industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

Universidad Politécnica de Cartagena

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Análisis de luz solar mediante diodos LED de bajo coste

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

Autor: **Francisco Franco Martínez**
Director: Manuel Jiménez Buendía
Codirector: José Alfonso Vera Repullo

Cartagena, Junio de 2017

Agradecimientos:

A José Alfonso Vera y Manuel Jiménez, por la oportunidad de participar en este proyecto.

A la empresa Industrias Peñalver S.L. y en especial a Evaristo Delgado, por la ayuda en el desarrollo el proyecto tanto en medios como en conocimientos y por la inspiración aportada.

A Martin Oates y a Antonio Ruiz Canales, por la oportunidad de poder trabajar con ellos a raíz del Trabajo Final de Grado.

Y en especial a mi familia que sin su apoyo, perseverancia y tesón no habría conseguido llegar a ser quien soy.

"Hay una fuerza motriz más poderosa que el vapor,

la electricidad y la energía atómica: la voluntad"

Albert Einstein.

Índice de contenidos

Índice de figuras

Índice de tablas

1

Introducción y objetivos

Vista general sobre las ideas y objetivos fundamentales del proyecto, además de las dos líneas principales del mismo.

1.1. Introducción

1.1.1 Luz solar

En el desarrollo de los cultivos, las condiciones de crecimiento pueden ser críticas para la futura cosecha. Estas condiciones van desde la calidad del suelo, humedad del suelo y temperatura. Pero cabe destacar otros factores como son la calidad y duración de luz solar disponible.

La cantidad o brillo de la luz que reciben los cultivos no es un buen indicador de la verdadera calidad de esa luz, ya que ha sido demostrado que la presencia o ausencia de ciertas longitudes de onda pueden afectar significativamente a la fotosíntesis y al crecimiento del cultivo. Longitudes de onda específicas, como la radiación ultravioleta, puede provocar daños en los tejidos vegetales por una sobreexposición a esta radiación. Por otro lado, la radiación infrarroja se dispersa fácilmente con niveles bajos de nubes y niebla y casi totalmente absorbida en altos niveles de nubosidad. La cantidad y calidad de luz solar recibida por la planta determinará la buena maduración de la misma y afectará a puntos como puede ser la cosecha o el tratamiento del cultivo a nivel de fertilizantes y pesticidas.

1.1.2 Humedad del suelo

Según avanzamos en el proyecto, se valoró la posibilidad de añadir una sonda de humedad al proyecto como elemento complementario y con el objetivo de ir desarrollándola en trabajos futuros. Conocer la humedad en los cultivos es fundamental para conocer la demanda de agua de la misma y nos permitirá saber cómo evolucionará el cultivo teniendo en cuenta las condiciones de luz solar.

Por tanto, un analizador de luz solar al que se puede añadir un sistema de control de la humedad es esencial para controlar el crecimiento del cultivo y la productividad de los mismos. Dentro de las tecnologías de bajo coste basadas en el cálculo de la humedad del suelo, existen las técnicas basadas en la Reflectometría en el Dominio de la Frecuencia (FDR) que han demostrado una mejor precisión a la hora de obtener los valores.

La técnica FDR nos permite conocer la humedad del suelo a través de la capacitancia eléctrica del suelo. Como el agua presenta una constante dieléctrica 80 veces mayor que el aire, la presencia de humedad en el suelo hace variar significativamente su capacitancia. La utilización de las técnicas basadas en FDR responden mejor en terrenos secos cuando se aplican altas frecuencias, pero al requerir esto equipos más costosos es importante buscar un equilibrio entre precio y precisión de los datos.

1.1.3 Visión global

El objetivo de este trabajo es el diseño, montaje y utilización de un sistema analizador de luz solar autónomo basado en diodos LEDs de bajo coste que será capaz de distinguir condiciones de luz específicas. Desde luz solar directa, pasando por niebla y nuboso hasta luna llena. Éste integrará un sistema complementario basado en las técnicas de la

Reflectometría en el Dominio de la Frecuencia, el cual nos aportará información de la humedad del suelo.

El dispositivo será capaz de recolectar valores de tipo de luz solar y humedad del suelo. Estará formado por un dispositivo emisor-receptor donde el emisor es diseñado para estar en un entorno donde no haya alimentación eléctrica y expuesto a condiciones medioambientales. Por otro lado, el receptor se comunica con el anterior vía radiofrecuencia 2.4 GHz con el que podremos recopilar toda la información y procesarla posteriormente con LabVIEW.

1.2. Objetivos del proyecto

1.2.1 Desarrollo de un dispositivo emisor para registro de datos.

Construcción y programación de un prototipo con tecnología Arduino para la identificación del estado de luz solar del lugar donde se sitúe. El dispositivo de medida está formado por una matriz de diodos LEDs con el que podemos recibir un juicio de ese estado meteorológico y poder automatizar procesos.

- Construcción con Arduino la matriz de LEDs empleando los datos, artículos y prácticas proporcionadas por el Dr. Martin Oates.
- Conexión con el dispositivo receptor.

1.2.2 Desarrollo de un dispositivo receptor para el procesamiento de datos

La información será transmitida a un dispositivo receptor (también tecnología Arduino) que, utilizando un módulo de Radiofrecuencia a 2.4 GHz, podemos procesar la información recibida de nuestro dispositivo emisor empleando un software de programación gráfica como LabVIEW.

- Construcción con Arduino el elemento receptor de la información vía radiofrecuencia para trasladarla al programa LabVIEW.
- Conexión con el dispositivo emisor.

1.2.3 Implementación de ambos dispositivos y toma de datos

Se enlazarán ambos dispositivos para que, mientras uno toma medidas de luz solar, el otro la recibe y procesa los datos.

- Comprobación y pruebas del dispositivo.
- Toma de medidas y análisis de datos.

2

Estado del Arte

Exposición de la ciencia y la teoría en la que se sustenta el trabajo desarrollado haciendo hincapié en el uso de LEDs como sensores y la técnica FDR.

2.1. Radiación Solar

2.1.1 Introducción

La radiación solar es el flujo de energía que recibimos del Sol como resultado del proceso de fusión nuclear que ocurre en él. Esta radiación se recibe en forma de ondas electromagnéticas de diferentes frecuencias (luz visible, infrarrojo y ultravioleta). La luz visible son las radiaciones comprendidas entre 0,4 μm y 0,7 μm pueden ser detectadas por el ojo humano. Existen radiaciones situadas en la parte infrarroja del espectro y en la parte ultravioleta.

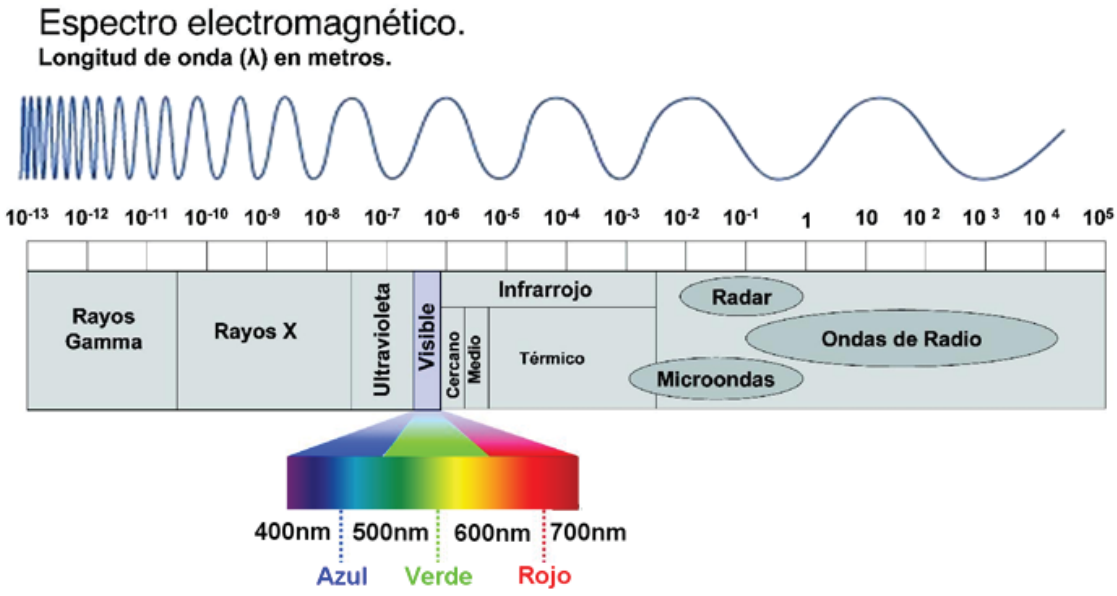


Figura 1. Espectro electromagnético.

La proporción aproximada de la radiación solar en las distintas regiones del espectro electromagnético es:

Radiación	Proporción
Ultravioleta	7 %
Luz visible	43 %
Infrarrojo	49 %
Resto	1 %

Tabla 1. Proporciones de ondas electromagnéticas en la radiación solar

2.1.2 Tipos de radiación e impacto en la Atmósfera

Los tipos de radiaciones que recibe la superficie de la Tierra son:

- Radiación directa: llega directamente del sol sin haber sufrido cambio alguno en su dirección. Se caracteriza por proyectar una sombra definida de los objetos opacos que la interceptan.
- Radiación Difusa: Parte de la radiación que atraviesa la atmósfera es reflejada por las nubes o absorbida por éstas. Esta radiación se denomina difusa, va en todas las direcciones, como consecuencia de las reflexiones y absorciones, no solo de las nubes sino de las partículas de polvo atmosférico, montañas, etc. Este tipo de radiación se caracteriza por no producir sombra alguna respecto a los objetos opacos interpuestos.
- Radiación Reflejada: Es aquella reflejada por la superficie terrestre. La cantidad de radiación depende del coeficiente de reflexión de la superficie. Las superficies verticales son las que más radiación reflejada reciben.
- Radiación global: Es la radiación total y la suma de las tres radiaciones anteriores.

La energía recibida del sol, al atravesar la atmósfera, provoca alteraciones en la densidad de los gases y desequilibrios que causan la circulación atmosférica. Esta energía produce la temperatura en la superficie terrestre y el efecto de la atmósfera es aumentarla por efecto invernadero y mitigar la diferencia de temperaturas entre el día y la noche y entre el polo y el ecuador.

La mayor parte de la energía utilizada por los seres vivos procede del sol, las plantas la absorben directamente y realizan la fotosíntesis, los herbívoros absorben indirectamente una pequeña cantidad de esta energía comiendo plantas, y los carnívoros absorben indirectamente una cantidad más pequeña comiendo a los herbívoros. La mayoría de las fuentes de energía usadas por el hombre derivan indirectamente del sol (Nave, 2012).

La fotosíntesis se caracteriza por la siguiente reacción química:

$$6\,CO_2 + 6\,H_2O + Radiación\ Solar \rightarrow C_6H_{12}O_6 + 6\,O_2$$

Principalmente son las clorofilas “a” y “b” las que absorben luz azul-violeta y roja sin absorber fotones con longitudes de onda entre los 500 y 600 nm. La clorofila a es el principal pigmento fotosintético y el único que puede actuar directamente para convertir la energía solar en energía química. Sin embargo, la clorofila b puede absorber fotones mientras que la anterior no. La clorofila b aumenta considerablemente la proporción de fotones en la luz del Sol que las plantas pueden utilizar.

Se demostró que existe un amplio rango de factores que afectan a la productividad de las plantas. Para algunos cultivos, regar durante condiciones de completa luz solar no es lo óptimo. Para otros, especialmente los que crecen en invernaderos de vidrio y en la maduración, puede ser beneficioso para garantizar niveles mínimos de luminosidad por medios artificiales cuando las condiciones de luz natural son insuficientes. A la inversa, puede ser beneficioso suministrar sombra artificial a algunos cultivos cuando las condiciones naturales de luz llegan a ser demasiado intensas o persisten demasiado tiempo. Para algunos cultivos, la presencia de luz de luna puede ser tanto beneficiosa como perjudicial debido al aumento de la probabilidad de polinizadores o de

depredadores de éstos. La detección de tales condiciones puede llevar a la aplicación de pesticidas de control más eficientes y sistemas de apoyo de polinización, bien químicos, auditivos o mecánicos (Oates, Ruiz-Canales, Molina-Martínez, & Vázquez de León, Resultados preliminares de una herramienta de bajo coste para análisis de la luz solar y gestión eficiente de recursos agrícolas, 2015).

Por tanto, las condiciones meteorológicas específicas y las condiciones temporales además de las exposiciones a distribuciones de espectros de luz y el reconocimiento de tales distribuciones pueden controlarse en tiempo real, así como el uso óptimo de recursos de gestión del cultivo tales como el riego, iluminación artificial y gestión del calor y la sombra.

2.2. Medición de la radiación solar

A grandes rasgos, la instrumentación que se emplea en la medición de la radiación solar es:

- Piranómetro: Instrumento que mide la irradiación solar global sobre la superficie de la Tierra.
- Pirheliómetro: Instrumento que mide la irradiación solar directa del Sol a la superficie de la Tierra.
- Heliómetro: Mide la intensidad lumínica solar.

2.2.1 Piranómetros

La radiación global incluye la recibida directamente del disco solar y también la radiación celeste difusa dispersada al atravesar la atmósfera. El instrumento necesario para medir la radiación global es el piranómetro.

La cúpula de cristal limita la respuesta al rango de 300 a 2800nm. preservando un campo de visión de 180 grados. Las bandas negras del sensor (termopila) absorben la radiación solar que se transforma en calor. Este calor fluye atravesando los sensores hacia el cuerpo del piranómetro, proporcionando una señal eléctrica proporcional a la radiación incidente (Pérez, s.f.).

Figura 2. Piranómetro.

Para medir una radiación solar, se requiere de la respuesta al flujo de radiación varíe con el coseno de ángulos de incidencia; por ejemplo, máxima respuesta cuando el flujo incide perpendicularmente sobre el sensor (0 grados), respuesta nula cuando el sol está en el horizonte (90 grados), o valores intermedios de respuesta, cuando el ángulo de incidencia está entre los anteriores.

Está construido con un adquiridor de datos y unas células solares fotovoltaica y una caja de bombones. Una vez conectado al software de la estación, da la radiación solar en porcentaje y en watios/m2, y sirve además para calcular la cobertura de nubes de forma automática. La tapa de la caja de bombones, transparente y perfecta en tamaño sirve para alojar y proteger de la intemperie a la célula solar (Pérez-Carrasco & Silva-Pérez).

Características	Valores medios
Tiempo de respuesta (95%)	< 30 s
Resolución (mínimo cambio detectable en $W \cdot m^{-2}$)	± 5
Estabilidad (porcentaje del fondo de escala, variación anual)	± 1.5
Respuesta direccional a la radiación directa (rango de errores debidos a asumir que la respuesta a la incidencia normal es válida para todas las direcciones cuando se mide, desde cualquier dirección, radiación directa normal de 1000 $W \cdot m^{-2}$)	± 20 $W \cdot m^{-2}$
Respuesta en temperatura (máximo error en % debido a la variación de temperatura ambiente en un intervalo de 50 K)	± 4
No-Linealidad (desviación en % de la respuesta a 500 W·m-2 debido a una variación de la irradiancia entre 100 y 1100 $W \cdot m^{-2}$)	± 1
Sensibilidad espectral (desviación en % del producto de la absortancia	± 5

espectral y la transmitancia espectral de la media correspondiente en el rango de 0.3 a 3 μm)	
Respuesta a la inclinación (desviación en % con respecto a la respuesta a inclinación de 0º debida a la variación de la inclinación desde 0 a 90º a 1000 W·m-2 de irradiancia)	± 2
Precio aproximado	600€

Tabla 2. Características medias de los piranómetros.

2.2.2 Pirheliómetro

La medida de la radiación solar directa total se realiza con el pirheliómetro, instrumento de tipo telescópico con una apertura de pequeño diámetro. Las superficies receptoras del pirheliómetro deben mantenerse en todo momento perpendiculares a la dirección de la radiación solar, por lo que el uso de un sistema de seguimiento adecuado (solar tracker) es ineludible (Pérez, s.f.).

Figura 3. Pirheliómetro.

Las aperturas de este dispositivo están dispuestas de forma que sólo la radiación procedente del disco solar y de una estrecha franja anular en torno al mismo alcanzan el receptor.

Los factores considerados para evaluar la precisión son: sensibilidad, estabilidad del factor de calibración, error máximo debido a las variaciones de temperatura ambiente, errores debidos a la respuesta espectral del receptor, no-linealidad de la respuesta, ángulo de apertura, constante de tiempo del sistema y efectos del equipo auxiliar.

Un pirheliómetro absoluto de cavidad está básicamente constituido por una cavidad (receptor) y sensores calorimétricos diferenciales autocalibrados eléctricamente. La radiación solar que atraviesa la apertura de precisión se absorbe en un receptor y se determina su valor mediante la sustitución de la radiación solar por el calor aportado por una corriente eléctrica, que se disipa en un bobinado calorífico muy próximo al lugar donde tiene lugar la absorción de la radiación (Pérez-Carrasco & Silva-Pérez).

Características	Valores medios
Tiempo de respuesta (95%)	< 30 s
Resolución (mínimo cambio detectable en $W \cdot m^{-2}$)	± 1
Estabilidad (porcentaje del fondo de escala, variación anual)	± 1
Respuesta en temperatura (máximo error en % debido a la variación de temperatura ambiente en un intervalo de 5 K)	± 2
No-Linealidad (desviación en % de la respuesta a 500 W·m-2 debido a una variación de la irradiancia entre 100 y 1100 $W \cdot m^{-2}$)	± 0.5
Sensibilidad espectral (desviación en % del producto de la absortancia espectral y la transmitancia espectral de la media correspondiente en el rango de 0.3 a 3 µm)	± 1
Respuesta a la inclinación (desviación en % con respecto a la respuesta a inclinación de 0º debida a la variación de la inclinación desde 0 a 90º a 1000 W·m-2 de irradiancia)	± 0.5

Tabla 3. Características medias de los pirheliómetros.

2.2.3 Heliógrafos

La heliofanía está directamente relacionada con la radiación solar y, en particular, con la radiación visible. De hecho, la condición de sol brillante puede asociarse a la aparición de sombras tras objetos iluminados. La WMO (1991) define el número de horas de sol o heliofanía como la suma del subperíodo para el que la irradiancia solar directa supera 120 W·m-2. La heliofanía tiene dimensiones de tiempo, y se mide en horas o segundos.

El interés de los datos de heliofanía va más allá de su utilización directa en estudios climáticos relacionados con la agricultura o la medicina, ya que constituye una de las fuentes principales –si no la principal- de datos para la estimación de la radiación solar mediante diversas técnicas. Incluso puede proporcionar una estimación detallada del potencial de radiación directa, fuente de energía primaria para los sistemas termosolares de concentración (Pérez, s.f.).

Figura 4. Heliógrafo.

2.3. LEDs como sensores

La conductividad eléctrica de los materiales semiconductores no es tan elevada como la de los metales; sin embargo, tienen algunas características eléctricas únicas que los hacen especialmente útiles. Cuando las características eléctricas están determinadas por átomos de impureza, se dice que se trata de un semiconductor extrínseco.

El comportamiento eléctrico está determinado por las impurezas, las cuales aún en pequeñas concentraciones introducen un exceso de electrones o huecos.

- Semiconductor extrínseco de tipo n

Un átomo de Si tiene cuatro electrones formando enlaces covalentes con cada uno de los cuatro atomos vecinos de Si. Supongamos que se añade como impureza sustitucional un átomo de impureza con valencia 5. Solamente cuatro de los cinco electrones formarán enlaces. El electrón extra que no se enlaza queda ligado débilmente a la región alrededor del átomo de impureza. Éste puede separarse fácilmente del átomo de impureza, en cuyo caso se convierte en un electrón libre o de conducción. De este modo, el número de electrones en la banda de conducción excede con mucho al número de huecos en la banda de valencia. Los electrones son los portadores mayoritarios de carga debido a su densidad o concentración; los huecos se denominan portadores minoritarios de carga.

- Semiconductor extrínseco de tipo p

Al añadir al silicio o al germanio impurezas sustitucionales trivales, uno de los enlaces covalentes alrededor de cada uno de los átomos carece de un electrón. Esta deficiencia

puede verse como un hueco débilmente ligado a un átomo de impureza. Este hueco puede ser liberado del átomo de impureza por la transferencia de un electrón desde un enlace adyacente. Un hueco móvil es considerado como un estado excitado y participa en el proceso de conducción. Los huecos están presentes en concentraciones mucho mayores que los electrones siendo las partículas de carga positiva las principales responsables de la conducción eléctrica.

Un rectificador o diodo es un dispositivo electrónico que permite que la corriente fluya solamente en una dirección. La unión rectificadora p-n se construye a partir de un semiconductor que es dopado a fin de que sea tipo n en un lado y tipo p en el otro. Si se unen piezas de los materiales tipo n y tipo p, se obtiene un rectificado deficiente, ya que la presencia de una superficie entre las dos secciones hace que el dispositivo sea muy ineficiente.

Antes de aplicar ningún potencial, los huecos serán los portadores mayoritarios en el lado p, y los electrones predominarán en la región n. Durante la aplicación de un potencial directo, los huecos en el lado p y los electrones en el lado n son atraídos hacia la unión. A medida que los electrones y los huecos se encuentran unos con otros cerca de la unión, se recombinan y aniquilan continuamente liberando energía. Este flujo de electrones desde el material tipo n al material tipo p genera una corriente desde el ánodo hasta el cátodo permitiendo el desarrollo de un voltaje a través de la unión dopada PN (Callister, 2009).

En el caso de los diodos LEDs, su funcionamiento es igual a diferencia de la liberación de energía. En un diodo convencional, la energía que se libera en la unión de un electrón y un hueco es energía térmica; pero en un diodo LED, la energía que se libera en esa unión es en forma de fotones. Según el material del diodo LEDs, las longitudes de onda emitidas son diferentes y los voltajes desarrollados también (Boylestad & Nashelsky, 2009).

Diodos emisores de luz

Color	Construcción	Voltaje en directa típico (V)
Ámbar	AlInGaP	2.1
Azul	GaN	5.0
Verde	GaP	2.2
Naranja	GaAsP	2.0
Rojo	GaAsP	1.8
Blanco	GaN	4.1
Amarillo	AlInGaP	2.1

Tabla 4. Diodos emisores de luz.

Sin embargo, si un fotón perdido choca en la región de empobrecimiento, el fotón puede ceder su energía a un electrón en la red cristalina, lo cual causa que el electrón se desprenda del átomo y recorra la red libremente dejando atrás un hueco en la red.

En condiciones normales, el par electrón-hueco podría recombinarse, pero el campo eléctrico producto de una polarización inversa los mantiene separados. El agujero es

atraído a la región n y el electrón a la región p. Una vez que alcancen sus respectivas regiones, se combinan y se cancelan con sus opuestos generando un movimiento de las cargas eléctricas y produciendo una corriente muy débil. Esta corriente será proporcional al nivel de brillo producido por la radiación incidente y al cual está sujeto el LED.

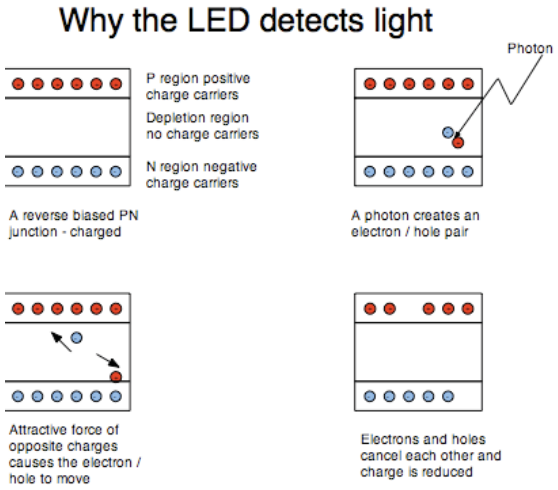


Figura 5. Incidencia de un fotón en un diodo LED.

Normalmente, para un LED sin lentes filtradas, la unión PN puede responder a longitudes de onda iguales o más cortas que las que han sido diseñadas para emitir. Obviamente, para LEDs con lentes filtradas, la respuesta es solamente adecuada a las longitudes de onda que pueden pasar a través del filtro (Oates, Timoschenko, Ruiz-Canales, Molina-Martínez, & Vázquez de León, 2016).

Sin embargo, estas corrientes generadas son excesivamente pequeñas y el funcionamiento observado de los LEDs bajo estas condiciones de uso se determina por otras características del dispositivo y de los métodos empleados para medirlos. En particular, estos incluyen capacitancia y resistencia efectiva. Dependiendo del tiempo empleado y de las condiciones de iluminación, se pueden generar intensidades de corriente del orden de nano y microamperios.

Aunque la corriente generada por la unión PN dopada se determina por el brillo y la longitud de ondas de la radiación, el máximo valor de tensión es determinado por las propiedades físicas del material. Éste se ve determinado por el valor de "Voltaje de Saturación", característico de cada diodo LED. Posteriormente, la capacitancia del LED dopado y del equipo y cables empleado puede producir que este voltaje pueda requerir una cantidad de tiempo significativa para acumular la corriente necesaria como para que el dispositivo empleado sea capaz de detectar condiciones de luz pobre.

Estas características pueden ser explotadas para producir un sensor capaz de detectar un amplio rango de niveles de brillo, observando voltajes desarrollados en escalas de tiempo cortas para ciertas condiciones de brillo y escalas de tiempo largas para niveles bajos de luz.

Luz	Longitud de onda (nm)	LED (nm)	Máxima lectura ADC (0 – 1023)	Máxima tensión (V)
Infrarroja	>760	940	195	0.95
Rojo	610-760	620	331	1.62
Verde	500-570	515	434	2.12
Azul	450-500	465	495	2.42
Ultravioleta	<400	405	423	2.07

Tabla 5. Lecturas de los LEDs.

Para obtener un desglose espectral lo más preciso posible, el sensor debe de estar formado por cinco diodos de 5 mm de encapsulado. Este sensor nos proporcionará un buen desglose de las longitudes de onda en la zona del rojo, azul, verde, infrarrojo y ultravioleta del espectro electromagnético; dándonos a conocer el tipo de longitudes de onda presentes en la radiación recibida. Todos los LEDs son conectados, vía una resistencia de 330Ω, a 0V.

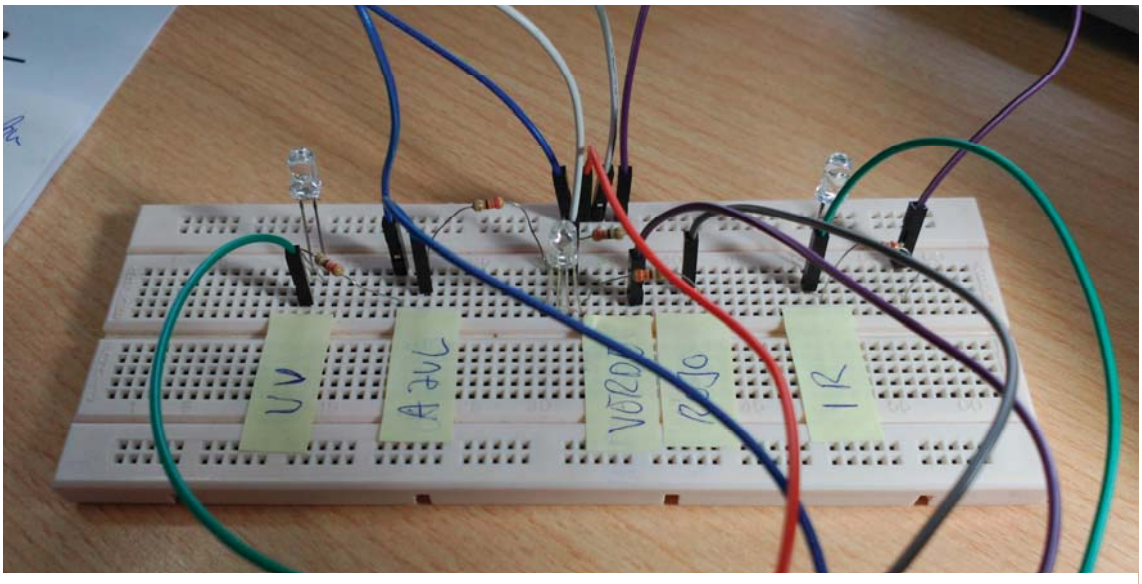


Figura 6. Test con los LEDs.

Es muy importante a tener en cuenta que, cuando el equipo esté conectado a un ordenador que está siendo alimentado con el cargador, es fundamental retirarlo y trabajar con la batería del ordenador. El ruido electromagnético que producen los cargadores de AC a DC es suficientemente elevado como para generar distorsiones en los valores de corriente en los LEDs y a su vez provocar lecturas erróneas en los puertos ADC del microcontrolador.

2.3.1 Toma de valores

El método empleado para la recogida de valores consiste en series de 11 lecturas de niveles de luz, cada una espaciada una unidad de tiempo respecto a la anterior. Antes de empezar las series de cada canal, el ánodo de cada LED es configurado como entradas de lectura al convertidor ADC y los cátodos a GND.

Lo que hacemos es usar sus débiles propiedades como condensador para cargarlo y descargarlo. Este tiempo de carga o duración de la exposición permite que la excitación del LED sea mayor o menor. Según esa excitación tendremos una sensibilidad determinada. Para tiempos de exposición altos, la excitación del LED será muy elevada y así su resultado. Sin embargo, tiempos de exposición más bajos, y por lo tanto menor excitación, nos permite obtener mayor sensibilidad a cambios en ese tipo de radiación.

Las primeras cuatro lecturas se toman en sucesiones inmediatas con las interrupciones del microcontrolador deshabilitadas. Como la tecnología utilizada son las placas de desarrollo Arduino, el tiempo requerido para la toma de las cuatro medidas es menor a 1 ms. Los retrasos entre medidas se incrementan a 1 ms y 2 ms para la 5º y 6º medida, pasando a 7 ms para la 7º medida. Luego, estos tiempos pasan a 20 ms, 70 ms, 200 ms y 700 ms para las lecturas 8º, 9º, 10º y 11º. El resultado es un grupo de lecturas espaciadas linealmente en el tiempo con cortas exposiciones de tiempo, seguidas con una serie de siete lecturas espaciadas con tiempos de exposición crecientes exponencialmente (Oates, Timoschenko, Ruiz-Canales, Molina-Martínez, & Vázquez de León, 2016).

2.4. Reflectometría en el dominio de la frecuencia (FDR)

2.4.1 Introducción

Después de avanzar profundamente con el analizador de luz solar de bajo coste teniendo como objetivo mejorar el control de la radiación que incide sobre los cultivos para controlar su crecimiento, se creyó conveniente iniciar un trabajo paralelo para incluir una sonda de control de la humedad. Ya que ella este parámetro determina también el crecimiento del cultivo.

Existe un amplio abanico de técnicas para la medida de la humedad del suelo en campo basadas en la electricidad. Estas incluyen métodos basados en la resistividad, como el método Wenner y el de Schlumberger, además de los métodos capacitivos como el del reflectómetro del dominio de la frecuencia “Frequency Domain Reflectometery” y la Reflectometría del dominio del tiempo “Time Domain Reflectometry” (TDR) y así como las técnicas basadas en la radiación como la sonda de neutrones. Cabe destacar que, por lo general, los dispositivos comerciales son caros oscilando entre 500 y 1600 dólares y careciendo de capacidad para integrar en sistemas de captación de datos (Oates, Ruiz-Canales, Molina-Martínez, & Vázquez de León, Compromiso entre coste y frecuencia de trabajo en sensores FDR de bajo coste para la gestión del riego, 2015).

Figura 7. Método Wenner.

Experimentos previos establecieron el punto de partida potencial del diseño de un sensor de bajo coste basado en la resistividad del terreno según el método Wenner. Se obtuvieron resultados con una alta correlación comparando con una sonda comercial. Sin embargo, las técnicas basadas en la resistividad del terreno son vulnerable a una variedad de condiciones diferentes del suelo como composición, textura, variación del pH, salinidad y temperatura.

