Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Ingeniería Industrial

MÁSTER EN INGENIERÍA INDUSTRIAL

ESCUELA DE INGENIERÍAS INDUSTRIALES UNIVERSIDAD DE VALLADOLID

TRABAJO FIN DE MÁSTER

RED INALÁMBRICA ENTRE DISPOSITIVOS ARDUINO/NODEMCU

Autor: D. Óscar Castrillejo García

Tutor: D. Francisco Plaza Pérez

Valladolid, octubre de 2020

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Ingeniería Industrial

MÁSTER EN INGENIERÍA INDUSTRIAL

ESCUELA DE INGENIERÍAS INDUSTRIALES UNIVERSIDAD DE VALLADOLID

TRABAJO FIN DE MÁSTER

RED INALÁMBRICA ENTRE DISPOSITIVOS ARDUINO/NODEMCU

Autor: D. Óscar Castrillejo García

Tutor: D. Francisco Plaza Pérez

Valladolid, octubre de 2020

RESUMEN

Este proyecto consiste en el desarrollo de una red WiFi estándar que los usuarios sin conocimientos avanzados de programación puedan adaptar a sus propios proyectos, enfocados, aunque no de forma exclusiva, en el ámbito de la domótica. Para ello, utilizo dispositivos tanto de software como de hardware libre centrándome en el entorno de desarrollo integrado de Arduino (IDE) y en la placa NodeMCU. Por tanto, desarrollo el código informático que configura tanto la red como la función de cada elemento que la forma. Las aplicaciones de este proyecto están dirigidas tanto a la domótica en que lo baso como a la gestión de cualquier red inalámbrica automatizada de sensores y actuadores.

This project consists of the development of a standard WiFi network that users without advanced programming knowledge can adapt to their own projects, focused, although not exclusively, in the field of home automation. To do this, I use both free software and hardware devices focusing on the Arduino integrated development environment (IDE) and the NodeMCU board. Therefore, I develop the computer code that configures both the network and the function of each element that forms it. The applications of this project are aimed both at home automation on which I base it and at the management of any automated wireless network of sensors and actuators.

AGRADECIMIENTOS

Quiero dar las gracias a las personas sin las cuales esto no habría sido posible:

A Marta, mi pareja, que me ha animado y apoyado durante muchos años.

A mis padres, que lo han hecho durante toda mi vida.

A Francisco Plaza, profesor y tutor de este TFM, que me ha guiado y aportado recursos para llevarlo a cabo.

ÍNDICE

1. Introducción 1
   1.1. Motivación 1
   1.2. Internet de las cosas 1
   1.3. Aplicaciones: Inmótica y domótica 2
2. Estado del arte 5
3. Microcontroladores y elementos de la red 7
   3.1. Arduino Uno 7
      3.1.1. Módulo ESP8266: Conexión WiFi con Arduino Uno 9
   3.2. Raspberry Pi 10
   3.3. NodeMCU 11
   3.4. Justificación del dispositivo elegido 13
   3.5. Sensores y actuadores 13
      3.5.1. Sensores 14
      3.5.2. Actuadores 16
      3.5.3. Otros materiales utilizados 18
4. Programación en NodeMCU 19
   4.1. Pluggin ESP8266 para IDE Arduino 19
   4.2. Instalación de librerías 21
   4.3. Ejemplos de programación para NodeMCU 22
      4.3.1. Parpadeo de un LED 23
      4.3.2. Lectura de la señal analógica de un fotorresistor 24
      4.3.3. Lectura de la señal digital de un PIR 25
      4.3.4. Control de un servomotor 26
   4.4. Conexión WiFi con NodeMCU 27
      4.4.1. Estación 27
      4.4.2. Punto de acceso 29
      4.4.3. Red en malla 30
      4.4.4. Conexión Bluetooth y comparativa 31
   4.5. Listado y descripción de las funciones utilizadas 33
   4.6. Comunicación WiFi entre módulos NodeMCU 35
      4.6.1. Protocolo HTTP 36
      4.6.2. Construcción de los mensajes 36
      4.6.3. Estructura Cliente-Servidor 38
   4.7. Punto de acceso y conexión a red doméstica 42
   4.8. Interfaz red-usuario 47

4.8.1. Comunicación usuario-interfaz-servidor 48
4.8.2. Estructura del Sketch 49
4.9. Conexión remota 54
5. Viabilidad económica y comparativa 57
5.1. Otras aplicaciones 61
6. Conclusión 63
7. Bibliografía 67

1. INTRODUCCIÓN

A lo largo de esta memoria explicaré de forma detallada los pasos seguidos a lo largo de la elaboración de mi trabajo de final de máster. Este consiste en la creación de una red formada por microcontroladores conectados entre sí por medio de conexión WiFi, capaces de recoger información de sensores y de controlar actuadores. A la vez, el usuario de esta red será capaz de visualizar la información del estado de los sensores, así como de controlar los actuadores y de configurar diversos modos de funcionamiento de la red desde cualquier lugar a través de su dispositivo móvil conectado a Internet. Este proyecto se enmarca dentro del software libre, con el uso del entorno de desarrollo integrado (IDE) de Arduino y del hardware libre por la utilización de la placa NodeMCU.

El objetivo es crear una red estándar, esto es, que una vez programadas unas funciones básicas sea capaz de adaptarse a distintas necesidades del usuario, por ejemplo, en el número o tipo de sensores o actuadores que configuren dicha red. Para aplicar esta red a una función práctica concreta, he elegido un ejemplo de automatización del hogar, es decir, domótica. Como he explicado anteriormente, el objetivo no es automatizar una casa concreta con determinados sensores y actuadores ya establecidos, sino elaborar una red adaptable a las necesidades de cada hogar o usuario.

Para este propósito he utilizado como elemento de control de la red la placa de desarrollo NodeMCULoLin V3. Esta placa será capaz de monitorizar la señal de los sensores y de controlar los actuadores a través de sus pines de propósito general. Pero la característica por la que destaca es que está basada en el SoC ESP8266, por lo que permite la conectividad WiFi. Además, su económico precio y el hecho de que sea una plataforma de código abierto han hecho que esta placa sea mi elección para el proyecto frente a otros dispositivos estudiados.

1.1. Motivación

Pese a que mi especialidad es la rama de Mecánica, el mundo de la automatización ha resultado de mi interés. Por una parte, la automatización de objetos inanimados, de modo que a partir de una programación basada en la lógica sean capaces de realizar cambios físicos a partir de las distintas señales que captan de su entorno, esto es, que unas líneas de texto no "se queden en el papel", sino que sean capaces de interactuar con el mundo físico. Por otra parte, la similitud entre el sistema cerebro-sentido-músculo y el sistema controlador-sensor-actuador me hace creer que la tecnología actual es el punto de partida para un cambio mucho mayor.

Desde un punto de vista más pragmático, poder realizar un Trabajo Fin de Máster dentro del ámbito de la electrónica me permite complementar mis estudios en mecánica, ya que considero conveniente la formación en ambas ramas de un ingeniero. A la vez, este proyecto tiene una gran parte práctica, con dispositivos económicos, lo que hace posible que pueda adaptarse a las necesidades concretas del día a día.

1.2. Internet de las cosas

Este proyecto se enmarca en el concepto del Internet de las cosas (en inglés, *Internet of Things*, IoT). Este concepto implica la extensión de la conectividad de Internet aplicada a dispositivos físicos y objetos cotidianos. Estos objetos, gracias a tener integradas la electrónica que les permite la recogida de datos mediante sensores y la conectividad a internet, pueden comunicarse e interactuar entre ellos, así como ser monitorizados y manejados de forma remota.

El Internet de las cosas ha podido darse gracias a la convergencia de distintas tecnologías, como redes de sensores, sistemas de control y automatización o circuitos integrados. Para el usuario de a pie, el Internet de las cosas tiene aplicación dentro del llamado "hogar inteligente", donde podrá manejar desde su dispositivo móvil, de forma remota, diferentes aparatos de su hogar, como la iluminación, el termostato o la seguridad y alarmas.

Pero el Internet de las cosas no se limita al hogar, sino que su capacidad de conexión va más allá. Tiene aplicaciones en cada industria, desde la agricultura y ganadería, donde el agricultor podrá monitorizar las condiciones de su plantación como la humedad y la temperatura y actuar sobre ello modificando el nivel del termostato o el caudal de agua, hasta la automoción, dentro del desarrollo del coche autónomo.

Así, el ámbito del Internet de las cosas no se circunscribe dentro de cada industria, sino que tiene como objetivo ser una red de redes, esto es, relacionar las redes del hogar con las del transporte, la industria o la energía y a su vez entre ellas, permitiendo recoger, analizar y distribuir información haciendo a cada industria pueda cubrir las necesidades de la población de una forma más eficiente.

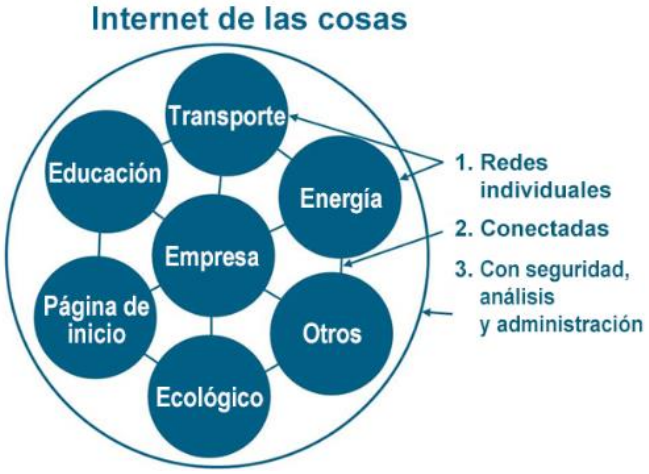


Figura 1: Ámbitos de aplicación de IoT

1.3. Aplicaciones: Inmótica y domótica

La inmótica es el conjunto de tecnologías que permiten la automatización integral de inmuebles, centralizando el control de calefacción, iluminación, ventilación o aire acondicionado, entre otros. Tiene como objetivos mejorar el confort de los ocupantes del edificio, aumentar la eficiencia energética y de costes y prolongar el ciclo de vida útil del edificio y los sistemas que lo componen.

Una red inmótica cuenta con un conjunto de sensores y de actuadores y con un sistema de control. Esta gama de sensores está compuesta, entre otros, por sensores de temperatura, humedad o presión. Como actuadores, se pueden encontrar motores eléctricos o relés. Los controladores serán pequeños ordenadores construidos específicamente para este propósito, capaces de recoger la información captada por los sensores y de enviar los comandos de control a los actuadores. Un ejemplo de estos controladores es un PLC.

La inmótica integra la domótica interna dentro de una estructura en red. La domótica es una rama de la inmótica circunscrita al ámbito de la vivienda habitual. Al igual que la inmótica, la domótica reúne un conjunto de tecnologías capaces de automatizar diversas funciones de la vivienda, como pueden ser la gestión energética, la seguridad, la comunicación o el confort.

Además, el control de estas funciones suele tener la capacidad de ser remoto, de modo que el usuario pueda monitorizarlo o controlarlo sin necesidad de estar dentro de la vivienda.

Figura 2: Aplicaciones dentro de la domótica

Dentro del marco de este proyecto, la red WiFi estándar está maquetada simulando un control domótico, aunque el ámbito de aplicación de dicha red es mayor, pudiéndose instalar por ejemplo en un edificio de oficinas o en unos talleres. Esto es así porque a la hora de construir la red no me he centrado en las características particulares de una vivienda concreta, sino en la estructura de la propia red, la comunicación WiFi entre nodos y la programación lógica básica para monitorizar y controlar diversos tipos de sensores y actuadores.

2. ESTADO DEL ARTE

Actualmente se está imponiendo la moda de la domótica, principalmente debido al impulso de las grandes marcas de tecnología, buscando un nuevo nicho de mercado donde vender sus productos. De hecho, no se trata simplemente de grandes empresas, sino que, a día de hoy, Apple, Alphabet, Microsoft y Amazon son las empresas más grandes del mundo por valor de mercado. Cada una de estas marcas lanza su kit de productos para domotizar nuestro hogar, y cada vez vemos con más frecuencia las campañas de marketing del "Amazon Echo" o del "Google Home".

			31 March 2019
Rank	Company name	Sector	Market capitalisation ($bn)
1	Microsoft	Technology	905
2	Apple	Technology	896
3	Amazon.com	Consumer Services	875
4	Alphabet	Technology	817
5	Berkshire Hathaway	Financials	494
6	Facebook	Technology	476
7	Alibaba	Consumer Services	472
8	Tencent	Technology	438
9	Johnson & Johnson	Healthcare	372
10	Exxon Mobil	Oil & Gas	342

Figura 3: Mayores empresas cotizadas por capitalización bursátil

Frente a estos gigantes nos encontramos los dispositivos de hardware y software libre, mediante los cuales, por un precio significativamente menor, podemos crear redes domésticas adaptadas a nuestras necesidades.

El software libre es un software informático cuyo código es distribuido bajo unas condiciones que permiten a los usuarios ejecutarlo para cualquier propósito, así como estudiarlo, editarlo y distribuir su versión adaptada. De la misma forma, el hardware libre consiste en dispositivos físicos o tecnología diseñada y distribuida bajo la misma filosofía de diseño abierto. Así, tanto su diseño como el software que maneja ese hardware son distribuidos en términos de libertad de uso.

Como ejemplo de software libre está Linux, una serie de sistemas operativos de código abierto. A partir de esta iniciativa se han desarrollado sistemas como Raspbian, el sistema operativo oficial de Raspberry Pi, o Android, muy extendido gracias a su implementación en teléfonos móviles.

Esta libertad a la hora de estudiar y modificar los elementos ha hecho que existan una gran comunidad de usuarios realizando distintos proyectos en estas plataformas y un constante flujo de *feedback*, que beneficia tanto a los usuarios como a los fabricantes. Así, podemos encontrar en la web una gran cantidad de información de las plataformas Arduino y Raspberry pi, dos de las iniciativas más populares de este tipo.

De esta manera, los usuarios no solo cuentan con la ventaja del precio a la hora de competir con los gigantes tecnológicos, sino que la ventaja más importante es la comunidad de usuarios respaldando estos proyectos. Así, a la hora de adaptar un proyecto a tus necesidades, no se parte de cero, sino que partiendo de la información existente se crea algo nuevo, aportando un pequeño grano de arena a la comunidad.

Respecto a la programación de los dispositivos de software libre, existe una amplia colección de funciones y librerías ya desarrolladas. Así, es fácil encontrar funciones básicas como la conexión del dispositivo a una red WiFi o el lanzamiento de una petición como cliente a un servidor. Por ello, para una persona con experiencia en programación no sería complicado hacer un pequeño proyecto, como por ejemplo leer sensores en diferentes dispositivos y monitorizar esa información en su ordenador.

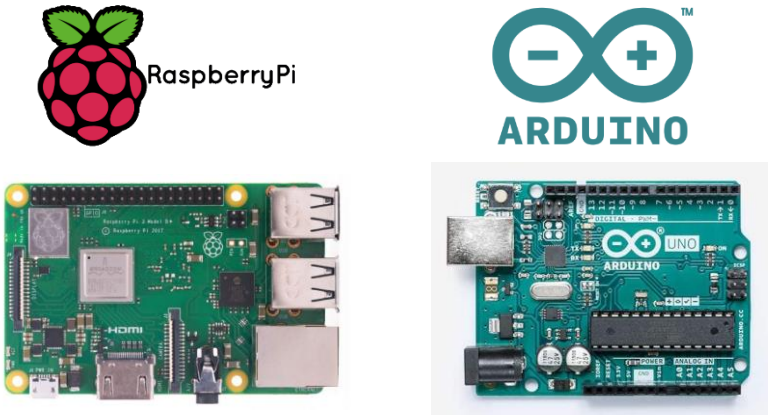


Figura 4: RaspberryPi y Arduino, ejemplos de software y hardware libre

El problema con el que me he encontrado es que toda esta información disponible dentro del software libre no está adaptada a un nivel de usuario, es decir, la información está disponible para que tú mismo seas capaz de programar los dispositivos y crear tu propio proyecto. Pero para alguien sin experiencia en programación esto puede resultar complicado y, aunque muestre interés por lo que los dispositivos son capaces de hacer, descartará la idea de utilizarlos por la aparente complejidad.

Por eso desde este proyecto quiero aportar una solución a ese problema: crear un código estándar para que el usuario no tenga que programar, y para que utilizar estos dispositivos sea tan sencillo como enchufarlos y que ya comiencen a funcionar.

3. MICROCONTROLADORES Y ELEMENTOS DE LA RED

A la hora de afrontar el proyecto, fue necesario elegir una placa de desarrollo que se adaptara a las necesidades del proyecto con las especificaciones necesarias y suficientes. El mercado de microcontroladores es amplio, y antes de decidirme por una placa de desarrollo en concreto, estudié distintas opciones: Arduino Uno, Raspberry Pi y NodeMCU.

Las placas de desarrollo, como Arduino y Raspberry Pi, suelen ser las elecciones habituales a la hora de prototipar nuevos dispositivos IoT. Estas placas de desarrollo son esencialmente "pequeños ordenadores", capaces de ser conectados y programados mediante un PC estándar. Una vez programadas, estas placas de desarrollo pueden conectar y controlar diversos sensores y actuadores.

Para formar parte de una red IoT, estas placas necesitan una conexión a Internet, siendo la más comoda y versátil la conexión inalámbrica. Sin embargo, tanto Arduino como Raspberry Pi (en sus placas más básicas) no cuentan con un soporte integrado para este tipo de conexión. Para ello es necesario conectar un módulo independiente y programar la conexión y acceso a dicho módulo u optar por una de las placas de mayor precio que tenga conectividad WiFi integrada.

La placa de desarrollo NodeMCU (Node Micro ControllerUnit) destaca por tener integrado un soporte de conectividad WiFi a un precio moderado, lo que hace que la aplicación y desarrollo de proyectos IoT sea algo más sencillo.

3.1. Arduino Uno

Arduino Uno es una de las numerosas placas de desarrollo de la compañía de fuente abierta Arduino. Es la placa emblema de la marca, siendo la primera de una serie de placas Arduino con conectividad USB, y ha sido nombrada así por el lanzamiento del IDE 1.0 de Arduino.

Esta placa está basada en el microcontrolador ATmega328P. Cuenta con 14 pines de entrada/salida digitales (de las cuales 6 pueden usarse como salidas PWM) y 6 entradas analógicas.

Figura 5: Arduino Uno

Gracias a su precio asequible y su facilidad de manejo y programación debido a su IDE oficial, es muy utilizada para proyectos de todo tipo, contando con una gran comunidad de usuarios. Esto hace posible encontrar una gran información respecto a su programación, con librerías y funciones ya desarrolladas, y amplios foros donde discutir nuevas aplicaciones.

El principal inconveniente es que esta placa, por sí misma, no dispone de conectividad WiFi. Este inconveniente se puede solucionar conectando elementos externos a Arduino Uno, por ejemplo, un módulo ESP8266 o una placa Shield WiFi.

Figura 6: Módulo ESP8266 (izquierda) y Shield WiFi Arduino (derecha)

Esto acarrea a su vez otros inconvenientes. En el caso del Shield WiFi, implica un aumento del precio, incluso superior al de la propia placa, teniendo en cuenta que sería necesaria una placa con su propio Shield WiFi para cada nodo de la red.

Por ello en primer lugar opté por realizar las primeras pruebas con Arduino Uno conectando un módulo ESP8266. La conexión no es sencilla, ya que el módulo ESP8266 cuenta con 8 pines, de los cuales 5 se utilizarán para la conexión.

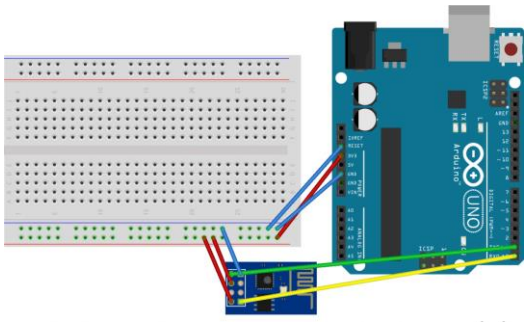


Figura 7: Conexión Arduino Uno a módulo ESP8266

De esta forma es posible comunicarse con el módulo ESP8266 desde el ordenador, utilizando la placa Uno como "puente" y el monitor serie del IDE de Arduino como interfaz de dicha comunicación. Esta comunicación se lleva a cabo mediante comandos AT (comandos Hayes), introduciéndolos en el monitor serie. De esta forma es posible, por ejemplo, conectarse a la red doméstica mediante el comando AT+CWJAP="WIFI_NAME","WIFI_PASSWORD".

Es posible también incluir estos comandos AT en el Sketch de Arduino, de forma que se ejecuten normalmente junto con su programación, eliminando la necesidad de introducir dichos comandos en el monitor serie.

Además de los ya mencionados Shield WiFi y el módulo ESP8266, Arduino cuenta con placas de desarrollo con conectividad WiFi incluida.

