Herramienta para el mantenimiento del amortiguador del tren de aterrizaje de helicópteros NH90

Rodrigo Álvarez Benéitez
Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación
75.663 TFG Arduino

Consultor: Antoni Morell Pérez
Profesor responsable de Área: Pere Tuset Peiró

3 de enero de 2021

“A Maitane y Aritz, a quienes les he robado el tiempo que les pertenecía y me han ayudado incondicionalmente a alcanzar mis metas. A mis amigos y compañeros, que me han animado a seguir esforzándome, sin bajar nunca los brazos. A toda la gente que creyó en mi, pero sobre todo a la gente que no lo hizo, verdadero motor que me hace constantemente superarme.”

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Herramienta para el mantenimiento del amortiguador del tren de aterrizaje de helicópteros NH90.*
Nombre del autor:	*Rodrigo Álvarez Benéitez*
Nombre del consultor/a:	*Antoni Morell Pérez*
Nombre del PRA:	*Pere Tuset Peiró*
Fecha de entrega (mm/aaaa):	01/2021
Titulación:	*Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación.*
Área del Trabajo Final:	*TFG Arduino*
Idioma del trabajo:	*Castellano*
Palabras clave	*Arduino, helicóptero, herramienta*

Resumen del Trabajo:

Se plantea crear una herramienta, para una tarea especifica de mantenimiento, que se realiza en las revisiones periódicas de diversas aeronaves. En este caso será una tarea descrita en el mantenimiento del helicóptero militar de transporte táctico HT-29 (denominación comercial NH90, fabricado por Airbus Helicopters).
La tarea de mantenimiento consiste en la comprobación del estado de los amortiguadores del tren de aterrizaje. Dicha tarea consiste en medir la temperatura del eje del amortiguador, el desplazamiento (longitud) del cilindro, y la presión del gas que contiene. Se realizará en cada uno de los tres ejes de los que dispone esta aeronave.
Como proyecto se propone crear un dispositivo, que debido a su morfología se pueda colocar (mediante una abrazadera o similar) en el eje del amortiguador, de forma que desde esa posición, pueda medir la temperatura del cilindro mediante infrarrojos, el desplazamiento del cilindro mediante una sonda ultrasónica, y por último envíe estos datos de forma inalámbrica a otro terminal, donde se podrán consultar las lecturas, además de valorar si son o no correctos. Este último dispositivo, nos indicará el rango de presión valido, para que sea comprobado mediante elementos externos. Por último almacenará las lecturas en una tarjeta SD.

Abstract:

It is proposed to create a tool for a specific maintenance task, which is carried out in the periodic inspection of several aircrafts. The task is described in the maintenance of the HT-29 military tactical transport helicopter (trade name NH90, manufactured by Airbus Helicopters).
The maintenance task consists of checking the condition of the landing gear shock absorbers. It is necessary to measure the temperature of the shock absorber shaft, the displacement (length) of the cylinder, and the pressure of the gas it contains. It will be carried out on each of the three axes available in this aircraft.
As a project, it is proposed to create a device, which due to its morphology can be placed (by means of a clamp or similar) on the axis of the shock absorber, so that from that position, it can measure the temperature of the cylinder with infrared sensor, the displacement of the cylinder with an ultrasonic probe, and finally send this data wirelessly to another terminal, where the readings can be consulted, in addition to assessing whether or not they are correct. This last device will indicate the valid pressure range to be checked by external elements. Finally, it will store the readings on an SD card.

ÍNDICE

TABLA DE ILUSTRACIONES

CAPITULO 1: INTRODUCCIÓN

1.1. ORGANIZACIÓN DE LA MEMORIA

En el capitulo 1, introducción, se tratan los orígenes de la aeronave sobre la cual se plantea el proyecto. En base a esto, se explica la necesidad de mantenimiento que se quiere cubrir. Se explica en qué consiste la tarea de mantenimiento descrita y los requisitos técnicos a los que tiene que sujetarse el prototipo de proyecto. Además se analiza la viabilidad del proyecto desde un punto de vista económico, operacional y de mercado.
En el capitulo 2, de estado del arte, se introducen las tecnologías empleadas. En este caso se trata de la plataforma Arduino. Además se detalla el funcionamiento de los diversos periféricos empleados, entre los que encontramos sistemas de medición, sondas de temperatura, emisores y receptores de radiofrecuencia, pantallas LCD, sistemas de almacenamiento, y otros.
En el capitulo 3, se detalla la propuesta del proyecto. En ella se explican los bloques hardware empleados, de forma individualizada para el elemento de medida y el elemento de control y visualización. Además, se detalla el código en los aspectos que se considere relevante.
En el capitulo 4, de diseño del dispositivo, se detalla el proceso de fabricación empleado para el prototipado, primeramente sobres protoboard, y posteriormente sobre placa shield. Se muestran las pruebas de precisión en banco, y pruebas de campo sobre el helicóptero. También se explica el funcionamiento del prototipo.
En el capitulo 5, se tratan las conclusiones. Concretamente las sinergias y experiencias obtenidas tras el empleo de la plataforma Arduino y de la elección de los periféricos empleados. Por otra parte se plantean posibles líneas de mejora sobre el prototipo. Además se analizan alternativas al presente proyecto, con ideas que pueden satisfacer las mismas necesidades de mantenimiento.

1.2. ANTECEDENTES

En los años 80, varios países pertenecientes a la Organización del Tratado del Atlántico Norte (OTAN-NATO), ante la necesidad de renovar parte de sus flotas de helicópteros de tamaño medio, decidieron crear un proyecto conjunto que les permitiera diseñar un helicóptero. Para lo cual, se marcaron como objetivo, la elaboración de un estudio común sobre las necesidades dictadas por cada uno, que permitieran la obtención de dicho helicóptero a un precio lo más ajustado posible, y a su vez permitiera a cada país, obtener unas sinergias positivas como el desarrollo de nuevas tecnologías o la creación de puestos de trabajo de alto valor. Estos países eran Francia, Alemania, Italia y Holanda. Posteriormente se unirían Portugal y España.

FIGURA 1 - NH90 DEL EJÉRCITO FRANCÉS

Ya en el año 2000, se formaliza el contrato para la producción de las primeras 298 unidades. El estado español entra en el proyecto en el año 2005 contratando un total de 104 unidades. Del total, 48 serían operadas por el Ejército de Tierra, para lo cual se diseña la versión para transporte táctico de tropas, 28 por la Armada, en su versión multipropósito naval y de lucha antisubmarina, y 28 unidades para el Ejército del Aire, en su versión para salvamento marítimo y transporte medicalizado.

FIGURA 2 - NH90 DEL EJERCITO ESPAÑOL

En el año 2016, el Ejército de Tierra, recibe sus primeras unidades, las cuales fueron designadas como HT-29 Caimán. Serían operados por el Batallón de Helicópteros de Maniobra III, sito en Agoncillo (La Rioja). El personal de dicho batallón tubo que realizar un proceso de migración desde los vetustos Bell UH-1H, que ya contaban con 52 años de servicio, y de los Eurocopter AS532 Cougar.

FIGURA 3 - BELL UH-1H

FIGURA 4 - EUROCOPTER AS532 COUGAR

A diferencia de los dos modelos anteriores, el nuevo HT29, además de unas capacidades hasta entonces desconocidas, gracias a su moderna tecnología, disponía de unos costes de mantenimiento muy elevados (entiéndase costes como: “horas de mantenimiento” / ”hora de vuelo”). Entre los cientos de diferentes tareas de mantenimiento previstas [15], la que atañe a esta memoria, es la que consiste en la revisión de los amortiguadores de los trenes de aterrizaje. Un componente vital para la seguridad de la aeronave [12].

1.3. MOTIVACIÓN Y OBJETIVOS

El NH90, dispone de un tren de aterrizaje retráctil formado por 3 ejes:

- Delantero, o NLG (Noise Landing Gear) [14].
- Dos traseros, o MLG (Main Landing Gear), uno a la izquierdo y otro a la derecha [13].

Una revisión que se realiza mensualmente consiste en medir la temperatura a la que se encuentra el cilindro del amortiguador [10] [11] (teniendo en cuenta también la temperatura ambiente, a la cual no puede existir una diferencia mayor de 1 grado centígrado), el desplazamiento de dicho amortiguador (es decir, la parte del cilindro que queda oculta dentro del casquillo del amortiguador), y la presión del nitrógeno de su interior. Para dichas comprobaciones se deben emplear herramientas individuales, como una simple regla, una sonda de temperatura Fluke, o un manómetro de presión.

La primera medida que se debe tomar es la correspondiente a la longitud del amortiguador que sobre sale del casquillo. El amortiguador completo mide 395 milímetros, con lo cual al querer conocer la longitud interior, tendremos que restar 395 menos la longitud del tramo que sobresale. A esta distancia total es a la que denominaremos “stroke”.

FIGURA 5 - MEDIDA DEL STROKE

Acto seguido, deberemos tomar una muestra de la temperatura a la que se encuentra el cilindro del amortiguador. Para ello emplearemos una sonda de temperatura con un multímetro FLUKE. Deberemos mantener el extremo de la sonda durante aproximadamente un minuto en permanente contacto con el cilindro. Además, deberemos mantener sin contacto la sonda durante aproximadamente otro minuto, para tomar el valor de la temperatura ambiental a la que se encuentra la aeronave.
Como una primera comprobación, deberemos de asegurarnos de que entre ambas temperaturas, no existe un diferencial mayor de 1 grado.

FIGURA 6 - MEDIDA DE TEMPERATURA DEL STROKE

Posteriormente, tomaremos el valor de presión a la que se encuentra el nitrógeno contenido en su interior. Para ello deberemos emplear un manómetro digital, y una llave fija. Con dicha llave tendremos que aflojar una tuerca, que permitirá comunicar el nitrógeno del cilindro con el manómetro. Con posterioridad a la medida, esa tuerca se deberá volver a cerrar. Dicha tuerca se encuentra en el vano del tren de aterrizaje, dificultando altamente su acceso.

FIGURA 7 - COMPROBACIÓN DE PRESIÓN

Todas estas medidas se realizan por un técnico en mantenimiento de aeronaves, que deberá estar recostado debajo del helicóptero en una posición incomoda que dificulta tomar las medidas con precisión al no haber apenas espacio. Además, el hecho de tener que arrastrarse por el suelo hasta introducirse debajo de la aeronave, supone un alto riesgo de sufrir un accidente, ya sea a través de traumatismo, o debido al riesgo de quedar atrapado debajo.

FIGURA 8 - MECÁNICO REALIZANDO COMPROBACIONES I

FIGURA 9 - MECÁNICO REALIZANDO COMPROBACIONES II

El operario deberá desplazarse hasta cada eje del tren de aterrizaje, para tomar las medidas (que deberá ir apuntando en un papel). Los datos tomados, son posteriormente representados en una grafica predefinida [9], que indica que valores son correctos y cuales no.
Dicha gráfica está representada para unos intervalos de temperatura ambiental (de -10º a 0º, de 0º a 10º, de 10º a 20º, de 20º a 30º, de 30º a 40º, y de 40 a 50º). Cada rango de temperaturas tiene una función máximo, y otra mínimo, por lo tanto los valores de presión deberán encontrarse entre dicho máximo y dicho mínimo para ser válidos.

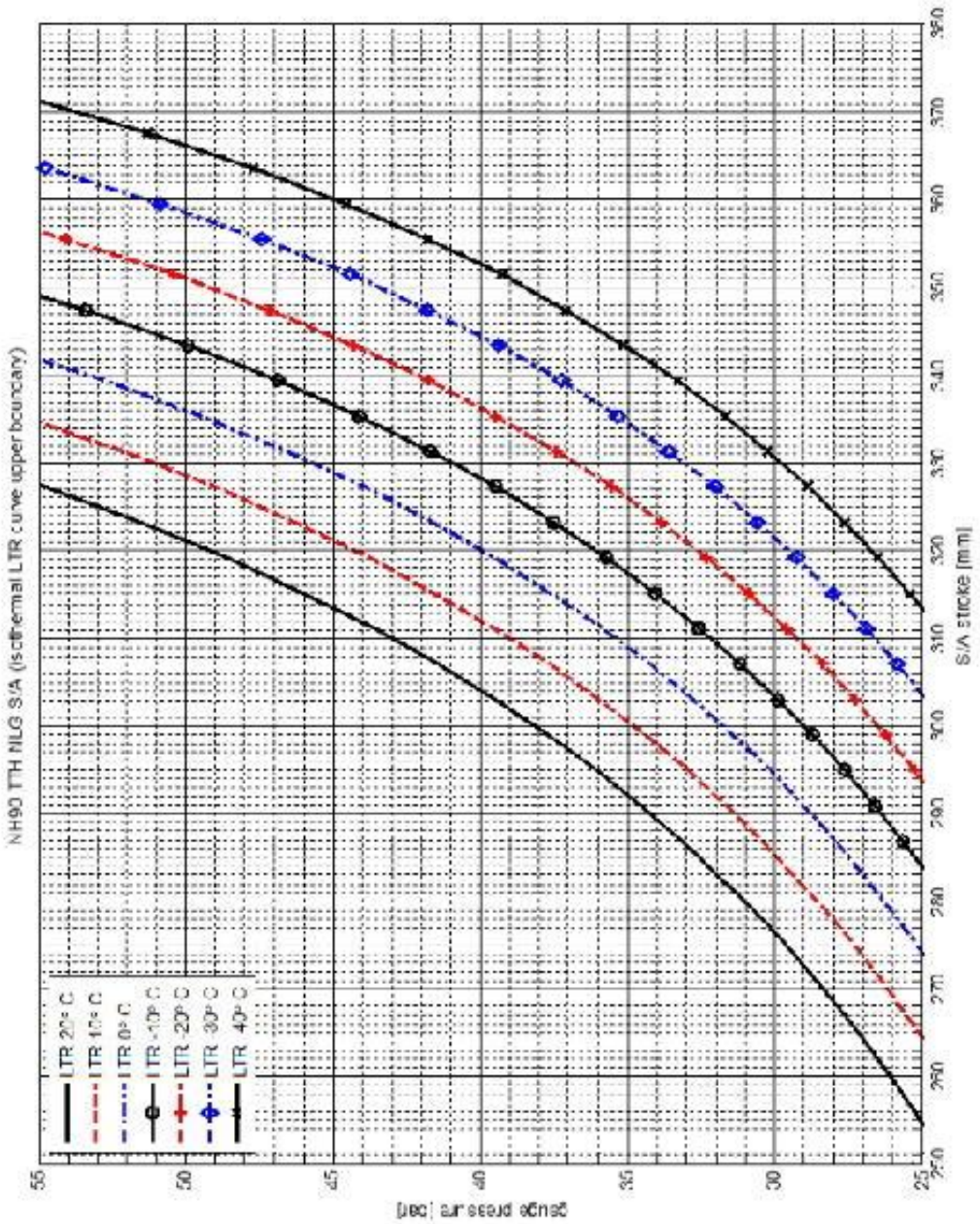


FIGURA 10 - TABLA DE VALORES DE PRESIÓN DEL NLG

Hasta la fecha, no existe ningún dispositivo que permita tomar todas las medidas a la vez, y de forma cómoda y segura para el mecánico.

En base a esta situación, el proyecto que se plantea es un conjunto de dos dispositivos que permitan facilitar la realización de dicha tarea de mantenimiento.

Por una parte se deberá contar con un dispositivo para la toma de las medidas necesarias (temperatura y desplazamiento del stroke), y que a partir de dichos parámetros, los cuales enviará a otro dispositivo (elemento de control y visualización) donde se muestren los valores, y se muestre automáticamente el rango de valores de presión que son válidos para dichos valores de temperatura y de stroke. Además, el elemento de control y visualización deberá permitir determinar si existe un diferencial superior a 1º entre la temperatura ambiental y la temperatura del stroke, lo cual consideraremos directamente como "T. fuera de limites".

1.3.1. REQUERIMIENTOS TÉCNICOS DEL PRODUCTO

El producto debe ser capaz de:

- Medir la temperatura del amortiguador sin contacto directo, mediante infrarrojos, con una resolución de $\pm$ 0,1º C.
- Medir la temperatura ambiental, con una resolución de $\pm$ 0,1º C.
- Calcular la longitud del stroke con una resolución de $\pm$ 3 mm.
- Enviar los parámetros medidos, desde el elemento de medida, hasta el elemento de control y visualización, mediante Radiofrecuencia, con una frecuencia central de 433 Mhz.
- Ambos dispositivos deberán contar con una pantalla LCD, que permita dar instrucciones básicas, o mostrar parámetros, según las necesidades o conveniencia de la tarea de mantenimiento.
- El elemento de control y visualización deberá contar con un lector de tarjetas SD, que permita almacenar los parámetros medidos.
- La toma de los valores de presiones queda fuera del alcance de este proyecto, debido a que los periféricos comercializados para la plataforma Arduino, no pueden soportar presiones de la magnitud de las necesarios.

1.4. VIABILIDAD DEL PROYECTO

Primeramente, a la hora de discutir la viabilidad de un proyecto como este, se puede citar algún que otro equipo, diseñado ad-hoc para este tipo de aeronaves militares. El ejemplo mas claro (aunque es evidente que supera en magnitud y complejidad al proyecto aquí planteado) que nos sirve para identificar la magnitud de los proyectos de defensa, es el "Banco SAME", diseñado por INDRA.

Se trata de un banco de pruebas y mantenimiento. Es de apariencia similar a dos armarios rack de 42 unidades, puestos uno al lado del otro. En su parte frontal dispone de diversas entradas y salidas que permiten conectar equipos y realizar pruebas. Las funciones que puede realizar el banco son las propias de osciloscopios, generadores de señales, etc. Mediante la conexión a un ordenador y con un software de pruebas, es capaz de realizar comprobaciones sobre todos y cada uno de los equipos contenidos en el helicóptero. Estos equipos pueden ser desmontados del helicóptero, como por ejemplo un DKU (Display Keyboard Unit), que es la interfaz de control por la que los pilotos controlan la aeronave. Esta se conecta al banco mediante cables auxiliares siguiendo las indicaciones de la interfaz del propio banco. El ordenador instalará en la DKU un software de pruebas, que indicará al operador las intervenciones necesarias (conectar o desconectar cables, así como realizar pulsaciones en la propia DKU). Una vez que termine las pruebas, mostrará el resultado y permitirá la exportación de los mismos, para ser empleados en la obtención de un certificado de aptitud de elemento.

Este banco de pruebas no hace nada que no se pudiera realizar de forma manual hasta ahora con diversos equipos, aunque si que nos aporta comodidad y precisión. El proyecto del banco SAME, adaptado al modelo de helicóptero Eurocopter NH90 (no es un desarrollo desde cero, ya que ya existía para otros modelos de aeronaves como el Eurocopter Tigre), tubo un presupuesto de 40,8 millones de Euros [1].

Es evidente que el proyecto que se plantea en esta memoria, es de una complejidad muy inferior al ejemplo mostrado, pero la cuestión es, que los proyectos de Defensa como este, con un buen funcionamiento, una buena presentación y que cubra una necesidad, puede ser el embrión para una herramienta que alcance un valor de varios o decenas de miles de euros, o que pueda ser evolucionado como parte de un equipo de mayores capacidades, que ayude en la realización de diversas tareas de mantenimiento distintas.

Por motivos de seguridad, no se pueden mostrar imágenes, ni más detalles de los necesarios relacionados con el Banco SAME para esta exposición.

Por otra parte, al ser el NH90 un proyecto común europeo, y que además el mismo modelo se ha exportado a distintos países fuera de los límites de la Unión, los posibles países compradores son amplios. Así mismo, con leves modificaciones puede adaptarse a distintos tipos de aeronaves.

1.4.1. VIABILIDAD ECONÓMICA

A la hora de analizar la viabilidad económica, se va a considerar el coste de las herramientas empleadas actualmente.

- Termómetro digital FLUKE 51 II de 1 canal. Este modelo es considerado como de "entrada de gama", con unas funciones muy básicas aunque suficientes. Su coste unitario es de 416,24€ (IVA incluido).
- Regla FACOM de acero inoxidable de 300 mm. Su coste unitario es de 12,68€ (IVA incluido).

El coste total [2] de las herramientas empleadas actualmente es de 428,92€ (IVA incluido).

En cuanto a los componentes empleados para el diseño del proyecto, podemos citar:

- Arduino MEGA 2560 Rev3. Se necesitarán 2 unidades. El coste unitario es de 39,94€ (IVA incluido).

- Conjunto de 2 conectores de alimentación para pila de 9V. Se necesitará 1 unidad. El coste unitario es de 4,95€ (IVA incluido).
- Pila de 9V. Se necesitarán 2 unidades. El coste unitario es de 2,99 (IVA incluido).
- Placas de prototipo PCB shield para Arduino. Se comercializa como un pack de mínimo 10 unidades. El coste del pack es de 14,87 (IVA incluido).
- Conjunto de Emisor y receptor RF 433 Mhz. Se necesitarán 2 unidades. El coste unitario es de 4,58€ (IVA incluido).
- Antena helicoidal para Arduino. Se comercializa en packs de 10 unidades. El coste del pack es de 5,25€ (IVA incluido).
- Sensor de temperatura infrarrojo MLX90614ESF. Se necesitará 1 unidad. El coste unitario es de 15,83€ (IVA incluido).
- Pantalla LCD 16x2 con interfaz i2c. Se necesitará 1 unidad. El coste unitario es 9,99€ (IVA incluido).
- Pantalla LCD 20x4 con interfaz i2c. Se necesitará 1 unidad. El coste unitario es de 10,99€ (IVA incluido).
- Lector de tarjetas SD. Se necesitará 1 unidad. El coste unitario es de 2,28€ (IVA incluido).
- Tarjeta SD. Se necesitará 1 unidad. El coste unitario es de 9,82€ (IVA incluido).

Se omiten al considerarse despreciable, los desechables como secciones de cable, estaño, pines, resistencias, leds, pulsadores, etc. El total para el montaje de 1 elemento de control y un elemento de medida, es de 169€ (IVA incluido).
Cabe decir, que al disponer previamente de varios componentes, que fueron reciclados de trabajos anteriores, se a podido reducir notablemente el presupuesto para el prototipado. En concreto se contaba con dos placas Arduino UNO rev3, que si bien no cuentan con las mismas capacidades que el modelo requerido (requerirá ajustes de código y ceñir el prototipo a un solo margen de temperaturas, de 0 a 10 grados y para un solo tren de aterrizaje), permiten ahorrar 79,88€ del total necesario. En resumen el presupuesto para el montaje del prototipo queda en 89,12€ (IVA incluido).

Se puede apreciar que el coste de las herramientas habituales con respecto al montaje de un conjunto de elemento de medida y elemento de control es 2,5 veces superior.

1.4.2. VIABILIDAD OPERACIONAL

Se omiten los gastos de fabricación, ya que el coste indicado es para la venta al por menor y se considera que al ser fabricado al por mayor, la reducción de costes en componentes es tal que permitirá cubrir la mano de obra para la fabricación, que en cualquier caso será externalizada.

