Escuela Politécnica Superior de Jaén

Grado en Ingeniería Eléctrica

UNIVERSIDAD DE JAÉN

Trabajo Fin de Grado

DISEÑO DE UN SISTEMA DE RIEGO PARA EL OLIVAR BASADO EN MICROCONTROLADOR

Alumno: Miguel Ángel Moya Garrido

Tutores: Prof. D. Fº José Sánchez Sutil
Prof. D. Antonio Cano Ortega
Dpto: Ingeniería Eléctrica

SEPTIEMBRE, 2019

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Ingeniería Eléctrica

Don Fº JOSÉ SÁNCHEZ SUTIL y Don ANTONIO CANO ORTEGA , tutores del Proyecto Fin de Carrera titulado: DISEÑO DE UN SISTEMA DE RIEGO PARA EL OLIVAR BASADO EN MICROCONTROLADOR, que presenta MIGUEL ÁNGEL MOYA GARRIDO, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, SEPTIEMBRE de 2019

El alumno:

MIGUEL ÁNGEL MOYA GARRIDO

Los tutores:

Fº JOSÉ SÁNCHEZ SUTIL

ANTONIO CANO ORTEGA

Índice

1. Introducción

1.1. Motivación del proyecto

El objeto del presente proyecto es el diseño de un sistema de riego que se controle mediante una aplicación Android.

La idea del Trabajo Fin de Grado surgió del inconveniente que tiene un familiar para regar su finca de olivos de regadío, ya que esta se encuentra a 50 km de su lugar de residencia y para regar debe desplazarse muy a menudo, acarreando gastos económicos y molestias. Esta circunstancia me motivó a la elaboración del proyecto, ya que podrá llevarse a cabo en mi entorno y podré ver su funcionamiento.

Además de la apertura de las válvulas de forma remota, se implementará un sistema automático que controle la humedad del terreno a partir de sensores de humedad del suelo, realizando un mejor aprovechamiento de los recursos hídricos.

La alimentación energética se llevará a cabo a partir de energía solar haciendo el proyecto responsable con el medio ambiente y reduciendo el tiempo de amortización de la instalación.

La elección de los componentes del proyecto se hará teniendo en cuenta abaratar al máximo el presupuesto, sin olvidar las garantías del correcto funcionamiento de la instalación.

1.2. Objetivos

Los requisitos técnicos que deberá cumplir el proyecto serán:

- Interacción del usuario con la instalación a larga distancia (orden de kilómetros).
- Comunicación inalámbrica de los elementos del proyecto.
- Apertura y cierre manual de las válvulas.
- Apertura y cierre automática de las válvulas, en función de la humedad del suelo.
- Implementación de una APP que sirva de interfaz de la instalación con el usuario.

1.3. Estructura del trabajo

Primeramente, se describen los antecedentes de la instalación para después explicar las modificaciones que se van a realizar y el funcionamiento que va a tener la instalación después de la consecución del trabajo.

Se continua con el estado del arte de los sensores y válvulas, y en el análisis de soluciones se explican las soluciones adoptadas.

Le sigue la descripción del universo Arduino y de los componentes más importantes empleados en el trabajo: módulo GSM/GPRS y módulo NRF24L01.

Una vez definidos todos los componentes se describirá la forma de alimentación adoptada y se hará su cálculo.

Finalmente se explicará la programación de las placas Arduino y la implementación de la aplicación Android.

2. Antecedentes

Se trata de una finca situada en el término municipal de Chiclana de Segura (Jaén), de olivos de regadío a la cual llega el agua a partir de una comunidad de regantes del pantano de Guadalmena.

El agua llega a la válvula general, a través de una tubería de PVC, ya filtrada, mediante dos filtros de anillas y dos filtros de arena, ubicados en la caseta de filtros de la comunidad de regantes.

Como se detalla en el plano I, de la válvula general, parte la tubería principal de PVC, que alimenta a las válvulas de cada sector. De cada válvula de sector, una tubería lateral alimenta a cada rama de goteros.

En su día la instalación de riego se dividió en tres sectores con sus tres respectivas válvulas y una válvula general, todas de accionamiento manual. Los sectores se dimensionaron teniendo en cuenta el gradiente máximo de presión que limita el funcionamiento correcto de la autocompensación de los goteros, esto

explica su tamaño desigual, pues la finca no tiene la misma pendiente en todos sus puntos.

En un principio, a cada olivo se le instalaron dos goteros autocompensantes de botón. Posteriormente se hizo una reforma de la instalación y se cambió el sistema de goteo. Este nuevo sistema consiste en goteros integrados en el tubo situado a 0.5 m de profundidad y ubicado en el punto medio entre fila y fila de olivos. El tubo contiene un gotero autocompensante integrado por cada metro.

3. Descripción de la instalación

3.1. Elementos del sistema

Las modificaciones en la instalación se limitan a la automatización de la misma, manteniendo los mismos sectores y la misma localización de las válvulas, pues se consideró que el diseño anterior funciona bien desde la experiencia y así no se tendría que cambiar ninguna tubería, abaratando el presupuesto.

Los elementos electrónicos se alojarán en cajas estancas de PVC y se fijarán sobre postes de madera de 1.5 m anclados al suelo. La alimentación será con baterías ion litio apoyadas con placas solares. Las placas solares se dispondrán de forma que sobre ellas incida la mayor cantidad de radiación, situadas arriba del poste de madera sobre una placa de plástico y orientadas hacia el sur con una inclinación de 30 º.

Los sensores se instalarán en el punto más desfavorable de la instalación, es decir, en la parte más cercana a la válvula de cada sector y más alejado de la tubería lateral. Su parte electrónica se impregnará de resina de poliuretano o epoxi para que no la dañe la humedad. Se insertarán en el suelo dejando las partes electrónicas recubiertas al aire, y se situarán a 40 cm de la tubería de goteros integrados.

Las electroválvulas se han escogido de la misma dimensión que las que había anteriormente por lo que su instalación se limita al intercambio de las mismas. Su accionamiento se llevará a cabo por medio de un relé.

3.2. Funcionamiento

El funcionamiento de la instalación se controla mediante la aplicación Android implementada en el capítulo 11. La aplicación se encarga de enviar ordenes en forma de SMS al Arduino principal. El Arduino principal es el que tiene contenido el segundo sensor y recibe los SMS gracias al módulo GSM/GPRS.

El Arduino principal controla las válvulas y los sensores a través de comunicaciones de radiofrecuencia por medio de los transceptores NRF24L01.

3.2.1. Formato de la orden en forma de SMS

Cuando se recibe un SMS a través del módulo GSM/GPRS, además del SMS en sí, se reciben una serie de datos que aparecen antes tales como la fecha, la hora y el número de teléfono. Por esto es necesario incluir al principio un caracter que permita identificar después el principio de la orden, en el caso de este trabajo la letra “u”.

A este caracter de inicio de la orden, le sigue el modo de funcionamiento que será 1 para el modo manual y 2 para el modo automático. La siguiente información de la orden depende del modo de funcionamiento escogido. Si es necesario se añadirán ceros delante de los datos para que sean de dos cifras en el caso de las horas y de tres cifras en el caso de lo setpoints y Arduino pueda leerlos correctamente. En el modo manual se incluye primero el número de la válvula (1, 2 o 3) y después el tiempo que se quiere regar, que tendrá un máximo de 12 h. En el modo automático se incluyen los setpoints de cada sector.

Así, si se quiere regar el sector 2 en modo manual durante 8 h, la orden será “u1208”. Si se quiere regar en modo automático y se indican unos setpoints de 40% para el primer sector, un 30% para el segundo sector y un 60% para el tercer sector, la orden será “u2040030060”.

3.2.2. Instrucciones de la Aplicación Android

Cuando abrimos la aplicación aparece la siguiente pantalla para que se escoja el modo de funcionamiento.

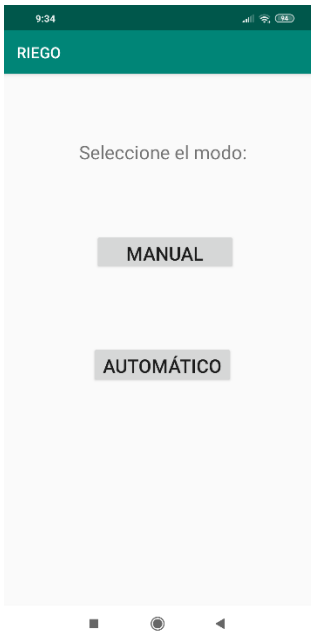


Fig. 3.1. Pantalla para escoger modo de funcionamiento

Si escoge el modo manual, aparecerá la siguiente pantalla para que se elija el sector que se desea regar.

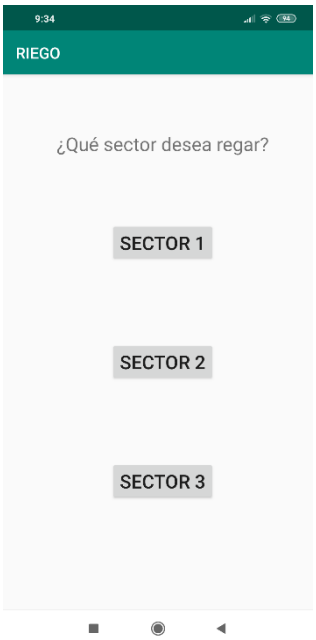


Fig. 3.2. Pantalla para escoger sector

Una vez se escoge el sector aparecerá la pantalla donde se introducirán las horas que se desea regar, siendo el máximo 12 h. Si se desea cerrar una válvula se introducirá 0 como número de horas. Una vez se introduce el tiempo deseado, para enviar la orden se selecciona el botón enviar.

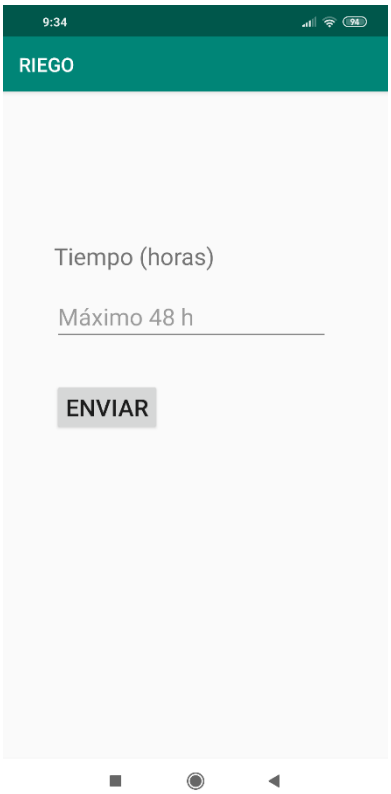


Fig. 3.3. Incluir horas que se desea regar

Si se selecciona el modo automático aparecerá la siguiente pantalla, donde se introducirán los setpoints de cada sector. Una vez introducidos se presiona el botón enviar para mandar la orden.

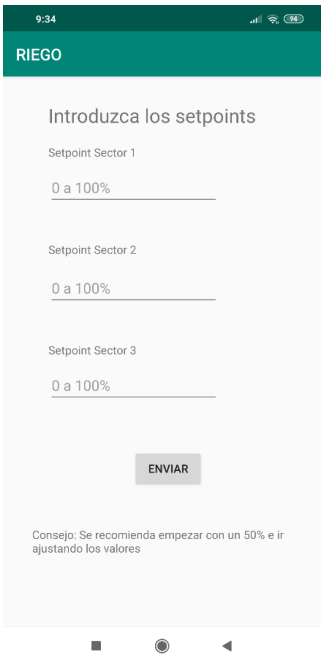


Fig. 3.4. Pantalla del modo automático

4. Estado del arte

En este apartado se explica el mercado actual de los elementos que se usan en este trabajo. Es un punto importante, pues sabiendo lo existente luego podemos elegir lo que más convenga al proyecto. La elección y su justificación se verá en el siguiente apartado en el análisis de soluciones.

4.1. Sensores de humedad del suelo

Existen dos tipos de sensores de humedad de suelo para usar en Arduino. Unos son de tipo resistivo, que miden la resistencia del suelo. Pero la resistencia del suelo además de la humedad depende de otros factores como la salinidad, presencia de compuestos químicos o la temperatura. Estos sensores tendrán poca fiabilidad en un terreno donde se lleva a cabo el cultivo convencional, dónde se añaden al terreno herbicidas y fertilizantes. Deben ir acompañados de un convertidor analógico-digital externo para comunicarlo con Arduino.

Fig. 4.1. Sensor tipo resistivo

Por otro lado, están los sensores capacitivos que se basan en la medida de las capacidades dieléctricas de la tierra. Esta forma de medir la humedad está mucho menos afectada por los factores externos que la que la de medir la resistencia del suelo. Disponen de un convertidor analógico-digital presente en el propio sensor.

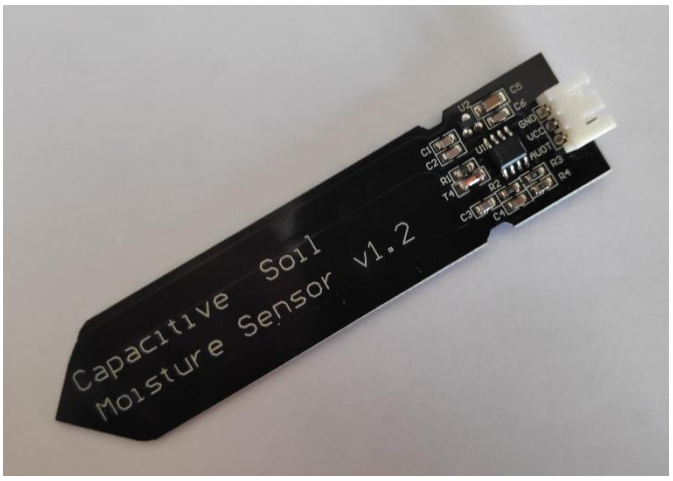


Fig. 4.2. Sensor tipo capacitivo

4.2. Electroválvulas

Principalmente las electroválvulas las podemos dividir en dos grupos en función de su actuación ya sea por solenoide o por un motor. Las accionadas por motor tienen la ventaja de tener un menor consumo, pues sólo consumen energía cuando se cierran o se abren, pero tienen un coste mayor.

También se pueden clasificar según su fabricación en metal o en plástico. Las de plástico tienen las ventajas de resistir mejor las heladas y tener un coste menor, pero las metálicas son más robustas.

En fusión de su alimentación las encontramos de corriente alterna a 220 V y de corriente continua.

5. Análisis de soluciones

A la hora de elegir el microcontrolador se eligió la plataforma Arduino por:

- Poseer el microcontrolador y todos los componentes necesarios para programar directamente.
- Por su bajo coste, tanto de las placas como de los componentes.
- La gran información existente en la Red.

En cuanto a la elección del sensor claramente se usarán los de tipo capacitivo por su mayor precisión y fiabilidad, aunque tienen un problema, y es que las partes electrónicas del convertidor analógico-digital están al aire, y el sensor debe ser

enterrado a la altura de las raíces del olivo. Este problema se solucionará impregnando los componentes electrónicos de una resina que los proteja de la humedad, como puede ser la resina de epoxi o poliuretano. Estos sensores por su bajo precio no están hechos para aguantar mucho tiempo enterrados por lo que se tendrán que renovar cada campaña de riego.

Es conocida en la zona de la finca la ocurrencia de heladas en invierno, por esto y por su menor precio se elegirán válvulas de plástico y accionadas por solenoide por el mismo motivo. Su alimentación será a 9 V.

En cuanto al tema de la alimentación de los elementos del proyecto nos encontramos con el problema de no tener disponible energía eléctrica en la finca, por lo que tendremos que alimentar desde baterías. Se han hecho algunos cálculos y sólo con baterías, el número de éstas sería elevado si queremos que duren un tiempo conveniente, además de tener el inconveniente de tener que cargarlas periódicamente. Es así que se instalarán placas solares de pequeña potencia acompañadas de gestores de carga y reguladores de tensión, que vayan cargando las baterías.

6. Arduino

6.1. ¿Qué es Arduino?

Arduino es una plataforma de código y hardware abierto que diseña y crea placas de desarrollo orientadas a la iniciación en la electrónica y la creación de programas que interactúen con el mundo real. Sus placas de desarrollo están fabricadas de modo que tanto las conexiones como la programación sea sencilla.

Además, Arduino consta de una gran comunidad lo que facilita la resolución de los problemas que puedan surgir a la hora de elaborar proyectos. Hay usuarios de esta comunidad que sin ánimo de lucro crea y comparte librerías que facilitan enormemente la programación.

Dada la difusión de Arduino, abundan en el mercado sensores y actuadores que se pueden conectar directamente a la placa y usar de forma realmente fácil.

6.2. ¿Por qué usar el microcontrolador de Arduino?

El trabajo se basa en un microcontrolador como su título indica. Se usa Arduino porque las placas Arduino disponen de un microcontrolador, de la compañía Atmel, y de todos los componentes electrónicos necesarios para usarlo ya ensamblados, evitando tener que diseñar y construir un circuito integrado.

Las placas Arduino llevan incorporados pines de lectura analógica, pines de entrada y salida digital y pines de salida PWM; un puerto USB de tipo B o mini que permite una comunicación serie tanto para la compilación de los programas como para la lectura de datos; regulador de voltaje de 5 o 3.3V según el modelo; cristal de cuarzo de 16 MHz; memorias Flash, SRAM, y EEPROM.

Cabe destacar la existencia de shield's de expansión que aumentan las posibilidades de las placas Arduino. Estos son módulos que se conectan a la placa original y contienen chips adicionales para aumentar las posibilidades de los proyectos.

Otro de los motivos para usarlo es su bajo coste si lo comparamos con otros microcontroladores. También es compatible con prácticamente todos los sistemas operativos ya sea Windows, Macintosh OSX o GNU/Linux.

Su entorno de programación es simple y claro cómo se verá en el siguiente apartado.

6.3. El IDE de Arduino

El entorno de programación o IDE (Integrated Development Environment) de Arduino es la herramienta oficial que nos ofrece la compañía para crear y compilar los sketches en la placa a través del puerto USB. El lenguaje de programación utilizado está inspirado en C++.

Los sketches constan básicamente de tres partes:

- Cabecera: en esta parte se definen las librerías precedidas del comando #include, se declaran las variables globales y se crean las instancias necesarias de cada librería.

- void setup(): aquí se lleva a cabo la configuración inicial de las variables.
- void loop(): es un bucle de repetición infinita, dónde se incluyen las tareas que se quieren realizar de manera continua.

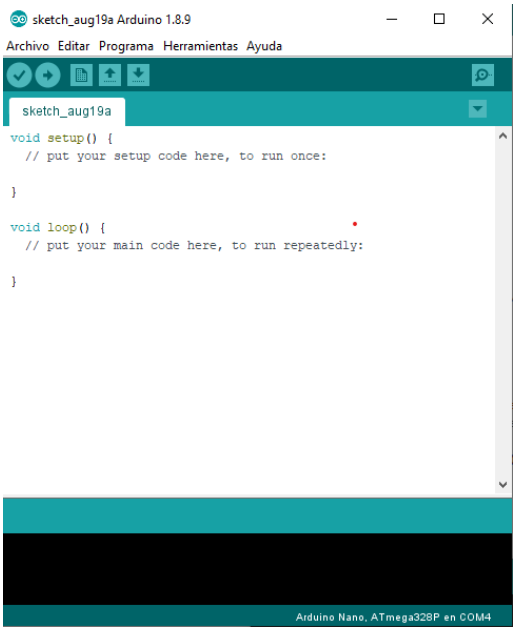


Fig. 6.1. IDE de Arduino

6.3.1. Las librerías de Arduino

Como se ha dicho anteriormente existen infinidad de librerías creadas por la comunidad Arduino que nos facilitan la programación. Para instalar una librería basta con descargarla en formato .zip y en el menú Programa/incluir librería, clicar en la opción añadir biblioteca .ZIP…

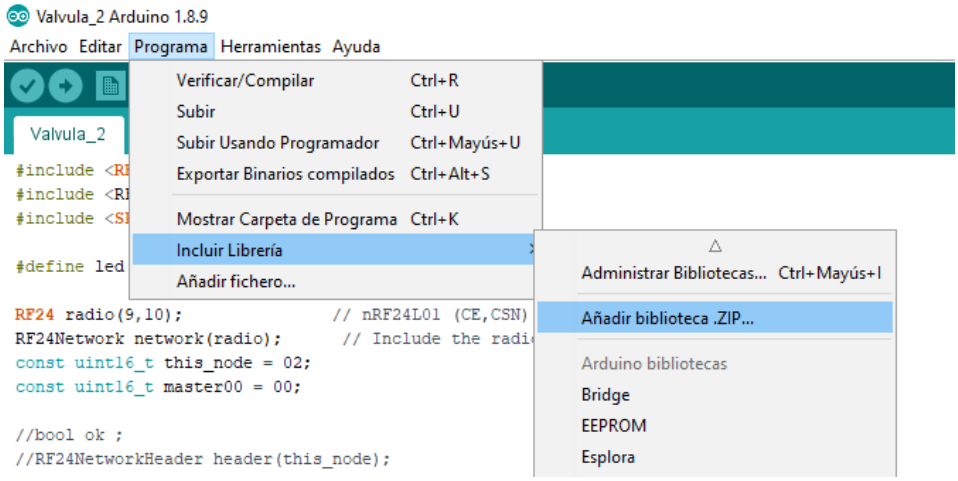


Fig. 6.2 Incluir librería Arduino

Para usarla se indica al principio del programa con el comando #include y el nombre de la librería entre signos de menor y mayor que. (Ej. #include <RF24.h>).