Las especificaciones técnicas de Arduino Uno son:

- Microcontrolador: ATmega328P
- Tensión de operación: 5 V
- Tensión de alimentación: 7-12 V
- Límite de tensión de alimentación: 6-20 V
- Pines de entrada/salida digitales: 14 (de los cuales 6 proporcionan salida PWM)
- Pines de entrada/salida PWM digitales: 6
- Pines de entrada analógicos: 6
- Corriente continua por pin de entrada/salida: 20 mA
- Corriente continua por pin de 3.3 V: 50 mA

- Memoria Flash: 32 KB (ATmega328P) de los cuales 0,5 KB son utilizados para el arranque
- SRAM: 2 KB (ATmega328P)
- EEPROM: 1 KB (ATmega328P)
- Velocidad de reloj: 16 MHz
- LED integrado: 13
- Dimensiones: 68,6 mm x 53,4 mm
- Peso: 25 g

3.1.1. Módulo ESP8266: Conexión WiFi con Arduino Uno

El ESP8266 es un SoC (*System on a Chip*), es decir, contiene los elementos fundamentales de un ordenador moderno: CPU, RAM, conectividad red (WiFi) e incluso un sistema operativo moderno y kits de desarrollo de software. Fue diseñado por Espressif Systems, y otra de sus ventajas es su competitivo coste, inferior a 2€, que ha hecho posible desarrollar una comunidad de usuarios bastante amplia.

Figura 8: ESP8266 Espressif Systems

Es posible adquirir el ESP8266 montado en un pequeño circuito con 8 pines GPIO, en su versión ESP01. Estos módulos están pre-programados con un firmware que consta de un set de comandos AT, lo que hace posible dotar a Arduino Uno de conectividad WiFi una vez establecida la comunicación entre Arduino Uno y el módulo ESP8266.

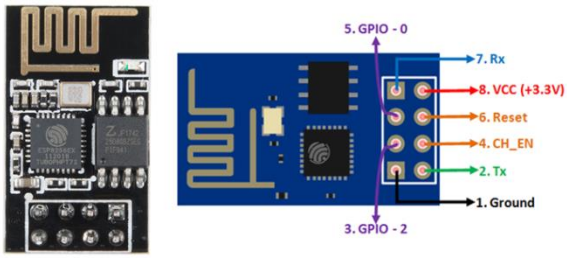


Figura 9: ESP01 Pinout

La conexión a la red WiFi se lleva a cabo mediante comandos AT. Estos comandos fueron desarrollados por Dennis Hayes para la configuración de los modem de la marca de su mismo nombre en 1981. Consisten en unas pequeñas cadenas de texto que pueden ser combinadas para producir comandos para operaciones como llamar, colgar o configurar los parámetros de la conexión.

Estos comandos AT son enviados desde Arduino al módulo ESP8266 vía comunicación en serie. Una vez conectado a la red WiFi, el módulo tiene la capacidad de enviar la información recibida por la puerta serie a una dirección IP y puerto que queramos.

Es posible trabajar introduciendo los comandos AT en el monitor serie del IDE de Arduino, una vez programada la comunicación en serie en Arduino Uno. Esta forma de trabajar sería poco

eficiente, y para cumplir las necesidades de la red lo correcto sería incluir los comandos AT en el Sketch de Arduino Uno.

La desventaja de trabajar con la configuración de Arduino Uno más el módulo ESP01 es que a la hora de comunicar la información entre varios nodos de la red tendremos que pasar por tres comunicaciones: de Arduino Uno (cliente) a su ESP01 vía puerta serie, del ESP01 (cliente) al ESP01 (servidor) vía WiFi, y del ESP01 al Arduino Uno (servidor) vía serie otra vez.

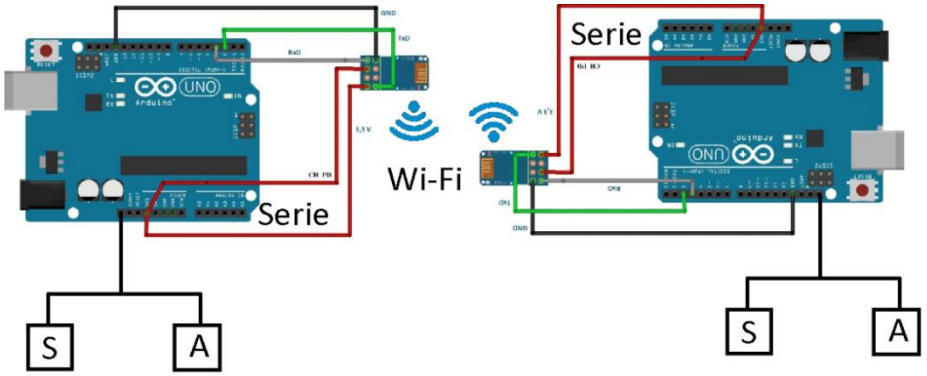


Figura 10: Comunicación WiFi Arduino Uno mediante ESP01

Las especificaciones técnicas del módulo ESP8266 versión ESP01 son:

- Tensión de alimentación: 3,3V
- Protocolos soportados: 802.11 b/g/n.
- Soporte de red: 2,4 GHz
- Banda: 2400 (MHz)
- WiFi Direct (P2p), Soft Access Point.
- Stack TCP/IP integrado.
- PLL, reguladores y unidades de manejo de energía integrados.
- Potencia de salida: 0,15 (W); +19,5 dBm en modo 802.11b
- Consumo en modo de baja energía: <10 uA.
- Procesador integrado de 32 bits.
- Soporta tres modos de funcionamiento: AP, STA, STA + AP
- Comunicación tipo de interfaz: Serial, UART
- Tamaño: 24 mm x 14 mm

3.2. Raspberry Pi

Raspberry Pi es un ordenador de placa reducida desarrollado por la fundación británica de su mismo nombre, dedicada a promover la enseñanza de informática básica en instituciones educativas. Al igual que Arduino, su software es de código abierto. Al no tratarse de una placa de desarrollo sino de un ordenador completo cuenta con mayores características: memoria RAM, GPU, puertos USB, HDM y Ethernet, 40 pines de entrada/salida, y conector para cámara.

Figura 11: Raspberry Pi Model 3 B+

De la misma forma que Arduino, Raspberry Pi cuenta con una amplia comunidad de usuarios y con numerosas páginas web en las que se puede encontrar información para desarrollar proyectos en ella. Estos proyectos se pueden desarrollar en su sistema operativo oficial basado en Debian, aunque permite utilizar otros sistemas, como Linux ARM o Windows. Su principal lenguaje de programación es Python, aunque también soporta otros lenguajes como C, Perl o Ruby.

Las especificaciones técnicas del modelo 3 B+ son:

- Chipset Broadcom BCM2837 a 1,2 GHz
- ARM Cortex-A53 de 64 bits y cuatro núcleos
- LAN inalámbrica 802.11 b/g/n
- Bluetooth 4.1 (Classic y Low Energy)
- Coprocesador multimedia de doble núcleo Videocore IV
- Memoria LPDDR2 de 1 GB
- Compatible con todas las últimas distribuciones de ARM GNU/Linux y Windows 10 IoT
- Conector micro USB para fuente de alimentación de 2,5 A
- 1 puerto Ethernet 10/100
- 1 conector de vídeo/audio HDMI y 1 conector de vídeo/audio RCA
- 1 conector de cámara CSI
- 4 x puertos USB 2.0
- 40 pines GPIO
- Antena de chip
- Conector de pantalla DSI
- Ranura de tarjeta microSD
- Dimensiones: 85 x 56 x 17 mm

Como se puede observar, cumple con las características necesarias para el proyecto. El inconveniente es que cumple en exceso, estando sobredimensionada para las necesidades del proyecto, aunque sí podría cumplir la función de nodo central de la red WiFi, optando por placas más económicas para el resto de nodos de la red.

3.3. NodeMCU

NodeMCU es un entorno de desarrollo de software abierto, construido partiendo de la base del ESP8266. El ESP8266 es un sistema que cuenta con las características de un microcontrolador en sí mismo. Es un dispositivo sencillo, de tamaño reducido y precio económico, lo que ha propiciado su uso dentro de aplicaciones IoT.

Por otra parte, el ESP8266 de forma individual (como chip) es de difícil acceso y uso. Sería necesario soldar conexiones a cada uno de sus pines para las tareas más simples, y habría que programarlo mediante instrucciones máquina de bajo nivel que puedan ser interpretadas por su hardware. Aunque todo esto no es problema en la fabricación a escala de sistemas integrados, es una carga para usuarios de menor nivel.

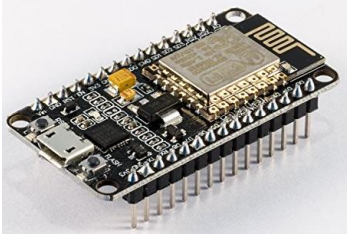


Figura 12: Placa NodeMCU ESP8266

Dada esta necesidad nace el proyecto NodeMCU, que trata de simplificar y acercar a los usuarios el desarrollo de los proyectos que utilizan el ESP8266. Para esto utiliza dos componentes clave:

- Un firmware abierto para el ESP8266, escrito sobre el kit de desarrollo de software del fabricante y basado en el entorno de programación eLua. Además, la comunidad de usuarios ha hecho posible programar este sistema mediante el IDE de Arduino.
- Un kit de desarrollo que incorpora el chip ESP8266 en un circuito estándar. Este circuito cuenta con un puerto USB, botón de reset, antena WiFi, luces LED y un conjunto de pines GPIO (General Purpouse Input Output) de tamaño estándar para ser conectado a las placas de pruebas.

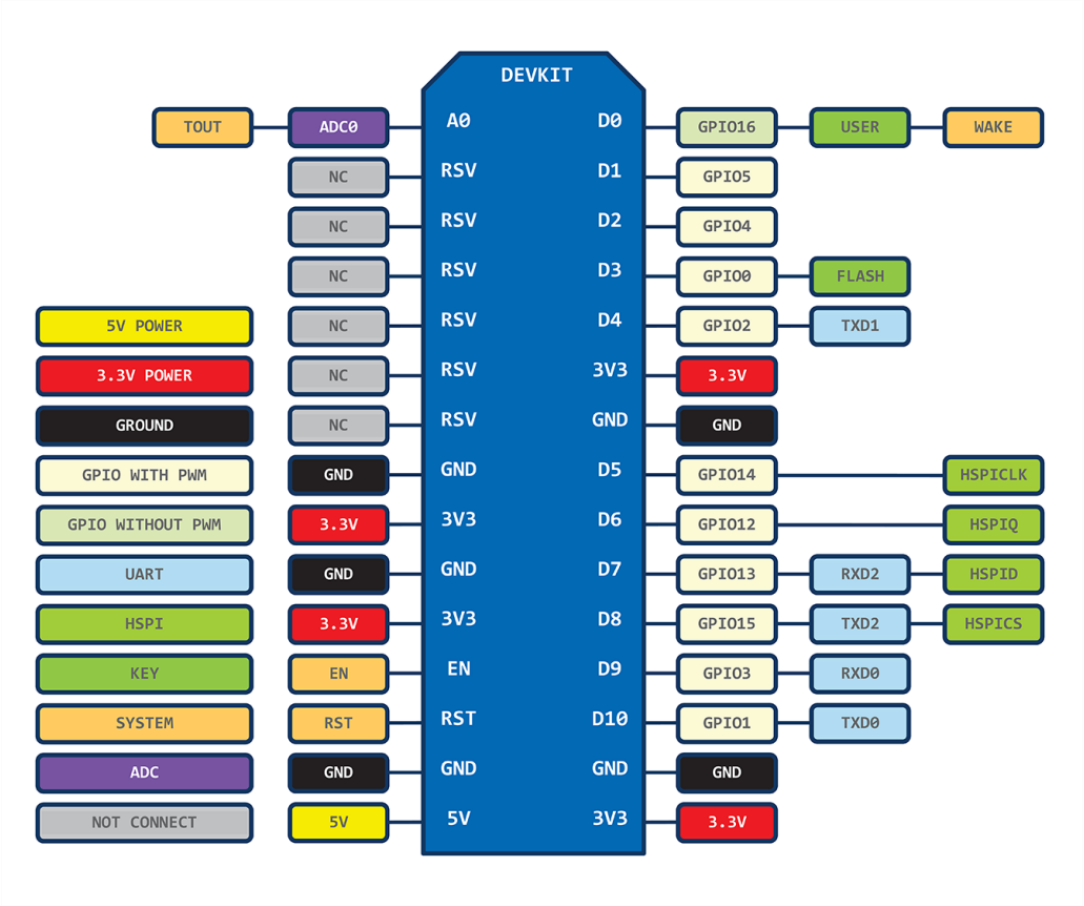


Figura 13: NodeMCU pinout

NodeMCU no es la única placa que integra el módulo ESP8266. Existe una amplia gama de placas con este módulo, por ejemplo, ESPresso Lite, Adafruit Feather, Olimex, Spark Fun ESP8266 y muchas más. NodeMCU es una de las placas más baratas, y a la vez con mejor integración del módulo ESP8266. Esto, unido a su diseño de open hardware, ha hecho a esta placa una de las más populares para los proyectos IoT, lo que hace que haya más información compartida entre una mayor comunidad de usuarios.

Las especificaciones técnicas de NodeMCU son:

- Procesador: ESP8266 @ 80MHz (3,3 V) (ESP-12E)
- 4MB de memoria FLASH (32 MBit)
- WiFi 802.11 b/g/n
- Regulador 3,3 V integrado (500 mA)
- Conversor USB-Serial CH340G / CH340G

- Función Auto-reset
- 9 pines GPIO con I2C y SPI
- 1 entrada analógica (1,0 V máx)
- 4 agujeros de montaje (3 mm)
- Pulsador de RESET
- Entrada alimentación externa VIN (20 V máx)

3.4. Justificación del dispositivo elegido

He elegido la placa NodeMCU como elemento controlador de la red, servirá tanto de nodos cliente encargados de recoger la información de sensores y actuadores como de nodo servidor encargado de administrar dicha información.

Los motivos son los siguientes:

- **Comunicación WiFi**: Si bien es cierto que Arduino Uno tiene capacidad de comunicación WiFi, es necesario conectar módulos adicionales, ya sea el Shield WiFi o el módulo ESP01. Por lo tanto, Raspberry Pi y NodeMCU, al contar con conexión WiFi sin necesidad de módulos adicionales, le superan en este aspecto.
- **Precio**: La placa NodeMCU se puede adquirir por un precio en torno a 3€. La placa Arduino Uno se puede adquirir por un precio alrededor de 3,5€ (en la página oficial cuesta 20€), a lo que habría que añadir o el módulo ESP01 por 1,5€ o el Shield WiFi por cerca de 3,5€. En la tienda oficial de Arduino es posible adquirir placas Arduino con WiFi ya integrado por un precio desde 30€. Raspberry Pi con su caja tiene un precio cercano a 40€. Por lo tanto, en cuestión de precios las placas más asequibles son NodeMCU y Arduino Uno con el módulo ESP01.
- **Necesidades del proyecto**: Todas las placas estudiadas cumplen con las necesidades de proyecto. Disponen de conectividad WiFi, suficientes pines GPIO para controlar sensores y actuadores y conexión al PC para poderse programar mediante software abierto. Además, todas ellas cuentan con una comunidad de usuarios amplia y con gran cantidad de información disponible en la web. Cabe destacar que la placa Raspberry no solo cumpliría con las necesidades del proyecto, sino que las sobrepasaría.

Por lo tanto, la placa NodeMCU, contando con las características necesarias y suficientes para el proyecto, destaca por una parte por su económico precio y por otra por tener integrada la conectividad WiFi, sin necesidad de combinarla con módulos adicionales.

3.5. Sensores y actuadores

Los sensores y actuadores son los elementos de la red que estarán conectados a los microcontroladores.

Por una parte, los sensores serán los elementos encargados de recoger información del entorno de la red. Existe una amplia gama de este tipo de dispositivos, compatibles con NodeMCU, que van desde elementos simples como termistores o fotorresistores hasta elementos más complejos como cámaras.

Por otra parte, los actuadores permitirán ejecutar diferentes acciones en función de la información recogida previamente por los sensores. Los actuadores también serán controlados por las placas NodeMCU y la elección de cada actuador dependerá de las necesidades del usuario, como por ejemplo un control de riego o de iluminación.

3.5.1. Sensores

Los sensores son elementos capaces de detectar distintas magnitudes físicas o químicas de su entorno, transformándolas con un transductor en señales eléctricas. Dentro de la red, la función de los sensores será transmitir estas señales al microcontrolador al que estén conectados.

Así, la información del entorno que puede recoger la red estará limitada por los sensores que la compongan. Como existe una amplia gama de sensores, el usuario podrá seleccionar los sensores que mejor se adapten a sus necesidades, por ejemplo, sensores de intensidad luminosa, de movimiento, de detección de gases, de humedad y temperatura…

Dada esta amplia gama de sensores, la red estándar tendrá que ser capaz de adaptarse a cada uno, pudiendo utilizar o instalar distintos tipos de sensores sin necesidad de modificar la configuración de la red.

Para simplificar el estudio de la implantación de los sensores en la red, he subdividido los sensores en dos tipos, analógicos y digitales. Dentro de cada uno de los tipos elegí un sensor para maquetar la red. El resto de sensores de un mismo tipo tendrá un funcionamiento análogo.

3.5.1.1. Sensor analógico: Fotorresistor

Un fotorresistor es un componente electrónico cuya resistencia disminuye con el aumento de intensidad de luz incidente, la radiación óptica aporta la energía necesaria para aumentar el número de electrones libres, disminuyendo su resistividad. Este sensor es un elemento útil dentro de una red domótica, con aplicaciones por ejemplo dentro del control de iluminación.

El fotorresistor, al tratarse de un sensor analógico, emite una señal comprendida entre un campo de valores. La señal varía en el tiempo, en función de la variación en la magnitud que mide. Dado que la señal que el sensor envía al microcontrolador es una señal continua, y éste solo trabaja con señales digitales, el microcontrolador llevará a cabo una conversión analógica-digital.

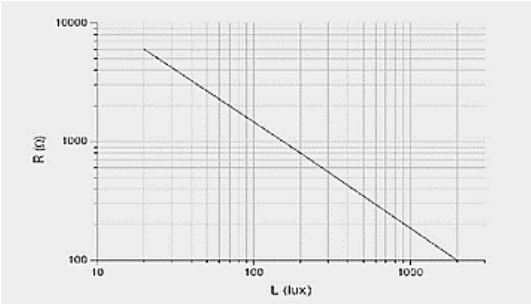


Figura 14: Curva Resistencia-Intensidad luminosa

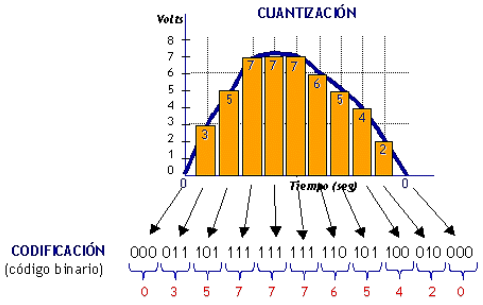


Figura 15: Resolución de la conversión analógica-digital

NodeMCU cuenta con un conversor analógico digital con una resolución de 10 bits, por tanto, la señal estará comprendida entre 0 y 1023.

Para este proyecto he utilizado un fotorresistor modelo GL5516.

Figura 16: Fotorresistor modelo GL5516

3.5.1.2. Sensor digital: sensor PIR

Los sensores PIR (*passive infrared sensor*) son elementos capaces de detectar la variación en la radiación infrarroja que reciben, y al hacerlo emiten una señal. Esto hace que los sensores PIR se utilicen frecuentemente como sensores de movimiento.

Los PIR se componen de un par de electrodos fijados de forma que un sensor ofrezca una señal de salida positiva y el otro, una señal de salida negativa. Mientras no se detecte ningún objeto, ambos sensores recibirán la misma intensidad de radiación infrarroja y sus señales se contrarrestarán. Por el contrario, cuando un cuerpo caliente atraviese el campo de detección de uno de los sensores, su valor diferirá del de su pareja, provocando que se registre un cambio en la salida.

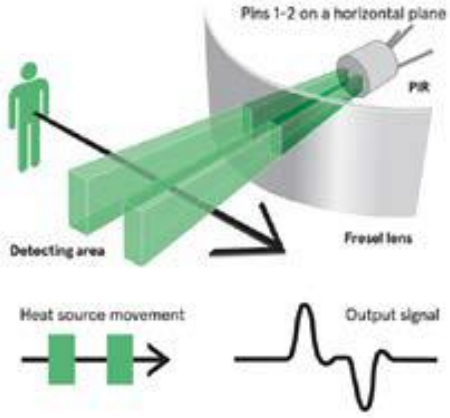


Figura 17: Esquema de funcionamiento del PIR

Se puede colocar una lente de Fresnel delante de la ventana para incrementar el rango de la zona de detección. Para este proyecto se ha utilizado el modelo de sensor HC-SR501, que cuenta con dicha lente, así como de un pequeño circuito de estabilización y control. Este circuito cuenta con 3 pines para su conexión: una entrada de tensión, una salida a tierra y la salida de la señal digital.

