UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Control Domótico Mediante Interfaz Móvil

Titulación: Grado en Ingeniería Electrónica Industrial y Automática

Alumno: Lucas Martínez Hernández

Tutor: Carlos Domínguez Montagud

Valencia, Junio 2016

El esfuerzo invertido en este trabajo, va dirigido a aquellas personas que nunca dudaron de mí, que a pesar de todo el esfuerzo y el tiempo empleado siempre estuvieron detrás dándome fuerzas, a mis padre Lucas y Conchi, y como no a Blanca por estar ahí día tras día.

Gracias.

ÍNDICE

PLIEGO DE CONDICIONES

Planos

1. Plano de planta de vivienda
2. Plano de instalación eléctrica
3. Plano de situación de actuadores
4. Plano de situación de sensores
5. Plano de instalación domótica

Anexos

1. Arduino Mega 2560
2. Cable FTP
3. Guía de uso PID para Arduino
4. Programación completa
5. Ficha técnica TIP120
6. Ficha técnica L293D
7. Ficha técnica sensor THD11
8. Ficha técnica sensor MQ-2

Bibliografía

ÍNDICE DE FIGURAS

Metodología

Diseño del sistema

Simulación

ÍNDICE DE TABLAS

RESUMEN

En este proyecto se plantea como objetivo, desarrollar un sistema de control domótico para una vivienda, mediante el uso de hardware libre Arduino, y el desarrollo de una interfaz de control y monitorización de procesos.

A medida que se desarrolle la memoria se mostrarán los distintos modos de poder solventar los objetivos planteados inicialmente. Del mismo modo se detallarán tanto los elementos utilizados para la implementación del sistema en sus distintas partes, como el funcionamiento del conjunto.

Durante la realización, se podrá apreciar que se han incorporado funciones adicionales a los objetivos originales con la finalidad de adaptar cada uno de los procesos a las necesidades del usuario, aumentando de esta forma su versatilidad. De este modo conseguimos que el resultado final sea lo más semejante a una aplicación comercial.

Posteriormente se plantearán una serie de objetivos y estudios que podrán mejorar la solución adoptada, con el fin de poder adaptarla a otras circunstancias u a otros procesos que se requieran, puesto que las mejoras y la evolución tecnológica en este campo son continuas.

ABSTRACT

In this project poses like aim, develop a system of control domotic for a house, by means of the use of free hardware Arduino, and the development of an interface of control and monitoring of processes.

To measure that develop the memory will show the distinct ways to be able to solve the aims posed initially. Of the same way will detail so much the elements used for the implementation of the system in his different parts, as the general functioning

During the realisation, will be able to appreciate that they have incorporated additional functions to the original aims with the purpose to adapt each one of the processes to the needs of the user, increasing of this form his versatility. In this way we achieve that the final result was the most similar to a commercial application.

Later they will pose a series of aims and studies that will be able to improve the solution adopted, with the end to be able to adapt it to other circumstances or to other processes that require, since the improvements and the technological evolution in this field are continuous.

1. INTRODUCCIÓN

Actualmente, es evidente que la sociedad cada vez está más influenciada por factores sociales y culturales que la impulsan al uso de la tecnología. Siendo esta un elemento prácticamente indispensable y que está en continua evolución, haciéndonos replantear los modelos ya existentes para poder mejorarlos y proponer otros que ayuden a nuestro bienestar.

Dotando de las infraestructuras de un sistema de automatización y gestión podemos conseguir para el usuario un incremento de confort y seguridad, de la misma forma que un ahorro energético considerable.

La domótica ha sido desarrollada para satisfacer este tipo de necesidades. Esta tecnología está basada en la integración de la informática y la automática en la vivienda para realizar de forma más eficiente algunos procesos cotidianos, como pueden ser el acondicionamiento térmico de la vivienda o el control de la iluminación.

En el presente trabajo se tratará de un punto de vista tanto teórico como práctico el diseño de un sistema domótico para dotar a una vivienda de una serie de beneficios, haciendo de la misma una “vivienda inteligente”, mediante el uso de hardware libre. Por ello se realizará un estudio de la situación actual de esta tecnología para poder desarrollar un posterior diseño e implementación.

Como apoyo a la programación realizada, se simularán cada uno de los procesos en una maqueta diseñada para este fin, donde se instalarán todos los dispositivos necesarios para un correcto funcionamiento.

1.1. ANTECEDENTES

Desde la introducción de la electricidad en la vivienda, el nivel de confort ha aumentado de forma progresiva con la introducción de nuevos electrodomésticos capaces de realizar las tareas cotidianas de forma autónoma.

La siguiente evolución es la domótica, este tipo de tecnología apareció en 1978 con la aparición del producto X10 por la compañía Pico *Electronics of Glenrothes*, que permitía un control remoto de dispositivos eléctricos, este fue el primer paso en la tecnología domótica. Posteriormente con la aparición de redes inalámbricas (WIRELESS) se desarrollaron otras nuevas formas de comunicación de entre dispositivos, de esta forma se facilitó una mejor comunicación entre los dispositivos y el usuario. Así mismo se mejoraron los sistemas de comunicación físicos (WIRED), lo que permitía una rápida transmisión y recepción de datos para poder optimizar estos procesos.

En la actualidad la domótica es capaz de optimizar prácticamente cualquier proceso de una vivienda, mejorando el confort del usuario.

1.2. VENTAJAS E INCONVENIENTES DEL USO DE LA DOMOTICA

VENTAJAS

- La mayor ventaja que posee la introducción de este tipo de tecnología al instalarse, tanto en viviendas, como en otro tipo de entornos, es el ahorro energético que produce, puesto que al programar los aparatos eléctricos cotidianos conseguimos que dejen de funcionar cuando no sean necesarios, del mismo modo que el control de la temperatura en la vivienda permite que el sistema no utilice más energía de la necesaria para establecer una temperatura de confort.
  Aunque no produzca un gran ahorro por sí solo, el sumatorio de todos ellos si consigue reducir considerablemente el consumo eléctrico.

- La seguridad es también un factor importante, ya que nos ofrece la tranquilidad de poder controlar desde el acceso a la vivienda hasta monitorización inalámbrica de estancias, incluso presencia de gases tóxicos o de humos, por lo que es una característica a tener en cuenta si se desea este tipo de instalación.

- Ofrece un incremento de la comodidad para el usuario ya que le permite monitorizar y realizar de forma remota acciones que antes debían ser presenciales, como el encendido de luces o la subida de persianas.

INCONVENIENTES

- La instalación de este tipo de sistemas conlleva una inversión inicial alta.
- Dependiendo del tipo de arquitectura del sistema, si se produce algún tipo de avería podría perjudicar a todo el conjunto, bloqueando toda la red.
- Dependiendo del tipo de transmisión de datos, si se trata de inalámbrica o mediante cableado, presenta una serie de desventajas, como podría ser la aparición de interferencias para el caso de los sistemas inalámbricos o la ralentización si se trata de transmisión mediante cable.
- Es necesario de sistemas de seguridad para evitar que gente externa tenga acceso a estos controles.

1.3. OBJETIVOS

1.3.1. Objetivos Generales

En el presente proyecto se pretende realizar el diseño e implementación de software y hardware de un sistema domótico, mediante el uso de Arduino, para una vivienda unifamiliar que dispone de:

- Dos cuartos de baño

- Un dormitorio
- Una cocina
- Un salón

Para simular todos estos procesos se dispone de una maqueta a escala de la vivienda, en la que se podrán instalar todos los componentes necesarios para que se asemejen a procesos reales.

1.3.2. Objetivos Específicos

En primer lugar se especificarán las necesidades y exigencias que tiene el usuario, y como poder satisfacerlas, para poder incrementar el nivel de confort de forma considerable, adaptándonos a la vivienda.

En segundo lugar se realizará un estudio de las posibilidades tecnológicas actuales y cuáles de ellas son las que mejor se adaptan a nuestros requerimientos.

Se seleccionará un método de control para cada una de las variables del sistema, así mismo se desarrollara una interfaz de control grafica inalámbrica que permita al usuario controlar y monitorizar su vivienda.

Se propondrán distintos tipos de arquitecturas de control domótico para poder seleccionar la más apropiada para nuestro sistema.

Esta interfaz deberá estar enlazada con el controlador central a tiempo real, para que el usuario pueda tomar decisiones en función de la situación de la vivienda.

Una vez se hayan implementado todos los componentes del sistema se realizará una comprobación general para establecer si se ha producido algún fallo, cuando cada una de las partes trabaje en conjunto, si es así se procederá a volver a implementar dicha parte y comprobar el sistema de nuevo.

A lo largo del desarrollo del proyecto, es posible que se incorporen otros objetivos, puesto que sean necesarios para cumplir los objetivos principales.

2. METODOLOGÍA

En los siguientes apartados, estudiaremos algunas de las herramientas que se van a utilizar, así como el abanico de posibilidades que se nos ofrece a la hora de poder seleccionar, tanto métodos de trabajo como componentes que actualmente cumplan las especificaciones necesarias para la elaboración del proyecto.

2.1. REQUERIMIENTOS DEL SISTEMA

A continuación se detallaran todas las necesidades y exigencias del usuario para un posterior diseño del sistema.

2.1.1. Control de iluminación

- Se requiere de un control total de la iluminación de la vivienda de forma remota, de este modo el usuario puede ser capaz de encender o apagar luces desde cualquier estancia en la que se encuentre.

- En segundo lugar, es necesario de la automatización de alguna de las lámparas para que se ilumine la estancia si se detecta la presencia del usuario.

- Otros de los requerimientos establecidos es la instalación de luces dimerizadas, es decir que pueda regularse su intensidad tanto de forma manual como remota, este tipo de iluminación se encontrará solamente en el salón y en el dormitorio.

2.1.2. Control térmico

Se desea disponer de un sistema de control térmico para que se autor regule de forma automática, manteniéndose siempre constante a

la temperatura que el usuario haya preestablecido como de confort, del mismo modo también se podrá modificar de forma manual.

El usuario podrá seleccionar de forma remota si desea que el sistema se mantenga constante o de otro modo, seleccionar el nivel de intensidad del sistema de ventilación.

2.1.3. Control Remoto

El usuario requiere de un sistema de control remoto mediante su dispositivo móvil, este debe funcionar de forma inalámbrica en todo el perímetro de la vivienda, e incluso se estudiará la posibilidad de que pueda funcionar en cualquier lugar fuera de la misma.
Este control debe constar de un sistema de autentificación de entrada para evitar que cualquier persona que acceda a la vivienda pueda tener acceso al mismo.

2.1.4. Automatización de persianas

Se requiere de un sistema de subida y bajada de persianas, tanto automáticas como manuales, el sistema automático dependerá del nivel de luz del exterior, estando subidas por la mañana y bajadas cuando el nivel de luz sea reducido.
El control de las persianas también podrá ser controlado de forma remota por el usuario.

2.1.5. Control de acceso y alarma

La vivienda contará con un control de acceso que irá enlazado a una alarma sonora, esta alarma se accionará si el sistema está activo y es detectada una presencia en la estancia, bajándose de forma instantánea las persianas de la vivienda como método de seguridad.
Para activar este control de acceso será necesario, que el usuario introduzca un código mediante un panel numérico situado en el recibidor de la vivienda.

Del mismo modo si la alarma es accionada por una presencia será necesario introducir esta contraseña para poder deshabilitarla.

2.1.6. Detector de gases

Será necesaria la instalación de un detector de presencia de humos y gases que eviten incendios y fugas, serán instalados en la cocina, puesto que es la estancia más propensa para detectar su aparición.

Estos detectores irán conectados a una alarma acústica que alertará en el caso de que existan niveles de presencia de gases que puedan resultar nocivos para la presencia en la vivienda de personas.

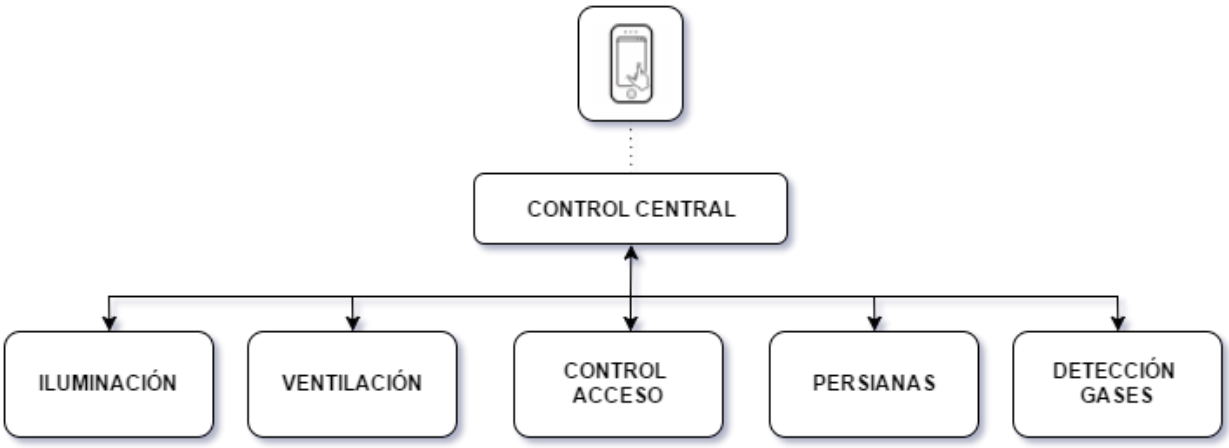


Figura 2.1: Diagrama de requerimientos del sistema

En el siguiente diagrama (Fig.2.1) se observan cada uno de los bloques que el usuario considera necesarias y que serán diseñadas e implementadas en apartados posteriores.

2.2. ARDUINO

Arduino es una plataforma de hardware libre basada en un microcontrolador Atmel AVR y que posee un entorno de programación muy sencillo y didáctico.

Arduino presenta las siguientes ventajas por la cual ha sido escogido para la realización de este proyecto:

- Sistema abierto: Arduino es una plataforma tanto de código como de hardware abierto, lo que permite fabricar un modelo reducido más económico, puesto que se cuenta con toda la información, archivos y librerías de diseño.

- Fácil programación: Arduino tiene la ventaja de no necesitar ninguna tarjeta de programación si no que la placa conecta mediante serial utilizando una conexión USB.
  Para su programación posee un código denominado *Processing*, que aun siendo de alto nivel facilita considerablemente su aprendizaje, esto no limita a que también pueda utilizarse otros lenguajes de programación distintos.

- Económico: Existen multitud de modelos capaces de adaptarse a las necesidades del programador, todos ellos a muy bajo coste, además de existir otros modelos versionados por distintos fabricantes que ofrecen las mismas características.

- Existencia de documentación: Existe multitud de documentación y ejemplos relacionada con la programación de Arduino lo que facilita considerablemente su aprendizaje.

- Librerías: Una de las mayores ventajas que se nos presenta es que esta plataforma es que posee librerías de libre distribución para prácticamente cualquier componente que se quiera utilizar.

- Accesorios compatibles: Otra de las herramientas que hacen de Arduino una elección lógica, es que posee multitud de componentes compatibles, por lo que facilita poder realizar pruebas de forma más sencilla. Así mismo, existen una gran cantidad de placas y periféricos capaces de poder ser acopladas a la tarjeta, y de este modo poder realizar funciones específicas, como podrían ser Teclados, LCD, sensores, Ethernet, etc.

2.2.1. MODELOS Y CARACTERISTICAS DE ARDUINO

En primer lugar antes de seleccionar que tipo de modelo utilizaremos para el desarrollo del proyecto, debemos saber de forma aproximada el número de periféricos y variables de la que vamos a necesitar. Con estos podemos adquirir una idea de la cantidad de pines analógicos y digitales necesarias para nuestro trabajo, de esta forma podemos descartar algunas placas más sencillas que no cumplan con nuestras necesidades.

También es necesario hacer una estimación del tamaño del código que se irá a generar, un programa muy extenso demandará de una mayor memoria flash, por lo que será indispensable seleccionar una placa adecuada.

En los Arduinos oficiales podemos diferenciar entre dos tipos de, los de 8 y de 32 bits, con microcontroladores basados en ATmega AVR y SMART basados en ARM de 32 bits, estos últimos proporcionan un rendimiento superior. Por ello es fundamental seleccionar un soporte que se adapte adecuadamente a nuestras exigencias antes de comenzar a realizar cualquier tarea.

Si se escoge un modelo con unas prestaciones menores a las requeridas, el resultado será que los procesos se ralentizarán y el funcionamiento general será inadecuado, por lo que es fundamental una buena elección como base para la realización del resto de tareas.

En la siguiente imagen *(Tabla 1)* se muestra los modelos más conocidos y algunas de sus características más significativas, a partir de las cuales

deberemos seleccionar los más adecuados en función de nuestras necesidades.

Característica de Arduino	UNO	Mega 2560	Leonardo	DUE
Tipo de microcontrolador	Atmega 328	Atmega 2560	Atmega 32U4	AT91SAM3X8E
Velocidad de reloj	16 MHz	16 MHz	16 MHz	84 MHz
Pines digitales de E/S	14	54	20	54
Entradas analógicas	6	16	12	12
Salidas analógicas	0	0	0	2 (DAC)
Memoria de programa (Flash)	32 Kb	256 Kb	32 Kb	512 Kb
Memoria de datos (SRAM)	2 Kb	8 Kb	2.5 Kb	96 Kb
Memoria auxiliar (EEPROM)	1 Kb	4 Kb	1 Kb	0 Kb

Tabla 1: Comparativa modelos de Arduino

2.2.2. ARDUINO UNO

Arduino UNO, en este caso la versión R3 es una de las placas más conocidas y utilizadas para el aprendizaje, este modelo fue el utilizado para realizar las primeras pruebas de programación puesto que se trata de un hardware muy compacto y sencillo.

Figura 2.2: Frontal y trasera Arduino modelo UNO R3

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm

Tabla 2: Características Arduino UNO R3

2.2.3. ARDUINO MEGA

Este modelo fue el seleccionado para un posterior desarrollo del proyecto, ya que nos proporciona un número de pines y una memoria flash muy superior al modelo UNO, siendo necesarios puesto que se acoplarán un gran número de periféricos.

Figura 2.3: Frontal y trasera Arduino modelo Mega

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
Length	101.52 mm
Width	53.3 mm

Tabla 3: Características Arduino Mega 2560

2.3. ENTORNO DE PROGRAMACIÓN

Para el diseño e implementación del sistema, se ha sido necesario el uso de distintas plataformas y herramientas diferentes en cada uno de los bloques de diseño, en función de las necesidades de cada uno.

A continuación mostraremos algunos de ellos, así como las características principales y sus protocolos de funcionamiento.

2.3.1. IDE ARDUINO

A pesar de existir más de un entorno de programación para Arduino, el más utilizado, y el seleccionado para esta ocasión es el propio de Arduino, este software que puede obtenerse de forma gratuita desde la página oficial *(Arduino 1.6.5).*

Se trata de un entorno de programación muy útil que facilita al programador poder depurar de forma rápida el código y poder acceder de forma sencilla a multitud de librerías que el propio programa proporciona, así como ejemplos ya implementados.

Para desarrollar nuestro proyecto, en primer lugar se genera el código para cada uno de los bloques, comprobando posteriormente que no existan fallos, el propio programa analiza las líneas introducidas y señala si existe algún error en la programación, posteriormente se compilará, volcando todo nuestro código a la memoria del microcontrolador.

Arduino también consta de un sistema que nos permite una comunicación entre nuestro PC y el hardware mediante el puerto USB, se trata del monitor Serial, este facilita tanto la lectura de datos de periféricos como el envío valores desde el PC hasta Arduino.

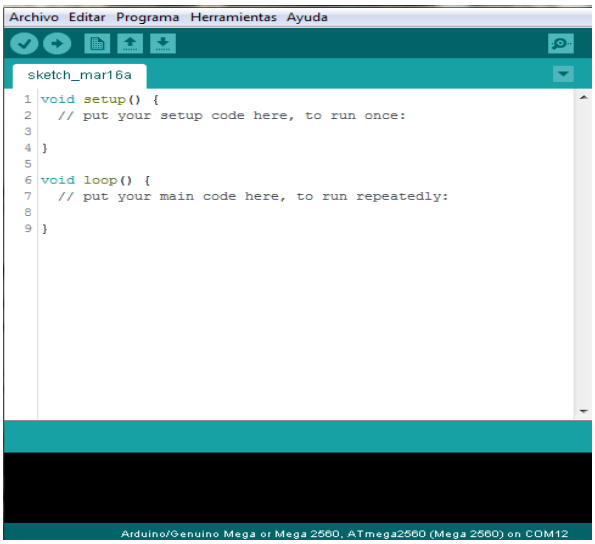


Figura: 2.4: IDE Arduino

2.3.2. APP INVENTOR

App Inventor es una página web de desarrollo de aplicaciones para entornos android, mediante esta página se le permite al programador desarrollar distintas aplicaciones e interfaces de forma muy sencilla sin la necesidad de conocer la programación en lenguaje Java u otras similares, puesto que se presenta un sistema de programación en módulos que hace de su aprendizaje algo rápido y sencillo, dando buenos resultados, de este modo se pretende crear una interfaz de usuario tal y como se requería en los objetivos iniciales.

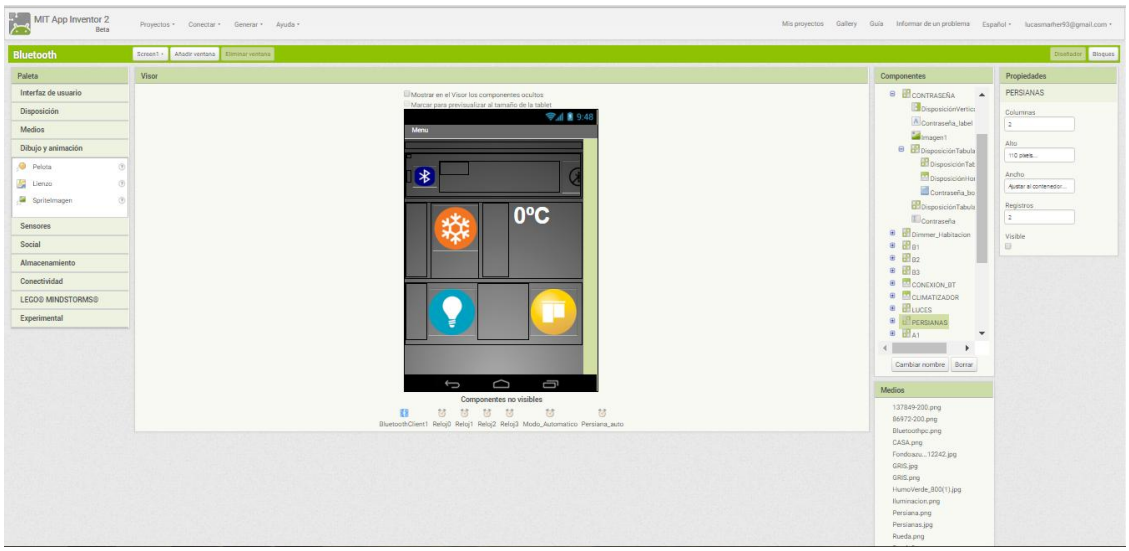


Figura: 2.5: App Inventor Menú diseño

Este tipo de página web es idónea para realizar este tipo de proyectos si no se disponen de conocimientos previos de Java u otros métodos de programación en android, puesto que es un aprendizaje relativamente rápido y que ofrece unos resultados muy aceptables.

En la siguiente imagen *(Fig. 2.5)* podemos observar una de las dos partes de las que se compone este método de generación de aplicaciones, se trata de la parte de diseño, es decir lo que posteriormente se verá en nuestro dispositivo móvil o tablet, es en este apartado donde se mejora el diseño de cada una de las partes a la que el usuario podrá acceder. Se debe tener en cuenta que el tiempo de aprendizaje para el usuario debe ser muy corto, por lo

que a la hora del diseño grafico de la interfaz sea fácil de relacionar con las acciones que se pretendan realizar, al mismo tiempo que visualmente atractivo.

Cada uno de estos botones por sí solo no realiza ningún tipo de acción, por lo que será necesario programarlos, para ello se va a mostrara el segundo de los bloques de los que consta este sistema.

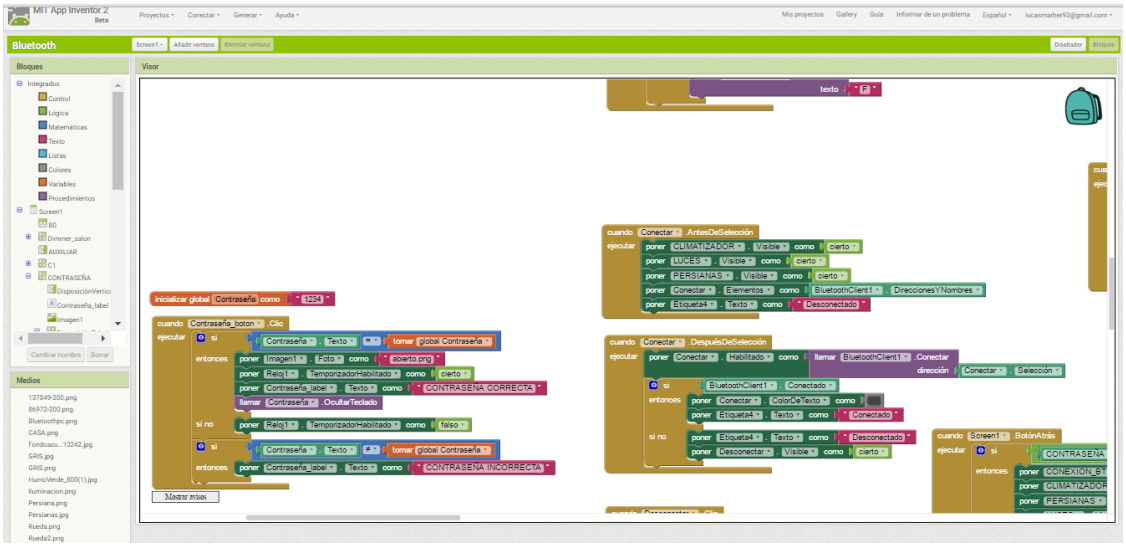


Figura: 2.6: App Inventor Menú bloques

En esta segunda parte *(fig.2.6)*, denominada bloques, se programa mediante módulos, las funciones de cada uno de las componentes que se han incluido en la parte de diseño, a diferencia de otros métodos en los que es necesario generar líneas de código.

Este sistema de bloques resulta bastante sencillo de aprender puesto que utiliza rutinas y condiciones muy semejantes a los utilizados en otros métodos de programación, como podría ser el uso de If, for, while, etc.

Además algunas de las estructuras más complejas podemos encontrarlas ya predefinidas, como podrían ser las conexiones ya sea bluetooth o wifi, estos bloques pueden encontrarse en multitud de paginas relacionadas con aplicaciones en android para Arduino.

Este sistema de programación tiene también como ventaja que permite probar la aplicación tanto en emuladores como directamente en el dispositivo, puesto

que dispone de un formato de descarga en (.apk) *(fig.1.7)* ya sea generando un archivo o mediante código QR, lo que facilita su rápida instalación.

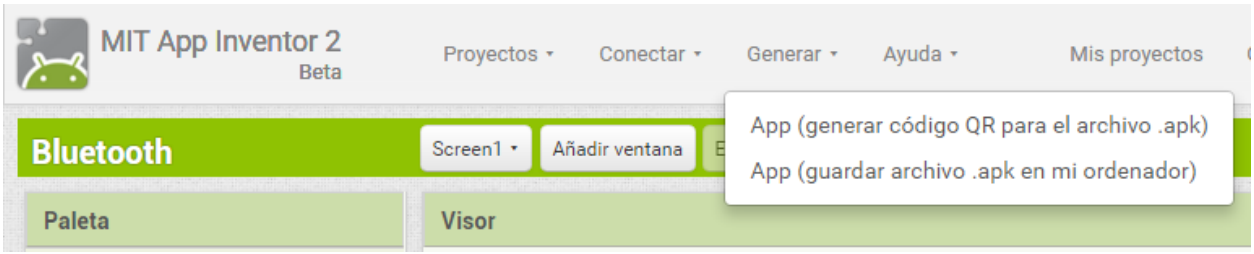


Figura 2.7: Generación de aplicaciones App Inventor

2.3.3. MATLAB

Matlab es un software de apoyo matemático que utiliza para su funcionamiento un lenguaje de alto nivel. Puesto que esta herramienta es ampliamente compleja solamente se ha estudiado las aplicaciones útiles para nuestro proyecto, aunque este software ofrezca muchas más posibilidades.

La versión de la que se dispones es la de MatlabR2016a, ya que cada versión de este software posee aplicaciones diferentes, esto depende de cuales vayan a ser las funciones que se necesiten.

Algunas de las herramientas generales de Matlab, nos permiten realizar las siguientes funciones:

- Análisis numérico
- Cálculo matricial
- Representaciones 2D y 3D
- Gráficos
- Procesamiento de señales
- Diseños de control
- Identificación de sistemas

En este caso la versión que se utilizará de este programa permite monitorizar tanto a tiempo real, como los datos anteriormente obtenidos de todos los sensores y actuadores que sean necesarios.

El principal uso que vamos a darle a este software, es el de diseño del sistema de ventilación, ya que posee herramientas que permiten la identificación de sistemas y el diseño de controladores lo que nos será de gran uso para facilitar los cálculos posteriores.

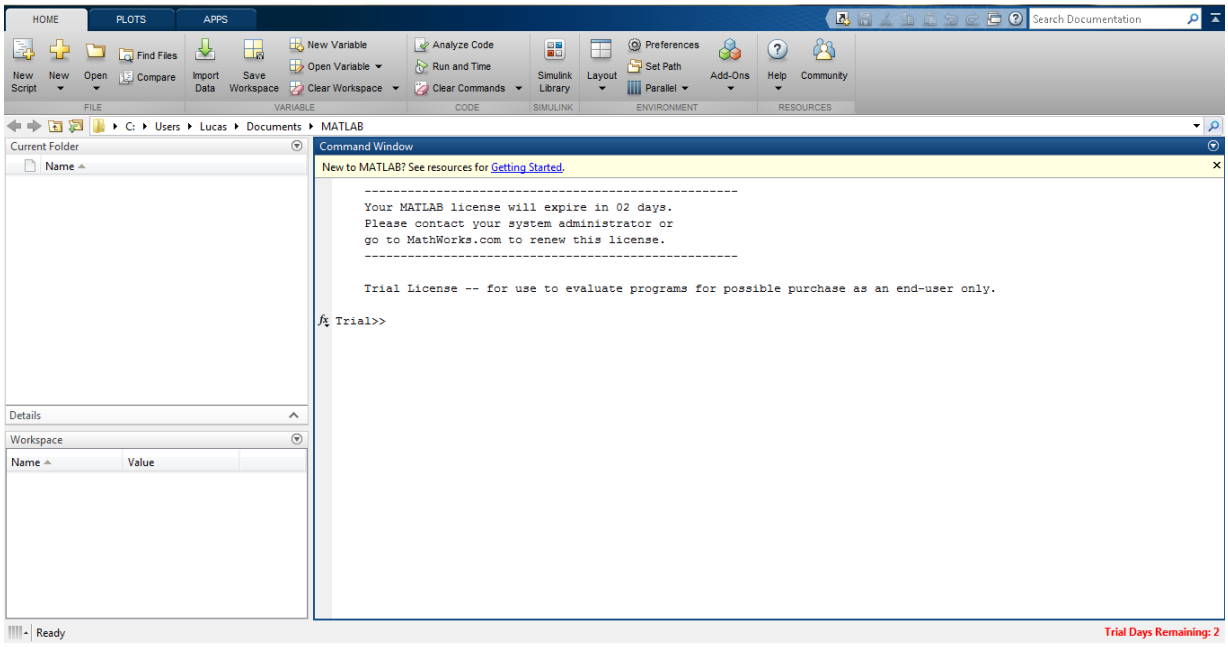


Figura: 2.8: Matlab

Matlab incluye una serie de aplicaciones internas *(fig 2.9)* que permiten realizar tareas especificas, como en este caso el diseño de reguladores PID y otros recursos que se irán viendo más adelante cuando aparezcan necesidades que requieran el uso de de este tipo de herramientas.

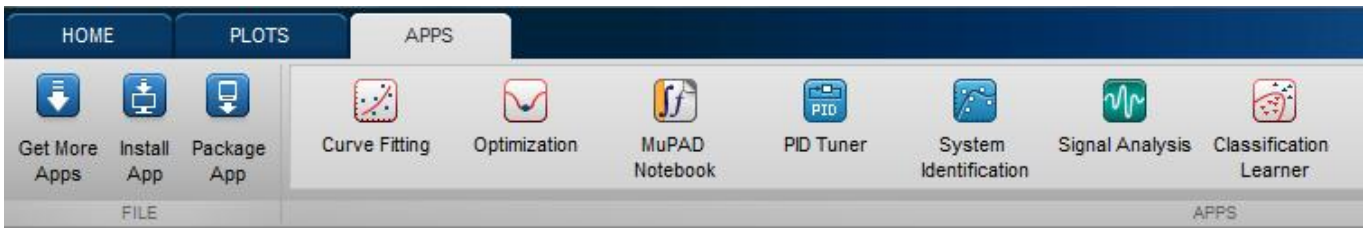


Figura 2.9: Apps Matlab

2.4. ARQUITECTURAS DE CONTROL DOMOTICO

Denominamos arquitecturas de control a la disposición de los componentes del sistema entre sí, es decir, de cómo se sitúe el núcleo central de control con respecto al resto de componentes periféricos, tantos sensores como actuadores con respecto al mismo.

A continuación se nombrarán algunos de los más comunes así como sus características principales para posteriormente poder seleccionar el más adecuado a nuestras necesidades.

2.4.1. ARQUITECTURA CENTRALIZADA

Este tipo de estructura de control domótico, consiste en situar el controlador central, el cual es el encargado de recibir toda la información de los dispositivos periféricos del sistema, es decir sensores y actuadores, para poder procesar dicha información y generar la respuesta pertinente para los actuadores.

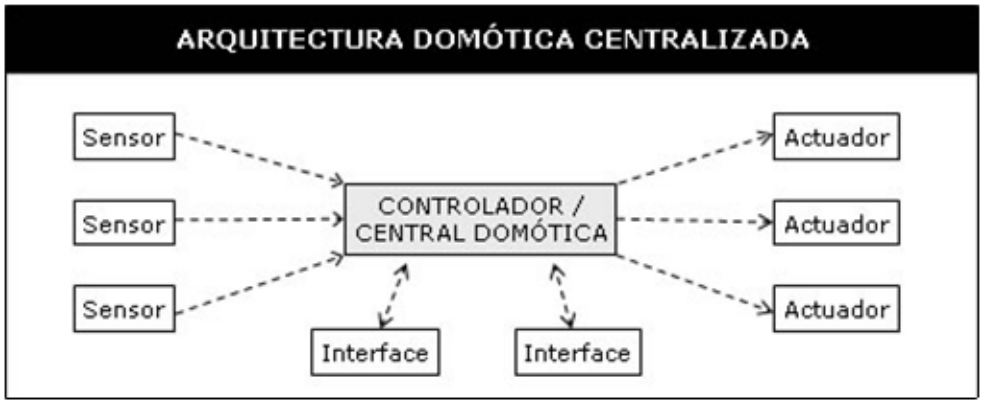


Figura 2.10: Arquitectura domótica centralizada

- La mayor característica de este tipo de distribución, es que posee una gran potencia, ideal para sistemas de gran complejidad en el que se deberán procesar gran cantidad de datos.

- Al ser un sistema centralizado tiene como inconveniente que toda la responsabilidad del funcionamiento del conjunto la posee el sistema central, si este falla todos los demás también lo harán, aunque este aspecto también caracteriza otros tipos de arquitectura que se verán posteriormente.

2.4.2. ARQUITECTURA DISTRIBUIDA

Este tipo de arquitectura se caracteriza porque cada uno de los dispositivos dispone de un procesador propio que es capaz de realizar cada una de las funciones por las que ha sido programado, realizando funciones específicas, actuando en función de la información entrante a partir del bus de datos, mediante los mismos, también se realiza la conexión con los demás dispositivos enviando información entre ellos tanto para sensores, actuadores como interfaces.

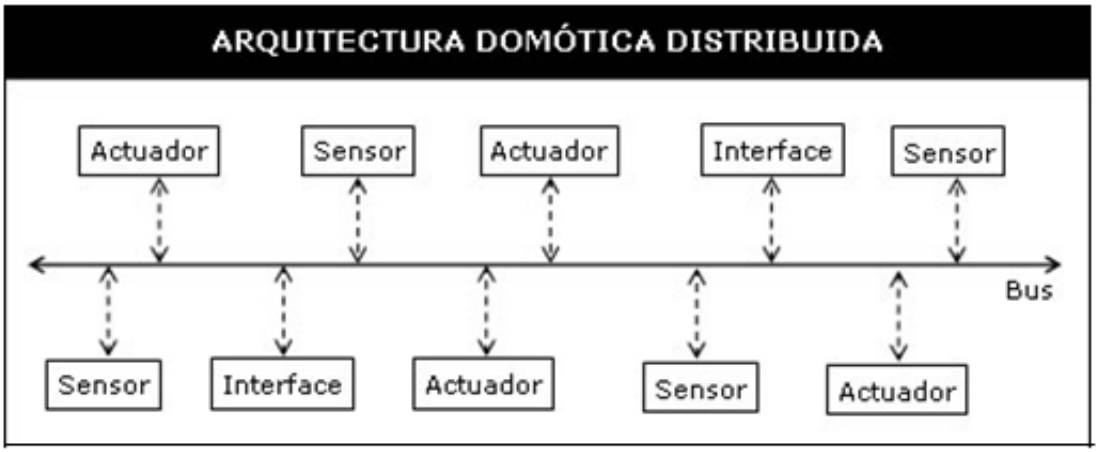


Figura 2.11: Arquitectura domótica distribuida

- Este tipo de distribución tiene como ventaja principal que cada uno de los dispositivos posee autonomía propia, lo cual proporciona una mayor

fiabilidad al sistema, por lo que si un dispositivo deja de funcionar no afecta al conjunto.

- Otra gran ventaja que presenta esta arquitectura es que es adecuada para lugares que no requieran de reformas puesto que no necesita de distribución de cableado hasta el cuadro central, por lo que facilita considerablemente la instalación.

- El principal inconveniente que presenta, es que al ser un sistema con sensores y actuadores distribuidos, estos solamente están programados para realizar funciones específicas, por lo que no se puede esperar que el sistema presente una gran potencia para realizar otro tipo de acciones más complejas.

2.4.3. ARQUITECTURA MIXTA

Este tipo de distribución combina características de los sistemas distribuidos y centralizados, por lo que puede disponer de un controlador central o un conjunto de controladores descentralizados, todos los dispositivos periféricos como sensores ya actuadores pueden procesar información del entorno, así mismo pueden compartir la información entre ellos si necesidad de que esta pase por el controlador central.

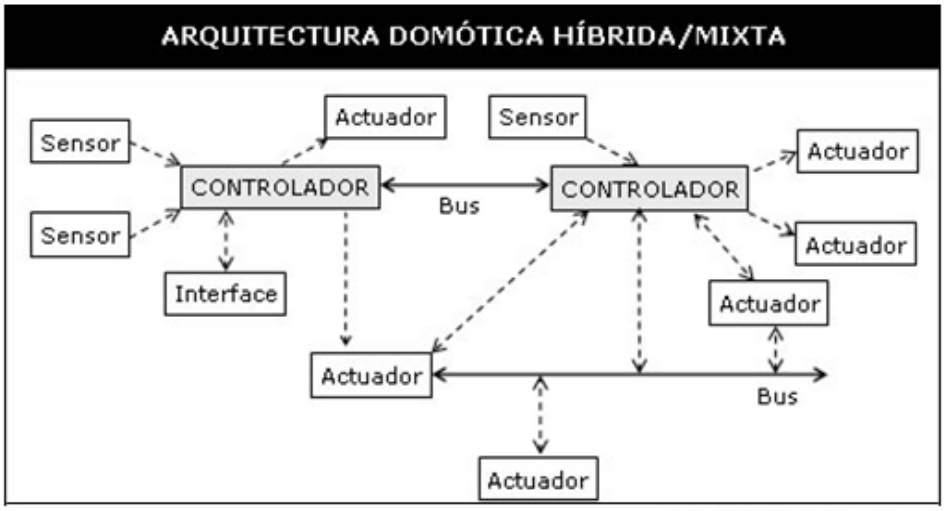


Figura 2.12: Arquitectura domótica mixta

- La ventaja que nos ofrece este tipo de arquitectura es que, a diferencia que los anteriores, si uno de los controladores principales deja de funcionar no conlleva un fallo general del sistema como sucedería con el sistema centralizado, o tenemos una potencia de procesamiento limitada como ocurre con los sistemas distribuidos.

- En este caso, los sensores y actuadores pueden procesar información sin dependencia del controlador central, pero sin la necesidad de descartar su instalación, por lo que posee una potencia superior a sistemas como los distribuidos de este modo presenta las características que aventajas a los sistemas anteriores.

2.5. MEDIOS DE INTERCONEXION

2.5.1. MEDIANTE CABLEADO

- **xDSL:** Este método es el más utilizado hoy en día para realizar las conexiones a internet, consiste en la utilización de un modem individual, puesto que este es necesario debido a que las redes telefónicas son algo analógicas y no permiten alcanzar velocidades de conexión óptimas. Dependiendo del tipo de necesidad se puede elegir entre ADSL, IDSL, VSDL, etc. Esta tecnología permite el flujo de información a alta velocidad.

Figura 2.13: Cable xDSL

- **Fibra Óptica:** Este método de transmisión se compone de filamentos ya sean de vidrio o plástico de un espesor que ronda entre 10 y 300 micrones, los cuales llevan información en forma de haces de luz de un extremo al otro de forma rápida y eficaz sin interrupciones. La fibra óptica pude usarse como los conectores de cobre convencionales, tanto en pequeños lugares como en redes de mayor tamaño como podrían ser líneas urbanas instaladas por compañías telefónicas.

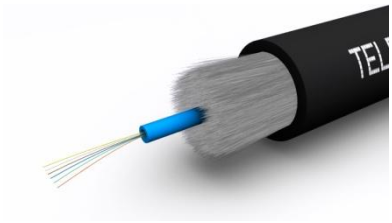


Figura 2.14: Fibra óptica

- **Cable Coaxial:** El cable coaxial está compuesto por un hilo de cobre en el núcleo, protegido por un material aislante y por una malla de metal trenzada.

Gracias a esta protección es el utilizado para cubrir grandes distancias a gran velocidad, aunque su uso más común es para la realización de instalaciones básicas.

Figura 2.15: Cable coaxial

- **Par trenzado:** Consiste en dos hilos de cobre enlazados entre sí, protegidos mediante un aislante. Esta disposición de los cables, se utiliza para reducir ruidos e interferencias eléctricas que puedan ocasionarse, e interfieran en la señal.

  Por lo general, este cable se utiliza con varios pares trenzados agrupados entre si y protegidos mediante una funda de protección. Este sistema, es adecuado para conexiones simples y de cortas distancias puesto que presenta una baja inmunidad al ruido.

Figura2.16: Par trenzado

2.5.2. INALAMBRICOS

- **Wifi:** Este métodos de conexión consistente en la comunicación mediante internet de forma inalámbrica. Para su uso solo se requiere de la instalación de un modem en la vivienda, este envía una señal mediante ondas que permite a los dispositivos que posean este tipo de conexión conectarse a la red en un rango máximo de entre 100 y 150 metros. Si el dispositivo no consta de esta tecnología, actualmente han aparecido terminales USB específicos que facilitan poder conectarse a la red.
- **Bluetooth:** La tecnología Bluetooth es una tecnología inalámbrica que permite una comunicación entre dispositivos digitales. El alcance es algo inferior al de otras tecnologías similares. Actualmente podemos clasificarlos en función de su alcance.

  - *Clase I* Alcance máximo 100 metros.

- *Clase II* Alcance máximo 25 metros.
- *Clase III* Alcance máximo 1 metro.

No requiere de instalación previa puesto que el módulo emisor es de fácil y rápida instalación. Además la mayor parte de dispositivos de última generación constan de este tipo de conexión, y del mismo modo que con las conexiones wifi, existen terminales USB que permiten automáticamente la conexión Bluetooth al dispositivo al que se conecte.

- **GPRS:** Estas siglas son la abreviatura de '*General Packet Radio'*, este método de transferencia de datos, consistente en el envío y recepción de paquetes de datos, utilizando para ello la red telefónica por satélite. Este tipo de tecnología, actualmente ha dejado paso a otras como podría ser la conexión 3G o 4G que permiten conexiones más rápidas, puesto que solamente ofrece una velocidad de transmisión de datos de 56kbps.

- **Radiofrecuencia*:*** Este método de transmisión de datos, está compuesto por un emisor y un receptor de señal, que permite la comunicación entre dispositivos a través de señales de radio, para ello es necesario que cada uno de las partes conste de estos dos componentes, la emisión de datos es de forma unidireccional, por lo que el receptor no puede enviar ningún tipo de señal al emisión y viceversa, por lo que limita su uso en algunos caso.
  Del mismo modo, la señal emitida es muy propensa a sufrir interferencias de elementos externos, dificultando de este modo una buena recepción de datos.

- **Zigbee**: Se trata de un conjunto de protocolos de alto nivel que permiten la comunicación inalámbrica mediante radiodifusión de datos. Este tipo de transmisión se realiza sin necesidad de una programación avanzada

ya que solamente requiere de aproximadamente 130kB de almacenamiento para la programación del código fuente.
Se trata de una tecnología con un coste muy reducido y que busca un ahorro energético considerable.
Dentro de estos dispositivos encontramos tres categorías distintas de nodos:

- **Coordinador Zigbee**: Es considerado el modulo de control central de toda la red.
- **Router Zigbee**: Es el responsable de conectar los nodos entre sí para poder ejecutar el código.
- **Dispositivo Zigbee**: Son los encargados de la recepción de la información y comunicarla al nodo central, la mayor ventaja de este dispositivo es que permanece desconectado durante los periodos de inactividad para reducir el consumo.

2.6. OTROS METODOS DE CONTROL

2.6.1. X-10

Este método se basa en la transmisión de información a través de la corriente eléctrica, esto permite controlar los dispositivos eléctricos de una vivienda a través de la red de alimentación.

Para poder enviar información a través del suministro eléctrico, se modifica el pulso de onda de la red, enviando impulsos de una frecuencia de 120 Khz, y en función de de la presencia de este pulso. En el receptor se le asigna un valor 1, o en caso contrario 0, actuando en consecuencia en función de su programación. Este pulso se envía junto a una señal de direccionamiento para que solo actúe sobre un receptor específico.