1.4.3. VIABILIDAD DE MERCADO

Este proyecto se plantea como una herramienta diseñada ad hoc por encargo de un cliente, por lo que la viabilidad de mercado no puede ser analizada. La venta de una sola unidad permite costear el proyecto. No obstante el proyecto puede servir como proyecto de la propia institución (en este caso el Ejército de Tierra), por lo que aun no habiendo rentabilidad económica, puede suponer gasto asumible, valorando la mejora de la seguridad.

CAPITULO 2: ESTADO DEL ARTE

En este capitulo hablaremos de la plataforma Arduino, haciendo hincapié en su concepto como plataforma abierta, su descripción física, su memoria y dos tecnologías relevantes para este proyecto: el bus I2C y el bus SPI.

2.1. INTRODUCCIÓN A LAS TECNOLOGÍAS USADAS: ARDUINO

Como se define a si misma en su página web corporativa [3], Arduino es una plataforma electrónica de código abierto basada en hardware y software fáciles de usar. La placa controladora está formada por un conjunto de pines Entrada/Salida que pueden ser activados o leídos a conveniencia mediante el código. El código, denominado "lenguaje de programación Arduino", y basado en Wiring (guarda similitudes al lenguaje C), puede ser trabajado con su propio IDE. En cuanto a su software, cabe decir que lo primero que sorprende al comenzar a trabajar con Arduino, es que su código se estructura en una función setup, que se ejecuta una sola vez al inicializar, y una función loop, que se repite en bucle constantemente.

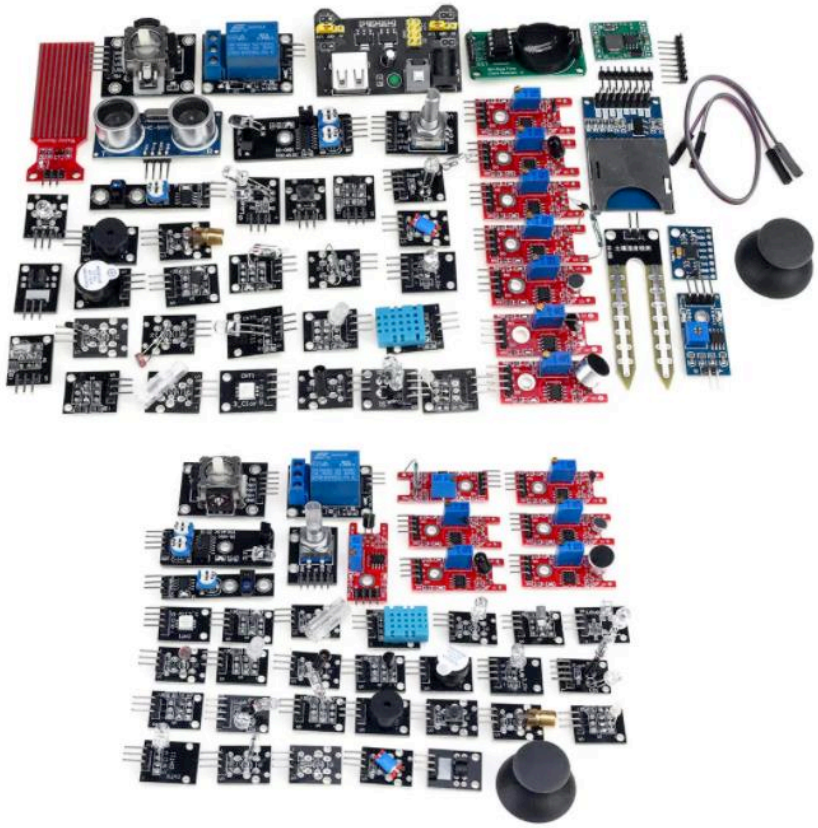


FIGURA 11 - DIVERSOS SENSORES PARA ARDUINO

La principal ventaja de Arduino como plataforma educativa, es que se pueden encontrar fácilmente en internet multitud de periféricos y componentes diseñados para Arduino, a un coste muy bajo. La ventaja de su bajo coste, supone también un hándicap, ya que la mayoría de los periféricos son de baja calidad, siendo en algunos casos una herramienta de aprendizaje, más que de desarrollo industrial. Además existe una amplia comunidad que genera documentación y librerías para cualquier aplicación, con lo que se pueden implementar multitud de proyectos de forma sencilla.

2.2. DESCRIPCIÓN FÍSICA.

La placa Arduino UNO rev.3 se compone [4] de los siguientes elementos:

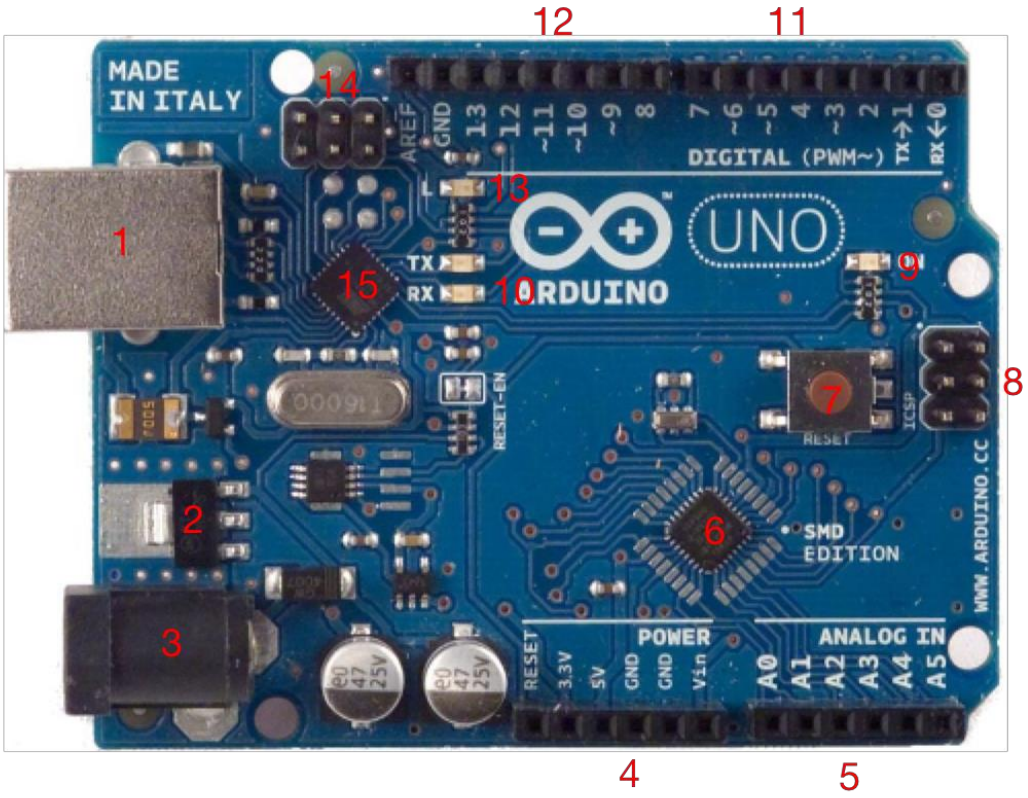


FIGURA 12 - PLACA ARDUINO UNO REV.3

1. USB: Permite por una parte, la comunicación para subir el sketch, y por otra nos permite alimentar la placa.
2. Regulador de tensión: convierte el voltaje de alimentación en 5 voltios.
3. Plug de conexión para una fuente de alimentación externa.
4. Puertos de conexiones de alimentaciones, masas y control de reset.
5. Entradas analógicas.
6. Microcontrolador ATMega328.
7. Botón de reseteo.
8. Pines de programación ICSP.
9. Led que indica que está en funcionamiento.
10. Leds de transmisión y recepción.
11. Puertos I/O. Algunos de ellos aceptan su uso como salidas controladas por ancho de pulso.
12. Más puertos I/O.
13. Led de estado.
14. Más pines de programación ICSP.
15. Chip de conversión de serial a USB.

2.3. BUS I2C

El bus I2C [5] es uno de los principales sistemas de comunicación de Arduino (junto con el puerto serie y el bus SPI). Es de enorme ayuda, ya que la mayoría de los periféricos encontrados en internet trabajan con el. Su principal ventaja radica en que nos permite conectar periféricos empleando tan solo 2 pines I/O, uno para la señal de reloj (CLK) ya que es un sistema síncrono, y otro para la transmisión de datos (SDA).

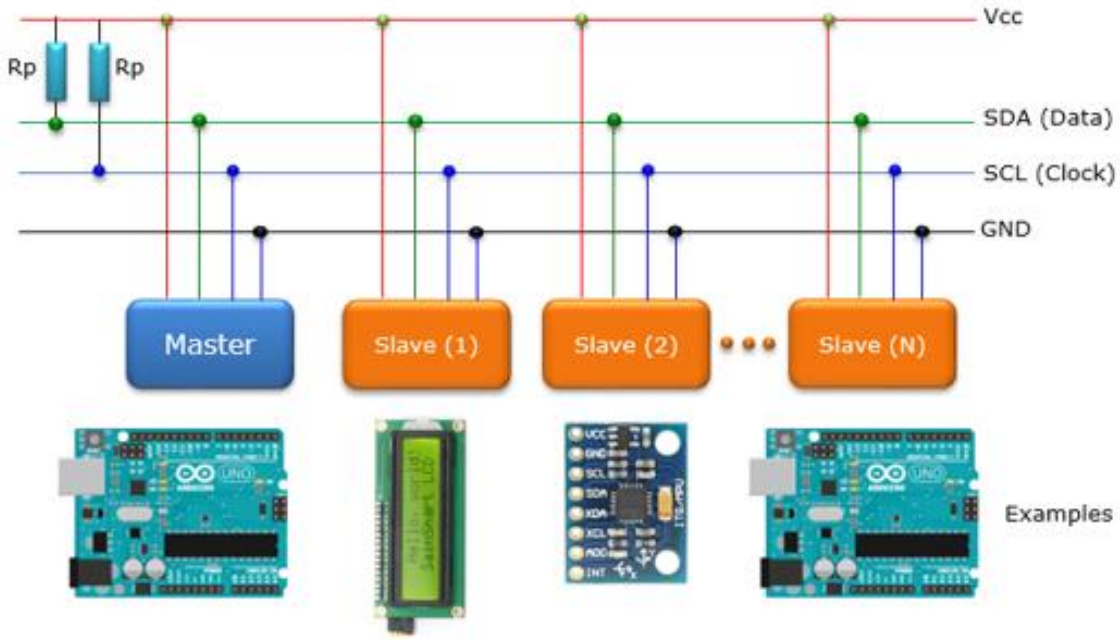


FIGURA 13 - REPRESENTACIÓN DEL BUS I2C

El bus I2C emplea una trama formada por 7 bits en los que identifica la dirección del periférico, 1 bit que indica si se va a realizar una lectura o una escritura, 1 bit de validación, 1 o más bytes de datos, y por último un bit de validación. Gracias a estas tramas, el bus puede comunicarse con tan solo un cable de datos.

Todos los periféricos conectados al bus, tienen que ser inicializados, y dispondrán de una dirección, que servirá para identificarlos. Normalmente estas direcciones se pueden modificar mediante un sistema de JUMPERS, de modo que todas sean únicas.

El protocolo I2C suele ir conectado a resistencias tanto para el SDA como para el CLK, que irán conectadas a VCC, no obstante mediante el empleo de la librería Wire.h, se pueden omitir y emplear las resistencias PULL-UP internas que nos ofrece Arduino. Estas serán suficiente para pequeñas longitudes de conexión.

2.4. BUS SPI

El bus SPI [6], es un protocolo de comunicación síncrona de 4 hilos. En este bus, la comunicación es entre maestro y esclavo, en líneas separadas, una para el maestro y otra para el esclavo, por lo que es Full Dúplex. Se puede enviar cualquier secuencia arbitraria de bits, ya que las tramas no siguen ninguna regla.

De los dos hilos de datos, tenemos uno de ellos, llamado MISO (Master Input Slave Output) empleado para recibir los datos desde otro elemento, y el hilo llamado MOSI (Master Output Slave Input), para transmitir datos hasta otro elemento. Por último tenemos el hilo CS, (también llamado en algunas ocasiones SS), empleado como selector de esclavo. Utiliza una línea para la señal de reloj, y dos para datos. Además de en Arduino, se emplea para comunicaciones en circuitos impresos.

FIGURA 14 - CONEXIONADO DEL BUS SPI

Cabe decir, que el bus SPI solo es válido para distancias cortas (unos 30 cm máximo). No dispone de ningún mecanismo de control de errores, por lo que no podemos saber si el mensaje ha sido recibido en el destino.

Arduino dispone de soporte para SPI por hardware, y este está predefinido a ciertos puertos. Para Arduino UNO tenemos CS en el puerto 10, MOSI en el puerto 11, MISO en el puerto 12 y SCK en el puerto 13.

2.5. MEMORIA

Arduino se compone de dos tipos de memoria [7], la memoria de programa o código, y la memoria de datos.

2.5.1. MEMORIA DE CÓDIGO O FLASH

Tiene como misión el albergar el código fuente o sketch de nuestro proyecto. Está formado por clústeres de 16 bits. Suelen ser de entre 4KB y 256KB. Para Arduino UNO rev.3, es de 32 KB.

2.5.2. MEMORIA DE DATOS.

Tiene como misión, el servir de espacio de almacenamiento temporal. Se agrupa en SRAM y EEPROM. La SRAM es empleada como memoria volátil para variables locales. Es un recurso crítico por su escaso tamaño. La EEPROM es memoria no volátil, que se emplea para almacenar datos que pueden ser mantenidos después de un reset. Tiene un tiempo de vida limitado. Para Arduino UNO rev.3. la SRAM (la que mas nos afectará en este proyecto) es de 2KB, siendo de 8KB para un Arduino MEGA, y de 32KB en un Arduino MKR1000.

CAPITULO 3: PROPUESTA DE PROYECTO

3.1. INTRODUCCIÓN

El proyecto consistirá en un elemento medidor, dotado de un sensor de ultrasonidos, un sensor infrarrojo, una pantalla, un pulsador y un sistema de comunicación RF, que permitirá capturar el valor de la distancia del stroke, y captar la temperatura ambiental y del stroke. Esta toma de valores se hará automáticamente desde que el dispositivo reciba la alimentación. En el momento que el operador pulse el único botón que tiene, enviará los datos al elemento de control.

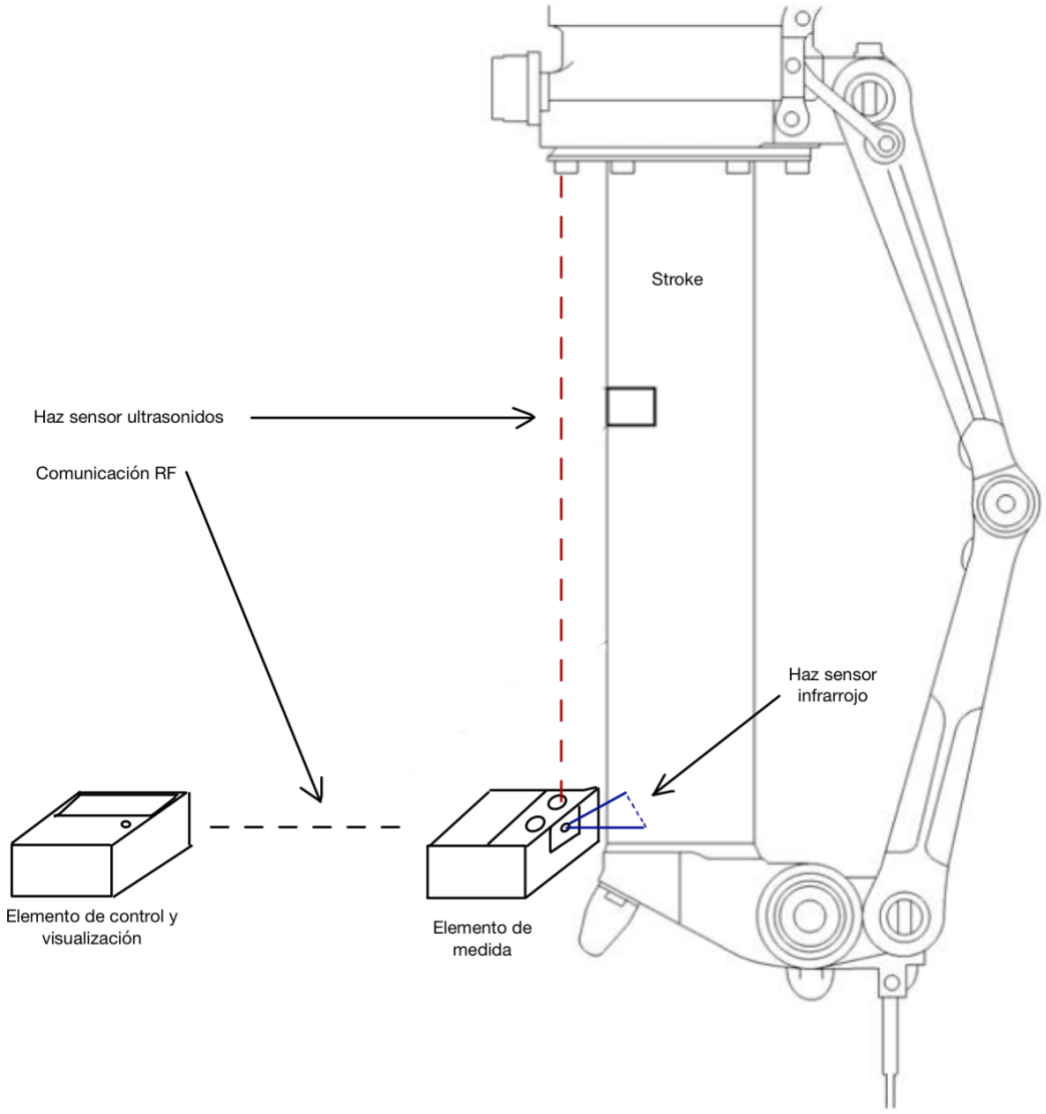


FIGURA 15 - REPRESENTACION DEL AMORTIGUADOR Y DE LOS PROTOTIPOS

El elemento de control contará con una pantalla, un sistema de almacenamiento SD, un pulsador, y un sistema de comunicación RF. Desde el momento en que el dispositivo reciba alimentación, quedará permanentemente a la espera de recibir parámetros por RF, por lo que no necesita la intervención del operador. El pulsador tiene el fin de indicarle al dispositivo, si estamos ante un amortiguador de tipo NLG o de tipo MLG (este será el elegido por defecto), ya que los parámetros de valores válidos son distintos. Alternaremos de un tipo al otro, con solo una pulsación. Además, el tipo elegido se mostrará mediante un led (color azul para MLG, color verde para NLG). En el mismo instante en que el dispositivo reciba los parámetros del elemento de medida por RF, los

mostrará por pantalla, y nos indicará el rango de presiones a las cuales puede funcionar ese amortiguador.

El diagrama de bloques que representa a ambos elementos es el siguiente.

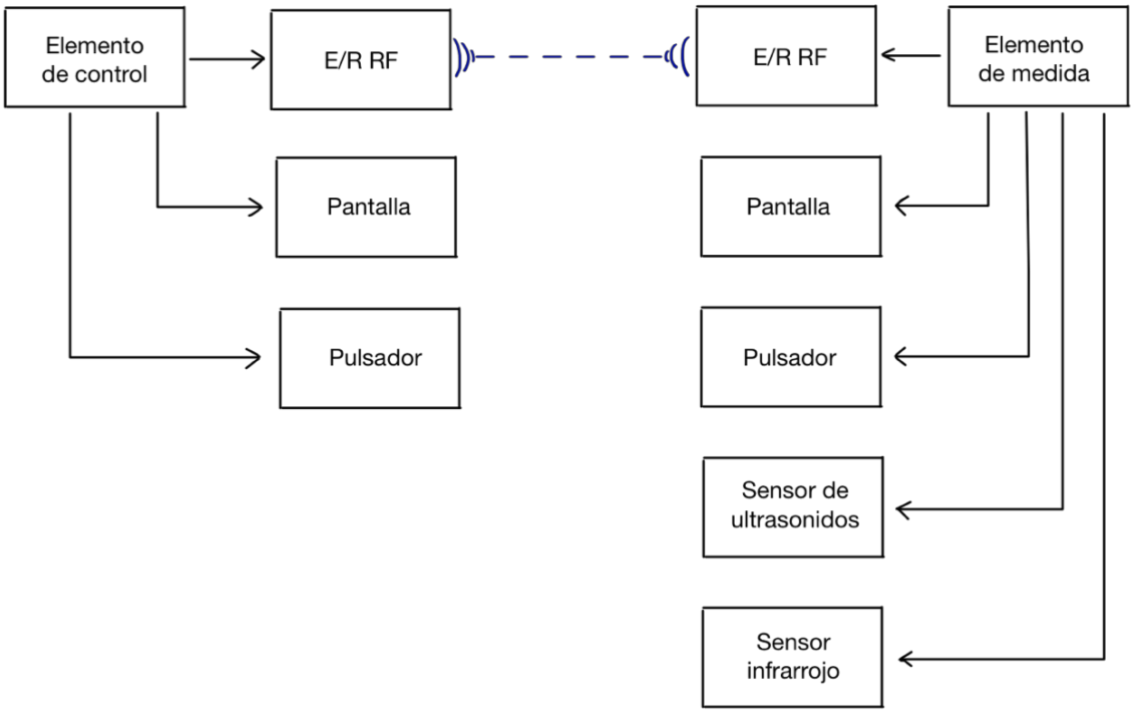


FIGURA 16 - DIAGRAMA DE BLOQUES

3.2. DESCRIPCIÓN DE LOS BLOQUES HARDWARE

A continuación se detallan los periféricos que se emplearán para el prototipado. Además se describirán las pruebas de funcionamiento de cada uno de estos bloques.

3.2.1. ELEMENTO DE MEDIDA

En este subapartado se van a detallar el funcionamiento de los bloques mas relevantes del elemento de medida, como son los de comunicación, captura de parámetros, y representación de información.

3.2.1.1. SISTEMAS DE REPRESENTACIÓN DE TEXTO: PANTALLAS LCD

El primero de los periféricos empleados, y además el más sencillo es la pantalla. Se han empleado dos modelos distintos, uno 16x2 y otro 20x4. El funcionamiento es idéntico (salvo la lógica diferencia de tamaño, y la distinción en el momento de la declaración mediante "LiquidCrystal_I2C lcd(0x27,20,4);"). Ambos modelos cuentan con comunicación I2C. Esta pantalla consta de 4 líneas de 20 caracteres (la 20x4), con retroiluminación azul. Para trabajar con este tipo de pantallas mediante el bus I2C, debemos emplear una librería. Existen varias aunque la elegida (debido a su sencillez) es la Liquid Crystal I2C, del autor Frank de Brabander.
El conexionado es sencillo. Tenemos 4 pines: GND, VCC, SDA, SCL. Conectaremos GND a una toma GND de Arduino, Vcc a toma 5V de Arduino, SDA a la entrada analógica a4 del Arduino, y SCL a la entrada analógica a5 del Arduino.

