UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Sistema de sensado mediante Arduino y una matriz de sensores de gases industriales

TRABAJO FIN DE GRADO

Ingeniería Electrónica Industrial y Automática

Universidad Politécnica de Valencia
Escuela Técnica Superior de Ingeniería del Diseño

Valencia, junio de 2019. Curso 2018/2019

Autor:
Alberto Díaz Paredes

Tutor:
Cristian Ariel Olguín Pinatti

Resumen

El presente Trabajado de Fin de Grado titulado *"Sistema de sensado mediante Arduino y una matriz de sensores de gases industriales"* tiene como objetivo el desarrollo de un sistema electrónico capaz de detectar sustancias volátiles en el aire. Este sistema está destinado a ser implementado en una nariz electrónica funcional, apta para realizar estudios e investigaciones.

Para la realización del proyecto, y una vez establecidos los requisitos que ha de cumplir el sistema de sensado, se ha comenzado por determinar y diseñar los principales subsistemas por los que está formado. Así, las partes que lo forman son una matriz compuesta por sensores de gases de la familia MQ y dos sensores de temperatura y humedad DHT11, el sistema de alimentación de esta matriz, un microcontrolador Arduino que procesa y envía los datos obtenidos de los sensores a un ordenador, y un software diseñado con Processing que permite la visualización a tiempo real de esa información, además de dar la posibilidad al usuario de exportar los datos en un fichero de texto con formato *.csv.*

Se ha llevado a cabo una simulación del sistema utilizando el software Proteus, con lo que se ha podido comprobar que ambos programas, el de Arduino y el de Processing, funcionan correctamente.

A continuación, se ha implementado un prototipo de la matriz de sensores y se han llevado a cabo dos ensayos con los que se ha podido corroborar que el sistema en su totalidad es viable y proporciona los resultados esperados.

Por último, se han diseñado las placas de circuito impreso en las que se montan tanto la matriz de sensores como la alimentación del sistema y que se usarán en la futura implementación de la nariz electrónica.

Abstract

The present Final Degree project titled "*Sistema de sensado mediante Arduino y una matriz de sensores de gases industriales*" – "*Sensing system using Arduino and a matrix of industrial gas sensors"* – aims to develop an electronic system capable of detecting volatile substances in the air. This system is intended to be implemented in a functional electronic nose, suitable for studies and research.

For the realization of the project, and once established the requirements that the sensing system must fulfill, its main subsystems by which it is formed have been determined and designed. Its main parts are a matrix composed of gas sensors of the MQ family and two temperature and humidity DHT11 sensors, its power supply system, an Arduino microcontroller that processes and sends the data obtained from the matrix to a computer, and a software designed with Processing that allows a real time visualization of the sensorized data and gives the user the possibility to export this information to a .csv text file.

A simulation of the system in Proteus was carried out, in order to verify that both programs, that of Arduino and that of Processing, work correctly.

Next, a prototype of the sensor array has been implemented and two experiments have been carried out. By doing that, it has been possible to corroborate that the system is viable and provides the expected results.

Finally, the printed circuit boards have been designed in which both the sensor array and the system power supply are mounted. This circuit boards will be used in the future implementation of the electronic nose.

Índice general

Capítulo 1. Memoria

Índice Capítulo 1. Memoria

Índice de Figuras

1. Objeto

El objeto de este proyecto es el diseño de un sistema de sensado utilizando el microcontrolador Arduino y una matriz de sensores de gases industriales. La intención final es que el conjunto desarrollado pueda ser empleado para la posterior creación de una nariz electrónica de laboratorio completamente funcional.

Por un lado, se desarrolla el sistema de adquisición de datos compuesto por la matriz de sensores y el microcontrolador. Este último se encarga de leer los valores de los sensores y transmitirlos a un ordenador donde puedan ser tratados.

Por otro lado, se crea un software encargado de leer y procesar los datos recibidos en el ordenador desde Arduino. Este programa es capaz de mostrar a tiempo real los valores de los diferentes sensores y de exportarlos a un archivo de texto con formato *.csv*. Con este archivo de texto se pueden tratar los datos obtenidos con otros programas como Matlab o Excel.

Además, durante el proyecto se realizan tanto una simulación del sistema como el montaje de un prototipo, y se realizan sendos ensayos para comprobar que el comportamiento tanto de los componentes individuales como del sistema completo es el esperado.

2. Justificación del proyecto

El presente proyecto se trata de un Trabajo de Fin de Grado realizado con vistas a la obtención del título del Grado en Ingeniería Electrónica Industrial y Automática por la Universidad Politécnica de Valencia.

Surge como una propuesta del profesor de la universidad y tutor del proyecto, quien encarga al alumno el diseño del sistema de sensado de una nariz electrónica, bajo las condiciones de que sea sencillo de implementar y de bajo coste.

3. Conceptos teóricos y estado del arte

3.1. Nariz electrónica

Una de las definiciones más populares de nariz electrónica fue enunciada por Gardner y Barlett en 1999: "*Instrumento que comprende una agrupación de sensores químicos con sensibilidades parcialmente solapadas junto a un sistema de reconocimiento de patrones, capaz de analizar y reconocer aromas simples o complejos*".

Una nariz electrónica se trata, por tanto, de un sistema que es capaz de detectar una sustancia volátil presente en el aire gracias a una matriz de sensores, e identificarla con ayuda de un algoritmo que analiza la respuesta de dicha matriz.

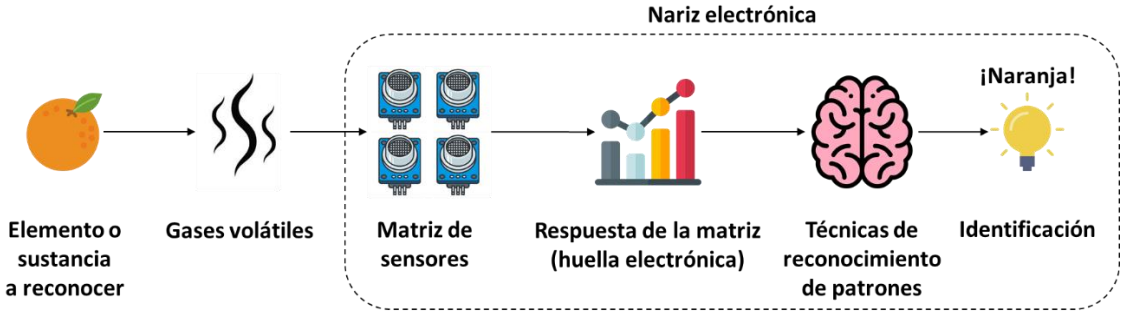


Figura 1 - Proceso de identificación de una sustancia

El primer dispositivo precursor de la nariz electrónica surgió en los años 60 de manos de la compañía *Bacharac Inc*., que fabricó un dispositivo llamado *Sniffer*. Sin embargo, tan sólo contaba con un único sensor de gas, por lo que no es considerado una nariz electrónica como tal.

Las primeras publicaciones relacionadas con las narices electrónicas surgieron a comienzos de la década de 1980 a manos de dos grupos de investigadores. El primero enfocó sus esfuerzos en entender los procesos del olfato biológico. El segundo, por su parte, desarrolló un aparato para detectar, identificar y medir una amplia variedad de productos químicos. Simultáneamente, en Japón se comenzó a utilizar matrices de sensores para investigar la frescura de los pescados. Todos estos avances dieron lugar a la posterior fabricación de narices electrónicas a nivel comercial.

En un primer momento fueron desarrolladas para sustituir a los expertos clasificadores de aromas, debido a que se consideraban costosos, difíciles de transportar y, en ocasiones, subjetivos, debido a que sus valoraciones podían verse afectadas por factores como el estado anímico o el cansancio.

En la actualidad las narices electrónicas siguen siendo usadas como clasificadores de aroma, pero también para la detección de otras sustancias. Es por ello por lo que su utilización se ha extendido a diferentes campos tales como la agroindustria, la contaminación ambiental, la seguridad o la medicina.

Existen múltiples tipos de narices electrónicas según el uso al que están destinadas. Algunas son pequeñas y portables; otras, por el contrario, son grandes y difíciles de transportar. Este último es el caso de las narices electrónicas de laboratorio, destinadas a realizar estudios científicos y ensayos.

Este tipo de narices disponen de dos cámaras: la cámara de la muestra, en la que se introduce la sustancia o gas que se desea identificar, y la cámara de medida, que contiene la matriz de sensores. Mediante un sistema de canalización del aire, se consigue un flujo que circula entre las dos cámaras de forma uniforme, haciendo que los gases volátiles de la cámara de la muestra pasen a la cámara de medida e impregnen los sensores de forma homogénea. De esta forma, se logra que la matriz reconozca el gas de una forma más fiable que midiendo directamente la muestra.

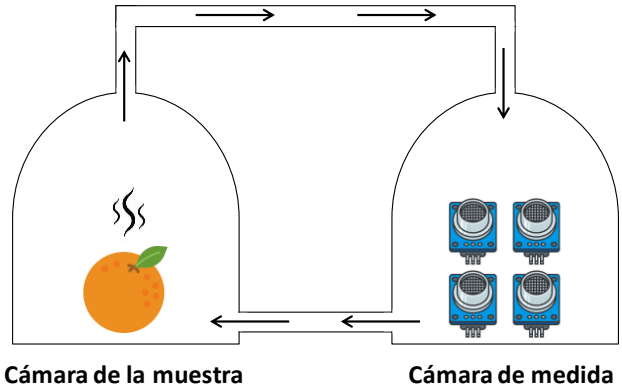


Figura 2 - Nariz electrónica de laboratorio

3.2. Java y Processing

En sus inicios, el lenguaje de programación *Java* era conocido como *Oak* o *Green* y fue desarrollado por James Gosling en el año 1991. La primera versión fue publicada en 1995, y fue en 1996 cuando pasó a llamarse *Java*, con el lanzamiento de la versión *JDK 1.0.2*.

El objetivo de James Gosling era crear un lenguaje de programación orientado a objetos, con una estructura y sintaxis parecidas a *C++*, pero con una máquina virtual propia. De esta forma, un programa escrito en Java podría ser usado en cualquier arquitectura bajo el principio *"Write once, run anywhere"*.

Fue con sus primeras versiones – *Java 1.1, 1.2 y 1.3* – cuando el lenguaje fue tomando la forma que tiene hoy en día, gracias a la inclusión de tecnologías como *JavaBeans*, *JDBC* para bases de datos, *RMI* para las invocaciones en remoto, *Collections* para la gestión de múltiples estructuras de datos o *AWT* para el desarrollo gráfico, entre otros.

En el año 2002, se lanzó la versión 1.4 de *Java*, que pasó a ser la primera gestionada por la comunidad mediante el *Java Community Process* (*JCP*). Posteriormente, en 2006 *Sun Microsystems* convirtió *Java* en un código libre mediante una licencia *GNU General Public License* (*GPL*).

A día de hoy, *Java* es uno de los lenguajes más importantes del mundo gracias a una comunidad extendida en todos los componentes y más de cuatro millones de desarrolladores. Además, con el auge de los smartphones y *Android*, se ha establecido como el lenguaje de programación para móviles más extendido.

Desde el lanzamiento de *Java*, otros lenguajes de programación han surgido basándose en este último. Uno de ellos es *Processing*, que vio la luz en 2001 de manos de Ben Fry y Casey Reas, heredando sus virtudes y convirtiéndose en una herramienta muy útil a la hora de desarrollar diferentes tipos de proyectos, tanto aplicaciones locales como para la web. Está destinado a desarrollar entornos gráficos y se caracteriza por su facilidad de uso y por la gran comunidad de usuarios que comparten código y librerías a través de foros especializados.

Processing es, además de un lenguaje de programación, un entorno de desarrollo de código abierto, que destaca por su simplicidad y en el que posteriormente se basó *Arduino* para desarrollar su propio IDE.

3.3. Microcontroladores y Arduino

Un microcontrolador es un circuito integrado programable, capaz de ejecutar las órdenes para las que ha sido programado y que están almacenadas en su memoria. Está compuesto por distintos bloques funcionales, desarrollando cada uno una tarea específica. Entre estos bloques se encuentran la unidad central de proceso o CPU, la memoria principal, los módulos de entrada y salida, y los buses de comunicación.

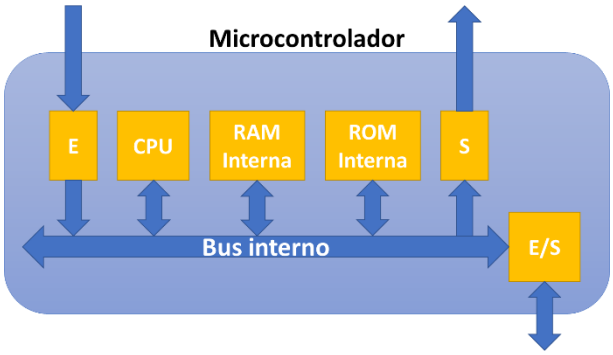


Figura 3 - Estructura de un microcontrolador

El primer microcontrolador fue desarrollado por *Texas Instruments* en la década de 1970. Se trataba de un microcontrolador de 4 bits con una función de ROM y RAM, y que la compañía utilizó internamente en sus productos mientras era perfeccionado. No fue hasta 1974 cuando *Texas Instruments* comercializó el *TMS 1000*, que se convirtió en el primer microcontrolador a la venta para la industria electrónica. Destinado a sistemas embebidos, combinaba memoria ROM, RAM, microprocesador y reloj en un único chip.

En 1977 se lanzó al mercado el *Intel 8048*. Fue uno de los primeros microcontroladores de *Intel* y fue utilizado en el teclado de la computadora personal de *IBM*. Le siguió el *8051* en 1980, que se convirtió en una de las familias de microcontroladores más populares. A día de hoy se siguen vendiendo variaciones de éste.

En aquella época, la mayoría de los microcontroladores contaban con memorias EPROM o PROM. Las primeras eran reprogramables, pero para realizar el borrado era necesario exponer la memoria a luz ultravioleta. Esto las convertía en una opción más cara que las memorias PROM, que presentaban la desventaja de que sólo eran programables una única vez.

En 1993 el microcontrolador *PIC16X84* de Microchip introdujo por primera vez las memorias EEPROM, que permitían un borrado eléctrico y más rápido que el de las memorias EPROM. Ese mismo año *Atmel* lanzó el primer microcontrolador que usaba memorias de tipo flash. Éstas permitían la creación rápida de prototipos y la programación en el sistema. A partir de este momento los costes de los microcontroladores se desplomaron y sus ventas aumentaron exponencialmente cada año.

A día de hoy los microcontroladores son baratos y fácilmente accesibles para el público general, lo que ha ayudado a que surjan grandes comunidades de usuarios que comparten contenido para ciertos microcontroladores a través de internet, como es el caso de *Arduino*.

Arduino surgió en el año 2005 como un proyecto de tesis del alumno del instituto *IVREA* Hernando Barragán. Su principal objetivo era crear una herramienta fácil de usar y de menor costo que el que suponían los microcontroladores que usaban los estudiantes en aquella época.

El proyecto consistía en una placa de desarrollo hardware basada en un microcontrolador *ATmega168*, un entorno de desarrollo integrado (IDE) basado en *Processing* y una biblioteca de funciones que facilitaban su programación.

Actualmente, *Arduino* se ha convertido en una compañía de hardware y software libre que diseña y manufactura placas de desarrollo hardware enfocadas en acercar y facilitar el uso de la electrónica y programación de sistemas embebidos a los estudiantes y aficionados.

Las diferentes propiedades que caracterizan a las placas de desarrollo Arduino son las siguientes:

- **Bajo coste.** Además de tener la posibilidad de comprar la placa de desarrollo por separado, se pueden encontrar en el mercado kits de desarrollo con los componentes necesarios para que los aficionados y estudiantes puedan iniciarse en él. Todo ello a un bajo precio.
- **Fácil de utilizar.** Está destinado a poder ser utilizado por personas con pocos conocimientos en programación y electrónica. Incluso por niños gracias a la posibilidad de ser programados con entornos gráficos que utilizan lenguajes visuales basados en bloques.
- **Plataforma abierta.** Permite construir gran cantidad de proyectos al poder combinarse con otras plataformas.
- **Alta flexibilidad.** Al ser de código libre, existen muchas herramientas que facilitan su uso y se puede trabajar en todas las plataformas informáticas.
- **Gran variedad de placas.** Existen multitud de placas, cada una enfocada a un uso concreto.
- **Placas de expansión.** Gracias a las placas de expansión existentes – llamadas *Shield* – se pueden extender las funcionalidades de *Arduino*, como por ejemplo WiFi, Ethernet o GPS.
- **Gran comunidad de usuarios.** Hay multitud de foros y páginas web donde los usuarios comparten sus proyectos y se ayudan entre sí para resolver sus dudas y problemas.

Las principales placas de desarrollo Arduino utilizadas hoy en día son las siguientes:

- **Arduino UNO.** Es la placa estándar. Más utilizada, conocida y documentada. Cuenta con un USB HID propio y un procesador *Atmega328p* con 32Kbytes de ROM para alojar el programa.
- **Arduino Mega 2560.** Es la placa más potente con microprocesador de 8 bits pensada para proyectos complejos. Su microcontrolador *Atmega2560* tiene más memoria, más RAM y más pines I/O. Existe la variante *Mega ADK*, basada en el *Mega 2560* pero con un USB host adicional para conectar dispositivos móviles basados en *Android*.
- **Arduino Micro.** Está basado en el *ATmega32u4* y se caracteriza por ser muy compacto.
- **Arduino Lilypad.** Diseñado para dispositivos wearables y textiles, permite coser un hilo conductor e instalarlo sobre prendas.
- **Arduino 101.** Es una de las placas más recientes y el sucesor de *Arduino UNO*. Cuenta con un procesador *Intel Curie Quark* de 32 bits de bajo consumo, 384 KB de memoria flash, 80 KB de SRAM y sensor DSP, Bluetooth, acelerómetro y giroscopio integrados.

En la siguiente tabla se puede observar una comparativa de sus características:

Arduino	Microcontrolador	Bits	Pines E/S digitales (PWM)	Pines analógicos	Memoria flash	SRAM	EEPROM	Velocidad de reloj
UNO	ATmega328	8	14 (6)	6	32 KB	2 KB	1 KB	16 MHz
Mega 2560	ATmega2560	8	54 (15)	16	256 KB	8 KB	4 KB	16 MHz
Micro	ATmega32u4	8	20 (7)	12	32 KB	2,5 KB	1 KB	16 MHz
Lilypad	ATmega168V	8	14 (6)	6	16 KB	1 KB	512 Bytes	8 MHz
101	Intel Curie	32	14 (4)	6	196 KB	24 KB	-	32 MHz

Figura 4 - Tabla comparativa de los modelos de Arduino

4. Factores a considerar

A la hora de desarrollar el proyecto se han tenido en cuenta una serie de condiciones preestablecidas que determinan algunas de las principales características que ha de cumplir el sistema de sensado. Estos factores son las condiciones del encargo, que recogen aquellos puntos que ha de cumplir el sistema, así como la normativa que ha de satisfacer para asegurar el correcto diseño y posterior funcionamiento del mismo.

4.1. Condiciones del encargo

Previo al comienzo del desarrollo del proyecto, se han establecido las siguientes condiciones del sistema:

- El sistema de sensado tiene como objetivo final el de **ser implementado en una nariz electrónica** funcional para su uso en laboratorio. Esta condición debe ser tenida en cuenta a lo largo de todo el proyecto.
- El sistema diseñado debe ser **sencillo** y lo más **barato** posible.
- La matriz de sensores ha de estar formada por la **mayor cantidad posible** de éstos. Además, deben ser **redundantes**. De esta forma se consigue que la nariz electrónica sea capaz de detectar los gases volátiles con mayor fiabilidad.
- En caso de que los sensores de gases empleados así lo requieran, se han de implementar también **sensores de temperatura y humedad**. Se consigue así poder compensar las variaciones de sensibilidad que estos factores producen sobre los sensores, por lo que la huella electrónica de la sustancia sensorizada no se verá afectada.
- El software desarrollado ha de permitir la **visualización a tiempo real** de las lecturas realizadas por los sensores.
- El usuario debe tener la posibilidad de **exportar los datos a un archivo de texto**. El objetivo es que esta información pueda ser utilizado posteriormente por otro software más potente que, mediante el algoritmo adecuado, sea capaz de reconocer los patrones de la huella electrónica.
- **La interfaz de usuario del programa ha de ser sencilla e intuitiva**, permitiendo que un usuario sin conocimientos previos pueda usarlo sin necesidad de formación previa.
- La cámara de medición de la nariz electrónica ha de ser lo más pequeña posible. De esta forma los gases volátiles se distribuyen de una forma más uniforme en su interior. Se asegura así que llegue a los sensores una concentración similar de gases. **La PCB diseñada debe ser lo más compacta posible** para ayudar a cumplir esta condición.
- La placa de circuito impreso donde se alojen los sensores estará expuesta a las sustancias volátiles del interior de la cámara de medida. Esta condición se ha de tener en cuenta en su diseño, de forma que se **facilite la limpieza de las sustancias** que precipitan sobre ésta.

4.2. Normativa

El proyecto ha de cumplir las siguientes normas UNE desarrolladas por la *Asociación Española de Normalización y Certificación* (AENOR):

- **UNE – EN 60617 – 4 y 5**. Recoge los símbolos gráficos normalizados de componentes electrónicos activos y pasivos a utilizar en dibujos, esquemas y planos de ingeniería, así como en la documentación técnica de productos relacionados.
- **UNE 20531.** Define valores preferidos para resistencias y condensadores. Se trata de valores nominales establecidos por década y clasificados en series según las tolerancias.

A la hora de diseñar las placas de circuito de impreso también se han de tener las siguientes normas IPC:

- **IPC 2221**. Define los requisitos generales para el diseño de PCBs y algunos aspectos sobre el ensamble de tarjetas electrónicas. Incluye principios y recomendaciones sobre el montaje y ensamble de componentes, ya sea a través de huecos pasantes o de montaje en superficie (SMT).
- **IPC – D – 325**. Recoge los requisitos para documentar correctamente el diseño de un circuito impreso.

5. Justificación de la solución adoptada

A continuación, se presenta la solución que se ha adoptado respecto al sistema de medición para que cumpla aquellos criterios y condiciones que se han establecido en el *Apartado 4.1.* Se detallan también las distintas alternativas que se han propuesto durante su desarrollo, así como los criterios utilizados para seleccionar la más adecuada.

5.1. Estructura general del sistema propuesto

El primer paso a llevar a cabo para realizar el proyecto es determinar qué partes o subsistemas son los que, en su conjunto, conformarán el sistema de medición al completo. Así, el sistema de sensado desarrollado durante este proyecto se divide en cuatro partes claramente diferenciadas.

1. **La matriz de sensores**. Formada por sensores de gases y de temperatura y humedad montados sobre una placa de circuito impreso. Proporciona la huella electrónica de las sustancias volátiles presentes en el aire sensado que se emplea para, posteriormente, identificar dichas sustancias.
2. **Un microcontrolador**. Se encarga de leer los sensores de la matriz, procesarlos y enviarlos a un ordenador. Además, controla que las medidas se realicen conforme al periodo de muestreo establecido.
3. **Un ordenador junto a un software** que lee los datos recibidos del microcontrolador y permite tanto su visualización a tiempo real como su guardado para un posterior uso, tal como viene definido en las condiciones del encargo.
4. **Una fuente de alimentación** que provee a la matriz de sensores la potencia necesaria para su funcionamiento.

En el siguiente esquema se puede observar la estructura general del sistema, así como la interacción entre sus diferentes partes:

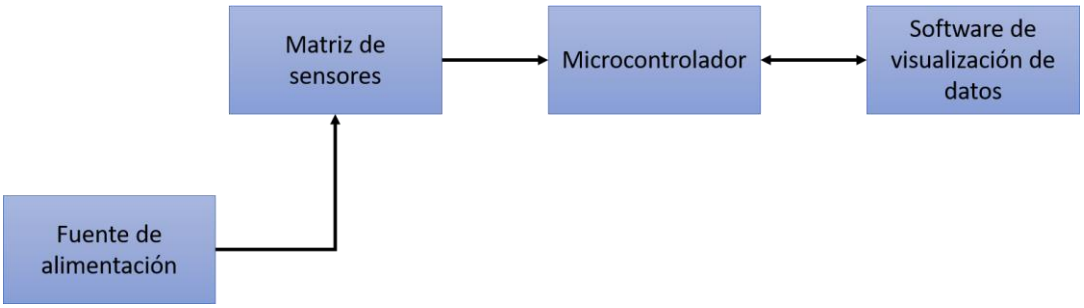


Figura 5 - Estructura general del sistema de sensado

5.2. Matriz de sensores

Una matriz de sensores se puede caracterizar por el tipo y tecnología de sensores que utiliza, por la cantidad de éstos o por cómo se encuentra implementada. A continuación, se justifica la solución adoptada en cada uno de estos factores que, en su conjunto, conforman la matriz de sensores.

5.2.1. Cantidad de sensores

La matriz está formada por la mayor cantidad posible de sensores que puede soportar el sistema para que, de esta forma, sea capaz de detectar una mayor cantidad de gases distintos. Además, los sensores aparecen redundados en la matriz – con al menos dos de cada tipo – para que el sistema sea lo más fiable y preciso posible.

La limitación en cuanto a la cantidad de sensores a emplear proviene de dos factores: la potencia e intensidad máximas que es capaz de proporcionar la fuente de alimentación, y el número de entradas digitales y/o analógicas de las que dispone el microcontrolador empleado.

En total, se emplean 16 sensores de gases y 2 de temperatura y humedad.

5.2.2. Sensores de gases

Existen múltiples tipos de sensores de gases que funcionan de forma distinta y depende de la tecnología que emplean. Según su método de operación, se pueden distinguir dos grupos fundamentales: el primero está formado por sensores que funcionan por medio de absorción, reacciones químicas y de contacto con el gas; el segundo, por sensores basados en emisiones infrarrojas o ultrasónicas.

A continuación, se presentan algunos de los tipos de sensores de gases más utilizados de acuerdo con su tipo de operación:

Sensores infrarrojos

Su funcionamiento se basa en emisores y receptores de luz infrarroja. El gas presente en el ambiente interfiere con la potencia de transmisión entre el emisor y el receptor. Esta alteración determina la presencia del gas.

Se trata de sensores analógicos que presentan una alta resolución, alta sensibilidad y estabilidad, y bajo consumo de energía. Sin embargo, sólo pueden detectar gases que contienen más de un tipo de átomo, como el dióxido de carbono (CO_2) o el metano (CH_4). No pueden, por ejemplo, detectar oxígeno (O_2) o hidrógeno (H_2).

Se descarta su uso en este proyecto por tratarse de sensores de alto rendimiento que ofrecen características superiores a las demandadas por el proyecto, lo que además encarece su coste económico.

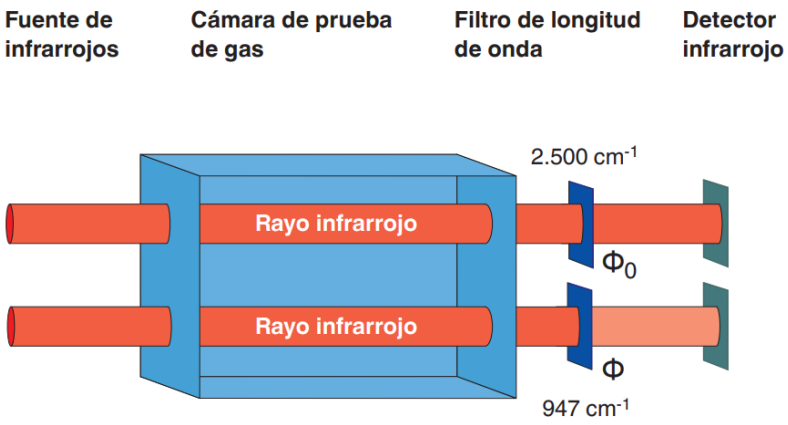


Figura 6 - Funcionamiento de un sensor de gases infrarrojo

Sensores ultrasónicos

Estos sensores usan emisiones de ultrasonidos con las que son capaces de detectar cambios en el ruido de fondo del ambiente. Son especialmente caros y se usan principalmente para detectar fugas en tuberías, por lo que no son aptos para el diseño del sistema de sensado propuesto.

Sensores electroquímicos

Cuentan con dos electrodos separados por una capa de electrolitos. Cuando el gas entra al sensor mientras una tensión de polarización se aplica a los electrodos, se produce una reacción de reducción – oxidación que genera una corriente eléctrica proporcional a la concentración de gas.

Su respuesta es rápida, precisa y estable. Además, son robustos y duraderos. Vienen calibrados de fábrica y pueden detectar concentraciones de gases especialmente bajas. Sin embargo, quedan descartados por el mismo motivo que los sensores infrarrojos: son productos caros que ofrecen más prestaciones de las necesarias.

Sensores semiconductores

Sensores con semiconductores de óxido metal. Su método de funcionamiento se basa en una película sensible al gas que está compuesta principalmente por cristales de óxido-metal. Cuando el gas entra en contacto con el sensor, provoca un cambio en la resistencia eléctrica de éste. Disponen también de un calentador – que no es más que una resistencia eléctrica - encargado de volatilizar los gases que entran en contacto con la película del sensor. Se consigue así un equilibrio entre los gases que precipitan sobre el sensor y los que se volatilizan, evitando su saturación con el paso del tiempo.

Se caracterizan por ser muy eficientes y por poder operar en un alto rango de humedad ambiental. Son baratos y existe una amplia gama de sensores capaces de detectar una multitud de gases distintos. Sin embargo, son sensibles a la temperatura y humedad, de forma que las mediciones realizadas se pueden ver afectadas por variaciones de estos dos factores ambientales.

Dadas sus propiedades, se escogen sensores basados en esta tecnología para crear la matriz de sensado. Concretamente, se seleccionan los de la familia MQ.

Éstos son sensores de gases cuyo principio de funcionamiento se basa en un sensor que varía su resistencia al entrar en contacto con las sustancias. Presentan una dinámica lenta, que necesita largos periodos de tiempo para estabilizarse tras un cambio en la concentración de los gases medidos.

Disponen además de un calentador que eleva su temperatura. Gracias a ello, se consigue que los gases que entran en contacto y precipitan sobre el sensor vuelvan a evaporarse, evitando así su saturación cuando son sometidos a un gas. Por tanto, para que estos sensores funcionen correctamente, es necesario depurarlos antes de su uso. Esto se consigue calentándolos durante un periodo de tiempo, que según el sensor puede llegar hasta 12 o 48 horas.

Cada modelo necesita su propia tensión para alimentar el calentador. En la mayoría de modelos es de 5 V, pero hay otros que requieren una alimentación especial. Este el caso, por ejemplo, del sensor MQ-7 que requiere una alimentación que alterna periódicamente entre 5 y 1,5 V. Además, debido al calor necesario para el funcionamiento del calentador, el consumo de estos sensores puede llegar hasta 900 mW en algunos modelos.

Por otro lado, cada modelo de sensor no es capaz de detectar un único gas, sino que es sensible a una gran cantidad de ellos. No obstante, la sensibilidad que presenta frente a cada gas es distinta y se ve afectada por los cambios en la temperatura y humedad ambientales.

Estos sensores ofrecen una salida analógica de tensión directamente proporcional a la concentración de gases presentes. Para ello, es necesario emplear una resistencia externa que, junto con el sensor, conforman un divisor resistivo.

Los motivos por los que se ha seleccionado la serie de sensores MQ son los siguientes:

- **Son baratos**. Se pueden encontrar en el mercado por un precio no superior a dos euros cada sensor, frente a los veinte euros que cuestan los de la serie TGS del fabricante Figaro, por ejemplo.
- **Son fáciles de utilizar**. La mayoría de los sensores funcionan con una tensión de alimentación continua de 5 V; aunque hay otros que deben ser alimentados a 6 V o alternando entre dos valores de tensión – generalmente 5 y 1,4 V –, pero que son descartados para esta aplicación dada su compleja integración con el resto de sensores de la matriz. Adem
- **Son sensibles a más de un gas**. Cada sensor de la serie MQ no detecta un único gas, sino que es capaz de detectar un conjunto de éstos, presentando una sensibilidad distinta para cada uno.

Existe una gran variedad. La familia MQ se compone de una gran cantidad de sensores que, entre todos, cubren la posibilidad de detectar una multitud de gases distintos.

Figura 7 - Sensor MQ

5.2.3. Sensores de temperatura y humedad

Como se ha mencionado en el *Apartado 5.2.2*, los sensores de gases escogidos son sensibles a variaciones en la temperatura y humedad del aire sensado. Es necesario medir estas magnitudes al mismo tiempo que se obtienen los datos de los sensores de gases, para que a la hora de analizar la huella electrónica generada por la matriz se puedan compensar estas variaciones en sus sensibilidades. Si no se hiciera de esta forma, al medir una misma sustancia en dos momentos distintos en los que estas magnitudes hayan variado, se obtendrían resultados distintos que llevarían a una identificación errónea de la sustancia.

Existen dos alternativas para llevar a cabo la medición de la temperatura y la humedad. Por un lado, utilizar dos sensores: uno para medir cada magnitud física. Por el otro, emplear un único sensor que sea capaz de medir ambas magnitudes simultáneamente.

Dos sensores para medir temperatura y humedad de forma separada

Esta opción presenta como principal ventaja la gran cantidad de sensores que se encuentran disponibles en el mercado actualmente. Algunos de ellos tienen características que se ajustan a la perfección a las necesidades del proyecto. Este es el caso de los sensores TMP36 y LM35, que ofrecen un alto rango de medida, alimentación a 5 V, resolución suficiente, bajo coste y simplicidad de montaje y utilización.

Sin embargo, no ocurre con lo mismo en el caso de los sensores de humedad relativa. Las alternativas son escasas y poco adecuadas para su aplicación en el sistema de medida diseñado dada su complejidad de uso y su alto coste. Este es el caso, por ejemplo, del sensor HS1101.

Un único sensor para medir temperatura y humedad

Utilizar un único sensor que mida simultáneamente la temperatura y la humedad relativa ambiental presenta como principal ventaja la sencillez en el diseño y la reducción del tamaño de la placa de circuito impreso gracias al uso de un menor número de dispositivos. Además, los sensores de este tipo proporcionan la información a través de una salida digital. Esto implica que no es necesario reservar el uso de uno de los pines analógicos de Arduino para conectar este sensor y, en consecuencia, se pueden usar más sensores de gases para la matriz.

Por estos motivos, en la solución adoptada se utiliza un único sensor para medir la temperatura y humedad ambientales. En concreto, se selecciona el sensor DHT11, que permite realizar la medición simultánea de estas dos magnitudes.

Además, este tipo de sensor dispone de un procesador interno que realiza el proceso de medición, y proporciona los resultados obtenidos a través de una señal digital. Por tanto, es muy sencillo obtener la medición a través de un microprocesador que interprete esa señal.

Sus principales características son:

- Medición de temperatura: entre 0 y 50º C, con una precisión de 2º C.
- Medición de humedad: entre 20 y 90 %, con una precisión del 5%.
- Frecuencia de muestreo: 1 muestra por segundo (1 Hz).

5.2.4. Implementación en PCB

Para que el sistema de sensado pueda ser implementado posteriormente en una nariz electrónica funcional, se plantea la creación de uno o varios circuitos impresos donde se ubiquen los sensores. Los principales requisitos que debe cumplir se han detallado en el *Apartado 4.1* son los siguientes:

- Debe poder ser implementada en una nariz electrónica funcional.
- Debe ser lo más compacta posible para facilitar que la cámara de medida sea también de un tamaño lo más reducido posible.
- Debe facilitar su limpieza tras la realización de los distintos ensayos.

La solución adoptada para cumplir estas condiciones implica el desarrollo de dos circuitos impresos. En el primero se montan únicamente la matriz con los sensores de gases, y de temperatura y humedad; y en el segundo se incluyen los conectores de la fuente de alimentación y del microcontrolador, siguiendo la siguiente estructura:

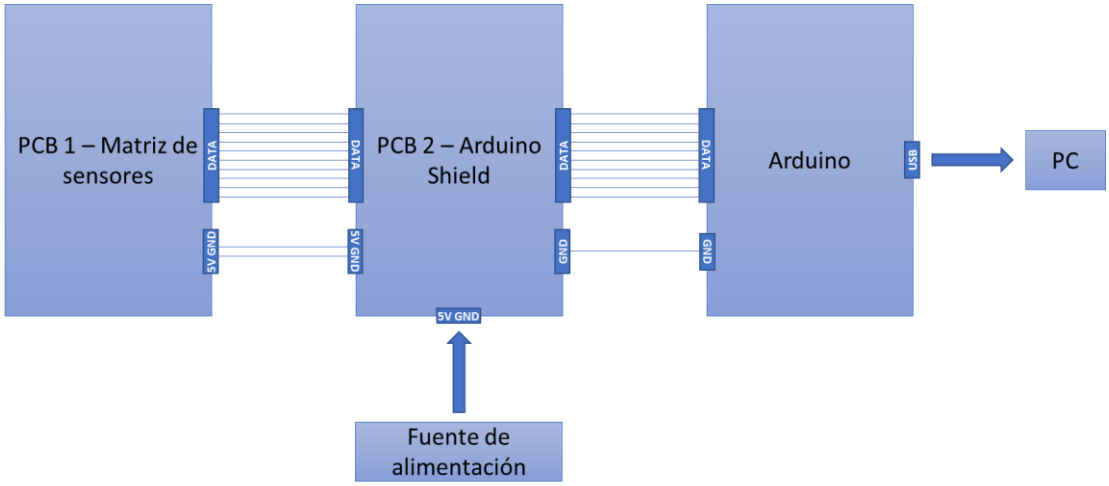


Figura 8 - Conexiones entre los circuitos impresos

Placa de circuito impreso de los sensores

Sobre esta placa se montan únicamente los sensores de temperatura y humedad y de gases, así como las resistencias y condensadores necesarios para su correcto funcionamiento. También se incluyen los conectores para la alimentación y la transmisión de los datos hacia el microcontrolador. Esta última se realiza a través de un cable de tipo plano flexible – FFC – de 18 hilos.

Los sensores se implementan en una de las caras del circuito impreso, que será la que se oriente hacia la cámara de medida y esté en contacto con los gases medidos. El resto de las componentes se montan en la otra cara de la placa utilizando tecnología SMD. Se consigue así que el lado de la placa de los sensores quede lo más libre posible de componentes para facilitar su limpieza.

Además, está provista de los taladros necesarios para que pueda ser atornillada a la cámara de medida de la nariz electrónica. De esta forma se facilita su montaje.

Placa de expansión para Arduino

La segunda placa, donde se conecta la fuente alimentación y Arduino, consiste en una placa de expansión – o *shield* – para el microcontrolador. Dispone de conectores para la alimentación del sistema a través de una fuente de alimentación y para la comunicación de los sensores con el microprocesador.

Se encarga de redirigir cada hilo de cada sensor hacia el pin de Arduino correspondiente, además de llevar la alimentación hacia la matriz de sensores a través del conector indicado y unir las masas de los diferentes componentes del sistema para que las lecturas sean correctas.

5.2.5. Alimentación

La potencia eléctrica requerida por la matriz de sensores se obtiene de una fuente de alimentación. Para cumplir con las condiciones del encargo, y teniendo en cuenta el tipo de sensores escogidos, esta fuente de alimentación debe cumplir los siguientes requisitos:

- **Salida de tensión de 5 Vdc**, pues es la empleada por la mayoría de los sensores de la serie MQ.
- Los sensores MQ son sensores resistivos, cuya salida está influenciada directamente por la tensión aplicada en su entrada. Por ello, la fuente de alimentación **debe proporcionar una tensión estable y sin fluctuaciones**, de forma que las medidas llevadas a cabo no se puedan ver influenciadas por este factor.
- **Debe ser barata.**
- **Debe ser lo más pequeña posible**, para facilitar su implementación en una nariz electrónica.

Teniendo en cuenta todos estos factores, se escoge la fuente de alimentación TXL 015-05S de la marca TRACO POWER. Sus principales características son:

- Potencia máxima de salida: 15W
- Tensión nominal de salida: 5 Vdc
- Corriente máxima de salida: 3,0 A
- Tensión de entrada: 100-240 Vac
- Rango de ajuste: 10%
- Variación máxima de la tensión de salida: 1%

Figura 9 - Fuente de alimentación TXL 015-05S

5.3. Microcontrolador

De cada sensor de gas que forma parte de la matriz del sistema de sensado se obtiene una lectura de tensión proporcional a la concentración del gas. Los sensores de temperatura y humedad escogidos, por su parte, envían la información codificada a través de una salida digital.

Un ordenador común no es capaz de leer estas señales de forma directa. Por ello, es necesario añadir un dispositivo al sistema de sensado que comunique la matriz de sensores con el ordenador para que el programa creado pueda tratar estos datos como corresponda.

Analizando las condiciones del encargo y las características de los sensores ya mencionadas, se establece que este dispositivo debe cumplir los siguientes puntos:

- **Entradas analógicas**. Por cada sensor de gas empleado, debe tener disponible una entrada analógica. La cantidad de estas entradas debe ser el mayor posible, para que la matriz pueda disponer de todos los sensores necesarios.
- **Entradas digitales.** Al igual que con las entradas analógicas, debe tener una entrada digital por cada sensor de temperatura y humedad empleado.
- **Barato.** El coste del dispositivo debe ser el menor posible para no encarecer en exceso el sistema completo.
- **Sencillo.** Dado que el sistema de sensado debe ser lo más sencillo posible, el dispositivo empleado debe ser fácil de montar y de programar.

Se consideran dos alternativas distintas para implementar este dispositivo:

Tarjeta de adquisición de datos

Una tarjeta de adquisición de datos es un dispositivo hardware que actúa como interfaz entre un ordenador y las señales provenientes de un sensor. Esta se encarga de digitalizar la información para que el computador sea capaz de leerla y procesarla.

A pesar de que muchas de estas tarjetas suelen ofrecer un buen rendimiento en cuanto a resolución, velocidad de muestreo y rango de entrada, no existe una opción barata que disponga de entradas digitales. Por tanto, no se pueden implementar los sensores de temperatura y humedad sin encarecer en exceso el sistema de sensado. Este problema hace que emplear una tarjeta de adquisición de datos sea inviable en este proyecto.

Microcontrolador

Tras analizar la opción anterior, se llega a la conclusión de que el empleo de un microcontrolador es la única alternativa realmente viable para transmitir la información entre el ordenador y la matriz de sensores.

A diferencia de las tarjetas de adquisición de datos, estos dispositivos sí que cuentan con entradas analógicas y digitales a un precio mucho más reducido. Además, a través de su programación se consigue no sólo adquirir los datos, sino también tratarlos para que el programa ejecutado en el ordenador se encargue únicamente de realizar las tareas relacionadas con la interfaz gráfica.

El microcontrolador escogido es *Arduino*, concretamente su versión *Mega 2560*. Posee un total de 16 entradas analógicas y 54 digitales, es barato, y muy sencillo de usar gracias a la cantidad de librerías existentes que facilitan y simplifican el desarrollo del código. Además, permite la

utilización de placas de expansión, lo que facilita el diseño de la placa de circuito impreso correspondiente.

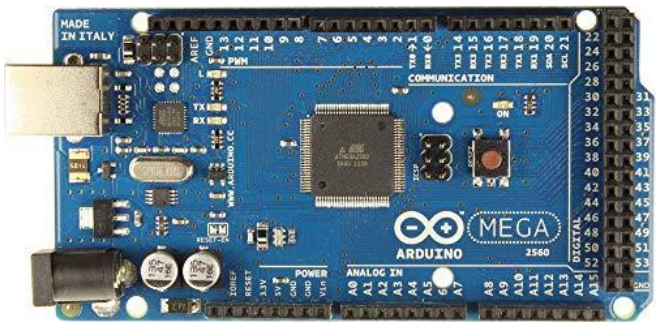


Figura 10 - Arduino Mega 2560

5.4. Visualización y guardado de datos

Una de las condiciones establecidas previas al comienzo del desarrollo del proyecto es la creación de un programa que, mientras se ejecuta en un ordenador, sea capaz de mostrar los datos recibidos de los sensores y permita guardarlos en un fichero de texto para su posterior análisis.

Existen múltiples lenguajes de programación enfocados a crear interfaces gráficas y que permiten la creación de un programa con las características demandadas. Algunos de los más populares y usados son *C++*, *Java*, *Javascript*, *Php* o *Glade*. No obstante, se emplea el lenguaje y entorno de desarrollo *Processing*. Los motivos por los que se ha escogido frente a las demás opciones son los siguientes:

- Está influido por otros lenguajes como *OpenGL* o *C*, pero principalmente por *Java*. Hereda de este último su **enfoque orientado a objetos** y su alta capacidad **para trabajar con entornos gráficos**.
- Su sintaxis es una versión simplificada de la de *Java*, lo que lo convierte en un lenguaje más **sencillo de usar**.
- Al estar basado en *Java*, también se ejecuta sobre una máquina virtual. Se asegura así la **portabilidad del programa**, independientemente de la arquitectura de la computadora en la que se ejecuta.
- El IDE de Arduino está basado en el de *Processing*. Esto implica una completa **compatibilidad** entre ambos, complementándose a la perfección.
- Existe una **gran cantidad de librerías** desarrolladas por los usuarios que facilitan la creación del programa y simplifican el código.

En cuanto a la funcionalidad del programa, éste dispone de las siguientes características que cumplen con los requisitos demandados:

- La **visualización a tiempo real** de los valores de tensión a la salida de cada uno **de los sensores de gases MQ**.
- La **visualización a tiempo real** de la temperatura y humedad leídas por los **sensores DHT11**.
- En caso de que la lectura de uno de los sensores se encuentre fuera del rango de medida, se **indica la existencia de un fallo** en el sensor correspondiente.
- Permite **visualizar gráficamente la evolución** en las medidas de los sensores a través de tres gráficos: uno para los sensores de gases, otro para la temperatura y un último para la humedad.
- Al iniciar el programa, da la opción al usuario de **introducir el periodo de muestreo** más adecuado para el ensayo que se va a realizar.

6. Descripción detallada de la solución adoptada

A continuación, se presentan en detalle las soluciones adoptadas para los distintos subsistemas que componen el sistema de medición completo, así como el dimensionado de sus componentes. Se detalla también el procedimiento seguido durante el desarrollo del proyecto.

6.1. Matriz de sensores

6.1.1. Selección de los sensores

El primer paso para desarrollar la matriz de sensores es la selección de los sensores de la serie MQ que se van a utilizar en ella. Para ello, se ha comenzado por analizar cuáles son los factores que limitan la cantidad de estos sensores. Son los siguientes:

- El Arduino Mega escogido posee un total de 16 entradas analógicas, por lo que se podrán escoger como máximo 16 sensores MQ.
- La corriente máxima de salida que es capaz de proporcionar la fuente de tensión escogida es de 3.000 mA a 5 Vdc, por lo que la intensidad requerida por los sensores no deberá ser superior a ésta.
- Sólo se pueden emplear aquellos sensores MQ que se alimentan a 5 Vdc para simplificar su implementación.

Una vez determinados los factores limitantes, se ha pasado a obtener información sobre cuáles son los distintos tipos de sensores de la serie MQ que hay disponibles, así como sus principales características eléctricas. Con ello, se ha elaborado la siguiente tabla:

Sensor	Tensión de alimentación	Potencia máxima	Intensidad máxima
MQ-2	5 V	900 mW	180 mA
MQ-3	5 V	900 mW	180 mA
MQ-4	5 V	900 mW	180 mA
MQ-5	5 V	900 mW	180 mA
MQ-6	5 V	900 mW	180 mA
MQ-7	Entre 5 V y 1,5 V	350 mW	70 mA
MQ-8	5 V	900 mW	180 mA
MQ-9	Entre 5 V y 1,5 V	350 mW	70 mA
MQ-135	5 V	900 mW	180 mA

Figura 11 - Comparativa de las características eléctricas de los sensores MQ

Como se puede observar, no todos los sensores presentan las mismas características eléctricas. Aunque la mayoría deben ser alimentados a 5 V de tensión continua, existen algunos que requieren una alimentación especial, en la que el sensor se alimenta a 5 V durante 60 segundos y, posteriormente, a 1,5 V durante 90 segundos.

Con esta información, se escogen los 7 sensores que se alimentan a 5 V, que son todos excepto los sensores MQ-7 y MQ-9. Éstos se descartan dado que su implementación en la matriz no es viable debido a la alimentación que requieren.

Puesto que aún quedan 9 entradas analógicas de Arduino sin usar del total de 16 de las que dispone, se decide duplicar todos los sensores. De esta forma, se cubren otras 7 entradas y se consigue la redundancia requerida en los sensores, con la que se consigue un sistema más fiable.