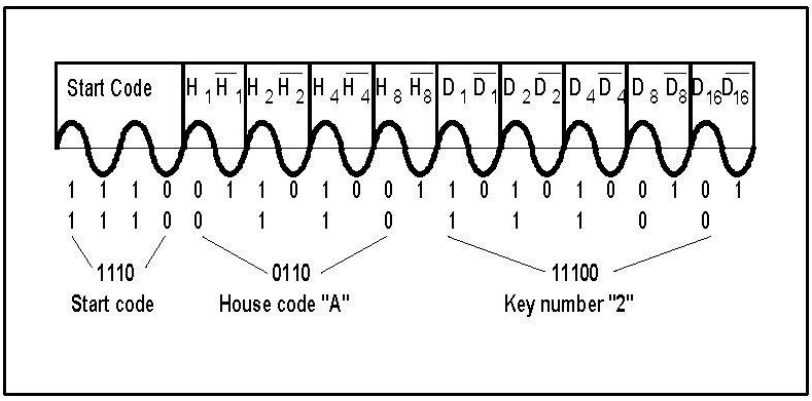


Figura 2.17: Envío de señal X-10

Ventajas e inconvenientes del uso del control *X-10*

- No necesita de instalación previa para su funcionamiento, ya que su instalación es rápida y sencilla.
- Se trata de un sistema abierto, por lo que se adapta con facilidad a otros dispositivos.
- Al ser un sistema modular se puede ampliar con facilidad.
- Son relativamente económicos.
- Son susceptibles a perturbaciones en la línea causando señales erróneas.
- Posee un uso limitado a viviendas, puesto que a nivel industrial las perturbaciones en la red son suficientemente elevadas como para desestimar su uso.

2.6.2. KNX

Este sistema de comunicación y control se considera abierto, es decir que puede incorporarse libremente a otros sistemas. Este tipo de control funciona gracias a que existe una interconexión entre todos los dispositivos a través de un bus de datos, se trata por lo tanto de un sistema de control distribuido.

Para su funcionamiento, solamente es necesario instalar un bus de dos hilos en paralelo con la alimentación eléctrica de 230v de la vivienda, transmitiendo las

señales de control a los dispositivos. Este bus de datos también permite que puedan comunicarse y enviar información entre los propios dispositivos.

Podemos clasificar los bus de datos según el medio de transmisión:

- PL100 Utilizando la red eléctrica
- TP1 Par trenzado
- RF Radio frecuencia
- Ethernet

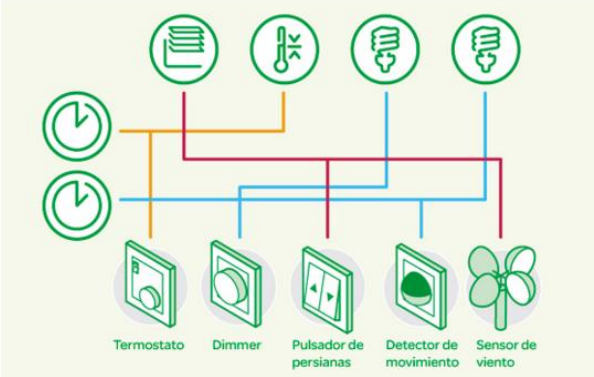


Figura2.18: Método de control KNX

Ventajas e inconvenientes del uso de sistemas de control KNX

- Se trata de un sistema modular, fácilmente ampliable sin necesidad de más instalaciones, solamente modificando la programación de los dispositivos que lo componen.
- Se pueden realizar mantenimientos fácilmente puesto que los dispositivos están conectados entres sí.
- Se trata de un sistema robusto y con capacidad de recuperación ante fallos puesto que es un sistema distribuido.
- Requiere de una instalación previa puesto que es necesario utilizar cables específicos para conectar el sistema.
- Su instalación en edificios ya construidos tiene un mal resultado estético.
- Tiene un precio elevado en comparación con otros sistemas.

2.6.3. EIB

Esta forma de control es muy similar a KNX, aunque en este caso se trata de un sistema descentralizado, puesto que cada dispositivo puede realizar funciones de forma autónoma sin necesidad de enviar datos al nodo central, de este modo si alguno de ellos falla no afecta de ninguna forma al funcionamiento general del sistema. Esto se produce porque, cada uno de los dispositivos unidos al bus de datos, dispone de un microprocesador propio que le proporciona un acceso al medio.

Según su medio de transmisión de datos podemos clasificarlos en:

- Par trenzado
- Red eléctrica de baja tensión
- RF Radio frecuencia
- Infrarrojos

Ventajas e inconvenientes del uso de sistemas EIB

- No requiere de una instalación compleja
- Permite utilizar dispositivos de otros fabricantes e interconectarse con otros sistemas ya instalados.
- Posee una resistencia a perturbaciones mayor que otros sistemas
- Su precio y el de los dispositivos es relativamente alto.
- Requiere de una alimentación de bus de datos mayor que otros sistemas similares.

2.7. SENSORES

En este apartado, se expondrán algunos de los dispositivos más conocidos y utilizados para este tipo de plataforma dentro de sus distintos campos, para posteriormente poder seleccionar los que mejor se adapten a las necesidades de cada uno de los bloques de diseño.

2.7.1. SENSORES DE MOVIMIENTO Y PRESENCIA

Módulo Sensor PIR HC-SR501

Este tipo de sensor de presencia está basado en la medición de radiación infrarroja. Todos los cuerpos emiten energía infrarroja de forma proporcional a su temperatura. Estos dispositivos disponen de un sensor piezoeléctrico que le permite captar este tipo de radiación y convertirla en señales eléctrica para poder ser monitorizadas por el procesador central.

Los sensores PIR funcionan gracias a la recepción de dos focos distintos de energía infrarroja dentro su campo de visión, estos son captados y procesados por la componente electrónica del dispositivo, generando una señal eléctrica si ambas lecturas no son iguales.

Figura2.19: Sensor PIR HC-SR501

Voltaje de entrada	*4.5~20V*
Corriente estática	*50uA*
Señal de salida	*0,3V ~5V*
Ángulo efectivo	*110 grados*
Distancia efectiva	*7 m.*
Tipo de sensor	*Digital*
Peso	*15g*

Tabla 4: Características PIR HC-SR501

Módulo sensor ultrasonidos HC-SR04

Este tipo de sensor se utiliza normalmente para medir distancias, aunque podemos también utilizarlo para conocer si hay alguna presencia en la estancia realizando algunas modificaciones en su programación habitual.

El funcionamiento de este dispositivo consiste en el envío de ultrasonidos a alta frecuencia a través de uno de los cilindros de los que se compone el sensor, esperando a que este rebote sobre cualquier superficie y retorne de nuevo al dispositivo, el tiempo invertido es enviado y procesado para obtener un cálculo de distancia.

Figura2.20: Sensor HC-SR04

Voltaje de entrada	*5 V*
Corriente en reposo	*<2 mA*
Ángulo Efectivo	*<15 grados*
Distancia Efectiva	*3cm ~3m.*
Resolución	*1cm*
Tipo de sensor	*Digital*
Peso	*12g*

Tabla5: Características sensor HC-SR04

Módulo sensor de Sonido LM393

Se trata de un sensor de sonido que dispone de un micrófono de alta sensibilidad, esta función puede ser utilizada para captar sonidos de bajas intensidad, por lo que podría utilizarse para funciones de detección de presencia en una estancia. Este micrófono tiene una baja respuesta a sonidos con tonos altos, lo cual presenta una ventaja si no se requiere trabajar con ultrasonidos. Es por lo tanto un sensor apto para captar sonidos débiles, puesto que con sonidos fuertes puede llegar a saturarse, también dispone de un regulador de sensibilidad para poder adaptarlo a las condiciones en las que se necesite. Se trata de un sensor de gran versatilidad y con un gran campo de usos dentro de la domótica como podría ser el control de luces u otros dispositivos de forma muy sencilla.

Figura 2.21: Sensor LM393

Voltaje de entrada	*4V~6V*
Frecuencia efectiva	*50Hz ~15KHz*
Sensibilidad	*-50dB ~ -70dB*
Peso	*5,00g*
Dimensiones	*3.8 cm x 1.6 cm x 1.0 cm*
Tipo de sensor	*Digital*

Tabla6: Características sensor LM393

2.7.2. SENSORES DE TEMPERATURA Y HUMEDAD

Sensor LM35

El sensor LM35 es un dispositivo de temperatura digital, este obtiene los valores de temperatura gracias a que está integrado con su propio circuito de control, proporcionando de forma lineal una señal de voltaje proporcional a la temperatura.
Este sensor es utilizado en multitud de tareas e integrado en otros dispositivos puesto que su uso es relativamente sencillo y muy económico.

Voltaje de entrada	4V~30V
Factor de escala lineal	+10.0 mV/ºC
Consumo	<60 µA
Rango de trabajo	-55ºC a +150ºC
Rango de tensiones	-550mV a 1500mV
Resolución	±0.5ºC a +25ºC
Tipo de sensor	Digital

Tabla7: Características sensor LM35

TMP36

Este sensor funciona de forma muy similar a LM35, aunque presenta algunas características que los diferencian.

Voltaje de entrada	2.7V~5.5V
Factor de escala lineal	+10.0 mV/ºC
Consumo	<50 µA
Rango de trabajo	-40ºC a +125ºC
Resolución	± 2ºC a +25ºC
Tipo de sensor	Digital

Tabla8: Características sensor TMP36

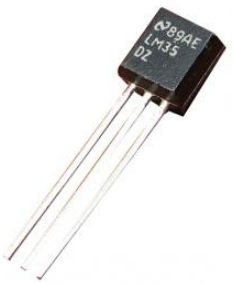


Figura 2.22 Sensor LM35

Figura 2.23: Sensor TMP36

Sensor DHT11 Y DHT22

Estos sensores son muy utilizados para la programación con la plataforma Arduino, puesto que ofrece unas altas prestaciones con un coste muy reducido.
Están compuestos por dos dispositivos, un sensor de humedad capacitivo y un termistor, ambas lecturas son procesadas por un circuito integrado que realiza una conversión de analógico a digital para posteriormente enviar esta señal digital con las lecturas realizadas.
Aunque ambos sensores son visualmente muy semejantes, y poseen una programación prácticamente igual, tienen unas prestaciones y características distintas.

Voltaje de entrada	3.3V~5V
Corriente máxima	2.5mA
Rango de humedad	20% a 90% HR
Precisión de humedad	± 5% HR
Rango de temperatura	0ºC a 50ºC
Precisión de temperatura	± 2 ºC
Frecuencia de muestreo	1Hz
Dimensiones	15.5mm x 12mm x 5.5mm

Tabla9: Características sensor DHT11

Voltaje de entrada	3.3V~5V
Corriente máxima	2.5mA
Rango de humedad	0% a 100% HR
Precisión de humedad	± 2% HR
Rango de temperatura	-40ºC a 80ºC
Precisión de temperatura	± 0.5ºC
Frecuencia de muestreo	0.5Hz
Dimensiones	15.1mm x 25mm x 7.7mm

Tabla10: Características sensor DHT22

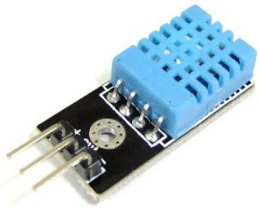


Figura2.24: Sensor DHT11

Figura2.25: Sensor DHT22

2.7.3. SENSORES DE CALIDAD DEL AIRE

Los dispositivos más destacados en este campo son MQ-2, MQ-3, MQ-7, MQ-5 y MQ-135, capaces de detectar la presencia de distintos gases en el aire, dependiendo del modelo. El funcionamiento de todos ellos es prácticamente el mismo, un componente del sensor se calienta cambiando su conductividad en función de la concentración de los distintos gases del exterior.

A continuación se enumerarán algunos de los principales gases que podemos encontrar en viviendas y que pueden perjudicar gravemente la salud de los inquilinos, por lo que es necesario poder controlar si se producen fugas no programadas o malas combustiones.

- **Metano (CH4):** El gas natural de uso común en viviendas, y está presente aproximadamente en un 90% de su composición, se utiliza en calderas, calentadores, cocina, etc.

- **Liquefied petroleum gas (LGP):** Compuesto por distintos gases provenientes del gas natural o el petróleo, principalmente propano y butano, se emplean en estado liquito debido a sus altas presiones. Su principal uso es el uso en calefacción.

- **Monóxido de carbono (CO):** Este gas se produce con la combustión deficiente de gas, gasolina, keroseno, en estufas, calderas y aparatos domésticos de combustión, es altamente tóxico y puede causar la muerte si se encuentra en cantidades elevadas.

- **Humos:** Se considera como la suspensión de pequeñas partículas en el aire debido a la combustión incompleta de combustibles, por lo que su detección puede resultar de ayuda para la prevención de incendios u otras fuentes de ignición.

- **Propano:** Gas utilizado en estado líquido, con altas prestaciones caloríficas y rendimiento, es muy utilizado en residencias para cocina, calefacción, climatizadores etc. Conlleva un riesgo elevado, puesto que es necesario para su uso almacenarlo en grandes cantidades en el interior de las viviendas.

Sensor Calidad del aire MQ-135

Voltaje de entrada	5V
Corriente máxima	200mA
Tipo de gases	Nh3, NOx, C2, Benceno

Detección partes por millón	10ppm~1000ppm
Tipo de sensor	Salida analógica y digital
Dimensiones	32mm x 22mm x 30mm

Tabla11: Características sensor MQ-135

Figura2.26: Sensor MQ-135

Sensor MQ-2

Voltaje de entrada	5V
Corriente máxima	150mA
Tipo de gases	H2, LGP,CH4, CO, Alcohol, humo, Propano
Detección partes por millón	300ppm~10000ppm
Tipo de sensor	Salida analógica y digital
Dimensiones	35mm x 20mm x 12mm

Tabla 12: Características sensor MQ-2

Figura2.27: Sensor MQ-2

Sensor MQ-3

Voltaje de entrada	5V
Corriente máxima	150mA
Tipo de gases	Alcohol
Detección partes por millón	500ppm~10000ppm
Tipo de sensor	Salida analógica y digital
Dimensiones	35mm x 20mm x 16mm

Tabla13: Características sensor MQ-3

Figura2.28: Sensor MQ-3

Sensor MQ-7

Voltaje de entrada	5V
Corriente máxima	180mA
Tipo de gases	CO2
Detección partes por millón	20ppm~2000ppm
Tipo de sensor	Salida analógica y digital
Dimensiones	36mm x 22mm x 17mm

Tabla14: Características sensor MQ-7

Figura2.29: Sensor MQ-27

Sensor MQ-5

Voltaje de entrada	5V
Corriente máxima	150mA
Tipo de gases	H2, LPG, CH4, CO, Alcohol
Detección partes por millón	100ppm~3000ppm
Tipo de sensor	Salida analógica y digital
Dimensiones	40mm x 23mm x 19mm

Tabla15: Características sensor MQ-5

Figura2.30: Sensor MQ-5

2.7.4. OTROS SENSORES ÚTILES EN DOMÓTICA

Sensor intensidad de luz

Estos sensores de luz se utilizan para detectar niveles de de intensidad lumínica externa, y producir una señal de salida eléctrica proporcional a la misma.

Este tipo de dispositivos utilizan un fotorresistor para poder detectar variaciones en la intensidad de la luz, estos, varían su resistencia en función de los cambios detectados, disminuyendo de forma lineal cuando la luz externa aumenta.

Voltaje de entrada	3.3V~5V
Corriente máxima	2.2mA
Chip comparado	LM393

Peso	5g
Dimensiones	36.0mm x 15.0mm x 6.0mm

Tabla 16: Características sensor intensidad de luz

Figura2.31: Sensor intensidad de luz

Sensor de gotas de lluvia

Estos dispositivos son idóneos para automatización en domótica puesto que son capaces de detectar la presencia de lluvia, para con ello poder desplegar un toldo o recoger las persianas. Este sensor funciona con el contacto de las gotas de lluvia sobre la placa del dispositivo, aumentando su resistencia, y enviando una señal al controlador.

Voltaje de entrada	3.3V~5V
Corriente máxima	2.5mA
Chip comparado	LM393
Tamaño de celda	5.0cm x 15.0cm
Peso	25g
Dimensiones	32.0mm x 14.0mm x 10mm

Tabla 17: Características sensor de lluvia

Figura2.32: Sensor de lluvia

3.0. DISEÑO DEL SISTEMA

En el siguiente apartado se diseñarán e implementarán cada uno de los bloques, seleccionado los dispositivos y las características más adecuadas para cada uno de los sistemas que lo componen. Del mismo modo, se detallarán todos los pasos que han sido necesarios seguir, para poder cumplir cada uno de los objetivos propuestos inicialmente para este proyecto.

3.1. ARQUITECTURA DEL SISTEMA

En primer lugar, es necesario seleccionar el tipo de arquitectura de control más adecuada para este sistema, a partir de la cual se distribuirán todos los sensores y actuadores.

ARQUITECTURA CENTRALIZADA

Se ha seleccionado este tipo de arquitectura de control debido a que proporciona una mayor facilidad de implementación e instalación, ya que permite poder sustituir tanto actuadores como sensores sin la necesidad de grandes cambios en la programación del microprocesador en el bloque principal.

Del mismo modo, este tipo de distribución permite incorporar nuevos sistemas al conjunto con unos cambios mínimos sin la necesidad de reducir la potencia del sistema, como sucedería en otro tipo de arquitecturas de control.

A continuación se enumerarán los bloques que posteriormente se diseñarán, y los elementos de los que se compondrán.

- Bloque Iluminación
- Bloque Ventilación
- Bloque de control de acceso y presencia
- Bloque de control de persianas
- Bloque de detección de gases
- Interfaz móvil

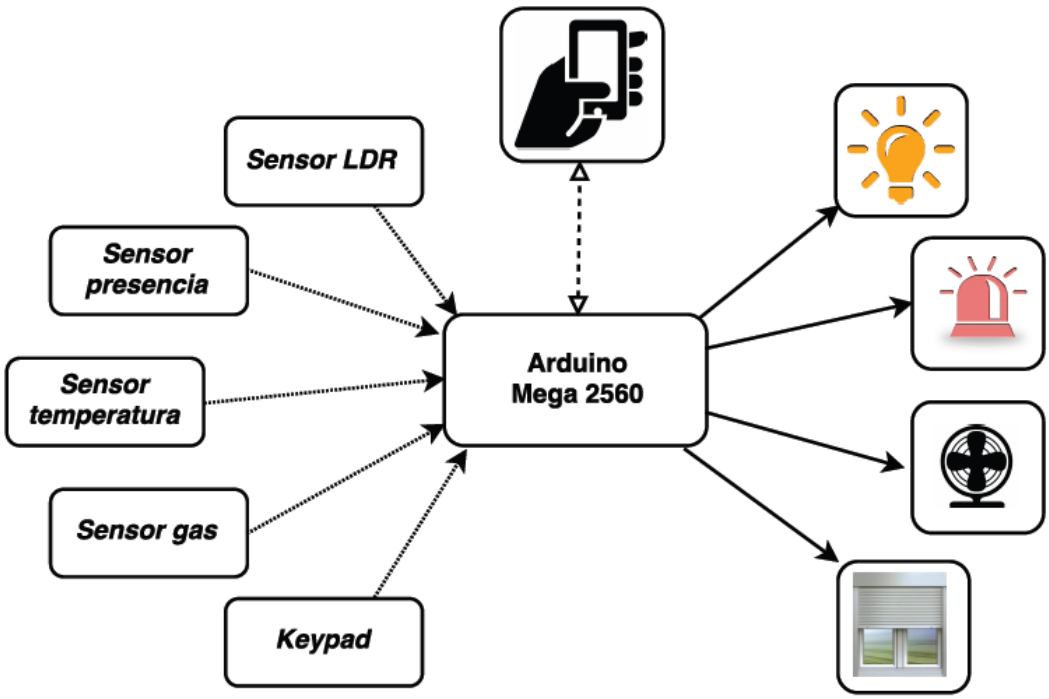


Figura 3.0: Arquitectura del sistema

En la *(fig 3.0)* podemos observar cómo se distribuirán los dispositivos dentro del sistema de control, en la parte izquierda del bloque principal se encuentran todos los sensores y dispositivos que se encargarán de captar información del entorno para posteriormente poder realizar una acción en consecuencia. Este protocolo de actuación vendrá predeterminado por la programación del microcontrolador dentro del bloque principal, siendo desarrollado en los apartados posteriores de diseño, para cada una de las funciones del sistema.

En el lado derecho, encontramos todos los actuadores, estos realizarán funciones diversas dependiendo de las lecturas de los sensores o de las exigencias del usuario.

Todos ellos, irán interconectados de forma física utilizando cableado al bloque de control, mediante los pines de Arduino. El tipo de cable que se adecúe a este sistema, lo seleccionaremos en apartados posteriores, del mismo modo que el método de conexión inalámbrico para conectar todo el sistema con los dispositivos móviles.

3.2. ARDUINO MEGA 2560

A pesar de que se hayan visto algunas de las características principales de este modelo de Arduino en apartados anteriores, en este apartado se incorporará información adicional, puesto que este dispositivo será el utilizado como bloque principal del sistema de control.

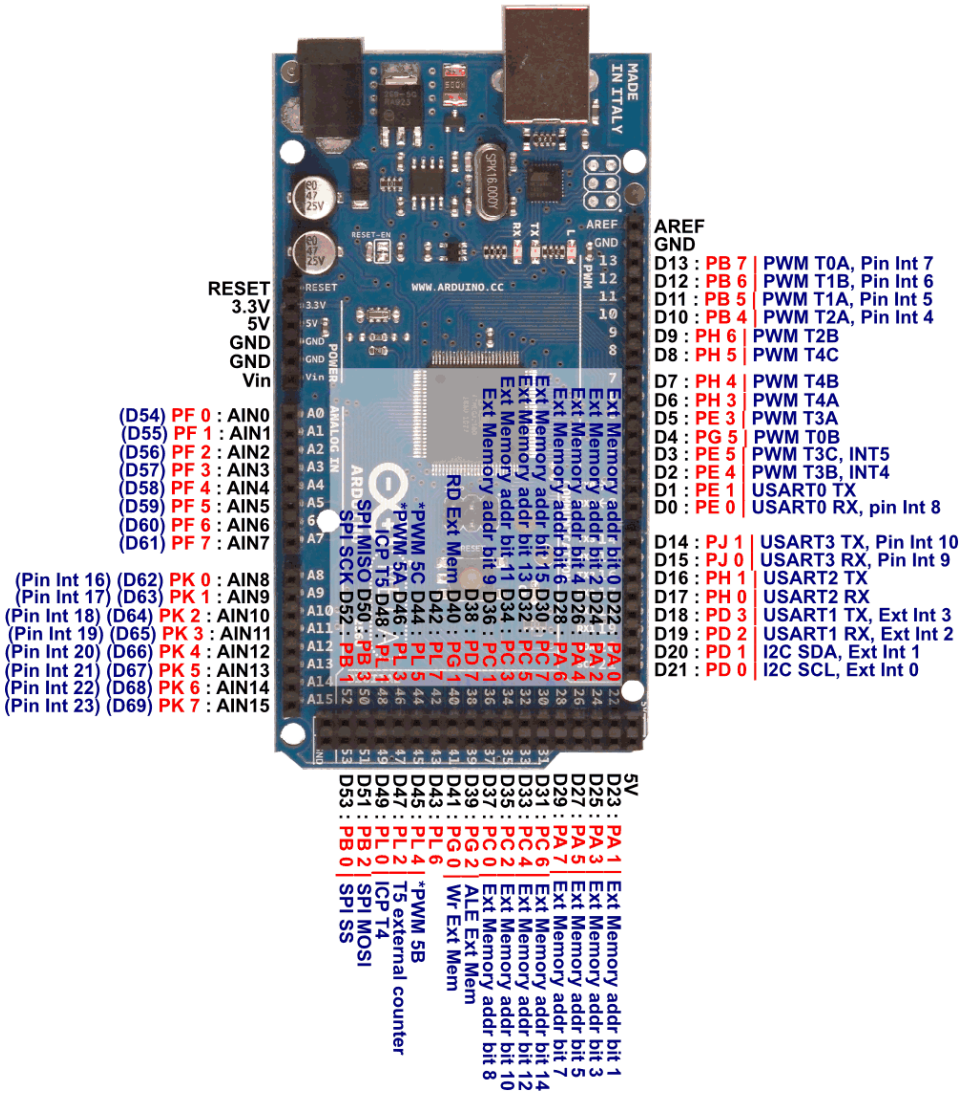


Figura 3.1 Pines Arduino Mega2560

En la (fig 3.1) se encuentran situados cada uno de los pines de los que se compone la placa Arduino, así como sus funciones para cada uno de ellos.

Cada una de estas funciones podremos encontrarlas de forma detallada en el *Anexo I* en los apartados finales del trabajo.

3.3. CONEXIONES DEL SISTEMA

A continuación, se seleccionarán los medios de interconexión para todos los elementos del sistema, tanto entre cada uno de los bloques con el núcleo de control, como la conexión para la interfaz móvil inalámbrica.

3.3.1. CONEXIÓN INALÁMBRICA

Puesto que uno de los objetivos principales es el desarrollo de un control con interfaz móvil, en primer lugar es necesario elegir que método de conexión se debe utilizar, dependiendo de que características se requieran para este sistema.

Como se especificó en los requerimientos del sistema, se desea implementar un de control inalámbrico, para poder tener una interfaz móvil en el interior de la vivienda, utilizando para ellos el propio teléfono móvil o una tablet.

Conociendo estas propiedades, se va a seleccionar un método de transmisión de datos inalámbrico, que proporcione unas características similares y que sea apto para esta función.

Se ha seleccionado el módulo de *Bluetooth HC-06*, se ha escogido este sistema de transmisión por su versatilidad y su facilidad de programación, además al no ser necesario un control remoto, cuando el usuario este fuera de la vivienda, la transmisión Bluetooth cumple con los requerimientos de distancias para este caso. Si se tratase de una vivienda de mayor tamaño, sería necesario utilizar otros métodos como podría ser una conexión mediante Ethernet.

A continuación se verán algunas de las características que definen este tipo de módulo.

Chip de radio	CSR BC417143
Frecuencia	2.4 GHz, banda ISM
Modulación	GFSK (Gaussian Frequency Shift Keying)
Antena de PCB incorporada	
Potencia de emisión	≤ 6 dBm, Clase II
Alcance	Entre 15m y 25 m
Sensibilidad	≤ -80 dBm a 0.1% BER
Velocidad: Asincrónica	2 Mbps (max.)/160 kbps, sincrónica: 1 Mbps/1 Mbps
Seguridad:	Autenticación y encriptación
Perfiles	Puerto serial Bluetooth
Consumo de corriente	30 mA a 40 mA
Voltaje de operación	3.6 V ~ 6 V
Dimensiones totales	1.7 cm x 4 cm
Temperatura de operación	-25 ºC a +75 ºC

Tabla18: Características módulo Bluetooth HC-06

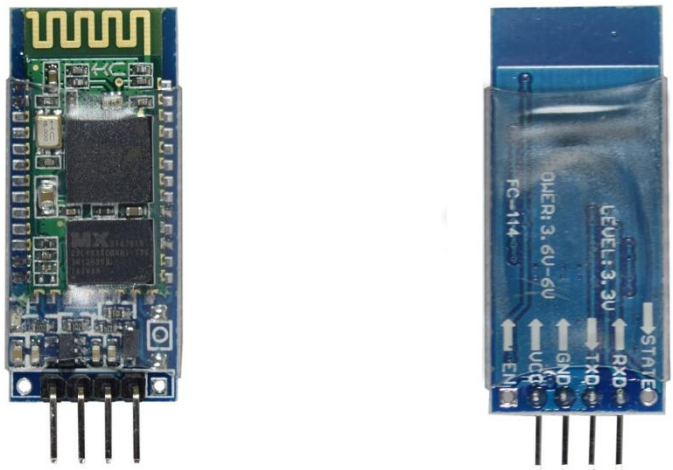


Figura 3.2: Módulo Bluetooth HC-06

3.3.2. CABLEADO

Para la realización de la instalación, será necesario la conexión de cada uno de los elementos periféricos con el núcleo central, para ello se utilizará cable de *par trenzado FTP*, ya que no se trata de grandes distancias y su coste es considerablemente reducido. La instalación del mismo se realizará utilizando lo que se considera cableado con estructura libre, es decir se utiliza la infraestructura de transmisión de energía de la vivienda para transportar los cables de transmisión de datos (*Bus de datos*) a cada uno de los elementos. De esta forma, se dispone al mismo tiempo en muchos de los casos de alimentación para dispositivos que requieran de una tensión elevada para funcionar, como sería el caso de motores o actuadores de mayor tamaño.

Del mismo modo se pueden utilizar las cajas de derivación instaladas. Este método de transmisión, reduce de forma considerable el tiempo de instalación, puesto que evita la realización de grandes cambios en la vivienda, así mismo se eliminan gran parte de los costes de instalación que provocarían otros métodos.

El cable FTP utilizado para la instalación consta de las siguientes características:

- Este cable posee un recubrimiento externo que reduce las interferencias provenientes de otros sistemas.
- Poseen una impedancia de 120 Ohmios.
- Según la velocidad de transmisión podemos clasificarlos en seis categorias, aunque las que se adaptan a nuestras exigencias son las categorias cuatro y cinco.

    - *Categoría 4*: soporta velocidades hasta 16 Mbits/seg
    - *Categoría 5*: hasta 100 Mbits/seg.

Se ha seleccionado la categoría cinco, para evitar limitar en exceso la velocidad de transmision de datos del bus.

Se incorporará mas informacion en el *Anexo II*, al final del trabajo, donde se incluirá la ficha técnica de este tipo de cable.

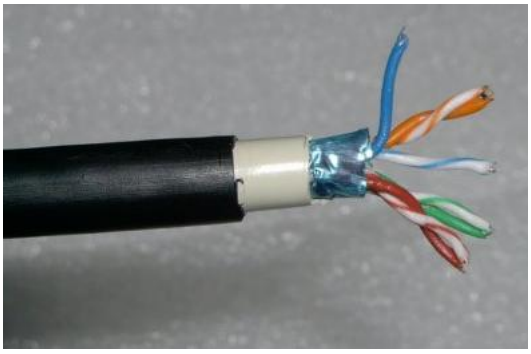


Figura 3.3: Cable de par trenzado FTP

3.4. MAQUETA DE SIMULACIÓN

Para poder simular cada uno de los procesos, se instalarán los dispositivos seleccionados en cada una de las partes de la vivienda. Para ellos se ha construido una maqueta simulando cada una de las estancias de la vivienda, consiguiendo un resultado mucho más visual para todo el conjunto del trabajo.

Se trata de una maqueta construida en cartón pluma de 5mm. de grosor.

Sus dimensiones son de 30 x 40 cm. A continuación se detalla la posición de cada uno de los elementos de dicha maqueta, incluyendo fotografías de cada una de las partes.

- Recibidor (8 x 12.5)cm
- Salón (4 x 24)cm
- Cocina (14 x 16)cm
- Dormitorio (16 x 18)cm
- Baño 1 (8x 12.5)cm
- Baño2 (16 x 9.5)cm

Figura 3.4: Foto planta de la maqueta

En la imagen (fig. 3.4) se observa una fotografía general de la planta de la maqueta construida, que servirá de soporte para cada uno de los sistemas que se diseñarán. Se ha querido recrear lo mejor posible el entorno de una vivienda, de este modo se obtiene un mayor resultado visual.

Figura 3.5: Foto maqueta 1

Figura 3.6: Foto maqueta 2

3.5. BLOQUE DE ILUMINACIÓN

En este apartado se realizará el diseño del sistema de iluminación, programando para ello el modelo seleccionado de Arduino. A continuación se explicarán cada uno de los pasos seguidos para el diseño e implementación del mismo.

Las necesidades requeridas para este bloque eran:

- Control manual y automático de las luces de la vivienda.
- Automatización para el encendido de lámparas con sensor de presencia.
- Control manual y automático de luces dimerizadas.

A continuación se detallará la programación realizada para la implementación de este bloque, se utiliza para ello la IDE de Arduino versión 1.6.5.

De la misma forma se explicarán las líneas de código más significativas.

3.5.1 CONTROL DE LÁMPARAS

- Declaración de pines y variables, para cada uno de los interruptores y lámparas de la vivienda.

```
int LUZ_COCINA=31;         // COCINA
int COCINA=30;             // Pin interruptor
int PULSADOR_COCINA;  // Variable que contendrá el estado del pulsador
int ESTADO_COCINA=0;  // Variable que contendrá el estado de la lámpara

int LUZ_SALON=33;          // SALÓN
int SALON=32;              // Pin interruptor
int PULSADOR_SALON; // Variable que contendrá el estado del pulsador
int ESTADO_SALON=0; // Variable que contendrá el estado de la lámpara

int LUZ_DORMITORIO=35;     // DORMITORIO
int DORMITORIO=34;         // Pin interruptor
int PULSADOR_DORMITORIO;  // Variable que contendrá el estado del pulsador
int ESTADO_DORMITORIO=0;  // Variable que contendrá el estado de la lámpara
```

```
int LUZ_ASEO1=37;         //  BAÑO 1
int ASEO1=36;             // Pin interruptor
int PULSADOR_ASEO1; // Variable que contendrá el estado del pulsador
int ESTADO_ASEO1=0; // Variable que contendrá el estado de la lámpara

int LUZ_ASEO2=39;             //BAÑO 2
int ASEO2=38;                 / / Pin interruptor
int PULSADOR_ASEO2; // Variable que contendrá el estado del pulsador
int ESTADO_ASEO2=0; // Variable que contendrá el estado de la lámpara
```

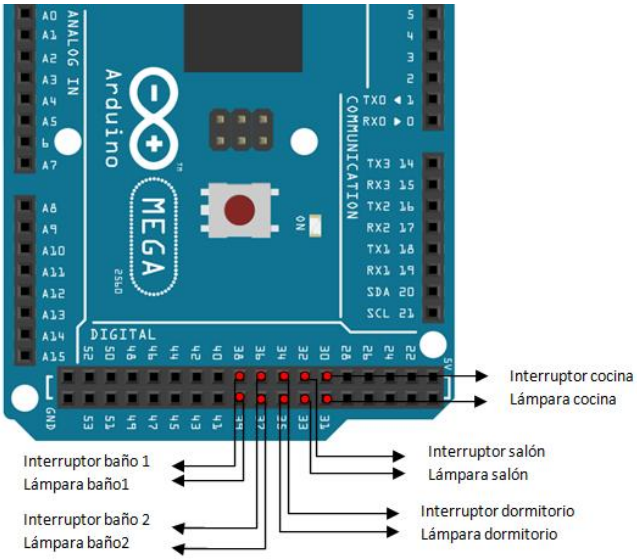


Figura 3.7: Pines bloque de iluminación

- Dentro de *void setup()* se incluye la función de los pines declarados, si actuarán como entradas, (*INPUT*) en el caso de pulsadores o interruptores, o salidas, (*OUTPUT*) en el caso de lámparas o actuadores.

```
void setup()
  pinMode(COCINA,INPUT);                //Pulsadores de las distintas estancias
  pinMode(DORMITORIO,INPUT);
  pinMode(SALON,INPUT);
  pinMode(ASEO1,INPUT);
  pinMode(ASEO2,INPUT);
```

```
 pinMode(LUZ_COCINA,OUTPUT);         //Lámparas
 pinMode(LUZ_SALON,OUTPUT);
 pinMode(LUZ_DORMITORIO,OUTPUT);
 pinMode(LUZ_ASEO1,OUTPUT);
 pinMode(LUZ_ASEO2,OUTPUT);
```

- Dentro de void loop(), se incluirán todos los procesos que se repetirán de forma cíclica, como puede ser la lectura de sensores o interruptores.

```
  void loop()
// Realiza las lecturas del estado de los interruptores para cada estancia
PULSADOR_COCINA=digitalRead(COCINA);
PULSADOR_SALON=digitalRead(SALON);
PULSADOR_DORMITORIO=digitalRead(DORMITORIO);
PULSADOR_ASEO1=digitalRead(ASEO1);
PULSADOR_ASEO2=digitalRead(ASEO2);
```

- Para la lectura de los pulsadores se emplea el comando *'digitalRead'*, y se asigna el valor leído a cada una de las variables.

```
if(PULSADOR_COCINA && Remoto_cocina ){              // Conmutación entre recepción serial y interruptor
 LUCES(LUZ_COCINA,1);     // Envía a la función la estancia y el estado encendido (1)
 ESTADO_COCINA=1;          //Cambia el estado de la variable a 1
}else{
  LUCES(LUZ_COCINA,0);    // Envía a la función la estancia y el estado apagado (0)
    ESTADO_COCINA=0;       //Cambia el estado de la variable a 0
}
if(PULSADOR_SALON && Remoto_salon){
 LUCES(LUZ_SALON,1);      // Envía a la función la estancia y el estado encendido (1)
 ESTADO_SALON=1;           //Cambia el estado de la variable a 1
}else{
  LUCES(LUZ_SALON,0);     // Envía a la función la estancia y el estado apagado (0)
   ESTADO_SALON=0;        //Cambia el estado de la variable a 0
}
if(PULSADOR_DORMITORIO && Remoto_dormitorio){
 LUCES(LUZ_DORMITORIO,1); // Envía a la función la estancia y el estado encendido (1)
 ESTADO_DORMITORIO=1;       //Cambia el estado de la variable a 1
}else{
  LUCES(LUZ_DORMITORIO,0); // Envía a la función la estancia y el estado apagado (0)
   ESTADO_DORMITORIO=0;      //Cambia el estado de la variable a 0
}
if(PULSADOR_ASEO1 && Remoto_aseo1){
 LUCES(LUZ_ASEO1,1);    // Envía a la función la estancia y el estado encendido (1)
```

```
 ESTADO_ASEO1=1;          //Cambia el estado de la variable a 1
}else{
  LUCES(LUZ_ASEO1,0);   // Envía a la función la estancia y el estado apagado (0)
  ESTADO_ASEO1=0;         //Cambia el estado de la variable a 0
}
if(PULSADOR_ASEO2 && Remoto_aseo2){
 LUCES(LUZ_ASEO2,1);   // Envía a la función la estancia y el estado encendido (1)
 ESTADO_ASEO2=1;         //Cambia el estado de la variable a 1
}else{
  LUCES(LUZ_ASEO2,0);  // Envía a la función la estancia y el estado apagado (0)
   ESTADO_ASEO2=0;      //Cambia el estado de la variable a 0
}
```

- En esta sección, se condiciona el estado de los interruptores, si estos están accionados envían a la función un estado “1” y esta activa o en caso contrario”0”, desactiva las lámparas para cada una de las habitaciones.

```
//FUNCION PARA ENCENDER LUCES. Envía estancia y el estado de la misma.
void LUCES(int Estancia,int estado){
  digitalWrite(Estancia,estado);
}
```

- Se emplean funciones con mucha frecuencia, para realizar tareas de forma ajena al bucle principal, estas solamente realizarán su función cuando se les llame dentro del mismo bucle. En este caso se emplean para encender o apagar luces utilizando el comando ‘*digitalWrite*’, esta función, recibe del bucle principal que estancia y en qué estado deben encontrarse, utilizando un 1 para un estado ‘*HIGH*’ y 0 para ‘*LOW*’.

3.5.2 CONTROL DE LÁMPARAS DIMERIZADAS

El control de lámparas mediante el uso de dimmers, permite al usuario poder regular su intensidad a placer para ajustarla a sus necesidades, este tipo de luces, se instalarán en la habitación y en el dormitorio para poder crear un ambiente de confort.

- Se emplean los mismos pasos que para el caso anterior, en primer lugar se declara la posición de cada uno de los pines que utilizará cada entrada o salida. En este caso, también incorporamos un sumatorio que se inicializa a 0 y será el encargado de incrementar o disminuir la intensidad de las lámparas, tanto para un control manual, como para una posterior implementación de modo remoto.

```
// DIMMER SALÓN

int DIMMER_SALON_UP=40;          //Pulsador dimmer salón incremento
int ESTADO_SALON1;               //Estado pulsador
int DIMMER_SALON_DOWN=41;        //Pulsador dimmer salón disminución
int ESTADO_SALON2;               //Estado pulsador
int LUZ_DIMMER_SALON=3;          //Lámpara dimmer salón
int SUM_DIMMER=0;                //Sumatorio Dimmer 1
int ESTADO_DIMMER=0;             //Estado luz dimerizada

// DIMMER HABITACIÓN
int DIMMER_HABITACION_UP=42;     //Pulsador dimmer dormitorio incremento
int ESTADO_HABITACION1;          //Estado pulsador
int DIMMER_HABITACION_DOWN=43;   //Pulsador dimmer dormitorio disminución
int ESTADO_HABITACION2;          //Estado pulsador
int LUZ_DIMMER_HABITACION=4;     //Lámpara dimmer dormitorio
int SUM2_DIMMER=0;               //Sumatorio Dimmer 2
int ESTADO_DIMMER2=0;            //Estado luz dimerizada
```

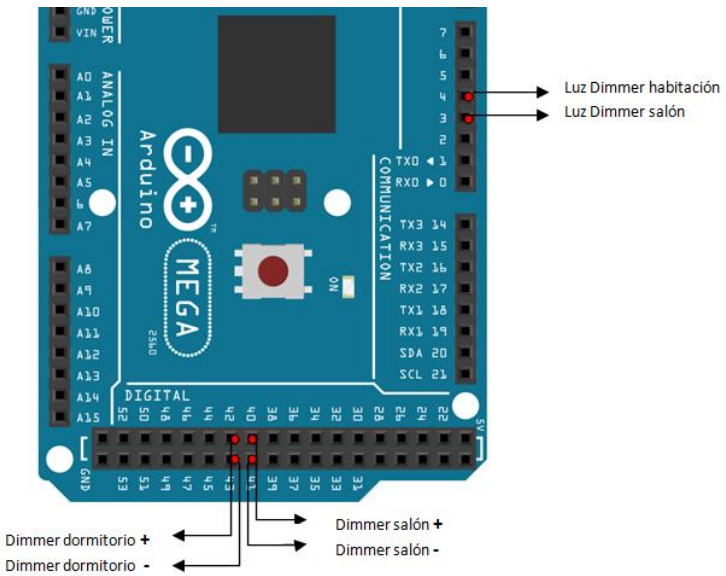


Figura 3.8: Pines bloque de iluminación con dimmers

- Se establecen la función de cada una de las variables, si trabajará como entrada o como salida. En este caso cada dimmer utilizará dos pulsadores, uno para incrementar y otro para disminuir por lo que cada uno constará de dos entradas y una sola salida.

```
pinMode(DIMMER_SALON_UP,INPUT);              // Pulsadores dimmer SALÓN
pinMode(DIMMER_SALON_DOWN,INPUT);
pinMode(DIMMER_HABITACION_UP,INPUT);     //Pulsadores dimmer HABITACIÓN
pinMode(DIMMER_HABITACION_DOWN,INPUT);

pinMode(LUZ_DIMMER_SALON,OUTPUT);          // Salidas
pinMode(LUZ_DIMMER_HABITACION,OUTPUT);
```

- Se realizan las lecturas del estado de los pulsadores y se asignan a las variables establecidas anteriormente.

```
// LECTURA PULSADORES DIMMER
ESTADO_SALON1=digitalRead(DIMMER_SALON_UP);                    //Botón para incrementar dimmer del salón
ESTADO_SALON2=digitalRead(DIMMER_SALON_DOWN) ;                 // Botón para disminuir dimmer del salón
ESTADO_HABITACION1=digitalRead(DIMMER_HABITACION_UP);      //Botón para incrementar dimmer de la habitación
ESTADO_HABITACION2=digitalRead(DIMMER_HABITACION_DOWN); // Botón para disminuir dimmer de la habitación
int Dim1=0; // Iniciación de las variables  que almacenarán el tiempo que ha sido pulsado cada botón.
int Dim2=0;

  //Dimmers
SUM_DIMMER=constrain(SUM_DIMMER,0,255);         //Limita el estado de los dimmers entre (0-255)
SUM2_DIMMER=constrain(SUM2_DIMMER,0,255);
```

- Se emplea la función '*constrain*' para limitar el contador entre los valores (0 y 255), puesto que son los valores mínimo y máximo respectivamente que acepta Arduino como entrada y salida.

```
if(ESTADO_SALON1==1){       // Mientras el pulsador este presionado, contador se irá incrementando hasta alcanzar un valor
 SUM_DIMMER++;                  máximo de 255
}
if(ESTADO_SALON2==1){      // Mientras este pulsador este accionado el contador disminuirá hasta alcanzar un valor
  SUM_DIMMER--;                 mínimo de 0
}
if(ESTADO_HABITACION1==1){
 SUM2_DIMMER++;
}
if(ESTADO_HABITACION2==1){
 SUM2_DIMMER--;
}
```

```
if(SUM_DIMMER!=0){            // Cambia el estado de las variables en función de si están encendidas o no las lámparas es decir ,
 ESTADO_DIMMER=1;                que el contador sea 0 o distinto de 0.
}else{
 ESTADO_DIMMER=0;}
if(SUM2_DIMMER!=0){
 ESTADO_DIMMER2=1;
}else{
 ESTADO_DIMMER2=0;}
```

- Esta función es muy similar al caso anterior, recibe por parte del bucle principal una llamada, enviándole la estancia y un valor entre (0-255), y esta aplica una salida de tensión proporcional a este valor.

```
//FUNCIN  DIMMERS
void DIMMER(int estancia,int valor){
valor=constrain(valor,0,255);          // Se limita de nuevo dentro de la propia función el valor recibido para evitar errores.
   analogWrite(estancia,valor);        // Envía el valor recibido al pin seleccionado
}
```

3.6. BLOQUE DE CONTROL TÉRMICO

En este apartado, se va a implementar una estructura de control tanto manual como automático, de un sistema de refrigeración para la vivienda.
De esta forma conseguimos mantener una temperatura constante, pudiendo predefinir la misma durante la programación del bloque.
Para ello, se van a diseñar una serie de controladores, capaces de regular automáticamente el estado de la ventilación, en función de la lectura de los sensores de temperatura instalados.

Para esta función se va a emplear el sensor de temperatura y humedad modelo *THD 11,* de cuyas características básicas se han tratado con anterioridad, pudiendo encontrar información técnica más detallada en los anexos finales.
Se ha seleccionado este tipo de sensor por su facilidad de instalación, la accesibilidad de sus librerías y documentación técnica, esto permite desarrollar con mayor facilidad todo el diseño de este bloque.

3.6.1. DISEÑO PID DE CONTROL DE TEMPERATURA

Podemos definir un control PID, como un mecanismo a través del cual podemos controlar una variable, ya se temperatura, presión, velocidad, u otras semejantes, mediante un lazo de retroalimentación. El controlador, obtiene la diferencia entre el valor actual de nuestra variable en contraposición con la deseada.

