Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Diseño de sistema de termografía para aplicaciones médicas

Autor: Rafael Antonio Marín Sánchez
Tutores: María del Mar Elena Pérez
Alfredo Pérez Vega-Leal

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Diseño de sistema de termografía para aplicaciones médicas

Autor:
Rafael Antonio Marín Sánchez

Tutores:
María del Mar Elena Pérez
Alfredo Pérez Vega-Leal

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Grado:

Diseño de sistema de termografía para aplicaciones médicas

Autor: Rafael Antonio Marín Sánchez

Tutor: María del Mar Elena Pérez

Alfredo Pérez Vega-Leal

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

Agradecimientos

A mi familia, en especial a mis padres y a mis hermanos, que siempre han estado ahí en las decisiones que he tomado a lo largo de mi vida, en los buenos y malos momentos que han ido acontenciendo. Sin su confianza y cariño no habría podido llegar hasta aquí

A Antonio López Marín, mi primo, que sin su ayuda y consejos posiblemente me habría quedado a medio camino, estar siempre dispuesto a echarme una mano y animarme en la medida de los posible.

A mis amigos de toda la vida, por todos los buenos ratos vividos y momentos con los que evadirme de la rutina de la carrera.

A mis tutores de proyecto, María del Mar Elena y Alfredo Pérez, por guiarme y ayudarme en todo momento durante la realización del proyecto

A los compañeros y profesores de la Universidad de Sevilla, que me ayudaron en tantas ocasiones mientras estudié allí y me enseñaron todo lo que sé.

Rafael Antonio Marín Sánchez
Sevilla, Junio de 2019

Resumen

La temperatura es una de las magnitudes más importantes, ya que influye en muchos procesos relevantes que se llevan a cabo diariamente a lo largo del mundo. El estado de la atmósfera, el rendmiendo de procesos industriales, el diagnostico de una enfermedad son algunos de los ejemplos para los cuales la temperatura juega un papel clave en esta era digital.

Existen muchos dispositivos en el mercado que dan solución a la medición de la temperatura y es por ello que es importante tenerla siempre en cuenta. Este proyecto a sido concevido para comprender y explorar por completo los procedimientos de cálculo de esta, desde su detención hasta la presentación de los datos.

Se ha diseñado un instrumento de medición de temperatura aplicando los conocimientos adquiridos a lo largo del grado. Para dicho propósito, se ha planteado la construcción de un sensor de temperatura encargado de la toma de muestras, las cuales serán acondicionadas mediante una etapa amplificadora. A partir de ahí, una placa de desarrollo es la responsable de la adquisición de la señal, procesamiento y su posterior trasmisión inlámbrica.

La visualización del valor final es llevada a cabo por una aplicación monitor alojada en un smartphone Android. Usando una resolución de 10bits, el termómetro que ha sido desarrollado optiene valores de temperatura, los procesa y los transmite a través de un enlace inlámbrico de hasta 10 metros.

Abstract

Temperature is one of the most important measurands as it influences many relevant daily events that undergo around the world. The state of the atmosphere, the performance of an industrial process or a successfully diagnosed disease are some of the examples for which temperature plays a key role in this digital age.

Many devices on the market already provide solutions to measure temperature whenever it is an essential value to keep under control. And yet, this project was conceived to understand and explore the whole calculation process, from the heat detection to the final data presentation.

A temperature measurement device has been designed by applying the knowledge acquired throughout the degree. For this purpose, the design of a thermal sensor in charge of taking samples has been considered, which will be treated by an amplifier stage. Thereafter, a microcontroller development board is responsible for the signal acquisition and further processing and wireless transmission.

The display of the final values is performed through a monitor application hosted on an Android smartphone. Using 10 bits of resolution, this thermometer has been developed to capture temperature signals, process and transmit them through a wireless link from up to 10 meters.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Termometría, o también conocida como **medición de temperatura**, describe el proceso de asociación de la temperatura como magnitud física a un valor interpretable, así como el método de obtención de este. (1)

En tiempos de Hippocrates solo se usaba la mano para la detección del calor o el frió del cuerpo humano, siendo la fiebre un síntoma de muerte. En Alejandría se empezó a controlar el pulso, dejando la observación de la temperatura, como síntoma de enfermedad. Fue en la Edad Media, cuando a la **teoría los cuatro humores** se le asociaron las cualidades de *calor, frío, seco y húmedo,* hicieron que el control de la fiebre volviese a ganar importancia.

Galileo Galilei (2) en 1592 diseño un burdo instrumento que mostraba cambios en la temperatura, pero no permitía cuantificarla ya que carecía de escala. Quien dio un gran salto en cuanto a la medición de la temperatura corporal fue **Santorio Santorio** (3) (4), un fisiólogo italiano profesor en Padua, quien cogiendo el *termoscopio* de Galileo y le añadiéndole una escala numérica creó el primer termómetro clínico (1612) aunque era inexacto por la influencia que la presión atmosférica ejercía sobre él, magnitud física aún desconocida. En 1654, **Fernando II de Médici Gran Duque de Toscana** aficionado a la ciencia, decidió añadir aguardiente en vez de agua y sellar el instrumento de Santorio, aportando la mejora de ser más sensible a los cambios de temperatura y no congelarse con tanta facilidad.

En 1665, **Christiaan Huygens** propone añadir un único punto fijo a la escala (punto de fusión del agua). **Newton** propuso añadir la temperatura del cuerpo humano como cota superior.

En 1714, **Daniel Fahrenheit** propone el uso de mercurio en vez de aguardiente ya que este presenta una amplia y uniforme expansión térmica, no se adhiere al vidrio y presentaba un mayor rango de temperatura en estado líquido, además de al ser plateado hacia más fácil su lectura. Es en 1724**,** cuando este establece la escala termométrica que es publicada en su libro *Philosophical Transactions*:

"Colocando el termómetro en un mezcla de sal de amonio o agua salada, hielo, y agua, un punto sobre la escala pudo ser encontrado el cual llamé cero. Un segundo punto fue obtenido de la misma manera, si la mezcla es usada sin sal. Denotando este punto como 30. Un tercer punto designado como 96 fue obtenido colocando el termómetro en la boca para adquirir el calor del cuerpo humano" (D.G Fahrenheit, Phil. Trans. (London) 33, 78, 1724).

En 1742, **Anders Celsius** reintroduce a la práctica el uso de la escala centígrada considerando valores para el punto de ebullición y congelación del agua de 100 y 0 ºC respectivamente. Años más tarde, **Jean-Pierre Christin** (1743) y **Carlos Linneo** (1745) invirtieron estos puntos. Este método tenía la ventaja de basarse en las propiedades físicas de los materiales

En 1821, uniendo dos hilos de metales diferentes (cobre y bismuto), **Thomas Johann Seebeck** descubrió accidentalmente que al producirse una diferencia de temperatura entre ambos hilos, se producía un campo magnético, siendo esto la base del funcionamiento de los **termopares** de la actualidad.

En 1866, **Thomas Clifford Allbutt** (5) diseño un termómetro de 6 pulgadas portable capaz de realizar medidas en 5min, que dejó obsoleto al usado en esa época que realizaba mediciones en 20min.

En 1868, **Carl Wunderlich** (6) estableció que el rango normal de temperatura del cuerpo humano era desde 36.3 hasta 37.5 ºC, tras la publicación de los resultados obtenidos tras realizar 1 millón de mediciones en la axila a un total de 25000 pacientes.

En la actualidad, existen diversos tipos diseño de termómetros siendo los digitales los más populares, por presentar ventajas (no son contaminantes al desecharlos, fácil lectura, respuesta rápida, memoria, etc) frente a los tradicionales de mercurio.

Según la referencia del 0 de la escala se pueden diferenciar dos tipos:

- **Escala absoluta**: Kelvin o Rankine
- **Escala relativa**: Celcius o Fahrenheit

La unidad de temperatura en el sistema internacional es el Kelvin (K).

1.1 Objetivos del Proyecto

El objetivo del proyecto es la construcción de un instrumento de medición de la temperatura del cuerpo humano aplicando los conocimientos adquiridos a lo largo del grado. Para dicho propósito se ha planteado la construcción de un sensor de temperatura que se encargará de la toma de muestras, dichas muestras serán acondicionadas, mediante una etapa amplificadora, a una placa de desarrollo que se encargará del análisis de estas y su trasmisión inlámbrica.

La visualización de estas muestras se hará a través de un monitor de forma inalámbrica.

1.2 Estado del arte

Realizando un estudio del arte previo a la realización de las especificaciones, se presentan una serie de productos a la venta, enmarcados en el tipo de dispositivo que se tiene como objetivo:

El FerverSense de Enfasmart (7) es un dispositivo de bajo consumo que cuenta con una autonomía de 7 dias.

Capaz de dar una muestra cada 16s y ser monitorizada a través de una app de móvil en tiempo real.

Se aportan algunos datos técnicos del producto:

- Precisión (+/-0.2°C)
- Distancia de trasmisión (40m)
- Bluetooth 4.1

Figura 1-1. FerverSense

El KSBT1 de koogeek (8) es un termómetro pensado para controlar la temperatura de bebes.

Capaz de dar una muestra cada 5s y ser monitorizada a través de una app de móvil en tiempo real.

- Precisión (±0.05°C)
- Bluetooth 4.0
- Batería reemplazable de 3V/200mAh

Figura 1-2. KSBT1

1.3 Requisitos y Especificaciones del Proyecto

En base a los objetivos marcados, se ha expresado un listado de los requisitos del proyecto:

Funcionales:

- **F.1.** El sistema debe medir temperatura.
- **F.2.** El sistema debe tener comunicación inalambrica con el display de visualización.
- **F.3.** El display debe mostrar la temperatura en grados centígrados
- **F.4.** Debe haber una relación entre la salida del circuito de acondicionamiento y la entrada del CAD.

Diseño:

- **D.2.1.** El MCU usará comunicación WiFi para interactuar con el display.
- **D.2.2.** Se usará un smartphone a modo de display.
- **D.3.1.** La temperatura debe ser mostrada de forma simple y clara.
- **D.3.2.** Se usarán colores fríos para la representación.
- **D.4.1.** La salida del circuito de acondicionamiento debe estar comprendida entre 0-3.3 V.
- **D.4.2.** La ganacia del circuito de acondicionamiento será ~10.
- **D.5.1.** El sistema se alimentará a 5V

Operaciones:

- **O.1.** El proyecto debería prensetar periódicamente informes de desarrollo a los tutores.

El diseño del sistema se llevará a cabo en base al siguiente diagrama de bloques, que cuenta con los siguientes bloques funcionales.

I. Adquisición de datos, que a partir de un sensor de temperatura se envíen las muestras digitalizados a la unidad de procesamiento.

II. Control y procesamiento, se encargará de la interpretación de las muestras y la gestión de los demás bloques.

III. Comunicación inalámbrica, que envíe los datos al *Monitor* en tiempo real.

Figura 1-3. Diagrama de bloques conceptual

1.4 Estructura de la memoria

Este documento se divide en varios capítulos que separan cada una de las partes del proyecto.

- Capítulo 2 (Diseño del Sistema), donde se define el sistema y componentes del mismo.
- Capítulo 3 (Implementación del Sistema), donde se detalla lo implementado en el sistema final.
- Capítulo 4 (Resultados), donde se presentan las pruebas de funcionamiento y la características del sistema.
- Capítulo 5 (Estudio económico), donde se detallan los gastos asumidos en el proyecto.
- Capítulo 6 (Cónclusiones), donde se valora el proyecto en sus conjuto.

Ademas de una serie de anexos con información extra referente a algunos de los puntos de los capítulos.

- Anexo A – Análisis de riesgos
- Anexo B – Cálculo de las desviaciones típicas
- Anexo C – Criterios de aceptación
- Anexo D - Presupuesto
- Anexo E – Código ESP8266
- Anexo F – Código App

2 DISEÑO DEL SISTEMA

Cualquier diseño requiere de una etapa de estudio previo en la que valorar la viabilidad del propio proyecto y además buscar las soluciones que mas se adecuen al mismo.

La elección de componentes debe responder tanto a factores de diseño como coste, tamaño o consumo, como a la posibilidad del propio laboratorio de conseguirlos y la disponibilidad de las herramientas para su programación e implementación en el sistema final. Además, cualquier decisión tomada en un apartado implicará restricciones en el diseño del resto.

2.1 Adquisición de datos

La finalidad de un sistema de instrumentación es la de estimar los valores de las magnitudes físicas, presentando el resultado de la estimación a un observador. Para este fin se usan los **transductores**, definidos cómo, un dispositivo o sistema que produce una señal eléctrica la cual es función de una magnitud física de entrada utilizando componentes sensibles que se comportan como elementos variables o como generadores de señal.

En el mercado existe cuatro grandes tipos de transductores de temperatura: **detectores de temperatura resistivos (RTD), termistores, sensores de IC y termopares**, de los cuales se comentarán las principales ventajas/desventajas de cada uno (9).

2.1.1 Detectores de temperatura resistivos (RTD)

Los **detectores de temperatura resistivos** se basan en la variación de la resistencia de un conductor en función de la temperatura, siendo el platino el material más empleado en la actualidad.

La variación de la resistencia del material se puede expresar cómo:

$$R_T = R_0\,(1 + \alpha T + \alpha_2 T^2 + \cdots + \alpha_n T^n),\ \ T\ en\ ^0C \qquad (1)$$

La utilización de un platino de alta calidad nos permite unas medidas más exactas y estables hasta los 500 ^{0}C, el cual encarece el detector. Además tiene efectos de autocalentamiento, la necesidad de alimentación y variaciones pequeñas de la resistencia.

2.1.2 Termistores

Al igual que las RTD, los **termistores** están basados en la variación de la resistividad del material, pero con la salvedad de la utilización de semiconductores en vez de conductores.

Son los detectores más sensibles, pero pierden la linealidad que presentan los RTD. Presentan altas velocidad de respuesta y bajo consumo.

Existen dos tipos de termistores: **NTC (Negative temperatura coefficent)** y **PTC (Positive temperatura coefficent)**. La resistencia disminuye/aumenta con la temperatura respectivamente.

2.1.3 Termopares

Los **termopares** son transductores de temperatura que generan una señal eléctrica sin necesidad de alimentación. Se basan en los *fenómenos termoeléctricos (efecto Seebeck)*. Su reducido precio y su amplia gama de temperatura hacen que sean los transductores de temperatura más usados.

2.1.4 Sensores de IC

Los tres tipos de transductores antes mencionados necesitan de circuitos de acondicionamiento de la señal para compensar las limitaciones que presentan. Los **sensores de circuito integrado** incorporan este circuito de acondicionamiento en su interior. Estos transductores, en gran parte, formados por diodos y transistores bipolares aprovechan las características de la unión PN y su dependencia con la temperatura.

	RTD	Termopar	Sensor de IC	Termistor
Ventajas	Mas lineal que el termopar El más sensible El más estable	Autoalimentado Barato Variedad formas físicas/temperaturas Simple Robusto	El más lineal El de mayor rendimiento Barato	Buen rendimiento Velocidad
Desventajas	Caro Pequeña ΔR Baja resistencia Necesita alimentación Autocalentamiento	No lineal Usa referencia El menos estable El menos sensible	Necesita alimentación Lento Limitación de configuraciones Autocalentamiento	No lineal Frágil Necesita alimentación Autocalentamiento

Tabla 2-1. Comparativa transductores de temperatura

Ya que proyecto tiene como principal objetivo la puesta en práctica de los conocimientos adquiridos durante el grado mediante la creación de un sistema real, es por ello que se hará uso de un transductor RTD, introducido en un **puente de Wheatstone**.

2.1.5 Puente de Wheatstone

El **puente de Wheatstone** es un circuito eléctrico cuyo fin es medir el valor de una resistencia desconocida a priori mediante una diferencia de tensión entre las dos ramas del circuito. Este está formado por 4 resistencias separadas 2 a 2 en cada rama.

En la figura 2-1 podemos ver una configuración de puente de Wheatstone en donde R_1, R_2, R_3 son conocidas y R_X es la resistencia que queremos conocer su valor. La relación que estas presentan se puede expresar como:

$$\frac{R_2}{R_1} = \frac{R_X}{R_3} \Rightarrow R_X = \frac{R_2}{R_1} \cdot R_3 \quad (2)$$

Cuando R_3 es igual a R_1, R_X es igual a R_2 en la condición de equilibrio que hayamos determinado la diferencia de tensión ente A y B será nula.

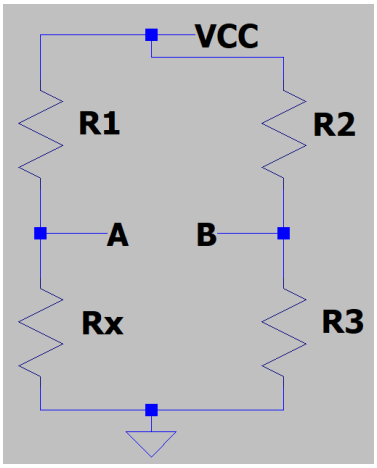


Figura 2-1. Puente de Wheatstone

La tensión diferencial V_{AB} quedaría de la forma:

$$V_{AB} = V_{CC} \cdot \frac{R_2 R_X - R_3 R_1}{(R_2 + R_3)(R_1 + R_X)} \tag{3}$$

Esta configuración ha sido aprovechada para, sustituyendo R_X por un transductor RTD, se haya podido captar variaciones de temperatura como diferencia de tensión entre el punto A y B.

$$V_{AB} = V_{CC} \cdot \frac{R_2 P_T - R_3 R_1}{(R_2 + R_3)(R_1 + P_T)} \tag{4}$$

2.2 Acondicionamiento de la Señal

En este apartado se explicará las pautas seguidas en el acordonamiento de la señal de tensión diferencial captada por el sensor.