Para el caso de la técnica FDR (Reflectometría en el dominio de la frecuencia), la humedad del suelo depende más de la constante dieléctrica del mismo que de la conductividad eléctrica y es teóricamente menos vulnerable a la salinidad. El principio de funcionamiento de una sonda de capacitancia FDR depende del hecho que la constante dieléctrica del agua y del aire difieren en un factor de 80. Por tanto, la presencia de agua en el suelo entre las placas de una sonda FDR produce un cambio altamente significativo en su capacitancia. Cuanta más alta es la concentración de agua, más alta es la capacitancia. Son dos los métodos eléctricos empleados para determinar la capacitancia efectiva de la sonda (Oates, Ruiz-Canales, Molina-Martínez, & Vázquez de León, Compromiso entre coste y frecuencia de trabajo en sensores FDR de bajo coste para la gestión del riego, 2015).

Al aislar eléctricamente las sondas no hay flujo de corriente directa en el suelo y el efecto conductivo de las sales del suelo se minimiza. Es por ello que a las sondas utilizadas se les aplica un recubrimiento con una pintura especial para reducir esos efectos de conductividad eléctrica y así sus efectos por la salinidad.

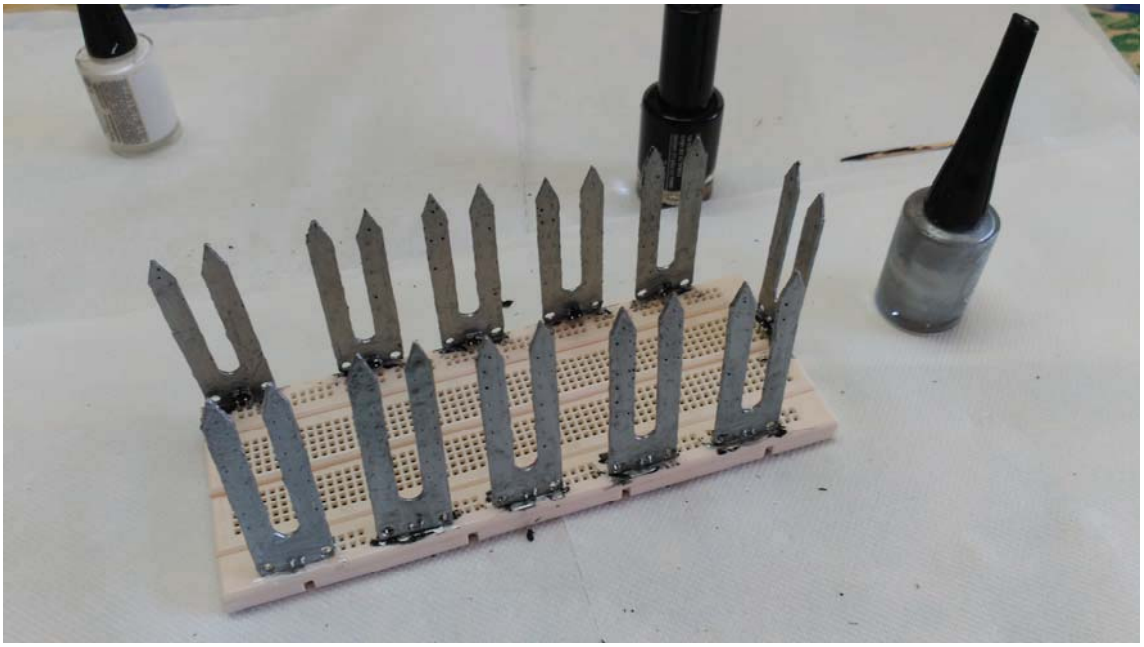


Figura 8. Tratamiento de pintura aplicado a las probetas para dispositivos FDR.

2.4.2 Primer método

Inicialmente se usa la sonda como el componente capacitivo de un filtro de paso bajo. Un microcontrolador genera una onda cuadrática de frecuencia fija de valores 250 kHz, 125 kHz, 62,5 kHz y 31 kHz. Estas frecuencias fueron seleccionadas por ser más altas que las que se usan en la práctica, dado el limitado producto de la ganancia en lazo abierto disponible y el ancho de banda (GBP) de los amplificadores operacionales de bajo coste.

Se hace pasar la señal cuadrada por una resistencia de 47 kΩ y se conecta con un terminal de la sonda FDR capacitiva. El otro extremo de la sonda se conecta a tierra. Adicionalmente, la unión de la sonda y la resistencia de 47 kΩ se conecta a la entrada de los amplificadores operacionales empleados. Estos se configuran para amortiguar los valores máximos con una resistencia retroalimentada de 330 Ω desde la salida hasta la entrada $-V_{entrada}$. La salida se conecta mediante un diodo 1N4001 a un circuito "detector de picos" consistente en una resistencia de 1,8 MΩ y un condensador de 100 nF conectados en paralelo y ambos conectados a tierra. La salida se conecta a una entrada del convertidor analógico-digital (ADC) del microcontrolador. Este voltaje se prueba después de un periodo de estabilización de 20 ms después de que se aplique una señal de onda cuadrática. Una vez la potencia de la señal se ha registrado, la señal de la onda cuadrática se para y el microcontrolador vuelve al estado de alta impedancia (Oates, Ruiz-Canales, Molina-Martínez, & Vázquez de León, Compromiso entre coste y frecuencia de trabajo en sensores FDR de bajo coste para la gestión del riego, 2015).

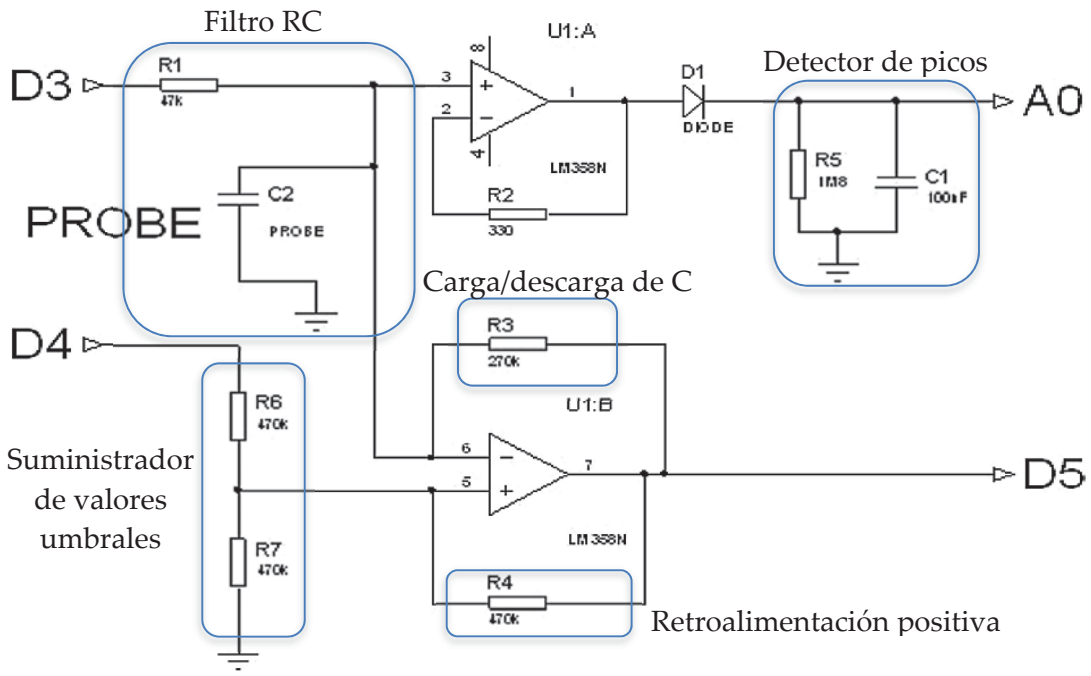


Figura 9. Circuito FDR.

Así, cuando la sonda capacitiva FDR está al aire o en un suelo muy seco, su capacitancia es baja (del orden de 30 pF). Cuando la sonda capacitiva está en un suelo muy húmedo, su capacitancia aumenta entre 200 a 400 pF.

La atenuación teórica de la frecuencia crítica de 3dB solamente se aplica a señales de onda senoidales mientras que en el experimento se emplean ondas cuadráticas. Esto significa que hay importantes componentes armónicos con amplitud decreciente, en la frecuencia base de múltiplos impares. Como se tiene solamente un valor pico de una señal y esos armónicos están normalmente sobre la frecuencia crítica del filtro de paso bajo, ello no influye significativamente en el comportamiento del sensor.

2.4.3 Segundo método

Seguidamente, usamos la sonda como el elemento capacitivo en un circuito oscilador, cargando y descargando repetidamente el condensador cuando la diferencia de potencial pasa entre dos entradas de control. El tiempo empleado por la diferencia de potencial para subir y bajar entre esas entradas se mide sobre cuatro ciclos de media onda para suministrar un indicador del valor de la capacitancia.

Para este método, la resistencia de 47 kΩ y el primer amplificador no juegan un papel significativo al encontrarse en un estado de alta impedancia. La sonda capacitiva FDR se conecta entre tierra y la entrada $-V_{entrada}$ del segundo amplificador. Ésta se conecta a una resistencia de retroalimentación de 270 kΩ hasta la salida del segundo amplificador. La salida también se retroalimenta con la entrada $+V_{entrada}$ del segundo amplificador para cambiar la tarjeta de entrada y determinar cuál es el estado de cambios de salida del amplificador. Esta salida también se determina por el punto medio del divisor de

potencial desde el control del microcontrolador de salida en la patilla D4 y 0v, consistente en dos resistencias de 470 kΩ adicionales.

El oscilador se enciende poniendo el pin de salida D4 en alto. La entrada $+V_{entrada}$ al segundo amplificador aumenta alrededor de 2.5 V mediante la resistencia de retroalimentación conectada a la salida del amplificador. Si esta salida es también alta, en el umbral $+V_{entrada}$ se alcanza el valor de 3.2 V y la sonda FDR capacitiva empieza a cargar mediante la resistencia de retroalimentación de 270 kΩ en el lado de entrada $-V_{entrada}$. Cuando esta carga alcanza los 3.2 V, la salida del segundo amplificador oscilará por debajo. Esto provoca que la sonda FDR capacitiva empiece a descargarse mediante la resistencia de 270 kΩ. Entonces la resistencia de retroalimentación de 470 kΩ en el lado $+V_{entrada}$ cambia la entrada $+V_{entrada}$ a alrededor de 1.8 V. Así la carga de la sonda FDR capacitiva alcanzaría valores por debajo de 1.8 V. En este momento, la salida del amplificador vuelve a cambiar a alta y el ciclo se repite. La salida del segundo amplificador se monitoriza con una patilla de entrada de microcontrolador (Oates, Ruiz-Canales, Molina-Martínez, & Vázquez de León, Compromiso entre coste y frecuencia de trabajo en sensores FDR de bajo coste para la gestión del riego, 2015).

El tiempo requerido para cargar y descargar la sonda FDR capacitiva se determina por su valor de capacitancia, que a su vez depende de la cantidad de agua en el área inmediatamente alrededor de la sonda. Por tanto, a valores mayores de capacitancia, los tiempos de carga y descarga son superiores.

3

Materiales y métodos

Descripción del hardware empleado en el montaje de los dispositivos y software diseñado para la obtención de los datos necesarios, además de una exposición del programa de adquisición de datos.

3.1. Hardware

3.1.1 Introducción a Arduino

Un sistema integrado es aquel sistema electrónico en el que confluyen hardware y software. Normalmente tiene un pequeño procesador que puede ser un microprocesador o un microcontrolador como parte central.

Los desarrolladores de sistemas electrónicos integrados tienen la necesidad de elegir dicho procesador de entre una gran variedad de fabricantes existentes. Por otro lado, dependiendo de la elección tendrán que disponer el software necesario y elegir uno de los diferentes compiladores existentes para la programación del microcontrolador usado.

La aparición de Arduino resuelve este caos para simplificar el uso de los microcontroladores. Arduino empieza con un proyecto de Massimo Banzi, un alumno en el Interaction Design Institute Ivrea (Italia) en el año 2005 y toma forma con la realización de una tesis sobre hardware. Finalmente, un grupo de investigadores realiza una optimización de éste para que sea económico y accesible para la comunidad (Lajara-Vizcaíno & Sebastià, 2014).

El ecosistema de Arduino está formado por tres elementos principales:

- Hardware: Placa electrónica básica para hacer desarrollos rápidos y económicos. El hardware es libre y permite una forma simple de aprender electrónica básica. También permite ampliar el hardware mediante el uso de otras placas que se conectan a la del microcontrolador.
- Entorno de programación: El entorno integrado de desarrollo o IDE (Integrated Development Environment) es un software que permite realizar Sketchs y compilarlos. Este software también permite realizar la programación del chip, pero para ello es necesario un programa extra llamado bootloader. Este es un pequeño programa ya incorporado que permite la comunicación con el IDE y el cambio del resto del código dentro del chip. La IDE es versátil para diferentes placas con diferentes microcontroladores como: ATMEGA328P, ATMEGA2560, ATtiny85, ATmega32U4…
- Comunidad: La participación activa de gente desinteresada en desarrollos e ideas mantienen la plataforma Arduino actualizada.

3.1.2 Arduino UNO

La placa Arduino UNO es la más famosa dentro de todas las placas “Open Source”. Está compuesta, fundamentalmente, por un microcontrolador ATmega328P de la casa Atmel, una serie de pines libres para acceder al microcontrolador y un sistema de comunicación USB – USART.

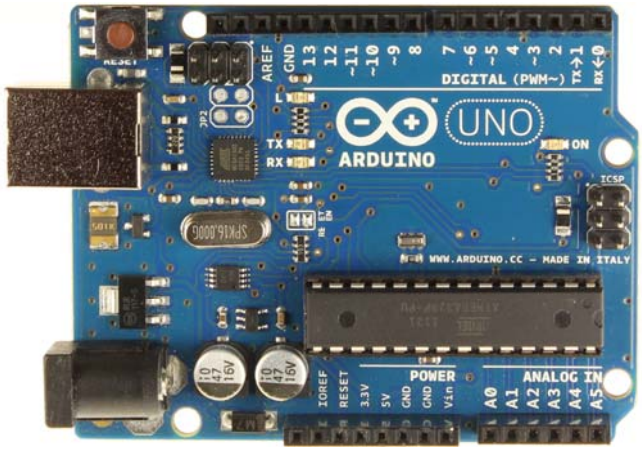


Figura 10. Arduino UNO.

3.1.2.1 Alimentación

La tensión a la que trabaja son 5 V, pudiendo venir desde una fuente externa al conector de la placa o al pin Vin, o via puerto USB. Esta alimentación está basada en un regulador NCP1117ST50T3G que se encarga de mantener una tensión estable de 5 V y proporcionando hasta 1 A de corriente. Además, los condensadores electrolíticos de 47 µF y 10 µF proporcionan estabilidad a la tensión, suministro instantáneo de corriente según la demanda y filtro de ruidos electromagnéticos.

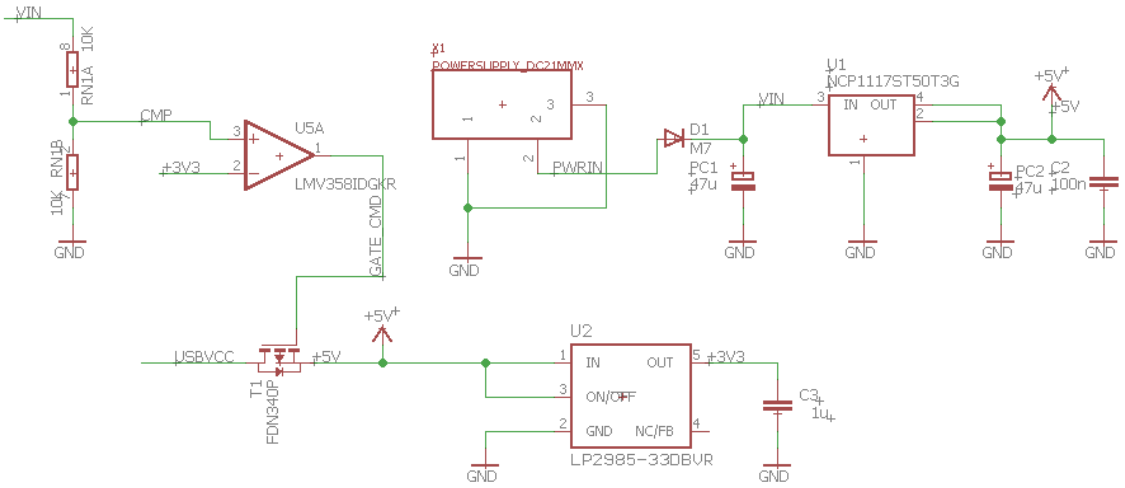


Figura 11. Circuito de alimentación de Arduino UNO.

Como diversos componentes trabajan a una tensión de 3,3 V, el regulador LP2985-33DBVR se encarga de proporcionar ese nivel de tensión y proporcionar hasta 150 mA.

El amplificador operacional U5A sirve para activar o desactivar el transistor T1, que funciona como interruptor. Si hay alimentación externa se utiliza esta y el transistor bloquea la alimentación del USB para que no haya conflictos entre ellas (Lajara-Vizcaíno & Sebastià, 2014).

3.1.2.2 Microcontrolador

El corazón del Arduino UNO es un microcontrolador ATmega328P de la compañía Atmel. La familia ATmega es un tipo de microcontrolador basado en la arquitectura Harvard (AVR) modificada de 8 bits con tecnología RISC (Reduced Instruction Set Computing), es decir, con memoria separadas para programar y datos y un conjunto de instrucciones simples para realizar la programación.

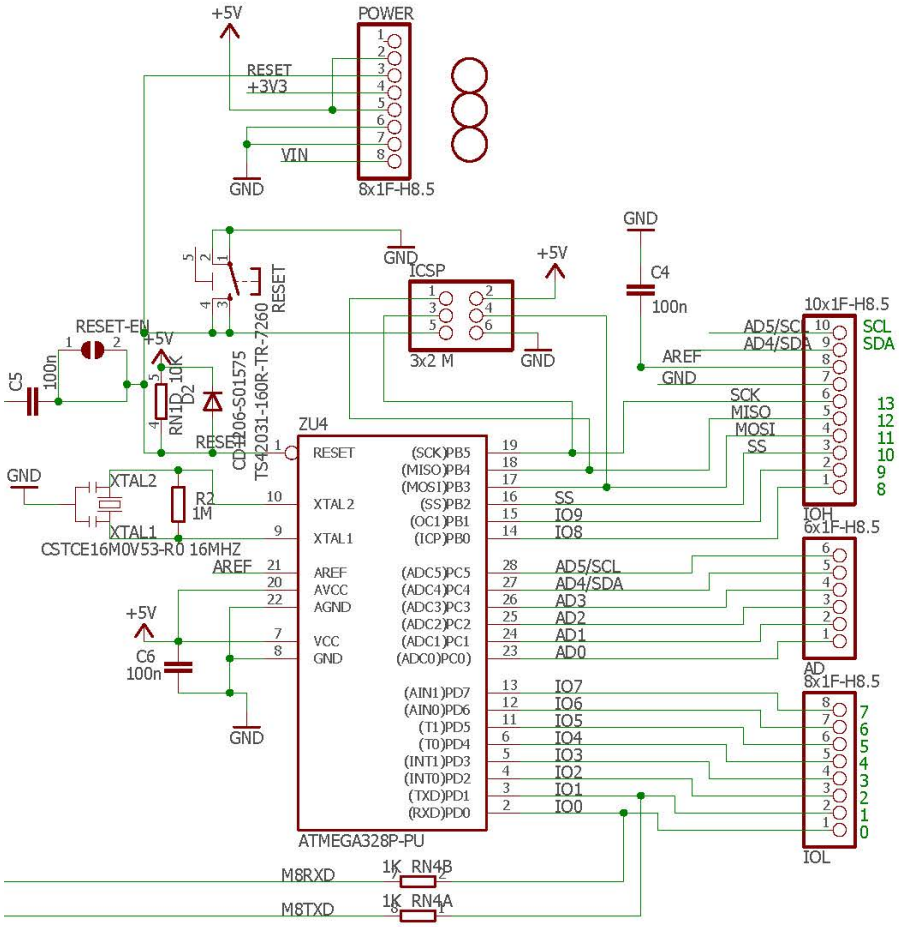


Figura 12. Circuito ATmega328P.

El microcontrolador combina una memoria Flash interna de 32 kB para almacenar el programa, una memoria EEPROM de 1 kB para datos no volátiles y 2 kB de memoria SRAM para datos volátiles. Tiene 23 pines de entrada/salida digital, 3 timers/contadores, interrupciones internas y externas, soporta comunicación serie/USART, I2C y SPI, tiene 6 canales de entrada a un convertidor de analógico-digital de 10 bits de resolución, tiene un timer watchdog con oscilador interno y 5 modos diferentes de ahorro de energía. Su rango de trabajo va desde tensiones de 1.8 V a 5,5 V y de -40 ºC a 85 ºC (Lajara-Vizcaíno & Sebastià, 2014).

Parámetros	Valor
Memoria Flash	32 kB

Memoria SRAM	2 kB
Memoria EEPROM	1024 B
CPU	AVR de 8 bits
Frecuencia del Reloj de Cuarzo	16 MHz
Pines totales / Pines I/O	32 / 23
Canales ADC/Resolución/Velocidad	8 canales / 10 bits / 15 kbps
Canales PWM	6
Interrupciones	24
Timers	2 de 8 bits y 1 de 16 bits
SPI	2
I2C	1
USART	1
Tensión de alimentación	1.8 – 5.5 V

Tabla 6. Características del ATmega328P.

3.1.2.3 USB

El sistema USB de la placa UNO está formado por otro microcontrolador, el ATmega8U2-MU. Éste presenta implementado internamente un bloque hardware para poder tener una conexión directa con un bus USB a través de dos resistencias de 22 Ω.

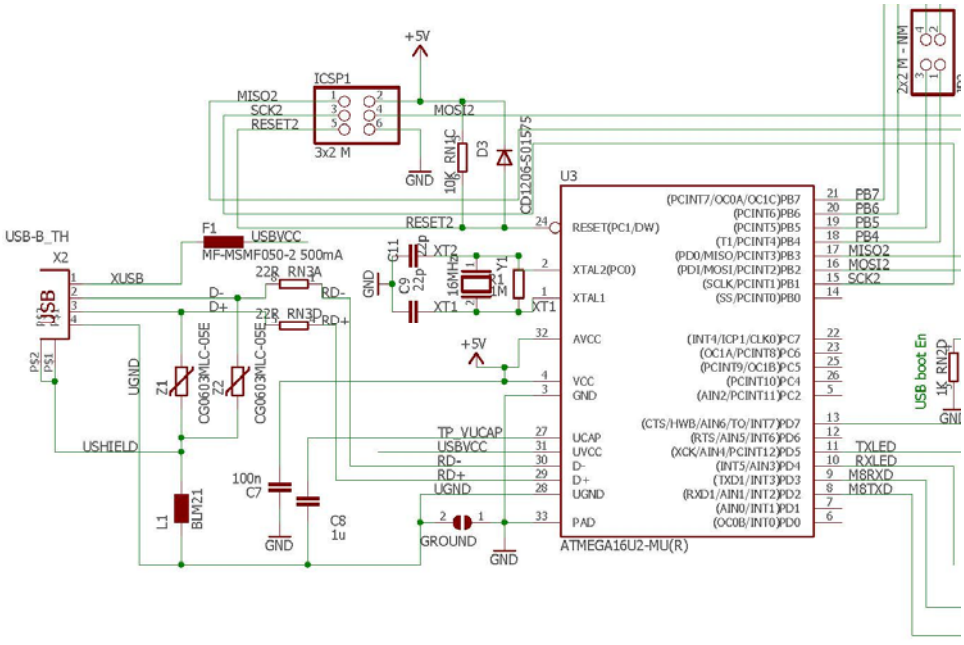


Figura 13. Circuito USB de Arduino UNO.

3.1.2.4 Otros periféricos

- Sistema de reloj: El microcontrolador tiene diferentes opciones de origen de reloj además de diferentes "prescaler" para poder reducir la frecuencia y optimizar el consumo. Por defecto, la frecuencia es 16 MHz.
- Timers: El microcontrolador presenta 3 "timers/counters", dos de 8 bits y uno de 16 bits. Son módulos que funcionan en paralelo a la CPU y de forma independiente a ella. El funcionamiento básico consiste en aumentar un valor del registro del contador al ritmo que marca su señal de reloj.
- PWM: La modulación por anchura de pulso es una técnica de modulación que se basa en la variación de la anchura del pulso de una señal digital en base a una señal analógica dada. Habitualmente se usa para el control de motores, regulación de potencia, rectificación y como una forma sencilla de conversión digital-analógico.
- Comunicación I2C: Permite la interconexión de hasta 128 dispositivos diferentes usando solo dos líneas bidireccionales de bus, una para el reloj (SCK) y otra para datos (SDA).
- Comunicación SPI: Permite la transferencia de datos síncrona a alta velocidad entre el microcontrolador y dispositivos periféricos haciendo uso de 4 buses (SCK, MISO, MOSI, SS).
- Comunicación USART: Se trata de un puerto de comunicaciones serie que puede ser tanto síncrono como asíncrono. Emplea dos buses, el RXD y TXD para recibir y transmitir.

3.1.3 Arduino Pro Mini

Esta placa Arduino es característica por presentar el mismo microcontrolador ATmega328P que la UNO, pero siendo el tamaño mucho más reducido.

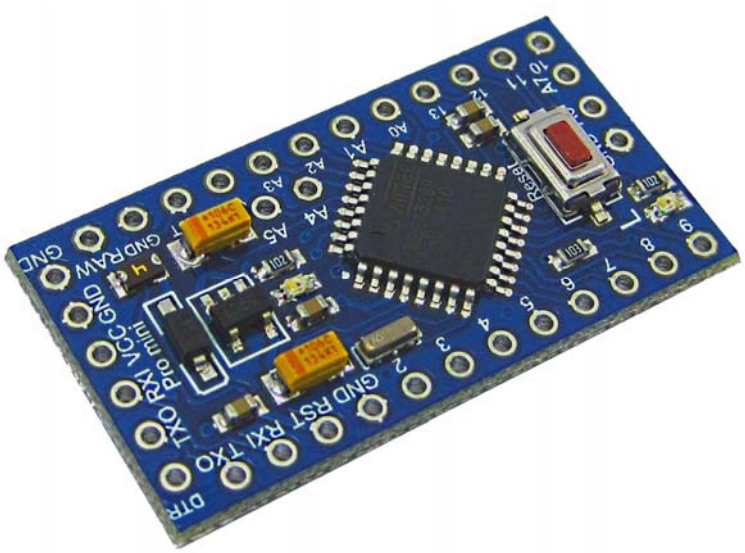


Figura 14. Arduino Pro Mini.

Tenemos las mismas prestaciones que nos presenta el microcontrolador, pero al ganar en menos espacio perdemos otras como:

- USB: La placa no contiene el chip ATmega8U2-MU y por lo tanto necesitaremos hardware externo a la placa para poder comunicar a la misma con el puerto USB de nuestro ordenador. Existen diferentes módulos USB que llevan integrado un chip como el CP2102 o el CH340G que nos permite la comunicación directa entre el ordenador y el Arduino Pro Mini.
- 3.3 V: Las Arduino Pro Mini se pueden encontrar de 5 V a 16 MHz y de 3.3 V a 8 MHz, pero ninguna de ambas presenta un regulador de voltaje para la otra tensión. Es por tanto necesario añadir, por ejemplo, un regulador HT7333 para 3.3 V siendo muy interesante para nuestro proyecto por su muy bajo consumo de corriente durante periodos de inactividad (4 μA).

3.1.4 NRF24L01

El NRF24L01 es un chip de comunicación inalámbrica fabricado por Nordic Semiconductor.

Éste integra un transceptor RF (transmisor + receptor) a una frecuencia entre 2.4GHz a 2.5GHz, pudiendo elegir entre 125 canales espaciados a razón de 1MHz. Se recomienda usar las frecuencias de 2501 a 2525 MHz para evitar interferencias con las redes Wifi.

La velocidad de transmisión es configurable entre 250 Kbps, 1Mbps, y 2 Mbps y permite la conexión simultánea con hasta 6 dispositivos. La tensión de alimentación del NRF24L01 es de 1.9 a 3.6V, aunque los pines de datos son tolerantes a 5V. El consumo eléctrico en Stand By es bajo, y de unos 15mA durante el envío y recepción, haciéndolo muy interesante para dispositivos alimentados con baterías que requieren un alto control del consumo de energía (Nordic Semiconductor, 2008).

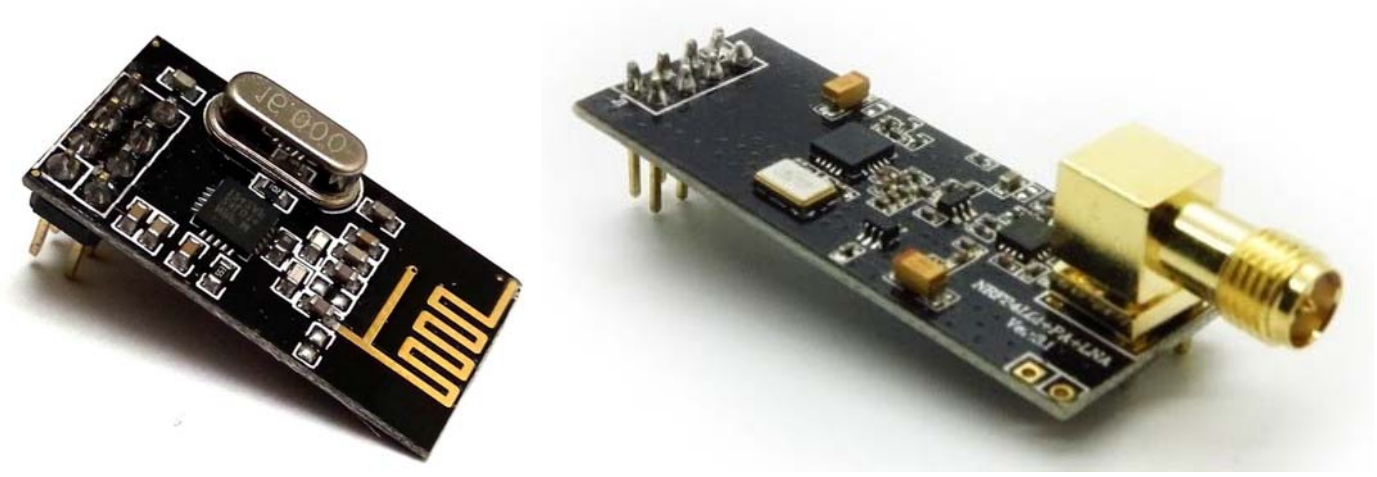


Figura 15. NRF24L01.

El NRF24L01 también incorpora la lógica necesaria para que la comunicación sea robusta, como corrección de errores y reenvío de datos si es necesario, liberando de esta tarea al procesador. El control del módulo se realiza a través de bus SPI, por lo que es sencillo controlarlo desde un procesador como Arduino.

Sin embargo, el alcance real se ve limitado por muchos factores, incluso en condiciones de visibilidad directa sin obstáculos. Con el módulo alimentado desde Arduino y velocidad de transmisión de 2 Mbps el alcance será apenas de 2-3 metros. Un factor de gran impacto en el alcance es la alimentación del módulo. Para conseguir el máximo alcance conviene alimentar el módulo con una fuente externa de 3.3V, estable y con potencia suficiente.

Este módulo ha sido mejorado en uno nuevo donde se incluye el NRF24L01 y se le añade un circuito amplificador de potencia (PA) y un circuito amplificador de bajo ruido (LNA). Nos permite transmitir a mayores distancias y lo hace más estable para trabajos simples en la industria. Al añadir el circuito PA y el circuito LNA, además de un interruptor RF y de un filtro de paso de banda obtenemos un módulo profesional de transmisión bidireccional con amplificación. El alcance de la transmisión puede alcanzar una distancia de 1000 metros con una antena de 2 dB que puede ser añadida, permitiendo una velocidad de 250 kbps en campo abierto.

Los módulos NRF24L01 son ampliamente empleados por su bajo precio y buenas características. Podemos emplearlos, por ejemplo, para recepción remota de sensores como temperatura presión, aplicaciones de domótica y edificios inteligentes, activación remota de dispositivos como iluminación, alarmas, y control o monitorización de robots en el rango de hasta 700 metros (Llamas, Comunicación inalámbrica a 2.4 GHz con Arduino y NRF24L01, 2016).