El algoritmo de control PID incluye tres parámetros:

- Ganancia proporcional (P)
- Integral (I)
- Derivativo (D)

La ganancia proporcional, calcula la diferencia entre el valor actual y el valor óptimo elegido, denominado ‘set-point’.
El parámetro integral, es el tiempo que conlleva llevar a cabo la corrección deseada.
La función derivativa, se encarga de prever el error e inicia las acciones pertinentes para evitar que este alcance un valor demasiado grande.
Una correcta sincronización de estos tres parámetros es la que consigue obtener unos resultados adecuados a las necesidades que se le exigen, para ello es necesario realizar una serie de cálculos y mediciones previas para evitar finalmente, fluctuaciones o vibraciones no deseadas.
Con este tipo de sistema de regulación, conseguimos un control automático mucho más completo, adaptado a las exigencias, capaz de autor regularse dependiendo de las condiciones externas en las que se encuentre, de este modo conseguimos un ahorro energético y un confort térmico mayor.

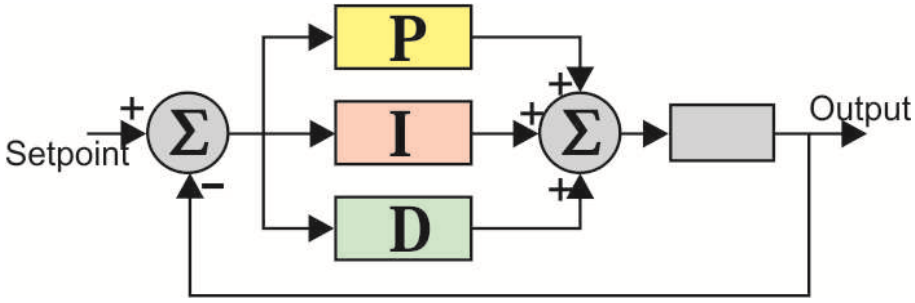


Figura 3.9: Esquema controlador PID

En primer lugar, se van a realizar una serie de mediciones de temperatura de forma controlada, se recrea un sistema cerrado y se eleva la temperatura del entorno hasta alcanzar una temperatura que consideremos adecuada, con ayuda de algún sistema térmico, como podría ser una estufa o un radiador, posteriormente, se apaga la fuente de calor y se deja enfriar el sistema realizando mediciones periódicas a medida que la temperatura se equilibra con la del entorno.

En la prueba realizada, se toman mediciones de temperatura una vez que el sistema ha alcanzado una temperatura aproximada de 45ºC, posteriormente se toman mediciones cada 5 segundos durante aproximadamente 13 minutos, este es el tiempo que tarda en alcanzar de nuevo los 25ºC que utilizaremos como temperatura de establecimiento o 'set point', obteniendo aproximadamente 160 muestras de temperatura durante dicho tiempo.

En este tipo de prueba no se utiliza ningún tipo de ventilación por lo que se consideraría una respuesta no forzada. Con ello conseguimos que una vez obtenido el control PID obtengamos una respuesta mucho más suave y menos brusca.

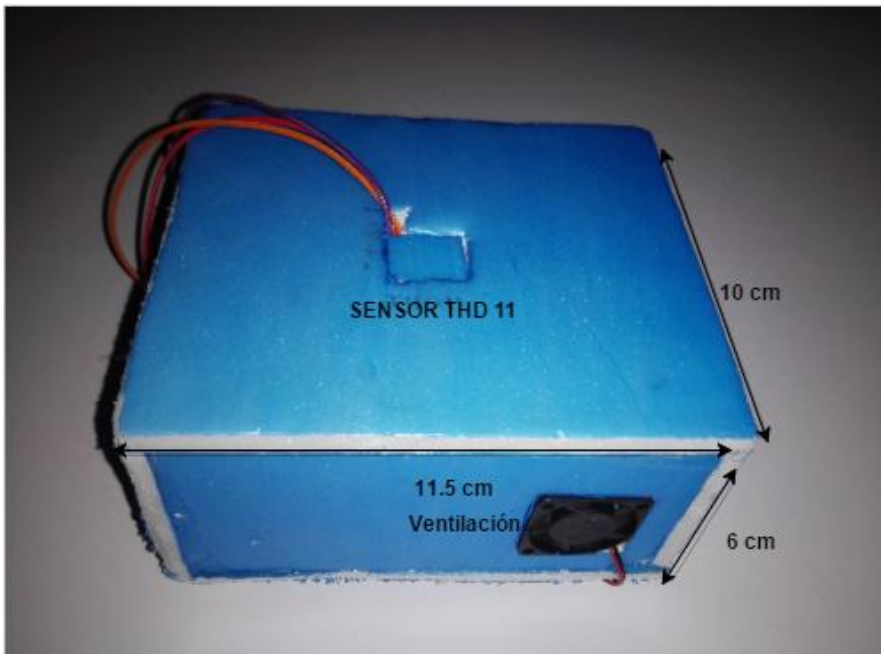


Figura 3.10: Diseño PID

Como observamos en la *(fig.3.10),* se ha elaborado un cubo de pequeñas dimensiones, utilizando como material polietileno, para que sea capaz de almacenar la temperatura el mayor tiempo posible, una vez se apague la fuente de calor, de este modo conseguiremos una lectura de datos mucho menos escalonada y por lo tanto una respuesta final del sistema más estable. El entorno creado consta también de ventiladores en ambos lados, por lo que antes de tomar solamente un tipo de lecturas, se ensayaron otros métodos como podría ser forzando el enfriamiento con ayuda de ellos. Finalmente se optó por tomar los datos con un sistema no forzado.

Una vez realizadas todas las mediciones, se exportan todas ellas al programa MATLAB, versión R2016a que incorpora aplicaciones de diseño de controladores PID, para con ello, obtener la función de transferencia del sistema y poder clasificar y diseñar el controlador al respecto.

Además de las lecturas obtenidas, es necesario introducir más información al programa, como podría ser la tensión de funcionamiento de los ventiladores que utilizaremos para la simulación, para que la respuesta que tome el sistema pueda ser apta con Arduino, y este entregue una salida PWM en consecuencia entre (0 y 255). Se trata de ventiladores de reducido tamaño que funcionan con pequeñas tensiones (5V), lo que facilita su funcionamiento mediante Arduino, sin la necesidad de cualquier adaptador electrónico de potencia.

Una vez introducidos todos los parámetros necesarios ya podremos obtener la función de transferencia de planta de nuestro sistema. Del mismo podo podemos obtener gráficas de las distintas variables introducidas. El objetivo de esto, es obtener una serie de constantes del controlador, las cuales, son necesarias para que el funcionamiento al introducirlo en Arduino sea el correcto.

Para ello Arduino consta de una serie de librerías y funciones que son capaces de realizar estos procesos de la misma forma que lo haría otro tipo de autómata. En este caso utilizaremos la librería '*PID_v1.h'* en la cual vienen

predefinidos todos los cálculos necesarios para el funcionamiento normal de un sistema de control PID.

Obtenemos una función de transferencia de planta utilizando el comando '*ident*' 'de Matlab:

```
Did you mean:
>> P1

P1 =
Process model with transfer function:
             Kp
  G(s) = ----------
          1+Tp1*s

        Kp = 0.084541
       Tp1 = 428.81

Name: P1
Parameterization:
    'P1'
   Number of free coefficients: 2
   Use "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using PROCEST on time domain data "mydata".
Fit to estimation data: 92.82%
```

Figura 3.11: Función de transferencia de planta

De la cual podemos deducir que se trata de una función de transferencia de un sistema de primer orden. Este tipo de sistemas se caracterizan por no presentar sobreoscilaciones, es decir, el sistema nunca alcanza el valor exacto y por lo tanto son normalmente sistemas relativamente lentos.

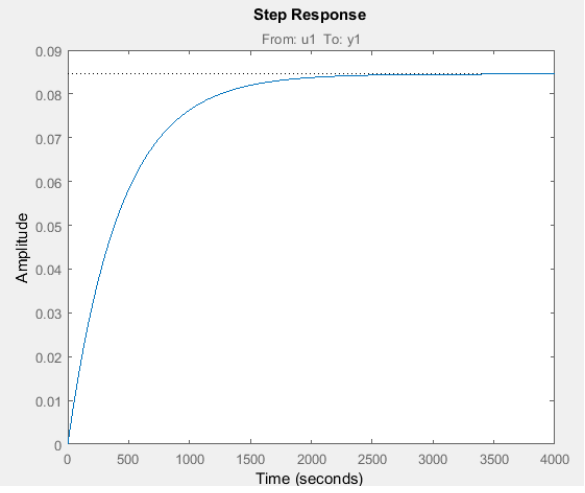


Figura 3.12: Respuesta en escalón a planta

Como observamos en la (fig 3.12) podemos ver de forma gráfica la respuesta en escalón a nuestra planta, y donde apreciamos un tiempo de establecimiento muy elevado (4000 seg.)

Posteriormente se introduce el controlador PID para modificar la respuesta que requerimos para nuestro sistema. En la figura siguiente, observamos el panel, a partir del cual se puede modificar algunas de las características del mismo, como puede ser el tiempo de respuesta o la brusquedad con la que actuará ante variaciones de temperatura.

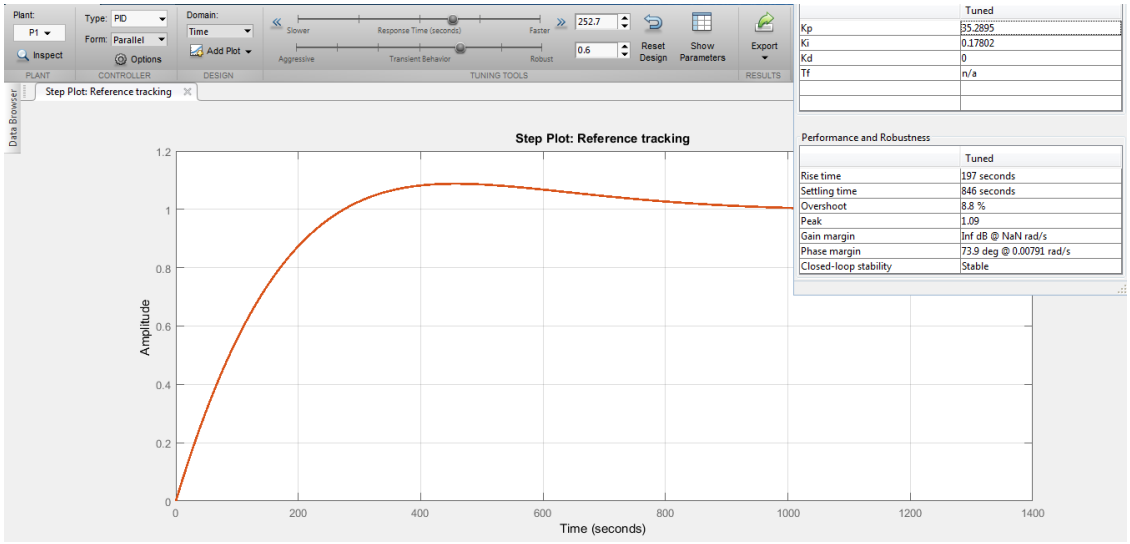


Figura 3.13: Diseño PID

Al tratarse de un control de temperatura, no es necesaria una respuesta inmediata del sistema, por lo que el tiempo de establecimiento no es necesario que sea reducido. Se realizan pruebas con distintas configuraciones, estableciendo un tiempo aproximado de entre 5 a 8 minutos.

Kp	35.2595
Ki	0.17802
Kd	0
Tiempo de subida	197
Tiempo de establecimiento	846seg
Sobreoscilación	8.8%
Estabilidad en lazo cerrado	Estable

Tabla 19: Características PID

A continuación introducimos los valores de las constantes obtenidas a través de Matlab en la IDE de Arduino, utilizando para ello la librería adecuada que nos realice las funciones de control PID.

- Para el uso de este sistema automático, se emplearán dos librerías distintas, una para la lectura del sensor THD11 ‘DHT.h’ y la anteriormente mencionada ’PID_v1.h’.

```
 #include <PID_v1.h>
#define ventilador1 12
#define ventilador2 11
#include "DHT.h"
#define DHTPIN A0   // Definimos la entrada analógica A0 como lectura del sensor de temperatura.
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
double Setpoint, Input, Output;                  // Declaramos las variables Setpoint, para la temperatura seleccionada
 int Error=0;                                    // Se inicializa el error a 0.
double Kp=105.1458, Ki=0.61146, Kd=0;            //Se inicializan los parámetros del controlador PID

PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, REVERSE); // Establecemos si el sistema funcionará de forma directa o inversa
```

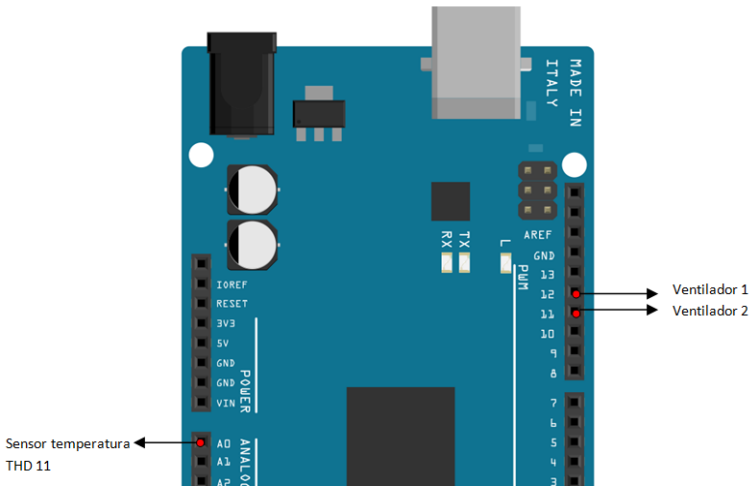


Figura 3.14: Pines PID

- Esta librería nos permite seleccionar si queremos que el control actúe por encima o por debajo de la temperatura seleccionada ’Set point’, es decir, si deseamos que actué como calefactor o como refrigerador. En

este caso debido a la dificultad para poder simular un sistema de calefacción, solamente utilizaremos el control inverso (*REVERSE*) para que el control se limite a la refrigeración mediante los ventiladores instalados.

```
void setup() {
  Setpoint = 25;    // Establecemos la temperatura a 25ºC, será a partir de la cual se accionará nuestra ventilación
  dht.begin();       //Se inicializa la lectura de temperatura.
  myPID.SetMode(AUTOMATIC); // Establecemos el modo de funcionamiento del sistema, si es manual o automático
void loop() {
  Input =dht.readTemperature() ;   //Realiza la lectura del sensor de temperatura.
  myPID.Compute();                  // Se llama a la función encargada de realizar los cálculos y las acciones pertinentes.
  Error=Setpoint-Input;             // Se calcula el error del sistema.
```

- Solamente se han expuesto las líneas de código más importantes, puesto que se trata de un sistema manual y automático, controlado vía remota, se explicarán las líneas de código del sistema manual en el bloque de control remoto, en el que podremos seleccionar el nivel de intensidad para cada uno de los ventiladores instalados.

  Del mismo modo, se incluirá la librería completa en el *Anexo III*, puesto que ha sido necesario comprenderla, para la correcta implementación del sistema de control PID.

3.7. BLOQUE DE CONTROL DE ACCESO

En este bloque se va a realizar el diseño e implementación de un sistema de control de acceso a la vivienda tal y como se expone en los requerimientos del sistema. Este control constará de un teclado que activará o desactivará una alarma acústica, que irá unida a un sensor de presencia. Para ello han sido seleccionados los dispositivos más aptos para estas funciones. Son los siguientes:

- Teclado 4x4 (Keypad)
- Sensor de presencia PIR HC-SR501
- Modulo de zumbador

Este sensor ha sido escogido porque ofrece la mejor detección de presencia de todos los analizados, se trata de un dispositivo de uso muy común, por lo que existen multitud de ejemplos ya implementados que facilitarán la programación.

A continuación se muestra un diagrama de flujo con el funcionamiento general que deberá tener el sistema de control de acceso:

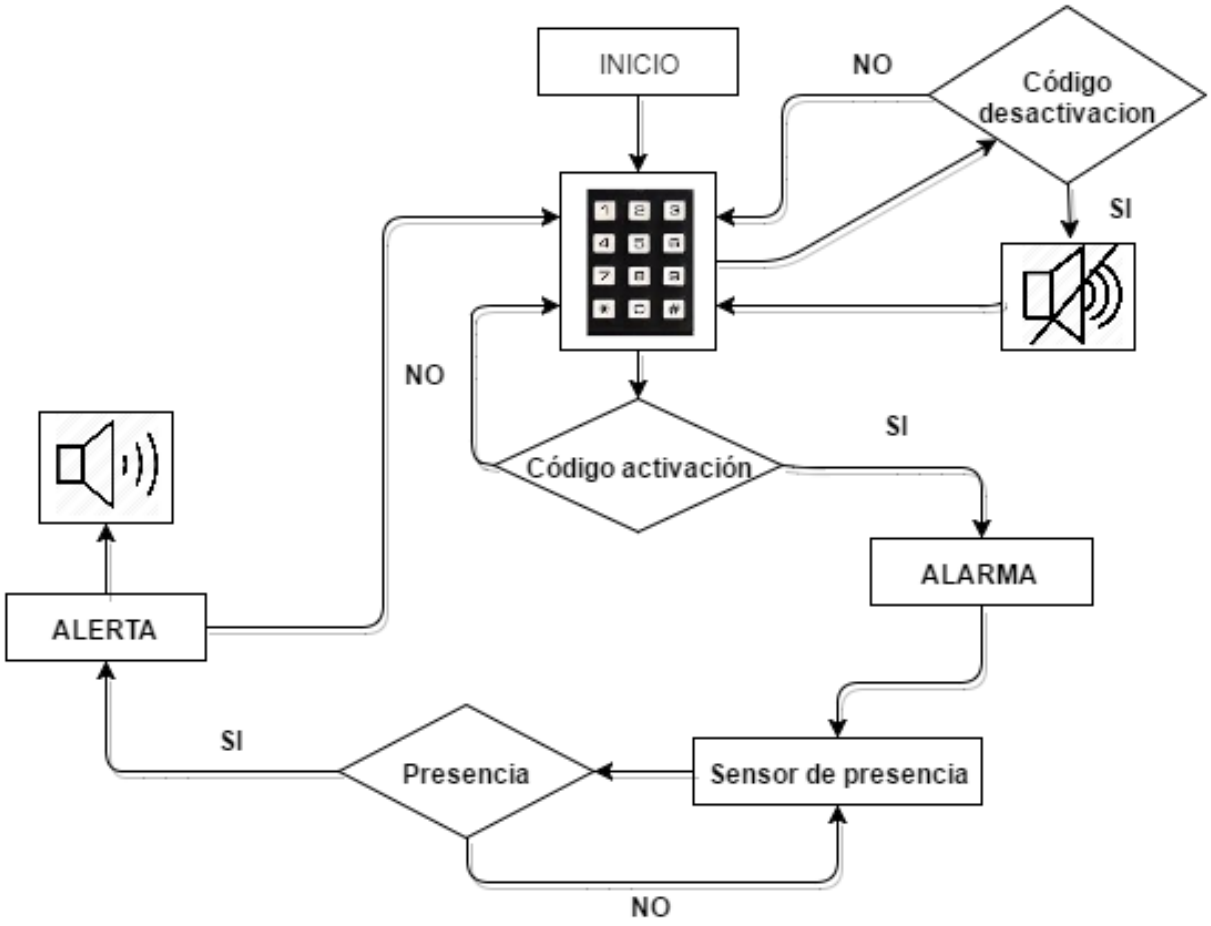


Figura 3.15: Diagrama de flujo bloque de control de acceso

En la (fig 3.15) se puede observar mediante un diagrama de flujo, las distintas acciones que se deben llevara a cabo en este bloque de control, en el se podrá accionar o desactivar un control de presencia en la vivienda. El sensor de presencia (PIR HC-SR501) deberá estar situado en un lugar cercano a la puerta principal de entrada a la vivienda, según las especificaciones previas.

Del mismo modo que en bloques anteriores, a continuación se explicarán las líneas de programación más características.

- Se va a emplear la librería para poder utilizar el teclado numérico ‘keypad.h’

```
 #include <Keypad.h>
char* secretCode = "1503"; // En esta línea se declara el vector que utilizaremos como contraseña de desactivación.
int position = 0;
int Sensor_presencia = 10;  //Declaramos el pin  que utilizaremos para conectar el sensor de presencia.
int Presencia = LOW;        //Definimos el estado inicial de algunas de las variables.
```

- Muchas de las líneas del código se pueden encontrar predefinidas en algunos ejemplos ya implementados por otros usuarios, por lo que facilita considerablemente la programación del teclado (keypad), solamente es necesario adaptar el código a nuestro sistema.

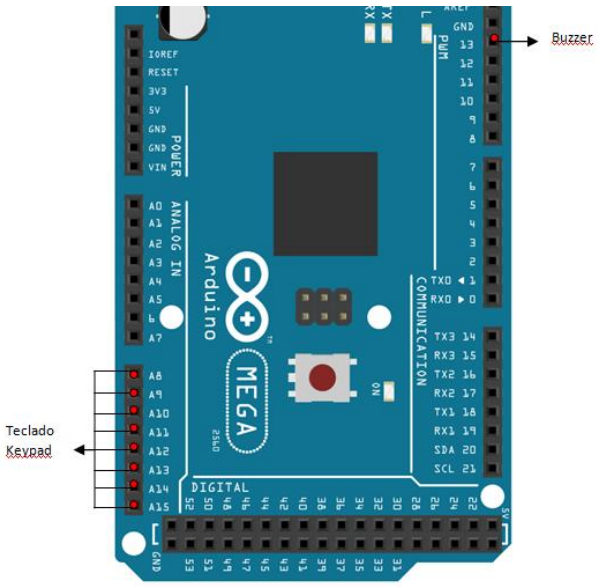


Figura 3.16: Pines control de acceso

```
void setup()
{
// Función de cada uno de los pines, si es como entrada o como salida
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(pin,OUTPUT);
  pinMode(Sensor_presencia, INPUT);
  pinMode(buzz, OUTPUT);
  ALARMA(false); //Estado inicial de la alarma es desactivada
```

```
void loop()
{

PASSWORD(); // En función de lo tecleado activaremos o no la alarma
if(ESTADO_ALARMA==HIGH){
SENSOR_PRESENCIA(1);
}
if(ALERTA==HIGH){
  ring(true);
}else{
  ring(false);
}
```

- El uso de funciones en este bloque es fundamental, puesto que la mayoría de acciones que van a realizarse no deben de hacerlo de forma cíclica en el bucle principal, si no que solamente se deben activar cuando se les requiera.

```
// FUNCION LECTURA KEYPAD
int PASSWORD(){
     char key = keypad.getKey();
  if (key == '#') {        // establecemos el character "#" como predeterminado para activar la alarma
    position = 0;

    lcd.setCursor(3,0);
    lcd.print("ACTIVANDO..");
    delay(5000);
    Buzz(1000);
    delay(500);// Tiempo de retardo hasta que se active para poder salir.
    Buzz(0);
    ALARMA(true);    }

  if (key == secretCode[position]){ //Condición en la que el caracter introducido debe ser igual al de la contraseña establecida
    position++;
    lcd.setCursor(0,0);
    lcd.print("Pass:");
    lcd.setCursor(5+position,0);
    lcd.print("*");
   if(position>=6){
    lcd.clear();
  }
if (position == 6) { // Si coinciden todas las posiciones la alarma se desactivará
  ALARMA(false);

}}
  delay(5); }
```

- Mientras esta función este activa, leerá la entrada del teclado, y dependiendo de si los caracteres introducidos de forma manual, coinciden con la contraseña establecida activará o desactivará otra función.

```
int ALARMA(int estado){
  if (estado==1) {
    digitalWrite(redPin, HIGH);
    digitalWrite(greenPin, LOW);
```

```
   Buzz(1000);
   delay(500);                //Señal acústica de activación
   Buzz(0);
   ESTADO_ALARMA=HIGH;
   lcd.clear();
   lcd.setCursor(0,0);
   lcd.print("ALARMA ACTIVADA");
   delay(1500);
   lcd.clear();

  } else {
   digitalWrite(redPin, LOW);
   digitalWrite(greenPin, HIGH);
    ESTADO_ALARMA=LOW;
    ALERTA=LOW;
    digitalWrite(pin, LOW);
   lcd.clear();
   lcd.setCursor(0,0);
   lcd.print("ALARMA DESACTIVA");
   delay(1500);
   lcd.clear();

}}
```

- Cuando el código correcto de activación ha sido introducido, se activa la función que regula el sensor de presencia, esta se encarga de realizar de forma periódica lecturas, de forma ajena al bucle principal. En el caso de que se detecte una presencia, esta función a su vez accionará otra, la cual será la encargada de activar los avisos.

```
void SENSOR_PRESENCIA(int estado){
   if(estado==1){
    int val = digitalRead(Sensor_presencia);  //Lectura del sensor PIR
    Serial.println(val);
     delay(5);
  if (val == HIGH) {
   digitalWrite(pin, HIGH);
   if (Presencia == LOW) {
     lcd.clear();
     lcd.setCursor(0,0);
     lcd.print("ALERTA PRESENCIA");
      Presencia = HIGH;
      ALERTA=HIGH;
      Persiana_Alarma(); // Llamada a la función de bajada de persianas.
   }}
  else {
     digitalWrite(pin, LOW);
if (Presencia == HIGH){
       Presencia = LOW;
        }}}}
```

- En esta función se a activa el zumbador, para ello se emplean dos frecuencias distintas a 2864 y 1000, esta permanecerá activa hasta que se introduzca el código de desactivación en el panel numérico. Dentro de esta función encontramos la llamada a otra, *'Persiana_Alarma()',* la cual será la encargada de hacer que las

persianas bajen de forma automática en el caso de que se detecte una presencia en la estancia. Esto se utiliza como método de persuasión, en caso de que exista una presencia no deseada en la vivienda.

```
 //FUNCION ALARMA SONORA
void ring (int estado){
  if(estado){
  analogWrite(buzz,2864);
  delay(100);
  analogWrite(buzz,1000),
  delay(100);
  }else{
   analogWrite(buzz,0);
  }}
```

- La función de bajada de persianas, contiene algunas llamadas a otras funciones que todavía no han sido explicadas, puesto que se verán en el bloque de control de persianas.

```
//FUNCION BAJADA DE PERSIANAS AUTOMATICA
void Persiana_Alarma(){
    if(Persiana!=0){
       PERSIANA_ABAJO();
       delay(3000); // Regular tiempo subida
       PERSIANA_ESTOP();
       Persiana=0;
    }}
```

- Se puede observar en algunas de las funciones, el uso de códigos para el control de una pantalla LCD, la implementación de la misma se verá en apartados posteriores, en sistemas auxiliares.

3.8 BLOQUE DE CONTROL DE PERSIANAS

Una de las ventajas que ofrece un sistema domótico, es el poder automatizar de forma tanto manual, como remota la subida o bajada de persianas, este tipo de control es muy frecuente en instalaciones de viviendas u otros lugares que no tengan instalados otros sistemas domóticos, en este apartado se explicará como se ha implementado la programación y posteriormente se expondrá el modo remoto y la simulación del mismo. Para poder realizar una programación más adecuada a lo que sería una instalación real, se han realizado una serie de mediciones que se explicarán a continuación.

Este sistema de control de persianas, según las especificaciones previas, constará de un control tanto manual como automático de las mismas, el control automático irá enlazado a un sensor de luz, el cual será el encargado de medir el nivel de intensidad lumínico externo y hacer que las persianas suban o bajen.

Según las especificaciones, estas deben subirse a primera hora de la mañana y bajarse cuando el nivel de luz sea reducido, para ello se ha realizado una medición del nivel de luz externo a lo largo de un día completo, realizando mediciones cada media hora, de este modo podremos tener una idea de los niveles de iluminación a los que puede estar sometido el sensor empleado.

Se ha utilizado un módulo con sensor lumínico, dicho sensor utiliza una resistencia LDR para realizar las mediciones, el módulo y sus especificaciones básicas ya fueron tratados anteriormente.

Este sensor ofrece lecturas con valores que oscilan entre (0-1000) y se ha situado con orientación norte para evitar luz solar directa con las primeras horas del día.

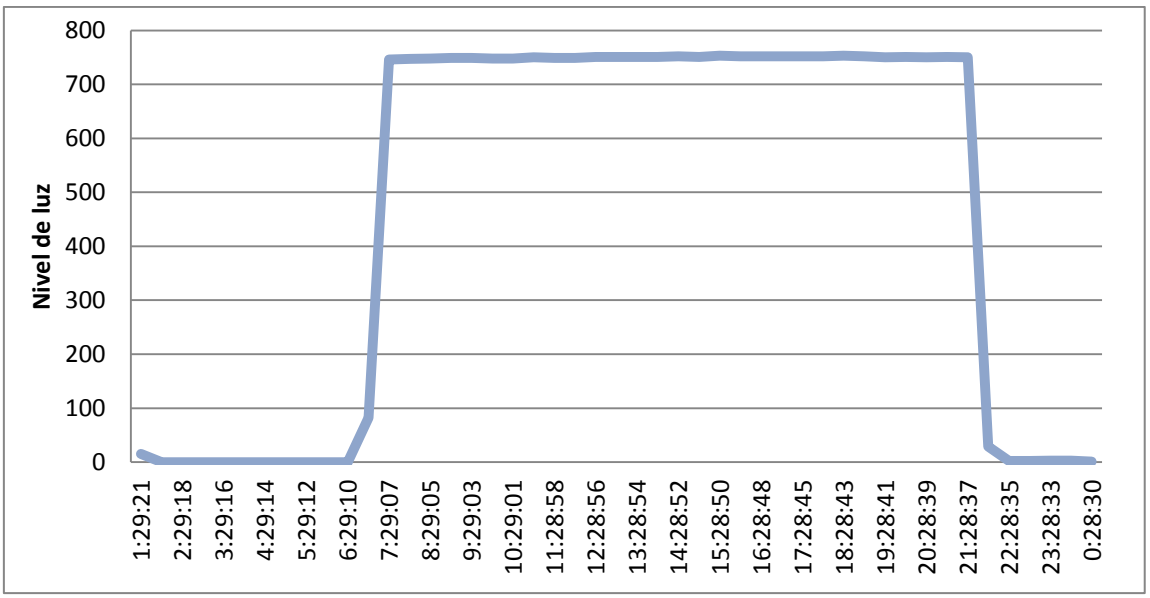


Figura 3.17: Gráfico nivel de luz externo

Como se puede observar en la (fig 3.17), el valor que coincide tanto en el amanecer como en el atardecer es aproximadamente 750, este valor se puede

considerar válido para nuestra programación puesto que se produce en un horario aceptable, 7:30 a.m y 21:30 p.m aproximadamente.

Del mismo modo que en todos los bloques, a continuación se expondrán las líneas de código más representativas, donde se explicarán las partes más importantes.

```
#define E1 9
#define I1 8      // Pines para el control del Puente H (L293D)
#define I2 7


int luz=0;                    //Variables para el control del modo automático
int SensorLuz=A1;

void setup()
  {
     for (int i = 8 ; i<11 ; i++)                 // Inicializamos los pines para el control del integrado
     pinMode( i, OUTPUT);
     pinMode(3,INPUT);
     pinMode(2,INPUT);
  }
```

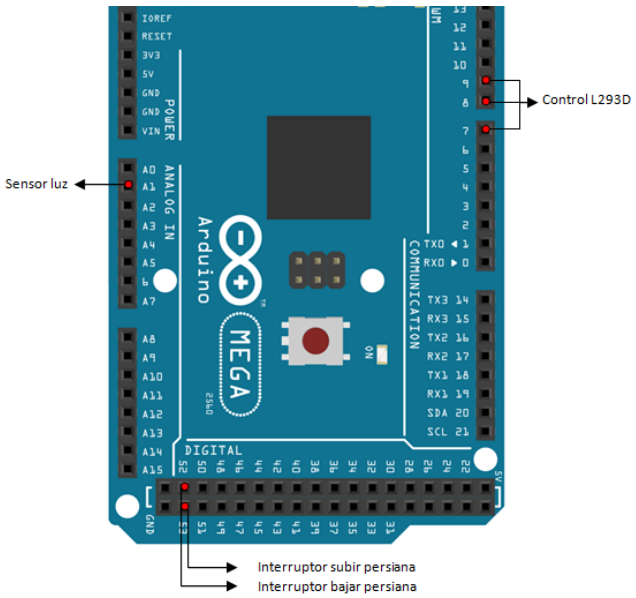


Figura 3.18: Pines control de persiana

- Para el control del motor de la persiana se va a utilizar un circuito integrado modelo *L293D,* sus características principales y conexiones, se verán en profundidad dentro del apartado de simulación. Este circuito

integrado nos permite variar el sentido de giro del motor en función de la polaridad de algunos de sus pines, en este caso se emplearán los pines de Arduino 7, 8,9.

Pin9 Activación	*Pin 7*	*Pin 8*	*Estado del motor*
0	-	-	Parado
1	1	0	Adelante
1	0	1	Atrás
1	1	0	Parado
1	0	1	Parado

Tabla 20: Control del sentido de giro del motor

- Para poder controlar el sentido de giro del motor, tanto para subir como para bajar las persianas, es completamente necesario conocer el estado en el que debe estar cada uno de estos pines.

```
void loop()
//LECTURA PULSADORES PERSIANA
Estado1=digitalRead(PERSIANA_BOTON_ARRIBA);    // Realizamos lecturas periódicas del estado de los pulsadores
Estado2=digitalRead(PERSIANA_BOTON_ABAJO);

Persiana=constrain(Persiana,0,50);  //Limitamos el movimiento para evitar que la persiana suba o baje en exceso
if((Persiana!=50)&&(Estado1==1)){
PERSIANA_ARRIBA();      //En el momento en el que se pulsa uno de los botones, el contador empieza a sumar
 Persiana++;
}else{
PERSIANA_ESTOP();
}
```

- Para limitar el tiempo de subida y bajada de las persianas se ha incluido un contador para cada uno de los pulsadores, de este modo evitamos que se fuerce de forma innecesaria el motor evitando fallos o averías. El número empleado en el contador es de 50, este valor ha sido obtenido de forma experimental, midiendo el tiempo de bajada o subida de forma manual e introduciéndolo en la programación posteriormente.

```
void PERSIANA_ARRIBA(){
    digitalWrite(E1, HIGH);     // Activamos Motor1
    digitalWrite(I1, HIGH);     // Arrancamos Motor
    digitalWrite(I2, LOW);
   delay(100);
 }

 void PERSIANA_ABAJO(){
     digitalWrite(E1, HIGH);     // Activamos Motor1
```

```
      digitalWrite(I1, LOW);      // Arrancamos con cambio de dirección
      digitalWrite(I2, HIGH);
      delay(100);
  }

  void PERSIANA_ESTOP(){
    digitalWrite(E1, LOW);
  }

void PERSIANA_AUTO(int estado){   // En esta función se introduce el sensor LDR este realiza la lectura de la luz externa y si esta
  if(estado==1){                      activo se encarga de subir o bajar las persianas.
 int luz =analogRead(SensorLuz);          // Realiza la lectura del sensor

if((luz<=750)&& (Persiana>=50)){      // Condiciona que si la persiana esta bajada y el nivel de luz es el adecuado, hace bajar las
   PERSIANA_ABAJO();                      persianas.
    delay(5000);// Regular tiempo subida
   PERSIANA_ESTOP();
   Persiana=0;
}}
```

- Tanto los tiempos de subida como el número de los contadores, son necesario variarlos dependiendo de algunos factores como la velocidad del motor empleado, o de la dimensión de cada persiana. Para ello es necesario realizar mediciones para posteriormente introducirlo en el código del sistema.

```
void Persiana_Alarma(){   // Esta función se acciona cuando el sensor de presencia detecta movimiento.
   if(Persiana!=0){
      PERSIANA_ABAJO();
      delay(5000);
      PERSIANA_ESTOP();
      Persiana=0;
```

- La función *'void Persiana_Alarma()'* se emplea en el bloque de control de acceso para hacer bajar de forma automáticas las persianas, en el caso de detección de presencias, cuando la alarma esta activada.

3.9 BLOQUE DE DETECCIÓN DE PRESENCIA DE GASES

La sensación de seguridad en instalaciones que posean flujo constante de gases tóxicos e inflamables es fundamental, para ello se va a implementar un sensor de presencia de gases, se ha seleccionado el modelo MQ-2 puesto que presenta un rango de lecturas que más se aproxima a los que podemos encontrar en una vivienda. La instalación de este sensor se realizará en la

cocina, puesto que es la zona más propensa para poder encontrar la mayor concentración de este tipo de gases.

Este sensor es capaz de detectar gas LP, butano, propano, metano e hidrógeno, posee una alta sensibilidad y un tiempo de respuesta muy corto, lo que lo hace muy útil para este tipo de entornos.

Se podrá encontrar más información técnica de este sensor en los anexos finales.

En este caso la programación es relativamente sencilla, puesto que no hay que ejercer ningún tipo de control sobre el dispositivo, este solamente se encarga de realizar lecturas periódicas mientras permanezca activo.

```
int SensorGas=A2; // Posición del sensor en la entrada A2
void setup(){
  pinMode(SensorGas,INPUT);
void loop(){
Gas(1); //Activa o desactiva el detector
}
```

- La activación o desactivación de esta alarma se puede ejercer desde el bucle principal, enviando el valor (0) o (1).

```
//FUNCION SENSOR DE GASES
void Gas(int estado){
  if(estado==1){
  int Lectura=0;
  Lectura=analogRead(SensorGas); //Lectura del sensor MQ-2
  if(Lectura>=500){
    Buzz(5000); //Llama a la función del zumbador y le envía una frecuencia
  }}}

// FUNCION BUZZER.
void Buzz(int sonido){
 analogWrite(buzz,sonido); //emite 1 sonido como alarma acústica
 }
```

Este tipo de dispositivos suelen ir asociados a otros sensores semejantes, de este modo se consigue un mayor rango de lecturas, como podrían ser detectores de concentración de humos, incendios o similares.

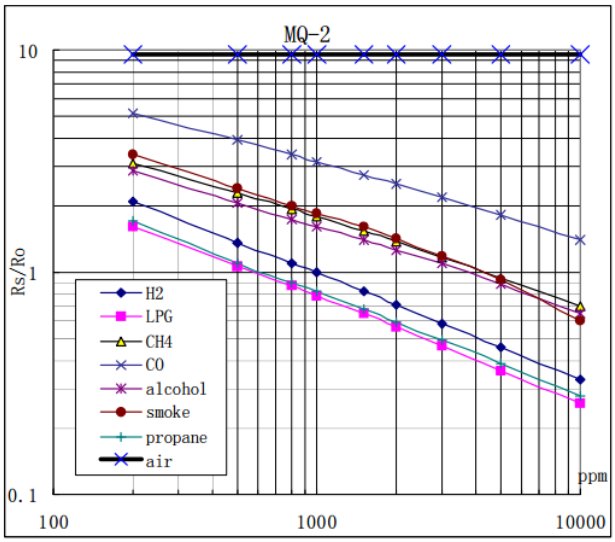


Figura 3.19: Rango de lecturas del sensor MQ-2

Este sensor es capaz de realizar mediciones de concentración de gases de hasta 10000 partes por millón, es por ello que es necesario que cuando se realice la instalación, el técnico evalúe características del entorno y de la instalación para poder ajustar algunos parámetros que hacen que este sistema de seguridad sea más efectivo. Algunas de estas características a tener en cuenta son, que tipo de instalación utiliza la vivienda, si se trata de gas natural, butano u otros, del mismo modo se ha de considerar las dimensiones del lugar de instalación así como la ventilación de la estancia. Estos factores variarán el ajuste que se hará al código para evitar que se produzcan grandes concentraciones que puedan llegar a ser peligrosas.

3.10 PROGRAMACIÓN AUXILIAR

Al margen de los bloques de diseño principales, tanto para mejorar como para aumentar la comunicación con el usuario, se introducen algunas de las funciones secundarias, es el caso, por ejemplo, del uso de pantallas de información como puede ser paneles táctiles o pantallas LCD. En este caso se

ha empleado una pantalla LCD 16x2 que proporcionará información en tiempo real al usuario, temperatura, humedad y estado de la alarma.

A continuación se mostrará el código de estos sistemas, tanto para la implementación de la pantalla LCD, como para mostrar información adicional, ya se del controlador PID o información básica de otros bloques.

```
#include <LiquidCrystal_I2C.h>  // Librerías para utilización de LCD
LiquidCrystal_I2C lcd(0x27,2,1,0,4,5,6,7,3, POSITIVE);
lcd.begin (16,2);  //Inicializamos la pantalla,  introduciendo sus dimensiones (16x2)
SERIAL_GENERAL(0);   // 0-Desactivar   1-Activar
SERIAL_PID(0);
CODIGO_LCD(1); //Activa o desactiva la pantalla LCD
```

- Utilizamos estas funciones auxiliares solamente si es necesario adquirir información adicional.

```
void SERIAL_GENERAL(int estado){
  if(estado==1){
char LUCES[8] =
{ESTADO_COCINA+48,ESTADO_SALON+48,ESTADO_DORMITORIO+48,ESTADO_ASEO1+48,ESTADO_ASEO2+48,ES
TADO_DIMMER+48,ESTADO_DIMMER2+48};  // Crea una cadena de caracteres con  el estado de la iluminación.
Serial.print("Estado luces:");
Serial.print("    ");
Serial.print(LUCES);
Serial.print("    ");
Serial.print("Temperatura: ");
Serial.print(t);
Serial.print("*C");
```

- Esta función ofrece mediante una impresión por pantalla, información general del sistema, desde una cadena de caracteres con el estado de la iluminación de cada estancia, incluyendo ambos dimmers, así como el estado de la lectura de los sensores de temperatura y humedad o el porcentaje de apertura de las persianas. Esta función actualmente solo esta implementada para ofrecer una mayor información de forma temporal, aunque podría utilizarse en aplicaciones futuras del propio sistema.

Ofrece la posibilidad de activar o desactivar esta impresión por pantalla, dependiendo de si se requiere o no, la información que ofrece, puesto que puede ralentizar al sistema.
En la siguiente imagen, se puede apreciar la información general del sistema mediante el monitor serial.

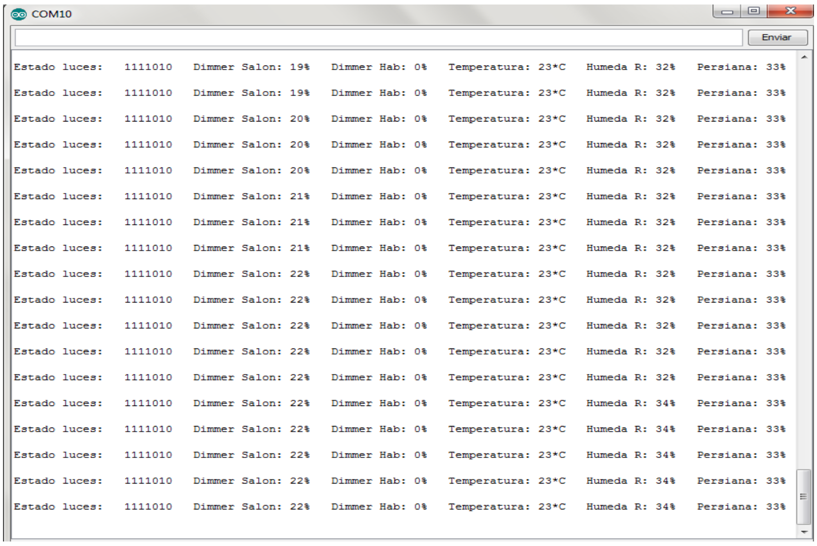


Figura 3.20: Monitor serial con información general del sistema

Del mismo modo también utilizamos este mismo monitor serial para conocer indicadores del controlador PID de temperatura.

```
void SERIAL_PID(int estado){
  if(estado==1){
  Serial.print("Temperatura:  ");
  Serial.print(Input);
  Serial.print("*C");
  Serial.print("        ");
  Serial.print("Setpoint:  ");
  Serial.print(Setpoint);
  Serial.print("*C");
  Serial.print("        ");
  Serial.print("Ventiladores:  ");
  Serial.print(map(Output,0,255,0,100));
  Serial.print("%");
  Serial.print("        ");
  Serial.print("Error:  ");
```

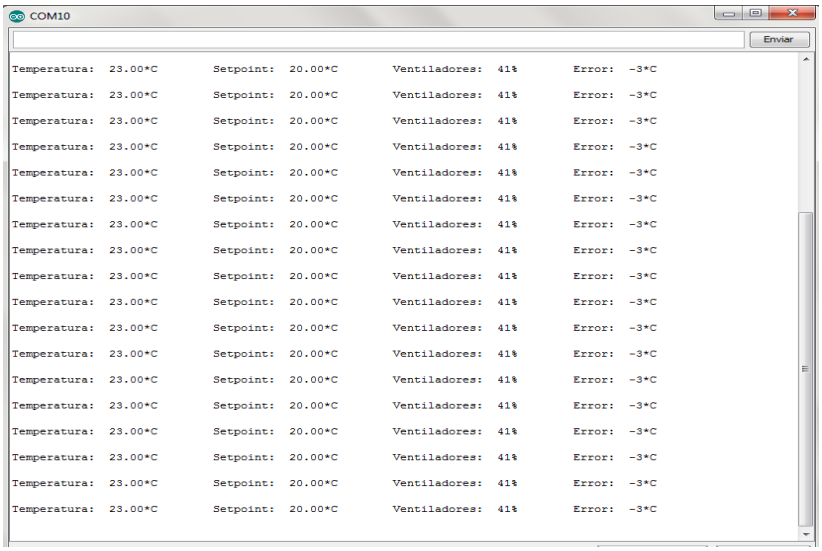


Figura 3.21: Monitor serial con información del controlador PID

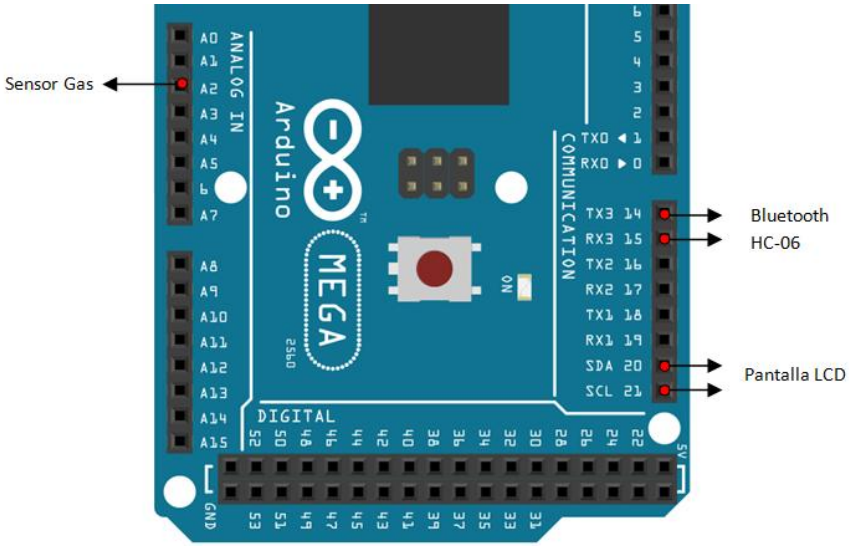


Figura 3.22: Pines auxiliares

En la (fig 3.22) se pueden ver los pines auxiliares utilizados, tanto para la conexión bluetooth como para la pantalla LCD o para el sensor MQ-2. La configuración de todos estos pines permite incorporar nuevos bloques y sensores al sistema general para adaptaciones futuras.