La necesidad de acondicionar los datos tomados varía dependiendo del sensor utilizado, es decir, no existe un único circuito que puede proporcionar acondicionamiento para todos los sensores.

La mayoría de las señales necesitan una preparación previa antes de ser digitalizadas. A continuación, se ha enumerado los tipos de acondicionamiento más comunes:

- **Amplificación:** se usa para incrementar el nivel de tensión y adaptarlo de manera que se utilice eficientemente el rango del convertidor analógico-digital (ADC)
- **Atenuación:** es lo contrario a la amplificación se utiliza cuando los valores a digitalizar están fuera de los rangos de entrada del convertidor analógico-digital.
- **Filtrado:** se utiliza para rechazar ruido no deseado dentro de nuestras frecuencias de interés.

- **Aislamiento:** se utiliza cuando se trabaja con señales que está muy alejadas del rango de trabajo del convertidor, además de la atenuación.
- **Linealización:** necesaria cuando los sensores producen señales eléctricas que no están linealmente relacionadas con la magnitud física medida.

En el proyecto se ha hecho uso de una etapa amplificadora para incrementar el nivel tensión diferencial proporcionado por el puente y adaptarlo al rango de entrada del convertidor analógico-digital que se ha usado en la digitalización.

Para dicha tarea se presentan dos alternativas:

- Desarrollar un circuito de acondicionamiento **desde cero** a partir de amplificadores y componentes pasivos controlando la ganancia del circuito según se estime.
- Utilizar un **circuito integrado** que realice dicha tarea.

A pesar de que la primera opción puede considerarse la mejor, el esfuerzo de diseño y tiempo necesarios hacen que se haya optado por usar un circuito integrado. En el mercado, destacan tres grandes tipos de etapas amplificadoras: **amplificador diferencial, amplificador de instrumentación** y **amplificador de instrumentación con ajuste lineal de ganancia**.

Para el diseño se ha optado por la primera opción, ya que tienen un mejor precio y cumplen las necesidades del proyecto. Ya que se disponian de ellos en el laboratorio, se ha hecho uso del **LT1990** de *Linear Technology*.

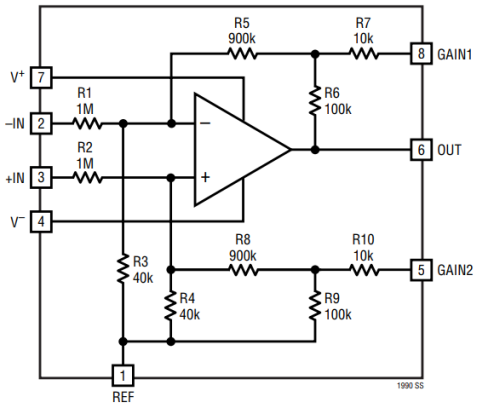


Figura 2-2. Diagrama del LT1990

El fabricante proporciona un datasheet detallado con las siguientes especificaciones:

	Condiciones	Valor típico
Ganancia	Pines 5 y 8 (sin conectar)	1
	Pines 5 y 8 = V_{REF}	10
Rango de entrada	Vs = 3V, 0V; V_{REF} =1.25V	(-5, 25) V
Voltaje de fuente mínimo	-	2.4 V
Voltaje mínimo a la salida	-	30 mV
Rechazo del modo común	Vs =3V, 0V; V_{REF} =1.25V; V_I = (-5, 25) V	68 dB
BW	G = 1	100 kHz
	G = 10	6.5 kHz

Tabla 2-2. Especificaciones LT1990

Adicionalmente el fabricante propone una configuración para ajustar a una ganancia intermedia en el rango de 1-10 añadiendo una R_{GAIN} entre los pines GAIN1 y GAIN2 dimensionada con la siguiente ecuación:

$$R_{GAIN} = \frac{180k}{G-1} - 20k\ [\Omega], siendo\ G\ la\ ganancia\ deseada \quad (5)$$

Cabe destacar que dicho circuito necesita una $V_{REF} \neq 0V$. Para este propósito se ha seleccionado una referencia de tensión, concretamente la **LM385Z-1.2**, que proporciona una tensión nominal de 1.235 V. Añadiendo error sistemático a la medida en forma de offset a la entrada del CAD que será compensado vía software.

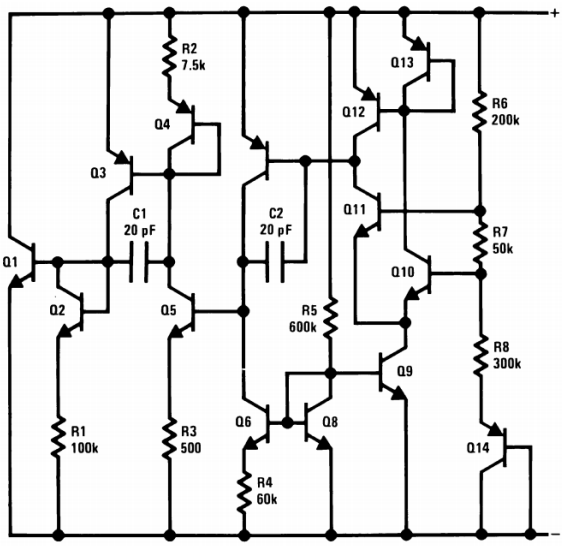


Figura 2-3. Esquemático LM385

El **LM385Z-1.2** se encuentra en diferentes encapsulados **T0-92**, **SOT-23, SOIC**.

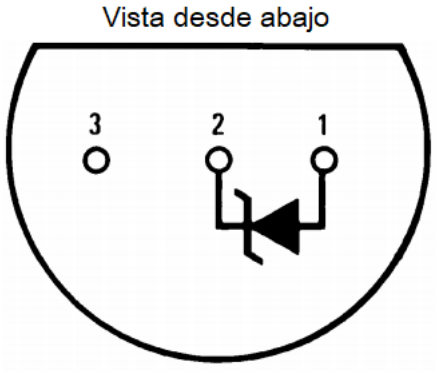


Figura 2-4. Diagrama del LM385Z-1.2

2.3 Dimensionamiento y cálculos teóricos

En este apartado se explicará el proceso y los cálculos teóricos seguidos en el proceso de diseño.

2.3.1 Transductor

En primer lugar se determinó el rango de salida del transductor de temperatura, conformado por el puente de Wheatstone.

La resistividad de la PT1000 ha sido calculada en base a la ecuación que nos proporciona el fabricante en el rango de temperatura del proyecto y atendiendo a la ecuación 1:

$$\left.\begin{array}{c} R_0 = 1000.000 \\ \alpha_1 = 3.909 \cdot 10^{-3} \\ \alpha_2 = -5.775 \cdot 10^{-7} \end{array}\right\} \Rightarrow \begin{cases} \xrightarrow[T=25℃]{} R_{PT} = 1085.710\ \Omega \\ \xrightarrow[T=45℃]{} R_{PT} = 1174.720\ \Omega \end{cases} \quad (6)$$

Dimensionando para los componentes seleccionados y haciendo referencia a la figura 2-1 y ecuación 4 del apartado 2.1.5.

$$V_{AB} = V_{CC} \cdot \frac{R_2 P_T - R_3 R_1}{(R_2 + R_3)(R_1 + P_T)} = 3.3V \cdot \frac{976\Omega \cdot P_T - 976\Omega \cdot 976\Omega}{(976\Omega + 976\Omega)(976\Omega + P_T)} =$$

$$\begin{cases} \xrightarrow[T=25℃]{} V_{AB} = 0.090\ V \\ \xrightarrow[T=45℃]{} V_{AB} = 0.155\ V \end{cases} \quad (7)$$

2.3.2 Rango de entrada del CAD

El rango de entrada del CAD varía según la placa de desarrollo, pudiendo estar comprendido entre 0-1V o 0-3.3V. Dada la controversia y la variedad de información acerca de las características de este se ha sometido a diferentes entradas de tensión con el fin de obtener la curva de calibración del convertidor de la placa.

Conociendo que el CAD es de 10 bits y obteniendo un valor de conversión de 1024 a 3V, el rango de entrada del convertidor ϵ (0, 3) V.

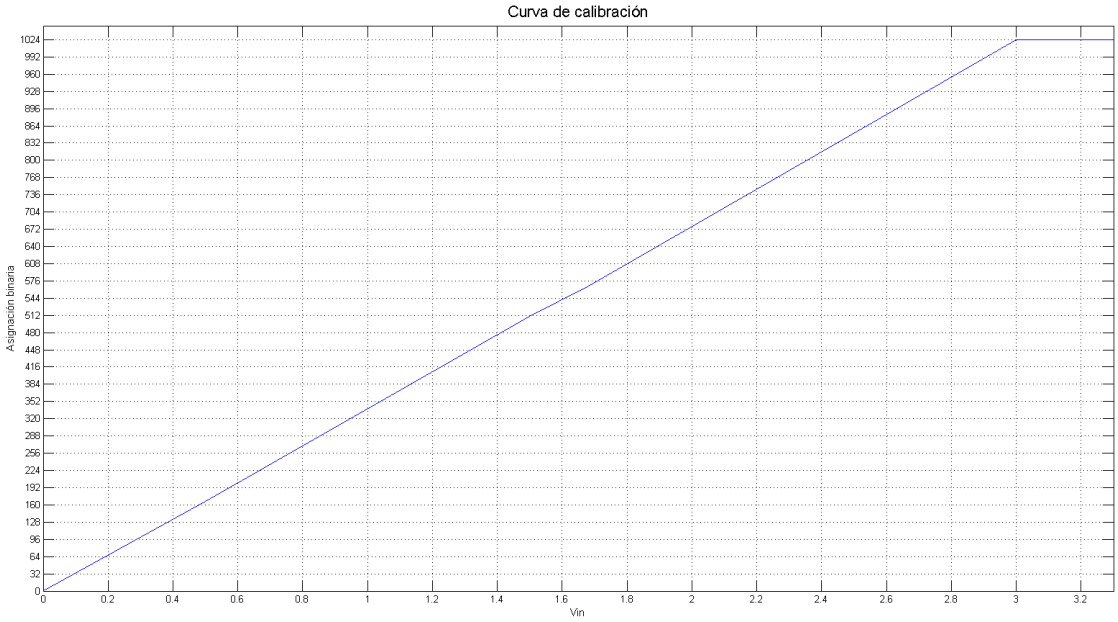


Figura 2-5. Curva de calibración del CAD

Siendo el rango de salida del transductor valores de tensión pequeños frente al rango de entrada del CAD, y para un mayor aprovechamiento de este, se ha realizado una etapa de amplificación como se ha comentado en el apartado 2-2 de acondicionamiento de la señal. Para el proyecto se ha selecionado una G=10, la ganancia máxima que nos ofrece el LT1990, atendiendo a la información que ofrece el fabricante:

$$V_{OUT} = G \cdot (V_{+IN} - V_{-IN}) + V_{REF} \tag{8}$$

siendo V_{REF} la tensión nominal del LM385Z-1.2, G la ganancia seleccionada y $(V_{+IN} - V_{-IN})$ la tensión diferencial V_{AB}, se obtuvo una tensión a la entrada del convertidor de:

$$V_{IN} = G \cdot V_{AB} + V_{REF} = 10 \cdot V_{AB} + 1.235V \Rightarrow$$
$$\Rightarrow \begin{cases} \xrightarrow{T=25℃} V_{IN} = 2.23\ V \\ \xrightarrow{T=45℃} V_{IN} = 2.78\ V \end{cases} \tag{9}$$

2.3.3 Obtención de la Medida

“El resultado de una medida viene afectado por causas no controladas en cualquier experimento, lo que implica una falta de repetibilidad total en esta, es decir, dos mediciones consecutivas del mismo observable, en idénticas condiciones experimentales, arrojan valores diferentes partiendo de un mismo nivel de precisión.” (10)

La obtención de la medida es el proceso por el cual se asigna un valor interpretable por un observador (ser humano, sistema de control, etc) a una magnitud física.

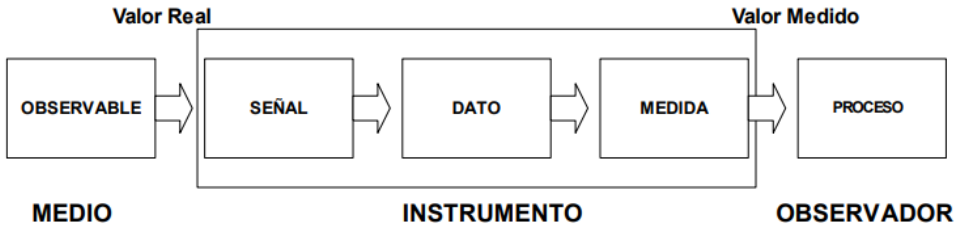


Figura 2-6. Diagrama del proceso de medida

La medida está formada por el **representante** y la **cota de error**.

$$Medida \equiv representante \pm cota\ de\ error$$

El representante es número que centraliza un *histograma de medida*, para el que se siguen dos criterios de selección:

- Criterio de moda: valor más repetido de los obtenidos.
- Criterio de media: media aritmética de los valores obtenidos.

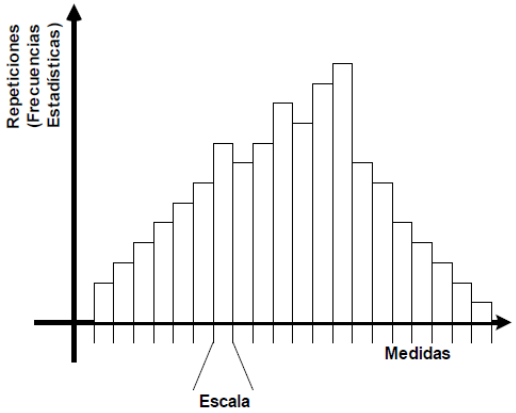


Figura 2-7. Histograma de medida

La cota de error es la desviación que presenta las muestras con respecto al representante. Existen dos enfoques:

- Cota absoluta: margen que ningún representante pasará en ningún caso.
- Cota estadística: define la variabilidad estadística del representante ($K \cdot \sigma$).
  - $K \equiv$ factor de cobertura.
  - $\sigma \equiv$ desviación típica.

A modo de resumen, existen dos enfoques para la obtención de la medida:

- **Enfoque operativo:** se usará un criterio de moda y una cota de error absoluta.
- **Enfoque estadístico:** se usará un criterio de media y una cota de error estadística.