Especificaciones	Valor
Voltaje	3-3.6V (recomendados 3.3V)
Máxima potencia de salida	+20 dBm
Corriente máxima en modo emisor	115 mA
Corriente máxima en modo receptor	45 mA
Corriente en modo apagado	4.2 uA
Rango de temperatura	-20 - 70 ℃
Sensibilidad en recibir a 2Mbps	-92 dBm
Sensibilidad en recibir a 1Mbps	-95 dBm
Sensibilidad en recibir a 250kbps	-104 dBm
Ganancia del amplificador de potencia (PA)	20 dB
Ganancia del amplificador de bajo ruido (LNA)	10 dB
Ruido del LNA	2.6 dB
Ganancia máxima de la antena	2 dBI
Alcance a 2Mb en campo abierto	520 metros
Alcance a 1Mb en campo abierto	750 metros
Alcance a 250Kb en campo abierto	>1000 metros

Medidas	38.00mm * 16.46mm * 0.8mm

Tabla 7. Características del NRF24L01.

3.1.5 Solar Charger Shield

El "Solar Charger Shield" es una placa de expansión compatible con Arduino y que puede ser montar sobre un Arduino UNO que adapta la energía proporcionada por baterías y panel solar para mantener la alimentación del Arduino UNO. Al mismo tiempo, es capaz de cargar la batería en situaciones con luz o simplemente abastecer de energía en momentos sin luz solar.

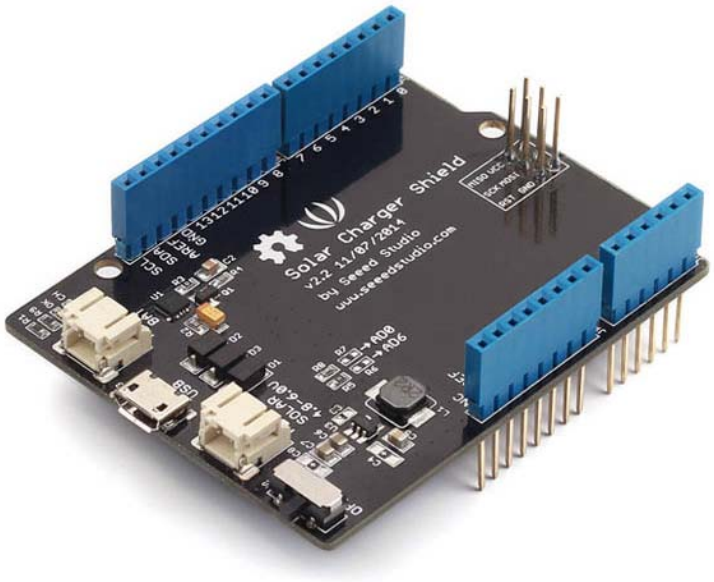


Figura 16. Solar Charger Shield.

Especificaciones:

Parámetros	Valor
Voltaje de entrada desde baterías	3.0 – 4.5 V
Voltaje de entrada desde USB	4.75 – 5.25 V
Voltaje de entrada desde panel solar	4.8 – 6 V
Potencia máxima de salida con baterías	3 W (600mA@5V)

Tabla 8. Características del Solar Charger Shield.

La placa presenta un convertidor/elevador de CC-CC. El ETA1036 es un elevador síncrono de alta eficiencia que puede suministrar hasta 3 W de potencia a una salida desde una entrada de tensión baja (para conseguir una tensión de 5 V desde una entrada mínima de 0.85 V). Lleva incorporado un circuito que es capaz de desconectar la entrada y salida durante un cortocircuito y elimina la necesidad de utilizar un transistor MOSFET externo (ETA Solutions, s.f.).

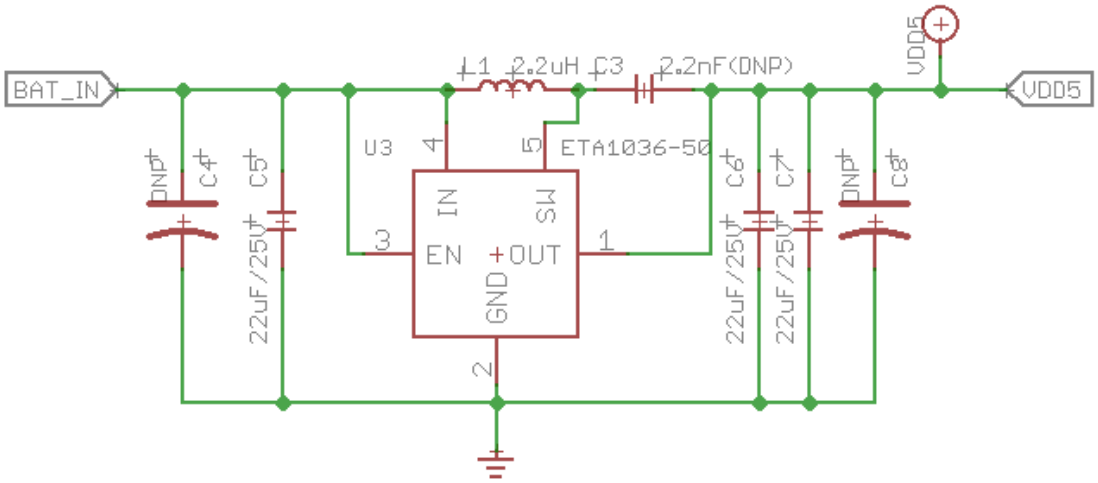


Figura 17. Convertidor CC-CC.

La alternativa más interesante es utilizar una batería LiPo que, más un panel solar, podemos hacer un sistema totalmente autónomo suministrando hasta 600 mA. El shield contiene un cargador lineal de tensión/intensidad constante CN3065 para baterías Li-ion y Li-Po recargables. El dispositivo contiene un MOSFET dentro del encapsulado y elimina la necesidad de una resistencia y diodo externo. Un convertidor ADC de 8 bits de resolución ajusta automáticamente el nivel de corriente de carga basada en la capacidad de la salida y la temperatura de la batería, ya que posee un sensor para ello. Cuando el suministro de energía en la entrada desaparece, el CN3065 entra automáticamente en un modo "durmiente" de baja energía provocando un consumo menor a 3 µA (Consonance Electronic, s.f.).

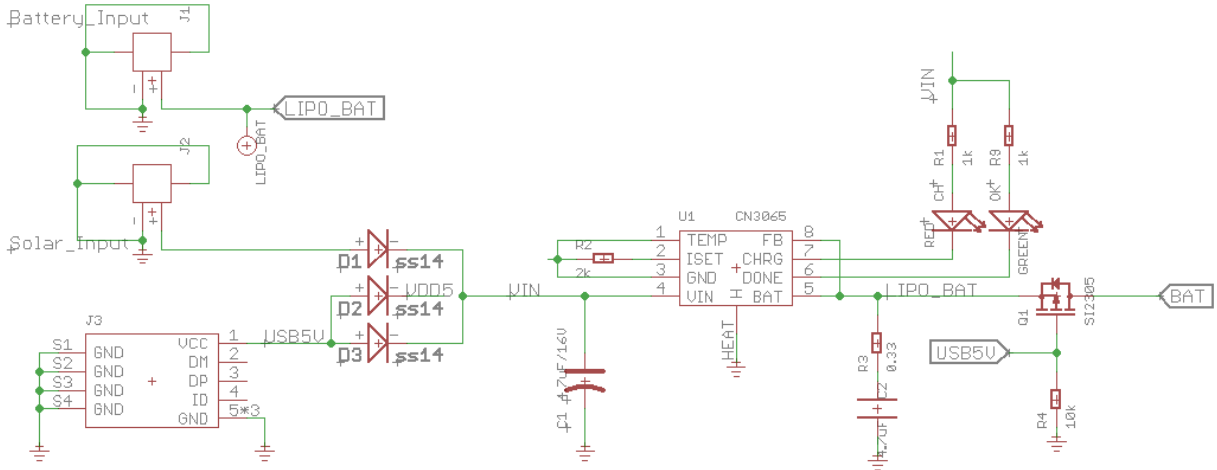


Figura 18. Circuito de carga de batería.

Para que se pueda cargar o alimentar el sistema correctamente mediante un panel fotovoltaico es importante tener en cuenta que la radiación que produce la energía eléctrica son la ultravioleta e infrarroja. Pero para optimizar esa radiación, se debe de buscar el ángulo de inclinación del panel que hace que la cantidad de luz solar incidente

sea máxima. Sabiendo que el sol se desplaza de este a oeste con un ángulo variable desde el zenit hacia el sur según la época del año, unos 20º de inclinación serán suficientes. El panel fue diseñado con el software Solidworks y posteriormente imprimido en una impresora 3D marca Witbox.

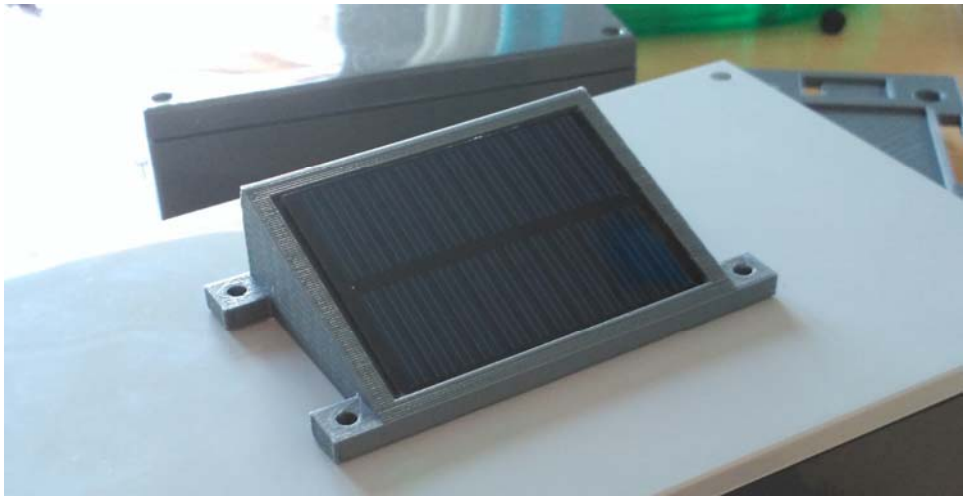


Figura 19. Panel solar.

Es importante tener en cuenta que, aunque los circuitos están diseñados para protegerse frente a cortocircuitos, es importante prevenirse frente a situaciones adversas. Realizando medidas del nivel de voltaje que se consigue con el panel solar mostrado en la imagen anterior, se registraron valores de 7.5 V durante el anochecer.

3.1.6 Pantalla OLED SSD1306

Una pantalla OLED está formada por OLEDs (Organic light-emitting diode), un tipo de LED en el que la capa emisiva está formada por un compuesto orgánico que emite luz en respuesta a la electricidad.

Figura 20. Display OLED 0.96''.

Ésta es una pequeña pantalla OLED de 0.96″ que incorpora el microcontrolador SSD1306 y tienen un tamaño muy reducido de 25mm x 14mm. Son monocromas y tienen una resolución de 128×64 pixels. Al igual que el resto de tipos de pantallas, las pantallas OLED necesitan un controlador específico que convierta los datos recibidos en las señales electrónicas para controlar la pantalla. La comunicación puede realizarse, según modelos, por bus SPI o por bus I2C por lo que es sencillo obtener los datos medidos. La tensión de alimentación admite voltajes de 3.3V y 5V (Llamas, Conectar Arduino a una pantalla OLED de 0.96'', 2016).

Las pantallas OLED tienen la ventaja de tener un consumo muy bajo, en torno a 20mA, dado que solo se enciende el pixel necesario y no requieren de backlight. Esto es especialmente interesante en aplicaciones que funcionan con baterías. Además, tienen una mejor visibilidad en ambientes luminosos, como bajo el sol.

3.1.7 Emisor

El emisor es el dispositivo encargado de tomar valores de luminosidad exterior usando los diodos LEDs como sensores, recolectar los parámetros de la técnica de la Reflectometría en el Dominio de la Frecuencia (FDR) y valores de temperatura del suelo. Es por ello necesario integrar un gran número de componentes en un mismo dispositivo haciéndolo, además, autónomo para largos usos en el tiempo e inalámbrico para la transmisión de información

El emisor estará formado por los siguientes componentes:

- Arduino UNO: Será el corazón del dispositivo que, con el microcontrolador, será el encargado de ejecutar las ordenes de toma de medidas y transmisión de datos, entre otras.
- NRF24L01: Módulo que se encargará de la transmisión de los datos vía RF.
- Solar Charger Shield: Serán los circuitos necesarios para poder brindarle al emisor la autonomía suficiente para que pueda trabajar sin necesidad de alimentación externa o remplazando baterías. Compuesto por éste y un panel solar localizado en la parte superior del emisor podemos usar la energía solar como suministro de potencia a nuestro dispositivo
- DS18B20: Sensor de temperatura resistente al agua que se encargará de monitorizar la temperatura del suelo.

Figura 21. DS18B20.

- Matriz de LEDs: Conjunto de cinco diodos LEDs de color Rojo, Verde, Azul, Infrarrojo y Ultravioleta para usarlos como sensores con el fin de realizar un análisis y desglose del espectro de radiación solar recibida.

Figura 22. Matriz de LEDs.

- Sonda: La sonda será el elemento utilizado para medir la capacitancia del suelo con el fin de conocer la humedad. Hay que tener en cuenta que la sonda debe ser tratada superficialmente ya que estará en contacto con la tierra.

  Durante las medidas, la sonda estará introducida en la tierra pudiendo presentar cantidades variables de agua. Si no se aísla adecuadamente, se pueden producir flujos de electrones entre la sonda y el agua del suelo. Para evitarlo hay que añadir una capa de pintura sobre la sonda para que actué como elemento dieléctrico. Esta pintura debe de presentar características como un coeficiente de dilatación mínimo frente a temperatura y humedad, además de una buena adherencia para evitar poros por donde puede infiltrarse el agua. Haciendo balance entre precio y características, la pintura óptima será algo tan cotidiano como el esmalte de uña.

Figura 23. Sonda FDR con tratamiento.

- PCB del emisor: Empleando una PCB y diferentes componentes, se diseñó y construyó los diferentes circuitos electrónicos necesarios para el buen desempeño de todas las funciones del emisor.

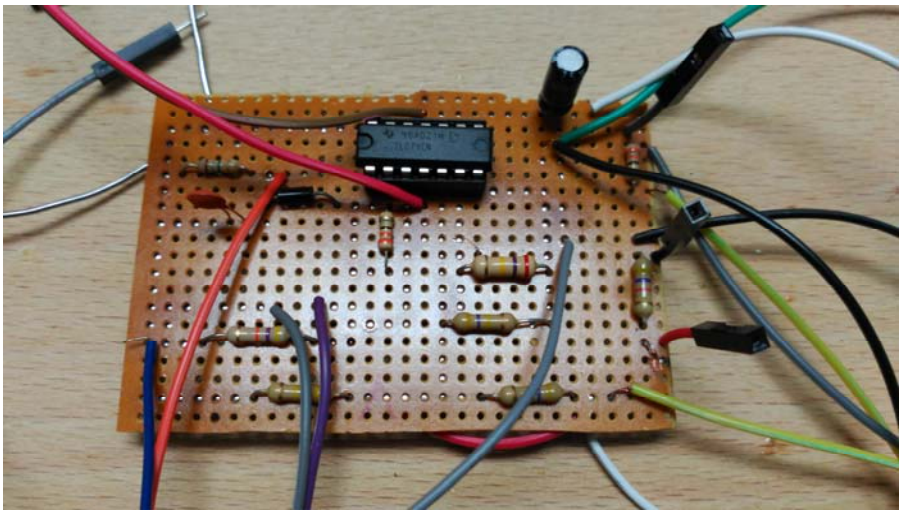


Figura 24. Circuito del emisor.

La versión más reciente de esta PCB está formada por el circuito que proporcionará la medida de la humedad del suelo mediante el método de la Reflectometría en el Dominio de la Frecuencia. Está formado por un amplificador TL074CN al que van ligados dos circuitos:

El primero consiste en usar la sonda como componente capacitivo de un filtro de paso bajo. Este circuito está formado por la resistencia R9, la sonda (C12) que, después del amplificador pasa por el circuito detector de picos hasta A5. El segundo consiste en usar la sonda como el elemento capacitivo de un circuito oscilador. Las resistencias R3 y R8 comienzan a cargar la sonda a través del amplificador y éste se descarga a su vez, viendo en D5 el tiempo que dura cada semiciclo.

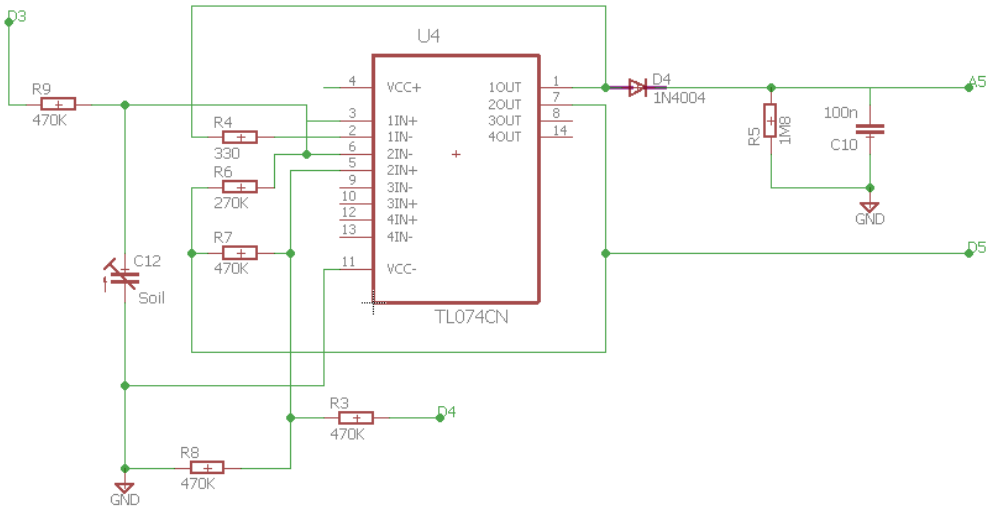


Figura 25. Diagrama del emisor.

Además, usamos una resistencia de 4K7 para el sensor de temperatura DS18B20, entre la alimentación y la línea de datos para recibir los valores de temperatura y un condensador de 10 µF para estabilizar la tensión de alimentación de 3.3 V para el módulo de transmisión de datos NRF24L01.

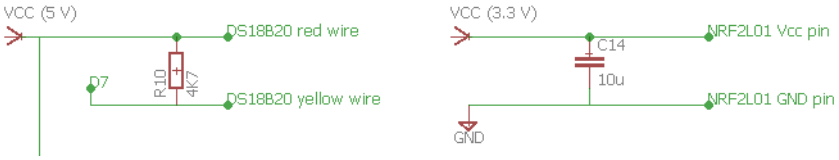


Figura 26. Diagrama del emisor II.

- PCB extra: Se habilitó una segunda PCB para añadir componentes extras que fueran necesarios. Para este caso fueron necesarios 5 resistencias de 330Ω para los LEDs como medida de protección.

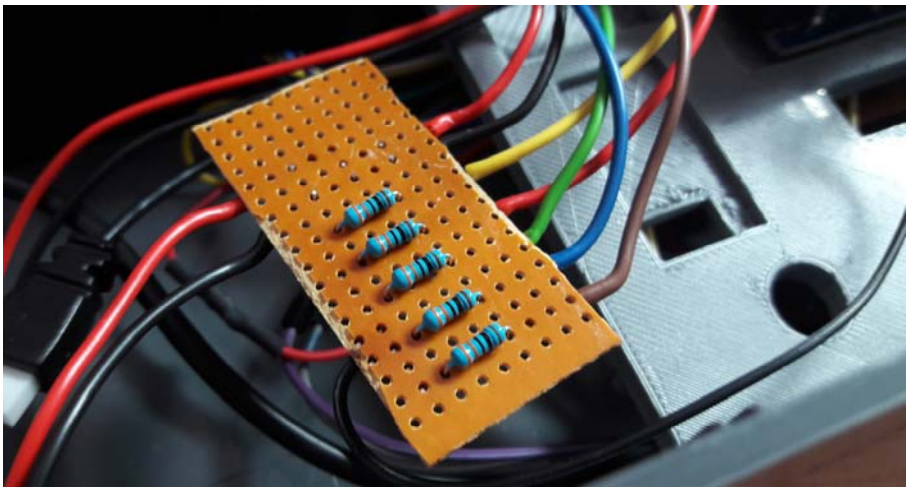


Figura 27. PCB del emisor.

La construcción del emisor siguió las siguientes etapas:

1. Impresión en 3D del soporte interno y del soporte el panel solar.
2. En cada PCB se fueron soldando los cinco LEDs que usaremos como sensores y en serie con resistencias de 330 Ω. Incluimos en esa PCB el regulador HT7305 y dos condensadores de 10 μF
3. Colocación de los LEDs en la parte superior del emisor dentro de agujeros equidistantes entre los colores rojo, verde y azul.

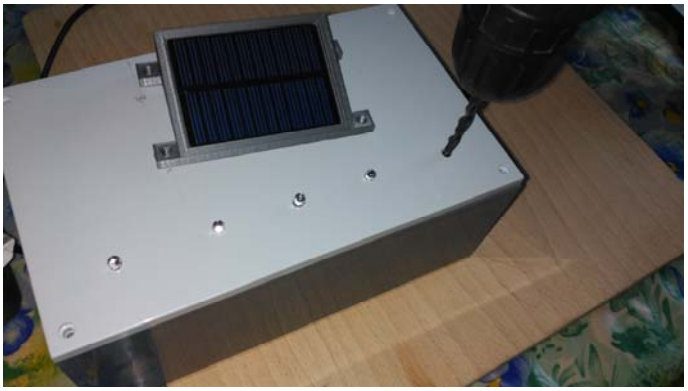


Figura 28. Realización de agujeros.

4. Fijación del panel solar a su soporte usando cola caliente. Después, fue taladrada y atornillada a la parte superior del emisor.

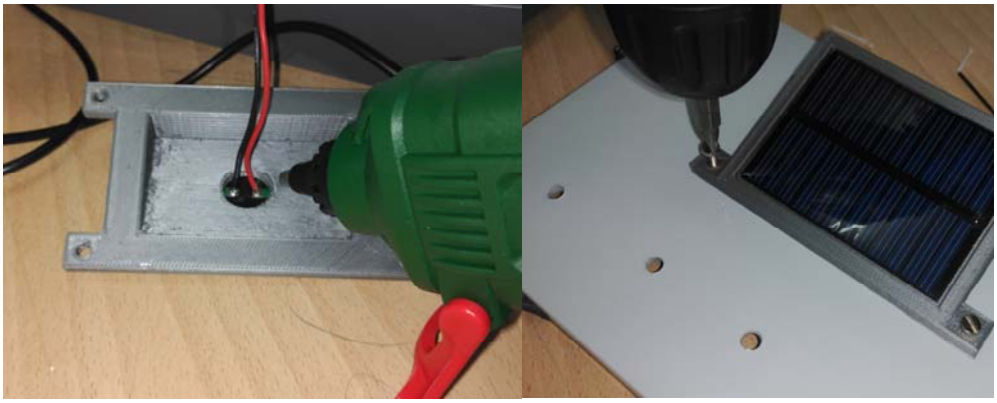


Figura 29. Implementación del panel solar

5. Unión de todos los componentes electrónicos y ensamblaje dentro de la caja del emisor. Se realizaron agujeros para la salida de la sonda y el sensor de temperatura además del conector para la antena usando pasamuros como elemento embellecedor.

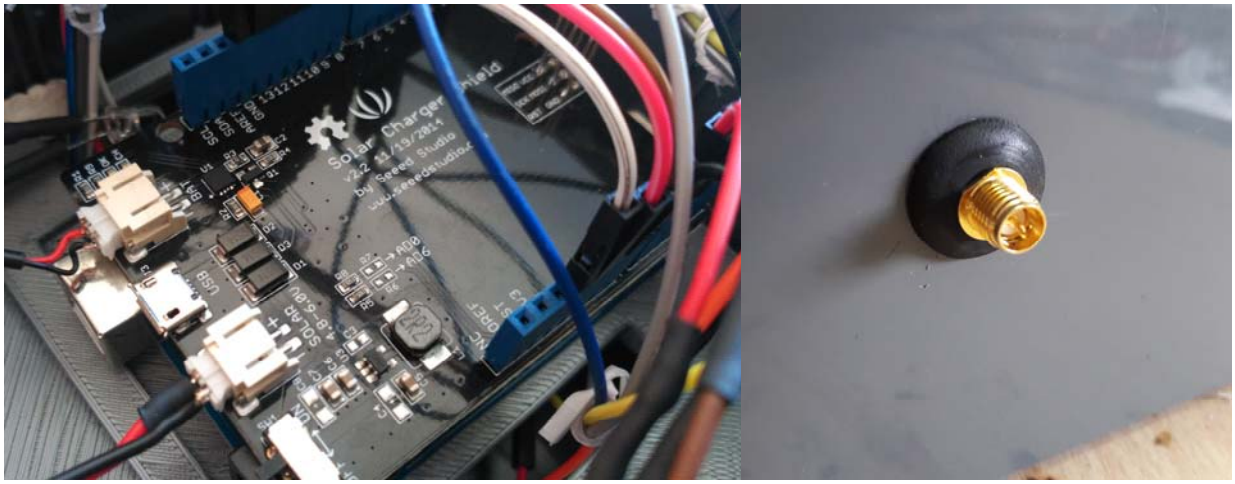


Figura 30. Conexiones del emisor.

3.1.8 Receptor

El receptor es el dispositivo encargado de recolectar la información enviada por el emisor vía radiofrecuencia y procesarla. El software que incorpora se encarga de reconocer las condiciones de luz en el lugar del emisor, a partir de los datos recogidos, mediante un algoritmo que reconoce valores y rangos típicos de cada condición.

Principalmente está compuesto por una PCB que integra el microcontrolador siendo el empleado un Arduino Pro Mini, el módulo de transmisión NRF24L01 y un display OLED de 0.96''. En adaptaciones posteriores se cambió el módulo de transmisión por la versión que incluía antena.

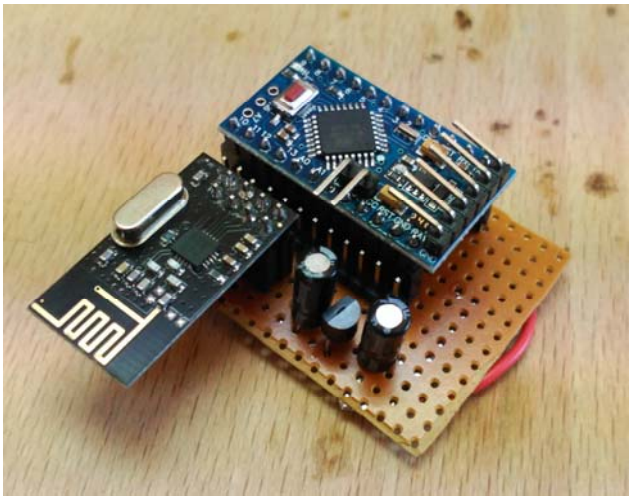


Figura 31. Circuito del receptor.

A partir de una caja de plástico ABS, se realizaron tres orificios por los que tendremos acceso al USB con el chip CP2102 para la comunicación USART – USB para recibir la información, otro para el display OLED para ver lo que estamos recibiendo y el último para poder conectar la antena.

Figura 32. Caja del receptor.

El resultado final es un dispositivo receptor con antena aérea. Contiene el display OLED donde podremos ver diversos parámetros que son enviados desde el emisor como es la temperatura, humedad, condición de luz y errores de producirse. Cuenta con un USB para poder conectarlo al ordenador y mediante el software LabVIEW realizar el procedimiento de adquisición de datos.

Figura 33. Receptor.

3.2. Software

3.2.1 Emisor

El código que se utiliza para el emisor está basado en funciones que realizaran los trabajos de adquisición de datos de condiciones de luz, humedad del suelo, transmisión de datos, modo de ahorro energético, etc.

Principalmente está divido en las siguientes partes:

- Librerías y declaración de variables

Lo primero de todo es incluir al programa todas las librerías necesarias para controlar los dispositivos o añadir funcionalidades al programa.

```
#include <OneWire.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <avr/sleep.h>
#include <avr/power.h>
```

La librería One Wire es necesaria para la obtención del valor de temperatura del sensor DS18B20, el cual transmite los datos por un único cable.

La librería SPI se encarga de todas las funciones necesarias para la comunicación con un dispositivo vía SPI. Se denomina interfaz de serie para periféricos y consiste en una transmisión de datos síncrona de tres hilos punto a punto. Permite la transferencia de datos síncrona a alta velocidad entre el microcontrolador y dispositivos periféricos.

Las librerías NRF24L01 y RF24 son necesarias para poder utilizar los módulos de transmisión por radiofrecuencia. Estas librerías nos aportan una amplia lista de funciones que nos permiten controlar estos módulos de forma rápida y sencilla, sin necesidad de entrar en materia de manipulación de registros.

Las librerías avr/sleep y avr/power se encargan de la gestión de la energía y de los modos para poner el microcontrolador en estado “off”. Podemos desde habilitar y deshabilitar el modo durmiente, seleccionar el modo y poner en modo “off” la CPU. La librería avr/power contiene una serie de registros de reducción de energía que permiten reducir el consumo habilitando o deshabilitando varios periféricos presentes en el microcontrolador. Además, podemos variar el registro que da valor a la prescala del reloj de cuarzo para reducir su consumo.

Lo siguiente será definir una serie de variables y definiciones necesarias para el correcto funcionamiento del software. Para el módulo de radio debemos de definir los pines CE y CSN, aparte de los pines de la comunicación SPI, para la transmisión de datos. También se define la dirección del canal al que se va a transmitir que tiene que ser igual en el

receptor para que haya comunicación directa entre ambos. Finalmente, para el sensor de temperatura hay que definir en qué pin del microcontrolador se encuentra.

```
//Módulo de radio
RF24 radio(9, 10);  //RF24 radio(CE_PIN, CSN_PIN)
const uint64_t ID[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

//Sensor de temperatura
OneWire  ds(7);
```

Para concluir, se definen las variables que serán necesarias para el código. Es muy importante dimensionar correctamente cada variable para poder declararla de un tipo u otro (byte, int, float,…). Con esto conseguimos ahorrar memoria dinámica, evitar posibles errores en cuanto a conflicto de tipos de datos y perder información numérica.

```
//Variables FDR
float datos[6], celsius;
byte i, present = 0, data[12], addr[8];
int pc; //Si lo quito de aqui pc es 0

//Variables LEDs
int Rojo[12], Verde[12], Azul[12], IR[12], UV[12], Muestra[12];
int r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11;

//Otros
int seq = 0;
```

- Condiciones de luz usando LEDs como sensores

La parte de obtención de datos a partir de lecturas usando diodos LEDs como sensores para conocer la condición de luz es un pilar de este trabajo. Lo que hacemos es tomar una serie de 11 lecturas de niveles de luz cada cierto tiempo de cada uno de los 5 LEDs. Realizaremos un bucle "for" para cada uno de ellos. Dentro de éste se tomarán las 11 medidas espaciadas un tiempo definido y luego serán almacenadas en el array correspondiente a ese LED.

Antes de empezar las series de cada canal, la patilla de entrada analógica se configura como salida digital a 0 y se hace una lectura simulada. Esto es necesario para descargar el LED y evitar interferencia en los datos por presencia de corrientes.

```
//Para inicializar las señales
pinMode(14 + x, OUTPUT);
digitalWrite(14 + x, LOW);
delay(3);
Muestra[0] = analogRead(x);
```

La toma de medidas es la parte más crítica de todo el funcionamiento del emisor. Estas microintensidades son tan pequeñas que son sensibles a muchos factores que puedan producir ruidos electromagnéticos y acarrear problemas en la precisión de los datos. Las primeras cuatro lecturas se toman en sucesiones inmediatas con las interrupciones del

microcontrolador deshabilitadas. Como este dispositivo es capaz de enviar varios miles de conversiones ADC por segundo, las cuatro primeras lecturas se toman en menos de 1 ms. Los retrasos incrementados de 1 ms y 2 ms se aplican para las lecturas 5º y 6º. Después de un retraso de 7 ms se toma la séptima medida y se vuelve a habilitar las interrupciones. Finalmente, se aplican tiempos de 20, 70, 200 y 700 ms para la 8º, 9º, 10º y 11º medida.

```
//Toma de medidas
noInterrupts();
pinMode(14 + x, INPUT);

Muestra[0] = analogRead(x);
Muestra[1] = analogRead(x);
Muestra[2] = analogRead(x);
Muestra[3] = analogRead(x);

delay(1);
Muestra[4] = analogRead(x);

delay(2);
Muestra[5] = analogRead(x);

delay(7);
Muestra[6] = analogRead(x);

interrupts();

delay(20);
Muestra[7] = analogRead(x);

delay(70);
Muestra[8] = analogRead(x);

delay(200);
Muestra[9] = analogRead(x);

delay(700);
Muestra[10] = analogRead(x);
```

Por último, según el LED que haya sido usado para tomar medidas, se almacenan esos datos del array "Muestra[11]" al array correspondiente a ese LED. Para el rojo sería el siguiente:

```
if (x == 0)
    {
      for (int i = 0; i < 12; i++)
      {
        Rojo[i] = Muestra[i];
      }
      Rojo[11] = 1;
    }
```

El comando `Rojo[11] = 1;` es un método el cual asigna al último término de cada array de las lecturas un valor entre 1 y 5 que usaremos en el receptor para que sepamos diferenciar a que LED pertenece la cadena de valores que estamos recibiendo.