3.11. BLOQUE DE CONTROL REMOTO

El objetivo de este bloque de diseño, es la creación de una sencilla aplicación en sistema operativo Android, para poder controlar de forma remota algunos de los sistemas de la vivienda. Para ello se va a utilizar la herramienta online App Inventor2, puesto que no se poseen conocimientos previos de programación en Java, requisito indispensable para la creación de aplicaciones más avanzadas. Este método de desarrollo, permite la creación de una aplicación sencilla pero con un amplio margen de posibilidades, puesto que solamente se requiere de una interfaz de control.

A lo largo de este bloque se explicará y se mostrarán las distintas fases de creación de la aplicación y su respectiva programación en Arduino.

Este método de creación de aplicaciones consta de dos partes, la parte visual y la parte de programación. La parte visual denominada diseñador, es la encargada de insertar todos los aspectos gráficos que posteriormente veremos en la aplicación, es decir, botones indicadores y deslizadores, estos no realizan ningún tipo de función por sí mismos, si no que requieren que se les atribuya una programación para que realicen una acción. Este proceso se incluye dentro del módulo de bloques, dentro de este apartado se realizará la programación para cada uno de las partes. La programación es relativamente sencilla puesto que esta herramienta se emplea como método de formación y dentro del desarrollo de aplicaciones posee multitud de bloques predefinidos y ejemplos.

En la siguiente figura, se muestra un esquema general de las distintas partes de las que consta la aplicación diseñada. Su diseño ha sido realizado para asemejarse lo máximo posible a una aplicación comercial dentro del campo de la domótica, consta de un menú principal con información a tiempo real de la temperatura y humedad relativa de la vivienda, y a partir de la cual se podrá acceder a cada uno de los sistemas. Esta parte como se ha dicho anteriormente, es solamente visual, es necesario realizar la programación de las distintas partes que lo compone.

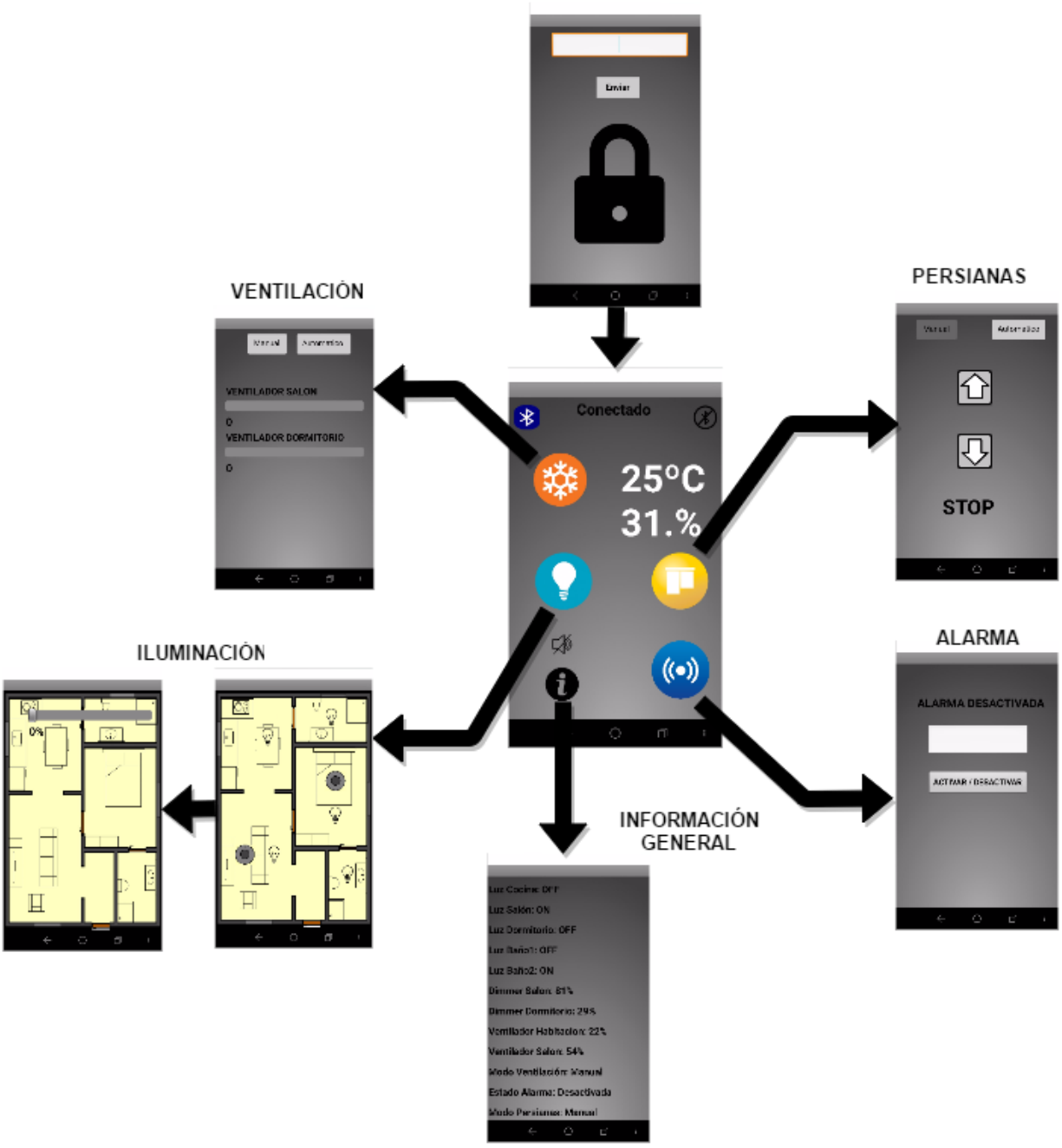
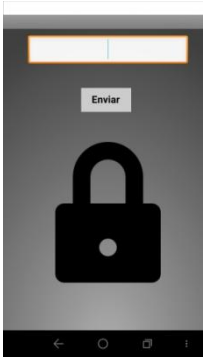


Figura 3.23 Esquema de partes de la aplicación

En la (fig 3.23) se puede encontrar algunos de los bloques que se han programado anteriormente, solamente aquellos que requerían un control remoto. A continuación se explicará cada uno de estos controles de forma más exhaustiva.



El primer sistema se trata de un control de acceso a la aplicación, debido a que este tipo de redes son muy vulnerables a accesos externos, de este modo aumentamos de forma considerable la seguridad al control de la vivienda.

Una vez se haya realizado la conexión con el dispositivo Bluetooth, aparecerá el menú principal, en el podremos acceder a cada uno de los controles principales, iluminación, climatizador, alarma y persianas. En la misma pantalla también podemos encontrar información a tiempo real acerca de la temperatura y humedad de la vivienda.

Dentro del sistema de climatización, encontramos dos modos de funcionamiento, el control manual y el control automático. El control manual permite al usuario asignar el nivel de potencia para cada uno de los ventiladores, mientras que el control automático, actúa de forma independiente en función de la temperatura captada por el sensor.

El sistema de control de iluminación está diseñado sobre un sencillo plano de las estancias de la vivienda, de esta forma es mucho más fácil y rápido poder situar cada una de las lámparas para poder accionarlas. Dentro de este control encontramos dos funciones distintas, la de encendido de lámparas y la de regulación de luces dimerizadas, es decir que es posible controlar su nivel de intensidad lumínica.

Desde el control remoto también se permite la opción de activar o desactivar la alarma. Del mismo modo que en Arduino, la aplicación utiliza las mismas contraseñas establecidas. De este modo el usuario puede ser capaz de desactivar la alarma antes de entrar a la vivienda evitando fallos en el sistema de seguridad.

Como información adicional, dentro de la propia aplicación se ha incluido un apartado de información genérica, estado de alarma, ventilación o iluminación, este solamente ofrece información del sistema remoto, es decir no incluyen los cambios que se produzcan manualmente, esto se incluirá en modificaciones futuras del sistema.

A continuación, se explicará como se programa cada una de las funciones de forma general, del mismo modo se expondrá el método de transmisión de datos entre la aplicación y Arduino.

App Inventor 2, utiliza para su programación un sistema de bloques, es decir, todas las funciones están predefinidas en bloques java, esto facilita considerablemente el desarrollo de una aplicación en Android, si no se poseen conocimientos previos puesto que el aprendizaje es mucho más rápido.

Además esta herramienta ofrece la posibilidad de probar, o descargar la aplicación creada para poder detectar fallos mediante simuladores en el propio ordenador, de esta forma se agiliza el proceso de desarrollo.

Puesto que para implementar la aplicación es necesario utilizar multitud de bloques de diseño, solamente se mostrarán los más representativos, como podría ser los de conexión o envío de datos.

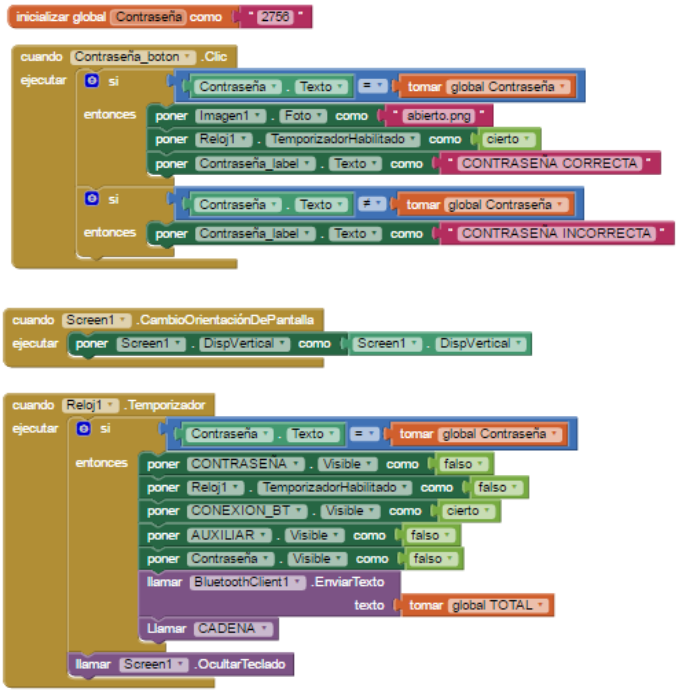


Figura 3.24: Programación en App Inventor 2

En la (fig 3.24) se muestra como ejemplo, la programación del sistema para realizar la conexión a la señal Bluetooth, puesto que para programar cada uno de los botones es necesario utilizar gran cantidad de bloques no se mostrara por completo la programación de cada uno de ellos.

Para realizar la transmisión de datos, utilizaremos el Serial 3 del cual dispone este modelo de Arduino, para establecerlo como canal de comunicación entre el dispositivo móvil y el controlador.

Para enviar datos a través de Bluetooth, se utilizará una cadena de caracteres, es decir, cada vez que modificamos un parámetro en la aplicación, modificamos un carácter dentro de una cadena, que posteriormente será enviada a Arduino. Se utiliza este método porque enviando caracteres de forma individual ralentizamos considerablemente la transmisión, de este modo

solamente se realizará un envío cada 2 segundos como se ha definido dentro de la propia aplicación. Posteriormente Arduino procesará esta cadena y realizará las acciones pertinentes.

Para ello asignamos una variable a cada uno de los valores que deseamos enviar, como puede ser el encendido de las luces de una estancia aplicando un 1 o 0, y lo unimos a la cadena en el espacio que corresponda, para poder enviarla posteriormente. Este sistema permite la posibilidad de introducir nuevas variables o modificar las existentes, aunque es fundamental conocer la posición de cada una de ellas, para poder reconocerla cuando sea recibida por el controlador.

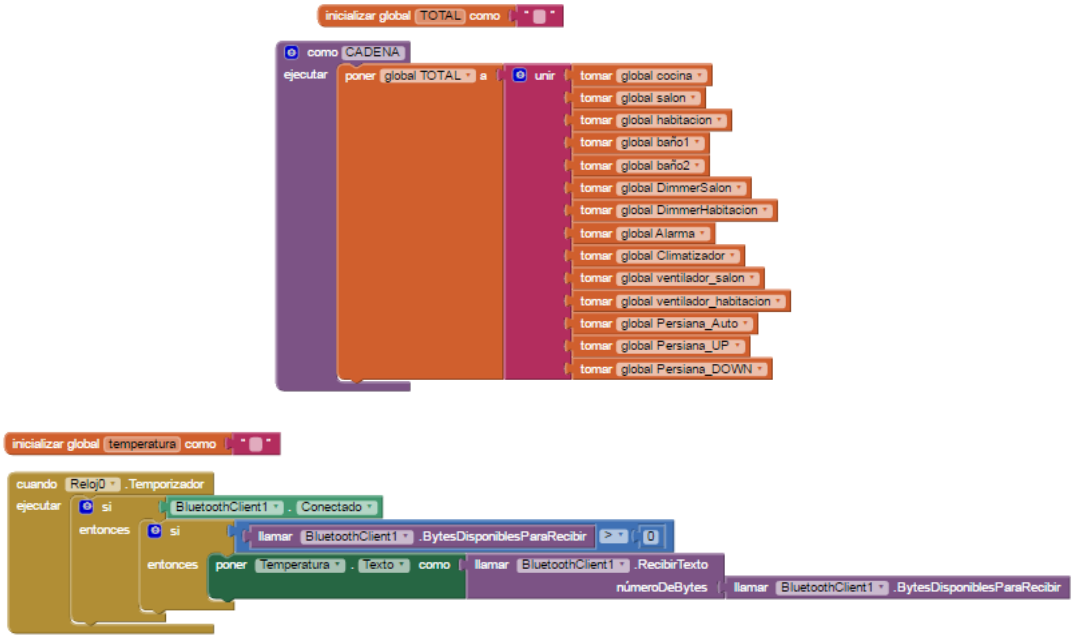


Figura 3.25: Transmisión y recepción de datos App inventor2

En la (fig 3.25) se observa como dentro de la programación de la aplicación, unimos todas las variables para formar una sola cadena que posteriormente se enviará de forma automática. Esta cadena, la cual está formada por 14 variables distintas, controla cada una de las acciones remotas que podemos llevar a cabo en la vivienda.

A continuación, se explicará la programación realizada en Arduino, para recibir y procesar la cadena de datos para posteriormente, realizar cada una de las tareas correspondientes.

Figura 3.26: Posiciones dentro de la cadena de transmisión

```
char Estados[14];       / /Creamos un array que almacenará los caracteres que
byte posicion=0;          escribiremos en la consola del PC.

Void setup()
 Serial3.begin(9600);

void loop() {


  //Recepcion Serial3 Bluetooth
  if(Serial3.available())
  {

 while(Serial3.available()>0)      //Mientras haya datos en el buffer ejecuta la
   {                                 función
     delay(5);
     Estados[posicion]=Serial3.read(); //Lee un carácter del string "cadena" de la "posición", luego lee el siguiente carácter con
"posicion++"
     posicion++;
   }}

int Remoto_cocina=Estados[0]-48;     // Asignamos posiciones para cada una de las variables del control remoto
int Remoto_salon=Estados[1]-48;
int Remoto_dormitorio=Estados[2]-48;
int Remoto_aseo1=Estados[3]-48;
int Remoto_aseo2=Estados[4]-48;
```

- Se crea una variable para cada una de las posiciones de la cadena recibida y posteriormente la asignamos a la posición correspondiente. De este modo se podrá utilizar cada una de ellas de forma independiente para realizar distintas tareas.

```
if(PULSADOR_COCINA && Remoto_cocina ){  // Conmutación entre recepción
LUCES(LUZ_COCINA,1);                          serial e interruptor
 ESTADO_COCINA=1;
}else{
  LUCES(LUZ_COCINA,0);
    ESTADO_COCINA=0;
```

- Se emplean estas variables para realizar funciones como encendido de lámparas, como sucede en este caso que se emplea como conmutador para encenderlas tanto de forma manual como remota.

```
//Dimmer Remoto
if(Remoto1_dim>=0){
Dim1=map(Remoto1_dim,0,9,0,255);   //Mapea el caracter recibido (0-9) y lo
}                                          Transforma en un valor entre (0-255)
if(SUM_DIMMER>=Dim1){
DIMMER(LUZ_DIMMER_SALON,SUM_DIMMER);
}else{
 DIMMER(LUZ_DIMMER_SALON,Dim1);
}
```

- Puesto que el caracter recibido para el control de luces dimerizadas, posee un valor entre (0 y 9) es necesario realizar un mapeo para poder utilizar las salidas de Arduino, de este modo obtenemos salidas ente (0 y 255)

```
Vent1=map(Remoto_Ventilador1,0,9,0,255);   //Mapeamos la variable de control de los ventiladores
Vent2=map(Remoto_Ventilador2,0,9,0,255);
if(Remoto_Climatizador==0){               // Si el modo manual del remoto esta activado,  realiza las siguientes acciones.
  myPID.SetMode(MANUAL);
 analogWrite(ventilador1,Vent1);          // Asigna a  la salida analógica el valor obtenido
 analogWrite(ventilador2,Vent2);
            }
```

- Este mismo caso sucede, para el controlador de ventiladores, cuando queremos asignar manualmente la potencia de cada uno de ellos, de modo contrario el sistema actuará de forma automática

```
Serial3.print(t);
Serial3.print("ºC");
Serial3.print("\n");
Serial3.print(h);
Serial3.print(".%");.
```

- Para enviar datos en otro sentido, es decir desde Arduino al dispositivo móvil, solamente es necesario utilizar el *Serial3* de Arduino, todas las lecturas realizadas serán recibidas en la aplicación.

- La programación del control remoto para cada uno de los distintos bloques es muy similar, se utiliza el estado de la variable recibida para condicionar una u otra acción. De este modo conseguimos controlar las acciones tanto manual como de forma remota desde el dispositivo móvil.

Podremos encontrar la programación completa de sistema en el *Anexo IV* ya que solamente se han utilizado las partes concretas más representativas para poder explicar su funcionamiento.

4.0 SIMULACIÓN

El objetivo de este apartado es explicar como se han simulado algunos de los procesos, con el objetivo de adaptarlos a un funcionamiento real. Se han recreado de forma física controles manuales como es el caso de interruptores o pulsadores, para el control de la iluminación o persianas, del mismo modo se han incorporado cada uno de los componentes a la maqueta para obtener un resultado final más funcional. A lo largo de este apartado se incorporarán fotografías de la instalación de los distintos sistemas a la maqueta.

4.1 SIMULACIÓN CONTROL MANUAL DE LÁMPARAS

Para la simulación de los distintos accionamientos manuales, es decir interruptores y pulsadores, para el control de luces dimerizadas, se van a emplear una serie de dispositivos electrónicos, para poder recrear de forma más efectiva todo el conjunto.

Se han empleado los siguientes materiales:

- 5 Interruptores de dos posiciones para el control de las lámparas.
- 4 Pulsadores para el control de las luces dimerizadas.
- 7 Resistencias de 10KΩ ±5%
- 2 Transistores modelos TIP120
- 1 Fuente de alimentación para protoboard.

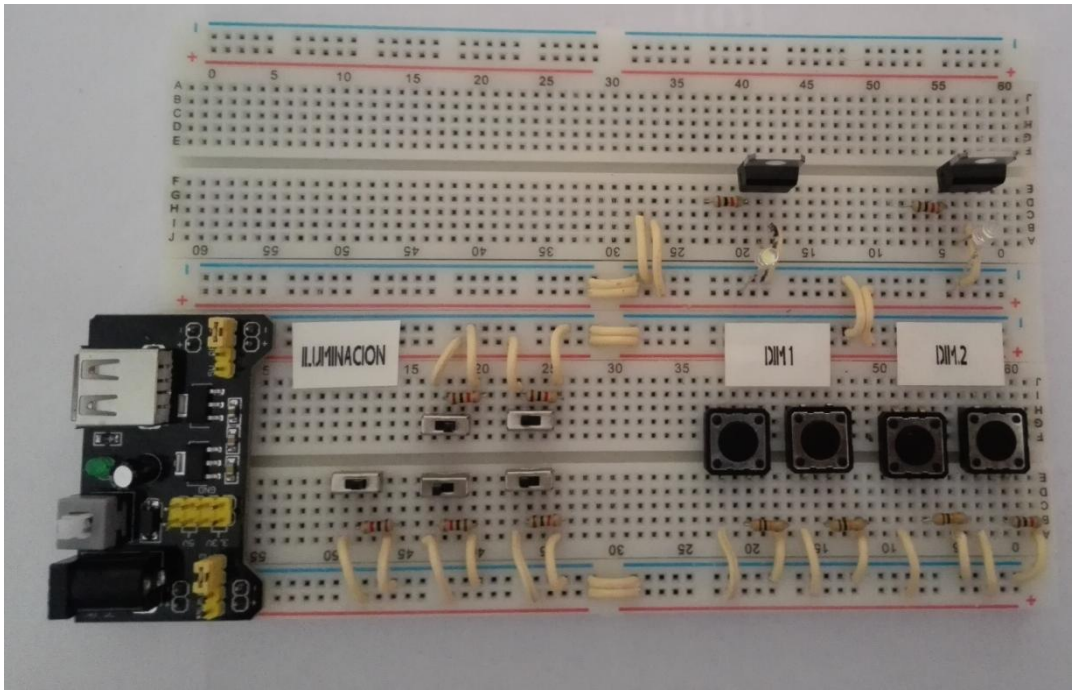


Figura 4.0: Control manual de iluminación

El montaje de estos distintos circuitos electrónicos, tiene como objetivo obtener finalmente una mejor simulación del sistema, de esta forma no solo se limita a

un control remoto, si no que se emplea un control manual de algunos de los accionamientos más utilizados en la vivienda. Ambos controles se aplicarán de forma simultánea a la maqueta.

En este apartado se explicará como se han conectado cada uno de estos elementos, para conseguir el funcionamiento requerido, así como algunas de sus características, aunque del mismo modo que en apartados anteriores, se incluirán sus fichas técnicas en los anexos finales del trabajo.

La conexión tanto de los pulsadores como de los interruptores esta realizado utilizando una conexión de 'Pull-Down' con la resistencia y el pin de lectura.

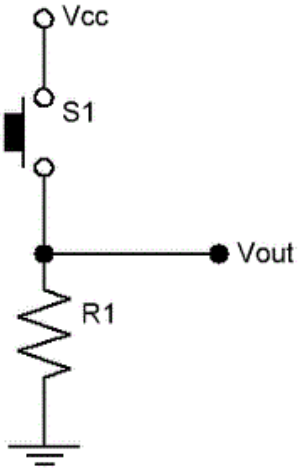


Figura 4.1: Conexión Pull-Down

A la hora de utilizar pulsadores, es necesario que funcionen en dos estados distintos, HIGH o LOW, pero cabe la posibilidad de que se obtenga un valor indefinido debido a ruidos electrónicos, interferencias con la fuente de alimentación u otros factores externos, estos factores impiden determinar si se encuentra en uno de los dos estados, para solucionar este problema se utilizan resistencias Pull-Down y Pull-Up.

Como se observa en el esquema de la (fig 4.1) la resistencia Pull-Down es conectada a tierra, de este modo cuando el interruptor está abierto la corriente es dirigida hasta la misma dejando un valor 0 (LOW), y en el caso que el pulsador o interruptor este accionado, la corriente se desplazará hacia Vout, en nuestro caso el pin de lectura de Arduino, obteniendo un valor 1(HIGH).

A continuación se explicará como se ha implementado el sistema para el control de las lámparas dimerizadas y algunos de los componentes utilizados.

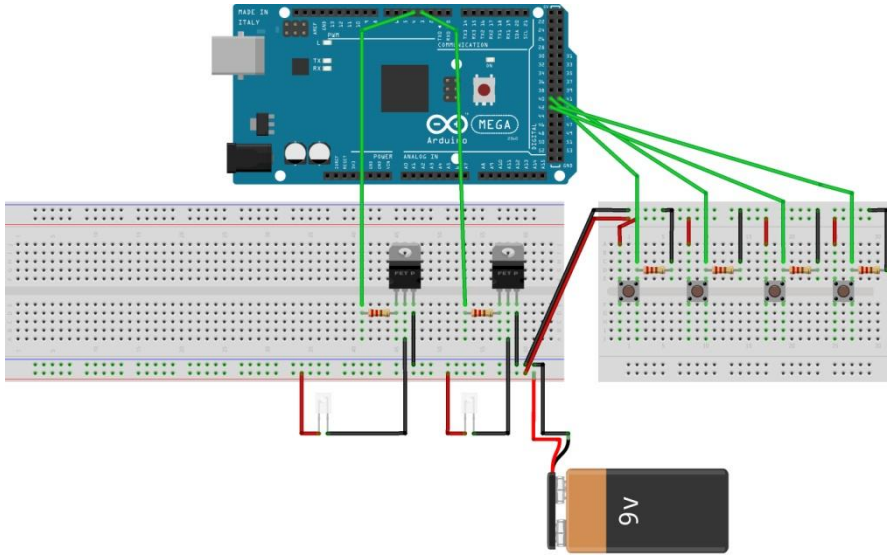


Figura 4.2: Conexión transistor TIP120

Para la simulación del control de luces dimerizadas se va a emplear el transistor modelo TIP 120, este tipo de dispositivos, son ideales para el control de intensidad de brillo en leds, ya que permiten el control de iluminación a partir de tensiones muy reducidas, en la (fig 4.2) se muestra el esquema de conexiones para un correcto funcionamiento. Se podrá encontrar información técnica de este dispositivo en el *Anexo V.*

Para implementar la iluminación en la maqueta, se ha utilizado tiras led de luz blanca, este tipo de led funciona a una tensión de 12v, lo que facilita su utilización en la simulación. Del mismo modo también se emplean leds blancos de 3.6v, para la simulación de las lámparas dimerizadas.

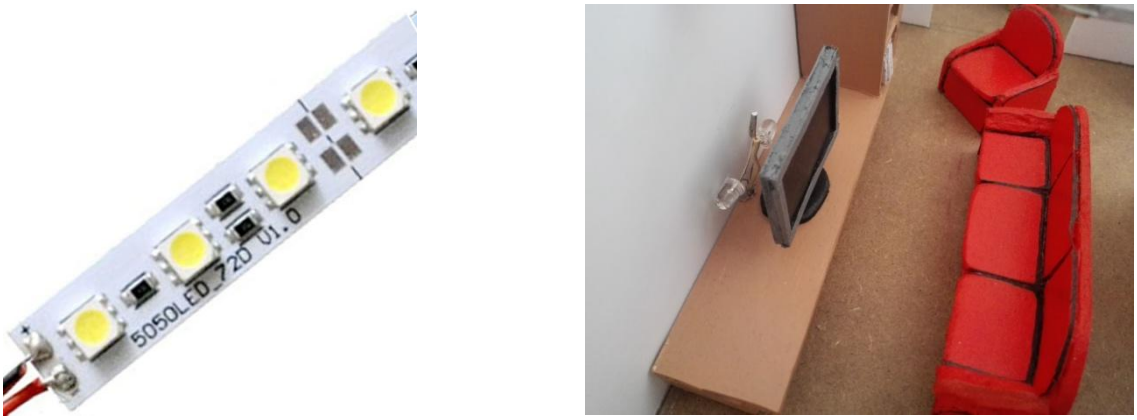


Figura 4.3: Lámparas utilizadas en la simulación

Para aumentar la similitud con un sistema real, aunque no es necesario para la seguridad, puesto que se utilizan tensiones muy reducidas, se emplearán bloques de relés para activar o desactivar la iluminación de cada estancia

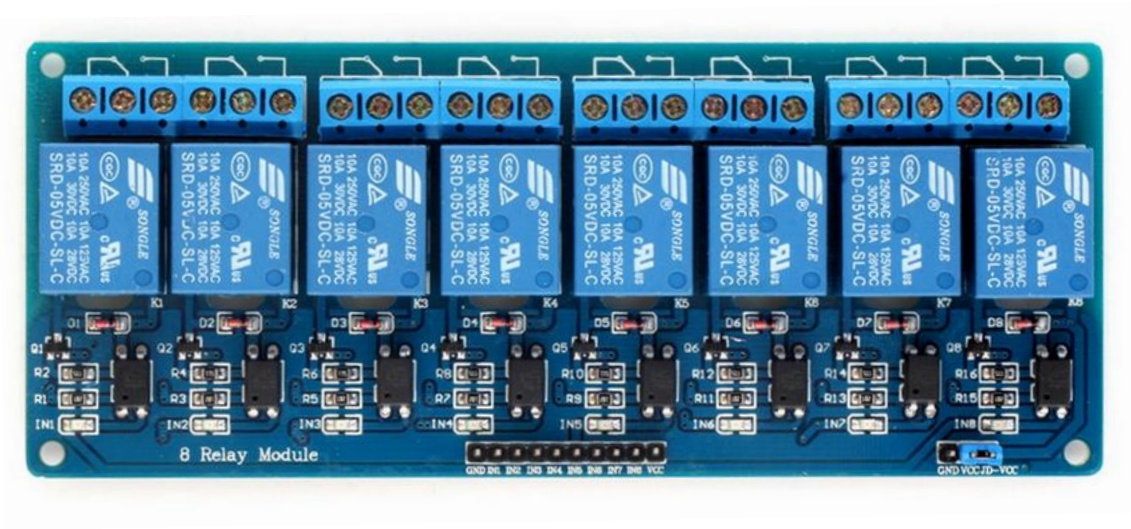


Figura 4.4: Módulo de relés para iluminación

Este tipo de relés son de fácil instalación, puesto que constan de jumpers de conexión adaptados a Arduino. De este modo se consigue crear un aislamiento entre el sistema de control y el de iluminación, evitando exponer algunos de los componentes a tensiones elevadas.

Estos módulos, son aptos para un uso real en una vivienda, puesto que son capaces de soportar corrientes elevadas.

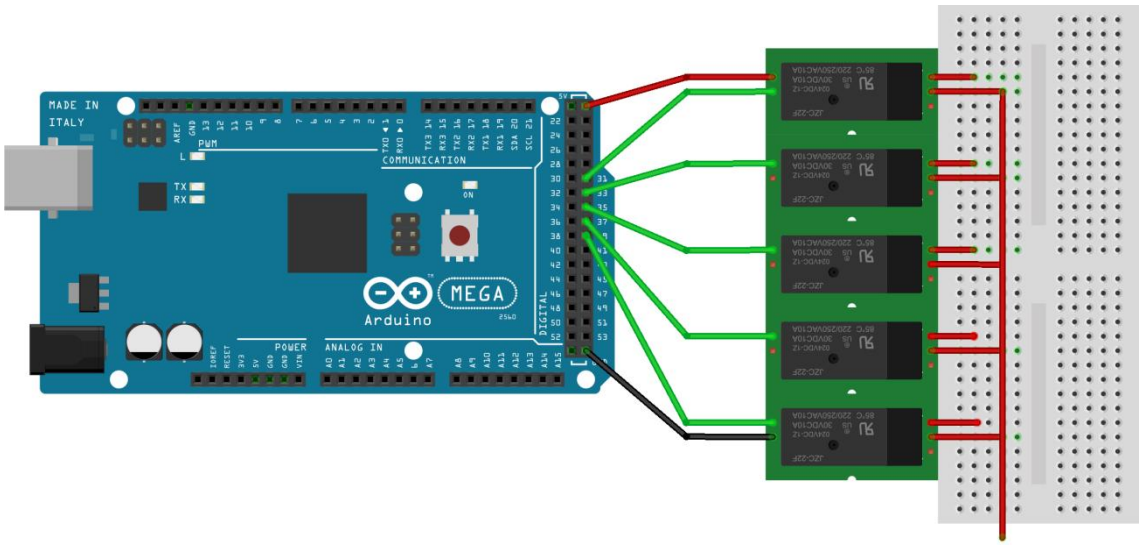


Figura 4.5: Conexión módulos de relés

4.2 SIMULACIÓN CONTROL MANUAL DE PERSIANAS

Para la simulación de este bloque de control se emplearán los siguientes componentes electrónicos:

- 2 Pulsadores
- 2 Resistencias de 10KΩ ±5%
- 1 Circuito integrado L293D
- 1 Motor DC12V 14RPM

La conexión de los pulsadores se ha realizado de la misma forma que en el bloque de iluminación, se han empleado dos resistencias Pull-Down para evitar estados indefinidos.

Se ha utilizado un circuito integrado modelo L263D, conocido como puente H para el control del motor, este permite controlar el sentido de giro y la velocidad de hasta dos motores.

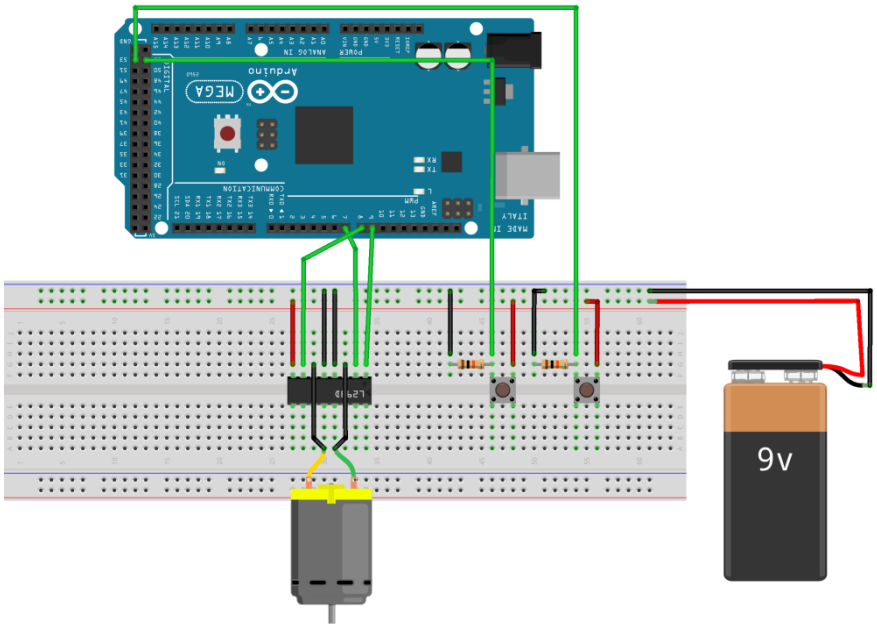


Figura 4.6: Conexiones integrado L293D

En la (fig 4.6) se observa como es necesario conectar cada uno de los pines de este circuito integrado para su correcto funcionamiento. Se podrá encontrar más información técnica en el *Anexo VI*.

El motor empleado para la simulación, se trata de un motor de reducidas revoluciones por minuto, a con una tensión de funcionamiento adecuada para esta tarea.

Figura 4.7: Motor persianas

Modelo	**S30K**
Voltaje	**DC 12V**
Velocidad de carga	**14 rpm**
Torque	**1.2 kg.cm**

Tabla 21: Características motor de persianas

Figura 4.8: Simulación de persianas

4.3 SIMULACIÓN DEL CONTROL DE ACCESO

Se ha empleado para la simulación de este sistema los siguientes materiales:

- 1 Teclado keypad
- 1 Pantalla LCD 16x2
- 1 Caja de derivación (15 x 22 x 5)
- 2 Leds
- 1 Zumbador
- 1 Sensor PIR HC-SR501

Para recrear el sistema de control, se han instalado todos los dispositivos en el interior de una caja de derivación, adaptándola para ello, del mismo modo se ha empleado esta caja como soporte para el control central de Arduino y para otros dispositivos.

Figura 4.9: Simulación de control de acceso

Este módulo de control, permite visualizar mediante la pantalla LCD, el código introducido para su desactivación, del mismo modo se han instalado dos diodos led de color rojo y verde para señalizar en todo momento el estado en el que se encuentra la alarma. Una vez que se ha activado o desactivado la misma, la pantalla mostrará información de temperatura y humedad del sistema a tiempo real. Esta información puede ser modificada dentro del código principal de programación, para poder visualizar otros aspectos, como podría ser el estado de las luces o el de las persianas.

La funcionalidad de este sistema podría mejorarse incorporando otro tipo de interfaces, como podría ser el uso de pantallas táctiles, o pantallas LCD de mayor tamaño y resoluciones, estos aspectos se incorporarán junto a otros, en el apartado de modificaciones futuras, donde se propondrán mejoras para el diseño actual.

Figura 4.10: Simulación sensor PIR control de acceso

En la (fig 4.10) se observa como se ha instalado el sensor PIR dentro de la maqueta en el lugar de instalación que ocuparía este sensor en una vivienda real, de este modo conseguimos una recreación mas visual.

Para realizar todas la conexiones se han empleado cables adaptados jumper para pines de Arduino, del mismo modo cuando es necesario realizar conexiones de mayor distancia se ha empleado cable telefónico *(Ø 0.5mm x2)* empleando adaptadores jumper para facilitar las conexiones.

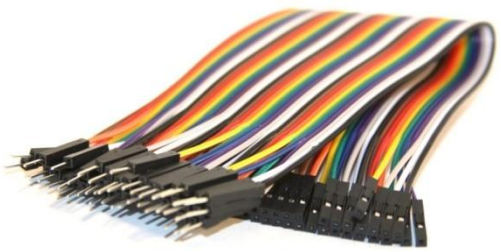


Figura 4.11: Cable jumper para conexiones

4.4. SIMULACIÓN DE CONTROL DE TEMPERATURA

Para simular el control térmico de la vivienda, se emplearán dos ventiladores de reducidas dimensiones, instalados en las estancias seleccionadas, con una tensión de funcionamiento de 5v, así mismo se empleará el sensor de temperatura y humedad THD 11.

Figura 4.12: Simulación de ventiladores

Figura 4.13: Ventilador de refrigeración

Modelo	**Sunon MC30060V2-A99**
Voltaje	**DC 5V**
Dimensiones	**25x25x6mm**
Potencia	**0.38W**

Tabla 22: Características ventilador

4.5. INSTALACIÓN DE OTROS DISPOSITIVOS

La instalación del resto de dispositivos se ha realizado utilizando una protoboard, para poder conectar y alimentar correctamente cada uno de estos elementos, por ello, se han ubicado en el interior de la caja de derivación, de este modo se consigue reducir de forma considerable el número de conexiones que es necesario realizar entre el control central y la maqueta, consiguiendo una simulación más compacta y una disminución de fallos en sus conexiones.

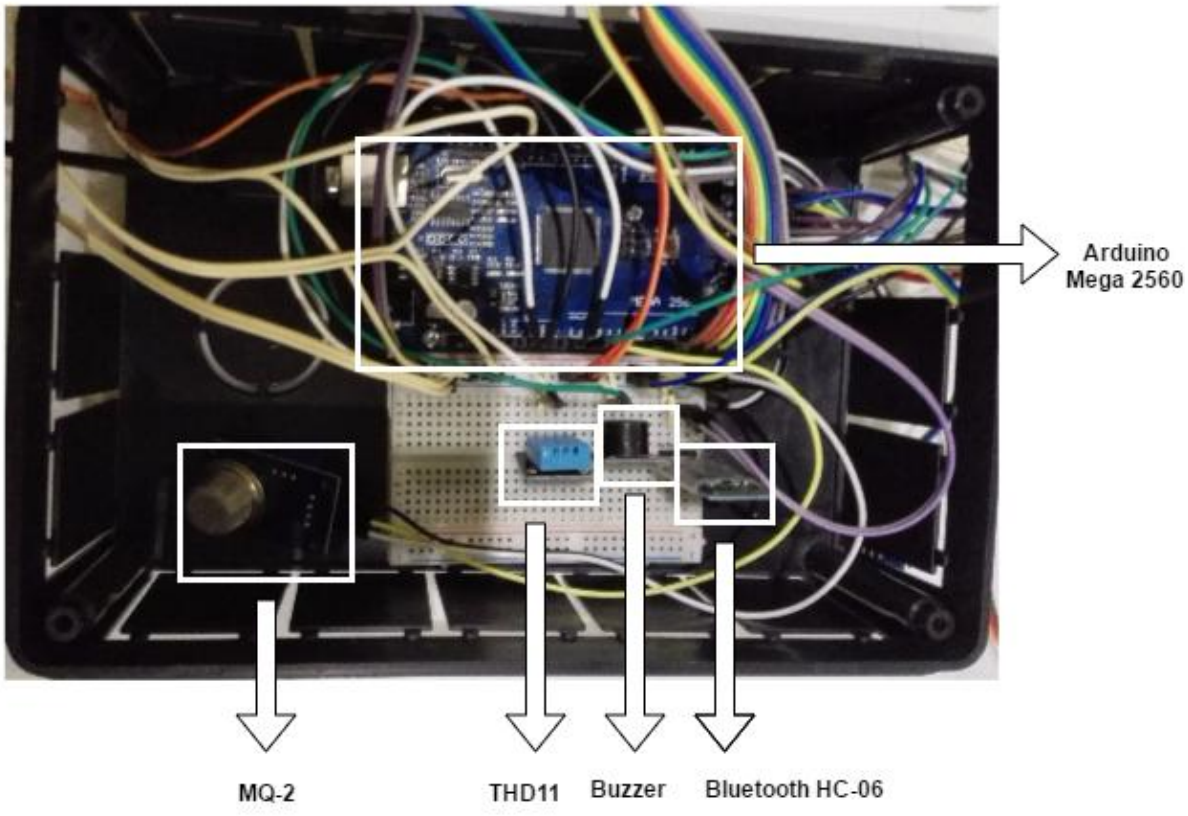


Figura 4.14: Ubicación de dispositivos

5.0. MODIFICACIONES FUTURAS

En este apartado, se propondrán algunas de las modificaciones que se podrán realizar posteriormente para mejorar el diseño actual del sistema, puesto que las posibilidades que ofrece este tipo de diseño son prácticamente infinitas, siempre es posible incorporar nuevos dispositivos o realizar cambios en la programación realizada, para ello se propondrán algunas correcciones que permitirán mejorar el funcionamiento general y facilitar al usuario tanto su accesibilidad como su uso.

Los cambios propuestos para el diseño actual son los siguientes en orden de preferencia:

- Mejorar la transmisión de información entre el control central y el control remoto, puesto que el sistema actual es prácticamente unidireccional, solamente es capaz de enviar señales del control remoto al control central, con la excepción de conocer a tiempo real la temperatura y humedad relativa del sistema. Será necesario establecer por ello, un método de transmisión bidireccional, para que el usuario sea capaz de conocer los cambios realizados de forma manual en la interfaz móvil, y tomar las decisiones de control pertinentes en consecuencia.

- Se propone la posibilidad de cambiar el sistema de transmisión remota de un control mediante dispositivo Bluetooth, a una transmisión mediante Ethernet, es decir, que establezca un control de los sistemas a través de internet, lo que permitiría al usuario conocer el estado de los sistemas fuera de la vivienda, mediante una conexión inalámbrica accediendo a un servidor propio desarrollado para esta función.

- Para un control de las persianas más eficaz, se pretende utilizar dispositivos como finales de carrera u otro tipo de accionamientos que permitan controlar la subida y bajada de las mismas, puesto que el diseño actual puede causar fallos si el módulo de control central se reinicia.

- Se podrá mejorar la interfaz de control, instalando una pantalla de información de mayor tamaño, incluso un panel táctil, esto facilitaría al usuario conocer con mayor facilidad el estado de los sistemas, puesto que la pantalla LCD utilizada actualmente, limita por su tamaño la cantidad de información que puede mostrarse.

- La mejora de la aplicación móvil desarrollada, también es un punto a tener en cuenta como modificación futura, puesto que se ha desarrollado desde un inicio, sin tener conocimientos previos de este tipo de implementación, se podrá mejorar tanto a nivel visual como a nivel de control, incorporando nuevas funciones y mandos.

- Para poder mejorar el sistema de regulación de temperatura, se podrá incorporar al control remoto, la posibilidad de que el usuario pueda seleccionar la temperatura de la vivienda para el control automático

- Se puede incorporar, para incrementar el nivel de seguridad, otros dispositivos, como podría ser una cámara de video vigilancia, sensores de apertura en las ventanas o sensores de presencia en otras estancias.

  Del mismo modo se podrán instalar sensores de incendio o de inundación.

6.0. CONCLUSIONES

Finalmente, una vez desarrollado e implementado el sistema, se puede observar que el funcionamiento del mismo es el deseado, y que se cumplen los requisitos propuestos al inicio del trabajo, por lo que podemos afirmar que la solución adoptada ha sido la correcta, a pesar de que existe la posibilidad de mejorar cada uno de los bloques diseñados, ya que el nivel de evolución en los sistemas Arduino está en continuo auge, con la aparición de nuevos dispositivos y métodos de control.

Durante el desarrollo de esta memoria he indagado en los métodos de control existentes, pudiendo conocer la gran ventaja que supone actualmente la instalación de controles domóticos en una instalación, mejorando de forma considerable el confort de una persona dentro de su propia casa, algunos de estos controles permiten incrementar el nivel de seguridad, el cual considero que es fundamental para el usuario, y del mismo modo se le facilita conocer el estado de cada uno de los sistemas de la propia vivienda, pudiendo hacer un uso más eficaz de los mismos, obteniendo con ello un gran ahorro energético.

Por otro lado, puedo afirmar haber cumplido satisfactoriamente mi objetivo personal con este proyecto, el cual era aprender a programar y controlar sensores y actuadores mediante la plataforma Arduino, puesto que no poseía conocimientos previos, este ha servido como iniciativa para poder impulsar el aprendizaje dentro de este área, obteniendo como resultado conocimientos muy útiles que me permitirá aplicarlos en otros sectores muy distintos, del mismo modo, durante desarrollo de diseño e implementación del proyecto, he podido indagar en algunos de los campos, en los que previamente me he formado durante el transcurso de mi carrera, como podría ser en el desarrollo de circuitos electrónicos, implementación de controles automáticos y programación de tarjetas electrónicas, por lo que considero que la elección de este trabajo ha sido la correcta para poder desarrollar de forma práctica algunos de los conocimientos adquiridos durante estos años.