Para cubrir las 2 entradas analógicas que quedan libres, se añade un tercer sensor de dos modelos para terminar de completar las entradas analógicas. Para llevar a cabo la selección de los dos modelos de los que se va a añadir una tercera unidad, se prepara la siguiente tabla resumen, donde se indican los principales gases que cada sensor es capaz de detectar:

Sensor	Gases detectables											
	Combustible gas	Smoke	Alcohol	Methane	Propane	Butane	Liquefied petroleum	LPG	Hydrogen	Ammonia	Sulfide	Benzene vapor
MQ2												
MQ3												
MQ4												
MQ5												
MQ6												
MQ8												
MQ135												
Total	1	1	1	2	3	3	1	1	1	1	1	1

Figura 12 - Comparativa de gases detectables por los sensores MQ

Como se puede observar, hay gases que pueden ser detectados por varios modelos simultáneamente, como es el caso del propano. Otros, por el contrario, sólo son detectados por un único sensor. Por ejemplo, el amoniaco que sólo puede ser detectado por el MQ-135.

Se decide añadir, pues, una tercera unidad de los sensores MQ-2 y MQ-3. Con esto se consigue que en la matriz haya la mayor cantidad posible de sensores sensibles a cada tipo de gas.

Una vez escogidos todos los sensores de gases que se van a emplear, se comprueba que la fuente de alimentación escogida es capaz de proporcionar toda la potencia que requieren. Para ello se calcula la intensidad máxima que puede llegar a consumir un sensor, y posteriormente el total que consume el conjunto de sensores:

$$I_{máx} = \frac{P_{máx}}{Vcc} = \frac{900\ mW}{5\ V} = 180\ mA \quad [1]$$

$$16\ sensores \cdot 180\ mA/sensor = 2.880\ mA \quad [2]$$

Estos 2.880 mA son menores que los 3.000 mA máximos que puede entregar la fuente. Por lo tanto, no tendrá problemas para alimentar la matriz de sensores y no es necesario descartar ninguno de los sensores escogidos debido a esta limitación.

Una vez seleccionados los sensores de gases, se pasa a determinar las unidades de los sensores DHT11 que se van a usar.

Los sensores DHT11 presentan un consumo de tan sólo 0,3 mA. Además, transmiten su información a través de una señal digital, por lo que cada sensor requiere disponer de un puerto digital libre en Arduino. Estos dos factores no son limitantes a la hora de escoger el número de sensores que se van a emplear, pues la fuente de alimentación es capaz de soportarlos sin problemas y el Arduino Mega escogido dispone de todas sus salidas digitales libres.

Se escogen finalmente 2 sensores DHT11. Con este número de unidades, se consigue una medida redundante, además de no aumentar en exceso el tamaño de la placa de circuito

impresa diseñada, lo que ayuda a cumplir las condiciones del encargo establecidas previamente al comienzo del proyecto.

6.1.2. Implementación en PCB

Todos los sensores escogidos y que forman parte de la matriz se montan sobre una placa de circuito impreso diseñada íntegramente para la aplicación de este proyecto, utilizando el programa de diseño electrónico y simulación *Proteus Design Suite*.

Antes de comenzar con el diseño de la placa de circuito impreso, es necesario crear los componentes correspondientes a los sensores MQ y DHT11, pues no vienen incluidos en las librerías de *Proteus*.

Para ello, en primer lugar, se crea el símbolo esquemático de cada sensor con ayuda de las herramientas de dibujo de gráficos 2D de las que dispone Proteus. Una vez realizados, se asignan los pines correspondientes a cada sensor utilizado la opción *Terminals mode.* El resultado obtenido es el siguiente:

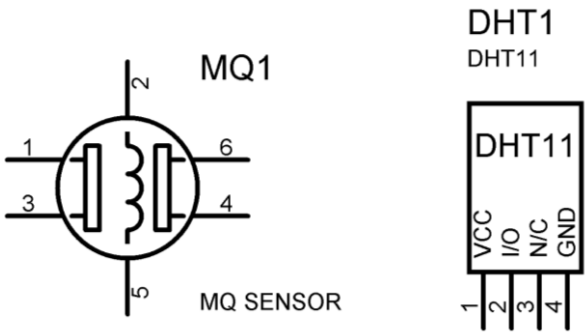


Figura 13 - Componentes MQ y DHT11

A continuación, se crea el *footprint* de cada componente. Esto es, los taladros y PADs donde se ubica cada componente una vez impresa la placa. Para ello, se hace uso de las herramientas *2D Graphics Mode* y *DIL PAD Mode* para generar el modelo de cada componente y, posteriormente, se generan sendos packages asignando el número correspondiente a cada pin. Los resultados obtenidos son los siguientes:

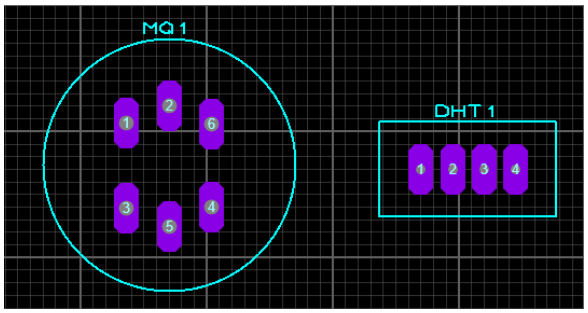


Figura 14 - Footprint de los componentes MQ y DHT11

Por último, se genera cada uno de los componentes, asignando un nombre y una librería, y asociando el footprint diseñado.

Una vez generados los componentes, se diseña el circuito electrónico necesario para cada uno de ellos y que incorpora los componentes electrónicos requeridos para su correcto funcionamiento. En el caso de los sensores MQ, se implementan en base al siguiente esquemático:

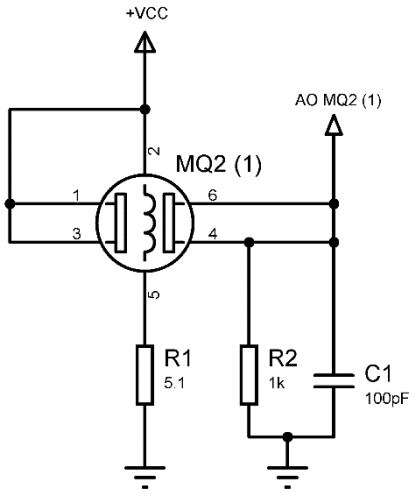


Figura 15 - Circuito electrónico para los sensores MQ

La funcionalidad de la resistencia R1 de 5,1 Ω es la de limitar la corriente que circula a través del calentador del sensor. Así, teniendo en cuenta que la resistencia del calentador es de 31 ± 3 Ω:

$$I_{H\,máx} = \frac{Vcc}{R_{H\,mín} + R_1} = \frac{5\ V}{28\ \Omega + 5{,}1\ \Omega} = 151\ mA \qquad [3]$$

La resistencia R2 de 1 kΩ conforma un divisor de tensión junto con el sensor, de forma que se obtiene una salida de tensión directamente proporcional a la cantidad del gas sensado. Por último, el condensador C1 de 100pF se utiliza para estabilizar la tensión de salida, evitando las fluctuaciones que se puedan producir en ésta. El encapsulado seleccionado para todos estos componentes es el 1206 de tecnología SMD. De ¼ W, es capaz de soportar la potencia máxima necesaria por ambas resistencias:

$$P_{R1\,máx} = I_{R1\,máx}{}^2 \cdot R_{1\,máx} = \left(\frac{Vcc}{R_{H\,mín} + R_1}\right)^2 \cdot R_1 = \left(\frac{5}{28 + 5{,}1}\right)^2 \cdot 5{,}1 = 116{,}37\ mW \qquad [4]$$

$$P_{R2\,máx} = \frac{V_{out\,máx}{}^2}{R_2} = \frac{5^2}{1\ k} = 25\ mW \qquad [5]$$

Los sensores de temperatura y humedad DHT11 se montan en base al siguiente circuito electrónico:

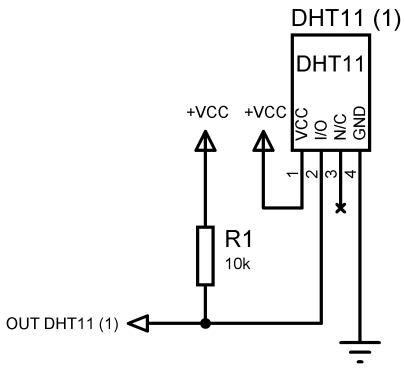


Figura 16 - Circuito electrónico para los sensores DHT11

La resistencia R_1 se trata de una resistencia de *Pull Up* de 10 kΩ y, al igual que en el caso de las empleadas en sensores MQ, su encapsulado es el 1206 de tecnología SMD.

$$P_{R1\ máx}=\frac{V_{out\ máx}{}^2}{R_2}=\frac{5^2}{10\ k}=2{,}5\ mW \qquad [6]$$

A continuación, se diseña la placa de circuito impreso. En ésta, los sensores MQ se disponen siguiendo una distribución de cuatro filas y cuatro columnas, en la que la posición que ocupa cada sensor se escoge específicamente con el objetivo de que los del mismo tipo no se sitúen en posiciones adyacentes. De esta forma se aumenta la robustez en la medida, pues se consigue que los resultados obtenidos por la nariz electrónica no se vean afectados por una concentración de gases volátiles más alta en una zona concreta de la placa que en el resto. La distribución escogida es la siguiente:

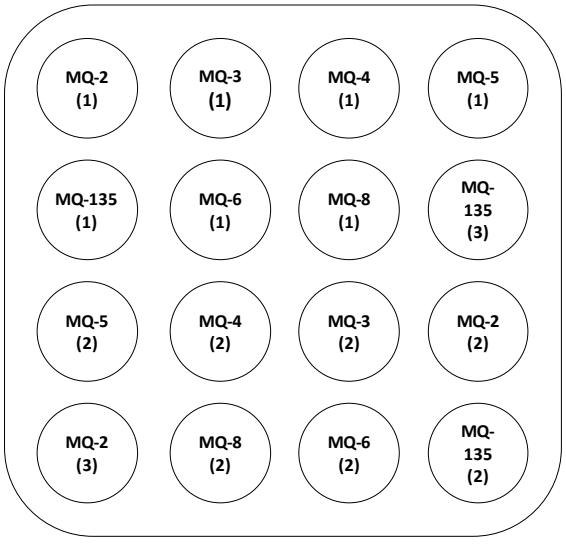


Figura 17 - Disposición de los sensores MQ

Los sensores de temperatura y humedad DHT11 se montan sobre este mismo circuito impreso, en una posición cercana a los sensores de gases para que su lectura sea lo más correcta posible, en el sentido de que la temperatura y la humedad ambientales obtenidas sean las mismas a las que están sometidas los sensores MQ.

La alimentación de todos estos sensores se obtiene desde la placa de expansión de Arduino – que proviene, a su vez, de la fuente de alimentación – utilizando un conector tipo T-Block de dos terminales, uno para la tensión de 5 V y otro para masa. Por otro lado, se emplea un conector de tipo FFC – *Flat Flexible Cable* – de 18 pines para transmitir los datos de la placa de los sensores al Shield de Arduino. En ambos conectores se emplea tecnología SMD de igual forma que en las resistencias y condensadores empleados.

Por último, los sensores se montan hacia un lado de la placa, que será el que se encuentre en contacto con los gases de la cámara de medida. Las resistencias, condensadores y conectores se montan en el lado contrario. De esta forma, se dispone de una placa con la menor cantidad posible de componentes y taladros en la zona orientada hacia la cámara de medida, facilitando así su limpieza tras llevar a cabo los ensayos.

El tamaño final de la placa diseñada es de 14 x 17 cm, y la zona reservada para los sensores es de 13 x 14 cm. Además, dispone de un plano de masa para reducir las interferencias eléctricas que se puedan producir y dispone de cuatro taladros en sus esquinas para facilitar su montaje en la cámara de medida de la nariz electrónica.

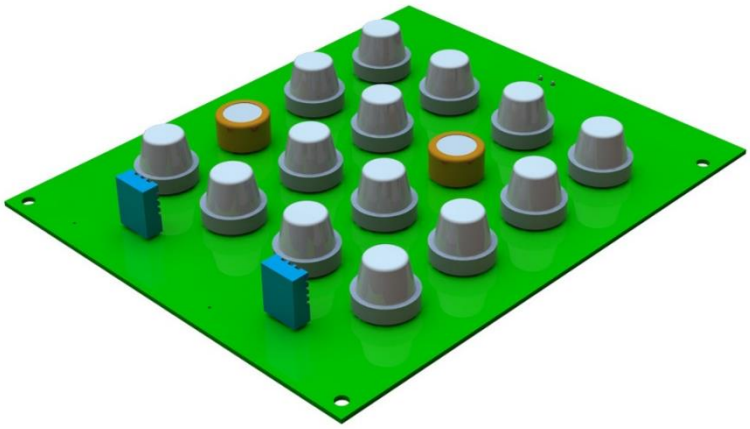


Figura 18 - PCB Sensores, cara superior

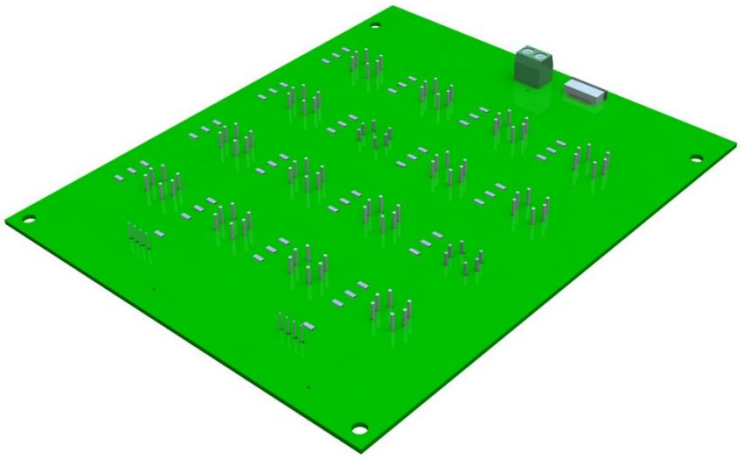


Figura 19 - PCB Sensores, cara inferior

6.2. Shield de Arduino

Se ha diseñado una segunda placa de circuito impreso con el objetivo de simplificar el circuito impreso de la matriz de los sensores, de forma que se consigue reducir su tamaño y su complejidad. Se ha empleado el mismo software que en el caso del circuito impreso para la matriz de los sensores: *Proteus Design Tool*.

Esta placa está diseñada para ser montada a modo de Shield sobre el Arduino Mega empleado. Por tanto, su tamaño y forma es similar al del propio Arduino para facilitar el montaje del conjunto en la nariz electrónica. Además, dispone de cuatro taladros en misma posición que aquellos de los que dispone Arduino para facilitar su montaje.

Consta de un conector de tipo FFC similar al de la PCB de la matriz de sensores, desde donde se reciben los datos sensados y se redirijen hasta los pines digitales y analógicos correspondientes a través de pistas de cobre. Los pines a los que se conecta cada sensor son los siguientes:

Sensor	Pin	Sensor	Pin
MQ-2 (1)	A15	MQ-2 (2)	A4
MQ-3 (1)	A14	MQ-3 (2)	A5
MQ-4 (1)	A13	MQ-4 (2)	A6
MQ-5 (1)	A12	MQ-5 (2)	A7
MQ-6 (1)	A10	MQ-6 (2)	A1
MQ-8 (1)	A9	MQ-8 (2)	A2
MQ-135 (1)	A11	MQ-135 (2)	A0
MQ-2 (3)	A3	MQ-135 (3)	A8
DHT11 (1)	D2	DHT11 (2)	D3

Figura 20 - Tabla de las conexiones de los sensores con Arduino

Dispone también de dos conectores tipo T-Block de dos terminales. A uno de ellos se conectan los cables de la fuente de alimentación, mientras que el segundo se emplea para llevar esta alimentación hasta la placa de los sensores. Por tanto, la placa dispone de rutas de cobre que conectan estos dos terminales. Además, también conecta la masa de la fuente de alimentación con la masa de Arduino, para que las medidas obtenidas sean correctas.

El resultado final es el siguiente:

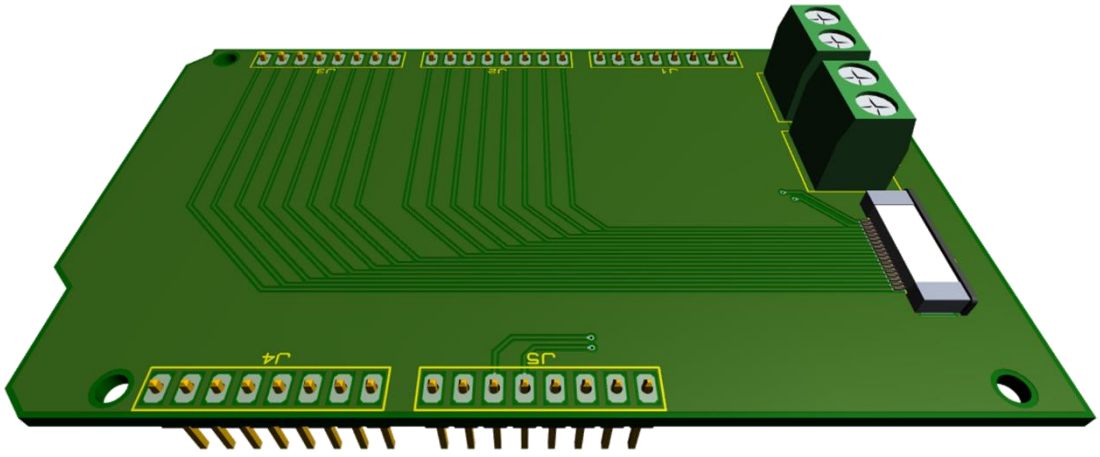


Figura 21 - Shield de Arduino

6.3. Alimentación

La potencia eléctrica requerida por la matriz de sensores proviene de la fuente de alimentación Traco Power TXL 015-05S escogida en el *Apartado 5.2.5*. Esta fuente dispone de las siguientes conexiones:

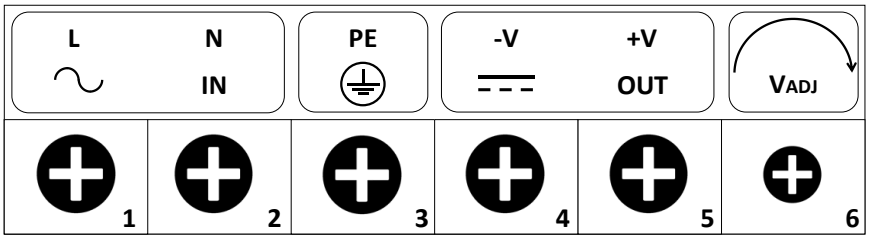


Figura 22 - Conexiones de la fuente de alimentación

1. **Alimentación, fase.** Este terminal se conecta a la fase de la red eléctrica a través de un cable de tres hilos y una clavija apta para los enchufes domésticos.
2. **Alimentación, neutro.** Al igual que el terminal anterior, se conecta al neutro de la red eléctrica usando el mismo cable.
3. **Alimentación, tierra.** De la misma forma, se conecta a la puesta a tierra de la instalación a través del cable empleado para la alimentación.
4. **Salida, masa.** La masa se conecta al terminal correspondiente del componente T-Block de la placa Shield de Arduino. Desde ahí, se conecta con las masas de los sensores, así como con la masa de Arduino para una correcta lectura de los datos.
5. **Salida, activo.** Al igual que la masa, se conecta al terminal del T-Block, desde donde se alimentan los sensores.
6. **Ajuste de tensión.** Se trata de un potenciómetro que permite ajustar el nivel de tensión de salida. Con ayuda de un destornillador y de un multímetro, se ajusta hasta conseguir una tensión de salida de 5 Vdc.

Por otro lado, para unir los terminales T-Block de las dos placas de circuito impreso diseñadas, de forma que se consigue alimentar la matriz de sensores, se emplea otro cable de similares características.

Realizando todas estas conexiones, se consigue que el sistema completo disponga de la alimentación eléctrica necesaria para su funcionamiento.

6.4. Arduino

Una vez diseñada la matriz de sensores y establecida la conexión de los sensores con los diferentes puertos del microcontrolador Arduino, se pasa a programarlo para que lea y procese los datos correctamente. Este programa se ha desarrollado utilizando el propio entorno de desarrollo de Arduino, y en su creación se ha seguido una metodología de modularización. Las principales funciones que realiza el programa son las siguientes:

- Lee el periodo de muestreo que recibe por el puerto serie desde el programa de Processing.
- Realiza el control sobre el periodo de muestreo.
- Lee los datos de los sensores y los almacena en paquetes de bytes.
- Envía los paquetes de bytes generados a Processing a través del puerto serie.

El código se encuentra dividido en los módulos *Main*, *Process* y *SerialCommunication*:

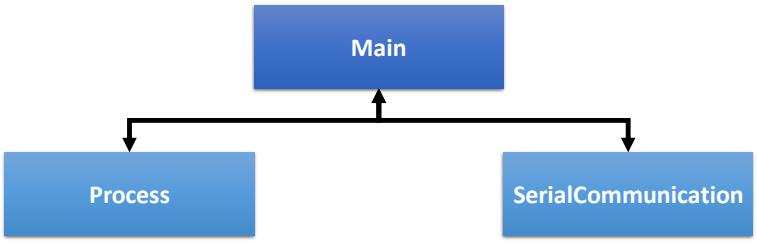


Figura 23 - Módulos del programa de Arduino

6.4.1. Main

En el módulo *Main* se declaran las variables globales utilizadas en los diferentes módulos e incluye la librería *DHT11.h* utilizada para leer los sensores de temperatura y humedad. Esta librería ha sido desarrollada por los usuarios de Arduino y permite declarar los sensores DHT11 empleados y obtener sus lecturas, de una forma sencilla y eficaz.

Se encuentra dividido en dos funciones principales. La primera, *setup*, se ejecuta una única vez al inicio del programa y en ella se realizan las siguientes acciones de forma secuencial:

1. Se inicializa el puerto serie a una velocidad de 9600 baudios.
```
Serial_Port_Init();
```
2. Se inicializan los pines analógicos donde se conectan los sensores MQ.
```
MQ_Sensors_Init();
```
3. Se recibe el periodo de muestreo de Processing a través del puerto serie y se almacena en la variable correspondiente.
```
Get_Sampling_Period();
```
4. Se establece el contacto con Processing para iniciar el envío de los paquetes de bytes que contienen la información de los sensores.
```
Establish_Contact();
```

La función *loop*, por su parte, realiza las tareas incluidas en su interior de forma cíclica, volviendo al inicio una vez termina de ejecutar todas las funciones. Estas tareas son las siguientes:

1. Comienza a contar el tiempo que tarda en leer los sensores y enviar la información.
```
uint32_t ts1 = millis();
```
2. Lee los datos de los sensores de gases MQ.
```
Read_MQ_Sensors();
```
3. Lee la temperatura y humedad obtenidas de los sensores DHT11.
```
Read_DHT11_Sensors();
```
4. Envía los datos a Processing.
```
Send_Data();
```
5. Termina de contar el tiempo.
```
uint32_t ts2 = millis();
```
6. Realiza una pausa igual al periodo de muestreo, teniendo en cuenta lo que se ha tardado en realizar las acciones anteriores. De esta forma, se asegura que el periodo de muestreo se cumple y no se ve afectado por el tiempo de ejecución del resto de las funciones.
```
delay(samplingPeriod - (ts2 - ts1));
```

6.4.2. Process

El módulo *Process* contiene las funciones relacionadas con la inicialización de los pines del microprocesador donde se conectan los sensores, y con la lectura y guardado de los datos de esos mismos sensores. Estas funciones son las siguientes:

MQ_Sensors_Init. Configura los 16 pines analógicos como entradas para la conexión de los sensores de gases MQ.

Read_MQ_Sensors. Lee el valor obtenido en los diferentes pines analógicos correspondientes a la tensión de salida de los sensores MQ. Posteriormente, lo divide en dos bytes y los guarda en un array - que contiene un total de 32 bytes, dos por cada sensor de gases – para ser enviados posteriormente a Processing.

Arduino proporciona un valor entero comprendido entre 0 y 1023 en función de la tensión aplicada en el puerto analógico correspondiente – 0 para 0,0 V y 1023 para 5,0 V –; por tanto, se determina que la resolución del convertidor ADC es de aproximadamente 5 mV:

$$\text{Resolución} = \frac{5\ \text{V}}{1023} = 4{,}89\ \text{mV} \approx 5\ \text{mV} \qquad [7]$$

Es por ello por lo que esta función se encarga también de proporcionar el valor leído con una resolución de 5 mV, pues transmitir el valor a Processing con mayor resolución de la del convertidor ADC de Arduino no tiene sentido. Esto se realiza aproximando el valor leído al valor correspondiente:

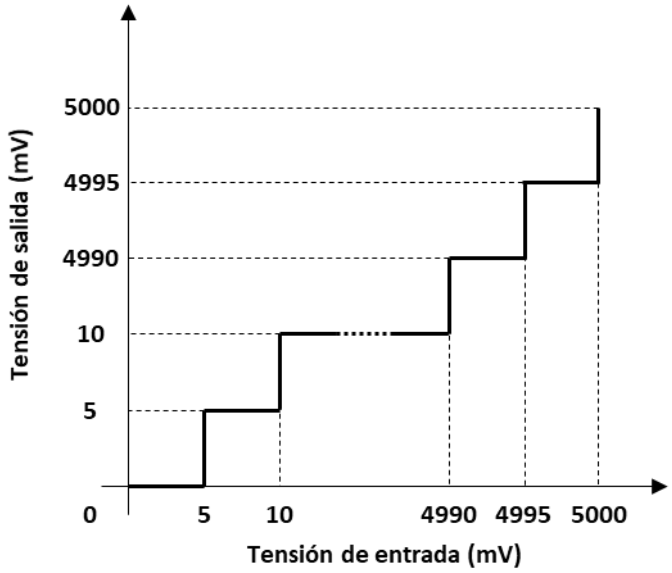


Figura 24 - Resolución en las lecturas de los sensores MQ

Read_DHT11_Sensors. Esta función hace uso de la librería *DHT11.h* para declarar los sensores en los puertos digitales correspondientes, leer la temperatura y humedad de cada sensor y guardar los datos en un array de cuatro bytes, uno por cada magnitud de cada sensor. A diferencia de los valores medidos por los sensores MQ, en este caso no es necesario dividir cada dato en dos bytes. Esto es debido a que uno es suficiente ya que las medidas de temperatura y humedad proporcionadas por el DHT11 son siempre inferiores a 255 – el máximo valor que es capaz de almacenar un byte –.

6.4.3. SerialCommunication

El módulo *SerialCommunication* contiene las funciones dedicadas a la comunicación con Processing; es decir, el envío y lectura de los datos a través del puerto serie. Estas funciones son:

Serial_Port_Init. Inicializa el puerto serie a 9600 baudios.

Get_Sampling_Period. Recibe el periodo de muestreo enviado por Processing. Incluye el proceso de requerir esta información a Processing, esperar su respuesta, y leer y almacenar el periodo de muestreo una vez lo recibe. Puesto que la comunicación se realiza byte a byte, este dato llega dividido en dos bytes; es por ello que esta función también incluye el cálculo en el que se unen los dos bytes en una única variable de tipo entero (*int*).

Establish_Contact. Establece un primer contacto con Processing iniciar el envío de los datos de los sensores.

Send_Data. Envía los paquetes de bytes a Processing a través del puerto serie. Estos paquetes contienen la información tanto de los sensores de gases como de los de temperatura y humedad.

6.4.4. Flujograma

El flujograma básico del programa de Arduino es el siguiente:

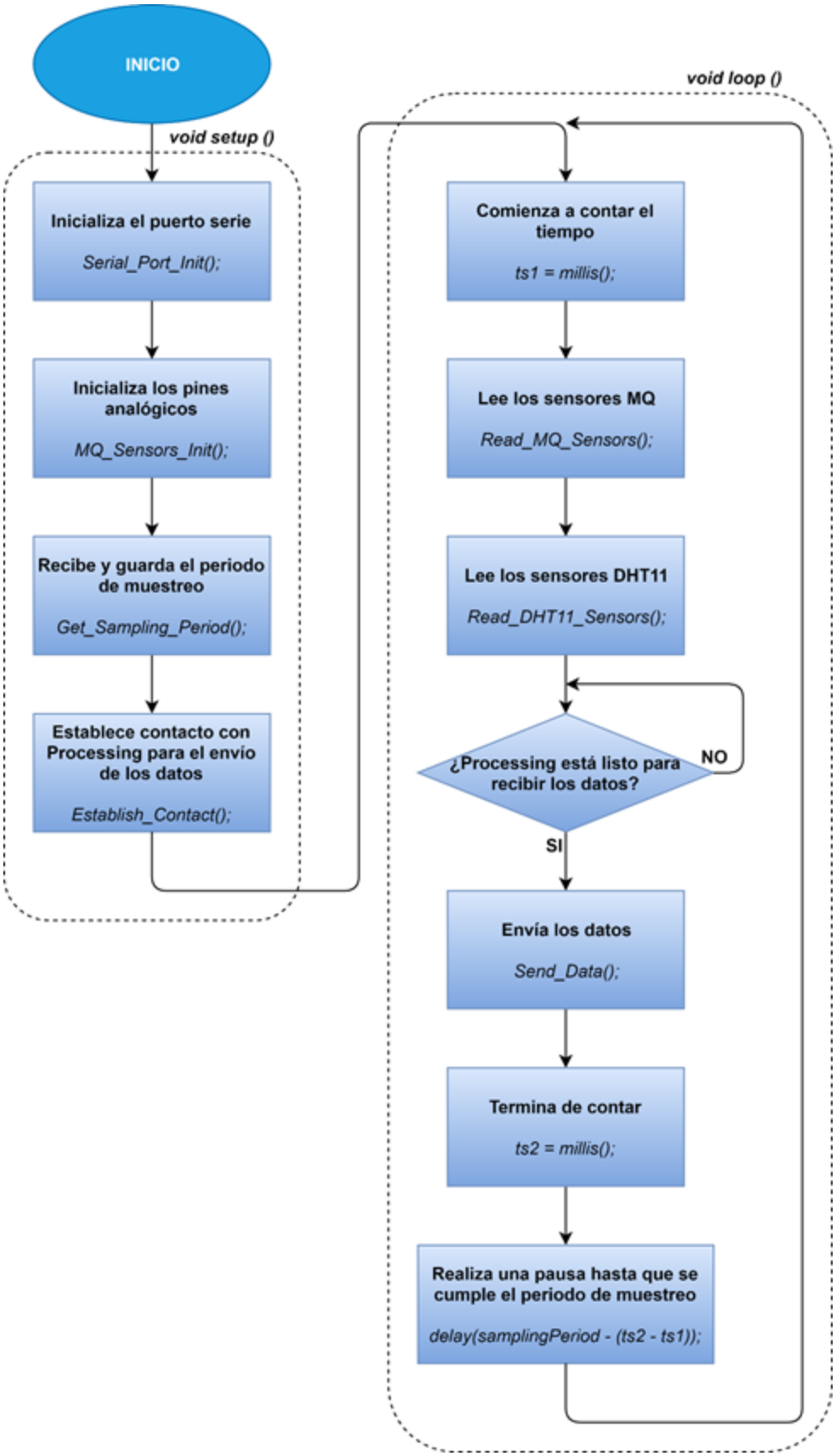


Figura 25 - Flujograma Arduino

6.5. Comunicación Arduino – Processing

Una vez definido el programa de Arduino, se determina el método de comunicación que se va a llevar a cabo entre Arduino y Processing para transmitir el periodo de muestreo y los datos de los sensores. Así, ésta consta principalmente de dos fases:

1. En primer lugar, y una vez el usuario ha introducido el periodo de muestreo en Processing, éste se envía a Arduino para que pueda comenzar a leer los sensores con la temporización adecuada.
2. Tras esto, se establece un contacto entre Arduino y Processing en el que ambos comunican si están listos para comenzar a enviar y recibir los datos de los sensores, respectivamente.
3. Finalmente, el microcontrolador comienza a enviar paquetes de datos con la información tomada de los sensores en cada periodo de muestreo.

Toda esta información se envía a través del puerto serie en forma de paquetes bytes. En total, se envían tres paquetes distintos, cada uno de los cuales contiene una información determinada:

- **El periodo de muestreo**. Este array se transmite una única vez al comienzo del programa y contiene un total de 3 bytes, suficientes para contener cualquier periodo de muestreo que pueda requerir el usuario en su aplicación.
- **Los datos de los sensores de gases**. Está compuesto por un total de 32 bytes, 2 por cada sensor.
- **Los datos de temperatura y humedad**. Contiene 4 bytes, uno por cada magnitud de cada sensor.

6.5.1. Secuencia de comunicación

Los distintos pasos que llevan a cabo los programas de Arduino y Processing son los siguientes:

1. En primer lugar, ambos programas inicializan el puerto serie a una velocidad de 9600 baudios o bits por segundo y realizan una pausa hasta que ésta se ha iniciado correctamente.
2. A continuación, Arduino pide a Processing que envíe el periodo de muestreo mediante la escritura del carácter ‘T’ en el puerto serie cada 300 ms. Cuando Processing recibe esta letra, la devuelve a Arduino para indicar que ha recibido la petición, y realiza una pausa de 100 ms antes de enviar el paquete de tres bytes en el que se incluye el periodo de muestreo. Arduino recibe este paquete de datos y lo guarda en la variable correspondiente.
3. Se inicia el proceso de lectura, envío y recepción de los datos leídos en los sensores. Este proceso comienza con la escritura del carácter ‘A’ en el puerto serie realizada por Arduino, indicando que está listo para comenzar a enviar los datos. Cuando Processing recibe este carácter, lo devuelve a Arduino para comunicar que también está listo para comenzar a recibir y guardar la información.
4. Es en este momento cuando Arduino lee la información de los sensores por primera vez, y la envía a través de los paquetes de bytes. Processing recibe la información, y la lee y guarda byte a byte. Cuando ha almacenado todos los bytes, indica a Arduino que ha completado el guardado a través del envío de ‘A’ por puerto serie.

5. En ese momento Arduino realiza una pausa hasta que se ha cumplido el periodo de muestreo. Entonces, vuelve a leer los sensores y envía los nuevos datos a Processing, repitiéndose el proceso hasta el apagado del sistema.

Todas las funciones empleadas por ambos programas para llevar a cabo la comunicación serial se incluyen en sendos módulos *SerialCommunication.* La explicación detalla de las funciones incluidas en estos dos módulos se halla a continuación.

6.5.2. Flujograma

El siguiente flujograma muestra de forma esquemática cómo se realiza la comunicación entre Arduino y Processing:

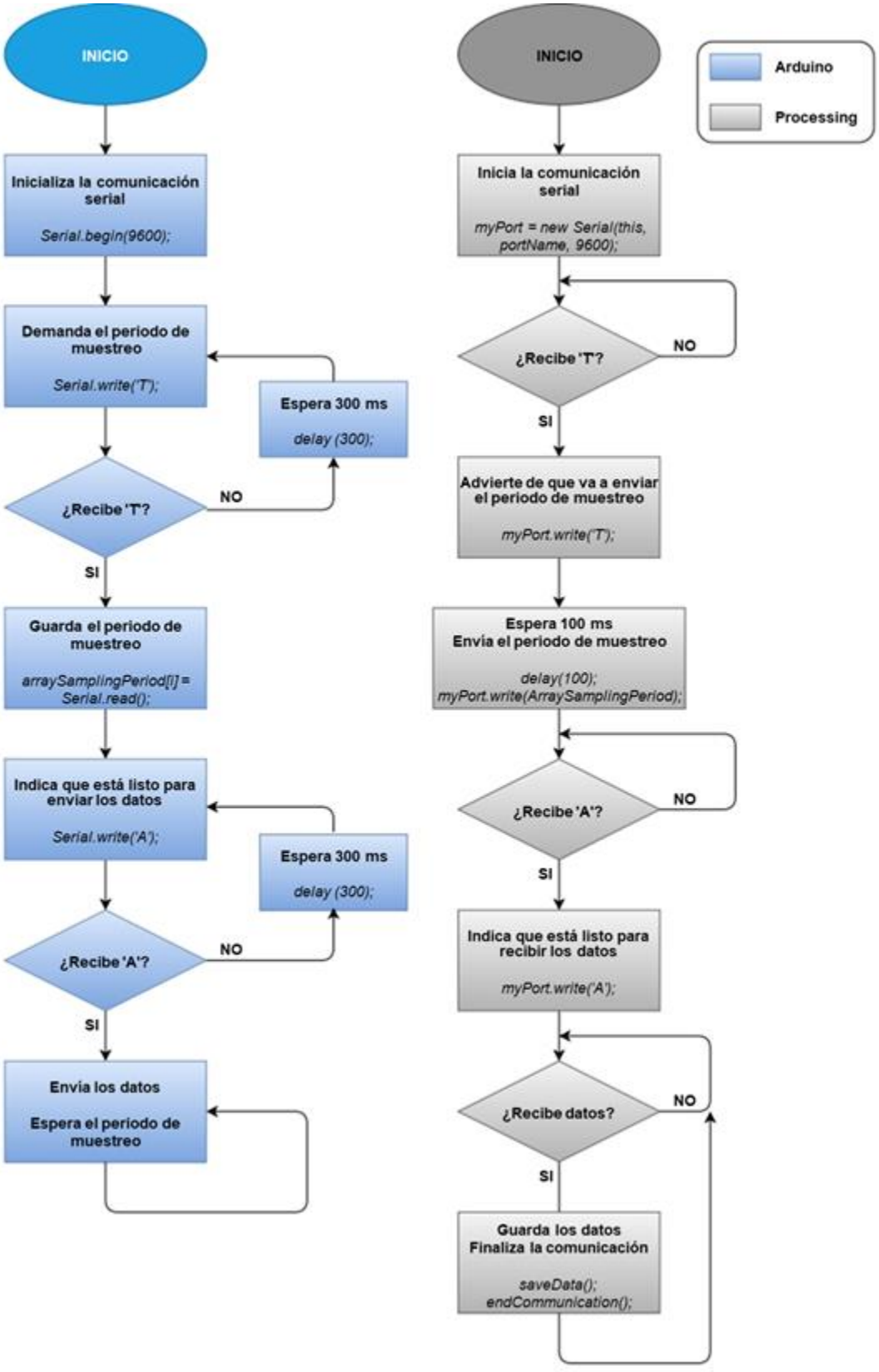


Figura 26 - Flujograma comunicación Arduino - Processing

6.6. Processing

El programa desarrollado en Processing tiene como principal objetivo proporcionar al usuario una interfaz amigable, sencilla e intuitiva que represente de forma visual los valores obtenidos de cada sensor y dibuje gráficos de la evolución de esta información a lo largo del tiempo. Permite además, al iniciar el programa, introducir el periodo de muestreo deseado para la aplicación y exportar los datos a un fichero de texto en formato *.csv*.

Respecto al código, este programa se ha dividido en los módulos *Main, Data, MainGUI, SerialCommunication, Styles, classGraph, classSensorCard* y *gui*. El flujo de información entre los diferentes módulos es el siguiente:

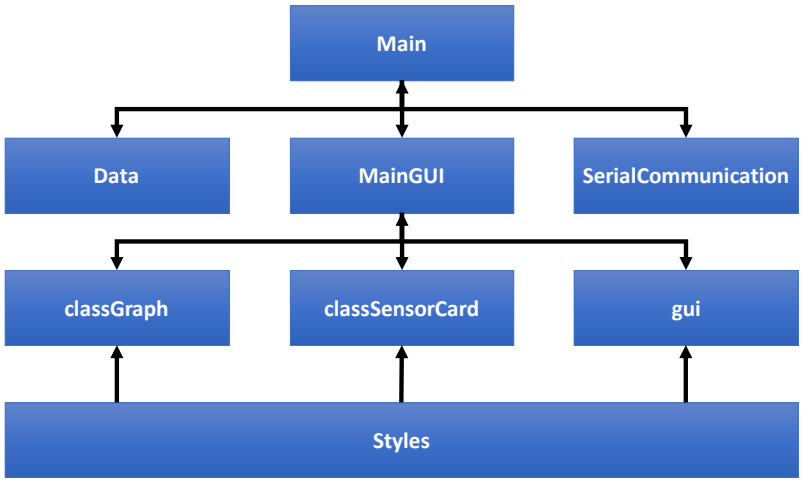


Figura 27 - Módulos del programa de visualización de datos

6.6.1. Main

Main es el módulo principal del programa. En él se inicializa la ventana principal del programa con un tamaño de 800 x 600 píxeles y se declaran todas las librerías empleadas en el resto del código:

- ***Processing.serial***. Empleada para realizar la comunicación serie con Arduino.
- ***G4p_controls***. Librería empleada por la herramienta *G4P GUI Builder* para crear los distintos elementos de la interfaz gráfica y que éstos funcionen correctamente.
- ***Peasy***. Librería utilizada también por la herramienta *G4P GUI Builder*. Utilizada para que las acciones realizadas con el ratón funcionen de una manera más eficiente.

Las tres funciones principales que componen este módulo son las siguientes:

Setup. Similar a la función de igual nombre en Arduino. Se ejecuta una única vez al inicio del programa. En ella, se inicializa la interfaz de usuario, el puerto serie, se declaran las tablas donde se guardan los datos y se realiza el envío del periodo de muestreo.

Draw. Es la función equivalente a *loop* en Arduino. Se ejecuta cíclicamente y se encarga de mostrar los elementos que corresponden en la interfaz de usuario.

***serialEvent*.** Esta función se ejecuta cada vez que se recibe un dato por el puerto serie. En ella, se inicia la comunicación con Arduino o se reciben y guardan los datos recibidos, según corresponda.

6.6.2. Data

Este módulo se encarga de tratar los datos recibidos, guardarlos en objetos de tipo *Table* y exportar estas tablas a los ficheros *.csv* correspondientes. Sus principales funciones son:

initData. Inicializa los dos objetos tipo *Table* donde se guardan los datos. La primera tabla –llamada *allDataRead* – almacena todos los datos recibidos desde el inicio del programa, y que se usan para dibujar los gráficos de los sensores. La segunda – *dataToSave* – contiene los datos recibidos mientras el guardado de datos está activo, y es la que posteriormente se exporta al fichero de texto con formato *.csv*.

defineTable. Define la cabecera de una tabla. Cada una contiene 22 columnas: una para los valores de cada sensor MQ, y dos por cada sensor DHT11 – una para temperatura y otra para humedad –.

insertDataIntoTable. Inserta los datos recibidos en el último periodo de muestreo en una nueva fila de la tabla indicada.

saveTheTable. Exporta la tabla indicada en un fichero *.csv*, en la ruta y con el nombre predefinidos.

cancelTable. Limpia los datos contenidos en una tabla, excepto las cabeceras declaradas con la función *defineTable*.

saveData. Convierte el paquete de bytes con los datos de los sensores recibido en variables de tipo entero. Puesto que cada valor de los sensores MQ viene dividido en dos bytes, es necesario unirlos para obtener el valor real de la medida. Esto se realiza mediante una operación lógica que combina una OR junto a un desplazamiento. Por ejemplo, para el caso en el que los dos bytes correspondientes a un sensor sean 9 y 109 en decimal:

$$1^{er} \text{ byte: } 9 \rightarrow 0000\ 1001 \quad [8]$$

$$2º \text{ byte: } 196 \rightarrow 1100\ 0100 \quad [9]$$

Se realiza un desplazamiento de 8 bits a la izquierda del 1er byte, obteniendo:

$$1^{er} \text{ byte: } 0000\ 1001 << 8 = 0000\ 1001\ 0000\ 0000 \quad [10]$$

$$2º \text{ byte: } \qquad 0000\ 0000\ 1100\ 0100 \quad [11]$$

A continuación, se realiza la operación lógica OR:

$$0000\ 1001\ 0000\ 0000 \mid 0000\ 0000\ 1100\ 0100 = 0000\ 1001\ 1100\ 0100 \rightarrow 2500 \quad [12]$$

Esto da, por tanto, un valor de 2500 mV.

splitSamplingPeriod. Divide la variable de tipo entero que almacena el periodo de muestreo en tres bytes para que pueda ser enviado a Arduino. Cada byte se calcula a través de operaciones lógicas AND y desplazamientos de bits, de forma análoga al procedimiento anterior.

6.6.3. SerialCommunication

Este módulo se encarga de realizar la comunicación vía puerto serie con Arduino. Sus principales funciones son:

initSerialPort. Inicializa el puerto serie a 9600 baudios.

sendSamplingPeriod. Realiza la secuencia de operaciones necesaria para enviar correctamente el periodo de muestreo a Arduino.

readInputByte. Lee el último byte recibido desde Arduino.

establishFirstContact. Establece el contacto con Arduino para que éste comience el envío de los datos de los sensores.

readInputArray. Lee y guarda en un *array* el paquete de bytes recibido.

endCommunication. Indica a Arduino que ha recibido y guardado correctamente los últimos datos recibidos.

6.6.4. MainGUI

El módulo *MainGUI* se encarga de gestionar la interfaz, mostrando en pantalla todo lo que corresponda en el momento adecuado. Algunas de las funciones más importantes que lo componen son:

drawMenuData. Muestra en pantalla los elementos correspondientes al menú *Valores*.

drawMenuGraphs. Muestra en pantalla los elementos correspondientes al menú *Gráficos*.

getSamplingPeriod. Lee el periodo de muestreo introducido por el usuario en la ventana *WindowSampligPeriod.*

6.6.5. classGraph

El módulo *classGraph* contiene las funciones empleadas para crear las clases *Graph* y *Line* y declarar los objetos de esta clase.

La clase *Graph* es la base de los gráficos de los sensores. Cada objeto de este tipo constituye, pues, un gráfico distinto. Vienen definidos por la posición origen del gráfico, el ancho y alto, las unidades de la magnitud a mostrar, y los valores límite de los ejes X e Y. Esta clase posee dos funciones básicas, que son:

drawAxis. Dibuja las líneas de los ejes X e Y el gráfico en pantalla.

writeAxis. Muestra los valores de los ejes.

Los objetos de la clase *Line* se corresponden con las líneas de cada sensor que se dibujan en cada gráfico y que muestran la evolución temporal de las medidas del sensor. Esta clase es hija de la clase padre *Graph*. Hereda de ésta, pues, sus propiedades. Además, se añaden otras como la posición de la leyenda correspondiente a la línea, la tabla donde se almacena la información del sensor, el ID – es decir, la columna de la tabla donde se sitúan los valores del sensor en la tabla, y el estilo de la línea. Las funciones base de esta clase son las siguientes:

drawLegend. Muestra en pantalla el color indicativo de la línea junto a la leyenda.

drawPoints. Dibuja los puntos de la línea, que se corresponden con el valor obtenido del sensor en cada instante.

drawConnectors. Une los puntos anteriores mediante una línea recta para facilitar la visualización.

Este módulo contiene además otras funciones que hacen uso de los objetos declarados. Estas son:

drawLegend. Muestra la leyenda del gráfico de los sensores de gases MQ.

drawGraphs. Muestra los objetos de tipo *Graph.* Es decir, dibuja todos los gráficos: el de los sensores de gases, el de temperatura y el de humedad.

drawLines. Al igual que la función *drawGraphs*, muestra los objetos de tipo *Line* que correspondan.

6.6.6. classSensorCard

La función de este módulo es similar a la de *classGraph*. Contiene las clases ***MQSensorCard*** y ***DHTSensorCard***. Los objetos de este tipo se corresponden con los distintos rectángulos presentes en el menú *Valores* que muestran la información de cada sensor.

Los elementos que caracterizan a la clase *MQSensorCard* son el nombre del sensor, la fila que ocupa el sensor en las tablas de datos – es decir, el ID –, y la posición. Los objetos de la clase *DHTSensorCard*, por su parte, se caracterizan por un título – que puede ser "*Temperatura*" o "*Humedad*", según corresponda –, un booleano para indicar si muestra la temperatura o la humedad – TRUE: temperatura, FALSE: humedad –, y la posición.

Ambas clases poseen la función ***drawCard***, que dibuja el rectángulo del sensor con el valor que se está leyendo a tiempo real. Además, muestra este rectángulo en verde para indicar que la lectura es correcta, o en rojo si es incorrecta. Esta condición se determina si el valor leído se encuentra dentro de los límites de lectura del sensor – de 0 a 5.000 mV para los sensores MQ, de 0 a 5 ºC para la temperatura y 20 a 90% para la humedad – o no.

Por último, este módulo contiene la función ***drawSensorCards***, que muestra en pantalla todos los objetos de estas dos clases que se han declarado.

6.6.7. Styles

Durante el desarrollo del programa se dibujan sobre la interfaz muchos elementos gráficos como rectángulos o líneas. Para cada uno de ellos es necesario definir algunas propiedades como color de relleno o del borde, grosor, etc.

Este módulo contiene una serie de clases que definen el estilo de diferentes objetos del programa. De esta forma se consigue reutilizar el código para simplificar el del resto de módulos, pues estas clases evitan tener que declarar los diferentes aspectos de los elementos gráficos uno a uno cada vez que se quiera dibujar uno. Por ejemplo, a la hora de mostrar en pantalla un objeto de la clase *sensorCard*, la serie de comandos de la izquierda se sustituyen por el de la derecha:

```
fill(mainFillColor);
stroke(strokeColor);          }   SensorCardCorrectStyle.setMainStyle();
strokeWeight(strokeWeight);
```

Además, permite cambiar los estilos de una forma más simple, pues sólo es necesario modificar el valor con el que se inicia la variable correspondiente.