- Obtención de los parámetros usando la Reflectometría en el Dominio de la Frecuencia.

Como mencionamos anteriormente, se emplean dos métodos eléctricos para determinar la capacitancia efectiva de la sonda que se encuentra en nuestro suelo a monitorizar. El primero consiste en el uso de la sonda como el componente capacitivo de un filtro de paso bajo. El ATmega328P genera una onda cuadrática de frecuencia de 65535 Hz que pasa por este filtro. Luego es amortiguada a través del amplificador y es pasada por un circuito detector de picos. Primero seleccionamos la referencia analógica por defecto para la conversión analógica-digital que es de 1.1 V. Posteriormente, hacemos un par de lecturas del canal ADC para estabilizar la señal. Definimos el pin por el que enviaremos la señal cuadrada. Con la función "tone()" activamos la generación de una onda cuadrada con un 50% de ciclo de trabajo y una frecuencia determinada, que será la máxima permitida por la función, es decir, 65535 Hz. Después de un periodo de estabilización leeremos el voltaje a la salida del primer amplificador y después del circuito detector de picos. Durante las lecturas del primer método, la segunda salida del amplificador se fuerza a 0 V por el microcontrolador en la patilla de salida D4 de control.

```
analogReference(DEFAULT);
datos[2] = analogRead(5);
datos[2] = analogRead(5);

//FILTRO PASO BAJO

pinMode(3, OUTPUT);
tone(3, 65535);
delay(300);
datos[2] = analogRead(5);
noTone(3);
```

El segundo método usa la sonda como el elemento capacitivo de un circuito oscilador, cargando y descargando repetidamente el condensador cuando la diferencia de potencial pasa entre dos entradas de control. El tiempo empleado por la diferencia de potencial para subir y bajar entre esas entradas se mide en la patilla D5 sobre cuatro ciclos de media onda para suministrar un indicador del valor de la capacitancia. El tiempo requerido para cargar y descargar la sonda FDR capacitiva se determina por su valor de capacitancia, que a su vez depende de la cantidad de agua en el área inmediatamente alrededor de la sonda.

Primero, ponemos el circuito del primer método en alta impedancia para que no influya en el segundo método. Alimentamos con 5 V DC en una pata del amplificador para el

circuito oscilador, que durante un periodo de estabilización pasará a tomar medidas del tiempo en μs que permanece en estado alto o bajo durante 4 semiciclos de 1 milisegundo.

```
//CIRCUITO OSCILADOR
pinMode(3, INPUT);
digitalWrite(4, HIGH);
delay(50);

noInterrupts();
pc = pulseIn(5, HIGH, 1000);
pc = pc + pulseIn(5, LOW, 1000);
pc = pc + pulseIn(5, HIGH, 1000);
pc = pc + pulseIn(5, LOW, 1000);
interrupts();

digitalWrite(4, LOW); // Fin de oscilación
```

- Voltaje de alimentación

Es posible conocer el voltaje de alimentación de nuestro microcontrolador a partir de la lectura del voltaje interno de referencia. Antes vamos a explicar que es y cómo funciona la conversión analógica-digital.

Trabajar con señales analógicas en un sistema electrónico es complicado. Hay que tener presente que una señal analógica que suba de 1 a 2 voltios en un segundo tiene infinitos niveles de tensión distintos en infinitos instantes de tiempo. Sin embargo, cualquier sistema informático tiene una memoria finita. Por tanto, habrá que guardar valores solo en algunos instantes de tiempo (muestrear) y aproximar su nivel a unos valores predefinidos (cuantificar). Así al final esta señal se transformará en una secuencia finita de números enteros dentro de un rango.

Un convertidor analógico-digital (ADC) es un dispositivo que se utiliza para convertir una señal analógica en una digital. Para ello primero se muestrean y luego se cuantifica la señal. El tiempo entre la adquisición de una muestra y otra es el periodo de muestreo. El rango de valores permitidos depende del número de bits que ofrece el convertidor.

El convertidor tiene dos señales de referencia, V+ y V-. La salida valdrá 0 cuando la entrada esté al nivel de V- y valdrá lo máximo cuando la entrada valga V+. Existe la posibilidad de seleccionar una referencia interna (en los ATmega328 es 1.1 V y en los ATmega8 es 2.56 V) o externa. La referencia externa proviene de la señal AREF y no debería sobrepasar a la de alimentación (Lajara-Vizcaíno & Sebastià, 2014).

Arduino dispone de 6 entradas para conversión analógico-digital que van desde los pines A0 a A5. Este tipo de conversión tiene una resolución de 10 bits, o lo que es lo mismo, 2^{10} = 1024 valores diferentes. La resolución de la lectura puede ir de 0 (0V) a 1023(5V) suponiendo cada incremento un aumento en la tensión de ±0.00488 V.

Figure 24-1. Analog to Digital Converter Block Schematic Operation,

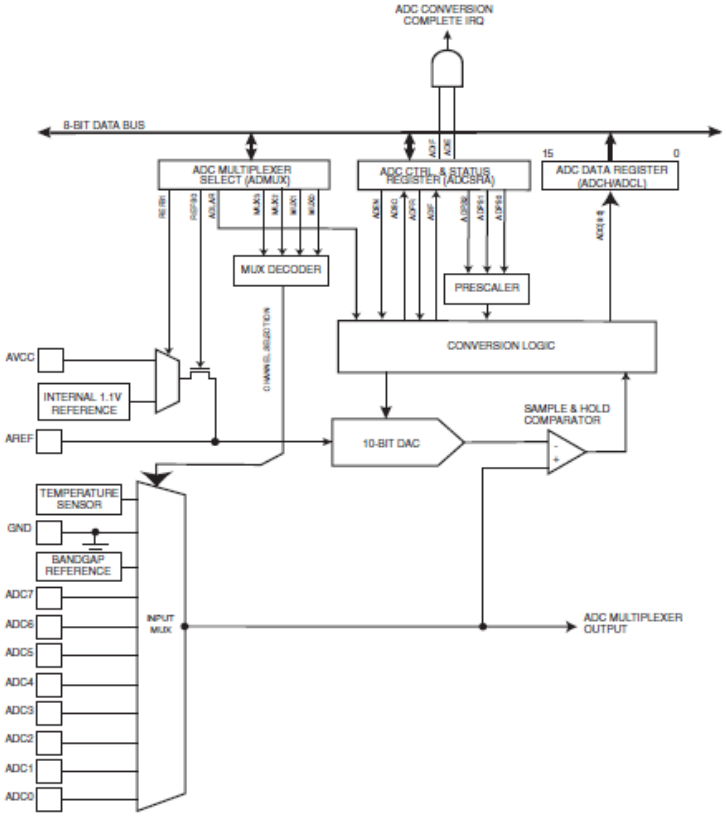


Figura 34. Diagrama del convertidor analógico-digital.

Otra utilidad que nos puede proporcionar un convertidor ADC es conocer el valor de la tensión de alimentación. Pueden darse casos que las baterías que alimentan al microcontrolador se descarguen hasta cierto punto que no consigan suministrar energía correctamente a todos los sistemas del ATmega328P. Por tanto, para saber que tensión de alimentación tenemos, vamos a realizar una conversión ADC con el voltaje de referencia interna que es conocido (Atmel, 2015).

Por ser un ATmega328P habilitamos los siguientes bits del registro ADMUX que manipula el multiplexor del ADC:

```
ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
```

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 35. Registro ADMUX.

Con esto podemos seleccionar que la alimentación del convertidor ADC sea 5 V y que la entrada analógica que leeremos para la conversión ADC no será ningún pin desde A0 a A5, sino que la entrada sea la tensión de referencia o 1.1 V.

```
ADCSRA |= _BV(ADSC); //Hacemos una conversión para inicializar
while (bit_is_set(ADCSRA,ADSC));
ADCSRA |= _BV(ADSC);                  //Iniciamos la conversión
while (bit_is_set(ADCSRA,ADSC));      //Midiendo
```

Ahora pasamos a realizar una conversión "falsa" ADC. Con el registro ADCSRA podemos controlar y conocer el estado del convertidor, que nos será útil para habilitar el inicio de la conversión de la señal de 1.1 V a digital.

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 36. Registro ADCSRA.

Mientras el bit ADSC del registro ADCSRA siga en 1, la conversión seguirá todavía realizándose siendo necesaria una segunda conversión ya que la primera se utiliza para inicializar el convertidor ADC.

```
uint8_t low = ADCL;
uint8_t high = ADCH;
long result = (high<<8) | low;
```

Como el resultado obtenido en una conversión puede ser de hasta 10 bits, el resultado será devuelto en dos bytes (ADCL y ADCH) ambos de 8 bits cada uno. La lectura debe de ser primer al registro ADCL que luego habilitará la lectura del registro ADCH.

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Figura 37. Registro ADCH y ADCL.

La expresión que relaciona los bits obtenidos por la conversión y las tensiones es:

$$\frac{V_{in}}{Vcc} = \frac{ADC}{1023}$$

Para este caso, la tensión de lectura o V_{in} es 1.1 V y el valor de ADC obtenido es 219.

Despejando la expresión anterior:

$$Vcc\ (mV) = \frac{V_{in}\ (mV) * 1023}{ADC} = \frac{1.1 * 1000 * 1023}{ADC} = \frac{1125300}{ADC} = 5138\ mV$$

Por tanto, la tensión de alimentación de este caso será aproximadamente 5.14 V. Realizamos un caso práctico para contrastar el valor obtenido por la conversión ADC con el valor proporcionado con un voltímetro con display para puerto USB de la casa Adafruit y con una resolución de 10 mV. El resultado fue el mostrado en la imagen:

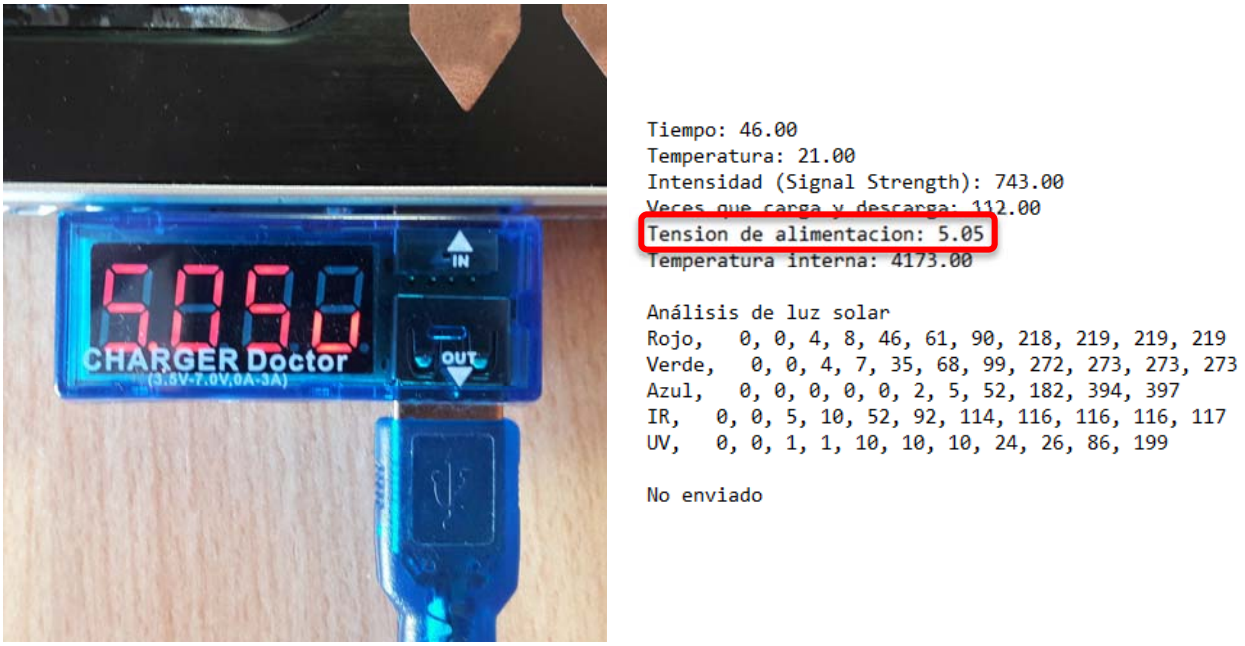


Figura 38. Test de voltaje.

Queda reflejada la gran precisión del valor de la tensión de alimentación mediante la conversión analógica a digital. Con ello podemos definir diferentes sistemas de alarma cuando el valor baja a valores críticos y es necesario cambiar dichas baterías, teniendo en cuenta que los periodos de empleo son muy largos y cualquier escenario es posible.

- "Sleep mode" o modo durmiente

Uno de los pilares básicos de nuestro proyecto es que sea capaz de operar durante largos periodos de tiempo para poder monitorizar las variables que deseamos. Es por ello que no siempre es necesario que el emisor esté consumiendo energía en aquellos momentos que ni toma datos, envía o realiza otras funciones. Para reducir el consumo de un microcontrolador hay algunos consejos, muchos de los cuales pueden parecer obvios: apagar todo aquel periférico que no se utilice, bajar la frecuencia de reloj, reducir la tensión de alimentación, no activar los pull-up si no son necesarios, poner los pines a entradas cuando no se utilicen, etc.

Para poder reducir el consumo de energía en periodos de inactividad existe un modo dentro del microcontrolador ATmega328P que es el "sleep mode" o modo durmiente. Si la CPU está dormida, el microcontrolador no ejecutará instrucciones, aunque conserva el estado de sus registros y memoria de datos. Cuando el microcontrolador está dormido necesitará ser despertado para volver a tener actividad. Despertar al mismo solo lo podrá hacer un periférico que esté activado (Lajara-Vizcaíno & Sebastià, 2014).

El microcontrolador ATmega328P tiene varios modos de dormido. En cada modo puede haber unos dispositivos activados y la CPU solo podrá ser despertada por los periféricos que no estén desactivados.

	Active Clock Domains					Oscillators		Wake-up Sources							
	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Figura 39. Sleep modes.

Como muestra la tabla, según el modo seleccionado tendremos una serie de Relojes activados y otros desactivados, la fuente de oscilación y las fuentes para despertar del modo durmiente. Es fácil observar que cuantos más periféricos desactivemos menor será el consumo, pero a veces no es lo ideal. Sino que conviene buscar un equilibrio entre consumo y mínima operatividad del microcontrolador porque cuantas más cosas sean apagadas más restringido se vuelve el método para despertar del "sleep mode".

Además, hay varias posibilidades a considerar cuando queremos minimizar el consumo de energía en un microcontrolador con sistema AVR. En general, los modos durmientes deben ser usados cuando sea posible y deben ser seleccionado para que la funcionalidad del dispositivo siga operativa. Todas las funcionalidades que no necesitemos deben de ser deshabilitadas. En particular, los siguientes periféricos necesitan una consideración especial cuando conseguimos alcanzar el nivel más bajo de consumo energético. Ellos son el convertidor ADC, que debe de ser deshabilitado antes de entrar en modo durmiente; el comparador analógico debe ser deshabilitado en cualquier caso en el que no sea utilizado; el Watchdog timer debe ser apagado si no es necesario en la aplicación porque en todos los modos durmientes posibles está activado y para modos en los que se busca un mínimo consumo, este timer puede ser el que se lleve el mayor consumo energético; los puertos para pines deben ser controlados para evitar que presenten cargas resistivas, en los modos durmientes son desactivados los relojes que gobiernan las entradas/salidas y la conversión ADC que supondrá que no haya consumo energético en las entradas (Atmel, 2015).

La implementación del "Sleep mode" en este proyecto ha sido usando el Watchdog Timer como sistema para despertar la CPU periódicamente. Este "temporizador Watchdog" o "Watchdog Timer" (WDT) es un temporizador que cuenta ciclos a partir de un oscilador independiente y dentro del encapsulado de 128 kHz. Provoca una

interrupción o un reseteo del sistema cuando el contador llega a un valor predefinido. En el modo de operación habitual, el Watchdog es utilizado para iniciar cuentas y reiniciarlas cuando finalizan, pero cuando el sistema no reinicia la cuenta del Watchdog, se resetea el sistema debido a que éste se ha quedado en algún bucle.

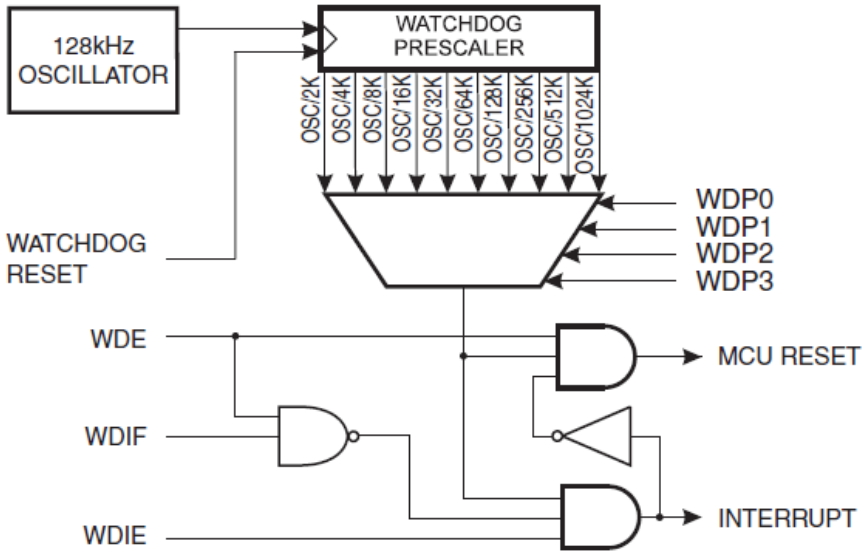


Figura 40. Diagrama del timer Watchdog.

En el modo de interrupción, el WDT produce una interrupción cuando su cuenta finaliza. Esta interrupción puede ser usada para despertar de modos durmientes y también como un timer más a parte de los tres ya integrados. Un ejemplo es limitar el tiempo máximo permitido por ciertas operaciones, produciendo una interrupción cuando la operación ha empleado más tiempo del definido. En el modo de reseteo del sistema, el Watchdog resetea el microcontrolador cuando el tiempo finaliza. Esto es muy típico en los sistemas para prevenir casos de cuando el sistema se queda colgado. El último modo es el que combina las interrupciones con el reseteo del sistema en el que primero provoca una interrupción y luego se resetea el microcontrolador. Esto nos permite realizar ciertas operaciones críticas como guardar ciertos parámetros críticos antes del reset (Atmel, 2015).

Ahora pasamos a definir el código. Lo primero es declarar una serie de tipos de datos llamados "wdt_prescalar_e". "Enum" son variables numéricas que solo pueden tener unos valores predefinidos. Internamente serían variables numéricas, pero de cara al programador se pueden asignar valores textuales, lo cual ayuda a entender mejor el código. Con la función "typedef" no generamos una nueva variable, sino un nuevo tipo. Mediante él se pueden ampliar los tipos de datos proporcionados por Arduino con nuevos tipos creados. Así podemos seleccionar el tiempo de desbordamiento del timer del Watchdog que provocará la salida del modo durmiente .

```
typedef enum { wdt_16ms = 0, wdt_32ms, wdt_64ms, wdt_128ms,
wdt_250ms, wdt_500ms, wdt_1s, wdt_2s, wdt_4s, wdt_8s}
wdt_prescalar_e;
```

Posteriormente definiremos una serie de variables para definir el tiempo que permanecerá la CPU en modo durmiente.

```
const short sleep_cycles_per_transmission = 2;

volatile short sleep_cycles_remaining = sleep_cycles_per_transmission;
```

Ahora haremos uso de dos funciones, la primera se encargará de manipular los registros en función del tiempo de desbordamiento para el Watchdog que hemos seleccionado. Primero seleccionamos la prescala que tiene que ser un número del 0 al 9 que corresponde la posición con "wdt_ prescalar_e". Es decir, 0 para wdt_16ms, 1 para wdt_32ms y así hasta 9 para wdt_8s. Luego es almacenada en una variable.

```
void setup_watchdog(uint8_t prescalar)
{
  prescalar = min(9, prescalar);
  uint8_t wdtcsr = prescalar & 7;
  if ( prescalar & 8 )
    wdtcsr |= _BV(WDP3);

  MCUSR &= ~_BV(WDRF);
  WDTCSR = _BV(WDCE) | _BV(WDE);
  WDTCSR = _BV(WDCE) | wdtcsr | _BV(WDIE);
}
```

El registro WDTCSR pertenece al registro de control del Watchdog. Se encarga de modificar los modos de operación, habilitar las interrupciones y seleccionar la prescala para configurar el tiempo que tarde en desbordarse (Atmel, 2015).

Bit	7	6	5	4	3	2	1	0	
(0x60)	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

Figura 41. Registro WDTCSR.

Para usar el Watchdog como método para salir de los "sleep modes" tendremos que configurarlo para que produzca una interrupción cuando se desborde el contador. Para ello hay que poner a cero el bit WDE y a uno el bit WDIE. Cuando éste es escrito a uno, habilitamos las interrupciones del Watchdog. Si el bit WDE es puesto a cero cuando el

WDTON(1)	WDE	WDIE	Mode	Action on Time-out
1	0	0	Stopped	None
1	0	1	Interrupt Mode	Interrupt
1	1	0	System Reset Mode	Reset
1	1	1	Interrupt and System Reset Mode	Interrupt, then go to System Reset Mode
0	x	x	System Reset Mode	Reset

Figura 42. Bits de control del Watchdog.

bit WDIE es puesto a uno, habilitamos el modo de interrupción sabiendo que el fusible WDTON no está programado y por tanto está a uno.

El bit WDE es compartido por el bit WDRF en el registro MCUSR. Esto quiere decir que el bit WDE está siempre a uno cuando el bit WDRF también lo está. Para poner a cero el bit WDE, antes debe ser puesto a cero el bit WDRF.

Para introducir el tiempo que tarde en desbordarse debemos de incluirlo en los bits WDP3, WDP2, WDP1 y WDP0. Pero antes debemos de activar el bit que se encarga de habilitar cambios, el bit WDCE. Este valor volverá a ser cero después de cuatro ciclos de reloj. Por tanto, debemos poner en estado uno los bits WDP3 y WDP0 para conseguir unos 8 segundos de tiempo hasta el desbordamiento del contador (Atmel, 2015).

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	16ms
0	0	0	1	4K (4096) cycles	32ms
0	0	1	0	8K (8192) cycles	64ms
0	0	1	1	16K (16384) cycles	0.125 s
0	1	0	0	32K (32768) cycles	0.25 s
0	1	0	1	64K (65536) cycles	0.5 s
0	1	1	0	128K (131072) cycles	1.0 s
0	1	1	1	256K (262144) cycles	2.0 s
1	0	0	0	512K (524288) cycles	4.0 s
1	0	0	1	1024K (1048576) cycles	8.0 s

Figura 43. Bits de control del Watchdog II.

Dentro de la función setup() ejecutaremos `setup_watchdog(wdt_8s);` para configurar el desbordamiento del Watchdog cada 8 segundos.

La segunda función se encargará de ejecutar el "sleep mode".

```
void do_sleep(void)
{
```

```
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  sleep_enable();

  sleep_mode();

  sleep_disable();
}
```

Primero seleccionamos el modo en función de las necesidades de ahorro energético y las posibilidades que nos interesen para salir de ese estado durmiente. El que mantiene todos los relojes desactivados y asegura un mínimo consumo es el modo "Power-down". Luego pasamos a habilitar el modo y con sleep_mode() entramos directamente en ese estado de mínimo consumo.

Dentro de ese modo se apagarán las unidades y periféricos del modo elegido y finalizará cuando recibamos una interrupción del Watchdog. Cuando eso ocurra, lo primero que hará será ejecutar el comando siguiente al previo que lo mete en el "sleep mode". Este comando deshabilita el modo durmiente y continua la ejecución habitual del programa.

La ejecución de estas funciones va al final de la función loop(), pero para conseguir periodos más largos en los que permanece en estado de mínimo consumo fue conseguido mediante la siguiente estructura de repetición:

```
while (sleep_cycles_remaining > 0) {
  do_sleep();
}
sleep_cycles_remaining = sleep_cycles_per_transmission;
```

La función do_sleep() se irá ejecutando mientras sleep_cycle_remaining sea mayor que cero. Lo interesante de esto es que podemos conseguir intervalos de tiempo que sean múltiplos de 8 segundos en los cuales el emisor permanecerá en el modo durmiente. El valor de sleep_cycles_remaining irá decrementando cada vez que se produzca una interrupción fruto de que el timer del Watchdog se desborda. De esto se encarga la función de interrupción ISR() con el vector correspondiente al Watchdog.

```
ISR(WDT_vect)
{
  --sleep_cycles_remaining;
}
```

- Transferencia de datos usando los módulos NRF24L01

Como ya hemos visto antes definiendo los pines CE y CSN y el canal al que se va a transmitir, debemos de inicializar el módulo NRF24L01 y abrir el canal de escritura mediante las siguientes funciones en la función setup().

```
radio.begin();
radio.openWritingPipe(ID[0]);
```

Luego, definimos una función que se encargará de enviar los 6 arrays de lecturas de luz de los cinco LEDs más uno extra para diferentes parámetros.

```
void envio_datos() {

  radio.stopListening();
  radio.write(Rojo, sizeof(Rojo));
  radio.write(Verde, sizeof(Verde));
  radio.write(Azul, sizeof(Azul));
  radio.write(IR, sizeof(IR));
  radio.write(UV, sizeof(UV));
  radio.write(datos, sizeof(datos));
  boolean ok = radio.write(datos, sizeof(datos));

  radio.startListening();
  if (ok) {
    Serial.println("Enviado");
  }
  else {
    Serial.println("No enviado");
  }
  Serial.println();
}
```

Lo primero es salir del estado de escucha para enviar la información, array por array hasta los seis. Como la función radio.write() devuelve un booleano, podemos saber si la información ha podido ser enviada o no.

3.2.2 Receptor

El código que se utiliza para el receptor está basado en funciones que realizarán los trabajos de recepción de datos desde el emisor, su manipulación para poder ser transmitidos vía comunicación USART y mostrar información en el display.

Principalmente está divido en las siguientes partes:

- Librerías y declaración de variables

Inicialmente incluimos en el programa las librerías que contienen las funciones que utilizaremos para poder manipular los dispositivos incluidos en el receptor. Estos son el módulo de transmisión por radiofrecuencia NRF24L01 y el display SSD1306 OLED 0.96''.

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

Como habíamos comentado anteriormente, la librería SPI se encarga de todas las funciones necesarias para la comunicación con un dispositivo via SPI. Se denomina interfaz de serie para periféricos y consiste en una transmisión de datos síncrona de tres hilos punto a punto. Permite la transferencia de datos síncrona a alta velocidad entre el microcontrolador y dispositivos periféricos.

Las librerías NRF24L01 y RF24 son necesarias para poder utilizar los módulos de transmisión por radiofrecuencia. Estas librerías nos aportan una amplia lista de funciones que nos permiten controlar estos módulos de forma rápida y sencilla, sin necesidad de entrar en materia de manipulación de registros.

La librería Wire se encarga de las comunicaciones I2C/TWI. La comunicación "Inter Integrated Circuit" (I2C) está basada en un bus con dos líneas bidireccionales, una para reloj (SCK) y otra para datos (SDA), permitiendo la interconexión de hasta 128 dispositivos diferentes. Cada uno de los dispositivos conectados al bus tienen una dirección única, esta dirección se utiliza para seleccionar un dispositivo en concreto del bus. La comunicación consistirá en que un dispositivo direcciona a otro, con lo que los siguientes eventos del bus solo atenderá ese dispositivo. Con un dispositivo direccionado se pueden enviar comandos o acceder a sus registros para leerlos o escribirlos.

Las librerías Adafruit_GFX y Adafruit_SSD1306 se encarga del uso del display SSD1306 OLED 0.96''. Las funciones que nos brindan estas librerías nos permiten escribir texto de diferentes tamaños y colores (blanco o negro), mover el texto dentro del display, realizar efectos con imágenes, etc. Ello nos quita una ardua tarea que sería ir encendiendo LED a LED del display hasta conseguir lo que queramos.

Lo siguiente será definir una serie de variables y definiciones necesarias para el correcto funcionamiento del software. Para el módulo de radio debemos de definir los pines CE y CSN, aparte de los pines de la comunicación SPI, para la transmisión de datos. También se define la dirección del canal al que se va a transmitir que tiene que ser igual en el emisor para que haya comunicación directa entre ambos. Finalmente, para el display definiremos un pin para el reseteo de la pantalla y se lo asignaremos a ella.

```
//Display
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

//nRF24L01
RF24 radio(9, 10);
const uint64_t ID[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };
```

Ahora pasamos a la definición de las variables que serán necesarias para el código. Es muy importante dimensionar correctamente cada variable para poder declararla de un tipo u otro (byte, int, float,…). Con esto conseguimos ahorrar memoria dinámica, evitar posibles errores en cuanto a conflicto de tipos de datos y perder información numérica.

```
//Variables
int Rojo[12], Verde[12], Azul[12], IR[12], UV[12];
float datos[6] , secuencia_OLD;
int OLD_IR , OLD_Verde;
boolean Sol_directo = false, Cielo_azul = false,
Ligera_contaminacion = false, Amanecer = false, Anochecer =
false;
boolean Nubes_ligeras = false, Nubes_densas = false,
Sombra_pobre = false, Noche = false, Neblina = false, flag =
false;
String datosTFG;
```

Lo primero de todo es definir los arrays en los cuales almacenaremos la información que recibamos vía 2.4 GHz. Es por ello que el tipo de dato del array debe ser igual al tipo de dato del array usado para tomar esos valores en el emisor ya que, por ejemplo, la temperatura es de tipo "float" porque no son valores enteros y debe ser almacenada en un array de tipo "float".

Crearemos una serie de variables booleanas para reconocer, a posteriori del procesamiento de los datos, el tipo de condición de luz solar existente. Con ello haremos mostrar por pantalla esa información además de temperatura, humedad y detección de errores.

La parte fundamental del receptor es el almacenamiento de la información en un string. Los strings son cadenas de caracteres en las que podemos ir añadiendo información y posteriormente imprimir toda esa información. No tiene una capacidad determinada, sino que se puede ir almacenando información a placer.

- Función setup()

La primera función que se ejecuta al activar la alimentación del Arduino es la función setup(). Esta solo se ejecuta una vez y al arranque para realizar los comandos que vamos a explicar.

```
void setup()
{
  //Módulo de radio
  radio.begin();
  radio.openReadingPipe(1, ID[0]);
  radio.startListening();

  //Display
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.clearDisplay();
  display.drawBitmap(30, 0, UPCT_logo_bmp, 64, 32, 1);
  display.display();
  delay(1000);

  Serial.begin(57600);
}
```

Como ya hemos visto antes definiendo los pines CE y CSN y el canal al que se va a transmitir, debemos de inicializar el módulo NRF24L01 y abrir el canal de lectura. Después de ello ponemos el módulo en modo "escucha" para que permanezca atento a la recepción de información.

Para el display OLED, inicializamos el mismo y le indicamos la dirección del dispositivo ya que usamos comunicación I2C y es necesario para ser llamado a recibir o transmitir. Limpiamos el buffer y mostramos por pantalla una imagen del logotipo de la universidad, más adelante explicaremos como fue obtenido.