2.3.4 Cálculos iniciales

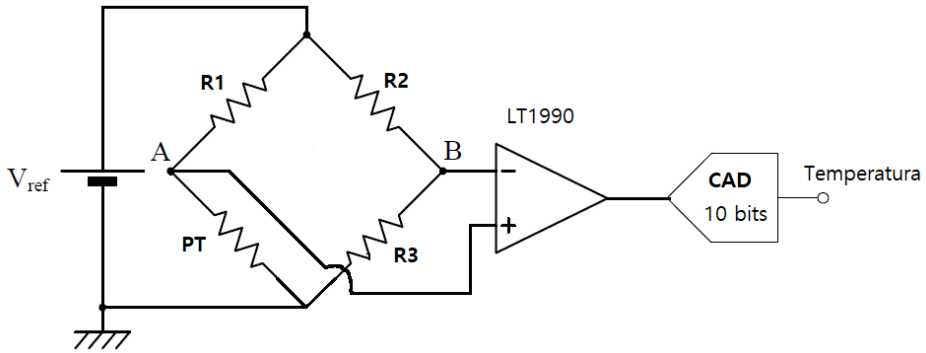


Figura 2-8. Diagrama diseño

Se ha usado como referencia el diagrama de la figura 2-8 y se ha caracterizado la temperatura de salida [T_{OUT}] como:

$$T_{OUT} = T_1 + \left(A_D V_{CC} \cdot \left[\frac{R_2 P_T - R_3 R_1}{(R_2 + R_3)(R_1 + P_T)}\right] - V_1\right) \cdot \frac{T_2 - T_1}{V_2 - V_1} \pm \Delta_{CAD} + e_{abs}^{edc} \cdot A_D \cdot \frac{T_2 - T_1}{V_2 - V_1} \quad (10)$$

siendo A_D la ganancia diferencial del amplificador. El término que multiplica el paréntesis dimensiona la conversión lineal del CAD presentado en la figura 2-5 y simplificado en la siguiente figura:

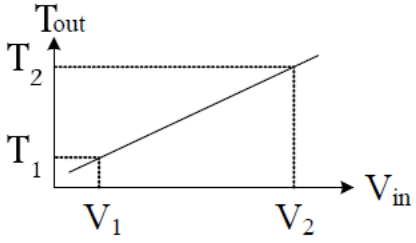


Figura 2-9. Relación E/S en el CAD

Los términos a la derecha del símbolo ± representan los errores de la medida:

- Error del CAD:

$$\Delta_{CAD} = q_{CAD} = \frac{SPAN}{2^{10} - 1} = \frac{3V}{2^{10} - 1} = 2{,}93\ mV \cdot \frac{45{,}00℃ - 25{,}00℃}{2.78V - 2.23V} = 0.11℃ \quad (11)$$

- Error debido al efecto de carga. Éste produce un error en la tensión V_{AB} que mide el amplificador:

$$e_{abs}^{e.d.c} = \left|V_{AB}^{real} - V_{AB}^{medido}\right| =$$

$$= \left| V_{CC} \cdot \frac{R_2 P_T - R_3 R_1}{(R_2 + R_3)(R_1 + P_T)} - V_{CC} \cdot \frac{R_2 P_T - R_3 R_1}{(R_2 + R_3)(R_1 + P_T)} \cdot \frac{R_{in}}{R_{in} + R_{th}} \right| \quad (12)$$

Siendo R_{in} la resistencia de entrada del amplificador y R_{th} la resistencia de salida del puente de Wheatstone. De forma,

$$e_{abs}^{e.d.c} = V_{CC} \cdot \frac{R_2 P_T - R_3 R_1}{(R_2 + R_3)(R_1 + P_T)} \cdot \frac{R_{th}}{R_{in} + R_{th}} = 0.000159\, V = 161\ \mu V \quad (13)$$

El **error total en la medida** se ha calculado como la contribución de ambos errores. Puesto que el debido al efecto de carga es en valor absoluto, ambos errores contribuyen con signo positivo:

$$\Delta_{total} = \sum_i \Delta x_i \cdot \left| \frac{dError}{dx_i} \right| = \Delta_{CAD} + e_{abs}^{edc} \cdot \left| A_D \cdot \frac{T_2 - T_1}{V_2 - V_1} \right| =$$
$$= 0.11℃ + 161\ \mu V \cdot \left| 10 \cdot \frac{20}{0.55} \right| = 0.17\ ℃ \quad (14)$$

El representante de la medida se obtiene ignorando los errores, por lo quedaría:

$$T_{OUT} = T_1 + (V_{IN} - V_1) \cdot \frac{T_2 - T_1}{V_2 - V_1} =$$
$$\begin{cases} \xrightarrow[T=25℃]{} T_{OUT} = 25 + (2.23 - 2.23) \cdot \dfrac{45 - 25}{2.78 - 2.23} = 25℃ \\ \xrightarrow[T=45℃]{} T_{OUT} = 45 + (2.78 - 2.23) \cdot \dfrac{45 - 25}{2.78 - 2.23} = 45℃ \end{cases} \quad (15)$$

Por lo tanto, el **error porcentual** en la medida obtenido (para el caso más desfavorable) es de:

$$e\% = 100 \cdot \frac{|Error_{total}|}{Valor_{real}} = 100 \cdot \frac{0.17}{45} = 0.38\ \% \quad (16)$$

Finalmente la medida obtenida sería de la forma:

$$T_{out} = T_{in} \pm 0.17\ ℃ \quad (17)$$

2.4 Tratamiento de las muestras

Hoy en día, la mayoría de los datos que se manejan se encuentran en el mundo digital, para ello es necesario digitalizar los datos para que un micro-controlador pueda interpretarlos. Dichos datos requieren un tratamiento para considerarlos como datos fiables y exigen el uso de la estadística debido a la variabilidad intrínseca de la medida (errores sistemáticos y aleatorios). (11)

Se ha efectuado un estudio estadístico de la medida con el fin de estimar el número de muestras necesarias para la obtención de un valor fiable en la medida.

Como se ha comentado en la conclusión de la introducción del apartado 2-3-3, en la realización de un estudio estadístico, la cota error asociada al representante viene dada por **K·σ**. Ya que para la obtención de la desviación típica del muestral habría que tomar infinitas muestras, algo irrealizable, se hará una estimación de esta con el correspondiente error asociado.

El **teorema central del límite asegura** que sea una población de muestras **Z** con media μ y varianza σ^2, sin ser necesariamente gaussiana, la variable normalizada de orden $Z = \frac{x-\mu}{\sigma/\sqrt{n}}$ es asintóticamente gaussiana. Siendo **x** el conjuntos de muestras, **μ** la media de los valores de **x**, **n** el número de muestras y **σ** la desviación típica. En el caso de no conocer **σ**, el muestral se puede aproximar mediante una distribución **t-student** de la forma $\tau = \frac{x-\mu}{s/\sqrt{n-1}}$, siendo **s** la desviación típica del muestral. Conforme el número de muestras aumenta (n > 30) la distribución t-student es idéntica a la normal.

Adicionalmente, haciendo referencia al **teorema de Chebyshev**, se puede asegurar que la probabilidad de que una muestra esté fuera del intervalo $(\mu - \varepsilon, \mu + \varepsilon)$ es despreciable si el cociente σ/ε es suficientemente pequeño, es decir:

$$P(|x - \mu| \geq \varepsilon) \leq \frac{\sigma^2}{\varepsilon^2} \tag{18}$$

2.4.1 Cálculos

Atendiendo a lo expuesto en la introducción de este apartado se ha calculado la varianza de la medida debido a los errores aleatorios introducidos por la referencia de tensión del circuito y la variabilidad estadística del CAD de la forma:

$$\sigma_{out}^2 = \sum_i \sigma_{x_i}^2 \cdot \left|\frac{\partial T_{out}}{\partial x_i}\right|^2 + \sigma_{CAD}^2 \tag{19}$$

Haciendo uso de la ecuación 10 y dado que la referencia de tensión del circuito presenta un error de 99 µV y el CAD variabilidad estadística de 0.11℃ [anexo B]:

$$\sigma_{out}^2 = \sigma_{V_{cc}}^2 \cdot \left| A_D \cdot \left[\frac{R_2 P_T - R_3 R_1}{(R_2 + R_3)(R_1 + P_T)} \right] \cdot \frac{T_2 - T_1}{V_2 - V_1} \right|^2 + \sigma_{CAD}^2 \rightarrow$$

$$\xrightarrow[T=45℃]{} \sigma_{out}^2 = (9.9 \cdot 10^{-5})^2 \cdot |10 \cdot 0.040 \cdot 36.364|^2 + 0.110^2 \rightarrow \quad (20)$$

$$\sigma_{out}^2 = 2.074 \cdot 10^{-6} + 0.0121 = 0.0121\ ℃^2 \rightarrow \sigma_{out} = 0.11℃$$

Con la obtención de la varianza se ha realizado una estimación de la probabilidad de obtener un error en la medida inferior a 0.1℃. Tomando como punto de partida que la variabilidad total es gaussiana y un muestral Z del diseño con solo una muestra:

$$\left.\begin{matrix} \varepsilon = 0.1℃ \\ \sigma = 0.11℃ \end{matrix}\right\} P\left(\frac{-\varepsilon}{\sigma} < z < \frac{\varepsilon}{\sigma}\right) = \quad (21)$$

$$P(-0.91 < z < 0.91) = 81{,}85\ \%$$

Puesto que el valor obtenido tras este análisis es pobre, se pasó a calcular el número de muestras necesarias para que la probabilidad de obtener un error inferior al 0.1℃ sea del 99%. Es decir, se ha buscado promediar n muestras para obtener dicha probabilidad.

Se consideró como punto de partida una aproximación a la normal:

$$\left.\begin{matrix} c = \frac{\varepsilon\sqrt{n}}{\sigma} \\ 99\% \rightarrow c = 2.58 \end{matrix}\right\} \rightarrow 2.58 = \frac{\varepsilon\sqrt{n}}{\sigma} \rightarrow n = \left(2.58 \cdot \frac{\sigma}{\varepsilon}\right)^2 = \left(2.58 \cdot \frac{0.11}{0.1}\right)^2 \quad (22)$$

$$n = 8.05 \rightarrow n = 9$$

Puesto que el número de muestras necesarias es menor a 30, la aproximación inicial no es válida. Recurriendo a la distribución t-student:

n	$\frac{\varepsilon\sqrt{n-1}}{s}$	r	Límite de la tabla
12	3.02	11	3.106
13	3.15	12	3.055
14	3.28	13	3.012
15	3.40	14	2.977

Tabla 2-3. Analísis en distribución t-student

Se obtuvo que el número de muestras necesarias para obtener un error inferior al 0.1℃ con un 99% de probabilidad es n=13. Este dato sirvió de punto de partida para el sketch diseñado para placa NodeMCU usada en este proyecto.

2.5 Procesamiento de los datos

El **procesamiento de datos** es la etapa en la que se digitalizan los datos, previamente acondicionados a un sistema. Dichos datos se analizan y se reenvían en caso de que fuese necesario. Para esta tarea son comúnmente usados los **convertidores analógico-digitales**, dispositivos electrónicos que convierten una señal analógica en una señal digital, capaz de ser interpretada por un micro-controlador. Dicha conversión tiene 3 etapas:

- **Muestreo:** consiste en la obtención de muestras (valores) de la señal analógica de entrada a una frecuencia constante. El muestreo está basado en el **teorema de Nyquist** que es la base para la representación discreta de una señal continua en banda limitada. El teorema demuestra que si la frecuencia de muestreo es **dos veces superior a la mayor frecuencia de la señal continua**, esta se puede recuperar totalmente a partir de sus muestras.
- **Cuantificación:** consiste en la asignación de un único valor de salida para un pequeño rango de entrada. Esto como cabe pensar provoca una pérdida de información dependiendo del número de bits que tenga el convertidor. A mayor número de bits, más fiel será el valor de salida con el valor muestreado a la entrada.
- **Codificación:** es la traducción esos valores obtenidos durante la cuantificación a un código entendido por el sistema que usará estos datos. El código más usado es el **código binario.**

Como ocurría en el apartado de A*condicionamiento de la señal*, la mejor solución sería hacer una implementación propia de un sistema de procesamiento, pero en el mercado existen numerosos sistemas de procesamiento especializados en esta tarea, además de reducir nuestra tarea de diseño nos ahorraría pasar certificaciones específicas para nuestro dispositivo.

En el mercado existen multitud de SoC , dispositivos de reducido tamaño, bajo consumo y bajo coste que integran gran parte de los módulos de un computador. Dos de los SoC más utilizados en el mercado son las tarjetas de **Arduino** (UNO, nano, …) y los diferentes modelos del **ESP8266** (NodeMCU, WeMos D1 mini, ….). En nuestro proyecto se ha optado por el *NodeMCU v2 de Amica* que integra un módulo WiFi de serie y nos facilitará la tarea de la comunicación inalámbrica con el display, la cual se comentará en apartados posteriores.

2.5.1 ESP8266

El **ESP8266** es un SoC (system on chip) fabricado por la compañía china *Espressif* con dimensiones muy reducidas y con un precio en torno a los 3-5 dólares.

Posee un procesador de 32bits (L106 de *Tensilica*) que posee un consumo muy bajo y un tamaño de instrucción de 16bits. El ESP8266 no dispone de memoria Flash, por lo que puede variar en función del módulo en el que se monta ya que tiene que ser proporcionada por este.

Del catálogo del fabricante podemos obtener las principales características de este: (12)

Categories	Items	Parameters
Wi-Fi	Certification	Wi-Fi Alliance
	Protocols	802.11 b/g/n (HT20)
	Frequency Range	2.4G ~ 2.5G (2400M ~ 2483.5M)
	TX Power	802.11 b: +20 dBm
		802.11 g: +17 dBm
		802.11 n: +14 dBm
	Rx Sensitivity	802.11 b: –91 dbm (11 Mbps)
		802.11 g: –75 dbm (54 Mbps)
		802.11 n: –72 dbm (MCS7)
	Antenna	PCB Trace, External, IPEX Connector, Ceramic Chip
Hardware	CPU	Tensilica L106 32-bit processor
	Peripheral Interface	UART/SDIO/SPI/I2C/I2S/IR Remote Control
		GPIO/ADC/PWM/LED Light & Button
	Operating Voltage	2.5V ~ 3.6V
	Operating Current	Average value: 80 mA
	Operating Temperature Range	–40°C ~ 125°C
	Package Size	QFN32-pin (5 mm x 5 mm)
	External Interface	-
Software	Wi-Fi Mode	Station/SoftAP/SoftAP+Station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network)
	Software Development	Supports Cloud Server Development / Firmware and SDK for fast on-chip programming
	Network Protocols	IPv4, TCP/UDP/HTTP
	User Configuration	AT Instruction Set, Cloud Server, Android/iOS App

Figura 2-10. Características del ESP8266

Dentro de esta familia, el ESP12E es el módulo con SoC ESP8266 más utilizado en placas de desarrollo ya que en este módulo tenemos disponible los pines importantes del ESP8266. (13)

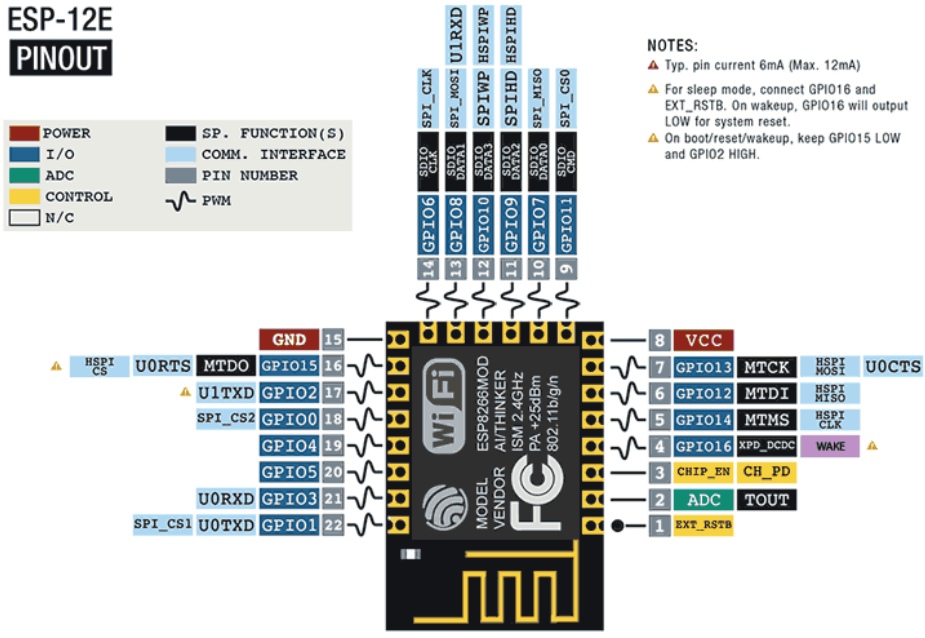


Figura 2-11. Pinout del ESP12E

Dichos módulos suelen estar montados en placas de desarrollo que facilitan mucho las tareas de programación de estos módulos. Es imposible seguir hablando de este SoC sin mencionar el entorno de desarrollo NodeMCU, nombre que también recibe una de las placas de desarrollo más populares del mercado.

2.5.2 NodeMCU (firmware)

En 2014, al poco de aparecer el ESP8266 se publicó en GitHub (una plataforma de desarrollo colaborativo, en la que se sube software de código abierto) se publicó el firmware **NodeMCU** que se podía grabar con facilidad en el ESP8266 permitiendo la programación en **Lua** (lenguaje basado en C y Perl, diseñado para ser muy ligero y con la facilidad para extender programas escritos a otros lenguajes C++, Java, etc). (14)

Aunque con la aparición del entorno de Arduino (C++) o MicroPython, el interés de Lua ha disminuido bastante, pero no se puede olvidar que a este lenguaje supuso el auge del ESP8266.