Para realizar todo ello, en este módulo se crea una clase padre – *GraphStyle* – que contiene algunos aspectos básicos a la hora de definir el aspecto de un objeto, como el color o el grosor. De esta clase padre se crean otras clases hijas específicas para unos objetos concretos, como por ejemplo la clase *cardStyle* para definir los estilos de las *sensorCard.*

Para poder definir los estilos desde otros módulos, cada clase contiene funciones que permiten declarar los aspectos. Por ejemplo, la clase *cardStyle* contiene las funciones *setMainStyle* y *setSecondaryStyle.*

6.6.8. gui

Este módulo es generado automáticamente por *G4P GUI Builder*. Esta herramienta se trata de una extensión para Processing que provee de un entorno visual para la creación y edición rápidas de las interfaces gráficas, haciendo uso de controles GUI y de la librería *G4P*.

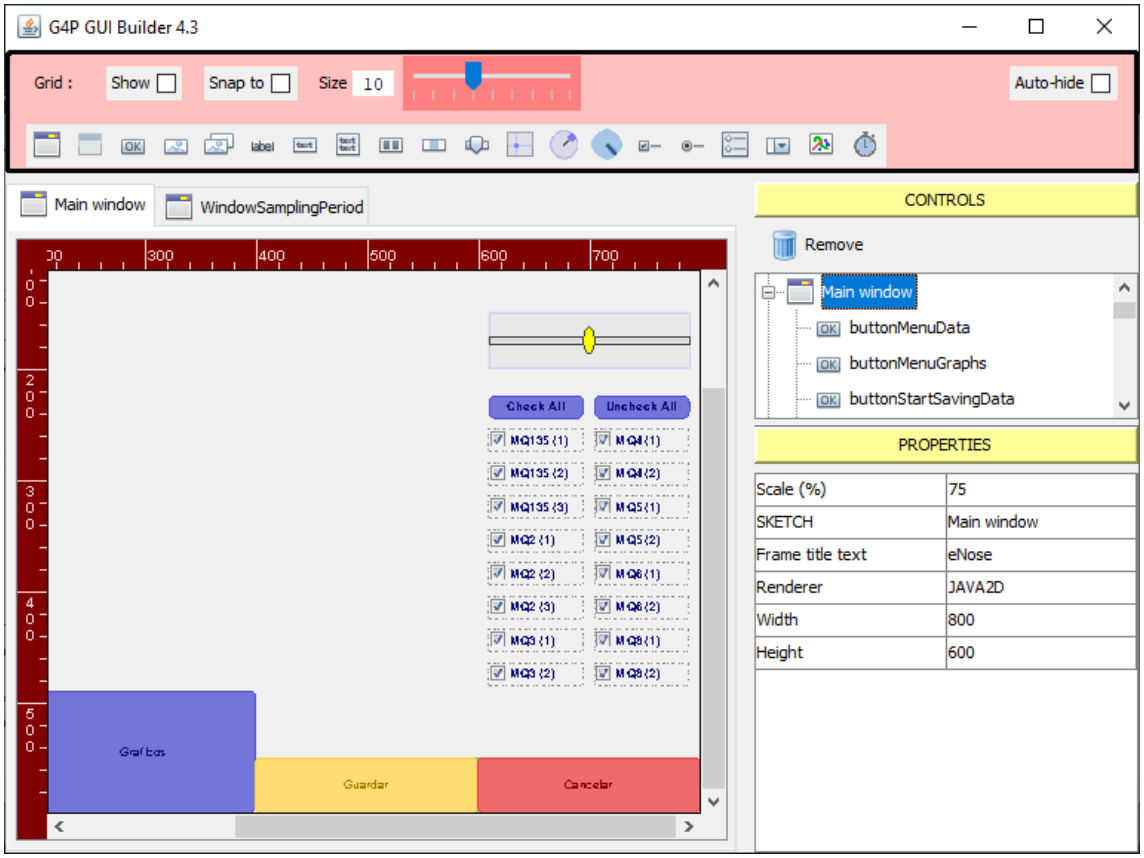


Figura 28 - G4P GUI Builder Tool

Este módulo incluye, por tanto, los objetos de las clases empleadas para desarrollar la interfaz gráfica como *checkboxes*, *sliders, buttons* o *labels*, entre otros.

La estructura de este módulo se puede dividir en dos partes. La primera contiene las funciones que son llamadas cuando el usuario interactúa con el objeto correspondiente. En su interior, se encuentra el código oportuno. Por ejemplo, la siguiente función lee y guarda el periodo de muestreo cuando el usuario presiona el botón *Aceptar* de la ventana *WindowSamplingPeriod*:

```
public void buttonAcceptPeriod_click(GButton source, GEvent event) {
  getSamplingPeriod();
}
```

En la segunda parte del módulo se declaran los diferentes objetos empleados, y se encuentra la función *createGUI*, que inicializa estos objetos.

Cada vez que se crea un nuevo objeto desde la herramienta *G4P GUI Builder*, se crea su función correspondiente en la primera parte del módulo, y se añaden las líneas de código necesarias para inicializarlo en la función *createGUI*.

6.6.9. Interfaz de usuario

La interfaz de usuario se divide en dos ventanas creadas con la herramienta *G4 GUI Builder*. La primera de ellas – *WindowSamplingPeriod* – se muestra en primer plano al iniciar el programa y permite al usuario introducir el periodo de muestreo deseado. Dado que la dinámica de los sensores empleados es lenta, esta ventana limita el periodo a un mínimo de 500 ms. Se asegura así además el correcto funcionamiento del sistema, pues se da tiempo suficiente para que se lleve a cabo la transmisión de los datos. Si el usuario introduce un periodo de muestreo inferior al indicado, muestra en pantalla un mensaje de error, indicando que el valor introducido no es válido.

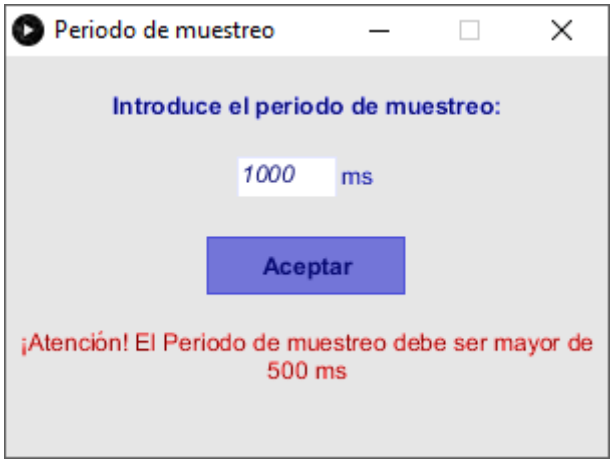


Figura 29 - Ventana de introducción del periodo de muestreo

La segunda ventana – *MainWindow* – incluye el resto de las funcionalidades del programa. Éstas se encuentran divididas en dos menús: el primero de ellos – *Valores* – muestra los valores de los sensores a través de los objetos *sensorCard*, mientras que el segundo – *Gráficos* –, muestra los gráficos generados a partir de la información de los sensores. La distribución de los elementos en esta ventana es la siguiente:

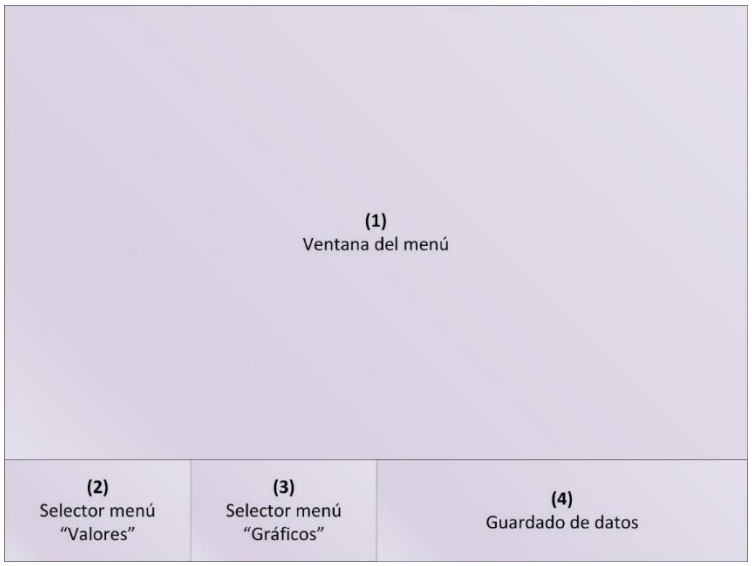


Figura 30 - Estructura de la interfaz el programa de visualización de datos

(1) – Ventana del menú. Esta sección de la interfaz se encuentra reservada para los elementos correspondientes al menú seleccionado mediante los botones (2) y (3).

(2) – **Selector menú "*Valores*"**. Selecciona el menú donde aparecen los valores de los datos recibidos desde Arduino.

(3) – **Selector menú "*Gráficos*".** Selecciona el menú donde aparecen los gráficos de los distintos sensores.

(4) – **Guardado de datos.** Submenú para comenzar o finalizar el guardado de los datos en el fichero de texto.

A continuación, se detalla el funcionamiento de los diferentes menús y de la funcionalidad del guardado de datos:

Guardado de datos

Las opciones de guardado de datos se encuentran siempre presentes en primer plano independientemente del menú activo. Inicialmente, consta de un único botón para iniciar el guardado de los datos. Desde el momento en que éste se pulsa, el programa comienza a guardar la información de los sensores, y el botón *Iniciar* deja paso a dos nuevos pulsadores: *Cancelar* y *Guardar*. El primero de ellos cancela el guardado de los datos. El segundo, finaliza el guardado y crea el fichero de formato *.csv* con la información. En cualquier caso, tras pulsar uno de los dos botones, se vuelve a mostrar el botón *Iniciar*.

Figura 31 - Submenú Exportar Datos (1)

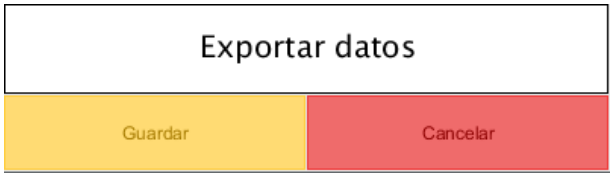


Figura 32 - Submenú Exportar Datos (2)

En caso de guardar los datos y para ayudar a la organización e identificación de los archivos generados, al fichero de texto generado se le asigna como nombre la fecha y hora actual siguiendo el formato *AAAA_MM_DD HHhMMm*, donde AAAA es el año actual, MM es el mes, DD es el día, HH la hora en formato 24h y MM los minutos. Por ejemplo: *2019_03_14 12h59m.*

Los ficheros creados un mismo día se sitúan dentro de una carpeta con la fecha actual, siguiendo un formato similar al del archivo de texto. Por ejemplo, *2019_03_14*. Por último, todas estas carpetas se crean dentro de la carpeta principal *data*, situada en el directorio raíz del programa.

Los archivos de texto se guardan en formato *.csv – Comma Separated Values –*. La primera fila se corresponde con la cabecera, que indica el sensor al que se refiere cada columna. El resto de las filas contienen la información medida en cada sensor en un periodo de muestreo. A continuación, se puede observar un extracto de un archivo de los archivos generados:

```
instant,MQ2 (1),MQ2 (2),MQ2 (3),MQ135 (3),Temperature 1,Temperature 2,Humidity 1,Humidity 2
0,      2385,   2205,   1975,   2745,     26,           25,           35,         33
1,      2415,   2230,   2005,   2765,     26,           25,           35,         33
2,      2390,   2210,   1980,   2745,     26,           25,           35,         33
```

Figura 33 - Ejemplo del fichero de datos .csv

Menú Valores

Este menú muestra los valores leídos por los sensores a tiempo real: la tensión en mV de los sensores de gases, y la temperatura en ºC y la humedad relativa en % de los sensores DHT11. Se muestran en distintos rectángulos, uno para cada sensor. Cada uno de ellos muestra la siguiente información:

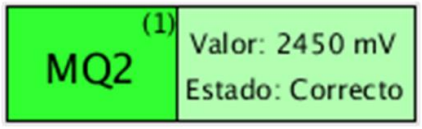


Figura 34 - SensorCard

- **Sensor**. Se indica tanto el tipo de sensor como el número de la fila de la matriz a la que pertenecen, de forma que se puede diferenciar cada sensor de los demás.
- **Valor**. El valor medido del sensor, junto con la magnitud de medida correspondiente.
- **Estado**. Indica si la lectura es correcta o incorrecta. En el caso de los sensores MQ, la lectura se determina como incorrecta si se encuentra fuera del rango comprendido entre 0 mV y 5000 mV. Para la temperatura, si es inferior a 0ºC o mayor a 50ºC; y para la humedad si está fuera del rango de 20 a 90%, que se corresponden con los rangos de medida del sensor DHT11.
- **Color**. Verde en caso de que el estado de la medida sea correcto, o rojo si es incorrecto. De esta forma, se puede observar a primera vista si hay algún problema en uno de los sensores.

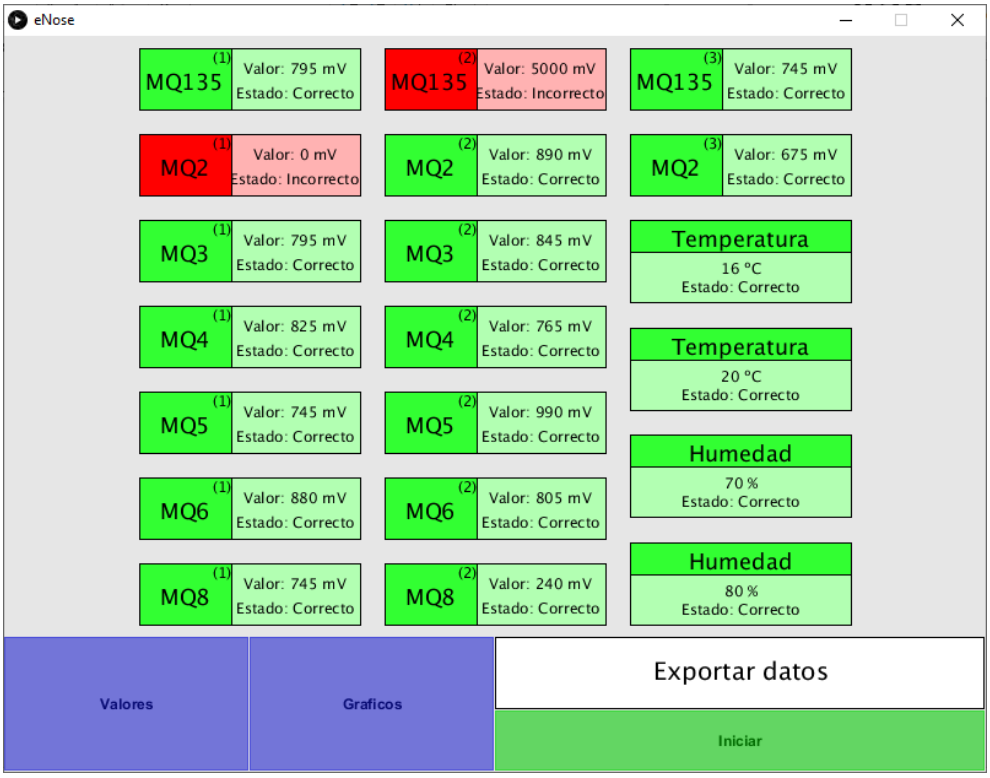


Figura 35 - Menú Valores

Menú Gráficos

Este menú muestra los gráficos que representan la evolución de los sensores a lo largo del tiempo. Incluye un total de tres gráficos: el grande y principal, para las medidas de los sensores de gases; y dos pequeños, para la temperatura y humedad. Además, en la parte derecha se incluye una leyenda en la que se muestra el color de la línea de cada sensor, junto a un checkbox que, en función de si se encuentra marcado o no, se mostrará la línea del sensor en el gráfico. El último elemento del menú es un slider donde se puede ajustar la escala del eje X de los gráficos.

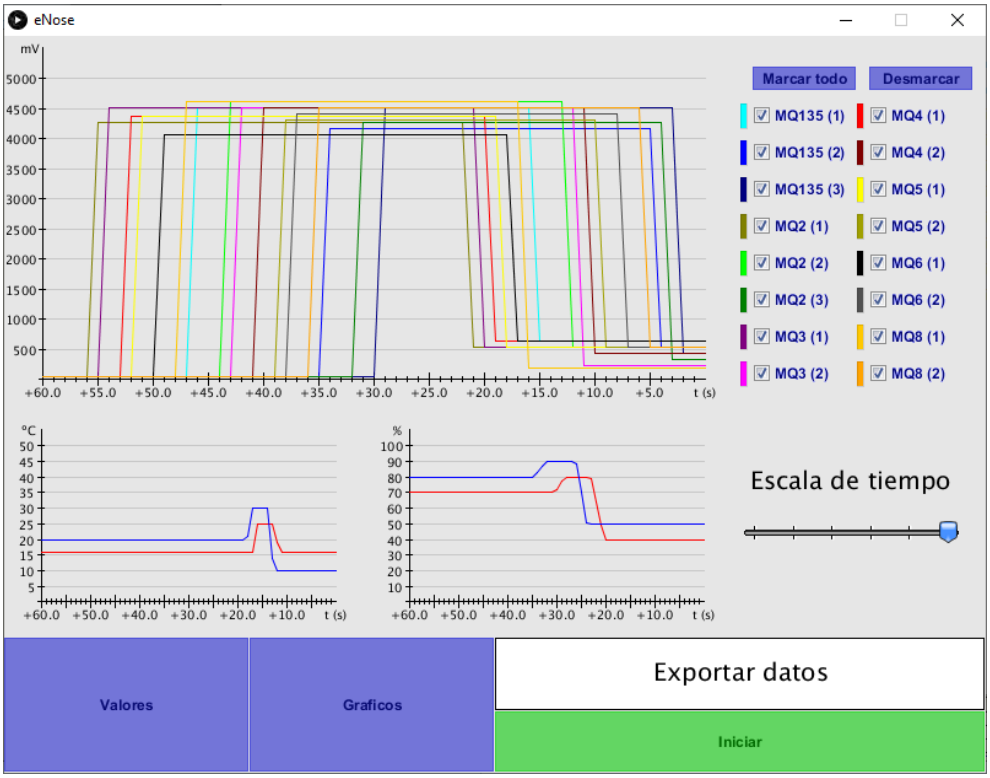


Figura 36 - Menú Gráficos

6.7. Simulación

Tras desarrollar el programa de visualización de datos con Processing se ha llevado a cabo una simulación de la matriz de sensores junto con Arduino, utilizando el software Proteus. El objetivo es asegurar que la funcionalidad de este programa es la esperada, asegurando que no existe ningún error en la visualización o guardado de los datos.

Los componentes que forman parte de esta simulación son los siguientes:

Simulino Mega. Para poder simular el microcontrolador Arduino Mega que se utiliza en el sistema real se ha utilizado el componente Simulino Mega de la librería Simulino. Esta librería ha sido desarrollada por los usuarios e incluye modelos de gran cantidad de placas de desarrollo Arduino, pudiéndose cargar un fichero binario con el código para realizar la simulación.

Las conexiones realizadas son similares a las que se hacen en el modelo real. Por un lado, se conectan las salidas de los sensores de gases a los puertos analógicos y de los sensores DHT11 a los pines digitales 2 y 3. Sin embargo, para simular la conexión con el puerto serie no se puede utilizar el puerto USB, sino que es necesario emplear los pines TX0 y RX0. No obstante, a efectos prácticos esto no supone ningún cambio en a la hora realizar la simulación, más allá de la conexión.

Además, no es necesario conectar la masa del componente a la masa utilizada en la simulación, pues el modelo ya tiene en cuenta este supuesto.

Por último, es necesario importar el programa de Arduino desarrollado previamente. Para ello, desde el IDE de Arduino se exportan los archivos binarios compilados y se cargan en el modelo Simulino a través de la opción *PROGRAM FILE* del menú Propiedades.

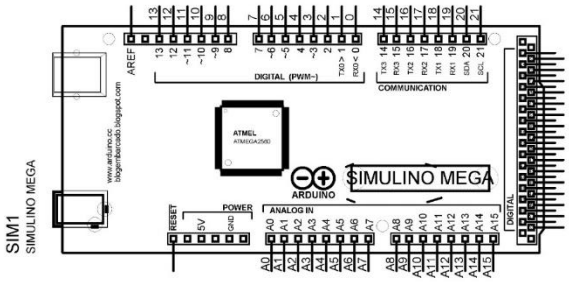


Figura 37 - Componente Simulino Mega 2560

COMPIM. Este componente viene incluido en la librería ACTIVE de Proteus. Se conecta a los pines RX0 y TX0 de Arduino y simula la conexión con el puerto serie, encargándose de enviar los datos correspondientes por el puerto serie. Para que funcione correctamente, es necesario introducir el puerto físico donde se conecta Arduino al ordenador, así como la velocidad del puerto serie en baudios desde el menú Propiedades del componente.

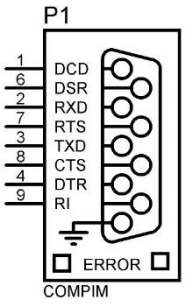


Figura 38 - Componente COMPIM

DHT11. Los sensores DHT11 se simulan con el componente del mismo nombre que viene incluido por defecto en el programa de simulación, en la librería TRXD. Estos componentes cuentan con tres terminales: VDD, que se conecta a la alimentación; GND, que se conecta a masa; y DATA, que constituye el terminal de salida del componente y se conecta al puerto digital correspondiente del componente Simulino Mega.

Además, cuenta con controles que permiten modificar los valores de temperatura y humedad a tiempo real mientras se ejecuta la simulación.

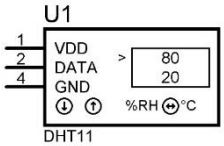


Figura 39 - Componente DHT11

MQ. En Proteus no existe ningún componente que permita la simulación de los sensores de gases MQ. Por ello, estos dispositivos se han recreado disponiendo dos resistencias en serie

junto con un potenciómetro de la librería ACTIVE – que permite modificar la posición del terminal de salida mientras se ejecuta la simulación – en configuración de divisor de tensión.

De esta forma, se puede modificar la tensión de salida, simulando la actuación de los sensores frente a la presencia de gases.

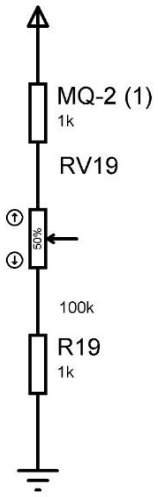


Figura 40 - Componente modelo de sensor MQ

A la hora de ejecutar esta simulación, no es posible llevarla a cabo sin un microcontrolador Arduino conectado al puerto USB del ordenador. Para evitar esta situación y poder realizarla sin necesidad de ningún dispositivo externo se emplea el programa *VSPE – Virtual Serial Ports Emulator –*, que como su propio nombre indica se encarga de emular un puerto virtual en el puerto físico indicado del ordenador.

Para ello, desde el programa se selecciona la opción *Create new device*. A continuación, en la ventana emergente que aparece, se selecciona *Connector* como tipo de dispositivo y se indica el puerto serie virtual que se desea crear – en este caso *COM3* –. Con ello, se crea el puerto serie virtual y ya se puede ejecutar la simulación normalmente sin necesidad de disponer de una tarjeta Arduino conectada al ordenador.

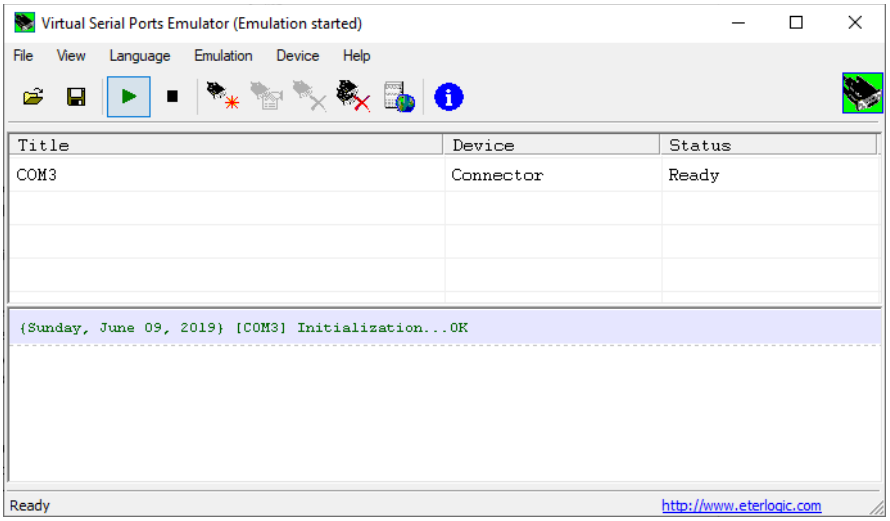


Figura 41 - Virtual Serial Ports Emulator

Una vez creado el esquemático necesario para llevar a cabo la simulación, y habiendo configurado todos los componentes y programas necesarios, se comienza con la realización del ensayo. En él se modifican los valores de los sensores uno a uno a través de la simulación. Con esto, se puede comprobar que la visualización y guardado de los datos es correcto.

En primer lugar, se inicia la simulación y se ejecuta el programa de visualización de datos. En este último, al iniciarse, se introduce un periodo de muestreo no demasiado elevado para que el tiempo del ensayo no sea excesivo. En este caso se escogen 1000 ms.

Una vez se ha introducido el periodo de muestreo y se muestra la ventana principal al usuario, se inicia el guardado de los datos haciendo uso del menú correspondiente. A continuación, se comienza a modificar las posiciones de los potenciómetros de los sensores de la simulación para que varíen las tensiones de salida. Se realiza de forma secuencial, en el siguiente orden preestablecido:

MQ-2 (1) → MQ-3 (1) → MQ-4 (1) → MQ-5 (1) → MQ-6 (1) → MQ-8 (1) → MQ-135 (1) → MQ-2 (2) → MQ-3 (2) → MQ-4 (2) → MQ-5 (2) → MQ-6 (2) → MQ-8 (2) → MQ-135 (2) → MQ-2 (3) → MQ-135 (3)

Mientras se cambian estos valores, se observa en el programa que los datos mostrados se corresponden con las modificaciones realizadas en los sensores de la simulación. Así, el gráfico de los sensores MQ obtenido es el siguiente:

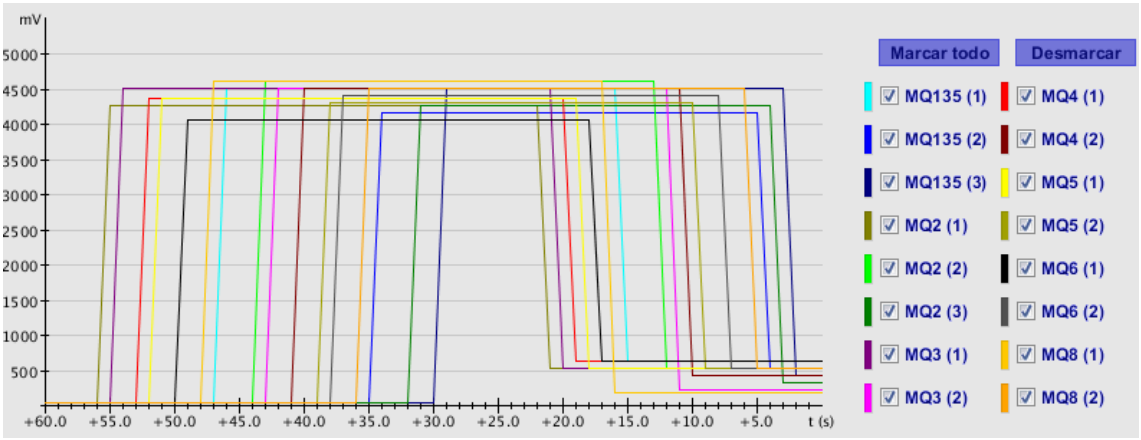


Figura 42 - Resultados de la simulación para los sensores MQ

A continuación, se realiza un procedimiento similar para los sensores de temperatura y humedad. El orden seguido es el siguiente:

Humedad (1) → Humedad (2) → Temperatura (1) → Temperatura (2)

Los gráficos obtenidos son los siguientes:

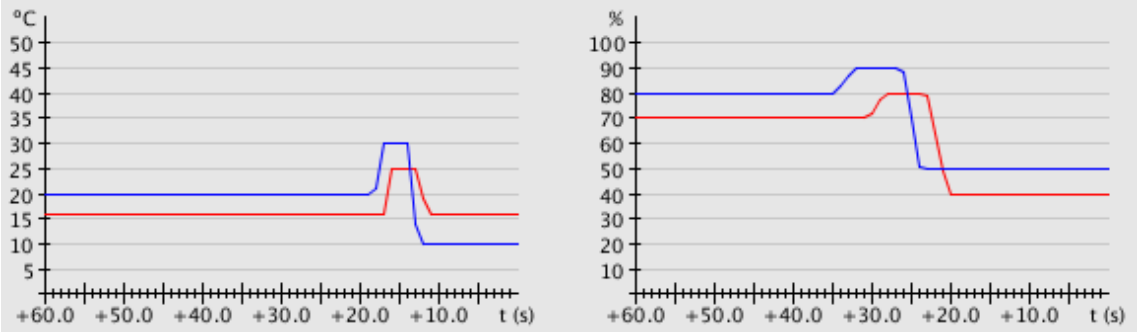


Figura 43 - Resultados de la simulación para los sensores DHT11

Por último, se comprueba que los datos guardados en el fichero de texto son correctos. Para ello se finaliza el guardado y se analiza la información contenida en el documento. Este análisis se puede llevar a cabo abriendo el fichero con un editor de texto, y observando directamente los valores de cada columna. Sin embargo, para realizar de una forma más rápida, visual y segura, se representan los datos en un gráfico y se observa que su evolución es la adecuada.

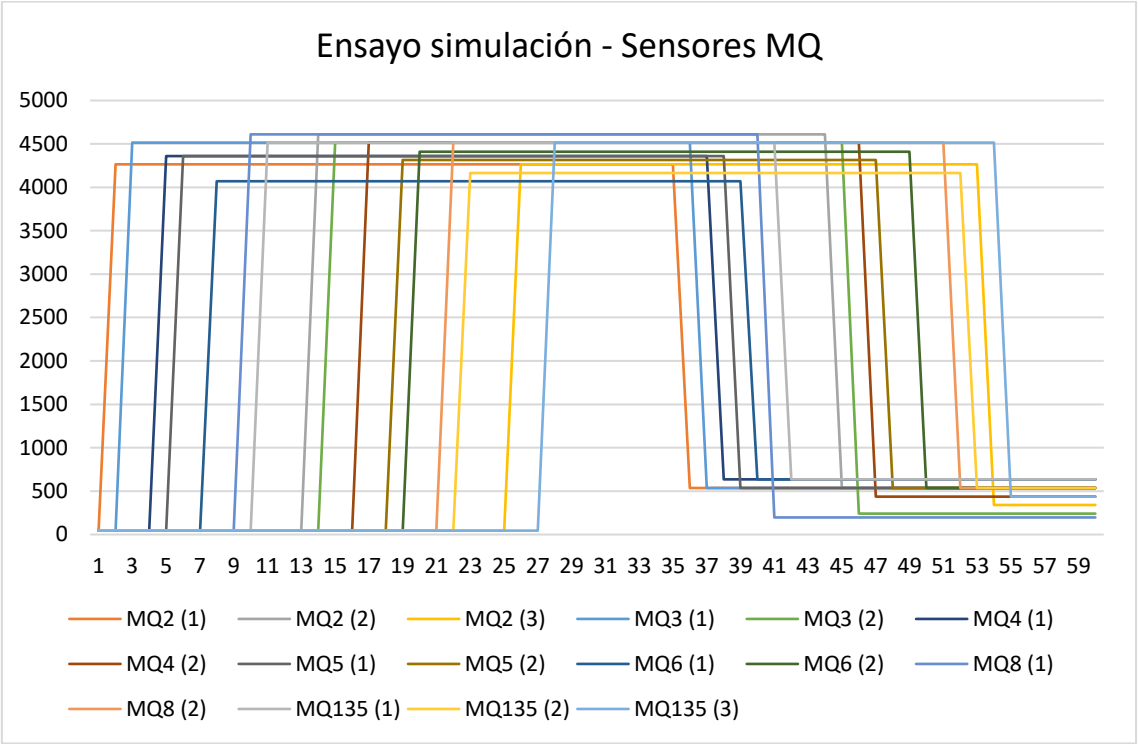


Figura 44 - Gráfico resulados de la simulación para los sensores MQ

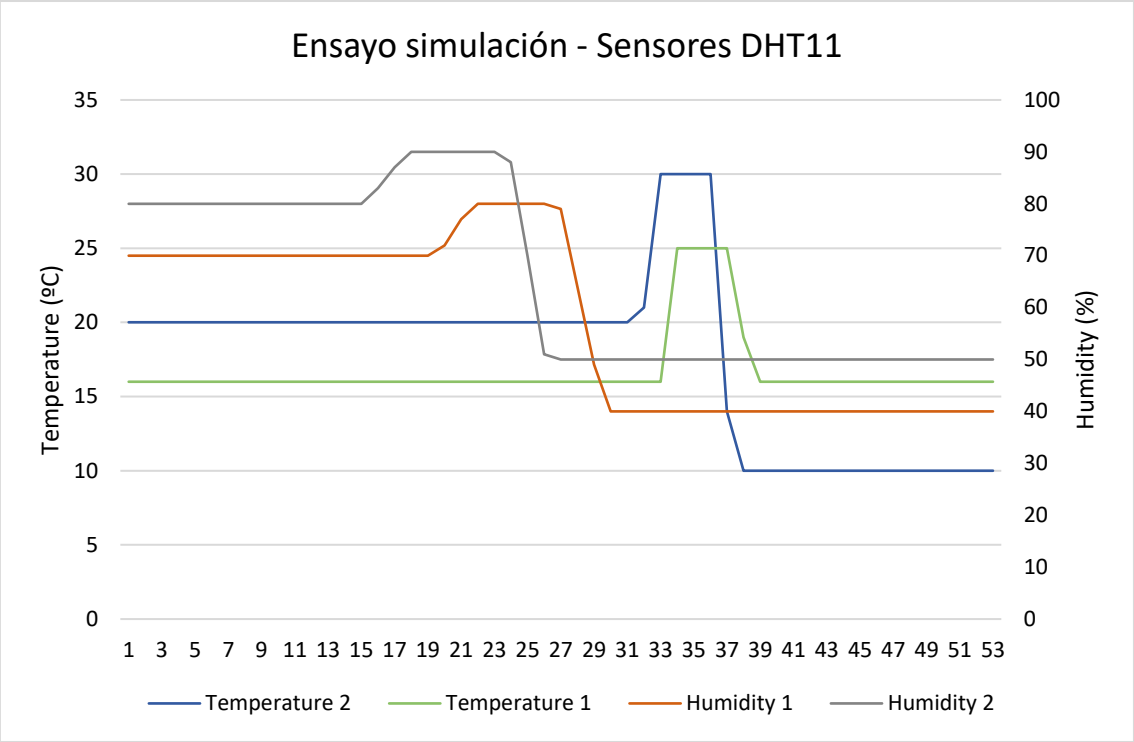


Figura 45 - Gráfico resultados de la simulación para los sensores DHT11

Como se puede observar, en todos los casos la evolución de los datos de los sensores ha sido la esperada. Los valores han cambiado en el orden y momentos adecuados tanto en el programa de visualización de datos a tiempo real, como en el gráfico de Excel utilizando el fichero de texto que ha generado previamente el programa. Por tanto, se puede determinar que se ha cumplido el objetivo del ensayo satisfactoriamente ya que se ha comprobado que el programa desarrollado muestra y guarda los datos correctamente.

6.8. Prototipo y ensayo final

Para finalizar el proyecto, se implementa un prototipo de la matriz de sensores y se llevan a cabo dos ensayos para comprobar el correcto funcionamiento y la viabilidad del sistema de sensado al completo, de forma que se pueda ser montado posteriormente en una nariz electrónica funcional.

El prototipo se crea utilizando tres placas de desarrollo o protoboards. En dos de ellas se montan los sensores de gases MQ y, en la tercera, los dos sensores de temperatura y humedad. Para realizar las conexiones necesarias se utilizan cables de tipo jumper. Para simular la cámara de medida de la nariz electrónica, las protoboards se introducen en un recipiente de plástico con una tapadera, el cual presenta un pequeño agujero por el que se pasan los cables de transmisión de datos y de alimentación. Es en este recipiente donde, durante los ensayos, se introducen los gases con los que se va a probar el sistema. A continuación, se puede observar una imagen del prototipo:

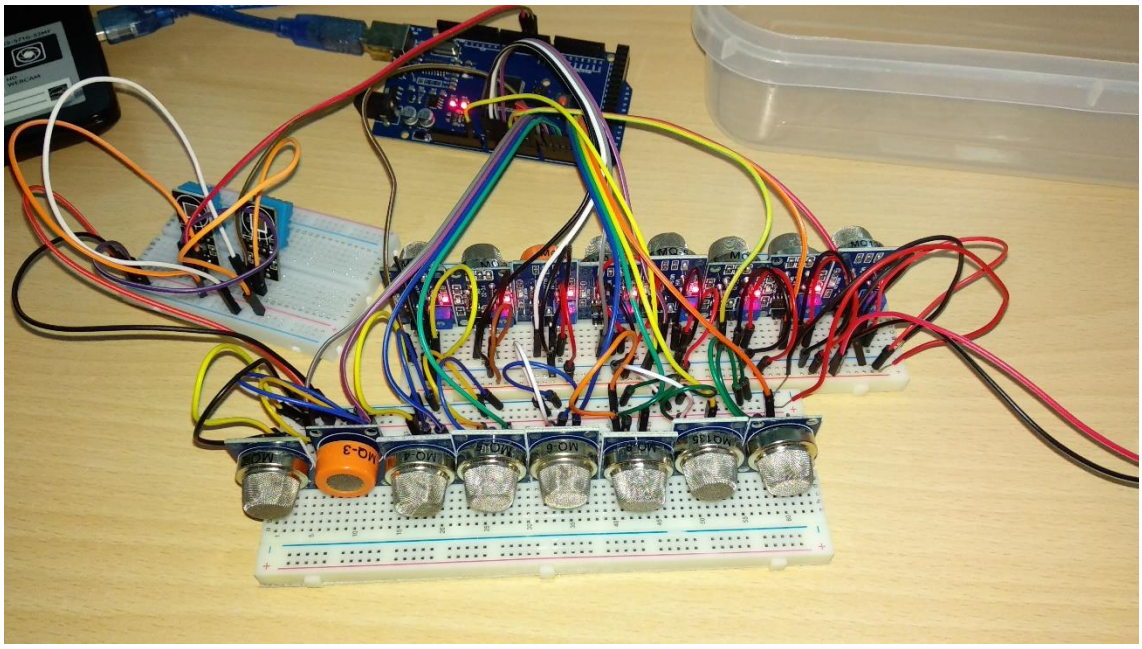


Figura 46 - Prototipo implementado en varias placas de pruebas

6.8.1. Ensayo con gas butano

En el primer ensayo se introduce gas butano dentro de la cámara de medida utilizando un mechero de tipo *Clipper*. La secuencia de pasos llevados a cabo es la siguiente:

1. Se dejan calentar los sensores durante 12 horas al aire libre para que se depuren correctamente, de forma que cualquier resto de gas que pueda existir en su superficie se volatilice correctamente y no influya en el resultado del ensayo.
2. Tras depurar los sensores, se da comienzo al ensayo. Para ello, se inicia el programa de visualización de datos, se establece un periodo de muestreo de 1000 ms y se comienza el guardado de los datos en el fichero de texto.
3. Se dejan reposar los sensores durante 1 minuto para asegurar que las medidas de los sensores son estables antes de introducir el gas en la cámara de medida.
4. Se introduce el gas en la cámara de medida. Este proceso requiere 15 segundos, aproximadamente.
5. Se cierra la cámara de medida, y se deja evolucionar el sistema durante 1 minuto.
6. Cuando la medida de los sensores se ha estabilizado tras este tiempo, se abre la cámara de medida permitiendo la renovación del aire que hay en su interior.
7. Cuando los datos de los sensores se vuelven a estabilizar pasados 2 minutos, se finaliza el guardado de los datos en el fichero y se da por terminado este ensayo.

Una vez realizado el ensayo, se carga el fichero *.csv* en una hoja de cálculo Excel para analizar la respuesta ofrecida por los sensores y extraer las conclusiones pertinentes. Por un lado, se dibuja un gráfico con los datos de los diferentes sensores. Por otro, se obtiene la diferencia de tensión generada en cada sensor durante el ensayo. Esta diferencia no se calcula tomando los valores máximo y mínimo de cada sensor debido a la gran sobre oscilación que presentan, sino que se calcula entre los valores de los sensores estabilizados antes de comenzar el ensayo, y una vez están siendo sometidos a la presencia del gas – instantes 59 y 134, respectivamente –. Los resultados obtenidos son los siguientes:

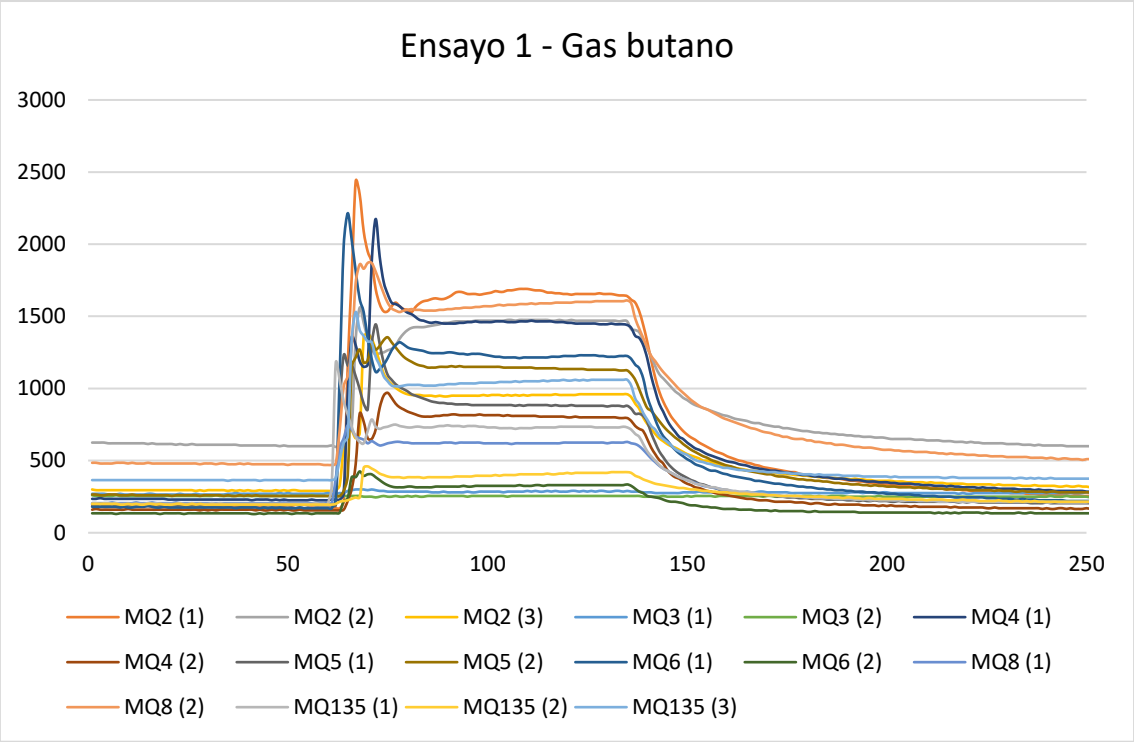


Figura 47 - Gráfico del ensayo con gas butano

Sensor	Medida (T59)	Media (T134)	Diferencia
MQ2 (1)	170	1645	1475
MQ4 (1)	225	1445	1220
MQ8 (2)	470	1605	1135
MQ6 (1)	170	1225	1055
MQ5 (2)	255	1125	870
MQ2 (2)	600	1465	865
MQ135 (3)	365	1060	695
MQ5 (1)	190	880	690
MQ2 (3)	290	960	670
MQ4 (2)	155	795	640
MQ135 (1)	200	730	530
MQ8 (1)	205	625	420
MQ135 (2)	195	420	225
MQ6 (2)	135	330	195
MQ3 (1)	275	290	15
MQ3 (2)	255	255	0

Figura 48 - Tabla del ensayo con gas butano

En los resultados obtenidos se puede observar cómo al aplicar el gas el sistema evoluciona hasta estabilizarse en unos valores distintos a los iniciales. Esto ocurre con todos los sensores salvo con los MQ-3, ya que todos ellos son sensibles en mayor o menor medida al gas butano. Los sensores MQ-3, por su parte, sólo son sensibles al alcohol, por lo que su respuesta no se ve afectada ante la presencia de este gas.

Además, se observan las siguientes anomalías en las respuestas del sistema:

- Aunque los estados transitorios de los sensores de un mismo tipo presentan una dinámica muy parecida entre sí, las sensibilidades de cada sensor son muy dispares. Esto se observa en la Figura *48* - Tabla del ensayo con gas butano, donde la diferencia de tensión en los sensores MQ-6, por ejemplo, es de 1055 mV y 195 mV.
- Inicialmente, y antes de introducir el gas en la cámara de medida, los sensores de un mismo tipo no ofrecen el mismo nivel de tensión a su salida. Por ejemplo, los sensores MQ-2 presentan 170, 600 y 290 mV, valores muy dispares entre sí.

Tras analizar la respuesta y medir las tensiones de alimentación de cada sensor, se llega a la conclusión de que estas anomalías son causadas por dos motivos:

1. Existen diferencias en la tensión de alimentación de cada sensor, debido a caídas de tensión producidas por su montaje en una placa de prototipos debido a malos contactos producidos entre las placas de prototipos y los cables. No obstante, este problema desaparece una vez montados los sensores en la placa de circuito impreso.
2. No se han calibrado los sensores. Esta variable afecta solamente a la segunda anomalía mencionada. Al no estar calibrados, sensores de un mismo tipo pueden proporcionar una salida distinta ante una misma concentración de gases.

Pese a ello, cabe indicar que estas anomalías no son un problema real en el sistema diseñado. A la hora de aplicarlo en una nariz electrónica el hecho de tener sensores del mismo tipo con sensibilidades distintas supone una ventaja, ya que aumenta la cantidad de información obtenida de la matriz que en el caso de que se tuvieran dos sensores exactamente iguales. Además, tampoco es necesario calibrar los sensores ya que la identificación de la sustancia no se realiza cuantificando los gases, sino a partir de la evolución en el tiempo de los sensores, que es lo que conforma la huella electrónica.

6.8.2. Ensayo con alcohol

En el segundo ensayo se emplean dos algodones bañados en alcohol para analizar la respuesta del sistema. La secuencia llevada a cabo durante el experimento es la siguiente:

1. Se depuran los sensores durante 12 horas al aire libre, de igual forma que en el ensayo anterior.
2. A continuación, se ejecuta el programa de visualización de datos, se establece un periodo de muestreo de 1000 ms y se inicia el guardado de los datos en el fichero de texto.
3. Se dejan reposar los sensores durante 1 minuto para asegurar que las medidas son estables.
4. Se introducen dos algodones bañados en alcohol en la cámara de medida, uno en cada esquina para intentar que se reparta lo más uniformemente posible. Este proceso dura, aproximadamente, 15 segundos.
5. A continuación, se cierra la cámara de medida y se deja evolucionar el sistema durante 5 minutos. En el transcurso de este tiempo, el alcohol de los algodones se volatiliza y pasa al aire contenido en la cámara de medida.
6. Tras este periodo, y una vez las medidas se han estabilizado, se abre la cámara de medida y se extraen los algodones. Se deja evolucionar el sistema durante otros 5 minutos, mientras el aire se renueva.
7. Finalmente, las medidas se estabilizan de nuevo y se finaliza el guardado de los datos en el fichero de texto, concluyendo así el ensayo.