Así mismo la incorporación al mundo de Arduino, me permite abrir un gran campo de posibilidades, que actualmente está en continua evolución, lo que me facilitará poder desarrollar posteriormente otro tipo de proyectos personales, utilizando para ello esta plataforma. Del mismo modo los conceptos adquiridos durante la creación de la aplicación móvil, serán utilizados posteriormente para el desarrollo de otras interfaces u otras funciones prácticas.

7.0. CLASIFICACIÓN DE LA INSTALACIÓN

Para poder clasificar una instalación domótica es necesario considerar las siguientes características:

TIPO DE DISPOSITIVO	NUMERO DE PUNTOS
Detector de presencia	1
Sirena interior	1
Teclado cifrado o llave electrónica	1
Detector de concentración de gas	1
Control de persianas	4
Regulación de iluminación	2
Control remotos de sistemas	4
Control térmico	1
TOTAL	**15**

Tabla 23: Clasificación de la instalación

- **Clase I.** Se tratar de instalaciones con un nivel mínimo de dispositivos u aplicaciones domóticas instaladas. Para que una instalación pueda clasificarse en este nivel, debe tener un mínimo de 13, siempre que se cubran al menos 3 aplicaciones distintas, es decir estos 13 puntos de control deben conseguirse mediante 3 aplicaciones distintas, como podría ser control de iluminación, alarma y control de persianas. No conseguiría esta clasificación si no se cumpliesen estas características.

- **Clase II.** Se trata de instalaciones con un nivel medio de dispositivos u aplicaciones domóticas instalados. Para que una instalación este dentro de este nivel, la suma de los puntos debe ser de un mínimo de 30, siempre que se cubran al menos 3 aplicaciones, para poder clasificarse dentro de este nivel.

- **Clase III.** Son instalaciones con un alto nivel de dispositivos u aplicaciones. Puesto que para clasificarse en este nivel, la suma de los puntos debe ser de un mínimo de 45, aunque en este caso se repartirán entre al menos 6 aplicaciones distintas.

Conociendo estos parámetros podemos clasificar la instalación diseñada como instalación de nivel I.

Una vez clasificada la instalación, es necesario conocer la normativa a la que deberá estar sujeto, puesto que toda instalación eléctrica en una vivienda, tanto de alta como de baja tensión debe estar regularizada.

7.1. NORMATIVA

Para los dispositivos de carácter voluntario que afectan a las instalaciones domóticas, encontramos las siguientes normativas aplicables a instalaciones domóticas e inmóticas tanto de carácter nacional como europeo:

- UNE-EN 50090
- UNE-EN 50491
- EA 0026 Especificación técnica nacional
- CLC/TR 50491-6-3

A continuación se presentan las principales prescripciones de cada una de ellas:

Normas UNE-EN 50090 para Sistemas Electrónicos de Viviendas y Edificios (HBES)

Las normas UNE-EN 50090 se encargan de normalizar el control de sistemas y aplicaciones de comunicación abierto destinadas a viviendas y edificios. Estas normativas, engloban todos los dispositivos electrónicos conectados a través de una red digital de transmisión, teniendo en cuenta los sistemas de automatización, tanto si son distribuidos como descentralizados.

Esta serie de normativas se enfocan en comunicaciones de clase I, es decir, comunicación de baja velocidad y destinada al control. Como podría ser alumbrado, calefacción, alarmas, control de persianas entre otros.

- *EN 50090-3-1:1994 Sistemas electrónicos para viviendas y edificios (HBES). Parte 3-1: Aspectos de la aplicación. Introducción a la estructura de la aplicación.*

- *EN 50090-3-2:2004 Sistemas electrónicos en viviendas y edificios (HBES). Parte 3-2: Aspectos de aplicación. Proceso de usuario para HBES de Clase 1.*

- *EN 50090-3-3:2009 Sistemas electrónicos para viviendas y edificios (HBES). Parte 3: Aspectos de aplicación. Modelo de interoperabilidad HBES y tipos de datos HBES comunes.*

- *EN 50090-4-1:2004 Sistemas electrónicos en viviendas y edificios (HBES). Parte 4-1: Capas independientes del medio. Capa de aplicación para HBES de Clase 1.*

- *EN 50090-4-2:2004 Sistemas electrónicos para viviendas y edificios (HBES). Parte 4-2: Capas independientes del medio. Capa de transporte, capa de red y partes generales de la capa de unión de datos para HBES de Clase 1.*

- *EN 50090-4-3:2007 Sistemas Electrónicos para Viviendas y Edificios (HBES). Parte 4-3: Capas independientes del medio. Comunicaciones IP.*

- *EN 50090-5-1:2005 Sistemas electrónicos para vivienda y edificios (HBES). Parte 5-1: Medio y capas dependientes. Potencia I para clase 1 HBES.*

- *EN 50090-5-2:2004 Sistemas electrónicos para viviendas y edificios (HBES). Parte 5-2: Medio y capas dependientes del medio. Red basada en HBES de Clase 1, Par trenzado.*

- *EN 50090-5-3:2006 Sistemas electrónicos para vivienda y edificios (HBES). Parte 5-3: Medio y capas dependientes. Radiofrecuencia.*

- *EN 50090-7-1:2004 Sistemas electrónicos para viviendas y edificios (HBES). Parte 7-1: Gestión del sistema. Procedimientos de gestión.*

- *EN 50090-8:2000 Sistemas electrónicos para viviendas y edificios (HBES). Parte 8: Evaluación de la conformidad de los productos.*

- *EN 50090-9-1:2004 Sistemas electrónicos en viviendas y edificios (HBES). Parte 9-1: Requisitos de instalación. Cableado genérico para par trenzado HBES de clase 1.*

- *CLC/TR 50090-9-2:2007 Sistemas electrónicos en viviendas y edificios (HBES). Parte 9-2: Requisitos de instalación. Inspección y ensayo de la instalación HBES.*

- *EN 50491-2:2010 Requisitos generales para sistemas electrónicos para viviendas y edificios (HBES) y sistemas de automatización y control de edificios (BACS). Parte 2: Condiciones ambientales.*

- *EN 50491-3:2009 Requisitos generales para sistemas electrónicos para viviendas y edificios (HBES) y sistemas de automatización y control de edificios (BACS). Parte 3: Requisitos de seguridad eléctrica.*

- *EN 50491-5-1:2010 Requisitos generales para sistemas electrónicos para viviendas y edificios (HBES) y sistemas de automatización y*

control de edificios (BACS). Parte 5-1: Requisitos de compatibilidad electromagnética (CEM), condiciones y montaje de ensayos.

- *EN 50491-5-2:2010 Requisitos generales para sistemas electrónicos para viviendas y edificios (HBES) y sistemas de automatización y control de edificios (BACS). Parte 5-2: Requisitos de compatibilidad electromagnética (CEM) para HBES/BACS utilizados en entornos residenciales, comerciales y de industria ligera.*

- *EN 50491-5-3:2010 Requisitos generales para sistemas electrónicos para viviendas y edificios (HBES) y sistemas de automatización y control de edificios (BACS). Parte 5-3: Requisitos de compatibilidad electromagnética (CEM) para HBES/BACS utilizados en entornos industriales.*

Normas UNE-EN 50491 para Sistemas Electrónicos de Viviendas y Edificios (HBES) y Sistemas de Automatización y Control de Edificios (BACS)

Estas normativas son independientes de los protocolos de comunicación del sistema, se encargan de recoger los requisitos generales de los sistemas electrónicos destinados a viviendas y edificios (HBES). Cubren los requisitos ambientales de compatibilidad electromagnética (CEM), de seguridad de los dispositivos, seguridad eléctrica.

- *EN 50491-5-1 Requisitos generales para sistemas electrónicos para viviendas y edificios (HBES) y sistemas de automatización y control de edificios (BACS). Parte 5-1: Requisitos de compatibilidad electromagnética (CEM), condiciones y montaje de ensayos.*

- *EN 61000-4-2 Compatibilidad electromagnética (CEM). Parte 4: Técnicas de ensayo y de medida. Sección 2: Ensayos de inmunidad a las descargas electrostáticas. Norma básica de CEM. (IEC 61000-4-2).*

- *EN 61000-4-3 Compatibilidad electromagnética (CEM). Parte 4-3:*

Técnicas de ensayo y de medida. Ensayos de inmunidad a los campos electromagnéticos, radiados y de radiofrecuencia. (IEC 61000-4-3).

- *EN 61000-4-4 Compatibilidad electromagnética (CEM). Parte 4-4: Técnicas de ensayo y de medida. Ensayos de inmunidad a los transitorios eléctricos rápidos en ráfagas. (IEC 61000-4-4).*

- *EN 61000-4-5 Compatibilidad electromagnética (CEM). Parte 4:-5: Técnicas de ensayo y de medida. Ensayos de inmunidad a las ondas de choque. (IEC 61000-4-5).*

- *EN 61000-4-6 Compatibilidad electromagnética (CEM). Parte 4-6: Técnicas de ensayo y de medida. Inmunidad a las perturbaciones conducidas, inducidas por los campos de radiofrecuencia. (IEC 61000-4-6).*

- *EN 61000-4-8 Compatibilidad electromagnética (CEM). Parte 4: Técnicas de ensayo y de medida. Sección 8: Ensayo de inmunidad a los campos magnéticos a frecuencia industrial. Norma básica de CEM. (IEC 61000-4-8).*
- *EN 61000-4-11 Compatibilidad electromagnética (CEM). Parte 4-11: Técnicas de ensayo y de medida. Ensayos de inmunidad a los huecos de tensión, interrupciones breves y variaciones de tensión. (IEC 61000-4-11).*

- *EN 61000-6-1 Compatibilidad electromagnética (CEM). Parte 6-1: Normas genéricas. Inmunidad en entornos residenciales, comerciales y de industria ligera. (IEC 61000-6-1).*

- *EN 61000-6-3 Compatibilidad Electromagnética (CEM). Parte 6-3: Normas genéricas. Norma de emisión en entornos residenciales, comerciales y de industria ligera. (IEC 61000-6-3).*

Especificación EA0026 para Instalaciones de Sistemas Domóticos de Viviendas

Las especificaciones EA0026 es un documento de rango inferior a la normativa y establece los mínimos requisitos que debe cumplir una instalación domótica de clase I, fijando las diferentes prescripciones generales de instalación, evaluación, y los distintos niveles de domotización a nivel residencial.

Estas especificaciones se emplean como referencia para certificar los sistemas domóticos y surgen con el objetivo de:

1. Impulsar el desarrollo del mercado domótico.
2. Aclarar la confusión existente en el mercado respecto a lo que es un sistema domótico.
3. Poder comparar entre las diferentes ofertas del mercado.

Especificación CLC/TR 50491-6-3 para Instalaciones de Sistemas Domóticos de Viviendas

Estas especificaciones se tratan de la propuesta española, realizada por el subcomité de normalización de AENOR para elevar la EA0026 como documento de referencia europea.

Esta especificación incluye una clasificación de niveles basada en la EA0026 y una clasificación en la que se indica el nivel de ahorro energético que proporcionan una instalación de sistemas domóticos, está basada en la norma UNE-ENE152332 de eficiencia energética en edificios.

- *EN 15232 Eficiencia energética de los edificios. Métodos de cálculo de las mejoras de la eficiencia energética mediante la aplicación de sistemas integrados de gestión técnica de edificios.*

- *EN 50491-3 Requisitos generales para sistemas electrónicos para viviendas y edificios (HBES) y sistemas de automatización y control de*

edificios (BACS). Parte 3: Requisitos de seguridad eléctrica.

- *EN 50491-5-1 Requisitos generales para sistemas electrónicos para viviendas y edificios (HBES) y sistemas de automatización y control de edificios (BACS). Parte 5-1: Requisitos de compatibilidad electromagnética (CEM), condiciones y montaje de ensayos.*

- *EN 50491-5-2 Requisitos generales para sistemas electrónicos para viviendas y edificios (HBES) y sistemas de automatización y control de edificios (BACS). Parte 5-2: Requisitos de compatibilidad electromagnética (CEM) para HBES/BACS utilizados en entornos residenciales, comerciales y de industria ligera.*

- *EN 50491-5-3 Requisitos generales para sistemas electrónicos para viviendas y edificios (HBES) y sistemas de automatización y control de edificios (BACS). Parte 5-3: Requisitos de compatibilidad electromagnética (CEM) para HBES/BACS utilizados en entornos industriales.*

- *EN 50491-6-1[1)] Requisitos generales para sistemas electrónicos para viviendas y edificios (HBES) y sistemas de automatización y control de edificios (BACS). Parte 6-1: Instalaciones HBES. Instalación y planificación*

Reglamentación de instalaciones domóticas e inmóticas

Las disposiciones legales que son de obligado cumplimiento para instalaciones domóticas e inmóticas, quedan recogidas por el Reglamento Electrónico de Baja Tensiones (REBT) y en el anexo Hogar Digital del Reglamento de Infraestructura Comunes de Telecomunicaciones (ICT).

Reglamento electrónico de baja tensión (REBT)

El REBT establece las condiciones técnicas y garantías de seguridad que debe reunir una instalación eléctrica de baja tensión para fines siguientes:

- Preservar la seguridad de las personas y los bienes.
- Asegurar el normal funcionamiento de dichas instalaciones y prevenir las perturbaciones en otras instalaciones y servicios.
- Contribuir a la fiabilidad técnica y a la eficiencia económica de las instalaciones.

El REBT incluye 51 instrucciones complementarias que hacen referencias a la normativa UNE, las cuales son aplicables a los sistemas de automatización y control.

ITC-BT 51

La Instrucción Técnica 51 del REBT establece los requisitos mínimos de la instalación de los sistemas domóticos y comprende a las instalaciones de sistemas no independientes que realizan una función de control automático.

Esta instrucción establece los requisitos mínimos de los que debe constar una instalación que incluya sistemas domóticos y comprende a las instalaciones de sistemas independientes que realizan funciones de control automático.

Reglamento ICT: Anexo Hogar Digital

El Anexo V del Reglamento de ICT (R.D. 345/2011) se puede aplicar de forma voluntaria, y tiene como objetivo de poder facilitar la incorporación de funciones de control digital a las viviendas, apoyándose en soluciones que se incluyen en el reglamento.

7.2. MANUAL DE INSTALACIÓN

Para un correcto funcionamiento de la instalación es necesario que se realice una correcta instalación, para ello se han establecido una serie de parámetros que se han de seguir a la hora de realizar el montaje dentro de la vivienda:

1. Identificar la instalación con datos del emplazamiento y características de la instalación previa.

2. Planos de la instalación
- Planta general de la vivienda
- Plano eléctrico con los trazados de los sistemas de conducción de cables, situación de cajas de derivación y cuadro general de distribución.
- Trazado y situación de dispositivos
- Esquema unifilar de la instalación identificando los circuitos de control y los de la red eléctrica asociada.

3. Las modificaciones en el lugar de instalación, deberán ser lo más reducidas posible, evitando realizar alteraciones de gran magnitud.

4. Con el fin de reducir las modificaciones, se empleará en la medida de lo posible la infraestructura previa de la instalación empleando cajas de derivación y canalizaciones ya instaladas.

5. Si la instalación no consta de ninguna preinstalación, se instalarán las canalizaciones necesarias, así como cajas de registro.

6. Instalación de dispositivos siguiendo las especificaciones técnicas y del fabricante en la localización detallada por los planos.

7. Programación de parámetros que se han establecido de acuerdo con las especificaciones de funcionamiento para cada dispositivo.

8. Programación de niveles de aviso de alarma.

9. Programación y ajuste de parámetros para dispositivos específicos.

10. Puesta en marcha y verificación del correcto funcionamiento con indicadores de etapa para asegurar que cada una de las partes, componentes, bloques de control y cableado están de acuerdo con las normas de instalación y la normativa vigente.

11. Instalación del sistema de control remoto e interfaz móvil, y descripción de cada una de sus funciones.

7.2.1. CONDICIONES PARTICULARES DE INSTALACIÓN

Además de las indicaciones generales establecidas en el apartado anterior, se establecen los siguientes requisitos particulares.

- **Requisitos para instalaciones que utilicen señales que transmiten por instalaciones de baja tensión**

Los módulos que introducen señales en la instalación de baja tensión, señales de entre 3kHZ hasta 148KHZ deben cumplir con la normativa UNE-EN 50.065-1 en lo relativo a compatibilidad electromagnética.

- **Requisitos para sistemas que utilizan para transmitir cables específicos**

Si el circuito transmisor de señal transcurre por la misma canalización que otro de baja tensión, el nivel de aislamiento de los cables de este circuito deben ser equivalentes a la de dos circuitos de baja tensión, ya sea en uno o varios aislamientos

- **Requisito para sistemas que usan señales radiales**

Los sistemas emisores de señales de radiofrecuencia, o señales de telecomunicaciones, deberán cumplir las especificaciones impuestas en la normativa vigente dentro del “Cuadro Nacional de Atribución de Frecuencias de Ordenación de las Telecomunicaciones”**.**

7.3. RECOMENDACIONES PARA LA INSTALACIÓN DE SISTEMAS DOMÓTICOS

En los proyectos de obras nuevas en los que no se requiera de la instalación de sistemas domóticos, se recomienda, con el fin de evitar costes posteriores debido a obras para la instalación de este tipo de sistemas, realizar preinstalaciones que faciliten la adecuación del sistema domótico a las necesidades del usuario para favorecer las futuras demandas en este campo.

Los elementos y características de una preinstalación son los siguientes:

- Introducir canalizaciones desde el punto de acceso de usuario a las instalaciones de telecomunicación hasta la caja de distribución.

- Se podrá instalar un cuadro general de distribución previsto para los dispositivos generales de mando.

- Se recomienda la instalación de una caja de 24 módulos DIN por cada 100 m^2 o bien por cada planta.

- Se recomienda instalar una caja de registros junto a cada caja de empalme y derivación de la instalación eléctrica o bien, instalar una caja de empalme o derivación un 50% más amplia, con el fin de ubicar los dispositivos del sistema de control domótico.

- Se considera instalar una canalización independiente entre las cajas de registro, para la instalación domótica, o en el caso de que se empleen cajas de empalme, se aumentará la sección de la canalización como mínimo 200mm.

- Se recomienda la instalación de cajas para situar los componentes de la instalación como pueden ser sensores o actuadores junto con sus correspondientes canalizaciones hasta la caja de registro.

8.0. PRESUPUESTO

Componentes	Unidades	Precio por unidad (€/ud)	Total (€)
INTERFACES			
Arduino Mega 2560	1	41,75	41,75
Módulo Bluetooth	1	8	8
Total Interfaces	2	49,75	
SEGURIDAD			
Sensor de movimiento PIR HC-SR501	1	3,99	3,99
Keypad	1	2,4	2,4
Módulo Buzzer	1	19,13	19,13
Reguladores	2	11,24	22,48
Módulo MQ-2	1	4,95	4,95
Total Seguridad	6	52,95	
CONFORT			
Modulo de relés	2	13,46	26,92
Motor de persianas	4	32	128
Actuador persianas	4	40	160
Sensor THD11	1	4,85	4,85
Módulo sensor LDR	4	4,95	19,8
Tiras led	2m	6,95	13,9
Total Confort	16	353,47	
INSTALACIÓN			
Cableado	400m	0,85	340
Canalizaciones	400m	0,18	72
Cuadro general	1	6,9	6,9
Cajas de registro	4	2,2	8,8
Configuración y puesta a punto	5h	22	110
Proyecto y planos	40h	30	1200
Instalador especializado	60h	22	1320
TOTAL			**3513,87**

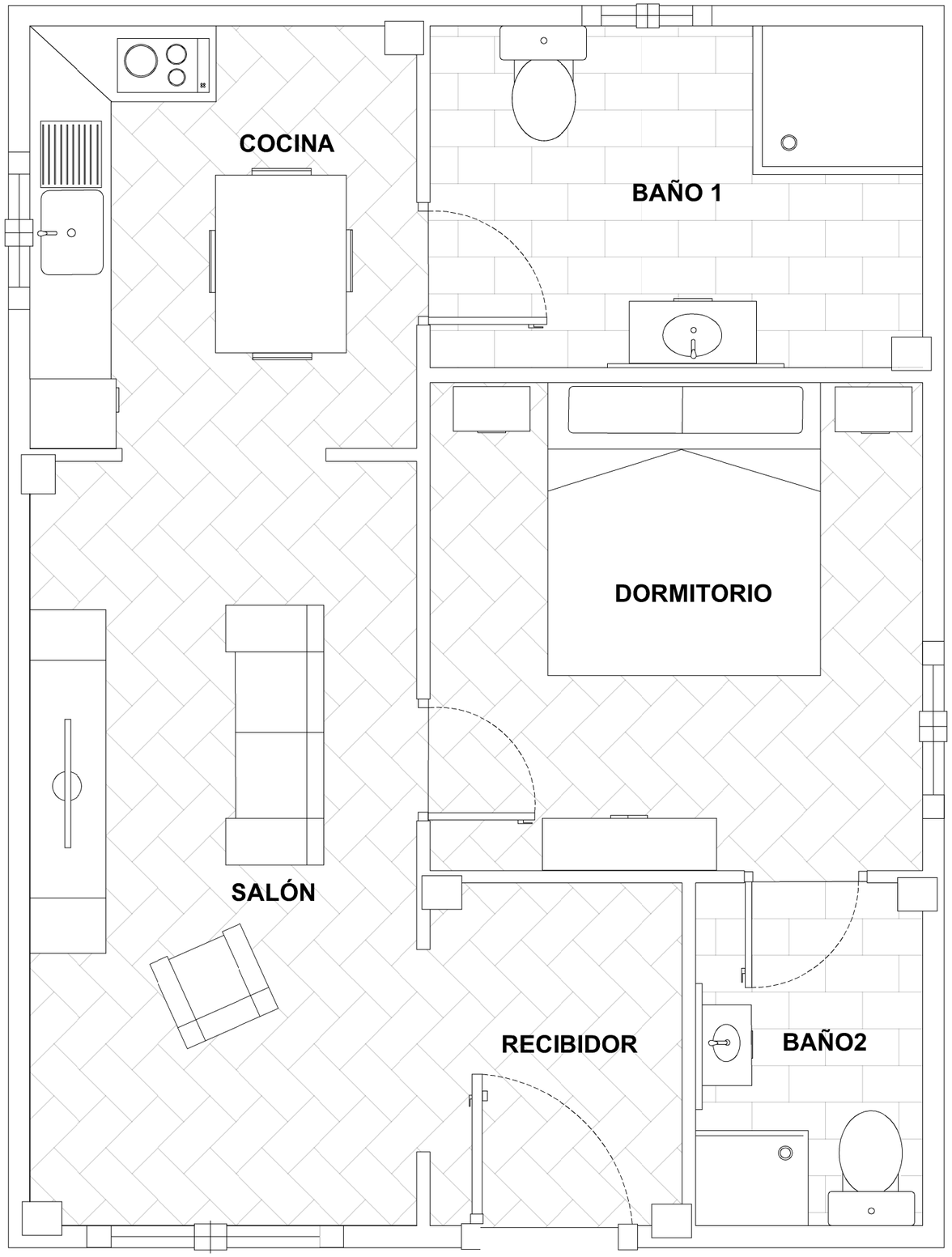
Tabla 24: Presupuesto instalación real domótica

Componentes	Unidades	Precio por unidad (€/ud)	Total (€)
ARDUINO			
Arduino Mega2560	1	41,75	41,75
Arduino Uno	1	6,99	6,99
MÓDULOS			
Bluetooth	1	6,75	6,75
Relé	1	13,46	13,46
Buzzer	1	1,05	1,05
TDH11	1	4,85	4,85
MQ-2	1	4,95	4,95
Sensor PIR	1	3,99	3,99
Keypad	1	2,89	2,89
Módulo sensor LDR	1	4,95	4.95
Modulo alimentación	1	2,39	2,39
LCD 16x2	1	7,99	7,99
ACTUADORES			
Motor DC12V 14rpm	1	3,49	3,49
Tira led	1	6,95	6,95
Leds	8	0,15	1,2
Ventiladores	2	10	20
COMPONENTES			
Pulsadores	6	0,3	1,8
Interruptores	5	0,1	0,5
Cables Jumper M-H	40	0,15	6
Cables Jumper M-M	40	0,15	6
Pin macho	40	0,06	2,4
TIP120	5	0,5	2,5
Protoboard	2	2,79	5,58
L293D	1	1,46	1,46
Cable	4m	0,13	0,52
Pilas 1,5v	2	1,1	2,2
Portapilas 3v	1	0,4	0,4
Portapilas 9V	2	0,3	0,6
Pila 9V	2	1,49	2,98
Caja de registro	1	4,95	4,95
Resistencias	15	0,03	0,45
MAQUETA			
Cartón pluma			18
Adhesivo			2,5
TOTAL			**192.50**

Tabla 25: Presupuesto de simulación

En el siguiente enlace se puede ver un video realizado sobre la simulación de cada uno de los bloques de control diseñados.

https://www.youtube.com/watch?v=5tK0gD92I2g&feature=youtu.be



SALÓN	22.5 m2	COCINA	15 m2
DORMITORIO	16.88 m2	BAÑO 1	9.37 m2
RECIBIDOR	6.2 m2	BAÑO 2	5.62 m2

UNIVERSIDAD POLITECNICA DE VALENCIA

TITULO DEL PROYECTO:

CONTROL DOMÓTICO MEDIANTE INTERFAZ MÓVIL

PLANO:

PLANTA VIVIENDA

PROMOTOR

Universitat Politècnica de València

FECHA: **Junio 2016**

Nº PLANO: **01**

ESCALA: **1:40**

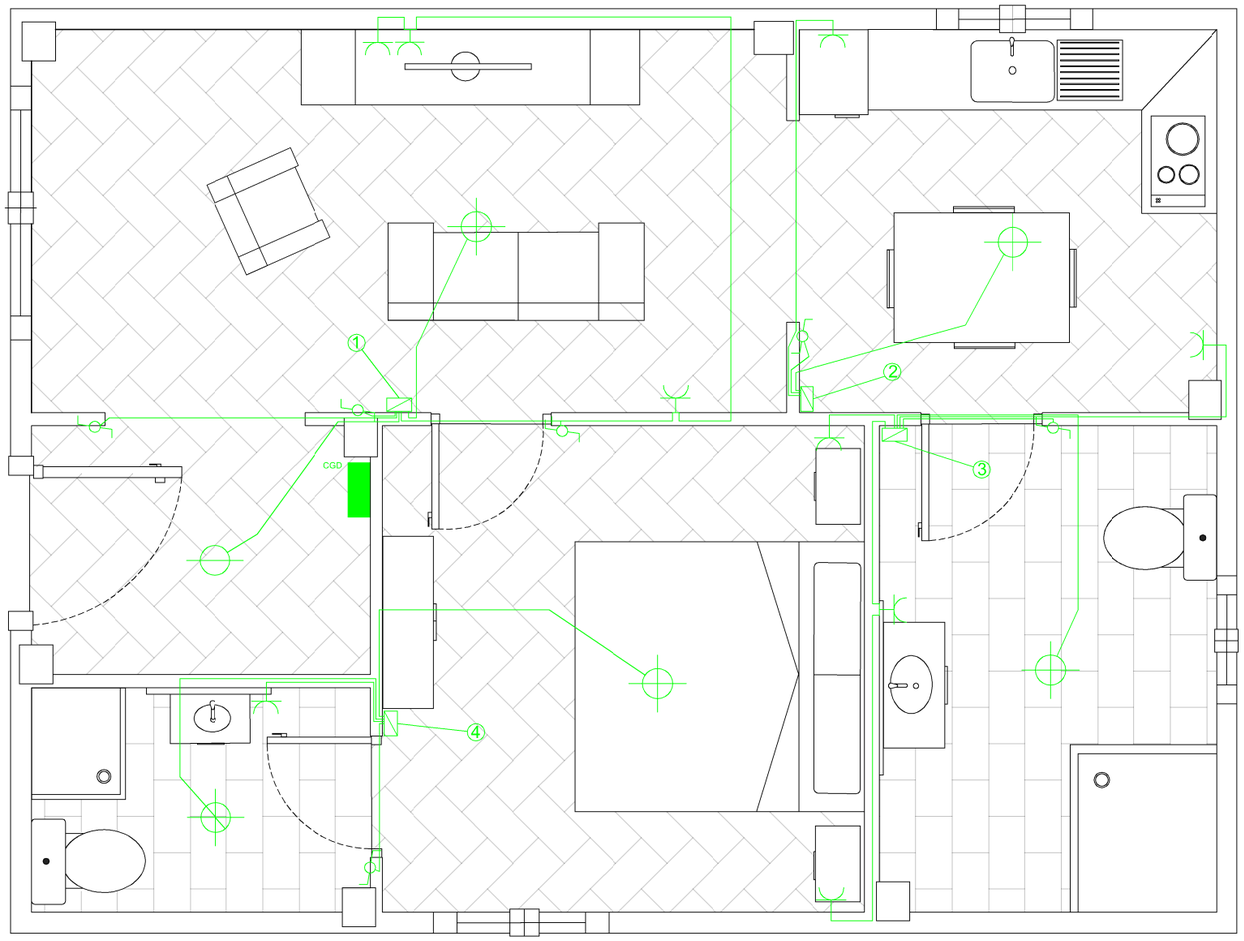
FIRMA:

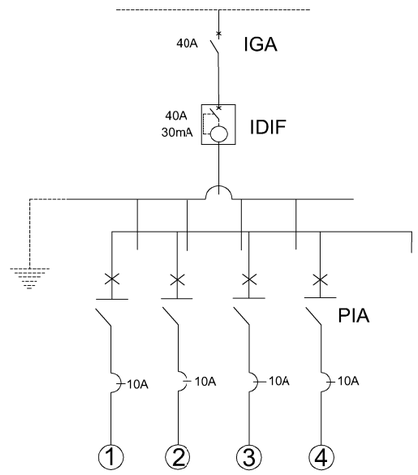
AUTOR:

Lucas Martínez Hernández

Grado en ing. Electrónica industrial y automática

Escuela Técnica Superior de Ingeniería del Diseño





ESQUEMA UNIFILAR

SIMBOLOGÍA			
	CONMUTADOR	PIA	PEQUEÑO INTERRUPTOR AUTOMÁTICO
	INTERRUPTOR	IDIF	INTERRUPTOR DIFERENCIAL
	ENCHUFE	IGA	INTERRUPTOR GENERAL AUTOMÁTICO
	CAJA DE DERIVACIÓN		INTERRUPTOR MAGNETOTÉRMICO
CGD	CUADRO GENERAL DE DISTRIBUCIÓN		PUNTO CENTRO

UNIVERSIDAD POLITECNICA DE VALENCIA

TITULO DEL PROYECTO:

CONTROL DOMÓTICO MEDIANTE INTERFAZ MÓVIL

PLANO:

INSTALACIÓN ELÉCTRICA

PROMOTOR

Universitat Politècnica de València

FECHA: **Junio 2016**

Nº PLANO: **02**

ESCALA: **1:50**

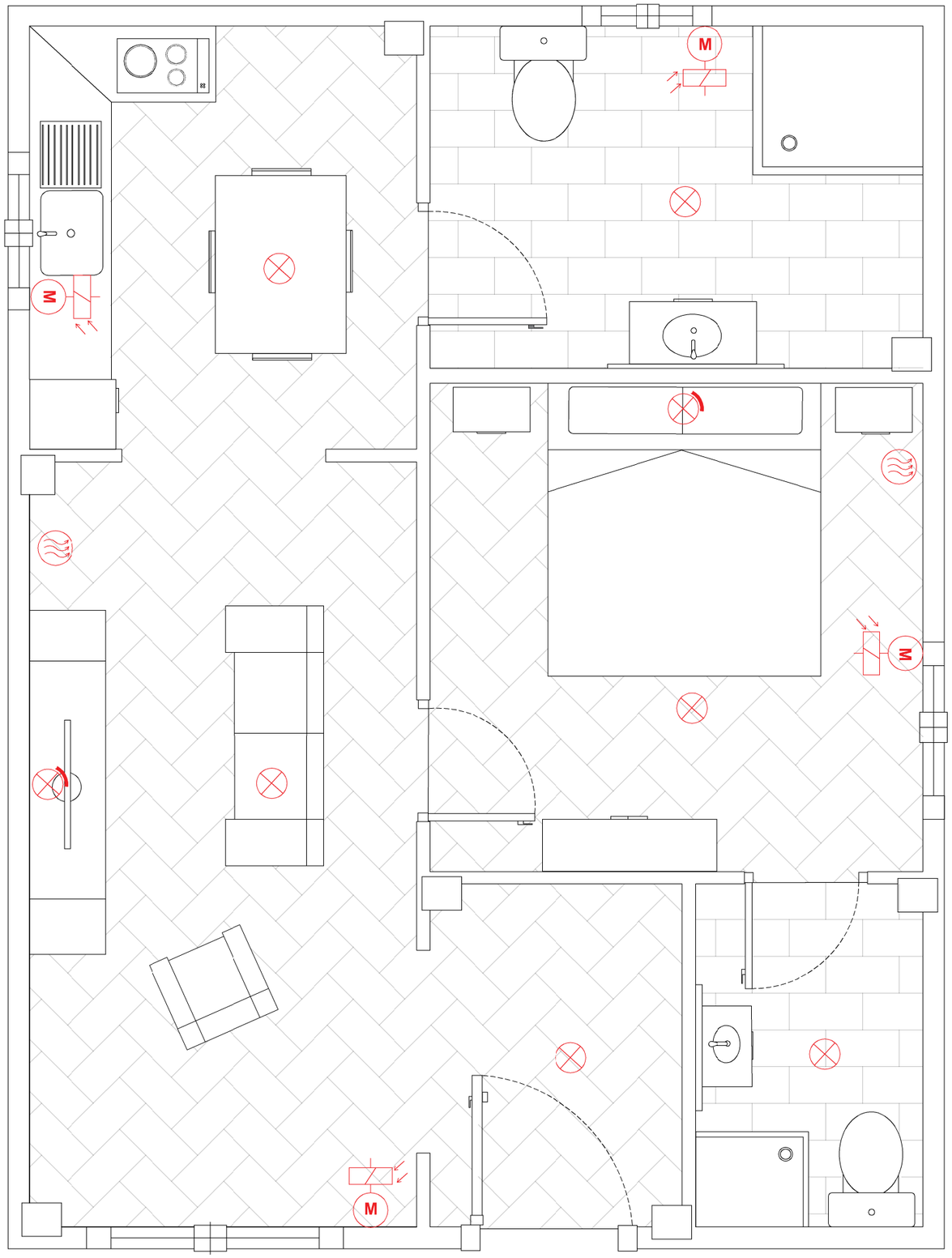
FIRMA:

AUTOR:

Lucas Martínez Hernández

Grado en ing. Electrónica industrial y automática

Escuela Técnica Superior de Ingeniería del Diseño



SIMBOLOGÍA	
	CONTROL DIMMER
	CONTROL DE ILUMINACIÓN
	VENTILACIÓN
M	MOTOR
	ACTUADOR DE PERSIANAS

UNIVERSIDAD POLITECNICA DE VALENCIA

TITULO DEL PROYECTO:

CONTROL DOMÓTICO MEDIANTE INTERFAZ MÓVIL

PLANO:

SITUACIÓN DE ACTUADORES

PROMOTOR

Universitat Politècnica de València

FECHA: Junio 2016

Nº PLANO: 03

ESCALA: 1:40

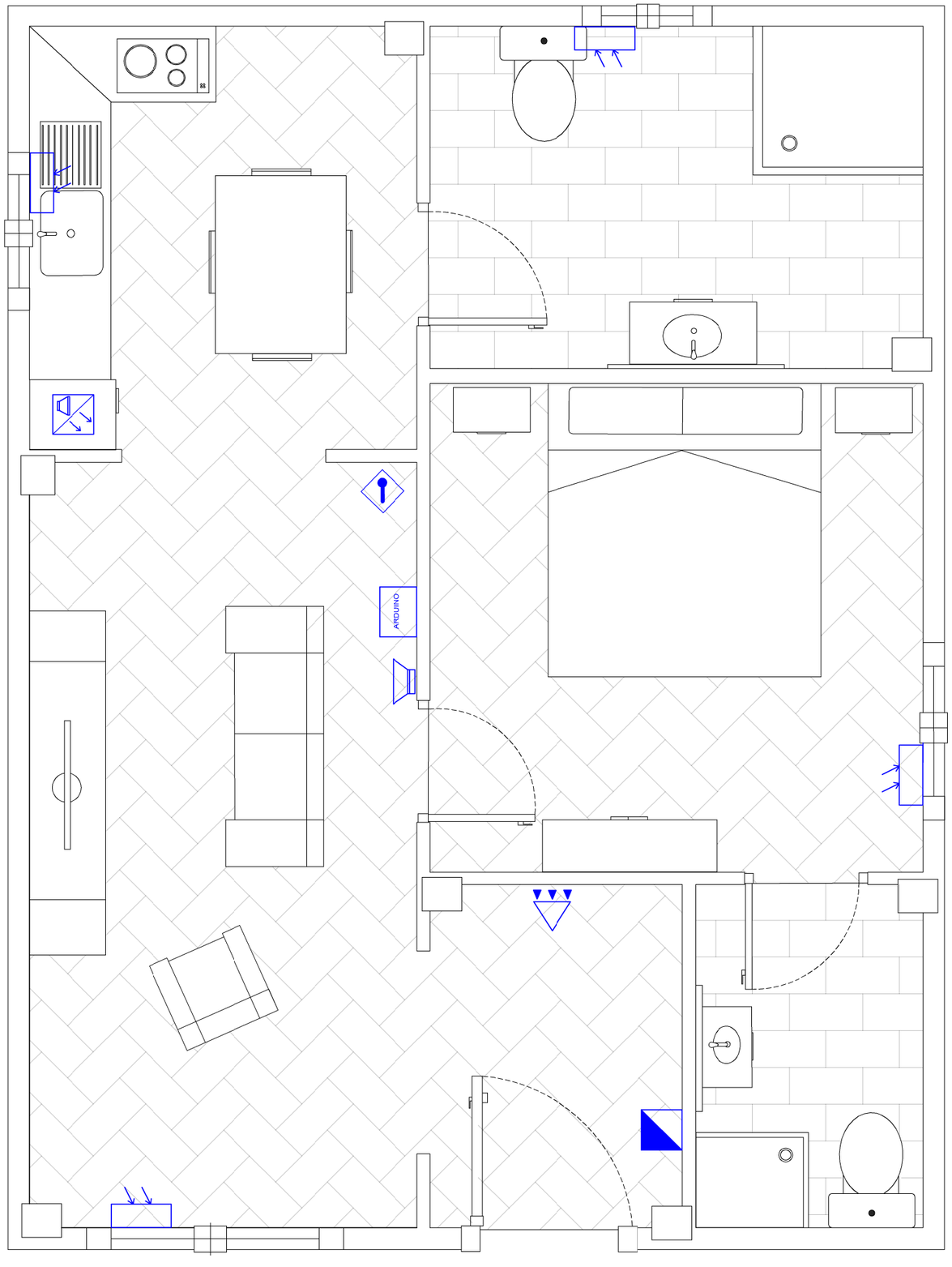
FIRMA:

AUTOR:

Lucas Martínez Hernández

Grado en ing. Eectrónica industrial y automática

Escuela Técnica Superior de Ingeniería del Diseño



SIMBOLOGÍA	
ARDUINO	CONTROL CENTRAL
	SENSOR DE TEMPERATURA
	SENSOR DE LUZ
	DETECTOR DE GASES
	ALARMA
	TECLADO CONTROL DE ACCESO
	DETECTOR DE MOVIMIENTOS

UNIVERSIDAD POLITECNICA DE VALENCIA

TITULO DEL PROYECTO:

CONTROL DOMÓTICO MEDIANTE INTERFAZ MÓVIL

PLANO:

SITUACIÓN DE SENSORES

PROMOTOR

Universitat Politècnica de València

FECHA: Junio 2016

Nº PLANO: 04

ESCALA: 1:40

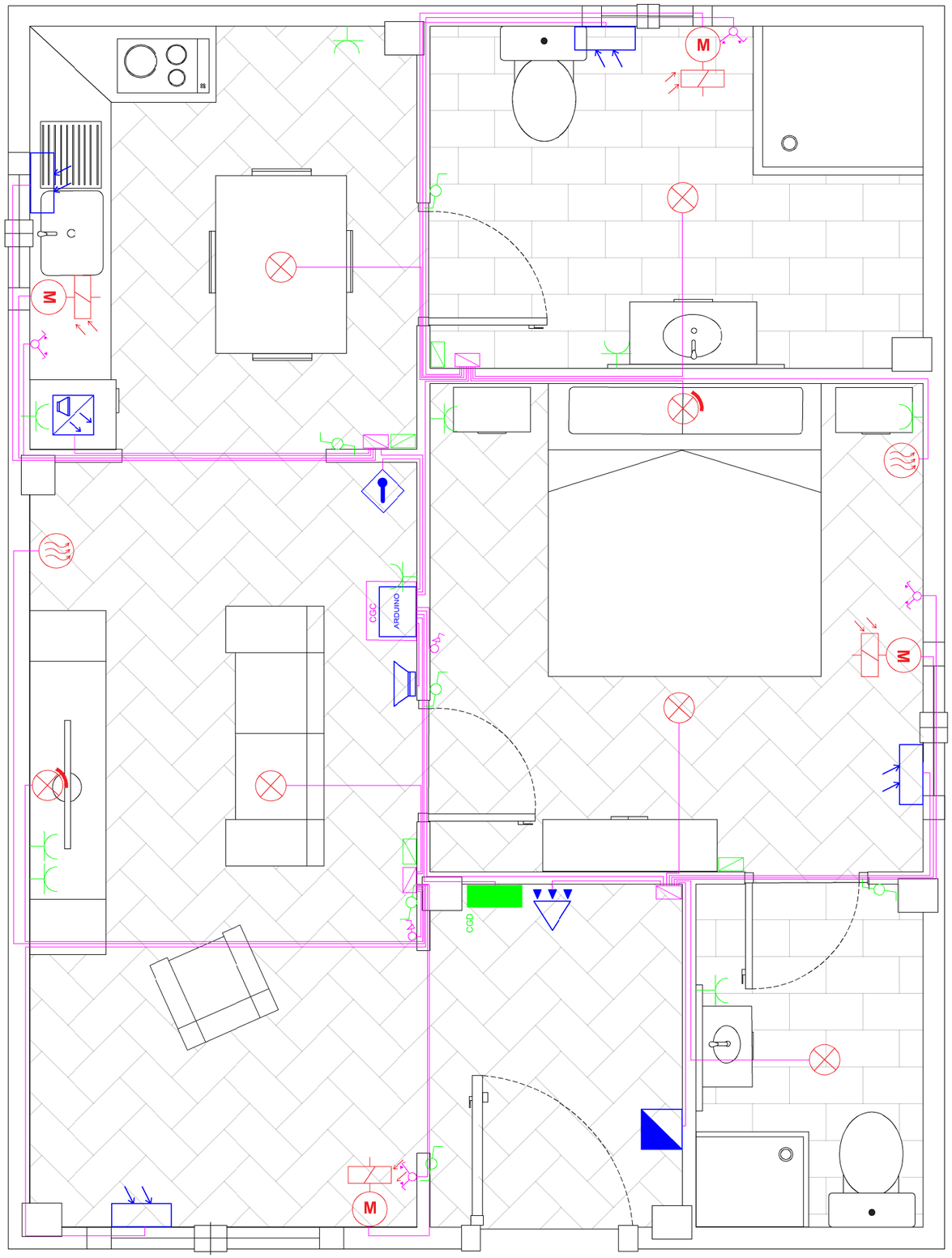
FIRMA:

AUTOR:

Lucas Martínez Hernández

Grado en ing. Eectrónica industrial y automática

Escuela Técnica Superior de Ingeniería del Diseño



SIMBOLOGÍA

	CONEXIONES DOMÓTICA
	ACTUADORES
	SENSORES
	INSTALACIÓN ELÉCTRICA

SIMBOLOGÍA

CGC	CUADRO GENERAL DE CONTROL
	REGULADOR
	CAJA DE REGISTRO
	INTERRUPTOR DE PERSIANAS

UNIVERSIDAD POLITECNICA DE VALENCIA

TITULO DEL PROYECTO:

CONTROL DOMÓTICO MEDIANTE INTERFAZ MÓVIL

PLANO:

INSTALACIÓN DOMÓTICA

PROMOTOR

Universitat Politècnica de València

FECHA: Junio 2016

Nº PLANO: 05

ESCALA: 1:40

FIRMA:

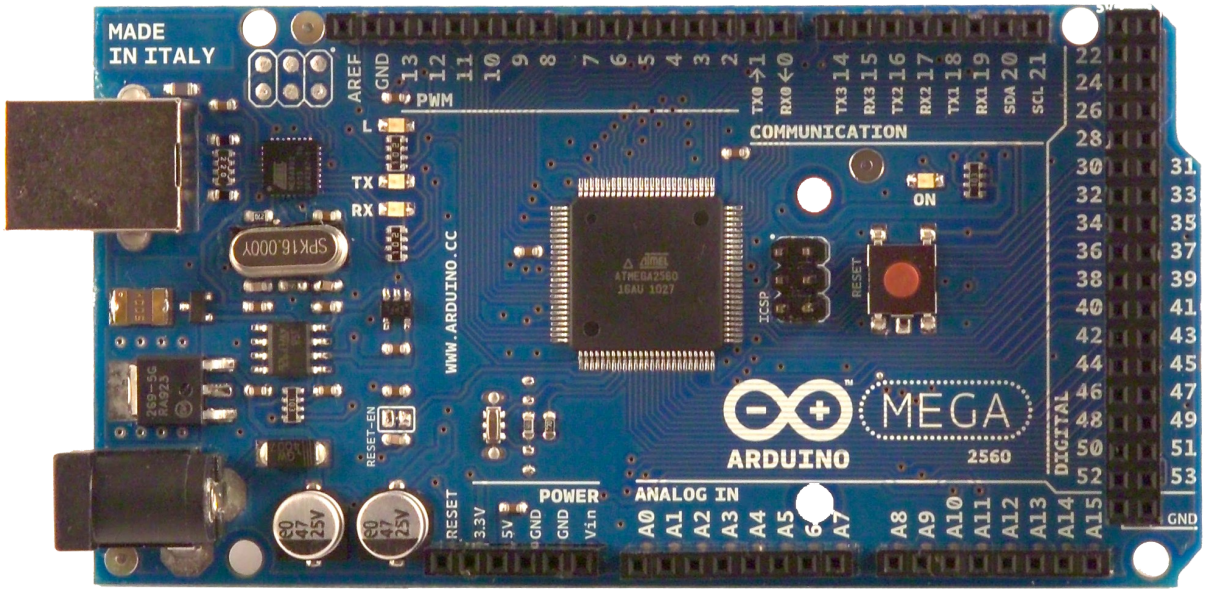
AUTOR:

Lucas Martínez Hernández

Grado en ing. Eectrónica industrial y automática

Escuela Técnica Superior de Ingeniería del Diseño

Arduino MEGA 2560



Product Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

Index

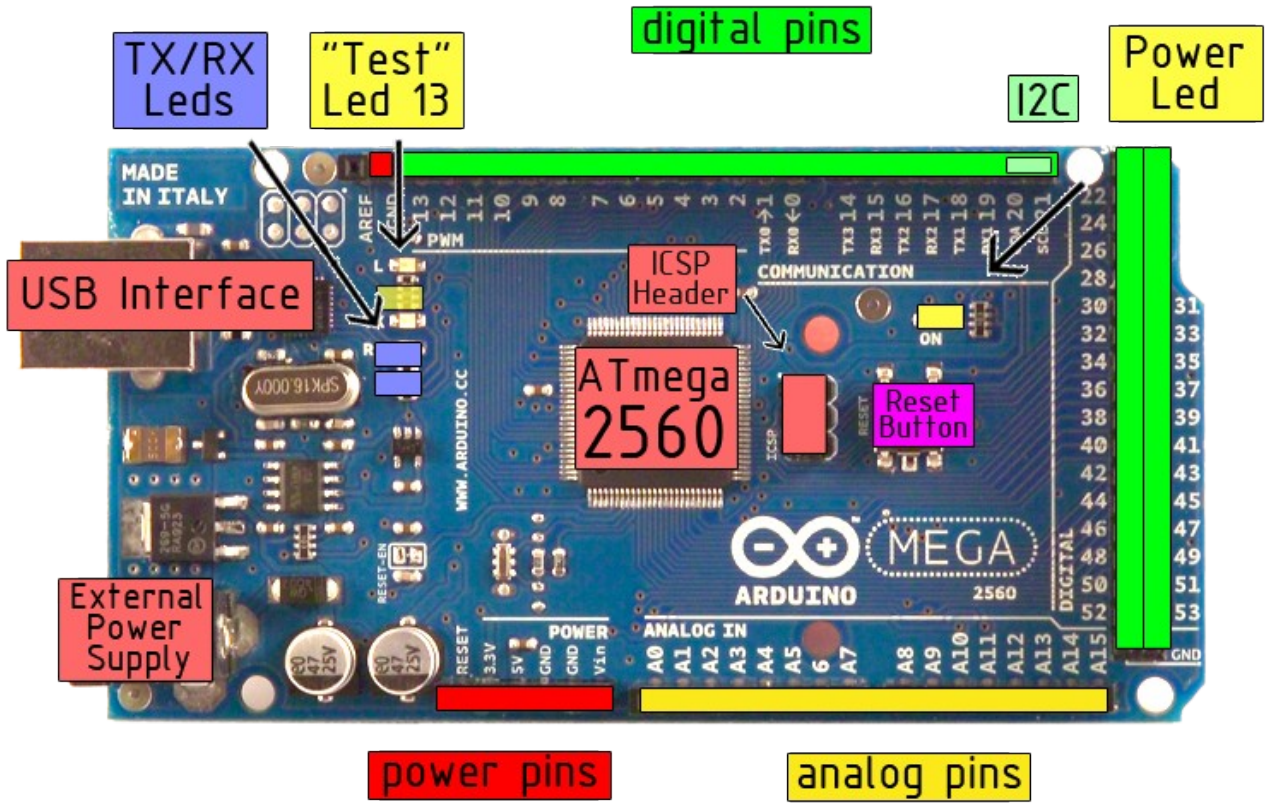
Technical Specification

EAGLE files: arduino-mega2560-reference-design.zip Schematic: arduino-mega2560-schematic.pdf

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

the board



Power

The Arduino Mega2560 can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **I^2C: 20 (SDA) and 21 (SCL).** Support I^2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I^2C pins on the Duemilanove.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega8U2 on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Mega's digital pins.