6.3.2. Cargar un programa en una placa Arduino

Para cargar un programa primero se debe seleccionar el modelo de placa Arduino que se va a usar en el submenú Herramientas/Placa:

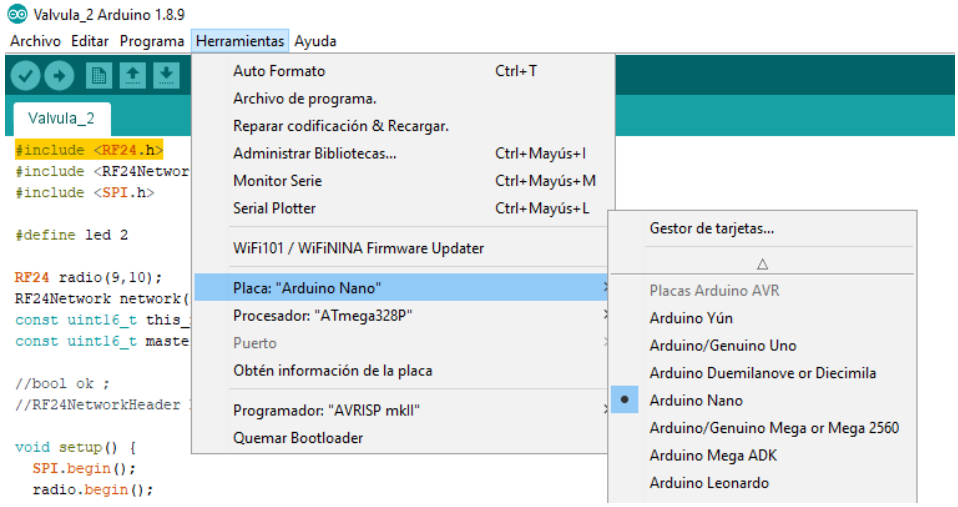


Fig. 6.3. Seleccionar placa Arduino

Después se selecciona el puerto de la placa en el submenú Herramientas/Puerto:

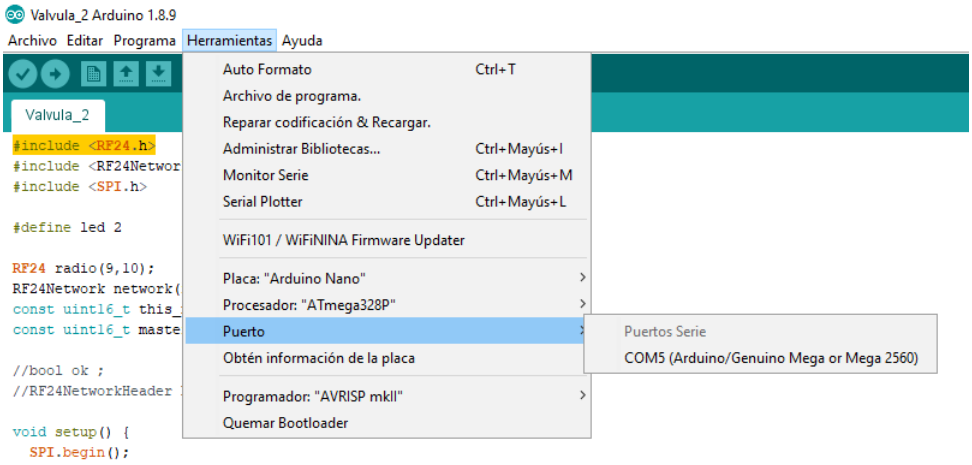


Fig. 6.4. Seleccionar puerto

Finalmente verificamos que el programa está escrito correctamente y lo cargamos en la placa con los siguientes botones:

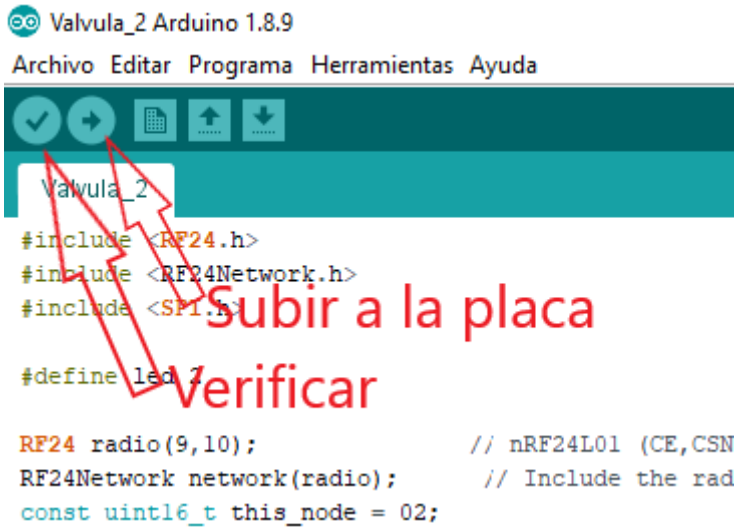


Fig. 6.5. Verificar y subir a la placa

6.4. Placas Arduino usadas en el proyecto

6.4.1. Arduino Mega

Se ha elegido esta placa Arduino para el programa principal por poseer una mayor memoria y un microcontrolador más potente teniendo así una mayor fiabilidad en el programa más importante.

Características técnicas del Arduino Mega [2]:

- Microcontrolador: ATmega2560
- Voltaje Operativo: 5V
- Voltaje de Entrada: 7-12V
- Voltaje de Entrada(límites): 20V
- Pines digitales de Entrada/Salida: 54 (de los cuales 15 proveen salida PWM)
- Pines análogos de entrada: 16
- Corriente DC por cada Pin Entrada/Salida: 40 mA
- Corriente DC entregada en el Pin 3.3V: 50 mA
- Memoria Flash: 256 KB (8KB usados por el bootloader)
- SRAM: 8KB
- EEPROM: 4KB
- Clock Speed: 16 MHz

Fig. 6.6. Arduino Mega

6.4.2. Arduino Nano

Para los demás nodos de la instalación se ha usado el Arduino Nano ya que tiene un menor consumo, lo cual interesa a la hora de alimentar por baterías, y es compatible con el Nano V3.0 I/O & Wireless shield.

Características técnicas del Arduino Nano [3]:

- Microcontrolador: ATMega328.
- Voltaje de operación: 5V.
- Voltaje de alimentación (Recomendado): 7-12V.
- I/O Digitales: 14 (6 son PWM)
- Entradas Analógicas: 8
- Memoria Flash: 32KB.
- EEPROM: 1KB.
- Frecuencia de trabajo: 16MHz.
- Dimensiones: 0.73″ x 1.70″

Fig. 6.7. Arduino Nano

7. Telemando de la instalación. Shield GSM/GPRS

La solución propuesta para comunicarnos con el proyecto a larga distancia es el shield GSM/GPRS. Este shield está basado en el chip SIM900 y da a Arduino las capacidades básicas de un teléfono móvil (Conexión a internet, enviar/recibir SMS, enviar/recibir llamadas).

Fig. 8.1. Shield GSM/GPRS

En este trabajo se usará su funcionalidad de recibir SMS, gracias al GSM (Global System for Global Communications) que es el estándar en Europa por el cual los teléfonos móviles envían y reciben tanto llamadas como SMS.

Este shield se comunica con Arduino vía UART (Universal Asynchronous Receiber-Transmitter) utilizando la librería Serial a través de comandos AT los cuales se incluyen en el apartado 7.2.

7.1. Conexiones

Para que funcione es necesario incluirle una tarjeta SIM de tamaño estándar, previamente activada por un teléfono móvil o Smart Phone. Actualmente, la mayoría de los Smart Phone funcionan con nano SIM, por lo que se necesita un adaptador.

Para poder encenderlo por software es necesario soldar el pad R13 del shield, y conectar el pin 9 del shield a un pin digital que tengamos libre en la placa Arduino.

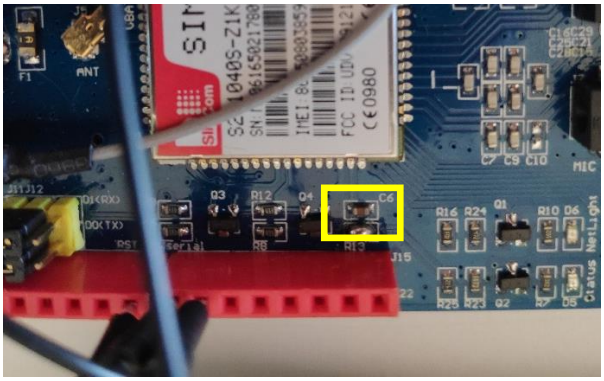


Fig. 8.2. Soldadura pad R13

Las conexiones para la comunicación vía UART son los pines 7 y 8 del módulo GSM/GPRS con los pines 10 y 11 del Aduino Mega que se usará como principal.

PIN	GSM/GPRS	UNO	MEGA
GND	1	GND	GND
Rx	7	7	10
Tx	8	8	11
ON/OFF	9	9	9

Tabla 8.1. Conexiones GSM/GPRS

7.2. Comandos AT

Los comandos AT (de attention) es la forma que tenemos de comunicarnos con el GSM a través de Arduino. A continuación, se explican los comandos usados en la programación de este trabajo:

- AT: Detecta que el módulo funciona correctamente. Devuelve OK si todo está correcto.
- AT+CPIN=”XXXX”: Con este comando se introduce el código PIN de la tarjeta SIM.
- AT+CREG?: Comprobación de la conexión de red. Devuelve “+CREG: 0,1”.
- AT+CMGF=1: Configura el módulo para enviar y recibir mensajes.
- AT+CNMI=2,2,0,0,0: Configura el módulo para que muestre los SMS.

7.3. Métodos que lo hacen operativo

Para que el shield GSP/GPRS funcione con total seguridad es necesario utilizar una serie de métodos que garanticen su correcto funcionamiento. Se usarán las funciones aportadas por prometec.net [4] con algunas modificaciones.

7.3.1. Método enviarAT()

```
1. int enviarAT(String
   ATcommand, char* resp_correcta, unsigned int tiempo)
2. {
3.
4.   int x = 0;
5.   bool correcto = 0;
6.   char respuesta[100];
7.   unsigned long anterior;
8.
9.   memset(respuesta, '\0', 100); // Inicializa el string
10.        delay(100);
11.        while ( SIM900.available() > 0) SIM900.read(); // Limpia el
   buffer de entrada
12.        SIM900.println(ATcommand); // Envia el comando AT
13.        x = 0;
14.        anterior = millis();
15.        // Espera una respuesta
16.        do {
17.          // si hay datos el buffer de entrada del UART lee y
   comprueba la respuesta
18.          if (SIM900.available() != 0)
19.          {
20.              respuesta[x] = SIM900.read();
21.              x++;
22.            // Comprueba si la respuesta es correcta
23.            if (strstr(respuesta, resp_correcta) != NULL)
24.            {
25.              correcto = 1;
26.            }
27.          }
28.        }
29.        // Espera hasta tener una respuesta
30.        while ((correcto == 0) && ((millis() - anterior) < tiempo));
31.        Serial.println(respuesta);
```

```
32.
33.         return correcto;
34.       }
```

Este método se usa para enviar un comando AT al shield GSM/GPRS y asegurarse de que lo recibe. Tiene tres parámetros, el primero es el comando AT a enviar, el segundo la respuesta esperada y el tercero el tiempo que espera a recibir respuesta. Devuelve 1 si el comando AT ha sido enviado con éxito.

7.3.2. Método power_on()

```
8. void power_on()
9. {
10.         int respuesta = 0;
11.
12.         // Comprueba que el modulo SIM900 esta arrancado
13.         if (enviarAT("AT", "OK", 2000) == 0)
14.         {
15.           Serial.println("Encendiendo el GPRS...");
16.
17.           pinMode(9, OUTPUT);
18.           digitalWrite(9, HIGH);
19.           delay(1000);
20.           digitalWrite(9, LOW);
21.           delay(1000);
22.
23.           // Espera la respuesta del modulo SIM900
24.           while (respuesta == 0) {
25.             // Envia un comando AT cada 2 segundos y espera la
    respuesta
26.             respuesta = enviarAT("AT", "OK", 2000);
27.             SIM900.println(respuesta);
28.           }
29.         }
30.       }
```

Este método se encarga de encender el módulo GSM/GPRS. No tiene ningún parámetro ni devuelve ningún valor.

7.3.3. Método iniciar()

```
8.   void iniciar()
9.   {
10.    enviarAT("AT+CPIN=\"7401\"", "OK", 1000);
11.    Serial.println("Conectando a la red...");
12.    delay (5000);
13.
14.    //espera hasta estar conectado a la red movil
15.    while ( enviarAT("AT+CREG?", "+CREG: 0,1", 1000) == 0 )
16.    {
17.    }
18.    Serial.println("Conectado a la red.");
19.    enviarAT("AT+CMGF=1\r", "OK", 1000); //Configura el modo texto para
     enviar o recibir mensajes
```

```
20.    enviarAT("AT+CNMI=2,2,0,0,0\r", "OK", 1000); //Configuramos el modulo
     para que nos muestre los SMS recibidos por comunicacion serie
21.    Serial.println("Preparado.");
22.  }
```

Este método introduce el código PIN, conecta el módulo a la red y configura la recepción y envío de SMS. También configura que los mensajes recibidos los muestro por el puerto serie.

8. Comunicación interna de la instalación. NRF24L01

El chip NRF24L01 [5] [6] [7] integra en un único chip todo lo necesario para llevar a cabo comunicaciones de radiofrecuencia. Se trata de un transceptor, pues es capaz de enviar y recibir datos, pero no hacerlo a la vez. Operan en la banda de 2.4 Ghz, que es de libre uso, a una velocidad configurable de 250 kb/s, 1 Mb/s o 2 Mb/s. Además, destacan por su bajo precio y su bajo consumo cuando no están funcionando.

Su tensión de alimentación es 3.3 V, aunque los pines de control soportan hasta 5 V. No obstante, es recomendable incluirle un shield que adapte el nivel de tensión.

Se comunica con Arduino vía SPI (Serial Peripheral Interface). Existen diversas librerías para facilitar la programación y disminuir los errores en la comunicación. La más difundida, y la que se usará en este proyecto, es la RF24.

Existen diversos fabricantes que lo comercializan en diferentes formas siendo el más básico el siguiente:

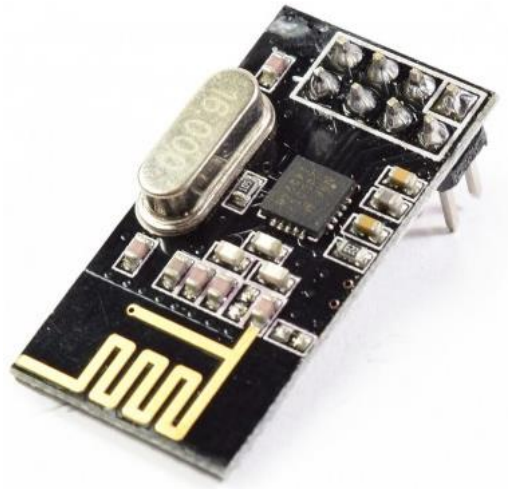


Fig. 8.1. Módulo NRF24L01

Este módulo lleva integrada la antena en la placa y su alcance llega hasta los 100 m en espacios abiertos y a una velocidad de comunicación baja. Pero en este trabajo se necesita una distancia mínima con algunos obstáculos de 200 m. Se encuentra la solución en el siguiente módulo que contiene un circuito amplificador de potencia (PA) para la transmisión y un circuito amplificador de bajo ruido (LNA) para la recepción. Incluye además una antena SMA.

Fig. 8.2. NRF24L01 + PA + LNA

8.3. Conexiones

El módulo posee los siguientes pines:

- V+, GND: Pines de alimentación.
- CSN, CE: Activan o desactivan el modo espera.

- MOSI, SCK, MISO: Pines de la comunicación SPI.
- IRQ: Sin conectar.

A continuación, se muestra su ubicación en el módulo y la tabla de conexiones para las placas Arduino más comunes.

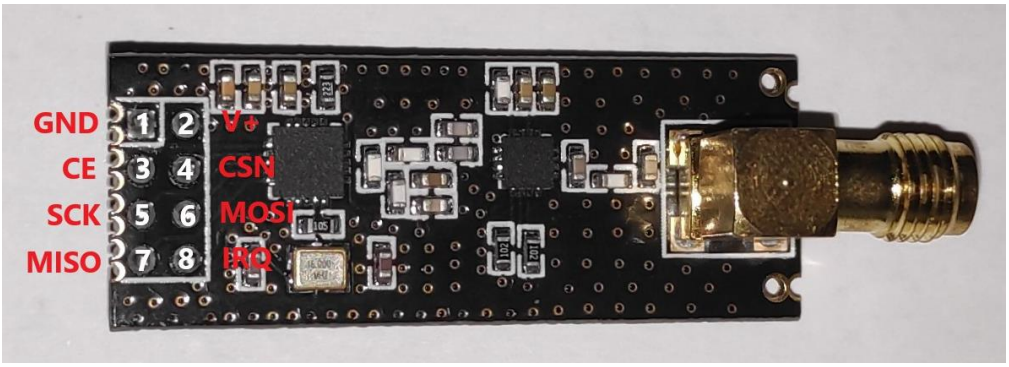


Fig. 8.3. Pines NRF24L01 + PA + LNA

PIN	NRF24L01	UNO, NANO	MEGA
GND	1	GND	GND
V+	2	3.3 V	3.3 V
CE	3	9	9
CSN	4	10	10
SCK	5	13	52
MOSI	6	11	51
MISO	7	12	50
IRQ	8	-	-

Tabla 8.1. Conexiones NRF24L01

En el Arduino principal será necesario para el correcto funcionamiento del módulo añadir un condensador de 10 uF entre los pines GND y V+.

En este trabajo por seguridad y facilidad en las conexiones se empleará el Nano V3.0 I/O & Wireless shield, placa de expansión preparada para conectar un Arduino Nano y el NRF24L01. Este shield lleva integrado un regulador de tensión a 3.3 V que adapta el nivel de tensión tanto a la alimentación del NRF24L01 como a los pines de control, lo cual es conveniente como ya se ha dicho anteriormente.

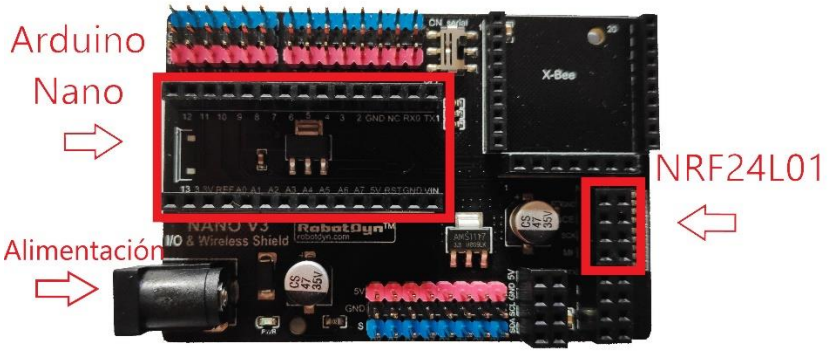


Fig. 8.4. Nano V3.0 I/O & Wireless shield

8.4. Red nodal

Se ha decidido la implementación de una red nodal basada en la referencia [8] para hacer que las comunicaciones de radiofrecuencia transcurran entre medias de las filas de olivos y no atravesándolos. Se consigue gracias a la librería RF24Network que identifica los nodos y los caminos a seguir por la información a partir de la dirección que se les da.