Los resultados obtenidos en este ensayo son los siguientes:

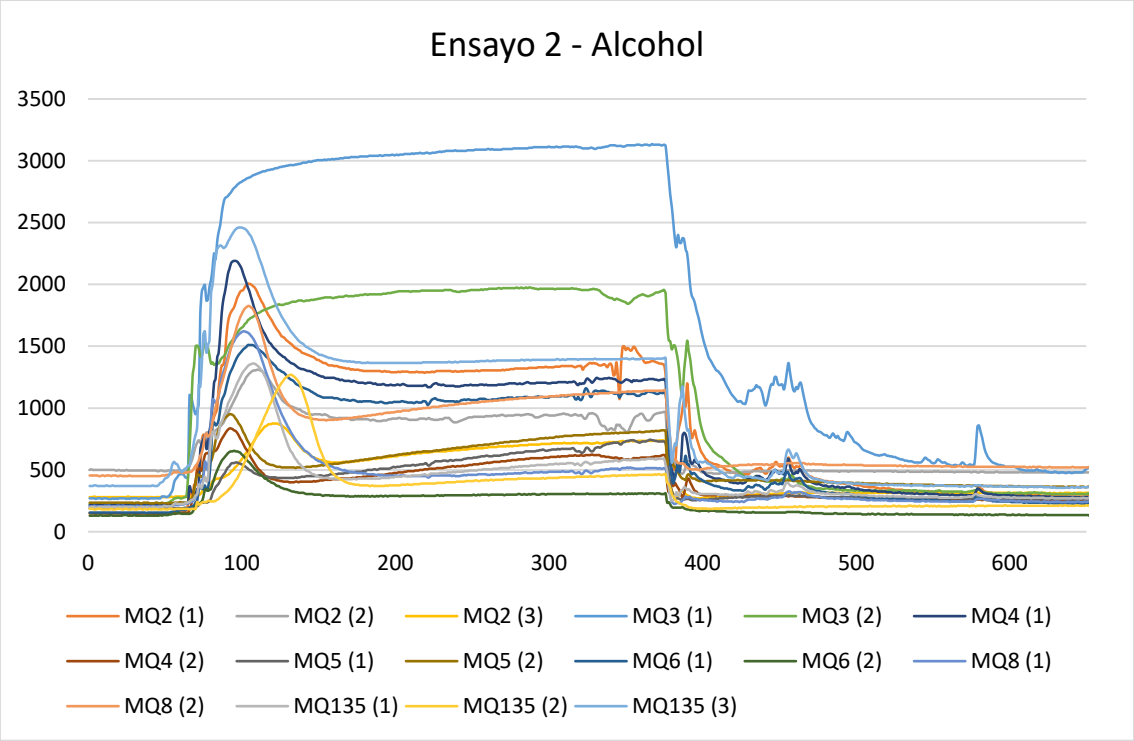


Figura 49 - Gráfico del ensayo con alcohol

Sensor	Medida (T30)	Medida (T370)	Diferencia
MQ3 (1)	270	3130	2860
MQ3 (2)	225	1930	1705
MQ2 (1)	145	1375	1230
MQ135 (3)	370	1395	1025
MQ4 (1)	215	1230	1015
MQ6 (1)	155	1120	965
MQ8 (2)	450	1140	690
MQ5 (2)	235	815	580
MQ5 (1)	190	735	545
MQ4 (2)	150	610	460
MQ2 (2)	495	950	455
MQ2 (3)	280	735	455
MQ135 (1)	200	585	385
MQ8 (1)	205	515	310
MQ135 (2)	185	465	280
MQ6 (2)	130	305	175

Figura 50 - Tabla del ensayo con alcohol

En este ensayo los sensores que más sensibles se muestran frente a la presencia de alcohol en el aire son los MQ-3, tal como era de esperar pues estos sensores están destinados principalmente a la detección de alcohol.

En cuanto al resto de sensores, sus respuestas se ven modificadas por la presencia de alcohol ya que todos ellos son sensibles en mayor o menor medida a esta sustancia, tal como se indica en sus datasheets.

Se observan además las mismas anomalías que en el ensayo realizado anteriormente con gas butano, las cuales son provocadas por los mismos causantes: las pérdidas de tensión en las placas de prototipos y la no calibración de los sensores. De la misma forma, no se consideran un problema para el sistema de medida una vez sea implementado en las placas de circuito impreso.

Por tanto y a la vista de los resultados obtenidos, se puede dar por válido el sistema diseñado ya que las anomalías encontradas no afectan al sistema final y no se han encontrado otros problemas con ninguno de los subsistemas que componen el sistema, o con el propio sistema al completo.

7. Conclusión y propuestas de mejora

A lo largo del desarrollo del proyecto se ha podido aplicar una gran cantidad de los conocimientos aprendidos durante el grado en Grado en Electrónica Industrial y Automática, entre los que destacan los de las asignaturas de Electrónica Digital e Instrumentación Electrónica para el diseño de la matriz de sensores, e Informática Industrial, para desarrollar los programas de Arduino y de Processing.

Como conclusión, se puede afirmar que el proyecto llevado a cabo ha logrado cumplir todas las condiciones impuestas previamente al comienzo de su desarrollo, además de implementar nuevas características adicionales a las requeridas. Además, se han llevado a cabo una serie de experiencias con las que se ha podido validar que el sistema de sensado es completamente funcional. Todo ello ha dado lugar a un sistema viable y apto para ser implementado posteriormente en una nariz electrónica funcional.

No obstante, de cara al futuro este proyecto puede ser ampliado y mejorado para lograr un sistema de sensado más versátil y fiable. Algunas de las mejoras propuestas son las siguientes:

- Implementar en la matriz los sensores de la serie MQ con alimentación distinta a 5 Vdc para lograr identificar una mayor cantidad de gases y sustancias.
- Alimentar los sensores redundantes con tensiones diferentes entre sí, por ejemplo 5 Vdc y 3,5 Vdc. De esta forma, se logra disponer de sensores del mismo tipo, pero con sensibilidades distintas, lo que aumenta la cantidad de información proporcionada por la matriz de sensores de cara a identificar una sustancia.
- Añadir un algoritmo de identificación de sustancias al propio programa de Processing, de forma que no sea necesario exportar los datos para analizarlos por un software distinto.
- Ampliar las funcionalidades del programa, añadiendo la posibilidad de elegir el nombre y formato del fichero donde se exportan los datos, de escoger los sensores que se desean utilizar durante el ensayo, de importar un fichero con datos y mostrarlos en el propio programa, etc.

8. Bibliografía

Díaz, M. J. (Octubre de 2015). Normas Básicas y Recomendaciones en el Diseño de PCBs.

Eun Gyeong Kim, Seok Lee, Jae Hun Kim, Chulki Kim, Young Tae Byun, Hyung Seok Kim, & Taikjin Lee. (2012). Pattern Recognition for Selective Odor Detection with Gas Sensor Arrays. *Sensors*, 1-12.

Moreno, I., Caballero, R., Galán, R., Matía, F., & Jiménez, A. (2009). La Nariz Electrónica: Estado del Arte. *Revista Iberoamericana de Automática e Informática Industrial*, 76-91.

Perellón, O. B. (2004). *Diseño y realización de una nariz electrónica para la maduración de quesos.*

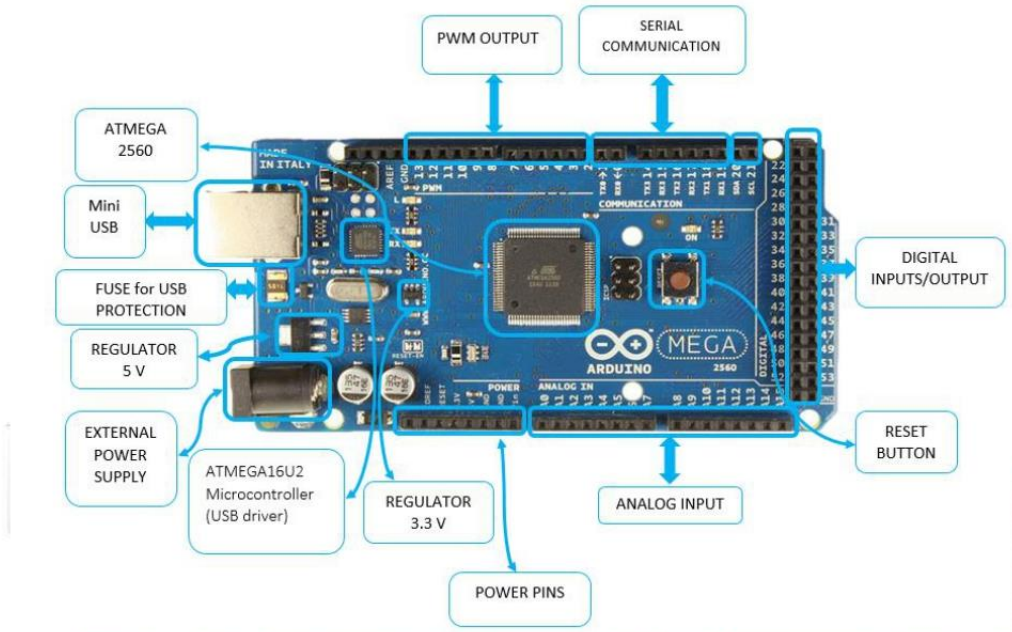
Prats, R. B. (2017). *Fabricación y puesta a punto de una nariz electrónica húmedad para la detección de gases y vapores.* Valencia: UPV.

Santos, R. S. (s.f.). *Ultimate Guide for Arduino Sensors/Modules.* Random Nerd Tutorials.

9. Anejos

9.1. Datasheets

ARDUINO MEGA



INTRODUCTION

The Arduino MEGA 2560 is designed for projects that require more I/O llines , more sketch memory and more RAM. With 54 digital I/O pins, 16 analog inputs so it is suitable for the complex projects like 3D printers and robotics projects.

ARDUINO MEGA PHYSICAL COMPONENTS

ATMEGA 2560 Microcontroller

Features

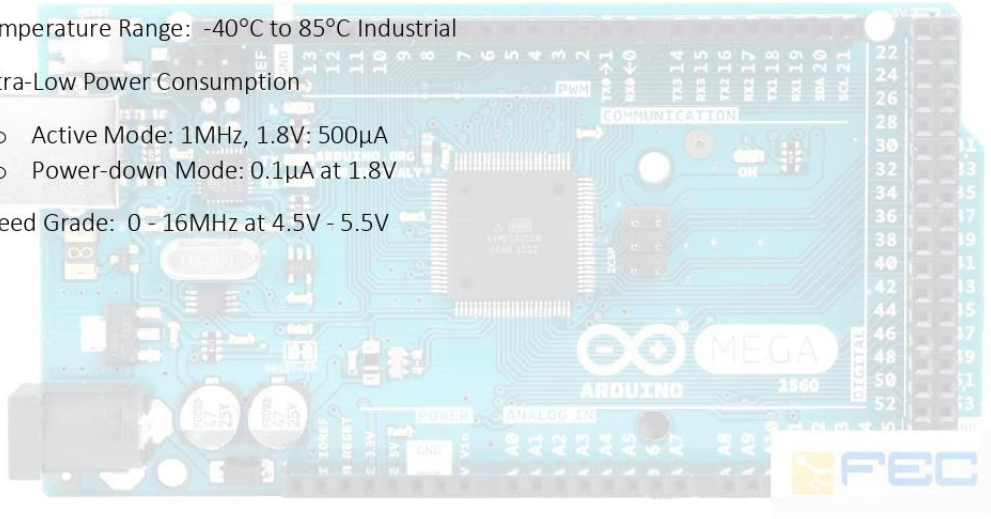
- 8-Bit Microcontroller
- High Performance, Low Power
- Advanced RISC Architecture
  - 135 Powerful Instructions
  - Most Single Clock Cycle Execution

- 32 × 8 General Purpose Working Registers
- Fully Static Operation
- Up to 16 MIPS Throughput at 16MHz
- On-Chip 2-cycle Multiplier

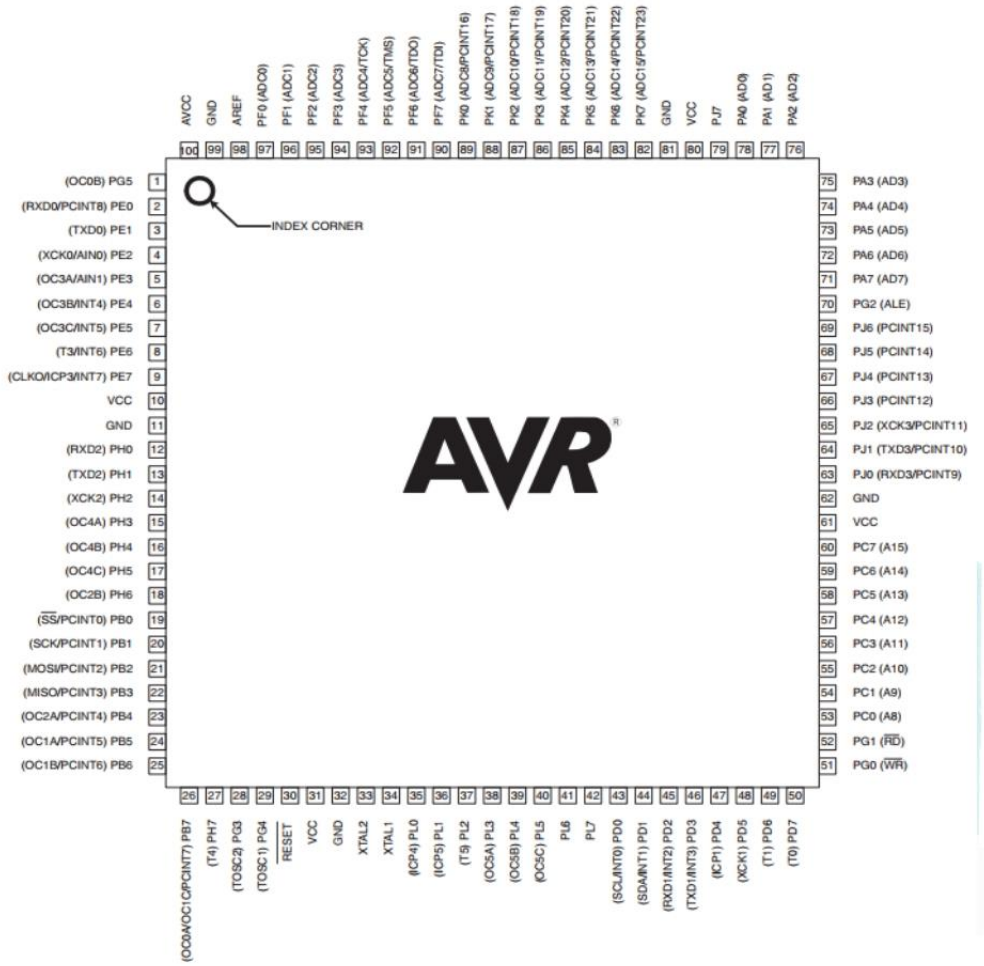
- High Endurance Non-volatile Memory Segments
  - 64K/128K/256KBytes of In-System Self-Programmable Flash
  - 4Kbytes EEPROM
  - 8Kbytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C
  - Optional Boot Code Section with Independent Lock Bits
- In-System Programming by On-chip Boot Program
- True Read-While-Write Operation
  - Programming Lock for Software Security
- Endurance: Up to 64Kbytes Optional External Memory Space
- Atmel QTouch library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix acquisition
  - Up to 64 sense channels

• Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - Four 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four 8-bit PWM Channels
  - Six/Twelve PWM Channels with Programmable Resolution from 2 to 16 Bits
  - 8/16-channel, 10-bit ADC
  - Two/Four Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte Oriented 2-wire Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change

- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 54/86 Programmable I/O Lines
  - 100-lead TQFP, 100-ball CBGA
  - RoHS/Fully Green
- Temperature Range: -40°C to 85°C Industrial
- Ultra-Low Power Consumption
  - Active Mode: 1MHz, 1.8V: 500µA
  - Power-down Mode: 0.1µA at 1.8V
- Speed Grade: 0 - 16MHz at 4.5V - 5.5V



- Pin configuration



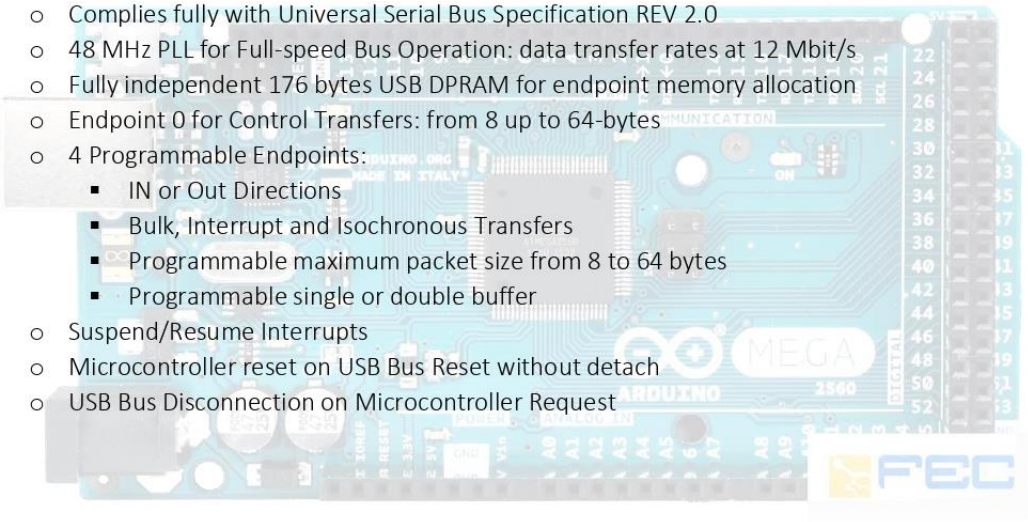
ATMEGA16U2 Microcontroller (USB driver)

Features

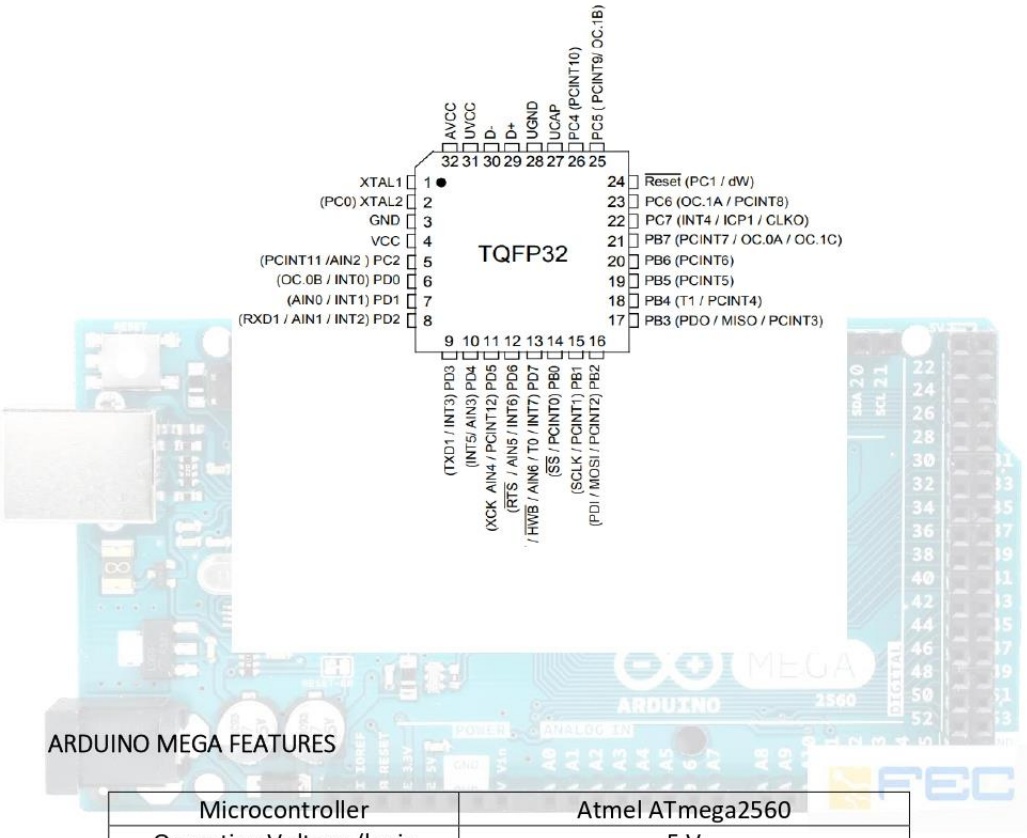
- High Performance, Low Power AVR
- Advanced RISC Architecture
  - 125 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation

- Non-volatile Program and Data Memories
  - 8K/16K/32K Bytes of In-System Self-Programmable Flash
  - 512/512/1024 EEPROM
  - 512/512/1024 Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/ 100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C
  - Optional Boot Code Section with Independent Lock Bits
  - In-System Programming by on-chip Boot Program hardware-activated after reset
  - True Read-While-Write Operation
  - Programming Lock for Software Security

- USB 2.0 Full-speed Device Module with Interrupt on Transfer Completion
  - Complies fully with Universal Serial Bus Specification REV 2.0
  - 48 MHz PLL for Full-speed Bus Operation: data transfer rates at 12 Mbit/s
  - Fully independent 176 bytes USB DPRAM for endpoint memory allocation
  - Endpoint 0 for Control Transfers: from 8 up to 64-bytes
  - 4 Programmable Endpoints:
    - IN or Out Directions
    - Bulk, Interrupt and Isochronous Transfers
    - Programmable maximum packet size from 8 to 64 bytes
    - Programmable single or double buffer
  - Suspend/Resume Interrupts
  - Microcontroller reset on USB Bus Reset without detach
  - USB Bus Disconnection on Microcontroller Request



- Pin configuration



ARDUINO MEGA FEATURES

Microcontroller	Atmel ATmega2560
Operating Voltage (logic level)	5 V
Input Voltage (recommended)	7-12 V
Input Voltage (limits)	6-20 V
Digital I/O Pins	54 (of which 14 provide PWM)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
Flash Memory	256Kbyte of which 8 KB used by boot loader
SRAM	8 Kbytes
EEPROM	4 Kbytes

TRACO POWER

AC/DC Enclosed Power Supply

TXL 015 Series, 15 Watt

- **Compact metal case with screw terminal block**
- **Universal input 85 - 264 Vac**
- **EMI/EMC compliance with EN 61000-6-3 and EN 61000-6-1**
- **Compliance to EN 61000-3-2**
- **Short circuit and overvoltage protection**
- **International safety approvals**
- **3 year product warranty**

ErP ready CE CB Scheme cULus UL 60950-1

The TRACO POWER TXL series is a family of encased power supplies designed for a wide range of cost critical applications. With a low profile metal case and screw terminal block connection, they are easy to install in any equipment. These power supplies have universal input and comply with European EMC standards and the Low Voltage Directive (LVD).

Models

Order Code	Power max.	Output voltage nom.	Output current max.	Efficiency typ.
TXL 015-3.3S	10.0 W	3.3 VDC	3.0 A	74 %
TXL 015-05S	15.0 W	5 VDC	3.0 A	79 %
TXL 015-12S	15.6 W	12 VDC	1.3 A	84 %
TXL 015-15S	15.0 W	15 VDC	1.0 A	85 %
TXL 015-24S	15.1 W	24 VDC	0.63 A	86 %
TXL 015-48S	15.4 W	48 VDC	0.32 A	87 %

TRACO POWER

TXL 015 Series, 15 Watt

Input Specifications			
Input voltage range	– nominal – AC range (universal input) – DC range		100 – 240 VAC 85 – 264 VAC 120 – 375 VDC
Input frequency			47 – 63 Hz
Input current at full load	– at 100 VAC		0.5 A max.
Input inrush current	– at 115 VAC / 230 VAC		30 A max. / 50 A max.
Zero load power consumption			0.3 W max. (green mode design)
Recommended circuit breaker (characteristic C or slow bow fuse)			5 A
Output Specifications			
Output voltage adjustment range			±10 %
Regulation	– Input variation – Load variation (20 - 100%)		1.0 % max. 1.0 % max.
Minimum load			not required
Temperature coefficient			0.02 %/K
Start-up time			1 s max.
Rise time			50 ms max.
Hold-up time	– at 230 VAC		60 ms min.
Ripple and noise (20Mhz Bandwidth)		3.3 & 5 Vout models: 12 Vout model: 15 Vout model: 24 & 48 Vout models:	80 mVp-p typ. 120 mVp-p typ. 150 mVp-p typ. 200 mVp-p typ.
	– measured with external capacitors:		0.1 µF and 47 µF parallel capacitor
Overload protection by current limitation			over 105 % of Iout max.
Short circuit protection			hiccup mode (automatic recovery)
Overvoltage protection (Limiting voltage latch)			115 – 140 % of nominal Vout
Capacitive load			www.tracopower.com/products/txl-capload.pdf
General Specifications			
Temperature ranges	– Operating – Storage		–20°C to +70°C (with derating) –40°C to +85°C
Output power derating	– Temperature		2.5 %/K above +50°C
Cooling			natural convection (20 lfm), no internal fan
Humidity (non condensing)			20 – 90 % rel. H max.
Altitude during operation			2000 m
Isolation voltage (60 sec.)	– Input / Output – Input / PE – Output / PE		3000 VAC 1500 VAC 500 VAC
Isolation resistance (at 500 VDC)			100 MOhm min.
Leakage current (at 264 VAC/60Hz)			0.5 mA max.
Switching frequency			105 kHz typ. (pulse width modulation)
Reliability (calculated MTBF)			> 630‘000 h

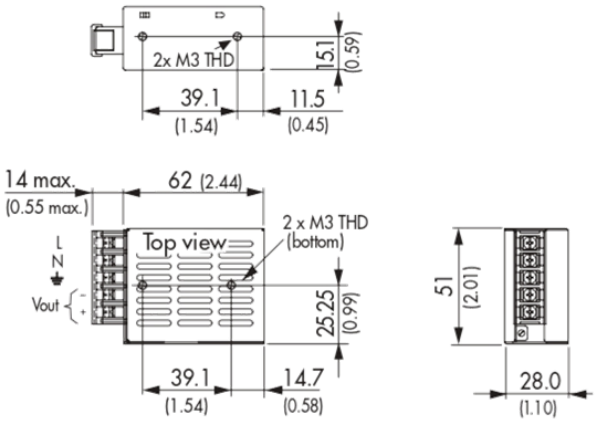
TRACO POWER

TXL 015 Series, 15 Watt

General Specifications (continued)

Electromagnetic compatibility (EMC), emissions		
	– Conducted input RI suppression	EN 55022 class B, FCC Part 15 level B
	– Harmonic current emissions	IEC/EN 61000-3-2, class A
	– Voltage flicker	IEC/EN 61000-3-3
Electromagnets compatibility (EMC), immunity		according EN 55024
	– Electrostatic discharge ESD	IEC/EN 61000-4-2, 4 kV / 8 kV, perf. criteria A
	– RF field immunity	IEC/EN 61000-4-3, 3 V/m, perf. criteria A
	– Electrical fast transients/burst immunity	IEC/EN 61000-4-4, ±2 kV, perf. criteria A
	– Surge	IEC/EN 61000-4-5, 1 kV / 2 kV, perf. criteria A
	– Conducted RF	IEC/EN 61000-4-6, 3 Vrms perf. criteria A
	– Magnetic field	IEC/EN 61000-4-8, 3 A/m perf. criteria A
	– Voltage dip	IEC/EN 61000-4-11 >95 %, perf. criteria A, 0.5 periods 30 %, perf. criteria A, 25 periods >95 % perf. criteria B, 250 periods
Safety standards		UL 60950-1, IEC/EN 60950-1
Safety approvals	– UL/cUL	www.ul.com → certifications → File: e188913
	– CB report	www.tracopower.com/overview/txl
Environmental compliance	– Reach	www.tracopower.com/products/reach-declaration.pdf
	– RoHS	RoHS directive 2011/65/EU
Connection		5 pole terminal pitch (7.62mm with plastic cover)
Casing material		aluminium (chassis and cover)
Weight		0.13 kg

Outline Dimensions



Dimensions in [mm], () = Inch
Tolerances ±0.5 (±0.02)

MQ-2 Semiconductor Sensor for Combustible Gas

Sensitive material of MQ-2 gas sensor is SnO_2, which with lower conductivity in clean air. When the target combustible gas exist, The sensor's conductivity is more higher along with the gas concentration rising. Please use simple electrocircuit, Convert change of conductivity to correspond output signal of gas concentration.

MQ-2 gas sensor has high sensitity to LPG, Propane and Hydrogen, also could be used to Methane and other combustible steam, it is with low cost and suitable for different application.

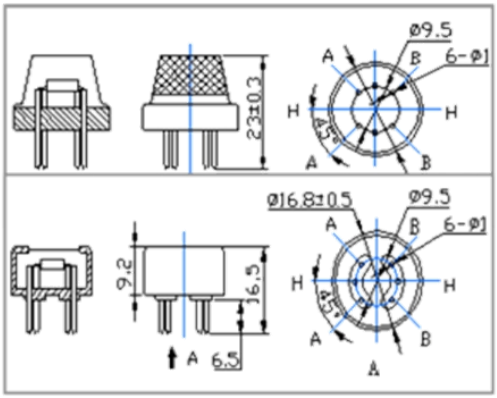
Character

*Good sensitivity to Combustible gas in wide range
* High sensitivity to LPG, Propane and Hydrogen
* Long life and low cost
* Simple drive circuit

Application

* Domestic gas leakage detector
* Industrial Combustible gas detector
* Portable gas detector

Configuration

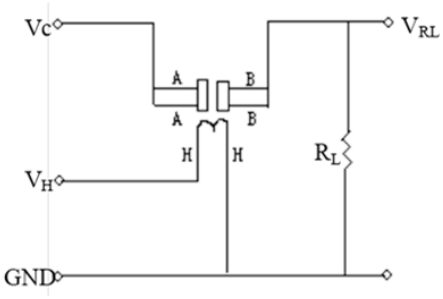


Technical Data

Model No.			MQ-2
Sensor Type			Semiconductor
Standard Encapsulation			Bakelite (Black Bakelite)
Detection Gas			Combustible gas and smoke
Concentration			300-10000ppm (Combustible gas)
Circuit	Loop Voltage	V_C	≤24V DC
	Heater Voltage	V_H	5.0V±0.2V ACorDC
	Load Resistance	R_L	Adjustable
Character	Heater Resistance	R_H	31Ω±3Ω（Room Tem.）
	Heater consumption	P_H	≤900mW
	Sensing Resistance	R_s	2KΩ-20KΩ(in 2000ppm C_4H_8)
	Sensitivity	S	Rs(in air)/Rs(1000ppm isobutane)≥5
	Slope	α	≤0.6($R_{5000ppm}/R_{3000ppm}$ CH_4)
Condition	Tem. Humidity		20℃±2℃；65%±5%RH
	Standard test circuit		Vc:5.0V±0.1V； V_H: 5.0V±0.1V
	Preheat time		Over 48 hours

Basic test loop

T



The above is basic test circuit of the sensor. The sensor need to be put 2 voltage, heater voltage（VH） and test voltage（VC）. VH used to supply certified working temperature to the sensor, while VC used to detect voltage (VRL) on load resistance （RL） whom is in series with sensor. The sensor has light polarity, Vc need DC power. VC and VH could use same power circuit with precondition to assure performance of sensor. In order to make the sensor with better performance, suitable RL value is needed:

Power of Sensitivity body(Ps):

$Ps=Vc^2\times Rs/(Rs+RL)^2$

Resistance of sensor(Rs): Rs=(Vc/VRL-1)×RL

Sensitivity Characteristics

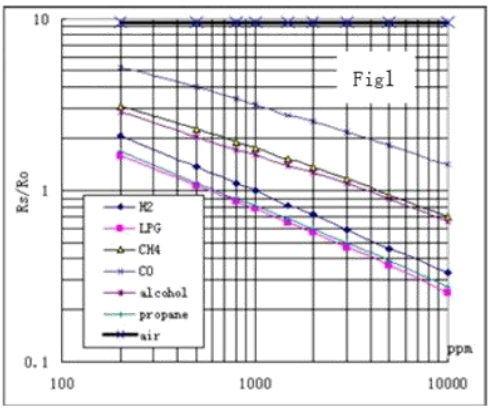


Fig.1 shows the typical sensitivity characteristics of the MQ-2, ordinate means resistance ratio of the sensor (Rs/Ro), abscissa is concentration of gases. Rs means resistance in different gases, Ro means resistance of sensor in 1000ppm Hyrogen. All test are under standard test conditions.

Influence of Temperature/Humidity

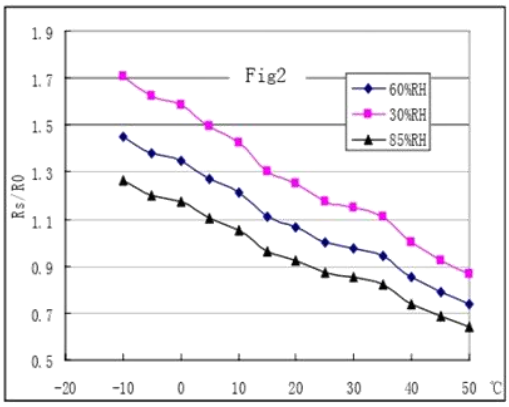
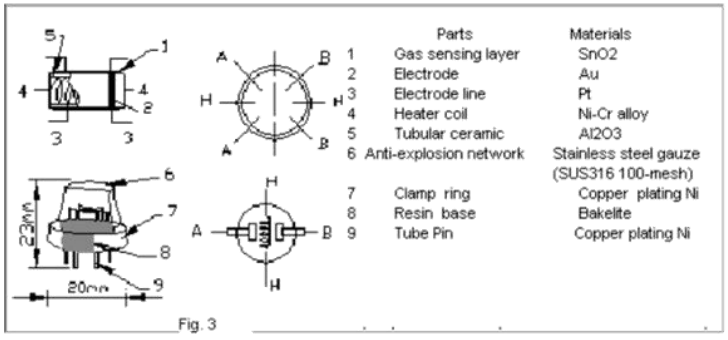


Fig.2 shows the typical temperature and humidity characteristics. Ordinate means resistance ratio of the sensor (Rs/Ro), Rs means resistance of sensor in 1000ppm Butane under different tem. and humidity. Ro means resistance of the sensor in environment of 1000ppm Methane, 20℃/65%RH

Structure and configuration



Structure and configuration of MQ-2 gas sensor is shown as Fig. 3, sensor composed by micro AL2O3 ceramic tube, Tin Dioxide (SnO2) sensitive layer, measuring electrode and heater are fixed into a crust made by plastic and stainless steel net. The heater provides necessary work conditions for work of sensitive components. The enveloped MQ-2 have 6 pin, 4 of them are used to fetch signals, and other 2 are used for providing heating current.

Notification

1 Following conditions must be prohibited

1.1 Exposed to organic silicon steam

Organic silicon steam cause sensors invalid, sensors must be avoid exposing to silicon bond, fixature, silicon latex, putty or plastic contain silicon environment

1.2 High Corrosive gas

If the sensors exposed to high concentration corrosive gas (such as H_2Sz, SO_X, Cl_2, HCl etc), it will not only result in corrosion of sensors structure, also it cause sincere sensitivity attenuation.

1.3 Alkali, Alkali metals salt, halogen pollution

The sensors performance will be changed badly if sensors be sprayed polluted by alkali metals salt especially brine, or be exposed to halogen such as fluorin.

1.4 Touch water

Sensitivity of the sensors will be reduced when spattered or dipped in water.

1.5 Freezing

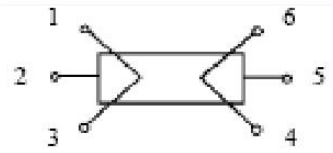
Do avoid icing on sensor'surface, otherwise sensor would lose sensitivity.

1.6 Applied voltage higher

Applied voltage on sensor should not be higher than stipulated value, otherwise it cause down-line or heater damaged, and bring on sensors' sensitivity characteristic changed badly.

1.7 Voltage on wrong pins

For 6 pins sensor, if apply voltage on 1、3 pins or 4、6 pins, it will make lead broken, and without signal when apply on 2、4 pins



2 Following conditions must be avoided

2.1 Water Condensation

Indoor conditions, slight water condensation will effect sensors performance lightly. However, if water condensation on sensors surface and keep a certain period, sensor' sensitivity will be decreased.

2.2 Used in high gas concentration

No matter the sensor is electrified or not, if long time placed in high gas concentration, if will affect sensors characteristic.

2.3 Long time storage

The sensors resistance produce reversible drift if it's stored for long time without electrify, this drift is related with storage conditions. Sensors should be stored in airproof without silicon gel bag with clean air. For the sensors with long time storage but no electrify, they need long aging time for stbility before using.

2.4 Long time exposed to adverse environment

No matter the sensors electrified or not, if exposed to adverse environment for long time, such as high humidity, high temperature, or high pollution etc, it will effect the sensors performance badly.

2.5 Vibration

Continual vibration will result in sensors down-lead response then repture. In transportation or assembling line, pneumatic screwdriver/ultrasonic welding machine can lead this vibration.

2.6 Concussion

If sensors meet strong concussion, it may lead its lead wire disconnected.

2.7 Usage

For sensor, handmade welding is optimal way. If use wave crest welding should meet the following conditions:

2.7.1 Soldering flux: Rosin soldering flux contains least chlorine
2.7.2 Speed: 1-2 Meter/ Minute
2.7.3 Warm-up temperature：100±20℃
2.7.4 Welding temperature：250±10℃
2.7.5 1 time pass wave crest welding machine

If disobey the above using terms, sensors sensitivity will be reduced.

MQ-3 Semiconductor Sensor for Alcohol

Sensitive material of MQ-3 gas sensor is SnO_2, which with lower conductivity in clean air. When the target alcohol gas exist, The sensor's conductivity is more higher along with the gas concentration rising. Please use simple electrocircuit, Convert change of conductivity to correspond output signal of gas concentration.

MQ-3 gas sensor has high sensitity to Alcohol, and has good resistance to disturb of gasoline, smoke and vapor. The sensor could be used to detect alcohol with different concentration, it is with low cost and suitable for different application.

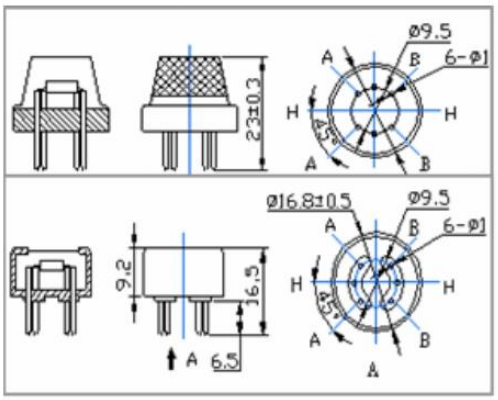
Character

* Good sensitivity to alcohol gas
* Long life and low cost
* Simple drive circuit

Application

* Vehicel alcohol detector
* Portable alcohol detector

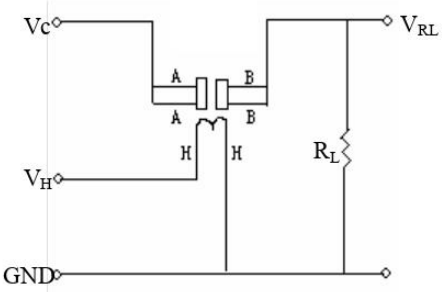
Configuration



Technical Data

Basic test loop

Model No.			MQ-3
Sensor Type			Semiconductor
Standard Encapsulation			Bakelite (Black Bakelite)
Detection Gas			Alcohol gas
Concentration			0.04-4mg/l alcohol
Circuit	Loop Voltage	V_c	≤24V DC
	Heater Voltage	V_H	5.0V±0.2V AC or DC
	Load Resistance	R_L	Adjustable
Character	Heater Resistance	R_H	31Ω±3Ω（Room Tem.）
	Heater consumption	P_H	≤900mW
	Sensing Resistance	R_s	2KΩ-20KΩ(in 0.4mg/l alcohol)
	Sensitivity	S	Rs(in air)/Rs(0.4mg/L Alcohol)≥5
	Slope	α	≤0.6(R_{300ppm}/R_{100ppm} Alcohol)
Condition	Tem. Humidity		20℃±2℃；65%±5%RH
	Standard test circuit		Vc:5.0V±0.1V；V_H: 5.0V±0.1V
	Preheat time		Over 48 hours



The above is basic test circuit of the sensor. The sensor need to be put 2 voltage, heater voltage（VH） and test voltage（VC）. VH used to supply certified working temperature to the sensor, while VC used to detect voltage (VRL) on load resistance （RL） whom is in series with sensor. The sensor has light polarity, Vc need DC power. VC and VH could use same power circuit with precondition to assure performance of sensor. In order to make the sensor with better performance, suitable RL value is needed:

Power of Sensitivity body(Ps):

$Ps=Vc^2 \times Rs/(Rs+RL)^2$

Resistance of sensor(Rs): Rs=(Vc/VRL-1)×RL

Sensitivity Characteristics

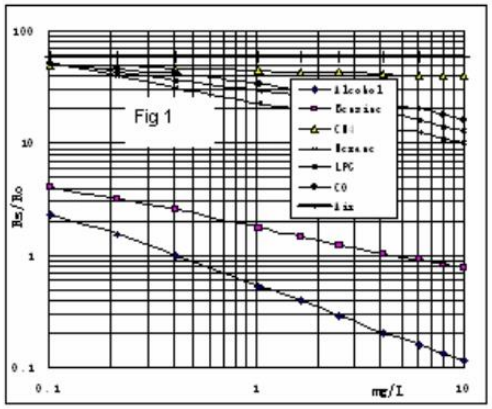


Fig.1 shows the typical sensitivity characteristics of the MQ-3, ordinate means resistance ratio of the sensor (Rs/Ro), abscissa is concentration of gases. Rs means resistance in different gases, Ro means resistance of sensor in 0. 4mg/l alcohol. All test are under standard test conditions.

P.S.: Sensitivity to smoke is ignite 10pcs cigarettes in $8m^3$ room, and the output equals to 0.1mg/l alcohol

Influence of Temperature/Humidity

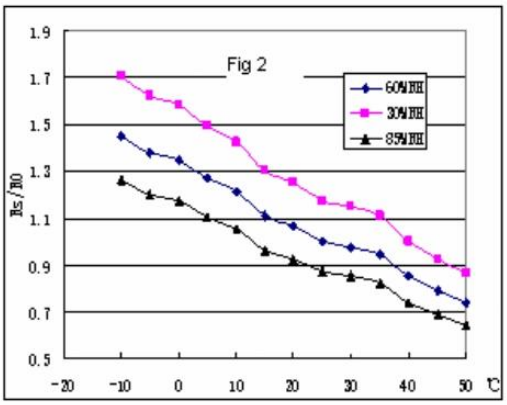
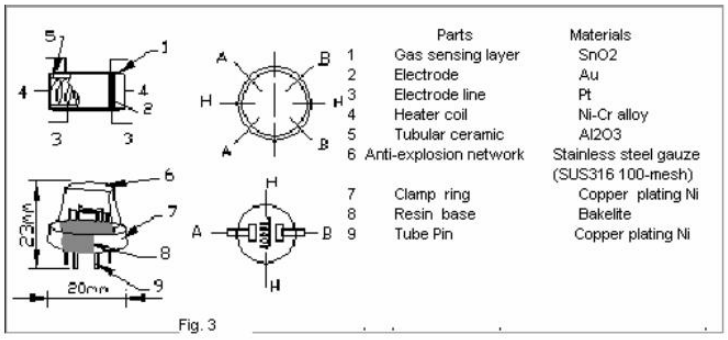


Fig.2 shows the typical temperature and humidity characteristics. Ordinate means resistance ratio of the sensor (Rs/Ro), Rs means resistance of sensor in 0.4mg/l alcohol under different tem. and humidity. Ro means resistance of the sensor in environment of 0.4mg/l alcohol, 20℃/65%RH

Structure and configuration



	Parts	Materials
1	Gas sensing layer	SnO2
2	Electrode	Au
3	Electrode line	Pt
4	Heater coil	Ni-Cr alloy
5	Tubular ceramic	Al2O3
6	Anti-explosion network	Stainless steel gauze (SUS316 100-mesh)
7	Clamp ring	Copper plating Ni
8	Resin base	Bakelite
9	Tube Pin	Copper plating Ni

Fig. 3

Structure and configuration of MQ-3 gas sensor is shown as Fig. 3, sensor composed by micro AL2O3 ceramic tube, Tin Dioxide (SnO2) sensitive layer, measuring electrode and heater are fixed into a crust made by plastic and stainless steel net. The heater provides necessary work conditions for work of sensitive components. The enveloped MQ-4 have 6 pin, 4 of them are used to fetch signals, and other 2 are used for providing heating current.

Notification

1 Following conditions must be prohibited

1.1 Exposed to organic silicon steam

Organic silicon steam cause sensors invalid, sensors must be avoid exposing to silicon bond, fixature, silicon latex, putty or plastic contain silicon environment

1.2 High Corrosive gas

If the sensors exposed to high concentration corrosive gas (such as H_2Sz, SO_X, Cl_2, HCl etc), it will not only result in corrosion of sensors structure, also it cause sincere sensitivity attenuation.

1.3 Alkali, Alkali metals salt, halogen pollution

The sensors performance will be changed badly if sensors be sprayed polluted by alkali metals salt especially brine, or be exposed to halogen such as fluorin.

1.4 Touch water

Sensitivity of the sensors will be reduced when spattered or dipped in water.

1.5 Freezing

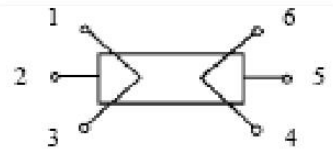
Do avoid icing on sensor'surface, otherwise sensor would lose sensitivity.

1.6 Applied voltage higher

Applied voltage on sensor should not be higher than stipulated value, otherwise it cause down-line or heater damaged, and bring on sensors' sensitivity characteristic changed badly.

1.7 Voltage on wrong pins

For 6 pins sensor, if apply voltage on 1、3 pins or 4、6 pins, it will make lead broken, and without signal when apply on 2、4 pins



2 Following conditions must be avoided

2.1 Water Condensation

Indoor conditions, slight water condensation will effect sensors performance lightly. However, if water condensation on sensors surface and keep a certain period, sensor' sensitivity will be decreased.

2.2 Used in high gas concentration

No matter the sensor is electrified or not, if long time placed in high gas concentration, if will affect sensors characteristic.

2.3 Long time storage

The sensors resistance produce reversible drift if it's stored for long time without electrify, this drift is related with storage conditions. Sensors should be stored in airproof without silicon gel bag with clean air. For the sensors with long time storage but no electrify, they need long aging time for stbility before using.

2.4 Long time exposed to adverse environment

No matter the sensors electrified or not, if exposed to adverse environment for long time, such as high humidity, high temperature, or high pollution etc, it will effect the sensors performance badly.

2.5 Vibration

Continual vibration will result in sensors down-lead response then repture. In transportation or assembling line, pneumatic screwdriver/ultrasonic welding machine can lead this vibration.

2.6 Concussion

If sensors meet strong concussion, it may lead its lead wire disconnected.

2.7 Usage

For sensor, handmade welding is optimal way. If use wave crest welding should meet the following conditions:

2.7.1 Soldering flux: Rosin soldering flux contains least chlorine
2.7.2 Speed: 1-2 Meter/ Minute
2.7.3 Warm-up temperature：100±20℃
2.7.4 Welding temperature：250±10℃
2.7.5 1 time pass wave crest welding machine

If disobey the above using terms, sensors sensitivity will be reduced.

MQ-4 Semiconductor Sensor for Natural Gas

Sensitive material of MQ-4 gas sensor is SnO_2, which with lower conductivity in clean air. When the target combustible gas exist, The sensor's conductivity is more higher along with the gas concentration rising. Please use simple electrocircuit, Convert change of conductivity to correspond output signal of gas concentration.

MQ-4 gas sensor has high sensitity to Methane, also to Propane and Butane. The sensor could be used to detect different combustible gas, especially Methane, it is with low cost and suitable for different application.

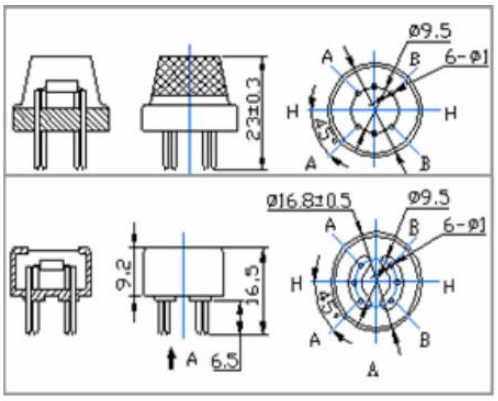
Character

* Good sensitivity to Combustible gas in wide range
* High sensitivity to Natural gas
* Long life and low cost
* Simple drive circuit

Application

* Domestic gas leakage detector
* Industrial Combustible gas detector
* Portable gas detector

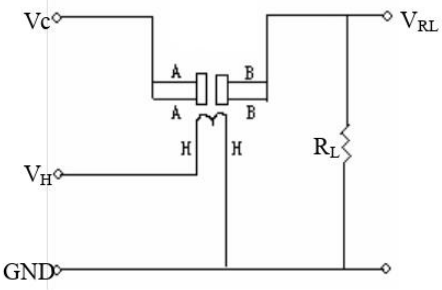
Configuration



Technical Data

Model No.			MQ-4
Sensor Type			Semiconductor
Standard Encapsulation			Bakelite (Black Bakelite)
Detection Gas			Natural gas/ Methane
Concentration			300-10000ppm (Natural gas / Methane)
Circuit	Loop Voltage	V_c	≤24V DC
	Heater Voltage	V_H	5.0V±0.2V ACorDC
	Load Resistance	R_L	Adjustable
Character	Heater Resistance	R_H	31Ω±3Ω（Room Tem.）
	Heater consumption	P_H	≤900mW
	Sensing Resistance	R_s	2KΩ-20KΩ(in 5000ppm CH_4)
	Sensitivity	S	Rs(in air)/Rs(5000ppm CH_4)≥5
	Slope	α	≤0.6($R_{5000ppm}/R_{3000ppm}$ CH_4)
Condition	Tem. Humidity		20℃±2℃；65%±5%RH
	Standard test circuit		Vc:5.0V±0.1V； V_H: 5.0V±0.1V
	Preheat time		Over 48 hours

Basic test loop



The above is basic test circuit of the sensor. The sensor need to be put 2 voltage, heater voltage（VH） and test voltage（VC）. VH used to supply certified working temperature to the sensor, while VC used to detect voltage (VRL) on load resistance （RL） whom is in series with sensor. The sensor has light polarity, Vc need DC power. VC and VH could use same power circuit with precondition to assure performance of sensor. In order to make the sensor with better performance, suitable RL value is needed:
Power of Sensitivity body(Ps):

$Ps=Vc^2\times Rs/(Rs+RL)^2$

Resistance of sensor(Rs): $Rs=(Vc/VRL-1)\times RL$

Sensitivity Characteristics

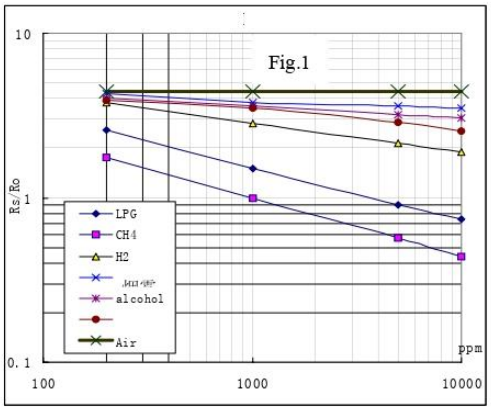


Fig.1 shows the typical sensitivity characteristics of the MQ-4, ordinate means resistance ratio of the sensor (Rs/Ro), abscissa is concentration of gases. Rs means resistance in different gases, Ro means resistance of sensor in 1000ppm Methane. All test are under standard test conditions.