La función para mostrar el logotipo es la siguiente:

```
drawBitmap(int16_t x, int16_t y, const uint8_t *bitmap, int16_t
w, int16_t h, uint16_t color)
```

Los parámetros X e Y representan la posición de la imagen dentro del display. El siguiente parámetro es un puntero que señala una zona de la memoria donde almacenamos la imagen. El resto de parámetros representa la anchura y altura de la imagen por si queremos modificar sus dimensiones y finalmente el color, que se puede elegir entre el color del propio display o negro retroiluminado. Todas las dimensiones de alto y ancho son en bits.

- Lectura de los datos transmitidos

Una vez ejecutada la función setup(), se ejecutará de forma repetida e indefinida la función loop(). Ella está dividida en dos estructuras de repetición "if" en la que una dependerá de la otra.

Cuando la sentencia `if(radio.available())` se cumple significa que hay datos disponibles en el buffer para leer. Entonces realizaremos un proceso de lectura de cada array de información y de su almacenamiento en el string.

```
radio.read(Rojo, sizeof(Rojo));
if (Rojo[11] == 1) {
  datosTFG = String(Rojo[0]);
  datosTFG += '\t';
  for (int i = 1; i < 10; i++) {
      datosTFG += String(Rojo[i]);
      datosTFG += '\t';
  }
  datosTFG += String(Rojo[10]);
  datosTFG += '\n';
}
delay(10);
```

Primero leemos y almacenamos la trama de información en el array correspondiente al color del LED que ha sido usado como sensor además de coincidir el tipo de variable (byte, int, float,…). Para evitar problemas se añadió un término más al array que contenía un 1 para rojo, 2 para verde, 3 para azul, 4 para infrarrojo, 5 para ultravioleta y 6 para

datos. Con esto conseguimos asegurar y evitar problemas a la hora de que se almacenen valores de un LED en un array que no corresponde a su color.

La forma en la que almacenemos los datos en el string será crucial para poder procesar esa información en el software de programación gráfica LabVIEW. Por ello, la información debe ir almacenada dato a dato y con tabulación "\t" entre ambos y para finalizar una trama de valores hay que añadir un salto de línea "\n". Con ello conseguimos que el programa informático muestre los valores en una tabla en la que cada columna corresponde a una medida tomada y las filas al color del LED del que fueron tomadas. El "delay" de 10 ms nos permite dar margen para que el emisor sea capaz de transmitir toda la información antes de que el receptor la lea.

```
radio.read(datos, sizeof(datos));
    if (datos[5] == 6)
    {
      for (int i = 0; i < 5; i++) {
        datosTFG += String(datos[i]);
        datosTFG += '\t';
      }
      datosTFG += String(float(datosTFG.length()));
      datosTFG += '\r';
      Serial.print(datosTFG);
      flag = true;
    }
    delay(10);
```

En la última trama leída, la que contiene los datos relativos al dispositivo FDR y otros valores, después de incluir todos los valores en el string introducimos un retorno de carro "\r". Este carácter es interpretado por LabVIEW como el fin de la transmisión y muestra los datos obtenidos. Después de ello, enviamos ese string por USART y habilitamos un booleano el cual nos permitirá procesar esa información y manipular el display.

- Algoritmo para el procesamiento de las medidas para reconocer la condición de luz exterior.

A partir de las lecturas IR L1, IR L11, Rojo L1, Rojo L4, Rojo L11, Verde L4, Verde L7, Verde L11, Azul L4, y UV L7 fue posible realizar una serie de reglas para clasificar fácilmente cuales de 20 diferentes posibilidades de condiciones de luz solar es la real. Entre ellas eran: sin luna, luz lunar, ligera contaminación, anochecer, amanecer, poca luz, nubes densas, sombra, nubes ligeras, niebla, cielo azul y sol directo. Como era de esperar, la distinción entre amanecer, anochecer, luz lunar y ligera contaminación mostró la complejidad para definir las condiciones de luz en función de los parámetros.

Direct sun/Poor light	Dawn/Dusk ...	Dark	Cloudy
(IR1 > 50) = DIRECT SUN (R4 < 20) = goto dark ... (R1 > IR1 *0.8) AND (R1 > 15) = BLUE SKY (IR1 > R1) AND (R1 > 4) goto cloudy ... (B4 > IR1) AND (B4 > 10) = SHADE POOR LIGHT (IR1 > 50) = DIRECT SUN	(IR11 < 55) AND (IR11 > previous IR11) = DAWN DUSK	(G11 > 30) = goto dawn/dusk ... (R11 > 15) AND (G4 > 0) = LIGHT POL (R11 > 5) AND (R11 * 3 > IR11) = LIGHT POL (IR11 < 12) OR (previous IR11 < 12) = NO MOON MOON	(G4 > 35) = HAZE (G4 > 15) = LIGHT CLOUD HEAVY CLOUD

Figura 44. Condiciones de luz.

Partiendo de que el sistema es de bajo coste y que hay alta correlación entre los valores obtenidos y la luz ambiental, es altamente viable para ser implementado en áreas donde sea necesario su uso. Pero es importante destacar que en el presente se sigue trabajando para afinar la precisión entre los valores obtenidos y la condición real de luz ya que factores como nubosidad, contaminación y otros generan variaciones que hacen difícil la búsqueda de patrones de comportamiento en los valores.

La implementación de esta información en el software es crucial para que el propio receptor pueda procesar los datos y obtener una conclusión de la condición de luz. Ello fue diseñado con una agrupación de estructuras de decisión "if" en la que se registraba el valor anterior de IR L11 y el anterior de Verde L7 necesarios para reconocer la condición de luz.

```
if (IR[0] > 50) {
  Sol_directo = true;
}

if (Rojo[0] >= 15 && Rojo[0] > IR[0] * 0.8)
{
  Cielo_azul = true;
}

if (Rojo[3] < 20)
{
  if (Rojo[10] > 15 && Verde[3] > 0) {
    Ligera_contaminacion = true;
  }
  if (Rojo[10] > 5 && Rojo[10] * 3 > IR[10]) {
    Ligera_contaminacion = true;
  }
  if (IR[10] < 12 || OLD_IR < 12) {
    Noche = true;
  }
}

if (Rojo[3] < 20 && Verde[10] > 30) {
  if (IR[10] < 55 && IR[10] > OLD_IR) {
    if (Verde[6] > OLD_Verde) {
```

```
      Amanecer = true;
    }
  }
  if (Verde[6] > OLD_Verde) {
    Anochecer = true;
  }
}

if (IR[0] > Rojo[0] && Rojo[0] > 4)
{
  if (Verde[3] > 35 && IR[0] < 100) {
    Neblina = true;
  }
  if (Verde[3] > 15 && IR[0] < 100) {
    Nubes_ligeras = true;
    if (Azul[3] > IR[0] && Azul[3] > 10) {
      Nubes_ligeras = false;
      Nubes_densas = true;
    }
  }
}

if (Azul[3] > IR[0] && Azul[3] < 30)
{
  Sombra_pobre = true;
}

OLD_IR = IR[10];
OLD_Verde = Verde[6];
```

Respecto a las publicaciones existentes, el algoritmo implementado fue modificado para solventar la ambigüedad que causaban ciertos parámetros en los que confluían en dos conclusiones iguales. En otros casos los parámetros fueron modificados ligeramente para reajustar los márgenes correctos.

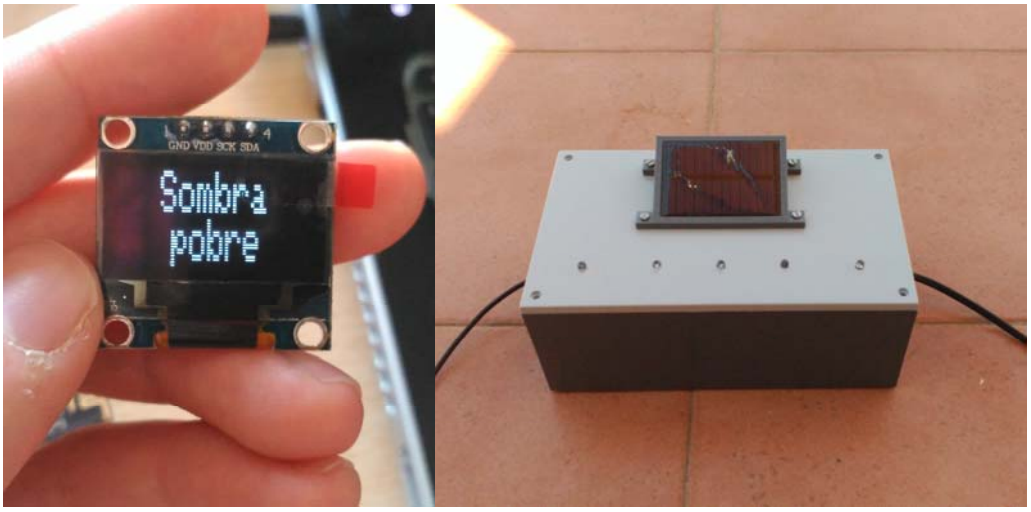


Figura 45. Test de condiciones de luz.

- Logotipo UPCT en el display

Cuando queremos mostrar algo por el display tenemos dos opciones, o hacer uso de las funciones que se encargan de manipular los registros para que se muestre la información que queremos y como la queremos o podemos manipular directamente esos registros y encender cada LED del display uno a uno hasta obtener lo que queramos. Esta ardua tarea es compleja y larga, por lo tanto, para conseguir mostrar más allá de texto existe la opción de mostrar imágenes. Y para evitar lo mencionado antes, existen diferentes programas que nos permiten transformar imágenes en una secuencia de valores hexadecimal que representa un mapa de bits en formato texto.

Para el proyecto se utilizó un software llamado "LCDAssistant". Es un software libre para la conversión de una imagen en formato Mapa de Bit en arrays de bytes, igual que los registros que manipulan la activación de los LEDs del display

Figura 46. LCD Assistant.

Primero convertimos la imagen a un formato llamado Mapa de Bits Monocromático usando algún software gráfico. Esta imagen es subida al programa en el que introducimos la orientación de los bytes para el código y las dimensiones de alto y ancho.

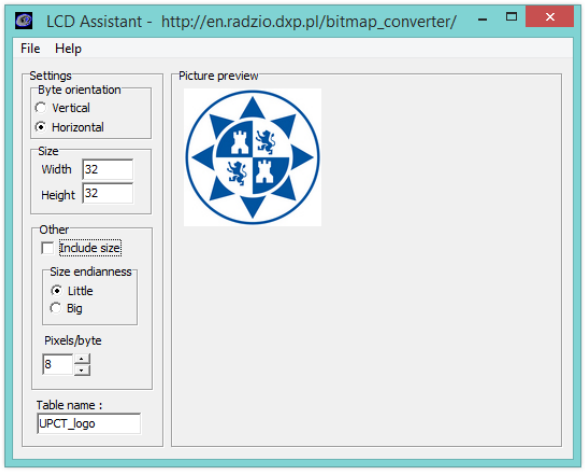


Figura 47. Interfaz LCD Assistant.

Con esta configuración, el programa nos devuelve un archivo que podemos abrir con el "Blog de Notas".

```
const unsigned char PROGMEM UPCT_logo_bmp [] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x00, 0x00, 0xE0, 0x00, 0x00,
  0x00, 0x00, 0x60, 0x01, 0x80, 0x06, 0x00, 0x00, 0x00, 0x06, 0x00, 0x03, 0xC0, 0x00, 0x60, 0x00,
  0x00, 0x18, 0x00, 0x07, 0xE0, 0x00, 0x18, 0x00, 0x00, 0x40, 0x00, 0x0F, 0xF0, 0x00, 0x02, 0x00,
  0x01, 0x8E, 0x00, 0x1C, 0x78, 0x00, 0x71, 0x80, 0x02, 0x07, 0xFC, 0x00, 0x00, 0x7F, 0xE0, 0x40,
  0x04, 0x07, 0xF0, 0x7F, 0x00, 0x0F, 0xE0, 0x20, 0x08, 0x03, 0xC7, 0xFF, 0x00, 0x03, 0xC0, 0x10,
  0x10, 0x03, 0x9E, 0x17, 0x27, 0x01, 0x80, 0x08, 0x20, 0x01, 0x3E, 0x07, 0x1E, 0x00, 0x00, 0x04,
  0x20, 0x00, 0x7F, 0x07, 0x07, 0x00, 0x00, 0x04, 0x40, 0x04, 0xFC, 0x03, 0x0F, 0x80, 0x20, 0x02,
  0x40, 0x79, 0xFC, 0x63, 0x00, 0x80, 0x1E, 0x02, 0x43, 0xF9, 0xFF, 0xFF, 0x00, 0x00, 0x1F, 0xC2,
  0x43, 0xF8, 0x00, 0x00, 0xFB, 0xFF, 0x9F, 0xC2, 0x40, 0x78, 0x04, 0xC0, 0xC0, 0x7F, 0x9E, 0x02,
  0x40, 0x04, 0x02, 0xC0, 0xE0, 0xFF, 0x20, 0x02, 0x20, 0x00, 0x01, 0xC0, 0xE0, 0xFE, 0x00, 0x04,
  0x20, 0x00, 0x01, 0x30, 0xC0, 0x3C, 0x00, 0x04, 0x10, 0x01, 0x80, 0x60, 0x84, 0x39, 0xC0, 0x08,
  0x08, 0x03, 0xC0, 0x00, 0xFF, 0xE3, 0xC0, 0x10, 0x04, 0x07, 0xF0, 0x00, 0xFE, 0x0F, 0xE0, 0x20,
  0x02, 0x07, 0xFE, 0x00, 0x00, 0x3F, 0xE0, 0x40, 0x01, 0x8E, 0x00, 0x1C, 0x18, 0x00, 0x71, 0x80,
  0x00, 0x40, 0x00, 0x0F, 0xF0, 0x00, 0x02, 0x00, 0x00, 0x18, 0x00, 0x07, 0xE0, 0x00, 0x18, 0x00,
  0x00, 0x06, 0x00, 0x03, 0xC0, 0x00, 0x60, 0x00, 0x00, 0x00, 0x60, 0x01, 0x80, 0x07, 0x00, 0x00,
  0x00, 0x00, 0x07, 0x00, 0x00, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
```

Esta imagen, al convertirse en arrays de bytes, es declarada como una constante de carácter "unsigned char". A efectos de ahorrar memoria dinámica y evitar problemas en la estabilidad del sistema, esta información será guardada en la memoria Flash junto con todo el programa y para ello hay que añadir el comando "PROGMEM" en la línea de declaración de la constante.

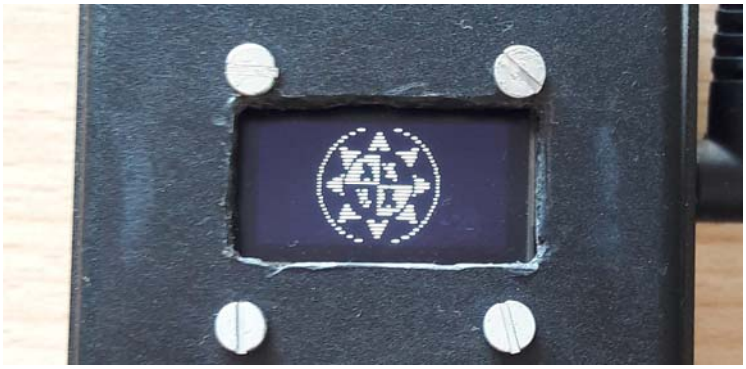


Figura 48. Logo UPCT en display.

- Manipulación del display SSD1306

Para poder mostrar por pantalla las condiciones de luz y humedad que han sido recogidas por el emisor y procesadas por el receptor debemos de utilizar las funciones proporcionadas por las librerías de Adafruit.

Según la condición solar que se ha analizado a partir de los datos, se activa un booleano que luego permitirá la ejecución de ciertos comandos para imprimir la información del análisis de datos por pantalla. Veamos un ejemplo.

```
if (Sombra_pobre) {
  sombra_pobre_display();
  Sombra_pobre = false;
  delay(1000);
}
```

Cuando los datos llevan a la conclusión de que la condición de luz es sombra pobre, se mostrará esta información por pantalla. Estos comandos están incluidos en la función

loop donde hay hasta once estructuras de decisión diferentes para las conclusiones del algoritmo a partir de los datos.

Para simplificar la comprensión del software del receptor se han creado once funciones que serán las que muestren la información por pantalla, como es el caso de `sombra_pobre_display();`

Esa función está compuesta por lo siguiente:

```
void sombra_pobre_display() {
  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(28, 0);
  display.setTextColor(WHITE);

  display.println("Sombra");

  display.setTextSize(2);
  display.setCursor(35, 16);
  display.setTextColor(WHITE);

  display.print("pobre");

  display.display();

}
```

Lo primero que hacemos es limpiar el display de información. Luego seleccionamos el tamaño de texto y la posición del cursor como referencia para que escriba. Cuando el texto requiere de una segunda línea, desplazamos el cursor hacia abajo además de hacia la derecha.

El resto de funciones son similares y varían en si requieren dos o una línea y por ende la posición del cursor según lo mencionado y la longitud de las palabras.

Figura 49. Condición de luz en display.

También incluimos funciones para mostrar en el display los valores de humedad y temperatura. La función para este último es como las anteriores, imprimiendo en el display el valor del array "Datos" correspondiente a la temperatura y seguido de "^C".

En cuanto a la sonda de humedad, no podemos concluir valores correctos de humedad. Esto es debido a que este subobjetivo fue iniciado una vez avanzado el proyecto y por falta de medios y tiempo se dejó la calibración, compensación de la temperatura y eliminación de ruidos para trabajos futuros. Estos problemas serán vistos en unos breves experimentos en el capítulo de Resultados.

Para no dejar esta parte vacía, se mostrará un valor ponderado a partir de los parámetros obtenidos. Para obtener un valor provisional pero no concluyente se realizó una interpolación de los parámetros de la técnica FDR usando como valores extremos los obtenidos en agua y aire, cuando esta al aire (0% de humedad) y cuando está en el agua (100% de humedad).

Obteniendo de un muestreo y realizando la media de los valores obtenidos, los parámetros que usaremos como márgenes son:

Lectura	Aire (0%)	Agua (100%)
Signal Strength	740	557
Tc/Td	112	1000

Tabla 9. Valores límite FDR.

Entonces, obtendremos dos expresiones para obtener un porcentaje de humedad provisional a partir de "Signal Strength" y "Tc/Td".

```
float SS_porcentaje = (-((datos[2] - 740.0) * 100.0)) / 183.0;
float tc_porcentaje = ((datos[3] - 112.0) * 100.0) / 888.0;
float hum_media = (SS_porcentaje + tc_porcentaje) / 2.0;
```

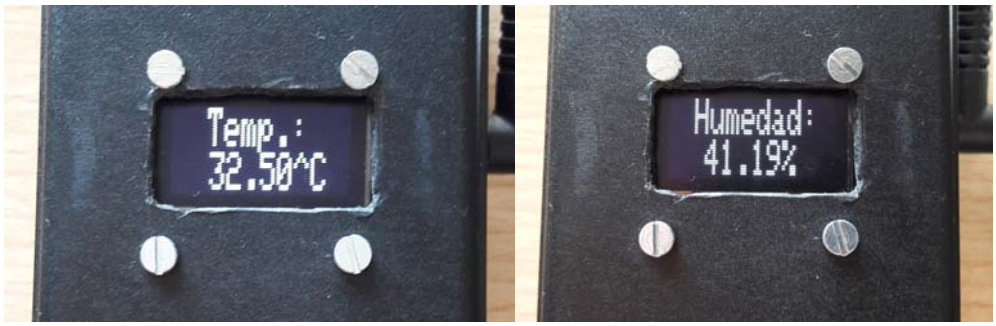


Figura 50. Parámetros en display.

3.3. Adquisición de datos con LabVIEW

3.3.1 Introducción a LabVIEW

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) es una plataforma y entorno de desarrollo para diseñar sistemas, con un lenguaje de programación visual gráfico. Recomendado para sistemas hardware y software de pruebas, control y diseño,

simulado o real y embebido, pues acelera la productividad. El lenguaje que usa se llama lenguaje G, donde la G simboliza que es lenguaje Gráfico.

Este programa fue creado por National Instruments (1976) para funcionar sobre máquinas MAC, salió al mercado por primera vez en 1986. Ahora está disponible para las plataformas Windows, UNIX, MAC y GNU/Linux. Los programas desarrollados con LabVIEW se llaman Instrumentos Virtuales, o VIs, y su origen provenía del control de instrumentos, aunque hoy en día se ha expandido ampliamente no sólo al control de todo tipo de electrónica (Instrumentación electrónica) sino también a su programación embebida, comunicaciones, matemáticas, etc.

Un lema tradicional de LabVIEW es: "La potencia está en el Software", que con la aparición de los sistemas multinúcleo se ha hecho aún más potente. Entre sus objetivos están el reducir el tiempo de desarrollo de aplicaciones de todo tipo (no sólo en ámbitos de Pruebas, Control y Diseño) y el permitir la entrada a la informática a profesionales de cualquier otro campo. LabVIEW consigue combinarse con todo tipo de software y hardware, tanto del propio fabricante (tarjetas de adquisición de datos, PAC, Visión, instrumentos y otro Hardware) como de otros fabricantes.

Como se ha dicho es una herramienta gráfica de programación, esto significa que los programas no se escriben, sino que se dibujan, facilitando su comprensión. Al tener ya pre-diseñados una gran cantidad de bloques, se le facilita al usuario la creación del proyecto, con lo cual en vez de estar una gran cantidad de tiempo en programar un dispositivo/bloque, se le permite invertir mucho menos tiempo y dedicarse un poco más en la interfaz gráfica y la interacción con el usuario final. Cada VI consta de dos partes diferenciadas:

- Panel Frontal: Es la interfaz con el usuario, la utilizamos para interactuar con el usuario cuando el programa se está ejecutando. Los usuarios podrán observar los datos del programa actualizados en tiempo real (como van fluyendo los datos, un ejemplo sería una calculadora, donde tú le pones las entradas, y te pone el resultado en la salida). En esta interfaz se definen los controles (los usamos como entradas y pueden ser botones, marcadores, etc.) e indicadores (los usamos como salidas y pueden ser gráficas, indicadores, etc.).

- Diagrama de Bloques: es el programa propiamente dicho, donde se define su funcionalidad. Aquí se colocan íconos que realizan una determinada función y se interconectan. Suele haber una tercera parte llamada “icono/conector” que son los medios utilizados para conectar un VI con otros VIs.

En el panel frontal, encontraremos todo tipos de controles o indicadores, donde cada uno de estos elementos tiene asignado en el diagrama de bloques una terminal, es decir el usuario podrá diseñar un proyecto en el panel frontal con controles e indicadores, donde estos elementos serán las entradas y salidas que interactuarán con la terminal del VI. Podemos observar en el diagrama de bloques, todos los valores de los controles e indicadores, como van fluyendo entre ellos cuando se está ejecutando un programa VI.

3.3.2 Programa de adquisición de datos

Esta parte consiste en la realización de una aplicación o instrumento virtual (VI) para la recepción de los datos transmitidos en bloques de datos en modo texto por el receptor vía comunicación serie y que fueron transmitidos por el emisor desde el terreno.

Para la obtención de los datos haremos uso de un protocolo de comunicación Serie a través de VISA. VISA es un API de alto nivel utilizado para comunicarse con buses de instrumentación. Es independiente de la plataforma, del bus y del entorno. Como USB es un bus de comunicación basado en mensajes, significa que una PC y un dispositivo USB se comunican enviando comandos y datos a través del bus en forma de texto o datos binarios. Cada dispositivo USB tiene su propio conjunto de comandos. Se pueden utilizar funciones de Lectura y Escritura NI-VISA para enviar estos comandos a un instrumento y leer la respuesta del mismo.

A partir de la versión 3.0, NI-VISA utiliza comunicación por USB. Se pueden utilizar dos clases de recursos VISA: USB INTR y USB RAW: Los dispositivos USB que cumplen con el protocolo USB Test and Measurement Class (USBTMC) utilizan la clase de recursos USB INSTR. Los dispositivos USBTMC cumplen con un protocolo que la clase de recursos USB INSTR de VISA puede entender. No se necesita ninguna configuración para comunicarse con un dispositivo USBTMC. Los instrumentos USB RAW son todos aquellos de USB que no cumplen con la especificación USBTMC.

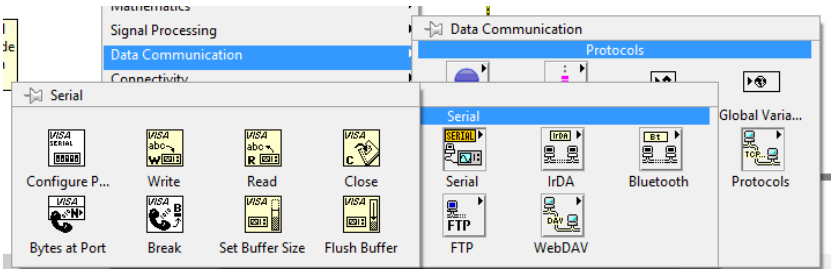


Figura 51. Menú comunicación Serie.

El menú de VISA para el protocolo Serie nos proporciona varios bloques con el que podemos configurar el puerto serie, escribir o leer, cerrar el puerto, leer en número de bytes en el puerto, interrumpir la transmisión, dimensionar el tamaño del buffer y su descarga.

La primera parte para el diseño del diagrama de bloques es insertar en bloque que configura el puerto serie.

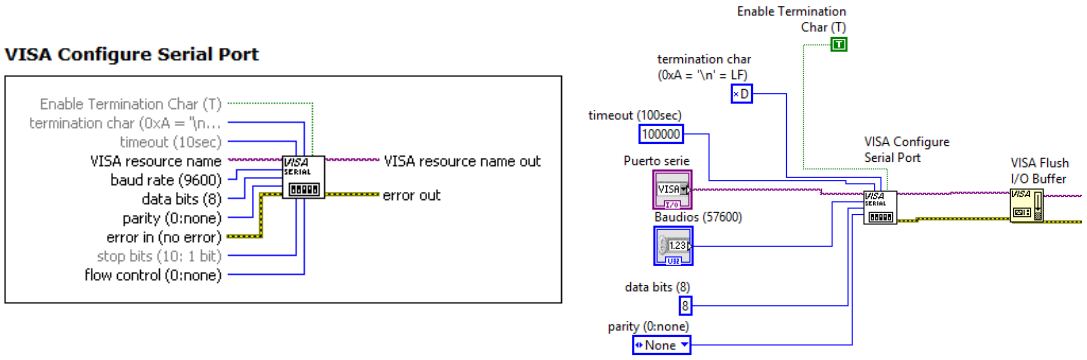


Figura 52. VISA Configure Serial Port.

En VISA Configure Serial Port.vi, introducimos un indicador en "VISA resource name" para poder seleccionar el puerto USB por donde estamos transmitiendo los datos. Luego añadimos una constante para introducir los baudios, es decir, la velocidad de transmisión de los datos que debe ser igual a los baudios introducidos en el programa de Arduino para la comunicación USART. Como en el presente caso no ha sido necesario manipular los registros que controlan la comunicación Serie para modificar el número de bits por trama, los bits de stop o paridad, debemos de incluir las condiciones por defecto que son 8 bits a transmitir y sin bit de paridad ya que el bit de stop va por defecto. Además se ha definido un tiempo de espera el cual finalizará el programa si no se recibe dato alguno en 100 segundos.

Una parte muy importante es la habilitación de un carácter de terminación. En la parte donde describíamos el software del receptor y construíamos el string que transmitiríamos se habló de finalizar la cadena de caracteres con un retorno de carro "\r". Habilitando la terminación por un carácter e indicando el retorno de carrera como tal en el bloque, conseguimos que el programa de LabVIEW reconozca tal carácter como el fin de la transmisión de la información para posteriormente poder manipular esa información. Además, usamos VISA Flush I/O Buffer.vi para borrar posibles caracteres que hubiera en el buffer de entrada.

La parte donde se leerá la información y se procesarán los datos estarán dentro de un bucle "While Loop". Ahí usaremos VISA Read.vi para leer los datos.

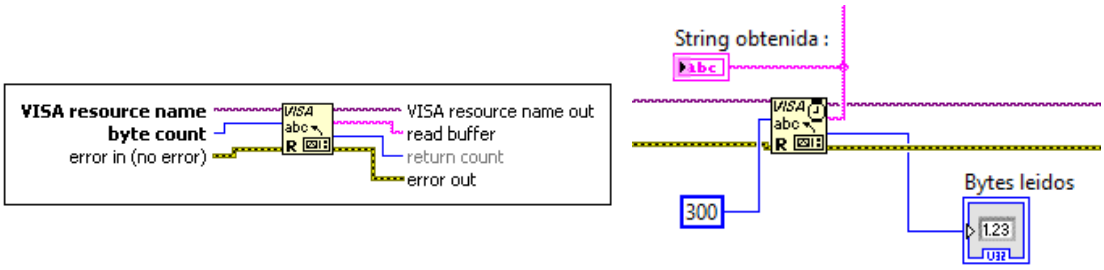


Figura 53. VISA Read.

Conectamos la salida de VISA resource name del bloque para la configuración del puerto con la entrada del bloque para la lectura e igualmente se hará para el bus de error. Creamos una constante para introducir los bytes que debe de leer, como ya introducimos un carácter de terminación de la trama, hay que añadir un número de bytes mayor al esperado recibir para así no perder información. En la salida del VISA Read.vi podemos conocer el número de bytes leídos y la información conseguida. Hay que destacar que la programación que estamos haciendo en el diagrama de bloques va enfocada a la recepción de un string, que es lo que enviamos desde el receptor. Por tanto, en "Read buffer" podremos poner un indicador donde leeremos toda la trama enviada. Con otros bloques podremos manipular esa información.

Recogida la trama de caracteres, la convertimos en un array de dos dimensiones con el bloque Spreadsheet String To Array.vi

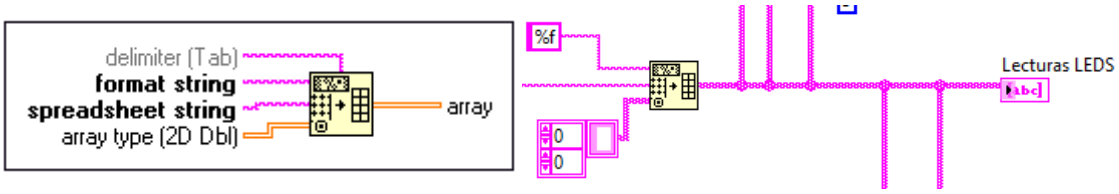


Figura 54. Spreadsheet String To Array.

Partimos de una entrada tipo string y obtenemos en la salida un array bidimensional. El formato del string lo definimos tipo "float" ya que no afecta a los valores numéricos de tipo "int" al ser enteros, pero si mantiene los decimales del array "Datos" que, como la temperatura, pueden contener. Para que este bloque convierta correctamente el string a un array de dos dimensiones hay que saber cómo indicar que cada valor va en una fila o columna en concreto. Para ello, debemos de modificar el string desde el receptor para que envíe todos esos datos pero separados con un carácter de tabulación "\t" para que el programa de LabVIEW lo reconozca como valores para columnas independientes. Cuando queramos pasar de la primera fila a la segunda tendremos que añadir el carácter de salto de línea "\n" que será interpretado por el programa como cambio de fila. Enviando los datos en string de esta forma conseguimos obtener la matriz bidimensional.

Los datos obtenidos de la medición usando los LEDs como sensores serán mostrados en una tabla dentro del Panel Frontal, pero para el resto de datos queda más gráfico usar indicadores que sean mostrados. Para poder usar estos indicadores debemos de seleccionar previamente el dato y convertirlo a valor numérico antes de mostrarlo.

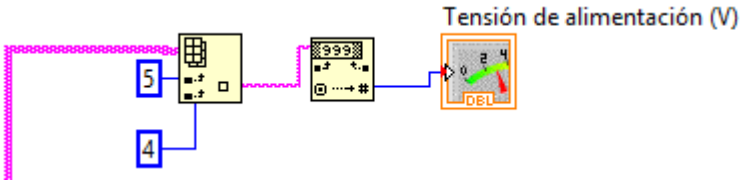


Figura 55. Conversión a número.

Para los cinco valores que mostramos en indicadores entre ellos la Tensión de alimentación usamos dos bloques para poder conseguirlo. El primero es Index Array Function.vi, es una funcionalidad dentro del menú Array el cual nos permite obtener un valor de dentro de un array. Para ello solo tenemos que introducir el valor de la fila y la columna del dato en cuestión.