La red tiene una topología de árbol y la denominación de los nodos se sigue tal como se muestra en la siguiente imagen. La base se denomina 00 y de ahí van partiendo los demás nodos, pudiendo conectar hasta 5 hijos por nodo. Las comunicaciones transcurren siguiendo la línea sucesoria, es decir, para que le nodo 011 se comunique con el 00, la información tiene que atravesar el nodo 01.

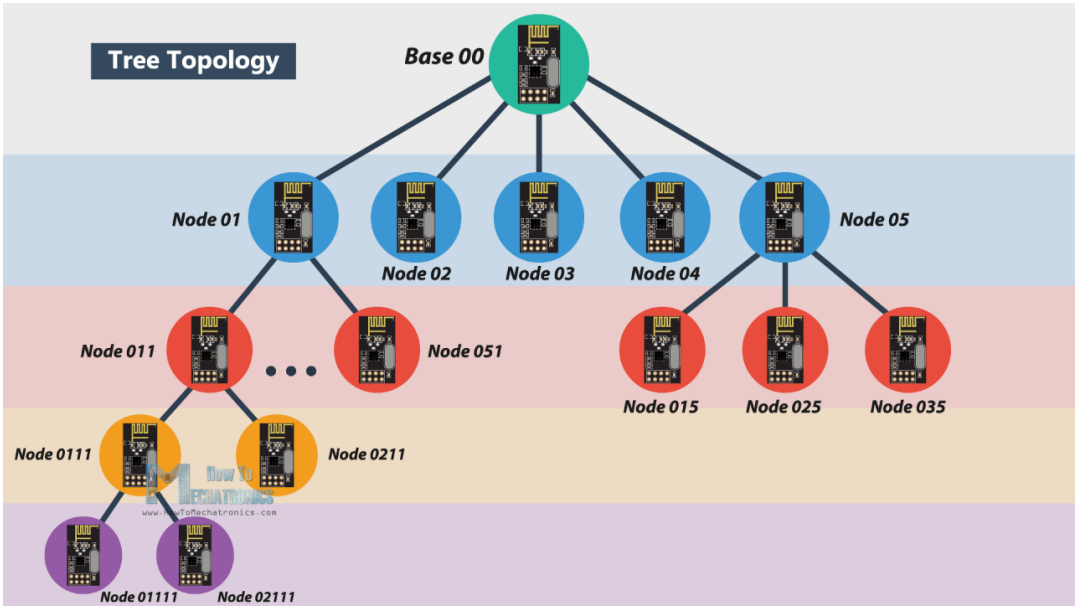
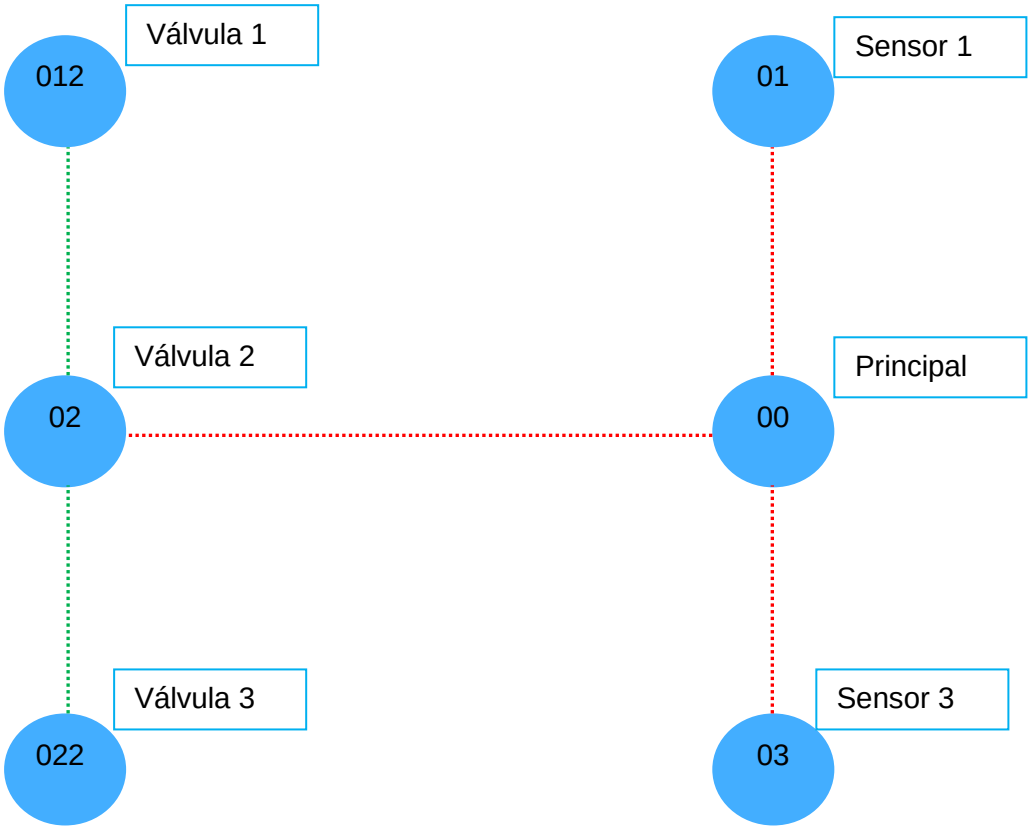


Fig. 8.5. Topología de la red

El Arduino principal y por tanto la base de la red nodal se ubicará junto con el segundo sensor. Desde esta posición se consiguen los caminos mínimos de comunicación. La red nodal del trabajo es de la siguiente manera:



9. Alimentación

Debido a que en la finca no se dispone de energía eléctrica los elementos del proyecto tendrán que alimentarse a partir de baterías. Además, para evitar la carga manual de las baterías se complementarán con placas solares de pequeña potencia. Los datos ofrecidos por el vendedor [9] son:

Característica	Valor
V	5V
P	2.5W
Imax	0.5A

Tabla 11.1. Características del panel solar

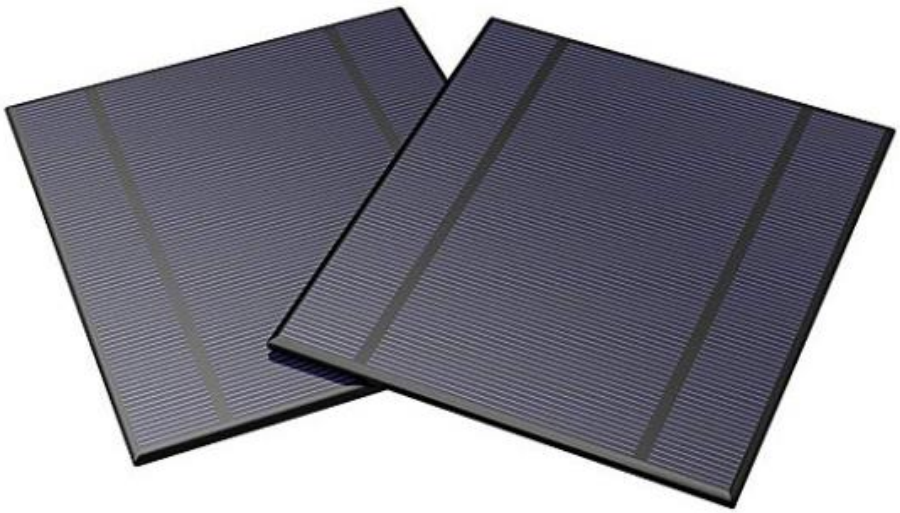


Fig. 13.1. Placas solares pequeña potencia

Las baterías escogidas son las 18650 por tener una gran capacidad en un espacio reducido y por ser las más usadas en este tipo de soluciones. Sus características extraídas de su datasheet [10] son:

Característica	Valor
$V_{nominal}$	3.7 V
V_{carga}	4.2 V
I_{carga}	0.52 A
$I_{descarga}$	0.52 A
Capacidad	2600 mAh

Tabla 11.2. Características batería 18650

Fig. 11.2. Baterías 18650

Por otro lado, será necesario un gestor de carga para cargar las baterías a partir de los paneles solares. La solución elegida es el gestor de carga TP4056 [11]**.** Este chip incluye todo lo necesario para cargar las baterías incluyendo una protección de descarga excesiva de la batería lo cual si ocurriese la dañaría. La batería se carga a una tensión constante de 4.2 V. En la siguiente tabla se muestran sus características [11]:

Característica	Valor
V_{in}	4 - 8 V
V_{carga}	4.2 V
I_{carga}	0.52 A

Tabla 11.3. Características gestor de carga TP4056

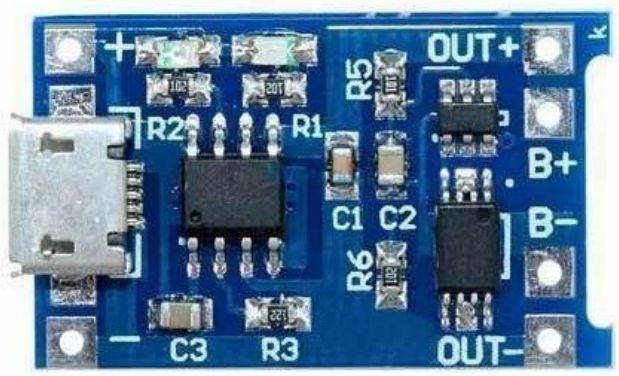


Fig. 11.3. Gestor de carga TP4056

Por último, serán necesarios convertidores de tensión DC-DC step up que den la tensión necesaria para los componentes. La solución escogida es el convertidor de tensión DC – DC regulable MT3608. A continuación, se exponen sus parámetros principales extraídos de su datasheet [12]:

Característica	Valor
V_{in}	4 - 8 V
V_{out}	4.2 V
I_{max}	0.52 A

Tabla 11.4. Características del convertidor de tensión DC-DC MT3608

Fig. 11.4. Convertidor de tensión DC-DC step up MT3608

9.3. Cálculos de la alimentación

Primeramente, se calculará la energía consumida por cada nodo de la instalación (válvula, sensor y principal) por día, multiplicando el consumo de cada uno de sus componentes por las horas que se prevé en funcionamiento y sumando estos consumos.

Una vez se sabe el consumo diario de cada nodo se puede calcular la capacidad necesaria, y con esta capacidad se calcula el número de baterías necesario y el número de placas solares necesarias para cargarlas.

11.1.1. Cálculo del consumo diario de cada nodo

Para asegurar que la instalación disponga de energía suficiente todo el tiempo, se supone que todo está funcionando las 24 horas del día, excepto las válvulas que se les pondrá un tiempo máximo de funcionamiento de 12 horas, el cual desde la experiencia del riego manual es más que suficiente. Los consumos se han medido con un multímetro, salvo los de la electroválvula que se preguntó al fabricante y el del NRF24L01 que se ha cogido de su hoja de características [13].

En cuanto al consumo del módulo NRF24L01 + PA + LNA se distinguen tres tipos de consumo, cuando envía datos, cuando recibe y cuando está en espera. En el principal la mayor parte del tiempo estará recibiendo, en las válvulas en espera y en los sensores enviando. Esto explica los diferentes consumos especificados en cada nodo para este componente.

La energía consumida se calcula multiplicando el consumo del componente por el tiempo considerado durante un día (fórmula 13.1).

Formula de la energía consumida diariamente:

$$E_{componente} = C * T \ (fórmula\ 13.1)$$

Donde:

$E_{componente}$ = energía consumida por el componente en un día

C = Consumo del componente

T = Tiempo considerado en un día

PRINCIPAL

Componente	Consumo	Tiempo en funcionamiento	Energía consumida
Arduino Mega	93 mA	24 h	2232 mAh
GSM/GPRS	29 mA	24 h	696 mAh
Sensor humedad	5 mA	24 h	120 mAh
NRF24L01 + PA + LNA	45 mA	24 h	1080 mAh

Total: 4128 mAh

SENSOR

Componente	Consumo	Tiempo en funcionamiento	Energía consumida
Arduino Nano	15 mA	24 h	360 mAh
Sensor humedad	5 mA	24 h	120 mAh
NRF24L01 + PA + LNA	115 mA	24 h	2760 mAh

Total: 3240 mAh

VÁLVULA

Componente	Consumo	Tiempo en funcionamiento	Energía consumida
Arduino Nano	15 mA	24 h	360 mAh
Válvula	200 mA	12 h	2400 mAh
Relé	74 mA	12 h	888 mAh
NRF24L01 + PA + LNA	4.2 uA	24 h	-

Total: 3648 mAh

11.1.2. Cálculo del número de baterías

La capacidad de las baterías escogidas es de 2600 mAh, pero no se pueden descargar por debajo de un 20%, por lo que la capacidad real se queda en 2080 mAh. Para calcular el número de baterías necesario, se divide la energía consumida diariamente por la capacidad de la batería y se redondea al número entero próximamente más alto (fórmula 13.2).

Fórmula del número de baterías:

$$N_B = \frac{E_{nodo}}{Q} \quad (fórmula\ 13.2)$$

Donde:

N_B = número de baterías

E_{nodo} = energía consumida por el nodo en un día

Q = capacidad de la batería (2080 mAh)

Tabla del cálculo del número de baterías:

Tipo de nodo	Energía consumida en un día	Número de baterías
Principal	4128 mA	2
Válvula	3648 mA	2
Sensor	3240 mA	2

11.1.3. Número de placas solares

La temporada de riego transcurre en la estación del verano por lo que se consideran 6 h de máxima potencia de las placas solares, esto equivale a una capacidad de cargar 3000 mAh si multiplicamos las horas de máxima potencia por la máxima corriente que entregan las placas que es de 0.5 A.

A continuación, se calcula el número de placas necesario. Para mayor seguridad, el cálculo se hará teniendo en cuenta las capacidades máximas de las baterías y no el consumo calculado de los nodos. El número de placas lo obtenemos de dividir la capacidad total de las baterías (2600 mAh) entre lo que es capaz de cargar cada placa (3000 mAh) según la fórmula 13.3.

Fórmula para calcular el número de placas solares:

$$N_{placas} = \frac{N_B * Q_{total}}{Q_{carga}} \quad (fórmula\ 13.3)$$

Dónde:

N_{placas} = número de placas solares necesarias

N_B =número de baterías

Q_{total} = Carga total de las baterías

Q_{carga} = Capacidad de carga de las placas solares

Tabla del cálculo del número de placas solares:

Tipo de nodo	Número de baterías	Número de placas solares
Principal	2	2
Válvula	2	2
Sensor	2	2

10. Programación

En este capítulo se explicarán los programas cargados en las placas Arduino del proyecto. El código completo se encuentra en el anexo I.

10.3. Principal

```
1.  //Librerías
2.  #include <RF24.h>
3.  #include <RF24Network.h>
4.  #include <SPI.h>
5.  #include <SoftwareSerial.h>
6.
7.  //Pin sensor 2 en el mismo arduino
8.  #define sensor2 A0
9.
10. SoftwareSerial SIM900(10, 11);   //SIM900(Tx,Rx);
11.
12. RF24 radio(48,49);               // nRF24L01 (CE,CSN)
13. RF24Network network(radio);      // Incluye the radio in the network
14.
15. // Dirección de los nodos
16.
17. //Nodo principal
18. const uint16_t this_node = 00;
19.
20. //Sensores
21. const uint16_t node01 = 01; //Sensor 1
22. const uint16_t node03 = 03; //Sensor 3
23.
24. //Válvulas
25. const uint16_t node012 = 012; //Válvula 1
26. const uint16_t node02 = 02; //Válvula 2
27. const uint16_t node022 = 022; //Válvula 3
28.
29. //Variables necesarias para NRF24L01
30. RF24NetworkHeader header;
31. RF24NetworkHeader header01(node01);
32. RF24NetworkHeader header03(node03);
33. RF24NetworkHeader header012(node012);
34. RF24NetworkHeader header02(node02);
35. RF24NetworkHeader header022(node022);
36.
37. //variables lectura de SMS
38. char lectura ;
39. String orden = "" ;
40. int modo = 0 ;
41. int valvula = 0 ;
42. int horas = 0 ;
43. unsigned long setpoint1, setpoint2, setpoint3;
44.
45. //Variable de estado de las válvulas
46. unsigned long estadoV ;
```

Primeramente, se incluyen las librerías que se van a usar en la programación (líneas 2-5). En la línea 8 se define el pin del sensor 2, conectado en el Arduino

principal. En las líneas 10, 12 y 13 se crean las instancias de las librerías SoftwareSerial, RF24 y RF24Network. A continuación de la línea 18 a la 27 se definen las direcciones de los diferentes nodos con los que le programa se va a comunicar para después definir los header de la comunicación. Por último, se declaran las variables para leer el SMS (líneas 38-43) y la variable del estado de la válvula.

```
47.   void setup() {
48.
49.     SIM900.begin(19200); //Configura velocidad del puerto serie para el
SIM900
50.     delay(1000);
51.     power_on();
52.     iniciar();
53.
54.     SPI.begin();
55.     radio.begin();
56.     network.begin(90, this_node);
57.     radio.setDataRate(RF24_2MBPS);
58. }
```

En el void setup() se inicia el funcionamiento de las librerías con el método begin(), se llaman los métodos power_on() e iniciar() y se configura la velocidad de comunicación de la comunicación por radio, con el método setDataRate().

```
59. void loop() {
60.
61.   if (SIM900.available()){
62.
63.     while (SIM900.available()){
64.       lectura = SIM900.read();
65.       orden = orden + lectura ;
66.       delay(25);
67.     }
68.     int pos = orden.indexOf("u"); //busca la posicion del inicio de la
    orden (letra u del SMS)
69.
70.     modo = int(orden.charAt(pos + 1)) - 48 ; //lee modo de
    funcionamiento, que viene en código ASCII y restándole 48 lo pasa
71.
72.     if (modo == 1){
73.       valvula = int(orden.charAt(pos + 2)) - 48 ;
74.
75.       horas = orden.substring(pos + 3, pos + 5).toInt();
76.     }
77.
78.     if (modo == 2){
79.       //Se leen los setpoints y se pasa a entero
80.       setpoint1 = orden.substring(pos + 2, pos + 5).toInt();
81.       setpoint2 = orden.substring(pos + 5, pos + 8).toInt();
82.       setpoint3 = orden.substring(pos + 8, pos + 11).toInt();
83.     }
84.
```

```
85.     orden = "" ;
86.   }
87.
```

Se inicia el void loop() y la primer bloque if() comprueba si hay alguna orden nueva y en caso de que la haya la lee y la guarda en la variable orden. A continuación, se procede a su lectura, identificando primero el modo escogido y en función del modo se leen las variables correspondientes. En la línea 85 se vacía la variable orden para dejarla preparada para escribir la siguiente.

```
88.   switch (modo){
89.
90.     case 1: //------------------------MODO MANUAL-------------------
91.
92.     unsigned long inicio; //Variable para temporizar
93.
94.     switch (valvula){
95.
96.       case 1: //ABRIR VÁLVULA 1
97.
98.       inicio = millis();
99.
100.         estadoV = 1;
101.         network.write(header012, &estadoV, sizeof(estadoV)); //Se abre
la válvula
102.         //delay(25);
103.
104.         while (SIM900.available() == 0 && millis()-
inicio <= horas*1000*3600){
105.           //Se espera a que pase el tiempo programado o a tener una
nueva orden
106.         }
107.
108.         if (SIM900.available() > 0 || millis()-
inicio >= horas*1000*3600){ //Se cierra la válvula cuando hay una orden
nueva o se pasa el tiempo programado
109.           estadoV = 0;
110.           network.write(header012, &estadoV, sizeof(estadoV));
111.           delay(100);
112.         }
113.
114.         break;
115.
116.  //(***)AQUÍ VAN LOS case 2 Y case 3(***)
117.     }//FIN del switch(valvula)
118.
119.      modo = 0 ;
120.
121.      break;//FIN del modo manual
```

Después de la lectura de la orden se encuentra un bloque switch() que en función del modo de funcionamiento escrito en el mensaje, dirigirá el programa al case 1, que es el modo manual o al case 2 que es el modo automático.