Figura 18: PIR HC-SR501

3.5.1.3. Sensor digital y comunicación I2C: sensor meteorológico BME280

Otra de las características de NodeMCU es que permite la comunicación mediante I2C. Se trata de un protocolo en serie, por sus siglas Circuito Inter Integrado, que hace posible la comunicación entre varios dispositivos maestros con varios dispositivos esclavos. Para ello utiliza dos vías de comunicación: SDA (Serial Data) donde se realiza el intercambio de información, y SCL (Serial Clock) donde se envía la señal de reloj, lo que permite que sea un protocolo síncrono. Los pines SDA y SCL de NodeMCU son los pines D2 y D1 respectivamente.

Esto permite a NodeMCU conectarse simultáneamente a varios dispositivos que utilicen este protocolo, para ello en el código habrá que especificar la dirección de cada dispositivo.

Un ejemplo de estos dispositivos es el sensor BME280, un sensor ambiental desarrollado por Bosch y capaz de medir humedad relativa, temperatura y presión barométrica.

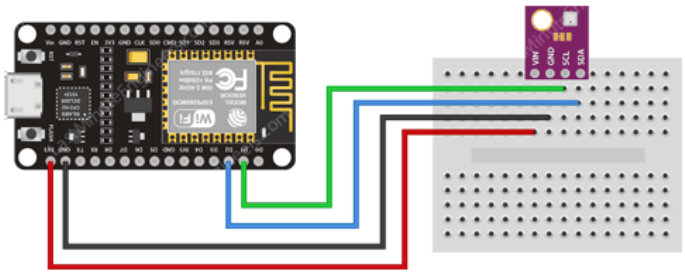


Figura 19: Conexión NodeMCU con sensor BME280

Para llevar a cabo la comunicación con este sensor será necesario incluir librerías al código, tanto las correspondientes a la comunicación I2C como las propias de este sensor.

3.5.2. Actuadores

Los actuadores son elementos mecánicos capaces de utilizar determinado tipo de energía para generar un efecto en un proceso automatizado, es decir, los microcontroladores de la red enviarán la señal a los actuadores para que resulten en la función que deseemos. En el caso de la domótica, será por ejemplo la subida o bajada de persianas, encendido de iluminación, apertura de válvulas, etc.

Para el maquetado del proyecto, he utilizado dos tipos de actuadores: diodos LED y servomotor.

3.5.2.1. Diodos LED

El diodo LED se trata de un diodo de unión p-n que emite luz cuando está activado. En este proyecto, el diodo LED tiene la finalidad de simular el funcionamiento de un actuador mediante la salida de una señal digital HIGH/LOW, por ejemplo, la activación de un relé.

Por ello he utilizado LED de colores simples, aunque existen otros tipos de LED como el RGB, que es capaz de emitir luz de distintos colores, y su manejo es a través de una señal PWM.

La utilización de LED permite maquetar circuitos de una forma sencilla. Cuentan con dos patillas, una larga (ánodo) y una corta (cátodo), fáciles de conectar a la placa de pruebas o protoboard. Dado que los LED están polarizados, es importante recordar que solo funcionarán si se han conectado en la posición correcta.

Figura 20: Luces LED

3.5.2.2. Servomotor

Los servomotores son un tipo especial de motor eléctrico, caracterizados por ser capaces de posicionarse en cualquier posición dentro de su intervalo de operación y mantenerse estables en dicha posición. Para conectar este elemento al microcontrolador cuenta con tres conexiones: la toma de corriente, la toma de tierra, y una entrada para la señal de control.

La característica de esta señal de control es que es una señal de tipo PWM (*Pulse Width Modulation*). Este sistema consiste en generar una onda cuadrada en la que se varía el tiempo que el pulso está en nivel alto, manteniendo el mismo periodo. Así, es posible modificar la posición del servo dentro de su rango de trabajo.

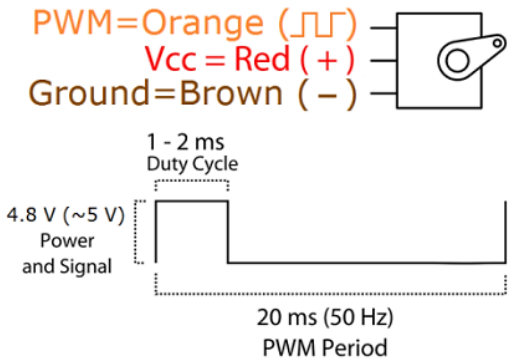


Figura 21: Esquema PWM Servo

Los servomotores se utilizan en diversas aplicaciones, principalmente en los campos de automatización industrial y en robótica, hasta en modelismo y juguetes radio control.

En este proyecto he utilizado el servomotor modelo SG90, cuyas especificaciones técnicas son:

- Velocidad: 0,3 s/60°
- Par motor: 1,5 kg · cm
- Temperatura de funcionamiento: 0°C a 55°C
- Tensión de funcionamiento: 4,2 V a 6 V
- Ángulo de rotación: 180°
- Dimensiones: 22,2 x 11,8 x 32,2 mm
- Peso: 9 g

Figura 22: MicroServo SG90

3.5.2.3. Pantalla LCD

La pantalla de cristal líquido, por sus siglas en inglés Liquid Crystal Display, consiste en una pantalla plana donde los píxeles son controlados individualmente haciendo pasar la corriente eléctrica por una sustancia líquida, haciendo que estos se vuelvan opacos, creando un contraste visible gracias a una iluminación trasera. Su bajo coste y consumo energético hace que sean comúnmente utilizadas para mostrar información en proyectos con microcontroladores.

Este dispositivo se controla desde NodeMCU mediante el protocolo I2C, como explicado anteriormente.

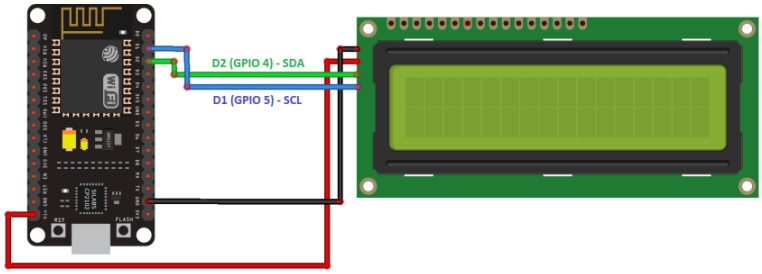


Figura 23: Conexión I2C entre NodeMCU y pantalla LCD

3.5.3. Otros materiales utilizados

El resto de materiales utilizados para realizar la maqueta de este proyecto son elementos básicos a la hora de construir pequeños circuitos eléctricos. Se pueden adquirir de forma conjunta en packs de aprendizaje o iniciación. Entre estos elementos se encuentran:

- Resistencias de 1-10 kOhmnios.
- Pulsadores.
- Placa de pruebas o protoboard.
- Cable de conexión USB.
- Cables de conexión para protoboard.
- Relés.

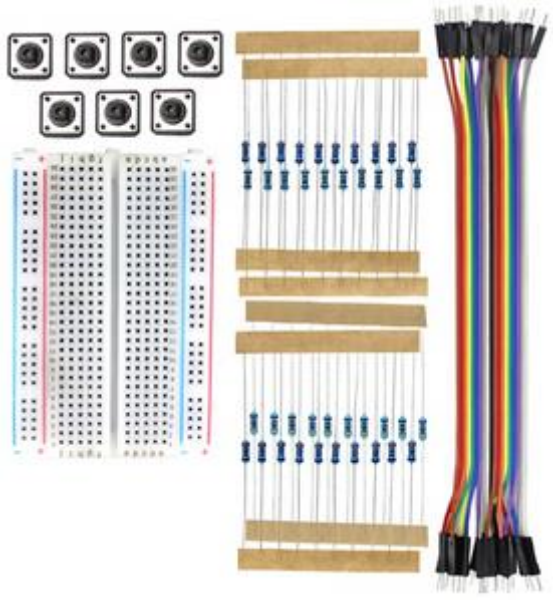


Figura 24: Pack de pulsadores, protoboard, resistencias y cables

4. PROGRAMACIÓN EN NODEMCU

En este apartado expondré los pasos seguidos para la programación de la red WiFi en NodeMCU, desde la configuración inicial para poder programar NodeMCU mediante el uso del IDE (*Integrated Development Enviroment*) de Arduino, pasando por diferentes ejemplos del control de sensores y actuadores y comunicación WiFi entre placas y con el usuario, hasta el código fuente de la red completa.

El lenguaje de programación de Arduino se basa en dos antecendentes. El primero es el lenguaje de programación y entorno de desarrollo Processing, el cual sienta la base para el IDE de Arduino. Este toma la estructura de los Sketch, diferenciando entre una función Set up inicial y una función Loop que se estará ejecutando de forma continua.

El segundo antecedente es la plataforma de desarrollo Wiring, del cual Arduino toma el lenguaje C/C++ simplificado y su compilador GCC.

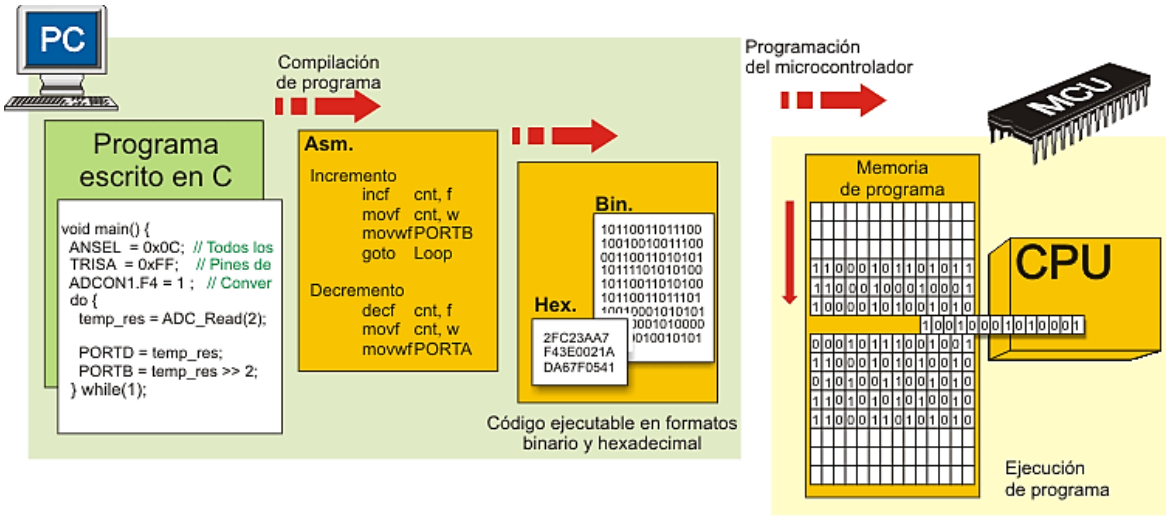


Figura 25: Jerarquía de programación de NodeMCU

4.1. Pluggin ESP8266 para IDE Arduino

El entorno de desarrollo de Arduino (IDE) es un programa que permite la programación de cada una de las placas de Arduino. Se puede obtener de forma gratuita descargándolo desde su página oficial (https://www.arduino.cc).

Antes de empezar programar NodeMCU desde el IDE, es necesario prepararlo mediante la instalación de un pluggin.

Se puede recurrir a dos opciones para esto, descargar un IDE con el pluggin ya instalado, o instalar el pluggin por separado. Este pluggin no sólo permitirá la programación en NodeMCU sino también la de una gran variedad de placas que utilicen el chip ESP8266.

En este caso, como ya contaba previamente con el IDE, ilustraré con diferentes imágenes como instalar el pluggin necesario. La versión utilizada del IDE es la 1.8.4, pero bastará con tener instalada la versión 1.6.4 o posteriores, a partir de la cual la versión permite la creación de definiciones de nuevas placas y su integración en el entorno.

Es necesario descargar el pluggin de internet. Para ello, abrimos el IDE y nos dirigimos a la Archivo\Preferencias. A continuación, en la parte inferior de la ventana que se abre, nos dirigimos a cuadro “Gestor de URLs Adicionales de Tarjetas:” e introducimos este vínculo: http://arduino.esp8266.com/stable/package_esp8266com_index.json

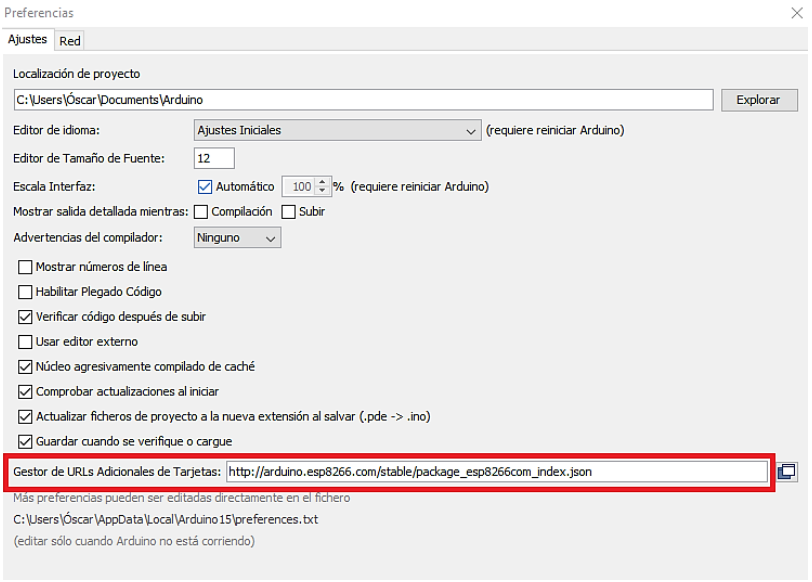


Figura 26: IDE Arduino – Preferencias

De esta forma se pueden instalar diferentes tarjetas, no solo para NodeMCU. Confirmamos pulsando OK y nos dirigimos a Herramientas\Placa\Board Manager. Aparecerá un listado de tarjetas, entre las que tenemos que elegir “esp8266 by ESP8266 Community”.

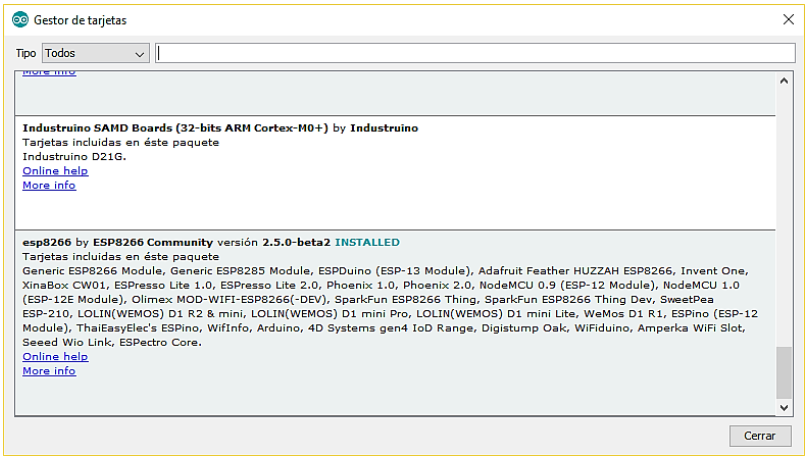


Figura 27: IDE Arduino – Gestor de tarjetas

En mi caso utilizaré la versión 2.5.0-beta2, la más reciente disponible. Confirmamos con el botón “Instalar”. Una vez finalizada la instalación, nos dirigimos a Herramientas\Placa, y seleccionamos de entre el listado “NodeMCU 1.0 (ESP-12E Module)”, que será la utilizada en este proyecto.

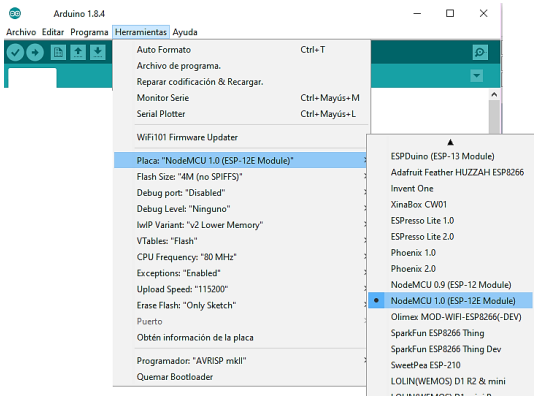


Figura 28: Selección de tarjeta NodeMCU 1.0

En la pestaña “Herramientas” encontramos más opciones de configuración. Para finalizar la configuración del IDE de Arduino para el uso de NodeMCU ha de seleccionarse la velocidad de subida (*Upload Speed*), que es la velocidad de comunicación desde el PC con la placa a la hora de transmitir desde el PC el Sketch previamente elaborado. La velocidad de subida por defecto está establecida en 9600 Baudios, en este caso se ha de seleccionar 115200 Baudios para NodeMCU.

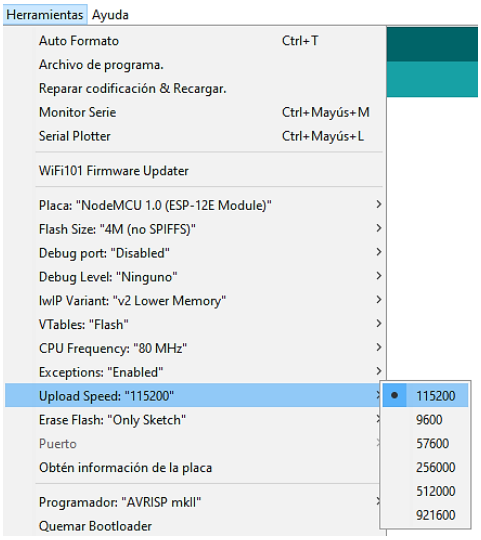


Figura 29: Selección de velocidad de subida

Finalizada la instalación del pluggin y una vez configurado el IDE, el IDE de Arduino nos permitirá elaborar programas para la placa NodeMCU utilizando el entorno de programación de Arduino. De esta forma la programación de la placa será más sencilla e intuitiva.

4.2. Instalación de librerías

Las librerías son archivos de código que se agrupan por incluir funciones similares. Por ejemplo, para poder programar el funcionamiento de un servo en el IDE de Arduino, es conveniente incluir al inicio del programa la librería “Servo.h”. Esta librería ya incluye funciones para controlar un servomotor mediante PWM, sin necesidad de crear tales funciones de cero. Se trata de una librería estándar, es decir, instalada por defecto en el IDE de Arduino.

Por otra parte, ya que las librerías estándar son escasas y limitadas, existen librerías no estándar para diversas funciones, como determinados sensores, pantallas, y para la comunicación entre dispositivos. Estas librerías no estándar son desarrolladas por la comunidad de usuarios, o por fabricantes de dispositivos y accesorios como los citados anteriormente. Para la comunicación WiFi entre módulos NodeMCU utilizaré librerías desarrolladas para este propósito. Estas librerías se instalaron en el pluggin descrito en el apartado anterior.

Ya que normalmente la instalación de librerías sigue un procedimiento distinto al del pluggin, y su uso es fundamental a la hora de programar dispositivos más complejos, describiré el proceso de instalación de librerías a continuación.

En primer lugar, se descargan los archivos de la librería comprimidos como un archivo .zip. Estos archivos se pueden encontrar en plataformas de desarrollo colaborativo, como por ejemplo GitHub. A continuación, en el IDE, en la pestaña Programa/Incluir librería, nos da la

opción de incluir librería como .zip. Seleccionando el fichero previamente descargado y confirmando, la librería ya estará instalada.

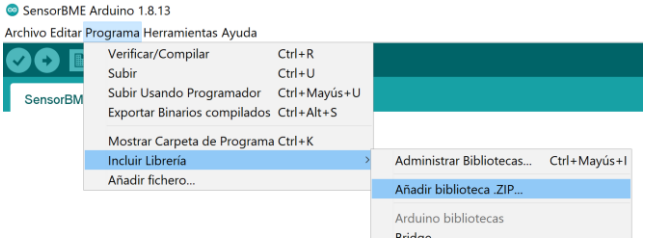


Figura 30: Instalación de biblioteca

Una vez instalada, para incluir una librería en el código se utiliza al comienzo de éste el comando #include seguido por el nombre de la librería, o automáticamente en la pestaña Programa/Incluir librería, y seleccionando la librería que queramos incluir.

Para llevar a cabo la comunicación WiFi entre módulos haré uso de varias librerías, que explicaré más en detalle en posteriores apartados.

4.3. Ejemplos de programación para NodeMCU

A la hora de afrontar este trabajo, en concreto el caso de la programación de NodeMCU, partí de las pruebas más básicas del control de sensores y actuadores con esta placa, desde el control de un led o la lectura de una señal digital hasta el control de un servomotor o la lectura de la señal analógica de un sensor.

Como he comentado anteriormente, el lenguaje de programación de Arduino se basa en el lenguaje C++. C++ es un lenguaje de programación desarrollado en 1979 con el objeto de extender al lenguaje C los mecanismos que habilitan la manipulación de objetos.

Al abrir el IDE de Arduino nos encontramos con un Sketch en blanco. Sketch es la denominación que utiliza Arduino para nombrar sus programas. Consta de un archivo con la extensión ".ino", en el que se pueden encontrar las líneas de código.