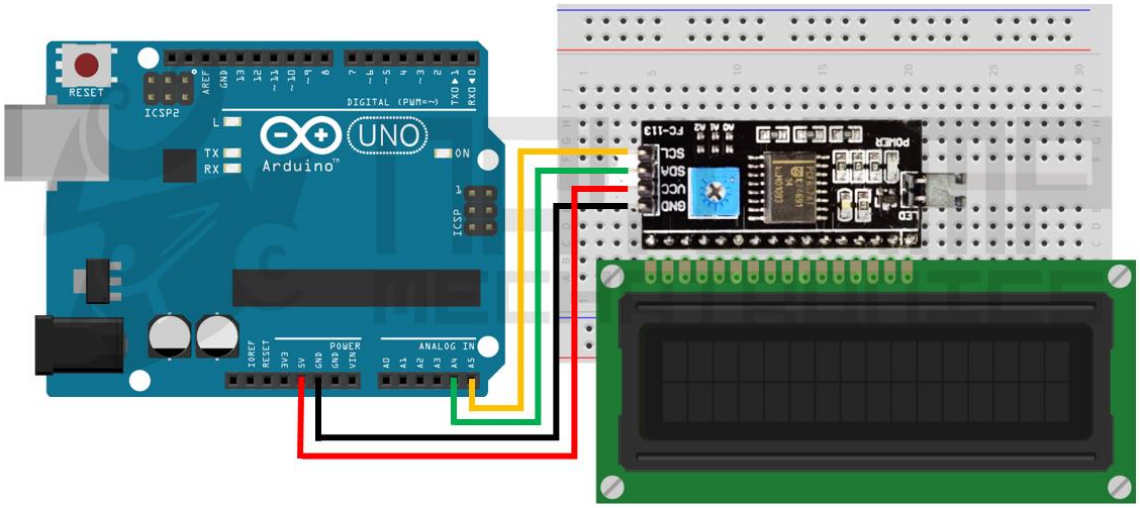


FIGURA 17 - CONEXIONADO DE PANTALLA LCD POR I2C

Esta imagen cuenta con una pantalla 16x2, aunque sus principios son válidos para una 20x4, ya que las conexiones y su funcionamiento, son idénticos. En algunos modelos de pantalla, como es el caso del que he adquirido, el módulo I2C viene directamente soldado por detrás de la pantalla.

En cuanto a su programación, podemos emplear algunas de las siguientes instrucciones:

- Lcd.init(); //Inicializa el módulo adaptador LCD a I2C
- Lcd.backlight(); //Enciende la retroiluminación.
- Lcd.noBacklight(); //Apaga la retroiluminación.
- Lcd.clear(); //Borra la pantalla
- Lcd.setCursor(columna,fila); //Posiciona el cursor del LCD
- Lcd.print("texto"); //Escribe un texto en la pantalla

- Lcd.scrollDisplayRight(); //Desplaza el contenido del texto y cursor 1 espacio hacia la derecha.
- Lcd.scrollDisplayLeft(); //Desplaza el contenido del texto y cursor 1 espacio hacia la izquierda.

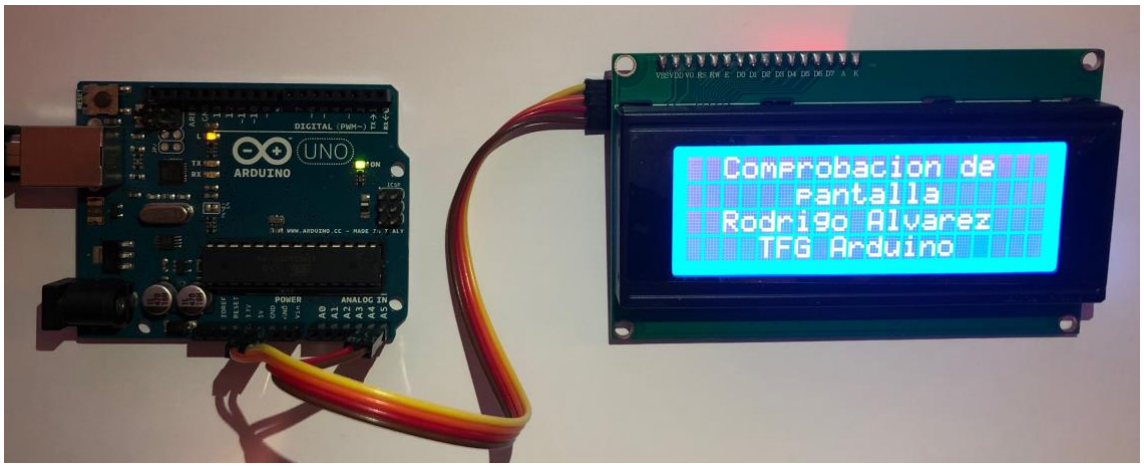


FIGURA 18 - MENSAJE POR PANTALLA

Se generó el siguiente código para realizar una prueba sencilla de funcionamiento. Todas las pruebas realizadas sobre los siguientes periféricos fueron realizadas sobre la base de este sketch, añadiéndoles cada vez mayores capacidades:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,20,4);  // set the LCD address to 0x27 for a 16
chars and 2 line display

void setup()
{
  lcd.init();
  lcd.backlight();
  lcd.setCursor(2,0);
  lcd.print("Comprobación de");
  lcd.setCursor(6,1);
  lcd.print("pantalla");
  lcd.setCursor(2,2);
  lcd.print("Rodrigo Álvarez");
  lcd.setCursor(4,3);
  lcd.print("TFG Arduino");
}

void loop()
{
}
```

3.2.1.2. SISTEMAS DE MEDICIÓN DE DISTANCIAS POR MEDIO DE ONDAS

Para la medición de distancias, se opto por un sensor de ultrasonidos. El modelo elegido es el HY-SRF05. Este modelo, que según su datasheet, cuenta con un rango de medida de entre 2cm y 450cm, y con una precisión de 0.3 cm, se caracteriza por tener un bajo coste. Está constituido por un transmisor de forma cilíndrica, por el que emite un ultrasonido, y un receptor de la misma apariencia, por la que detecta la onda emitida. Mediante la medida del tiempo que tarda en volver la onda, se calcula la distancia.
Tras las pruebas de uso, se desprende que en este tipo de sensores, las medidas oscilan constantemente, con lo que se opta por recoger nueve muestras y hacer la mediana, de modo que se consiga suavizar su imprecisión. Esto será realizado gracias a la librería RunningMedian.

Su conexionado es sencillo. VCC a 5V, GND con GND, TRIG a la entrada 8, ECHO a la patilla 9, y OUT sin conectar.
A través de su pin TRIG, damos orden de disparo, de modo que el periférico emite una señal. Y a través del pin ECHO, nos da la señal de llegada de la onda.

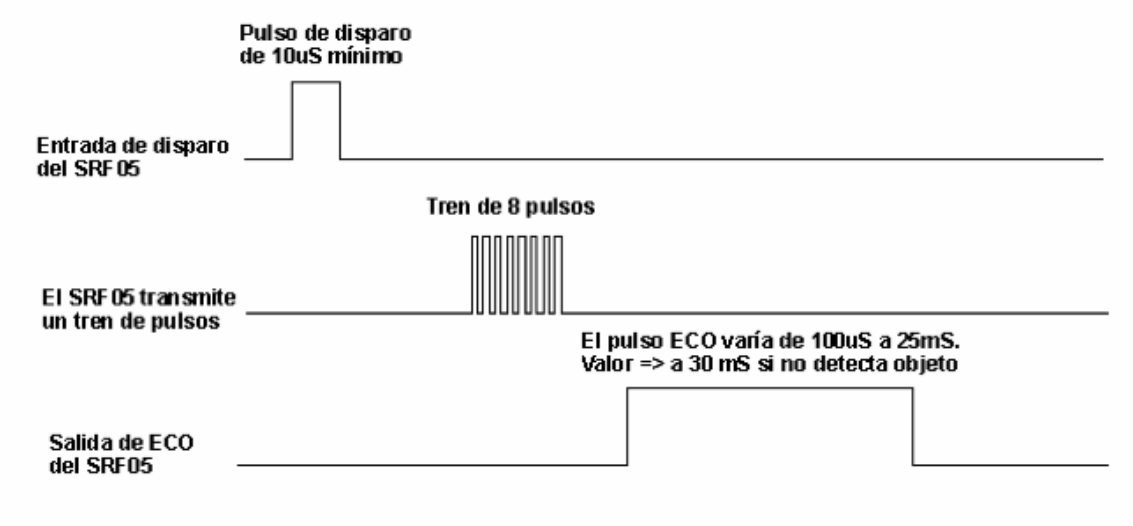


FIGURA 19 - REPRESENTACIÓNM DE LA SEÑAL

Tal y como se muestra en el diagrama de tiempos [8], el modo de empleo es muy sencillo. Externamente se aplica, por parte del usuario, un pulso de disparo o trigger de 10 µS de duración mínima. Se inicia la secuencia. El módulo transmite un tren de pulsos o "burst" de 8 ciclos a 40KHz. En ese momento la señal de salida ECHO pasa a nivel 1 lógico. Cuando el receptor recibe la señal transmitida como eco, esta salida pasa de nuevo a nivel 0 lógico. El usuario debe medir la duración del pulso de esta señal, es decir, el tiempo en que la señal eco se mantiene a 1 lógico.

3.2.1.3. SISTEMAS DE MEDICIÓN DE TEMPERATURAS MEDIANTE INFRARROJOS

Para la medición de temperaturas, se eligió el modelo MLX90614. Este modelo, que detecta la temperatura a distancia mediante infrarrojos, funciona mediante una interfaz I2C. El motivo de su elección fue la precisión que ofrece en las medidas tomadas (asegura en su datasheet una precisión de 0,02º para un rango de temperaturas de -70ºC hasta 382,2ºC). Además no necesita contacto con el objeto para tomar valores. En cuanto a esto último, se ha podido comprobar que el sensor es mas preciso cuanto mas cerca se encuentra del objeto. Esto es debido a que el haz infrarrojo generado es poco direccional (aproximadamente 80º). Existe una versión mejorada con

un haz de emisión de tan solo 3º, aunque su coste se multiplica por 8, con lo que se desechó por una cuestión presupuestaria. Por otra parte, el modelo elegido dispone también de una segunda sonda de temperatura ambiental, que nos ofrece datos también muy precisos.
En cuanto al conexionado, optamos por conectarlo de forma paralela a la pantalla, ya que ambos trabajarán con el bus I2C.

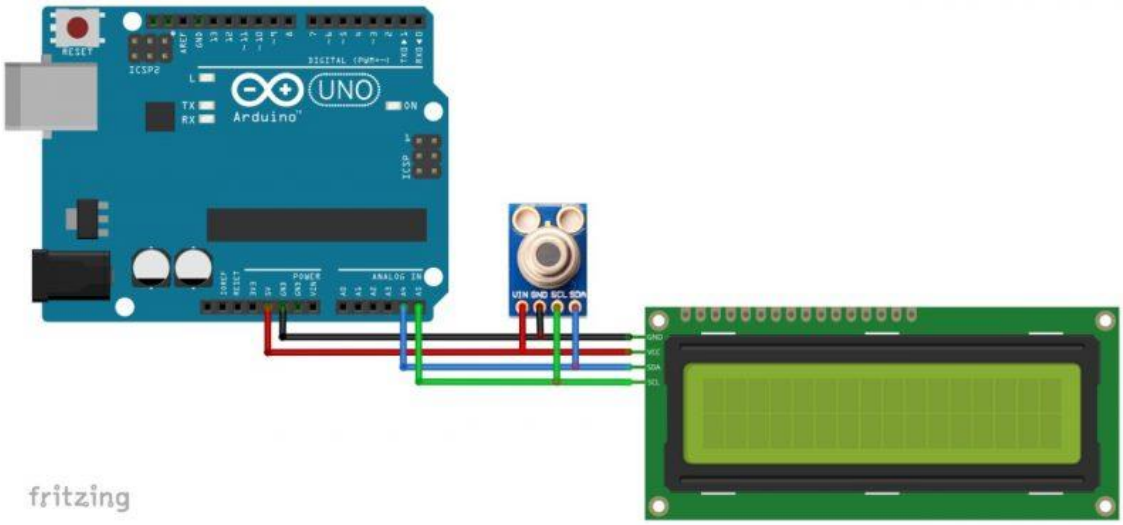


FIGURA 20 - CONEXIONADO DEL SENSOR INFRARROJO

Para el funcionamiento del propio sensor, tendremos que emplear la librería Adafruit_MLX90614. Creando un objeto de la clase Adafruit_MLX90614, que tendremos que inicializar mediante la función "begin" podremos obtener los valores de temperatura del objeto mediante infrarrojos con la función "readObjectTempC" (para el valor en centígrados), y el valor de la temperatura ambiental mediante "readAmbientTempC".

En base al código generado para las pruebas del medidor de ultrasonidos, se genera uno nuevo que muestra por pantalla las temperaturas junto con la distancia medida por el sensor de ultrasonidos.

```
#include <Wire.h>

#include <LiquidCrystal_I2C.h>

#include <RunningMedian.h>

#include <Adafruit_MLX90614.h>


LiquidCrystal_I2C lcd(0x27,20,4);

RunningMedian mediana = RunningMedian(9);

Adafruit_MLX90614 TemperaturaIR = Adafruit_MLX90614();


int trigger = 8;

int echo = 9;

int i = 0;
```

```
void setup()
{
  //Inicializo la pantalla
  lcd.init();
  //Activo la retroiluminación de la pantalla
  lcd.backlight();

  //Establezco los dos pines del sensor de ultrasonidos
  pinMode(trigger, OUTPUT);
  pinMode(echo, INPUT);
  digitalWrite(trigger, LOW);

  //Inicializo el sensor de temperaturaIR
  TemperaturaIR.begin();
}

void loop()
{
  float d;     //distancia en centímetros

  digitalWrite(trigger, LOW);
  delayMicroseconds(5);    //Estabiliza el sensor.
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);    //Enviamos un pulso de 10 us.
  digitalWrite(trigger, LOW);

  //Relleno el Runningmedian para calcular la mediana de las medidas tomadas.
  for(i=0; i<10; i++){
    mediana.add(MedirTiempo());
  }
```

```
  d = (0.034 * mediana.getMedian()) / 2;              //Escalamos el tiempo
  lcd.setCursor(0,0);
  lcd.print("Distancia:          ");
  lcd.setCursor(11,0);
  lcd.print(d);
  delay(1000);

  //Prueba de sensor IR
  lcd.setCursor(0,1);
  lcd.print("Temp. objeto:       ");
  lcd.setCursor(14,1);
  lcd.print(TemperaturaIR.readObjectTempC());
  lcd.setCursor(0,2);
  lcd.print("Temp. ambien:       ");
  lcd.setCursor(14,2);
  lcd.print(TemperaturaIR.readAmbientTempC());
  lcd.setCursor(0,3);
}