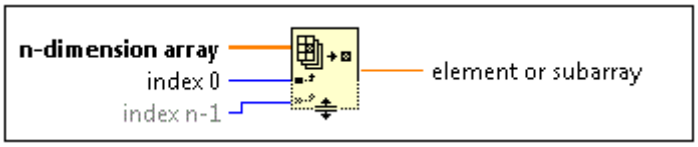


Figura 56. Index Array Function.

El segundo bloque es Decimal String To Number Function.vi, una funcionalidad del menú String que nos permite convertir un número en formato String a un número decimal.

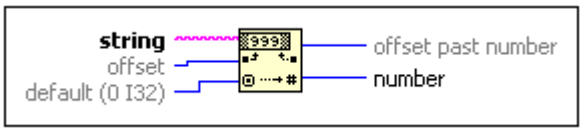


Figura 57. Decimal String To Number Function.

Finalmente, y con los bloques anteriores, podemos mostrar en indicadores los valores de tensión de alimentación, Signal Strength, Time to charge/discharge, temperatura y el número de la secuencia obtenida.

3.3.3 Panel frontal del programa

En el Panel frontal se ha diseñado la interfaz del Programa de adquisición de datos sobre condiciones lumínicas y parámetros de FDR que interactuará con el usuario. Está compuesto por la parte de configuración del puerto serie. En él podemos seleccionar el puerto serie/USB por donde estamos recibiendo la información, así como la velocidad de transmisión de datos o "Baudios". Además de que todo programa cuenta con su botón de stop para interrumpir el programa.

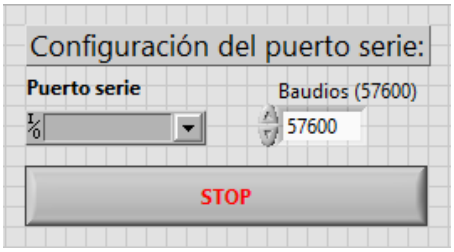


Figura 58. Configuración del puerto serie.

La segunda parte es la de exposición de los resultados obtenidos. Todos ellos serán expuestos en una tabla donde recogeremos las muestras de cada uno de los LEDs usados como sensores y los datos relativos a la técnica FDR, temperatura y demás.

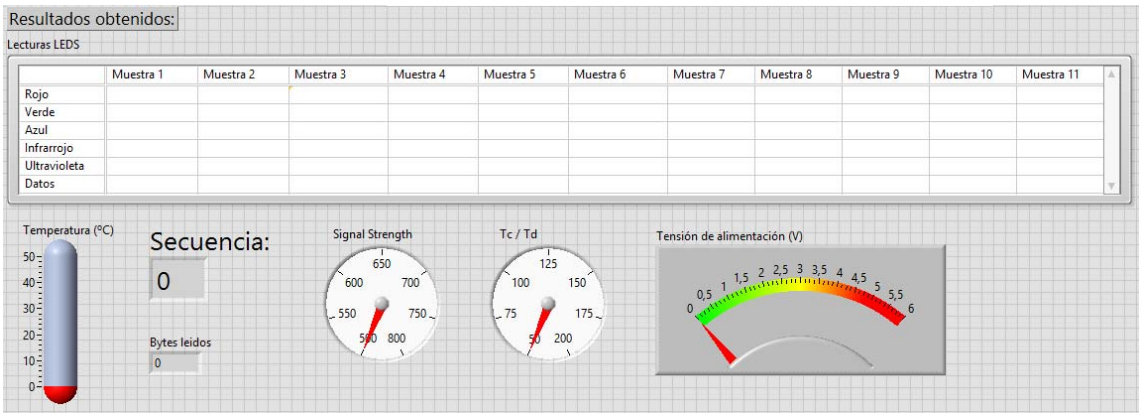


Figura 59. Panel de resultado.

Para un caso práctico, el emisor está enviando información al receptor y éste a su vez al programa vía USB. Indicando que el puerto serie será el "COM5" y una velocidad de 57600 baudios obtenemos lo siguiente:

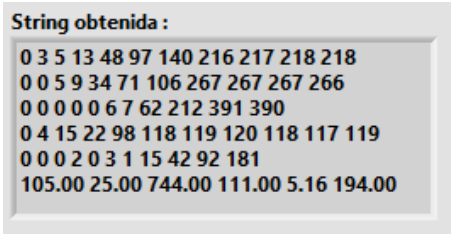


Figura 60. String recibida.

Esta string obtenida es convertida a un array bidimensional y ciertos valores mostrados en indicadores con lo que obtendremos una vista del Panel frontal como la siguiente:

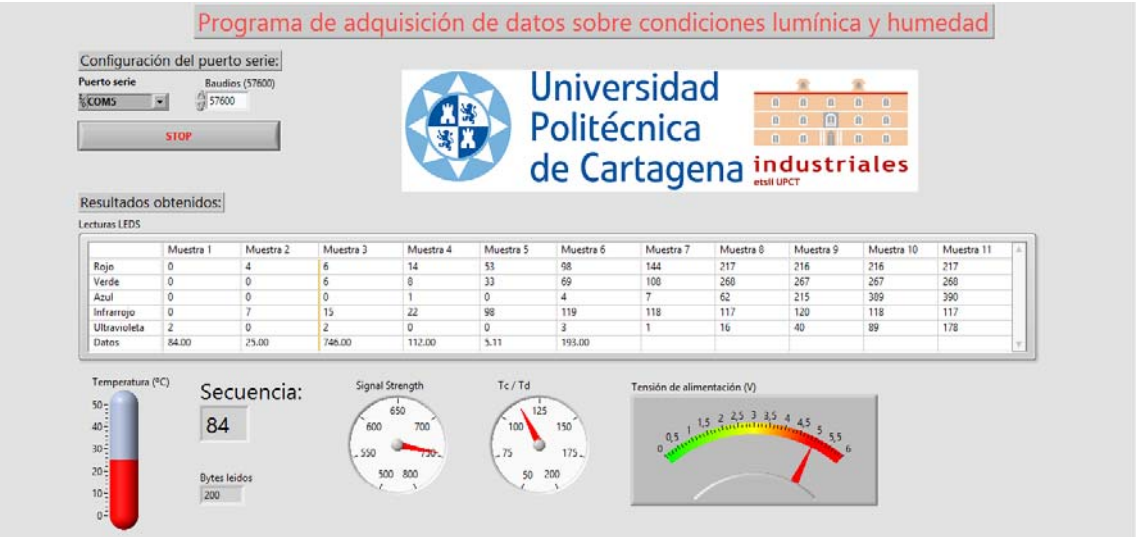


Figura 61. Vista del Panel Frontal.

Podemos ver mostrados los valores en la tabla y como varían según recibimos nuevos datos. Además, los indicadores de abajo nos dan una visión más intuitiva de ciertos parámetros que son controlados.

4

Resultados

Recopilación de los experimentos realizados basándonos en los LEDs como sensores para la obtención de las condiciones de luz y en las técnica de la Reflectometría en el Dominio de la Frecuencia.

4.1. Introducción

4.1.1 Adaptación de los dispositivos para almacenamiento masivo

Los dispositivos anteriormente expuestos fueron desarrollados para una monitorización en continuo de los parámetros recogidos por el emisor y transmitidos al receptor.

Figura 62. Emisor tomando medidas.

Para la exposición de los resultados y conclusiones a partir de los LEDs como sensores y del dispositivo FDR, se ha desarrollado una versión adaptada al fin de conseguir un almacenamiento masivo de la información en unidades de memorias no volátiles para su posterior procesamiento y análisis.

La forma más eficaz y económica para conseguir esto es la implementación de un módulo para SD con comunicación SPI que, haciendo uso de los mismos buses que utiliza el módulo de comunicaciones NRF24L01, podremos almacenar grandes cantidades de información de lecturas de luminosidad y parámetros de FDR durante largos periodos de tiempo.

Tanto al emisor como al receptor se le pueden incluir estos elementos para ese fin:

- Módulo SD

Un lector SD es un dispositivo que permite emplear como almacenamiento una tarjeta SD. Estas tarjetas y las micro SD se han convertido en un estándar, desplazando a otros medios de almacenamiento de datos debido a su gran capacidad y pequeño tamaño. Por este motivo han sido integradas en una gran cantidad de dispositivos, siendo en la actualidad componentes frecuentes en ordenadores, tablets y smartphones, entre otros.

Dentro del mundo de Arduino, es posible encontrar lectores de bajo coste tanto para tarjetas SD como micro SD. En ambos tipos de lectores, la lectura puede realizarse a

través de bus SPI. Aunque pueden disponer de otros interfaces, como bus I2C o USART, normalmente es preferible emplear SPI por su alta tasa de transferencia.

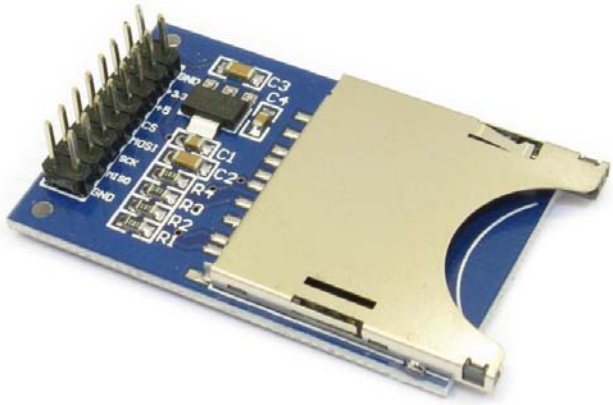


Figura 63. Módulo SD.

Respecto a las tarjetas empleadas, podemos emplear tarjetas SD o SDSC (Standard Capacity) o SDHC (High Capacity), pero no SDXC (Extended Capacity). Deberá estar formateada en sistema de archivos FAT16 o FAT32. La tensión de alimentación es de 3.3V, pero en la mayoría de los módulos se incorpora la electrónica necesaria para conectarlo de forma sencilla a Arduino, lo que frecuentemente incluye un regulador de voltaje que permite alimentar directamente a 5V.

Emplear una tarjeta SD o micro SD con Arduino tiene la ventaja de proporcionar una memoria casi ilimitada para nuestros proyectos. Además, es no volátil (es decir, persiste cuando se elimina la alimentación), y puede ser extraída y conectada a un ordenador con facilidad. La gran desventaja es que supone una importante carga de trabajo para Arduino. Sólo el programa ocupará el 40% de la memoria Flash, y casi el 50% de la memoria dinámica. El uso del procesador también es exigente.

En los proyectos se suelen emplear las tarjetas SD y micro SD, principalmente, en proyectos de tipo datalogger, es decir, para mantener el registro de las mediciones de un sensor. También pueden ser empleadas, por ejemplo, para llevar movimientos o rutas de robots precalculadas, y cargarlos bajo demanda.

La conexión es sencilla y similar tanto para lectores SD como Micro SD. Simplemente alimentamos el módulo desde Arduino mediante 5V y GND. Por otro lado, conectamos los pines del bus SPI a los correspondientes de Arduino.

- Pulsador

Uno de los problemas que pueden presentarse al quitar la tarjeta de memoria o apagar el sistema sin cerrar correctamente el archivo es generar un fichero corrupto el cual aparecerá vacío. Esto provocará que perdamos todos los datos que hemos estado almacenando y ser un gran problema.

```
if (digitalRead(PIN_SAVE) == PIN_SAVE_ACTIVE) {
```

```
      dataFile.close();
      save = true;

      digitalWrite(2, HIGH); //Zumbador
      delay(1000);
      digitalWrite(2, LOW);
    }
```

La solución fue usar un pulsador conectado a GND y a un pin de Arduino con las resistencias "Pull-Up" activadas para, cuando éste sea pulsado, cerrar correctamente el archivo para poder retirarlo con seguridad.

En muchas ocasiones un periférico conectado al microcontrolador tendrá una configuración conocida como colector abierto. Esta configuración hace que cuando el periférico quiere sacar un nivel bajo cortocircuitará el pin con la masa, por lo que tendrá 0V. Cuando el periférico quiere sacar un nivel alto simplemente desconecta el pin de masa, por lo que se queda "al aire". Como este es el caso, habría que forzar de algún modo un nivel de tensión alto cuando el pin esté en el aire. Esto se consigue con una configuración que se conoce como resistencia de "pull-up".

Las resistencias de "pull-up" se colocan en línea de datos, entre ellas y la alimentación. Se aprovechan del hecho que las entradas digitales apenas consumen corriente. Si no hay corriente por las resistencias cuando hay un nivel alto en la línea, no tendrán caída de tensión, por lo que la tensión en la línea será la de la alimentación y el microcontrolador leerá un nivel alto.

Como las resistencias de "pull-up" son un elemento común en los sistemas integrados, el microcontrolador ATmega328P las incorpora internamente y de un valor de 20 KΩ.

- Zumbador

El zumbador nos permite comprender que está ocurriendo en el código ya que no contamos con ningún display para ello. Según la frecuencia del pitido que emita podemos saber si ha sido reconocida la tarjeta y el archivo correctamente abierto (2 Hz) o si ha ocurrido algún error en su apertura (0.5 Hz) que generalmente suele ocurrir cuando nos olvidamos de introducir la tarjeta o el archivo esta corrupto, además de indicador para extraer la tarjeta de memoria después de cerrarse correctamente (1 Hz).

Desarrollando el código para el receptor con esta adaptación hemos llegado al límite de la memoria Flash y SRAM del ATmega328P ya que el espacio que ocupan las librerías que controlan las funciones del display OLED y las funciones del módulo para la SD ocupan demasiada memoria. Hemos llegado a alcanzar el uso del 95% de la memoria lo cual da problema de estabilidad y no es interesante. Por ello, en ese código, se opta por deshabilitar el display OLED. Igualmente ocurrió en el emisor, que tuvimos que deshabilitar las librerías y códigos del módulo NRF24L01.

4.2. Resultados usando LEDs como sensores

Para la recolección de datos, el emisor ha sido colocado en una zona abierta sin elementos que produzcan sombra y con una amplia visión del cielo. Los valores guardados en la memoria SD para su procesamiento son los siguientes: R1, R4, R11, G4, G7, G11, B4, B11, IR1, IR11 y UV7. Estas medidas han sido seleccionadas por ser las más representativas de las 55 lecturas recogidas (11 por LED), ya que explican mejor como varía el espectro luminoso recogido por los sensores según el estado meteorológico. A partir de esas medidas se obtuvieron la clasificación simple que vimos anteriormente de las condiciones lumínicas. Esta clasificación final incluyó los siguientes términos para determinar el grado de iluminación:

- Sin Luna
- Luz de Luna
- Ligeramente contaminado
- Anochecer
- Amanecer
- Luz pobre
- Muy nublado
- Ligeramente nublado
- Niebla
- Cielo azul
- Sol directo

Como se podía esperar, la diferencia entre amanecer, anochecer, luz de luna y ligeramente contaminado muestran una dificultad de diferenciación.

Para la representación de los valores obtenidos en gráficas y para una mejor visualización de la misma, representaremos los valores en porcentajes según el valor de saturación máximo para ese LED.

4.2.1 Experimento 1: Tarde y noche del 21/05 con nubosidad

El dispositivo se dejó conectado y tomando medidas desde las 17:25 del 21 de mayo hasta las 3:00 del 22 de mayo. Durante la tarde se veía en el cielo una nubosidad poco densa pero uniforme que cubría todo el cielo hasta entrada la noche.

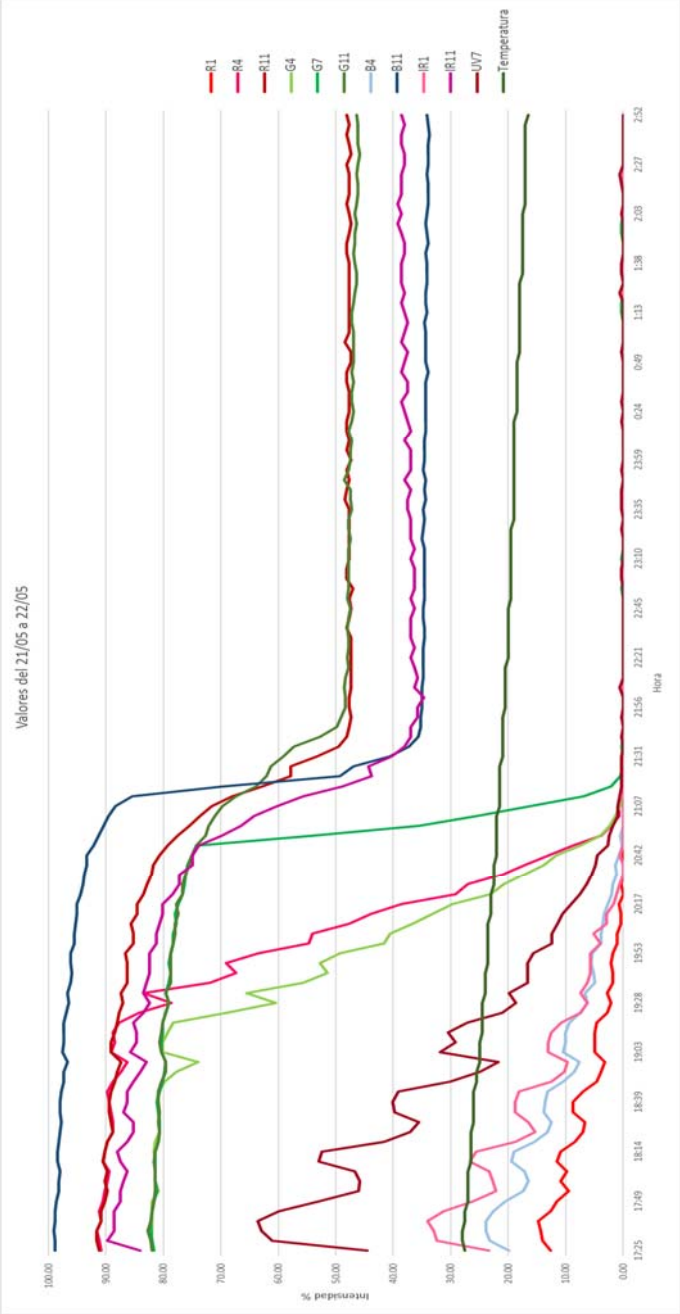


Figura 64. Experimento 1 LEDs.

Podemos observar como la temperatura de la tierra de la planta que se monitorizaba iba descendiendo según evolucionaba el día. Se aprecia que durante la tarde los valores fluctúan coincidiendo crestas y valles entre las distintas curvas de los diferentes valores además de presentar una tendencia decreciente según se acerca el ocaso.

Para el análisis de la tarde usaremos las siguientes gráficas:

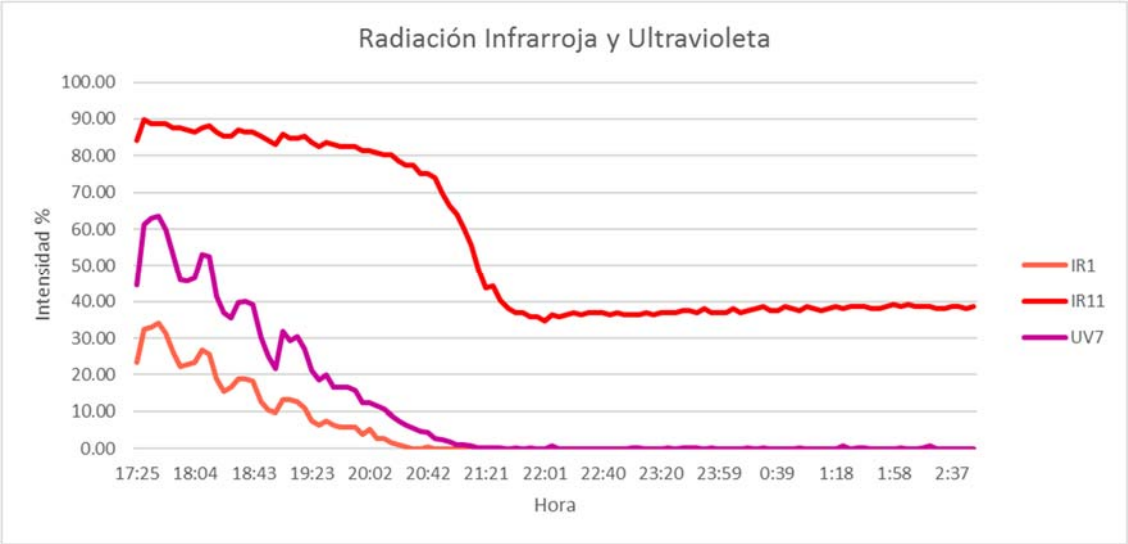


Figura 65. Experimento 1 LEDs IR UV.

En esta grafica se representan los valores de radiación infrarroja y ultravioleta durante el experimento. Vemos que IR1 y UV7 presentan los mismos valles y crestas además de IR11, pero tímidamente. Esto es porque las nubes son capaces de dispersar la radiación infrarroja y viendo que la curva IR1 tiene forma senoidal decreciente, nos lleva a comprender que durante la tarde la nubosidad iba variando. Además, esa nubosidad evita que parte de la radiación ultravioleta emitida por el Sol llegue a nuestro sensor y es por ello que su valor se reduce según la nubosidad, igual que hace la radiación infrarroja.

Figura 66. Experimento 1 LEDs RGB.

Para el caso de las lecturas R1, G4 y B4 que han sido tomadas en tiempos de exposición muy cortos vemos la misma tendencia que representan las lecturas de IR y UV. Aparte

de que ya no se pueden excitar durante la noche debido a que no tienen tiempo suficiente para ello, son útiles para reconocer ciertas condiciones.

Cuando anochece vemos que los valores R1 y B4 de baja exposición se mantienen con una intensidad mínima hasta que en la noche no pueden registrar datos. Por otro lado, vemos que R4 y G4 decrecen linealmente desde que el sol presenta cierto grado crítico con respecto al suelo hasta que se esconde. A pesar de ello, seguimos viendo en ellos esos dientes de sierra que indican que la nubosidad va variando durante ese periodo.

Finalmente, vemos que los valores de mayor exposición R11, G11 y B11 presentan altos niveles de intensidad durante toda la tarde hasta que decrecen de forma similar a una arco-tangente negativa y muy rápidamente. Son solo estos últimos los que registrarán medidas en el periodo nocturno debido a que la noche, a pesar de ser sin luna, tiene niveles moderados de contaminación lumínica.

4.2.2 Experimento 2: Tarde del 22/05 soleada

Durante la tarde del día 22 de mayo se fueron adquiriendo datos de un día completamente soleado y sin nubes aparentes. Fue colocado desde las 14:50 hasta las 22:00. El resultado fue el siguiente:

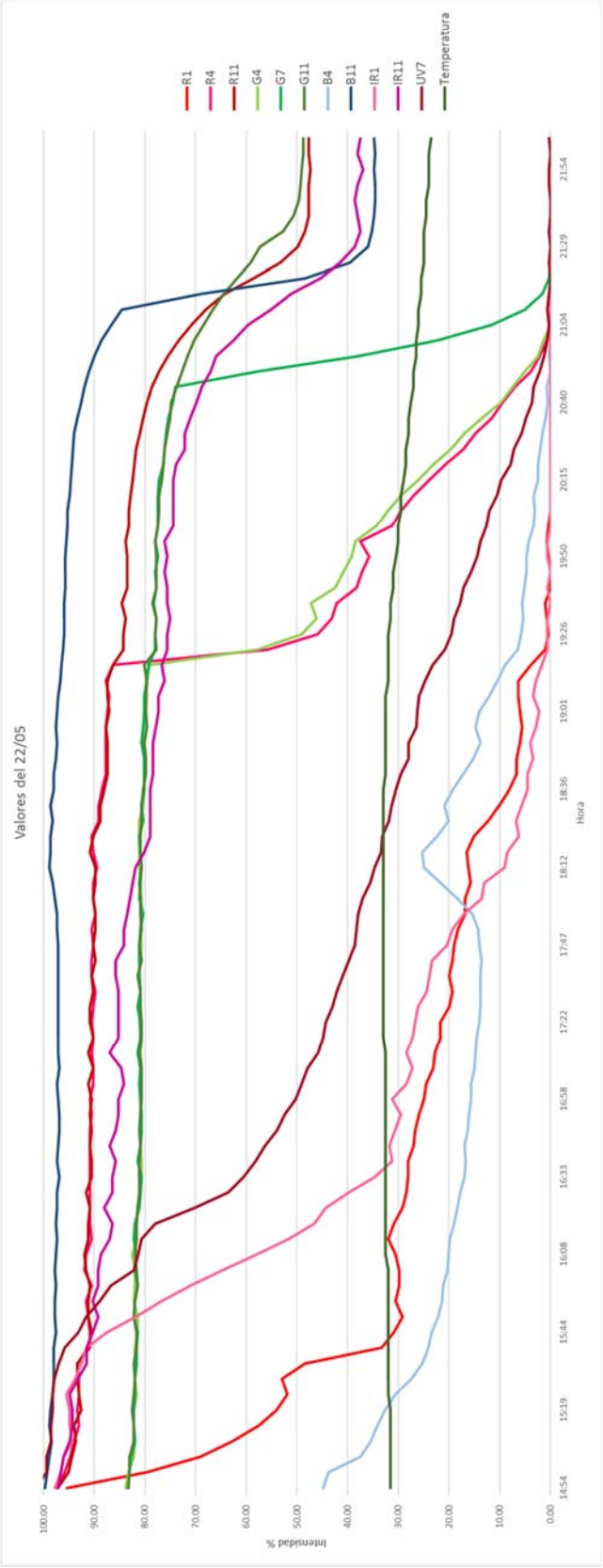


Figura 67. Experimento 2 LEDs.

Observamos al inicio que la exposición solar es alta y según va pasando la tarde van decreciendo hasta que se llega a un umbral que es donde se reconoce que comienza el ocaso donde ciertas lecturas caen en picado hasta la noche.

Para el análisis de la tarde usaremos las siguientes gráficas:

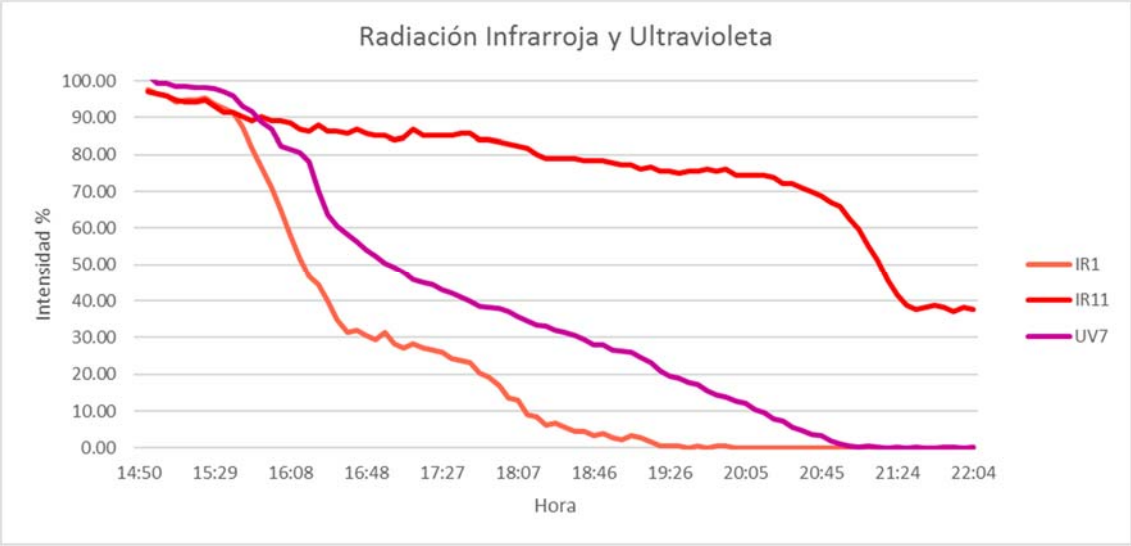


Figura 68. Experimento 2 LEDs UV IR.

Los valores recogidos para IR11 son habituales durante sol, decreciendo linealmente y cayendo en el ocaso. Ahora vemos que IR1 y UV7 no fluctúan como vimos antes, sino que la radiación solar es constante ya que no hay presencia de nubes. La caída de IR1 y UV7 es debido a que el sol comienza a alejarse aún más del punto más alto que es donde los sensores están apuntando.

Figura 69. Experimento 2 LEDs RGB.

Observamos que las lecturas de alta exposición R4 y R11, además de G4 y G 11 presentan el mismo valor durante la tarde hasta que comienza el atardecer y las lecturas de tiempos de exposición menores R4 y G4 decrecen drásticamente hasta decrecer linealmente con una fuerte pendiente. Observamos que R1 y B4 decrecen linealmente durante la tarde menos en un periodo en el que B4 cambia de tendencia, producto de la proyección de la sombra de un edificio sobre el sensor. Por otro lado, las lecturas R11, G11 y B11 presentan la misma tendencia que el experimento anterior las cuales tienen valores próximos a la saturación y luego caen siguiendo una función aproximada del tipo:

$$y = -\arctan(a * x)$$

Quedando luego estabilizadas en un valor durante la noche viendo además que se conservan en ambos experimentos el mismo porcentaje de intensidad. Valores que variarían en función de la cantidad de luz nocturna, bien por la contaminación lumínica o bien por luz lunar.

4.2.3 Experimento 3: Amanecer y día del 24/05

El experimento fue iniciado a las 4:45 de la madrugada del 24 de mayo hasta las 20:00 de la tarde de ese mismo día. Transcurrió totalmente soleado y sin ningún tipo de nubes o capa de contaminación. Los datos recogidos han sido los siguientes:

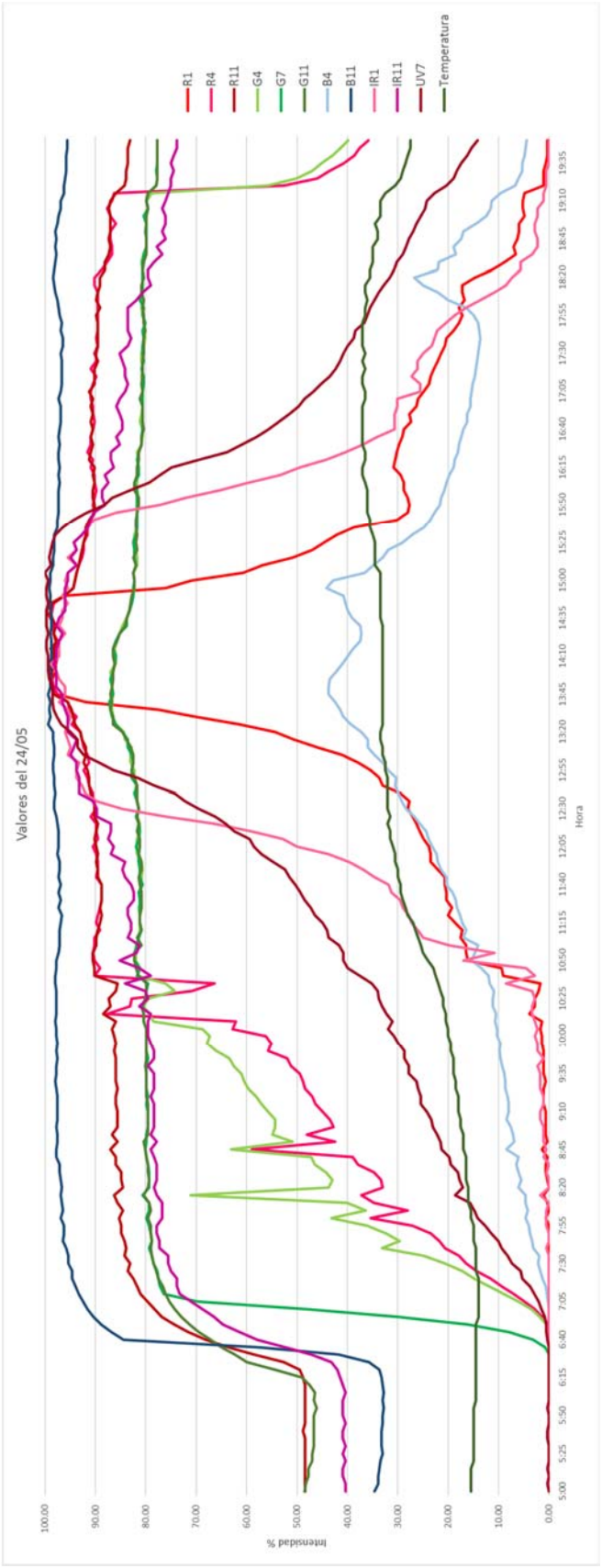


Figura 70. Experimento 3 LEDs.