La estructura de un Sketch es sencilla, y diferenciada en dos partes, o dos funciones principales:

- *Set Up*: la función setup() es llamada una única vez, al iniciarse el Sketch. Como su nombre indica, dentro de esta función se ejecutan los comandos de "preparación", es decir, inicializar librerías o modos de funcionamiento de los pines.
- *Loop*: la función loop() es ejecutada una y otra vez, de forma indefinida, y es donde se encuentra el núcleo o la parte principal de la programación del Sketch.

```
void setup(){
 // put your set up code here, to run once:
}

void loop(){
 // put your main code here, to run repeatedly:
}
```

Es necesario declarar ambas funciones, independientemente de que se encuentren vacías o no.

A continuación, ilustraré cómo programar el control de los elementos más básicos de la red, esto es, el control de sensores y actuadores mediante NodeMCU.

4.3.1. Parpadeo de un LED

```
void setup(){
 pinMode(D1,OUTPUT);
}

void loop(){
 digitalWrite(D1,HIGH);
 delay(500);
 digitalWrite(D1,LOW);
 delay(500);
}
```

La prueba de control de un LED es considerada una de las pruebas de programación más básicas en las placas de desarrollo. Se utiliza como toma de contacto, además de para comprobar que tanto el IDE de Arduino como la placa funcionan correctamente.

Este código ilustra los comandos necesarios para controlar un diodo emisor de luz, en este caso conectado al pin D1.

En primer lugar se encuentra la función Setup, descrita anteriormente. Es precedida por la palabra “void”, esto es, al ejecutar la función Setup ésta no devuelve ningún valor.

Al ser una función de preparación, dentro de ella indicamos el modo de funcionamiento de los pines. Esto se hace mediante el uso de la función “pinMode”, donde podremos indicar si el pin tiene función de entrada o salida de señal.

El primer argumento de la función es el nombre del pin, en este caso D1, y el segundo argumento es el modo de funcionamiento (OUTPUT para salida o INPUT para entrada).

A continuación, entramos en la función Loop, que, como se describe anteriormente, se estará ejecutando en bucle.

Para conseguir el encendido del pin se utiliza la función “digitalWrite”, con argumentos el pin y el estado, en este caso HIGH. El estado HIGH en el pin D1 establecerá una señal digital de 3,3 V en dicho pin, consiguiendo el encendido del LED.

Mediante la función “delay” estableceremos un tiempo de espera, siendo las unidades de su argumento milisegundos, antes de pasar otra vez a la función digitalWrite en la que en este caso con el argumento LOW, apagaremos el LED.

De esta forma, alternando los estados HIGH y LOW, en intervalos de 500 ms, conseguiremos el parpadeo del led.

La conexión se realiza de la siguiente forma:

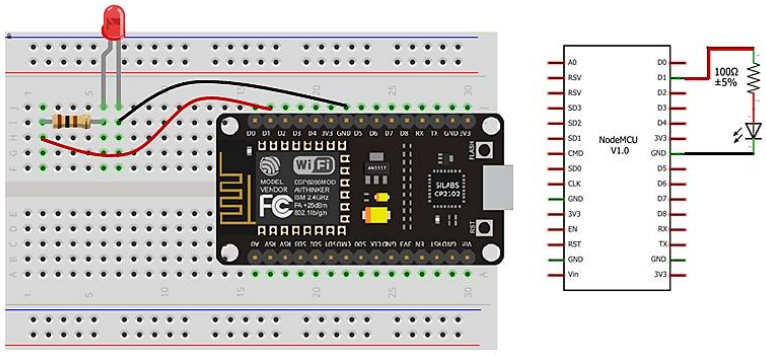


Figura 31: Circuito LED NodeMCU

Dado que la señal proporcionada por NodeMCU es de 3,3 V, es necesario conectar una resistencia en el circuito.

4.3.2. Lectura de la señal analógica de un fotorresistor

```
void setup(){

 Serial.begin(9600);

}

void loop(){

 Serial.println(analogRead(A0));
 delay(500);

}
```

Para la lectura de las señales analógicas, NodeMCU cuenta con un solo pin, el pin A0:

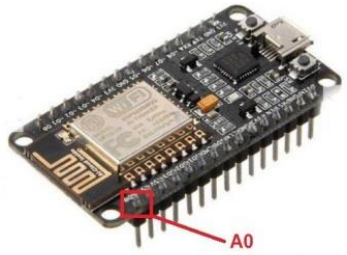


Figura 32: Pin analógico A0 NodeMCU

El pin analógico permitirá la lectura de valores dentro de un rango. Este rango está determinado por la resolución del conversor ADC, esto es, el conversor analógico digital.

En el caso de NodeMCU, el pin analógico permite la lectura de un rango entre 0 y 3,3 V, y su resolución es de 10 bits. Por lo tanto, la lectura que obtendremos en dicho pin estará comprendida 0 y 1023.

Dentro del Sketch se utiliza la función “AnalogRead(A0)” para la lectura de la señal analógica en el pin A0. De forma análoga, se utiliza la función “DigitalRead()” para la lectura de señales digitales en los distintos pines.

En el Sketch del ejemplo, la función AnalogRead() se anida dentro de la función “Serial.println()”. La función Serial.println comunica la placa con el PC, de forma que la placa le manda la información contenida dentro de la función como argumento, y esta información puede ser visualizada en el monitor serie del IDE.

Podemos acceder al monitor serie desde el IDE, en la pestaña Herramientas\Monitor serie. Esta herramienta resulta útil para visualizar la diferente información y variables con la que está trabajando la placa. A su vez, gracias al monitor serie, se puede enviar información a la placa desde el ordenador en tiempo real.

Para que esta comunicación se pueda llevar a cabo, es necesario iniciar previamente en el Sketch la función “Serial.begin()”, que tiene como argumento la velocidad en Baudios.

En este caso, debido a que el fotorresistor utilizado es el modelo GL5516, la conexión se realiza de la siguiente forma:

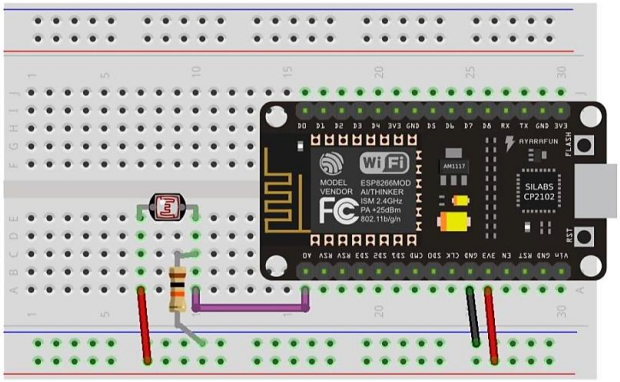


Figura 33: Circuito fotorresistor NodeMCU

4.3.3. Lectura de la señal digital de un PIR

Para la lectura de señales digitales, NodeMCU cuenta con 13 pines numerados del D0 al D12. La lectura que registrará NodeMCU solo tendrá dos estados, HIGH o LOW, que en este caso se corresponderán con las salidas del sensor PIR.

Dentro del Sketch, la función DigitalRead() será la que haga posible la lectura de esta señal, utilizando como argumento el pin del que queremos registrar la señal. Antes es necesario definir el pin al que está conectado el PIR como INPUT, mediante la función PinMode().

A la hora de conectar el PIR a NodeMCU, si el PIR está montado sobre un circuito de soporte, dispondrá de tres conexiones: toma de corriente, toma de tierra y salida digital.

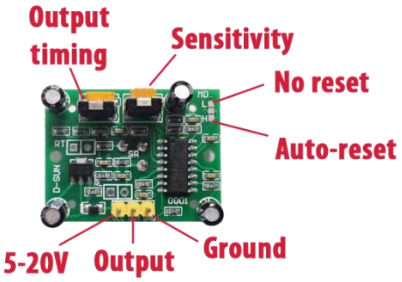


Figura 34: Elementos del circuito PIR HC-SR501

La conexión quedaría de la siguiente manera:

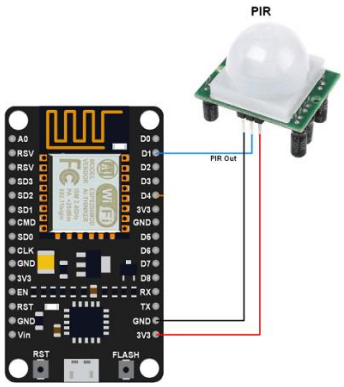


Figura 35: Conexión PIR a NodeMCU

4.3.4. Control de un servomotor

```
#include<Servo.h>

Servo servo;

void setup(){

 servo.attach(14);//Pin D5
 servo.write(0);
 delay(2000);

}

void loop(){

 servo.write(90);
 delay(1000);
 servo.write(0);
 delay(1000);

}
```

Este Sketch muestra cómo controlar un servomotor mediante NodeMCU. En este caso, el servo efectuará un movimiento de 0° a 90° de forma intermitente. Para ello, el IDE de Arduino cuenta con una librería denominada Servo.h. Esta librería se incluirá al inicio del Sketch mediante el comando #include.

A continuación, se declara un objeto Servo, nombrado “servo” en este caso. Este objeto se vinculará o enlazará a uno de los pines mediante la función attach(). Esta vinculación se realiza dentro del Setup, y en este caso al pin 14, que corresponde al D5. También dentro del Setup se posicionará el servo en su posición inicial, en este caso 0°, mediante la función write(), y esperará 2 segundos antes de entrar en la función Loop.

Una vez dentro de la función Loop, mediante dos funciones write(), separadas por delays de 1 segundo, se conseguirá el posicionamiento intermitente del servo entre las posiciones 0° y 90°.

El servomotor que he utilizado es el Micro Servo modelo SG90 del fabricante Osoyoo.

La conexión del servomotor a NodeMCU se realiza mediante tres conexiones: la entrada de potencia, la salida a tierra y la señal de control.

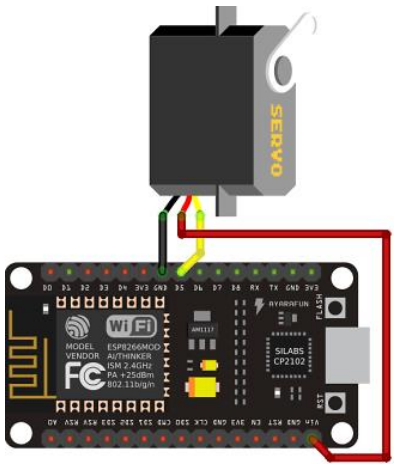


Figura 36: Circuito Servo NodeMCU

La particularidad del control de un servomotor mediante NodeMCU se encuentra en dicha señal de control. Los servomotores utilizan la modulación por ancho de pulsos (PWM) para controlar su posición o dirección. Este control es simplificado gracias a la librería que incluye Arduino, denominada Servo.h.

4.4. Conexión WiFi con NodeMCU

Los distintos aparatos conectados a una red WiFi son denominados estaciones. La conexión a la red se realiza a través de un punto de acceso, que actúa como centro para varias estaciones. Normalmente, los puntos de acceso se integran dentro de un router, que permite el acceso a Internet a través de la red WiFi.

La conexión WiFi mediante NodeMCU permite tres distintos modos de configuración, que son como estación, como punto de acceso, y el modo combinado estación más punto de acceso. Estas distintas configuraciones son posibles gracias a las librerías WiFi creadas para el módulo ESP8266.

Figura 37: Librería ESP8266WiFi.h

La librería ESP8266WiFi toma como base la nomenclatura y filosofía funcional de la librería WiFi de Arduino. La librería consta de un compendio de funciones y propiedades en lenguaje C++ para configurar y operar el módulo ESP8266 tanto en modo estación como punto de acceso. Es de destacar que esta librería no es exclusiva de NodeMCU, sino que es general para todas las placas que tengan el módulo ESP8266 integrado.

4.4.1. Estación

El modo estación es usado para hacer que el módulo ESP8266 se conecte a la red WiFi a través de un punto de acceso ya establecido.

```
#include<ESP8266WiFi.h>
IPAddress staticIP145_39(192,168,1,39);
IPAddress gateway145_39(192,168,1,1);
IPAddress subnet145_39(255,255,255,0);
WiFiServer server(80);
Constchar ssid[]="XXXXX";
Constchar password[]="xxxxx";

void setup(){
WiFi.disconnect();
 delay(3000);
 WiFi.config(staticIP145_39,gateway145_39,subnet145_39);
 WiFi.begin(ssid,password);
 while((!(WiFi.status()==WL_CONNECTED))){
 delay(300);
 }
server.begin();
}
```

Para ello, dentro del Sketch, se incluye en primer lugar la librería ESP8266WiFi.h. Esta incluye funciones en sus diferentes clases que permiten configurar y operar la conexión WiFi de módulo ESP8266.

A continuación, se establece el número de IP que usará NodeMCU. De esta forma se consigue que cuente con IP estático, y con ello la IP no variará cada vez que se reinicie el módulo. Esto es importante, ya que la conexión entre módulos se llevará a cabo realizando peticiones a la IP correspondiente.

Dado que el módulo se conectará a un punto de acceso establecido, es necesario indicar el nombre de la red a la que se va a conectar y, en su caso, la contraseña de ésta. Para ello se declaran dos cadenas de caracteres que contienen dicha información para conectarse, por ejemplo, a la red WiFi doméstica.

Esto puede ser un inconveniente para un usuario que quiera conectar su NodeMCU a distintas redes, o que no disponga del IDE de Arduino y su conexión a NodeMCU, ya que para cambiar la información de la red a conectarse sería necesario acceder al Sketch, desde ahí cambiar la información, y volver a subir el programa. Este inconveniente es solucionado en el siguiente apartado, usando NodeMCU en primer lugar como punto de acceso.

Siguiendo con el Sketch, estableceremos la IP previamente descrita mediante la función WiFi.config, y se inicia la conexión a la red doméstica mediante la función WiFi.begin(), que tiene como argumentos el nombre y contraseña de la red. Si en la función WiFi.begin() no se introducen argumentos, se conectará al último punto de acceso usado.

Posteriormente, gracias a un bucle While, nos aseguraremos de que NodeMCU se haya conectado a la red. NodeMCU estará ya preparado para enviar y recibir información a través de dicha red.

Es de destacar que no es necesario ningún elemento extra al que conectar físicamente NodeMCU para realizar la conexión WiFi, ya que éste cuenta con los elementos necesarios para realizar la conexión.

ESP8266 operating in the **Station** mode

Figura 38: Esquema de conexión nodo-estación

La clase estación tiene varias características que facilitan la administración de la conexión WiFi. En caso de que se pierda la conexión, el módulo ESP8266 se reconectará automáticamente al último punto de acceso usado, una vez que se encuentre disponible. Esto es posible gracias a que el ESP guarda las credenciales utilizadas para acceder al último punto de acceso en la memoria Flash.

4.4.1.1. Conexión WPS

Además de los habituales protocolos de seguridad WEP y WPA, ESP8266 es capaz de conectarse a la red WiFi utilizando el protocolo WPS (*WiFi Protected Setup*), aunque sólo en la configuración con pulsado de botón. Esto consiste en pulsar un botón físico en el router que permite durante unos minutos la conexión de otros dispositivos.

La configuración WPS fue creada en 2006 con el objetivo de ofrecer a los usuarios de redes domésticas la posibilidad de establecer accesos WiFi protegidos, a la vez que conectar de forma sencilla nuevos dispositivos a dicha red.

Para realizar esta conexión con NodeMCU, hay que introducir la función WiFi.begin WPSConfig() en el Sketch, y pulsar el botón correspondiente en el router.

Este método no es recomendable ya que permite conectarse a la red sin introducir la contraseña, así que utilizaré la conexión mediante contraseña WPA.

4.4.2. Punto de acceso

Un punto de acceso es un dispositivo que permite el acceso a una red WiFi a otros dispositivos, denominados estaciones. A su vez, éstos son conectados a Internet a través de una red cableada. El ESP8266 tiene una función similar pero no idéntica, ya que éste no permite por sí mismo el acceso a la red cableada. A este modo de funcionamiento se le denomina “Soft Access Point”.

Para el caso de este proyecto, el modo punto de acceso servirá como primer paso en el que el usuario podrá introducir el nombre y contraseña de la red doméstica sin tener que modificar el Sketch. De esta forma, al iniciarse NodeMCU, en primer lugar iniciará un punto de acceso, al cual el usuario podrá conectarse desde su ordenador o teléfono móvil. Al usuario se le indicará al acceder que tiene que introducir la información de la red, y una vez introducida NodeMCU pasará del modo punto de acceso al modo estación y se conectará a la red doméstica. Este procedimiento será ilustrado con más detalle en posteriores apartados.

```
#include<ESP8266WiFi.h>

WiFiServer server(80);
Constchar ssid[]="XXXXX";
Constchar password[]="xxxxx";
HTTPClient http;

void setup(){
Serial.begin(9600);
server.begin();
WiFi.mode(WIFI_AP);
WiFi.softAP(ssid,password);
Serial.print("Direccion IP Access Point - por defecto: ");
Serial.println(WiFi.softAPIP());
Serial.print("Direccion MAC Access Point: ");
Serial.println(WiFi.softAPmacAddress());
}

void loop(){

 WiFiClientclient=server.available();
 if(!client){
 return;
 }
}
```

En este Sketch se destaca el uso de las funciones WiFi.mode y WiFi.softAP. Mediante la primera función, configuramos el modo de funcionamiento de NodeMCU como punto de acceso, y posteriormente mediante WiFi.softAP iniciamos dicho punto de acceso. El nombre de la red será el introducido previamente en el Sketch, dentro de la cadena de caracteres SSID, y la contraseña para acceder a dicha red será la información de la cadena PASSWORD.

Gracias al uso de NodeMCU como punto de acceso, se consigue mejorar la seguridad de la red, ya que para acceder a la red de NodeMCU será necesario conocer su contraseña.

ESP8266 operating in the **Soft Access Point** mode

Figura 39: Esquema de la conexión con el punto de acceso

Es importante mencionar que el máximo número de estaciones que se pueden conectar al soft-AP puede ser fijado desde 0 a 8, y por defecto está en 4.

4.4.3. Red en malla

El ESP8266 no sólo tiene la capacidad de actuar en modo estación o soft-AP, sino que es capaz de actuar en ambos modos de forma simultánea. Esto hace posible al módulo ESP8266 actuar como nodo de una red en malla.

Una red en malla es una topología de red, donde cada nodo está conectado al resto de nodos. Con esto se consigue la posibilidad de enviar información de un nodo a otro utilizando diferentes caminos. Así, al no requerir un nodo central, se reduce el riesgo de fallos.

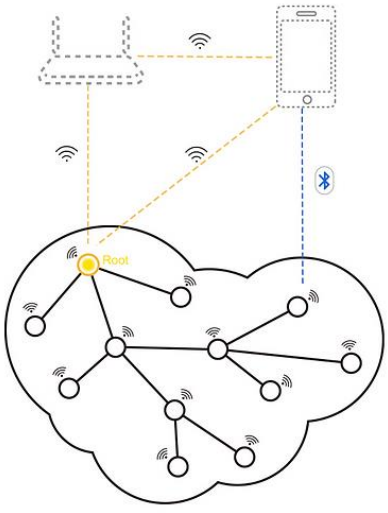


Figura 40: Red en malla

Las redes en malla también encuentran aplicaciones dentro de la domótica, u otras aplicaciones dentro de aparcamientos, fábricas u oficinas, donde los sensores con conectividad WiFi también puedan actuar como nodos de la red.

En este proyecto no utilizo esta configuración del ESP8266, sino que me limito a los modos estación y soft-ap, ya que los nodos de la red estarán conectados al router de la red WiFi doméstica.

4.4.4. Conexión Bluetooth y comparativa

La comunicación inalámbrica entre dispositivos no se limita a la comunicación WiFi. Arduino es compatible, por ejemplo, con la comunicación Bluetooth. La placa Arduino Uno no cuenta con esta característica integrada, por lo que sería necesario conectar un módulo Bluetooth, de la misma forma que se conecta el módulo WiFi ESP01, explicado anteriormente. Los módulos bluetooth son de la serie HC, por ejemplo, el HC-05, HC-08 o HC-12.

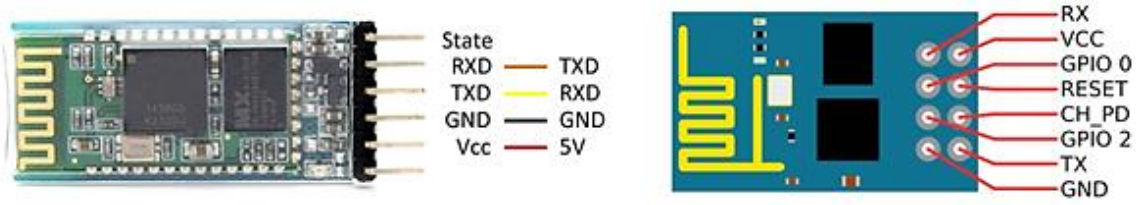


Figura 41: Comparativa pinout módulo HC-05 (Bluetooth) (izquierda) y módulo ESP01 (WiFi) (derecha)

La conexión se realiza mediante el uso de un puerto serie de la placa Arduino. Esta conexión utiliza los pines TXD (pin de transmisión) y RXD (pin de recepción). Dentro del Sketch de Arduino tendremos que programar la comunicación serie. Otros dispositivos, como Raspberry Pi, se pueden adquirir con Bluetooth ya integrado.