int MedirTiempo(){
  int t;   //tiempo que tarda en llegar el eco
  digitalWrite(trigger, LOW);
  delayMicroseconds(5);    //Estabiliza el sensor.
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);    //Enviamos un pulso de 10 us.
  digitalWrite(trigger, LOW);
  t = pulseIn(echo, HIGH);    //Obtenemos el ancho del pulso
  delay(100);
  return(t);
}
```

3.2.1.4. SISTEMAS DE COMUNICACIÓN MEDIANTE RADIOFRECUENCIA

Para la comunicación entre el elemento de medida y el elemento de control y visualización, se decidió emplear un emisor y receptor de RF. En concreto los modelos elegidos son el FS1000A (emisor) y el XY-MK-5V (receptor). Ambos suelen comercializarse sin antena, con lo que su alcance queda reducido a unos escasos centímetros. Por tanto, debemos soldar antenas helicoidales a unos puntos de la placa diseñados para tal fin (identificados por ANT). Estos E/R's trabajan con una frecuencia de operación de 433 Mhz, por tanto una banda libre (de uso gratuito). Existen modelos similares con una frecuencia de trabajo de 315 Mhz. En función de la tensión de alimentación y de la longitud de la antena, podemos alcanzar hasta un máximo de 300 metros en exteriores (en las pruebas de laboratorio, a partir del metro de distancia, la comunicación fue poco fiable).
Estos periféricos se basan en una comunicación simplex mediante modulación ASK de 2400bps, no obstante es suficiente para la transmisión de pequeños parámetros.
Se optó por este tipo de comunicación, al disponerse de 2 parejas de E/R que pudieron ser reciclados de proyectos anteriores, lo que nos permitió reducir los costes de prototipado.

FIGURA 21 - EMISOR Y RECEPTOR DE 433 MHZ

En cuanto a su conexionado, contamos con un emisor de 3 patillas, DATA (conectado al pin 6), VCC y GND. En cuanto al receptor, contamos con VCC, un pin DATA sin conectar, DATA (conectado a pin 5) y GND.
Para la comunicación se emplearon las librerías "RH_ASK.h" y "RHreliableDatagram.h". Y dentro de los métodos que estas nos permiten para la realización de la transmisión, se empleo sendtoWait, que envía el paquete, y se queda a la espera. A través de la función waitPacketSent, podemos controlar la llegada de los paquetes al destino, y conocer si tuvimos una perdida de datos (devuelve un booleano con valor FALSE si la comunicación no ha sido completa).
Las pruebas se realizaron sobre los sketches de pruebas de los apartados anteriores, no obstante, al parecerse demasiado al código final, se omite aquí su presentación. Puede ser consultado el código final en el ANEXO correspondiente.

3.2.2. ELEMENTO DE CONTROL Y VISUALIZACIÓN

En este subapartado se va a detallar el funcionamiento de los bloques del elemento de control y visualización (se omiten los que ya han sido detallados para el elemento de medida, ya que son idénticos).

3.2.2.1. SISTEMA DE ALMACENAMIENTO SD

El dispositivo de almacenamiento SD, es un modelo con comunicación SPI. En cuanto a su conexionado tenemos 6 patillas, CS (I/O 10), SCK (I/O 13), MOSI (I/O 11), MISO (I/O 12), VCC y GND.

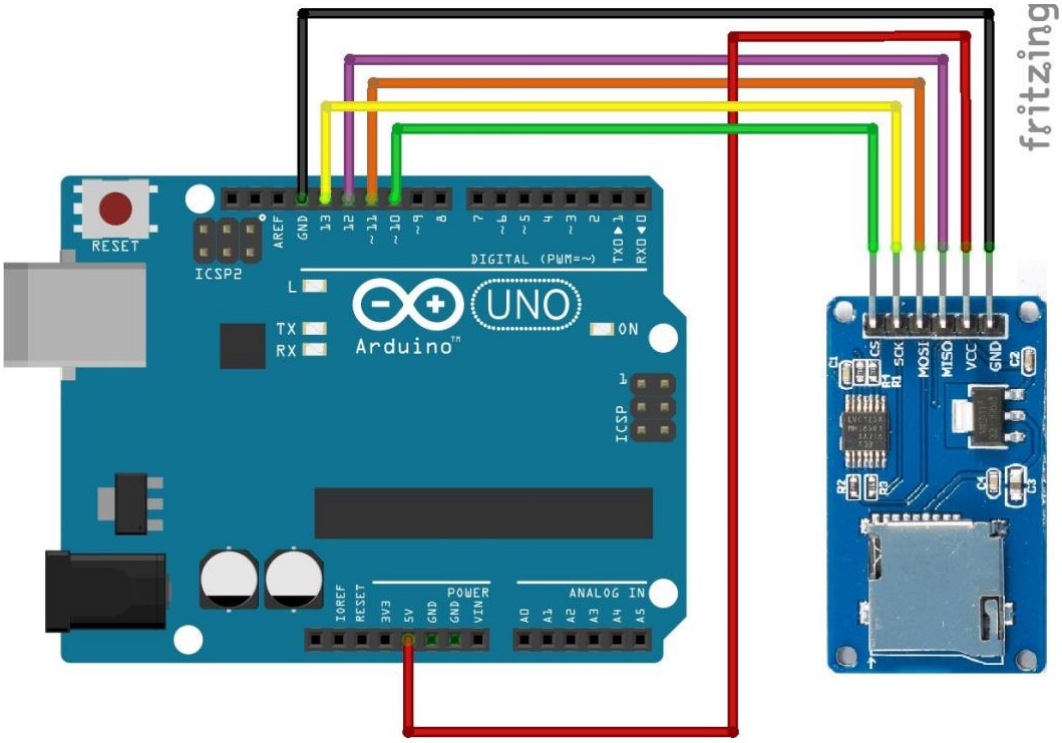


FIGURA 22 - CONEXIONADO LECTOR SD

La asignación de puertos I/O viene predefinida en la librería “SD.h”, la cual nos permite gestionar los archivos. Para ello solo tendremos que declarar un objeto a través de “File myFile;”, que posteriormente podremos abrir mediante la función “SD.open” y cerrar mediante “SD.close”. El resto del funcionamiento es similar a la salida por pantalla: myFile.print(“…”). Para las pruebas de funcionamiento, no se genero ningún código, ya que la librería “SD.h” cuenta con un ejemplo de escritura muy sencillo, cuyo código posteriormente fue reciclado para incrustarlo en el código fuente del proyecto.

CAPITULO 4: DISEÑO DEL DISPOSITIVO

En este capitulo, se explicará y mostrará el montaje realizado, primeramente sobre una protoboard, y posteriormente sobre un shield. Además se realizarán unas pruebas básicas de precisión de los valores medidos, tanto de distancia como de temperatura, para asegurarnos de que se ajustan a los requerimientos técnicos previos del proyecto, así como si se cumplen los valores indicados en los correspondientes datasheet. El funcionamiento básico de los distintos periféricos no será explicado aquí, ya que ha sido detallado en el capitulo anterior.

4.1. MONTAJE SOBRE PROTOBOARD.

En cuanto a los pulsadores, decir que no fueron montados con resistencias dedicadas, ya que se opto por omitirlas, y configurar en el código, las entradas como “INPUT_PULLUP”.

En cuanto a los diodos led, hay que tener en cuenta que necesitan conectarse a través del positivo del diodo, a una resistencia, y esta resistencia al puerto correspondiente de Arduino. El terminal negativo del diodo se conectará a GND. El valor de la resistencia, esta condicionado al color del led, ya que cada color tiene una tensión umbral distinta.

$$R = \frac{V_{cc} - V_{umbral}}{I_{nominal}}$$

El valor de la intensidad nominal de un led, decir que es variable en función del led y habitualmente se normaliza entorno a un valor de 0,02A. Para el led verde, la tensión umbral tiene un valor de 2,4V, por lo tanto:

$$R_{verde} = \frac{5 - 2{,}4}{0{,}02} = 130\ ohmios\ de\ valor\ minimo$$

Para el led azul, la tensión umbral tiene un valor de 3,4V, por lo que:

$$R_{azul} = \frac{5 - 3{,}4}{0{,}02} = 80\ ohmios\ de\ valor\ minimo$$

En este caso, para un diodo led de color verde, necesitamos una resistencia de al menos 130 ohmios, mientras que para la azul, necesariamente debe de ser de al menos 80 ohmios. En base a las resistencias de las cuales se disponía, y siendo que el valor calculado es el mínimo, se opto por usar sendas resistencias de 560 ohmios, que como se pudo comprobar, permitían un funcionamiento correcto.

4.1.1. CONEXIONADO DEL ELEMENTO DE CONTROL Y VISUALIZACIÓN

El conexionado será el siguiente:

Módulo LCD	Arduino
GND	GND
VCC	5V
SDA	A4
SDL	A5

Emisor RF	Arduino
DATA	6
VCC	5V
GND	GND

Receptor RF	Arduino
VCC	5V
DATA	Sin conectar
DATA	5
GND	GND

Lector SD	Arduino
CS	10
SCK	13
MOSI	11
MISO	12
VCC	5V
GND	GND

Pulsador	Arduino
1	3
2	GND

Led verde	Arduino
Positivo (a resistencia)	8
Negativo	GND

Led azul	Arduino
Positivo (a resistencia)	9
Negativo	GND

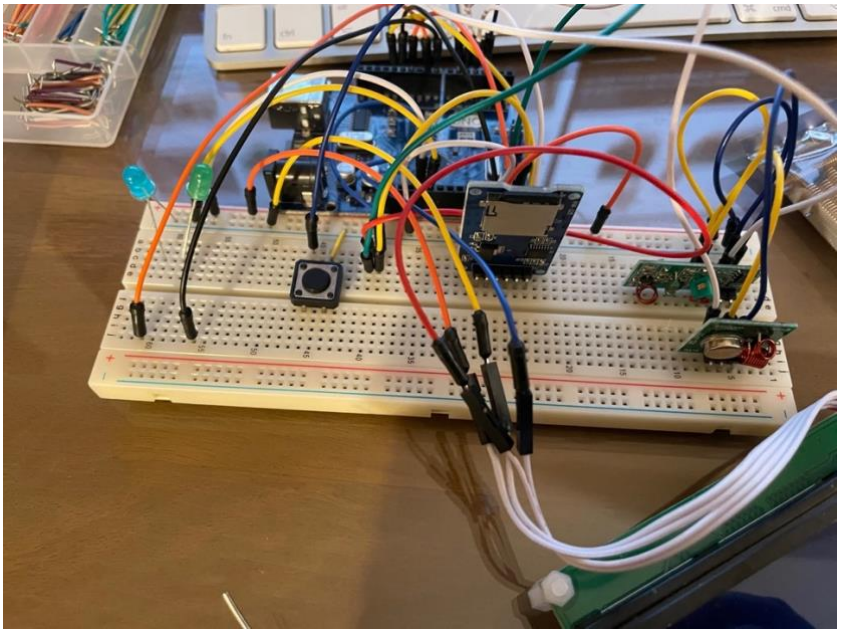


FIGURA 23 - MONTAJE DE ELEMENTO DE CONTROL SOBRE PROTOBOARD

4.1.2. CONEXIONADO DEL ELEMENTO DE MEDIDA

El conexionado será el siguiente:

Módulo LCD	Arduino
GND	GND
VCC	5V
SDA	A4
SDL	A5

Emisor RF	Arduino
DATA	6
VCC	5V
GND	GND

Receptor RF	Arduino
VCC	5V
DATA	Sin conectar
DATA	5
GND	GND

Módulo ultrasonidos	Arduino
VCC	5V
GND	GND
TRIG	8
ECHO	9
OUT	Sin conectar

Módulo infrarrojos	Arduino
GND	GND
VCC	5V
SDA	A4
SCL	A5

Pulsador	Arduino
1	3
2	GND

4.2. PRUEBAS DE PRECISIÓN.

4.2.1. PRUEBAS DE PRECISIÓN: SENSOR DE TEMPERATURA POR INFRARROJOS

Para la comprobación de la precisión del sensor de temperatura por infrarrojos, se coloco a una distancia de aproximadamente 12 cm, un recipiente metálico. Así mismo se empleó un termómetro de cocina para comprobar la exactitud en la medida. Tras las pruebas se pudo confirmar que el sensor es capaz de medir con precisión, incluso a una decena de centímetros de distancia. Lo que se considera un resultado excelente.

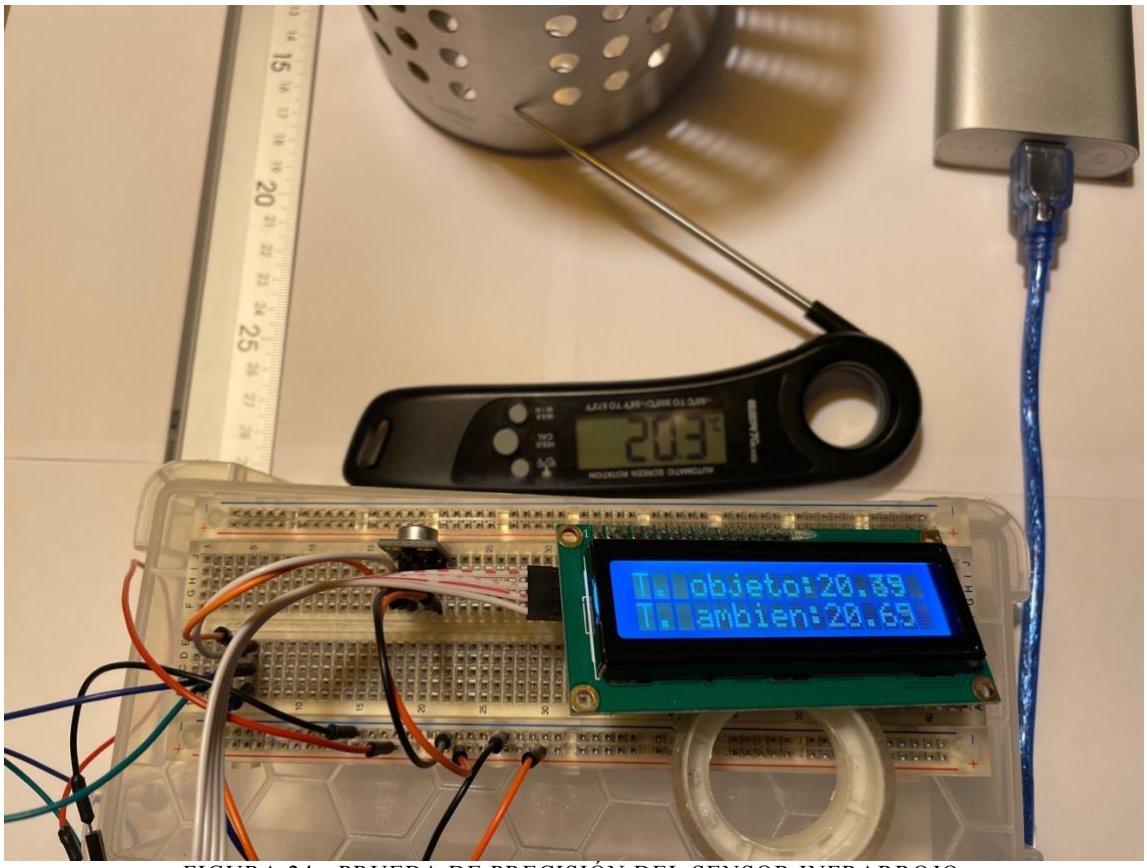


FIGURA 24 - PRUEBA DE PRECISIÓN DEL SENSOR INFRARROJO

4.2.2. PRUEBAS DE PRECISIÓN: SENSOR DE DISTANCIA POR ULTRASONIDOS

Para comprobar la precisión del sensor de distancia por ultrasonidos, se colocó el mismo en una posición ligeramente elevada, tratando de separarlo de la mesa. Tras colocar el sensor a 30 cm de distancia de una caja de cartón, se probó el funcionamiento. Realmente la distancia medida fue prácticamente exacta.

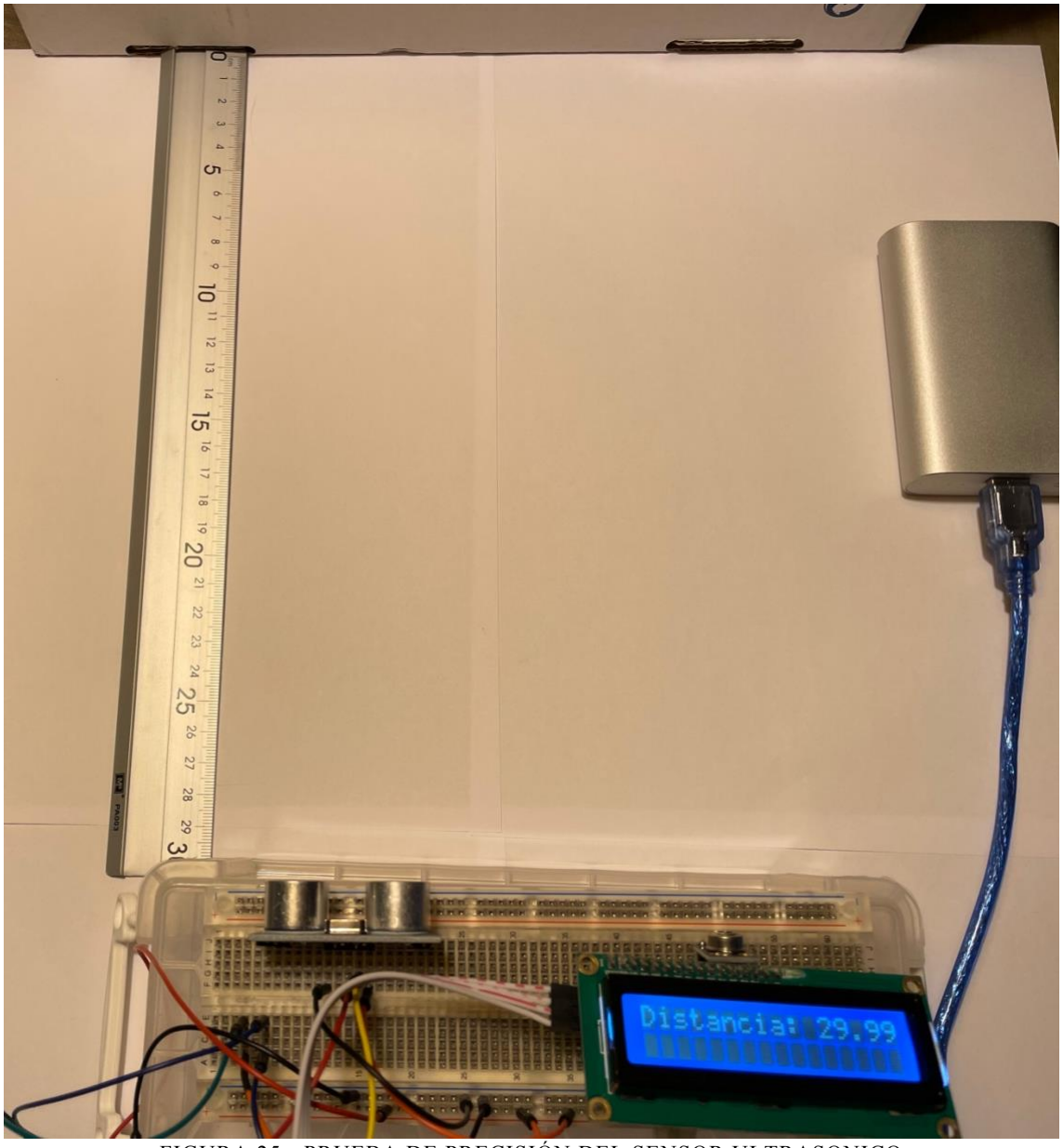


FIGURA 25 - PRUEBA DE PRECISIÓN DEL SENSOR ULTRASONICO

4.3. MONTAJE SOBRE SHIELD

Dado que el montaje de los distintos periféricos de ambos dispositivos, así como de la infinidad de cables que permiten las conexiones, hace que el prototipo sea algo "engorroso", se optó por trasladar todos los componentes a una placa shield. Esta está diseñada para encajar encima de un Arduino UNO, con lo que el prototipo obtenido, resultara en un conjunto más sólido y manejable. Esta placa shield, pensada para un prototipo, no sustituiría a una placa impresa que se diseñaría ad-hoc para una versión comercial, que además debería contar con un empaque exterior diseñado para tal fin, gracias al modelado para una impresora 3D.

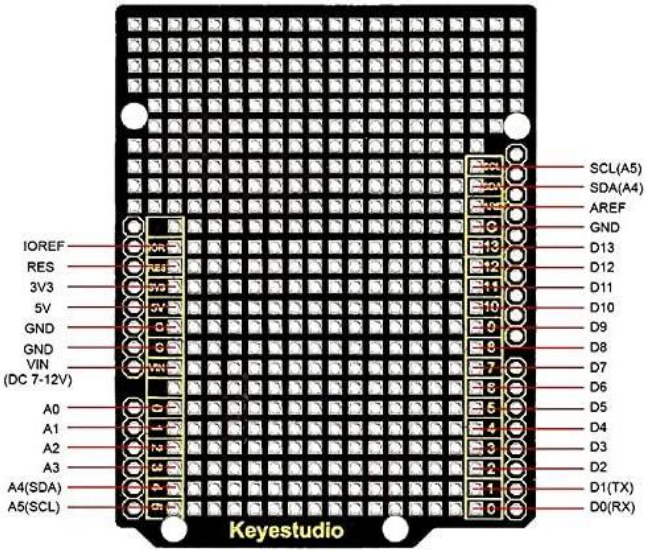


FIGURA 26 - ESQUEMA DE LA PLACA SHIELD

Este modelo de placa shield tiene serigrafiados los puntos que corresponden a puertos I/O, alimentaciones, etc. Para poder trabajar con ella, primeramente tenemos que soldar pines, que nos permitan conectarlo a la placa Arduino.

FIGURA 27 - PINES PARA PLACA SHIELD

Una vez que estos se encuentran soldados, llega el momento de colocar los componentes en una posición que nos permita, de la forma mas limpia y ordenada posible, trazar las pistas. Este modelo de placa shield, no cuenta con ningún modo de trazar esas pistas, con lo que se optó por realizar las conexiones gracias a un juego de pequeñas secciones de cable de aluminio con aislamiento plástico, que nos permitirán rápidamente obtener un prototipo.