P.S.: Sensitivity to smoke is ignite 10pcs cigarettes in $8m^3$ room, and the output equals to 200ppm Methane

Influence of Temperature/Humidity

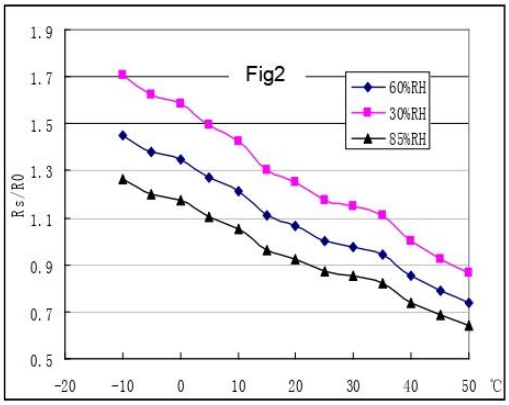
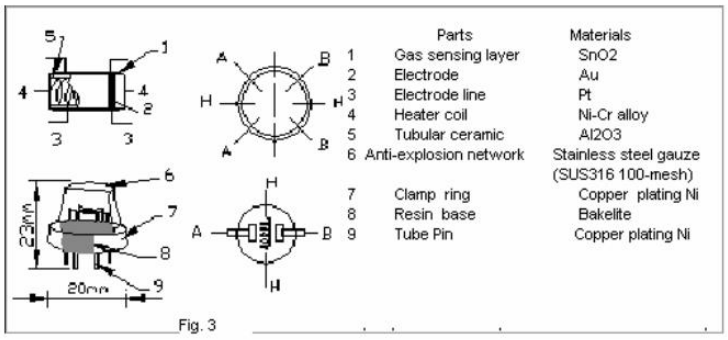


Fig.2 shows the typical temperature and humidity characteristics. Ordinate means resistance ratio of the sensor (Rs/Ro), Rs means resistance of sensor in 1000ppm Methane under different tem. and humidity. Ro means resistance of the sensor in environment of 1000ppm Methane, 20℃/65%RH

Structure and configuration



Structure and configuration of MQ-4 gas sensor is shown as Fig. 3, sensor composed by micro AL2O3 ceramic tube, Tin Dioxide (SnO2) sensitive layer, measuring electrode and heater are fixed into a crust made by plastic and stainless steel net. The heater provides necessary work conditions for work of sensitive components. The enveloped MQ-4 have 6 pin, 4 of them are used to fetch signals, and other 2 are used for providing heating current.

Notification

1 Following conditions must be prohibited

1.1 Exposed to organic silicon steam

Organic silicon steam cause sensors invalid, sensors must be avoid exposing to silicon bond, fixature, silicon latex, putty or plastic contain silicon environment

1.2 High Corrosive gas

If the sensors exposed to high concentration corrosive gas (such as H_2Sz, SO_X, Cl_2, HCl etc), it will not only result in corrosion of sensors structure, also it cause sincere sensitivity attenuation.

1.3 Alkali, Alkali metals salt, halogen pollution

The sensors performance will be changed badly if sensors be sprayed polluted by alkali metals salt especially brine, or be exposed to halogen such as fluorin.

1.4 Touch water

Sensitivity of the sensors will be reduced when spattered or dipped in water.

1.5 Freezing

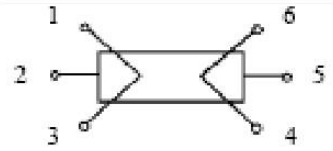
Do avoid icing on sensor'surface, otherwise sensor would lose sensitivity.

1.6 Applied voltage higher

Applied voltage on sensor should not be higher than stipulated value, otherwise it cause down-line or heater damaged, and bring on sensors' sensitivity characteristic changed badly.

1.7 Voltage on wrong pins

For 6 pins sensor, if apply voltage on 1、3 pins or 4、6 pins, it will make lead broken, and without signal when apply on 2、4 pins



2 Following conditions must be avoided

2.1 Water Condensation

Indoor conditions, slight water condensation will effect sensors performance lightly. However, if water condensation on sensors surface and keep a certain period, sensor' sensitivity will be decreased.

2.2 Used in high gas concentration

No matter the sensor is electrified or not, if long time placed in high gas concentration, if will affect sensors characteristic.

2.3 Long time storage

The sensors resistance produce reversible drift if it's stored for long time without electrify, this drift is related with storage conditions. Sensors should be stored in airproof without silicon gel bag with clean air. For the sensors with long time storage but no electrify, they need long aging time for stbility before using.

2.4 Long time exposed to adverse environment

No matter the sensors electrified or not, if exposed to adverse environment for long time, such as high humidity, high temperature, or high pollution etc, it will effect the sensors performance badly.

2.5 Vibration

Continual vibration will result in sensors down-lead response then repture. In transportation or assembling line, pneumatic screwdriver/ultrasonic welding machine can lead this vibration.

2.6 Concussion

If sensors meet strong concussion, it may lead its lead wire disconnected.

2.7 Usage

For sensor, handmade welding is optimal way. If use wave crest welding should meet the following conditions:

2.7.1 Soldering flux: Rosin soldering flux contains least chlorine
2.7.2 Speed: 1-2 Meter/ Minute
2.7.3 Warm-up temperature：100±20℃
2.7.4 Welding temperature：250±10℃
2.7.5 1 time pass wave crest welding machine

If disobey the above using terms, sensors sensitivity will be reduced.

TECHNICAL DATA MQ-5 GAS SENSOR

FEATURES

* High sensitivity to LPG, natural gas , town gas
* Small sensitivity to alcohol, smoke.
* Fast response . * Stable and long life * Simple drive circuit

APPLICATION

They are used in gas leakage detecting equipments in family and industry, are suitable for detecting of LPG, natural gas , town gas, avoid the noise of alcohol and cooking fumes and cigarette smoke.

SPECIFICATIONS

A. Standard work condition

Symbol	Parameter name	Technical condition	Remarks
Vc	Circuit voltage	5V±0.1	AC OR DC
V_H	Heating voltage	5V±0.1	ACOR DC
P_L	Load resistance	20KΩ	
R_H	Heater resistance	31±10%	Room Tem
P_H	Heating consumption	less than 800mw	

B. Environment condition

Symbol	Parameter name	Technical condition	Remarks
Tao	Using Tem	-10℃-50℃	
Tas	Storage Tem	-20℃-70℃	
R_H	Related humidity	less than 95%Rh	
O_2	Oxygen concentration	21%(standard condition)Oxygen concentration can affect sensitivity	minimum value is over 2%

C. Sensitivity characteristic

Symbol	Parameter name	Technical parameter	Remarks
Rs	Sensing Resistance	10KΩ - 60KΩ (5000ppm methane)	Detecting concentration scope: 200-10000ppm LPG,LNG Natural gas, iso-butane, propane Town gas
α (5000ppm/1000 ppm CH_4)	Concentration slope rate	≤0.6	
Standard detecting condition	Temp: 20℃±2℃ Humidity: 65%±5%	Vc:5V±0.1 Vh: 5V±0.1	
Preheat time	Over 24 hour		

D. Strucyure and configuration, basic measuring circuit

	Parts	Materials
1	Gas sensing layer	SnO_2
2	Electrode	Au
3	Electrode line	Pt
4	Heater coil	Ni-Cr alloy
5	Tubular ceramic	Al_2O_3
6	Anti-explosion network	Stainless steel gauze (SUS316 100-mesh)
7	Clamp ring	Copper plating Ni
8	Resin base	Bakelite
9	Tube Pin	Copper plating Ni

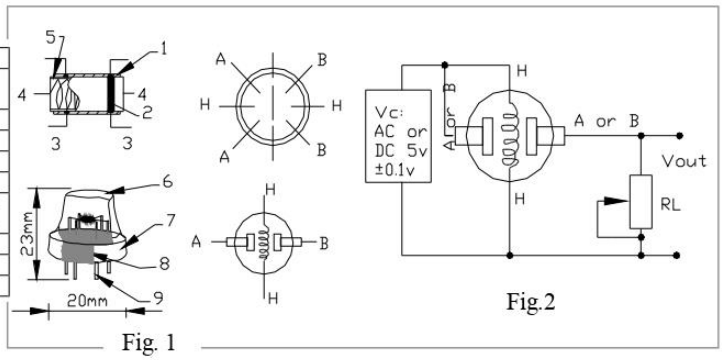
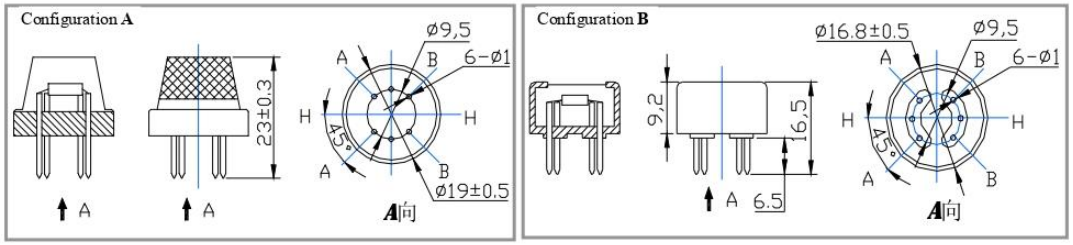


Fig. 1 Fig.2



Structure and configuration of MQ-5 gas sensor is shown as Fig. 1 (Configuration A or B), sensor composed by

micro AL_2O_3 ceramic tube, Tin Dioxide (SnO_2) sensitive layer, measuring electrode and heater are fixed into a crust made by plastic and stainless steel net. The heater provides necessary work conditions for work of sensitive components. The enveloped MQ-5 have 6 pin ,4 of them are used to fetch signals, and other 2 are used for providing heating current.

Electric parameter measurement circuit is shown as Fig.2

E. Sensitivity characteristic curve

Fig.2 sensitivity characteristics of the MQ-5

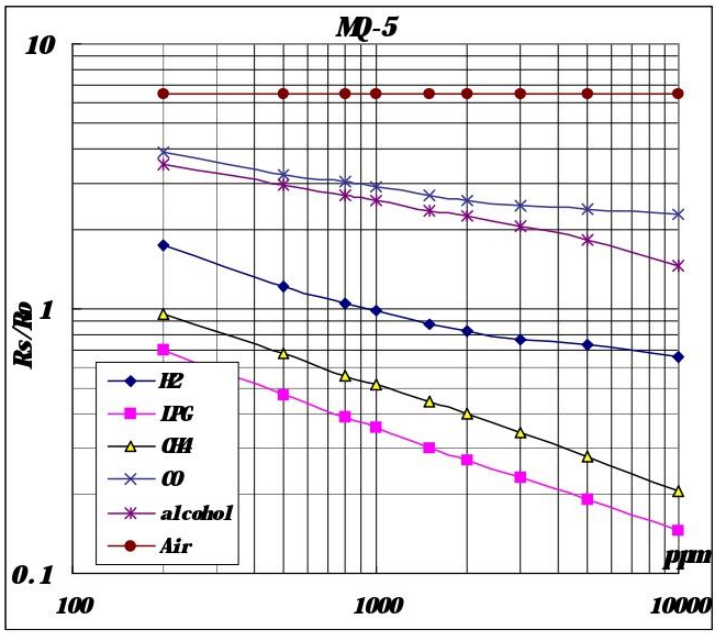


Fig.3 is shows the typical sensitivity characteristics of the MQ-5 for several gases.
in their: Temp: 20℃、
Humidity: 65%、
O_2 concentration 21%
RL=20kΩ
Ro: sensor resistance at 1000ppm of H_2 in the clean air.
Rs:sensor resistance at various concentrations of gases.

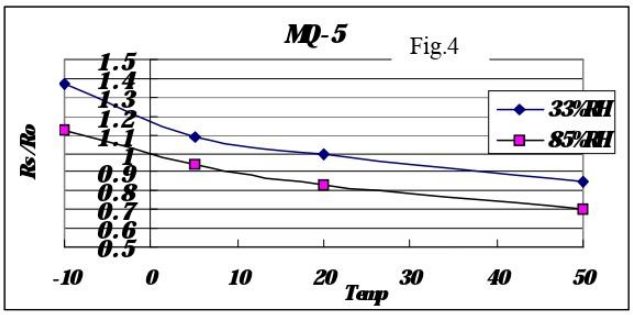


Fig.4 is shows the typical dependence of the MQ-5 on temperature and humidity.
Ro: sensor resistance at 1000ppm of H_2 in air at 33%RH and 20 degree.
Rs: sensor resistance at different temperatures and humidities.

SENSITVITY ADJUSTMENT

Resistance value of MQ-5 is difference to various kinds and various concentration gases. So, When using this components, sensitivity adjustment is very necessary. we recommend that you calibrate the detector for 1000ppm H_2 or LPG concentration in air and use value of Load resistance (R_L) about 20 KΩ (10KΩ to 47KΩ).

When accurately measuring, the proper alarm point for the gas detector should be determined after considering the temperature and humidity influence.

MQ-6 Semiconductor Sensor for LPG

Sensitive material of MQ-6 gas sensor is SnO_2, which with lower conductivity in clean air. When the target combustible gas exist, The sensor's conductivity is more higher along with the gas concentration rising. Please use simple electrocircuit, Convert change of conductivity to correspond output signal of gas concentration.

MQ-6 gas sensor has high sensitity to Propane, Butane and LPG, also response to Natural gas. The sensor could be used to detect different combustible gas, especially Methane, it is with low cost and suitable for different application.

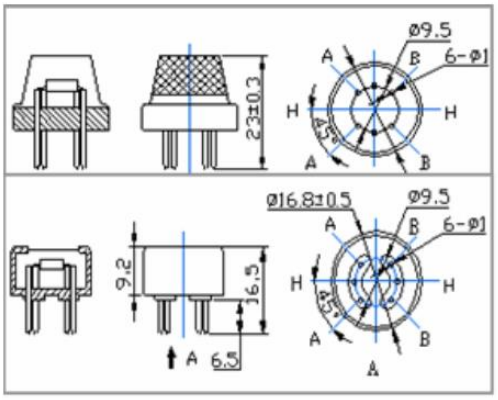
Character

* Good sensitivity to Combustible gas in wide range
* High sensitivity to Propane, Butane and LPG
* Long life and low cost
* Simple drive circuit

Application

* Domestic gas leakage detector
* Industrial Combustible gas detector
* Portable gas detector

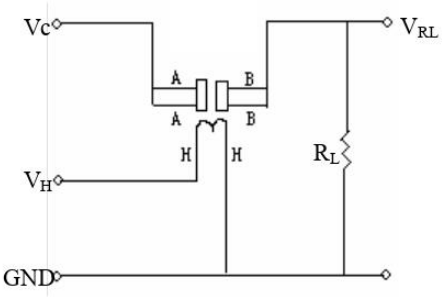
Configuration



Technical Data

Model No.			MQ-6
Sensor Type			Semiconductor
Standard Encapsulation			Bakelite (Black Bakelite)
Detection Gas			Isobutane, Butane, LPG
Concentration			300-10000ppm (Butane, Propane, LPG)
Circuit	Loop Voltage	V_c	≤24V DC
	Heater Voltage	V_H	5.0V±0.2V ACorDC
	Load Resistance	R_L	Adjustable
Character	Heater Resistance	R_H	31Ω±3Ω（Room Tem.）
	Heater consumption	P_H	≤900mW
	Sensing Resistance	R_s	2KΩ-20KΩ(in 2000ppm C_3H_8)
	Sensitivity	S	Rs(in air)/Rs(1000ppm C_4H_{10})≥5
	Slope	α	≤0.6($R_{2000ppm}/R_{1000ppm}$ LPG)
Condition	Tem. Humidity		20℃±2℃；65%±5%RH
	Standard test circuit		Vc:5.0V±0.1V； V_H: 5.0V±0.1V
	Preheat time		Over 48 hours

Basic test loop



The above is basic test circuit of the sensor. The sensor need to be put 2 voltage, heater voltage（VH） and test voltage（VC）. VH used to supply certified working temperature to the sensor, while VC used to detect voltage (VRL) on load resistance （RL） whom is in series with sensor. The sensor has light polarity, Vc need DC power. VC and VH could use same power circuit with precondition to assure performance of sensor. In order to make the sensor with better performance, suitable RL value is needed:
Power of Sensitivity body(Ps):

$Ps=Vc^2\times Rs/(Rs+RL)^2$

Resistance of sensor(Rs): $Rs=(Vc/VRL-1)\times RL$

Sensitivity Characteristics

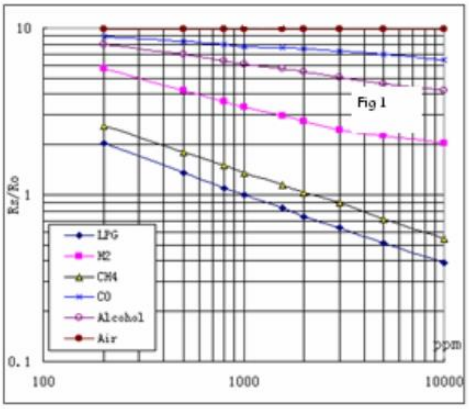


Fig.1 shows the typical sensitivity characteristics of the MQ-6, ordinate means resistance ratio of the sensor (Rs/Ro), abscissa is concentration of gases. Rs means resistance in different gases, Ro means resistance of sensor in 1000ppm LPG. All test are under standard test conditions.

Influence of Temperature/Humidity

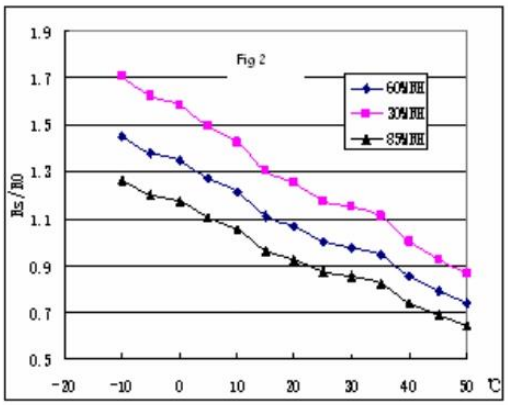
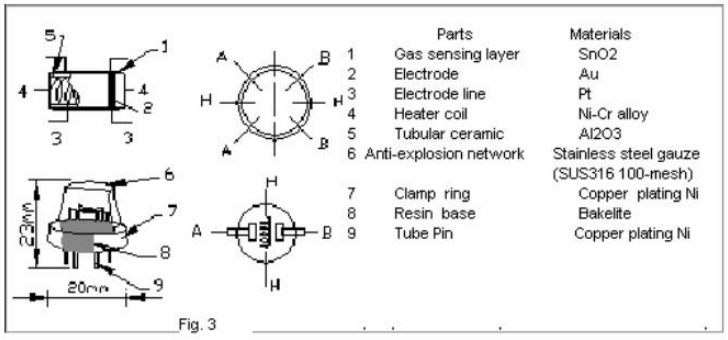


Fig.2 shows the typical temperature and humidity characteristics. Ordinate means resistance ratio of the sensor (Rs/Ro), Rs means resistance of sensor in 1000ppm Methane under different tem. and humidity. Ro means resistance of the sensor in environment of 1000ppm Propane, 20℃/65%RH

Structure and configuration



Structure and configuration of MQ-6 gas sensor is shown as Fig. 3, sensor composed by micro AL2O3 ceramic tube, Tin Dioxide (SnO2) sensitive layer, measuring electrode and heater are fixed into a crust made by plastic and stainless steel net. The heater provides necessary work conditions for work of sensitive components. The enveloped MQ-4 have 6 pin, 4 of them are used to fetch signals, and other 2 are used for providing heating current.

Notification

1 Following conditions must be prohibited

1.1 Exposed to organic silicon steam

Organic silicon steam cause sensors invalid, sensors must be avoid exposing to silicon bond, fixature, silicon latex, putty or plastic contain silicon environment

1.2 High Corrosive gas

If the sensors exposed to high concentration corrosive gas (such as H_2Sz, SO_X, Cl_2, HCl etc), it will not only result in corrosion of sensors structure, also it cause sincere sensitivity attenuation.

1.3 Alkali, Alkali metals salt, halogen pollution

The sensors performance will be changed badly if sensors be sprayed polluted by alkali metals salt especially brine, or be exposed to halogen such as fluorin.

1.4 Touch water

Sensitivity of the sensors will be reduced when spattered or dipped in water.

1.5 Freezing

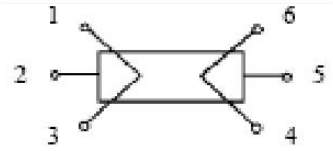
Do avoid icing on sensor'surface, otherwise sensor would lose sensitivity.

1.6 Applied voltage higher

Applied voltage on sensor should not be higher than stipulated value, otherwise it cause down-line or heater damaged, and bring on sensors' sensitivity characteristic changed badly.

1.7 Voltage on wrong pins

For 6 pins sensor, if apply voltage on 1、3 pins or 4、6 pins, it will make lead broken, and without signal when apply on 2、4 pins



2 Following conditions must be avoided

2.1 Water Condensation

Indoor conditions, slight water condensation will effect sensors performance lightly. However, if water condensation on sensors surface and keep a certain period, sensor' sensitivity will be decreased.

2.2 Used in high gas concentration

No matter the sensor is electrified or not, if long time placed in high gas concentration, if will affect sensors characteristic.

2.3 Long time storage

The sensors resistance produce reversible drift if it's stored for long time without electrify, this drift is related with storage conditions. Sensors should be stored in airproof without silicon gel bag with clean air. For the sensors with long time storage but no electrify, they need long aging time for stbility before using.

2.4 Long time exposed to adverse environment

No matter the sensors electrified or not, if exposed to adverse environment for long time, such as high humidity, high temperature, or high pollution etc, it will effect the sensors performance badly.

2.5 Vibration

Continual vibration will result in sensors down-lead response then repture. In transportation or assembling line, pneumatic screwdriver/ultrasonic welding machine can lead this vibration.

2.6 Concussion

If sensors meet strong concussion, it may lead its lead wire disconnected.

2.7 Usage

For sensor, handmade welding is optimal way. If use wave crest welding should meet the following conditions:

2.7.1 Soldering flux: Rosin soldering flux contains least chlorine
2.7.2 Speed: 1-2 Meter/ Minute
2.7.3 Warm-up temperature：100±20℃
2.7.4 Welding temperature：250±10℃
2.7.5 1 time pass wave crest welding machine

If disobey the above using terms, sensors sensitivity will be reduced.

TECHNICAL DATA MQ-8 GAS SENSOR

FEATURES

* High sensitivity to Hydrogen (H_2)
* Small sensitivity to alcohol, LPG,cooking fumes
* Stable and long life

APPLICATION

They are used in gas leakage detecting equipments in family and industry, are suitable for detecting of Hydrogen (H_2), avoid the noise of alcohol and cooking fumes, LPG,CO.

SPECIFICATIONS

A. Standard work condition

Symbol	Parameter name	Technical condition	Remarks
Vc	Circuit voltage	5V±0.1	AC OR DC
V_H	Heating voltage	5V±0.1	ACOR DC
P_L	Load resistance	10KΩ	
R_H	Heater resistance	31±5%	Room Tem
P_H	Heating consumption	less than800mW	

B. Environment condition

Symbol	Parameter name	Technical condition	Remarks
Tao	Using Tem	-10℃-50℃	
Tas	Storage Tem	-20℃-70℃	
R_H	Related humidity	less than 95%Rh	
O_2	Oxygen concentration	21%(standard condition)Oxygen concentration can affect sensitivity	minimum value is over 2%

C. Sensitivity characteristic

Symbol	Parameter name	Technical parameter	Ramark 2
Rs	Sensing Resistance	10KΩ - 60KΩ (1000ppm H_2)	Detecting concentration scope: 100-10000ppm Hydrogen (H_2)
α (1000ppm/ 500ppmH_2)	Concentration slope rate	≤0.6	
Standard detecting condition	Temp: 20℃±2℃ Humidity: 65%±5%	Vc:5V±0.1 Vh: 5V±0.1	
Preheat time	Over 24 hour		

D. Structure and configuration, basic measuring circuit

	Parts	Materials
1	Gas sensing layer	SnO_2
2	Electrode	Au
3	Electrode line	Pt
4	Heater coil	Ni-Cr alloy
5	Tubular ceramic	Al_2O_3
6	Anti-explosion network	Stainless steel gauze (SUS316 100-mesh)
7	Clamp ring	Copper plating Ni
8	Resin base	Bakelite
9	Tube Pin	Copper plating Ni

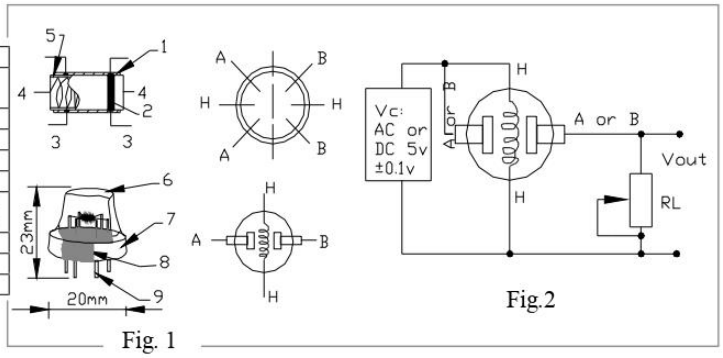
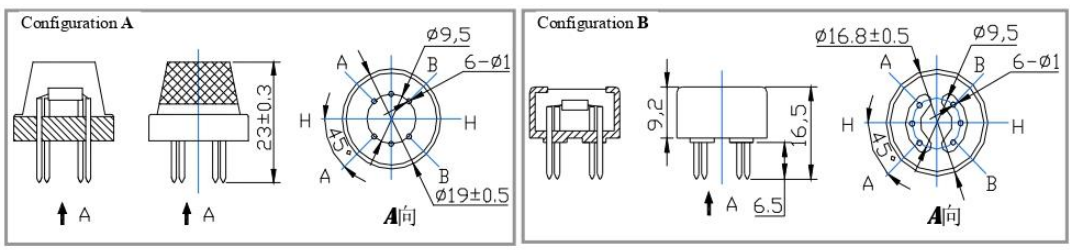


Fig. 1 Fig.2



Structure and configuration of MQ-8 gas sensor is shown as Fig. 1 (Configuration A or B), sensor composed by micro AL_2O_3 ceramic tube, Tin Dioxide (SnO_2) sensitive layer, measuring electrode and heater are fixed into a crust made by plastic and stainless steel net. The heater provides necessary work conditions for work of sensitive components. The enveloped MQ-8 have 6 pin ,4 of them are used to fetch signals, and other 2 are used for providing heating current.

Electric parameter measurement circuit is shown as Fig.2

E. Sensitivity characteristic curve

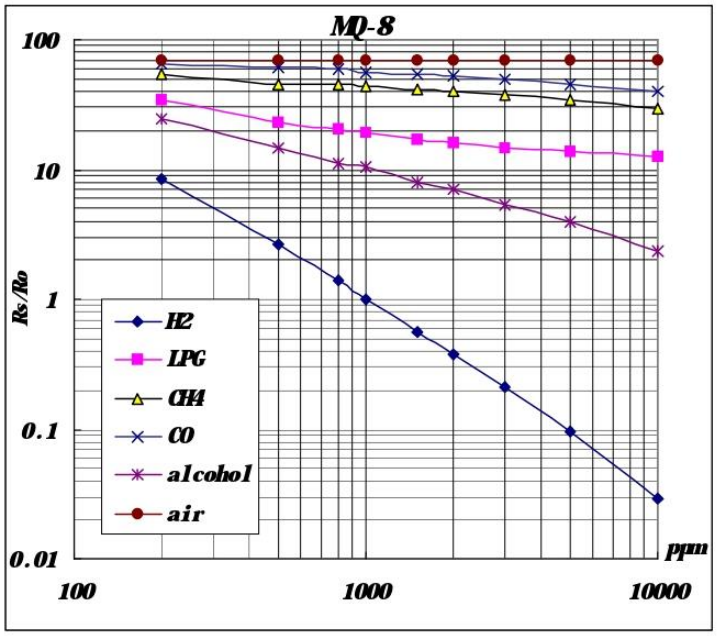


Fig.3 is shows the typical sensitivity characteristics of the MQ-8 for several gases.
in their: Temp: 20℃、
Humidity: 65%、
O_2 concentration 21%
RL=10kΩ
Ro: sensor resistance at 1000ppm H_2 in the clean air.
Rs:sensor resistance at various concentrations of gases.

Fig.2 sensitivity characteristics of the MQ-8

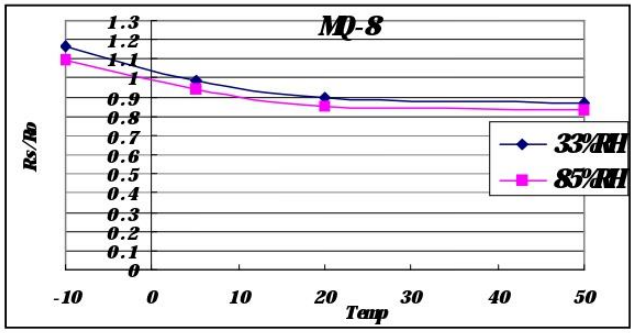


Fig.4 is shows the typical dependence of the MQ-8 on temperature and humidity.
Ro: sensor resistance at 1000ppm of H_2 in air at 33%RH and 20 degree.
Rs: sensor resistance at 1000ppm of H_2 in air at different temperatures and humidities.

SENSITVITY ADJUSTMENT

Resistance value of MQ-8 is difference to various kinds and various concentration gases. So,When using this components, sensitivity adjustment is very necessary. we recommend that you calibrate the detector for 1000ppm H_2 concentration in air and use value of Load resistance (R_L) about 10 KΩ (5KΩ to 33 KΩ).

When accurately measuring, the proper alarm point for the gas detector should be determined after considering the temperature and humidity influence.

TECHNICAL DATA MQ-135 GAS SENSOR

FEATURES

Wide detecting scope — Fast response and High sensitivity
Stable and long life — Simple drive circuit

APPLICATION

They are used in air quality control equipments for buildings/offices, are suitable for detecting of NH3,NOx, alcohol, Benzene, smoke,CO_2,etc.

SPECIFICATIONS

A. Standard work condition

Symbol	Parameter name	Technical condition	Remarks
Vc	Circuit voltage	5V±0.1	AC OR DC
V_H	Heating voltage	5V±0.1	ACOR DC
R_L	Load resistance	can adjust	
R_H	Heater resistance	33Ω ±5%	Room Tem
P_H	Heating consumption	less than 800mw	

B. Environment condition

Symbol	Parameter name	Technical condition	Remarks
Tao	Using Tem	-10℃-45℃	
Tas	Storage Tem	-20℃-70℃	
R_H	Related humidity	less than 95%Rh	
O_2	Oxygen concentration	21%(standard condition)Oxygen concentration can affect sensitivity	minimum value is over 2%

C. Sensitivity characteristic

Symbol	Parameter name	Technical parameter	Ramark 2
Rs	Sensing Resistance	30KΩ-200KΩ (100ppm NH_3)	Detecting concentration scope: 10ppm-300ppm NH_3 10ppm-1000ppm Benzene 10ppm-300ppm Alcohol
α (200/50) NH_3	Concentration Slope rate	≤0.65	
Standard Detecting Condition	Temp: 20℃±2℃ Humidity: 65%±5%	Vc:5V±0.1 Vh: 5V±0.1	
Preheat time	Over 24 hour		

D. Structure and configuration, basic measuring circuit

	Parts	Materials
1	Gas sensing layer	SnO_2
2	Electrode	Au
3	Electrode line	Pt
4	Heater coil	Ni-Cr alloy
5	Tubular ceramic	Al_2O_3
6	Anti-explosion network	Stainless steel gauze (SUS316 100-mesh)
7	Clamp ring	Copper plating Ni
8	Resin base	Bakelite
9	Tube Pin	Copper plating Ni

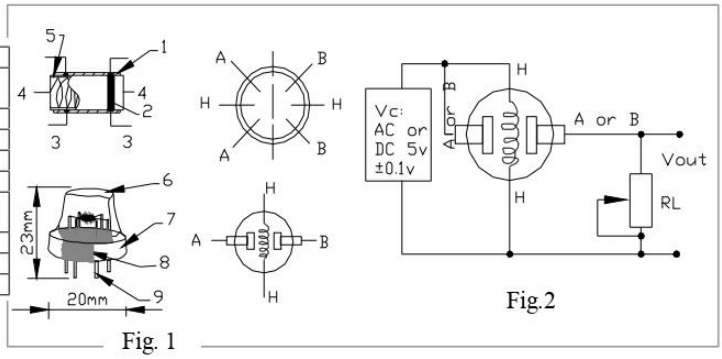
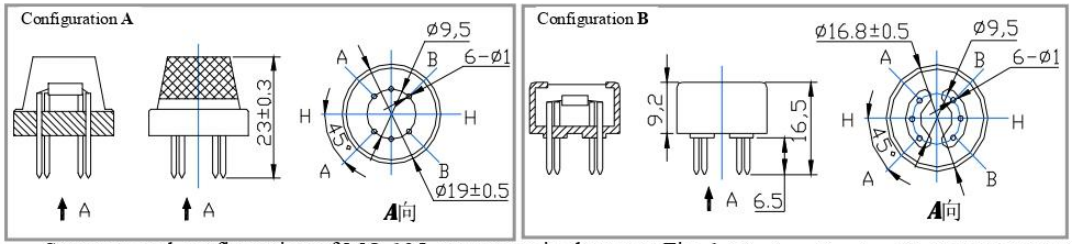


Fig. 1 Fig.2



Structure and configuration of MQ-135 gas sensor is shown as Fig. 1 (Configuration **A or B**), sensor composed by micro AL_2O_3 ceramic tube, Tin Dioxide (SnO_2) sensitive layer, measuring electrode and heater are fixed into a crust made by plastic and stainless steel net. The heater provides necessary work conditions for work of

sensitive components. The enveloped MQ-135 have 6 pin ,4 of them are used to fetch signals, and other 2 are used for providing heating current.

Electric parameter measurement circuit is shown as Fig.2

E. Sensitivity characteristic curve

Fig.2 sensitivity characteristics of the MQ-135

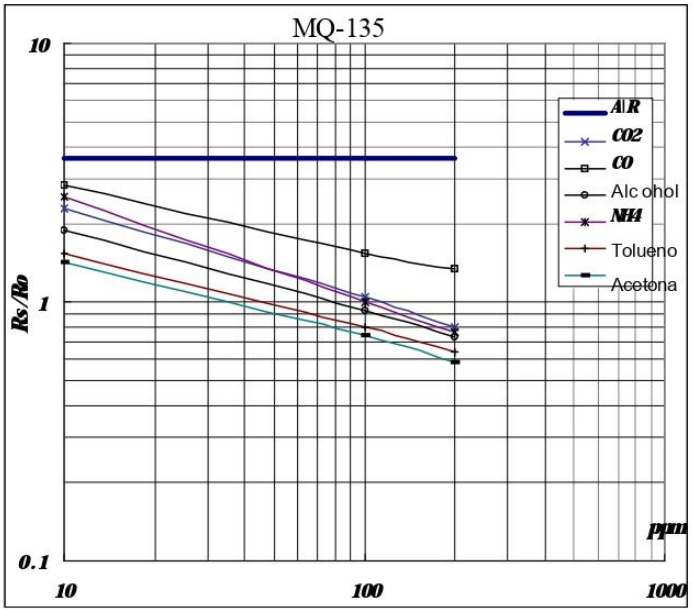


Fig.3 is shows the typical sensitivity characteristics of the MQ-135 for several gases.
in their: Temp: 20℃、
Humidity: 65%、
O_2 concentration 21%
RL=20kΩ
Ro: sensor resistance at 100ppm of NH_3 in the clean air.
Rs:sensor resistance at various concentrations of gases.

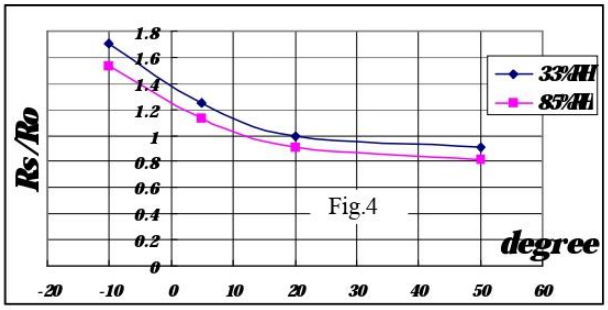


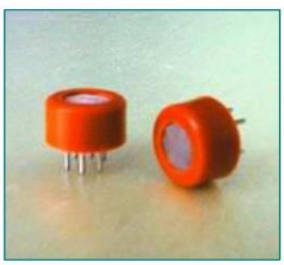
Fig.4

Fig.4 is shows the typical dependence of the MQ-135 on temperature and humidity.
Ro: sensor resistance at 100ppm of NH_3 in air at 33%RH and 20 degree.
Rs: sensor resistance at 100ppm of NH_3 at different temperatures and humidities.

SENSITIVITY ADJUSTMENT

Resistance value of MQ-135 is difference to various kinds and various concentration gases. So,When using this components, sensitivity adjustment is very necessary. we recommend that you calibrate the detector for 100ppm NH_3 or 50ppm Alcohol concentration in air and use value of Load resistancethat(R_L) about 20 KΩ (10KΩ to 47 KΩ).

When accurately measuring, the proper alarm point for the gas detector should be determined after considering the temperature and humidity influence.



Notification

1 Following conditions must be prohibited

1.1 Exposed to organic silicon steam

Organic silicon steam cause sensors invalid, sensors must be avoid exposing to silicon bond, fixature, silicon latex, putty or plastic contain silicon environment

1.2 High Corrosive gas

If the sensors exposed to high concentration corrosive gas (such as H_2Sz, SO_X, Cl_2, HCl etc), it will not only result in corrosion of sensors structure, also it cause sincere sensitivity attenuation.

1.3 Alkali, Alkali metals salt, halogen pollution

The sensors performance will be changed badly if sensors be sprayed polluted by alkali metals salt especially brine, or be exposed to halogen such as fluorin.

1.4 Touch water

Sensitivity of the sensors will be reduced when spattered or dipped in water.

1.5 Freezing

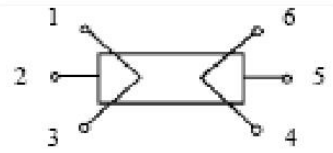
Do avoid icing on sensor'surface, otherwise sensor would lose sensitivity.

1.6 Applied voltage higher

Applied voltage on sensor should not be higher than stipulated value, otherwise it cause down-line or heater damaged, and bring on sensors' sensitivity characteristic changed badly.

1.7 Voltage on wrong pins

For 6 pins sensor, if apply voltage on 1、3 pins or 4、6 pins, it will make lead broken, and without signal when apply on 2、4 pins



2 Following conditions must be avoided

2.1 Water Condensation

Indoor conditions, slight water condensation will effect sensors performance lightly. However, if water condensation on sensors surface and keep a certain period, sensor' sensitivity will be decreased.

2.2 Used in high gas concentration

No matter the sensor is electrified or not, if long time placed in high gas concentration, if will affect sensors characteristic.

2.3 Long time storage

The sensors resistance produce reversible drift if it's stored for long time without electrify, this drift is related with storage conditions. Sensors should be stored in airproof without silicon gel bag with clean air. For the sensors with long time storage but no electrify, they need long aging time for stbility before using.

2.4 Long time exposed to adverse environment

No matter the sensors electrified or not, if exposed to adverse environment for long time, such as high humidity, high temperature, or high pollution etc, it will effect the sensors performance badly.

2.5 Vibration

Continual vibration will result in sensors down-lead response then repture. In transportation or assembling line, pneumatic screwdriver/ultrasonic welding machine can lead this vibration.

2.6 Concussion

If sensors meet strong concussion, it may lead its lead wire disconnected.

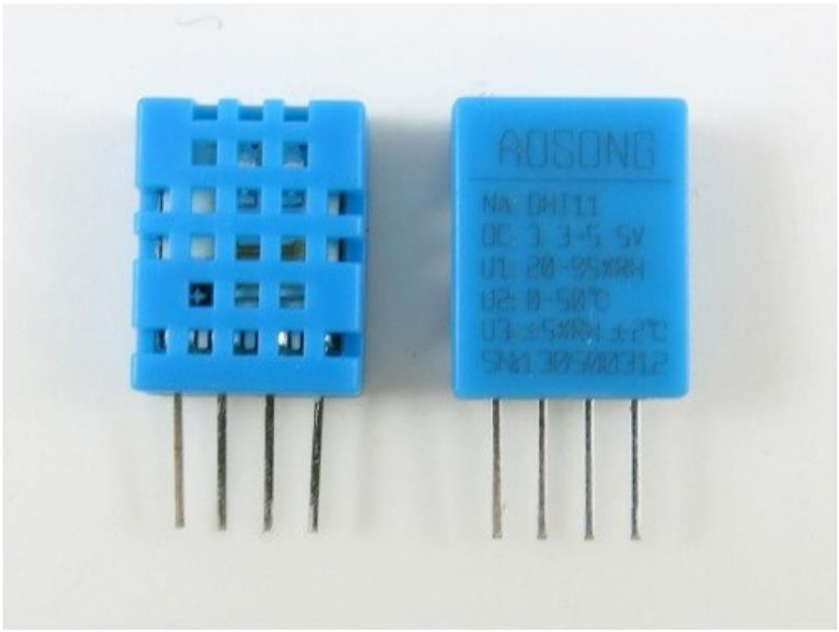
2.7 Usage

For sensor, handmade welding is optimal way. If use wave crest welding should meet the following conditions:

2.7.1 Soldering flux: Rosin soldering flux contains least chlorine
2.7.2 Speed: 1-2 Meter/ Minute
2.7.3 Warm-up temperature：100±20℃
2.7.4 Welding temperature：250±10℃
2.7.5 1 time pass wave crest welding machine

If disobey the above using terms, sensors sensitivity will be reduced.

Temperature and humidity module DHT11 Product Manual



1、Product Overview

DHT11 digital temperature and humidity sensor is a composite Sensor contains a calibrated digital signal output of the temperature and humidity. Application of a dedicated digital modules collection technology and the temperature and humidity sensing technology, to ensure that the product has high reliability and excellent long-term stability. The sensor includes a resistive sense of wet components and an NTC temperature measurement devices, and connected with a high-performance 8-bit microcontroller.

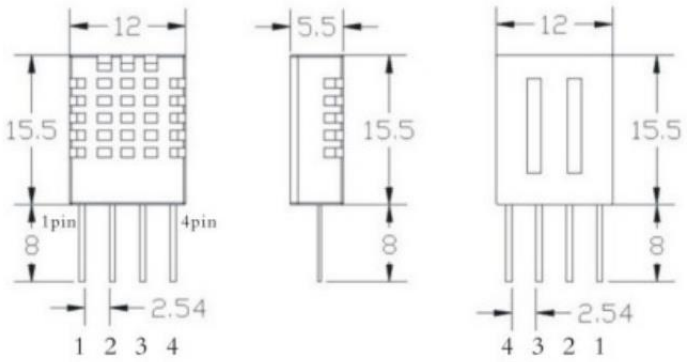
2、Applications

HVAC, dehumidifier, testing and inspection equipment, consumer goods, automotive, automatic control, data loggers, weather stations, home appliances, humidity regulator, medical and other humidity
measurement and control.

3、Features

Low cost, long-term stability, relative humidity and temperature measurement, excellent quality, fast response, strong anti-interference ability, long distance signal transmission, digital signal output, and precise calibration.

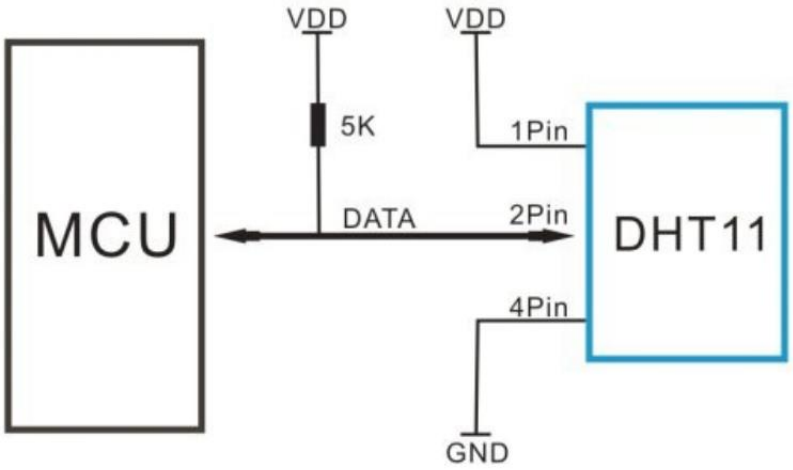
4、Dimensions (unit: mm)



5、Product parameters

- Relative humidity
- Resolution: 16Bit
- Repeatability: ±1% RH
- Accuracy: At 25℃ ±5% RH
- Interchangeability: fully interchangeable
- Response time: 1 / e (63%) of 25℃ 6s
- 1m / s air 6s
- Hysteresis: <± 0.3% RH
- Long-term stability: <± 0.5% RH / yr in
- Temperature
- Resolution: 16Bit
- Repeatability: ±0.2℃
- Range: At 25℃ ±2℃
- Response time: 1 / e (63%) 10S
- Electrical Characteristics
- Power supply: DC 3.5 ~ 5.5V
- Supply Current: measurement 0.3mA standby 60μ A
- Sampling period: more than 2 seconds
- Pin Description
- 1, the VDD power supply 3.5 ~ 5.5V DC
- 2 DATA serial data, a single bus
- 3, NC, empty pin
- 4, GND ground, the negative power

6、Typical circuit



Note: 3Pin – Null; MCU = Micro-computer Unite or single chip Computer
When the connecting cable is shorter than 20 metres, a 5K pull-up resistor is recommended; when the connecting cable is longer than 20 metres, choose a appropriate pull-up resistor as needed.

7、Serial communication instructions (single-wire bi-directional)

7-1 Single bus Description
DHT11 uses a simplified single-bus communication. Single bus that only one data line, the system of data exchange, control by a single bus to complete. Device (master or slave) through an open-drain or tri-state port connected to the data line to allow the device does not send data to release the bus, while other devices use the bus; single bus usually require an external one about 5.1kΩ pull-up resistor,

so that when the bus is idle, its status is high. Because they are the master-slave structure, and only when the host calls the slave, the slave can answer, the host access devices must strictly follow the single-bus sequence, if the chaotic sequence, the device will not respond to the host.

7-2 Single bus to transfer data defined
DATA For communication and synchronization between the microprocessor and DHT11, single-bus data format, a transmission of 40 data, the high first-out.

Data format:
The 8bit humidity integer data + 8bit the Humidity decimal data +8 bit temperature integer data + 8bit fractional temperature data +8 bit parity bit.