2.5.3 NodeMCU (hardware)

La placa de desarrollo **NodeMCU** está basada en el ESP12E, teniendo las características de este y añadiendo funcionalidades propias:

- Puerto micro-USB y conversor serie-USB
- LED y botón de reset integrados
- Alimentación y programación sencilla a través del puerto micro-USB

Existen varios modelos y versiones de estas placas de desarrollo siendo los tres principales fabricantes: *Amica, Lolin/Wemos y DOIT/SmartArduin.*

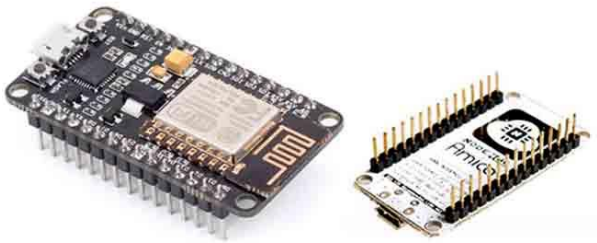


Figura 2-12. Placa NodeMCU de Amica

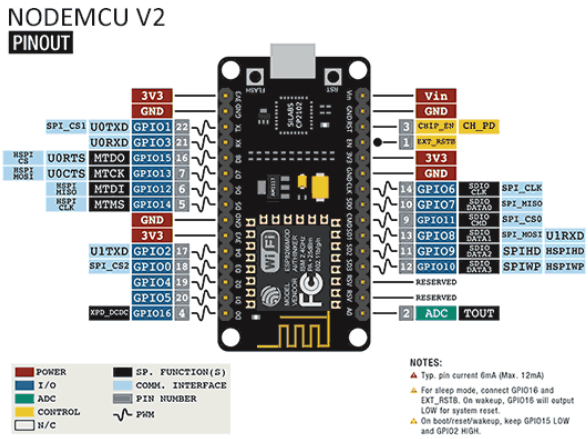


Figura 2-13. Pinout placa NodeMCU

Cualquier modelo o versión de estas placas de desarrollo presentan un convertidor analógico-digital de **aproximaciones sucesivas** con una resolución de 10 bits, en donde la principal diferencia es el rango de entrada de este. El ESP8266 presenta un rango de entrada de 0 a 1V, sin embargo la mayoría de las placas de desarrollo presentan un divisor de tensión a la entrada de este ampliando el rango de entrada de 0 a 3.3V. La conversión se efectuará a un número entre 0 y 1024 (2^{10}-1), siendo 0 la tensión mínima representada y 1024 la máxima. De la figura 2-5 se ha podido caracterizar la placa usada:

SPAN	3 V
Resolución	2.93 mV
Tasa de muestreo	50Khz
Nº de bits	10 bits

Tabla 2-4. Características CAD

2.5.4 Entorno de desarrollo

Existen diferentes formas de programar el ESP8266 (scripts lua, SDK de expressif en C++, Api de NodeMCU, MicroPython y Arduino IDE en C++), para el proyecto se ha decidido usar el IDE de Arduino por la gran cantidad de información que dispone de este y la gran comunidad que hay detrás de este. Está basado en Java y consta de un editor de código, compilador y depurador.

A la hora de usar el IDE de Arduino hay que realizar una serie de pasos para que este reconozca la placa NodeMCU:

1. Añadir la dirección donde bajar la información de nuestra placa y poder añadirla al gestor de placas, accediendo desde *Archivo* > *Preferencias* y añadiendo la URL *"http://arduino.esp8266.com/stable/package_esp8266com_index.json*".

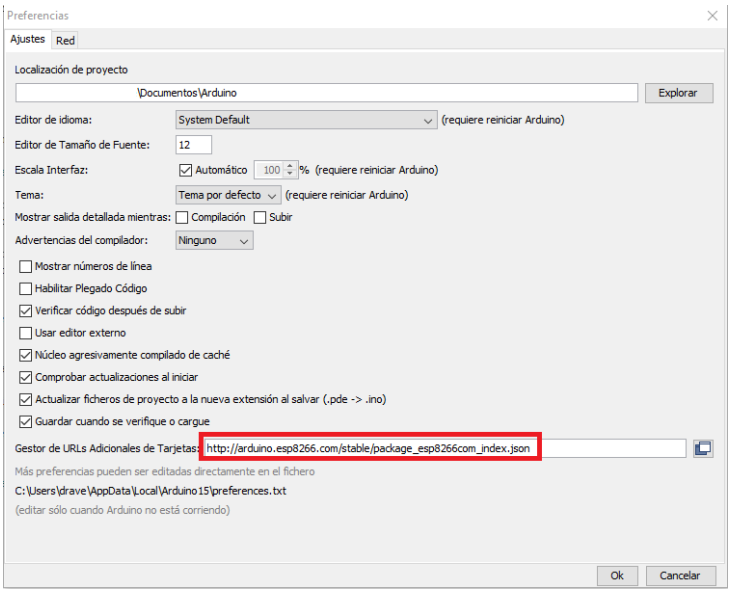


Figura 2-14. Añadir URL librería

Se accede al menú de gestión de tarjetas.

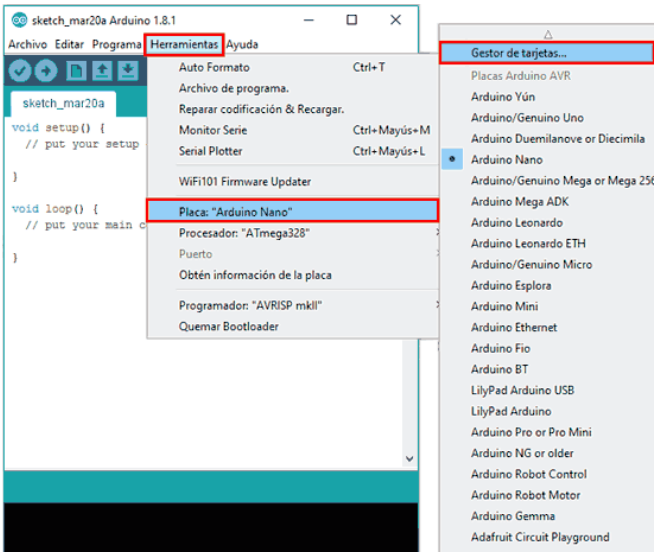


Figura 2-15. Menú de gestión de tarjetas

2. Se busca la librería y se instala.

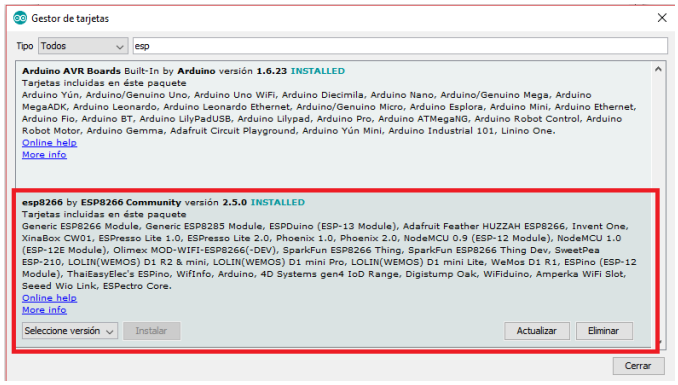


Figura 2-16. Instalación de la librería

3. Se selecciona el módulo y se habrá configurado el entorno para cargar los sketch en la placa.

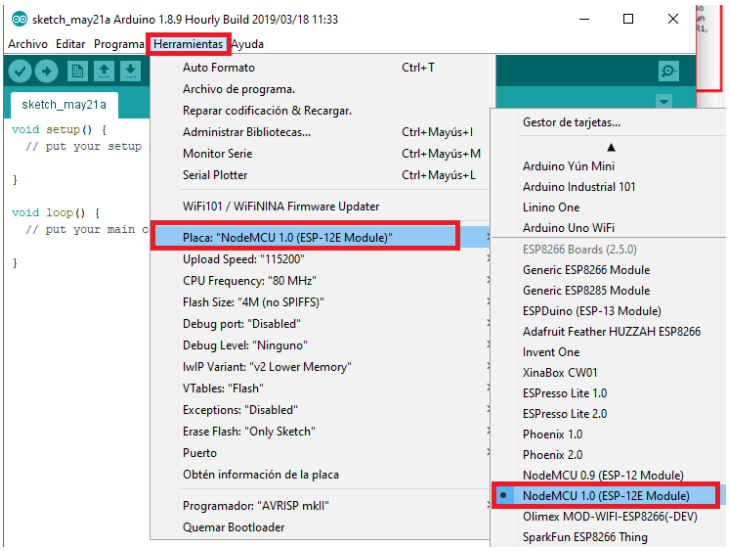


Figura 2-17. Selección del modelo

Cualquier código creado con Arduino consta de dos funciones:

- **void setup():** En esta función se colocan las líneas que solo se ejecutarán una vez (configuración de los pines, scripts en java, etc). Es la primera función que se ejecuta al arrancar el módulo.
- **void loop():** Esta función se ejecutará de forma continuada hasta la desconexión del módulo, es decir, cuando llegue a la última línea volverá a la primera y así sucesivamente.

Una vez finalizado el código a implementar la carga de este se realiza pulsando el *botón de subida* del IDE, durante la que se realizará una depuración del código y la propia carga de este.

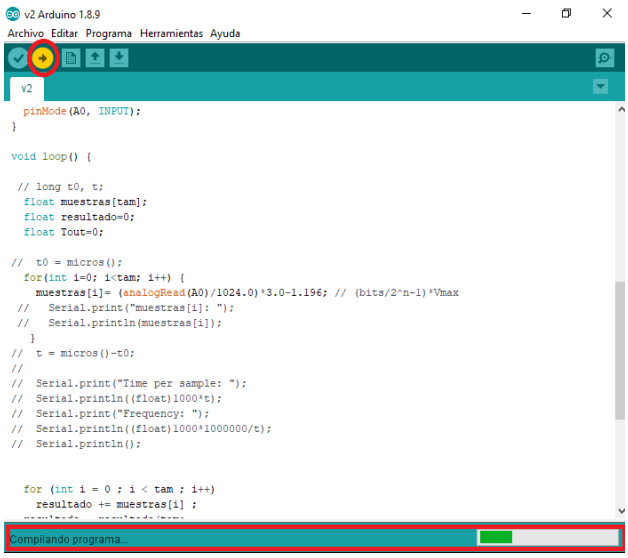


Figura 2-18. Depuración del código

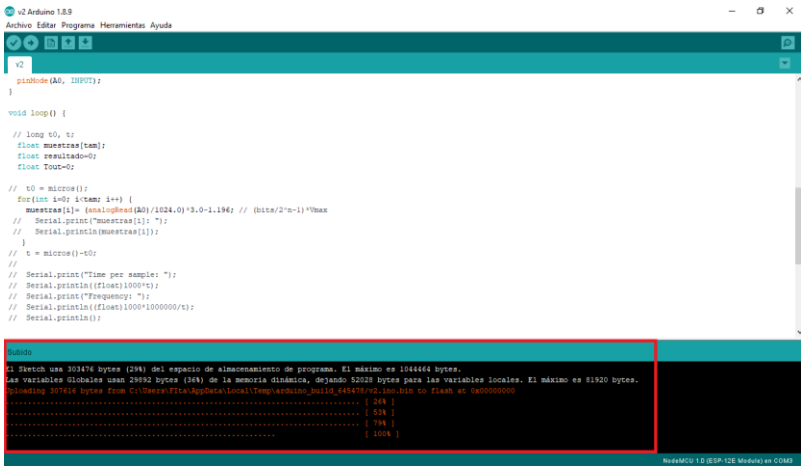


Figura 2-19. Carga del código

2.5.5 Sketch del proyecto

En este apartado se explicará cómo se ha planteado el procesamiento de datos que se ha llevado a cabo en la placa de desarrollo. El sketch se iniciará al alimentar la placa sin condición para salir del reposo:

- **Inicio de la comunicación serie**: se iniciará la comunicación con el monitor serie interno del IDE de Arduino a una velocidad de 115200 baudios.
- **Creación del punto de acceso Wi-Fi:** habilitaremos el modo punto de acceso Wi-Fi del ESP12-E que se usará para realizar la comunicación inalámbrica entre el servidor Web y la app.
- **Asignación IP:** asignaremos una IP a nuestro punto de acceso. Por defecto se asigna "192.168.4.1".
- **Inicio servidor web:** iniciaremos un servidor web alojado en la placa de desarrollo con IP 192.168.4.1 y escuchando peticiones en el puerto 80.
- **Creamos un socket web:** iniciaremos un socket web (TCP) asociado al servidor web, concretamente, el puerto 8080.

- **Captura entrada analógica:** capturaremos el número de muestras estimado en el apartado [nº apartado cálculos] y las iremos almacenando en un array para su posterior tratamiento.
- **Promediado y cálculo de la medida:** Realizaremos la media aritmética de los valores contenidos en el array, realizando un sumatorio de los valores del mismo y dividiendo por el número de muestras, así como el cálculo de la temperatura medida.
- **Envío de la medida al socket:** el servidor web enviará por difusión el valor de la medida.

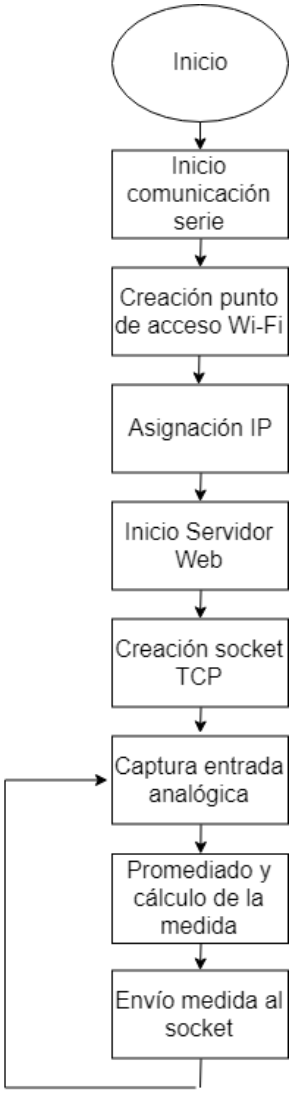


Figura 2-20. Digrama flujo del programa

3 IMPLEMENTACIÓN DEL SISTEMA

Una vez establecidos todos los parámetros necesarios a seguir en el diseño se llevó a cabo la implementación real, usando placa de pruebas y probando cada sección por separado, en la medida de lo posible. [Anexo C]

En la figura 3-1 se muestra el diseño final del proyecto en la que se encuentran diferenciadas por colores las diferentes partes del mismo. Como alimentación se ha usado los propios pines de 3.3V de los que dispone la placa.

Durante el desarrollo del prototipo se han realizado pruebas con la aplicación *de visualización*, desarrollada conjuntamente. Estas han permitido tanto descubrir errores en la implementación del código como asegurar el correcto funcionamiento del sistema completo.

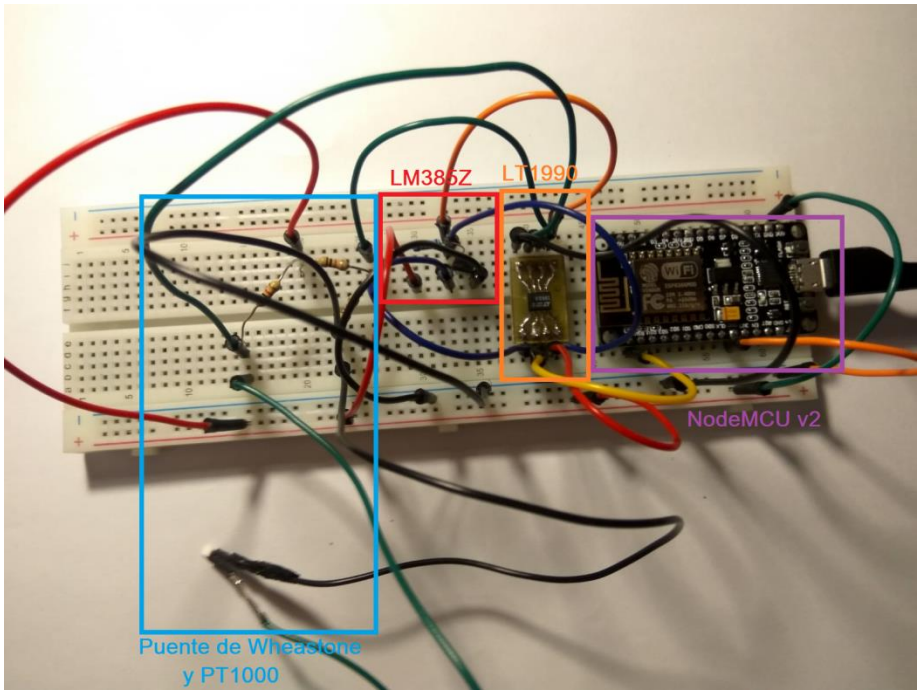


Figura 3-1. Diseño completo

3.1 Visualización de la medida

En este apartado se comentará las decisiones tomadas a la hora de representar los valores de la medida.

La visualización se ha llevado a cabo a través de una aplicación Android alojada en un Smartphone, con el fin de cumplir el requisito de visualizar los datos de inalámbricamente. Ya que el objetivo del proyecto se centra en la creación de una app, se ha usado el entorno de desarrollo App Inventor con el fin de agilizar la creación de esta.

3.1.1 App Inventor

App Inventor es un entorno de desarrollo creado por Google Labs para la creación de aplicaciones destinadas al sistema operativo Android. De forma visual y a partir de un conjunto de herramientas básicas, el usuario puede ir enlazando una serie de bloques a modo de *puzzle* para crear la aplicación. Es un sistema totalmente gratuito y se puede acceder a través de la misma página. Las aplicaciones creadas están limitadas por su simplicidad, aunque permite realizar multitud de proyectos que cumplen abarcan las funcionalidades básicas de un dispositivo móvil. (15)

Presenta una interfaz gráfica muy intuitiva que se divide en dos grandes secciones:

- **Diseño gráfico:** es la sección que nos permite configurar la parte visual de nuestra configuración, así como crear los objetos a los que asignaremos código en la siguiente sección.