En la descripción de los dispositivos y sus características, he nombrado en varios de ellos que soportan protocolos 802.11 b/g/n. Esto se corresponde a unos estándares para redes del IEEE (Instituto de ingenieros eléctricos y electrónicos). El 802.11 se corresponde al WiFi, y b/g/n indica la revisión, son actualizaciones en la velocidad de transmisión y la banda de trabajo. El estándar correspondiente al Bluetooth es el 802.15.

Bluetooth es una especificación industrial para redes inalámbricas de área personal, desarrollado por la compañía Bluetooth Special Interest Group. Su utilidad es la transmisión de datos entre distintos dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz. Los primeros dispositivos en incorporar este protocolo de comunicación fueron los teléfonos móviles, a los que se han ido añadiendo ordenadores portátiles y sus periféricos (tablets) o televisiones, entre otros.

El Bluetooth se clasifica en distintas clases en función de su potencia y alcance:

- Bluetooth Clase 3: Potencia de 1 mW y un metro de alcance.
- Bluetooth Clase 2: Potencia de 2,5 mW máxima y entre 5 y 10 metros de alcance. Aquí se encuentran los dispositivos más comunes.
- Bluetooth Clase 1: Potencia de 100 mW y cien metros de alcance.

Con respecto al WiFi, existen dispositivos WiFi con un alcance mayor a 100 metros, aunque no es recomendable utilizar un alcance mayor, debido a que estará expuesto a un riesgo de interferencias.

Comparando la tecnología WiFi con la Bluetooth en su frecuencia de operación, ambas tienen capacidad para operar en la banda de 2,4 GHz. A mayores, el WiFi dispone desde 2013 de un

estándar (802.11ac) que le permite operar en la banda de 5 GHz, reduciendo el riesgo de interferencias, pero con un alcance menor.

Los dispositivos Bluetooth también se pueden clasificar por su capacidad de canal, de entre 1 Mbit/s y 50 Mbit/s. Su velocidad es por tanto menor que la del WiFi, que se mueve en un rango de entre 11 Mbit/s y 300 Mbit/s.

Por otra parte, si queremos seleccionar una u otra tecnología por su compatibilidad y accesibilidad a otros dispositivos, nos encontraremos con que los dispositivos actuales, como Smartphone, ordenadores, tablets o televisores, disponen de ambos modos de comunicación. En este aspecto, es de destacar que la tecnología WiFi, tanto por su velocidad como por su alcance, se ha posicionado por encima del Bluetooth respecto a la conectividad a Internet. Aunque es posible la conexión a Internet mediante Bluetooth, este se encuentra más enfocado a la comunicación local entre dispositivos. Así, el Bluetooth también entra dentro del ámbito de aplicación del IoT.

De esta forma, el último punto a favor de la tecnología WiFi, además de su velocidad y alcance respecto al Bluetooth, es que gran parte de los hogares cuentan con un router como centro de una red WiFi doméstica. Por tanto, podremos conectar los dispositivos de la red de este proyecto a un mayor número de hogares.

Además de la conexión WiFi y Bluetooth, Arduino cuenta con otros dispositivos para diferentes tipos de conexión:

- **Sigfox:** red para dispositivos que necesiten un bajo un consumo, utiliza la banda de radio para aplicaciones industriales, médicas y científicas (ISM). En Europa esta banda corresponde a los 868MHz.
- **GSM / 3G:** es la red que utilizan los teléfonos móviles (Global System for Mobile Communications). Ya que esta red cubre gran parte del mundo, es interesante cuando no sea posible la conexión con otro tipo de red. Además, utilizando el dispositivo junto con una tarjeta SIM es posible mandar mensajes SMS o realizar llamadas.
- **Narrowband:** es la red de bajo consumo diseñada para operar en redes de teléfonos móviles y LTE (Long Term Evolution), un estándar para la comunicación a alta velocidad. También se destaca por poder utilizarse en aplicaciones que requieran un bajo consumo o una situación remota.

Figura 42: Módulos MKR FOX 1200 (izquierda), MKR GSM 1400 (centro) y MKR NB 1500 (derecha) de Arduino

4.5. Listado y descripción de las funciones utilizadas

Para llevar a cabo la comunicación mediante WiFi entre módulos NodeMCU, utilizo funciones ya existentes integradas dentro de librerías disponibles online. A continuación, describiré las más utilizadas, así como en qué situación utilizarlas.

Funciones pertenecientes a la clase servidor:

- **WifiServer server(80):** crea un objeto servidor, que escucha las comunicaciones a través del puerto que tiene como argumento. Utilizo el puerto 80 porque es el puerto por defecto por el que los navegadores web se comunican con los servidores.
  Añado esta función en el código después de incluir las librerías y antes de la función setup(), junto con la declaración del resto de variables universales. Esto es así ya que es la función previa a cualquier comunicación WiFi, sin la cual no puede tener lugar. Del mismo modo existe la función **WifiClient()**, que crea un cliente capaz de conectarse a la IP y puerto especificados.

- **Server.begin():** inicializa el servidor para que comience a escuchar las peticiones entrantes. Previamente se ha de crear un objeto servidor mediante WifiServer. Esta función no tiene argumentos, y la añado dentro de la función setup().

- **Server.available():** Devuelve un cliente que se ha conectado al servidor y tiene datos preparados para su lectura. Por lo tanto en el código para iniciar la comunicación, creo en primer lugar un objeto client, y le asigno un valor mediante server.available. Si no hay un cliente conectado reinicio el bucle y espero a que haya un cliente conectado.

```
WiFiClient client = server.available();
if (!client) { return; }
```

Funciones pertenecientes a la clase cliente:

- **WifiClient()**: Como he indicado anteriormente, crea un objeto client.

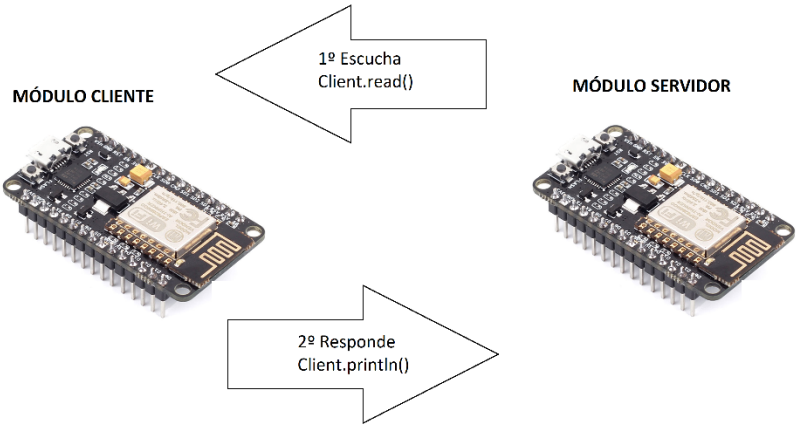
- **Client.available():** Devuelve el número de bytes disponibles para su lectura, escritos en el cliente correspondientes a la respuesta del servidor al que se ha conectado.

- **Client.readStringUntil('\r'):** Lee la cadena de texto correspondiente a client.available(). Utilizo como argumento '\r', que se corresponde al caracter de control de carro, lo que me permite leer la cadena completa. Si utilizase la función **Client.read()** tendría que ir leyendo byte a byte. El paso en la comunicación inmediatamente posterior al anteriormente descrito, sería de la siguiente manera:

```
while(!client.available()){  delay(1); }
resactuador = (client.readStringUntil('\r'));
```

  Esto es, compruebo que hay información disponible. Si no la hay, espero, y en cuento haya información leo la cadena completa y la almaceno en una variable String.

- **Client.println():** Envía datos al servidor al que el cliente se ha conectado. Dicho de otra forma, si **Client.read()** es la escucha, **Client.println()** es la contestación. De forma esquemática, el código en el módulo cliente seguiría esta estructura:



- **Client.stop():** Desconecta del servidor.

- **Client.flush():** Descarta los bytes escritos en el cliente, pero no leídos. Junto con client.stop() utilizo esta función para marcar el fin de la comunicación.

Además de las funciones de comunicación, se han de utilizar funciones para configurar el modo de funcionamiento del WiFi y las conexiones a red. A continuación, describo tales funciones:

- **WiFi.mode():** Establece el modo de funcionamiento del módulo utilizando los siguientes argumentos: (WIFI_STA) para modo estación, (WIFI_AP) para punto de acceso, y (WIFI_AP_STA) para estación y punto de acceso.

- **WiFi.config():** Permite establecer una IP estática, y cambiar las direcciones DNS, gateway y subnet. Utilizo esta función ya que los módulos lanzarán peticiones a IP concretas para llevar a cabo la comunicación, y de este modo les asigno una IP estática para que las peticiones se lancen de forma correcta.

- **WiFi.begin():** Utilizo esta función para conectarme a una red WiFi existente. Tiene como argumentos el nombre de la red, y la contraseña.

- **WiFi.softAP():** Con esta función el módulo comienza a trabajar como punto de acceso creando su propia red. El nombre de la red y la contraseña son los argumentos de esta función.

- **WiFi.disconnect():** Desconecta el módulo de la red WiFi a la que está conectado.

- **Wifi.status():** función sin argumentos, indica el estado de la conexión. Se utiliza para asegurarse que la conexión se ha establecido antes de continuar con el programa, usándola en un bucle while de la siguiente forma:

```
while ((!(WiFi.status() == WL_CONNECTED))){
   delay(300);}
```

Por último, para el módulo interfaz, utilizo funciones de librería ESP8266WebServer. Esta librería contiene funciones más complejas que permiten programar aplicaciones más avanzadas, especializándose en el manejo de peticiones HTTP. Utilizo principalmente tres funciones:

- **Server.on():** asocia las funciones de callback que se ejecutan al recibir la petición en una URL determinada. Estas funciones callback son argumentos de otras funciones y se ejecutan dentro de estas. Dicho de otra forma, al recibir una petición con la URL "ejemplo", se ejecutará el código dentro de la función Server.on("ejemplo"). En el código sería de la siguiente forma:

```
server.on("/ejemplo", [](){

  Serial.println(ejemplo);
   delay(1000);

 });
```

- **Server.HandleClient():** espera las peticiones de los clientes y lanza las funciones de callback. Se ha de ejecutar de forma continua dentro de Loop (). Este será todo el Loop() dentro del módulo interfaz:

```
void loop(void){

  server.handleClient();

 }
```

- **Server.send():** envía la respuesta al cliente como HTTP, es decir, línea inicial, cabecera y cuerpo. Éstos serán sus argumentos.

Para utilizar en el código las funciones listadas en este apartado, es necesario incluir al inicio del código las librerías correspondientes. Estas librerías fueron instaladas en el pluggin, pero no basta con instalarlas si no que deben estar explícitas en el código:

```
#include <WiFiClient.h>
#include <ESP8266HTTPClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266WiFi.h>
```

4.6. Comunicación WiFi entre módulos NodeMCU

Gracias a la gran comunidad de usuarios detrás de proyectos con plataformas de software libre, es posible encontrar una amplia cantidad de información. La información respecto al manejo de sensores y actuadores con estos módulos se encuentra de forma amplia en la web. De forma algo más reducida, es posible encontrar información respecto a la comunicación WiFi entre un solo módulo y el usuario, por ejemplo, la monitorización del estado de los sensores conectados a un único módulo NodeMCU desde el ordenador o dispositivo móvil.

El ámbito donde menos información disponible hay es precisamente en la creación de redes con estos módulos y cómo realizar la comunicación entre ellos y, simultáneamente, con el usuario. En este aspecto, es donde con este proyecto creo poder aportar un pequeño grano de arena al desarrollo y evolución de este campo.

4.6.1. Protocolo HTTP

La comunicación entre módulos se realizará mediante el protocolo HTTP. El protocolo de transferencia de hipertexto está orientado a transacciones de información, y se basa en el esquema petición-respuesta entre un cliente y un servidor. Esto es importante, ya que encaja con la estructura de la red de este proyecto, donde un conjunto de módulos clientes recogerán la información de los sensores, que será enviada a un módulo principal o servidor, y éste les dará respuesta. Así, el flujo de información ha de ser bidireccional. Además, HTTP permite la comunicación a través de Internet.

Como su nombre indica, HTTP utiliza mensajes de tipo texto. Esto hace que los mensajes sean más legibles a costa de hacerlos más largos. Así, la información que necesitemos enviar de un NodeMCU a otro deberá estar codificada dentro de cadenas de texto.

A la hora de programar, las cadenas de texto pueden ser representadas de dos formas. Se puede usar el tipo de dato String o se puede crear una cadena de texto encadenado datos del tipo Char (caracteres individuales). Los caracteres son almacenados como números, correspondientes al código ASCII.

Ejemplo de declaración de variables “char” y “String”:

```
char myChar = 'A';
String stringOne = "HelloString";
```

Dentro del Sketch podremos hacer toda clase de operaciones con las cadenas de texto. Esto permite una gran flexibilidad a la hora de trabajar con la información en forma de texto. Por ejemplo, se pueden sumar cadenas, extraer subcadenas, recortarlas, buscar información concreta dentro de la cadena, realizar operaciones de comparación e incluso pasarlas a formato numérico.

Dicho de otra forma, las cadenas de texto son vectores que guardan información en cada uno de sus componentes. Por lo tanto, manejar la información de los sensores y actuadores de la red en forma de texto es de por sí una buena opción, a lo que hay que añadir la ventaja de que dichas cadenas pueden ser transmitidas mediante HTTP.

Para poder transmitirse mediante HTTP, los mensajes han de seguir una estructura concreta:

- Línea inicial: aquí se debe incluir el método de petición en el caso de que el mensaje sea una petición o el código de respuesta si el mensaje se trata de una respuesta.
- Cabecera: se compone de metadatos, es decir, datos que describen los datos que se expondrán a continuación.
- Cuerpo del mensaje: contendrá los datos que se intercambian entre servidor y cliente.

A continuación, describiré cómo he aplicado estos tres puntos a la hora de crear mensajes para comunicar los distintos módulos NodeMCU de la red.

4.6.2. Construcción de los mensajes

En el apartado de línea inicial, dentro de los métodos de petición, HTTP define un conjunto predefinido de métodos que se pueden utilizar. El método define la acción que se pretende ejecutar sobre el recurso identificado. Como método de petición he usado el método GET. Mediante este método, al lanzar una petición, estoy enviando un mensaje del nodo 1 al nodo 2, a la vez que, con ese mismo mensaje, el nodo 1 le está pidiendo datos al nodo 2 y este se los entrega. Así consigo realizar transmisión de información de forma bidireccional, tanto del nodo 1 al nodo 2 como del nodo 2 al nodo 1, utilizando una sola petición.

Siguiendo en el apartado de línea inicial, las respuestas a las peticiones las he codificado con el código 200. Los códigos de la serie 2xx se utilizar para indicar que las peticiones se han realizado correctamente.

Las cabeceras se utilizan para describir los datos del mensaje. En este caso, como el contenido de los mensajes es texto, utilizo la cabecera “Content-Type: text/html”.

El contenido del mensaje será la información que quiero transmitir de un nodo a otro. Por una parte, la información que deben enviar los nodos cliente al servidor es la información recogida por los sensores. Como esta información ha de recogerse en cadenas de texto, la lectura de los sensores, tanto digitales como analógicos, las guardaré en variables tipo String.

```
String sen1;
sen1 = digitalRead(D4);
```

Dentro del Sketch, defino variables String para los sensores, tantas como el número de sensores conectados a ese nodo. Si ese nodo tiene conectados tres sensores, se definirían tres variables String: “sen1”, “sen2” y “sen3”.

A la hora de hacer la lectura del sensor, guardo la información directamente en la variable que corresponde al sensor conectado al pin correspondiente.

A continuación, uniré la información de todos los sensores en un único String. Para hacer esto, concateno los String “sen1”, “sen2”, “sen3”… poniendo delante de cada cadena un pequeño código en forma de texto (S1, S2, S3…) para diferenciar la información de cada sensor dentro del mensaje final.

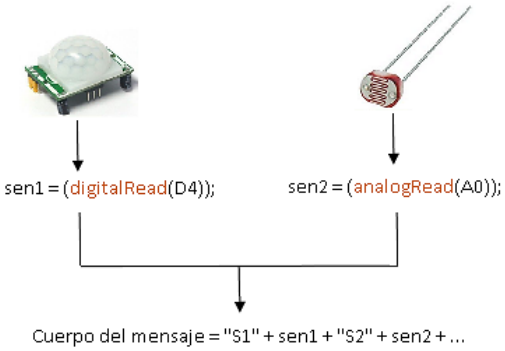


Figura 43: Esquema de construcción de un mensaje

De esta forma ya consigo unir en un solo String toda la información recogida por los sensores unidos a un NodeMCU.

Este mismo esquema lo utilizo para recoger la información de los actuadores. La información que mandará el nodo servidor a los nodos clientes con la señal a transmitir a los actuadores se codificará como “A1” + act1 + “A2” + act2 + … donde A1 y A2 son texto, y act1 y act2 son las variables String donde se almacena la información de cada actuador.

```
client.println("HTTP/1.1 200 OK");              //LINEA INICIAL
client.println("Content-Type: text/html");      //CABECERA
client.println("");
client.println(respuesta);                      //CUERPO
client.stop();
delay(1);
client.flush();
```

Una vez tenemos completa la estructura del mensaje, es decir, línea inicial, cabecera, y cuerpo, se puede responder a la petición GET.

Para lanzar la petición GET, utilizo las siguientes funciones:

- **http.begin(httpurl):** Inicia la conexión. Como argumento se ha de indicar la URL de la conexión, en este caso yo uso la IP del dispositivo a conectarse y el mensaje.
- **http.GET():** Realiza la petición a la dirección indicada en http.begin.
- **http.getString():** Devuelve la respuesta del cliente como una variable string.
- **http.end():** Finaliza la conexión.

El módulo que lanzó la petición GET recibirá la respuesta. Ese módulo podrá extraer la información correspondiente a cada sensor o a cada actuador del String completo, gracias a la codificación realizada anteriormente, y el uso de operaciones con variables String.

```
sensor1=lectura.substring(lectura.lastIndexOf("S1")+2,lectura.indexOf
("S2"));
```

Por ejemplo, si queremos extraer la información del sensor 1, en primer lugar, se busca dentro de la cadena de texto el código “S1”, y el código “S2”, con las funciones IdexOf. Posteriormente, gracias a la función SubString, se recorta esa parte de la información que queremos. Así, se extrae de la misma forma la información de un sensor analógico que de uno digital, independientemente del número de dígitos de esa información. Del mismo modo aplicamos esto al resto de sensores y a los actuadores.

4.6.3. Estructura Cliente-Servidor

Una vez conseguida la conexión WiFi de cada módulo NodeMCU a la red doméstica, el siguiente paso es la comunicación entre ambos. La información transmitida entre los distintos NodeMCU que compongan la red será la correspondiente a la lectura de sensores y las señales para los actuadores que se encuentre de dicha red.

La comunicación seguirá un esquema servidor-cliente o nodo central-nodos auxiliares. En este esquema el módulo cliente estará conectado a diferentes sensores y actuadores. El módulo servidor recogerá la información del estado de los sensores que le envíen los diferentes clientes vía WiFi. Una vez recibida la información de los sensores, establecerá la señal que corresponda para cada actuador y enviará dicha información a los clientes. Por último, estos clientes ejecutarán sobre sus actuadores la señal recibida desde el servidor.

La justificación de esta estructura es aportar flexibilidad física a la red. Es decir, los sensores y actuadores se distribuirán por el hogar de acuerdo con las necesidades del usuario, y también los NodeMCU a los que se conectan. Así, puede ser que un actuador conectado a un NodeMCU ubicado en una zona de la casa se active cuando varíe la señal de un sensor ubicado en la zona opuesta de la casa. Dicho de otra forma, la distribución de sensores y actuadores en el hogar será diferente para cada usuario y la red estándar necesita poder adaptarse y ser capaz de comunicar sensores y actuadores independientemente de su ubicación.

Por tanto, es necesario un nodo central que administre toda la información de la red, la recoja y la distribuya a los diferentes nodos. Además, es conveniente reunir toda la información en un nodo, para que ésta pueda ser monitorizada por el usuario.

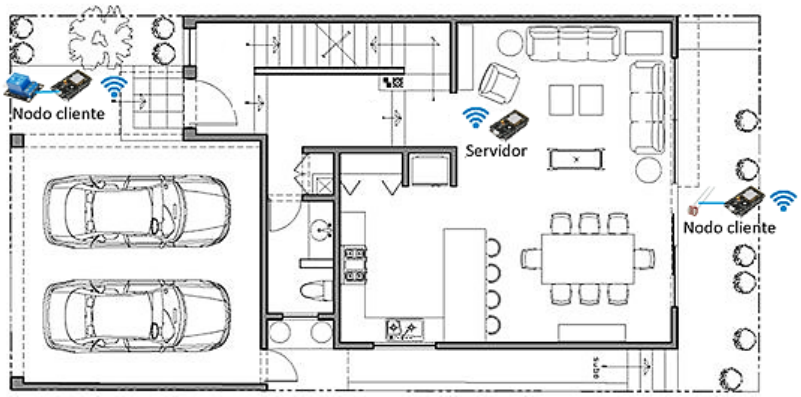


Figura 44: Distribución NodeMCU en una vivienda

4.6.3.1. Servidor

El servidor se encarga de recoger la información de los distintos sensores, conectados a los clientes, y a su vez devolver a los clientes la información sobre las señales que deben enviar a los actuadores que tengan conectados. Con esto se consigue unificar en un módulo la gestión de toda la información de la red. Esto hace posible que un actuador ubicado en una zona concreta actúe en relación con diferentes sensores ubicados en otros lugares, que pueden estar próximos entre sí o no.