4.3.1. PROTOTIPO ELEMENTO DE CONTROL SOBRE PLACA SHIELD

Tras el montaje y soldadura de los distintos componentes, como resistencias, periféricos, diodos, pulsador y conexiones, se obtuvo el siguiente prototipo de elemento de control y visualización. Como se puede apreciar en la imagen, la pantalla LCD queda separada del conjunto y conectada a través de cuatro pines macho que quedan habilitados para tal fin, al igual que en el elemento de control y visualización.

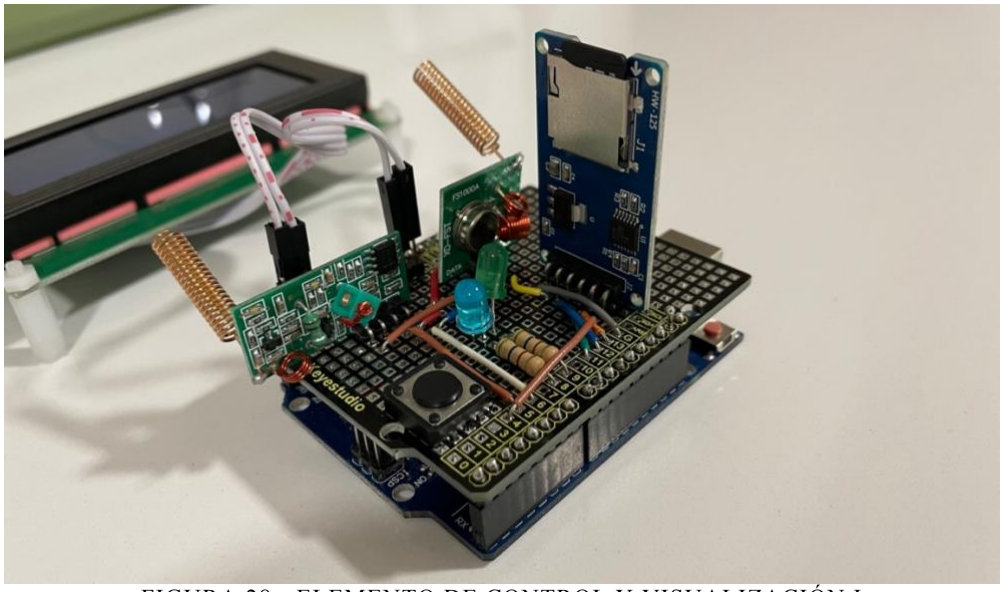


FIGURA 28 - ELEMENTO DE CONTROL Y VISUALIZACIÓN I

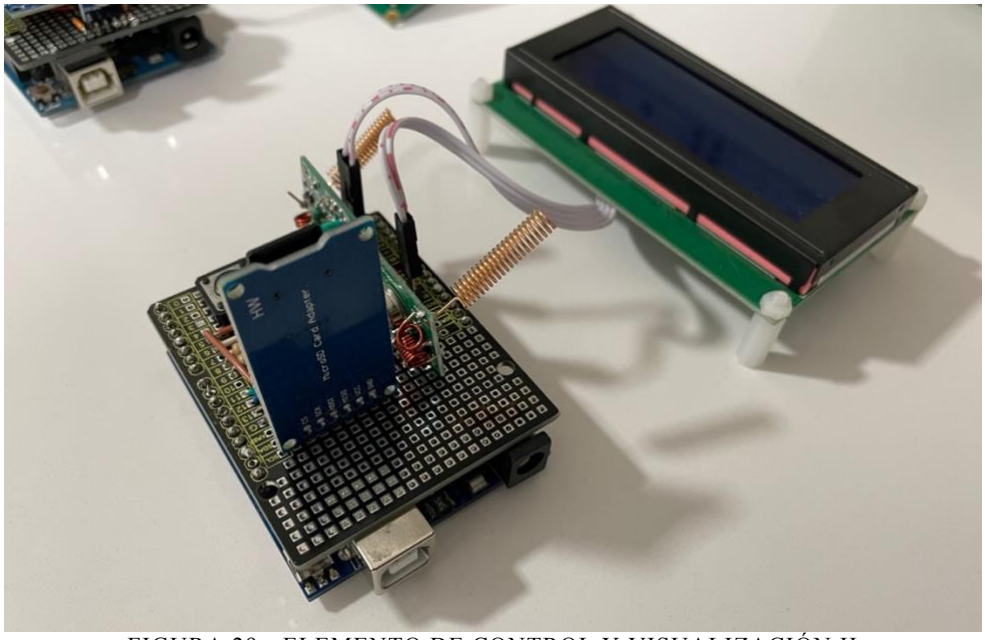


FIGURA 29 - ELEMENTO DE CONTROL Y VISUALIZACIÓN II

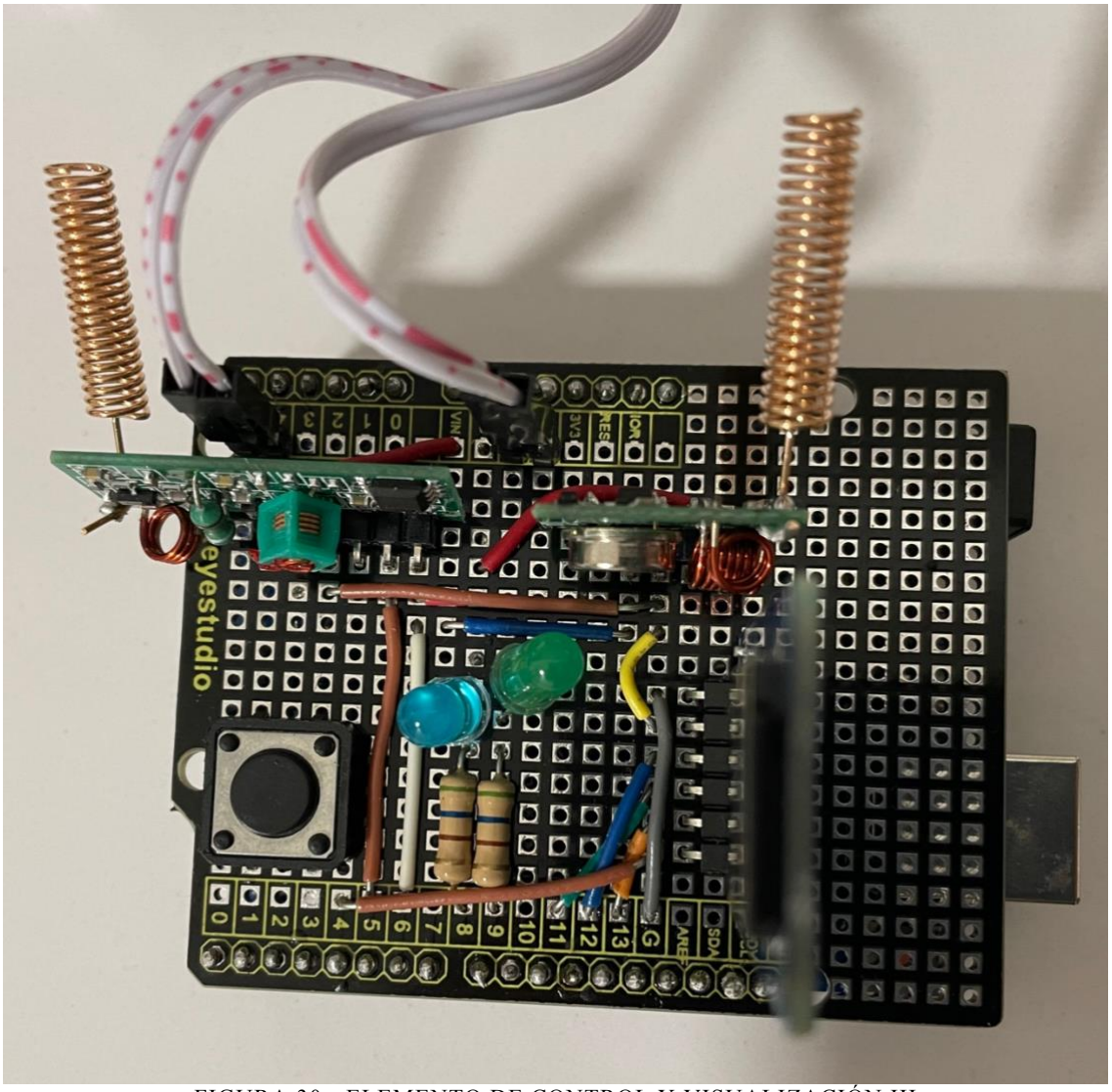


FIGURA 30 - ELEMENTO DE CONTROL Y VISUALIZACIÓN III

4.3.2. PROTOTIPO ELEMENTO DE MEDIDA SOBRE PLACA SHIELD

Tras el montaje y soldadura de los distintos componentes , periféricos, pulsador y conexiones, se obtuvo el siguiente prototipo de elemento de medida. Como se puede apreciar en la imagen, la pantalla LCD queda separada del conjunto y conectada a través de cuatro pines macho que quedan habilitados para tal fin, al igual que en el elemento de control y visualización.

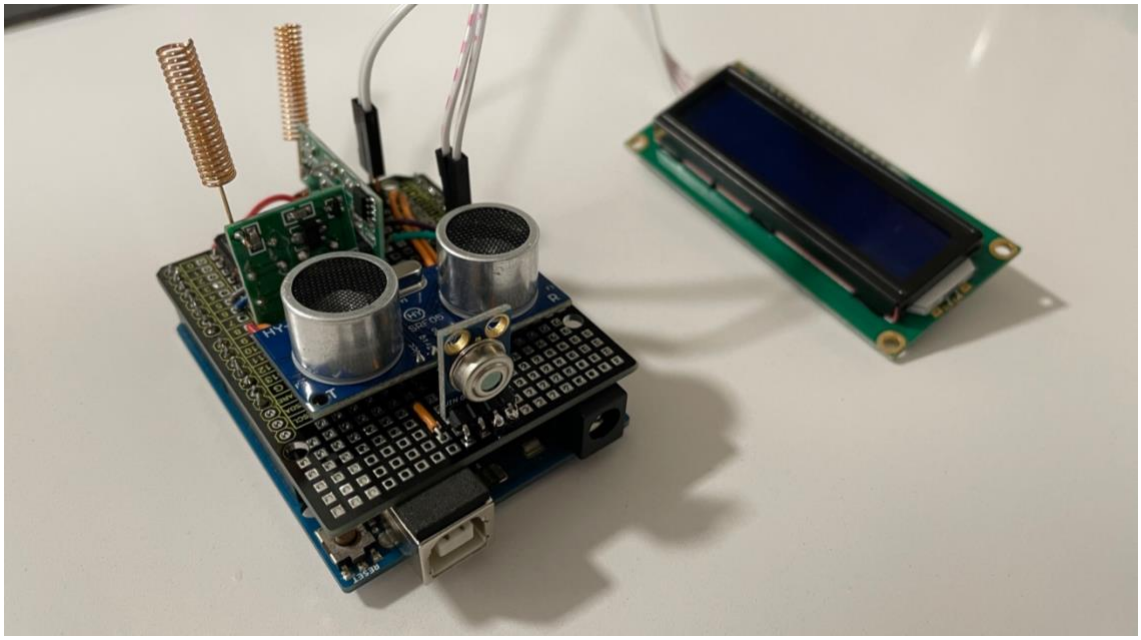


FIGURA 31 - ELEMENTO DE MEDIDA I

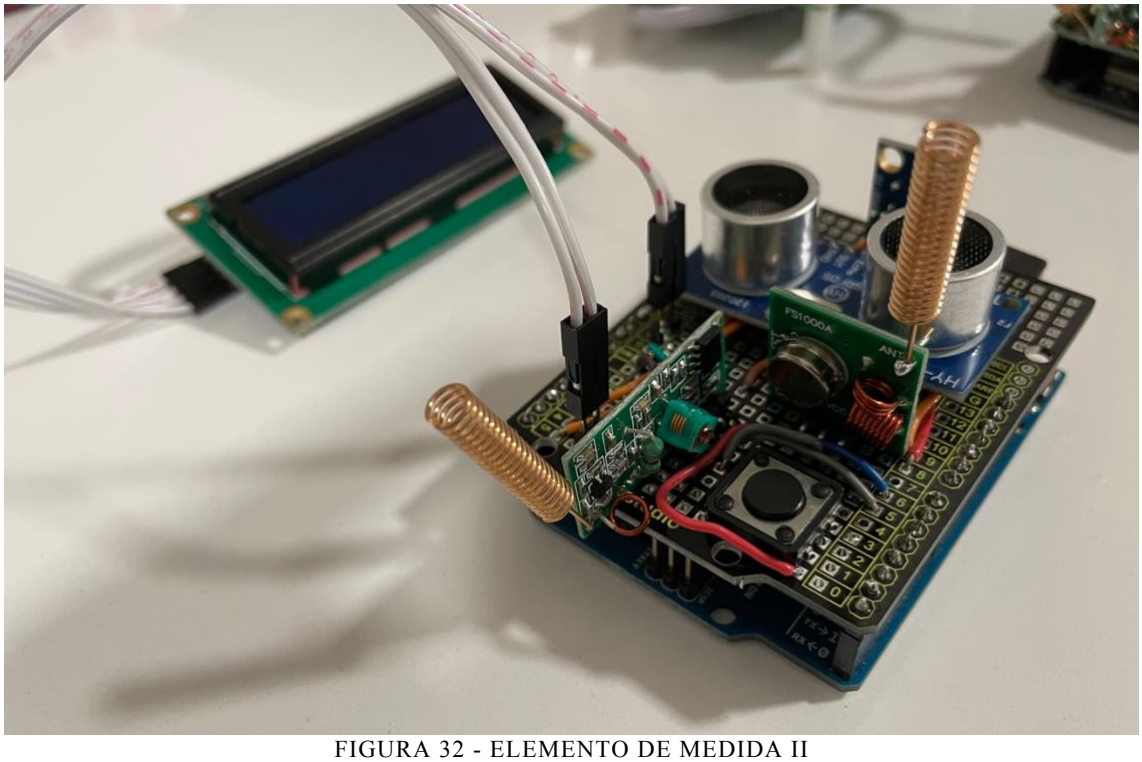


FIGURA 32 - ELEMENTO DE MEDIDA II

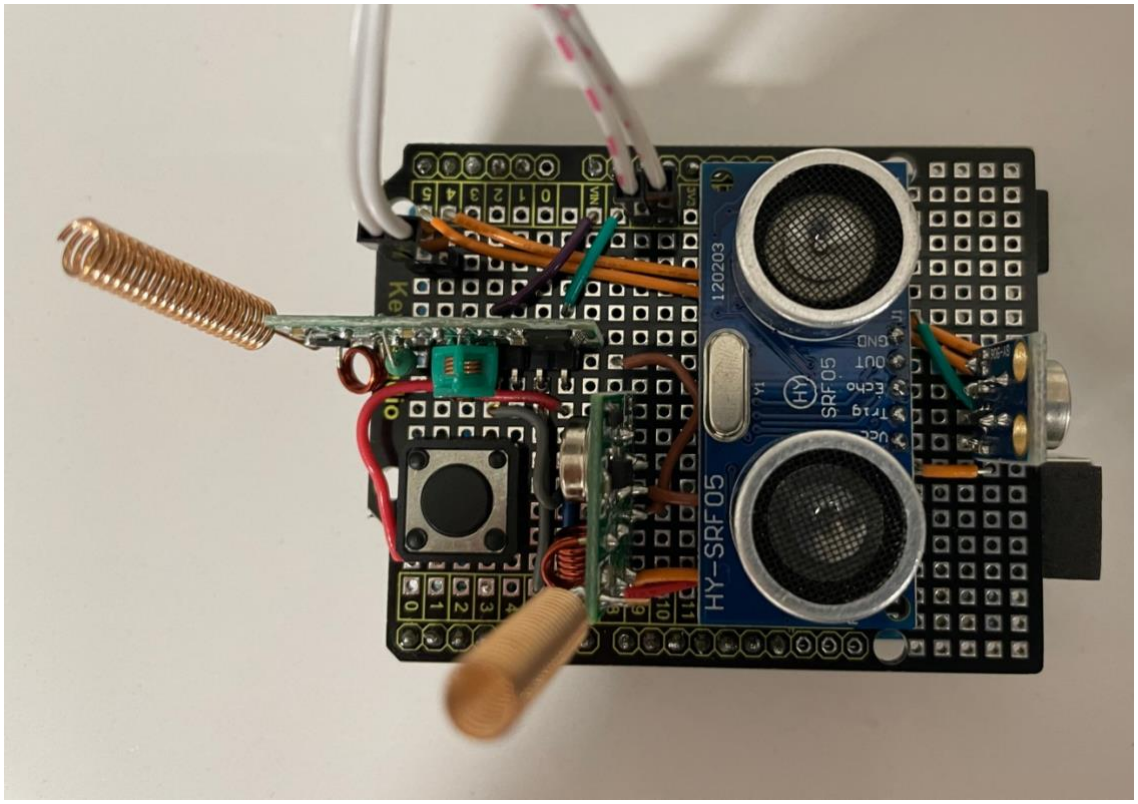


FIGURA 33 - ELEMENTO DE MEDIDA III

4.3. PRUEBA DE LOS PROTOTIPOS

Para probar el correcto desempeño del prototipo, se realizaron pruebas en un entorno real, en este caso, el helicóptero para el cual ha sido diseñado.

Las pruebas consistieron en la revisión de los valores del amortiguador del tren de aterrizaje delantero (NLG, Noise Landing Gear). Se opto por hacer las pruebas sobre el delantero, ya que este permitía tomar imágenes del proceso de forma mas correcta.

FIGURA 34 - HELICOPTERO PARA PRUEBAS DE CAMPO

Recordemos, que la prueba consiste en medir la longitud del stroke del tren de aterrizaje. El Stroke hace referencia a la parte del cilindro que esta dentro del amortiguador, por lo que la prueba consistirá en medir la parte que sobresale (en este caso la medida sobre una regla era de 96 mm).

FIGURA 35 - MEDIDA DEL STROKE DEL NLG

El cilindro entero tiene para este modelo de helicóptero, una longitud de 395 mm, sobre la cual debemos deducir la longitud medida (96 mm), obteniendo un stroke de 299 mm.
Seguidamente se comprobó la temperatura con la sonda fluke.

FIGURA 36 - MEDIDA DE LA TEMPERATURA DEL STROKE

La temperatura del stroke obtenida era de 10,5º centígrados. Posteriormente se comprobó la temperatura ambiental.

FIGURA 37 - MEDIDA DE LA TEMPERATURA AMBIENTAL

Como se puede apreciar, la medida obtenida fue de 11,3º centígrados, lo cual nos indica que la temperatura del stroke y la temperatura ambiental, no difieren en mas de un grado.

A continuación se hizo la prueba del dispositivo. La prueba consistió en colocar el elemento de medida en la posición de medida y posteriormente alimentarlo (momento en el cual este empieza a tomar valores). A continuación, se alimentó el elemento de control y visualización, para posteriormente pulsar su botón y así indicarle que se trata de un NLG (led azul). Acto seguido, se pulsó el botón del elemento de medida, que mandó los parámetros correctamente al elemento de control. Se pudo comprobar tras sucesivas pruebas, que los dispositivos debían encontrarse próximos, ya que de lo contrario en algunas ocasiones no llegaban a transferirse los parámetros.

FIGURA 38 - PRUEBA DEL PROTOTIPO I

FIGURA 39 - PRUEBA DEL PROTOTIPO II

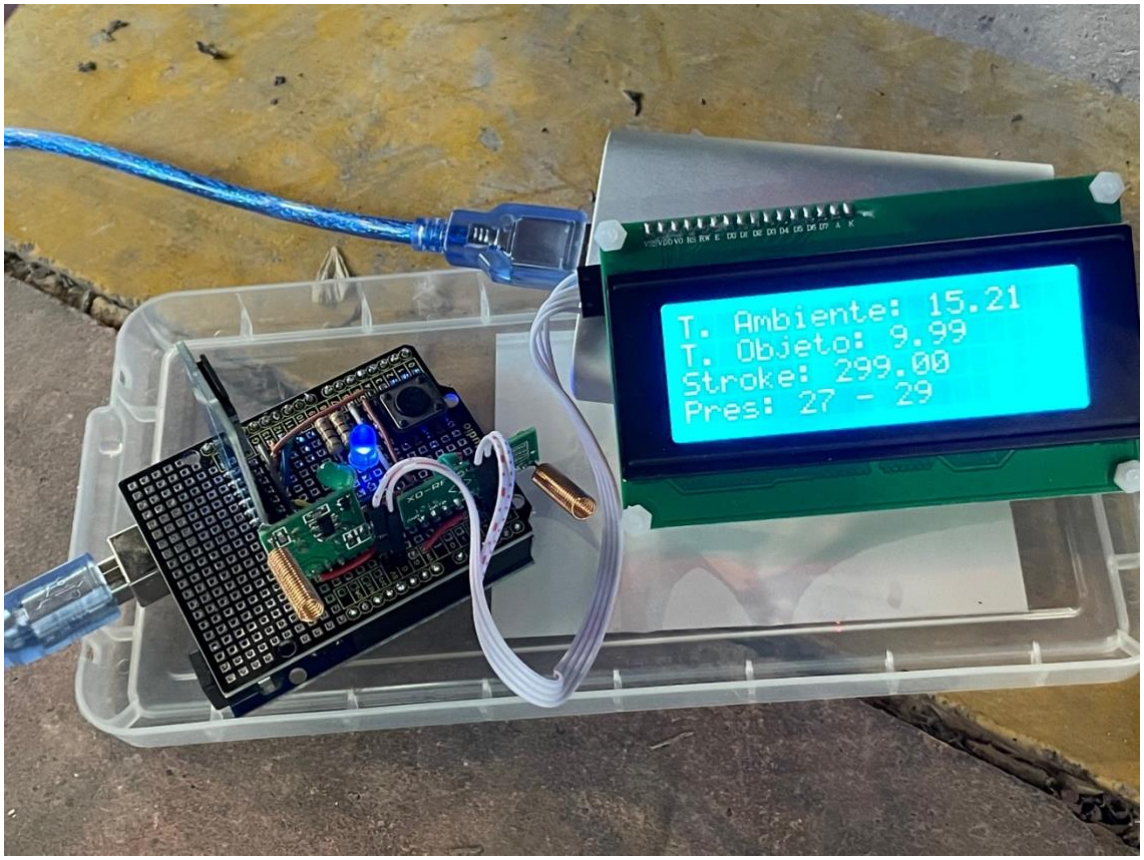


FIGURA 40 - PRUEBA DEL PROTOTIPO III

Los valores obtenidos tras las pruebas con el prototipo fueron de un stroke de 299 (misma longitud que la obtenida con la regla), una temperatura del stroke de 9,99º centígrados (con la sonda fluke la medida tomada fue de 10,5º centígrados), y una temperatura ambiental de 15,21º centígrados. Como se puede apreciar, la única medida que no correspondía con los valores reales es la temperatura ambiental. Tras pruebas más exhaustivas se comprobó, que la temperatura ambiental siempre se incrementaba al operar el aparato con las manos (calor corporal), así como que se incrementaba con las sucesivas pruebas (podría verse afectado por el calor que desprendiera la placa Arduino a medida que esta se va calentando). No obstante, tuvo que ser eliminada la condición por la cual las temperaturas no pueden distar más de 1 grado (se estableció una tolerancia de 10 grados para la demostración).
En cualquier caso al ser exactos los valores de temperatura del stroke y de longitud, se obtuvo unos valores de presión correctos de entre 27 y 29 bares. La comprobación de presión no fue realizada, ya que al no condicionar al prototipo, no da a lugar.

En la siguiente tabla, se muestran los primeros valores de presión para temperaturas de entre 0º y 10º.

Presión	Stroke mínimo	Stroke máximo
25	285,7416667	291,7916667
26	288,725	294,875
27	291,9083333	299,1583333
28	294,8916667	301,2416667
29	297,775	304,225
30	300,5583333	307,1083333

Como se puede apreciar, los valores entre los cuales tenemos 299 mm son los correspondientes a 27, 28 y 29 bares, con lo que los valores mostrados por el prototipo son correctos.
Los valores mostrados en la tabla, son obtenidos de la gráfica del fabricante, con lo cual la precisión de estos valores es relativa.

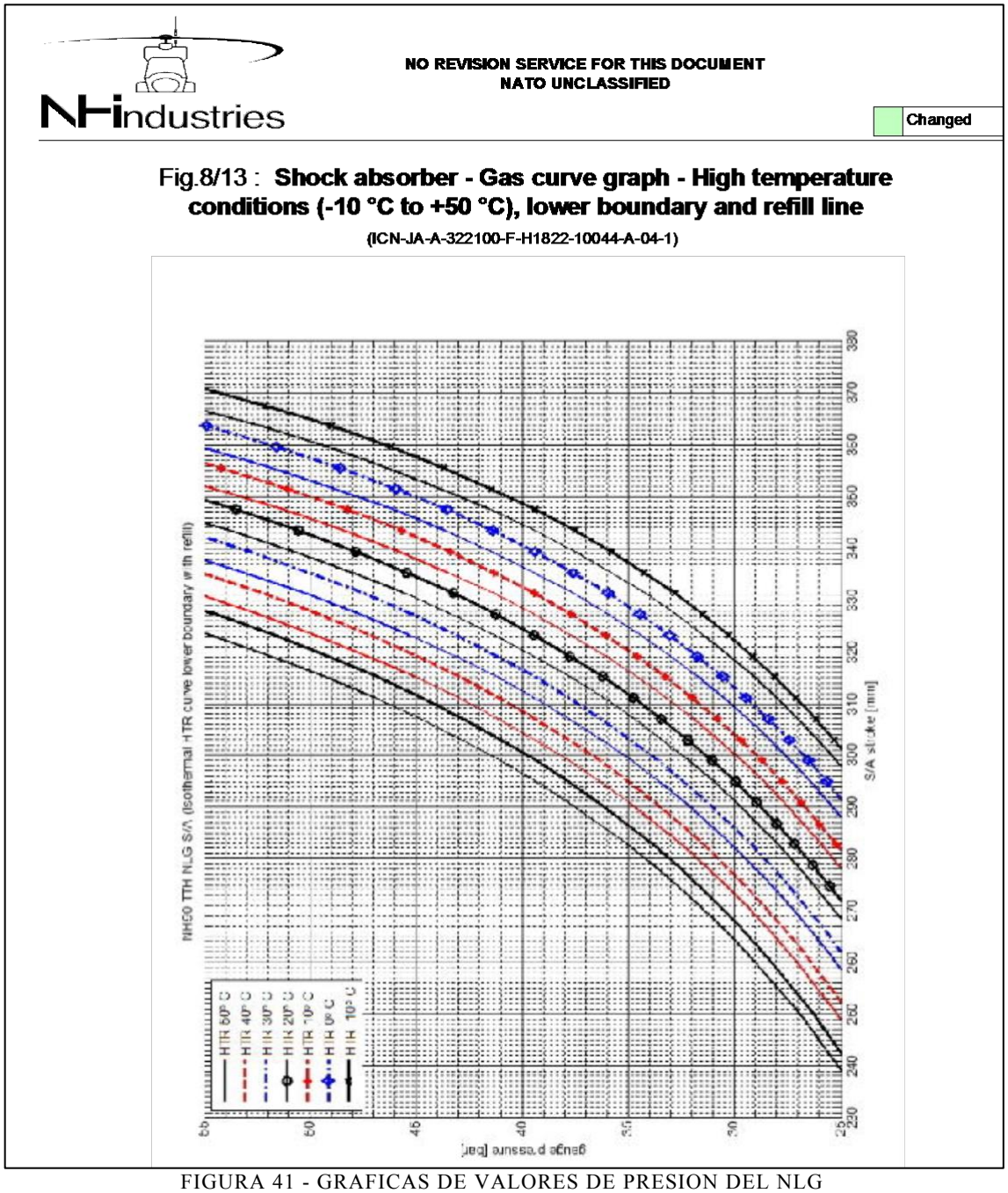


FIGURA 41 - GRAFICAS DE VALORES DE PRESION DEL NLG

En resumen, se puede afirmar que el prototipo no cumple los requerimientos técnicos previos en materia de precisión para los parámetros de temperatura ambiental y temperatura del stroke, si bien, esta última podría ser salvada con un contacto total entre la sonda de temperatura y el stroke (la morfología de Arduino no lo permite al sobresalir el puerto USB). Los valores de temperatura ambiental están bastante alejados de los correctos. Por otra parte, si que se cumplieron los requerimientos técnicos previos en materia de precisión para la medida de la longitud del stroke. En cuanto al almacenamiento de los datos en la tarjeta SD, fue implementado aunque desactivado a través de indicadores de comentario en el código, ya que el peso en memoria que tenía su implementación, no permitía el empleo de Arduino UNO.

CAPITULO 5: CONCLUSIONES Y LÍNEAS FUTURAS DE DESARROLLO

En conclusión podemos afirmar, que los periféricos empleados pueden ser suficientes si trabajan en las condiciones ideales de funcionamiento. Es decir:

- Para el periférico de medida mediante ultrasonidos, su funcionamiento a sido correcto si existía una buena alineación. Mediante el empleo de una carcasa externa que lo haga encajar en el cilindro, esta alineación sería perfecta. En cualquier caso, en las pruebas de campo, la medida fue excepcional de igual modo.
- Para el periférico de medición de temperatura mediante infrarrojos, se pudo comprobar que, si bien en las pruebas en banco de trabajo se comportó con precisión, en la prueba de campo hubo variaciones ($\pm$ 0,5º) que nos impiden cumplir con los requerimientos técnicos. Para la medición de temperatura ambiental, el calor generado por nuestras manos y por Arduino, impedía tomar valores precisos (variaciones de incluso 6º).
- Los Emisores y receptores de RF, aunque cumplieron la función, demostraron no ser totalmente confiables, ya que sufrían perdidas de las transmisiones en algunos casos.
- El funcionamiento del almacenamiento en tarjeta SD funciono correctamente de forma aislada, si bien no pudo ser implementado por escasez de memoria de la placa Arduino de la cual se disponía.

La programación y el concepto de diseño fueron correctos y no se apreciaron fallos destacables, aunque si que es cierto que mediante la elección de otro modelo de placa, y empleando otra solución distinta a la placa shield utilizada, se podrían cumplir los requisitos técnicos previos. De hecho, algunas de estas modificaciones se plantean en las líneas futuras de desarrollo.
En conclusión, no puede garantizarse con el prototipo desarrollado, que puedan garantizarse las precisiones que un estándar aeronáutico necesita.

5.1. LÍNEAS FUTURAS DE DESARROLLO

Las líneas futuras de desarrollo son amplias. Primeramente, si nos ceñimos exclusivamente al prototipo fabricado, es evidente que la principal mejora sería emplear una placa Arduino sin tanta limitación de memoria, como por ejemplo un Arduino MKR1000. Este nos permitiría implementar sin limitaciones, todos los valores de temperaturas para ambos tipos de amortiguadores (así como el almacenamiento SD). Además, al contar con una interfaz WIFI de serie, podríamos evitar el empleo de emisores y receptores RF, con lo que podríamos ahorrar mucho espacio, a la vez que conseguimos una comunicación mas confiable y con mayor alcance.