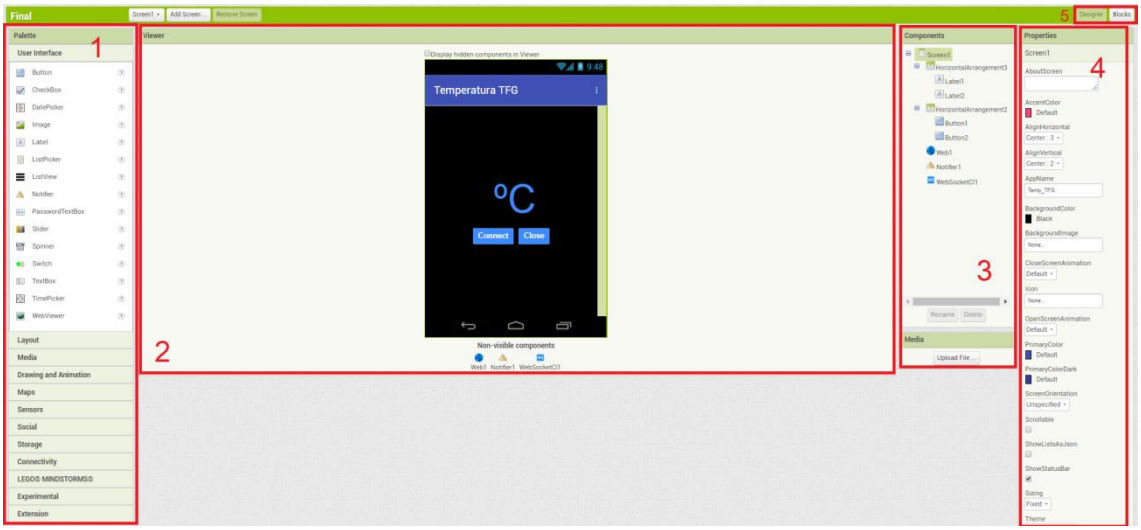


Figura 3-2. Entorno diseño gráfico

1. Menú de selección de los objetos que se podrán añadir a nuestro diseño.
2. Visualizador en el que se irá viendo el resultado gráfico final de la aplicación.
3. Índice de los componentes que se han ido añadiendo.
4. Menú de propiedades del objeto que se ha seleccionado, en el que se podrá modificar los parámetros de este.
5. Botones para cambiar en el modo de diseño gráfico o el modo programación.

- **Modo programación:** es la sección en la que podemos asignar código a los objetos seleccionados añadiendo así las funcionalidades deseadas.

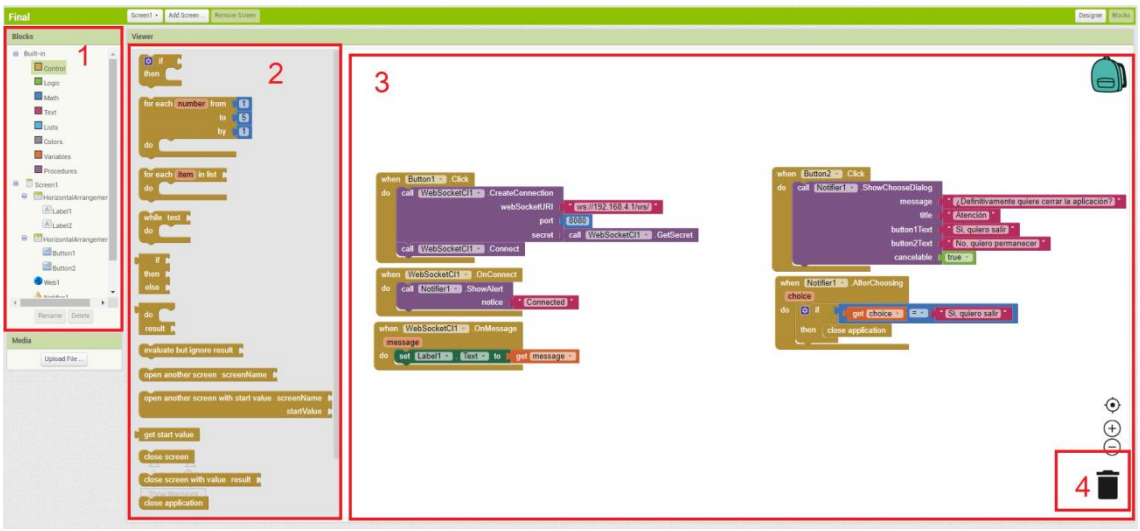


Figura 3-3. Entorno programación

1. Menú de selección de selección de los bloques generales y propios de cada objeto que se usarán para la asignación de código a estos.
2. Menú deslizante que aparecerá al seleccionar cualquiera de los bloques antes mencionado y nos mostrará los posibles bloques asociados a dicha elección.
3. Espacio donde se colocarán los bloques deseados a modo de puzle con el fin de crear estructura de programación del lenguaje usado.
4. Espacio en donde si arrastramos cualquier bloque podremos eliminarlo.

Por último, el entorno dispone de un sistema de compilación muy cómodo mediante el cual podemos exportar el proyecto a través de un código QR directamente al dispositivo móvil o un archivo *".apk"* que se puede cargar manualmente en este.

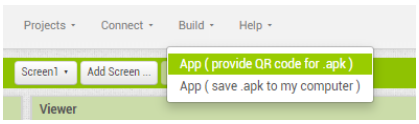


Figura 3-4. Compilación app

Referente a la aplicación del proyecto, se ha realizado de una forma simple la conexión con el servidor Web creado en la placa de desarrollo, la lectura del socket abierto en este y la representación de los datos leídos.

Partiendo de un estado de reposo permanece en dicho estado hasta que se pulsa el botón *"connect"*, momento en el cual la aplicación manda una petición al servidor para conectarse y poder leer el puerto previamente creado. Tras la lectura de los datos se muestran en la parte central de la pantalla del Smartphone.

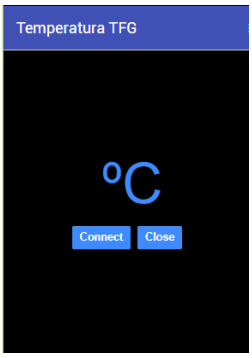


Figura 3-5. Aplicación temperatura

Adicionalmente, dispone de un botón de “close” con el que tras un cuadro de diálogo podremos cerrar la aplicación.

En la siguiente figura se muestra, a modo de resumen, el funcionamiento de la aplicación.

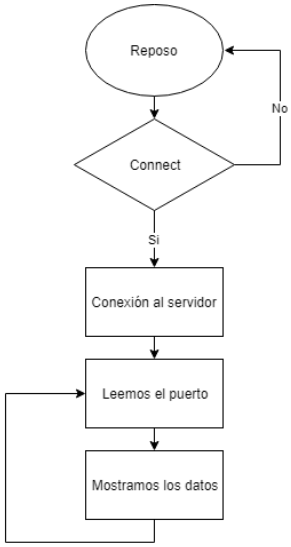


Figura 3-6. Diagrama de flujo de la app

3.2 Calibración del instrumento

Se denomina **calibración** al proceso por el que con un patrón de referencia (o estándar) se comparan las medidas realizadas por un instrumento y dicho patrón de una magnitud física. Según la Oficina Internacional de Pesas y Medidas, la calibración es "una operación que, bajo condiciones específicas, establece en una primera etapa una relación entre los valores y las incertidumbres de medida provistas por estándares e indicaciones correspondientes con las incertidumbres de medida asociadas y, en un segundo paso, usa esta información para establecer una relación para obtener un resultado de la medida a partir de una indicación".

La calibración del diseño se ha hecho en torno a dos valores, 0℃ y 100℃. Los patrones para estos valores serán:

- 0℃: Se ha tomado un vaso el cual se llenó de hielo, se le añadió agua y se dejó reposar 10min. Tras este tiempo, por equilibrio térmico la masa de agua próxima al hielo o entre estos se puede

asegurar que está a 0℃. Se introdujo el sensor en dicha masa de agua, se dejó 5min para que se estabilizase la medida y se comenzó a registrar las medidas.

- 100℃: Usando un criterio similar al anteriormente mencionado se calentó agua hasta que comenzó a hervir, en ese momento podemos asegurar que la masa de agua en la superficie (masa burbujeante) se encuentra a 100℃, momento en el cual empieza a evaporarse. Se introdujo el sensor en dicha masa de agua y se dejó 5min para que se estabilizase la medida y se comenzó a registrar las medidas.

El objetivo de este apartado es cuantificar el error cometido en la medida al partir de los cálculos teóricos realizados. Usando como referencia las ecuaciones [nº ec. calculo resistencia y Vab del apartado de transducor-dimensionamiento // nº ec. vout apart. rango entrada cad] se obtuvo los valores teóricos asociados a las temperaturas de los patrones con el fin de realizar la camparación:

$$V_{IN} = G \cdot V_{AB} + V_{REF}$$

$$= 10 \cdot V_{AB} + 1.235V = \begin{cases} \xrightarrow[T=0℃]{} 0.025 + 1.235 = 1.26V \\ \xrightarrow[t=100℃]{} 0.285 + 1.235 = 1.52V \end{cases} \quad (23)$$

Para el calibrado se ha usado el amplificador con ganancia unidad para que al medir los 100℃ la salida del amplificador no sature, ya que este está alimentado a 3.3V.

Las mediciones de los patrones se efectuarán a la salida del amplificador, donde se han obtenido valores de $0.133 \pm 0.05\ V$ para la temperatura de 0℃ y $0.47 \pm 0.03\ V$ para los 100℃.

En el caso de nuestro proyecto se ha ajustado tomando como referencia los datos obtenidos en los 0℃, ya que al volver a las condiciones reales de nuestro proyecto (G=10) no se pudo verificar los resultados en la medida de 100℃ debido a lo anteriormente comentado.

Con los resultados de los patrones y haciendo uso de la ecuación [ec. tout cálculos iniciales-calibración] se pudo observar que nuestro diseño sufre una desviación en la temperatura de aproximadamente $-5.6℃$. Conociendo esto, se procedió a corregir dicho offset en el sketch de la placa y se pasó a realizar una segunda toma de medidas del patrón.

En una segunda comparación, en esta ocasión con el sistema completo y tras la estabilización de la medida, se obtuvo el siguiente histograma:

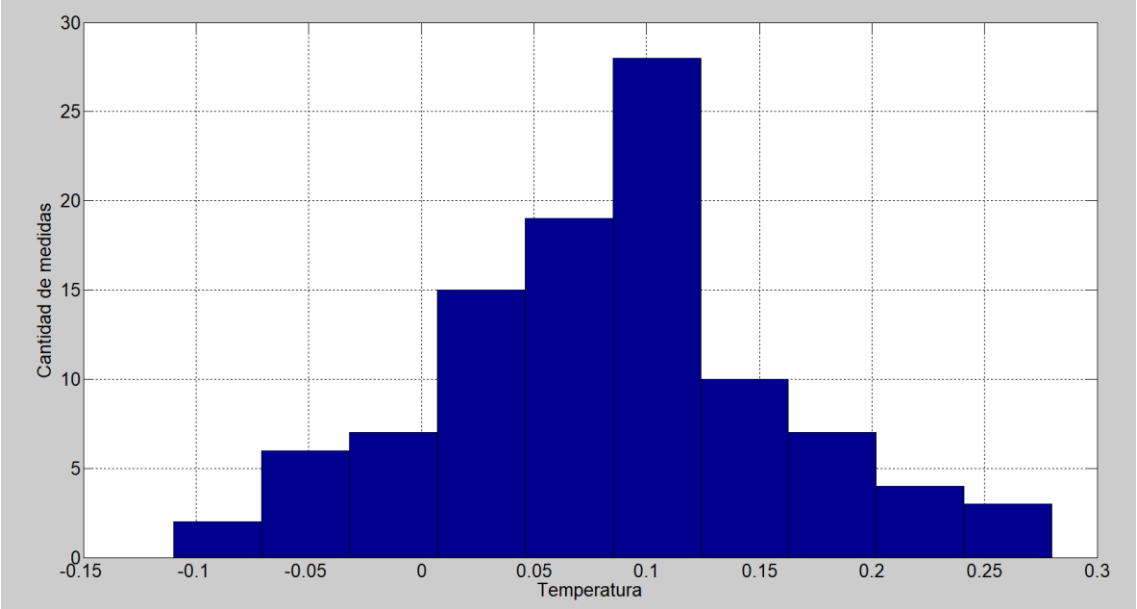


Figura 3-7. Histograma de la medida

Dicho histograma se ha realizado a partir de 101 muestra del cual se pudieron estimar las siguientes características de la medida:

$$\mu = \frac{\sum valor\_medida}{n° \; de \; medidas} = 0.08 \; ℃ \tag{24}$$

$$\sigma = \sqrt{\frac{\sum(\mu - valor\_medida)^2}{n° \; de \; medidas}} = \; 0.07 \; ℃ \tag{25}$$

El histograma de medida, tras el calibrado, presenta una media de 0.08℃ y una desviación típica de 0.07℃.

4 RESULTADOS

Una vez concluida la impletación del sistema, sólo quedaría la comprobación del correcto funcionamiento del sistema completo. Además se ha presentado un listado con las características principales de nuestro sistema.

4.1 Sistema completo

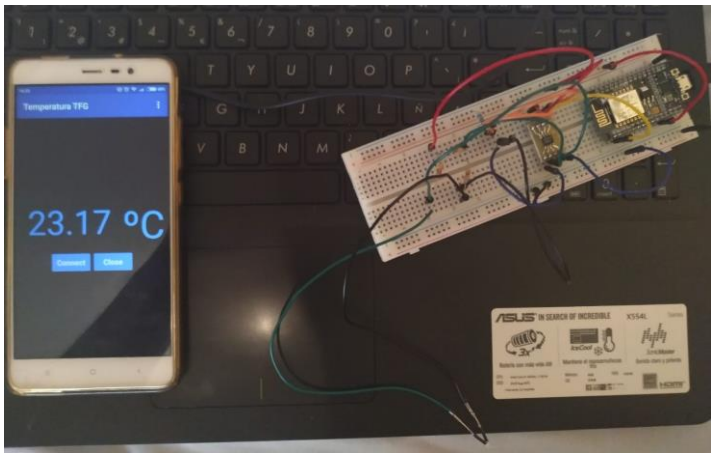


Figura 4-1. Temperatura en reposo

Como podemos ver en la figura 4-1, el sistema muestra valores de temperatura razonables para una habitación domestica.

En la siguiente figura se aprecia como ante la colocación de los dedos en la PT1000 se podruce un incremento de la temperatura. [anexo D]

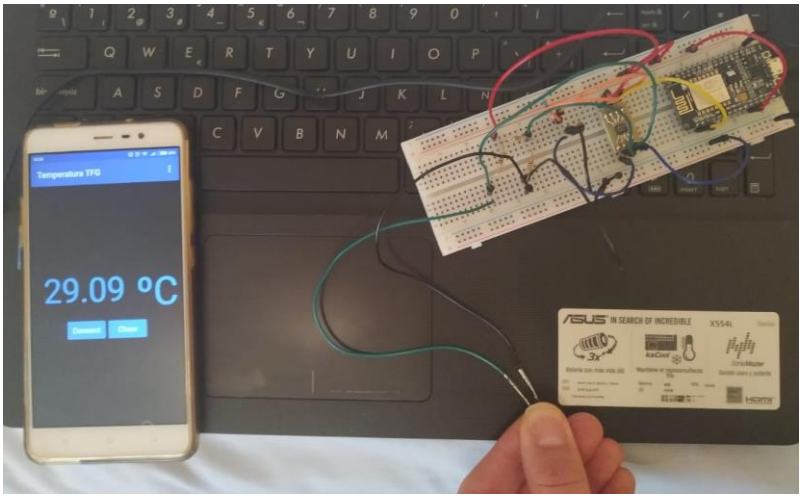


Figura 4-2. Temperatura de la mano

4.2 Características

En este apartado se presenta las características principales del diseño:

Rango de temperatura	0 - 63°C
Distancia máxima de funcionamiento[1]	10 m
Resolución	10 bits
Precisión	0.11°C
Alimentación	5V o USB

Tabla 4-1. Tabla de características del dispositivo

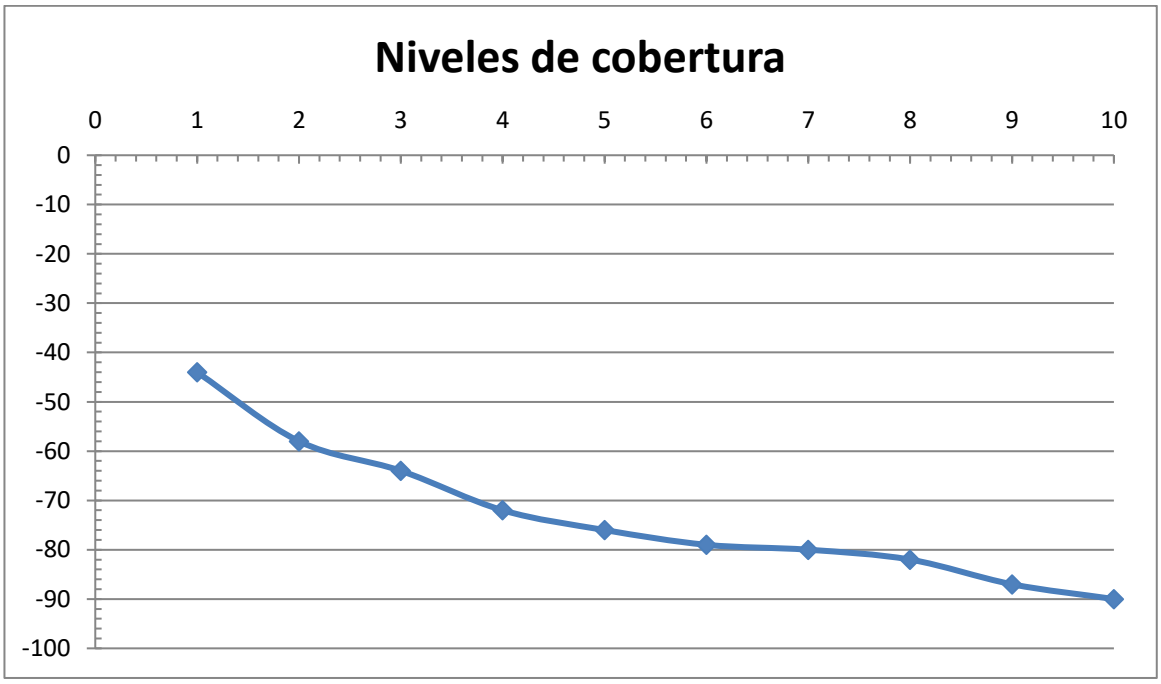


Figura 4-3. Cobertura del dispotivo metros-dBm

[1] Sin obstáculos entre el placa y el dispositivo.