El módulo servidor será el que realice las peticiones a los clientes. El módulo servidor "preguntará" a los clientes qué información han recogido los sensores. En el contenido de la propia petición se incluye la señal que deben enviar los clientes a los actuadores.

Así, el servidor lanzará peticiones a las IP de cada uno de los clientes, a la vez que les manda la información de los actuadores. Esta información sobre los actuadores no es específica para cada cliente y sus actuadores específicos, sino que toda la información de los actuadores se incluye en una única línea de texto y se envía a todos los clientes, independientemente del número o tipo de actuadores que tengan conectados.

Dicho de otra manera, el servidor realiza la misma pregunta a todos los clientes, incluyendo en la propia pregunta información que necesitan los clientes sobre sus actuadores. Así, los clientes no necesitan "preguntar" o lanzar peticiones. Esto es así porque si fuesen los clientes los que lanzasen la petición al servidor, llegarían al servidor un gran número de peticiones simultáneas que no sería capaz de responder. Así, un único nodo es que el que se encarga de administrar la información y decidir con quién y cuándo comunicar un mensaje.

Para esta gestión de la comunicación cliente-servidor, dentro del Sketch servidor he creado una función que tiene como argumentos la información que manda el servidor y la IP de nodo al que mandar dicha información y que devuelve la información recogida en el cliente.

```
String TheAnswerOfClient;
String SendWithAnswer(String IPcache, String message){
httpurl="http://";
httpurl+=IPcache;
httpurl+="/";
httpurl+=message;
http.begin(httpurl);
http.GET();
TheAnswerOfClient=(http.getString());
http.end();
return TheAnswerOfClient;
}
```

Para ello, utiliza el protocolo HTTP. La función construye un String con los argumentos declarados y con el método GET lanza la petición a la IP indicada. La petición debe ser incluida entre las llamadas http.begin y http.end. Mediante la función http.getString() la función recoge la información devuelta por el cliente.

He utilizado este método como una función, ya que una vez declarada en Void() se puede utilizar en Loop() las veces que sea necesaria, manteniendo limpio el Sketch. Además, aporta mayor facilidad a la hora de introducir más módulos cliente a la red: esto solo supone utilizar una vez más esta función, poniendo como argumento la IP del nuevo módulo añadido.

Esta función se ejecutará dentro del bucle Loop() al menos una vez por cada módulo cliente. El retorno de la función, es decir, la información enviada por el nodo cliente, está codificada para diferenciar la señal de cada sensor dentro de una cadena de texto. Esto es, la señal del sensor 1 estará precedida dentro de la cadena por “S1”, la señal del sensor 2 estará precedida por “S2”, y así sucesivamente. De esta forma se consigue segregar la señal de cada sensor mediante la función String.substring().

Será dentro del módulo servidor donde se procese la información de los sensores y se calcule la señal de los actuadores. Aquí se programará la lógica del sistema. En el Sketch del servidor es donde se habrá de programar si el “actuador 7” se enciende cuando esté activo el “sensor 3”, por ejemplo. Se utilizará la lógica necesaria, por ejemplo, funciones “If”, o “while”, según los sensores y actuadores que formen parte de la red.

Finalmente, el servidor construirá el String con la información que los clientes han de mandar a los actuadores, en función de la señal recibida de los sensores. Se codificará de similar forma a la información de los sensores, esto es, la señal del actuador 1 estará precedida por “A1”, la señal del actuador 2 por “A2”, y así sucesivamente. Este String será uno de los argumentos que se usen en la función de comunicación servidor-cliente.

4.6.3.2. Cliente

La función de los clientes será recoger la información de los sensores, transmitirla al servidor y mandar la señal a los actuadores en función del mensaje enviado desde el servidor.

Dado que el servidor estará ejecutando la función de comunicación en bucle, como he explicado anteriormente, el cliente estará esperando la petición desde el servidor mediante el siguiente bucle:

```
WiFiClientclient = server.available();
if(!client){return;}
while(!client.available()){ delay(1);}
```

Si la conexión es exitosa, el cliente saldrá de este bucle para responder al servidor mediante protocolo HTTP. Se leerá la petición del servidor mediante la función client.readStringUntil('\r'). Y en este caso el código de respuesta enviado al servidor será el 200, es decir, la respuesta estándar para peticiones correctas. A continuación, enviará al servidor la información en forma de String con la señal recogida mediante los sensores.

```
resactuador=(client.readStringUntil('\r'));
```

El cliente ya tiene entonces la información que deberá enviar a los actuadores que tenga conectados. Como se ha indicado anteriormente, segregará la información de cada actuador mediante la función string.substring().

En este Sketch se utilizan varios métodos de la clase Client. La clase Client permite crear clientes capaces de conectarse a servidores, y enviar y recibir datos.

4.6.3.3. Esquema del código de comunicación entre cliente-servidor

En los apartados anteriores he descrito las funciones utilizadas para la comunicación. Estas funciones se han de ordenar de forma correcta tanto el código del cliente como en el del servidor. Así, por ejemplo, la ejecución de la función de petición por parte de un módulo debe coincidir con la ejecución de la función de escucha del otro módulo. A continuación, utilizo un esquema para aclarar el orden de ejecución.

MÓDULO CLIENTE

MÓDULO SERVIDOR

En primer lugar, el módulo cliente debe estar esperando la conexión. Para asegurarse de que esté ejecutando esta función, la incluyo al inicio de Loop(), y reinicio el bucle mientras no se haya realizado la conexión.

```
WiFiClient client = server.available();
if (!client) { return; }
while(!client.available()){ delay(1); }
```

El módulo servidor entonces lanzará la petición.

```
http.begin(httpurl);
http.GET();
```

El cliente sale del bucle, lee la petición y responde.

```
client.readStringUntil('\r');
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html");
client.println("");
client.println(respuesta);
```

El servidor recoge la respuesta del cliente.

```
http.getString();
```

Para finalizar, ambos módulos finalizan la conexión.

```
client.stop();
client.flush();
```

```
http.end();
```

4.7. Punto de acceso y conexión a red doméstica

En este apartado ilustraré cómo el usuario puede conectar NodeMCU a su red doméstica introduciendo la información desde su ordenador o teléfono móvil.

Este punto se plantea con dos motivaciones. En primer lugar, es un paso que aporta seguridad a la red, ya que la conexión a la red estará protegida mediante dos contraseñas, una para la conexión a NodeMCU y otra para la conexión a la red doméstica. La segunda motivación es evitar al usuario tener que modificar el Sketch para introducir los datos de su propia red doméstica, esto es, ser capaz de configurar la conexión de NodeMCU a tu propia red doméstica de una forma sencilla mediante el teléfono móvil u ordenador con conexión WiFi.

La estructura de programación para conseguir este propósito se basa en una serie de páginas HTML que son enviadas como respuesta a la petición realizada por el usuario mediante su dispositivo móvil. Estas páginas HTML permiten una forma visual e intuitiva de introducir la información de la red doméstica.

El usuario se conectará al punto de acceso de NodeMCU desde el navegador de su teléfono móvil u ordenador. Entonces realizará peticiones con el método GET mediante el navegador de su dispositivo. Si la conexión es exitosa, NodeMCU recogerá y guardará la información incluida en la petición y responderá con una nueva página HTML al usuario. Así, se introducirá el nombre y contraseña de la red doméstica y aparecerá una última página donde el usuario confirmará la información introducida.

Para ello, en primer lugar, se configurará NodeMCU como punto de acceso:

```
WiFi.mode(WIFI_AP);
WiFi.softAP(ssid,password);
```

La función WiFi.softAP() tiene como argumentos el nombre y contraseña de la red que creará NodeMCU.

El usuario entonces podrá conectarse a la red WiFi de NodeMCU desde otro dispositivo.

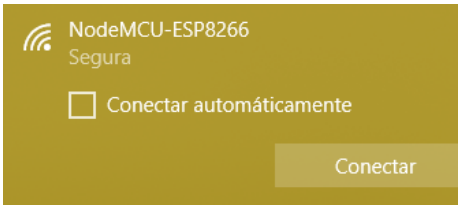


Figura 45: Conexión a punto de acceso desde ordenador portátil

NodeMCUesperará la conexión del usuario mediante el bucle descrito anteriormente:

```
WiFiClientclient=server.available();
if(!client){
return;
}
Serial.println("nuevo cliente");
while(!client.available()){
delay(1);
}
```

Una vez conectado, el usuario tendrá que introducir la IP por defecto de NodeMCU en el navegador de su dispositivo. Esta IP por defecto es 192.168.4.1. En la figura inferior muestro esta IP por el monitor serie mediante las funciones serial.print() y WiFisoftAPIP(). NodeMCU responderá a la petición del usuario, y éste podrá ver en la pantalla de su dispositivo un

mensaje de confirmación. El funcionamiento de la comunicación es el mismo que en el caso cliente-servidor. Ahora es el usuario el que lanza la petición, y el módulo interfaz responde mediante la función client.println.

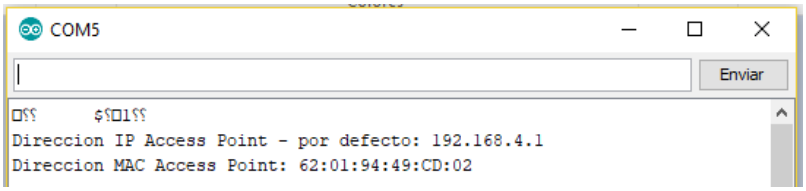


Figura 46: IP por defecto mostrada en monitor serie

Este mensaje aparecerá en la pantalla del navegador del dispositivo móvil u ordenador gracias a una pequeña página HTML programada en el Sketch de NodeMCU, descrita a continuación:

```
 client.println("HTTP/1.1 200 OK");
 client.println("");
 client.println("<!DOCTYPE HTML>");
 client.println("<meta charset='UTF-8'>");
 client.println("<html>");
 client.println("<p>Por favor introducir nombre red</p>");
 client.println("<form> Nombre red: <input type='text'
name='NombreRed' value=''><br>");
 client.println("<input type='submit' value='Enviar'></form>");
 client.println("</html>");
```

La página resultante de este código se ilustra en la figura a continuación:

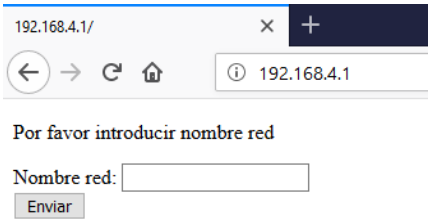


Figura 47: Página enviada por NodeMCU una vez realizada la petición a la IP por defecto

El esquema de estas páginas HTML es sencillo, consta de una línea de texto donde se solicita la información, un cuadro de texto donde el usuario introducirá la información y un botón con el que el usuario lanzará la petición.

NodeMCU guardará la información introducida en el cuadro de texto en una variable String, de la misma forma que en la comunicación cliente-servidor, con la función client.readStringUntil('\r'). El código para esto es el siguiente:

```
if (pascontra == 1)
{
   rede = client.readStringUntil('\r');

        if (rede != "GET /favicon.ico HTTP/1.1" && rede
!= "GET / HTTP/1.1"){

         red = rede.substring(rede.lastIndexOf("GET
      /")+16,rede.indexOf("HTTP/1.1")-1);
   }
}
```

En primer lugar, tengo un condicional que marca en qué paso de la introducción de datos se encuentra el usuario. “Pascontra == 1” indica que aún no introdujo ni el nombre de la red ni la contraseña, “Pascontra == 2” indica que introdujo el nombre de la red pero no la contraseña, y “Pascontra == 3” indica que ya introdujo ambos datos y sólo falta el paso de confirmación de datos.

Hago esto ya que debo utilizar varias veces la función client.read(), una para almacenar en un string el nombre de la red, otra para la contraseña, y otra para la confirmación. Como el código se ejecuta en bucle, ese primer condicional ha sido mi forma de diferenciar los pasos introducción de datos.

El siguiente condicional *if (rede != "GET /favicon.ico HTTP/1.1" && rede != "GET / HTTP/1.1")* lo utilize para confirmer que se ha introducido el nombre de la red y no se ha dejado el campo vacío. Esas líneas de texto aparecen cuando se deja el campo de introducción de datos vacío.

Por último, extraigo el nombre de la red de la cadena de texto. Ya que el protocolo HTTP utiliza línea inicial, cabecera y cuerpo, extraigo el texto que me interesa (en este caso el nombre de la red) mediante la función string.substring(), y marco la posición de la información relevante con la función string.indexOf().

De la misma forma, el usuario podrá introducir la contraseña de su red WiFi doméstica. Esto lo he conseguido declarando variables de texto distintas para el nombre de la red y la contraseña. Si la variable de la red está vacía, NodeMCU responderá a las peticiones a su IP con la página HTML preguntando por el nombre de la red. Si se envía una petición correcta dentro de esta página, NodeMCU guardará la información en la variable de texto correspondiente a la red. Una vez la variable del nombre de la red contenga información, la siguiente vez que se envíe una petición a la IP de NodeMCU, responderá con una página HTML preguntando por la contraseña.

Esto lo llevo a cabo mediante condicionales, y dentro de cada condicional la página HTML correspondiente:

- String “red”, que almacena el nombre de la red, vacío, mostrará al usuario un mensaje que para que introduzca el nombre de la red

```
    if (red == ""){
  client.println("HTTP/1.1 200 OK");
  client.println("");
  client.println("<!DOCTYPE HTML>");
  client.println("<meta charset='UTF-8'>");
  client.println("<html>");
  client.println("<p>Por favor introducir nombre red</p>");
  client.println("<form> Nombre red: <input type='text'
name='NombreRed' value=''><br>");
  client.println("<input type='submit' value='Enviar'>
</form>");
  client.println("</html>"); }
```

- String “red” no está vacío y string “pass” que almacena la contraseña está vacío, mostrará al usuario un mensaje para que introduzca la contraseña. Además, marco el inicio del siguiente paso igualando a 2 la variable “pascontra”.

```
      if (red != "" && pass == ""){
  client.println("HTTP/1.1 200 OK");
  client.println("");
  client.println("<!DOCTYPE HTML>");
  client.println("<meta charset='UTF-8'>");
  client.println("<html>");
  client.println("<h1>El nombre de la red es: ");
  client.println(red);
  client.println("");
  client.println("Por favor introduce la contraseña</h1>");
  client.println("<form> Password: <input type='text'
name='Password' value=''><br>");
  client.println("<input type='submit' value='Enviar'>
</form>");
  client.println("</html>");
  pascontra = 2;}
```

- String “pass” no está vacío, mostrará al usuario un mensaje para que confirme los datos introducidos. También marco el inicio del paso 3 (Pascontra == 3).

```
       if (pass != ""){
  client.println("HTTP/1.1 200 OK");
  client.println("");
  client.println("<!DOCTYPE HTML>");
  client.println("<meta charset='UTF-8'>");
  client.println("<html>");
  client.println("<h1>El nombre de la red es: </h1>");
  client.println(red);
  client.println("");
  client.println("<h1>La contraseña es:</h1>");
  client.println(pass);
  client.println("");
  client.println("<p>Si los datos son incorrectos, por favor
resetee el controlador. De lo contrario introduzca OK </p>");
  client.println("<p><a href='OK'>Confirmacion OK</a></p>");
  client.println("</html>");

  pascontra = 3;
   }
```

Figura 48: Página de confirmación de datos introducidos en NodeMCU

Una vez introducida la red y contraseña, el usuario confirmará dicha información. NodeMCU entonces finalizará su conexión WiFi como punto de acceso e iniciará una nueva conexión en modo estación. En modo de conexión estación, NodeMCU se conectará entonces a la red WiFi doméstica gracias a la información transmitida previamente por el usuario. El código es el descrito a continuación:

```
red.toCharArray(redy, 50);
pass.toCharArray(passy, 50);
WiFi.disconnect();
delay(2000);
Serial.println("START");
WiFi.begin(redy, passy);
while ((!(WiFi.status() == WL_CONNECTED))){
  delay(300);
  Serial.print("..");

}
Serial.println("Connected");
Serial.println("Your IP is");
Serial.println((WiFi.localIP().toString()));
server.begin();
Serial.println("Se conecto a casa");
```

El código de conexión al WiFi doméstico es idéntico al descrito en el apartado “Estación”, excepto porque en primer lugar, paso los string del nombre de la red y de la contraseña a un array de caracteres. Hago esto porque la función WiFi.begin() no trabaja con argumentos string, si no con array de caracteres. En segundo lugar utilizo la función WiFi.disconnect() para finalizar el modo de funcionamiento como punto de acceso, ya que a partir de entonces NodeMCU trabajará como estación. El resto del código ya ha sido explicado con anterioridad.

El conjunto de páginas HTML y el proceso de introducción de la información se encuentran dentro del Loop() del Sketch. Esto es así porque es necesario ejecutar la función de gestión de peticiones (client.read()) varias veces, es decir, gestionar la petición inicial, la introducción de red y contraseña, y la confirmación final.

Cuando una petición es exitosa y el usuario ha introducido información, por ejemplo, el nombre de la red, utilizo el comando Return, para volver al inicio del bucle. Al inicio del Loop(), NodeMCU volverá a estar esperando la petición.

```
WiFiClientclient= server.available();
if(!client){
return;
}
```

Es importante que esta función se encuentre al inicio del Loop(). De esta forma se consigue que NodeMCU esté de forma constante esperando a las peticiones. De lo contrario, si esta función se colocase en otra parte de Loop(), NodeMCU no estará de forma continua esperando por peticiones, y entonces las peticiones que se le lancen podrían no captarse porque NodeMCU se encuentre en ese momento ejecutando otras líneas de código.

La conexión a la red doméstica se hará también dentro de Loop(). Una vez conectado de forma exitosa a la red doméstica, NodeMCU entrará en el bucle de funcionamiento normal, es decir, en el de cliente si es un nodo cliente o en el de servidor si es un nodo servidor. Esto es así porque he dividido Loop() en dos bucles, un primer bucle de introducción de la información por el usuario, del que saldrá una vez conectado a la red doméstica, y un segundo bucle que será el que se esté ejecutando de forma constante, donde está programado su funcionamiento normal.

Para el segundo bucle, el de funcionamiento habitual, he utilizado la función goto por motivo de claridad a la hora de visualizar el código, pero se también se puede utilizar un condicional if.

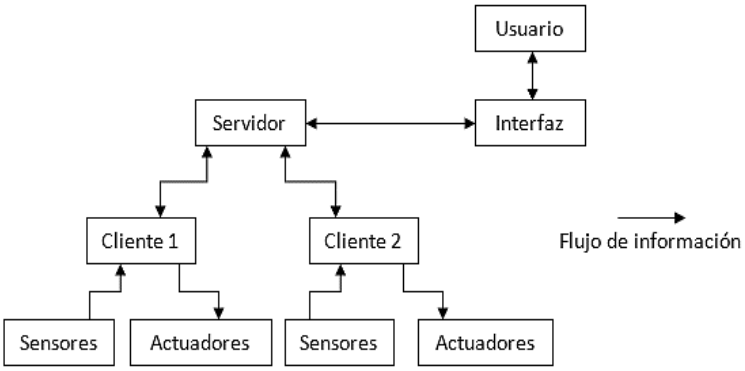
```
bucle:

lectura = (SendWithAnswer("192.168.1.39", respuestaa));
    sensorr1 =
lectura.substring(lectura.lastIndexOf("S1")+2,lectura.index
Of("S2"));
    sensorr2 =
lectura.substring(lectura.lastIndexOf("S2")+2);

  Serial.println(lectura);
  Serial.println(sensorr1);
  Serial.println(sensorr2);
  respuestaa = "";
  respuestaa+="A1";
  respuestaa+=sensorr1;
  respuestaa+="A2";
  respuestaa+=sensorr2;
  Serial.println(respuestaa);
  delay(50);

goto bucle;
```

En el código se puede ver el uso de la función goto, se establece un marcador en el código y al llegar a goto la ejecución del código vuelve al marcador.

4.8. Interfaz red-usuario

Este punto surge de la necesidad de que el usuario sea capaz de interactuar con la red. La red es capaz de funcionar de forma automática sin interacción por parte del usuario mediante la transmisión de información entre sus elementos, sensores, actuadores y placas NodeMCU. Pero para poder modificar el modo de funcionamiento de dicha red, el usuario debería modificar individualmente los Sketch de cada NodeMCU y cargarlos en su placa correspondiente. Esto sería poco práctico y no funcional.

Para ello he desarrollado una interfaz en la que el usuario podrá monitorizar, configurar y actuar sobre la red de forma sencilla desde un dispositivo con conexión WiFi, como por ejemplo su teléfono móvil u ordenador portátil. El usuario podrá, desde el navegador de su teléfono móvil, comprobar el estado de sensores y actuadores, manejar manualmente los actuadores de la red y configurar modos de funcionamiento de la red.