En el case 1 del modo manual se declara la variable inicio para guardar el tiempo que ha pasado desde que se inició el programa y poder compararlo después para la temporización. Una vez dentro del case 1 se encuentra otro bloque switch() que corresponde a la elección de la válvula: case 1 para la válvula 1, case 2 para la válvula 2 y case 3 para la válvula 3.

Los tres case hacen la misma función cambiando la dirección de los nodos: primero se guarda el tiempo transcurrido desde que empezó el programa a funcionar con el método millis() en la variable inicio. Después se procede a encender la válvula correspondiente y con el bucle while() de la línea 104 se mantiene abierta hasta que no pase el tiempo indicado o haya una orden nueva. Cuando se sale del bucle while() por alguno de los dos motivos descritos, se manda apagar la válvula. Se ha incluido la orden de apagar la válvula dentro de un if() para incluir un delay con el objetivo de dar tiempo a la comunicación de la orden.

Antes de salir del bloque switch(valvula) se le da el valor 0 a la variable modo para que no se vuelva a repetir el case.

```
155.      case 2: //------------------------MODO AUTOMÁTICO----------------
 ---
156.
157.      bool primera_vez; //Variables del modo automático
158.      bool leido;
159.      unsigned long humedad = 0 ;
160.      unsigned long giveMe ;
161.
162.      while (SIM900.available()== 0){
163.
164.  //---------------PRIMER SECTOR----------------
165.        primera_vez = true ;// Inicialización de las variables
166.        leido = false ;
167.        humedad = 0;
168.
169.        //Se le envía al sensor la orden de que envíe datos de humedad
170.        giveMe = 1 ;
171.        network.write(header01, &giveMe, sizeof(giveMe))
172.
173.        while (humedad < setpoint1 && SIM900.available() == 0){
174.
175.          network.update(); //carga la red
176.
177.          while (network.available()){ //Mientras que haya datos
 disponibles los lee
178.            network.read(header, &humedad, sizeof(humedad));
179.            leido = true;
180.          }
181.
182.          //Solo manda la orden de abrir la válvula una vez
```

```
183.
   if (leido == true && humedad < setpoint1 && primera_vez == true){
184.             estadoV = 1;
185.             network.write(header012, &estadoV, sizeof(estadoV));//Manda
 abrir la válvula
186.             primera_vez = false;
187.             delay(100);
188.           }
189.         }
190.         giveMe = 0;
191.         network.write(header01, &giveMe, sizeof(giveMe));//Manda que el
 sensor deje de enviar datos
192.
193.         //Si se ha abierto la válvula la cierra
194.         if (primera_vez == false){
195.           estadoV = 0;
196.           network.write(header012, &estadoV, sizeof(estadoV));//Manda
 cerrar la válvula
197.           delay(100);
198.         }
199.         inicio = millis();
200.         while (millis()-inicio < 2000){}//Espera para que no se mezclen
      los datos de los sensores
201.
```

El case 2 del switch(modo) empieza declarando las variables necearías para este modo de funcionamiento: la variable primera_vez para saber la primera vez que se abre la válvula y no volverla a abrir hasta que no se cierre, la variable leido para saber si se han leído datos procedentes del sensor, la variable humedad para guardar el dato de la humedad del sensor y la variable giveMe, que valdrá 1 cuando se quiere que el sensor nos mande datos y 0 cuando se quiere que deje de mandarlos.

Una vez declaradas las variables se inicia un bucle while() que obliga al programa a permanecer en el modo automático hasta que no se reciba una orden nueva.

En la parte del sector 1, primero se inicializan las variables, después se manda al sensor la orden de que envíe datos. A continuación se inicia un bucle while() donde permanecerá el programa hasta que la humedad sea superior al setpoint correspondiente o haya una orden nueva. En este bucle while() se van leyendo los datos del sensor y guardando en la variable humedad. El bloque if() de la línea 183 manda abrir la válvula e impide que la orden de abrir la válvula se repita.

Al salir del bucle while de la línea 173, se manda al sensor que deje de enviar datos y si la válvula ha sido abierta se manda cerrar.

Los demás sectores funcionan de la misma manera con la salvedad de que en el sector 2 los datos de humedad se cogen directamente del mismo Arduino utilizando el método analogRead().

10.4. Sensores

```
1.  //Librerías
2. #include <RF24.h>
3. #include <RF24Network.h>
4. #include <SPI.h>
5.
6. //Pin del sensor
7. #define pinSensor A0
8.
9. unsigned long humedad;
10.      unsigned long giveMe = 0 ;
11.
12.      //Instancias de las librerías
13.      RF24 radio(9,10);
14.      RF24Network network(radio);
15.
16.      //Dirección de los nodos
17.      const uint16_t this_node = 01;
18.      const uint16_t master00 = 00;
19.
20.      //Objeto de lectura de la librería RF24Network
21.      RF24NetworkHeader header;
22.
23.      //Objeto de escritura de la librería RF24Network
24.      RF24NetworkHeader header00(master00);
25.
26.      void setup() {
27.        SPI.begin();
28.        radio.begin();
29.        network.begin(90, this_node);
30.        radio.setDataRate(RF24_2MBPS);
31.      }
32.
33.      void loop() {
34.
35.        network.update();
36.
37.        while (network.available()){
38.          network.read(header, &giveMe, sizeof(giveMe));
39.        }
40.
41.        if (giveMe == 1){
42.
43.          humedad = analogRead(pinSensor);
44.          humedad = map(humedad,0,1023,0,100);
45.
46.          network.write(header00, &humedad, sizeof(humedad));

47.          delay(500);
48.        }
49.      }
```

Al principio del programa, se incluyen las librerías que se van a usar, se define el pin donde se encuentra el sensor y se declaran las variables necesarias para la programación. Después se crean las instancias de las librerías, se definen las direcciones de los nodos involucrados en el programa y se crean los objetos necesarios para la comunicación mediante la librería RF24Network.

En el void setup() se inician las librerías y se especifica la velocidad de comunicación por radio.

En el void loop() el programa va comprobando si hay datos que leer, y si los hay los almacena en la variable giveMe, que valdrá 1 si se le ha ordenado que mande datos. Cuando la variable giveMe cambia a 1 se procede a la lectura del sensor y el envío de datos cada 500 ms.

10.3. Válvula

```
1. //Librerías
2. #include <RF24.h>
3. #include <RF24Network.h>
4. #include <SPI.h>
5.
6. //Pin de la válvula
7. #define pinValvula 3
8.
9. unsigned long estadoV ;
10.
11.      //Instancias de las librerías
12.      RF24 radio(9,10);
13.      RF24Network network(radio);
14.      const uint16_t this_node = 012;
15.      const uint16_t master00 = 00;
16.
17.      //Objeto de lectura de la librería RF24Network
18.      RF24NetworkHeader header;
19.
20.      void setup() {
21.        SPI.begin();
22.        radio.begin();
23.        network.begin(90, this_node);
24.        radio.setDataRate(RF24_2MBPS);
25.        pinMode(pinValvula, OUTPUT);
26.      }
27.
28.      void loop() {
29.        network.update();
30.
31.        while (network.available()){
32.
33.          network.read(header, &estadoV, sizeof(estadoV));
34.          if (estadoV == 1){
35.            digitalWrite(pinValvula, HIGH);
```

```
36.           } else {
37.             digitalWrite(pinValvula, LOW);
38.           }
39.         }
40.       }
```

Primero se incluyen las librerías necesarias para el funcionamiento del programa, se define el pin del relé que acciona la válvula y se declara la variable estadoV que contiene el estado de la válvula. Después se crean las instancias de las librerías y se definen las direcciones de los nodos involucrados en el programa y el objeto necesario para la comunicación mediante la librería RF24Network.

En el void setup() se inician las librerías, se define la velocidad de comunicación por radio y se indica con el método pinMode() que el pin del relé es un pin de salida.

En el void loop() se comprueba constantemente si hay datos para la lectura, y si los hay se leen y se guardan en la variable estadoV. El bloque if() que se incluye a continuación activa el relé si la variable estadoV vale 1 y lo desactiva si vale 0.

11. APP Android

Una aplicación se divide en diferentes pantallas denominadas Activities con las que el usuario va interactuando. Cada pantalla posee dos partes de programación: una que describe su aspecto gráfico, y otra que define su comportamiento lógico.

A continuación, se describe el proceso de creación de una aplicación con Android Studio a nivel básico, para después explicar la programación de la aplicación de este trabajo.

11.3. Android Studio

Android Studio [7] es un entorno de desarrollo integrado (IDE), nombrado oficial por Google, para la creación de aplicaciones móviles. Está basado en IntelliJ IDEA. Los lenguajes de programación que se pueden usar son java y kotlin.

Para crear un nuevo proyecto se selecciona menú File, se accede al menú new… y se elige la opción New Project...

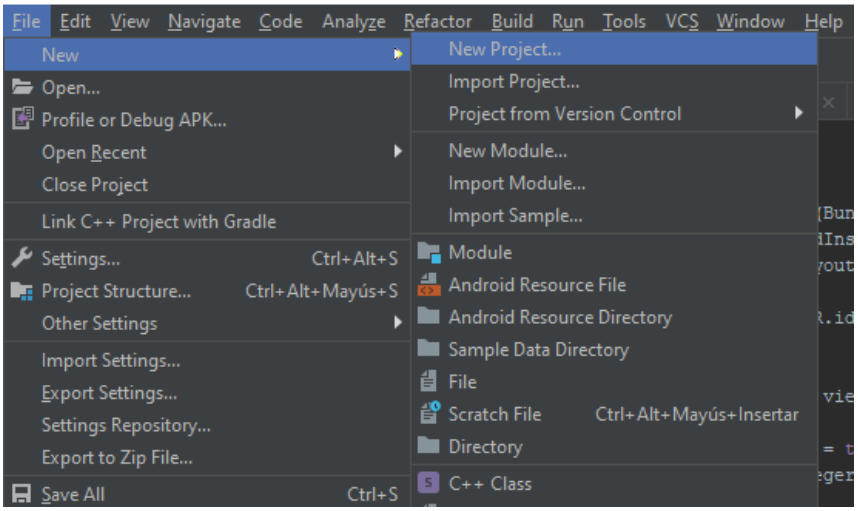


Fig. 10.1. Crear nuevo proyecto

A continuación, aparece la siguiente ventana, donde se selecciona la opción Empty Activity, para que se cree una Activity vacía, y se clica en siguiente.

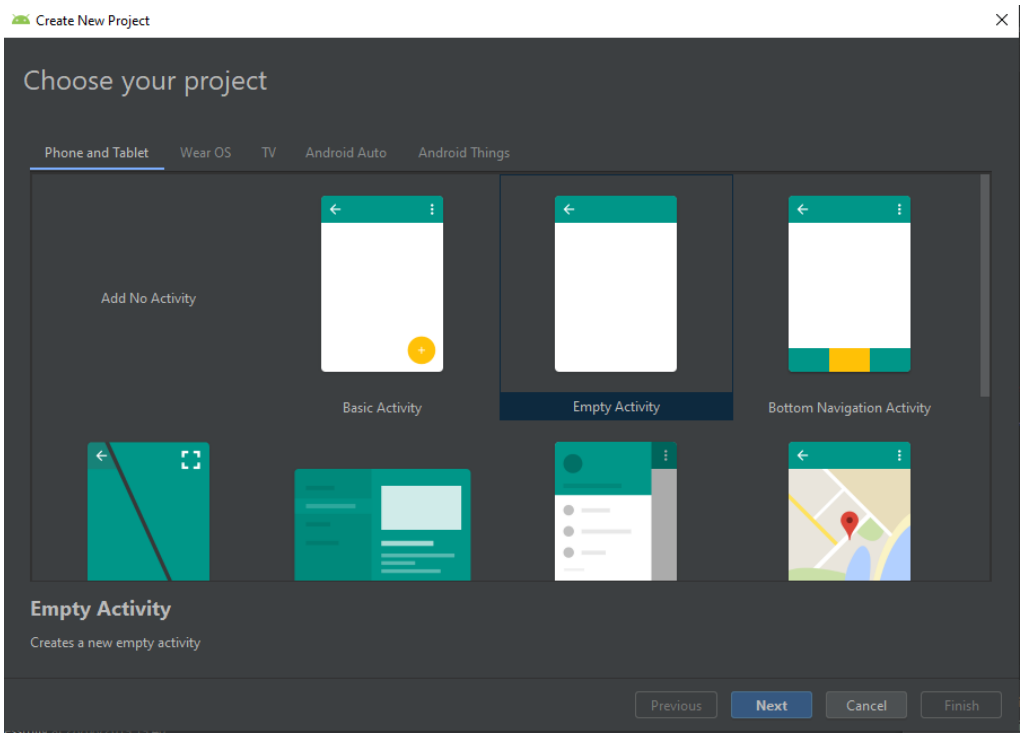


Fig. 10.2. Crear Activity vacía

En la siguiente ventana se elige el nombre de la APP, así como la ubicación de los archivos que genera archivos en el PC. También se elige el lenguaje de programación en el que se desea programar la aplicación. Por último, se selecciona la versión de Android. Se recomienda usar la que nos sale por defecto, pues es la que el programa considera que podrán usar más usuarios.

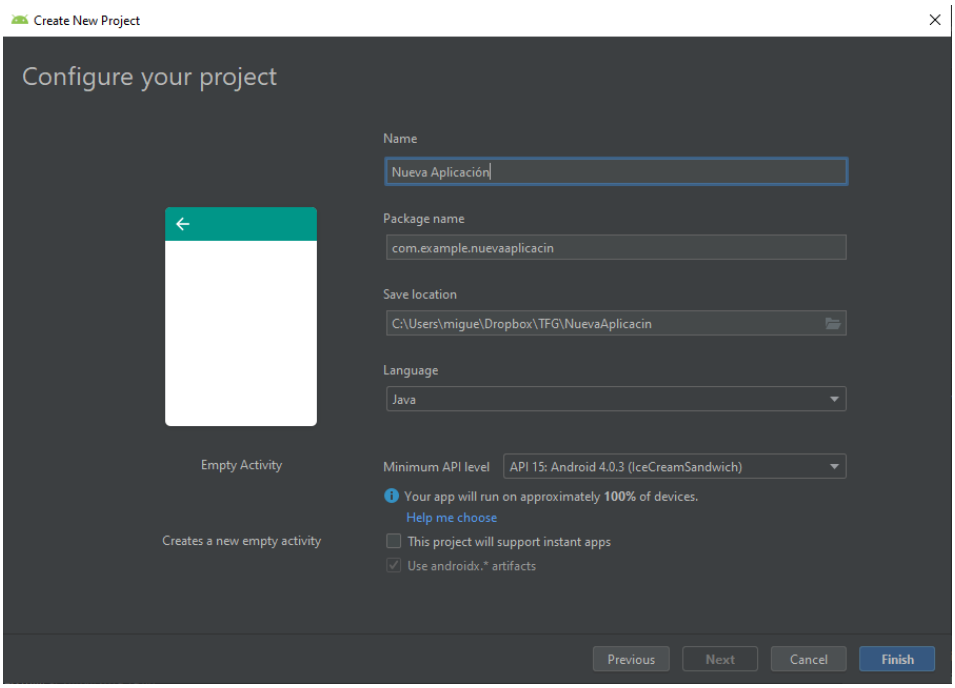


Fig. 10.3. Configuración de la aplicación

Una vez finalizada la configuración, el programa genera los archivos necesarios para empezar a programar la aplicación. En la parte izquierda de la pantalla aparece el árbol de archivos de la aplicación. Caben destacar las carpetas:

- Manifest: Contiene el fichero AndroidManifest.xml, dónde se describe la aplicación Android. Aquí también se declaran los permisos de la aplicación.
- java: Aquí se ubica la parte lógica de la aplicación en archivos .java.
- res: Esta carpeta contiene los recursos de la aplicación. La subcarpeta layout contiene los ficheros XML que describen la interfaz de las diferentes Activities que creemos.

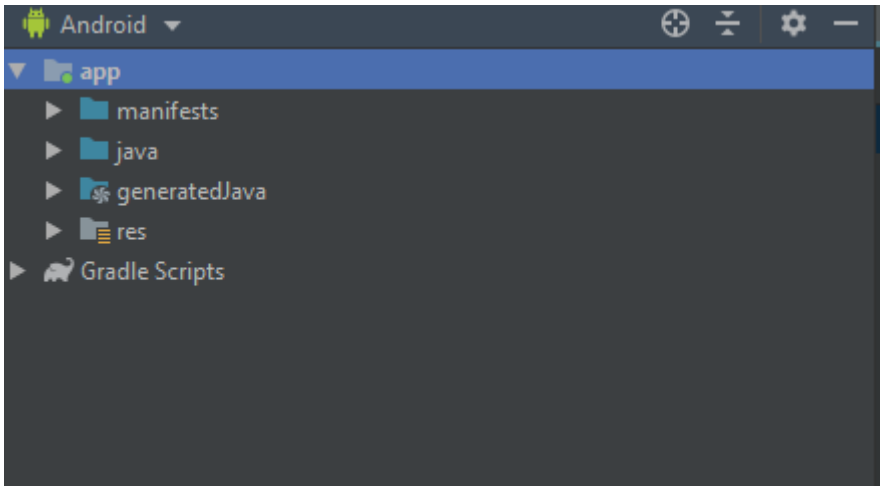


Fig. 10.4. Árbol de archivos de la aplicación

Cada Activity dispone de un fichero .java para programar su parte lógica y un archivo .xml para describirla gráficamente.

En el archivo .java aparece ya creada la clase del Acivity (MainActivity), clase en la que estará incluido el código que programemos. El programa también crea el método onCreate que será llamado cuando se inicie la aplicación. En esté método se realiza la inicialización y la configuración de la interfaz gráfica.

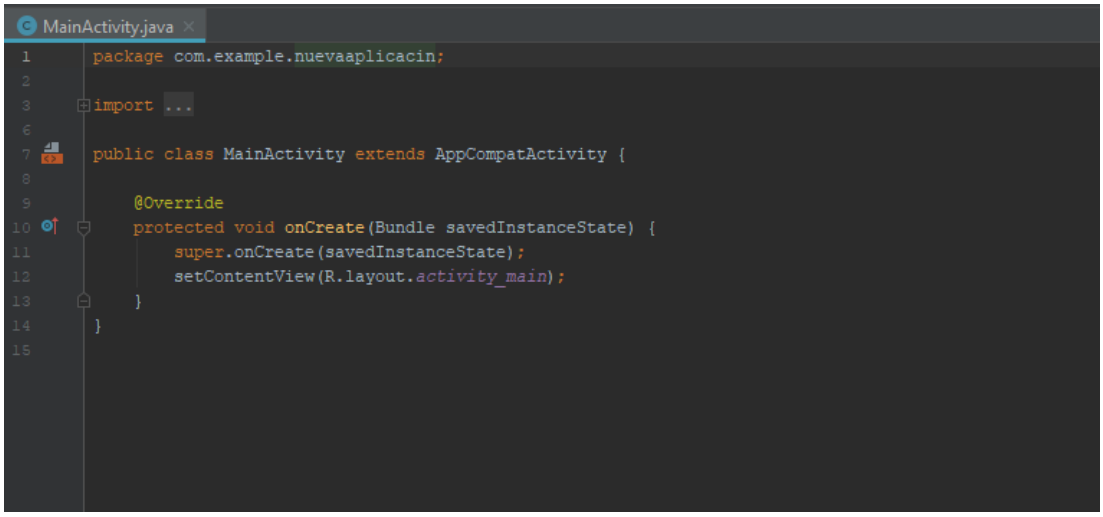


Fig. 10.5. Archivo .java de un activity

El archivo .xml contiene lo relacionado con la parte gráfica del activity. Existen dos formas de visualizarlo: en forma de código xml o mediante un asistente de diseño que incluye el programa. Se puede cambiar de una vista a otra seleccionándola en la parte inferior izquierda de la pantalla.