7-3 Parity bit data definition
"8bit humidity integer data + 8bit humidity decimal data +8 bit temperature integer data + 8bit temperature fractional data" 8bit checksum is equal to the results of the last eight.
Example 1: 40 data is received:

0011 0101	0000 0000	0001 1000	0000 0000	0100 1101
High humidity 8	Low humidity 8	High temp. 8	Low temp. 8	Parity bit

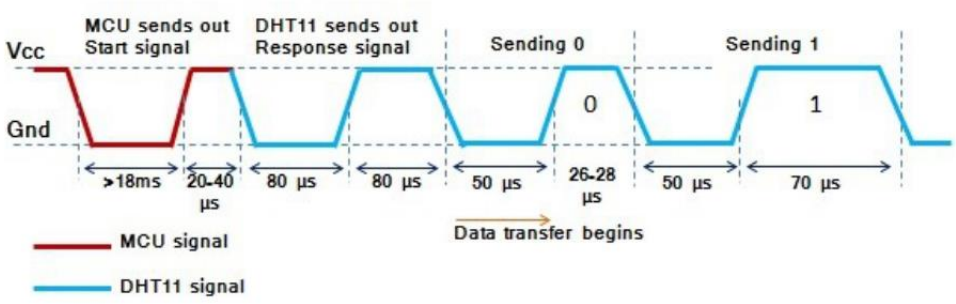
Calculate :
0011 0101+0000 0000+0001 1000+0000 0000= 0100 1101
Received data is correct :
Humidity : 0011 0101=35H=53%RH
Temperature : 0001 1000=18H=24℃

Example 2: 40 data is received:

0011 0101	0000 0000	0001 1000	0000 0000	0100 1001
High humidity 8	Low humidity 8	High temp. 8	Low temp. 8	Parity bit

Calculate :
0011 0101+0000 0000+0001 1000+0000 0000 = 0100 1101
01001101≠0100 1001
The received data is not correct, give up, to re-receive data.

7-4 Data Timing Diagram
User host (MCU) to send a signal, DHT11 converted from low-power mode to high-speed mode, until the host began to signal the end of the DHT11 send a response signal to send 40bit data, and trigger a letter collection. The signal is sent as shown.

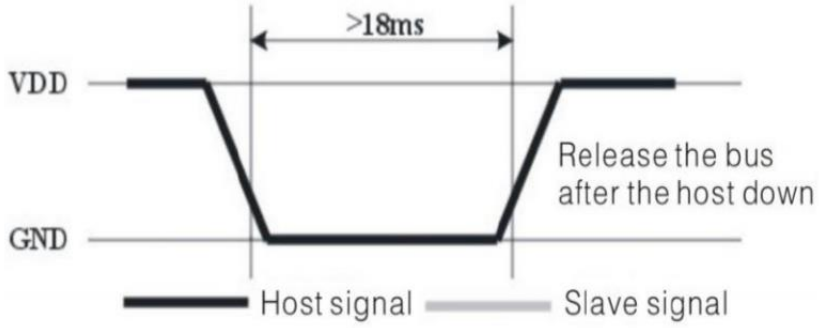


Data Timing Diagram
Note: The host reads the temperature and humidity data from DHT11 always the last measured value, such as twice the measured interval of time is very long, continuous read twice to the second value of real-time temperature and humidity values.

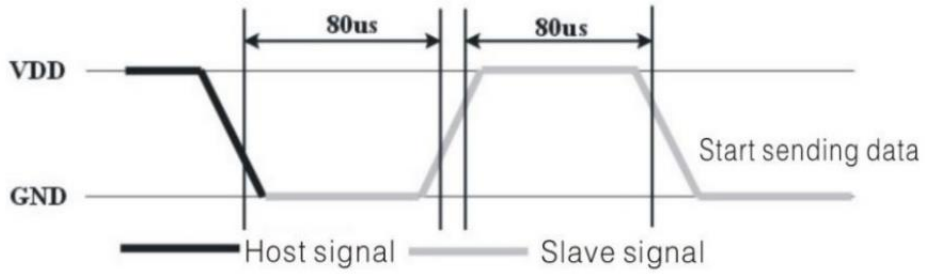
7-5 Peripherals read steps
Communication between the master and slave can be done through the following steps (peripherals (such as microprocessors) read DHT11 the data of steps).

Step 1:
After power on DHT11 (DHT11 on after power to wait 1S across the unstable state during this period can not send any instruction), the test environment temperature and humidity data, and record the data, while DHT11 the DATA data lines pulled by pull-up resistor has been to maintainhigh; the DHT11 the DATA pin is in input state, the moment of detection of external signals.

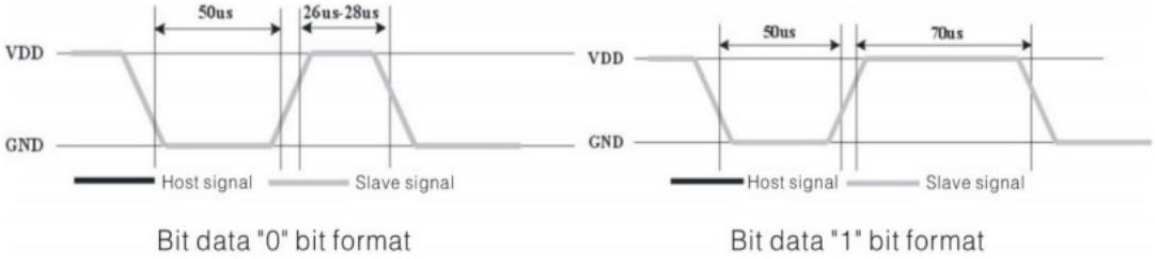
Step 2:
Microprocessor I / O set to output at the same time output low, and low hold time can not be less than 18ms, then the microprocessor I / O is set to input state, due to the pull-up resistor, a microprocessor/ O DHT11 the dATA data lines also will be high, waiting DHT11 to answer signal, send the signal as shown:



Step 3:
DATA pin is detected to an external signal of DHT11 low, waiting for external signal low end the delay DHT11 DATA pin in the output state, the output low of 80 microseconds as the response signal, followed by the output of 80 micro-seconds of high notification peripheral is ready to receive data, the microprocessor I / O at this time in the input state is detected the I / O low (DHT11 response signal), wait 80 microseconds highdata receiving and sending signals as shown:



Step 4:
Output by DHT11 the DATA pin 40, the microprocessor receives 40 data bits of data "0" format: the low level of 50 microseconds and 26-28 microseconds according to the changes in the I / O levellevel, bit data "1" format: the high level of low plus, 50 microseconds to 70 microseconds. Bit data "0", "1" signal format as shown:



Bit data "0" bit format

Bit data "1" bit format

End signal:
Continue to output the low 50 microseconds after DHT11 the DATA pin output 40 data, and changed the input state, along with pull-up resistor goes high. But DHT11 internal re-test environmental temperature and humidity data, and record the data, waiting for the arrival of the external signal.

8、Application of information

8-1. Work and storage conditions
Outside the sensor the proposed scope of work may lead to temporary drift of the signal up to 300%RH. Return to normal working conditions, sensor calibration status will slowly toward recovery. To speed up the recovery process may refer to "resume processing". Prolonged use of non-normal operating conditions, will accelerate the aging of the product.

Avoid placing the components on the long-term condensation and dry environment, as well as the following environment.

A, salt spray
B, acidic or oxidizing gases such as sulfur dioxide, hydrochloric acid

Recommended storage environment

Temperature: 10 ~ 40 ℃ Humidity: 60% RH or less

8-2. The impact of exposure to chemicals
The capacitive humidity sensor has a layer by chemical vapor interference, the proliferation of chemicals in the sensing layer may lead to drift and decreased sensitivity of the measured values. In a pure environment, contaminants will slowly be released. Resume processing as described below will accelerate this process. The high concentration of chemical pollution (such as ethanol) will lead to the complete damage of the sensitive layer of the sensor.

8-3. The temperature influence
Relative humidity of the gas to a large extent dependent on temperature. Therefore, in the measurement of humidity, should be to ensure that the work of the humidity sensor at the same temperature. With the release of heat of electronic components share a printed circuit board, the installation should be as far as possible the sensor away from the electronic components and mounted below the heat source, while maintaining good ventilation of the enclosure.
To reduce the thermal conductivity sensor and printed circuit board copper plating should be the smallest possible, and leaving a gap between the two.

8-4. Light impact
Prolonged exposure to sunlight or strong ultraviolet radiation, and degrade performance.

8-5. Resume processing
Placed under extreme working conditions or chemical vapor sensor, which allows it to return to the status of calibration by the following handler. Maintain two hours in the humidity conditions of 45℃ and <10% RH (dry); followed by 20-30℃ and> 70% RH humidity conditions to maintain more than five hours.

8-6. Wiring precautions
The quality of the signal wire will affect the quality of the voltage output, it is recommended to use high quality shielded cable.

8-7. Welding information
Manual welding, in the maximum temperature of 300℃ under the conditions of contact time shall be less than 3 seconds.

8-8. Product upgrades
Details, please the consultation Aosong electronics department.

9、The license agreement

Without the prior written permission of the copyright holder, shall not in any form or by any means, electronic or mechanical (including photocopying), copy any part of this manual, nor shall its contents be communicated to a third party. The contents are subject to change without notice.

The Company and third parties have ownership of the software, the user may use only signed a contract or software license.

10、Warnings and personal injury

This product is not applied to the safety or emergency stop devices, as well as the failure of the product may result in injury to any other application, unless a particular purpose or use authorized. Installation, handling, use or maintenance of the product refer to product data sheets and application notes. Failure to comply with this recommendation may result in death and serious personal injury. The Company will bear all damages resulting personal injury or death, and waive any claims that the resulting subsidiary company managers and employees and agents, distributors, etc. that may arise, including: a variety of costs, compensation costs, attorneys' fees, and so on.

11、Quality Assurance

The company and its direct purchaser of the product quality guarantee period of three months (from the date of delivery). Publishes the technical specifications of the product data sheet shall prevail. Within the warranty period, the product was confirmed that the quality is really defective, the company will provide free repair or replacement. The user must satisfy the following conditions:

1. The product is found defective within 14 days written notice to the Company;
2. The product shall be paid by mail back to the company;
3. The product should be within the warranty period.

The Company is only responsible for those used in the occasion of the technical condition of the product defective product. Without any guarantee, warranty or written statement of its products used in special applications.

Company for its products applied to the reliability of the product or circuit does not make any commitment.

9.2. Código

i. Arduino – Main

```
//Libraries
#include <DHT11.h>

//Global variables
byte inByte;
int samplingPeriod;
byte MqSensorsData[32];

void setup() {
  Serial_Port_Init();
  MQ_Sensors_Init();
  Get_Sampling_Period();
  Establish_Contact();
}

void loop() {
  uint32_t ts1 = millis(); //Start counting time
  Read_MQ_Sensors(); //Read values from MQ sensors
  Read_DHT11_Sensors(); //Read values from DHT sensors

  //When Processing is ready to receive the data, Arduino sends it
  if (Serial.available() > 0) {
    inByte = Serial.read();
    if (inByte == 'A') {
      Send_Data();
    }
  }
  uint32_t ts2 = millis(); //Read how long communication has taken
  delay(samplingPeriod - (ts2 - ts1)); //Wait until the sampling period is met
}
```

ii. Arduino – Process

```
//Global variables
byte pinDHT1 = 2, pinDHT2 = 3;
float temp1, temp2, hum1, hum2;
byte DhtSensorsData[4];
int sensorsPinsMask[16] = {15,14,13,12,10,9,11,3,4,5,6,7,1,2,0,8}; //Used to read the sensors in the correct order

void MQ_Sensors_Init(void) {
  for(int i=0;i<16;i++) pinMode(i,INPUT);
}

void Read_MQ_Sensors(void) {
  //Variables
  uint16_t analogValue[16];
  int tensionValue[16];

  for(int i=0;i<16;i++) { //Read the value of the sensors (0 - 1023)
    analogValue[i] = analogRead(sensorsPinsMask[i]);
    delay(10);
  }

  for(int i=0;i<16;i++) { //Convert the analog value to voltage (mV) with a resolution of 5 mV
    tensionValue[i] = (5.0/1023.0)*analogValue[i]*1000;
    if((tensionValue[i]%10)<5) {
      tensionValue[i] = tensionValue[i]/10;
      tensionValue[i] = tensionValue[i]*10;
  }
    else {
      tensionValue[i] = tensionValue[i]/10;
      tensionValue[i] = tensionValue[i]*10;
      tensionValue[i] += 5;
    }
  }

  for(int i=0;i<16;i++) { //Split every integer value in two bytes and save them into an array
    MqSensorsData[2*i] = tensionValue[i]&0xFF;
    MqSensorsData[2*i+1] = (tensionValue[i]>>8)&0xFF;
```

```
  }
}

void Read_DHT11_Sensors (void) {
  //Initialize DHT Sensors
  DHT11 dht1(pinDHT1);
  DHT11 dht2(pinDHT2);

  //Read temperature and humidity
  dht1.read(hum1,temp1);
  dht2.read(hum2,temp2);

  //Save the values into an array
  DhtSensorsData[0] = hum1;
  DhtSensorsData[1] = temp1;
  DhtSensorsData[2] = hum2;
  DhtSensorsData[3] = temp2;
}

```

iii. Arduino – SerialCommunication

```
void Serial_Port_Init(void) {
  Serial.begin(9600);
  while (!Serial) {}
}

void Get_Sampling_Period (void) {
  //Variables
  boolean receivedSamplingPeriod = false;
  int arraySamplingPeriod[3];

  //While sampling period has not been received yet
  do {
    //Ask Processing to send the sampling period
    while (Serial.available() <= 0) {
      Serial.write('T');
      delay(300);
    }

    //Read the sampling period
    inByte = Serial.read();
    if (inByte == 'T') {
      //Read the three bytes of the sampling period
      for (int i = 0; i < 3; i++) {
        while (Serial.available() <= 0) {}
        arraySamplingPeriod[i] = Serial.read();
      }

      //Join the bytes into an integer variable
      samplingPeriod = arraySamplingPeriod[0] | arraySamplingPeriod[1] << 8 |
arraySamplingPeriod[2] << 16;
      receivedSamplingPeriod = true;
    }
  } while (!receivedSamplingPeriod);
}

void Establish_Contact(void) { //Establish contact with Processing to start sending
the sensors data
  while (Serial.available() <= 0) {
    Serial.write('A');
    delay(300);
  }
}

void Send_Data (void) {
  for (int i = 0; i < 32; i++) {
    Serial.write(MqSensorsData[i]);
  }
  for (int i = 0; i < 4; i++) {
    Serial.write(DhtSensorsData[i]);
  }
}

```

iv. Processing – Main

```
//Libraries
import processing.serial.*;
import g4p_controls.*;
import peasy.*;

public void setup(){
  //GUI
  size(800, 600, JAVA2D); //Define dimensions and renderer of the display
  initGUI();
  //Serial Communication
  while(!establishedSamplingPeriod) {delay(10);} //Wait until user introduces a sampling period
  initSerialPort();
  //Data
  initData();
  sendSamplingPeriod(); //Send the sampling period to Arduino
}

public void draw(){
  drawActiveMenu();
}

void serialEvent (Serial myPort) {
  if(SentSamplingPeriod) { //Sampling period has already been sent to Arduino
    readInputByte();
    if(firstContact == false) {
      establishFirstContact();
    }
    else {
      readInputArray();
      if(serialCount>35) {
        saveData();
        endCommunication();
        insertDataIntoTable(allDataRead);
        if(mustSaveData) insertDataIntoTable(dataToSave);
      }
    }
  }
}

```

v. Processing – Data

```
//Global variables
boolean mustSaveData = false;
boolean establishedSamplingPeriod = false;
int samplingPeriod;

int mq2_1, mq3_1, mq4_1, mq5_1, mq6_1, mq8_1, mq135_1;
int mq2_2, mq3_2, mq4_2, mq5_2, mq6_2, mq8_2, mq135_2;
int mq2_3, mq135_3;
int hum1, temp1, hum2, temp2;

//Data tables
Table allDataRead = new Table();
Table dataToSave = new Table();

void initData() { //Create both tables to save sensors data
  defineTable(allDataRead);
  defineTable(dataToSave);
}

void defineTable(Table table) {  //Create a new table to save sensors data
  table.addColumn("instant");
  table.addColumn("MQ2 (1)");
  table.addColumn("MQ2 (2)");
  table.addColumn("MQ2 (3)");
  table.addColumn("MQ3 (1)");
  table.addColumn("MQ3 (2)");
  table.addColumn("MQ4 (1)");
  table.addColumn("MQ4 (2)");
  table.addColumn("MQ5 (1)");
  table.addColumn("MQ5 (2)");
```

```
31    table.addColumn("MQ6 (1)");
32    table.addColumn("MQ6 (2)");
33    table.addColumn("MQ8 (1)");
34    table.addColumn("MQ8 (2)");
35    table.addColumn("MQ135 (1)");
36    table.addColumn("MQ135 (2)");
37    table.addColumn("MQ135 (3)");
38
39    table.addColumn("Temperature 1");
40    table.addColumn("Temperature 2");
41    table.addColumn("Humidity 1");
42    table.addColumn("Humidity 2");
43  }
44
45  void insertDataIntoTable(Table table) { //Insert and save current data into a table
46    TableRow newRow = table.addRow();
47    newRow.setInt("instant", table.getRowCount() - 1);
48
49    newRow.setInt("MQ2 (1)",mq2_1);
50    newRow.setInt("MQ2 (2)",mq2_2);
51    newRow.setInt("MQ2 (3)",mq2_3);
52    newRow.setInt("MQ3 (1)",mq3_1);
53    newRow.setInt("MQ3 (2)",mq3_2);
54    newRow.setInt("MQ4 (1)",mq4_1);
55    newRow.setInt("MQ4 (2)",mq4_2);
56    newRow.setInt("MQ5 (1)",mq5_1);
57    newRow.setInt("MQ5 (2)",mq5_2);
58    newRow.setInt("MQ6 (1)",mq6_1);
59    newRow.setInt("MQ6 (2)",mq6_2);
60    newRow.setInt("MQ8 (1)",mq8_1);
61    newRow.setInt("MQ8 (2)",mq8_2);
62    newRow.setInt("MQ135 (1)",mq135_1);
63    newRow.setInt("MQ135 (2)",mq135_2);
64    newRow.setInt("MQ135 (3)",mq135_3);
65
66    newRow.setInt("Temperature 1", temp1);
67    newRow.setInt("Temperature 2", temp2);
68    newRow.setInt("Humidity 1", hum1);
69    newRow.setInt("Humidity 2", hum2);
70  }
71
72  void saveTheTable(Table table) { //Export data from a table to a .csv file
73    String fileRoute;
74    String fileName;
75    int d = day();
76    int m = month();
77    int y = year();
78    int h = hour();
79    int mn = minute();
80
81    fileRoute = "data/"+y+"_"+nf(m,2)+"_"+nf(d,2)+"/";
82    fileName = ""+nf(y,4)+"_"+nf(m,2)+"_"+nf(d,2)+" "+nf(h,2)+"h"+nf(mn,2)+"m.csv";
83    saveTable(table, fileRoute+fileName);
84    table.clearRows();
85  }
86
87  void cancelTable(Table table) { //Remove all data from a table
88    table.clearRows();
89  }
90
91  public void saveData() {
92    mq2_1 = serialInArray[0] | (serialInArray[1]<<8);
93    mq3_1 = serialInArray[2] | (serialInArray[3]<<8);
94    mq4_1 = serialInArray[4] | (serialInArray[5]<<8);
95    mq5_1 = serialInArray[6] | (serialInArray[7]<<8);
96    mq6_1 = serialInArray[8] | (serialInArray[9]<<8);
97    mq8_1 = serialInArray[10] | (serialInArray[11]<<8);
98    mq135_1 = serialInArray[12] | (serialInArray[13]<<8);
99
100   mq2_2 = serialInArray[16] | (serialInArray[17]<<8);
101   mq3_2 = serialInArray[18] | (serialInArray[19]<<8);
102   mq4_2 = serialInArray[20] | (serialInArray[21]<<8);
103   mq5_2 = serialInArray[22] | (serialInArray[23]<<8);
104   mq6_2 = serialInArray[24] | (serialInArray[25]<<8);
105   mq8_2 = serialInArray[26] | (serialInArray[27]<<8);
106   mq135_2 = serialInArray[28] | (serialInArray[29]<<8);
107
```

```
108    mq2_3 = serialInArray[14] | (serialInArray[15]<<8);
109    mq135_3 = serialInArray[30] | (serialInArray[31]<<8);
110
111    hum1 = serialInArray[32];
112    temp1 = serialInArray[33];
113    hum2 = serialInArray[34];
114    temp2 = serialInArray[35];
115  }
116
117  void splitSamplingPeriod () {
118    arraySamplingPeriod[0] = samplingPeriod & 0x0000FF;
119    arraySamplingPeriod[1] = (samplingPeriod & 0x00FF00) >> 8;
120    arraySamplingPeriod[2] = (samplingPeriod & 0xFF0000) >> 16;
121  }
122
```

vi. Processing – SerialCommunication

```
1   //Global variables
2   Serial myPort;
3   int inByte;
4   int[] arraySamplingPeriod = new int [3];
5   int[] serialInArray = new int[36];
6   byte serialCount = 0;
7   boolean firstContact = false;
8   boolean SentSamplingPeriod = false;
9
10  public void initSerialPort () { //Initialize serial communication
11    String portName = Serial.list()[0];
12    myPort = new Serial(this, portName, 9600);
13  }
14
15  void sendSamplingPeriod() { //Send the established sampling period to Arduino
16    do {
17      //Wait until Arduino requests the sampling period
18      while(myPort.available() <= 0) {delay(10);} //
19      inByte = myPort.read();
20
21      //When Arduino has requested the sampling period
22      if (inByte == 'T') {
23        myPort.write('T'); //Warn Arduino that the sampling period will be sent
24        delay(100);
25        for(int i=0; i<3; i++) { //Send the sampling period divided in three bytes
26          myPort.write(arraySamplingPeriod[i]);
27        }
28        SentSamplingPeriod = true;
29      }
30    } while (!SentSamplingPeriod);
31  }
32
33  public void readInputByte() {
34      inByte = myPort.read();
35  }
36
37  public void establishFirstContact() { //Get ready to receive sensors data from
Arduino
38    if (inByte == 'A') {
39        myPort.clear();
40        firstContact = true;
41        myPort.write('A');
42    }
43  }
44
45  public void readInputArray() { //Read all the bytes received from Arduino that
include sensors data
46    serialInArray[serialCount] = inByte;
47    serialCount++;
48  }
49
50  public void endCommunication () { //Confirm Arduino that all the data has been
received
51    myPort.write('A');
52    serialCount=0;
53  }
```

vii. Processing – MainGUI

```
//Global variables
int activeMenu = 1;
int sliderValue = 10;

void initGUI () { //Initialize GUI
    createGUI();
}

void drawActiveMenu() { //Draw the active menu interface
  background(230);
  switch (activeMenu) {
    case 1: drawMenuData(); break;
    case 2: drawMenuGraphs(); break;
  }
}

void drawMenuData() { //Draw the data menu interface
  //Hide elements belonging to graphs menu
  showCheckboxes(false);
  showSlider(false);

  //Show elements belonging to data menu
  drawSensorCards();

  //Draw "Export Data" rectangle
  fill(255,255,255);
  rect(400,490,398,58);
  fill(0,0,0);
  textAlign(CENTER,CENTER);
  textSize(20);
  text("Exportar datos",600,515);
}

void drawMenuGraphs() {
  //Show elements belonging to graphs menu
  showCheckboxes(true);
  showSlider(true);

  //Draw graphs
  drawGraphs();
  drawLines();

  stroke(0,0,0);
  strokeWeight(1);
  fill(255,255,255);
  rect(400,490,398,58);
  fill(0,0,0);
  textAlign(CENTER,CENTER);
  textSize(20);
  text("Exportar datos",600,515);

  //Draw export data rectangle
  setExportDataRectStyle();
  rect(400,490,398,58);
  DefaultTextStyle.setStyle();
  text("Exportar datos",600,515);

  //Write slide bar title
  DefaultTextStyle.setStyle();
  text("Escala de tiempo", 690, 355);
}

void showCheckboxes(boolean visibility) {
  checkbox_mq135_1.setVisible(visibility);
  checkbox_mq135_2.setVisible(visibility);
  checkbox_mq135_3.setVisible(visibility);
  checkbox_mq2_1.setVisible(visibility);
  checkbox_mq2_2.setVisible(visibility);
  checkbox_mq2_3.setVisible(visibility);
  checkbox_mq3_1.setVisible(visibility);
  checkbox_mq3_2.setVisible(visibility);
  checkbox_mq4_1.setVisible(visibility);
  checkbox_mq4_2.setVisible(visibility);
  checkbox_mq5_1.setVisible(visibility);
```

```
    checkbox_mq5_2.setVisible(visibility);
    checkbox_mq6_1.setVisible(visibility);
    checkbox_mq6_2.setVisible(visibility);
    checkbox_mq8_1.setVisible(visibility);
    checkbox_mq8_2.setVisible(visibility);

    buttonCheckAll.setVisible(visibility);
    buttonUncheckAll.setVisible(visibility);
  }

  void showSlider(boolean visibility) { //Show or hide the slider
   sliderTime.setVisible(visibility);
  }

  void getSamplingPeriod() { //Read the sampling period introduced by the user
    String textfield = textfieldPeriod.getText();
    try {
      samplingPeriod = Integer.parseInt(textfield);
      if(samplingPeriod<500 || samplingPeriod > 100000) {
        labelError.setVisible(true);
      }
      else {
        WindowSamplingPeriod.forceClose();
        establishedSamplingPeriod = true;
      }
    }
    catch (NumberFormatException e){
      labelError.setVisible(true);
    }
    splitSamplingPeriod();
  }
```

viii. Processing – ClassGraph

```
//Legend
int legendWidth = 5;
int legendHeight = 20;

int legendPx_mq135_1 = 600; int legendPy_mq135_1 = 55;
int legendPx_mq135_2 = 600; int legendPy_mq135_2 = 85;
int legendPx_mq135_3 = 600; int legendPy_mq135_3 = 115;

int legendPx_mq2_1 = 600; int legendPy_mq2_1 = 145;
int legendPx_mq2_2 = 600; int legendPy_mq2_2 = 175;
int legendPx_mq2_3 = 600; int legendPy_mq2_3 = 205;

int legendPx_mq3_1 = 600; int legendPy_mq3_1 = 235;
int legendPx_mq3_2 = 600; int legendPy_mq3_2 = 265;

int legendPx_mq4_1 = 695; int legendPy_mq4_1 = 55;
int legendPx_mq4_2 = 695; int legendPy_mq4_2 = 85;

int legendPx_mq5_1 = 695; int legendPy_mq5_1 = 115;
int legendPx_mq5_2 = 695; int legendPy_mq5_2 = 145;

int legendPx_mq6_1 = 695; int legendPy_mq6_1 = 175;
int legendPx_mq6_2 = 695; int legendPy_mq6_2 = 205;

int legendPx_mq8_1 = 695; int legendPy_mq8_1 = 235;
int legendPx_mq8_2 = 695; int legendPy_mq8_2 = 265;

//MQ Graph
int mqPox = 30;
int mqPoy = 280;
int mqGraphWidth = 540;
int mqGraphHeight = 270;
int MQUpperLimit = 5500;
int MQLowerLimit = 0;
String MQGraphUnits = "mV";

//Temperature Graph
int tempPox = 30;
int tempPoy = 460;
int tempGraphWidth = 240;
int tempGraphHeight = 140;
```

```
42  int TempUpperLimit = 55;
43  int TempLowerLimit = 0;
44  String TempGraphUnits = "ยบC";
45
46  //Humidity Graph
47  int humPox = 330;
48  int humPoy = 460;
49  int humGraphWidth = 240;
50  int humGraphHeight = 140;
51  int HumUpperLimit = 110;
52  int HumLowerLimit = 0;
53  String HumGraphUnits = "%";
54
55  //Column of every sensor in the data tables
56  int columnMq2_1 = 1;
57  int columnMq2_2 = 2;
58  int columnMq2_3 = 3;
59  int columnMq3_1 = 4;
60  int columnMq3_2 = 5;
61  int columnMq4_1 = 6;
62  int columnMq4_2 = 7;
63  int columnMq5_1 = 8;
64  int columnMq5_2 = 9;
65  int columnMq6_1 = 10;
66  int columnMq6_2 = 11;
67  int columnMq8_1 = 12;
68  int columnMq8_2 = 13;
69  int columnMq135_1 = 14;
70  int columnMq135_2 = 15;
71  int columnMq135_3 = 16;
72
73  //Initialize Graphs
74  Graph MQGraph = new Graph (mqPox, mqPoy, mqGraphWidth, mqGraphHeight, MQGraphUnits,
MQUpperLimit, MQLowerLimit);
75  Graph TempGraph = new Graph(tempPox, tempPoy, tempGraphWidth, tempGraphHeight,
TempGraphUnits, TempUpperLimit, TempLowerLimit);
76  Graph HumGraph = new Graph(humPox, humPoy, humGraphWidth, humGraphHeight,
HumGraphUnits, HumUpperLimit, HumLowerLimit);
77
78  //Initialize lines
79  GraphLine MQ135_1_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq135_1, legendPy_mq135_1,
allDataRead, columnMq135_1, mq135_1_Style);
80  GraphLine MQ135_2_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq135_2, legendPy_mq135_2,
allDataRead, columnMq135_2, mq135_2_Style);
81  GraphLine MQ135_3_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq135_3, legendPy_mq135_3,
allDataRead, columnMq135_3, mq135_3_Style);
82
83  GraphLine MQ2_1_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq2_1, legendPy_mq2_1, allDataRead,
columnMq2_1, mq2_1_Style);
84  GraphLine MQ2_2_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq2_2, legendPy_mq2_2, allDataRead,
columnMq2_2, mq2_2_Style);
85  GraphLine MQ2_3_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq2_3, legendPy_mq2_3, allDataRead,
columnMq2_3, mq2_3_Style);
86
87  GraphLine MQ3_1_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq3_1, legendPy_mq3_1, allDataRead,
columnMq3_1, mq3_1_Style);
88  GraphLine MQ3_2_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq3_2, legendPy_mq3_2, allDataRead,
columnMq3_2, mq3_2_Style);
89
90  GraphLine MQ4_1_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq4_1, legendPy_mq4_1, allDataRead,
columnMq4_1, mq4_1_Style);
91  GraphLine MQ4_2_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq4_2, legendPy_mq4_2, allDataRead,
columnMq4_2, mq4_2_Style);
92
93  GraphLine MQ5_1_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq5_1, legendPy_mq5_1, allDataRead,
columnMq5_1, mq5_1_Style);
```

```
94  GraphLine MQ5_2_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq5_2, legendPy_mq5_2, allDataRead,
columnMq5_2, mq5_2_Style);
95
96  GraphLine MQ6_1_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq6_1, legendPy_mq6_1, allDataRead,
columnMq6_1, mq6_1_Style);
97  GraphLine MQ6_2_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq6_2, legendPy_mq6_2, allDataRead,
columnMq6_2, mq6_2_Style);
98
99  GraphLine MQ8_1_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq8_1, legendPy_mq8_1, allDataRead,
columnMq8_1, mq8_1_Style);
100 GraphLine MQ8_2_Line = new GraphLine (mqPox, mqPoy, mqGraphWidth, mqGraphHeight,
MQGraphUnits, MQUpperLimit, MQLowerLimit, legendPx_mq8_2, legendPy_mq8_2, allDataRead,
columnMq8_2, mq8_2_Style);
101
102 GraphLine Temp1_Line = new GraphLine (tempPox, tempPoy, tempGraphWidth,
tempGraphHeight, TempGraphUnits, TempUpperLimit, TempLowerLimit, 0, 0, allDataRead, 17,
DHT1_Style);
103 GraphLine Temp2_Line = new GraphLine (tempPox, tempPoy, tempGraphWidth,
tempGraphHeight, TempGraphUnits, TempUpperLimit, TempLowerLimit, 0, 0, allDataRead, 18,
DHT2_Style);
104 GraphLine Hum1_Line = new GraphLine (humPox, humPoy, humGraphWidth, humGraphHeight,
HumGraphUnits, HumUpperLimit, HumLowerLimit, 0, 0, allDataRead, 19, DHT1_Style);
105 GraphLine Hum2_Line = new GraphLine (humPox, humPoy, humGraphWidth, humGraphHeight,
HumGraphUnits, HumUpperLimit, HumLowerLimit, 0, 0, allDataRead, 20, DHT2_Style);
106
107
108 //Graph classes
109 public class Graph {
110   protected int OX;
111   protected int OY;
112   protected int graphWidth;
113   protected int graphHeight;
114   protected String units;
115   protected int upperLimit;
116   protected int lowerLimit;
117
118   public Graph() {
119   }
120
121   Graph(int OX, int OY, int graphWidth, int graphHeight, String units, int
upperLimit, int lowerLimit) { //Constructor
122     this.OX = OX;
123     this.OY = OY;
124     this.graphWidth = graphWidth;
125     this.graphHeight = graphHeight;
126     this.units = units;
127     this.upperLimit = upperLimit;
128     this.lowerLimit = lowerLimit;
129   }
130
131   public void drawGraph() {
132     drawAxis();
133     writeAxis();
134   }
135
136   private void drawAxis() {
137     strokeWeight(graphAxisStroke);
138     stroke(colorBlack);
139     line(OX,OY,OX+graphWidth,OY); //X Axis
140     line(OX,OY,OX,OY-graphHeight); //Y Axis
141
142     //X Axis
143     for(int i=0; i<sliderValue; i++) {
144       line(OX+i*graphWidth/sliderValue, OY+2, OX+i*graphWidth/sliderValue, OY-2);
145     }
146     MQGraphTextStyle.setStyle();
147     text("t (s)",OX+graphWidth, OY+10);
148
149     //Y Axis
150     MQGraphTextStyle.setStyle();
151     text(units,OX-10,OY-graphHeight);
152     for(int i=0; i<11; i++) {
153       stroke(colorBlack);
```

```
154       line(OX-3, OY-i*graphHeight/11, OX+3, OY-i*graphHeight/11);
155       stroke(200,200,200);
156       if(i!=0) {
157       line(OX+3, OY-i*graphHeight/11, OX+graphWidth, OY-i*graphHeight/11);
158       }
159     }
160     stroke(colorBlack);
161   }
162
163   private void writeAxis() {
164     //Y Axis
165     for(int i=1;i<11;i++) {
166       MQGraphTextStyle.setStyle();
167       textAlign(RIGHT,CENTER);
168       text(i*(upperLimit-lowerLimit)/11, OX-5, OY-i*graphHeight/11);
169     }
170     //X Axis
171     //10
172     if(sliderValue < 15) {
173       for(int i=0; i<10; i++) {
174         line(OX+i*graphWidth/10,OY+5,OX+i*graphWidth/10,OY-5);
175         MQGraphTextStyle.setStyle();
176         if(!(graphHeight<200 && i%2==0)) {
177           text("+"+(float(i+1)*samplingPeriod/1000.0), OX+graphWidth-
(i+1)*graphWidth/10, OY+10);
178         }
179       }
180     }
181     //20
182     else if (sliderValue >= 15 && sliderValue < 25) {
183       for(int i=0; i<10; i++) {
184         line(OX+i*graphWidth/10,OY+5,OX+i*graphWidth/10,OY-5);
185         MQGraphTextStyle.setStyle();
186         if(!(graphHeight<200 && i%2==0)) {
187           text("+"+(float(i+1)*2.0*samplingPeriod/1000.0), OX+graphWidth-
(i+1)*graphWidth/10, OY+10);
188         }
189       }
190     }
191     //30
192     else if (sliderValue >= 25 && sliderValue < 35) {
193       for(int i=0; i<6; i++) {
194         line(OX+i*graphWidth/6,OY+5,OX+i*graphWidth/6,OY-5);
195         MQGraphTextStyle.setStyle();
196         if(!(graphHeight<200 && i%2==0)) {
197           text("+"+(float(i+1)*5.0*samplingPeriod/1000.0), OX+graphWidth-
(i+1)*graphWidth/6, OY+10);
198         }
199       }
200     }
201     //40
202     else if (sliderValue >= 35 && sliderValue < 45) {
203       for(int i=0; i<8; i++) {
204         line(OX+i*graphWidth/8,OY+5,OX+i*graphWidth/8,OY-5);
205         MQGraphTextStyle.setStyle();
206         if(!(graphHeight<200 && i%2==0)) {
207           text("+"+(float(i+1)*5.0*samplingPeriod/1000.0), OX+graphWidth-
(i+1)*graphWidth/8, OY+10);
208         }
209       }
210     }
211     //50
212     else if (sliderValue >= 45 && sliderValue < 55) {
213       for(int i=0; i<10; i++) {
214         line(OX+i*graphWidth/10,OY+5,OX+i*graphWidth/10,OY-5);
215         MQGraphTextStyle.setStyle();
216         if(!(graphHeight<200 && i%2==0)) {
217           text("+"+(float(i+1)*5.0*samplingPeriod/1000.0), OX+graphWidth-
(i+1)*graphWidth/10, OY+10);
218         }
219       }
220     }
221     //60
222     else if (sliderValue >= 55) {
223       for(int i=0; i<12; i++) {
224         line(OX+i*graphWidth/12,OY+5,OX+i*graphWidth/12,OY-5);
225         MQGraphTextStyle.setStyle();
```

```
226         if(!(graphHeight<200 && i%2==0)) {
227           text("+"+(float(i+1)*5.0*samplingPeriod/1000.0), OX+graphWidth-
(i+1)*graphWidth/12, OY+10);
228         }
229       }
230     }
231   }
232
233 }
234
235 //Class Line inherited from class Graph
236 public class GraphLine extends Graph {
237   int legendPx;
238   int legendPy;
239   Table dataTable;
240   int sensorID;
241   GraphStyle MyStyle;
242
243   public GraphLine() { //Superconstructor
244     super();
245   }
246
247   //Constructor
248   GraphLine(int OX, int OY, int graphWidth, int graphHeight, String units, int
upperLimit, int lowerLimit, int legendPx, int legendPy, Table dataTable, int sensorID,
GraphStyle MyStyle) {
249     super(OX, OY, graphWidth, graphHeight, units, upperLimit, lowerLimit);
250     this.legendPx = legendPx;
251     this.legendPy = legendPy;
252     this.dataTable = dataTable;
253     this.sensorID = sensorID;
254     this.MyStyle = MyStyle;
255   }
256
257   public void drawLine() {
258     drawPoints();
259     drawConnectors();
260   }
261
262    private void drawLegend() {
263       MyStyle.setLegendStyle();
264       rect(legendPx, legendPy, legendWidth, legendHeight);
265   }
266
267   private void drawPoints() {
268     int tableNumRows = dataTable.getRowCount();
269     for (int i = 0; i<sliderValue; i++) {
270       int index = tableNumRows - (i+1);
271       int currentValue = 0;
272       if(index >= 0) {
273         currentValue = dataTable.getInt(index, sensorID);
274         int px = OX+graphWidth-i*graphWidth/sliderValue;
275         int py = OY-(currentValue-lowerLimit)*graphHeight/(upperLimit-lowerLimit);
276         MyStyle.setPointStyle();
277         point(px,py);
278       }
279     }
280   }
281
282   private void drawConnectors() {
283     int tableNumRows = dataTable.getRowCount();
284     for (int i = 0; i<sliderValue; i++) {
285       int index = tableNumRows - (i+1);
286       int currentValue = 0, previousValue = 0;
287       if(index > 0) {
288         currentValue = dataTable.getInt(index, sensorID);
289         previousValue = dataTable.getInt(index-1, sensorID);
290         int px = OX+graphWidth-i*graphWidth/sliderValue;
291         int py = OY-(currentValue-lowerLimit)*graphHeight/(upperLimit-lowerLimit);
292         int pox = OX+graphWidth-(i+1)*graphWidth/sliderValue;
293         int poy = OY-(previousValue-lowerLimit)*graphHeight/(upperLimit-lowerLimit);
294         MyStyle.setLineStyle();
295         line(px,py,pox,poy);
296       }
297     }
298   }
299 }
```

```
300
301 //Functions
302 void drawLegend() { //Show legend of the MQ Sensors data graph
303   MQ135_1_Line.drawLegend();
304   MQ135_2_Line.drawLegend();
305   MQ135_3_Line.drawLegend();
306   MQ2_1_Line.drawLegend();
307   MQ2_2_Line.drawLegend();
308   MQ2_3_Line.drawLegend();
309   MQ3_1_Line.drawLegend();
310   MQ3_2_Line.drawLegend();
311   MQ4_1_Line.drawLegend();
312   MQ4_2_Line.drawLegend();
313   MQ5_1_Line.drawLegend();
314   MQ5_2_Line.drawLegend();
315   MQ6_1_Line.drawLegend();
316   MQ6_2_Line.drawLegend();
317   MQ8_1_Line.drawLegend();
318   MQ8_2_Line.drawLegend();
319 }
320
321 public void drawGraphs() { //Draw graphs axis and legend
322   MQGraph.drawGraph();
323   TempGraph.drawGraph();
324   HumGraph.drawGraph();
325   drawLegend();
326 }
327
328 public void drawLines() { //Draw sensors lines in the graphs
329   if(checkbox_mq135_1.isSelected() == true) MQ135_1_Line.drawLine();
330   if(checkbox_mq135_2.isSelected() == true) MQ135_2_Line.drawLine();
331   if(checkbox_mq135_3.isSelected() == true) MQ135_3_Line.drawLine();
332
333   if(checkbox_mq2_1.isSelected() == true) MQ2_1_Line.drawLine();
334   if(checkbox_mq2_2.isSelected() == true) MQ2_2_Line.drawLine();
335   if(checkbox_mq2_3.isSelected() == true) MQ2_3_Line.drawLine();
336
337   if(checkbox_mq3_1.isSelected() == true) MQ3_1_Line.drawLine();
338   if(checkbox_mq3_2.isSelected() == true) MQ3_2_Line.drawLine();
339
340   if(checkbox_mq4_1.isSelected() == true) MQ4_1_Line.drawLine();
341   if(checkbox_mq4_2.isSelected() == true) MQ4_2_Line.drawLine();
342
343   if(checkbox_mq5_1.isSelected() == true) MQ5_1_Line.drawLine();
344   if(checkbox_mq5_2.isSelected() == true) MQ5_2_Line.drawLine();
345
346   if(checkbox_mq6_1.isSelected() == true) MQ6_1_Line.drawLine();
347   if(checkbox_mq6_2.isSelected() == true) MQ6_2_Line.drawLine();
348
349   if(checkbox_mq8_1.isSelected() == true) MQ8_1_Line.drawLine();
350   if(checkbox_mq8_2.isSelected() == true) MQ8_2_Line.drawLine();
351
352   Temp1_Line.drawLine();
353   Temp2_Line.drawLine();
354
355   Hum1_Line.drawLine();
356   Hum2_Line.drawLine();
357
358 }
359
```

ix. Processing – ClassSensorCard

```
1   //Initialize sensor cards
2   MQSensorCard SensorCardMQ135_1 = new MQSensorCard("MQ135", 1, 110, 10);
3   MQSensorCard SensorCardMQ135_2 = new MQSensorCard("MQ135", 2, 310, 10);
4   MQSensorCard SensorCardMQ135_3 = new MQSensorCard("MQ135", 3, 510, 10);
5   MQSensorCard SensorCardMQ2_1   = new MQSensorCard("MQ2", 1, 110, 80);
6   MQSensorCard SensorCardMQ2_2   = new MQSensorCard("MQ2", 2, 310, 80);
7   MQSensorCard SensorCardMQ2_3   = new MQSensorCard("MQ2", 3, 510, 80);
8   MQSensorCard SensorCardMQ3_1   = new MQSensorCard("MQ3", 1, 110, 150);
9   MQSensorCard SensorCardMQ3_2   = new MQSensorCard("MQ3", 2, 310, 150);
10  MQSensorCard SensorCardMQ4_1   = new MQSensorCard("MQ4", 1, 110, 220);
11  MQSensorCard SensorCardMQ4_2   = new MQSensorCard("MQ4", 2, 310, 220);
12  MQSensorCard SensorCardMQ5_1   = new MQSensorCard("MQ5", 1, 110, 290);
13  MQSensorCard SensorCardMQ5_2   = new MQSensorCard("MQ5", 2, 310, 290);
```

```
14  MQSensorCard SensorCardMQ6_1   = new MQSensorCard("MQ6", 1, 110, 360);
15  MQSensorCard SensorCardMQ6_2   = new MQSensorCard("MQ6", 2, 310, 360);
16  MQSensorCard SensorCardMQ8_1   = new MQSensorCard("MQ8", 1, 110, 430);
17  MQSensorCard SensorCardMQ8_2   = new MQSensorCard("MQ8", 2, 310, 430);
18
19  DHTSensorCard sensorDHT_T1     = new DHTSensorCard("Temperatura", true, 510, 150);
20  DHTSensorCard sensorDHT_T2     = new DHTSensorCard("Temperatura", true, 510, 238);
21  DHTSensorCard sensorDHT_H1     = new DHTSensorCard("Humedad", false, 510, 325);
22  DHTSensorCard sensorDHT_H2     = new DHTSensorCard("Humedad", false, 510, 413);
23
24  //Classes
25  private class MQSensorCard {
26    String sensor;
27    int row;
28    int xpos;
29    int ypos;
30
31    MQSensorCard(String sensor, int row, int xpos, int ypos) { //Constructor
32      this.sensor = sensor;
33      this.row = row;
34      this.xpos = xpos;
35      this.ypos = ypos;
36    }
37
38    public void drawCard (int reading) {
39      boolean failure;
40      String estado;
41
42      //Check if there is a problem when reading the sensor
43      if (reading == 0 || reading >= 5000) failure = true;
44      else failure = false;
45
46      if(failure) {
47        estado = "Incorrecto";
48        SensorCardIncorrectStyle.setMainStyle();
49        rect(xpos, ypos, 75, 50);
50        SensorCardIncorrectStyle.setSecondaryStyle();
51        rect(xpos+75, ypos, 105,50);
52      }
53      else {
54        estado = "Correcto";
55        SensorCardCorrectStyle.setMainStyle();
56        rect(xpos, ypos, 75, 50);
57        SensorCardCorrectStyle.setSecondaryStyle();
58        rect(xpos+75, ypos, 105,50);
59      }
60
61      fill(colorBlack);
62      textSize(18);
63      textAlign(CENTER,CENTER);
64      text(sensor,xpos+75/2, ypos+25);
65
66      fill(colorBlack);
67      textSize(12);
68      textAlign(RIGHT,TOP);
69      text("("+row+")", xpos+75, ypos);
70
71      textSize(12);
72      textAlign(CENTER,CENTER);
73      text("Valor: "+reading+" mV", xpos+75+105/2,ypos+15);
74      text("Estado: "+estado, xpos+75+105/2, ypos+35);
75    }
76  }
77
78  private class DHTSensorCard {
79    String title;
80    boolean magnitudeReading; //True = Temperature, False = Humidity
81    int xpos;
82    int ypos;
83
84    DHTSensorCard(String title, boolean magnitudeReading, int xpos, int ypos) {
//Constructor
85      this.title = title;
86      this.magnitudeReading = magnitudeReading;
87      this.xpos = xpos;
88      this.ypos = ypos;
89    }
```

```
90
91    public void drawCard(int reading) {
92      String estado;
93      String textString = "";
94      boolean failure;
95
96      //Check if there is a problem when reading the sensor
97      if((magnitudeReading && (reading < 0)) || (magnitudeReading && (reading > 50)))
{ //If reading temperature
98        failure = true;
99      }
100     else if((!magnitudeReading && (reading < 20)) || (!magnitudeReading && (reading
> 90))) { //If reading humidity
101       failure = true;
102     }
103     else failure = false;
104
105     if(failure) {
106       estado = "Incorrecto";
107       SensorCardIncorrectStyle.setMainStyle();
108       rect(xpos, ypos, 180, 26);
109       SensorCardIncorrectStyle.setSecondaryStyle();
110       rect(xpos, ypos+26, 180,41);
111     }
112     else {
113       estado = "Correcto";
114       SensorCardCorrectStyle.setMainStyle();
115       rect(xpos, ypos, 180, 26);
116       SensorCardCorrectStyle.setSecondaryStyle();
117       rect(xpos, ypos+26, 180,41);
118     }
119
120     fill(colorBlack);
121     textSize(18);
122     textAlign(CENTER,CENTER);
123     text(title,xpos+90, ypos+13);
124
125     textSize(12);
126     textAlign(CENTER,CENTER);
127     if(title == "Temperatura") {
128       textString = reading + " ยบC";
129     }
130     else if (title == "Humedad") {
131       textString = reading + " %";
132     }
133     text(textString, xpos+90,ypos+13+25);
134     text("Estado: "+estado, xpos+90, ypos+13+40);
135   }
136 }
137
138 //Functions
139 void drawSensorCards () {
140   SensorCardMQ135_1.drawCard(mq135_1);
141   SensorCardMQ135_2.drawCard(mq135_2);
142   SensorCardMQ135_3.drawCard(mq135_3);
143   SensorCardMQ2_1.drawCard(mq2_1);
144   SensorCardMQ2_2.drawCard(mq2_2);
145   SensorCardMQ2_3.drawCard(mq2_3);
146   SensorCardMQ3_1.drawCard(mq3_1);
147   SensorCardMQ3_2.drawCard(mq3_2);
148   SensorCardMQ4_1.drawCard(mq4_1);
149   SensorCardMQ4_2.drawCard(mq4_2);
150   SensorCardMQ5_1.drawCard(mq5_1);
151   SensorCardMQ5_2.drawCard(mq5_2);
152   SensorCardMQ6_1.drawCard(mq6_1);
153   SensorCardMQ6_2.drawCard(mq6_2);
154   SensorCardMQ8_1.drawCard(mq8_1);
155   SensorCardMQ8_2.drawCard(mq8_2);
156
157   sensorDHT_T1.drawCard(temp1);
158   sensorDHT_T2.drawCard(temp2);
159   sensorDHT_H1.drawCard(hum1);
160   sensorDHT_H2.drawCard(hum2);
161 }
```

x. Processing – Styles

```
1   //Strokes
2   int graphAxisStroke = 1;
3   int graphPointStroke = 7;
4   int graphLineStroke = 1;
5   int lineStroke = 1;
6   int legendStrokeWeight = 0;
7   int textMQGraphSize = 10;
8   int textRectSize = 20;
9   //Colors
10  color colorMQ135_1  = #00ffff;
11  color colorMQ135_2  = #0000ff;
12  color colorMQ135_3  = #000080;
13  color colorMQ2_1    = #808000;
14  color colorMQ2_2    = #00ff00;
15  color colorMQ2_3    = #008000;
16  color colorMQ3_1    = #800080;
17  color colorMQ3_2    = #ff00ff;
18  color colorMQ4_1    = #ff0000;
19  color colorMQ4_2    = #800000;
20  color colorMQ5_1    = #ffff00;
21  color colorMQ5_2    = #a0a000;
22  color colorMQ6_1    = #000000;
23  color colorMQ6_2    = #505050;
24  color colorMQ8_1    = #ffc800;
25  color colorMQ8_2    = #ffa300;
26  color colorDHT1     = #ff0000;
27  color colorDHT2     = #0000ff;
28
29  color colorRed         = #FF0000;
30  color colorGreen       = #00FF00;
31  color colorBlack       = #000000;
32  color colorMainGreen   = #32FF32;
33  color colorLightRed   = #FFB2B2;
34  color colorLightGreen = #B2FFB2;
35
36  //Initialize styles
37  GraphStyle mq135_1_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ135_1);
38  GraphStyle mq135_2_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ135_2);
39  GraphStyle mq135_3_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ135_3);
40
41  GraphStyle mq2_1_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ2_1);
42  GraphStyle mq2_2_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ2_2);
43  GraphStyle mq2_3_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ2_3);
44
45  GraphStyle mq3_1_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ3_1);
46  GraphStyle mq3_2_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ3_2);
47
48  GraphStyle mq4_1_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ4_1);
49  GraphStyle mq4_2_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ4_2);
50
51  GraphStyle mq5_1_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ5_1);
52  GraphStyle mq5_2_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ5_2);
53
54  GraphStyle mq6_1_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ6_1);
55  GraphStyle mq6_2_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ6_2);
56
57  GraphStyle mq8_1_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ8 1);
58  GraphStyle mq8_2_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorMQ8_2);
59
60  GraphStyle DHT1_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorDHT1);
```

```
61  GraphStyle DHT2_Style = new GraphStyle(graphLineStroke, graphPointStroke,
colorDHT2);
62
63  CardStyle SensorCardCorrectStyle = new CardStyle(colorMainGreen, colorLightGreen,
colorBlack, lineStroke);
64  CardStyle SensorCardIncorrectStyle = new CardStyle(colorRed, colorLightRed,
colorBlack, lineStroke);
65
66  TextStyle MQGraphTextStyle = new TextStyle(colorBlack, textMQGraphSize);
67  TextStyle DefaultTextStyle = new TextStyle(colorBlack, textRectSize);
68
69  //Style classes
70  public class GraphStyle {
71    int lineStrokeWeight;
72    int pointStrokeWeight;
73    color strokeColor;
74
75    GraphStyle (int lineStrokeWeight, int pointStrokeWeight, color strokeColor) {
76      this.lineStrokeWeight = lineStrokeWeight;
77      this.pointStrokeWeight = pointStrokeWeight;
78      this.strokeColor = strokeColor;
79    }
80
81    public void setLineStyle() {
82     strokeWeight(lineStrokeWeight);
83     stroke(strokeColor);
84    }
85
86    public void setPointStyle() {
87      strokeWeight(pointStrokeWeight-sliderValue/10);
88      stroke(strokeColor);
89    }
90
91    public void setLegendStyle() {
92      strokeWeight(legendStrokeWeight);
93      fill(strokeColor);
94    }
95  }
96
97  public class CardStyle {
98    color mainFillColor;
99    color secondaryFillColor;
100   color strokeColor;
101   int strokeWeight;
102
103   CardStyle (color mainFillColor, color secondaryFillColor, color strokeColor, int
strokeWeight) {
104     this.mainFillColor = mainFillColor;
105     this.secondaryFillColor = secondaryFillColor;
106     this.strokeColor = strokeColor;
107     this.strokeWeight = strokeWeight;
108   }
109
110   public void setMainStyle() {
111     fill(mainFillColor);
112     stroke(strokeColor);
113     strokeWeight(strokeWeight);
114   }
115
116   public void setSecondaryStyle() {
117     fill(secondaryFillColor);
118     stroke(strokeColor);
119     strokeWeight(strokeWeight);
120   }
121 }
122
123 public class TextStyle {
124  color textColor;
125  int textSize;
126
127  TextStyle (color textColor, int textSize) {
128   this.textColor = textColor;
129   this.textSize = textSize;
130  }
131
132  void setStyle () {
133    textAlign(CENTER,CENTER);
```

```
    textSize(textSize);
    fill(textColor);
  }
 }

void setExportDataRectStyle() {
  stroke(0,0,0);
  strokeWeight(1);
  fill(255,255,255);
}