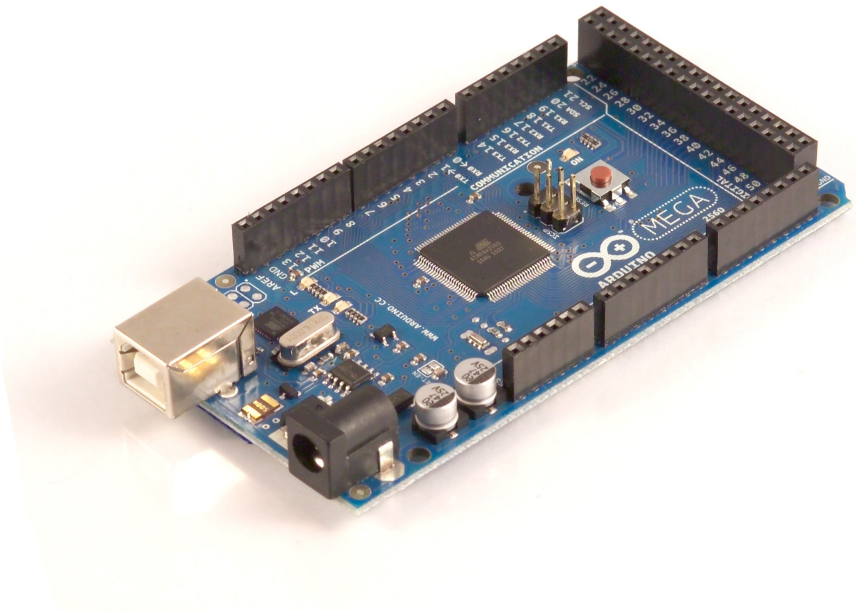
The ATmega2560 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website for details. To use the SPI communication, please see the ATmega2560 datasheet.

Programming

The Arduino Mega2560 can be programmed with the Arduino software (download). For details, see the reference and tutorials.

The Atmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.



Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Mega contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Mega has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega is designed to be compatible with most shields designed for the Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega and Duemilanove / Diecimila. **Please note that I^2C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).**

How to use Arduino

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).

Arduino is a cross-platoform program. You'll have to follow different instructions for your personal OS. Check on the Arduino site for the latest instructions. *http://arduino.cc/en/Guide/HomePage*

Linux Install

Windows Install

Mac Install

Once you have downloaded/unzipped the arduino IDE, you can Plug the Arduino to your PC via USB cable.

Blink led

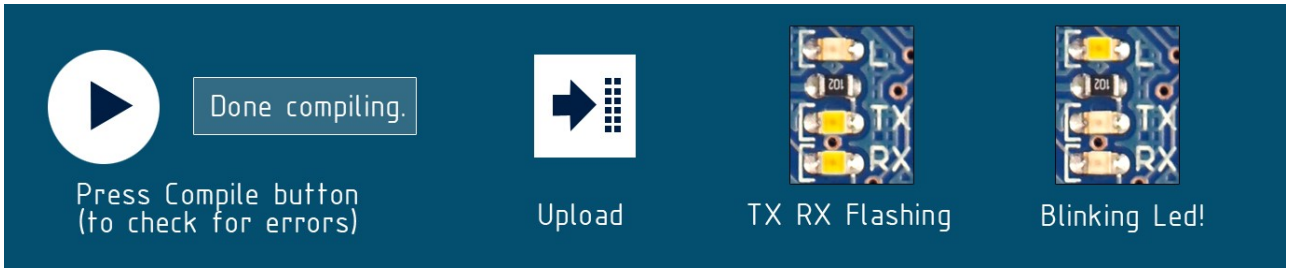
Now you're actually ready to "burn" your first program on the arduino board. To select "blink led", the physical translation of the well known programming "hello world", select

File>Sketchbook>
Arduino-0017>Examples>
Digital>Blink

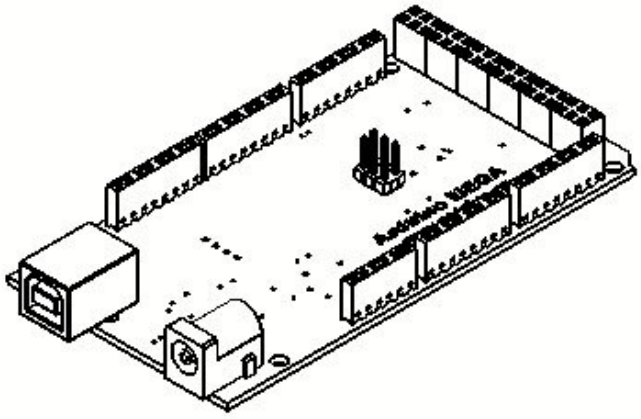
Once you have your skecth you'll see something very close to the screenshot on the right.

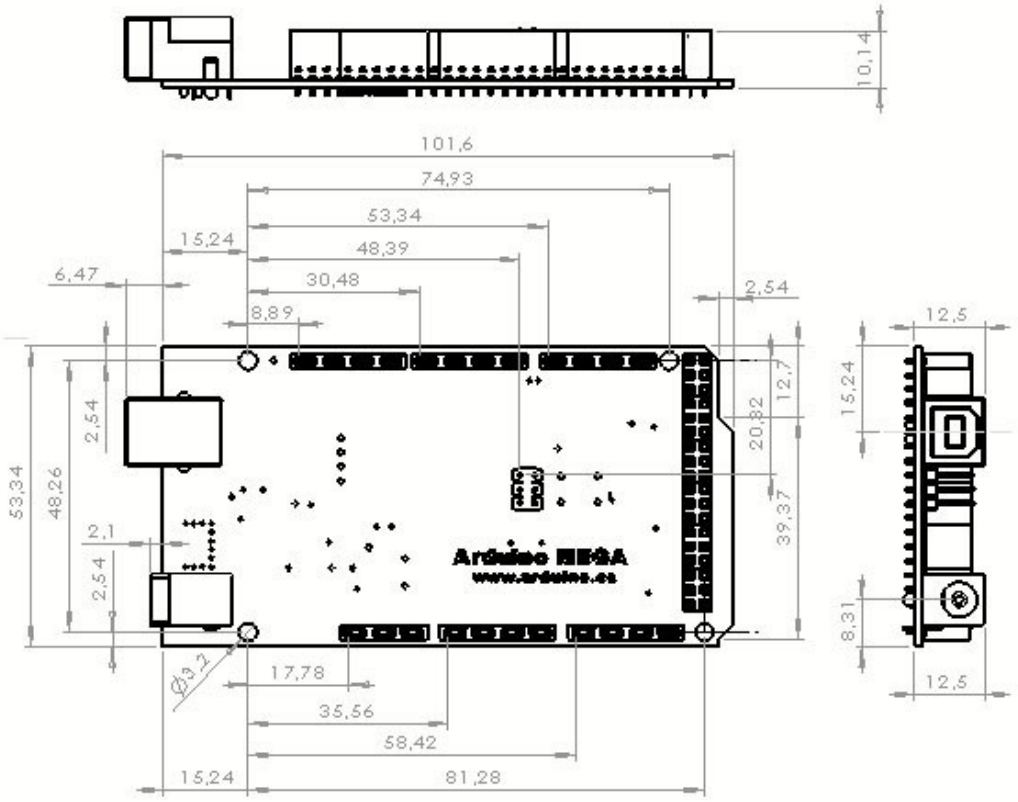
In **Tools>Board** select MEGA

Now you have to go to
Tools>SerialPort
and select the right serial port, the one arduino is attached to.



Dimensioned Drawing





Terms & Conditions

1. Warranties

1.1 The producer warrants that its products will conform to the Specifications. This warranty lasts for one (1) years from the date of the sale. The producer shall not be liable for any defects that are caused by neglect, misuse or mistreatment by the Customer, including improper installation or testing, or for any products that have been altered or modified in any way by a Customer. Moreover, The producer shall not be liable for any defects that result from Customer's design, specifications or instructions for such products. Testing and other quality control techniques are used to the extent the producer deems necessary.

1.2 If any products fail to conform to the warranty set forth above, the producer's sole liability shall be to replace such products. The producer's liability shall be limited to products that are determined by the producer not to conform to such warranty. If the producer elects to replace such products, the producer shall have a reasonable time to replacements. Replaced products shall be warranted for a new full warranty period.

1.3 EXCEPT AS SET FORTH ABOVE, PRODUCTS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS." THE PRODUCER DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

1.4 Customer agrees that prior to using any systems that include the producer products, Customer will test such systems and the functionality of the products as used in such systems. The producer may provide technical, applications or design advice, quality characterization, reliability data or other services. Customer acknowledges and agrees that providing these services shall not expand or otherwise alter the producer's warranties, as set forth above, and no additional obligations or liabilities shall arise from the producer providing such services.

1.5 The Arduino™ products are not authorized for use in safety-critical applications where a failure of the product would reasonably be expected to cause severe personal injury or death. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Arduino™ products are neither designed nor intended for use in military or aerospace applications or environments and for automotive applications or environment. Customer acknowledges and agrees that any such use of Arduino™ products which is solely at the Customer's risk, and that Customer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

1.6 Customer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products and any use of Arduino™ products in Customer's applications, notwithstanding any applications-related information or support that may be provided by the producer.

2. Indemnification

The Customer acknowledges and agrees to defend, indemnify and hold harmless the producer from and against any and all third-party losses, damages, liabilities and expenses it incurs to the extent directly caused by: (i) an actual breach by a Customer of the representation and warranties made under this terms and conditions or (ii) the gross negligence or willful misconduct by the Customer.

3. Consequential Damages Waiver

In no event the producer shall be liable to the Customer or any third parties for any special, collateral, indirect, punitive, incidental, consequential or exemplary damages in connection with or arising out of the products provided hereunder, regardless of whether the producer has been advised of the possibility of such damages. This section will survive the termination of the warranty period.

4. Changes to specifications

The producer may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." The producer reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

Enviromental Policies

The producer of Arduino™ has joined the Impatto Zero® policy of LifeGate.it. For each Arduino board produced is created / looked after half squared Km of Costa Rica's forest's.

Ficha Técnica

GESCABLE, S.L.,
C/Afrodita, 2
Pol.Ind. R-2 Meco
28880 Meco - MADRID
Telf.: 91 830 70 84 / 902 230 234
Fax.: 91 830 70 89

Referencia:

FTP CAT 5E ANTI-ROEDOR

Descripción y Uso:

Cable de uso exterior , avanzada tecnología para trasmitir datos a alta velocidad. Proporcionan unas excelentes características que superan los requerimientos de la Cat 5E, obteniendo unos valores de rendimiento muy superiores a los cables existentes en el mercado para esta categoría.

CARACTERÍSTICAS TÉCNICAS

Descripción Constructiva

Conductor	24 AWG (0,51mm) Cobre recocido sólido
Aislamiento	Polioefina
Pareado	4 pares de conductores
Pantalla	Cinta de aluminio poliéster
Conductor de Drenaje	24 AWG (0,51mm) cobre estañado solido
Cubierta interior	PVC Gris IEC 60332-1
Armado	Trenza de hilos de acero 32 x 0,20mm
Cubierta exterior	PVC Color Negro

Aplicaciones

- 4/16 Mbps Token Ring
- 100 Mbps TP-PMDD
- 10 BASE-T (IEE 802.3)
- 100 BASE-VG-AnyLAN
- 100 BASE- T (IEE 802.3)
- 1000 BASE-T (Gigabit Ethernet)
- 155/622 Mbps ATM

Valores Eléctricos Constructivos

Resistencia en corriente continua (máx) OHMS/100M (328 ft) @ 20ºC		8,90
Capacidad mutua (máx) nF/100m (328 ft) @ 1kHz		4,59
Velocidad nominal de propagación (NVP) % Velocidad de la luz		70
Impedancia característica (Ohms) Frecuencia	772 MHz	87-117
	10-200 MHz	85-115
Retardo de propagación (máx)	ns @ 10 MHz:	518
Retardo diferencial (máx)	ns/100 m:	45
Diámetro Exterior (mm)		7,50
Espesor cubierta		0,70
Radio mínimo curvature	4 x Diámetro Exterior = 27,48 mm	
Temperatura máxima de servicio		70ºC

Información adicional:

Las normas que cumplen el material utilizado para la fabricación de la cubierta son:
UNE 50265-2-1 No propagador de la llama.
UNE 50265-2-2 No propagador de la llama.
UNE 50266-2 No propagador del incendio.
Otras normas de fabricación:
ANSI7TIA7EIA 568-5 (Categoría 5E)
ISO/IEC 11801
EN 50173

Código Colores

Pares	Combinación de colores	
1	blanco azul	azul
2	blanco naranja	naranja
3	blanco verde	verde
4	blanco marrón	marrón

Observaciones:

La presentación estándar será en bobinas de 500 m y 1000 m.

Certificado CE Comunidad Europea

Arduino PID - Guía de uso de la librería

Traducción del trabajo de Brett Beauregard:
http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

Licencia:

Moyano Jonathan Ezequiel [jonathan215.mza@gmail.com]

PID para principiantes, primer acercamiento:

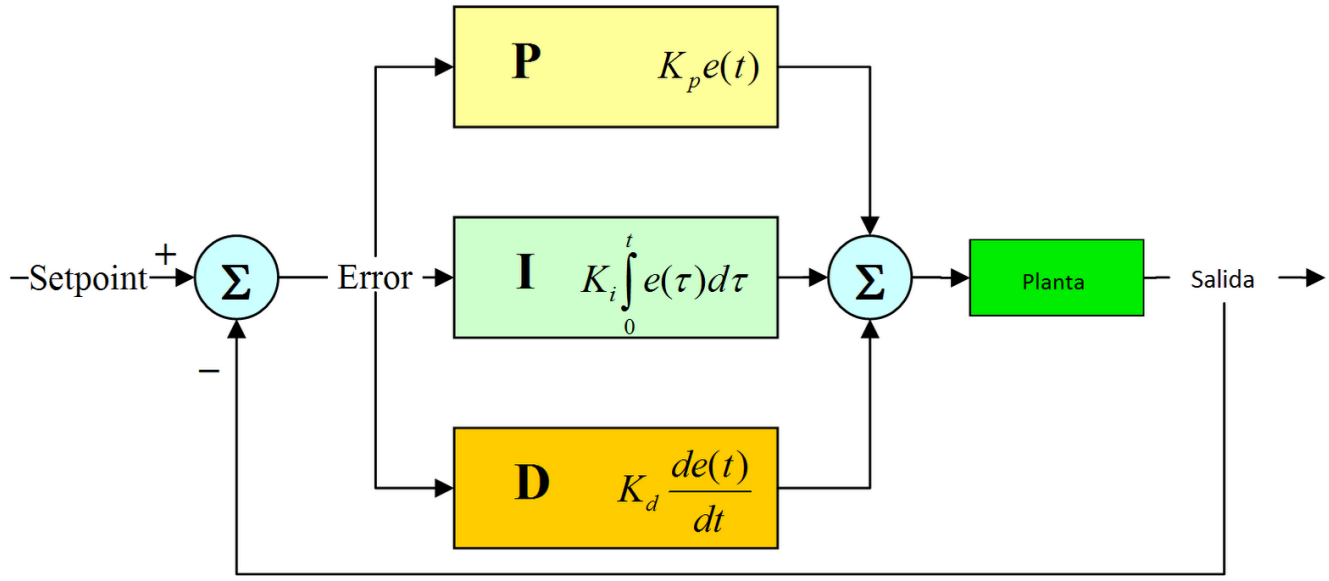
En esta introducción, veremos los parámetros básicos a tener en cuenta sobre el control propocional, integral , derivativo (PID); el objetivo de este tutorial no es introducirnos en los análisis teóricos del PID, sinó ver su aplicación en un sistema real, utilizando un microcontrolador programado en un lenguaje de alto nivel, como puede ser C.

La ecuación del PID:

De la documentación existente sobre sistemas de control, podemos destacar la siguiente ecuación.

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(t)dt + K_p T_d \frac{de(t)}{dt}$$

Para tener una idea más clara, recurrimos al siguiente diagrama



De la ecuación, podemos hacer las siguientes afirmaciones:

- **e(t)** es el error de la señal.
- **u(t)** salida del controlador y entrada de control al proceso.
- **Kp** es la ganancia proporcional.
- **Ti** es la constante de tiempo integral.
- **Td** es la constante de tiempo derivativa.

Del diagrama de flujo determinamos lo siguiente:

- El primer bloque de control (***proporcional***) consiste en el producto entre la señal de error y la constante proporcional, quedando un error en estado estacionario casi nulo.
- El segundo bloque de control (***integral***) tiene como propósito disminuir y eliminar el error en estado estacionario, provocado por el modo proporcional. El control integral actúa cuando hay una desviación entre la variable y el punto de consigna, integrando esta desviación en el tiempo y sumándola a la acción proporcional.
- El tercer bloque de control (***Derivativo***) considera la tendencia del error y permite una repercusión rápida de la variable después de presentarse una perturbación en el proceso.

Explicado lo anterior, tenemos el siguiente código:

```
/* Variables utilizadas en el controlador PID. */
unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastErr;
double kp, ki, kd;

void Compute()
{
   /* Cuanto tiempo pasó desde el último cálculo. */
   unsigned long now = millis();
   double timeChange = (double)(now - lastTime);

   /* Calculamos todas las variables de error. */
   double error = Setpoint - Input;
   errSum += (error * timeChange);
   double dErr = (error - lastErr) / timeChange;
/* Calculamos la función de salida del PID. */
```

```
   Output = kp * error + ki * errSum + kd * dErr;

   /* Guardamos el valor de algunas variables para el próximo ciclo de cálculo. */
   lastErr = error;
   lastTime = now;
}

/* Establecemos los valores de las constantes para la sintonización. */
void SetTunings(double Kp, double Ki, double Kd)
{
   kp = Kp;
   ki = Ki;
   kd = Kd;
}
```

El programa anterior funciona correctamente, pero tiene limitaciones en cuanto a su aplicación a un sistema real. Para que se comporte como un PID de nivel industrial, hay que tener en cuenta otros parámetros; el algoritmo del PID funciona mejor si se ejecuta a intervalos regulares, si se incorpora el concepto del tiempo dentro del PID, se pueden llegar a simplificar los cálculos.

El problema:

Los PID principiantes, están diseñados para ejecutarse a periodos irregulares, esto puede traer 2 problemas:

- Se tiene un comportamiento inconsistente del PID, debido a que en ocaciones se lo ejecuta regularmente y a veces no.
- Hay que realizar operaciones matemáticas extras para calcular los términos correspondientes a la parte derivada e integral del PID, ya que ambos son dependientes del tiempo.

La solución:

Hay que asegurarse que la función que ejecuta el PID lo haga regularmente. Basado en un tiempo de ejecución predeterminado, el PID decide si debe hacer cálculos o retornar de la función. Una vez que nos aseguramos que el PID se ejecuta a intervalos regulares, los cálculos correspondientes a la parte derivada e integral se simplifican.

```
// Variables utilizadas en el controlador PID.
unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastErr;
double kp, ki, kd;
int SampleTime = 1000; // Seteamos el tiempo de muestreo en 1 segundo.

void Compute()
{
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   // Determina si hay que ejecutar el PID o retornar de la función.
   if(timeChange>=SampleTime)
   {
      // Calcula todas las variables de error.
      double error = Setpoint - Input;
      errSum += error;
      double dErr = (error - lastErr);

      // Calculamos la función de salida del PID.
      Output = kp * error + ki * errSum + kd * dErr;

      // Guardamos el valor de algunas variables para el próximo ciclo de cálculo.
      lastErr = error;
      lastTime = now;
   }
}

/* Establecemos los valores de las constantes para la sintonización.
   Debido a que ahora sabemos que el tiempo entre muestras es constante,
   no hace falta multiplicar una y otra vez por el cambio de tiempo; podemos
   ajustar las constantes Ki y Kd, obteniendose un resultado matemático equivalente
   pero más eficiente que en la primera versión de la función. */

void SetTunings(double Kp, double Ki, double Kd)
{
  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {

/* si el usuario decide cambiar el tiempo de muestreo durante el funcionamiento, Ki y Kd tendrán
que ajustarse para reflejar este cambio. */

      double ratio  = (double)NewSampleTime / (double)SampleTime;
      ki *= ratio;
```

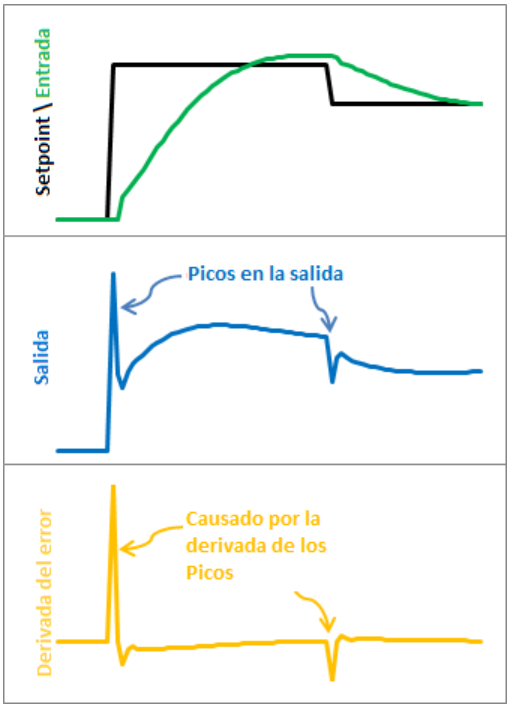
```
    kd /= ratio;
    SampleTime = (unsigned long)NewSampleTime; }}
```

Los resultados:

- Independientemente de cuán frecuente es llamada la función para calcular el PID, el algoritmo de control será evaluado a tiempos regulares.
- Debido la expresión (`int timeChange = (now - lastTime);`) no importa cuado `millis()` se hace cero nuevamente, ya que al tiempo actual, se le resta el tiempo transcurrido con anterioridad.
- Como el tiempo de muestreo ahora es constante, no necesitamos calcular permanentemente las constantes de sintonización. Con lo cuál nos ahorramos cálculos cada vez que procesamos el PID.
- Tener en cuenta que es posible mejorar la gestión de los tiempos de muestreos mediante interrupciones, pero queda a cargo del diseñador la implementación y prueba de este concepto.

Derivative Kick

Esta modificación que presentaremos a continuación, cambiará levemente el termino derivativo con el objetivo de eliminar el fenómeno "***Derivative Kick***". Este fenómeno, se produce por variaciones rápidas en la señal de referencia r(t), que se magnifican por la acción derivativa y se transforman en componentes transitorios de gran amplitud en la señal de control.



La imagen de arriba ilustra el problema. Siendo el *error = setpoint - entrada*, cualquier cambio en la consigna, causa un cambio instantáneo en el error; la derivada de este cambio es infinito (en la práctica, *dt* no es cero, igualmente, el valor termina siendo muy grande). Esto produce un sobrepico muy alto en la salida, que podemos corregir de una manera muy sencilla.

La solución:

$$\frac{d\text{Error}}{dt} = \frac{d\text{Setpoint}}{dt} - \frac{d\text{Input}}{dt}$$

Cuando el setpoint es constante

$$\frac{d\text{Error}}{dt} = -\frac{d\text{Input}}{dt}$$

Resulta que la derivada del error es igual a la derivada negativa de la entrada, salvo cuando el setpoint está cambiando, esto acaba siendo una solución perfecta. En lugar de añadir (*Kd * error derivado*), restamos (*Kd * valor de entrada derivado*). Esto se conoce como el uso de "*Derivada de la medición*".

El código:

```
// Variables utilizadas en el controlador PID.

unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastInput;
double kp, ki, kd;
int SampleTime = 1000; // Tiempo de muestreo de 1 segundo.

void Compute()
{
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      // Calcula todas las variables de errores.
      double error = Setpoint - Input;
      errSum += error;
      double dInput = (Input - lastInput);
```

```
      // Calculamos la función de salida del PID.
      Output = kp * error + ki * errSum - kd * dInput;

      // Guardamos el valor de algunas variables para el próximo ciclo de cálculo.
      lastInput = Input;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}
```

Las modificaciones son bastante sencillas, estamos reemplazando la derivada positiva del error con la derivada negativa de la entrada. En vez de recordar el último valor del error, ahora recordamos el último valor que tomó la entrada.

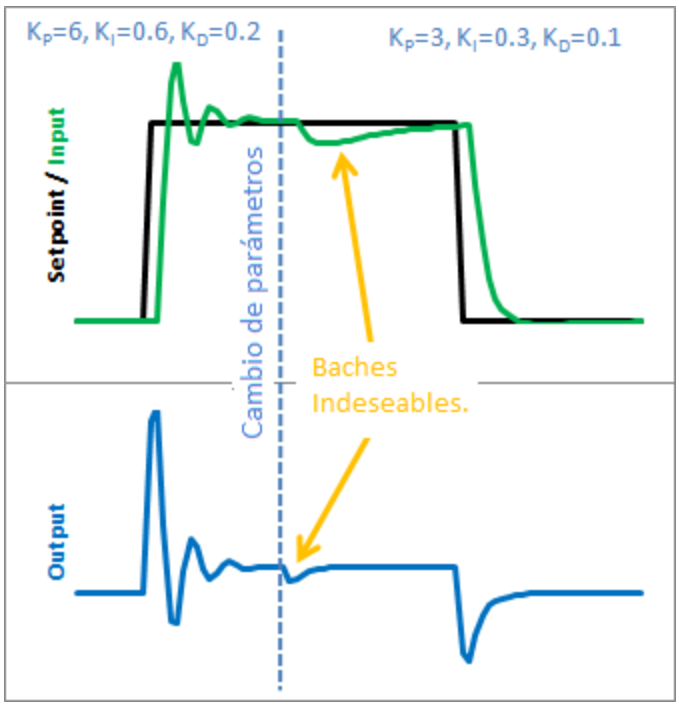
El resultado:

Podemos ver como los picos en la salida han sido eliminados. Este factor de corrección se aplica a sistemas muy sensibles a dichas variaciones; en un horno por ejemplo, donde la inercia térmica es muy grande, no le afectan en lo más mínimo dichos picos. Por lo tanto no sería necesario tener en cuenta esta corrección.

Cambios en la sintonización

El problema:

La posibilidad de cambiar la sintonización del PID, mientras el sistema está corriendo, es la característica más respetable del algoritmo del sistema de control.



Los PID principiantes, de hecho, actúan de manera errática si queremos setear los valores de la sintonización, mientras el sistema está corriendo. Veamos por que. Aquí se muetra el estado del PID antes y después de que los parámetros han cambiado.

La salida se reduce a la mitad

	Output	= kp	* error	+ ki	* errSum	- kd	* dInput
Justo Antes	0.98	6	-0.01	0.6	1.73	-0.2	0.02
Justo Después	0.49	3	-0.01	0.3	1.72	-0.1	-0.01

Debido a que el término integral rápidamente se ha reducido a la mitad.

De inmediato podemos ver que el culpable de este bache en la señal de salida es el término integral; es el único término que cámbia drásticamente cuando la señal de sintonización se modifica. Esto sucede debido a la interpretación de la integral.

$$K_I \int e\,dt \approx K_{I_n}[e_n + e_{n-1} + \ldots]$$

Esta interpretación funciona bien hasta que Ki cambia. De repente, la suma de todos los errores se multiplica con el valor de Ki, esto no es lo que necesitamos. Nosotros solo queremos que afecte a los valores que estén por delante. Por ejemplo: Si nosotros modificamos Ki, en un tiempo t=5s. Necesitamos que el impacto de este cambio solo afecte a valores de Ki que se modifican en un tiempo mayor a t=5s.

La solución:

La solución a este error no queda muy elegante, pero consiste en reescalar la suma del error, doblando el valor de Ki o cortando la suma de los errores a la mitad. Esto quita el bache del termino integral, solucionando el problema.

$$K_I \int e\,dt = \int K_I\, e\,dt$$

$$\int K_I\, e\,dt \approx K_{I_n} e_n + K_{I_{n-1}} e_{n-1} + \ldots$$

En lugar de tener el término Ki fuera de la integral, lo introducimos dentro del cálculo. Al parecer, no hemos realizado nada extraño, pero en la práctica está

acción resulta en una grán diferencia en la función de salida del PID.

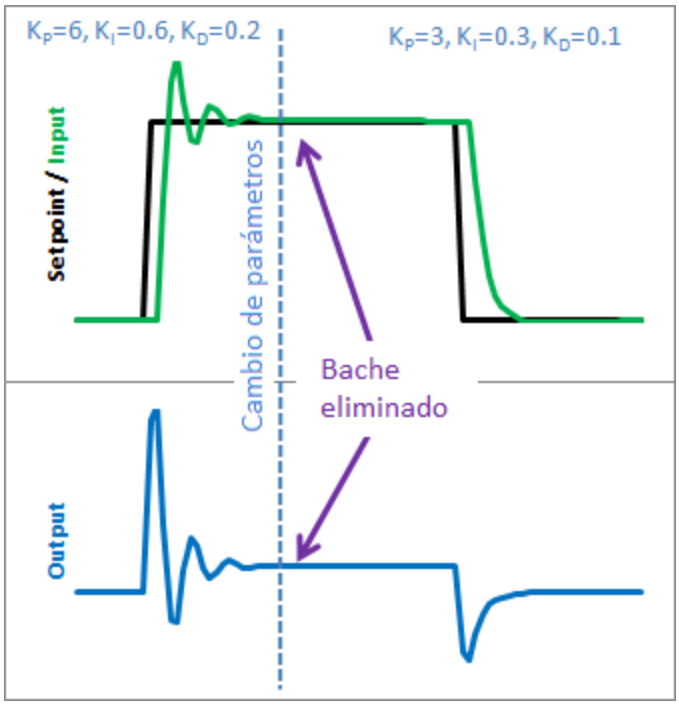
Ahora tomamos el error y lo multiplicamos por el valor de Ki en ese momento, luego almacenamos la suma de los diferentes errores multiplicados por la constante Ki. Esto resulta en una función de salida, suave y sin sobresaltos, con la ventaja de no tener que utilizar matemática adicional para ello.

```
// Variables utilizadas en el controlador PID.
unsigned long lastTime;
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; // Tiempo de muestreo: 1 segundo.

void Compute()
{
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      // Calcula todos los errores.
      double error = Setpoint - Input;
      ITerm += (ki * error);
      double dInput = (Input - lastInput);

      // Calculamos la función de salida del PID.
      Output = kp * error + ITerm - kd * dInput;

      // Guardamos el valor de algunas variables para el próximo recálculo.
      lastInput = Input;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
```

```
    }
}
```

Reemplazamos la variable errSuma, por una variable compuesta llamada Iterm. Suma *Ki*error* en lugar de solamente el error. Por último, como el cálculo de Ki está embebido en el término integral, se elimina de la ecuación principal del PID.

El resultado:



No hay bache en la salida.

	Output	= kp * error +	ITerm	- kd * dInput	
Justo Antes	0.98	6 -0.01	1.04	-0.2	0.02
Justo Después	1.01	3 -0.01	1.04	-0.1	-0.01

Con las modificaciones hechas, los cambios en la sintonización del término integral no afectan al rendimiento general de nuestro sistema, ya que tiene en cuenta la modificación en cada instancia de error, sumandose al total.

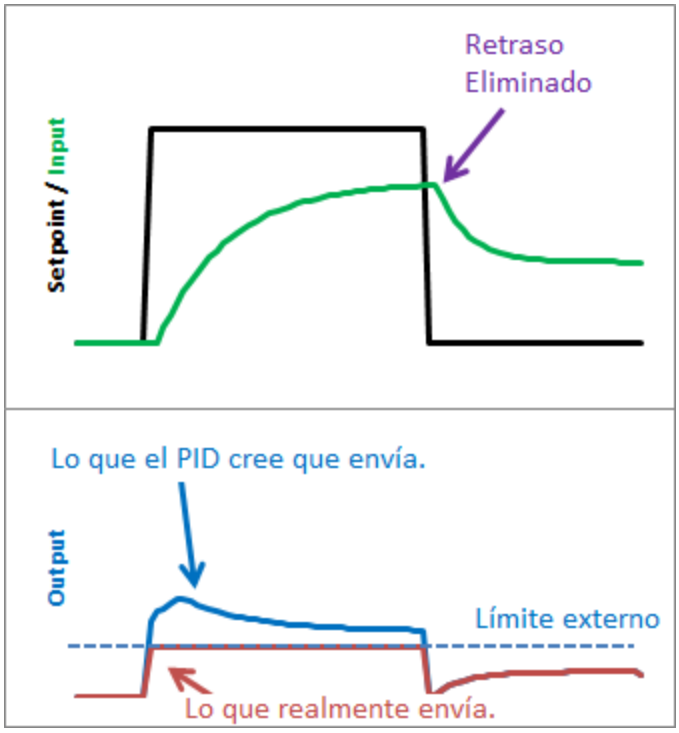
Reset WindUp

El problema:

El efecto windup aparece al arrancar el sistema o en cualquier otra situación, donde aparece un error muy grande durante un tiempo prolongado. Esto hará que el término integral aumente para reducir el error. Pero si nuestro actuador es limitado, con esto me refiero que la tensión que podemos aplicarle esta entre 0 y 5V (0 a 255 , pwm de 8 bits), se saturará, pero el termino integral seguirá creciendo. Cuando el error se reduce, la parte integral también comenzará a reducirse, pero desde un valor muy alto, llevando mucho tiempo hasta que logre la estabilidad, generando fluctuaciones exageradamente grandes.

El problema se manifiesta en forma de retrasos extraños. En la imagen podemos ver que el valor de la salida, está muy por encima del límite. Cuando el valor del setpoint cae por debajo de un valor determinado, el valor de salida decrece por debajo de la línea límite de 255 (5v).

La solución - Paso 1:



Hay varias formas para mitigar el efecto del WindUp, pero la elegida es la siguiente: decirle al PID cuáles son los límites de salida. En el código de abajo, veremos que ahora hay una función SetOuputLimits. Una vez que ya se alcanza el límite, el PID detiene el funcionamiento del término integral.

La solución - Paso 2:

Observe en el gráfico anterior, si bien nos libramos del retraso inducido por el WindUp, no hemos resuelto todo el problema. Todavía hay una diferencia, entre lo que el pid piensa que está enviando, y lo que está enviando. ¿Por qué? Veamos el término proporcional y (en menor medida) el término derivativo.

Aunque el término integral ha sido acotado de forma segura, el termino Proporcional y Derivativo están añadiendo pequeños valores adicionales, dando un resultado superior al límite de salida. Esto es inaceptable. Si el usuario llama a

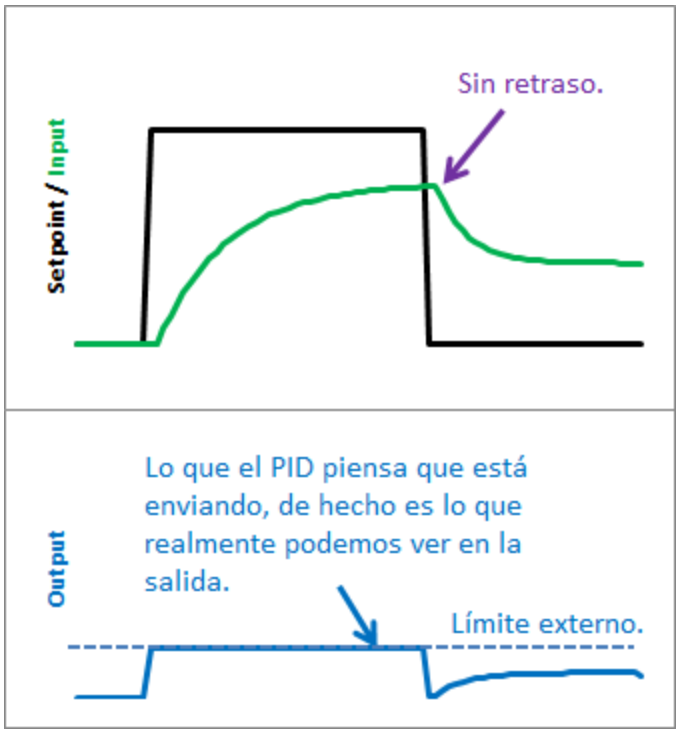
la función "SetOutputLimits" tiene que asumir que eso significa "la salida se mantendrá dentro de estos valores." Así que en el paso 2, hacemos una suposición válida. Además de la restricción del término Integral, hay que acotar el valor de salida para que se mantenga dentro de los límites.

Uno se preguntará, por que acotamos el termino integral y la salida. Esto se debe a lo siguiente: Por más que pongamos límites al valor que puede tomar la salida, el término integral seguiría creciendo, introduciendo errores en la salida.

El código:

```
// Variables de trabajo.
unsigned long lastTime;
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; // Tiempo de muestreo de 1 segundo.
double outMin, outMax;
void Compute()
{
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      // Calcula todos los errores.
      double error = Setpoint - Input;
      ITerm+= (ki * error);
      if(ITerm> outMax) ITerm= outMax;
      else if(ITerm< outMin) ITerm= outMin;
      double dInput = (Input - lastInput);

      // Calculamos la función de salida del PID.
      Output = kp * error + ITerm- kd * dInput;
      if(Output > outMax) Output = outMax;
      else if(Output < outMin) Output = outMin;

      // Guardamos el valor de algunas variables para el próximo recálculo.
      lastInput = Input;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
```

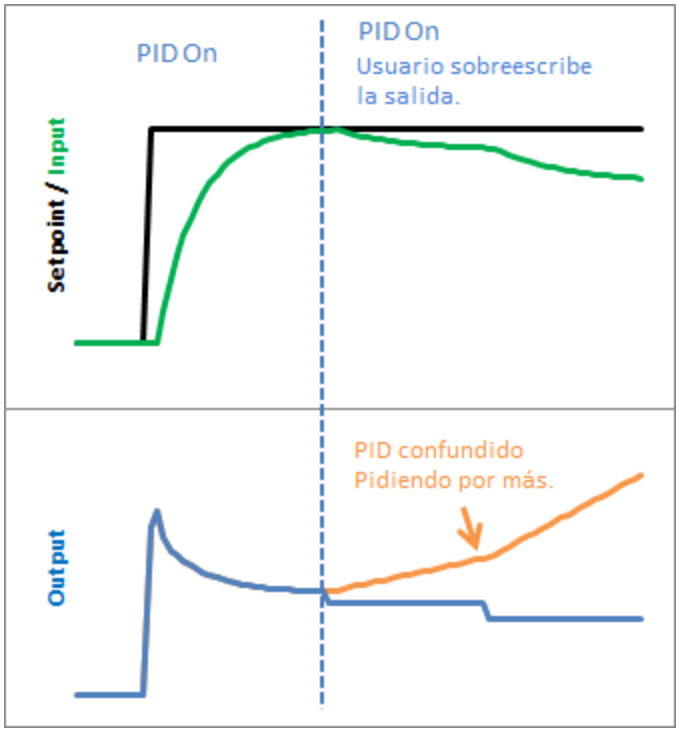
```
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}

void SetOutputLimits(double Min, double Max)
{
   if(Min > Max) return;
   outMin = Min;
   outMax = Max;

   if(Output > outMax) Output = outMax;
   else if(Output < outMin) Output = outMin;

   if(ITerm> outMax) ITerm= outMax;
   else if(ITerm< outMin) ITerm= outMin;
}
```



Como podemos ver el fenómeno del WindUp es eliminado. Además, podemos ver que la salida permanece dentro del rango que deseamos. Esto significa que podemos configurar el rango de valores máximos y mínimos que necesitamos en la salida.

PID: On/Off

El problema:



Digamos que en algún momento del programa deseamos forzar la salida a un valor determinado (0 por ejemplo), usando la siguiente rutina:

```
void loop()
{
Compute();
Output=0;    }
```

De esta manera no importa el valor de salida que haya computado el PID, nosotros simplemente determinamos su valor manualmente. Esto en la práctica es erróneo ya que introducirá errores en el PID: Dirá, yo estoy variando la función de salida, pero en realidad no pasa nada. Como resultado, cuando pongamos nuevamente el PID en funcionamiento, tendremos un cambio brusco y repentino en el valor de la función de salida.

La solución:

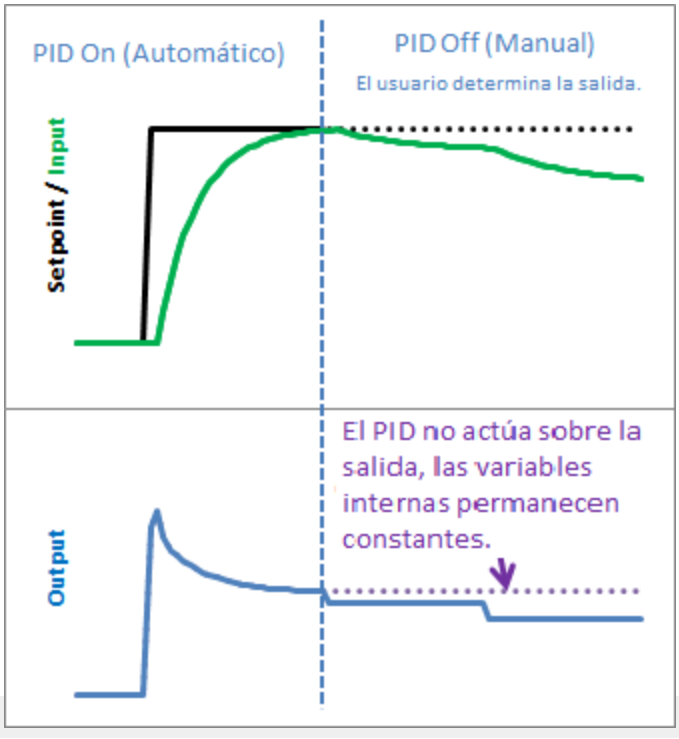
La solución a este problema es tener un medio para encender o apagar el PID de vez en cuando. Los términos comunes para estos estados son "**Manual**" (*ajustar el valor de la salida manualmente*) y "**Automatic**" (*el PID ajusta automáticamente la salida*). Vamos a ver cómo se hace esto en el código:

```
// Variables de trabajo.
unsigned long lastTime;
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; // Tiempo de muestreo 1 segundo.
double outMin, outMax;
bool inAuto = false;

#define MANUAL 0
#define AUTOMATIC 1

void Compute()
{
   if(!inAuto) return;
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      // Calculamos todos los errores.
      double error = Setpoint - Input;
      ITerm+= (ki * error);
      if(ITerm> outMax) ITerm= outMax;
      else if(ITerm< outMin) ITerm= outMin;
      double dInput = (Input - lastInput);

      // Calculamos la función de salida del PID.
      Output = kp * error + ITerm- kd * dInput;
      if(Output > outMax) Output = outMax;
      else if(Output < outMin) Output = outMin;
```

```
      // Guardamos el valor de algunas variables para el próximo recálculo.
      lastInput = Input;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}

void SetOutputLimits(double Min, double Max)
{
   if(Min > Max) return;
   outMin = Min;
   outMax = Max;

   if(Output > outMax) Output = outMax;
   else if(Output < outMin) Output = outMin;

   if(ITerm> outMax) ITerm= outMax;
   else if(ITerm< outMin) ITerm= outMin;
}

void SetMode(int Mode)
{
  inAuto = (Mode == AUTOMATIC);
}
```

Una solución bastante simple. Si no está en modo automático, sale inmediatamente de la función de cómputo del PID, sin ajustar la salida ni las variables internas del mismo.