Podemos observar cómo se va desarrollando el día desde que comienza a amanecer a las 6:45 de la mañana y cómo van incrementando los valores hasta pasado el mediodía. Luego empieza la tarde y acaba durante el anochecer. También observamos como la evolución de la temperatura de la tierra va aumentando hasta las seis de la tarde en claro desfase a la temperatura ambiental.

Para el análisis de la tarde usaremos las siguientes gráficas:

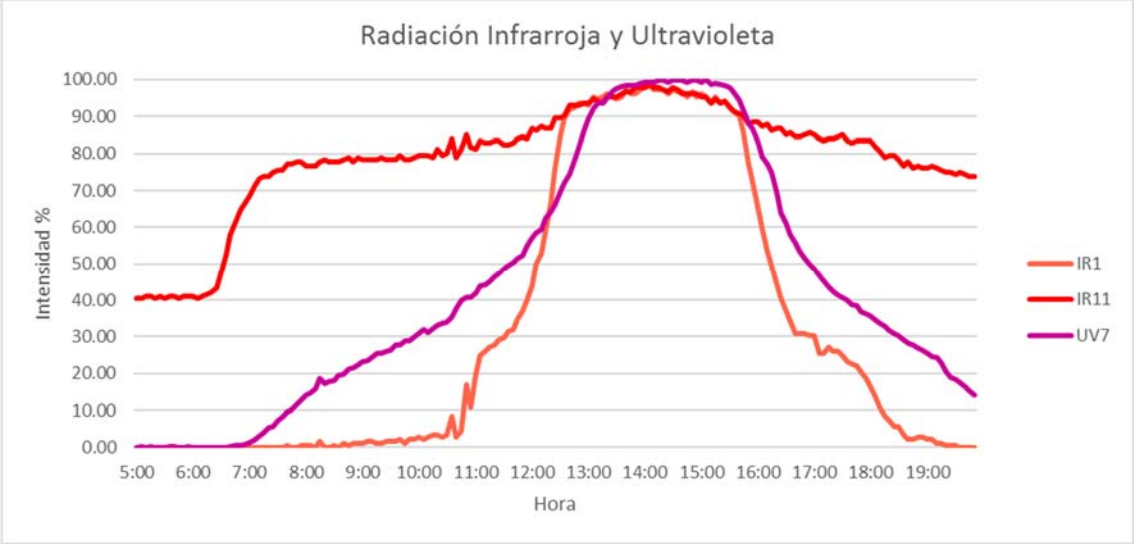


Figura 71. Experimento 3 LEDs UV IR.

Lo primero que se observa es la forma de campana tan característica que presentan las lecturas IR11 y UV7. Ello es un indicativo de como la radiación se vuelve más intensa según nos acercamos al medio día. Sobre la radiación ultravioleta, apreciamos tendencia lineal desde el amanecer hasta el mediodía donde se dispara de forma logarítmica hasta que el sensor satura. El descenso es similar pero la caída es mayor antes de empezar la fase lineal. En cuanto a las lecturas de valores infrarrojos, no es hasta que el Sol está en el zenit cuando ambas lecturas coinciden. Ello es producto del ángulo del haz de luz que no es lo suficientemente perpendicular al sensor como para poder obtener medidas. Es destacable el breve rizado durante las 10:30 hasta las 11:00 horas que, como vimos en el experimento anterior, representa el paso de nubes ya que dispersan la radiación infrarroja.

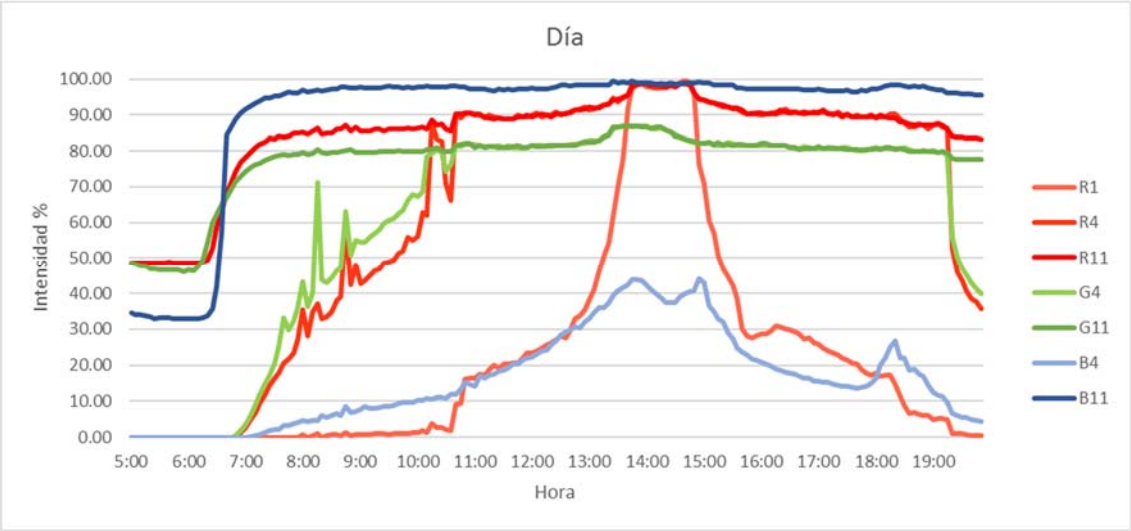


Figura 72. Experimento 3 RGB.

Sobre los valores de mayor exposición R11, G11 y B11 suben logarítmicamente hasta un valor estable. B11 permanece muy estable durante todo el día y próximo al valor de saturación, mientras que R11 y G11 sufren un aumento durante el momento de mayor radiación llegando el sensor rojo a saturarse.

Cuando el amanecer ya se ha establecido, y los valores de R4, G4 y B4 comienzan a aumentar, observamos un rizado acentuado desde las 7:45 hasta las 9:15 muy destacado en R4 y G4 pero mínimamente apreciable en B4. Después de una investigación exhaustiva se llegó a la conclusión que era producto de una sombra que generaba una palmera sobre el sensor que, al proporcionar una sombra irregular por la morfología de las palmas, ha generado un falso reconocimiento de "Nubes" como condición de luz. Para que realmente fuesen nubes debería de existir un rizado similar en los valores de radiación infrarroja durante ese periodo producto de la dispersión de esta radiación.

Es curioso destacar las inversiones que se producen en las gráficas de R1 y B4 a media mañana y a media tarde. Inicialmente la curva azul predomina sobre la roja, pero observamos que a las 10:30 de la mañana comienza a dispararse los valores de IR1 y como la radiación roja dentro de la luz visible está muy próxima a la infrarroja, también se dispara, como observamos, en los valores del R1. Estos valores crecen exponencialmente hasta saturar durante una hora y media volviendo a descender rápidamente. La tendencia es casi lineal durante la tarde.

Los valores de B4 aumentan linealmente hasta el periodo de mayor nivel de radiación donde fluctúa a un nivel de un 40% de intensidad. Luego comienza a descender linealmente, pero durante las 18:00 hasta las 19:30 se produce un cambio en esa tendencia producto de la sombra que generan los edificios sobre el sensor y es por ello que el azul es un indicador del grado de nubosidad. Se observa que R1 también sufre esta perturbación.

4.3. Resultados usando un dispositivo FDR para medir la humedad

Después de usar los LEDs como sensores, se inició una complementación al trabajo añadiéndole la funcionalidad de conocer la humedad usando una sonda FDR. Los resultados mostrados son a partir de unos experimentos muy breves y realizados sin llevar a cabo una calibración u optimización por falta de recursos. Queda abierto para líneas futuras la continuación y desarrollo de esta parte.

Recordemos que para medir la humedad usábamos el método de la Reflectometría en el Dominio de la Frecuencia para conocer la capacidad eléctrica del suelo, ya que como la capacidad del agua y el aire difieren en un factor de 80 podemos saber esa concentración de agua sabiendo la capacidad del suelo. Está claro que, a mayor agua, mayor capacidad eléctrica del suelo, pero con el método FDR no obtenemos directamente los Faradios. Lo que hacíamos era usar la sonda que estaba insertada en el suelo como el elemento capacitivo de dos circuitos, un filtro de paso bajo y un circuito oscilador.

En el primer circuito obtendremos una lectura que se verá condicionada de la capacidad del condensador (capacidad de la tierra) para acumular energía en el filtro de paso bajo. Como el voltaje que consume un condensador es inversamente proporcional a la capacitancia del mismo, cuanto mayor sea su capacitancia, menor será el voltaje que consumirá. Esto quiere decir que según aumente la humedad del suelo que estamos analizando, menor será el nivel de tensión que leamos. Luego, la salida es retroalimentada en lazo cerrado y pasada por un circuito detector de picos. El segundo circuito proporciona una señal continua de 5V que hará cargar y descargar la sonda. Claramente, según el valor de capacidad del suelo, la sonda tendrá más capacidad para almacenar energía y lo conoceremos mediante el control de los tiempos que emplean en cargar y descargar ese condensador. A mayor tiempo empleado, mayor capacidad.

4.3.1 Experimento 1: 24/05

Durante las 5 de la madrugada hasta las 8 de la tarde, se hicieron mediciones de humedad y temperatura del suelo. El registro fue en una maceta con turba y tierra y durante las mediciones no se hizo aporte de agua y se tapó la maceta para evitar la evapotranspiración, pero los valores reflejan conclusiones bastante diferentes.

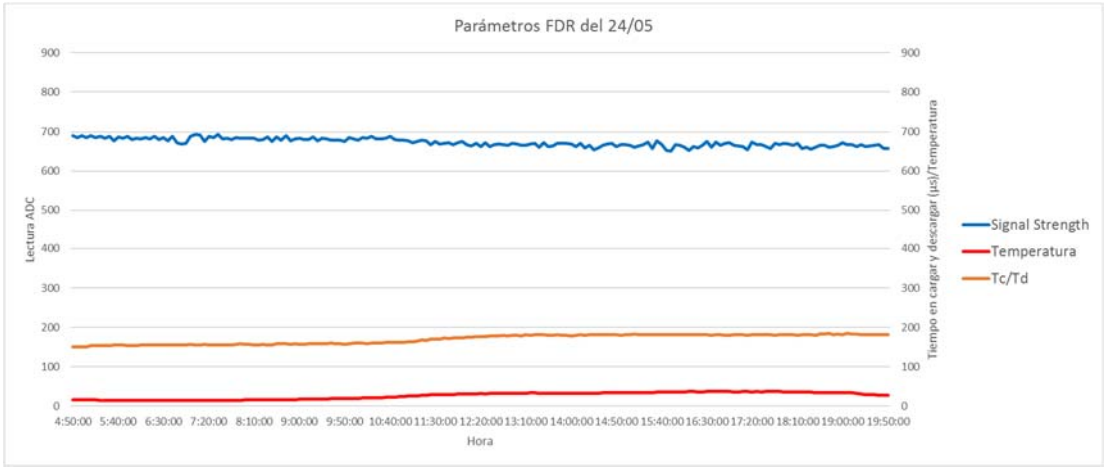


Figura 73. Experimento 1 FDR.

Si nos fijamos en el periodo de las 10:00 a las 13:00, observamos que la temperatura aumenta. A la misma vez y en ese periodo, el valor de "Signal Strenght" se reduce, mientras que "Time to charge/discharge" aumenta. Por otro lado observamos la cantidad de ruido que se genera en los datos de "Signal Strenght".

A simple vista podemos decir que en la muestra ha aumentado la humedad durante ese periodo de tiempo, pero realmente no fue así ya que no fue regada. Observamos, además, que los cambios de tendencia en las curvas presentan simetría con la evolución de la temperatura. Esto ocurre porque el valor de capacitancia del suelo obtenido mediante la técnica FDR se ve afectada tanto por la temperatura del suelo como por la temperatura crítica de los componentes del circuito de medición debido a la sensibilidad que presenta esta técnica a estos valores.

Resultados obtenidos a partir de experimentos realizados por el Doctor Martin Oates en la Universidad Miguel Hernández muestran que estos dos efectos que producen la temperatura del suelo y de los componentes electrónicos sobre las mediciones son complementarios. La temperatura del suelo incrementa el valor de la capacidad obtenida, mientras que la temperatura de los componentes electrónicos la reducen. Los perfiles diarios de la temperatura del suelo y de los componentes, que ambos aumentan durante el día y disminuyen durante la noche, muestran fases significativamente diferenciadas y difíciles de cancelar. Por otro lado, la compensación de la temperatura requerida debe de variar para cada técnica y niveles de humedad.

De este estudio se obtuvieron una serie de fórmulas para solventar la compensación de la temperatura en las mediciones de humedad del suelo.

$$SS_{62kHz\ Compensada} = SS_{62kHz} - \frac{T_{interna} - 12}{2.67} - \frac{T_{suelo} - 20}{1.5}$$

$$Tc_{Compensada} = Tc + \frac{T_{interna} - 12}{2.75} - (T_{suelo} - 20) * \frac{Tc - 45}{45}$$

Debido a que las condiciones del experimento realizado se alejan de las de nuestro proyecto tanto en que usan un amplificador operacional LM358 y no un TL074CN como el nuestro además de que la frecuencia de la señal es de 65525 Hz, debemos de reajustando ciertos parámetros quedando las fórmulas como se muestra:

$$SS_{62kHz\ Compensada} = SS_{62kHz} - \frac{T_{interna} - 12}{25} - \frac{T_{suelo} - 20}{16}$$

$$Tc_{Compensada} = Tc + \frac{T_{interna} - 12}{4.4} - (T_{suelo} - 20) * \frac{Tc - 60}{60}$$

La compensación deja los valores del siguiente modo:

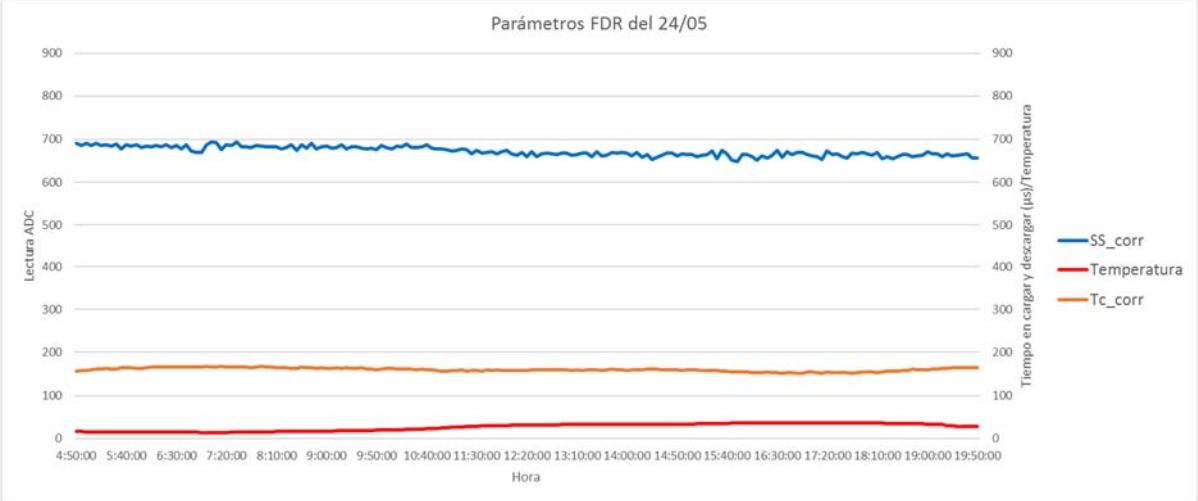


Figura 74. Experimento 1 FDR II.

Es difícil buscar expresiones generales para la compensación de la temperatura, ya que las condiciones particulares de cada experimento harán variar las mediciones y por ello una mala compensación. Este método es novedoso y está aún en desarrollo, pendiente de ser implementado en dispositivos de monitorización de muestras de suelo en la UMH.

4.3.2 Experimento 2: 29/05

Haciendo uso de la muestra de suelo anterior, la inundamos con 300 ml de agua teniendo en cuenta que las dimensiones de la maceta son de 30 cm de diámetro y 30 cm de alto. Después de media hora de reposo se empezó a monitorizar los parámetros desde las 16:30 hasta media noche.

Figura 75. Experimento 2 FDR.

Observamos que en la señal de "Signal Strength" aparece mucho ruido y fue descartada. También observamos como "Time to charge/discharge" disminuye logarítmicamente pero aumentando su valor en comparación con el experimento anterior.

Esta tendencia logarítmica nos deja de manifiesto cómo va disminuyendo la humedad de la muestra según avanzan las horas producto de la evapotranspiración y la absorción de la planta. Los resultados también nos llevan a valorar que la concentración de agua en la maceta no es homogénea. Debido a que la sonda se encontraba insertada en la parte mitad superior de la maceta y que la tierra estaba prácticamente seca, inicialmente detectaba alta concentración de agua, pero con el paso de las horas y por efecto de la evapotranspiración, la concentración de agua va disminuyendo en la parte superior donde se encontraba la sonda.

5

Conclusiones

Finalmente se hará un análisis de los resultados obtenidos y las conclusiones que arrojan. Se describirá y valorará sus costes frente a otras alternativas.

5.1. Análisis general

5.1.1 Análisis de LEDs como sensores para condiciones de luz

Estos resultados demuestran que un Analizador de Luz Solar puede medir la intensidad de varias longitudes de ondas que componen la luz solar y clasificarlas en un amplio rango de condiciones de luz.

A partir de los datos obtenidos y durante prácticas realizadas en las que el receptor concluía que condición de luz solar había presente, se llegó a la conclusión de que las clasificaciones más simples son para sol directo, cielo azul, nublado y noche. Estas condiciones se pueden determinar mediante un monitoreo de corta duración con infrarrojo y niveles de rojo. La diferencia entre sol directo y cielo azul es clara, ya que cuando la radiación infrarroja y ultravioleta es máxima, indica exposición a sol directo. Sin embargo, es difícil de diferenciar la densidad de la nubosidad entre densa o ligera ya que el parámetro determinante es la dispersión de la radiación infrarroja y se ve afectada por otros factores, provocando también problemas entre niebla y nubes. En cuanto a la noche, surgen problemas entre noche con luna y contaminación, ya que los valores que provocan la luz de la luna sobre los sensores son similares a las condiciones de contaminación.

Queda demostrado que el grado de nubosidad afecta a la dispersión de la radiación infrarroja y más aún a valores de corta duración. Por otro lado, la cantidad de radiación ultravioleta que llega a la superficie se puede ver afectada por la concentración de nubes ya que parte es absorbida por ellas.

Lecturas de media exposición como en el sensor azul nos dan una medida de la cantidad de nubosidad presente y se han observado tendencias similares en los valores de corta exposición del sensor rojo. Además, las lecturas de media exposición de los sensores verde y rojo nos permiten conocer cuando comienza el amanecer o el atardecer desde el punto de vista de la emisión de radiación electromagnética ya que sus valores crecen o decrecen con una gran pendiente en cuestión de pocas horas.

Los valores obtenidos en lecturas de alta exposición nos proporcionan un valor más estable de las condiciones de luz. Se mantienen prácticamente constantes durante la noche y cada una con su nivel en función de la luz nocturna y la contaminación. Durante el amanecer y el atardecer crecen o decrecen de forma logarítmica hasta establecerse en torno a un valor.

La elección de LEDs como sensores frente a fotodiodos reside en que estos primeros están diseñados para generar una intensidad en función de la intensidad lumínica recibida pero dentro de un amplio rango de longitudes de onda y no discriminan entre unas u otras. Sin embargo, usando los diodos LEDs como sensores, obtenemos una excitación a la longitud de onda y por debajo de la que está diseñado para emitir. Es decir, los LEDs responden a una banda mucho más estrecha de longitudes de onda que los fotodiodos.

Con estas conclusiones podemos realizar un sistema que controle en tiempo real la gestión de recursos para la agricultura, tales como riego, iluminación artificial, calefacción y sombreado, gestión de pesticidas o soporte de polinización.

5.1.2 Análisis de la técnica FDR para humedad

Como hemos mencionado al principio, estos son los resultados preliminares de una sonda de humedad en vías de desarrollo. Hemos tomado contacto con los principales problemas a solventar como calibración, compensación de temperatura, eliminación de ruido, mejor combinación de frecuencia y amplificador, etc. que será objetivo de trabajo en futuros desarrollos. De los breves experimentos que hemos realizado hemos sacado una serie de conclusiones que son interesantes destacar.

El primer parámetro que muestreábamos es la lectura a través de un puerto ADC. No es otra cosa que el voltaje que consume la sonda en el filtro de paso bajo y después de pasar por un circuito amplificador retroalimentado y un circuito detector de picos. La sensibilidad de la lectura es mínima y su estabilidad ha quedado constatada. El segundo parámetro es el tiempo que la sonda carga y descarga a través de un circuito oscilador. Ese tiempo es la lectura de cuatro semiciclos de onda. Durante las observaciones vemos que es muy sensible a los cambios que se pueden producir además de requerir más tiempo para llegar al régimen estacionario.

Es importante resaltar que los experimentos nos arrojan la conclusión de que se produce una descompensación en los valores con el efecto de la temperatura. Esto ocurre porque el valor de capacitancia del suelo obtenido mediante la técnica FDR se ve afectada tanto por la temperatura del suelo como por la temperatura crítica de los componentes del circuito de medición. La temperatura del suelo incrementa el valor de la capacidad obtenida, mientras que la temperatura de los componentes electrónicos la reducen.

Las sondas FDR capacitivas requieren una separación física determinada para prevenir acumulaciones de finas capas de agua alrededor de la superficie de la sonda. Esto se consigue con un tratamiento de pintura que permitan protegerla de la incursión de agua y le permita de medio dieléctrico como parte de un condensador.

La posición de la sonda afecta también a los valores registrados y a su evolución. Al regarse una muestra de suelo, el gradiente de concentración de agua decrecerá hacia el fondo de la misma. Pero su evolución será diferente según la cantidad de humedad presente en la muestra previamente. Si la tierra se encuentra muy seca, gran parte del agua será absorbida en las capas superiores y poca llegará al fondo. Por otro lado, regar una tierra con un buen nivel de humedad hará saturar de agua las primeras capas de tierra quedando suficiente para las capas más inferiores. La clave está en buscar un equilibrio entre nivel de humedad remanente y aporte de agua necesario para regar. Además, disponer de varias sondas que monitoricen distintas profundidades de la muestra nos proporcionaran unas mejores conclusiones.

El nivel de humedad determinado por la sonda FDR capacitiva se puede usar como sistema complementario en un analizador de luz solar para tener más información y

llevar un control más exhaustivo de los factores que afectan al crecimiento de los cultivos.

5.1.3 Análisis del sistema de adquisición de datos inalámbrico

Un sistema compuesto por un emisor sobre el terreno que obtiene medidas y las transmite a un receptor que las procesa y envía a un ordenador con un programa de adquisición de datos con LabVIEW es una forma sencilla, versátil y económica de tener una monitorización continua sobre una serie de parámetros como en este caso de condiciones de luz solar y humedad.

El sistema de comunicación es importante de dimensionar correctamente para los objetivos del proyecto. Existen diferentes opciones según el alcance, consumo o precio que nos interesa. Podemos hacer uso de los repetidores de telefonía vía GPRS pero pagando a una compañía por el servicio, hacer uso de sistemas de comunicaciones en VHF o HF pero con equipos y antenas más grandes y caros, hacer uso de sistemas de comunicaciones que transitan en UHF o en la frecuencia WiFi (2.4 GHz) pero con menos alcance. En el presente proyecto usamos módulos NRF24L01 que trabajan a 2.4 GHz y su consumo es muy bajo. Perdemos intensidad de señal y alcance (hasta 1 Km), pero ganamos en precio, tamaño y mínimo consumo, siendo ideal para sistemas formados por baterías.

Por otro lado, un programa informático que muestre de forma clara los valores obtenidos y los represente en tablas e iconos ayudará a una mejor interacción del usuario con los datos. Para hacerlo lo más dinámico posible, se puede modificar el programa de LabVIEW según los requisitos y necesidades del cliente final.

5.2. Presupuesto y valoración

5.2.1 Presupuesto empleado

El desglose económico del emisor es el siguiente:

Artículo	Unidades	Precio
Arduino UNO	1	4.67 €
Solar Panel Shield V2.2	1	16.20 €
Panel Solar	1	6.00 €
NRF24L01 + antena	1	1.80 €
DS18B20 Sumergible	1	1.20 €
Sonda de humedad + tratamiento	1	2.00 €
Antena 2.4 GHz 390 mm	1	4.00 €
Batería LiPo 3.7 V 600mAh	1	9.00 €
Componentes electrónicos	27	5.00 €
Caja de plástico ABS gris	1	10.75 €
Soporte interno	1	4.00 €

Soporte panel solar	1	2.00 €
Portes totales	1	3.00 €
Total	**42**	**69.62 €**

Tabla 10. Costes emisor.

Por otro lado, el desglose de componentes electrónicos es:

Componentes electrónicos	**Unidades**
LEDs	5
Resistencia 330 Ω	6
Resistencia 270 KΩ	1
Resistencia 4K7 Ω	1
Resistencia 47 KΩ	1
Resistencia 470 KΩ	3
Resistencia 1M8 Ω	1
Condensador 100 nF	1
Condensador 10 μF	1
Diodo 1N4004	1
Amplificador TL074CN	1
PCB 100x80	1
Cables (metros)	4
Total	**27**

Tabla 11. Componentes emisor.

Casi la mitad del precio va dedicado al sistema de alimentación mediante energía solar usando un panel y con batería LiPo para cargar y uso en horarios sin luz. Con el tiempo se verá amortizado por la falta de necesidad de un control permanente ni en reponer baterías. Un cuarto del total va dedicado a la electrónica que constituye el Arduino UNO y el circuito FDR, además del sensor de temperatura y la sonda de humedad. El otro cuarto va dedicado a los materiales necesarios para construir el chasis donde irán alojados los componentes.

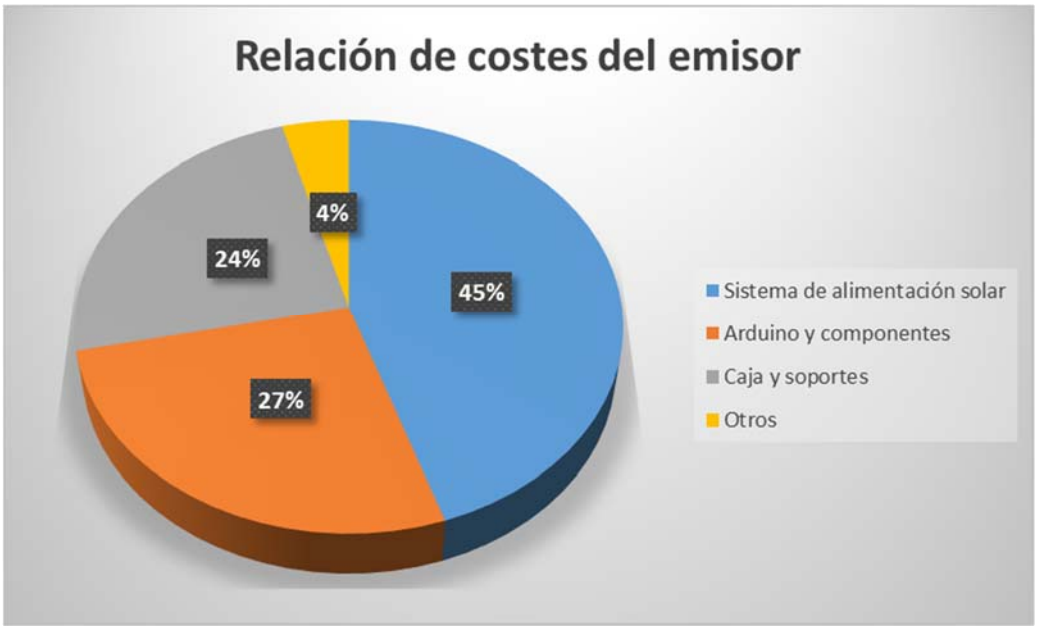


Figura 76. Proporciones de coste del emisor.

El desglose económico del receptor es el siguiente:

Artículo	Unidades	Precio
Arduino Pro Mini	1	4.67 €
NRF24L01 + antena	1	1.80 €
CP2021 USB a USART	1	1.10 €
Display OLED 0.96''	1	2.44 €
Componentes electrónicos	5	1.00 €
Caja de plástico ABS	1	3.97 €
Portes totales	1	2.00 €
Total	**11**	**16.98 €**

Tabla 12. Costes receptor.

Por otro lado, el desglose de componentes electrónicos es:

Componentes electrónicos	Unidades
Condensador 10 µF	2
HT7333	1
PCB 45x35	1
Cables (metros)	0.1
Total	**5**

Tabla 13. Componentes receptor.

A diferencia del emisor, el receptor no necesita de autonomía desde el punto de vista energético y es por ello que el 65% de su coste se dedique a los componentes electrónicos y al Arduino Pro Mini. El resto está dedicado a la caja donde irá alojado el circuito y los portes de los componentes.

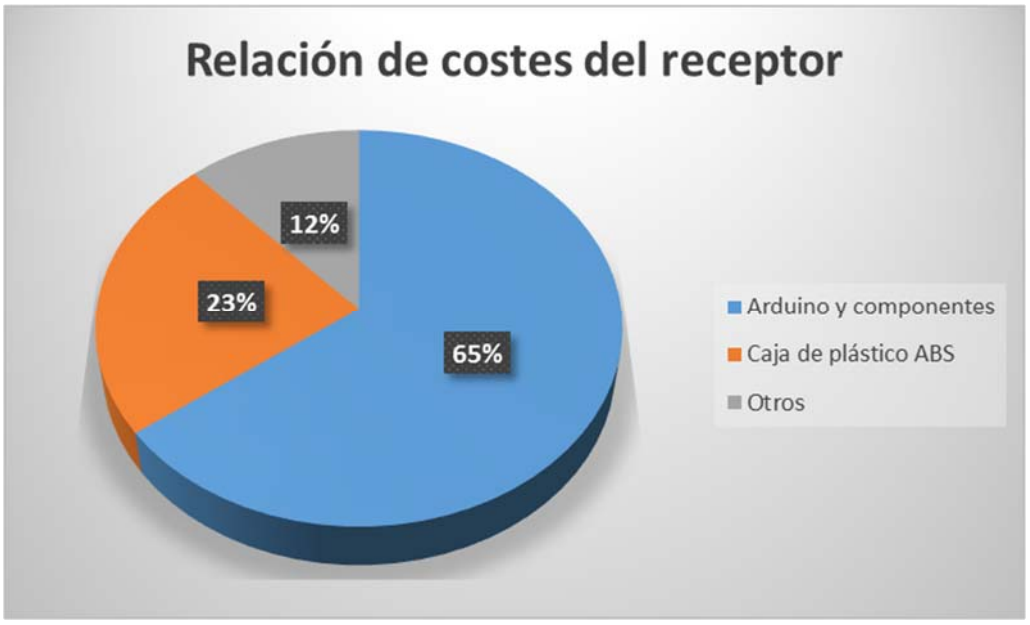


Figura 77. Proporciones de coste del receptor.

5.2.2 Valoración general

El sistema desarrollado brinda al usuario de un dispositivo flexible capaz de poder adaptarlo al terreno que deseamos, monitorizar las variables que queremos y transmitirlas de forma inalámbrica. Lo interesante de ser implementado el software de adquisición de datos en LabVIEW es que su sencilla programación gráfica nos permite modificarlo según nuestro último fin.

Frente a otras alternativas, un sistema de 86.6 € compuesto por emisor, receptor y software es muy competitivo frente a otros instrumentos ya existentes en el mercado. Éstos, aunque nos ofrecen una gran precisión, tienen un precio desbordado para realmente el uso que le queremos dar que es enfocado a obtener lecturas de condiciones de luz. Por otro lado, los instrumentos existentes no van compuestos de ningún sistema de adquisición de datos ni de una transmisión inalámbrica de los mismos.

6

Bibliografía

A continuación, se detallan las referencias bibliográficas utilizadas en la redacción de este Trabajo Fin de Grado.

Atmel. (2015). Atmel 8-bit Microcontroller with 4/8/16/32 kB in-system programmable flash Datasheet. Recuperado de http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf

Boylestad, & Nashelsky. (2009). Electrónica: Teoría de circuitos y dispositivos electrónicos. Pearson.

Callister. (2009). Introducción a la ciencia e ingeniería de los materiales. Limusa.