En cuanto a la temperatura ambiental, habría que emplear un sensor de temperatura distinto al que incorpora el sensor de infrarrojos. Esto nos permitiría tener un valor más fiable.
Por otra parte, la situación ideal seria una en la cual pudiéramos tomar los valores de presión directamente con el prototipo. No existe ningún periférico para Arduino que nos permita medir presiones de estas magnitudes, no obstante con un mayor presupuesto se podría adquirir alguna sonda industrial diseñada para tal fin, que nos permitiera mediante un puerto serie leer los valores. Otra posibilidad podría ser emplear un manómetro digital, que siendo desmontado, podría permitirnos leer los valores.
También, la placa shield debería ser sustituida por una placa PCB diseñada específicamente para este fin. Además, debería dotarse al proyecto de una envuelta (diseñada en 3D) que permita que el elemento de medida encaje en la forma cilíndrica del amortiguador, de modo que mediante imanes o algún tipo de anclaje, puede sujetarse el solo en la posición perfecta para la toma de valores. Esto permitiría que la sonda infrarroja entre en casi en contacto con el stroke. Con esto conseguiríamos una toma de valores ideal para longitud y temperatura del stroke.
Si y solo si se consiguieran solventar todas estas mejoras necesarias, podría ser empleado el prototipo con el suficiente rigor como para ser empleado en el campo de la aeronáutica.

Finalmente decir, que posiblemente la mayor mejora que podría presentarse va al origen de su concepción como herramienta de mantenimiento. En una aeronave como el NH90, donde todos los aspectos de la aeronave se controlan de forma electrónica, desde el control de la combustión, los controles de vuelo Fly-By-Wire desde los FCS's (Flight Control System), etc… el sistema debería de contar con un dispositivo embarcado como el elemento de medida que formara parte del amortiguador, de modo que estos parámetros se entregaran on Fly a las computadoras de la aeronave. De este modo seria una comprobación mas de las realizadas por los computadores en los test realizados tanto de el momento del encendido (PBit) como constantemente durante el vuelo (CBit). Este parámetro podría ser mostrado por pantalla al piloto en todo momento.

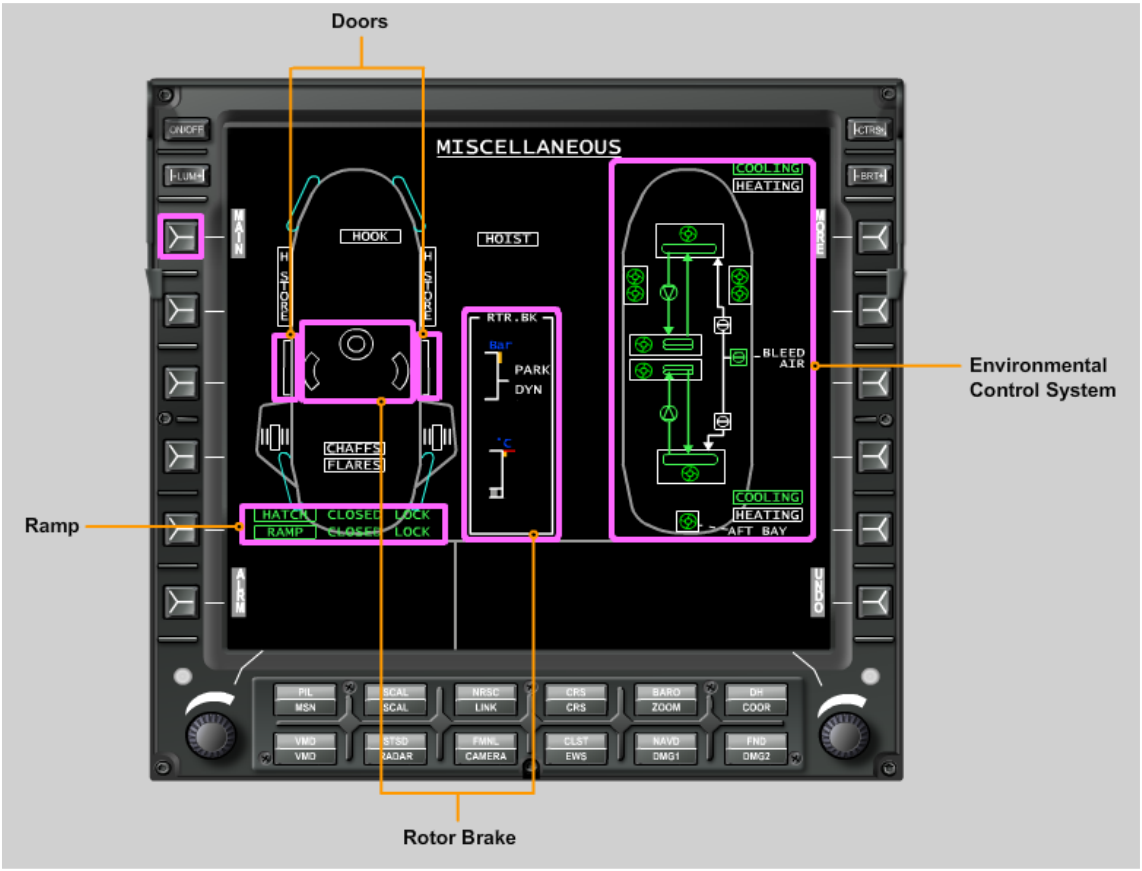
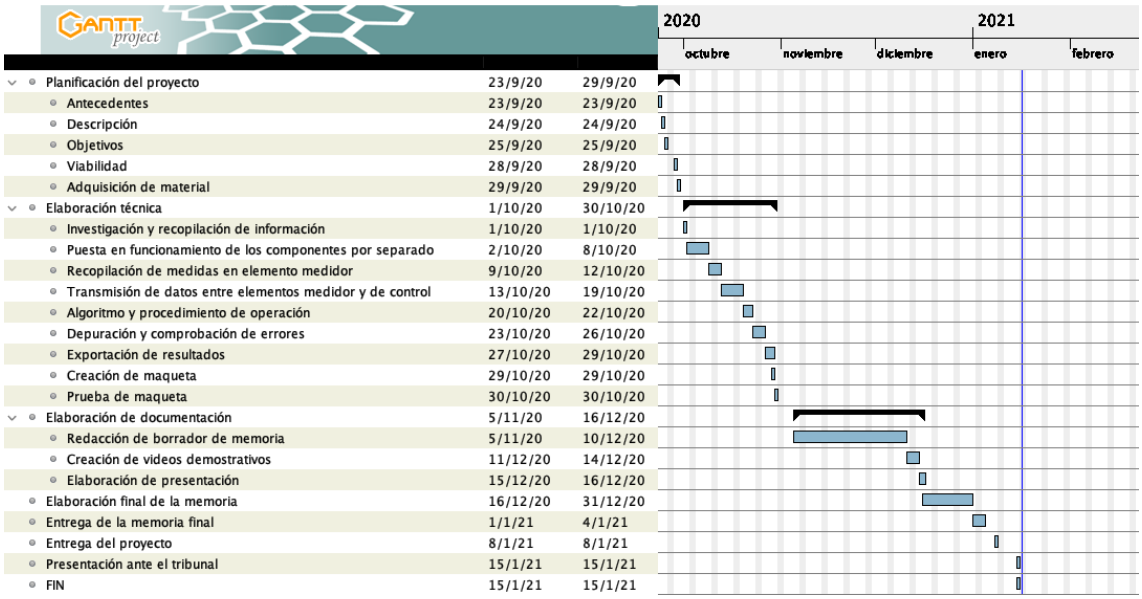


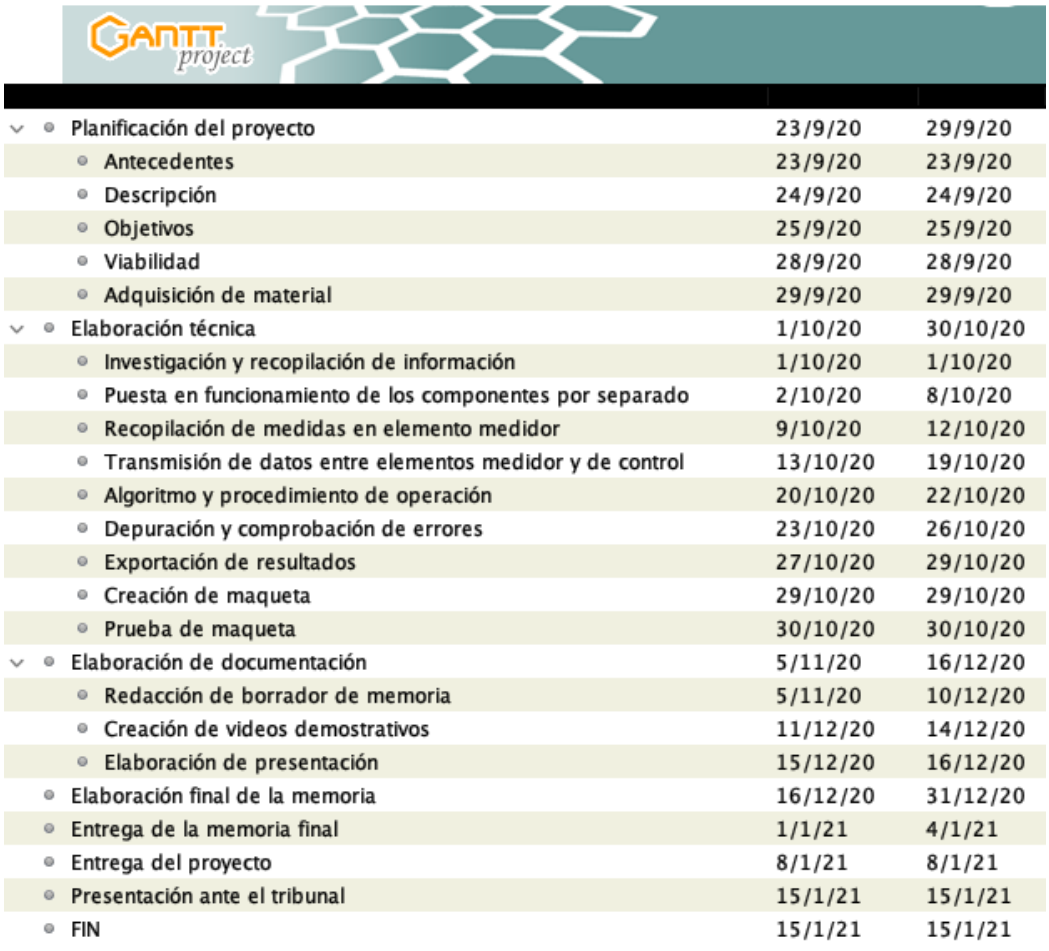
FIGURA 42 - VMD (VEHICULE MANAGEMENT DISPLAY) DE UN NH90

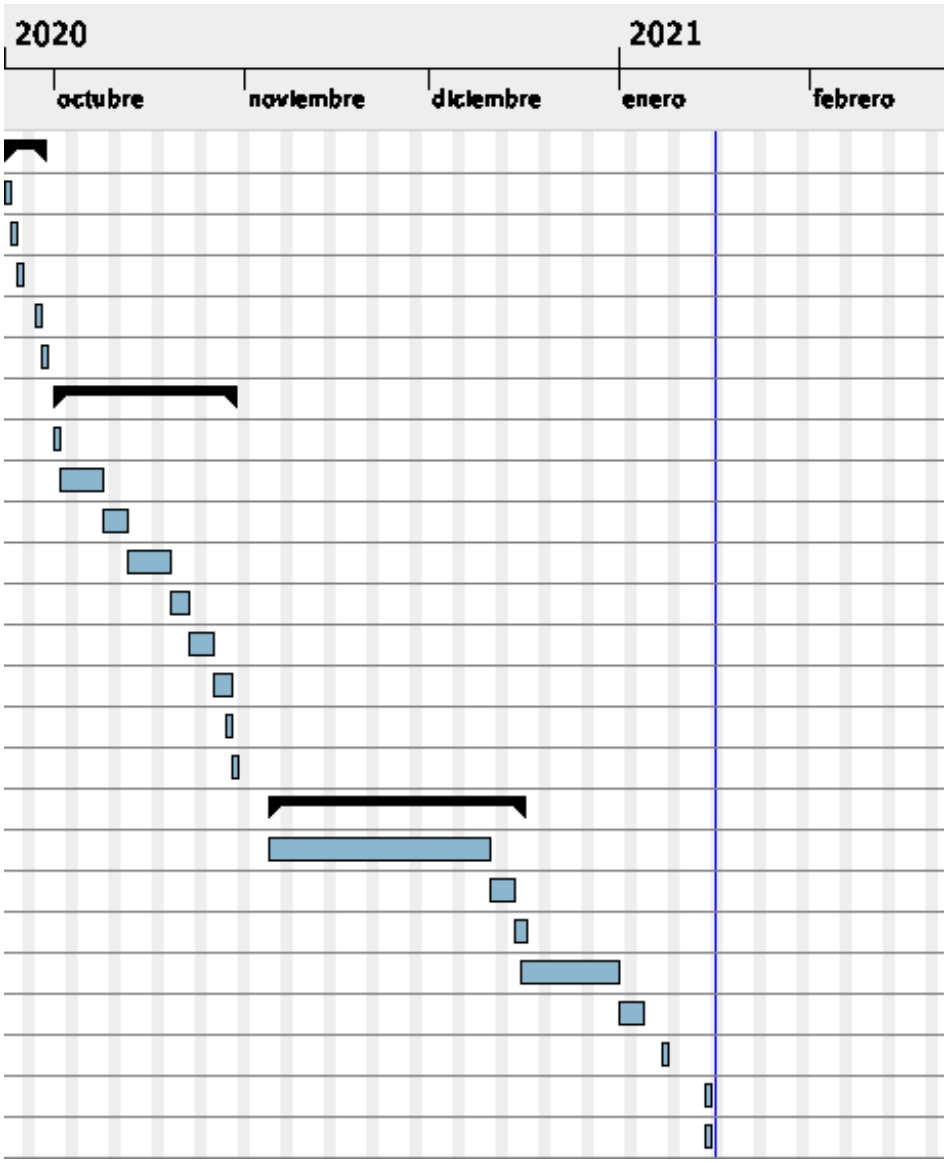
CAPITULO 6: PLANIFICACIÓN TEMPORAL

La planificación del proyecto se representa mediante el siguiente diagrama de Gantt:



Se añaden las siguientes ampliaciones de la imagen anterior, donde puede resultar más sencilla su lectura.




2020
2021
octubre
noviembre
diciembre
enero
febrero

BIBLIOGRAFÍA

[1] *Infodefensa* [en línea] [fecha de publicación: 27 de diciembre de 2013]. Disponible en: http://www.infodefensa.com/es/2013/12/27/noticia-indra-adjudica-sistema-automatico-mantenimiento-helicopteros-millones.html

[2] *Comercial RS* [en línea] [fecha de consulta: 1 de diciembre de 2020]. Disponible en: https://es.rs-online.com/web/

[3] *What is Arduino?* [en línea] [fecha de consulta: 1 de diciembre de 2020]. Disponible en: https://www.arduino.cc/en/Guide/Introduction

[4] TAPIA AYALA, Carlos Hipólito y MANZANO YUPA, Héctor Mauricio. *Evaluación de la plataforma Arduino e implementación de un sistema de control de posición horizontal* [en línea]. Tesis: Universidad Politécnica Salesiana de Ecuador, 2013. Disponible en: https://dspace.ups.edu.ec/bitstream/123456789/5522/1/UPS-GT000511.pdf

[5] LLAMAS, Luis. El bus I2C en Arduino [en línea]. *Luis Llamas*. (Fecha de publicación: 18 de mayo de 2016). [Consulta: 1 de diciembre de 2020]. Disponible en: https://www.luisllamas.es/arduino-i2c/

[6] LLAMAS, Luis. El bus SPI en Arduino [en línea]. *Luis Llamas*. (Fecha de publicación: 14 de mayo de 2016). [Consulta: 1 de diciembre de 2020]. Disponible en: https://www.luisllamas.es/arduino-spi/

[7] CRESPO, José Enrique. Memoria Arduino [en línea]. *Aprendiendo Arduino*. (Fecha de publicación: 8 de noviembre de 2016). [Consulta: 1 de diciembre de 2020]. Disponible en: https://aprendiendoarduino.wordpress.com/2016/11/08/memoria-arduino/

[8] Ingeniería de Microsistemas Programados S.L. SRF05 [Datasheet]. Disponible en: http://riobotics-test.weebly.com/uploads/9/3/0/9/9309609/medidor_ultrasonico_srf05.pdf

[9] Airbus Helicopters España S.A. Landing Gears - Introduction of high and low temperature conditions curves. SIIP-JA-A-32-00-00-06S-A-A-002 [Instrucción técnica]. Versión 002. (2019)

[10] Airbus Helicopters España S.A. Main Landing Gear (MLG) retraction actuator/shock absorber - Mandatory inspections. JA-A-04-40-32-11A- 000A-A [Instrucción técnica]. Versión 001. (2019)

[11] Airbus Helicopters España S.A. Nose Landing Gear (NLG) shock strut - Mandatory inspections. JA-A-04-40-32-21A-000A-A [Instrucción técnica]. Versión 001. (2019)

[12] Airbus Helicopters España S.A. Flight maneuvering limitations - Description (for aircrew). JA-A-15-15-30-00A-043A-A [Instrucción técnica]. Versión 001. (2019)

[13] Airbus Helicopters España S.A. Main Landing Gear (MLG) retraction actuator/shock absorber - Serviceable check. JA-A-32-11-01-00A- 200D-A [Instrucción técnica]. Versión 001. (2019)

[14] Airbus Helicopters España S.A. Nose Landing Gear (NLG) shock strut – Serviceable check. JA-A-32-21-01-00A-200D-A [Instrucción técnica]. Versión 001. (2019)

[15] Airbus Helicopters España S.A. Every 30 days – Scheduled inspections. JA-A-05-42-00-02A-281A-A [Instrucción técnica]. Versión 001. (2019)

ANEXOS

CÓDIGO FUENTE: ELEMENTO DE MEDIDA

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <RunningMedian.h>
#include <Adafruit_MLX90614.h>
#include <RH_ASK.h>
#include <SPI.h>
#include <RHreliableDatagram.h>


#define RX_PIN 5
#define TX_PIN 6
#define SPEED 2000
#define CLIENT_ADDRESS 1
#define SERVER_ADDRESS 2


LiquidCrystal_I2C lcd(0x27,16,2);
RunningMedian mediana = RunningMedian(9);
Adafruit_MLX90614 TemperaturaIR = Adafruit_MLX90614();
float temperatura = 0;
float humedad = 0;
int pulsador = 3;
int trigger = 8;
int echo = 9;
int i = 0;
float d=0;
int presion=50;
char *msg = "                                        ";
RH_ASK driver(SPEED, RX_PIN, TX_PIN);
RHReliableDatagram manager(driver, CLIENT_ADDRESS);
String floatToString(float,int=8,int=2,boolean=true);


void setup()
{
  lcd.init();
  delay(1000);
  lcd.backlight();
  Serial.begin(9600);
  pinMode(pulsador, INPUT_PULLUP);

  //Establezco los dos pines del sensor de ultrasonidos
  pinMode(trigger, OUTPUT);
  pinMode(echo, INPUT);
  digitalWrite(trigger, LOW);

  //Inicializo el sensor de temperaturaIR
  TemperaturaIR.begin();
```

```
  if (!driver.init()){
    lcd.setCursor(0,0);
    lcd.print("Fallo de iniciacion");
  }
}



void loop()
{
  digitalWrite(trigger, LOW);
  delayMicroseconds(5);    //Estabiliza el sensor.
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);    //Enviamos un pulso de 10 us.
  digitalWrite(trigger, LOW);

  //Relleno el Runningmedian para calcular la mediana de las medidas
tomadas.
  for(int i=0; i<10; i++){
    mediana.add(MedirTiempo());
  }
  d = (0.034 * mediana.getMedian()) / 0.2;                //Escalamos
el tiempo a una distancia en cm

  //Al pulsar el boton, envia los parametros al receptor:
  if(digitalRead(pulsador) == LOW)
  {
    TomarParametros();
    delay(2000);
    EnviarParametros();
  }
}


int MedirTiempo(){
  int t=0;   //tiempo que tarda en llegar el eco
  digitalWrite(trigger, LOW);
  delayMicroseconds(5);    //Estabiliza el sensor.
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);    //Enviamos un pulso de 10 us.
  digitalWrite(trigger, LOW);
  t = pulseIn(echo, HIGH);    //Obtenemos el ancho del pulso
  delay(100);
  return(t);
}


//Recopila los datos y conforma el mensaje
//Temperatura   amiental;temperatura   del   objeto;distancia;presion
simulada (en bares)
void TomarParametros(){
```

```
  String flotante = floatToString(TemperaturaIR.readAmbientTempC(),6)
+ ";" + floatToString(TemperaturaIR.readObjectTempC(),6) + ";" + (int)
d + ";" + presion + ";";
  flotante.toCharArray(msg, flotante.length()+1);
  lcd.setCursor(0,0);
  lcd.print(" Tomando Datos  ");
  lcd.setCursor(0,1);
  lcd.print("                ");
}

//Envia los parametros con el emisor RF
void EnviarParametros(){
  lcd.setCursor(0,1);
  lcd.print("    Enviando    ");
  manager.sendtoWait((uint8_t *)msg, strlen(msg), SERVER_ADDRESS);

  if(driver.waitPacketSent()==true){
    lcd.setCursor(0,1);
    lcd.print(" Datos enviados ");
    lcd.setCursor(0,0);
    lcd.print("                ");
  }
  else{
    lcd.setCursor(0,1);
    lcd.print(" Fallo de envio ");
    lcd.setCursor(0,0);
    lcd.print("                ");
  }

  delay(200);
}

// Convierte un float en una cadena.
String floatToString( float n, int l, int d, boolean z){
  char c[l+1];
  String s;
  dtostrf(n,l,d,c);
  s=String(c);

  if(z){
    s.replace(" ","");
  }
  return s;
}
```