En la pestaña Design aparece la paleta con los componentes que se pueden usar a la hora de diseñar la interfaz gráfica de la APP. Los componentes se crean arrastrándolos desde la paleta a la vista del activity.

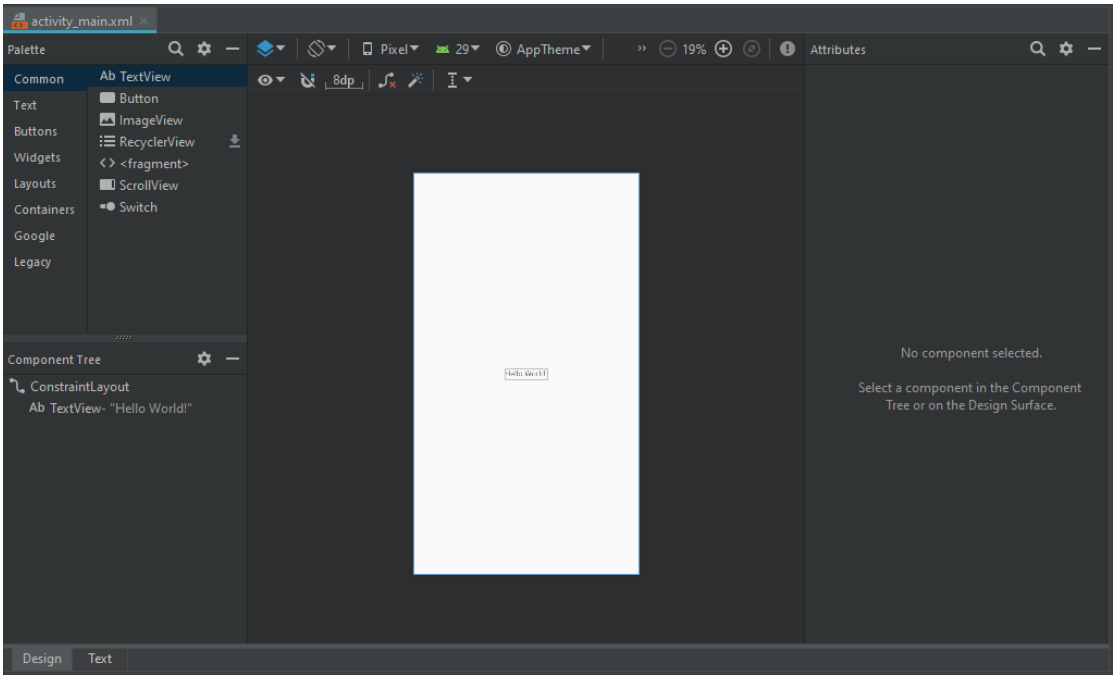


Fig. 10.6. Vista Design del archivo activity_main.xml

Una vez que se crea un elemento, arrastrándolo a la vista del activity, se definen las distancias con los márgenes y con otros elementos. Esto se hace pinchando en el elemento, y luego arrastrando de los puntos que aparecen en él hacia dónde se quiere definir la distancia.

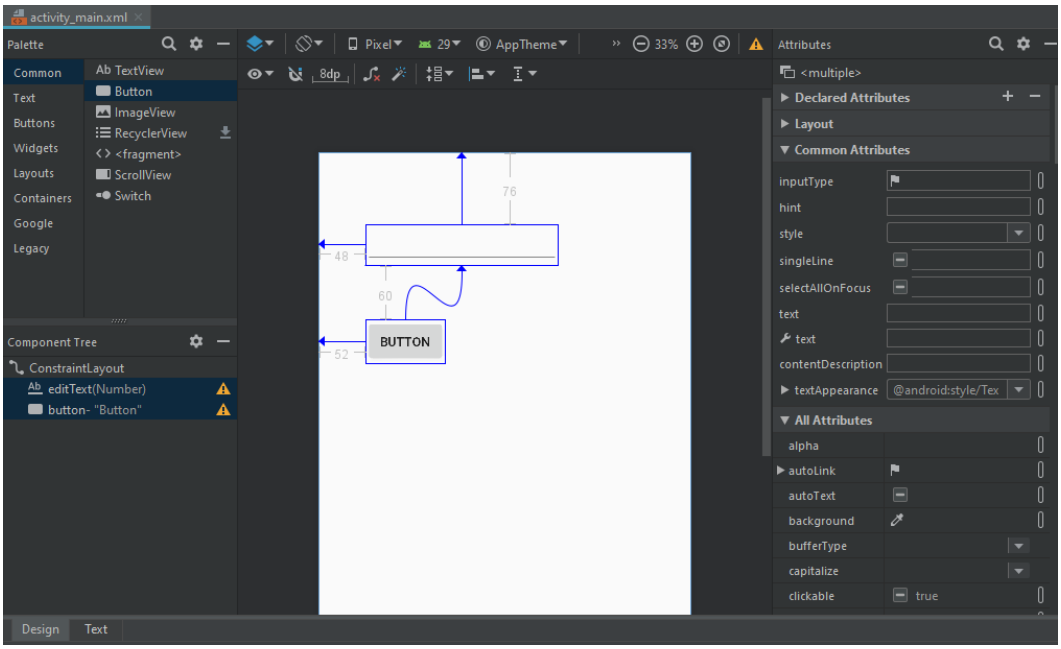


Fig. 10.7. Definiendo las distancias

Los atributos de cada elemento se pueden modificar en la ventana que aparece a la derecha cuando se selecciona un elemento.

11.4. El código de la aplicación

11.4.1. Activity “MainActivity”

Fig. 10.8. Aspecto de MainActivity

En este activity se crea un botón para seleccionar el modo manual y otro botón para seleccionar el modo automático. Se crea también un TextView para preguntar al usuario el modo que quiera escoger.

Parte lógica

```
1. package com.example.riego;
2.
3. import androidx.appcompat.app.AppCompatActivity;
4.
5. import android.content.Intent;
6. import android.os.Bundle;
7. import android.view.View;
8.
9. public class MainActivity extends AppCompatActivity {
10.
11.          @Override
12.          protected void onCreate(Bundle savedInstanceState) {
13.              super.onCreate(savedInstanceState);
14.              setContentView(R.layout.activity_main);
15.          }
16.
17.          //Método para el botón MANUAL
18.          public void Manual(View view){
19.              Intent manual = new Intent(this, MANUAL.class);
20.              startActivity(manual);
21.          }
22.
23.          //Método para el botón AUTOMÁTICO
24.          public void Automatico(View view){
25.              Intent automatico = new Intent(this, AUTOMATICO.class);
26.              startActivity(automatico);
27.          }
28.      }
```

En la parte lógica se crean los métodos “Manual” y “Automatico” (líneas 18 y 24) que se usan para que cuando el usuario pulse los botones se cambie al activity correspondiente. Para cambiar de activity se crea un objeto de tipo Intent que tiene dos parámetros, en el primero se incluye el activity del que se parte y en el segundo el activity al que se quiere ir. Después se usa el método “startActivity” que tiene como único parámetro el objeto de tipo Intent que se creó anteriormente.

11.4.2. Activity “AUTOMATICO”

Parte gráfica

Fig. 10.9. Aspecto de AUTOMATICO

En la parte gráfica de este activity se crea un primer TextView para dicirle al usuario que introduzca los setpoints, tres TextView que especifican que setpoint corresponde a cada sector, tres EditText que permitan al usuario introducir los setpoints de cada sector, el botón enviar que envía la orden a Arduino y por último un TextView que indica un consejo de utilización.

Parte lógica

```
1. package com.example.riego;
2.
3. import androidx.appcompat.app.AppCompatActivity;
4.
5. import android.os.Bundle;
6. import android.telephony.SmsManager;
7. import android.view.View;
8. import android.widget.EditText;
9. import android.widget.Toast;
10.
11.      public class AUTOMATICO extends AppCompatActivity {
12.
13.          private EditText et1, et2, et3;
14.
15.          @Override
16.          protected void onCreate(Bundle savedInstanceState) {
17.              super.onCreate(savedInstanceState);
18.              setContentView(R.layout.activity_automatico);
```

```
19.
20.             et1 = findViewById(R.id.txt_num1);
21.             et2 = findViewById(R.id.txt_num2);
22.             et3 = findViewById(R.id.txt_num3);
23.         }
24.
25.         //Método del boton ENVIAR
26.         public void Enviar(View view){
27.
28.             String valor1 = et1.getText().toString();
29.             String valor2 = et2.getText().toString();
30.             String valor3 = et3.getText().toString();
31.
32.             int num1 = Integer.parseInt(valor1);
33.             int num2 = Integer.parseInt(valor2);
34.             int num3 = Integer.parseInt(valor3);
35.
36.
    if (num1>100 || num2>100 || num3>100 || num1<0 || num2<0 ||num3<0)
  {
37.                 Toast.makeText(this, "ATENCIÓN: Los porcentajes
  deben estar entre 0 y 100", Toast.LENGTH_SHORT).show();
38.             }else{
39.
40.                 if (num1!=100 && num1 >= 10){
41.                     valor1 = "0" + valor1;
42.                 }
43.                 if (num2!=100 && num2 >= 10){
44.                     valor2 = "0" + valor2;
45.                 }
46.                 if (num3!=100 && num2 >= 10){
47.                     valor3 = "0" + valor3;
48.                 }
49.                 if (num1 < 10){
50.                     valor1 = "00" + valor1;
51.                 }
52.                 if (num2 < 10){
53.                     valor2 = "00" + valor2;
54.                 }
55.                 if (num3 < 10){
56.                     valor3 = "00" + valor3;
57.                 }
58.
59.                 String mensaje = "u2" + valor1 + valor2 + valor3;
60.
61.                 String phone = "XXXXXXXXX";
62.
63.                 SmsManager sms = SmsManager.getDefault();
64.                 sms.sendTextMessage(phone, null, mensaje
  , null, null);
65.
66.                 Toast.makeText(this, "Orden enviada correctamente",
  Toast.LENGTH_LONG).show();
67.             }
68.         }
69.     }
```

Primeramente, se crea tres objetos de tipo EditText para los setpoints (línea 13). En el método “onCreate” se indica de donde provienen los valores de dichos objetos con el método “findViewById ()”.

En el método del botón se recopilan los valores escritos por el usuario y se guardan en una variable tipo string (líneas 28, 29 y 30) para después convertirlos a variable tipo int (líneas 32, 33 y 34) para poder validar los datos. A continuación, en el bloque if de la línea 36 se validan los valores, que al ser porcentajes deben estar entre 0 y 100. Una vez validados, para que el mensaje esté escrito de forma que Arduino pueda leerlo correctamente, se añade un cero delante si el número es de dos cifras y dos ceros si el número es de dos cifras (líneas 40-57).

Por último, se declaran las variables tipo string que contienen el número de teléfono y el mensaje (líneas 59 y 61), y se crea un objeto de la librería “SmsManager” (línea 63) para después usar el método “sendTextMessage” de la librería, para enviar la orden en forma de SMS (línea 64) incluyendo los parámetros del método necearios: mensaje y número de teléfono. Una vez enviado el mensaje la aplicación nos muestra un mensaje emergente informándonos de que el SMS ha sido enviado.

11.4.3. Activity “MANUAL”

Fig. 10.10. Aspecto de MANUAL

Se incluye aquí un TextView preguntado al usuario qué sector desea regar y tres botones para que se seleccione el sector que se desee.

Parte lógica

```
1. package com.example.riego;
2.
3. import androidx.appcompat.app.AppCompatActivity;
4.
5. import android.content.Intent;
6. import android.os.Bundle;
7. import android.view.View;
8. import android.widget.Toast;
9.
10.      public class MANUAL extends AppCompatActivity {
11.
12.          @Override
13.          protected void onCreate(Bundle savedInstanceState) {
14.              super.onCreate(savedInstanceState);
15.              setContentView(R.layout.activity_manual);
16.          }
17.
18.          //Método botón SECTOR 1
19.          public void SECTOR1(View view){
20.              Intent sector1 = new Intent(this, SECTOR1.class);
21.              startActivity(sector1);
22.
23.          }
24.
25.          //Método botón SECTOR 2
26.          public void SECTOR2(View view){
27.              Intent sector2 = new Intent(this, SECTOR2.class);
28.              startActivity(sector2);
29.          }
30.
31.          //Método botón SECTOR 3
32.          public void SECTOR3(View view){
33.              Intent sector3 = new Intent(this, SECTOR3.class);
34.              startActivity(sector3);
35.          }
36.      }
```

En la parte lógica de este activity se crean los métodos de los botones, que tienen como función dirigir la aplicación hacia el activity del sector seleccionado.

11.4.4. Activity “SECTOR1”, “SECTOR2”, y “SECTOR3”

Parte gráfica

Fig. 10.11. Aspecto de SECTOR1, SECTOR2 Y SECTOR3

La parte gráfica de los activities SECTOR1, SECTOR2 y SECTOR3 constan de un TextView a modo de título, un EditText para que el usuario indique las horas que quiere regar y un botón para enviar la orden.

Parte lógica

```
1. package com.example.riego;
2.
3. import androidx.appcompat.app.AppCompatActivity;
4.
5. import android.os.Bundle;
6. import android.telephony.SmsManager;
7. import android.view.View;
8. import android.widget.EditText;
9. import android.widget.Toast;
10.
11.      public class SECTOR1 extends AppCompatActivity {
12.
13.          EditText tv1;
14.
15.          @Override
16.          protected void onCreate(Bundle savedInstanceState) {
17.              super.onCreate(savedInstanceState);
18.              setContentView(R.layout.activity_sector1);
19.
20.              tv1 = findViewById(R.id.txt_tiempo);
21.          }
22.
23.          //Método para el botón ENVIAR
24.          public void Enviar(View view){
```

```
25.
26.             String horas_string = tv1.getText().toString();
27.             int horas_int = Integer.parseInt(horas_string);
28.
29.             if (horas_int<0){
30.                 Toast.makeText(this,"El número de horas no puede
   ser inferior a 0", Toast.LENGTH_SHORT).show();
31.             }else if(horas_int>12) {
32.                 Toast.makeText(this, "No puede regar más de 12 h
   seguidas", Toast.LENGTH_SHORT).show();
33.             }else if(horas_int<10){
34.                 horas_string = "0" + horas_string;
35.             }else{
36.                 String mensaje = "u11" + horas_string;
37.                 String phone = "XXXXXXXXX";
38.
39.                 SmsManager sms = SmsManager.getDefault();
40.                 sms.sendTextMessage(phone, null, mensaje
   , null, null);
41.
42.                 Toast.makeText(this, "Orden enviada correctamente",
   Toast.LENGTH_LONG).show();
43.
44.             }
45.         }
46.     }
```

Se crea un objeto de tipo EditText para recoger el dato de las horas en la línea 13 y en el método onCreate(), en la línea 20, se le indica al programa la dirección de donde tiene que coger el valor de las horas de la parte gráfica.

En el método del botón, primero se recopila el valor de horas en string y después se pasa a entero para poder validarlo. A continuación, se procede a su validación, incluyendo mensajes de error en caso de que el valor introducido por el usuario no sea válido. Por último, se construye el mensaje y se indica el número de teléfono para después enviar el SMS tal como se hizo en el activity “AUTOMATICO”.

12. Maqueta

Para ilustrar el funcionamiento de la instalación se ha construido una maqueta. Los sensores serán representados por potenciómetros y las electroválvulas con diodos LED.

La maqueta posee todos los elementos referentes a las telecomunicaciones, cada nodo posee un módulo NRF24L01 y al principal, ubicado junto al sensor del segundo sector se le ha incluido el módulo GSM/GPRS.

Se alimentan los elementos con pilas de 9 V, salvo el módulo GSM/GPRS que su alimentación será a partir de un adaptador AC/DC a 9 V.

La ubicación de los elementos en la maqueta tiene la misma disposición que se tendrá en la instalación real.

Los programas cargados en las placas Arduino son los mismos que los que se cargarán en la instalación real, excepto el cambio de la temporización en el modo manual de horas a segundos para poder ver mejor el funcionamiento.

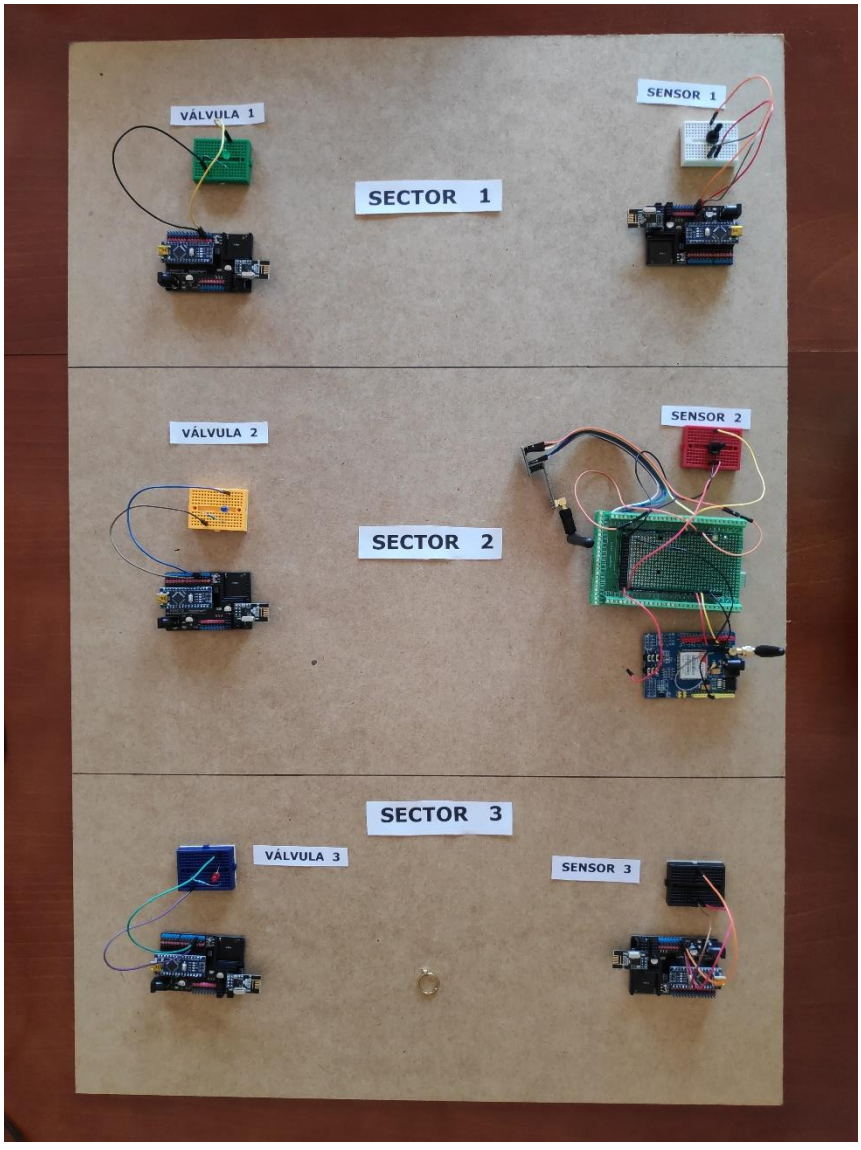


Fig. 12.1. Maqueta de la instalación

13. Presupuesto

El trabajo ha sido concebido sin ánimo de lucro por lo que no se tendrá en cuenta ni la mano de obra ni el beneficio industrial. El IVA está incluido en los precios. Se calculan los costes de las unidades básicas para después en el resumen calcular el precio total.