5 ESTUDIO ECONÓMICO

Como capítulo final antes de llegar a las conclusiones se muestra el estudio económico de los de diferentes gastos sujetos al proyecto. Algunos costes aquí presentados se encuentran desarrollados en el Anexo D.

Concepto	**Coste (€)**
Placa de desarrollo NodeMCU v2	8,49
Componentes electrónicos	7,12
Material de labaratorio	4,55
TOTAL	20,16

Tabla 5-1. Coste del diseño

Los costes de los componentes pueden variar según el momento de compra, fabricante, tecnología y distribuidor, por lo que el coste total derivado en este proyecto debe considerarse como un gasto aproximado.

6 CONCLUSIONES

El objetivo de este proyecto era diseñar un dispositivo capaz de capturar temperatura, para ser procesadas, enviadas y mostradas en tiempo real en un monitor inlámbrico. El diseño parte como idea conceptual motivado por plasmar los conocimientos adquiridos durante el grado.

Cada una de las partes se ha desarrollado y probado de forma aislada en primera instacia, aunque la mayor parte del tiempo se ha invertido en hacer funcionar el sistema completo.

6.1 Conclusiones

La primera y más clara conclusión que se llega tras abordar este proyecto es que en ingeniería cualquier diseño, por trivial que parezca a priori, tiene su complejidad y necesita de tiempo de maduración.

En base a un diagrama conceptual y unas especificaciones, se ha diseñado cada uno de los bloques que compenen el proyecto. En primer lugar por separado, para después unirlos en un diseño final.

Se han planteado diferentes formas de operar y se ha elegido las que han resultado más convenientes para el proyecto en particular. Empezando por la comparativa de diferentes transductores de temperatura, diferentes etapas amplificadoras y sistemas SoC.

Este proyecto podría ser una parte de un proyecto más amplio en el que se recogiesen otras magnitudes del cuerpo humano como el pulso, la actividad celebral, etc, con el fin de usarse en estudios concretos.

Por último, centrandonos en los SoC, por su reducido coste, tamaño y su gran capacidad de integración, el ESP8266 es un SoC excelente para el mundo IoT y los sistemas embebidos, ofreciendo gran variedad de aplicaciones y posibildidades en un mercado que lleva en auge durante varios años y seguirá creciendo.

6.2 Lineas de mejora

De cara a nuevos desarrollos, una de las principales mejores sería la integración del sistema en una única placa realizando su diseó en entornos como Eagle o Altium Designer. Así como sustituir el sensor por uno comercial facilitando así su integración y la tarea de acondicionamiento de la señal o realizar el diseño intregado del mismo. Con el fin de poder realizar medidas en zonas del cuerpo fiables para la realización de estudios basados en este medida.

Adicionalmente, una optimización de la etapa amplificadora podría darnos un mayor aprovechamiento del rango de entrada del CAD, así como la incorporación de diferentes canales con el fin de muestrear otras magnitudes.

Respecto a la parte de programación, se podría realizar un optmización de código con el fin de aumentar la seguridad de la comunicación MCU-App y la velocidad de respuesta de este. Así como la creación de nuevas funciones para ampliar su versatilidad.

En el lado de la App, optar por una progración desde cero de la misma con el fin de tener el control total de esta, pudiendo así añadir nuevas funcionalidades en el caso de que se quiera añadir otras mediciones, realizar un registro de las mimas, etc.

ANEXO A – ANÁLISIS DE RIESGOS

Los posibles riesgos que puede enfretar el proyecto son:

- Retraso en la entrega
- Recorte del presupuesto
- Asuencia de los tutores
- Cambios de última hora
- Corrupción del código del programa
- Destrucción total/parcial del hardware

Riesgo		Probabilidad				
		1	2	3	4	5
Severidad	1					
	2		Recorte del presupuesto			
	3		Ausencia de los tutores	Retraso en la entrega		
	4			Cambios de última hora	Corrupción del código del programa	
	5				Destrucción del hardware	

ANEXO B – CÁLCULO DE LAS DESVIACIONES TÍPICAS

Haciendo referencia al datasheet del AMS1117 se ha obtenido que:

$$RMS\ Output\ Noise = 0.003\% \xrightarrow[V_{out}=3.3V]{} \sigma_{V_{cc}} = 99\ \mu V \tag{26}$$

El cálculo para el CAD se ha realizado atendiendo a:

$$LSB = \frac{SPAN}{2^{10}} \rightarrow \sigma_{CAD} = LSB \cdot \frac{45℃ - 25℃}{2.78V - 2.23V} = 0.11\ ℃ \tag{27}$$

ANEXO C – CRITERIOS DE ACEPTACIÓN

Matriz de verificación

La matriz de verificación del diseño es:

Requisito	Nombre del requirimiento	Verificacion				Nombre de la pruebra	Estado
		I	A	D	T		
D.2.1	Conmunicación MCU-Display				**X**	Test MCU - Display	Terminada
D.3.1	Visualización resultados				**X**	Test app	Terminada
D.4.1 / D.4.2	Salida Am. Inst.				**X**	Test A.I - MCU	Terminada
D.4.3	Puente sensor				**X**	Test puente Wheastone	Terminada

Plan de pruebas

El objetivo es verificar que el sistema cumple con las especificaciones con las que se ha diseñado.

Prueba 1: MCU-Display

Se comprobará que la conexión MCU-smartphone es correcta y que se reciben los datos correctamente. Con ayuda del monitor serie del IDE de Arduino se ha podido comprobar los datos enviados.

Prueba 2: Test app

Se volcará el código de la app en el Smartphone y se comprobará que el el formato y la visualización de los resultados cumple con los requisitos marcados.

Prueba 3: Amplificador-MCU

Con la ayuda de voltímetro se compróbara si los valores tras la amplificación son correctos y están dentros de los niveles del CAD (0 -3 V)

Prueba 4: Test puente sensor

Con la ayuda de un voltimetro se comprobará que la tensión diferencial en la entrada del Amplificador varía antes diferentes variaciones de temperatura atendiendo en primera aproximación al datasheet del fabricante.

ANEXO D – PRESUPUESTO

Componente	Nombre	Cantidad	Min. Compra	Precio (unidad)
ESP8266	XX	1	1	8.49 €
1K RESISTENCIA 1/4W 5%	R1,R2,R3	3	1	0.02 €
LT1990	Amplicador	1	2	2.595 €
Cabledo y placa de pruebas	XX	1	1	4.55 €
PT1000	PT	1	1	1.85 €
TOTAL				**20.16 €**

ANEXO E – CÓDIGO ESP8266

Código E.1 – main.ino

```
#include <ESP8266WiFi.h>                    // Librerias propias de la placa
#include <ESP8266WebServer.h>
#include <WebSocketsServer.h>              // Libreria adicional para el
manejo de sockets

// Declaración e inicialización de las variables globales
const int tam = 13;      // número de muestras calculadas en el apartado
2.4.1
const float t1= 25;
const float t2= 45;
const float v1= 0.99;
const float v2= 1.55;

// Configuración del AP
char * ssid = "TFG";
char * password = "prueba01";

// Arranque del servidor Web en el puerto 80
ESP8266WebServer server(80);

// Addición del socketTCP en el puerto 8080
WebSocketsServer webSocket = WebSocketsServer(8080);

void setup() {
  Serial.begin(115200);      // Establecimento de la conexión serie a 115200
  Serial.println("");
  WiFi.mode(WIFI_AP);           //Only Access point
  WiFi.softAP(ssid, password);  //Start HOTspot removing password will
disable security

  IPAddress myIP = WiFi.softAPIP(); //Get IP address
  Serial.print("HotSpt IP:");
  Serial.println(myIP);

  server.begin();
  Serial.printf("Web server started, open %s in a web browser\n",
WiFi.softAPIP().toString().c_str());
  webSocket.begin();
  pinMode(A0, INPUT);
}

void loop() {

// Declaración e inicialización de las variables locales
  float muestras[tam];
  float resultado=0;
  float Tout=0;

// Realización del muestreo
  for(int i=0; i<tam; i++) {
    muestras[i]= (analogRead(A0)/1024.0)*3.0-1.235; // (bits/2^n-1)*SPAN-
offset
  }
```

```
  for (int i = 0 ; i < tam ; i++)
    resultado += muestras[i] ;
  resultado = resultado/tam;

// punto de prueba intermedio
  Serial.print("resultado: ");
  Serial.println(resultado);

// Cálculo de la temperatura
  Tout = t1+(resultado - v1)*((t2-t1)/(v2-v1))+5.6;

// Visualización del resultado por el puerto serie
  Serial.print("Temperatura: ");
  Serial.println(Tout);

// Conversión a string para ser enviado por el socketTCP
  String json = " ";
  json += Tout;

// Envio por difusión a través del socket
  webSocket.broadcastTXT( json.c_str(), json.length());

  webSocket.loop();
  server.handleClient();
  delay(3000);
}
```

Código E.2 – WebSocketsServer.h

```
#ifndef WEBSOCKETSSERVER_H_
#define WEBSOCKETSSERVER_H_

#include "WebSockets.h"

#ifndef WEBSOCKETS_SERVER_CLIENT_MAX
#define WEBSOCKETS_SERVER_CLIENT_MAX  (5)
#endif



class WebSocketsServer: protected WebSockets {
public:

#ifdef __AVR__
        typedef void (*WebSocketServerEvent)(uint8_t num, WStype_t type,
uint8_t * payload, size_t length);
        typedef bool (*WebSocketServerHttpHeaderValFunc)(String headerName,
String headerValue);
#else
        typedef std::function<void (uint8_t num, WStype_t type, uint8_t *
payload, size_t length)> WebSocketServerEvent;
        typedef std::function<bool (String headerName, String headerValue)>
WebSocketServerHttpHeaderValFunc;
#endif

        WebSocketsServer(uint16_t port, String origin = "", String protocol =
"arduino");
```

```
        virtual ~WebSocketsServer(void);

        void begin(void);
        void close(void);

#if (WEBSOCKETS_NETWORK_TYPE != NETWORK_ESP8266_ASYNC)
        void loop(void);
#else
        // Async interface not need a loop call
        void loop(void) __attribute__ ((deprecated)) {}
#endif

        void onEvent(WebSocketServerEvent cbEvent);
        void onValidateHttpHeader(
                WebSocketServerHttpHeaderValFunc validationFunc,
                const char* mandatoryHttpHeaders[],
                size_t mandatoryHttpHeaderCount);


        bool sendTXT(uint8_t num, uint8_t * payload, size_t length = 0, bool
headerToPayload = false);
        bool sendTXT(uint8_t num, const uint8_t * payload, size_t length =
0);
        bool sendTXT(uint8_t num, char * payload, size_t length = 0, bool
headerToPayload = false);
        bool sendTXT(uint8_t num, const char * payload, size_t length = 0);
        bool sendTXT(uint8_t num, String & payload);

        bool broadcastTXT(uint8_t * payload, size_t length = 0, bool
headerToPayload = false);
        bool broadcastTXT(const uint8_t * payload, size_t length = 0);
        bool broadcastTXT(char * payload, size_t length = 0, bool
headerToPayload = false);
        bool broadcastTXT(const char * payload, size_t length = 0);
        bool broadcastTXT(String & payload);

        bool sendBIN(uint8_t num, uint8_t * payload, size_t length, bool
headerToPayload = false);
        bool sendBIN(uint8_t num, const uint8_t * payload, size_t length);

        bool broadcastBIN(uint8_t * payload, size_t length, bool
headerToPayload = false);
        bool broadcastBIN(const uint8_t * payload, size_t length);

        bool sendPing(uint8_t num, uint8_t * payload = NULL, size_t length =
0);
        bool sendPing(uint8_t num, String & payload);

        bool broadcastPing(uint8_t * payload = NULL, size_t length = 0);
        bool broadcastPing(String & payload);

        void disconnect(void);
        void disconnect(uint8_t num);

        void setAuthorization(const char * user, const char * password);
        void setAuthorization(const char * auth);

        int connectedClients(bool ping = false);

#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266) || (WEBSOCKETS_NETWORK_TYPE
== NETWORK_ESP8266_ASYNC) || (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP32)
        IPAddress remoteIP(uint8_t num);
#endif
```

```
protected:
        uint16_t _port;
        String _origin;
        String _protocol;
        String _base64Authorization; ///< Base64 encoded Auth request
        String * _mandatoryHttpHeaders;
        size_t _mandatoryHttpHeaderCount;

        WEBSOCKETS_NETWORK_SERVER_CLASS * _server;

        WSclient_t _clients[WEBSOCKETS_SERVER_CLIENT_MAX];

        WebSocketServerEvent _cbEvent;
        WebSocketServerHttpHeaderValFunc _httpHeaderValidationFunc;

        bool _runnning;

        bool newClient(WEBSOCKETS_NETWORK_CLASS * TCPclient);

        void messageReceived(WSclient_t * client, WSopcode_t opcode, uint8_t
* payload, size_t length, bool fin);

        void clientDisconnect(WSclient_t * client);
        bool clientIsConnected(WSclient_t * client);

#if (WEBSOCKETS_NETWORK_TYPE != NETWORK_ESP8266_ASYNC)
        void handleNewClients(void);
        void handleClientData(void);
#endif

        void handleHeader(WSclient_t * client, String * headerLine);

        /**
         * called if a non Websocket connection is coming in.
         * Note: can be override
         * @param client WSclient_t *  ptr to the client struct
         */
        virtual void handleNonWebsocketConnection(WSclient_t * client) {
            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader] no Websocket
connection close.\n", client->num);
            client->tcp->write("HTTP/1.1 400 Bad Request\r\n"
                    "Server: arduino-WebSocket-Server\r\n"
                    "Content-Type: text/plain\r\n"
                    "Content-Length: 32\r\n"
                    "Connection: close\r\n"
                    "Sec-WebSocket-Version: 13\r\n"
                    "\r\n"
                    "This is a Websocket server only!");
            clientDisconnect(client);
        }

        /**
         * called if a non Authorization connection is coming in.
         * Note: can be override
         * @param client WSclient_t *  ptr to the client struct
         */
        virtual void handleAuthorizationFailed(WSclient_t *client) {
             client->tcp->write("HTTP/1.1 401 Unauthorized\r\n"
                    "Server: arduino-WebSocket-Server\r\n"
                    "Content-Type: text/plain\r\n"
                    "Content-Length: 45\r\n"
                    "Connection: close\r\n"
```

```
                    "Sec-WebSocket-Version: 13\r\n"
                    "WWW-Authenticate: Basic realm=\"WebSocket Server\""
                    "\r\n"
                    "This Websocket server requires Authorization!");
            clientDisconnect(client);
        }

        /**
         * called for sending a Event to the app
         * @param num uint8_t
         * @param type WStype_t
         * @param payload uint8_t *
         * @param length size_t
         */
        virtual void runCbEvent(uint8_t num, WStype_t type, uint8_t *
payload, size_t length) {
            if(_cbEvent) {
                _cbEvent(num, type, payload, length);
            }
        }

        /*
         * Called at client socket connect handshake negotiation time for
each http header that is not
         * a websocket specific http header (not Connection, Upgrade, Sec-
WebSocket-*)
         * If the custom httpHeaderValidationFunc returns false for any
headerName / headerValue passed, the
         * socket negotiation is considered invalid and the upgrade to
websockets request is denied / rejected
         * This mechanism can be used to enable custom authentication schemes
e.g. test the value
         * of a session cookie to determine if a user is logged on /
authenticated
         */
        virtual bool execHttpHeaderValidation(String headerName, String
headerValue) {
            if(_httpHeaderValidationFunc) {
                  //return the value of the custom http header validation
function
                  return _httpHeaderValidationFunc(headerName, headerValue);
            }
            //no custom http header validation so just assume all is good
            return true;
        }

private:
        /*
         * returns an indicator whether the given named header exists in the
configured _mandatoryHttpHeaders collection
         * @param headerName String ///< the name of the header being checked
         */
        bool hasMandatoryHeader(String headerName);

};


#endif /* WEBSOCKETSSERVER_H_ */
```