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <RH_ASK.h>
#include <RHReliableDatagram.h>
#include <SPI.h>
#include <SD.h>


#define RX_PIN 5
#define TX_PIN 6
#define SPEED 2000
#define CLIENT_ADDRESS 1
#define SERVER_ADDRESS 2
#define ToleranciaDiferencialDeTemperatura 10
#define PinLedAzul 9
#define PinLedVerde 8
#define pulsador 3


//    Valores para MLG
float MLG_menos10_Stroke_minimo[] = {227.5, 228.5, 229, 230, 230.7,
231.5, 232.2, 233, 233.6, 234.1, 234.9, 235.5, 236, 236.5, 237.1,
237.9, 238.2, 238.9, 239.2, 239.9, 240.3, 240.9, 241.3, 241.8, 242.2,
242.5, 243, 243.4, 243.9, 244.2, 244.6, 245, 245.3, 245.8, 246, 246.6,
246.9, 247.1, 247.3, 247.6, 248, 248.3, 248.6, 248.9, 249.2, 249.4,
249.7, 249.9, 250.1, 250.5, 250.7, 251, 251.3, 251.5, 251.7, 251.9,
252.1, 252.4, 252.6, 252.8, 253};
float MLG_menos10_Stroke_maximo[] = {235.1, 236.1, 236.6, 237.6,
238.3, 239.1, 239.8, 240.6, 241.2, 241.7, 242.5, 243.1, 243.6, 244.1,
244.7, 245.5, 245.8, 246.5, 246.8, 247.5, 247.9, 248.5, 248.9, 249.4,
249.8, 250.1, 250.6, 251, 251.5, 251.8, 252.2, 252.6, 252.9, 253.4,
253.6, 254.2, 254.5, 254.7, 254.9, 255.2, 255.6, 255.9, 256.2, 256.5,
256.8, 257, 257.3, 257.5, 257.7, 258.1, 258.3, 258.6, 258.9, 259.1,
259.3, 259.5, 259.7, 260, 260.2, 260.4, 260.6};
float MLG_cero_Stroke_minimo[] = {222.2116667, 223.2233333, 223.735,
224.7466667, 225.4583333, 226.27, 226.9816667, 227.7933333, 228.405,
228.9166667, 229.7283333, 230.34, 230.8516667, 231.3633333, 231.975,
232.7866667, 233.0983333, 233.81, 234.1216667, 234.8333333, 235.245,
235.8566667, 236.2683333, 236.78, 237.1916667, 237.5033333, 238.015,
238.4266667, 238.9383333, 239.25, 239.6616667, 240.0733333, 240.385,
240.8966667, 241.1083333, 241.72, 242.0316667, 242.2433333, 242.455,
242.7666667, 243.1783333, 243.49, 243.8016667, 244.1133333, 244.425,
244.6366667, 244.9483333, 245.16, 245.3716667, 245.7833333, 245.995,
246.3066667, 246.6183333, 246.83, 247.0416667, 247.2533333, 247.465,
247.7766667, 247.9883333, 248.2, 248.4116667};
float MLG_cero_Stroke_maximo[] = {229.8116667, 230.8233333, 231.335,
232.3466667, 233.0583333, 233.87, 234.5816667, 235.3933333, 236.005,
236.5166667, 237.3283333, 237.94, 238.4516667, 238.9633333, 239.575,
240.3866667, 240.6983333, 241.41, 241.7216667, 242.4333333, 242.845,
243.4566667, 243.8683333, 244.38, 244.7916667, 245.1033333, 245.615,
246.0266667, 246.5383333, 246.85, 247.2616667, 247.6733333, 247.985,
248.4966667, 248.7083333, 249.32, 249.6316667, 249.8433333, 250.055,
```

```
250.3666667, 250.7783333, 251.09, 251.4016667, 251.7133333, 252.025,
252.2366667, 252.5483333, 252.76, 252.9716667, 253.3833333, 253.595,
253.9066667, 254.2183333, 254.43, 254.6416667, 254.8533333, 255.065,
255.3766667, 255.5883333, 255.8, 256.0116667};
float MLG_mas10_Stroke_minimo[] = {216.9233333, 217.9466667, 218.47,
219.4933333, 220.2166667, 221.04, 221.7633333, 222.5866667, 223.21,
223.7333333, 224.5566667, 225.18, 225.7033333, 226.2266667, 226.85,
227.6733333, 227.9966667, 228.72, 229.0433333, 229.7666667, 230.19,
230.8133333, 231.2366667, 231.76, 232.1833333, 232.5066667, 233.03,
233.4533333, 233.9766667, 234.3, 234.7233333, 235.1466667, 235.47,
235.9933333, 236.2166667, 236.84, 237.1633333, 237.3866667, 237.61,
237.9333333, 238.3566667, 238.68, 239.0033333, 239.3266667, 239.65,
239.8733333, 240.1966667, 240.42, 240.6433333, 241.0666667, 241.29,
241.6133333, 241.9366667, 242.16, 242.3833333, 242.6066667, 242.83,
243.1533333, 243.3766667, 243.6, 243.8233333};
float MLG_mas10_Stroke_maximo[] = {224.5233333, 225.5466667, 226.07,
227.0933333, 227.8166667, 228.64, 229.3633333, 230.1866667, 230.81,
231.3333333, 232.1566667, 232.78, 233.3033333, 233.8266667, 234.45,
235.2733333, 235.5966667, 236.32, 236.6433333, 237.3666667, 237.79,
238.4133333, 238.8366667, 239.36, 239.7833333, 240.1066667, 240.63,
241.0533333, 241.5766667, 241.9, 242.3233333, 242.7466667, 243.07,
243.5933333, 243.8166667, 244.44, 244.7633333, 244.9866667, 245.21,
245.5333333, 245.9566667, 246.28, 246.6033333, 246.9266667, 247.25,
247.4733333, 247.7966667, 248.02, 248.2433333, 248.6666667, 248.89,
249.2133333, 249.5366667, 249.76, 249.9833333, 250.2066667, 250.43,
250.7533333, 250.9766667, 251.2, 251.4233333};
float MLG_mas20_Stroke_maximo[] = {219.235, 220.27, 220.805, 221.84,
222.575, 223.41, 224.145, 224.98, 225.615, 226.15, 226.985, 227.62,
228.155, 228.69, 229.325, 230.16, 230.495, 231.23, 231.565, 232.3,
232.735, 233.37, 233.805, 234.34, 234.775, 235.11, 235.645, 236.08,
236.615, 236.95, 237.385, 237.82, 238.155, 238.69, 238.925, 239.56,
239.895, 240.13, 240.365, 240.7, 241.135, 241.47, 241.805, 242.14,
242.475, 242.71, 243.045, 243.28, 243.515, 243.95, 244.185, 244.52,
244.855, 245.09, 245.325, 245.56, 245.795, 246.13, 246.365, 246.6,
246.835};
float MLG_mas20_Stroke_minimo[] = {211.635, 212.67, 213.205, 214.24,
214.975, 215.81, 216.545, 217.38, 218.015, 218.55, 219.385, 220.02,
220.555, 221.09, 221.725, 222.56, 222.895, 223.63, 223.965, 224.7,
225.135, 225.77, 226.205, 226.74, 227.175, 227.51, 228.045, 228.48,
229.015, 229.35, 229.785, 230.22, 230.555, 231.09, 231.325, 231.96,
232.295, 232.53, 232.765, 233.1, 233.535, 233.87, 234.205, 234.54,
234.875, 235.11, 235.445, 235.68, 235.915, 236.35, 236.585, 236.92,
237.255, 237.49, 237.725, 237.96, 238.195, 238.53, 238.765, 239,
239.235};
float MLG_mas30_Stroke_minimo[] = {206.3466667, 207.3933333, 207.94,
208.9866667, 209.7333333, 210.58, 211.3266667, 212.1733333, 212.82,
213.3666667, 214.2133333, 214.86, 215.4066667, 215.9533333, 216.6,
217.4466667, 217.7933333, 218.54, 218.8866667, 219.6333333, 220.08,
220.7266667, 221.1733333, 221.72, 222.1666667, 222.5133333, 223.06,
223.5066667, 224.0533333, 224.4, 224.8466667, 225.2933333, 225.64,
226.1866667, 226.4333333, 227.08, 227.4266667, 227.6733333, 227.92,
228.2666667, 228.7133333, 229.06, 229.4066667, 229.7533333, 230.1,
230.3466667, 230.6933333, 230.94, 231.1866667, 231.6333333, 231.88,
```

```
232.2266667, 232.5733333, 232.82, 233.0666667, 233.3133333, 233.56,
233.9066667, 234.1533333, 234.4, 234.6466667};
float MLG_mas30_Stroke_maximo[] = {213.9466667, 214.9933333, 215.54,
216.5866667, 217.3333333, 218.18, 218.9266667, 219.7733333, 220.42,
220.9666667, 221.8133333, 222.46, 223.0066667, 223.5533333, 224.2,
225.0466667, 225.3933333, 226.14, 226.4866667, 227.2333333, 227.68,
228.3266667, 228.7733333, 229.32, 229.7666667, 230.1133333, 230.66,
231.1066667, 231.6533333, 232, 232.4466667, 232.8933333, 233.24,
233.7866667, 234.0333333, 234.68, 235.0266667, 235.2733333, 235.52,
235.8666667, 236.3133333, 236.66, 237.0066667, 237.3533333, 237.7,
237.9466667, 238.2933333, 238.54, 238.7866667, 239.2333333, 239.48,
239.8266667, 240.1733333, 240.42, 240.6666667, 240.9133333, 241.16,
241.5066667, 241.7533333, 242, 242.2466667};
float MLG_mas40_Stroke_minimo[] = {201.0583333, 202.1166667, 202.675,
203.7333333, 204.4916667, 205.35, 206.1083333, 206.9666667, 207.625,
208.1833333, 209.0416667, 209.7, 210.2583333, 210.8166667, 211.475,
212.3333333, 212.6916667, 213.45, 213.8083333, 214.5666667, 215.025,
215.6833333, 216.1416667, 216.7, 217.1583333, 217.5166667, 218.075,
218.5333333, 219.0916667, 219.45, 219.9083333, 220.3666667, 220.725,
221.2833333, 221.5416667, 222.2, 222.5583333, 222.8166667, 223.075,
223.4333333, 223.8916667, 224.25, 224.6083333, 224.9666667, 225.325,
225.5833333, 225.9416667, 226.2, 226.4583333, 226.9166667, 227.175,
227.5333333, 227.8916667, 228.15, 228.4083333, 228.6666667, 228.925,
229.2833333, 229.5416667, 229.8, 230.0583333};
float MLG_mas40_Stroke_maximo[] = {208.6583333, 209.7166667, 210.275,
211.3333333, 212.0916667, 212.95, 213.7083333, 214.5666667, 215.225,
215.7833333, 216.6416667, 217.3, 217.8583333, 218.4166667, 219.075,
219.9333333, 220.2916667, 221.05, 221.4083333, 222.1666667, 222.625,
223.2833333, 223.7416667, 224.3, 224.7583333, 225.1166667, 225.675,
226.1333333, 226.6916667, 227.05, 227.5083333, 227.9666667, 228.325,
228.8833333, 229.1416667, 229.8, 230.1583333, 230.4166667, 230.675,
231.0333333, 231.4916667, 231.85, 232.2083333, 232.5666667, 232.925,
233.1833333, 233.5416667, 233.8, 234.0583333, 234.5166667, 234.775,
235.1333333, 235.4916667, 235.75, 236.0083333, 236.2666667, 236.525,
236.8833333, 237.1416667, 237.4, 237.6583333};
float MLG_mas50_Stroke_minimo[] = {195.77, 196.84, 197.41, 198.48,
199.25, 200.12, 200.89, 201.76, 202.43, 203, 203.87, 204.54, 205.11,
205.68, 206.35, 207.22, 207.59, 208.36, 208.73, 209.5, 209.97, 210.64,
211.11, 211.68, 212.15, 212.52, 213.09, 213.56, 214.13, 214.5, 214.97,
215.44, 215.81, 216.38, 216.65, 217.32, 217.69, 217.96, 218.23, 218.6,
219.07, 219.44, 219.81, 220.18, 220.55, 220.82, 221.19, 221.46,
221.73, 222.2, 222.47, 222.84, 223.21, 223.48, 223.75, 224.02, 224.29,
224.66, 224.93, 225.2, 225.47};
float MLG_mas50_Stroke_maximo[] = {203.37, 204.44, 205.01, 206.08,
206.85, 207.72, 208.49, 209.36, 210.03, 210.6, 211.47, 212.14, 212.71,
213.28, 213.95, 214.82, 215.19, 215.96, 216.33, 217.1, 217.57, 218.24,
218.71, 219.28, 219.75, 220.12, 220.69, 221.16, 221.73, 222.1, 222.57,
223.04, 223.41, 223.98, 224.25, 224.92, 225.29, 225.56, 225.83, 226.2,
226.67, 227.04, 227.41, 227.78, 228.15, 228.42, 228.79, 229.06,
229.33, 229.8, 230.07, 230.44, 230.81, 231.08, 231.35, 231.62, 231.89,
232.26, 232.53, 232.8, 233.07};


//    Valores para NLG
```

```
float  NLG_menos10_Stroke_minimo[]  =  {295.6916667,  297.3833333,
298.575, 300.2666667, 301.6583333, 303.15, 304.5416667, 306.0333333,
307.325, 308.5166667, 310.0083333, 311.3, 312.4916667, 313.6833333,
314.975, 316.4666667, 317.4583333, 318.85, 319.8416667, 321.2333333,
322.325, 323.6166667, 324.7083333, 325.9, 326.9916667, 327.9833333,
329.175, 330.2666667, 331.4583333, 332.45, 333.5416667, 334.6333333,
335.625, 336.8166667, 337.7083333, 339, 339.9916667, 340.8833333,
341.775, 342.7666667, 343.8583333, 344.85, 345.8416667, 346.8333333,
347.825, 348.7166667, 349.7083333, 350.6, 351.4916667, 352.5833333,
353.475, 354.4666667, 355.4583333, 356.35, 357.2416667, 358.1333333,
359.025, 360.0166667, 360.9083333, 361.8, 362.6916667};
float  NLG_menos10_Stroke_maximo[]  =  {301.7416667,  303.4833333,
304.725, 306.4666667, 307.9083333, 309.45, 310.8916667, 312.4333333,
313.775, 315.0166667, 316.5583333, 317.9, 319.1416667, 320.3833333,
321.725, 323.2666667, 324.3083333, 325.75, 326.7916667, 328.2333333,
329.375, 330.7166667, 331.8583333, 333.1, 334.2416667, 335.2833333,
336.525, 337.6666667, 338.9083333, 339.95, 341.0916667, 342.2333333,
343.275, 344.5166667, 345.4583333, 346.8, 347.8416667, 348.7833333,
349.725, 350.7666667, 351.9083333, 352.95, 353.9916667, 355.0333333,
356.075, 357.0166667, 358.0583333, 359, 359.9416667, 361.0833333,
362.025, 363.0666667, 364.1083333, 365.05, 365.9916667, 366.9333333,
367.875, 368.9166667, 369.8583333, 370.8, 371.7416667};
float NLG_cero_Stroke_minimo[] = {285.7416667, 287.4833333, 288.725,
290.4666667, 291.9083333, 293.45, 294.8916667, 296.4333333, 297.775,
299.0166667, 300.5583333, 301.9, 303.1416667, 304.3833333, 305.725,
307.2666667, 308.3083333, 309.75, 310.7916667, 312.2333333, 313.375,
314.7166667, 315.8583333, 317.1, 318.2416667, 319.2833333, 320.525,
321.6666667, 322.9083333, 323.95, 325.0916667, 326.2333333, 327.275,
328.5166667, 329.4583333, 330.8, 331.8416667, 332.7833333, 333.725,
334.7666667, 335.9083333, 336.95, 337.9916667, 339.0333333, 340.075,
341.0166667, 342.0583333, 343, 343.9416667, 345.0833333, 346.025,
347.0666667, 348.1083333, 349.05, 349.9916667, 350.9333333, 351.875,
352.9166667, 353.8583333, 354.8, 355.7416667};
float NLG_cero_Stroke_maximo[] = {291.7916667, 293.5833333, 294.875,
296.6666667, 298.1583333, 299.75, 301.2416667, 302.8333333, 304.225,
305.5166667, 307.1083333, 308.5, 309.7916667, 311.0833333, 312.475,
314.0666667, 315.1583333, 316.65, 317.7416667, 319.2333333, 320.425,
321.8166667, 323.0083333, 324.3, 325.4916667, 326.5833333, 327.875,
329.0666667, 330.3583333, 331.45, 332.6416667, 333.8333333, 334.925,
336.2166667, 337.2083333, 338.6, 339.6916667, 340.6833333, 341.675,
342.7666667, 343.9583333, 345.05, 346.1416667, 347.2333333, 348.325,
349.3166667, 350.4083333, 351.4, 352.3916667, 353.5833333, 354.575,
355.6666667, 356.7583333, 357.75, 358.7416667, 359.7333333, 360.725,
361.8166667, 362.8083333, 363.8, 364.7916667};
float NLG_mas10_Stroke_minimo[] = {275.7916667, 277.5833333, 278.875,
280.6666667, 282.1583333, 283.75, 285.2416667, 286.8333333, 288.225,
289.5166667, 291.1083333, 292.5, 293.7916667, 295.0833333, 296.475,
298.0666667, 299.1583333, 300.65, 301.7416667, 303.2333333, 304.425,
305.8166667, 307.0083333, 308.3, 309.4916667, 310.5833333, 311.875,
313.0666667, 314.3583333, 315.45, 316.6416667, 317.8333333, 318.925,
320.2166667, 321.2083333, 322.6, 323.6916667, 324.6833333, 325.675,
326.7666667, 327.9583333, 329.05, 330.1416667, 331.2333333, 332.325,
333.3166667, 334.4083333, 335.4, 336.3916667, 337.5833333, 338.575,
```

```
339.6666667, 340.7583333, 341.75, 342.7416667, 343.7333333, 344.725,
345.8166667, 346.8083333, 347.8, 348.7916667};
float NLG_mas20_Stroke_maximo[] = {265.8416667, 267.6833333, 269.025,
270.8666667, 272.4083333, 274.05, 275.5916667, 277.2333333, 278.675,
280.0166667, 281.6583333, 283.1, 284.4416667, 285.7833333, 287.225,
288.8666667, 290.0083333, 291.55, 292.6916667, 294.2333333, 295.475,
296.9166667, 298.1583333, 299.5, 300.7416667, 301.8833333, 303.225,
304.4666667, 305.8083333, 306.95, 308.1916667, 309.4333333, 310.575,
311.9166667, 312.9583333, 314.4, 315.5416667, 316.5833333, 317.625,
318.7666667, 320.0083333, 321.15, 322.2916667, 323.4333333, 324.575,
325.6166667, 326.7583333, 327.8, 328.8416667, 330.0833333, 331.125,
332.2666667, 333.4083333, 334.45, 335.4916667, 336.5333333, 337.575,
338.7166667, 339.7583333, 340.8, 341.8416667};
float NLG_mas20_Stroke_minimo[] = {281.8416667, 283.6833333, 285.025,
286.8666667, 288.4083333, 290.05, 291.5916667, 293.2333333, 294.675,
296.0166667, 297.6583333, 299.1, 300.4416667, 301.7833333, 303.225,
304.8666667, 306.0083333, 307.55, 308.6916667, 310.2333333, 311.475,
312.9166667, 314.1583333, 315.5, 316.7416667, 317.8833333, 319.225,
320.4666667, 321.8083333, 322.95, 324.1916667, 325.4333333, 326.575,
327.9166667, 328.9583333, 330.4, 331.5416667, 332.5833333, 333.625,
334.7666667, 336.0083333, 337.15, 338.2916667, 339.4333333, 340.575,
341.6166667, 342.7583333, 343.8, 344.8416667, 346.0833333, 347.125,
348.2666667, 349.4083333, 350.45, 351.4916667, 352.5333333, 353.575,
354.7166667, 355.7583333, 356.8, 357.8416667};
float NLG_mas10_Stroke_maximo[] = {271.8916667, 273.7833333, 275.175,
277.0666667, 278.6583333, 280.35, 281.9416667, 283.6333333, 285.125,
286.5166667, 288.2083333, 289.7, 291.0916667, 292.4833333, 293.975,
295.6666667, 296.8583333, 298.45, 299.6416667, 301.2333333, 302.525,
304.0166667, 305.3083333, 306.7, 307.9916667, 309.1833333, 310.575,
311.8666667, 313.2583333, 314.45, 315.7416667, 317.0333333, 318.225,
319.6166667, 320.7083333, 322.2, 323.3916667, 324.4833333, 325.575,
326.7666667, 328.0583333, 329.25, 330.4416667, 331.6333333, 332.825,
333.9166667, 335.1083333, 336.2, 337.2916667, 338.5833333, 339.675,
340.8666667, 342.0583333, 343.15, 344.2416667, 345.3333333, 346.425,
347.6166667, 348.7083333, 349.8, 350.8916667};
float NLG_mas30_Stroke_minimo[] = {255.8916667, 257.7833333, 259.175,
261.0666667, 262.6583333, 264.35, 265.9416667, 267.6333333, 269.125,
270.5166667, 272.2083333, 273.7, 275.0916667, 276.4833333, 277.975,
279.6666667, 280.8583333, 282.45, 283.6416667, 285.2333333, 286.525,
288.0166667, 289.3083333, 290.7, 291.9916667, 293.1833333, 294.575,
295.8666667, 297.2583333, 298.45, 299.7416667, 301.0333333, 302.225,
303.6166667, 304.7083333, 306.2, 307.3916667, 308.4833333, 309.575,
310.7666667, 312.0583333, 313.25, 314.4416667, 315.6333333, 316.825,
317.9166667, 319.1083333, 320.2, 321.2916667, 322.5833333, 323.675,
324.8666667, 326.0583333, 327.15, 328.2416667, 329.3333333, 330.425,
331.6166667, 332.7083333, 333.8, 334.8916667};
float NLG_mas30_Stroke_maximo[] = {261.9416667, 263.8833333, 265.325,
267.2666667, 268.9083333, 270.65, 272.2916667, 274.0333333, 275.575,
277.0166667, 278.7583333, 280.3, 281.7416667, 283.1833333, 284.725,
286.4666667, 287.7083333, 289.35, 290.5916667, 292.2333333, 293.575,
295.1166667, 296.4583333, 297.9, 299.2416667, 300.4833333, 301.925,
303.2666667, 304.7083333, 305.95, 307.2916667, 308.6333333, 309.875,
311.3166667, 312.4583333, 314, 315.2416667, 316.3833333, 317.525,
318.7666667, 320.1083333, 321.35, 322.5916667, 323.8333333, 325.075,
```