PRINCIPAL

Componente	Precio	Cantidad	Total
Arduino Mega	13.99 €	1	13.99 €
GSM/GPRS	22.99 €	1	22.99 €
Sensor humedad capacitivo	4.76 €	1	4.76 €
NRF24L01 + PA + LNA	4.66 €	1	4.66 €
Panel solar 5 V 2.5W	6.5 €	2	13 €
Batería litio 18650 2600 mAh	6.75 €	2	13.5 €
Gestor de carga TP4056	1.6 €	1	1.6 €
Convertidor elevador de tensión MT3608	1.5 €	1	1.5 €
Estaca de madera de 2 m	1.33 €	1	1.33 €
Caja estanca de PVC 220x170x140 mm	18.11 €	1	18.11 €
Pequeño material	5 €	1	5 €

Total: 100.44 €

SENSOR

Componente	Precio	Cantidad	Total
Arduino Nano	7.49 €	1	7.49 €
Sensor humedad capacitivo	4.76 €	1	4.76 €
NRF24L01 + PA + LNA	4.66 €	1	4.66 €
Panel solar 5 V 2.5W	6.5 €	2	13 €
Batería litio 18650 2600 mAh	6.75 €	2	13.5 €
Gestor de carga TP4056	1.6 €	1	1.6 €
Convertidor elevador de tensión MT3608	1.5 €	1	1.5 €
Estaca de madera de 2 m	1.33 €	1	1.33 €
Caja estanca de PVC 220x170x140 mm	18.11 €	1	18.11 €

Pequeño material	5 €	1	5 €

Total: 70.95 €

VÁLVULA

Componente	Precio	Cantidad	Total
Arduino Nano	7.49 €	1	7.49 €
Electroválvula Cepex 9 V latch	21 €	1	21 €
Relé	1.8 €	1	1.8 €
NRF24L01 + PA + LNA	4.66 €	1	4.66 €
Panel solar 5 V 2.5W	6.5 €	2	13 €
Batería litio 18650 2600 mAh	6.75 €	2	13.5 €
Gestor de carga TP4056	1.6 €	1	1.6 €
Convertidor elevador de tensión MT3608	1.5 €	1	1.5 €
Estaca de madera de 2 m	1.33 €	1	1.33 €
Caja estanca de PVC 220x170x140 mm	18.11 €	1	18.11 €
Pequeño material	5 €	1	5 €

Total: 88.99 €

RESUMEN

Unidad básica	Precio	Cantidad	Total
Principal	100.44 €	1	100.44 €
Sensor	70.95 €	2	141.9 €
Válvula	88.89 €	3	266.67 €

Total: 509.01 €

14. Conclusiones

El proyecto facilitará la vida enormemente al usuario, pues ya no tendrá que visitar la finca con tanta asiduidad. Además de esta ventaja, indirectamente, se han conseguido las siguientes:

- Mejor aprovechamiento del agua.
- Se evita el estrés hídrico al que se somete al olivo cuando se cambia su estado de humedad de forma brusca, cosa que sucede en los riegos manuales.
- Utilización de energía limpia.
- Universalidad del sistema para cualquier finca.
- Fácil instalación de los componentes.
- Aprovechamiento de la instalación hidráulica existente.

Como beneficio personal, mis conocimientos tecnológicos han aumentado considerablemente aprendiendo a implementar aplicaciones Android, programando comunicaciones de radiofrecuencia y GSM, aprendiendo a diseñar una alimentación con placas solares de pequeña potencia y todos los conocimientos que indirectamente han intervenido el trabajo.

15. Mejoras en el futuro

El trabajo tiene algunos inconvenientes que dan pie a mejoras futuras tales como:

- Mejora de la interfaz gráfica de la aplicación
- Que la aplicación recuerde los datos
- Ahorro de energía de los componentes del proyecto

16. Referencias

- [1]: http://brioagro.es/sensores-humedad-olivar/
- [2]: https://www.elarduino.com/mega/
- [3]: https://www.elarduino.com/nano/

- [4]: https://www.prometec.net/gprs-enviar-recibir-llamadas-sms/
- [5]: https://howtomechatronics.com/tutorials/arduino/how-to-build-an-arduino-wireless-network-with-multiple-nrf24l01-modules/
- [4]: https://www.prometec.net/nrf2401/
- [5]: https://naylampmechatronics.com/blog/16_Tutorial-b%C3%A1sico-NRF24L01-con-Arduino.html
- [6]: https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/

- [7]: El gran libro de Android. Jesús Tomás. Ed. Marcombo
- [8]: https://howtomechatronics.com/tutorials/arduino/how-to-build-an-arduino-wireless-network-with-multiple-nrf24l01-modules/
- [9]: https://www.hackster.io/igorF2/solar-charged-battery-powered-arduino-uno-645d89
- [10]: https://www.amazon.es/ALLPOWERS-Encapsulado-Bater%C3%ADa-Cargador-130x150mm/dp/B073XKPWY7/ref=sr_1_1?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=placa+solar+5v&qid=1567094237&s=gateway&sr=8-1#customerReviews
- [11]: https://www.ineltro.ch/media/downloads/SAAItem/45/45958/36e3e7f3-2049-4adb-a2a7-79c654d92915.pdf
- [12]: https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/TP4056.pdf
- [13]: https://www.olimex.com/Products/Breadboarding/BB-PWR-3608/resources/MT3608.pdf
- [14]: https://www.robotshop.com/media/files/pdf/datasheet-wir020.pdf

Anexo I. Código Arduino

Principal

```
0. //Librerías
1. #include <RF24.h>
2. #include <RF24Network.h>
3. #include <SPI.h>
4. #include <SoftwareSerial.h>
5.
6. //Pin sensor 2 en el mismo arduino
7. #define sensor2 A0
8.
9. SoftwareSerial SIM900(10, 11);
10.
11.      RF24 radio(48,49);
12.      RF24Network network(radio);
13.
14.      // Dirección de los nodos
15.
16.      //Nodo principal
17.      const uint16_t this_node = 00;
18.
19.      //Sensores
20.      const uint16_t node01 = 01; //Sensor 1
21.      const uint16_t node03 = 03; //Sensor 3
22.
23.      //Válvulas
24.      const uint16_t node012 = 012; //Válvula 1
25.      const uint16_t node02 = 02; //Válvula 2
26.      const uint16_t node022 = 022; //Válvula 3
27.
28.      //Variables necesarias para NRF24L01
29.      RF24NetworkHeader header;
30.      RF24NetworkHeader header01(node01);
31.      RF24NetworkHeader header03(node03);
```

```
32.      RF24NetworkHeader header012(node012);
33.      RF24NetworkHeader header02(node02);
34.      RF24NetworkHeader header022(node022);
35.
36.      //variables lectura de SMS
37.      char lectura ;
38.      String orden = "" ;
39.      int modo = 0 ;
40.      int valvula = 0 ;
41.      int horas = 0 ;
42.      unsigned long setpoint1, setpoint2, setpoint3;
43.
44.      //Variable de estado de las válvulas
45.      unsigned long estadoV ;
46.
47.      void setup() {
48.
49.        SIM900.begin(19200); //Configura velocidad del puerto serie para el SIM900
50.        Serial.begin(19200); //Configura velocidad del puerto serie del Arduino
51.        delay(1000);
52.        power_on();
53.        iniciar();
54.
55.        SPI.begin();
56.        radio.begin();
57.        network.begin(90, this_node);
58.        radio.setDataRate(RF24_2MBPS);
59.      }
60.
61.      void loop() {
62.
63.        if (SIM900.available()){
64.
65.          while (SIM900.available()){
66.            lectura = SIM900.read();
67.            orden = orden + lectura ;
68.            delay(25);
69.          }
70.          int pos = orden.indexOf("u"); //busca la posicion del inicio de la orden (letra u del SMS)
71.
```

```
72.          modo = int(orden.charAt(pos + 1)) - 48 ; //lee modo de funcionamiento, que viene en código ASCII y
   restándole 48 lo pasa
73.
74.          if (modo == 1){
75.            valvula = int(orden.charAt(pos + 2)) - 48 ;
76.
77.            horas = orden.substring(pos + 3, pos + 5).toInt();
78.          }
79.
80.          if (modo == 2){
81.            //Se leen los setpoints y se pasa a entero
82.            setpoint1 = orden.substring(pos + 2, pos + 5).toInt();
83.            setpoint2 = orden.substring(pos + 5, pos + 8).toInt();
84.            setpoint3 = orden.substring(pos + 8, pos + 11).toInt();
85.          }
86.
87.          orden = "" ;
88.        }
89.        switch (modo){
90.
91.          case 1: //------------------------MODO MANUAL-------------------
92.
93.          unsigned long inicio; //Variable para temporizar
94.
95.          switch (valvula){
96.
97.            case 1: //ABRIR VÁLVULA 1
98.
99.            inicio = millis();
100.
101.           estadoV = 1;
102.           network.write(header012, &estadoV, sizeof(estadoV)); //Se abre la válvula
103.           //delay(25);
104.
105.           while (SIM900.available() == 0 && millis()-inicio <= horas*1000*3600){
106.             //Se espera a que pase el tiempo programado o a tener una nueva orden
107.           }
108.
109.           if (SIM900.available() > 0 || millis()-inicio >= horas*1000*3600){ //Se cierra la válvula cuando
   hay una orden nueva o se pasa el tiempo programado
```

```
110.             estadoV = 0;
111.             network.write(header012, &estadoV, sizeof(estadoV));
112.             delay(100);
113.           }
114.
115.           break;
116.           case 2://ABRIR VÁLVULA 2
117.
118.           inicio = millis(); // Tiempo desde que empezo a funcionar Arduino en milisegundos
119.
120.           estadoV = 1;
121.           network.write(header02, &estadoV, sizeof(estadoV)); //Se abre la válvula
122.
123.           while (SIM900.available() == 0 && millis()-inicio <= horas*1000*3600){
124.             //Se espera a que pase el tiempo programado o a tener una nueva orden
125.           }
126.           if (SIM900.available() > 0 || millis()-inicio >= horas*1000*3600){ //Se cierra la válvula cuando
   hay una orden nueva o se pasa el tiempo programado
127.             estadoV = 0;
128.             network.write(header02, &estadoV, sizeof(estadoV));
129.             delay(100);
130.           }
131.           break;
132.
133.           case 3://ABRIR VÁLVULA 3
134.
135.           inicio = millis();
136.
137.           estadoV = 1;
138.           network.write(header022, &estadoV, sizeof(estadoV)); //Se abre la válvula
139.
140.           while (SIM900.available() == 0 && millis()-inicio <= horas*1000*3600){
141.             //Se espera a que pase el tiempo programado o a tener una nueva orden
142.           }
143.           if (SIM900.available()>0 || millis()-inicio >= horas*1000*3600){//Se cierra la válvula cuando hay
   una orden nueva o se pasa el tiempo programado
144.             estadoV = 0;
145.             ok = network.write(header022, &estadoV, sizeof(estadoV));
146.             delay(100);
147.           }
```

```
148.           break;
149.         }//FIN del switch(valvula)
150.
151.         modo = 0 ;
152.
153.         break;//FIN del modo manual
154.
155.         case 2: //------------------------MODO AUTOMÁTICO-------------------
156.
157.         bool primera_vez; //Variables del modo automático
158.         bool leido;
159.         unsigned long humedad = 0 ;
160.         unsigned long giveMe ;
161.
162.         while (SIM900.available()== 0){
163.
164.     //---------------PRIMER SECTOR----------------
165.           primera_vez = true ;// Inicialización de las variables
166.           leido = false ;
167.           humedad = 0;
168.           inicio = millis();
169.
170.           //Se le envía al sensor la orden de que envíe datos de humedad
171.           giveMe = 1 ;
172.           network.write(header01, &giveMe, sizeof(giveMe));
173.
174.           while (humedad < setpoint1 && SIM900.available() == 0){
175.
176.             network.update(); //carga la red
177.
178.             while (network.available()){ //Mientras que haya datos disponibles los lee
179.               network.read(header, &humedad, sizeof(humedad));
180.               leido = true;
181.             }
182.
183.             //Solo manda la orden de abrir la válvula una vez
184.             if (leido == true && humedad < setpoint1 && primera_vez == true){
185.               estadoV = 1;
186.               network.write(header012, &estadoV, sizeof(estadoV));//Manda abrir la válvula
187.               primera_vez = false;
```

```
188.               delay(100);
189.             }
190.           }
191.           giveMe = 0;
192.           network.write(header01, &giveMe, sizeof(giveMe));//Manda que el sensor deje de enviar datos
193.
194.           //Si se ha abierto la válvula la cierra
195.           if (primera_vez == false){
196.             estadoV = 0;
197.             network.write(header012, &estadoV, sizeof(estadoV));//Manda cerrar la válvula
198.             delay(100);
199.           }
200.           inicio = millis();
201.           while (millis()-inicio < 2000){}//Espera para que no se mezclen los datos de los sensores
202.
203.       //---------SEGUNDO SECTOR-----------
204.           primera_vez = true ;// Inicialización de las variables
205.           humedad = 0;
206.           inicio = millis();
207.
208.           while (humedad < setpoint2 && SIM900.available() == 0){
209.
210.             humedad = analogRead(sensor2); //Lectura del sensor 2 contenido en el principal
211.             humedad = map(humedad,0,1023,100,0); //Mapeo de la lectura
212.
213.             //Solo manda la orden de abrir la válvula una vez
214.             if (humedad < setpoint2 && primera_vez == true){
215.               estadoV = 1;
216.               network.write(header02, &estadoV, sizeof(estadoV));//Manda abrir la válvula 2
217.               primera_vez = false ;
218.               delay(100);
219.             }
220.           }
221.
222.           if (primera_vez == false){ //Si se ha abierto la válvula, se cierra
223.             estadoV = 0;
224.             network.write(header02, &estadoV, sizeof(estadoV));
225.             delay(100);
226.           }
227.           inicio = millis();
```

```
228.           while (millis()-inicio < 2000){}//Espera para que no se mezclen los datos de los sensores
229.
230.
231.     //---------------TERCER SECTOR----------------
232.           primera_vez = true ;// Inicialización de las variables
233.           leido = false ;
234.           humedad = 0;
235.           inicio = millis();
236.
237.           //Se le envía al sensor la orden de que envíe datos de humedad
238.           giveMe = 1 ;
239.           network.write(header03, &giveMe, sizeof(giveMe));
240.
241.           while (humedad < setpoint3 && SIM900.available() == 0){
242.
243.             network.update(); //carga la red
244.
245.             while (network.available()){ //Mientras que haya datos disponibles los lee
246.               network.read(header, &humedad, sizeof(humedad));
247.               leido = true;
248.             }
249.
250.             //Solo manda la orden de abrir la válvula una vez
251.             if (leido == true && humedad < setpoint3 && primera_vez == true){
252.               estadoV = 1;
253.               network.write(header022, &estadoV, sizeof(estadoV));//Manda abrir la válvula
254.               primera_vez = false;
255.               delay(100);
256.             }
257.           }
258.           giveMe = 0;
259.           network.write(header03, &giveMe, sizeof(giveMe));//Manda que el sensor deje de enviar datos
260.
261.           //Si se ha abierto la válvula la cierra
262.           if (primera_vez == false){
263.             estadoV = 0;
264.             network.write(header022, &estadoV, sizeof(estadoV));//Manda cerrar la válvula
265.             delay(100);
266.           }
267.           inicio = millis();
```

```
268.           while (millis()-inicio < 2000){}//Espera para que no se mezclen los datos de los sensores
269.
270.         }
271.         break;
272.       }
273.     }
274.
275.
276.
277.     //Funciones del módulo GSM/GPRS
278.     int enviarAT(String ATcommand, char* resp_correcta, unsigned int tiempo)
279.     {
280.
281.       int x = 0;
282.       bool correcto = 0;
283.       char respuesta[100];
284.       unsigned long anterior;
285.
286.       memset(respuesta, '\0', 100); // Inicializa el string
287.       delay(100);
288.       while ( SIM900.available() > 0) SIM900.read(); // Limpia el buffer de entrada
289.       SIM900.println(ATcommand); // Envia el comando AT
290.       x = 0;
291.       anterior = millis();
292.       // Espera una respuesta
293.       do {
294.         // si hay datos el buffer de entrada del UART lee y comprueba la respuesta
295.         if (SIM900.available() != 0)
296.         {
297.             respuesta[x] = SIM900.read();
298.             x++;
299.           // Comprueba si la respuesta es correcta
300.           if (strstr(respuesta, resp_correcta) != NULL)
301.           {
302.             correcto = 1;
303.           }
304.         }
305.       }
306.       // Espera hasta tener una respuesta
307.       while ((correcto == 0) && ((millis() - anterior) < tiempo));
```

```
308.       Serial.println(respuesta);
309.
310.       return correcto;
311.     }
312.
313.     void power_on()
314.     {
315.       int respuesta = 0;
316.
317.       // Comprueba que el módulo SIM900 está arrancado
318.       if (enviarAT("AT", "OK", 2000) == 0)
319.       {
320.         Serial.println("Encendiendo el GPRS...");
321.
322.         pinMode(9, OUTPUT);
323.         digitalWrite(9, HIGH);
324.         delay(1000);
325.         digitalWrite(9, LOW);
326.         delay(1000);
327.
328.         // Espera la respuesta del modulo SIM900
329.         while (respuesta == 0) {
330.           // Envia un comando AT cada 2 segundos y espera la respuesta
331.           respuesta = enviarAT("AT", "OK", 2000);
332.           SIM900.println(respuesta);
333.         }
334.       }
335.     }
336.
337.     void iniciar()
338.     {
339.       enviarAT("AT+CPIN=\"7401\"", "OK", 1000);
340.       Serial.println("Conectando a la red...");
341.       delay (5000);
342.
343.       //espera hasta estar conectado a la red movil
344.       while ( enviarAT("AT+CREG?", "+CREG: 0,1", 1000) == 0 )
345.       {
346.       }
347.       Serial.println("Conectado a la red.");
```

```
348.       enviarAT("AT+CLIP=1\r", "OK", 1000); // Activamos la identificación de llamadas
349.       enviarAT("AT+CMGF=1\r", "OK", 1000); //Configura el modo texto para enviar o recibir mensajes
350.       enviarAT("AT+CNMI=2,2,0,0,0\r", "OK", 1000); //Configuramos el módulo para que nos muestre los SMS
   recibidos por comunicación serie
351.       Serial.println("Preparado.");
352.     }
```

Sensor 1

```
1. //Librerías
2. #include <RF24.h>
3. #include <RF24Network.h>
4. #include <SPI.h>
5.
6. //Pin del sensor
7. #define pinSensor A0
8.
9. unsigned long humedad;
10.      unsigned long giveMe = 0 ;
11.
12.      //Instancias de las librerías
13.      RF24 radio(9,10);
14.      RF24Network network(radio);
15.
16.      //Dirección de los nodos
17.      const uint16_t this_node = 01;
18.      const uint16_t master00 = 00;
19.
20.      //Objeto de lectura de la librería RF24Network
21.      RF24NetworkHeader header;
22.
23.      //Objeto de escritura de la librería RF24Network
24.      RF24NetworkHeader header00(master00);
25.
26.      void setup() {
27.        SPI.begin();
28.        radio.begin();
29.        network.begin(90, this_node);
30.        radio.setDataRate(RF24_2MBPS);
```

```
31.       }
32.
33.       void loop() {
34.
35.         network.update();
36.
37.         while (network.available()){
38.           network.read(header, &giveMe, sizeof(giveMe));
39.         }
40.
41.         if (giveMe == 1){
42.
43.           humedad = analogRead(pinSensor);
44.           humedad = map(humedad,0,1023,100,0);
45.
46.           network.write(header00, &humedad, sizeof(humedad));
47.           delay(500);
48.         }
49.       }
```

Sensor 3

```
1. //Librerías
2. #include <RF24.h>
3. #include <RF24Network.h>
4. #include <SPI.h>
5.
6. //Pin del sensor
7. #define pinSensor A0
8.
9. unsigned long humedad;
10.       unsigned long giveMe = 0 ;
11.
12.       //Instancias de las librerías
```

```
13.      RF24 radio(9,10);
14.      RF24Network network(radio);
15.
16.      //Dirección de los nodos
17.      const uint16_t this_node = 03;
18.      const uint16_t master00 = 00;
19.
20.      //Objeto de lectura de la librería RF24Network
21.      RF24NetworkHeader header;
22.
23.      //Objeto de escritura de la librería RF24Network
24.      RF24NetworkHeader header00(master00);
25.
26.      void setup() {
27.        SPI.begin();
28.        radio.begin();
29.        network.begin(90, this_node);
30.        radio.setDataRate(RF24_2MBPS);
31.      }
32.
33.      void loop() {
34.
35.        network.update();
36.
37.        while (network.available()){
38.          network.read(header, &giveMe, sizeof(giveMe));
39.        }
40.
41.        if (giveMe == 1){
42.
43.          humedad = analogRead(pinSensor);
44.          humedad = map(humedad,0,1023,100,0);
45.
46.          network.write(header00, &humedad, sizeof(humedad));
47.          delay(500);
48.        }
49.      }
```