```

xi. Processing – gui

```
/* =========================================================
 * ====                   WARNING                       ===
 * =========================================================
 * The code in this tab has been generated from the GUI form
 * designer and care should be taken when editing this file.
 * Only add/edit code inside the event handlers i.e. only
 * use lines between the matching comment tags. e.g.
 */

 public void buttonMenuData_click(GButton source, GEvent event) {
//_CODE_:buttonMenuData:660030:
   activeMenu = 1;
 } //_CODE_:buttonMenuData:660030:

 public void buttonMenuGraphs_click(GButton source, GEvent event) {
//_CODE_:buttonMenuGraphs:317833:
   activeMenu = 2;
 } //_CODE_:buttonMenuGraphs:317833:

 public void buttonStartSavingData_click(GButton source, GEvent event) {
//_CODE_:buttonStartSavingData:668410:
   mustSaveData = true;
   buttonStartSavingData.setVisible(false);
   buttonFinishSavingData.setVisible(true);
   buttonCancelSavingData.setVisible(true);
 } //_CODE_:buttonStartSavingData:668410:

 public void buttonFinishSavingData click(GButton source, GEvent event) {
//_CODE_:buttonFinishSavingData:426868:
   mustSaveData = false;
   buttonStartSavingData.setVisible(true);
   buttonFinishSavingData.setVisible(false);
   buttonCancelSavingData.setVisible(false);
   saveTheTable(dataToSave);
 } //_CODE_:buttonFinishSavingData:426868:

 public void buttonCancelSavingData_click(GButton source, GEvent event) {
//_CODE_:buttonCancelSavingData:662516:
   mustSaveData = false;
   buttonStartSavingData.setVisible(true);
   buttonFinishSavingData.setVisible(false);
   buttonCancelSavingData.setVisible(false);
   cancelTable(dataToSave);
 } //_CODE_:buttonCancelSavingData:662516:

 public void checkbox_mq135_1_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq135_1:455375:
 } //_CODE_:checkbox_mq135_1:455375:

 public void checkbox_mq3_2_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq3_2:458706:
 } //_CODE_:checkbox_mq3_2:458706:

 public void checkbox_mq3_1_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq3_1:447250:
 } //_CODE_:checkbox_mq3_1:447250:

 public void checkbox_mq2_1_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq2_1:974596:
 } //_CODE_:checkbox_mq2_1:974596:

```

```
53  public void checkbox_mq2_2_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq2_2:413909:
54  } //_CODE_:checkbox_mq2_2:413909:
55
56  public void checkbox_mq2_3_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq2_3:214695:
57  } //_CODE_:checkbox_mq2_3:214695:
58
59  public void checkbox_mq135_3_clicked(GCheckbox source, GEvent event) {
// CODE :checkbox mq135 3:577298:
60  } //_CODE_:checkbox_mq135_3:577298:
61
62  public void checkbox_mq135_2_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq135_2:709201:
63  } //_CODE_:checkbox_mq135_2:709201:
64
65  public void checkbox_mq8_2_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq8_2:220191:
66  } //_CODE_:checkbox_mq8_2:220191:
67
68  public void checkbox_mq5_2_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq5_2:503728:
69  } //_CODE_:checkbox_mq5_2:503728:
70
71  public void checkbox_mq8_1_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq8_1:483290:
72  } //_CODE_:checkbox_mq8_1:483290:
73
74  public void checkbox_mq6_2_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq6_2:426810:
75  } //_CODE_:checkbox_mq6_2:426810:
76
77  public void checkbox_mq6_1_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq6_1:259495:
78  } //_CODE_:checkbox_mq6_1:259495:
79
80  public void checkbox_mq5_1_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq5_1:722537:
81  } //_CODE_:checkbox_mq5_1:722537:
82
83  public void checkbox_mq4_2_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq4_2:627600:
84  } //_CODE_:checkbox_mq4_2:627600:
85
86  public void checkbox_mq4_1_clicked(GCheckbox source, GEvent event) {
//_CODE_:checkbox_mq4_1:898831:
87  } //_CODE_:checkbox_mq4_1:898831:
88
89  public void sliderTime_change1(GCustomSlider source, GEvent event) {
//_CODE_:sliderTime:697298:
90    sliderValue = sliderTime.getValueI();
91  } //_CODE_:sliderTime:697298:
92
93  public void buttonCheckAll_click(GButton source, GEvent event) {
//_CODE_:buttonCheckAll:845685:
94    checkbox_mq135_1.setSelected(true);
95    checkbox_mq135_2.setSelected(true);
96    checkbox_mq135_3.setSelected(true);
97
98    checkbox_mq2_1.setSelected(true);
99    checkbox_mq2_2.setSelected(true);
100   checkbox_mq2_3.setSelected(true);
101
102   checkbox_mq3_1.setSelected(true);
103   checkbox_mq3_2.setSelected(true);
104
105   checkbox_mq4_1.setSelected(true);
106   checkbox_mq4_2.setSelected(true);
107
108   checkbox_mq5_1.setSelected(true);
109   checkbox_mq5_2.setSelected(true);
110
111   checkbox_mq6_1.setSelected(true);
112   checkbox_mq6_2.setSelected(true);
113
114   checkbox_mq8_1.setSelected(true);
115   checkbox_mq8_2.setSelected(true);
```

```
116 } //_CODE_:buttonCheckAll:845685:
117
118 public void buttonUncheckAll_click(GButton source, GEvent event) {
// CODE :buttonUncheckAll:535983:
119   checkbox_mq135_1.setSelected(false);
120   checkbox_mq135_2.setSelected(false);
121   checkbox_mq135_3.setSelected(false);
122
123   checkbox_mq2_1.setSelected(false);
124   checkbox_mq2_2.setSelected(false);
125   checkbox_mq2_3.setSelected(false);
126
127   checkbox_mq3_1.setSelected(false);
128   checkbox_mq3_2.setSelected(false);
129
130   checkbox_mq4_1.setSelected(false);
131   checkbox_mq4_2.setSelected(false);
132
133   checkbox_mq5_1.setSelected(false);
134   checkbox_mq5_2.setSelected(false);
135
136   checkbox_mq6_1.setSelected(false);
137   checkbox_mq6_2.setSelected(false);
138
139   checkbox_mq8_1.setSelected(false);
140   checkbox_mq8_2.setSelected(false);
141 } //_CODE_:buttonUncheckAll:535983:
142
143 synchronized public void win_draw(PApplet appc, GWinData data) {
//_CODE_:WindowSamplingPeriod:701704:
144   appc.background(230);
145 } //_CODE_:WindowSamplingPeriod:701704:
146
147 public void closeMethod(GWindow window) { //_CODE_:WindowSamplingPeriod:734249:
148 } //_CODE_:WindowSamplingPeriod:734249:
149
150 public void textfieldPeriod_change1(GTextField source, GEvent event) {
//_CODE_:textfieldPeriod:542979:
151 } //_CODE_:textfieldPeriod:542979:
152
153 public void buttonAcceptPeriod_click(GButton source, GEvent event) {
//_CODE_:buttonAcceptPeriod:352744:
154   getSamplingPeriod();
155 } //_CODE_:buttonAcceptPeriod:352744:
156
157
158
159 // Create all the GUI controls.
160 // autogenerated do not edit
161 public void createGUI(){
162   G4P.messagesEnabled(false);
163   G4P.setGlobalColorScheme(GCScheme.BLUE_SCHEME);
164   G4P.setMouseOverEnabled(false);
165   GButton.useRoundCorners(false);
166   surface.setTitle("eNose");
167   buttonMenuData = new GButton(this, 0, 490, 200, 110);
168   buttonMenuData.setText("Valores");
169   buttonMenuData.setTextBold();
170   buttonMenuData.addEventHandler(this, "buttonMenuData_click");
171   buttonMenuGraphs = new GButton(this, 200, 490, 200, 110);
172   buttonMenuGraphs.setText("Graficos");
173   buttonMenuGraphs.setTextBold();
174   buttonMenuGraphs.addEventHandler(this, "buttonMenuGraphs_click");
175   buttonStartSavingData = new GButton(this, 400, 550, 400, 50);
176   buttonStartSavingData.setText("Iniciar");
177   buttonStartSavingData.setTextBold();
178   buttonStartSavingData.setLocalColorScheme(GCScheme.GREEN_SCHEME);
179   buttonStartSavingData.addEventHandler(this, "buttonStartSavingData_click");
180   buttonFinishSavingData = new GButton(this, 400, 550, 200, 50);
181   buttonFinishSavingData.setText("Guardar");
182   buttonFinishSavingData.setLocalColorScheme(GCScheme.GOLD_SCHEME);
183   buttonFinishSavingData.addEventHandler(this, "buttonFinishSavingData_click");
184   buttonCancelSavingData = new GButton(this, 600, 550, 200, 50);
185   buttonCancelSavingData.setText("Cancelar");
186   buttonCancelSavingData.setLocalColorScheme(GCScheme.RED_SCHEME);
187   buttonCancelSavingData.addEventHandler(this, "buttonCancelSavingData_click");
188   checkbox_mq135_1 = new GCheckbox(this, 610, 55, 85, 20);
```

```
189   checkbox_mq135_1.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
190   checkbox_mq135_1.setText("MQ135 (1)");
191   checkbox_mq135_1.setTextBold();
192   checkbox_mq135_1.setOpaque(false);
193   checkbox_mq135_1.addEventHandler(this, "checkbox_mq135_1_clicked");
194   checkbox_mq135_1.setSelected(true);
195   checkbox_mq3_2 = new GCheckbox(this, 610, 265, 85, 20);
196   checkbox_mq3_2.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
197   checkbox_mq3_2.setText("MQ3 (2)");
198   checkbox_mq3_2.setTextBold();
199   checkbox_mq3_2.setOpaque(false);
200   checkbox_mq3_2.addEventHandler(this, "checkbox_mq3_2_clicked");
201   checkbox_mq3_2.setSelected(true);
202   checkbox_mq3_1 = new GCheckbox(this, 610, 235, 85, 20);
203   checkbox_mq3_1.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
204   checkbox_mq3_1.setText("MQ3 (1)");
205   checkbox_mq3_1.setTextBold();
206   checkbox_mq3_1.setOpaque(false);
207   checkbox_mq3_1.addEventHandler(this, "checkbox_mq3_1_clicked");
208   checkbox_mq3_1.setSelected(true);
209   checkbox_mq2_1 = new GCheckbox(this, 610, 145, 85, 20);
210   checkbox_mq2_1.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
211   checkbox_mq2_1.setText("MQ2 (1)");
212   checkbox_mq2_1.setTextBold();
213   checkbox_mq2_1.setOpaque(false);
214   checkbox_mq2_1.addEventHandler(this, "checkbox_mq2_1_clicked");
215   checkbox_mq2_1.setSelected(true);
216   checkbox_mq2_2 = new GCheckbox(this, 610, 175, 85, 20);
217   checkbox_mq2_2.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
218   checkbox_mq2_2.setText("MQ2 (2)");
219   checkbox_mq2_2.setTextBold();
220   checkbox_mq2_2.setOpaque(false);
221   checkbox_mq2_2.addEventHandler(this, "checkbox_mq2_2_clicked");
222   checkbox_mq2_2.setSelected(true);
223   checkbox_mq2_3 = new GCheckbox(this, 610, 205, 85, 20);
224   checkbox_mq2_3.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
225   checkbox_mq2_3.setText("MQ2 (3)");
226   checkbox_mq2_3.setTextBold();
227   checkbox_mq2_3.setOpaque(false);
228   checkbox_mq2_3.addEventHandler(this, "checkbox_mq2_3_clicked");
229   checkbox_mq2_3.setSelected(true);
230   checkbox_mq135_3 = new GCheckbox(this, 610, 115, 85, 20);
231   checkbox_mq135_3.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
232   checkbox_mq135_3.setText("MQ135 (3)");
233   checkbox_mq135_3.setTextBold();
234   checkbox_mq135_3.setOpaque(false);
235   checkbox_mq135_3.addEventHandler(this, "checkbox_mq135_3_clicked");
236   checkbox_mq135_3.setSelected(true);
237   checkbox_mq135_2 = new GCheckbox(this, 610, 85, 85, 20);
238   checkbox_mq135_2.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
239   checkbox_mq135_2.setText("MQ135 (2)");
240   checkbox_mq135_2.setTextBold();
241   checkbox_mq135_2.setOpaque(false);
242   checkbox_mq135_2.addEventHandler(this, "checkbox_mq135_2_clicked");
243   checkbox_mq135_2.setSelected(true);
244   checkbox_mq8_2 = new GCheckbox(this, 705, 265, 85, 20);
245   checkbox_mq8_2.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
246   checkbox_mq8_2.setText("MQ8 (2)");
247   checkbox_mq8_2.setTextBold();
248   checkbox_mq8_2.setOpaque(false);
249   checkbox_mq8_2.addEventHandler(this, "checkbox_mq8_2_clicked");
250   checkbox_mq8_2.setSelected(true);
251   checkbox_mq5_2 = new GCheckbox(this, 705, 145, 85, 20);
252   checkbox_mq5_2.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
253   checkbox_mq5_2.setText("MQ5 (2)");
254   checkbox_mq5_2.setTextBold();
255   checkbox_mq5_2.setOpaque(false);
256   checkbox_mq5_2.addEventHandler(this, "checkbox_mq5_2_clicked");
257   checkbox_mq5_2.setSelected(true);
258   checkbox_mq8_1 = new GCheckbox(this, 705, 235, 85, 20);
259   checkbox_mq8_1.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
260   checkbox_mq8_1.setText("MQ8 (1)");
261   checkbox_mq8_1.setTextBold();
262   checkbox_mq8_1.setOpaque(false);
263   checkbox_mq8_1.addEventHandler(this, "checkbox_mq8_1_clicked");
264   checkbox_mq8_1.setSelected(true);
265   checkbox_mq6_2 = new GCheckbox(this, 705, 205, 85, 20);
```

```
266   checkbox_mq6_2.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
267   checkbox_mq6_2.setText("MQ6 (2)");
268   checkbox_mq6_2.setTextBold();
269   checkbox_mq6_2.setOpaque(false);
270   checkbox_mq6_2.addEventHandler(this, "checkbox_mq6_2_clicked");
271   checkbox_mq6_2.setSelected(true);
272   checkbox_mq6_1 = new GCheckbox(this, 705, 175, 85, 20);
273   checkbox_mq6_1.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
274   checkbox_mq6_1.setText("MQ6 (1)");
275   checkbox_mq6_1.setTextBold();
276   checkbox_mq6_1.setOpaque(false);
277   checkbox_mq6_1.addEventHandler(this, "checkbox_mq6_1_clicked");
278   checkbox_mq6_1.setSelected(true);
279   checkbox_mq5_1 = new GCheckbox(this, 705, 115, 85, 20);
280   checkbox_mq5_1.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
281   checkbox_mq5_1.setText("MQ5 (1)");
282   checkbox_mq5_1.setTextBold();
283   checkbox_mq5_1.setOpaque(false);
284   checkbox_mq5_1.addEventHandler(this, "checkbox_mq5_1_clicked");
285   checkbox_mq5_1.setSelected(true);
286   checkbox_mq4_2 = new GCheckbox(this, 705, 85, 85, 20);
287   checkbox_mq4_2.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
288   checkbox_mq4_2.setText("MQ4 (2)");
289   checkbox_mq4_2.setTextBold();
290   checkbox_mq4_2.setOpaque(false);
291   checkbox_mq4_2.addEventHandler(this, "checkbox_mq4_2_clicked");
292   checkbox_mq4_2.setSelected(true);
293   checkbox_mq4_1 = new GCheckbox(this, 705, 55, 85, 20);
294   checkbox_mq4_1.setIconAlign(GAlign.LEFT, GAlign.MIDDLE);
295   checkbox_mq4_1.setText("MQ4 (1)");
296   checkbox_mq4_1.setTextBold();
297   checkbox_mq4_1.setOpaque(false);
298   checkbox_mq4_1.addEventHandler(this, "checkbox_mq4_1_clicked");
299   checkbox_mq4_1.setSelected(true);
300   sliderTime = new GCustomSlider(this, 600, 375, 180, 50, "blue18px");
301   sliderTime.setShowValue(false);
302   sliderTime.setShowLimits(false);
303   sliderTime.setLimits(10, 10, 60);
304   sliderTime.setNbrTicks(6);
305   sliderTime.setStickToTicks(true);
306   sliderTime.setShowTicks(true);
307   sliderTime.setNumberFormat(G4P.INTEGER, 0);
308   sliderTime.setOpaque(false);
309   sliderTime.addEventHandler(this, "sliderTime_change1");
310   buttonCheckAll = new GButton(this, 610, 25, 85, 20);
311   buttonCheckAll.setText("Marcar todo");
312   buttonCheckAll.setTextBold();
313   buttonCheckAll.addEventHandler(this, "buttonCheckAll_click");
314   buttonUncheckAll = new GButton(this, 705, 25, 85, 20);
315   buttonUncheckAll.setText("Desmarcar");
316   buttonUncheckAll.setTextBold();
317   buttonUncheckAll.addEventHandler(this, "buttonUncheckAll_click");
318   WindowSamplingPeriod = GWindow.getWindow(this, "Periodo de muestreo", 500, 300,
300, 200, JAVA2D);
319   WindowSamplingPeriod.noLoop();
320   WindowSamplingPeriod.setActionOnClose(G4P.KEEP_OPEN);
321   WindowSamplingPeriod.addDrawHandler(this, "win_draw");
322   WindowSamplingPeriod.addOnCloseHandler(this, "closeMethod");
323   labelPeriod = new GLabel(WindowSamplingPeriod, 50, 10, 200, 30);
324   labelPeriod.setTextAlign(GAlign.CENTER, GAlign.MIDDLE);
325   labelPeriod.setText("Introduce el periodo de muestreo:");
326   labelPeriod.setTextBold();
327   labelPeriod.setOpaque(false);
328   textfieldPeriod = new GTextField(WindowSamplingPeriod, 115, 50, 50, 20,
G4P.SCROLLBARS_NONE);
329   textfieldPeriod.setPromptText("1000");
330   textfieldPeriod.setOpaque(true);
331   textfieldPeriod.addEventHandler(this, "textfieldPeriod_change1");
332   labelMs = new GLabel(WindowSamplingPeriod, 165, 50, 20, 20);
333   labelMs.setText("ms");
334   labelMs.setOpaque(false);
335   buttonAcceptPeriod = new GButton(WindowSamplingPeriod, 100, 90, 100, 30);
336   buttonAcceptPeriod.setText("Aceptar");
337   buttonAcceptPeriod.setTextBold();
338   buttonAcceptPeriod.addEventHandler(this, "buttonAcceptPeriod_click");
339   labelWarning1 = new GLabel(WindowSamplingPeriod, 0, 130, 300, 40);
340   labelWarning1.setTextAlign(GAlign.CENTER, GAlign.MIDDLE);
```

```
341   labelWarning1.setText("¡Atención! El Periodo de muestreo debe ser mayor de 500
ms");
342   labelWarning1.setLocalColorScheme(GCScheme.RED_SCHEME);
343   labelWarning1.setOpaque(false);
344   labelError = new GLabel(WindowSamplingPeriod, 0, 170, 300, 30);
345   labelError.setTextAlign(GAlign.CENTER, GAlign.MIDDLE);
346   labelError.setText("¡ERROR! El periodo introducido no es válido.");
347   labelError.setTextBold();
348   labelError.setLocalColorScheme(GCScheme.RED_SCHEME);
349   labelError.setOpaque(false);
350   labelError.setVisible(false);
351   buttonStartSavingData.setVisible(true);
352   buttonFinishSavingData.setVisible(false);
353   buttonCancelSavingData.setVisible(false);
354   WindowSamplingPeriod.loop();
355 }
356
357 // Variable declarations
358 // autogenerated do not edit
359 GButton buttonMenuData;
360 GButton buttonMenuGraphs;
361 GButton buttonStartSavingData;
362 GButton buttonFinishSavingData;
363 GButton buttonCancelSavingData;
364 GCheckbox checkbox_mq135_1;
365 GCheckbox checkbox_mq3_2;
366 GCheckbox checkbox_mq3_1;
367 GCheckbox checkbox_mq2_1;
368 GCheckbox checkbox_mq2_2;
369 GCheckbox checkbox_mq2_3;
370 GCheckbox checkbox_mq135_3;
371 GCheckbox checkbox_mq135_2;
372 GCheckbox checkbox_mq8_2;
373 GCheckbox checkbox_mq5_2;
374 GCheckbox checkbox_mq8_1;
375 GCheckbox checkbox_mq6_2;
376 GCheckbox checkbox_mq6_1;
377 GCheckbox checkbox_mq5_1;
378 GCheckbox checkbox_mq4_2;
379 GCheckbox checkbox_mq4_1;
380 GCustomSlider sliderTime;
381 GButton buttonCheckAll;
382 GButton buttonUncheckAll;
383 GWindow WindowSamplingPeriod;
384 GLabel labelPeriod;
385 GTextField textfieldPeriod;
386 GLabel labelMs;
387 GButton buttonAcceptPeriod;
388 GLabel labelWarning1;
389 GLabel labelError;
39
```

Capítulo 2. Pliego de condiciones

Índice Capítulo 2. Pliego de condiciones

1. Definición y alcance del pliego

El objeto de este documento es determinar las condiciones que deben cumplir los materiales y componentes empleados para el diseño del sistema, así como el proceso de ejecución del proyecto.

El ámbito de aplicación se extiende tanto a los elementos electrónicos como al software desarrollado que forman parte del sistema. No se incluye, por el contrario, todo aquello relacionado con el diseño de la parte mecánica, estudios sobre flujos del fluido en las cámaras de medida o el desarrollo de algoritmos de identificación de las sustancias.

2. Condiciones y normas de carácter general

Durante la ejecución del proyecto se deberán tener en cuenta las siguientes normas:

- **Real Decreto 614/2011**, de 8 de junio, sobre disposiciones mínimas para la protección de la salud y seguridad de los trabajadores frente al riesgo eléctrico.
- **Real Decreto 773/1997** sobre disposiciones mínimas de seguridad y salud relativas a la utilización por los trabajadores de equipos de protección individual.
- **Real Decreto 1407/1992** sobre comercialización de equipos de protección individual.
- **Real Decreto 486/1997** sobre disposiciones mínimas de seguridad y salud en los lugares de trabajo.
- **Real Decreto 485/1997** sobre disposiciones mínimas en materia de señalización de seguridad y salud en el trabajo.
- **Real Decreto 1215/1997** por el que se establecen las disposiciones mínimas de seguridad y salud para la utilización por los trabajadores de los equipos de trabajo.

3. Condiciones de los materiales

Todos los componentes y materiales empleados deberán cumplir con la legislación vigente que aplique en cada caso concreto. Su uso y montaje se realizará de acuerdo a las recomendaciones indicadas por el fabricante en las fichas técnicas.

3.1. Sensores

Sensor de gases

Sensores MQ-2, MQ-3, MQ-4, MQ-5, MQ-6, MQ-8 y MQ-135 con encapsulado tipo A: metálico, 23 mm de altura, con un diámetro exterior de 16,8 mm y 6 pines dispuestos a lo largo de una circunferencia de 6 mm de diámetro.

Sensores MQ-3 con encapsulado tipo B: cobertura de plástico naranja de 9,2 mm de altura, con un diámetro exterior de 16,8 mm y 6 pines dispuestos a lo largo de una circunferencia de 6 mm de diámetro.

Sensor de temperatura y humedad

Sensor DHT11 con encapsulado de 15,5 mm de alto, 12 mm de ancho y 5,5 mm de grosor. Con 4 pines de 8 mm de largo con separación de 2,54 mm entre ellos.

3.2. Conectores

Conectores para cable plano

Conectores tipo FFC/FPC hembra para cable flexible plano, de 18 terminales. Con separación entre pines de 0,5 mm y montaje superficial sobre PCB.

Conectores de alimentación

Conectores tipo T-Block de dos terminales, con tornillos para la sujeción del cable y montaje en PCB a través de agujeros pasantes. Con cubierta de plástico aislante de color verde o azul y orientación en ángulo recto. Distancia entre pines de 5 mm.

Cabezales macho tipo pin

Cabezales macho tipo pin para insertar en los zócalos de Arduino. Diámetro de pin de 2,54 mm. Compuestos por plástico y metal.

3.3. Resistencias y condensadores

Resistencias de 1 kΩ

Resistencias de 1 kΩ con encapuslado 1206 de tipo SMD/SMT. Potencia de ¼ W y tolerancia del 5%.

Resistencias de 5.1 Ω

Resistencias de 1 kΩ con encapuslado 1206 de tipo SMD/SMT. Potencia de ¼ W y tolerancia del 5%.

Resistencias de 5 kΩ

Resistencias de 1 kΩ con encapuslado 1206 de tipo SMD/SMT. Potencia de ¼ W y tolerancia del 5%.

Condensadores de 100 pF

Condensador cerámico de 100 pF con encapuslado 1206 de tipo SMD/SMT. Potencia de ¼ W y tolerancia del 15%.

3.4. Cables

Cable USB de Arduino

Cable USB 2.0 con conectores Tipo A y Tipo B machos. Longitud de 30 cm y color azul.

Cable de 3 hilos

Cable eléctrico de tres hilos de sección 0,75 mm^2 cada uno y diámetro exterior total de 6 mm. Tensión nominal de 300 Vac y aislamiento de PVC.

Clavija

Clavija para enchufes de tipo Schuko con toma de tierra. Con carcasa de plástico. Intensidad máxima de 16 A.

Cable FFC

Cable plano flexible de 18 hilos, con separación de 0,5 mm y longitud de 50 cm. Intensidad máxima de 0,5 A y tensión de 30 Vac.

4. Condiciones de ejecución

Deberán indicarse en la memoria del proyecto todas las pautas para realizar un correcto montaje de los diferentes componentes que forman el sistema de sensado, así como las necesarias para su uso.

5. Pruebas y ensayos

Todos los dispositivos y materiales empleados durante la ejecución del proyecto deberán ser comprobados antes de su uso para asegurar el correcto funcionamiento de los mismos.

Además, todos los sistemas y programas software diseñados deberán ser sometidos a sendos ensayos, de forma que se pueda comprobar que los resultados son correctos y acorde a lo esperado.

6. Condiciones de entrega

El contratista deberá proporcionar todos los recursos necesarios para la correcta implementación del sistema de medida, entre los que se incluyen:

- Archivos gerber para la fabricación de las placas de circuito impreso.
- Ejecutable del programa para la visualización de datos.
- El microcontrolador de Arduino con el código correctamente cargado en su memoria.
- Todos los cables y la fuente de alimentación necesarios.

Además, será necesaria la entrega de la documentación generada durante el proyecto:

- Planos de las placas de circuito impreso.
- Archivos Proteus con el diseño de las placas.
- Archivo Proteus con la simulación.
- Código de Arduino en formato .ino
- Código fuente del programa de Processing.

Capítulo 3. Presupuesto

Índice Capítulo 3. Presupuesto

1. Cuadro de precios elementales

Se muestran a continuación el coste unitario de los materiales empleados durante el proyecto, así como el de la mano de obra y los medios auxiliares.

Cuadro de precios elementales			
Ref	**Ud**	**Designación**	**Precio**
Materiales			
M1	Ud	Sensor MQ-2	2,75 €
M2	Ud	Sensor MQ-3	2,75 €
M3	Ud	Sensor MQ-4	2,50 €
M4	Ud	Sensor MQ-5	2,50 €
M5	Ud	Sensor MQ-6	2,50 €
M6	Ud	Sensor MQ-8	2,50 €
M7	Ud	Sensor MQ-135	2,75 €
M8	Ud	Sensor DHT11	1,50 €
M9	Ud	Fuente de alimentación TRACO POWER TXL 015-05S	34,09 €
M10	Ud	Arduino Mega 2560	35,00 €
M11	Ud	Resistencia SMD 5,1 Ω 1206 1/4W	0,05 €
M12	Ud	Resistencia SMD 1 kΩ 1206 1/4W	0,05 €
M13	Ud	Resistencia SMD 5 kΩ 1206 1/4W	0,05 €
M14	Ud	Condensador SMD 100 pF 1206 1/4W	0,25 €
M15	Ud	Conector T-BLOCK 2 terminales	1,05 €
M16	Ud	Conector FFC/FPC 18 pines 1,0 mm pitch	2,13 €
M17	Ud	Cabezal macho de 8 pines	0,83 €
M18	Ud	Placa de prototipos 830 puntos	1,86 €
M19	Ud	Placa de prototipos 400 puntos	1,46 €
M20	Ud	Cable tipo Jumper para placas de prototipos	2,49 €
M21	Ud	Cable USB Tipo A a B, 50 cm	0,90 €
M22	ud	Cable FFC 18 pines 1,0 mm pitch, 50 cm	2,55 €
M23	Ud	Clavija bipolar 6 - 10 Amp Blanc FAMATEL 1001	2,99 €
M24	m	Cable 3 hilos, 3 x 1,75 mm2	1,75 €
M25	Ud	PCB Matriz de sensores, 170 x 140 mm	12,00 €
M26	Ud	PCB Shield de Arduino, 100 x 60 mm	8,00 €
M27	Ud	Licencia Windows 10 LTSC	299,99 €
M28	Ud	Licencia Proteus PCB Design Starter Kit	222,71 €
M29	Ud	Licencia Virtual Serial Ports Emulator	24,95 €
M30	Ud	Ordenador Acer Aspire E5 571G	599,95 €
Mano de obra			
P1	h	Ingeniero en electrónica	14,35 €
P2	h	Doctor especialista en electrónica	20,75 €
Medios auxiliares			
AUX	%	Medios auxiliares	2%

2. Cuadro de amortización

Se muestran a continuación el coste de amortización de los equipos empleados durante la realización del proyecto, teniendo en cuenta el porcentaje de horas de uso de los equipos.

Cuadro de amortización de equipos					
Ref	**Ud**	**Designación**	**Precio**	**Amortización**	**Total**
M27	Ud	Licencia Windows 10	299,99 €	1,5%	4,50 €
M28	Ud	Licencia Proteus PCB Design Starter Kit	222,71 €	5%	11,14 €
M29	Ud	Licencia Virtual Serial Ports Emulator	24,95 €	10%	2,50 €
M30	Ud	Ordenador Acer Aspire E5 571G	599,95 €	1,5%	9,00 €

3. Cuadro de precios descompuestos

Se detalla a continuación la cantidad de cada material empleado durante el proyecto, así como de la mano de obra y medios auxiliares.

Cuadro de precios descompuestos					
Ref	**Ud**	**Designación**	**Precio**	**Cantidad**	**Parcial**
D1	Ud	**PCB de 170 x 140 mm con la matriz sensores. Montada y lista para su uso.**	464,78 €	1	**464,78 €**
		Materiales			
M1	Ud	Sensor MQ-2	2,75 €	3	8,25 €
M2	Ud	Sensor MQ-3	2,75 €	2	5,50 €
M3	Ud	Sensor MQ-4	2,50 €	2	5,00 €
M4	Ud	Sensor MQ-5	2,50 €	2	5,00 €
M5	Ud	Sensor MQ-6	2,50 €	2	5,00 €
M6	Ud	Sensor MQ-8	2,50 €	2	5,00 €
M7	Ud	Sensor MQ-135	2,75 €	3	8,25 €
M8	Ud	Sensor DHT11	1,50 €	2	3,00 €
M11	Ud	Resistencia SMD 5,1 Ω 1206 1/4W	0,05 €	16	0,80 €
M12	Ud	Resistencia SMD 1 kΩ 1206 1/4W	0,05 €	16	0,80 €
M13	Ud	Resistencia SMD 5 kΩ 1206 1/4W	0,05 €	2	0,10 €
M14	Ud	Condensador SMD 100 pF 1206 1/4W	0,25 €	16	4,00 €
M15	Ud	Conector T-BLOCK 2 terminales	1,05 €	1	1,05 €
M16	Ud	Conector FFC/FPC 18 pines 1,0 mm pitch	2,13 €	1	2,13 €
M25	Ud	PCB Matriz de sensores, 170 x 140 mm	12,00 €	1	12,00 €
M27	Ud	Licencia Windows 10	4,50 €	1	4,50 €
M28	Ud	Licencia Proteus PCB Design Starter Kit	11,14 €	1	11,14 €
M30	Ud	Ordenador Acer Aspire E5 571G	9,00 €	1	9,00 €
		Mano de obra			
P1	h	Ingeniero en electrónica	14,35 €	24	344,40 €
P2	h	Doctor especialista en electrónica	20,75 €	1	20,75 €
		Medios auxiliares			
AUX	%	Medios auxiliares sobre los costes directos	2%	455,66 €	9,11 €

Ref	Ud	Designación	Precio		
D2	Ud	**Microcontrolador Arduino junto a la placa de expansión de 100 x 60 mm. Montada y lista para su**	790,84 €	1	**790,84 €**
		Materiales			
M15	Ud	Conector T-BLOCK 2 terminales	1,05 €	2	2,10 €
M16	Ud	Conector FFC/FPC 18 pines 1,0 mm pitch	2,13 €	1	2,13 €
M17	Ud	Cabezal macho de 8 pines	0,83 €	5	4,15 €
M26	Ud	PCB Shield de Arduino, 100 x 60 mm	8,00 €	1	8,00 €
M21	Ud	Cable USB Tipo A a B, 50 cm	0,90 €	1	0,90 €
M10	Ud	Arduino Mega 2560	35,00 €	1	35,00 €
M27	Ud	Licencia Windows 10	4,50 €	1	4,50 €
M30	Ud	Ordenador Acer Aspire E5 571G	9,00 €	1	9,00 €
		Mano de obra			
P1	h	Ingeniero en electrónica	14,35 €	48	688,80 €
P2	h	Doctor especialista en electrónica	20,75 €	1	20,75 €
		Medios auxiliares			
AUX	%	Medios auxiliares sobre los costes directos	2%	775,33 €	15,51 €

Ref	Ud	Designación	Precio		
D3	Ud	**Alimentación y conexiones entre circuitos impresos**	765,06 €	1	**765,06 €**
		Materiales			
M22	Ud	Cable FFC 18 pines 1,0 mm pitch, 50 cm	2,55 €	1	2,55 €
M23	Ud	Clavija bipolar 6 - 10 Amp Blanc FAMATEL 1001	2,99 €	1	2,99 €
M24	m	Cable 3 hilos, 3 x 1,75 mm2	1,75 €	0,5	0,88 €
M9	Ud	Fuente de alimentación TRACO POWER TXL 015-05S	34,09 €	1	34,09 €
		Mano de obra y maquinaria			
P1	h	Ingeniero en electrónica	14,35 €	48	688,80 €
P2	h	Doctor especialista en electrónica	20,75 €	1	20,75 €
		Medios auxiliares			
AUX	%	Medios auxiliares sobre los costes directos	2%	750,06 €	15,00 €

Ref	Ud	Designación	Precio		
D4	Ud	**Creación del programa de visualización de datos**	327,67 €	1	**327,67 €**
		Materiales			
M27	Ud	Licencia Windows 10	4,50 €	1	4,50 €
M30	Ud	Ordenador Acer Aspire E5 571G	9,00 €	1	9,00 €
		Mano de obra			
P1	h	Ingeniero en electrónica	14,35 €	20	287,00 €
P2	h	Doctor especialista en electrónica	20,75 €	1	20,75 €
		Medios auxiliares			
AUX	%	Medios auxiliares sobre los costes directos	2%	321,25 €	6,42 €

Ref	Ud	Designación	Precio		
D5	Ud	**Simulación de la matriz de sensores y ensayo para comprobar el correcto funcionamiento de los**	195,21 €	1	195,21 €
		Materiales			
M27	Ud	Licencia Windows 10	4,50 €	1	4,50 €
M28	Ud	Licencia Proteus PCB Design Starter Kit	11,14 €	1	11,14 €
M29	Ud	Licencia Virtual Serial Ports Emulator	2,50 €	1	2,50 €
M30	Ud	Ordenador Acer Aspire E5 571G	9,00 €	1	9,00 €
		Mano de obra y maquinaria			
P1	h	Ingeniero en electrónica	14,35 €	10	143,50 €
P2	h	Doctor especialista en electrónica	20,75 €	1	20,75 €
		Medios auxiliares			
AUX	%	Medios auxiliares sobre los costes directos	2%	191,38 €	3,83 €

Ref	Ud	Designación	Precio		
D6	Ud	**Implementación de prototipo y ensayos para comprobar el correcto funcionamiento del sistema**	481,11 €	1	**481,11 €**
		Materiales			
M18	Ud	Placa de prototipos 830 puntos	1,86 €	2	3,72 €
M19	Ud	Placa de prototipos 400 puntos	1,46 €	1	1,46 €
M20	Ud	Cable tipo Jumper para placas de prototipos	2,49 €	50	124,50 €
M27	Ud	Licencia Windows 10	4,50 €	1	4,50 €
M30	Ud	Ordenador Acer Aspire E5 571G	9,00 €	1	9,00 €
		Mano de obra y maquinaria			
P1	h	Ingeniero en electrónica	14,35 €	20	287,00 €
P2	h	Doctor especialista en electrónica	20,75 €	2	41,50 €
		Medios auxiliares			
AUX	%	Medios auxiliares sobre los costes directos	2%	471,68 €	9,43 €

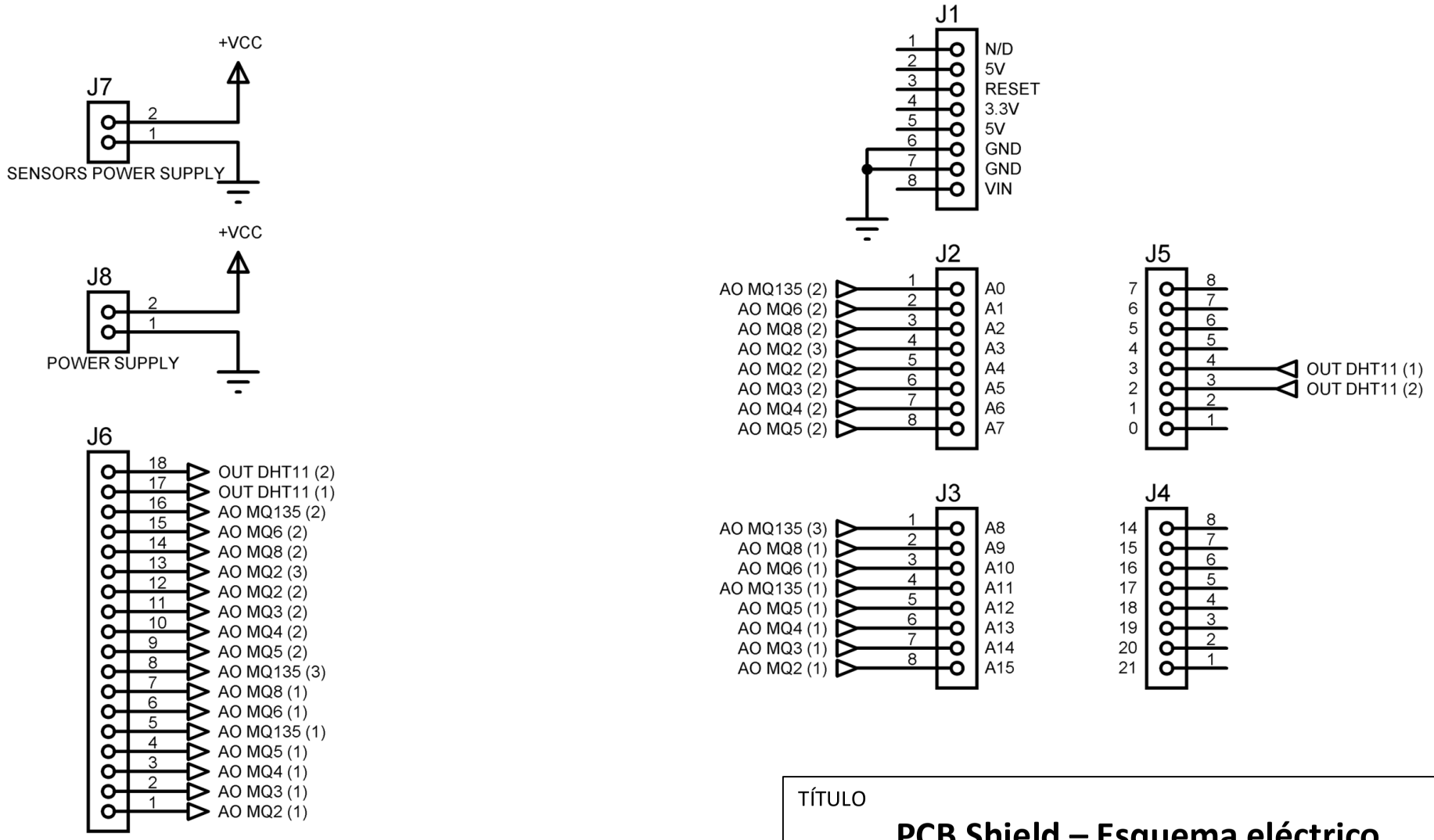
4. Valoración del presupuesto

Valoración del presupuesto					
Ref	**Ud**	**Designación**	**Precio**	**Cantidad**	**Total**
D1	Ud	PCB de 170 x 140 mm con la matriz sensores. Montada y lista para su uso.	464,78 €	1	464,78 €
D2	Ud	Microcontrolador Arduino junto a la placa de expansión de 100 x 60 mm. Montada y lista para su	790,84 €	1	790,84 €
D3	Ud	Alimentación y conexiones entre circuitos impresos	765,06 €	1	765,06 €
D4	Ud	Creación del programa de visualización de datos	327,67 €	1	327,67 €
D5	Ud	Simulación de la matriz de sensores y ensayo para comprobar el correcto funcionamiento de los	195,21 €	1	195,21 €
D6	Ud	Implementación de prototipo y ensayos para comprobar el correcto funcionamiento del sistema	481,11 €	1	481,11 €
				Total (Sin impuestos)	3.024,66 €
				IVA (21%)	635,18 €
				Total	**3.659,84 €**

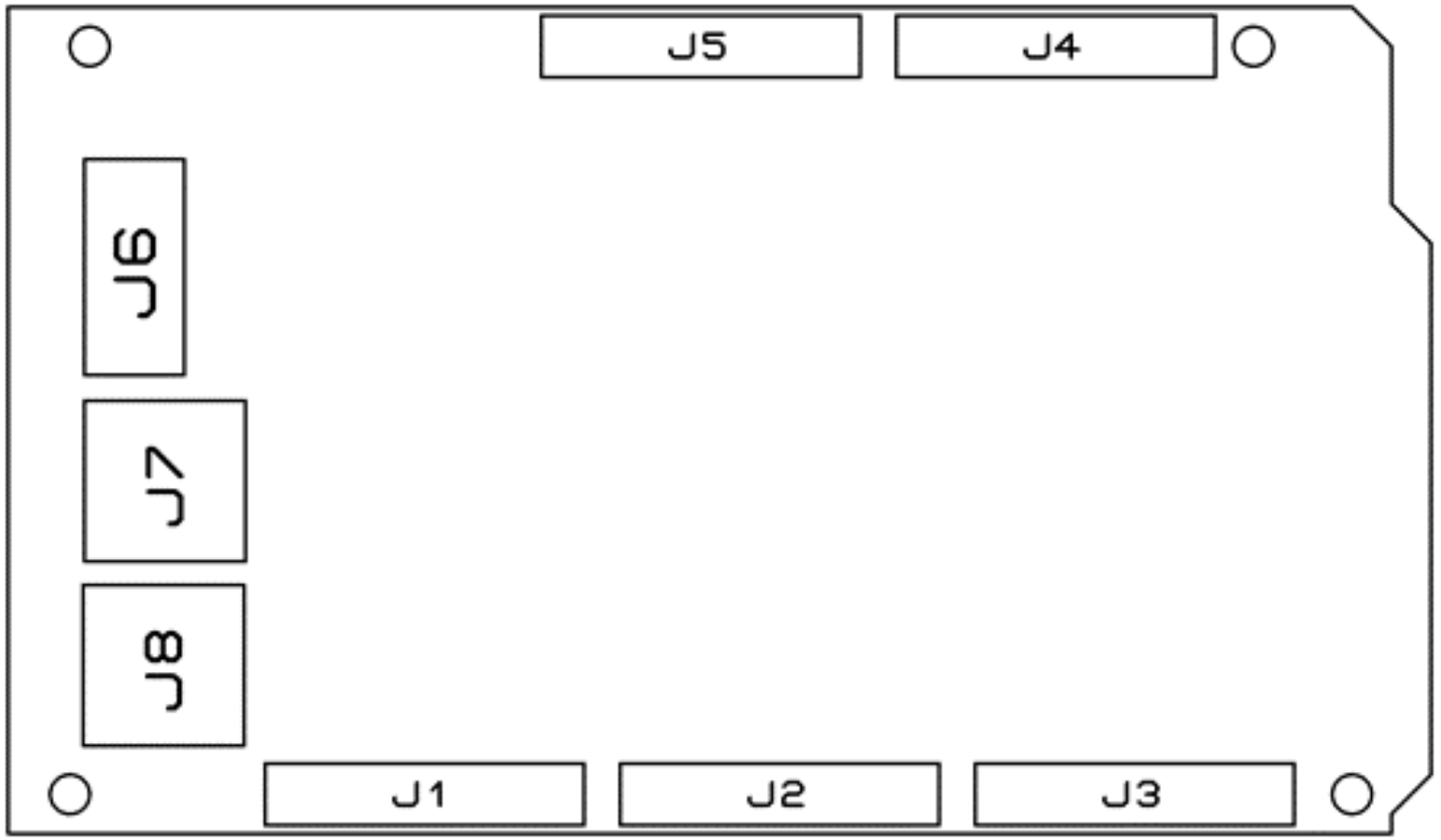
El coste total de ejecución del proyecto es de **TRES MIL SEISCIENTOS CINCUENTA Y NUEVE EUROS CON OCHENTA Y CUATRO CÉNTIMOS.**