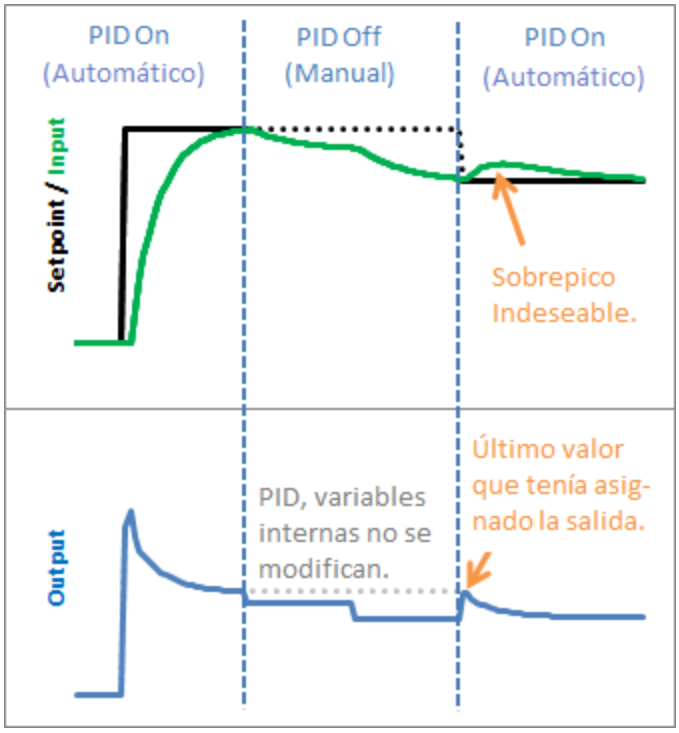
El resultado:



Ciertamente podríamos conseguir un efecto similar sin llamar a la función que calcula el PID, pero esta solución mantiene las variables del PID contenidas. Al mantener el valor de dichas variables, podemos hacer un seguimiento de los valores de las mismas, y lo más importante, vamos a saber cuando podemos cambiar los modos.

PID: Inicialización

Anteriormente habíamos implementado la posibilidad de encender o apagar el PID de vez en cuando. Ahora vamos a ver lo que pasa cuando volvemos a encenderlo:



Aquí tenemos un problema, el PID entrega a la salida el última valor computado, luego comienza a corregir a partir de ahí. Esto resulta en un sobrepico en la entrada que es preferible no tener.

La solución:

Esto es bastante fácil de solucionar. Ahora sabemos que al pasar de manual a automático, sólo tenemos que inicializar los parámetros para una transición sin problemas. Esto significa, inicializar el valor de la entrada con el último valor almacenado, e inicializar el término integral con el último valor que tomó la salida, para evitar los sobrepicos en la salida.

El código:

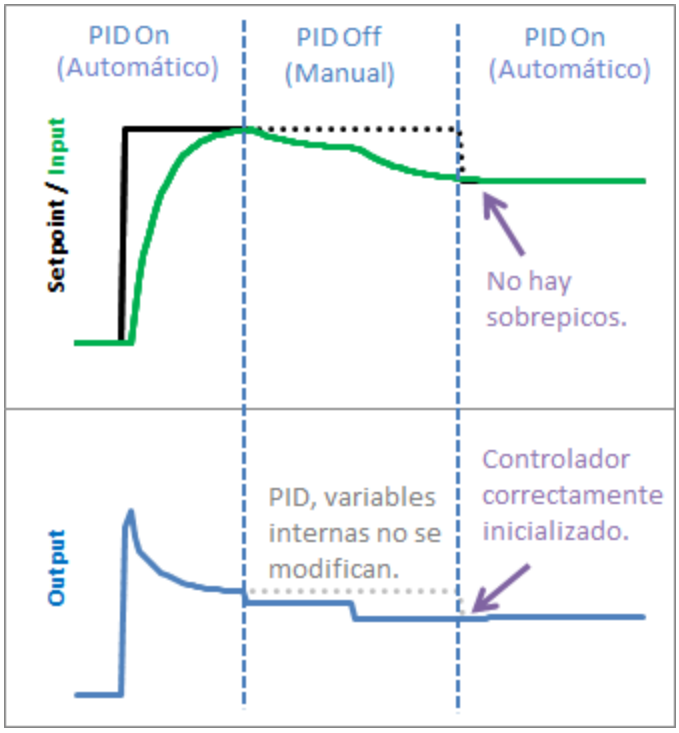
```
// Variables de trabajo.
unsigned long lastTime;
```

```
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; // Tiempo de muestreo 1 segundo.
double outMin, outMax;
bool inAuto = false;

#define MANUAL 0
#define AUTOMATIC 1

void Compute()
{
   if(!inAuto) return;
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      // Calculamos todos los errores.
      double error = Setpoint - Input;
      ITerm+= (ki * error);
      if(ITerm> outMax) ITerm= outMax;
      else if(ITerm< outMin) ITerm= outMin;
      double dInput = (Input - lastInput);

      // Calculamos la función de salida del PID.
      Output = kp * error + ITerm- kd * dInput;
      if(Output > outMax) Output = outMax;
      else if(Output < outMin) Output = outMin;

      // Guardamos el valor de algunas variables para el próximo recálculo.
      lastInput = Input;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}
```

```
void SetOutputLimits(double Min, double Max)
{
   if(Min > Max) return;
   outMin = Min;
   outMax = Max;

   if(Output > outMax) Output = outMax;
   else if(Output < outMin) Output = outMin;

   if(ITerm> outMax) ITerm= outMax;
   else if(ITerm< outMin) ITerm= outMin;
}


void SetMode(int Mode)
{
    bool newAuto = (Mode == AUTOMATIC);
    if(newAuto && !inAuto)
    {  // Para cambiar de manual a automático, inicializamos algunos parámetros.
        Initialize();
    }
    inAuto = newAuto;
}

void Initialize()
{
   lastInput = Input;
   ITerm = Output;
   if(ITerm> outMax) ITerm= outMax;
   else if(ITerm< outMin) ITerm= outMin;
}
```

Hemos modificado setMode () para detectar el paso de manual a automático y hemos añadido nuestra función de inicialización. En él se establecen (`iTerm = salida`) cuidar de que el término integral, y (`LastInput = Entrada`) para mantener la derivada de la adición. El término proporcional no se basa en la información del pasado, por lo que no necesita ningún tipo de inicialización.

El resultado:



Vemos en el gráfico anterior que una inicialización adecuada, da como resultado, una transferencia de manual a automático sin perturbaciones: exactamente lo que estábamos buscando.

PID: Dirección

El problema:

Los procesos a los cuáles un PID estará enlazado, se dividen 2 grupos: de acción directa y de acción inversa. Todos los ejemplos vistos hasta el momento han sido de acción directa, por lo tanto, un incremento en la entrada, da como resultado un incremento en la salida. En el caso de los procesos de acción reversa, es todo lo contrario.

En un refrigerador, por ejemplo, un aumento en la acción de enfriamiento, causa una disminución de la temperatura. Para que el PID funcione en un proceso de acción inversa, los signos de Kp, Ki, y Kd deben ser negativos.

Esto no es un problema por si mismo, pero el usuario debe elegir el signo correcto, y asegúrese de que todos los parámetros tengan el mismo signo.

La solución:

Para hacer el proceso un poco más simple, se requiere que los parámetros Kp, Ki, y kd sean > = 0. Si el usuario está trabajando en un proceso de acción inversa, se especifica por separado, utilizando la función SetControllerDirection. esto asegura que los parámetros tienen el mismo signo.

El código:

```
// Variables de trabajo.
unsigned long lastTime;
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; // Tiempo de muestreo 1 segundo.
double outMin, outMax;
bool inAuto = false;

#define MANUAL 0
#define AUTOMATIC 1
#define DIRECT 0
#define REVERSE 1
int controllerDirection = DIRECT;


void Compute()
{
   if(!inAuto) return;
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      // Calculamos todos los errores.
      double error = Setpoint - Input;
      ITerm+= (ki * error);
      if(ITerm> outMax) ITerm= outMax;
      else if(ITerm< outMin) ITerm= outMin;
      double dInput = (Input - lastInput);

      // Calculamos la función de salida del PID.
      Output = kp * error + ITerm- kd * dInput;
      if(Output > outMax) Output = outMax;
      else if(Output < outMin) Output = outMin;
```

```
      // Guardamos el valor de algunas variables para el próximo recálculo.
      lastInput = Input;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
   if (Kp<0 || Ki<0|| Kd<0) return;

  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;

  if(controllerDirection ==REVERSE)
   {
      kp = (0 - kp);
      ki = (0 - ki);
      kd = (0 - kd);
   }
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}

void SetOutputLimits(double Min, double Max)
{
   if(Min > Max) return;
   outMin = Min;
   outMax = Max;

   if(Output > outMax) Output = outMax;
   else if(Output < outMin) Output = outMin;

   if(ITerm> outMax) ITerm= outMax;
   else if(ITerm< outMin) ITerm= outMin;
}


void SetMode(int Mode)
{
    bool newAuto = (Mode == AUTOMATIC);
```

```
    if(newAuto && !inAuto)
    {  // Para cambiar de manual a automático, inicializamos algunos parámetros.
        Initialize();
    }
    inAuto = newAuto;
}

void Initialize()
{
   lastInput = Input;
   ITerm = Output;
   if(ITerm> outMax) ITerm= outMax;
   else if(ITerm< outMin) ITerm= outMin;
}

void SetControllerDirection(int Direction)
{
   controllerDirection = Direction;
}
```

PROGRAMACIÓN COMPLETA DE ARDUINO

```
#include <Keypad.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,2,1,0,4,5,6,7,3, POSITIVE);
char* secretCode = "ABCD";
int position = 0;
int Sensor_presencia = 6;
int Presencia = LOW;
int ESTADO_ALARMA=LOW;
int ALERTA= LOW;
int buzz=13;
int redPin = 46;
int greenPin = 47;
int pin=52;

const byte rows = 4;
const byte cols = 4;
char keys[rows][cols] = {
 {'1','2','3', 'A'},
 {'4','5','6', 'B'},
 {'7','8','9', 'C'},
 {'#','0','*', 'D'}
};
byte rowPins[rows] = {A8, A9, A10, A11};          //Establece las filas del keypad
 byte colPins[cols] = {A12, A13, A14, A15};       //Establece las columnas del keypad
 Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, rows, cols );

#include <PID_v1.h>
#define ventilador1 12
#define ventilador2 11
#include "DHT.h"
#define DHTPIN A0 //INPUT
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
double Setpoint, Input, Output;
int Error=0;
//introducción de parámetros PID
double Kp=35.2895, Ki=0.17802, Kd=0;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, REVERSE);
int t;
int h;

int LUZ_COCINA=31;       // COCINA
int COCINA=30;           // Pin interruptor
int PULSADOR_COCINA;
int ESTADO_COCINA=0;

int LUZ_SALON=33;        // SALON
int SALON=32;            // Pin interruptor
int PULSADOR_SALON;
int ESTADO_SALON=0;

int LUZ_DORMITORIO=35;   // DORMITORIO
int DORMITORIO=34;       // Pin interruptor
int PULSADOR_DORMITORIO;
int ESTADO_DORMITORIO=0;
```

```
int LUZ_ASEO1=37;        //  BAÑO 1
int ASEO1=36;            // Pin interruptor
int PULSADOR_ASEO1;
int ESTADO_ASEO1=0;

int LUZ_ASEO2=39;        //BAÑO 2
int ASEO2=38;            // Pin interruptor
int PULSADOR_ASEO2;
int ESTADO_ASEO2=0;

int DIMMER_SALON_UP=40;
int ESTADO_SALON1;
int DIMMER_SALON_DOWN=41;
int ESTADO_SALON2;          // DIMMER SALON
int LUZ_DIMMER_SALON=3;
int SUM_DIMMER=0;
int ESTADO_DIMMER=0;

int DIMMER_HABITACION_UP=42;
int ESTADO_HABITACION1;
int DIMMER_HABITACION_DOWN=43;
int ESTADO_HABITACION2;          // DIMMER HABITACION
int LUZ_DIMMER_HABITACION=4;
int SUM2_DIMMER=0;
int ESTADO_DIMMER2=0;
char Estados[14];    //Creamos un array que almacenará los caracteres
byte posicion=0;
#define E1 9    // Enable Pin  para motor 1
#define I1 8    // Control pin 1 para el motor 1
#define I2 7    // Control pin 2 para el motor 1
int PERSIANA_BOTON_ARRIBA=52;
int PERSIANA_BOTON_ABAJO=53;
int Estado1=0;
int Estado2=0;
int Persiana=0;
int luz=0;
int SensorLuz=A1;
int SensorGas=A2;

void setup() {
 Serial.begin(9600);
 Serial3.begin(9600);
 lcd.begin (16,2);
 Setpoint = 25;
 dht.begin();
 pinMode(redPin, OUTPUT);
 pinMode(greenPin, OUTPUT);
 pinMode(pin,OUTPUT);
 pinMode(Sensor_presencia, INPUT);
 pinMode(buzz, OUTPUT);
 ALARMA(false);      //Estado inicial de la alarma es desactivada

 pinMode(COCINA,INPUT);
 pinMode(DORMITORIO,INPUT);
 pinMode(SALON,INPUT);
 pinMode(ASEO1,INPUT);
 pinMode(ASEO2,INPUT);

 pinMode(LUZ_COCINA,OUTPUT);
 pinMode(LUZ_SALON,OUTPUT);
 pinMode(LUZ_DORMITORIO,OUTPUT);
 pinMode(LUZ_ASEO1,OUTPUT);
```

```
 pinMode(LUZ_ASEO2,OUTPUT);

  pinMode(DIMMER_SALON_UP,INPUT);
 pinMode(DIMMER_SALON_DOWN,INPUT);
 pinMode(DIMMER_HABITACION_UP,INPUT);
 pinMode(DIMMER_HABITACION_DOWN,INPUT);

 pinMode(LUZ_DIMMER_SALON,OUTPUT);
 pinMode(LUZ_DIMMER_HABITACION,OUTPUT);

  for (int i = 8 ; i<11 ; i++)              // Inicializamos persiana
     pinMode( i, OUTPUT);
     pinMode(PERSIANA_BOTON_ARRIBA,INPUT);
     pinMode(PERSIANA_BOTON_ABAJO,INPUT);
     pinMode(SensorLuz,INPUT);
     pinMode(SensorGas,INPUT);
}


void loop() {

 //Recepción Serial3 Bluetooth
 if(Serial3.available())
 {

 while(Serial3.available()>0) //Mientras haya datos en el buffer ejecuta la función
  {
   delay(5);                    //Ponemos un pequeño delay para mejorar la recepción de datos
   Estados[posicion]=Serial3.read();  //Lee un carácter del string "cadena" de la "posicion", luego lee el siguiente carácter con
   posicion++;                        "posicion++"
  }}


 // LECTURA INTERRUPTORES
PULSADOR_COCINA=digitalRead(COCINA);
PULSADOR_SALON=digitalRead(SALON);
PULSADOR_DORMITORIO=digitalRead(DORMITORIO);
PULSADOR_ASEO1=digitalRead(ASEO1);
PULSADOR_ASEO2=digitalRead(ASEO2);

// LECTURA PULSADORES DIMMER
ESTADO_SALON1=digitalRead(DIMMER_SALON_UP);
ESTADO_SALON2=digitalRead(DIMMER_SALON_DOWN);
ESTADO_HABITACION1=digitalRead(DIMMER_HABITACION_UP);
ESTADO_HABITACION2=digitalRead(DIMMER_HABITACION_DOWN);

//LECTURA PULSADORES PERSIANA
Estado1=digitalRead(PERSIANA_BOTON_ARRIBA);
Estado2=digitalRead(PERSIANA_BOTON_ABAJO);

int Dim1=0;
int Dim2=0;
int Vent1=0;
int Vent2=0;
int Remoto_cocina=Estados[0]-48;
int Remoto_salon=Estados[1]-48;
int Remoto_dormitorio=Estados[2]-48;  // Recepción mediante serial de una cadena de datos
int Remoto_aseo1=Estados[3]-48;
int Remoto_aseo2=Estados[4]-48;
int Remoto1_dim=Estados[5]-48;
int Remoto2_dim=Estados[6]-48;
int Remoto_Alarma=Estados[7]-48;
```

```
int Remoto_Climatizador=Estados[8]-48;
int Remoto_Ventilador1=Estados[9]-48;
int Remoto_Ventilador2=Estados[10]-48;
int Remoto_PersianaAuto=Estados[11]-48;
int Remoto_PersianaUP=Estados[12]-48;
int Remoto_PersianaDOWN=Estados[13]-48;

if(PULSADOR_COCINA && Remoto_cocina ){   // Conmutación entre recepción serial y interruptor
 LUCES(LUZ_COCINA,1);
 ESTADO_COCINA=1;
}else{
  LUCES(LUZ_COCINA,0);
    ESTADO_COCINA=0;
}
if(PULSADOR_SALON && Remoto_salon){
 LUCES(LUZ_SALON,1);
 ESTADO_SALON=1;
}else{
  LUCES(LUZ_SALON,0);
   ESTADO_SALON=0;
}
if(PULSADOR_DORMITORIO && Remoto_dormitorio){
 LUCES(LUZ_DORMITORIO,1);
 ESTADO_DORMITORIO=1;
}else{
  LUCES(LUZ_DORMITORIO,0);
   ESTADO_DORMITORIO=0;
}
if(PULSADOR_ASEO1 && Remoto_aseo1){
 LUCES(LUZ_ASEO1,1);
 ESTADO_ASEO1=1;
}else{
  LUCES(LUZ_ASEO1,0);
  ESTADO_ASEO1=0;
}
if(PULSADOR_ASEO2 && Remoto_aseo2){
 LUCES(LUZ_ASEO2,1);
 ESTADO_ASEO2=1;
}else{
  LUCES(LUZ_ASEO2,0);
   ESTADO_ASEO2=0;
}
      //Dimmers
SUM_DIMMER=constrain(SUM_DIMMER,0,255);         //Limita el estado de los dimmers entre (0-255)
SUM2_DIMMER=constrain(SUM2_DIMMER,0,255);
if(ESTADO_SALON1==1){
  SUM_DIMMER++;
}
if(ESTADO_SALON2==1){
   SUM_DIMMER--;
}
if(ESTADO_HABITACION1==1){
  SUM2_DIMMER++;
}
if(ESTADO_HABITACION2==1){
  SUM2_DIMMER--;
}
if(SUM_DIMMER!=0){          // Cambia el estado de los dimmers
 ESTADO_DIMMER=1;
}else{
 ESTADO_DIMMER=0;}
if(SUM2_DIMMER!=0){
```

```
 ESTADO_DIMMER2=1;
}else{
 ESTADO_DIMMER2=0;}
 //Dimmer Remoto
if(Remoto1_dim>=0){
Dim1=map(Remoto1_dim,0,9,0,255);  //Mapea el caracter recibido (0-9) y lo transforma en un valor entre (0-255)
}
if(SUM_DIMMER>=Dim1){
DIMMER(LUZ_DIMMER_SALON,SUM_DIMMER);
}else{
 DIMMER(LUZ_DIMMER_SALON,Dim1);
}
if(Remoto2_dim>=0){
Dim2=map(Remoto2_dim,0,9,0,255);
}
if(SUM2_DIMMER>=Dim2){
DIMMER(LUZ_DIMMER_HABITACION,SUM2_DIMMER);
}else{
 DIMMER(LUZ_DIMMER_HABITACION,Dim2);
}
if(ESTADO_HABITACION1==0 && ESTADO_HABITACION2==0 && ESTADO_SALON1==0 && ESTADO_SALON2==0 && Estado1==0
&& Estado2==0){
 h = dht.readHumidity();        // Lee la humedad
 t= dht.readTemperature();    //Lee la temperatura
}
//Envío de datos mediante serial3
Serial3.print(t);
Serial3.print("ºC");
Serial3.print("\n");
Serial3.print(h);
Serial3.print(".%");

//Inicio modo PID ventilación
Input=t;
Vent1=map(Remoto_Ventilador1,0,9,0,255);
Vent2=map(Remoto_Ventilador2,0,9,0,255);
if(Remoto_Climatizador==1){
  myPID.SetMode(AUTOMATIC);   // Selección del modo de funcionamiento (MANUAL/AUTOAMATIC)
  myPID.Compute();
  Error=Setpoint-Input;
  analogWrite(ventilador1, Output);
  analogWrite(ventilador2, Output);
}
if(Remoto_Climatizador==0){
  myPID.SetMode(MANUAL);
 analogWrite(ventilador1,Vent1);
 analogWrite(ventilador2,Vent2);
}


Persiana=constrain(Persiana,0,50);   //Limita el movimiento de la persiana si esta subida o bajada
if((Persiana!=50)&&(Estado1==1)){   //Se debe regular el valor 50 en función del tiempo de subida de la persiana para no forzar el
PERSIANA_ARRIBA();                    motor
 Persiana++;
}else{
PERSIANA_ESTOP();
}
if((Persiana!=0)&&(Estado2==1)){
 PERSIANA_ABAJO();
 Persiana--;
}else{
PERSIANA_ESTOP();
```

```
}
if(Remoto_PersianaAuto==1){
 PERSIANA_AUTO(1);
}
if(Remoto_PersianaUP==1){
  PERSIANA_ARRIBA();
  Persiana++;
}else{
  PERSIANA_ESTOP();
}
if(Remoto_PersianaDOWN==1){
  PERSIANA_ABAJO();
  Persiana--;
}else{
  PERSIANA_ESTOP();
}

PASSWORD();    // En función de lo tecleado activaremos o no la alarma
if(ESTADO_ALARMA==HIGH){
SENSOR_PRESENCIA(1);
}
if(ALERTA==HIGH){
  ring(1);

}else{
  ring(0);
}
if(Remoto_Alarma==2){
ALARMA(1);
}
if(Remoto_Alarma==1){
ALARMA(0);
}
//FUNCIONES AUXILIARES
Gas(1);
SERIAL_GENERAL(0);  // 0-Desactivar   1-Activar
SERIAL_PID(0);
CODIGO_LCD(1);
PERSIANA_AUTO(0);
posicion=0;
}

//*******************************FUNCIONES*****************************************


//FUNCION PARA ENCENDER LUCES. Envía estancia y el estado de la misma.
void LUCES(int Estancia,int estado){
  digitalWrite(Estancia,estado);
}

//FUNCION REGULACION DIMMERS
void DIMMER(int estancia,int valor){
valor=constrain(valor,0,255);
   analogWrite(estancia,valor);

}
//FUNCION ACTIVA O DESACTIVA VISUAL PUERTO SERIE ESTADO DE VARIABLES
void SERIAL_GENERAL(int estado){
  if(estado==1){
char LUCES[8] =
{ESTADO_COCINA+48,ESTADO_SALON+48,ESTADO_DORMITORIO+48,ESTADO_ASEO1+48,ESTADO_ASEO2+48,ESTADO_DIMME
R+48,ESTADO_DIMMER2+48};
```

```
Serial.print("Estado luces:");
Serial.print("   ");
Serial.print(LUCES);
Serial.print("   ");
Serial.print("Dimmer Salon: ");
Serial.print(map(SUM_DIMMER,0,255,0,100));
Serial.print("%");
Serial.print("   ");
Serial.print("Dimmer Hab: ");
Serial.print(map(SUM2_DIMMER,0,255,0,100));
Serial.print("%");
Serial.print("   ");
Serial.print("Temperatura: ");
Serial.print(t);
Serial.print("*C");
Serial.print("   ");
Serial.print("Humeda R: ");
Serial.print(h);
Serial.print("%");
Serial.print("   ");
Serial.print("Persiana: ");
Serial.print(map(Persiana,0,130,0,100));
Serial.print("%");
Serial.print("\n");
Serial.print("       ");
Serial.print("\n");
}}


 // FUNCION LECTURA KEYPAD    si la pass es correcta activa la alarma
int PASSWORD(){
  char key = keypad.getKey();

  if (key == '#') {
    position = 0;
    lcd.clear();
    lcd.setCursor(3,0);
    lcd.print("ACTIVANDO..");
    delay(5000);
    Buzz(2000);    // Función Buzz, enviando frecuencia de sonido.
    delay(500);    // Tiempo de retardo hasta que se active para poder salir.
    Buzz(0);
    ALARMA(1);
  }
  if (key == secretCode[position]){
    lcd.clear();
    position++;
    lcd.setCursor(0,0);
    lcd.print("Pass:");
    lcd.setCursor(5+position,0);
    lcd.print("*");
   if(position>=6){
  }
if (position == 6) {
    Buzz(2000);         // Función Buzz, enviando frecuencia de sonido.
    delay(500);         // Tiempo de retardo hasta que se active para poder salir.
    Buzz(0);
  ALARMA(0);
   lcd.clear();
   digitalWrite(greenPin,LOW);
   digitalWrite(redPin,HIGH);
     delay(1000);
```

```
    position=0;
}}
}
```

```
//FUNCION ALARMA  Dependiendo de el código introducido activamos o desactivamos la alarma
int ALARMA(int estado){
  if (estado==1) {
    Buzz(1000);
    delay(500);
    Buzz(0);
    ESTADO_ALARMA=HIGH;
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("ALARMA ACTIVADA");
    digitalWrite(greenPin,HIGH);
    digitalWrite(redPin,LOW);
    delay(1500);
    lcd.clear();

  } else {
     Buzz(1000);
    delay(500);
     Buzz(0);
     ESTADO_ALARMA=LOW;
     ALERTA=LOW;
     digitalWrite(pin, LOW);
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("ALARMA DESACTIVA");
    digitalWrite(greenPin,LOW);
    digitalWrite(redPin,HIGH);
    delay(1500);
    lcd.clear();

}}
```

```
//FUNCION SENSOR DE PRESENCIA. Si la alarma esta activa el sensor de presencia estará ON junto conto con sus actuadores
  void SENSOR_PRESENCIA(int estado){
    if(estado==1){
     int val = digitalRead(Sensor_presencia);
     Serial.println(val);
      delay(5);
  if (val == HIGH) {
    digitalWrite(pin, HIGH);
  if (Presencia == LOW) {
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("ALERTA PRESENCIA");
       Presencia = HIGH;
       ALERTA=HIGH;
       Persiana_Alarma();
    }}
  else {
      digitalWrite(pin, LOW);
      if (Presencia == HIGH){
        Presencia = LOW;
          }}}}
```

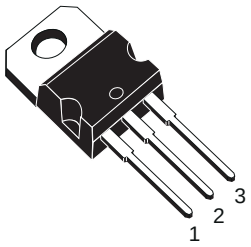
```
 //FUNCION ALARMA SONORA
void ring (int estado){
 if(estado==1){
 analogWrite(buzz,2864);
 delay(100);
 analogWrite(buzz,1000),
 delay(100);
 }else{
  analogWrite(buzz,0);
 }}

// FUNCION BUZZER.
void Buzz(int sonido){
 analogWrite(buzz,sonido); //emite 1 sonido
 }



 // FUNCION ACTIVA O DESACTIVA MOSTRAR INFORMACION PID MEDIANTE SERIAL
 void SERIAL_PID(int estado){
 if(estado==1){

 Serial.print("Temperatura:  ");
 Serial.print(Input);
 Serial.print("*C");
 Serial.print("       ");
 Serial.print("Setpoint:  ");
 Serial.print(Setpoint);
 Serial.print("*C");
 Serial.print("       ");
 Serial.print("Ventiladores:  ");
 Serial.print(map(Output,0,255,0,100));
 Serial.print("%");
 Serial.print("       ");
 Serial.print("Error:  ");
 Serial.print(Error);
 Serial.print("*C");
 Serial.print("\n");
 Serial.print("       ");
 Serial.print("\n");
 }
 }
  void CODIGO_LCD(int estado){
   if(estado==1){

     lcd.setCursor(1,1);
     lcd.print("T:");
     lcd.print(t);
     lcd.print("\337C");
     lcd.setCursor(9,1);
     lcd.print("HR:");
     lcd.print(h);
     lcd.print("%");
   }}


  void PERSIANA_ARRIBA(){
    digitalWrite(E1, HIGH);    // Activamos Motor1
    digitalWrite(I1, HIGH);    // Arrancamos
    digitalWrite(I2, LOW);
   delay(100);
  }
```

```
 void PERSIANA_ABAJO(){
     digitalWrite(E1, HIGH);    // Activamos Motor1
     digitalWrite(I1, LOW);     // Arrancamos con cambio de dirección
     digitalWrite(I2, HIGH);
     delay(100);
 }

 void PERSIANA_ESTOP(){
  digitalWrite(E1, LOW);
 }

void PERSIANA_AUTO(int estado){
 if(estado==1){
 int luz =analogRead(SensorLuz);

if((luz<=200)&& (Persiana>=40)){
  PERSIANA_ABAJO();
   delay(5000);
  PERSIANA_ESTOP();
  Persiana=0;
}
if((luz>750)&&(Persiana<=1)){
  PERSIANA_ARRIBA();
   delay(5000);
  PERSIANA_ESTOP();
  Persiana=50;
}
}}
void Persiana_Alarma(){
   if(Persiana!=0){
     Persiana=0;
     PERSIANA_ABAJO();
     delay(5000);
     PERSIANA_ESTOP();

   }}

//FUNCION DE LECTURA DE GASES
void Gas(int estado){
 if(estado==1){
 int Lectura=0;
 Lectura=analogRead(SensorGas);
 if(Lectura>=500){
  Buzz(5000);
 }}}
```

TIP120/121/122
TIP125/126/127

COMPLEMENTARY SILICON POWER DARLINGTON TRANSISTORS

- STMicroelectronics PREFERRED SALESTYPES

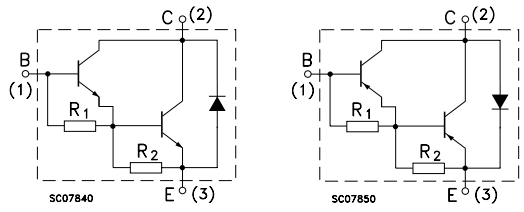


TO-220

DESCRIPTION

The TIP120, TIP121 and TIP122 are silicon Epitaxial-Base NPN power transistors in monolithic Darlington configuration mounted in Jedec TO-220 plastic package. They are intented for use in power linear and switching applications. The complementary PNP types are TIP125, TIP126 and TIP127, respectively.

INTERNAL SCHEMATIC DIAGRAM



R_1 Typ. = 5 KΩ R_2 Typ. = 150 Ω

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter		Value			Unit
		NPN	TIP120	TIP121	TIP122	
		PNP	TIP125	TIP126	TIP127	
V_{CBO}	Collector-Base Voltage (I_E = 0)		60	80	100	V
V_{CEO}	Collector-Emitter Voltage (I_B = 0)		60	80	100	V
V_{EBO}	Emitter-Base Voltage (I_C = 0)		5			V
I_C	Collector Current		5			A
I_{CM}	Collector Peak Current		8			A
I_B	Base Current		0.1			A
P_{tot}	Total Dissipation at $T_{case} \leq 25\ ^oC$ $T_{amb} \leq 25\ ^oC$		65 2			W W
T_{stg}	Storage Temperature		-65 to 150			^oC
T_j	Max. Operating Junction Temperature		150			^oC

* For PNP types voltage and current values are negative.

THERMAL DATA

$R_{thj\text{-}case}$	Thermal Resistance Junction-case	Max	1.92	$^oC/W$
$R_{thj\text{-}amb}$	Thermal Resistance Junction-ambient	Max	62.5	$^oC/W$

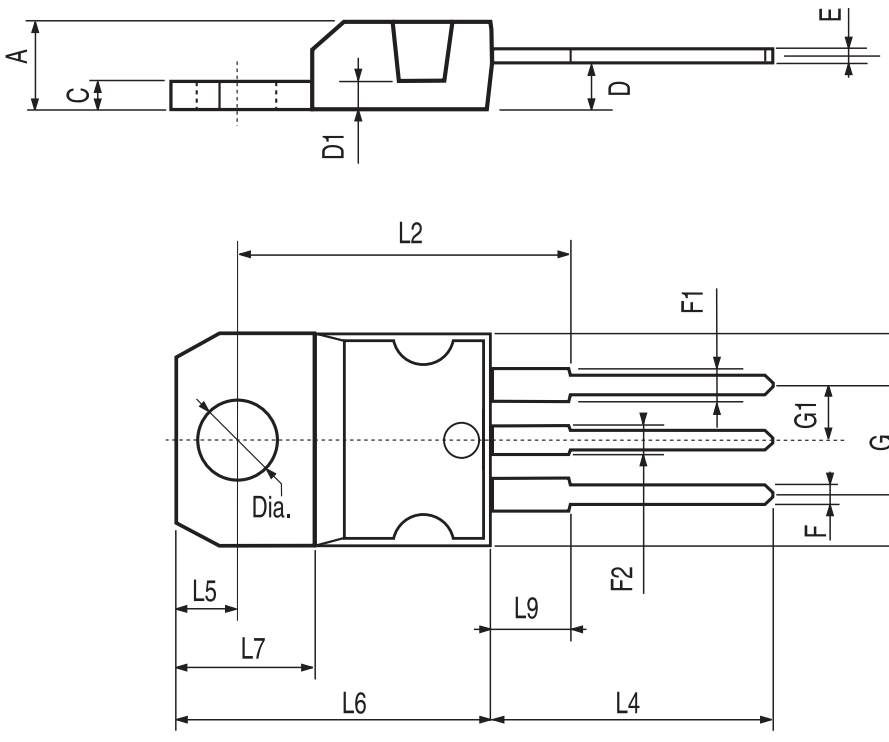
ELECTRICAL CHARACTERISTICS (T_{case} = 25 ^oC unless otherwise specified)

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
I_{CEO}	Collector Cut-off Current (I_B = 0)	for **TIP120/125** for **TIP121/126** for **TIP122/127**	V_{CE} = 30 V V_{CE} = 40 V V_{CE} = 50 V			0.5 0.5 0.5	mA mA mA
I_{CBO}	Collector Cut-off Current (I_B = 0)	for **TIP120/125** for **TIP121/126** for **TIP122/127**	V_{CB} = 60 V V_{CB} = 80 V V_{CB} = 100 V			0.2 0.2 0.2	mA mA mA
I_{EBO}	Emitter Cut-off Current (I_C = 0)	V_{EB} = 5 V				2	mA
$V_{CEO(sus)}$*	Collector-Emitter Sustaining Voltage (I_B = 0)	I_C = 30 mA for **TIP120/125** for **TIP121/126** for **TIP122/127**		 60 80 100			 V V V
$V_{CE(sat)}$*	Collector-Emitter Saturation Voltage	I_C = 3 A I_C = 5 A	I_B = 12 mA I_B = 20 mA			2 4	V V
$V_{BE(on)}$*	Base-Emitter Voltage	I_C = 3 A	V_{CE} = 3 V			2.5	V
h_{FE}*	DC Current Gain	I_C = 0.5 A I_C = 3 A	V_{CE} = 3 V V_{CE} = 3 V	1000 1000			

∗ Pulsed: Pulse duration = 300 µs, duty cycle < 2 %
For PNP types voltage and current values are negative.

TO-220 MECHANICAL DATA

DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A	4.40		4.60	0.173		0.181
C	1.23		1.32	0.048		0.051
D	2.40		2.72	0.094		0.107
D1		1.27			0.050	
E	0.49		0.70	0.019		0.027
F	0.61		0.88	0.024		0.034
F1	1.14		1.70	0.044		0.067
F2	1.14		1.70	0.044		0.067
G	4.95		5.15	0.194		0.203
G1	2.4		2.7	0.094		0.106
H2	10.0		10.40	0.393		0.409
L2		16.4			0.645	
L4	13.0		14.0	0.511		0.551
L5	2.65		2.95	0.104		0.116
L6	15.25		15.75	0.600		0.620
L7	6.2		6.6	0.244		0.260
L9	3.5		3.93	0.137		0.154
DIA.	3.75		3.85	0.147		0.151

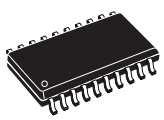


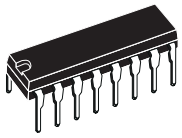
P011C

L293D
L293DD

PUSH-PULL FOUR CHANNEL DRIVER WITH DIODES

- 600mA OUTPUT CURRENT CAPABILITY PER CHANNEL
- 1.2A PEAK OUTPUT CURRENT (non repetitive) PER CHANNEL
- ENABLE FACILITY
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)
- INTERNAL CLAMP DIODES





SO(12+4+4) **Powerdip (12+2+2)**

ORDERING NUMBERS:

L293DD L293D

DESCRIPTION

The Device is a monolithic integrated high voltage, high current four channel driver designed to accept standard DTL or TTL logic levels and drive inductive loads (such as relays solenoides, DC and stepping motors) and switching power transistors.

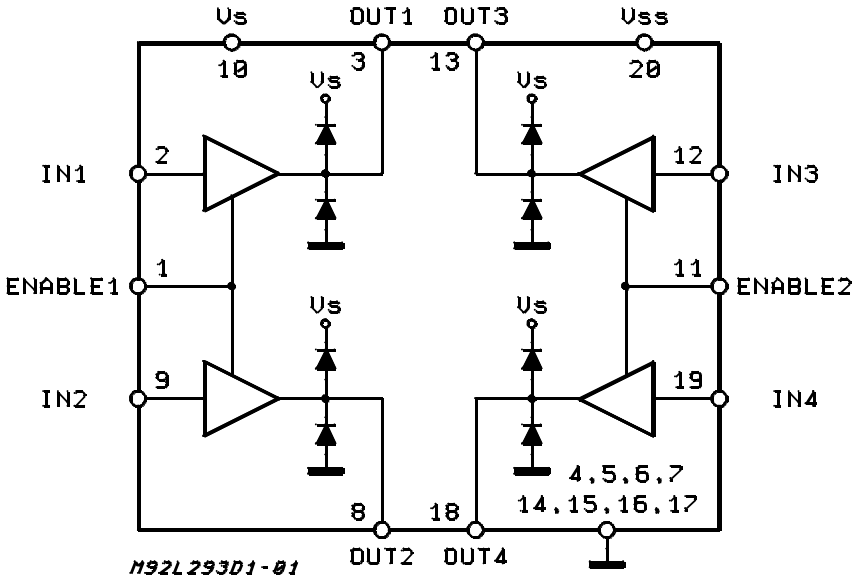
To simplify use as two bridges each pair of channels is equipped with an enable input. A separate supply input is provided for the logic, allowing operation at a lower voltage and internal clamp diodes are included.

This device is suitable for use in switching applications at frequencies up to 5 kHz.

The L293D is assembled in a 16 lead plastic packaage which has 4 center pins connected together and used for heatsinking

The L293DD is assembled in a 20 lead surface mount which has 8 center pins connected together and used for heatsinking.

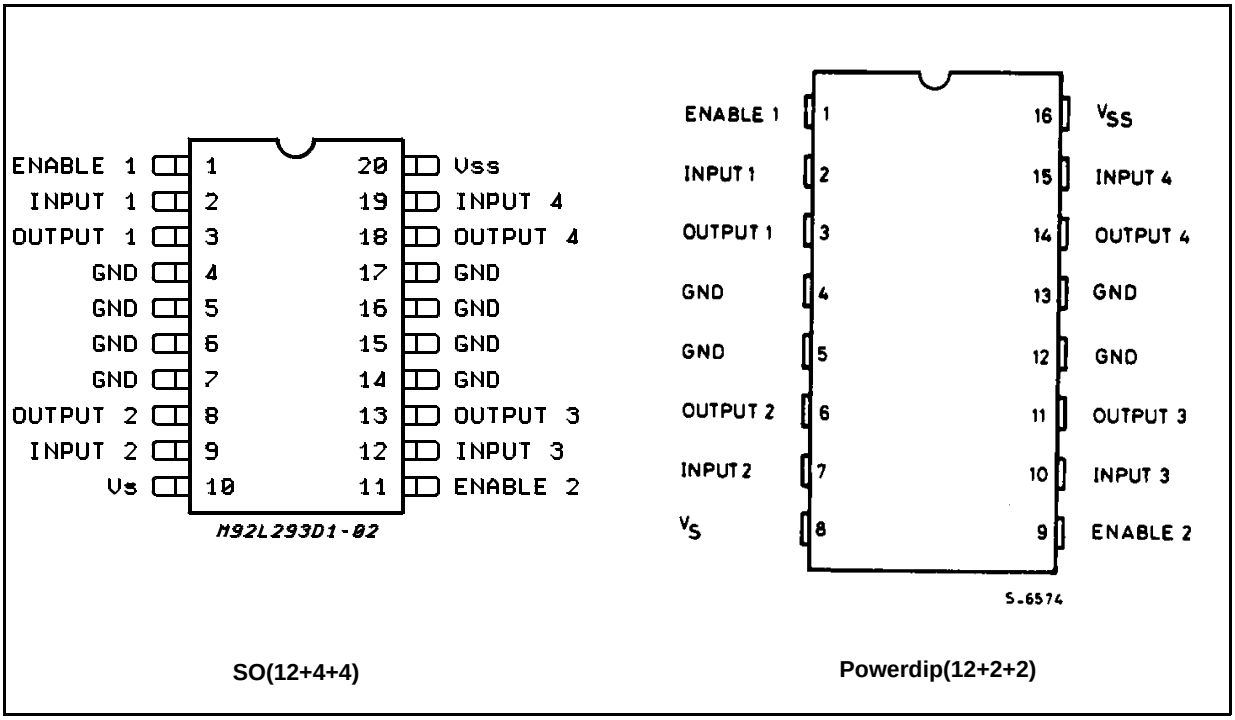
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Supply Voltage	36	V
V_{SS}	Logic Supply Voltage	36	V
V_i	Input Voltage	7	V
V_{en}	Enable Voltage	7	V
I_o	Peak Output Current (100 µs non repetitive)	1.2	A
P_{tot}	Total Power Dissipation at T_{pins} = 90 °C	4	W
T_{stg}, T_j	Storage and Junction Temperature	– 40 to 150	°C

PIN CONNECTIONS (Top view)



SO(12+4+4) **Powerdip(12+2+2)**

THERMAL DATA

Symbol	Decription		DIP	SO	Unit
$R_{th\ j\text{-}pins}$	Thermal Resistance Junction-pins	max.	–	14	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance junction-ambient	max.	80	50 (*)	°C/W
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	max.	14	–	

(*) With 6sq. cm on board heatsink.

ELECTRICAL CHARACTERISTICS (for each channel, V_S = 24 V, V_{SS} = 5 V, T_{amb} = 25 °C, unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 10)		V_{SS}		36	V
V_{SS}	Logic Supply Voltage (pin 20)		4.5		36	V
I_S	Total Quiescent Supply Current (pin 10)	V_i = L ; I_O = 0 ; V_{en} = H		2	6	mA
		V_i = H ; I_O = 0 ; V_{en} = H		16	24	mA
		V_{en} = L			4	mA
I_{SS}	Total Quiescent Logic Supply Current (pin 20)	V_i = L ; I_O = 0 ; V_{en} = H		44	60	mA
		V_i = H ; I_O = 0 ; V_{en} = H		16	22	mA
		V_{en} = L		16	24	mA
V_{IL}	Input Low Voltage (pin 2, 9, 12, 19)		– 0.3		1.5	V
V_{IH}	Input High Voltage (pin 2, 9, 12, 19)	$V_{SS} \leq 7$ V	2.3		V_{SS}	V
		$V_{SS} > 7$ V	2.3		7	V
I_{IL}	Low Voltage Input Current (pin 2, 9, 12, 19)	V_{IL} = 1.5 V			– 10	µA
I_{IH}	High Voltage Input Current (pin 2, 9, 12, 19)	2.3 V ≤ V_{IH} ≤ V_{SS} – 0.6 V		30	100	µA
$V_{en\ L}$	Enable Low Voltage (pin 1, 11)		– 0.3		1.5	V
$V_{en\ H}$	Enable High Voltage (pin 1, 11)	$V_{SS} \leq 7$ V	2.3		V_{SS}	V
		$V_{SS} > 7$ V	2.3		7	V
$I_{en\ L}$	Low Voltage Enable Current (pin 1, 11)	$V_{en\ L}$ = 1.5 V		– 30	– 100	µA
$I_{en\ H}$	High Voltage Enable Current (pin 1, 11)	2.3 V ≤ $V_{en\ H}$ ≤ V_{SS} – 0.6 V			± 10	µA
$V_{CE(sat)H}$	Source Output Saturation Voltage (pins 3, 8, 13, 18)	I_O = – 0.6 A		1.4	1.8	V
$V_{CE(sat)L}$	Sink Output Saturation Voltage (pins 3, 8, 13, 18)	I_O = + 0.6 A		1.2	1.8	V
V_F	Clamp Diode Forward Voltage	I_O = 600nA		1.3		V
t_r	Rise Time (*)	0.1 to 0.9 V_O		250		ns
t_f	Fall Time (*)	0.9 to 0.1 V_O		250		ns
t_{on}	Turn-on Delay (*)	0.5 V_i to 0.5 V_O		750		ns
t_{off}	Turn-off Delay (*)	0.5 V_i to 0.5 V_O		200		ns

(*) See fig. 1.