Válvula 1

```
1. //Librerías
2. #include <RF24.h>
3. #include <RF24Network.h>
4. #include <SPI.h>
5.
6. //Pin de la válvula
7. #define pinValvula 3
8.
9. unsigned long estadoV ;
10.
11.      //Instancias de las librerías
12.      RF24 radio(9,10);
13.      RF24Network network(radio);
14.      const uint16_t this_node = 012;
15.      const uint16_t master00 = 00;
16.
17.      //Objeto de lectura de la librería RF24Network
18.      RF24NetworkHeader header;
19.
20.      void setup() {
21.        SPI.begin();
22.        radio.begin();
23.        network.begin(90, this_node);
24.        radio.setDataRate(RF24_2MBPS);
25.        pinMode(pinValvula, OUTPUT);
26.      }
27.
28.      void loop() {
29.        network.update();
30.
31.        while (network.available()){
32.
33.          network.read(header, &estadoV, sizeof(estadoV));
34.          if (estadoV == 1){
35.            digitalWrite(pinValvula, HIGH);
```

```
          } else {
            digitalWrite(pinValvula, LOW);
          }
        }
      }
```

Válvula 2

```
//Librerías
#include <RF24.h>
#include <RF24Network.h>
#include <SPI.h>

//Pin de la válvula
#define pinValvula 3

unsigned long estadoV ;

      //Instancias de las librerías
      RF24 radio(9,10);
      RF24Network network(radio);
      const uint16_t this_node = 02;
      const uint16_t master00 = 00;

      //Objeto de lectura de la librería RF24Network
      RF24NetworkHeader header;

      void setup() {
        SPI.begin();
        radio.begin();
        network.begin(90, this_node);
        radio.setDataRate(RF24_2MBPS);
        pinMode(pinValvula, OUTPUT);
      }

      void loop() {
```

```
        network.update();

        while (network.available()){

          network.read(header, &estadoV, sizeof(estadoV));
          if (estadoV == 1){
            digitalWrite(pinValvula, HIGH);
          } else {
            digitalWrite(pinValvula, LOW);
          }
        }
      }
```

Válvula 3

```
//Librerías
#include <RF24.h>
#include <RF24Network.h>
#include <SPI.h>

//Pin de la válvula
#define pinValvula 3

unsigned long estadoV ;

      //Instancias de las librerías
      RF24 radio(9,10);
      RF24Network network(radio);
      const uint16_t this_node = 022;
      const uint16_t master00 = 00;

      //Objeto de lectura de la librería RF24Network
      RF24NetworkHeader header;

      void setup() {
        SPI.begin();
```

```
22.         radio.begin();
23.         network.begin(90, this_node);
24.         radio.setDataRate(RF24_2MBPS);
25.         pinMode(pinValvula, OUTPUT);
26.       }
27.
28.       void loop() {
29.         network.update();
30.
31.         while (network.available()){
32.
33.           network.read(header, &estadoV, sizeof(estadoV));
34.           if (estadoV == 1){
35.             digitalWrite(pinValvula, HIGH);
36.           } else {
37.             digitalWrite(pinValvula, LOW);
38.           }
39.         }
40.       }
```

Anexo II. Código de la aplicación

MainActivity

Archivo XML

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent"
7.     tools:context=".MainActivity">
8.
9.      <Button
10.             android:id="@+id/button_automatico"
```

```
11.             android:layout_width="181dp"
12.             android:layout_height="49dp"
13.             android:layout_marginStart="112dp"
14.             android:layout_marginLeft="112dp"
15.             android:layout_marginTop="96dp"
16.             android:onClick="Automatico"
17.             android:text="@string/txt_automatico"
18.             android:textSize="24sp"
19.             app:layout_constraintStart_toStartOf="parent"
20.             app:layout_constraintTop_toBottomOf="@+id/button_manual" />
21.
22.         <Button
23.             android:id="@+id/button_manual"
24.             android:layout_width="180dp"
25.             android:layout_height="48dp"
26.             android:layout_marginStart="116dp"
27.             android:layout_marginLeft="116dp"
28.             android:layout_marginTop="88dp"
29.             android:onClick="Manual"
30.             android:text="@string/txt_manual"
31.             android:textSize="24sp"
32.             app:layout_constraintStart_toStartOf="parent"
33.             app:layout_constraintTop_toBottomOf="@+id/tv1" />
34.
35.         <TextView
36.             android:id="@+id/tv1"
37.             android:layout_width="wrap_content"
38.             android:layout_height="wrap_content"
39.             android:layout_marginStart="96dp"
40.             android:layout_marginLeft="96dp"
41.             android:layout_marginTop="84dp"
42.             android:text="@string/txt_tv1"
43.             android:textSize="24sp"
44.             app:layout_constraintStart_toStartOf="parent"
45.             app:layout_constraintTop_toTopOf="parent" />
46.
47.     </androidx.constraintlayout.widget.ConstraintLayout>
```

Archivo Java

```
1. package com.example.riego;
2.
3. import androidx.appcompat.app.AppCompatActivity;
4.
5. import android.content.Intent;
6. import android.os.Bundle;
7. import android.view.View;
8.
9. public class MainActivity extends AppCompatActivity {
10.
11.          @Override
12.          protected void onCreate(Bundle savedInstanceState) {
13.              super.onCreate(savedInstanceState);
14.              setContentView(R.layout.activity_main);
15.          }
16.
17.          //Método para el botón MANUAL
18.          public void Manual(View view){
19.              Intent manual = new Intent(this, MANUAL.class);
20.              startActivity(manual);
21.          }
22.
23.          //Método para el botón AUTOMÁTICO
24.          public void Automatico(View view){
25.              Intent automatico = new Intent(this, AUTOMATICO.class);
26.              startActivity(automatico);
27.          }
28.      }
```

AUTOMATICO

Archivo XML

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".AUTOMATICO">

     <TextView
              android:id="@+id/textView2"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_marginStart="56dp"
              android:layout_marginLeft="56dp"
              android:layout_marginTop="20dp"
              android:text="@string/txt_automatico_sector1"
              app:layout_constraintStart_toStartOf="parent"
              app:layout_constraintTop_toBottomOf="@+id/textView6" />

          <EditText
              android:id="@+id/txt_num1"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_marginStart="56dp"
              android:layout_marginLeft="56dp"
              android:layout_marginTop="12dp"
              android:ems="10"
              android:hint="@string/txt_automatico_rango"
              android:inputType="number"
              app:layout_constraintStart_toStartOf="parent"
              app:layout_constraintTop_toBottomOf="@+id/textView2" />
```

```
32.
33.          <TextView
34.             android:id="@+id/textView3"
35.             android:layout_width="wrap_content"
36.             android:layout_height="wrap_content"
37.             android:layout_marginStart="56dp"
38.             android:layout_marginLeft="56dp"
39.             android:layout_marginTop="44dp"
40.             android:text="@string/txt_automatico_sector2"
41.             app:layout_constraintStart_toStartOf="parent"
42.             app:layout_constraintTop_toBottomOf="@+id/txt_num1" />
43.
44.          <EditText
45.             android:id="@+id/txt_num2"
46.             android:layout_width="wrap_content"
47.             android:layout_height="wrap_content"
48.             android:layout_marginStart="56dp"
49.             android:layout_marginLeft="56dp"
50.             android:layout_marginTop="16dp"
51.             android:ems="10"
52.             android:hint="@string/txt_automatico_rango"
53.             android:inputType="number"
54.             app:layout_constraintStart_toStartOf="parent"
55.             app:layout_constraintTop_toBottomOf="@+id/textView3" />
56.
57.          <TextView
58.             android:id="@+id/textView4"
59.             android:layout_width="wrap_content"
60.             android:layout_height="wrap_content"
61.             android:layout_marginStart="56dp"
62.             android:layout_marginLeft="56dp"
63.             android:layout_marginTop="44dp"
64.             android:text="@string/txt_automatico_sector3"
65.             app:layout_constraintStart_toStartOf="parent"
66.             app:layout_constraintTop_toBottomOf="@+id/txt_num2" />
67.
68.          <EditText
69.             android:id="@+id/txt_num3"
70.             android:layout_width="wrap_content"
71.             android:layout_height="wrap_content"
```

```
72.             android:layout_marginStart="56dp"
73.             android:layout_marginLeft="56dp"
74.             android:layout_marginTop="12dp"
75.             android:ems="10"
76.             android:hint="@string/txt_automatico_rango"
77.             android:inputType="number"
78.             app:layout_constraintStart_toStartOf="parent"
79.             app:layout_constraintTop_toBottomOf="@+id/textView4" />
80.
81.         <TextView
82.             android:id="@+id/textView5"
83.             android:layout_width="334dp"
84.             android:layout_height="37dp"
85.             android:layout_marginStart="36dp"
86.             android:layout_marginLeft="36dp"
87.             android:layout_marginTop="48dp"
88.             android:text="@string/txt_automatico_consejo"
89.             app:layout_constraintStart_toStartOf="parent"
90.             app:layout_constraintTop_toBottomOf="@+id/button4" />
91.
92.         <TextView
93.             android:id="@+id/textView6"
94.             android:layout_width="wrap_content"
95.             android:layout_height="wrap_content"
96.             android:layout_marginStart="56dp"
97.             android:layout_marginLeft="56dp"
98.             android:layout_marginTop="32dp"
99.             android:text="@string/txt_automatico_titulo"
100.            android:textSize="24sp"
101.            app:layout_constraintStart_toStartOf="parent"
102.            app:layout_constraintTop_toTopOf="parent" />
103.
104.        <Button
105.            android:id="@+id/button4"
106.            android:layout_width="wrap_content"
107.            android:layout_height="wrap_content"
108.            android:layout_marginStart="160dp"
109.            android:layout_marginLeft="160dp"
110.            android:layout_marginTop="56dp"
111.            android:onClick="Enviar"
```

```
            android:text="@string/enviar"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/txt_num3" />
    </androidx.constraintlayout.widget.ConstraintLayout>
```

Archivo Java

```
package com.example.riego;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

    public class AUTOMATICO extends AppCompatActivity {

        private EditText et1, et2, et3;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_automatico);

            et1 = findViewById(R.id.txt_num1);
            et2 = findViewById(R.id.txt_num2);
            et3 = findViewById(R.id.txt_num3);
        }

        //Método del boton ENVIAR
        public void Enviar(View view){

            String valor1 = et1.getText().toString();
            String valor2 = et2.getText().toString();
            String valor3 = et3.getText().toString();

```

```
32.             int num1 = Integer.parseInt(valor1);
33.             int num2 = Integer.parseInt(valor2);
34.             int num3 = Integer.parseInt(valor3);
35.
36.             if (num1>100 || num2>100 || num3>100 || num1<0 || num2<0 ||num3<0) {
37.                 Toast.makeText(this, "ATENCIÓN: Los porcentajes deben estar entre 0 y 100",
   Toast.LENGTH_SHORT).show();
38.             }else{
39.
40.                 if (num1!=100 && num1 >= 10){
41.                     valor1 = "0" + valor1;
42.                 }
43.                 if (num2!=100 && num2 >= 10){
44.                     valor2 = "0" + valor2;
45.                 }
46.                 if (num3!=100 && num2 >= 10){
47.                     valor3 = "0" + valor3;
48.                 }
49.                 if (num1 < 10){
50.                     valor1 = "00" + valor1;
51.                 }
52.                 if (num2 < 10){
53.                     valor2 = "00" + valor2;
54.                 }
55.                 if (num3 < 10){
56.                     valor3 = "00" + valor3;
57.                 }
58.
59.                 String mensaje = "u2" + valor1 + valor2 + valor3;
60.
61.                 String phone = "XXXXXXXXX";
62.
63.                 SmsManager sms = SmsManager.getDefault();
64.                 sms.sendTextMessage(phone, null, mensaje , null, null);
65.
66.                 Toast.makeText(this, "Orden enviada correctamente", Toast.LENGTH_LONG).show();
67.             }
68.         }
69.     }
```

MANUAL

Archivo XML

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MANUAL">

     <Button
              android:id="@+id/button"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_marginStart="136dp"
              android:layout_marginLeft="136dp"
              android:layout_marginTop="100dp"
              android:onClick="SECTOR2"
              android:text="@string/txt_Sector2"
              android:textSize="24sp"
              app:layout_constraintStart_toStartOf="parent"
              app:layout_constraintTop_toBottomOf="@+id/button3" />

          <Button
              android:id="@+id/button3"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_marginStart="136dp"
              android:layout_marginLeft="136dp"
              android:layout_marginTop="84dp"
              android:onClick="SECTOR1"
              android:text="@string/txt_Sector1"
              android:textSize="24sp"
```

```
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/textView" />

        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="136dp"
            android:layout_marginLeft="136dp"
            android:layout_marginTop="100dp"
            android:onClick="SECTOR3"
            android:text="@string/txt_Sector3"
            android:textSize="24sp"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/button" />

        <TextView
            android:id="@+id/textView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="68dp"
            android:layout_marginLeft="68dp"
            android:layout_marginTop="72dp"
            android:text="@string/tv_activity_manual"
            android:textSize="24sp"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
    </androidx.constraintlayout.widget.ConstraintLayout>
```

Archivo Java

```
package com.example.riego;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
```

```
6. import android.os.Bundle;
7. import android.view.View;
8. import android.widget.Toast;
9.
10.      public class MANUAL extends AppCompatActivity {
11.
12.          @Override
13.          protected void onCreate(Bundle savedInstanceState) {
14.              super.onCreate(savedInstanceState);
15.              setContentView(R.layout.activity_manual);
16.          }
17.
18.          //Método botón SECTOR 1
19.          public void SECTOR1(View view){
20.              Intent sector1 = new Intent(this, SECTOR1.class);
21.              startActivity(sector1);
22.
23.          }
24.
25.          //Método botón SECTOR 2
26.          public void SECTOR2(View view){
27.              Intent sector2 = new Intent(this, SECTOR2.class);
28.              startActivity(sector2);
29.          }
30.
31.          //Método botón SECTOR 3
32.          public void SECTOR3(View view){
33.              Intent sector3 = new Intent(this, SECTOR3.class);
34.              startActivity(sector3);
35.          }
36.      }
```

SECTOR1

Archivo XML

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent"
7.     tools:context=".SECTOR1">
8.
9.      <TextView
10.             android:id="@+id/textView7"
11.             android:layout_width="wrap_content"
12.             android:layout_height="wrap_content"
13.             android:layout_marginStart="52dp"
14.             android:layout_marginLeft="52dp"
15.             android:layout_marginTop="152dp"
16.             android:text="@string/txt_sector_tiempo"
17.             android:textSize="24sp"
18.             app:layout_constraintStart_toStartOf="parent"
19.             app:layout_constraintTop_toTopOf="parent" />
20.
21.         <EditText
22.             android:id="@+id/txt_tiempo"
23.             android:layout_width="wrap_content"
24.             android:layout_height="wrap_content"
25.             android:layout_marginStart="52dp"
26.             android:layout_marginLeft="52dp"
27.             android:layout_marginTop="20dp"
28.             android:ems="10"
29.             android:hint="@string/txt_sector_hint"
30.             android:inputType="number"
31.             android:textSize="24sp"
32.             app:layout_constraintStart_toStartOf="parent"
33.             app:layout_constraintTop_toBottomOf="@+id/textView7" />
34.
35.         <Button
```

```
            android:id="@+id/button5"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="52dp"
            android:layout_marginLeft="52dp"
            android:layout_marginTop="40dp"
            android:onClick="Enviar"
            android:text="@string/enviar"
            android:textSize="24sp"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/txt_tiempo" />
    </androidx.constraintlayout.widget.ConstraintLayout>
```

Archivo Java

```
package com.example.riego;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

    public class SECTOR1 extends AppCompatActivity {

        EditText tv1;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_sector1);

```

```
20.             tv1 = findViewById(R.id.txt_tiempo);
21.         }
22.
23.         //Método para el botón ENVIAR
24.         public void Enviar(View view){
25.
26.             String horas_string = tv1.getText().toString();
27.             int horas_int = Integer.parseInt(horas_string);
28.
29.             if (horas_int<0){
30.                 Toast.makeText(this,"El número de horas no puede ser inferior a 0",
   Toast.LENGTH_SHORT).show();
31.             }else if(horas_int>12) {
32.                 Toast.makeText(this, "No puede regar más de 12 h seguidas", Toast.LENGTH_SHORT).show();
33.             }else if(horas_int<10){
34.                 horas_string = "0" + horas_string;
35.             }else{
36.                 String mensaje = "u11" + horas_string;
37.                 String phone = "XXXXXXXXX";
38.
39.                 SmsManager sms = SmsManager.getDefault();
40.                 sms.sendTextMessage(phone, null, mensaje , null, null);
41.
42.                 Toast.makeText(this, "Orden enviada correctamente", Toast.LENGTH_LONG).show();
43.
44.             }
45.         }
46.     }
```

SECTOR2

Archivo XML

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent"
7.     tools:context=".SECTOR2">
8.
9.      <TextView
10.             android:id="@+id/textView7"
11.             android:layout_width="wrap_content"
12.             android:layout_height="wrap_content"
13.             android:layout_marginStart="52dp"
14.             android:layout_marginLeft="52dp"
15.             android:layout_marginTop="152dp"
16.             android:text="@string/txt_sector_tiempo"
17.             android:textSize="24sp"
18.             app:layout_constraintStart_toStartOf="parent"
19.             app:layout_constraintTop_toTopOf="parent" />
20.
21.         <EditText
22.             android:id="@+id/txt_tiempo"
23.             android:layout_width="wrap_content"
24.             android:layout_height="wrap_content"
25.             android:layout_marginStart="52dp"
26.             android:layout_marginLeft="52dp"
27.             android:layout_marginTop="20dp"
28.             android:ems="10"
29.             android:hint="@string/txt_sector_hint"
30.             android:inputType="number"
31.             android:textSize="24sp"
32.             app:layout_constraintStart_toStartOf="parent"
33.             app:layout_constraintTop_toBottomOf="@+id/textView7" />
34.
35.         <Button
36.             android:id="@+id/button5"
37.             android:layout_width="wrap_content"
38.             android:layout_height="wrap_content"
39.             android:layout_marginStart="52dp"
40.             android:layout_marginLeft="52dp"
41.             android:layout_marginTop="40dp"
42.             android:onClick="Enviar"
```

```
            android:text="@string/enviar"
            android:textSize="24sp"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/txt_tiempo" />

    </androidx.constraintlayout.widget.ConstraintLayout>
```

Archivo Java

```
package com.example.riego;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

    public class SECTOR2 extends AppCompatActivity {

        EditText tv1;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_sector2);

            tv1 = findViewById(R.id.txt_tiempo);

        }

        //Método para el botón ENVIAR
        public void Enviar(View view){

            String horas_string = tv1.getText().toString();
```

```
28.              int horas_int = Integer.parseInt(horas_string);
29.
30.              if (horas_int<0){
31.                  Toast.makeText(this,"El número de horas no puede ser inferior a 0",
   Toast.LENGTH_SHORT).show();
32.              }else if(horas_int>12){
33.                  Toast.makeText(this,"No se puede regar más de 12 h en el modo manual",
   Toast.LENGTH_SHORT).show();
34.              }else if(horas_int<10){
35.                  horas_string = "0" + horas_string;
36.              }else{
37.                  String mensaje = "u12" + horas_string;
38.                  String phone = "XXXXXXXXX";
39.
40.                  SmsManager sms = SmsManager.getDefault();
41.                  sms.sendTextMessage(phone, null, mensaje , null, null);
42.
43.                  Toast.makeText(this, "Orden enviada correctamente", Toast.LENGTH_LONG).show();
44.
45.              }
46.          }
47.      }
```

SECTOR3

Archivo XML

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent"
7.     tools:context=".SECTOR3">
8.
```