Capítulo 4. Planos

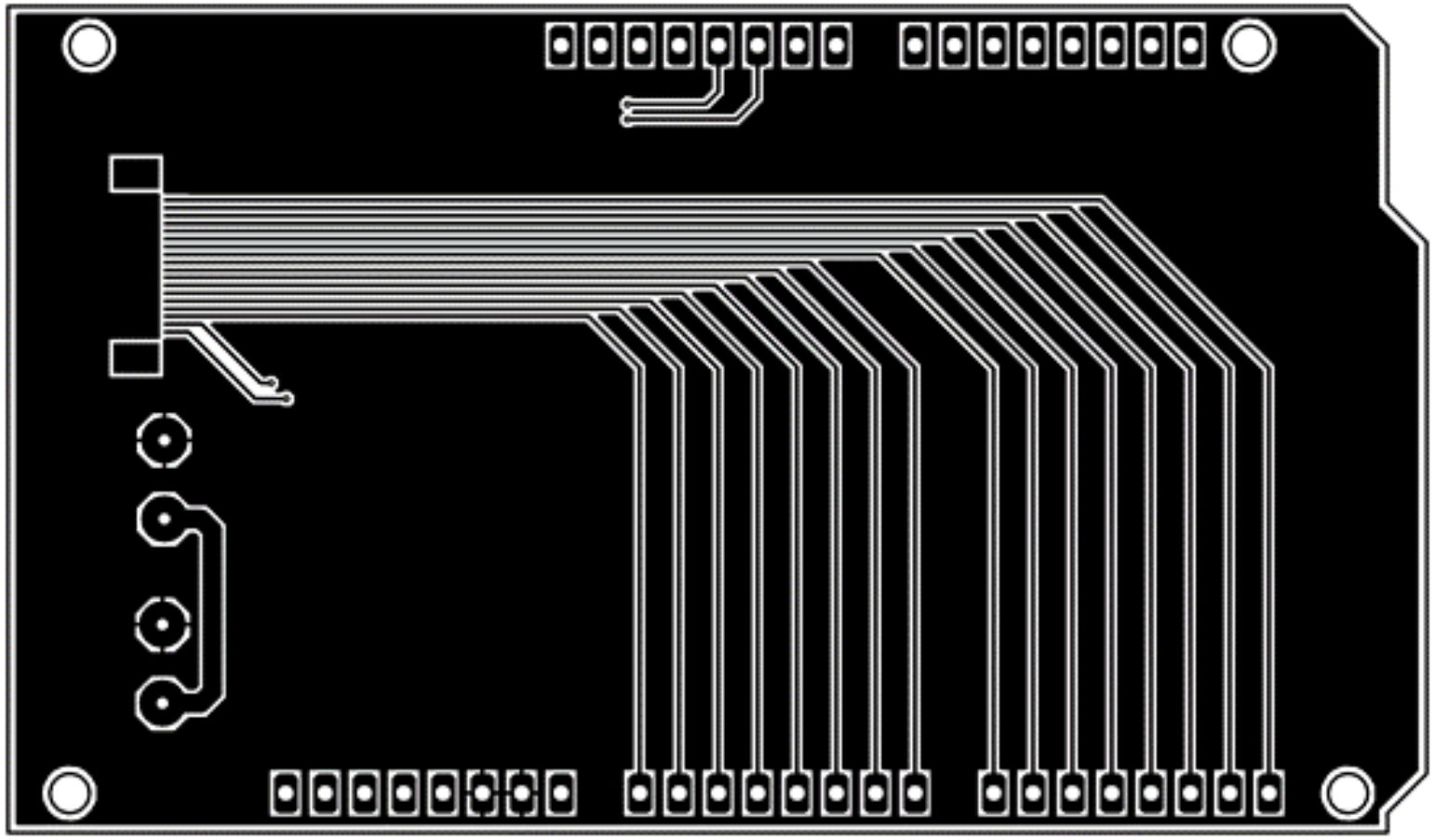
Índice Capítulo 4. Planos



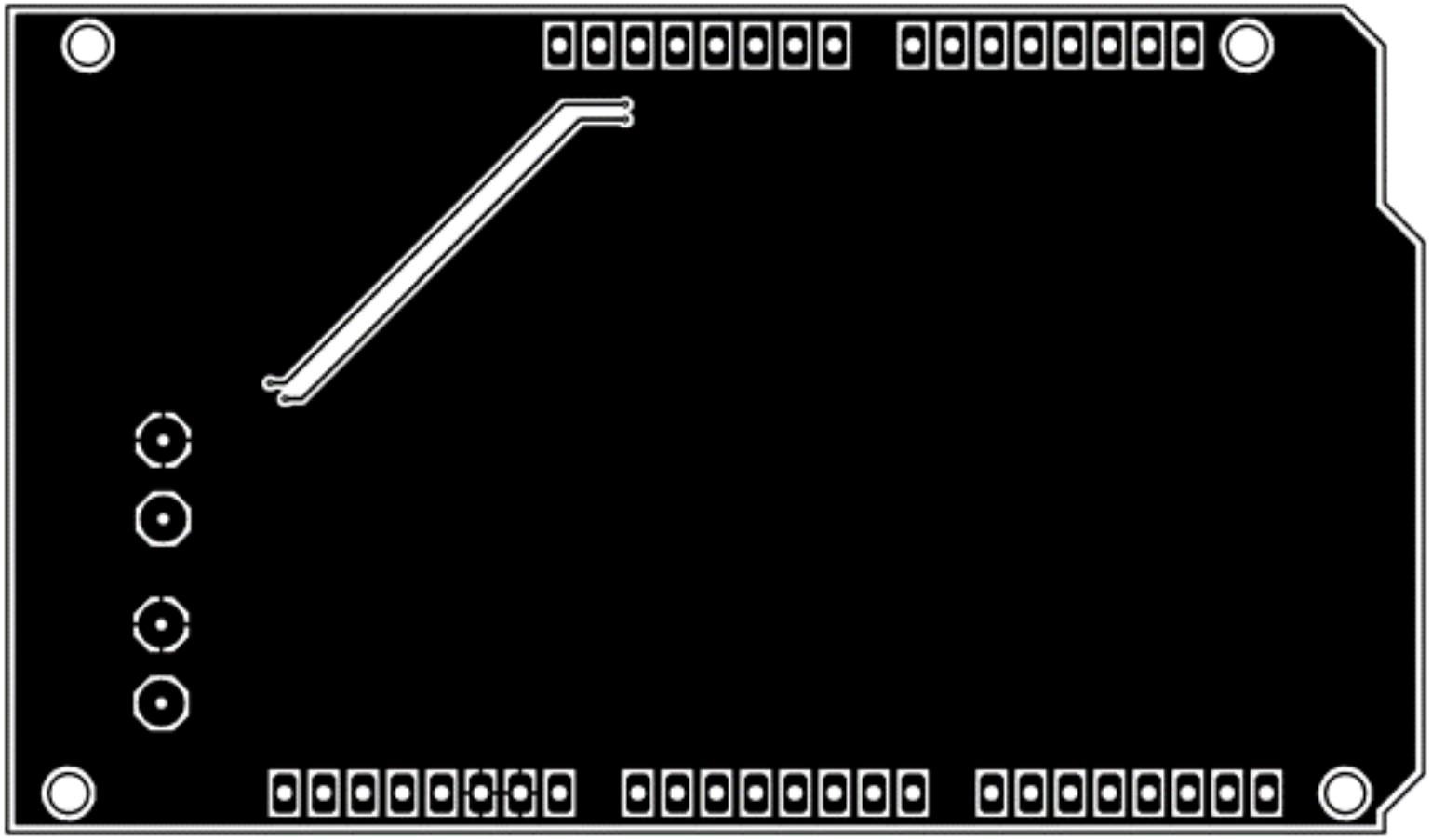
TÍTULO: **PCB Shield – Esquema eléctrico**		
CREADO POR: Alberto Díaz Paredes	FECHA: 01/06/2019	Nº DE PLANO: SH.01
APROBADO POR: Cristian Ariel Olguín Pinatti	REVISIÓN: 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA



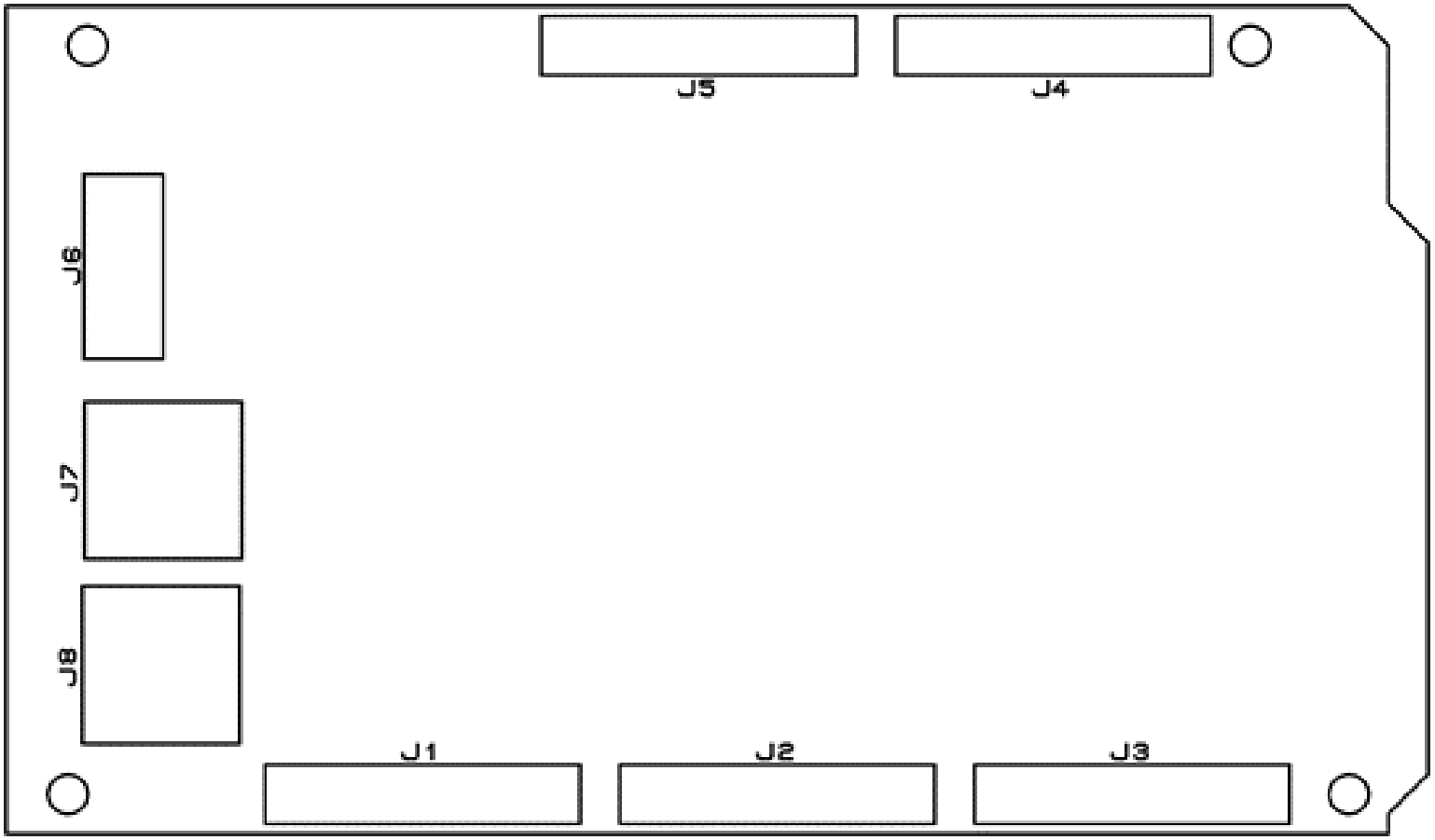
TÍTULO		
PCB Shield – Top Assembly		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SH.02
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA



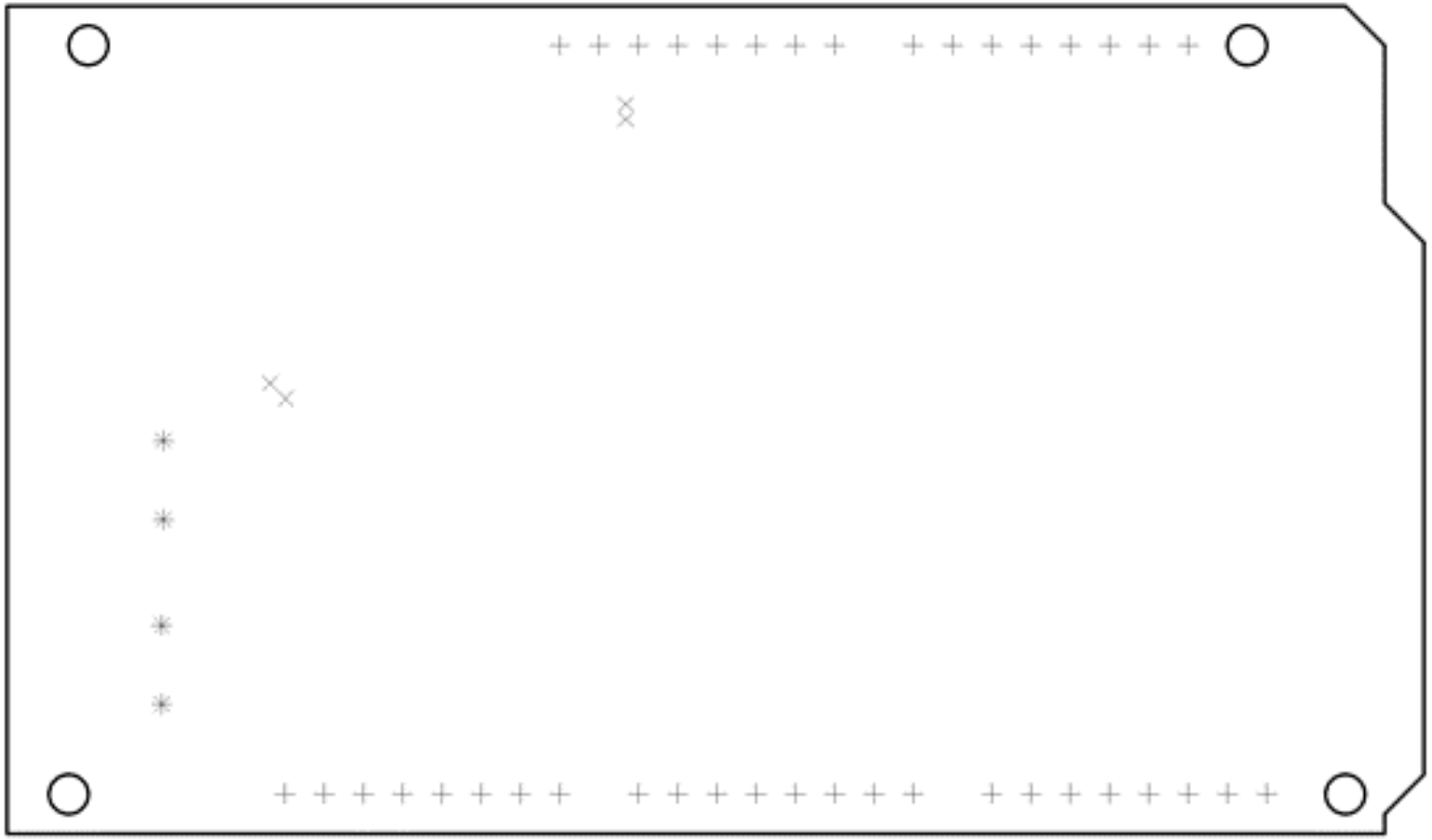
TÍTULO		
PCB Shield – Top Copper		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SH.03
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA



TÍTULO		
PCB Shield – Bottom Copper		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SH.04
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA

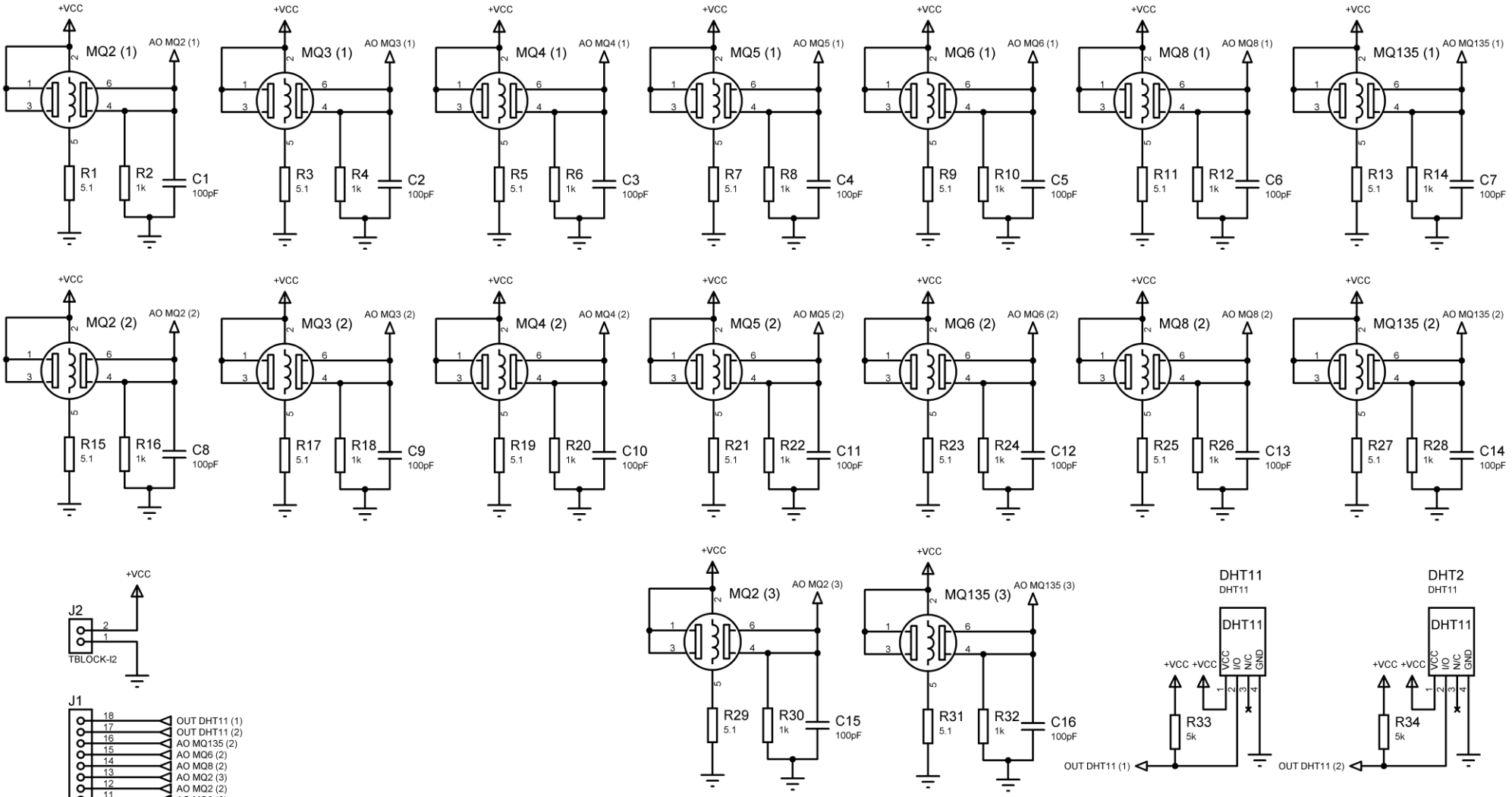


TÍTULO		
PCB Shield – Top Silk		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SH.05
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA

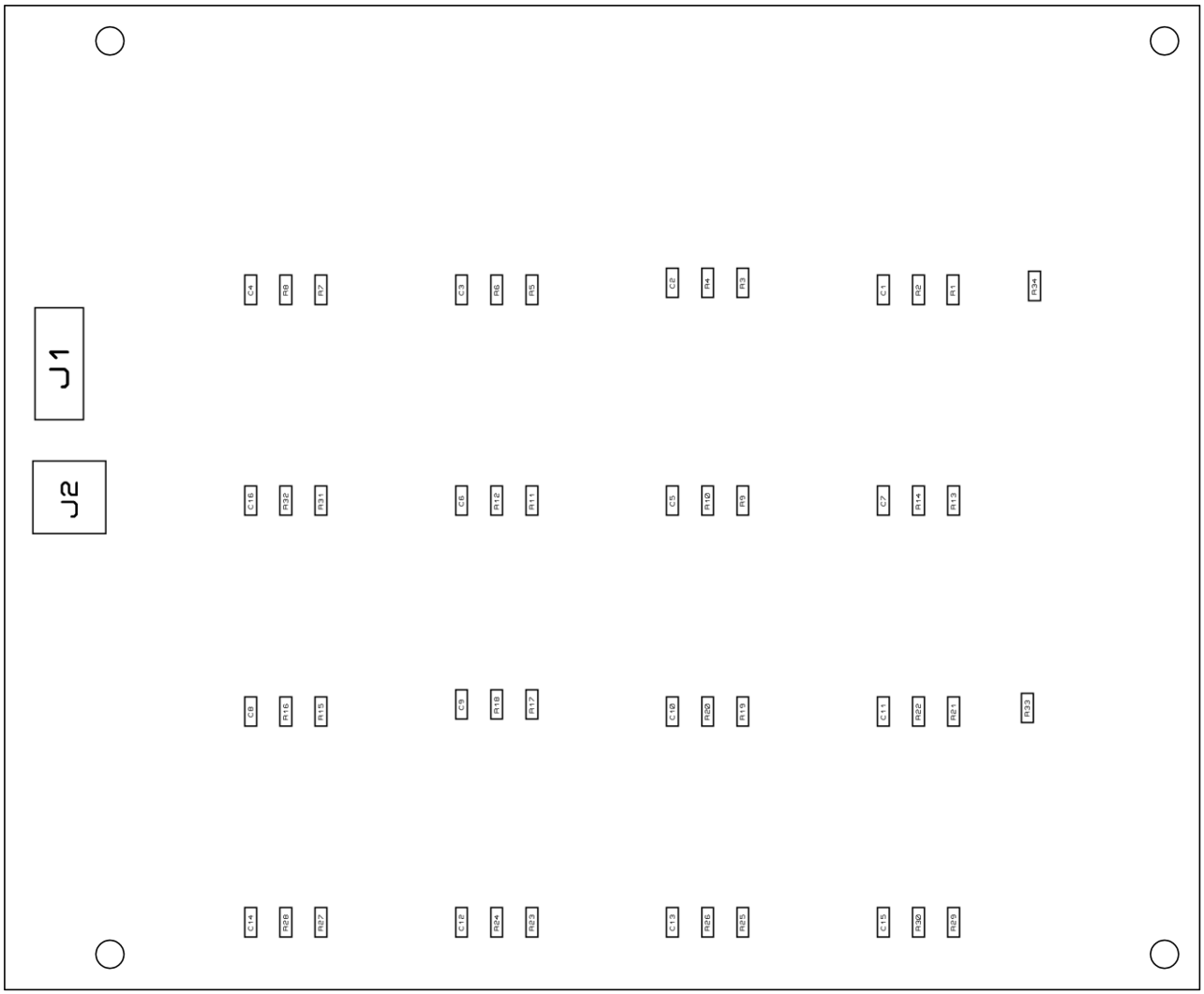


SYM	SIZE	PLATED	QTY
+	1mm	YES	40
×	10th	YES	4
✳	60th	YES	4

TÍTULO **PCB Shield – Drill**		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SH.06
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA

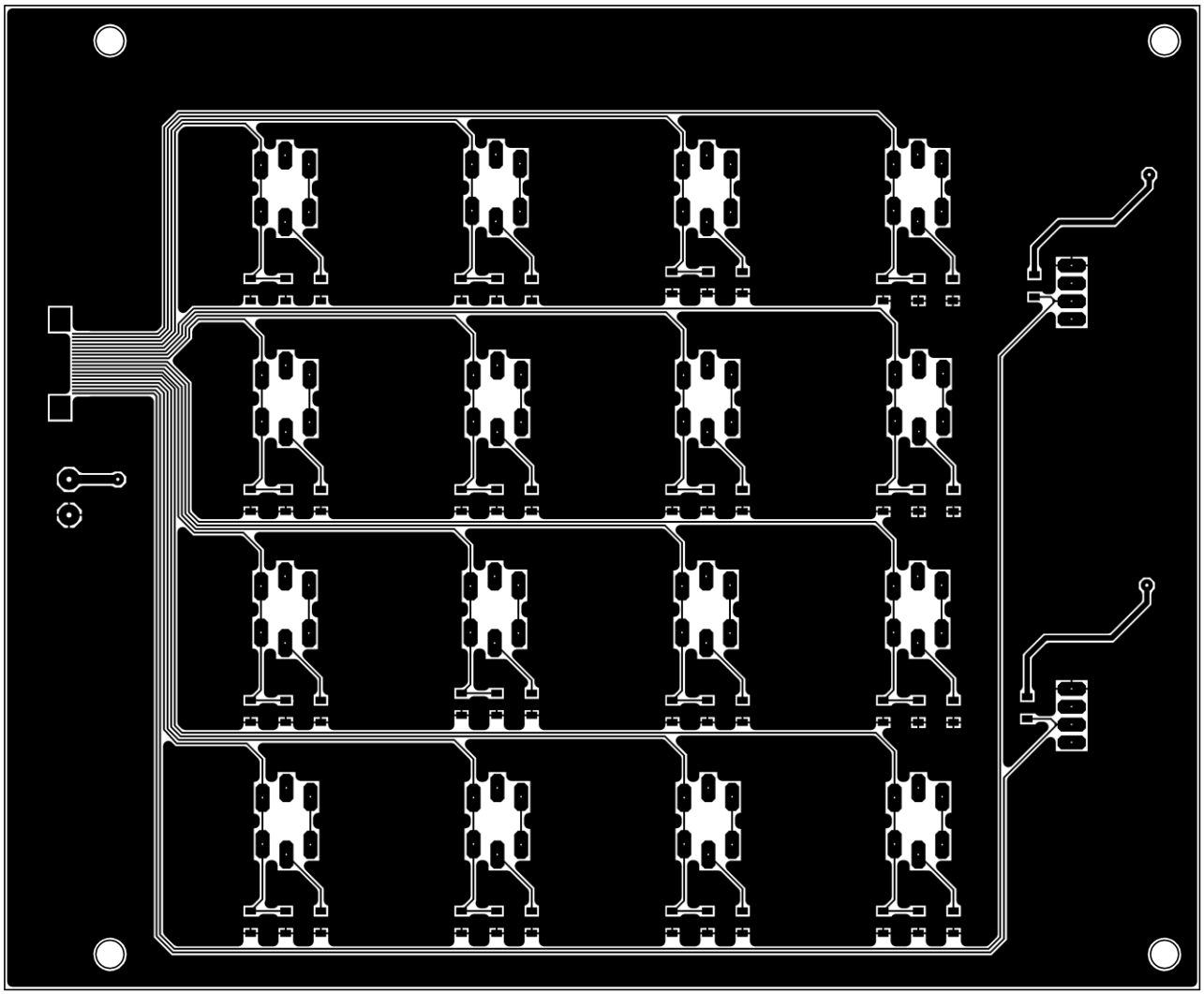


TÍTULO		
PCB Sensores – Esquema eléctrico		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SE.01
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA

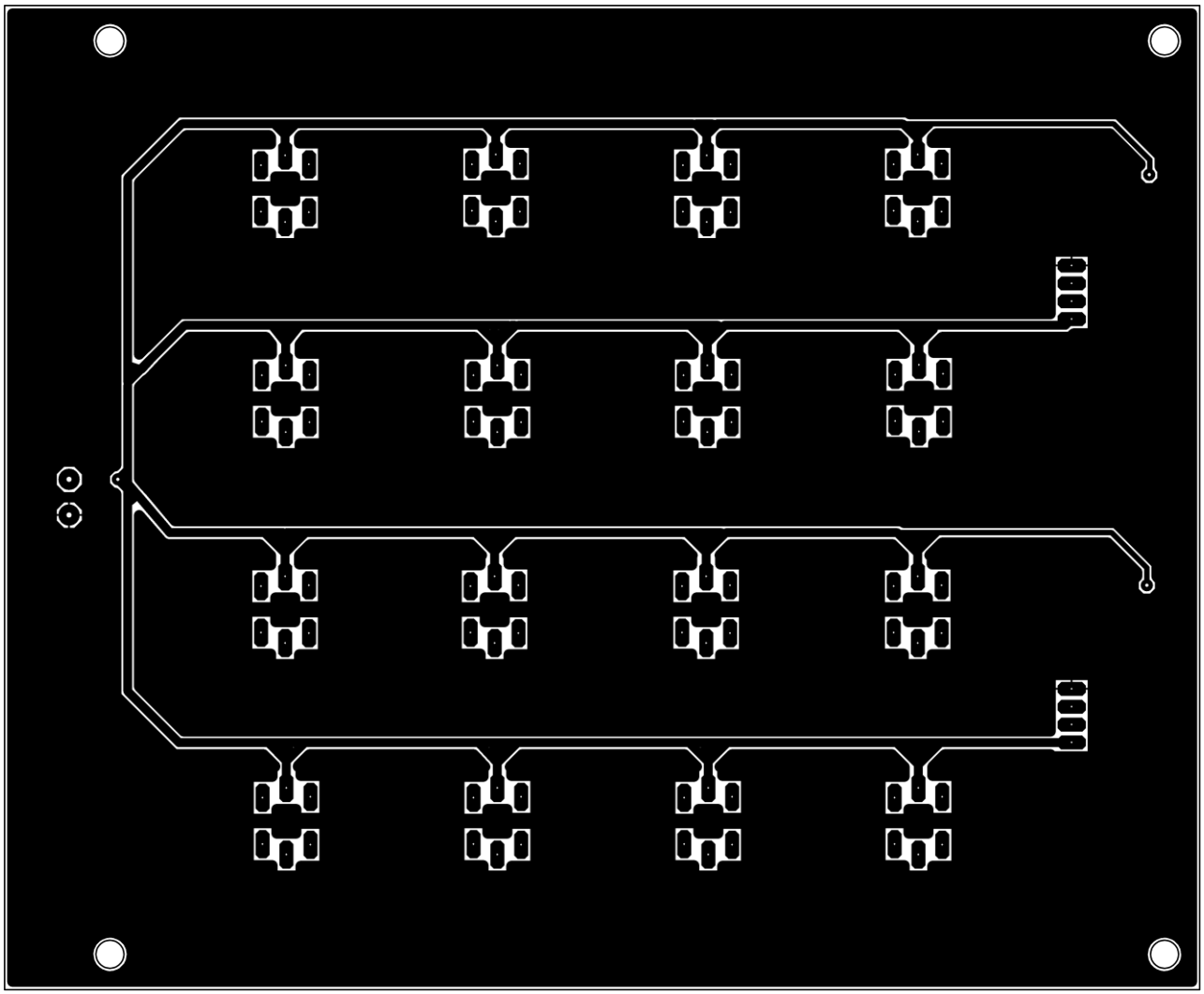


TÍTULO		
PCB Sensores – Top Assembly		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SE.02
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA

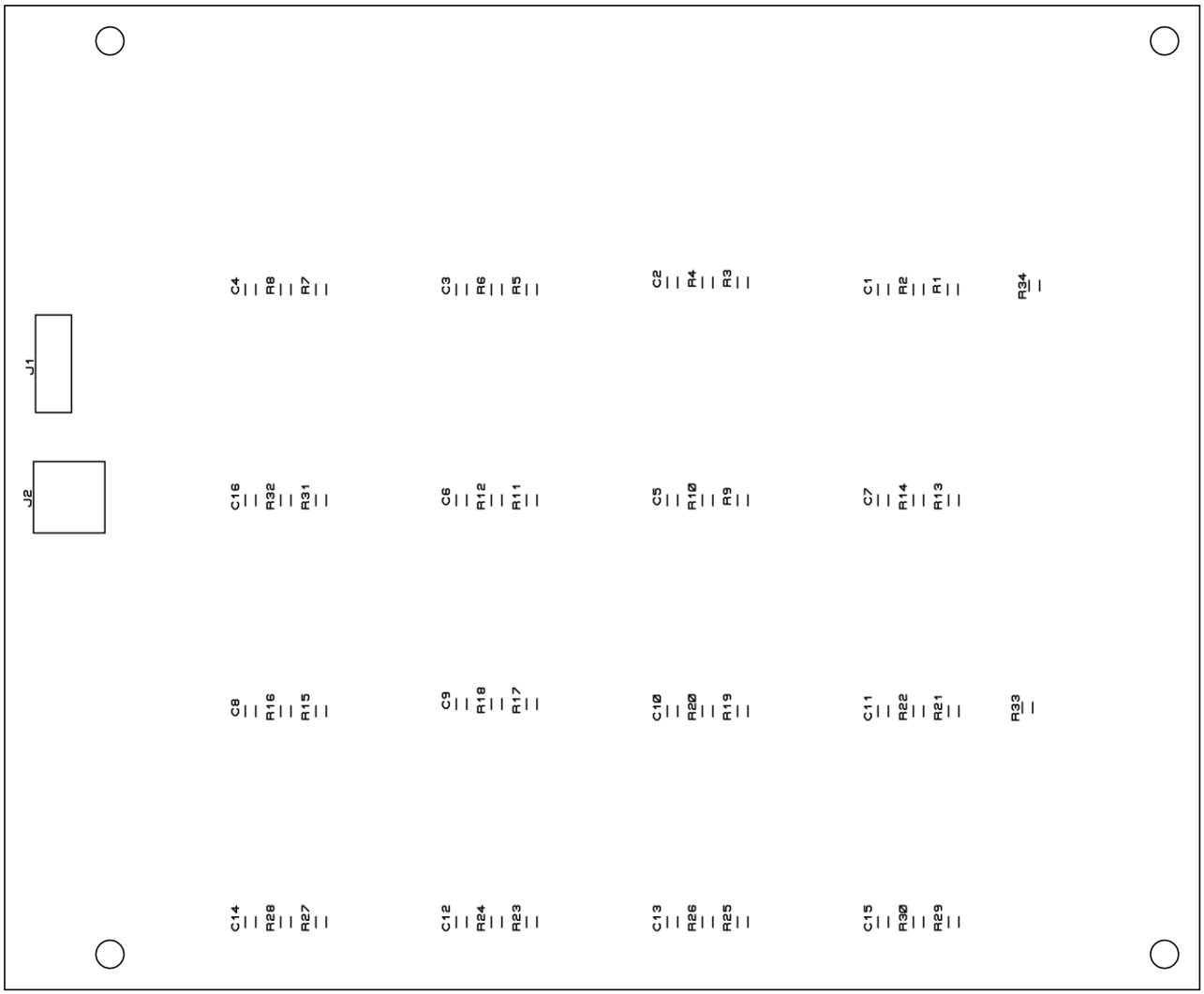
TÍTULO		
PCB Sensores – Bottom Assembly		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SE.03
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA



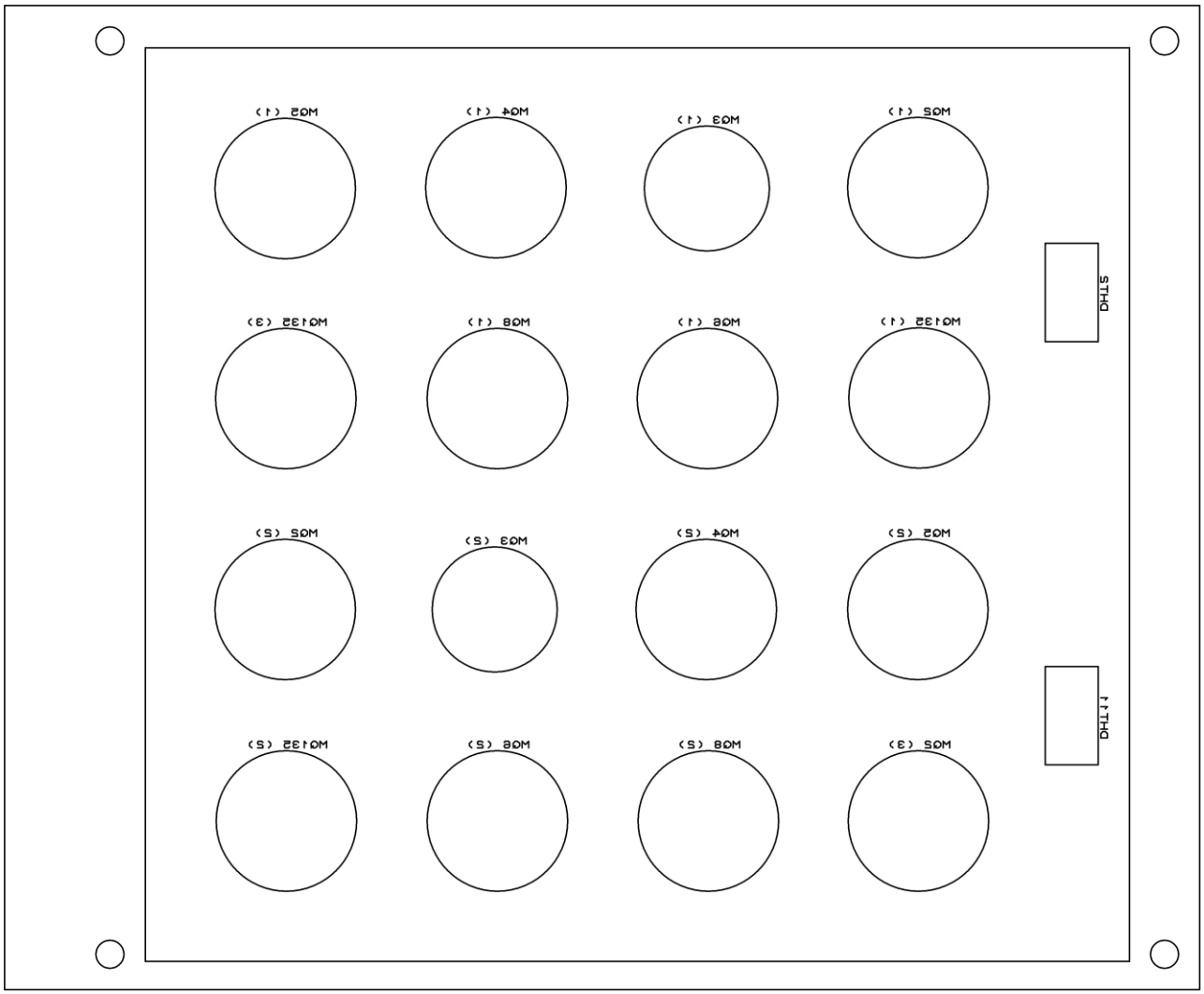
TÍTULO		
PCB Sensores – Top Copper		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SE.04
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA



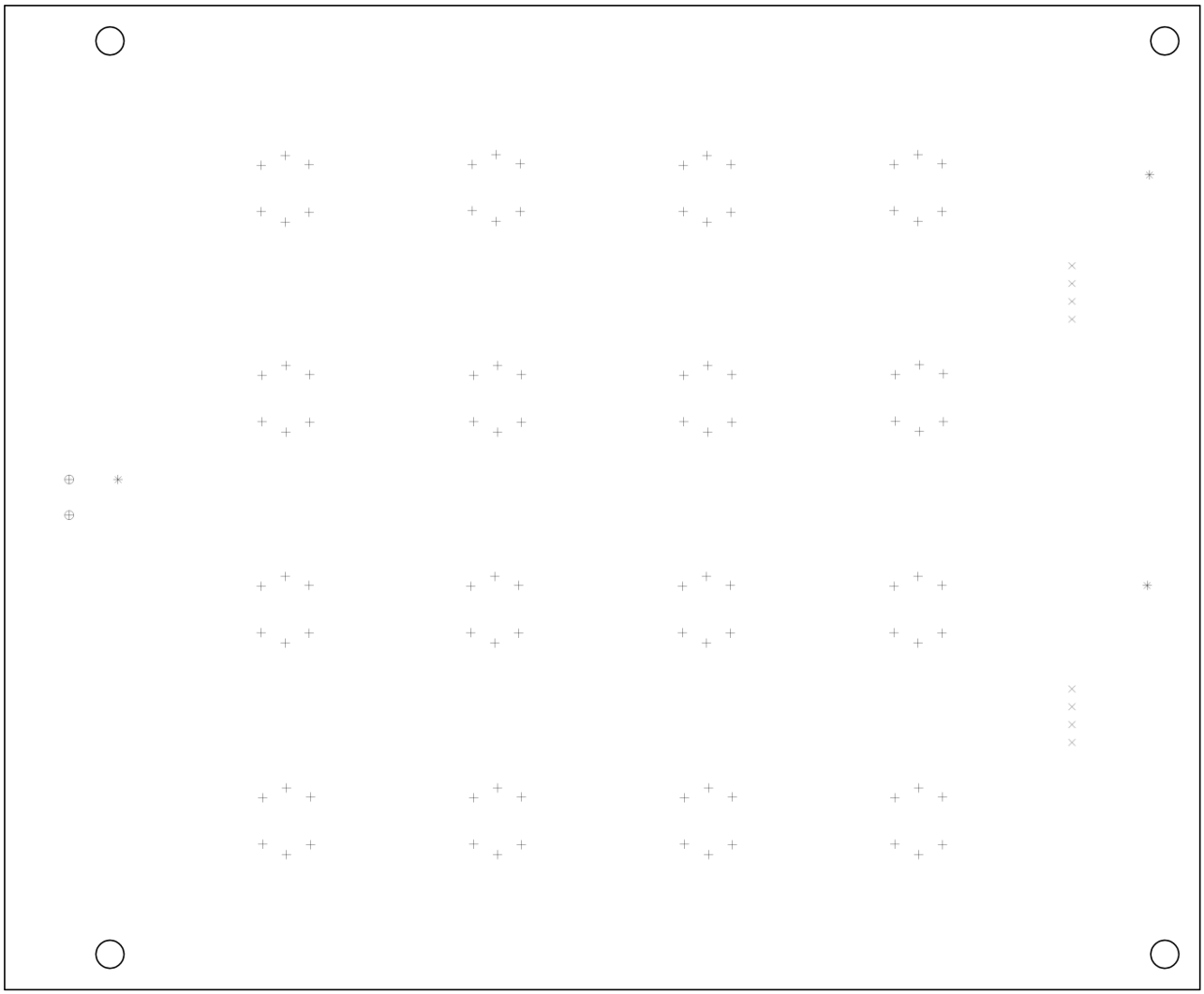
TÍTULO		
PCB Sensores – Bottom Copper		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SE.05
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA



TÍTULO **PCB Sensores – Top Silk**		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SE.06
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA



TÍTULO **PCB Sensores – Bottom Silk**		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SE.07
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA

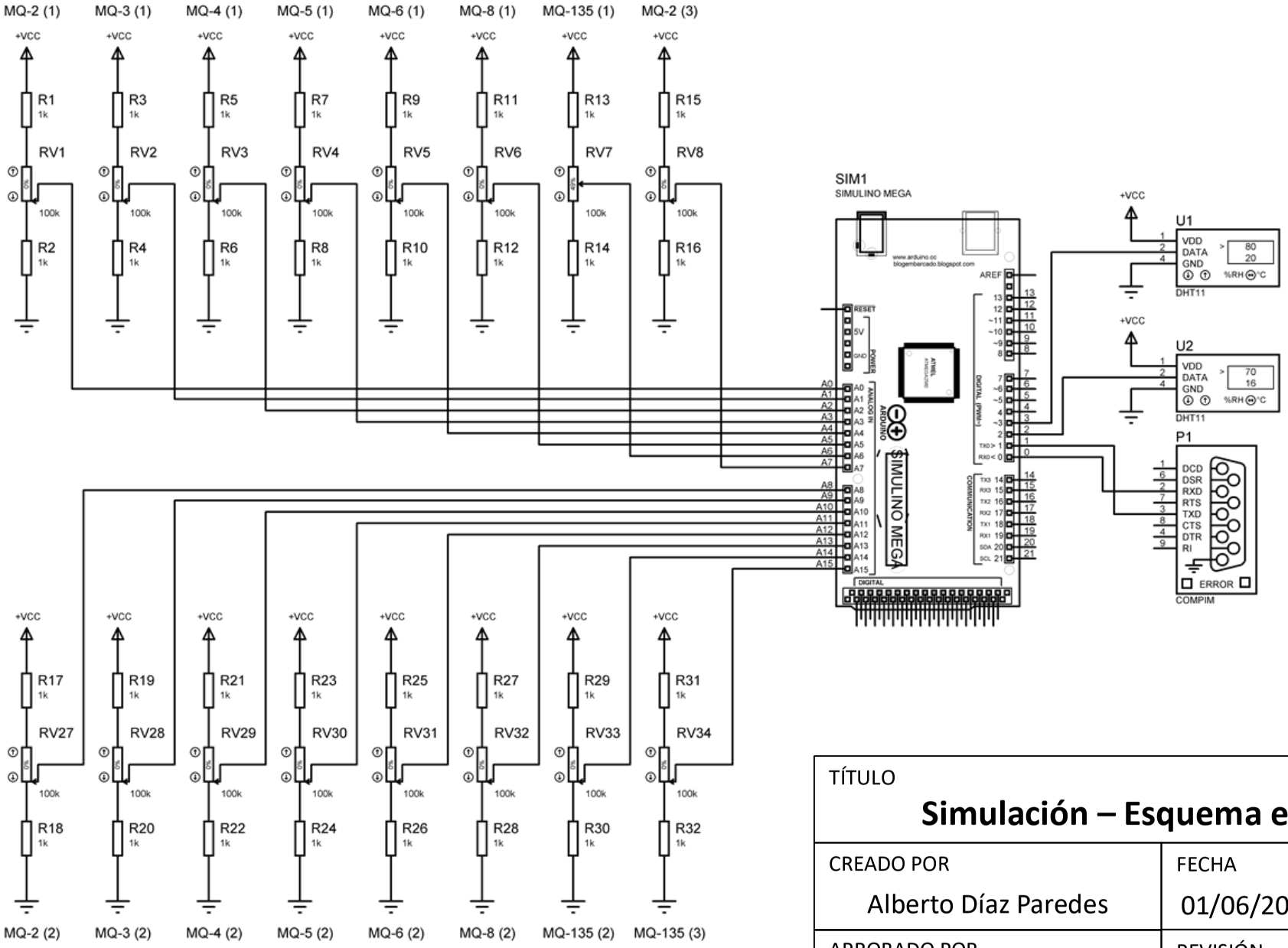


SYM	SIZE	PLATED	QTY
+	1.2MM	YES	96
×	1MM	YES	8
✳	30th	YES	3
⊕	60th	YES	2

TÍTULO

PCB Sensores – Drill

CREADO POR	FECHA	Nº DE PLANO
Alberto Díaz Paredes	01/06/2019	SE.08
APROBADO POR	REVISIÓN	
Cristian Ariel Olguín Pinatti	01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA



TÍTULO **Simulación – Esquema eléctrico**		
CREADO POR Alberto Díaz Paredes	FECHA 01/06/2019	Nº DE PLANO SI.01
APROBADO POR Cristian Ariel Olguín Pinatti	REVISIÓN 01	UNIVERSITAT POLITÈCNICA DE VALÈNCIA