TRUTH TABLE (one channel)

Input	Enable (*)	Output
H	H	H
L	H	L
H	L	Z
L	L	Z

Z = High output impedance
(*) Relative to the considered channel

Figure 1: Switching Times

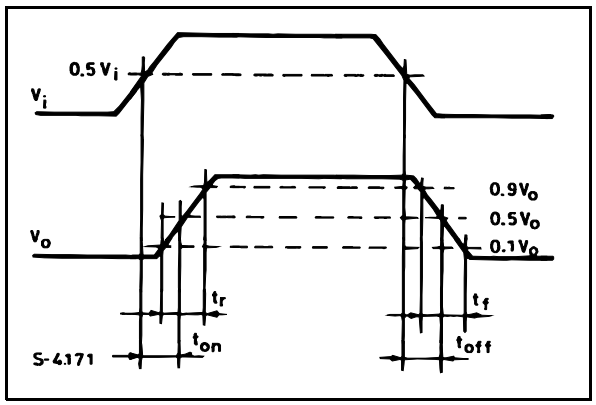
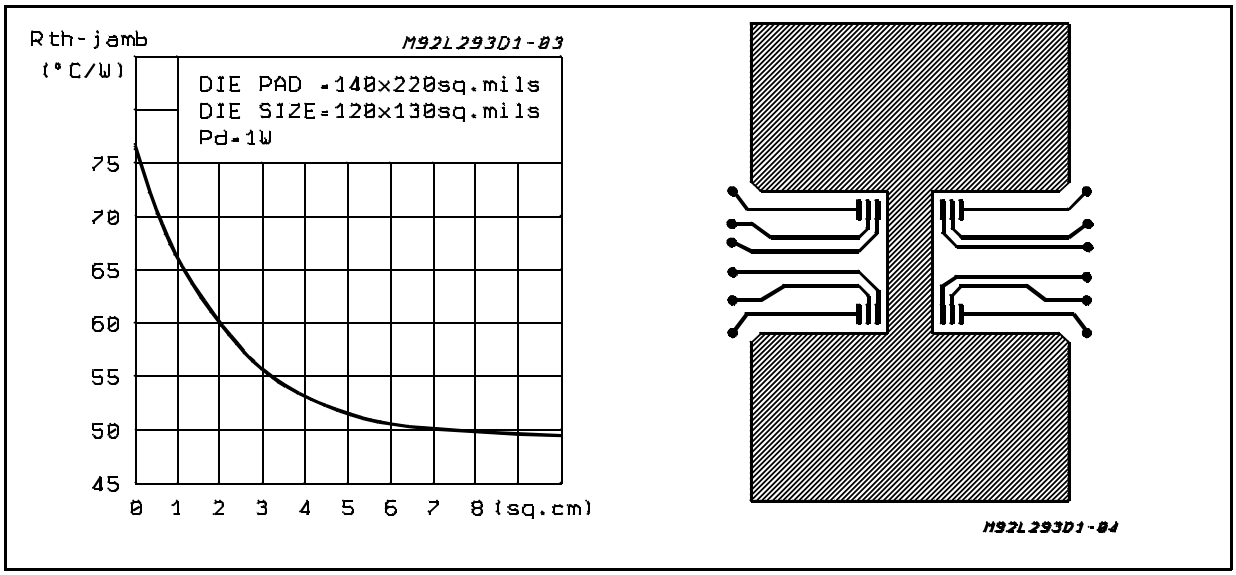
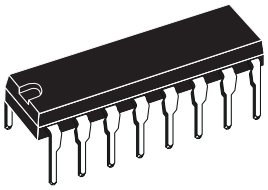


Figure 2: Junction to ambient thermal resistance vs. area on board heatsink (SO12+4+4 package)

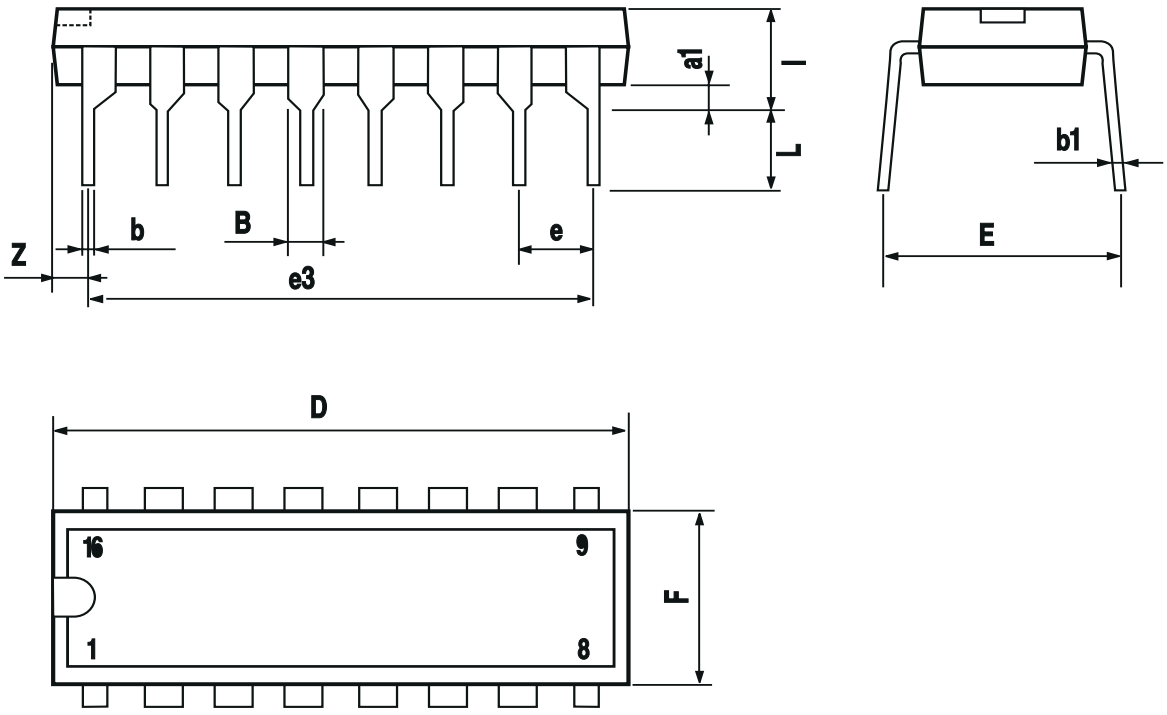


DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
a1	0.51			0.020		
B	0.85		1.40	0.033		0.055
b		0.50			0.020	
b1	0.38		0.50	0.015		0.020
D			20.0			0.787
E		8.80			0.346	
e		2.54			0.100	
e3		17.78			0.700	
F			7.10			0.280
I			5.10			0.201
L		3.30			0.130	
Z			1.27			0.050

OUTLINE AND MECHANICAL DATA

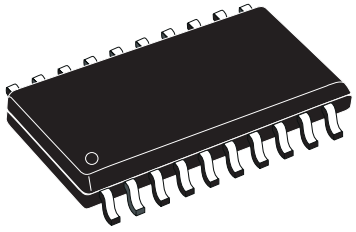


Powerdip 16

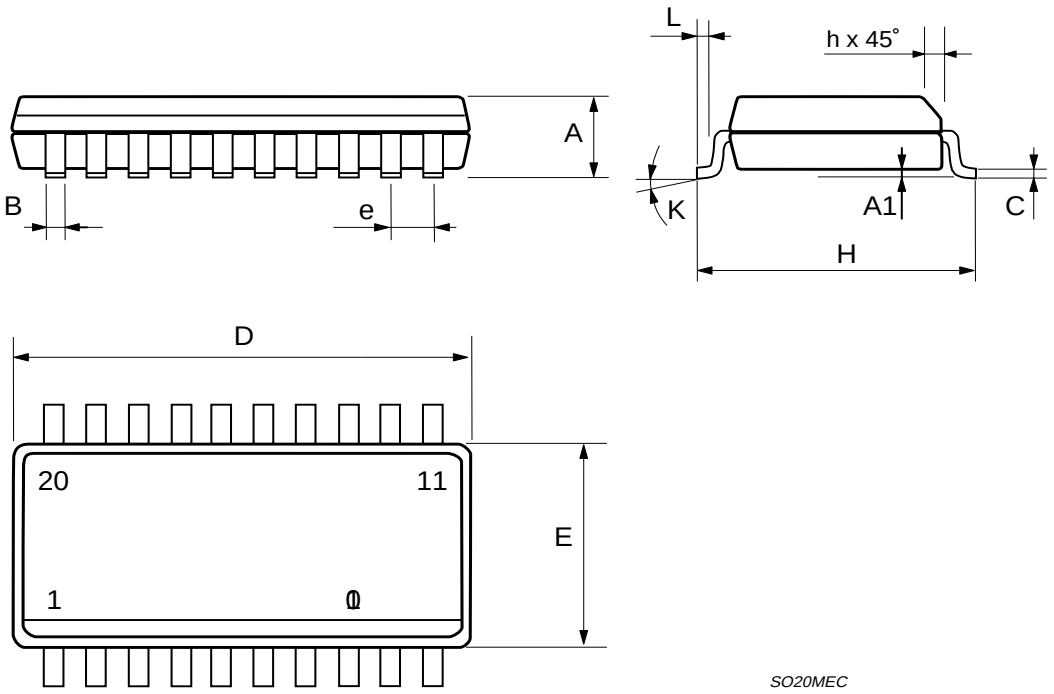


DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A	2.35		2.65	0.093		0.104
A1	0.1		0.3	0.004		0.012
B	0.33		0.51	0.013		0.020
C	0.23		0.32	0.009		0.013
D	12.6		13	0.496		0.512
E	7.4		7.6	0.291		0.299
e		1.27			0.050	
H	10		10.65	0.394		0.419
h	0.25		0.75	0.010		0.030
L	0.4		1.27	0.016		0.050
K	0˚ (min.)8˚ (max.)					

OUTLINE AND MECHANICAL DATA



SO20



SO20MEC

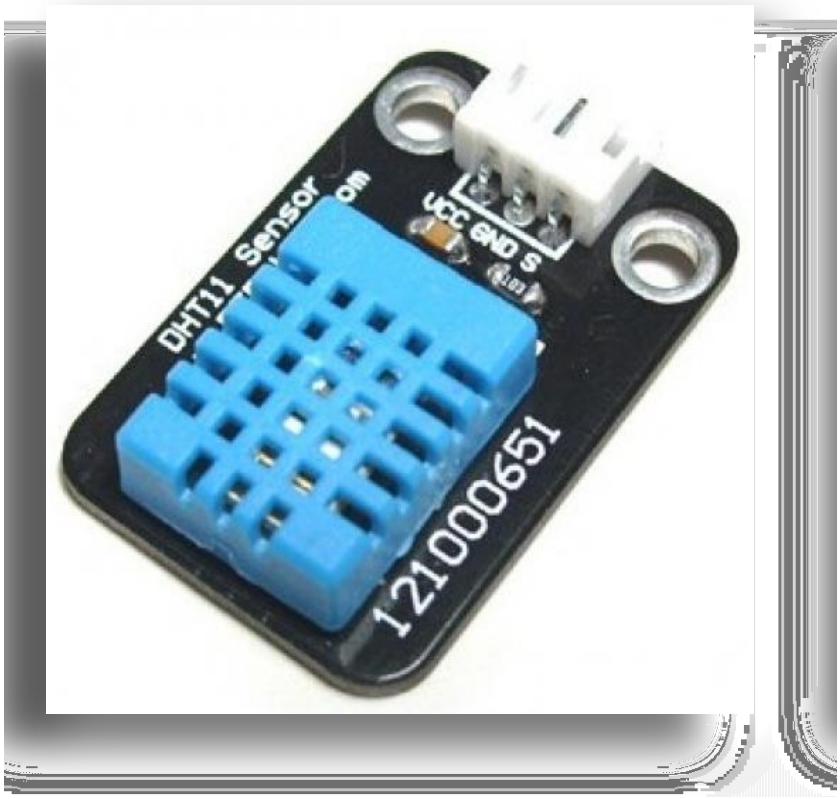
DHT11 Humidity & Temperature Sensor

D-Robotics UK (www.droboticsonline.com)

DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output.

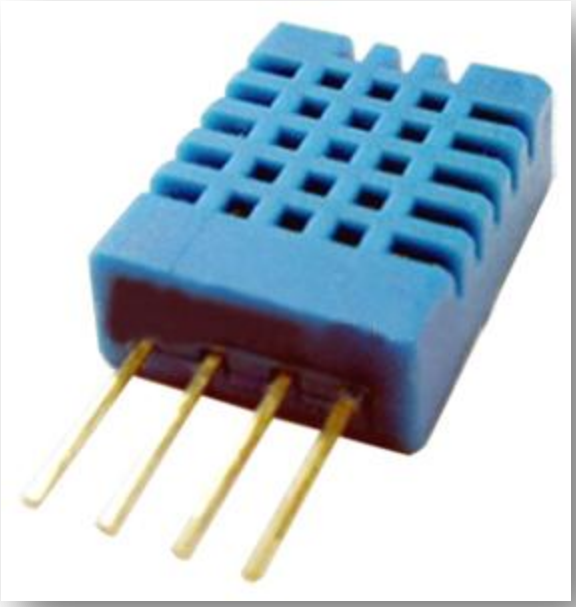
D-Robotics
7/30/2010

DHT 11 Humidity & Temperature Sensor



1. Introduction

This DFRobot DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.



Each DHT11 element is strictly calibrated in the laboratory that is extremely accurate on humidity calibration. The calibration coefficients are stored as programmes in the OTP memory, which are used by the sensor's internal signal detecting process. The single-wire serial interface makes system integration quick and easy. Its small size, low power consumption and up-to-20 meter signal transmission making it the best choice for various applications, including those most demanding ones. The component is 4-pin single row pin package. It is convenient to connect and special packages can be provided according to users' request.

2. Technical Specifications:

Overview:

Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 ℃	±5%RH	±2℃	1	4 Pin Single Row

Detailed Specifications:

Parameters	Conditions	Minimum	Typical	Maximum
Humidity				
Resolution		1%RH	1%RH	1%RH
			8 Bit	
Repeatability			±1%RH	
Accuracy	25℃		±4%RH	
	0-50℃			±5%RH
Interchangeability	Fully Interchangeable			
Measurement Range	0℃	30%RH		90%RH
	25℃	20%RH		90%RH
	50℃	20%RH		80%RH
Response Time (Seconds)	1/e(63%)25℃, 1m/s Air	6 S	10 S	15 S
Hysteresis			±1%RH	
Long-Term Stability	Typical		±1%RH/year	
Temperature				
Resolution		1℃	1℃	1℃
		8 Bit	8 Bit	8 Bit
Repeatability			±1℃	
Accuracy		±1℃		±2℃
Measurement Range		0℃		50℃
Response Time (Seconds)	1/e(63%)	6 S		30 S

3. Typical Application (Figure 1)

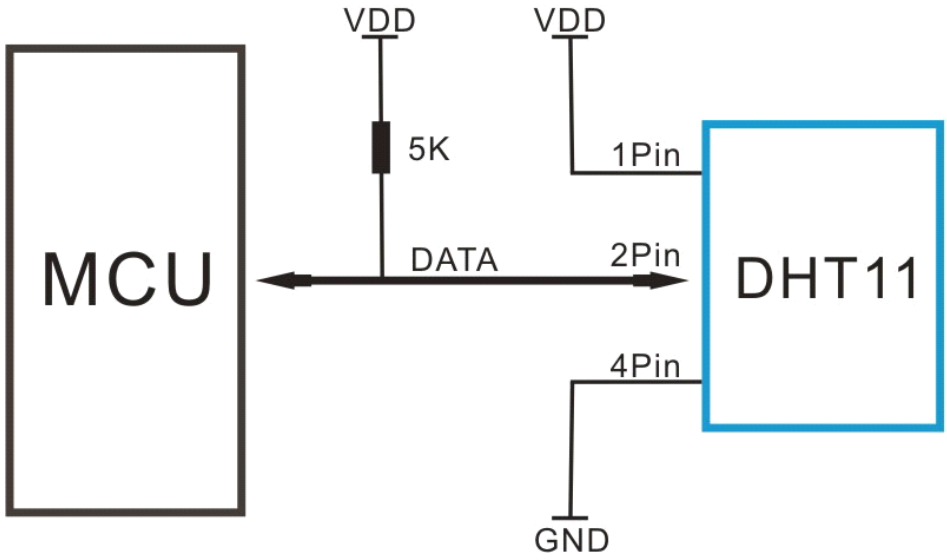


Figure 1 Typical Application

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.

4. Power and Pin

DHT11's power supply is 3-5.5V DC. When power is supplied to the sensor, do not send any instruction to the sensor in within one second in order to pass the unstable status. One capacitor valued 100nF can be added between VDD and GND for power filtering.

5. Communication Process: Serial Interface (Single-Wire Two-Way)

Single-bus data format is used for communication and synchronization between MCU and DHT11 sensor. One communication process is about 4ms.

Data consists of decimal and integral parts. A complete data transmission is **40bit**, and the sensor sends **higher data bit** first.

Data format: 8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum. If the data transmission is right, the check-sum should be the last 8bit of "8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data".

5.1 Overall Communication Process (Figure 2, below)

When MCU sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for MCU completing the start signal. Once it is completed, DHT11 sends a response signal of 40-bit data that include the relative humidity and temperature information to MCU. Users can choose to collect (read) some data. Without the start signal from MCU, DHT11 will not give the response signal to MCU. Once data is collected, DHT11 will change to the low-power-consumption mode until it receives a start signal from MCU again.

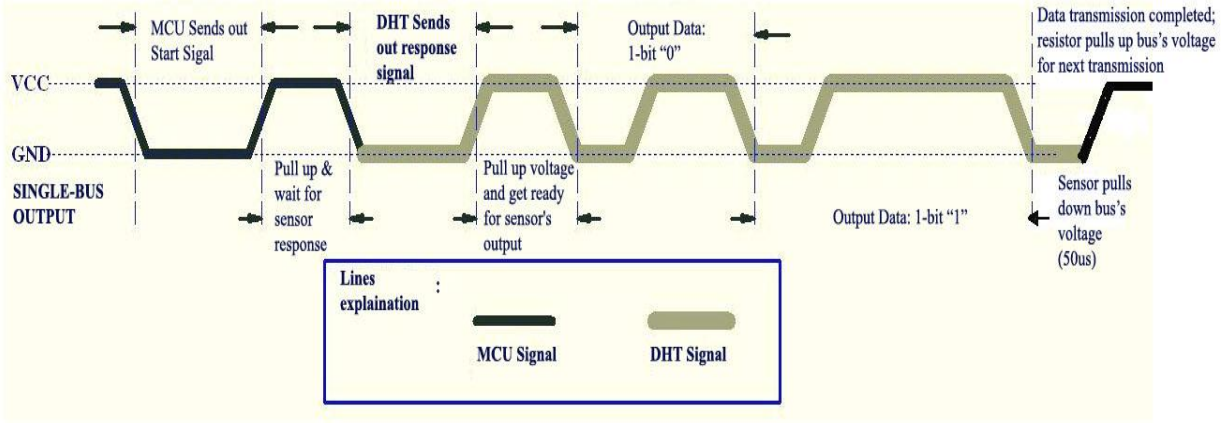


Figure 2 Overall Communication Process

5.2 MCU Sends out Start Signal to DHT (Figure 3, below)

Data Single-bus free status is at high voltage level. When the communication between MCU and DHT11 begins, the programme of MCU will set Data Single-bus voltage level from high to low and this process must take at least 18ms to ensure DHT's detection of MCU's signal, then MCU will pull up voltage and wait 20-40us for DHT's response.

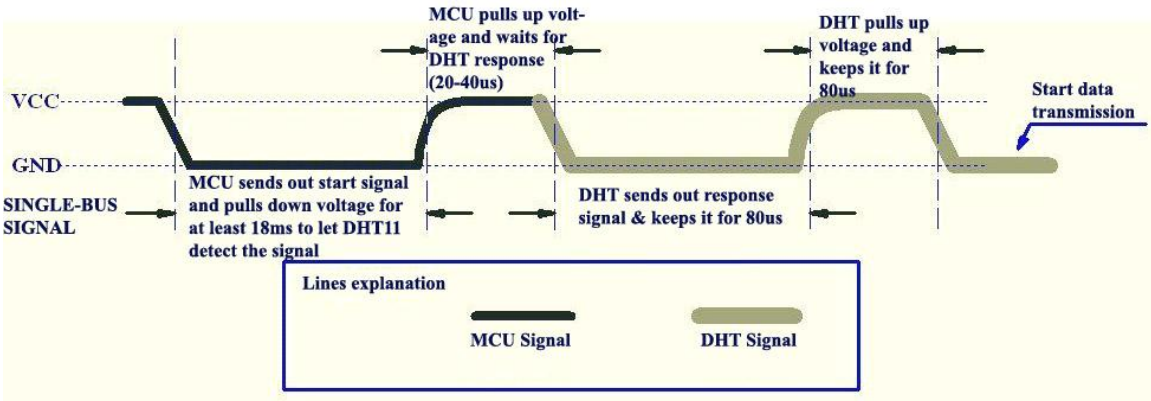


Figure 3 MCU Sends out Start Signal & DHT Responses

5.3 DHT Responses to MCU (Figure 3, above)

Once DHT detects the start signal, it will send out a low-voltage-level response signal, which lasts 80us. Then the programme of DHT sets Data Single-bus voltage level from low to high and keeps it for 80us for DHT's preparation for sending data.

When DATA Single-Bus is at the low voltage level, this means that DHT is sending the response signal. Once DHT sent out the response signal, it pulls up voltage and keeps it for 80us and prepares for data transmission.

When DHT is sending data to MCU, every bit of data begins with the 50us low-voltage-level and the length of the following high-voltage-level signal determines whether data bit is "0" or "1" (see Figures 4 and 5 below).

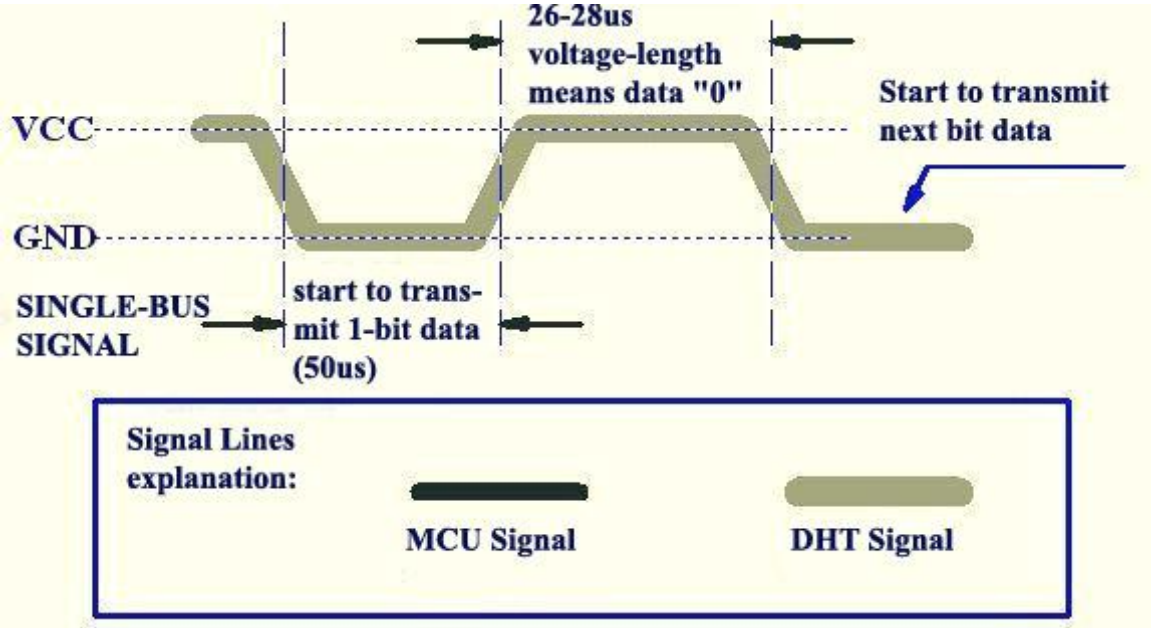


Figure 4 Data "0" Indication

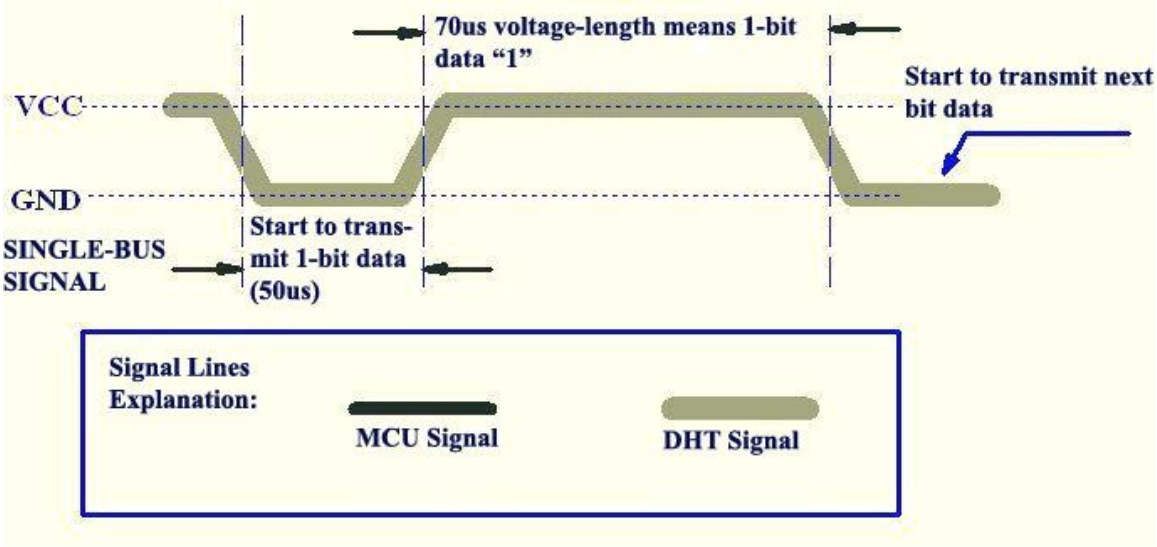


Figure 5 Data "1" Indication

If the response signal from DHT is always at high-voltage-level, it suggests that DHT is not responding properly and please check the connection. When the last bit data is transmitted, DHT11 pulls down the voltage level and keeps it for 50us. Then the Single-Bus voltage will be pulled up by the resistor to set it back to the free status.

6. Electrical Characteristics

VDD=5V, T = 25℃ (unless otherwise stated)

	Conditions	Minimum	Typical	Maximum
Power Supply	DC	3V	5V	5.5V
Current Supply	Measuring	0.5mA		2.5mA
	Average	0.2mA		1mA
	Standby	100uA		150uA
Sampling period	Second	1		

Note: Sampling period at intervals should be no less than 1 second.

7. Attentions of application

(1) Operating conditions

Applying the DHT11 sensor beyond its working range stated in this datasheet can result in 3%RH signal shift/discrepancy. The DHT11 sensor can recover to the calibrated status gradually when it gets back to the normal operating condition and works within its range. Please refer to (3) of

this section to accelerate its recovery. Please be aware that operating the DHT11 sensor in the non-normal working conditions will accelerate sensor's aging process.

(2) Attention to chemical materials

Vapor from chemical materials may interfere with DHT's sensitive-elements and debase its sensitivity. A high degree of chemical contamination can permanently damage the sensor.

(3) Restoration process when (1) & (2) happen

Step one: Keep the DHT sensor at the condition of Temperature 50~60Celsius, humidity <10%RH for 2 hours;
Step two:K keep the DHT sensor at the condition of Temperature 20~30Celsius, humidity >70%RH for 5 hours.

(4) Temperature Affect

Relative humidity largely depends on temperature. Although temperature compensation technology is used to ensure accurate measurement of RH, it is still strongly advised to keep the humidity and temperature sensors working under the same temperature. DHT11 should be mounted at the place as far as possible from parts that may generate heat.

(5) Light Affect

Long time exposure to strong sunlight and ultraviolet may debase DHT's performance.

(6) Connection wires

The quality of connection wires will affect the quality and distance of communication and high quality shielding-wire is recommended.

(7) Other attentions

* Welding temperature should be bellow 260Celsius and contact should take less than 10 seconds.
* Avoid using the sensor under dew condition.
* Do not use this product in safety or emergency stop devices or any other occasion that failure of DHT11 may cause personal injury.
* Storage: Keep the sensor at temperature 10-40℃, humidity <60%RH.

Declaim:

This datasheet is a translated version of the manufacturer's datasheet. Although the due care has been taken during the translation, D-Robotics is not responsible for the accuracy of the information contained in this document.

D-Robotics: www.droboticsonline.com

Email contact: d_robotics@hotmail.co.uk

TECHNICAL DATA MQ-2 GAS SENSOR

FEATURES

Wide detecting scope — Fast response and High sensitivity
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, i-butane, propane, methane ,alcohol, Hydrogen, 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
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	-20℃-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	3KΩ-30KΩ (1000ppm iso-butane)	Detecting concentration scope: 200ppm-5000ppm LPG and propane 300ppm-5000ppm butane 5000ppm-20000ppm methane 300ppm-5000ppm H_2 100ppm-2000ppm Alcohol
α (3000/1000) isobutane	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

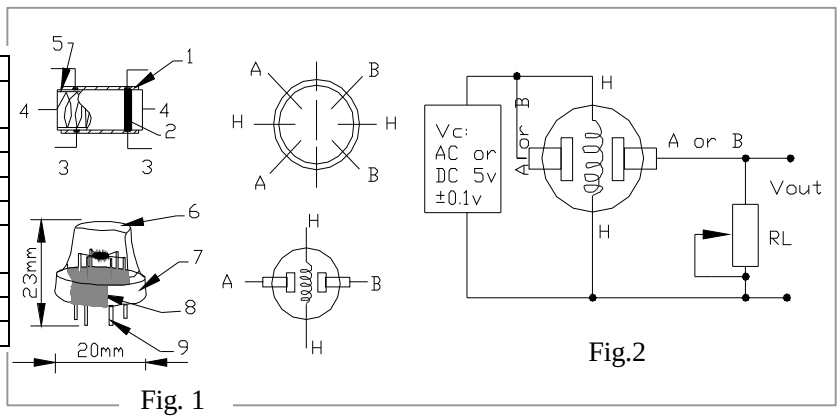
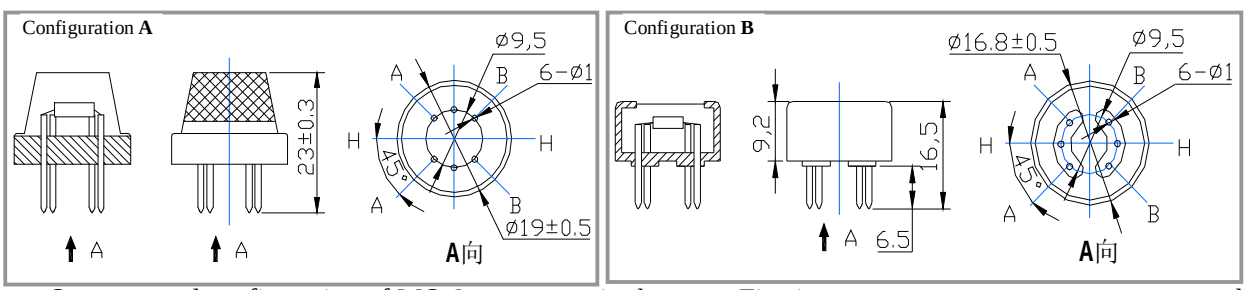


Fig. 1 Fig.2



Structure and configuration of MQ-2 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-2 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

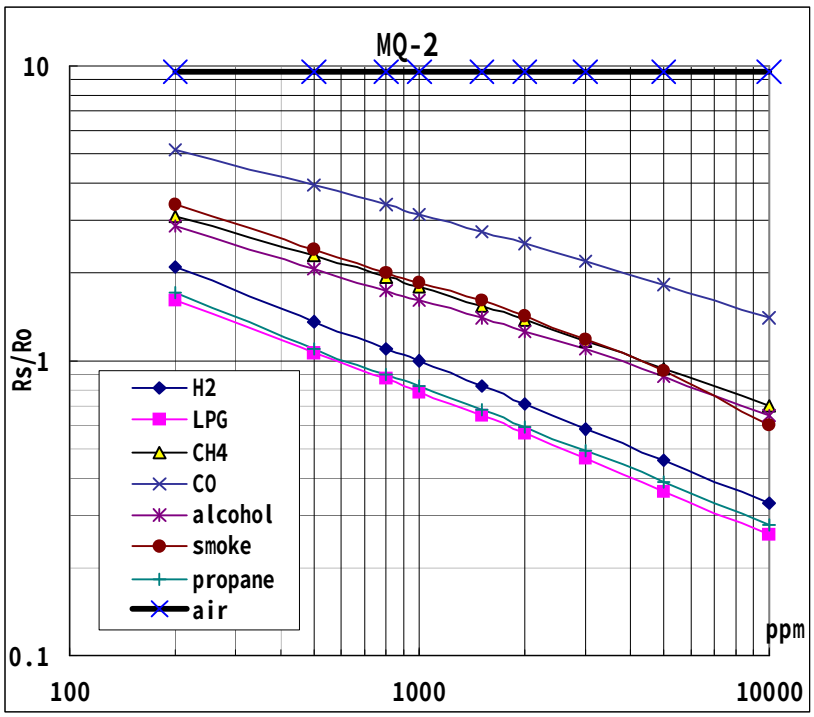


Fig.2 sensitivity characteristics of the MQ-2

Fig.3 is shows the typical sensitivity characteristics of the MQ-2 for several gases.
in their: Temp: 20℃、
Humidity: 65%、
O_2 concentration 21%
RL=5kΩ
Ro: sensor resistance at 1000ppm of H_2 in the clean air.
Rs:sensor resistance at various concentrations of gases.

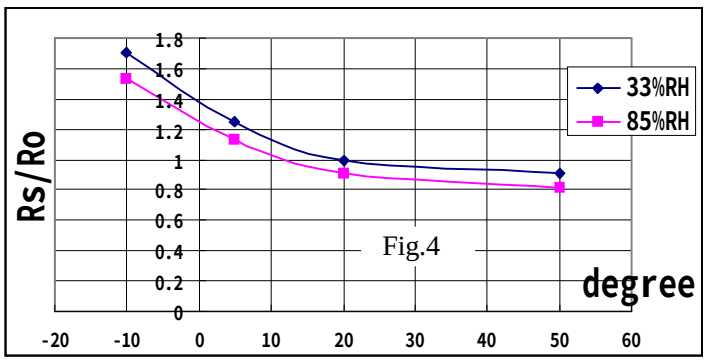


Fig.4

Fig.4 is shows the typical dependence of the MQ-2 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 at different temperatures and humidities.

SENSITVITY ADJUSTMENT

Resistance value of MQ-2 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 liquified petroleum gas<LPG>,or 1000ppm iso-butane<i-C_4H_{10}>concentration in air and use value of Load resistance that(R_L) about 20 KΩ (5KΩ to 47 KΩ).

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

ABC Proyectos electrónicos (2014). Sensor de humedad y temperatura DHTxxx. [En línea] Disponible en: http://www.abcelectronica.net/proyectos/con-microcontrolador/dht1122/ [Acceso 12 mayo 2016].

Adafruit (2014). Using a PIR. [En línea] Disponible en: https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/using-a-pir [Acceso 4 mayo 2016].

AENOR (2010). Norma UNE-EN 50491-5-2:2010. [En línea] Disponible en: http://www.aenor.es/aenor/normas/normas/fichanorma.asp?tipo=N&codigo=N0046285#.V2A_WNSLRkh [Acceso 8 junio 2016].

AENOR (2011). Norma UNE-EN 50090-1:2011. [En línea] Disponible en: http://www.aenor.es/aenor/normas/normas/fichanorma.asp?tipo=N&codigo=N0048154#.V06y_vmLTcc [Acceso 8 junio 2016].

AENOR (2013). Publicación de la versión europea de la EA0026. [En línea] Disponible en: http://www.aenor.es/aenor/actualidad/actualidad/noticias.asp?campo=1&codigo=28632#.V17AVdSLRkh [Acceso 20 mayo 2016]

Alldatasheet (2016). L293D Datasheet, PDF. [En línea] Disponible en: http://www.alldatasheet.com/view.jsp?Searchword=L293d%20datasheet&gclid=CPDwnILHo80CFfMV0wodUfIPHg [Acceso 4 junio 2016].

Analog Devices (2010). Low Voltage Temperature Sensors. [En línea] Disponible en: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Temp/TMP35_36_37.pdf [Acceso 25 mayo 2016].

Arduino (2016). Arduino UNO & Genuino UNO. [En línea] Disponible en: https://www.arduino.cc/en/Main/ArduinoBoardUno [Acceso 30 abril 2016].

Arduino. (2016). Arduino MEGA 2560 & Genuino MEGA 2560. [En línea] Disponible en: https://www.arduino.cc/en/Main/ArduinoBoardMega2560 [Acceso abril 2016].

Asociación Española de Domótica e Inmótica – CEDOM (2016). Tabla de niveles para evaluación de instalaciones domóticas. [En línea] Disponible en: http://www.aenor.es/aenor/actualidad/actualidad/noticias.asp?campo=1&codigo=28632#.V2A9itSLRki [Acceso 8 junio 2016].

Aulaclic (2005). WIFI - Comunicación Inalámbrica. [En línea] Disponible en: http://www.aulaclic.es/articulos/wifi.html [Acceso 8 junio 2016].

Beauregard, Brett (2011). Arduino PID - Guía de uso de la librería. [En línea] Disponible en: http://brettbeauregard.com/blog/wp-content/uploads/2012/07/Gu%C3%ADa-de-uso-PID-para-Arduino.pdf [Acceso 14 mayo 2016].

Blog Redes (2010). Tipos de cable. [En línea] Disponible en: https://pondalpar113.wordpress.com/tipos-de-cable/ [Acceso 15 mayo 2016].

Caffelli, Paolo (2010). ¿Qué es y cómo funciona el sistema Bluetooth? [En línea] Disponible en: http://etecnologia.com/gadgets/funcionamiento-bluetooth [Acceso 25 mayo2016].

Casadomo. Todo sobre edificios inteligentes (2004). Domótica Inmótica. Introdomótica. [En línea] Disponible en: https://www.casadomo.com/noticias/-2185 [Acceso 6 junio 2016].

Casadomo. Todo sobre edificios inteligentes (2014). I Congreso EI: Marco legislativo de los sistemas domóticos e inmóticos. [En línea] Disponible en: https://www.casadomo.com/comunicaciones/i-congreso-ei-marco-legislativo-de-los-sistemas-domoticos-e-inmoticos [Acceso 8 junio 2016].

CCM (2016). Transmisión de datos: Cableado. [En línea] Disponible en: http://es.ccm.net/contents/685-transmision-de-datos-cableado [Acceso 21 mayo 2016].

Díaz Puerta, Andoni (2011). Librería PID con Arduino. [En línea] Disponible en: http://blog.bricogeek.com/noticias/arduino/libreria-pid-con-arduino/ [Acceso 4 abril 2016].

Domoprac (2016). Protocolos de Red: Tipos y Utilidades - DomoPrac - Domotica practica paso a paso. [En línea]. Disponible en: http://www.domoprac.com/protocolos-de-comunicacion-y-sistemas-domoticos/protocolos-de-red-tipos-y-utilidades.html [Acceso 18 abril 2016].

Domótica 365 (2016). Ventajas e inconvenientes de la domótica. [En línea] Disponible en: http://www.domotica365.com/articulos/ventajas-e-inconvenientes-de-la-domotica [Acceso 2 mayo 2016].

D-Robotics UK (2010). DHT11 Humidity & Temperature Sensor DHT11 Humidity & Temperature Sensor. [En línea] Disponible en: http://www.micropik.com/PDF/dht11.pdf [Acceso 10 mayo 2016].

EDCA. El cajón de Ardu (2014). Tutorial: sensor ultrasonidos HC-SR04. [En línea] Disponible en: http://elcajondeardu.blogspot.com.es/2014/03/tutorial-sensor-ultrasonidos-hc-sr04.html [Acceso 12 mayo 2016].

Elecom Electronics Supply (2016). Arduino Uno R3 with USB cable_Arduino Boards_Arduino_Australia, Elecom Electronics Supply. [En línea] Disponible en: http://www.elecomes.com/goods.php?id=173 [Acceso 15 mayo 2016].

Gabriel Rivemar, Alfredo (2013). Tabla comparativa de Arduinos. [En línea] Disponible en: http://sabetecnologia.blogspot.com.es/2013/03/tabla-comparativa-de-arduinos.html [Acceso 11 abril 2016].

García González, Antony (2014). DHT22: Sensor de humedad/temperatura de precisión para Arduino. [En línea] Disponible en: http://panamahitek.com/dht22-sensor-de-humedadtemperatura-de-precision-para-arduino/ [Acceso 20 mayo 2016].

García, Pamela (2013). [Blog] ¿Qué es el control PID? [En línea] Disponible en: https://franklinelinkmx.wordpress.com/2013/09/05/que-es-el-control-pid/ [Acceso 25 abril 2016].

Google Site (2016). Arduino - Diseño y Manufactura. [En línea] Disponible en: https://sites.google.com/site/temasdedisenoymanufactura/arduino [Acceso 15 mayo 2016]

Grupo de Trabajo IFLICA (2006). Instalación Física y lógica de una red cableada e inalámbrica en un aula. [En línea] Disponible en: http://informatica.iescuravalera.es/mod/resource/view.php?id=257 [Acceso 4 mayo 2016].

Gutiérrez, Manuel J (2015). Todo sobre ZigBee, la tecnología barata para comunicación inalámbrica. [En línea] Disponible en: http://www.elandroidelibre.com/2015/08/todo-sobre-zigbee-la-tecnologia-ultrabarata-para-comunicacion-inalambrica.html [Acceso 25 abril 2016].

Haibrain (2016). X10 Protocol - X10 Home Automation and Security solutions. [En línea] Disponible en: http://www.haibrain.com/informatie-domotica-x10-protocol-c-131_148_141.html?language=en [Acceso 29 abril 2016].

Huarí E, Félix (2001). Tecnología XDSL para comunicaciones. [En línea] Disponible en: http://sisbib.unmsm.edu.pe/bibvirtual/publicaciones/indata/v04_n1/tecnologia.htm [Acceso 25 mayo 2016].

Igor R.(2014).Maqueta de control PID con arduino [En línea] Disponible en: http://real2electronics.blogspot.com.es/2011/07/maqueta-de-control-pid-con-arduino.html[Acceso 16 mayo 2016].

Innergy (2016). GAS NATURAL - ¿Qué es la gas natural? [En línea] Disponible en: http://www.innergy.cl/quees.htm [Acceso 4 mayo 2016].

Lazaridis, Giorgos (2011). PID Theory. [En línea] Disponible en: http://www.pcbheaven.com/wikipages/PID_Theory/ [Acceso 14 mayo 2016]

Llamas, Luis (2015). Detector de movimiento con arduino y sensor PIR. [En línea] Disponible en: http://www.luisllamas.es/2015/07/detector-de-movimiento-con-arduino-y-sensor-pir/ [Acceso 3 mayo 2016].

Llamas, Luis (2015). Medir distancia con Arduino y sensor de ultrasonidos HC-SR04. [En línea] Disponible en: http://www.luisllamas.es/2015/06/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/ [Acceso 10 mayo 2016].

Llamas, Luis (2015). Medir temperatura con Arduino y sensor LM35. [En línea] Disponible en: http://www.luisllamas.es/2015/07/medir-temperatura-con-arduino-y-sensor-lm35/ [Acceso 6 abril 2016].

Marian, P (2012). LM35 Datasheet. [En línea] Disponible en: http://www.electroschematics.com/6393/lm35-datasheet/ [Acceso 8 mayo 2016].

Mastermagazine (2016). Definición de GPRS. [En línea] Disponible en: http://www.mastermagazine.info/termino/5172.php [Acceso 9 mayo 2016].

Morocho Maita, Luis Andrés (2014). Sistema de Comunicación Domótico X-10. [En línea] Disponible en: http://www.monografias.com/trabajos99/sistema-domotico-x-10/sistema-domotico-x-10.shtml [Acceso 12 mayo 2016].

Naula Dutan, Byron (2014). Domótica mediante Protocolo X10. [En línea] Disponible en: http://www.monografias.com/trabajos101/domotica-protocolo-x10/domotica-protocolo-x10.shtml#ventajasya [Acceso 30 mayo 2016].

Nexans (2016). Nexans breaks new ground in the Baltic region with xDSL frame agreement for Citrus Solutions. [En línea] Disponible en: http://www.nexans.com/eservice/Corporate-en/navigatepub_142508_-3664/Nexans_breaks_new_ground_in_the_Baltic_region_with.html [Acceso 4 abril 2016].

PE, Isaac. (2014). Comparativa de todas las placas Arduino. [En línea]. Disponible en: http://comohacer.eu/analisis-comparativo-placas-arduino-oficiales-compatibles/ [Acceso x].

Robot Platform (2016). Dual H-bridge Motor Driver - L293D IC [En línea] Disponible en: http://www.robotplatform.com/howto/L293/motor_driver_1.html [Acceso 6 mayo 2016].

Rodriguez, Yurisay (2003). Fibra óptica. [En línea] Disponible en: http://www.monografias.com/trabajos13/fibropt/fibropt.shtml [Acceso 21 abril 2016].

Rosalía (2008). Domótica. [En línea] Disponible en: http://domotica-ortegon.blogspot.com.es/2008_05_01_archive.html [Acceso 2 abril 2016].

Sánchez, Jorge (2015). [Blog] Transmision de Datos por Radiofrecuencia. [En línea] Disponible en: http://blanquitorules.blogspot.com.es/2015/02/transmision-de-datos-por-radiofrecuencia.html [Acceso 10 mayo 2016].

Schneider Electric (2016). KNX Building Control System. [En línea] Disponible en: https://www.schneiderelectric.es/sites/spain/es/productos-servicios/product-launch/knx/how-does-knx-works.page [Acceso 12 abril 2016].

Seeed Wiki (2016). Grove - Gas Sensor(MQ2). [En línea] Disponible en: http://seeedstudio.com/wiki/Twig_-_Gas_Sensor(MQ2) [Acceso 20 abril 2016].

Temporizadores y Sensores (2011). [Blog] Javier temporizadores. [En línea] Disponible en: http://javier-temporizadores.blogspot.com.es/2011/03/sensores-fotoelectricos-un-sensor.html [Acceso 18 mayo 2016].

Tuelectrónica. Telecomunicaciones, electrónica e informática (2012). Resistencia Pull Up y Pull Down. [En línea] Disponible en: http://www.tuelectronica.es/tutoriales/electronica/resistencia-pull-up-y-pull-down.html [Acceso 28 mayo 2016].

Unitel - Soluciones e infraestructuras Tecnológicas (2016). Domótica KNX – más que ventajas. [En línea] Disponible en: http://unitel-tc.com/domotica-knx/ [Acceso 9 mayo 2016].

Wikipedia (2016). Gas licuado del petróleo. [En línea] Disponible en: https://es.wikipedia.org/wiki/Gas_licuado_del_petr%C3%B3leo [Acceso 25 mayo 2016].

Wikipedia (2016). Monóxido de carbono. [En línea] Disponible en: https://es.wikipedia.org/wiki/Mon%C3%B3xido_de_carbono [Acceso 7 mayo 2016].

Zapata, Francisco (2014). [Blog] Clases de programación en Arduino. [En línea] Disponible en: http://clasesdearduinoinem.blogspot.com.es/ [Acceso 7 abril 2016].

Zhengzhou Winsen Electronics Technology Co., Ltd (2014). Toxic Gas Sensor. Manual. [En línea] Disponible en: https://cdn.sparkfun.com/datasheets/Sensors/Biometric/MQ-7%20Ver1.3%20-%20Manual.pdf [Acceso 13 mayo 2016].