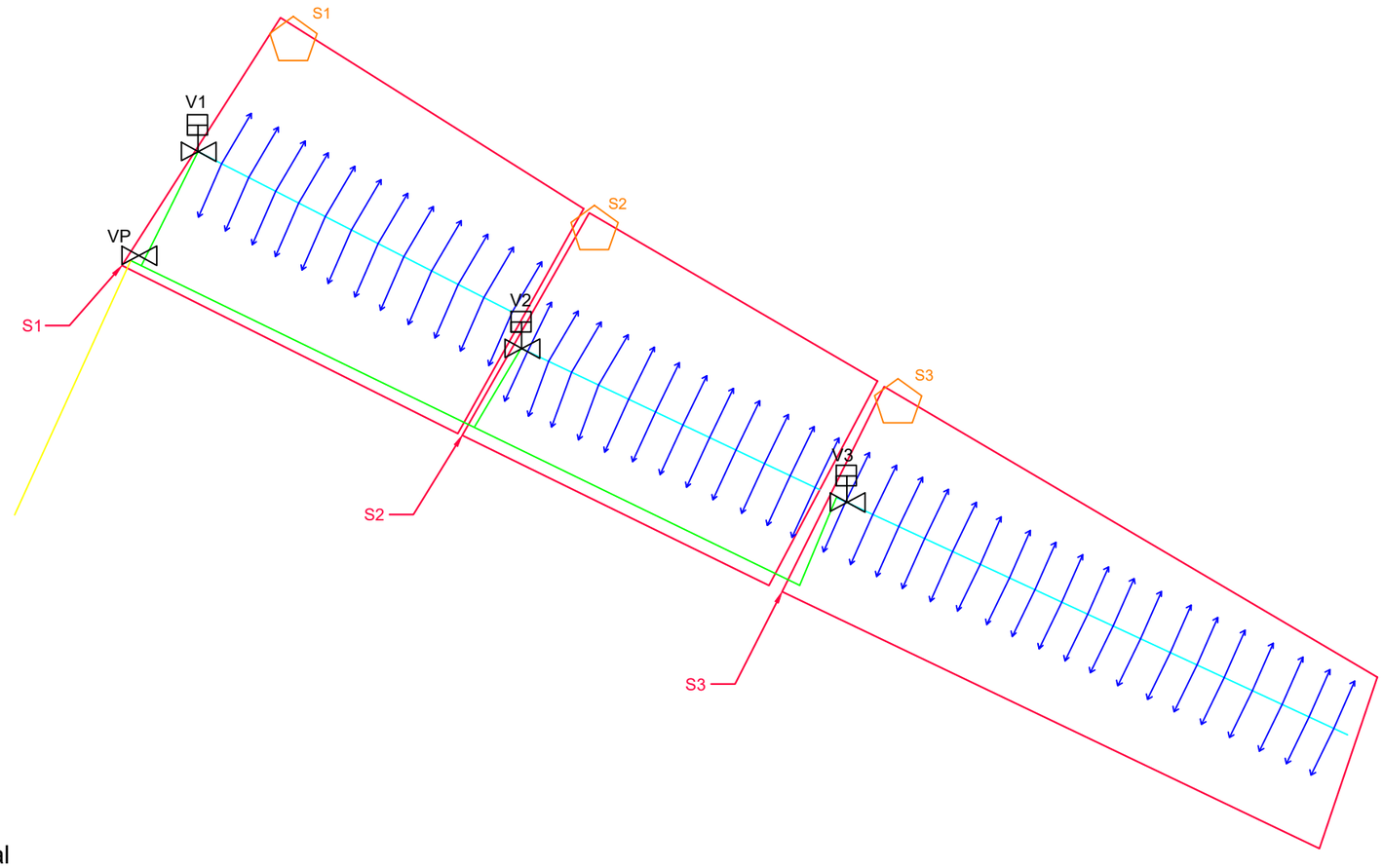
```
9.      <TextView
10.             android:id="@+id/textView7"
11.             android:layout_width="wrap_content"
12.             android:layout_height="wrap_content"
13.             android:layout_marginStart="52dp"
14.             android:layout_marginLeft="52dp"
15.             android:layout_marginTop="152dp"
16.             android:text="@string/txt_sector_tiempo"
17.             android:textSize="24sp"
18.             app:layout_constraintStart_toStartOf="parent"
19.             app:layout_constraintTop_toTopOf="parent" />
20.
21.         <EditText
22.             android:id="@+id/txt_tiempo"
23.             android:layout_width="wrap_content"
24.             android:layout_height="wrap_content"
25.             android:layout_marginStart="52dp"
26.             android:layout_marginLeft="52dp"
27.             android:layout_marginTop="20dp"
28.             android:ems="10"
29.             android:hint="@string/txt_sector_hint"
30.             android:inputType="number"
31.             android:textSize="24sp"
32.             app:layout_constraintStart_toStartOf="parent"
33.             app:layout_constraintTop_toBottomOf="@+id/textView7" />
34.
35.         <Button
36.             android:id="@+id/button5"
37.             android:layout_width="wrap_content"
38.             android:layout_height="wrap_content"
39.             android:layout_marginStart="52dp"
40.             android:layout_marginLeft="52dp"
41.             android:layout_marginTop="40dp"
42.             android:onClick="Enviar"
43.             android:text="@string/enviar"
44.             android:textSize="24sp"
45.             app:layout_constraintStart_toStartOf="parent"
46.             app:layout_constraintTop_toBottomOf="@+id/txt_tiempo" />
47.
48.       </androidx.constraintlayout.widget.ConstraintLayout>
```

Archivo Java

```
1. package com.example.riego;
2.
3. import androidx.appcompat.app.AppCompatActivity;
4.
5. import android.os.Bundle;
6. import android.telephony.SmsManager;
7. import android.view.View;
8. import android.widget.EditText;
9. import android.widget.Toast;
10.
11.      public class SECTOR3 extends AppCompatActivity {
12.
13.          EditText tv1;
14.
15.          @Override
16.          protected void onCreate(Bundle savedInstanceState) {
17.              super.onCreate(savedInstanceState);
18.              setContentView(R.layout.activity_sector3);
19.
20.              tv1 = findViewById(R.id.txt_tiempo);
21.          }
22.
23.          public void Enviar(View view){
24.
25.              String horas_string = tv1.getText().toString();
26.              int horas_int = Integer.parseInt(horas_string);
27.
28.              if (horas_int<0){
29.                  Toast.makeText(this,"El número de horas no puede ser inferior a 0",
   Toast.LENGTH_SHORT).show();
30.              }else if(horas_int>12){
31.                  Toast.makeText(this,"No se puede regar más de 12 h en el modo manual",
   Toast.LENGTH_SHORT).show();
32.              }else if(horas_int<10){
33.                  horas_string = "0" + horas_string;
```

```
34.                 }else{
35.                     String mensaje = "u13" + horas_string;
36.                     String phone = "XXXXXXXXX";
37.
38.                     SmsManager sms = SmsManager.getDefault();
39.                     sms.sendTextMessage(phone, null, mensaje , null, null);
40.
41.                     Toast.makeText(this, "Orden enviada correctamente", Toast.LENGTH_LONG).show();
42.
43.                 }
44.             }
45.         }
```

PLANOS



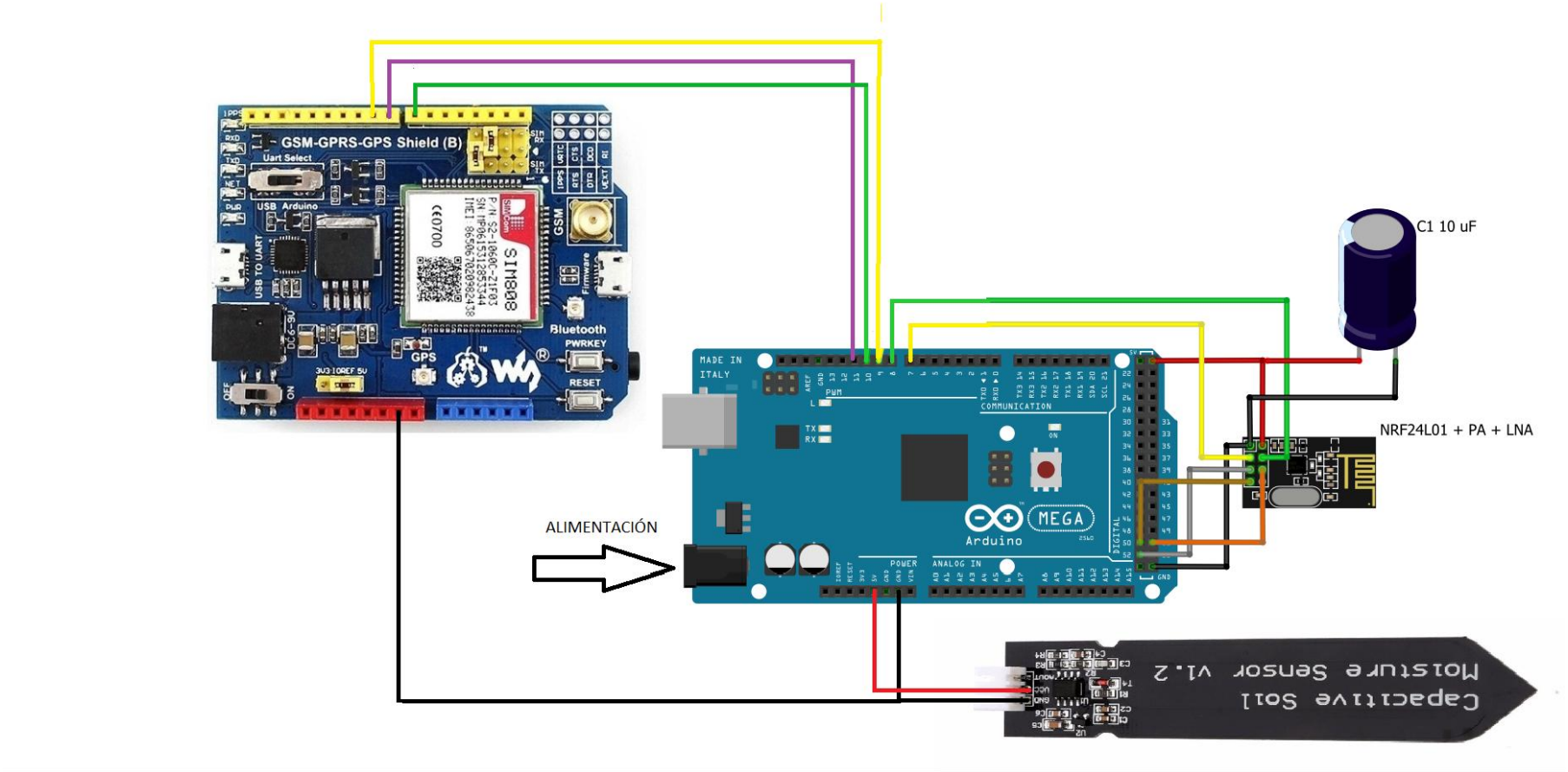
Leyenda:

- Tubería origen
- Tubería principal
- Tubería lateral
- Fila de goteros
- Válvula manual
- Electroválvula
- Sensor

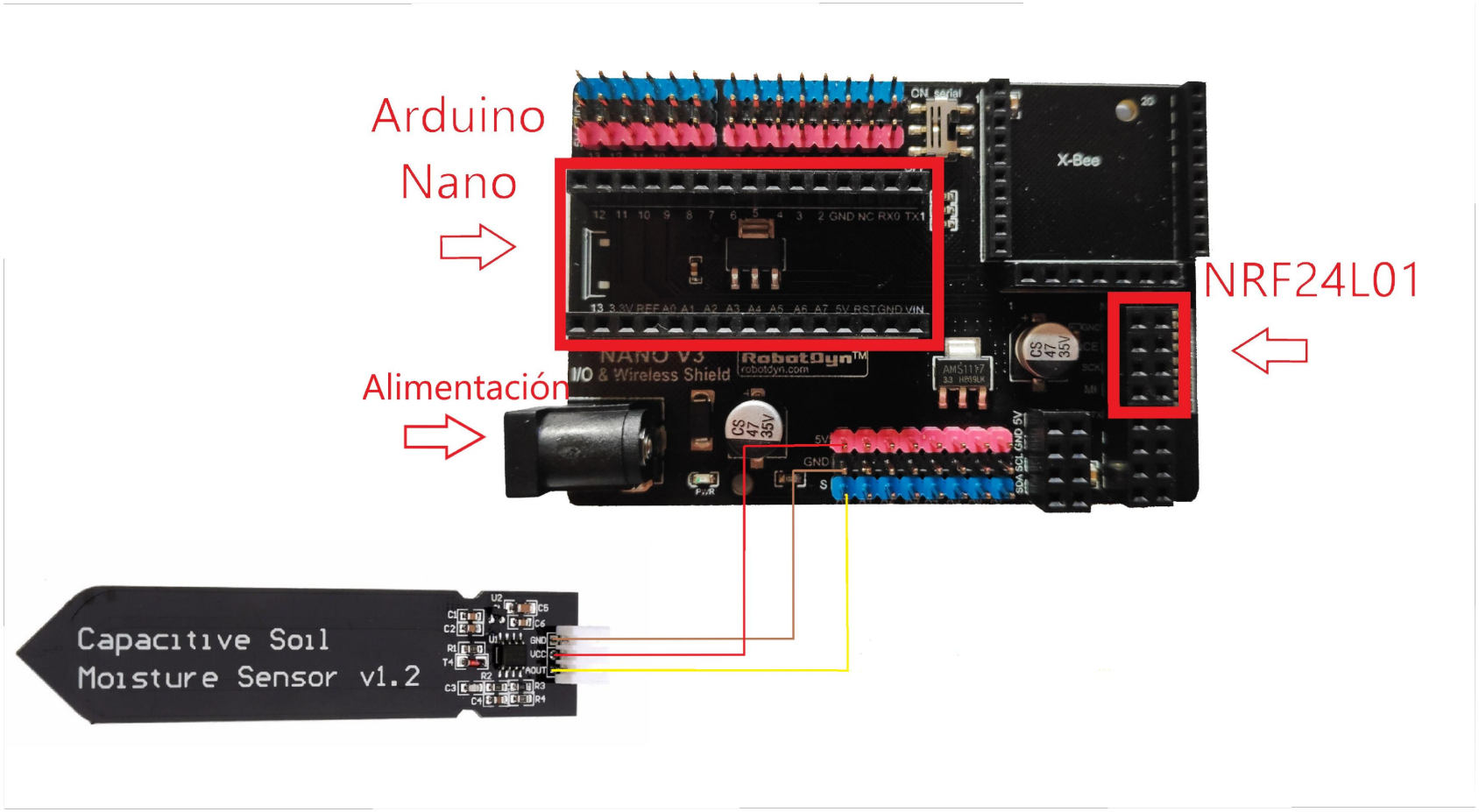
UJA		Fecha:	Nombre:
	Dibujado:	Septiembre, 2019	M. Ángel Moya Garrido
	Comprobado:		
Escala: 1:2000	DISEÑO DE UN SISTEMA DE RIEGO PARA EL OLIVAR BASADO EN MICROCONTROLADOR		

PLANTA

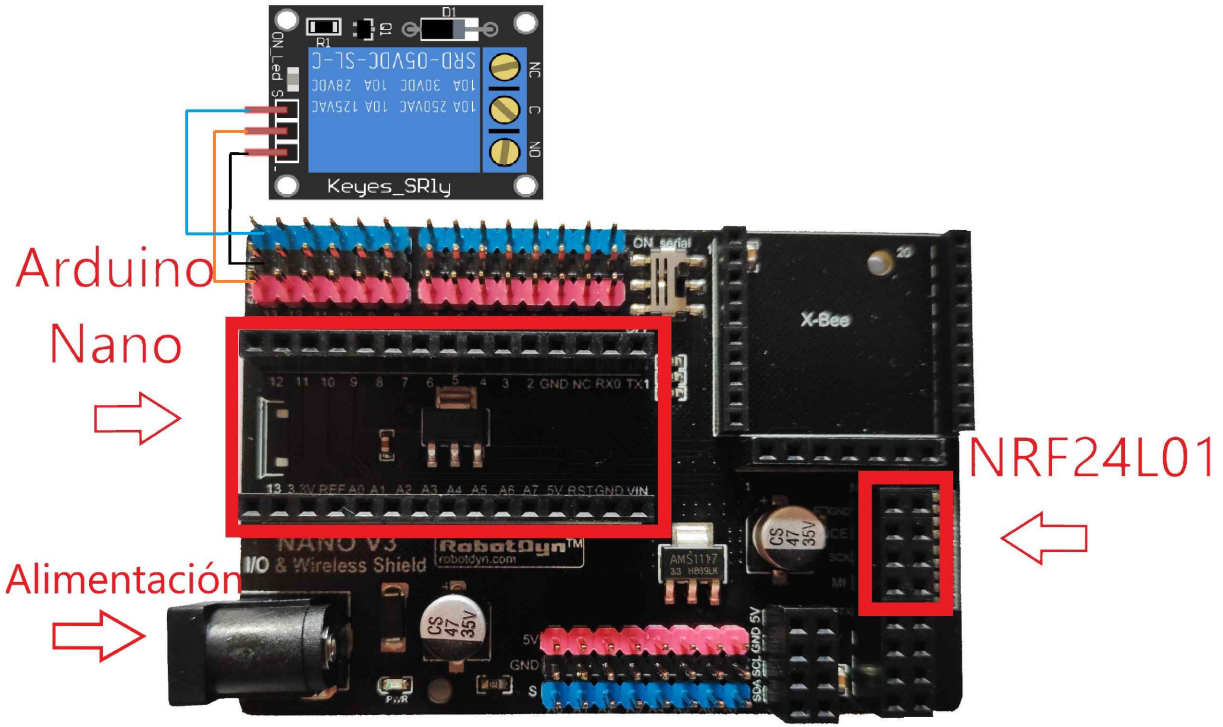
PLANO 1



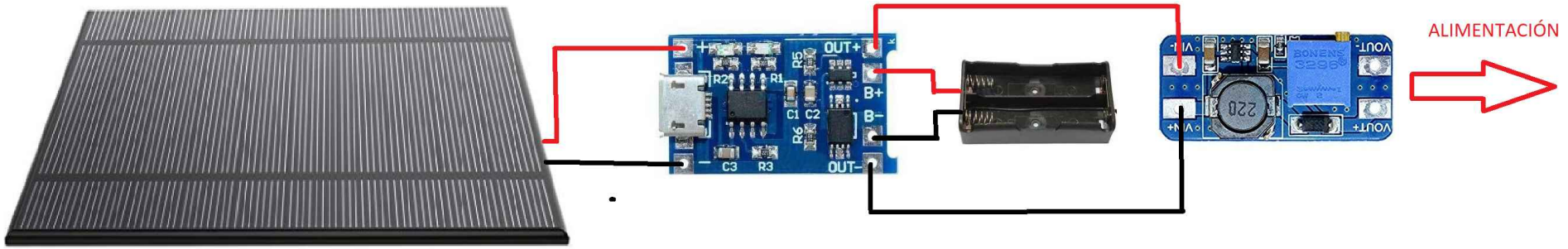
UJA		Fecha:	Nombre:	Conexiones unidad principal
	Dibujado:	Septiembre, 2019	M. Ángel Moya Garrido	
	Comprobado:			
Escala: S/E	DISEÑO DE UN SISTEMA DE RIEGO PARA EL OLIVAR BASADO EN MICROCONTROLADOR			PLANO 2



UJA		Fecha:	Nombre:	Conexiones unidad sensor
	Dibujado:	Septiembre, 2019	M. Ángel Moya Garrido	
	Comprobado:			
Escala: S/E	DISEÑO DE UN SISTEMA DE RIEGO PARA EL OLIVAR BASADO EN MICROCONTROLADOR			PLANO 3



UJA		Fecha:	Nombre:	Conexiones unidad válvula
	Dibujado:	Septiembre, 2019	M. Ángel Moya Garrido	
	Comprobado:			
Escala: S/E	DISEÑO DE UN SISTEMA DE RIEGO PARA EL OLIVAR BASADO EN MICROCONTROLADOR			PLANO 4



UJA		Fecha:	Nombre:	Conexiones alimentación
	Dibujado:	Septiembre, 2019	M. Ángel Moya Garrido	
	Comprobado:			
Escala: S/E	DISEÑO DE UN SISTEMA DE RIEGO PARA EL OLIVAR BASADO EN MICROCONTROLADOR			PLANO 5