El Sketch de interfaz se cargará en un módulo NodeMCU individual en lugar de ser incluido en el módulo servidor. Esto es así porque su estructura de programación será diferente a la utilizada en los módulos cliente y servidor, y de este modo no interferirá en el funcionamiento automático de la red. De hecho, en el módulo interfaz, el grueso de Sketch estará dentro de la función SetUp() y la función Loop() sólo contará con una línea de código, como ilustraré a continuación.

4.8.1. Comunicación usuario-interfaz-servidor

Las peticiones enviadas al módulo interfaz se dividen en dos tipos: las peticiones que realiza el usuario a la interfaz y las peticiones que realiza el servidor a la interfaz.

En el caso del usuario, las peticiones permiten dos resultados diferentes: el cambio de configuración y la interacción manual del usuario con los elementos de la red.

En cuanto al cambio de modo de funcionamiento o configuración de la red, el usuario podrá cambiar el modo de funcionamiento de la red a modo automático o a modo manual. De la misma forma podrá establecer diferentes modos preestablecidos, como por ejemplo un modo "noche" o un modo "vacaciones", que serán unos modos automáticos en los que se modifique el output hacia los actuadores dependiendo de la señal recogida por los sensores. Esto es, por ejemplo, en el modo noche se podrían encender determinadas luminarias que en el modo estándar no sería necesario encender.

Por otra parte, el usuario podrá actuar de forma manual sobre los actuadores de la red si ha establecido previamente el modo manual. En modo manual, el usuario enviará peticiones al nodo interfaz, del tipo "encendido" o "apagado" de los distintos actuadores, e incluso peticiones más complejas, por ejemplo, para modificar la posición de un servomotor. Además, la interfaz mostrará mediante páginas HTML la información recogida por los sensores.

La comunicación entre usuario e interfaz también se llevará a cabo mediante páginas HTML. El nodo interfaz se encontrará de forma continua esperando peticiones. Si se lanza una petición a su IP sin información adicional, el nodo interfaz responderá con la página HTML principal. Esta página cuenta con botones y con texto para mostrar información.

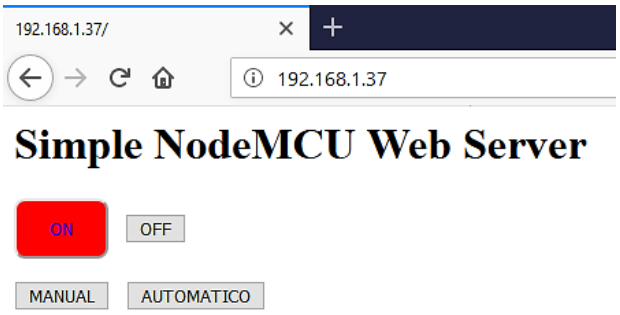


Figura 49: Página básica de la interfaz

Los botones permiten modificar los modos de funcionamiento, y también manejar de forma manual los actuadores. Pulsando los diferentes botones, el usuario realiza peticiones a NodeMCU. NodeMCU registra la información enviada mediante peticiones por el usuario, pero es el módulo servidor y no el interfaz el que gestiona la información completa de la red. Así que además de intercambiar información con el usuario, el nodo interfaz se comunicará de forma constante con el nodo servidor. Será el nodo servidor el encargado de lanzar peticiones, de la misma forma que envía peticiones a los nodos cliente.

Por una parte, el servidor enviará a la interfaz la información sobre el estado de los sensores. Esta información sobre el estado de los sensores es utilizada por el nodo interfaz para mostrársela al usuario.

Por otra parte, el servidor realizará peticiones del tipo "modo", es decir, preguntará a la interfaz en qué modo debe trabajar la red. En caso de que la red deba trabajar en modo manual, porque así lo haya indicado previamente el usuario, el nodo interfaz indicará al servidor la señal que debe enviarse a los actuadores. En modo automático, la señal enviada a

los actuadores depende directamente de la señal recogida por los sensores. En el modo manual, la señal de los actuadores dependerá de las peticiones previas que haya realizado el usuario, independientemente de la información recogida por los sensores.

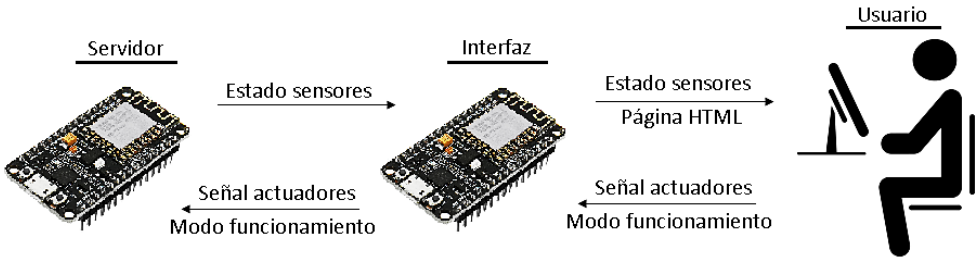


Figura 50: Comunicación Red-Usuario

Es decir, en el nodo servidor estarán programados varios modos de funcionamiento. El servidor lanzará peticiones periódicas al nodo interfaz, preguntando por el modo en el que debe trabajar.

4.8.2. Estructura del Sketch

En el Sketch del módulo interfaz, la función SetUp() incluirá un conjunto de instrucciones que se ejecutarán bajo la condición de que el usuario o servidor realice determinadas peticiones a la IP del módulo interfaz. Esto se consigue mediante la función server.on(). Cuando mediante una petición haya una conexión exitosa a NodeMCU, se ejecutará el código que esté dentro del server.on() correspondiente a la petición realizada. La respuesta del nodo interfaz a las peticiones que le envíen se realiza a través de la función server.send. Esta función tiene tres argumentos, que son los ya descritos dentro del apartado del protocolo HTTP: línea inicial, cabecera, y cuerpo del mensaje.

Antes, es importante diferenciar el comando del objeto servidor. En el código de los módulos cliente y servidor, el objeto se creaba mediante el comando WiFiServer(). Para el módulo interfaz utilizo una librería distinta, por lo que el comando es ESP8622WebServer(). Del mismo modo lo incluyo antes de la función set up().

```
ESP8622WebServer server(80);
```

En los siguientes extractos del código de la interfaz, se ilustran dos ejemplos de la función server.on(). En la primera de ellas, el código que contiene se ejecutará si al módulo interfaz se le envía la petición “manual”. La petición del cambio a modo manual la realizará el usuario a través de la página web HTML que crea el propio NodeMCU interfaz. Una vez enviada la petición, la variable “modo” recogerá el string “manual”, se construirá una página HTML a partir de información previa, y enviará dicha página mediante la función server.send().

```
server.on("/manual",[](){
modo="manual";

pagcom="";
pagcom+=page;
pagcom+="<p>";
pagcom+=modo;
pagcom+="</p>";
server.send(200,"text/html",pagcom);
Serial.println(modo);
delay(1000);
});
```

De la misma forma ocurrirá si el usuario envía la petición “automático”, como se ilustra a continuación.

```
server.on("/automatico",[](){
modo="automatico";

pagcom="";
pagcom+=page;
pagcom+="<p>";
pagcom+=modo;
pagcom+="</p>";

server.send(200,"text/html",pagcom);
Serial.println(modo);
delay(1000);
});
```

En este fragmento de código se puede observar cómo cuando el usuario realiza la petición “LEDOn” se ejecutará lo recogido dentro de la función server.on(LEDOn), esto es, el cambio de información de determinadas variables, el envío de la página HTML con la información actualizada y el cambio de estado de uno de los pines.

```
server.on("/LEDOn",[](){

estado="encendido";
enviomanualS1="S11";

pagcom="";
pagcom+=page;
pagcom+="<p>";
pagcom+=estado;
pagcom+="</p>";

server.send(200,"text/html",pagcom);
digitalWrite(LEDSetPin,HIGH);
Serial.println(estado);
delay(1000);
});
```

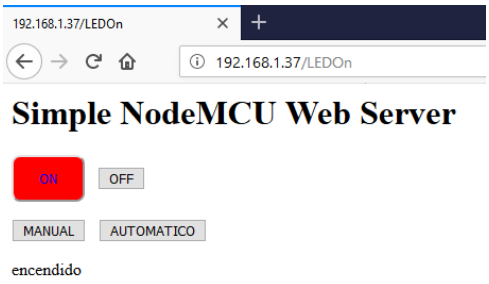


Figura 51: Interfaz con petición LED On tras pulsar "ON"

Para construir la página HTML se suman diferentes variables de tipo String partiendo de un String base donde se recogen los elementos invariantes de dicha página. Estos serán texto y su estilo, botones de acción, distribución de los elementos en distintos párrafos, etc. De esta forma y agrupando todos los elementos comunes en un string base, la modificación de la página HTML será automática cada vez que se actualice la información de las variables string que la componen.

```
 page="<style> .button{ background-color:red; color:blue; border-radius:
8px; padding: 12px 24px; }</style><h1>Simple NodeMCU Web
Server</h1><p><a href=""LEDOn""><buttonclass=""button"">ON</button></a>
  <a href=""LEDOff""><button>OFF</button></a></p><p><a
href=""manual""><button>MANUAL</button></a>  <a
href=""automatico""><button>AUTOMATICO</button></a></p>";

        server.on("/LEDOn",[](){

 estado="encendido";
 pagcom="";
 pagcom+=page;
 pagcom+="<p>";
 pagcom+=estado;
 pagcom+="</p>";

 server.send(200,"text/html",pagcom);
 });
```

Toda esta gestión de peticiones por parte de la interfaz se realiza a través de la función server.handleClient(), que será la única función dentro de Loop(), y por tanto se estará ejecutando de forma continua.

```
 Void loop(void){
 server.handleClient();
 }
```

El registro de las diferentes peticiones se hará a través de un conjunto de variables declaradas al inicio de Sketch, y la conexión de la interfaz a la red WiFi doméstica se realizará de forma idéntica a la explicada anteriormente para los módulos clientes y servidor.

Por otra parte, el módulo servidor también lanzará peticiones al módulo interfaz. De forma periódica lanzará una petición preguntando por el modo de funcionamiento. El siguiente fragmento de código corresponde a la respuesta de la interfaz, simplemente devuelve el modo de funcionamiento asignado previamente por el usuario, guardado en el string “modo”.

```
    server.on("/modo", [](){
       server.send(200, "text/html", modo);
       delay(100);
     });
```

El módulo servidor pasará a trabajar con condicionales. Si el string “modo” es igual a “automatico”, el servidor trabajará con el código explicado anteriormente en su apartado. Si el string “modo” es igual a “manual”, el servidor enviará una petición a la interfaz preguntando por el estado de los actuadores. Este fragmento del código pertenece al servidor:

```
   if (modo == "manual"){
    datosmanual = (SendWithAnswer("192.168.1.37",
"datosmanual"));
    sensorm1 =
datosmanual.substring(datosmanual.lastIndexOf("S1")+2,
datosmanual.indexOf("S2"));
    sensorm2 =
datosmanual.substring(datosmanual.lastIndexOf("S2")+2);
  respuestaa = "";
  respuestaa+="A1";
  respuestaa+=sensorm1;
  respuestaa+="A2";
  respuestaa+=sensorm2;
  delay(50);}
```

A continuación, detallo el orden de ejecución del código en un esquema, para que resulte más fácil su compresión.

MODULO INTERFAZ

MÓDULO SERVIDOR

El usuario envía la petición a la interfaz para que la red trabaje de forma manual. La interfaz gestiona la petición con server.handleClient() y almacena “manual” en el string “modo”.

```
void loop(void){
 server.handleClient();}

  server.on("/manual", [](){
   modo = "manual";

   pagcom = "";
   pagcom += page;
   pagcom += "<p>";
   pagcom += modo;
   pagcom += "</p>";

   server.send(200, "text/html", pagcom);

   delay(100);});
```

El servidor pregunta de forma periódica (una vez por bucle) a la interfaz el modo de funcionamiento.

```
    modo = (SendWithAnswer("192.168.1.37", "modo"));
```

La interfaz gestiona la petición handleClient(), y le responde el string “modo”, en el que previamente se guardó la información “manual”.

```
void loop(void){
 server.handleClient();}

    server.on("/modo", [](){

     server.send(200, "text/html", modo);

     delay(100);});
```

El servidor guarda en su string “modo” la información “manual”. Entrará en el bucle de funcionamiento manual, y preguntará a la interfaz los datos de los actuadores (“datosmanual”).

```
if (modo == "manual"){
datosmanual = (SendWithAnswer("192.168.1.37", "datosmanual"));
```

La interfaz gestiona la petición y le responde el string “enviomanual”, donde se guarda la información de todos los actuadores.

```
server.on("/datosmanual", [](){

enviomanual = enviomanualS1 + enviomanualS2;
server.send(200, "text/html", enviomanual);

delay(100);});
```

Previamente el usuario mandó las peticiones para modificar manualmente el estado de los actuadores, y la interfaz guardó esa información en los string “enviomanualS1” y “enviomanualS2”.

El servidor extrae del string “datosmanual” la información correspondiente a cada uno de los actuadores, y prosigue su funcionamiento normal.

```
   sensorm1 =
datosmanual.substring(datosmanual.lastIndexOf("S1")+2,
datosmanual.indexOf("S2"));
    sensorm2 =
datosmanual.substring(datosmanual.lastIndexOf("S2")+2)
;
```

Aclaro que el nombre de las variables utilizadas en los sketch es arbitrario.

En resumen, el servidor pregunta cada bucle a la interfaz por el modo de funcionamiento. En modo automático, el servidor pregunta a cada cliente por el estado de sus sensores y en función de esa información les da la orden de los actuadores. En modo manual, el servidor pregunta a la interfaz por la orden de los actuadores. Sigue preguntando a cada cliente por el estado de sus sensores, pero la orden de los actuadores ya no está en función del estado de los sensores, si no de la respuesta que recibió de la interfaz.

Del mismo modo que he programado entre modo manual y automático, se podría programar un modo “vacaciones” o modo “noche”, en función de las necesidades del usuario.

4.9. Conexión remota

Dado que las diferentes placas que componen la red WiFi estarán conectadas a una red doméstica, es posible acceder a esta red desde cualquier lugar a través de internet.

Para ello el usuario desde una conexión remota, por ejemplo, desde su teléfono móvil conectado a una red distinta a la red doméstica, realizará la petición a la IP pública del router de la red doméstica, especificando el puerto de conexión entre el router y el módulo NodeMCU. La unión de estos dos parámetros, la IP pública y el puerto de comunicación, se conoce como socket.

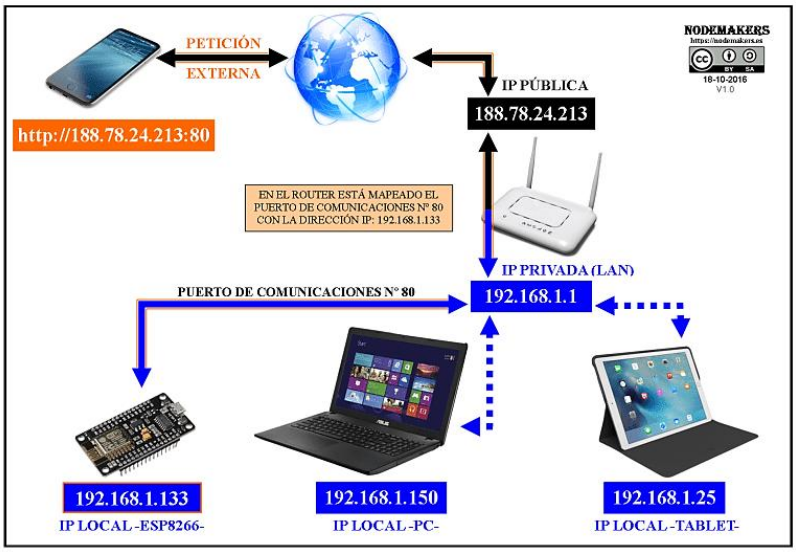
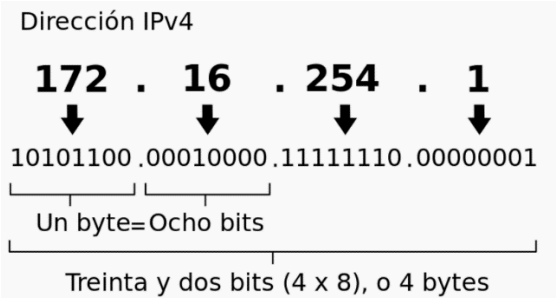


Figura 52: Conexión a NodeMCU vía Internet

Las direcciones IP (Internet Protocol) son códigos numéricos que se utilizan para identificar a los clientes de una red y así poder dirigir paquetes de información IP entre ellos. Actualmente se utiliza el protocolo IPv4, donde las direcciones se expresan como un entero de 32 bit, por lo tanto, admite un total de 2^{32} direcciones. Normalmente estas direcciones se expresan como números en decimal separados por puntos, de la siguiente forma:



Es importante también conocer la diferencia entre IP pública e IP privada. La IP pública identifica a la red doméstica desde el exterior, y la determina el proveedor de Internet. La IP privada identifica a cada uno de los dispositivos conectados a la red doméstica. A estos dispositivos con IP privada no nos podemos conectar de forma remota, excepto mediante el socket.

Esto se consigue habilitando una conexión entre la información que llega al router de forma remota y la IP local con la que se pretende conectar. Dicho de otra forma, consiste en reenviar la información que llega a determinado puerto del router a una IP local de un dispositivo

conectado a dicho router. A este proceso se le denomina mapeado de puertos. En este caso, sólo será necesario habilitar la conexión a un módulo NodeMCU, el módulo interfaz.

El mapeado en cada router depende de su fabricante. De forma genérica, en primer lugar hay que acceder al router a través de un dispositivo conectado a la red local, introduciendo la dirección de acceso en el navegador (192.168.1.1).

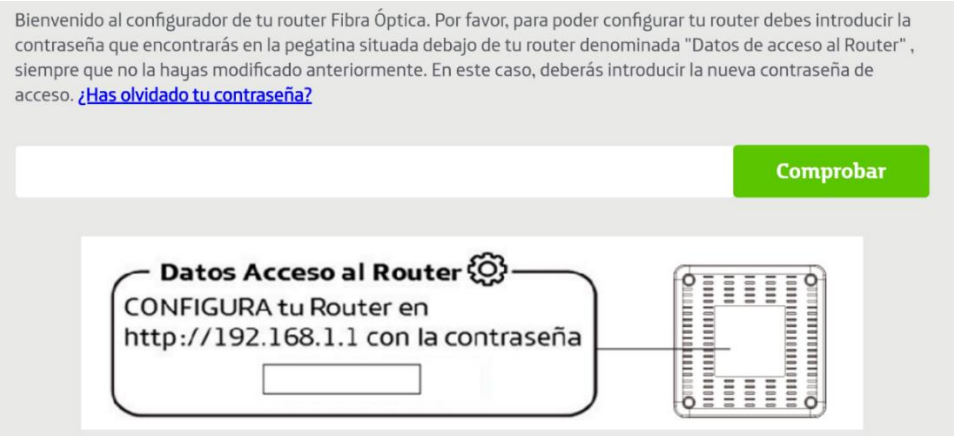


Figura 53: Página de configuración del router, IP 192.168.1.1

Una vez en la página de configuración del router, nos dirigimos a la pestaña de aplicaciones/puertos. En esta página crearemos una nueva aplicación, en la que elegimos el protocolo de comunicación y el puerto. Finalmente, vinculamos la IP local del módulo NodeMCU interfaz al puerto que elegimos previamente.

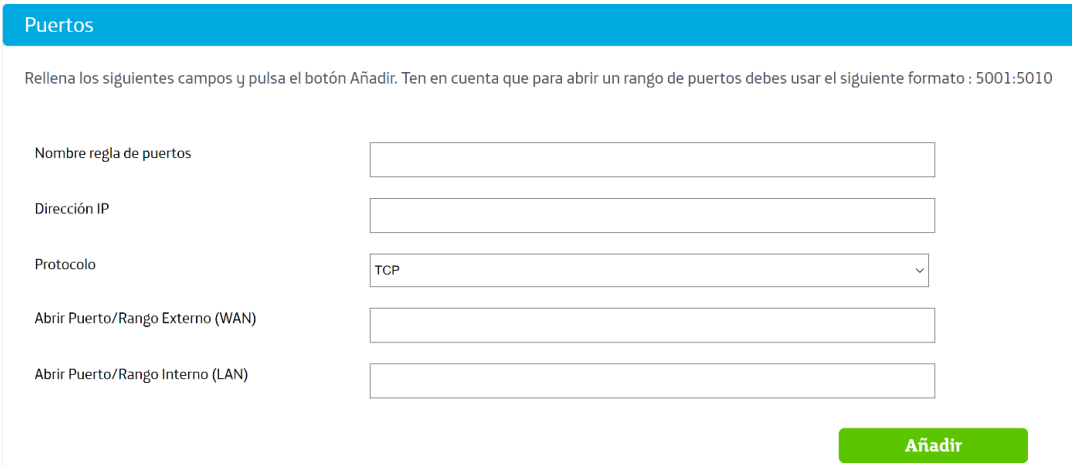


Figura 54: Página de configuración del router, Puertos

Una vez finalizada la configuración, para poder acceder a NodeMCU desde Internet, sólo tendremos que introducir en el navegador la IP pública y el puerto, por ejemplo 83.52.163.57:80.