Código E.2 – WebSocketsServer.cpp

```
#include "WebSockets.h"
#include "WebSocketsServer.h"

WebSocketsServer::WebSocketsServer(uint16_t port, String origin, String
protocol) {
    _port = port;
    _origin = origin;
    _protocol = protocol;
    _runnning = false;

    _server = new WEBSOCKETS_NETWORK_SERVER_CLASS(port);

#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266_ASYNC)
    _server->onClient([](void *s, AsyncClient* c){
        ((WebSocketsServer*)s)->newClient(new AsyncTCPbuffer(c));
    }, this);
#endif

    _cbEvent = NULL;

    _httpHeaderValidationFunc = NULL;
    _mandatoryHttpHeaders = NULL;
    _mandatoryHttpHeaderCount = 0;

    memset(&_clients[0], 0x00, (sizeof(WSclient_t) *
WEBSOCKETS_SERVER_CLIENT_MAX));
}


WebSocketsServer::~WebSocketsServer() {
    // disconnect all clients
      close();

    if (_mandatoryHttpHeaders)
        delete[] _mandatoryHttpHeaders;

    _mandatoryHttpHeaderCount = 0;
}

/**
 * called to initialize the Websocket server
 */
void WebSocketsServer::begin(void) {
    WSclient_t * client;

    // init client storage
    for(uint8_t i = 0; i < WEBSOCKETS_SERVER_CLIENT_MAX; i++) {
        client = &_clients[i];

        client->num = i;
        client->status = WSC_NOT_CONNECTED;
        client->tcp = NULL;
#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266) || (WEBSOCKETS_NETWORK_TYPE
== NETWORK_ESP32)
        client->isSSL = false;
```

```
        client->ssl = NULL;
#endif
        client->cUrl = "";
        client->cCode = 0;
        client->cKey = "";
        client->cProtocol = "";
        client->cVersion = 0;
        client->cIsUpgrade = false;
        client->cIsWebsocket = false;

        client->base64Authorization = "";

        client->cWsRXsize = 0;

#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266_ASYNC)
        client->cHttpLine = "";
#endif
    }

#ifdef ESP8266
    randomSeed(RANDOM_REG32);
#elif defined(ESP32)
    #define DR_REG_RNG_BASE 0x3ff75144
    randomSeed(READ_PERI_REG(DR_REG_RNG_BASE));
#else
    // TODO find better seed
    randomSeed(millis());
#endif

    _runnning = true;
    _server->begin();

    DEBUG_WEBSOCKETS("[WS-Server] Server Started.\n");
}

void WebSocketsServer::close(void) {
      _runnning = false;
       disconnect();

#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266)
    _server->close();
#elif (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP32) || (WEBSOCKETS_NETWORK_TYPE
== NETWORK_ESP8266_ASYNC)
    _server->end();
#else
    // TODO how to close server?
#endif

}

#if (WEBSOCKETS_NETWORK_TYPE != NETWORK_ESP8266_ASYNC)
/**
 * called in arduino loop
 */
void WebSocketsServer::loop(void) {
      if(_runnning) {
            handleNewClients();
            handleClientData();
      }
}
#endif

/**
```

```
 * set callback function
 * @param cbEvent WebSocketServerEvent
 */
void WebSocketsServer::onEvent(WebSocketServerEvent cbEvent) {
    _cbEvent = cbEvent;
}

/*
 * Sets the custom http header validator function
 * @param httpHeaderValidationFunc WebSocketServerHttpHeaderValFunc ///<
pointer to the custom http header validation function
 * @param mandatoryHttpHeaders[] const char* ///< the array of named http
headers considered to be mandatory / must be present in order for websocket
upgrade to succeed
 * @param mandatoryHttpHeaderCount size_t ///< the number of items in the
mandatoryHttpHeaders array
 */
void WebSocketsServer::onValidateHttpHeader(
      WebSocketServerHttpHeaderValFunc validationFunc,
      const char* mandatoryHttpHeaders[],
      size_t mandatoryHttpHeaderCount)
{
      _httpHeaderValidationFunc = validationFunc;

      if (_mandatoryHttpHeaders)
            delete[] _mandatoryHttpHeaders;

      _mandatoryHttpHeaderCount = mandatoryHttpHeaderCount;
      _mandatoryHttpHeaders = new String[_mandatoryHttpHeaderCount];

      for (size_t i = 0; i < _mandatoryHttpHeaderCount; i++) {
            _mandatoryHttpHeaders[i] = mandatoryHttpHeaders[i];
      }
}

/*
 * send text data to client
 * @param num uint8_t client id
 * @param payload uint8_t *
 * @param length size_t
 * @param headerToPayload bool  (see sendFrame for more details)
 * @return true if ok
 */
bool WebSocketsServer::sendTXT(uint8_t num, uint8_t * payload, size_t length,
bool headerToPayload) {
    if(num >= WEBSOCKETS_SERVER_CLIENT_MAX) {
        return false;
    }
    if(length == 0) {
        length = strlen((const char *) payload);
    }
    WSclient_t * client = &_clients[num];
    if(clientIsConnected(client)) {
        return sendFrame(client, WSop_text, payload, length, false, true,
headerToPayload);
    }
    return false;
}

bool WebSocketsServer::sendTXT(uint8_t num, const uint8_t * payload, size_t
length) {
    return sendTXT(num, (uint8_t *) payload, length);
}
```

```
bool WebSocketsServer::sendTXT(uint8_t num, char * payload, size_t length,
bool headerToPayload) {
    return sendTXT(num, (uint8_t *) payload, length, headerToPayload);
}

bool WebSocketsServer::sendTXT(uint8_t num, const char * payload, size_t
length) {
    return sendTXT(num, (uint8_t *) payload, length);
}

bool WebSocketsServer::sendTXT(uint8_t num, String & payload) {
    return sendTXT(num, (uint8_t *) payload.c_str(), payload.length());
}

/**
 * send text data to client all
 * @param payload uint8_t *
 * @param length size_t
 * @param headerToPayload bool  (see sendFrame for more details)
 * @return true if ok
 */
bool WebSocketsServer::broadcastTXT(uint8_t * payload, size_t length, bool
headerToPayload) {
    WSclient_t * client;
    bool ret = true;
    if(length == 0) {
        length = strlen((const char *) payload);
    }

    for(uint8_t i = 0; i < WEBSOCKETS_SERVER_CLIENT_MAX; i++) {
        client = &_clients[i];
        if(clientIsConnected(client)) {
            if(!sendFrame(client, WSop_text, payload, length, false, true,
headerToPayload)) {
                ret = false;
            }
        }
#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266)
        delay(0);
#endif
    }
    return ret;
}

bool WebSocketsServer::broadcastTXT(const uint8_t * payload, size_t length) {
    return broadcastTXT((uint8_t *) payload, length);
}

bool WebSocketsServer::broadcastTXT(char * payload, size_t length, bool
headerToPayload) {
    return broadcastTXT((uint8_t *) payload, length, headerToPayload);
}

bool WebSocketsServer::broadcastTXT(const char * payload, size_t length) {
    return broadcastTXT((uint8_t *) payload, length);
}

bool WebSocketsServer::broadcastTXT(String & payload) {
    return broadcastTXT((uint8_t *) payload.c_str(), payload.length());
}

/**
```

```
 * send binary data to client
 * @param num uint8_t client id
 * @param payload uint8_t *
 * @param length size_t
 * @param headerToPayload bool  (see sendFrame for more details)
 * @return true if ok
 */
bool WebSocketsServer::sendBIN(uint8_t num, uint8_t * payload, size_t length,
bool headerToPayload) {
    if(num >= WEBSOCKETS_SERVER_CLIENT_MAX) {
        return false;
    }
    WSclient_t * client = &_clients[num];
    if(clientIsConnected(client)) {
        return sendFrame(client, WSop_binary, payload, length, false, true,
headerToPayload);
    }
    return false;
}

bool WebSocketsServer::sendBIN(uint8_t num, const uint8_t * payload, size_t
length) {
    return sendBIN(num, (uint8_t *) payload, length);
}

/**
 * send binary data to client all
 * @param payload uint8_t *
 * @param length size_t
 * @param headerToPayload bool  (see sendFrame for more details)
 * @return true if ok
 */
bool WebSocketsServer::broadcastBIN(uint8_t * payload, size_t length, bool
headerToPayload) {
    WSclient_t * client;
    bool ret = true;
    for(uint8_t i = 0; i < WEBSOCKETS_SERVER_CLIENT_MAX; i++) {
        client = &_clients[i];
        if(clientIsConnected(client)) {
            if(!sendFrame(client, WSop_binary, payload, length, false, true,
headerToPayload)) {
                ret = false;
            }
        }
#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266)
        delay(0);
#endif
    }
    return ret;
}

bool WebSocketsServer::broadcastBIN(const uint8_t * payload, size_t length) {
    return broadcastBIN((uint8_t *) payload, length);
}


/**
 * sends a WS ping to Client
 * @param num uint8_t client id
 * @param payload uint8_t *
 * @param length size_t
 * @return true if ping is send out
 */
```

```
bool WebSocketsServer::sendPing(uint8_t num, uint8_t * payload, size_t
length) {
    if(num >= WEBSOCKETS_SERVER_CLIENT_MAX) {
        return false;
    }
    WSclient_t * client = &_clients[num];
    if(clientIsConnected(client)) {
        return sendFrame(client, WSop_ping, payload, length);
    }
    return false;
}

bool WebSocketsServer::sendPing(uint8_t num, String & payload) {
    return sendPing(num, (uint8_t *) payload.c_str(), payload.length());
}

/**
 *  sends a WS ping to all Client
 * @param payload uint8_t *
 * @param length size_t
 * @return true if ping is send out
 */
bool WebSocketsServer::broadcastPing(uint8_t * payload, size_t length) {
    WSclient_t * client;
    bool ret = true;
    for(uint8_t i = 0; i < WEBSOCKETS_SERVER_CLIENT_MAX; i++) {
        client = &_clients[i];
        if(clientIsConnected(client)) {
            if(!sendFrame(client, WSop_ping, payload, length)) {
                ret = false;
            }
        }
#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266)
        delay(0);
#endif
    }
    return ret;
}

bool WebSocketsServer::broadcastPing(String & payload) {
    return broadcastPing((uint8_t *) payload.c_str(), payload.length());
}


/**
 * disconnect all clients
 */
void WebSocketsServer::disconnect(void) {
    WSclient_t * client;
    for(uint8_t i = 0; i < WEBSOCKETS_SERVER_CLIENT_MAX; i++) {
        client = &_clients[i];
        if(clientIsConnected(client)) {
            WebSockets::clientDisconnect(client, 1000);
        }
    }
}

/**
 * disconnect one client
 * @param num uint8_t client id
 */
void WebSocketsServer::disconnect(uint8_t num) {
    if(num >= WEBSOCKETS_SERVER_CLIENT_MAX) {
```

```
        return;
    }
    WSclient_t * client = &_clients[num];
    if(clientIsConnected(client)) {
        WebSockets::clientDisconnect(client, 1000);
    }
}


/*
 * set the Authorization for the http request
 * @param user const char *
 * @param password const char *
 */
void WebSocketsServer::setAuthorization(const char * user, const char *
password) {
    if(user && password) {
        String auth = user;
        auth += ":";
        auth += password;
        _base64Authorization = base64_encode((uint8_t *)auth.c_str(),
auth.length());
    }
}

/**
 * set the Authorizatio for the http request
 * @param auth const char * base64
 */
void WebSocketsServer::setAuthorization(const char * auth) {
    if(auth) {
        _base64Authorization = auth;
    }
}

/**
 * count the connected clients (optional ping them)
 * @param ping bool ping the connected clients
 */
int WebSocketsServer::connectedClients(bool ping) {
    WSclient_t * client;
    int count = 0;
      for(uint8_t i = 0; i < WEBSOCKETS_SERVER_CLIENT_MAX; i++) {
            client = &_clients[i];
            if(client->status == WSC_CONNECTED) {
                  if(ping != true || sendPing(i)) {
                        count++;
                  }
            }
      }
    return count;
}

#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266) || (WEBSOCKETS_NETWORK_TYPE
== NETWORK_ESP8266_ASYNC) || (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP32)
/**
 * get an IP for a client
 * @param num uint8_t client id
 * @return IPAddress
 */
IPAddress WebSocketsServer::remoteIP(uint8_t num) {
    if(num < WEBSOCKETS_SERVER_CLIENT_MAX) {
        WSclient_t * client = &_clients[num];
```

```
        if(clientIsConnected(client)) {
            return client->tcp->remoteIP();
        }
    }

    return IPAddress();
}
#endif

//##############################################################################
######
//##############################################################################
######
//##############################################################################
######

/**
 * handle new client connection
 * @param client
 */
bool WebSocketsServer::newClient(WEBSOCKETS_NETWORK_CLASS * TCPclient) {
    WSclient_t * client;
    // search free list entry for client
    for(uint8_t i = 0; i < WEBSOCKETS_SERVER_CLIENT_MAX; i++) {
        client = &_clients[i];

        // state is not connected or tcp connection is lost
        if(!clientIsConnected(client)) {

            client->tcp = TCPclient;

#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266) || (WEBSOCKETS_NETWORK_TYPE
== NETWORK_ESP32)
            client->isSSL = false;
            client->tcp->setNoDelay(true);
#endif
#if (WEBSOCKETS_NETWORK_TYPE != NETWORK_ESP8266_ASYNC)
            // set Timeout for readBytesUntil and readStringUntil
            client->tcp->setTimeout(WEBSOCKETS_TCP_TIMEOUT);
#endif
            client->status = WSC_HEADER;
#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266) || (WEBSOCKETS_NETWORK_TYPE
== NETWORK_ESP8266_ASYNC) || (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP32)
            IPAddress ip = client->tcp->remoteIP();
            DEBUG_WEBSOCKETS("[WS-Server][%d] new client from %d.%d.%d.%d\n",
client->num, ip[0], ip[1], ip[2], ip[3]);
#else
            DEBUG_WEBSOCKETS("[WS-Server][%d] new client\n", client->num);
#endif


#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266_ASYNC)
            client->tcp->onDisconnect(std::bind([](WebSocketsServer * server,
AsyncTCPbuffer * obj, WSclient_t * client) -> bool {
                DEBUG_WEBSOCKETS("[WS-Server][%d] Disconnect client\n",
client->num);

                AsyncTCPbuffer ** sl = &server->_clients[client->num].tcp;
                if(*sl == obj) {
                    client->status = WSC_NOT_CONNECTED;
                    *sl = NULL;
                }
                return true;
```

```
            },  this, std::placeholders::_1, client));


            client->tcp->readStringUntil('\n', &(client->cHttpLine),
std::bind(&WebSocketsServer::handleHeader, this, client, &(client-
>cHttpLine)));
#endif

            return true;
            break;
        }
    }
    return false;
}

/**
 *
 * @param client WSclient_t *  ptr to the client struct
 * @param opcode WSopcode_t
 * @param payload  uint8_t *
 * @param length size_t
 */
void WebSocketsServer::messageReceived(WSclient_t * client, WSopcode_t
opcode, uint8_t * payload, size_t length, bool fin) {
    WStype_t type = WStype_ERROR;

    switch(opcode) {
        case WSop_text:
            type = fin ? WStype_TEXT : WStype_FRAGMENT_TEXT_START;
            break;
        case WSop_binary:
            type = fin ? WStype_BIN : WStype_FRAGMENT_BIN_START;
            break;
        case WSop_continuation:
            type = fin ? WStype_FRAGMENT_FIN : WStype_FRAGMENT;
            break;
        case WSop_close:
        case WSop_ping:
        case WSop_pong:
        default:
            break;
    }

    runCbEvent(client->num, type, payload, length);

}

/**
 * Disconnect an client
 * @param client WSclient_t *  ptr to the client struct
 */
void WebSocketsServer::clientDisconnect(WSclient_t * client) {


#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266) || (WEBSOCKETS_NETWORK_TYPE
== NETWORK_ESP32)
    if(client->isSSL && client->ssl) {
        if(client->ssl->connected()) {
            client->ssl->flush();
            client->ssl->stop();
        }
        delete client->ssl;
        client->ssl = NULL;
```

```
        client->tcp = NULL;
    }
#endif

    if(client->tcp) {
        if(client->tcp->connected()) {
#if (WEBSOCKETS_NETWORK_TYPE != NETWORK_ESP8266_ASYNC)
            client->tcp->flush();
#endif
            client->tcp->stop();
        }
#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266_ASYNC)
        client->status = WSC_NOT_CONNECTED;
#else
        delete client->tcp;
#endif
        client->tcp = NULL;
    }

    client->cUrl = "";
    client->cKey = "";
    client->cProtocol = "";
    client->cVersion = 0;
    client->cIsUpgrade = false;
    client->cIsWebsocket = false;

    client->cWsRXsize = 0;

#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266_ASYNC)
    client->cHttpLine = "";
#endif

    client->status = WSC_NOT_CONNECTED;

    DEBUG_WEBSOCKETS("[WS-Server][%d] client disconnected.\n", client->num);

    runCbEvent(client->num, WStype_DISCONNECTED, NULL, 0);

}

/**
 * get client state
 * @param client WSclient_t *  ptr to the client struct
 * @return true = connected
 */
bool WebSocketsServer::clientIsConnected(WSclient_t * client) {

    if(!client->tcp) {
        return false;
    }

    if(client->tcp->connected()) {
        if(client->status != WSC_NOT_CONNECTED) {
            return true;
        }
    } else {
        // client lost
        if(client->status != WSC_NOT_CONNECTED) {
            DEBUG_WEBSOCKETS("[WS-Server][%d] client connection lost.\n",
client->num);
            // do cleanup
            clientDisconnect(client);
        }
```

```
    }

    if(client->tcp) {
        // do cleanup
        DEBUG_WEBSOCKETS("[WS-Server][%d] client list cleanup.\n", client-
>num);
        clientDisconnect(client);
    }

    return false;
}
#if (WEBSOCKETS_NETWORK_TYPE != NETWORK_ESP8266_ASYNC)
/**
 * Handle incoming Connection Request
 */
void WebSocketsServer::handleNewClients(void) {

#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266) || (WEBSOCKETS_NETWORK_TYPE
== NETWORK_ESP32)
    while(_server->hasClient()) {
#endif
        bool ok = false;

#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266) || (WEBSOCKETS_NETWORK_TYPE
== NETWORK_ESP32)
        // store new connection
        WEBSOCKETS_NETWORK_CLASS * tcpClient = new
WEBSOCKETS_NETWORK_CLASS(_server->available());
#else
        WEBSOCKETS_NETWORK_CLASS * tcpClient = new
WEBSOCKETS_NETWORK_CLASS(_server->available());
#endif

        if(!tcpClient) {
            DEBUG_WEBSOCKETS("[WS-Client] creating Network class failed!");
            return;
        }

        ok = newClient(tcpClient);

        if(!ok) {
            // no free space to handle client
#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266) || (WEBSOCKETS_NETWORK_TYPE
== NETWORK_ESP32)
            IPAddress ip = tcpClient->remoteIP();
            DEBUG_WEBSOCKETS("[WS-Server] no free space new client from
%d.%d.%d.%d\n", ip[0], ip[1], ip[2], ip[3]);
#else
            DEBUG_WEBSOCKETS("[WS-Server] no free space new client\n");
#endif
            tcpClient->stop();
        }

#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266) || (WEBSOCKETS_NETWORK_TYPE
== NETWORK_ESP32)
        delay(0);
    }
#endif

}


/**
```