```
326.2166667, 327.4583333, 328.6, 329.7416667, 331.0833333, 332.225,
333.4666667, 334.7083333, 335.85, 336.9916667, 338.1333333, 339.275,
340.5166667, 341.6583333, 342.8, 343.9416667};
float NLG_mas40_Stroke_minimo[] = {245.9416667, 247.8833333, 249.325,
251.2666667, 252.9083333, 254.65, 256.2916667, 258.0333333, 259.575,
261.0166667, 262.7583333, 264.3, 265.7416667, 267.1833333, 268.725,
270.4666667, 271.7083333, 273.35, 274.5916667, 276.2333333, 277.575,
279.1166667, 280.4583333, 281.9, 283.2416667, 284.4833333, 285.925,
287.2666667, 288.7083333, 289.95, 291.2916667, 292.6333333, 293.875,
295.3166667, 296.4583333, 298, 299.2416667, 300.3833333, 301.525,
302.7666667, 304.1083333, 305.35, 306.5916667, 307.8333333, 309.075,
310.2166667, 311.4583333, 312.6, 313.7416667, 315.0833333, 316.225,
317.4666667, 318.7083333, 319.85, 320.9916667, 322.1333333, 323.275,
324.5166667, 325.6583333, 326.8, 327.9416667};
float NLG_mas40_Stroke_maximo[] = {251.9916667, 253.9833333, 255.475,
257.4666667, 259.1583333, 260.95, 262.6416667, 264.4333333, 266.025,
267.5166667, 269.3083333, 270.9, 272.3916667, 273.8833333, 275.475,
277.2666667, 278.5583333, 280.25, 281.5416667, 283.2333333, 284.625,
286.2166667, 287.6083333, 289.1, 290.4916667, 291.7833333, 293.275,
294.6666667, 296.1583333, 297.45, 298.8416667, 300.2333333, 301.525,
303.0166667, 304.2083333, 305.8, 307.0916667, 308.2833333, 309.475,
310.7666667, 312.1583333, 313.45, 314.7416667, 316.0333333, 317.325,
318.5166667, 319.8083333, 321, 322.1916667, 323.5833333, 324.775,
326.0666667, 327.3583333, 328.55, 329.7416667, 330.9333333, 332.125,
333.4166667, 334.6083333, 335.8, 336.9916667};
float NLG_mas50_Stroke_minimo[] = {235.9916667, 237.9833333, 239.475,
241.4666667, 243.1583333, 244.95, 246.6416667, 248.4333333, 250.025,
251.5166667, 253.3083333, 254.9, 256.3916667, 257.8833333, 259.475,
261.2666667, 262.5583333, 264.25, 265.5416667, 267.2333333, 268.625,
270.2166667, 271.6083333, 273.1, 274.4916667, 275.7833333, 277.275,
278.6666667, 280.1583333, 281.45, 282.8416667, 284.2333333, 285.525,
287.0166667, 288.2083333, 289.8, 291.0916667, 292.2833333, 293.475,
294.7666667, 296.1583333, 297.45, 298.7416667, 300.0333333, 301.325,
302.5166667, 303.8083333, 305, 306.1916667, 307.5833333, 308.775,
310.0666667, 311.3583333, 312.55, 313.7416667, 314.9333333, 316.125,
317.4166667, 318.6083333, 319.8, 320.9916667};
float NLG_mas50_Stroke_maximo[] = {242.0416667, 244.0833333, 245.625,
247.6666667, 249.4083333, 251.25, 252.9916667, 254.8333333, 256.475,
258.0166667, 259.8583333, 261.5, 263.0416667, 264.5833333, 266.225,
268.0666667, 269.4083333, 271.15, 272.4916667, 274.2333333, 275.675,
277.3166667, 278.7583333, 280.3, 281.7416667, 283.0833333, 284.625,
286.0666667, 287.6083333, 288.95, 290.3916667, 291.8333333, 293.175,
294.7166667, 295.9583333, 297.6, 298.9416667, 300.1833333, 301.425,
302.7666667, 304.2083333, 305.55, 306.8916667, 308.2333333, 309.575,
310.8166667, 312.1583333, 313.4, 314.6416667, 316.0833333, 317.325,
318.6666667, 320.0083333, 321.25, 322.4916667, 323.7333333, 324.975,
326.3166667, 327.5583333, 328.8, 330.0416667};


LiquidCrystal_I2C lcd(0x27,20,4);
File myFile;
int CodigoDeFallo=0;

float Parametros[4];
```

```
int PresionMaximaInt = 0;
int PresionMinimaInt = 0;

RH_ASK driver(SPEED, RX_PIN, TX_PIN);
RHReliableDatagram manager(driver, SERVER_ADDRESS);
String floatToString(float,int=8,int=2,boolean=true);
int SeleccionDeTren = 0;     //MLG=0 (led azul) y NLG=1 (led verde)



void setup()
{
    lcd.init();
    lcd.backlight();
    Serial.begin(9600);
    if (!manager.init()){
        lcd.setCursor(0,0);
        lcd.print("Fallo de iniciacion");
    }
    //Establezco por defecto el modo MLG
    pinMode(pulsador, INPUT_PULLUP);
    pinMode(PinLedAzul, OUTPUT);
    pinMode(PinLedVerde, OUTPUT);
    digitalWrite(PinLedAzul, HIGH);
    digitalWrite(PinLedVerde, LOW);


}

void loop()
{
    if(digitalRead(pulsador) == LOW)
    {
        if(SeleccionDeTren == 1){
            SeleccionDeTren = 0;
            digitalWrite(PinLedAzul, HIGH);
            digitalWrite(PinLedVerde, LOW);
            delay(1000);
        }
        else if(SeleccionDeTren == 0){
            SeleccionDeTren = 1;
            digitalWrite(PinLedVerde, HIGH);
            digitalWrite(PinLedAzul, LOW);
            delay(1000);
        }

    }
    uint8_t data[] = "And hello back to you";
    uint8_t buf[RH_ASK_MAX_MESSAGE_LEN];

    uint8_t len = sizeof(buf);
    uint8_t from;

    if (manager.available()){
```

```
        if(manager.recvfromAck(buf, &len, &from)){

              String AmbientTempString="";
              String ObjectTempString="";
              String StrokeString="";
              String PresionString="";
              int numParametro=0;
              String msg = (char*)buf;
              for(int i=0; i<len; i++){
                    if (msg.charAt(i) != ';'){
                        if(numParametro==0){
                            AmbientTempString = AmbientTempString +
msg.charAt(i);
                        }
                        else if(numParametro==1){
                            ObjectTempString = ObjectTempString +
msg.charAt(i);
                        }
                        else if(numParametro==2){
                            StrokeString    =    StrokeString    +
msg.charAt(i);
                        }
                        else if(numParametro==3){
                            PresionString   =    PresionString   +
msg.charAt(i);
                        }
                    }
                    else if(msg.charAt(i) == ';'){
                        numParametro++;
                    }
              }
              Parametros[0]= AmbientTempString.toFloat();
              Parametros[1]= ObjectTempString.toFloat();
              Parametros[2]= 395 - StrokeString.toFloat();
              Parametros[3]= PresionString.toFloat();

              bool fallo = VerificacionDeTemperatura();
              if(fallo==false){
                  MostrarParametros(false);
              }
              else{
                  MostrarParametros(true);
              }
        }
    }
}



void MostrarParametros (bool fallo){
    if(fallo==true){
        lcd.setCursor(0,0);
        lcd.print("T. Ambiente: ");
        lcd.print(Parametros[0]);
```

```
        lcd.setCursor(19,0);
        lcd.print("!");
        lcd.setCursor(0,1);
        lcd.print("T. Objeto: ");
        lcd.print(Parametros[1]);
        lcd.setCursor(0,2);
        lcd.print("Stroke: ");
        lcd.print(Parametros[2]);
        lcd.setCursor(0,3);
        lcd.print("T. FUERA DE LIMITES ");

    }
    else{
        lcd.setCursor(0,0);
        lcd.print("T. Ambiente: ");
        lcd.print(Parametros[0]);
        lcd.setCursor(0,1);
        lcd.print("T. Objeto: ");
        lcd.print(Parametros[1]);
        lcd.setCursor(0,2);
        lcd.print("Stroke: ");
        lcd.print(Parametros[2]);
        lcd.setCursor(0,3);
        lcd.print("Pres: ");
        if(SeleccionDeTren == 0){
          CalculoDePresionValidaMLG();
        }
        else if(SeleccionDeTren == 1){
          CalculoDePresionValidaNLG();
        }

        if(CodigoDeFallo == 0){
            lcd.print(PresionMinimaInt);
            lcd.print(" - ");
            lcd.print(PresionMaximaInt);
            myFile      =      SD.open("test_de_amortiguadores.txt",
FILE_WRITE);
            if (myFile) {
                myFile.print("T. Ambiente: ");
                myFile.print(Parametros[0]);
                myFile.print(";T. Objeto: ");
                myFile.print(Parametros[1]);
                myFile.print(";Stroke: ");
                myFile.print(Parametros[2]);
                myFile.print(";Pres: ");
                myFile.print(PresionMinimaInt);
                myFile.print(" - ");
                myFile.print(PresionMaximaInt);
                myFile.close();
            }
        }
        else if (CodigoDeFallo==2){
            lcd.setCursor(0,3);
            lcd.print("STROKE MUY BAJO     ");
```

```
        }
        else if (CodigoDeFallo==3){
            lcd.setCursor(0,3);
            lcd.print("STROKE MUY ALTO     ");
        }
    }
    return;
}


// Verifica que no haya mayor diferencia de temperatura
// entre la temperatura del objeto y la ambiental, que la
//  definida  en  ToleranciaDiferencialDeTemperatura  (originalmente  1
grado), pero la controlaremos con un define
bool VerificacionDeTemperatura(){
    if((Parametros[0]-
Parametros[1])>ToleranciaDiferencialDeTemperatura){
        CodigoDeFallo=1;
        MostrarParametros(true);
        return true;
    }
    else                                                if((Parametros[1]-
Parametros[0])>ToleranciaDiferencialDeTemperatura){
        CodigoDeFallo=1;
        MostrarParametros(true);
        return true;
    }
    else{
        return false;
    }
}


void CalculoDePresionValidaMLG(){
    CodigoDeFallo=0;
    if(Parametros[1]>=(10) && Parametros[1]<(20)){
        if(Parametros[2] > MLG_mas10_Stroke_minimo[0])  {
              if(Parametros[2] < MLG_mas10_Stroke_maximo[59])  {
                  PresionMinimaInt = MLG_mas10_Stroke_minimo[0];
                  PresionMaximaInt = MLG_mas10_Stroke_maximo[59];
                  for(int i=0;i<60; i++){
                        if(Parametros[2]                                <
MLG_mas10_Stroke_maximo[i]){
                            PresionMinimaInt = (i+1)+50;
                        }
                        if(Parametros[2]                                >
MLG_mas10_Stroke_minimo[i]){
                            PresionMaximaInt = i+50;
                        }
                   }
              }
              else{
                    //Stroke por encima del maximo
                    CodigoDeFallo=3;
```

```
            }
        }
        else{
              //Stroke por debajo del minimo
              CodigoDeFallo=2;
        }
    }
    else if(Parametros[1]>=(20) && Parametros[1]<(30)){
        if(Parametros[2] > MLG_mas20_Stroke_minimo[0])  {
             if(Parametros[2] < MLG_mas20_Stroke_maximo[59])  {
                 PresionMinimaInt = MLG_mas20_Stroke_minimo[0];
                 PresionMaximaInt = MLG_mas20_Stroke_maximo[59];
                 for(int i=0;i<60; i++){
                       if(Parametros[2]                              >
MLG_mas20_Stroke_maximo[i]){
                            PresionMinimaInt = (i+1)+50;
                       }
                       if(Parametros[2]                              >
MLG_mas20_Stroke_minimo[i]){
                            PresionMaximaInt = i+50;
                       }
                  }
             }
             else{
                   //Stroke por encima del maximo
                   CodigoDeFallo=3;
             }
        }
        else{
              //Stroke por debajo del minimo
              CodigoDeFallo=2;
        }
    }
    else if(Parametros[1]>=(-10) && Parametros[1]<(0)){
          if(Parametros[2] > MLG_menos10_Stroke_minimo[0])  {
              if(Parametros[2] < MLG_menos10_Stroke_maximo[59])  {
                  PresionMinimaInt = MLG_menos10_Stroke_minimo[0];
                  PresionMaximaInt = MLG_menos10_Stroke_maximo[59];
                  for(int i=0;i<60; i++){
                        if(Parametros[2]                              >
MLG_menos10_Stroke_maximo[i]){
                             PresionMinimaInt = (i+1)+50;
                        }
                        if(Parametros[2]                              >
MLG_menos10_Stroke_minimo[i]){
                             PresionMaximaInt = i+50;
                        }
                   }
              }
              else{
                    //Stroke por encima del maximo
                    CodigoDeFallo=3;
              }
          }
```

```
          else{
                //Stroke por debajo del minimo
                CodigoDeFallo=2;
                return;
          }
    }
    else if(Parametros[1]>=(0) && Parametros[1]<(10)){
       if(Parametros[2] > MLG_cero_Stroke_minimo[0])  {
              if(Parametros[2] < MLG_cero_Stroke_maximo[59])  {
                  PresionMinimaInt = MLG_cero_Stroke_minimo[0];
                  PresionMaximaInt = MLG_cero_Stroke_maximo[59];
                  for(int i=0;i<60; i++){
                        if(Parametros[2]                              >
MLG_cero_Stroke_maximo[i]){
                              PresionMinimaInt = (i+1)+50;
                        }
                        if(Parametros[2]                              >
MLG_cero_Stroke_minimo[i]){
                              PresionMaximaInt = i+50;
                        }
                   }
              }
              else{
                    //Stroke por encima del maximo
                    CodigoDeFallo=3;
              }
          }
          else{
                //Stroke por debajo del minimo
                CodigoDeFallo=2;
          }
    }

    else if(Parametros[1]>=(30) && Parametros[1]<(40)){
        if(Parametros[2] > MLG_mas30_Stroke_minimo[0])  {
              if(Parametros[2] < MLG_mas30_Stroke_maximo[59])  {
                  PresionMinimaInt = MLG_mas30_Stroke_minimo[0];
                  PresionMaximaInt = MLG_mas30_Stroke_maximo[59];
                  for(int i=0;i<60; i++){
                        if(Parametros[2]                              >
MLG_mas30_Stroke_maximo[i]){
                              PresionMinimaInt = (i+1)+50;
                        }
                        if(Parametros[2]                              >
MLG_mas30_Stroke_minimo[i]){
                              PresionMaximaInt = i+50;
                        }
                   }
              }
              else{
                    //Stroke por encima del maximo
                    CodigoDeFallo=3;
              }
          }
```

```
          else{
                //Stroke por debajo del minimo
                CodigoDeFallo=2;
          }
    }
    else if(Parametros[1]>=(40) && Parametros[1]<(50)){
       if(Parametros[2] > MLG_mas40_Stroke_minimo[0])  {
              if(Parametros[2] < MLG_mas40_Stroke_maximo[59])  {
                  PresionMinimaInt = MLG_mas40_Stroke_minimo[0];
                  PresionMaximaInt = MLG_mas40_Stroke_maximo[59];
                  for(int i=0;i<60; i++){
                        if(Parametros[2]                                >
MLG_mas40_Stroke_maximo[i]){
                              PresionMinimaInt = (i+1)+50;
                        }
                        if(Parametros[2]                                >
MLG_mas40_Stroke_minimo[i]){
                              PresionMaximaInt = ((i)/2)+50;
                        }
                   }
              }
              else{
                    //Stroke por encima del maximo
                    CodigoDeFallo=3;
              }
          }
          else{
                //Stroke por debajo del minimo
                CodigoDeFallo=2;
          }
    }
    else if(Parametros[1]>=(50) && Parametros[1]<(60)){
       if(Parametros[2] > MLG_mas50_Stroke_minimo[0])  {
              if(Parametros[2] < MLG_mas50_Stroke_maximo[59])  {
                  PresionMinimaInt = MLG_mas50_Stroke_minimo[0];
                  PresionMaximaInt = MLG_mas50_Stroke_maximo[59];
                  for(int i=0;i<60; i++){
                        if(Parametros[2]                                >
MLG_mas50_Stroke_maximo[i]){
                              PresionMinimaInt = (i+1)+50;
                        }
                        if(Parametros[2]                                >
MLG_mas50_Stroke_minimo[i]){
                              PresionMaximaInt = i+50;
                        }
                   }
              }
              else{
                    //Stroke por encima del maximo
                    CodigoDeFallo=3;
              }
          }
          else{
                //Stroke por debajo del minimo
```

```
                CodigoDeFallo=2;
            }
        }

}


void CalculoDePresionValidaNLG(){
    CodigoDeFallo=0;
    if(Parametros[1]>=(0) && Parametros[1]<(10)){
       if(Parametros[2] > NLG_cero_Stroke_minimo[0])  {
              if(Parametros[2] < NLG_cero_Stroke_maximo[59])  {
                  PresionMinimaInt = NLG_cero_Stroke_minimo[0];
                  PresionMaximaInt = NLG_cero_Stroke_maximo[59];
                  for(int i=0;i<60; i++){
                        if(Parametros[2]                                    >
NLG_cero_Stroke_maximo[i]){
                            PresionMinimaInt = ((i+1)/2)+25;
                        }
                        if(Parametros[2]                                    >
NLG_cero_Stroke_minimo[i]){
                            PresionMaximaInt = ((i)/2)+25;
                        }
                   }
              }
              else{
                    //Stroke por encima del maximo
                    CodigoDeFallo=3;
              }
         }
         else{
               //Stroke por debajo del minimo
               CodigoDeFallo=2;
         }
    }
    else if(Parametros[1]>=(10) && Parametros[1]<(20)){
        if(Parametros[2] > NLG_mas10_Stroke_minimo[0])  {
              if(Parametros[2] < NLG_mas10_Stroke_maximo[59])  {
                  PresionMinimaInt = NLG_mas10_Stroke_minimo[0];
                  PresionMaximaInt = NLG_mas10_Stroke_maximo[59];
                  for(int i=0;i<60; i++){
                        if(Parametros[2]                                    >
NLG_mas10_Stroke_maximo[i]){
                            PresionMinimaInt = ((i+1)/2)+25;
                        }
                        if(Parametros[2]                                    >
NLG_mas10_Stroke_minimo[i]){
                            PresionMaximaInt = ((i)/2)+25;
                        }
                   }
              }
              else{
                    //Stroke por encima del maximo
                    CodigoDeFallo=3;
```

```
            }
        }
        else{
              //Stroke por debajo del minimo
              CodigoDeFallo=2;
        }
    }
    else if(Parametros[1]>=(20) && Parametros[1]<(30)){
        if(Parametros[2] > NLG_mas20_Stroke_minimo[0])  {
             if(Parametros[2] < NLG_mas20_Stroke_maximo[59]){
                 PresionMinimaInt = NLG_mas20_Stroke_minimo[0];
                 PresionMaximaInt = NLG_mas20_Stroke_maximo[59];
                 for(int i=0;i<60; i++){
                       if(Parametros[2]                              >
NLG_mas20_Stroke_maximo[i]){
                            PresionMinimaInt = ((i+1)/2)+25;
                       }
                       if(Parametros[2]                              >
NLG_mas20_Stroke_minimo[i]){
                            PresionMaximaInt = ((i)/2)+25;
                       }
                  }
             }
             else{
                   //Stroke por encima del maximo
                   CodigoDeFallo=3;
             }
        }
        else{
              //Stroke por debajo del minimo
              CodigoDeFallo=2;
        }
    }
    else if(Parametros[1]>=(-10) && Parametros[1]<(0)){
          if(Parametros[2] > NLG_menos10_Stroke_minimo[0])  {
              if(Parametros[2] < NLG_menos10_Stroke_maximo[59])  {
                 PresionMinimaInt = NLG_menos10_Stroke_minimo[0];
                 PresionMaximaInt = NLG_menos10_Stroke_maximo[59];
                 for(int i=0;i<60; i++){
                       if(Parametros[2]                              >
NLG_menos10_Stroke_maximo[i]){
                            PresionMinimaInt = ((i+1)/2)+25;
                       }
                       if(Parametros[2]                              >
NLG_menos10_Stroke_minimo[i]){
                            PresionMaximaInt = ((i)/2)+25;
                       }
                  }
             }
             else{
                   //Stroke por encima del maximo
                   CodigoDeFallo=3;
             }
        }
```

```
          else{
                //Stroke por debajo del minimo
                CodigoDeFallo=2;
                return;
          }
    }
    else if(Parametros[1]>=(30) && Parametros[1]<(40)){
        if(Parametros[2] > NLG_mas30_Stroke_minimo[0])  {
              if(Parametros[2] < NLG_mas30_Stroke_maximo[59])  {
                  PresionMinimaInt = NLG_mas30_Stroke_minimo[0];
                  PresionMaximaInt = NLG_mas30_Stroke_maximo[59];
                  for(int i=0;i<60; i++){
                        if(Parametros[2]                                >
NLG_mas30_Stroke_maximo[i]){
                            PresionMinimaInt = ((i+1)/2)+25;
                        }
                        if(Parametros[2]                                >
NLG_mas30_Stroke_minimo[i]){
                            PresionMaximaInt = ((i)/2)+25;
                        }
                   }
              }
              else{
                    //Stroke por encima del maximo
                    CodigoDeFallo=3;
              }
          }
          else{
                //Stroke por debajo del minimo
                CodigoDeFallo=2;
          }
    }
    else if(Parametros[1]>=(40) && Parametros[1]<(50)){
       if(Parametros[2] > NLG_mas40_Stroke_minimo[0])  {
              if(Parametros[2] < NLG_mas40_Stroke_maximo[59])  {
                  PresionMinimaInt = NLG_mas40_Stroke_minimo[0];
                  PresionMaximaInt = NLG_mas40_Stroke_maximo[59];
                  for(int i=0;i<60; i++){
                        if(Parametros[2]                                >
NLG_mas40_Stroke_maximo[i]){
                            PresionMinimaInt = ((i+1)/2)+25;
                        }
                        if(Parametros[2]                                >
NLG_mas40_Stroke_minimo[i]){
                            PresionMaximaInt = ((i)/2)+25;
                        }
                   }
              }
              else{
                    //Stroke por encima del maximo
                    CodigoDeFallo=3;
              }
          }
          else{
```

```
                //Stroke por debajo del minimo
                CodigoDeFallo=2;
          }
    }
    else if(Parametros[1]>=(50) && Parametros[1]<(60)){
       if(Parametros[2] > NLG_mas50_Stroke_minimo[0])  {
              if(Parametros[2] < NLG_mas50_Stroke_maximo[59])  {
                  PresionMinimaInt = NLG_mas50_Stroke_minimo[0];
                  PresionMaximaInt = NLG_mas50_Stroke_maximo[59];
                  for(int i=0;i<60; i++){
                        if(Parametros[2]                                    >
NLG_mas50_Stroke_maximo[i]){
                            PresionMinimaInt = ((i+1)/2)+25;
                        }
                        if(Parametros[2]                                    >
NLG_mas50_Stroke_minimo[i]){
                            PresionMaximaInt = ((i)/2)+25;
                        }
                   }
              }
              else{
                    //Stroke por encima del maximo
                    CodigoDeFallo=3;
              }
          }
          else{
                //Stroke por debajo del minimo
                CodigoDeFallo=2;
          }
    }

}

String floatToString( float n, int l, int d, boolean z){
  char c[l+1];
  String s="";
  dtostrf(n,l,d,c);
  s=String(c);

  if(z){
    s.replace(" ","");
  }
  return s;
}
```