Mediante este sistema es posible acceder a nuestra red doméstica por Internet, pudiendo configurar los modos de funcionamiento de la red, monitorizar el estado de los sensores, o actuar sobre los actuadores desde cualquier lugar.

Es importante tener en cuenta que al conectarse desde cualquier lugar, no es necesario conocer la clave de la red doméstica para acceder a la red, por lo que a nivel de seguridad habría que establecer una contraseña a mayores para las conexiones remotas, y no hacer pública la IP de acceso a la red.

5. VIABILIDAD ECONÓMICA Y COMPARATIVA

Como he comentado al inicio de esta memoria, las grandes compañías tecnológicas (Apple, Alphabet, Microsoft o Amazon) lanzan con más frecuencia campañas de marketing para promocionar sus productos dentro del ámbito de la domótica. Venden kits, normalmente solo compatibles con sus propios dispositivos, con los que se pueden automatizar diversas funciones del hogar.

La red que he desarrollado también es capaz de automatizar distintas funcionalidades gracias a que puede adaptar el uso de diferentes sensores o actuadores. Además, como sistema de control se utilizan las placas NodeMCU, capaces de comunicarse entre ellas y con el usuario mediante comunicación WiFi. El uso de este sistema de software y hardware libre hace posible adaptar la programación de estas placas a las distintas necesidades del usuario.

A continuación, ilustraré diferentes aplicaciones dentro del ámbito de la domótica, y la comparativa entre las soluciones comerciales y las soluciones que aporta este proyecto mediante código abierto.

- **Sistema de control de riego**

La compañía Apple, dentro de su gama Apple HomeKit, ofrece un "programador de riego inteligente EveAqua que convierte tu toma de riego en una válvula de salida que puedes abrir y cerrar automáticamente con el iPhone, Siri o el botón integrado." Se puede adquirir desde 99,95€.

Figura 55: Programador de riego Apple HomeKit

Se puede implantar este mismo sistema mediante aplicaciones de código libre, utilizando la placa NodeMCU como controlador, y como actuador un servomotor o una electroválvula. El precio total sería de 28€ aproximadamente, 3€ de la placa NodeMCU y en torno a 25€ para la electroválvula. Por 1€ más, podríamos añadir un higrómetro, capaz de medir la humedad del suelo, dotando al equipo de una funcionalidad de la que no dispone la alternativa de Apple.

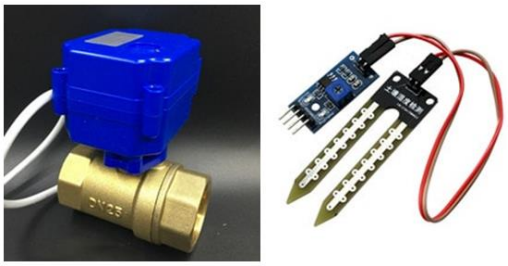


Figura 56: Electroválvula e higrómetro

- **Cámara de vigilancia**

Siguiendo con el Apple HomeKit, Apple ofrece una "cámara de seguridad por cable Circle 2 de Logitech que te permite vigilar la casa a través del iPhone o el iPad cuando no estás". Adquirible desde 199,95€.

Figura 57: Cámara de vigilancia Apple HomeKit

La alternativa que ofrece este proyecto es una placa NodeMCU a la que conectar una cámara ArduCam mini, o una cámara PTC06. Estas cámaras solo necesitan de conexión a Vcc, GND, y la comunicación serie RX y TX, trabajando la cámara PTC06 con una imagen de 640 x 480 a 30 fps. El precio de esta cámara es de 20€. Además, si la montamos sobre un servomotor, que podemos adquirir por 2€, dotaremos a la cámara de la funcionalidad de poder rotarla diferentes ángulos, pudiendo monitorizar distintas zonas.

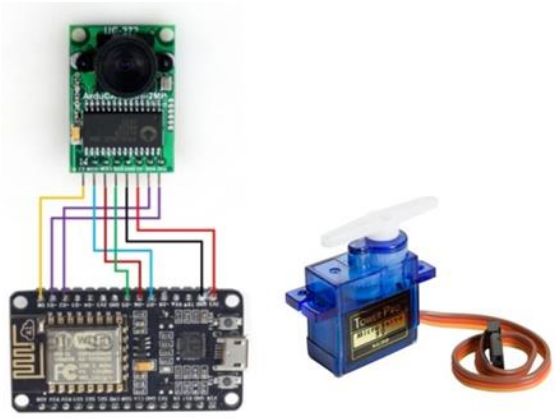


Figura 58: Conexión ArduCam con NodeMCU, y servomotor

- **Control de iluminación**

Los diferentes fabricantes de luminarias han sacado al mercado una serie de "bombillas inteligentes", que en realidad se tratan de bombillas LED a las que se ha incorporado conectividad. Por ejemplo, la compañía LIFX, dispone de bombillas con conexión WiFi, compatibles con diversos sistemas como Android, Alexa, o Google Assistant. Estas bombillas se pueden adquirir desde 28,80€.

Figura 59: Bombilla inteligente LIFX

El control de iluminación mediante NodeMCU se puede realizar mediante relés. Son dispositivos que se pueden adquirir por menos de 1€, y pueden abrir o cerrar el paso de la corriente mediante una señal de control digital. Además, podemos añadir otra funcionalidad, que es la iluminación automática al detectar presencia utilizando un sensor PIR, de un precio también menor a 1€.

Figura 60: Relé (izquierda) y PIR (derecha)

- **Termostato**

Otra de las aplicaciones dentro de la domótica es el control de la temperatura. Para ello, las grandes marcas han sacado a la venta termostatos inteligentes, que se tratan de termostatos con conectividad WiFi. Por ejemplo, la marca Tado ofrece un termostato compatible con Google Home. El producto se ofrece como una forma de controlar la calefacción desde el dispositivo móvil, y su precio es de 129,99€.

Figura 61: Termostato inteligente Tado

El control remoto de la caldera se puede conseguir de la misma forma con la placa NodeMCU y un relé para activar la caldera. Además, se pueden utilizar sensores de humedad y temperatura, adquiribles por un precio en torno a 3€, para monitorizar el estado de distintas zonas del hogar.

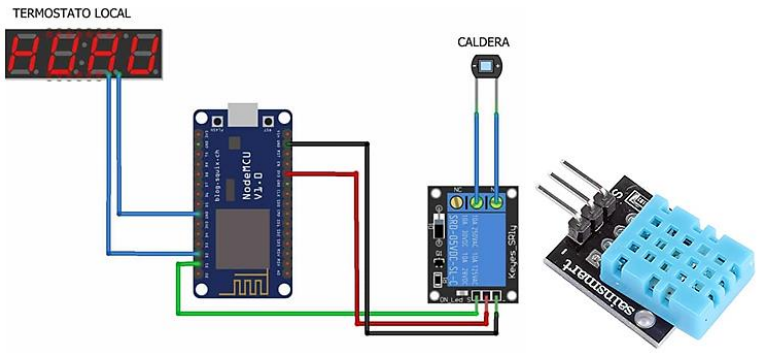


Figura 62: Conexión NodeMCU con caldera (izqueirda) y sensor DHT11 (derecha)

Las placas de desarrollo NodeMCU se pueden alimentar de varias formas. La más sencilla, es utilizar un adaptador de 7 V y conectarlo a la entrada microUSB de la placa. Este tipo de adaptadores son los que se utilizan, por ejemplo, para cargar los teléfonos móviles. Se pueden encontrar, con cable microUSB ya incluido, por cerca de 4€.

Figura 63: Adaptador 7V 2A

En la siguiente tabla muestro de forma resumida el presupuesto de los materiales que permitirían dotar a la vivienda de las funcionalidades anteriormente descritas, esto es, control de riego, cámara de vigilancia, control de iluminación (3 estancias) y control de temperatura.

Producto	**Precio (€)**	**Unidades**	**Total (€)**
Electroválvula	25	1	25
Higrómetro	1	1	1
Cámara	20	1	20
Servomotor	2	1	2
PIR	1	3	3
S. Humedad y Tª	3	3	9
Relé	2	4	8
NodeMCU	3	7	21
Adaptador 7 V 2A	4	7	28
			117

Como se puede observar, el precio es significativamente menor al de las alternativas comerciales. Además del precio, la red NodeMCU cuenta con la ventaja de poder manejar todos los dispositivos desde la misma página web, mientras que los dispositivos comerciales requerirán una App distinta según la marca. También, la fácil adaptación de la red a otro tipo de sensores y actuadores hace que sea adaptable a las necesidades de cada usuario, siendo cada red única.

5.1. Otras aplicaciones

En este TFM he desarrollado una red de dispositivos NodeMCU tomando como posible aplicación el ámbito de la domótica. La conectividad WiFi de estos dispositivos, así como la gran cantidad y variedad de sensores y actuadores que pueden controlar hace que el ámbito de este TFM no acabe en la domótica, si no que hay multitud de aplicaciones distintas en las la red NodeMCU puede resultar de gran utilidad. A continuación, listo varios ejemplos:

- **Estación meteorológica:** mediante sensores de humedad, presión, temperatura y un anemómetro, conectados a NodeMCU, es posible monitorizar por Internet el clima de distintas zonas.

- **Invernadero:** otra aplicación para los sensores de humedad y temperatura es su uso en invernaderos. También en esta aplicación son interesantes los sensores de intensidad luminosa y humedad del suelo. Además, NodeMCU puede actuar abriendo válvulas para el riego, por lo que se puede conseguir un invernadero automatizado y monitorizarlo desde cualquier lugar.

- **Control de instalaciones fotovoltaicas:** dentro de esta aplicación resulta interesante el uso de NodeMCU para, por ejemplo, comprobar si las placas están funcionando correctamente, un sistema de vigilancia con cámara, o monitorizar en tiempo real la potencia que están suministrando las placas.

- **Aplicaciones industriales:** en pequeñas industrias podría tener variedad de aplicaciones, como el conteo de piezas que salen de una línea, avisos si alguna máquina se para, e incluso aplicaciones más complejas como la conexión a controladores industriales tipo PLC o control de motores.

Figura 64: Otras aplicaciones para una red WiFi

6. CONCLUSIÓN

El objetivo de este proyecto era lograr una red WiFi estándar para domótica, es decir, unas estructuras de Sketch básicas sobre las que fuese posible programar diversas redes WiFi, con distinto número y tipo de sensores y actuadores, pero compartiendo determinados bloques de programación sobre los que fuese fácilmente adaptable la programación a las características de la red deseada.

Para ello se ha elegido como módulo de dicha red la placa NodeMCU. El motivo de su elección es que de entre las opciones de hardware y software libre, cuenta con conectividad WiFi integrada, es más económica, y a la hora de programar se encuentran disponibles una serie de librerías para la conexión WiFi. Además, es capaz de dar versatilidad a la red al poder trabajar con sensores tanto digitales como analógicos.

A la hora de programar los dispositivos, he diferenciado los Sketch según la función de cada elemento en la red, esto es, en 3 tipos: cliente, servidor e interfaz. Los módulos cliente tendrán la función de recoger la información de los sensores y enviar la señal a los actuadores, el módulo servidor recogerá la información de cliente e interfaz y el módulo interfaz permitirá al usuario la conexión desde un dispositivo con conexión WiFi el acceso a la red.

Esta división en cliente, servidor e interfaz se debe a lo siguiente:

- Los módulos cliente se encontrarán en diferentes lugares físicos de la red, donde sea necesario recoger datos de sensores, o activar actuadores. Por ello, en función de la cantidad de sensores y actuadores de la red, y su disposición física en la red, será necesario contar con varios módulos cliente.
- El módulo servidor será el nodo que cuente con toda la información de la red, y la distribuya a los módulos que la necesiten. Es decir, no todos los módulos cliente necesitan tener la información de todos los sensores y actuadores de la red, será el módulo servidor el encargado de recoger esta información enviar a cada cliente sólo la información para su funcionamiento.
- El módulo interfaz será el nodo de comunicación entre la red y el usuario. A pesar de que el módulo servidor ya recoge toda la información de la red, descarto utilizarlo como interfaz debido a que el servidor está continuamente recogiendo y enviando información a los clientes, por ello no puede atender de forma rápida y precisa a las peticiones del usuario. El módulo interfaz cuenta con una sola función dentro de su bucle, “handleClient”, con la que estará pendiente en todo momento de las peticiones del usuario a través de las páginas web que este módulo crea.

Así la red estará conformada por varios módulos cliente, un módulo servidor y un módulo interfaz.

En este sentido, el esquema de cada uno de los Sketch ha resultado de la siguiente forma.

Sketch Cliente

- Declaración de librerías.
- Declaración de variables.
- Set up de los pines (IN/OUT).
- Conexión WiFi.
- Recogida información sensores.
- Comunicación Servidor-Cliente.
- Envío señal a actuadores.

Este esquema es estándar, y es independiente del tipo de sensores y actuadores conectados, así como la cantidad de ellos en la red. Si queremos añadir o quitar sensores y actuadores de la red sólo habrá que repetir en mayor o menor medida las instrucciones ya definidas de cada uno de los bloques. Por ejemplo, si queremos añadir un sensor, las únicas modificaciones que habrá que realizar al Sketch serán:

- Una nueva variable que recoja la información de dicho sensor.
- Declaración de otro pin de entrada (INPUT).
- Añadir otra función digital/analog.read() que recoja la información del sensor.
- Añadir una cadena al String de comunicación con el servidor.

Cada una de estas modificaciones supone sólo añadir una línea de código, copiándola de las líneas ya programadas para otros sensores.

Sketch Servidor

- Declaración de librerías.
- Declaración de variables.
- Conexión WiFi.
- Comunicación Servidor-Cliente y Servidor-Interfaz.
- Cálculo de la señal para los actuadores.

Como se puede observar, el Sketch servidor tiene en común 4 bloques con el Sketch cliente. En este caso, al no utilizar los pines de entrada/salida, añadir un sensor o actuador a la red sólo supone añadir líneas de código en el cálculo de la señal para los actuadores.

Sketch Interfaz

- Declaración de librerías.
- Declaración de variables.
- Conexión WiFi.
- Comunicación Interfaz-Usuario e Interfaz-Servidor.

Del mismo modo que el nodo servidor, el nodo interfaz no utiliza los pines de entrada/salida. Añadir un número distinto de actuadores o sensores a la red sólo supone copiar y pegar las funciones ya programadas en el SetUp(), y cambiar los nombres de las variables por las nuevas variables que correspondan.

El protocolo de comunicación elegido para el intercambio de información entre estos módulos es el HTTP, ya que no es necesaria una velocidad de comunicación muy rápida. En diferentes pruebas he comprobado que desde que envío una señal de encendido de un led desde un dispositivo externo (ordenador o teléfono móvil) al módulo interfaz, y se enciende el led en el módulo cliente, normalmente no pasan más de 0,5 segundos, pero esto varía en función del dispositivo externo desde el que te conectas, el navegador, o la calidad de la red WiFi doméstica, por lo que en ocasiones puede ser un tiempo en torno a 1 segundo.

Respecto a la seguridad, he programado los módulos para que actúen en primer lugar como servidor, donde crearán una red WiFi a la que el usuario habrá de conectarse mediante contraseña. Las contraseñas pueden ser diferentes entre los distintos módulos y se pueden generar de forma aleatoria, de forma similar a los router domésticos. Esta configuración inicial puede llevarse a cabo de forma aún más segura añadiendo un simple pulsador conectado a un pin digital como input, con una resistencia pull-up o pull-down, y un condicional en el código, de forma que hasta que no se activa ese pulsador el módulo no creará la red WiFi de configuración inicial.

Una vez conectado a la red de NodeMCU, el usuario tendrá que conectarlo a su propia red WiFi doméstica, también mediante contraseña si la tuviera. El ESP8266 es compatible con los protocolos de seguridad WPA/WPA2, WEP y WPS. Una vez establecida la conexión la red doméstica, será posible configurar el router para acceder a la red desde cualquier lugar con acceso a internet, mediante conexión remota.

Respecto a la viabilidad económica, cada módulo NodeMCU tiene un coste inferior a 3€, por lo que el coste de la red no se deriva del controlador tanto como de los dispositivos que queramos controlar. Con este módulo podemos controlar luces, persianas, cámaras, riego, calefacción… en definitiva, cualquier dispositivo controlable mediante una señal digital o analógica (PWM). Del mismo modo, también puede recoger información de sensores digitales y analógicos y monitorizar variables como temperatura, humedad, intensidad luminosa… Esto hace que el módulo sea adaptable a las necesidades de usuario a distintos niveles de control y monitorización, es decir, se parte de unos módulos con programación estándar a los que se pueden conectar diferentes cantidades y tipos de sensores y actuadores según los requerimientos del usuario.

En definitiva, se ha conseguido una estructura de red básica sobre la que es posible programar diferentes configuraciones de redes de una forma rápida y sencilla.

7. BIBLIOGRAFÍA

- Evans, D. **Internet de las cosas. Cómo la próxima evolución de Internet lo cambia todo** [texto electrónico, PDF]. Cisco Internet Business Solutions Group (IBSG): abril de 2011. Recuperable en: https://www.cisco.com/c/dam/global/es_mx/solutions/executive/assets/pdf/internet-of-things-iot-ibsg.pdf
- **ARDUINO©** [Internet] [primer acceso en noviembre de 2018]. Disponible en: https://www.arduino.cc/
- **ESPRESSIF©** [Internet] [primer acceso en noviembre de 2018]. Disponible en: https://www.espressif.com/en/home
- Espressif. **ESP8266. Technical Reference** [texto electrónico, PDF]. Espressif: 2020. Recuperable en: https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf
- Iberobotics. Robots personales y de servicios [Internet] [acceso en enero de 2019]. **ESP-01 Módulo WiFi basado en ESP8266 Serial**. Disponible en: https://www.iberobotics.com/producto/esp-01-modulo-wifi-esp8266/
- **Raspberry Pi Foundation** [Internet] [primer acceso en enero de 2019]. Disponible en: https://www.raspberrypi.org/
- Raspberry Shop [Internet]. **Raspberry Pi 3** [acceso en enero de 2019]. Disponible en: https://www.raspberryshop.es/raspberry-pi-3.php
- BricoGeek [Internet]. **NodeMCU V3 Wifi - ESP8266, CH340** [acceso en enero de 2019]. Disponible en: https://tienda.bricogeek.com/wifi/1033-nodemcu-v3-wifi-esp8266-ch340.html
- Serna Ruiz, A., Ros García, F. A., Rico Noguera, J. C. **Guía práctica de sensores**. España: Creaciones Copyright; 2010.
- Bosch-sensortec [Internet]. **Humidity sensor – BME280** [acceso en diciembre de 2018]. https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/
- OSOYOO [Internet]. **Micro Servo – SG90** [acceso en diciembre de 2018]. Disponible en: https://osoyoo.com/2017/05/08/micro-servo-mg90-9g/
- **Aprendiendo Arduino. Aprendiendo a manejar Arduino en profundidad** [Internet] [primer acceso en enero de 2019]. Disponible en: https://aprendiendoarduino.wordpress.com/
- **ESPloradores** [Internet] [primer acceso en febrero de 2019]. Disponible en: https://www.esploradores.com/
- **GitHub** [Internet] [primer acceso en febrero de 2019]. Disponible en: https://github.com/
- **TUNIOT** [Internet] [primer acceso en febrero de 2019]. Disponible en: http://easycoding.tn/

- **ESP8266 Arduino Core** [Internet] [primer acceso en febrero de 2019]. Disponible en: https://arduino-esp8266.readthedocs.io/en/latest/libraries.html
- ARDUINO a muete [Internet] [acceso en febrero de 2019]. **Librería ESP8266WebServer (Arduino IDE)** [aprox. 4 pantallas]. Disponible en: http://arduinoamuete.blogspot.com/2016/02/libreria-esp8266webserver-arduino-ide.html
- Luis Llamas. Ingeniería, informática y diseño [Internet] [acceso en marzo de 2019]. **Conectar Arduino por Bluetooth con los módulos HC-05 ó HC-06** [aprox. 8 pantallas]. Disponible en: https://www.luisllamas.es/conectar-arduino-por-bluetooth-con-los-modulos-hc-05-o-hc-06/
- ComoFunciona [Internet] [acceso en marzo de 2019]. **Cómo funciona el Bluetooth** [aprox. 5 pantallas]. Disponible en: https://como-funciona.co/el-bluetooth/
- **IEEE 802.11™** [Internet] [primer acceso en febrero de 2019]. Disponible en: http://www.ieee802.org/11/
- Cassiopeia Ltd. The joy of making stuff [Internet] [acceso en abril de 2019]. **WiFi/FTP camera with ESP8266 and serial camera PTC06 (and PTC02/PTC08)** [aprox. pantallas]. Disponible en: https://cassiopeia.hk/wificam/
- Apple [Internet]. **HomeKit** [acceso en junio de 2019]. Disponible en: https://www.apple.com/es/shop/accessories/all-accessories/homekit
- **LIFX Europe. WiFi enabled LED smart lights** [Internet] [acceso en junio de 2019]. Disponible en: https://eu.lifx.com/
- Tado [Internet]. **Termostato inteligente** [acceso en junio de 2019]. Disponible en: https://www.tado.com/es-es/termostato-inteligente