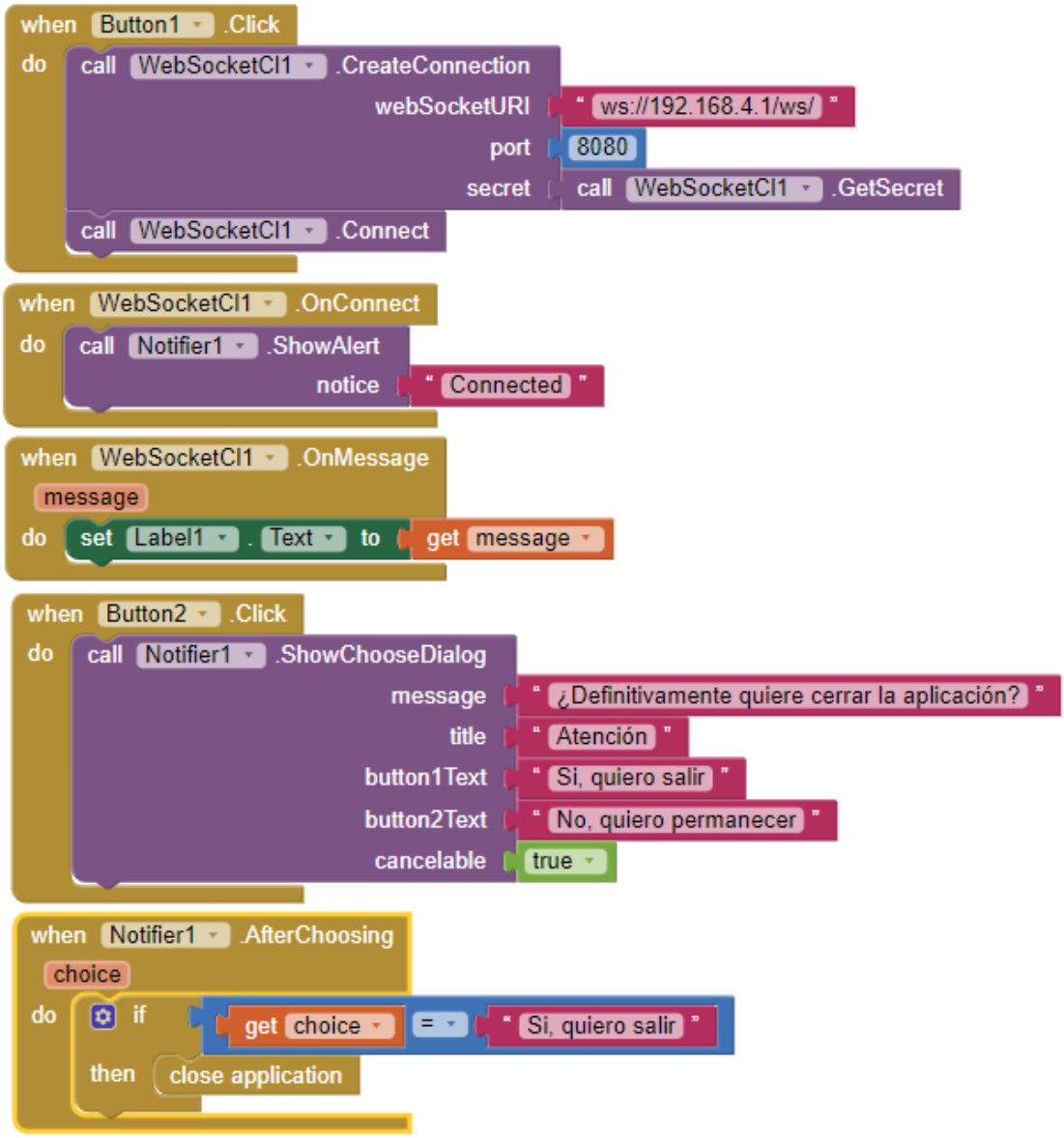
```
 * Handel incomming data from Client
 */
void WebSocketsServer::handleClientData(void) {

    WSclient_t * client;
    for(uint8_t i = 0; i < WEBSOCKETS_SERVER_CLIENT_MAX; i++) {
        client = &_clients[i];
        if(clientIsConnected(client)) {
            int len = client->tcp->available();
            if(len > 0) {
                //DEBUG_WEBSOCKETS("[WS-Server][%d][handleClientData] len:
%d\n", client->num, len);
                switch(client->status) {
                    case WSC_HEADER:
                    {
                        String headerLine = client->tcp-
>readStringUntil('\n');
                        handleHeader(client, &headerLine);
                    }
                        break;
                    case WSC_CONNECTED:
                        WebSockets::handleWebsocket(client);
                        break;
                    default:
                        WebSockets::clientDisconnect(client, 1002);
                        break;
                }
            }
        }
#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266)
        delay(0);
#endif
    }
}
#endif

/*
 * returns an indicator whether the given named header exists in the
configured _mandatoryHttpHeaders collection
 * @param headerName String ///< the name of the header being checked
 */
bool WebSocketsServer::hasMandatoryHeader(String headerName) {
      for (size_t i = 0; i < _mandatoryHttpHeaderCount; i++) {
            if (_mandatoryHttpHeaders[i].equalsIgnoreCase(headerName))
                  return true;
      }
      return false;
}


/**
 * handles http header reading for WebSocket upgrade
 * @param client WSclient_t * ///< pointer to the client struct
 * @param headerLine String ///< the header being read / processed
 */
void WebSocketsServer::handleHeader(WSclient_t * client, String * headerLine)
{

      static const char * NEW_LINE = "\r\n";

      headerLine->trim(); // remove \r

      if(headerLine->length() > 0) {
```

```
            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader] RX: %s\n", client-
>num, headerLine->c_str());

            // websocket requests always start with GET see rfc6455
            if(headerLine->startsWith("GET ")) {

                  // cut URL out
                  client->cUrl = headerLine->substring(4, headerLine-
>indexOf(' ', 4));

                  //reset non-websocket http header validation state for this
client
                  client->cHttpHeadersValid = true;
                  client->cMandatoryHeadersCount = 0;

            } else if(headerLine->indexOf(':')) {
                  String headerName = headerLine->substring(0, headerLine-
>indexOf(':'));
                  String headerValue = headerLine->substring(headerLine-
>indexOf(':') + 1);

                  // remove space in the beginning (RFC2616)
                  if(headerValue[0] == ' ') {
                        headerValue.remove(0, 1);
                  }


      if(headerName.equalsIgnoreCase(WEBSOCKETS_STRING("Connection"))) {
                        headerValue.toLowerCase();
                        if(headerValue.indexOf(WEBSOCKETS_STRING("upgrade"))
>= 0) {
                              client->cIsUpgrade = true;
                        }
                  } else
if(headerName.equalsIgnoreCase(WEBSOCKETS_STRING("Upgrade"))) {

      if(headerValue.equalsIgnoreCase(WEBSOCKETS_STRING("websocket"))) {
                              client->cIsWebsocket = true;
                        }
                  } else
if(headerName.equalsIgnoreCase(WEBSOCKETS_STRING("Sec-WebSocket-Version"))) {
                        client->cVersion = headerValue.toInt();
                  } else
if(headerName.equalsIgnoreCase(WEBSOCKETS_STRING("Sec-WebSocket-Key"))) {
                        client->cKey = headerValue;
                        client->cKey.trim(); // see rfc6455
                  } else
if(headerName.equalsIgnoreCase(WEBSOCKETS_STRING("Sec-WebSocket-Protocol")))
{
                        client->cProtocol = headerValue;
                  } else
if(headerName.equalsIgnoreCase(WEBSOCKETS_STRING("Sec-WebSocket-
Extensions"))) {
                        client->cExtensions = headerValue;
                  } else
if(headerName.equalsIgnoreCase(WEBSOCKETS_STRING("Authorization"))) {
                        client->base64Authorization = headerValue;
                  } else {
                        client->cHttpHeadersValid &=
execHttpHeaderValidation(headerName, headerValue);
                        if(_mandatoryHttpHeaderCount > 0 &&
hasMandatoryHeader(headerName)) {
                              client->cMandatoryHeadersCount++;
```

```
                        }
                    }

                } else {
                        DEBUG_WEBSOCKETS("[WS-Client][handleHeader] Header error
(%s)\n", headerLine->c_str());
                }

            (*headerLine) = "";
#if (WEBSOCKETS_NETWORK_TYPE == NETWORK_ESP8266_ASYNC)
            client->tcp->readStringUntil('\n', &(client->cHttpLine),
std::bind(&WebSocketsServer::handleHeader, this, client, &(client-
>cHttpLine)));
#endif
        } else {

            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader] Header read fin.\n",
client->num);
            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader]  - cURL: %s\n",
client->num, client->cUrl.c_str());
            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader]  - cIsUpgrade: %d\n",
client->num, client->cIsUpgrade);
            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader]  - cIsWebsocket:
%d\n", client->num, client->cIsWebsocket);
            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader]  - cKey: %s\n",
client->num, client->cKey.c_str());
            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader]  - cProtocol: %s\n",
client->num, client->cProtocol.c_str());
            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader]  - cExtensions:
%s\n", client->num, client->cExtensions.c_str());
            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader]  - cVersion: %d\n",
client->num, client->cVersion);
            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader]  -
base64Authorization: %s\n", client->num, client-
>base64Authorization.c_str());
            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader]  - cHttpHeadersValid:
%d\n", client->num, client->cHttpHeadersValid);
            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader]  -
cMandatoryHeadersCount: %d\n", client->num, client->cMandatoryHeadersCount);

            bool ok = (client->cIsUpgrade && client->cIsWebsocket);

                if(ok) {
                    if(client->cUrl.length() == 0) {
                        ok = false;
                    }
                    if(client->cKey.length() == 0) {
                    ok = false;
                }
                if(client->cVersion != 13) {
                    ok = false;
                }
                if(!client->cHttpHeadersValid) {
                    ok = false;
                }
                if (client->cMandatoryHeadersCount != _mandatoryHttpHeaderCount)
{
                    ok = false;
                }
            }

            if(_base64Authorization.length() > 0) {
                    String auth = WEBSOCKETS_STRING("Basic ");
```

```
                auth += _base64Authorization;
                if(auth != client->base64Authorization) {
                    DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader] HTTP
Authorization failed!\n", client->num);
                    handleAuthorizationFailed(client);
                    return;
                }
        }

        if(ok) {

            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader] Websocket
connection incoming.\n", client->num);

            // generate Sec-WebSocket-Accept key
            String sKey = acceptKey(client->cKey);

            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader]  - sKey: %s\n",
client->num, sKey.c_str());

            client->status = WSC_CONNECTED;

            String handshake = WEBSOCKETS_STRING("HTTP/1.1 101 Switching
Protocols\r\n"
                    "Server: arduino-WebSocketsServer\r\n"
                    "Upgrade: websocket\r\n"
                    "Connection: Upgrade\r\n"
                    "Sec-WebSocket-Version: 13\r\n"
                    "Sec-WebSocket-Accept: ");
                handshake += sKey + NEW_LINE;

            if(_origin.length() > 0) {
                handshake += WEBSOCKETS_STRING("Access-Control-Allow-Origin:
");
                handshake +=_origin + NEW_LINE;
            }

            if(client->cProtocol.length() > 0) {
                handshake += WEBSOCKETS_STRING("Sec-WebSocket-Protocol: ");
                handshake +=_protocol + NEW_LINE;
            }

            // header end
            handshake += NEW_LINE;

            DEBUG_WEBSOCKETS("[WS-Server][%d][handleHeader] handshake %s",
client->num, (uint8_t*)handshake.c_str());

            write(client, (uint8_t*)handshake.c_str(), handshake.length());

            headerDone(client);

            // send ping
            WebSockets::sendFrame(client, WSop_ping);

            runCbEvent(client->num, WStype_CONNECTED, (uint8_t *) client-
>cUrl.c_str(), client->cUrl.length());

        } else {
            handleNonWebsocketConnection(client);
        }
    }
}
```

ANEXO F – CÓDIGO APP



REFERENCIAS

1. *A brief history of the clinical thermometer.* **Pearce, J.M.S.** 4, s.l. : QJM: An International Journal of Medicine, April 2002, Vol. 95, págs. 251-252.

2. **Drinkwater, J.E.** Life of Galileo Galilei. 1832, pág. 41.

3. **Petrucelli, Albert S Lyons R Joseph.** *Medicine: An Illustrated History.* s.l. : Abradale/Abrams, 1997.

4. **The Galileo Project: Santorio Santorio. [En línea] [Citado el: 04 de 25 de 2019.] http://galileo.rice.edu/sci/santorio.html.**

5. ***Allbutt TC. Medical thermometry. Brit Med Chir Rev 1870; 45:429–41. Cited in: Norman JM, ed. Garrison & Morton's Medical Bibliography, 5th edn. Aldershot, Scholar Press, 1991:431.***

6. ***Wunderlich CRA. Das Verhalten der Eigenwarme in Krankheiten . Leipzig, O. Wigand 1868. Translated by New Sydenham Society, 1871.***

7. Enfasmart Baby Temperature Thermometer. [En línea] [Citado el: 17 de 3 de 2019.] https://www.enfasmart.com/.

8. Best Home Automation and Smart Health Home Devices Online . [En línea] Koogeek.com. [Citado el: 17 de 3 de 2019.] https://www.koogeek.com/.

9. Granda Miguel, Mercedes y Mediavilla Bolado, Elena. ***Instrumentación electrónica : transductores y acondicionadores de señal.*** **Santander : Editorial de la Universidad de Cantabria, 2015.**

10. GITT, Intrumentación Electrónica. Tema 2: Carácteristimas Metrológicas. 2018.

11. GITT, Instrumentación Electrónica. Tema 3: Errores en la medida. 2018.

12. Espressif Inc., "Overview ESP8266" . [En línea] [Citado el: 22 de 05 de 2019.] https://espressif.com/en/products/hardware/esp8266ex/overview.

13. Luis Llamas. [En línea] [Citado el: 19 de 05 de 2019.] https://www.luisllamas.es/.

14. LUA.org. [En línea] [Citado el: 22 de 05 de 2019.] https://www.lua.org/about.html.

15. MIT App Inventor. [En línea] Google Labs. [Citado el: 27 de 05 de 2019.] http://appinventor.mit.edu/explore/about-us.html.