Consonance Electronic. Lithium Ion Battery Charger for Solar-Powered Systems CN3065. Recuperado de https://lib.chipdip.ru/164/DOC001164905.pdf

ETA Solutions. 3W, 0.85V Startup Voltage, Synchronous Step-Up Converter with Real-Shutdown and Short-Circuit Protection in SOT23-5. Recuperado de http://www.etasolution.com/upload/product/ETA1036.pdf

Franco-Martínez, F. & Oates, M.J. & Ferrández-Villena, M. & Molina-Martínez, J.M. & Fernández-López, A. & Ruiz-Canales, A. (2017). Red de dispositivos de bajo coste para la monitorización de parámetros ambientales relacionados con la gestión del agua y la energía en riego de hortícolas. VIII Congresso Ibérico de Ciências Hortícolas. Coimbra.

Lajara-Vizcaíno, J.R. & Sebastià, J.P. (2014). Sistemas integrados con Arduino. (1ª ed.) Barcelona: Ediciones Marcombo.

Llamas, L. (2016). Comunicación inalámbrica a 2.4 GHz con Arduino y NRF24L01. Zaragoza, España: Luis Llamas. Recuperado de https://www.luisllamas.es/comunicacion-inalambrica-a-2-4ghz-con-arduino-y-nrf24l01/

Llamas, L. (2016). Conectar Arduino a una pantalla OLED de 0.96''. Zaragoza, España: Luis Llamas. Recuperado de https://www.luisllamas.es/conectar-arduino-a-una-pantalla-oled-de-0-96/

Nave, C.R. (2012). HyperPhysics. Atlanta, EEUU.: Radiación solar. Recuperado de http://hyperphysics.phy-astr.gsu.edu/hbasees/vision/solirrad.html

Nordic Semiconductor. (2008). nRF24L01+ Single Chip 2.4 GHz Transceiver Datasheet. Recuperado de https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P

Oates, M.J. & Ruiz-Canales, A. & Molina-Martínez, J.M. & Vázquez de León, A.L. (2015). Compromiso entre coste y frecuencia de trabajo en sensores FDR de bajo coste para la gestión del riego. VIII Congreso Ibérico Agroingeniería. Sociedade de Ciencias Agrarias de Portugal. Orihuela-Algorfa.

Oates, M.J. & Ruiz-Canales, A. & Molina-Martínez, J.M. & Vázquez de León, A.L. (2015). Resultados preliminares de una herramienta de bajo coste para análisis de la luz solar y

gestión eficiente de recursos agrícolas. VIII Congreso Ibérico Agroingeniería. Sociedade de Ciencias Agrarias de Portugal. Orihuela-Algorfa.

Oates, M.J. & Ramadan, K. & Molina-Martínez, J.M. & Ruiz-Canales, A. (2016). Automatic fault detection in a low cost frequency domain (capacitance based) soil moisture sensor. Elsevier, 8.

Oates, M.J. & Timoschenko, N. & Ruiz-Canales, A. & Molina-Martínez, J.M. & Vázquez de León, A.L. (2016). Posibilidad del empleo de un analizador de luz solar de bajo coste y registrador de datos en la medida de radiación para agroclimatología. II Simposio Nacional de Ingeniería Hortícola. Sociedad Española de Ciencias Hortícolas. Almería.

Oates, M.J. & Fernández-López, A. & Ferrández-Villena, M. & Ruiz-Canales, A. (2016). Temperature compensation in a low cost frequency domain (capacitance based) soil moisture sensor. Elsevier, 8.

Pérez, R.E. Escuela Técnica IPEM 56 Abraham Juarez. Córdoba, Argentina: Radiación solar. Recuperado de http://www.oni.escuelas.edu.ar/2008/CORDOBA/1324/trabajo/radiacionsolar.html

Pérez-Carrasco, D. & Silva-Pérez M.A. Procedimiento de mantenimiento y calibración de estación radiométrica. Universidad de Sevilla, Sevilla.

7

Anexos

Recopilación de planos y códigos desarrollados durante la realización del presente trabajo.

7.1. Código Emisor

```
#include <OneWire.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <avr/sleep.h>
#include <avr/power.h>

//Módulo de radio
RF24 radio(9, 10);  //RF24 radio(CE_PIN, CSN_PIN)
const  uint64_t  ID[2]  =  {  0xF0F0F0F0E1LL,  0xF0F0F0F0D2LL  };
//Variable con la dirección del canal por donde se va a transmitir

//Sensor de temperatura
OneWire  ds(7);

//Variables FDR
float datos[6], celsius;
byte i, present = 0, data[12], addr[8];
int pc; //Si lo quito de aqui pc es 0

//Variables LEDs
int Rojo[12], Verde[12], Azul[12], IR[12], UV[12], Muestra[12];
int r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11;
//D14, D15, D16, D17 equivale a A0, A1, A2, A3,...
//Usamos el último valor de cada array para identificar a que LED
corresponde

//Otros
int seq = 0;

void temperatura() {
  if ( !ds.search(addr)) {
    ds.reset_search();
    delay(250);
  }
  ds.reset();
  ds.select(addr);
  ds.write(0x44, 1);
  delay(1000);
  present = ds.reset();
  ds.select(addr);
  ds.write(0xBE);
  for ( i = 0; i < 9; i++) {
    data[i] = ds.read();
  }
  int16_t raw = (data[1] << 8) | data[0];
  byte cfg = (data[4] & 0x60);
  if (cfg == 0x00) raw = raw & ~7;
  else if (cfg == 0x20) raw = raw & ~3;
  else if (cfg == 0x40) raw = raw & ~1;
```

```
  celsius = (float)raw / 16.0;
  datos[1] = celsius;
  Serial.print("Temperatura: ");
  Serial.println(datos[1]);
}

void FDR() {

  analogReference(DEFAULT);
  datos[2] = analogRead(5);
  datos[2] = analogRead(5); //Para estabilizar la referencia
analógica

  //FILTRO PASO BAJO INICIO
  pinMode(3, OUTPUT);
  tone(3, 65535);            //Activa la generación de una onda
cuadrada
  delay(300);               //Esperamos a que se estabilice
  datos[2] = analogRead(5);
  noTone(3);                //Desactiva la generación de la onda

  Serial.print("Intensidad (Signal Strength): ");
  Serial.println(datos[2]);
  //FILTRO PASO BAJO FIN

  //CIRCUITO OSCILADOR INICIO
  pinMode(3, INPUT);
  digitalWrite(4, HIGH);    //Inicio de oscilación
  delay(50);                //Esperamos a que se estabilice

  noInterrupts();
  pc = pulseIn(5, HIGH, 1000);
  pc = pc + pulseIn(5, LOW, 1000);
  pc = pc + pulseIn(5, HIGH, 1000);
  pc = pc + pulseIn(5, LOW, 1000);
  interrupts();
  //La función mide el tiempo en us que un pin esta en nivel alto
o bajo durante 1000 us
  //La función devuelve un 0 si no se completa un pulso en 1000
us
  if(pc == 0){
    pc = 1000;
  }

  digitalWrite(4, LOW); // Fin de oscilación
  datos[3] = pc;
  Serial.print("Veces que carga y descarga: ");
  Serial.println(datos[3]);
  //CIRCUITO OSCILADOR FIN

  datos[5] = 6;
}
```

```
void get_vcc()
{
#if defined(__AVR_ATmega32U4__) || defined(__AVR_ATmega1280__) ||
defined(__AVR_ATmega2560__)
  ADMUX = _BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) |
_BV(MUX1);
#elif defined (__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) ||
defined(__AVR_ATtiny84__)
  ADMUX = _BV(MUX5) | _BV(MUX0);
#elif defined (__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) ||
defined(__AVR_ATtiny85__)
  ADMUX = _BV(MUX3) | _BV(MUX2);
#else
  ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
#endif

  delay(2);                                  //Esperamos a que se
estabilice la tensión de referencia
  ADCSRA |= _BV(ADSC);                    //Hacemos una conversión
para inicializar
  while (bit_is_set(ADCSRA, ADSC));
  ADCSRA |= _BV(ADSC);                   //Iniciamos la conversión
  while (bit_is_set(ADCSRA, ADSC));   //Midiendo

  uint8_t low  = ADCL;                  //Primero se lee ADCL para
desbloquear la lectura de ADCH
  uint8_t high = ADCH;
  long result = (high << 8) | low;

  result = 1125300L / result;         // Calcula Vcc (mV); 1125300
= 1.1*1023*1000

  datos[4] = float(result / 1000.0);
  Serial.print("Tension de alimentacion: ");
  Serial.println(datos[4]);
}

void LEDs() {
  Serial.println();
  Serial.println("Análisis de luz solar");
  for (int x = 0; x < 5; x++) {
    if (x == 0)
    {
      Serial.print("Rojo");
    }
    if (x == 1)
    {
      Serial.print("Verde");
    }
    if (x == 2)
    {
      Serial.print("Azul");
    }
```

```
if (x == 3)
{
  Serial.print("IR");
}
if (x == 4)
{
  Serial.print("UV");
}

//Para inicializar las señales
pinMode(14 + x, OUTPUT);
digitalWrite(14 + x, LOW);
delay(3);
Muestra[0] = analogRead(x);

//Toma de medidas
noInterrupts();
pinMode(14 + x, INPUT);
Muestra[0] = analogRead(x); //r1
Muestra[1] = analogRead(x); //r2
Muestra[2] = analogRead(x);
Muestra[3] = analogRead(x);
delay(1);
Muestra[4] = analogRead(x);
delay(2);
Muestra[5] = analogRead(x);
delay(7);
Muestra[6] = analogRead(x);
interrupts();
delay(20);
Muestra[7] = analogRead(x);
delay(70);
Muestra[8] = analogRead(x);
delay(200);
Muestra[9] = analogRead(x);
delay(700);
Muestra[10] = analogRead(x); //r11

Serial.print(",   ");
Serial.print(Muestra[0]);
Serial.print(", ");
Serial.print(Muestra[1]);
Serial.print(", ");
Serial.print(Muestra[2]);
Serial.print(", ");
Serial.print(Muestra[3]);
Serial.print(", ");
Serial.print(Muestra[4]);
Serial.print(", ");
Serial.print(Muestra[5]);
Serial.print(", ");
Serial.print(Muestra[6]);
Serial.print(", ");
```

```
    Serial.print(Muestra[7]);
    Serial.print(", ");
    Serial.print(Muestra[8]);
    Serial.print(", ");
    Serial.print(Muestra[9]);
    Serial.print(", ");
    Serial.println(Muestra[10]);

    if (x == 0)
    {
      for (int i = 0; i < 12; i++)
      {
        Rojo[i] = Muestra[i];
      }
      Rojo[11] = 1;
    }
    if (x == 1)
    {
      for (int i = 0; i < 12; i++)
      {
        Verde[i] = Muestra[i];
      }
      Verde[11] = 2;
    }
    if (x == 2)
    {
      for (int i = 0; i < 12; i++)
      {
        Azul[i] = Muestra[i];
      }
      Azul[11] = 3;
    }
    if (x == 3)
    {
      for (int i = 0; i < 12; i++)
      {
        IR[i] = Muestra[i];
      }
      IR[11] = 4;
    }
    if (x == 4)
    {
      for (int i = 0; i < 12; i++)
      {
        UV[i] = Muestra[i];
      }
      UV[11] = 5;
    }
  }
  Serial.println();
}

void envio_datos() {
```

```
  radio.stopListening();
  radio.write(Rojo, sizeof(Rojo));
  radio.write(Verde, sizeof(Verde));
  radio.write(Azul, sizeof(Azul));
  radio.write(IR, sizeof(IR));
  radio.write(UV, sizeof(UV));
  radio.write(datos, sizeof(datos));
  boolean ok = radio.write(datos, sizeof(datos));

  radio.startListening();
  if (ok) {
    Serial.println("Enviado");
  }
  else {
    Serial.println("No enviado");
  }
  Serial.println();
}

typedef enum { wdt_16ms = 0, wdt_32ms, wdt_64ms, wdt_128ms,
wdt_250ms,   wdt_500ms,   wdt_1s,   wdt_2s,   wdt_4s,   wdt_8s}
wdt_prescalar_e;

void setup_watchdog(uint8_t prescalar);
void do_sleep(void);

const short sleep_cycles_per_transmission = 1;
volatile         short         sleep_cycles_remaining         =
sleep_cycles_per_transmission;

void setup() {
  radio.begin();                 //Inicializamos el NRF24L01
  radio.openWritingPipe(ID[0]); //Abrimos un canal de escritura

  pinMode(5, INPUT);             // Para contar las oscilaciones
  pinMode(4, OUTPUT);          // Para el control de la oscilación
  digitalWrite(4, LOW);          // Para la oscilación

  Serial.begin(57600);
  setup_watchdog(wdt_8s);
}

void loop() {

  //Toma de datos
  seq = seq + 1;
  datos[0] = seq;
  Serial.print("Secuencia: ");
  Serial.println(datos[0]);
  temperatura();
  FDR();
  get_vcc();
  LEDs();
```

```
  envio_datos();

  for (int i = 0; i < 6; i++) {
    Serial.print(datos[i]);
    Serial.print('\t');
  }

  while (sleep_cycles_remaining > 0) {
    do_sleep();
  }
  sleep_cycles_remaining = sleep_cycles_per_transmission;

}

void setup_watchdog(uint8_t prescalar)
{
  prescalar = min(9, prescalar);
  uint8_t wdtcsr = prescalar & 7;
  if ( prescalar & 8 )
    wdtcsr |= _BV(WDP3);

  MCUSR &= ~_BV(WDRF);
  WDTCSR = _BV(WDCE) | _BV(WDE);
  WDTCSR = _BV(WDCE) | wdtcsr | _BV(WDIE);
}

ISR(WDT_vect)
{
  --sleep_cycles_remaining;
}

void do_sleep(void)
{
  set_sleep_mode(SLEEP_MODE_PWR_DOWN); //Elegimos el "sleep mode"
que deseemos
  sleep_enable();

  sleep_mode();                        //En esta función entra el
sistema en modo durmiente

  sleep_disable();                    //Lo primero que se ejecuta
después del desbordamiento del Watchdog
}
```

7.2. Código Receptor

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

//Display
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

//nRF24L01
RF24 radio(9, 10); //RF24 radio(CE_PIN, CSN_PIN)
const uint64_t ID[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };
//Variable con la dirección del canal por donde se va a transmitir

//Variables
int Rojo[12], Verde[12], Azul[12], IR[12], UV[12];
float datos[6], secuencia_OLD;
int OLD_IR , OLD_Verde;
boolean Sol_directo = false, Cielo_azul = false,
Ligera_contaminacion = false, Amanecer = false, Anochecer = false;
boolean Nubes_ligeras = false, Nubes_densas = false, Sombra_pobre
= false, Noche = false, Neblina = false, flag = false;
String datosTFG;

const unsigned char PROGMEM UPCT_logo_bmp [] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x07, 0x00, 0x00, 0xE0, 0x00, 0x00,
  0x00, 0x00, 0x60, 0x01, 0x80, 0x06, 0x00, 0x00, 0x00, 0x06,
0x00, 0x03, 0xC0, 0x00, 0x60, 0x00,
  0x00, 0x18, 0x00, 0x07, 0xE0, 0x00, 0x18, 0x00, 0x00, 0x40,
0x00, 0x0F, 0xF0, 0x00, 0x02, 0x00,
  0x01, 0x8E, 0x00, 0x1C, 0x78, 0x00, 0x71, 0x80, 0x02, 0x07,
0xFC, 0x00, 0x00, 0x7F, 0xE0, 0x40,
  0x04, 0x07, 0xF0, 0x7F, 0x00, 0x0F, 0xE0, 0x20, 0x08, 0x03,
0xC7, 0xFF, 0x00, 0x03, 0xC0, 0x10,
  0x10, 0x03, 0x9E, 0x17, 0x27, 0x01, 0x80, 0x08, 0x20, 0x01,
0x3E, 0x07, 0x1E, 0x00, 0x00, 0x04,
  0x20, 0x00, 0x7F, 0x07, 0x07, 0x00, 0x00, 0x04, 0x40, 0x04,
0xFC, 0x03, 0x0F, 0x80, 0x20, 0x02,
  0x40, 0x79, 0xFC, 0x63, 0x00, 0x80, 0x1E, 0x02, 0x43, 0xF9,
0xFF, 0xFF, 0x00, 0x00, 0x1F, 0xC2,
  0x43, 0xF8, 0x00, 0x00, 0xFB, 0xFF, 0x9F, 0xC2, 0x40, 0x78,
0x04, 0xC0, 0xC0, 0x7F, 0x9E, 0x02,
  0x40, 0x04, 0x02, 0xC0, 0xE0, 0xFF, 0x20, 0x02, 0x20, 0x00,
0x01, 0xC0, 0xE0, 0xFE, 0x00, 0x04,
  0x20, 0x00, 0x01, 0x30, 0xC0, 0x3C, 0x00, 0x04, 0x10, 0x01,
0x80, 0x60, 0x84, 0x39, 0xC0, 0x08,
```

```
  0x08, 0x03, 0xC0, 0x00, 0xFF, 0xE3, 0xC0, 0x10, 0x04, 0x07,
0xF0, 0x00, 0xFE, 0x0F, 0xE0, 0x20,
  0x02, 0x07, 0xFE, 0x00, 0x00, 0x3F, 0xE0, 0x40, 0x01, 0x8E,
0x00, 0x1C, 0x18, 0x00, 0x71, 0x80,
  0x00, 0x40, 0x00, 0x0F, 0xF0, 0x00, 0x02, 0x00, 0x00, 0x18,
0x00, 0x07, 0xE0, 0x00, 0x18, 0x00,
  0x00, 0x06, 0x00, 0x03, 0xC0, 0x00, 0x60, 0x00, 0x00, 0x00,
0x60, 0x01, 0x80, 0x07, 0x00, 0x00,
  0x00, 0x00, 0x07, 0x00, 0x00, 0xE0, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

void leyendo_display() {
  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(5, 8);
  display.setTextColor(WHITE);

  display.print("Leyendo...");
  display.display();
}

void sol_directo_display() {
  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(49, 0);
  display.setTextColor(WHITE);

  display.println("Sol");

  display.setTextSize(2);
  display.setCursor(25, 16);
  display.setTextColor(WHITE);

  display.print("directo");

  display.display();

}

void cielo_azul_display() {
  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(6, 10);
  display.setTextColor(WHITE);

  display.print("Cielo azul");

  display.display();
```

```
}

void ligera_contaminacion_display() {
  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(28, 0);
  display.setTextColor(WHITE);

  display.println("Ligera");

  display.setTextSize(2);
  display.setCursor(15, 16);
  display.setTextColor(WHITE);

  //display.print("contaminacion");
  display.print("polucion");
  display.display();

}

void amanecer_display() {
  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(18, 8);
  display.setTextColor(WHITE);

  display.print("Amanecer");
  display.display();
}

void anochecer_display() {
  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(13, 8);
  display.setTextColor(WHITE);

  display.print("Anochecer");
  display.display();

}

void neblina_display() {
  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(25, 8);
  display.setTextColor(WHITE);

  display.print("Neblina");
  display.display();
```

```
}

void nubes_ligeras_display() {
  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(35, 0);
  display.setTextColor(WHITE);

  display.println("Nubes");

  display.setTextSize(2);
  display.setCursor(23, 16);
  display.setTextColor(WHITE);

  display.print("ligeras");

  display.display();

}

void sombra_pobre_display() {
  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(28, 0);
  display.setTextColor(WHITE);

  display.println("Sombra");

  display.setTextSize(2);
  display.setCursor(35, 16);
  display.setTextColor(WHITE);

  display.print("pobre");

  display.display();

}

void nubes_densas_display() {

  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(35, 0);
  display.setTextColor(WHITE);

  display.println("Nubes");

  display.setTextSize(2);
  display.setCursor(28, 16);
```

```
  display.setTextColor(WHITE);

  display.print("densas");
  display.display();
}

void noche_display() {
  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(35, 8);
  display.setTextColor(WHITE);

  display.println("Noche");
  display.display();
}

void sol_display() {
  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(47, 8);
  display.setTextColor(WHITE);

  display.print("Sol");
  display.display();
}

void humedad_display() {

  //Interpolación entre 0%, a % y 100%
  float SS_porcentaje = (-((datos[2] - 740.0) * 100.0)) / 183.0;
  float tc_porcentaje = ((datos[3] - 112.0) * 100.0) / 888.0;
  float hum_media = (SS_porcentaje + tc_porcentaje) / 2.0;

  if (hum_media < 0) {
    hum_media = 0;
  }

  if (hum_media > 100) {
    hum_media = 100;
  }

  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(20, 0);
  display.setTextColor(WHITE);

  display.println("Humedad:");

  display.setTextSize(2);
  display.setCursor(28, 16);
```

```
  display.setTextColor(WHITE);

  display.print(hum_media);
  display.print("%");
  display.display();
}

void temperatura_display() {

  display.clearDisplay();

  display.setTextSize(2);
  display.setCursor(26, 0);
  display.setTextColor(WHITE);

  display.println("Temp.:");

  display.setTextSize(2);
  display.setCursor(28, 16);
  display.setTextColor(WHITE);

  display.print(datos[1]);
  display.print("^C");
  display.display();
}


void setup()
{
  radio.begin();                    //Inicializamos el NRF24L01
  radio.openReadingPipe(1, ID[0]);  //Leo informacion del emisor
  radio.startListening();

  display.begin(SSD1306_SWITCHCAPVCC, 0x3C); //Inicializamos la
pantalla OLED
  display.clearDisplay();                        //Limdpiamos el
buffer

  display.drawBitmap(30,   0,   UPCT_logo_bmp,   64,   32,   1);
//Mostramos el logotipo de la UPCT
  display.display();
  delay(1000);

  Serial.begin(57600);
}

void loop() {

  if ( radio.available() ) {
    leyendo_display();

    radio.read(Rojo, sizeof(Rojo));
    if (Rojo[11] == 1) {
```

```
  datosTFG = String(Rojo[0]);
  datosTFG += '\t';
  for (int i = 1; i < 10; i++) {
    datosTFG += String(Rojo[i]);
    datosTFG += '\t';
  }
  datosTFG += String(Rojo[10]);
  datosTFG += '\n';
}
delay(10);

radio.read(Verde, sizeof(Verde));
if (Verde[11] == 2) {
  for (int i = 0; i < 10; i++) {
    datosTFG += String(Verde[i]);
    datosTFG += '\t';
  }
  datosTFG += String(Verde[10]);
  datosTFG += '\n';
}
delay(10);

radio.read(Azul, sizeof(Azul));
if (Azul[11] == 3) {
  for (int i = 0; i < 10; i++) {
    datosTFG += String(Azul[i]);
    datosTFG += '\t';
  }
  datosTFG += String(Azul[10]);
  datosTFG += '\n';
}
delay(10);

radio.read(IR, sizeof(IR));
if (IR[11] == 4) {
  for (int i = 0; i < 10; i++) {
    datosTFG += String(IR[i]);
    datosTFG += '\t';
  }
  datosTFG += String(IR[10]);
  datosTFG += '\n';
}
delay(10);

radio.read(UV, sizeof(UV));
if (UV[11] == 5) {
  for (int i = 0; i < 10; i++) {
    datosTFG += String(UV[i]);
    datosTFG += '\t';
  }
  datosTFG += String(UV[10]);
  datosTFG += '\n';
}
```

```
    delay(10);

    radio.read(datos, sizeof(datos));
    if (datos[5] == 6)
    {
      for (int i = 0; i < 5; i++) {
        datosTFG += String(datos[i]);
        datosTFG += '\t';
      }
      datosTFG += String(float(datosTFG.length()));
      datosTFG += '\r';
      Serial.print(datosTFG);
      flag = true;
    }
    delay(10);

  }

  if (flag) {

    int n = Rojo[11] + Verde[11] + Azul[11] + IR[11] + UV[11] +
datos[5];

    if (n != 21) {
      flag = false;
      display.clearDisplay();

      display.setTextSize(2);
      display.setCursor(40, 8);
      display.setTextColor(WHITE);

      display.print("ERROR");
      display.display();
      delay(1000);
    }

    if(datos[0]-secuencia_OLD > 1.0){
      flag = false;
      display.clearDisplay();

      display.setTextSize(2);
      display.setCursor(10, 8);
      display.setTextColor(WHITE);

      display.print("INFO LOST");
      //display.print(datos[0]-secuencia_OLD);
      display.display();
      delay(1000);
    }
    secuencia_OLD = datos[0];
    //EVALUACION DE DATOS//
```

```
////////////////////////////////////////////////////////////////
////////////////////////////////////
    if (IR[0] > 50) {
      Sol_directo = true;
    }

    if (Rojo[0] >= 15 && Rojo[0] > IR[0] * 0.8)
    {
      Cielo_azul = true;
    }

    if (Rojo[3] < 20)
    {
      if (Rojo[10] > 15 && Verde[3] > 0) {
        Ligera_contaminacion = true;
      }
      if (Rojo[10] > 5 && Rojo[10] * 3 > IR[10]) {
        Ligera_contaminacion = true;
      }
      if (IR[10] < 12 || OLD_IR < 12) {
        Noche = true;
      }
    }

    if (Rojo[3] < 20 && Verde[10] > 30) {
      if (IR[10] < 55 && IR[10] > OLD_IR) {
        if (Verde[6] > OLD_Verde) {
          Amanecer = true;
        }
      }
      if (Verde[6] > OLD_Verde) {
        Anochecer = true;
      }
    }

    if (IR[0] > Rojo[0] && Rojo[0] > 4)
    {
      if (Verde[3] > 35 && IR[0] < 100) {
        Neblina = true;
      }
      if (Verde[3] > 15 && IR[0] < 100) {
        Nubes_ligeras = true;
        if (Azul[3] > IR[0] && Azul[3] > 10) {
          Nubes_ligeras = false;
          Nubes_densas = true;
        }
      }
    }

    if (Azul[3] > IR[0] && Azul[3] < 30)
    {
      Sombra_pobre = true;
```

```
    }

    OLD_IR = IR[10];
    OLD_Verde = Verde[6];

////////////////////////////////////////////////////////////////
////////////////////////////////////

    //Mostrar en el display temperatura y humedad
    temperatura_display();
    delay(2000);
    humedad_display();
    delay(2000);

    //CONDICIONES DE LUZ
    //Mostrar en display las condiciones de luz
    if (Sol_directo && Cielo_azul) {
      sol_display();
      Sol_directo = false;
      Cielo_azul = false;
      delay(1000);
    }

    if (Sol_directo) {
      sol_directo_display();
      Sol_directo = false;
      delay(1000);
    }

    if (Cielo_azul) {
      cielo_azul_display();
      Cielo_azul = false;
      delay(1000);
    }

    if (Ligera_contaminacion) {
      ligera_contaminacion_display();
      Ligera_contaminacion = false;
      delay(1000);
    }

    if (Amanecer) {
      amanecer_display();
      Amanecer = false;
      delay(1000);
    }

    if (Anochecer) {
      anochecer_display();
      Anochecer = false;
      delay(1000);
    }
```

```
    if (Nubes_ligeras) {
      nubes_ligeras_display();
      Nubes_ligeras = false;
      delay(1000);
    }

    if (Nubes_densas) {
      nubes_densas_display();
      Nubes_densas = false;
      delay(1000);
    }

    if (Sombra_pobre) {
      sombra_pobre_display();
      Sombra_pobre = false;
      delay(1000);
    }

    if (Noche) {
      noche_display();
      Noche = false;
      delay(1000);
    }

    if (Neblina) {
      neblina_display();
      Neblina = false;
      delay(1000);
    }

    //Reseteo de los arrays
    for (byte a = 0; a < 12; a++) {
      Rojo[a] = 0;
      Verde[a] = 0;
      Azul[a] = 0;
      IR[a] = 0;
      UV[a] = 0;
    }

    //Cuando no se consigue reconocer la condición de luz
   /*if (Sol_directo + Cielo_azul + Ligera_contaminacion +
Amanecer + Anochecer + Nubes_ligeras + Nubes_densas + Sombra_pobre
+ Noche + Neblina == 0) {
      display.clearDisplay();

      display.setTextSize(1);
      display.setCursor(0, 8);
      display.setTextColor(WHITE);

      display.print("Condicion de luz");

      display.setTextSize(1);
      display.setCursor(0, 18);
```

```
      display.setTextColor(WHITE);

      display.print("indeterminada");
      display.display();
      delay(1000);
    }*/
    flag = false;


  }
}
```

7.3. Código Emisor con SD

```
#include <SD.h>
#include <OneWire.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <avr/sleep.h>
#include <avr/power.h>

//SD
#define PIN_SAVE  6
#define chipSelect  8
File dataFile;

//Módulo de radio
RF24 radio(9, 10);  //RF24 radio(CE_PIN, CSN_PIN)
const uint64_t ID[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };
//Variable con la dirección del canal por donde se va a transmitir

//Sensor de temperatura
OneWire  ds(7);

//Variables FDR
float datos[6], celsius;
byte i, present = 0, data[12], addr[8];
int pc; //Si lo quito de aqui pc es 0

//Variables LEDs
int Rojo[12], Verde[12], Azul[12], IR[12], UV[12], Muestra[12];
int r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11;
//D14, D15, D16, D17 equivale a A0, A1, A2, A3,...
//Usamos el último valor de cada array para identificar a que LED
corresponde

//Otros
int seq = 0;
bool save, SDOK, PIN_SAVE_ACTIVE;
String datosTFG;

void temperatura() {
  if ( !ds.search(addr)) {
    ds.reset_search();
    delay(250);
  }
  ds.reset();
  ds.select(addr);
  ds.write(0x44, 1);
  delay(1000);
  present = ds.reset();
  ds.select(addr);
  ds.write(0xBE);
```

```
  for ( i = 0; i < 9; i++) {
    data[i] = ds.read();
  }
  int16_t raw = (data[1] << 8) | data[0];
  byte cfg = (data[4] & 0x60);
  if (cfg == 0x00) raw = raw & ~7;
  else if (cfg == 0x20) raw = raw & ~3;
  else if (cfg == 0x40) raw = raw & ~1;
  celsius = (float)raw / 16.0;
  datos[1] = celsius;
}

void FDR() {

  analogReference(DEFAULT);
  datos[2] = analogRead(5);
  datos[2] = analogRead(5); //Para estabilizar la referencia
analógica

  //FILTRO PASO BAJO INICIO
  pinMode(3, OUTPUT);
  tone(3, 65535);            //Activa la generación de una onda
cuadrada
  delay(300);               //Esperamos a que se estabilice
  datos[2] = analogRead(5);
  noTone(3);                //Desactiva la generación de la onda
  //FILTRO PASO BAJO FIN

  //CIRCUITO OSCILADOR INICIO
  pinMode(3, INPUT);
  digitalWrite(4, HIGH);    //Inicio de oscilación
  delay(50);                //Esperamos a que se estabilice

  noInterrupts();
  pc = pulseIn(5, HIGH, 1000);
  pc = pc + pulseIn(5, LOW, 1000);
  pc = pc + pulseIn(5, HIGH, 1000);
  pc = pc + pulseIn(5, LOW, 1000);
  interrupts();
  //La función mide el tiempo en us que un pin esta en nivel alto
o bajo durante 1000 us
  //La función devuelve un 0 si no se completa un pulso en 1000
us

  digitalWrite(4, LOW); // Fin de oscilación
  datos[3] = pc;
  //CIRCUITO OSCILADOR FIN

  datos[5] = 6;
}

void get_vcc()
{
```

```
#if defined(__AVR_ATmega32U4__) || defined(__AVR_ATmega1280__) ||
defined(__AVR_ATmega2560__)
  ADMUX = _BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) |
_BV(MUX1);
#elif defined (__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) ||
defined(__AVR_ATtiny84__)
  ADMUX = _BV(MUX5) | _BV(MUX0);
#elif defined (__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) ||
defined(__AVR_ATtiny85__)
  ADMUX = _BV(MUX3) | _BV(MUX2);
#else
  ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
#endif

  delay(2);                                  //Esperamos a que se
estabilice la tensión de referencia
  ADCSRA |= _BV(ADSC);                    //Hacemos una conversión
para inicializar
  while (bit_is_set(ADCSRA, ADSC));
  ADCSRA |= _BV(ADSC);                  //Iniciamos la conversión
  while (bit_is_set(ADCSRA, ADSC));   //Midiendo

  uint8_t low  = ADCL;                  //Primero se lee ADCL para
desbloquear la lectura de ADCH
  uint8_t high = ADCH;
  long result = (high << 8) | low;

  result = 1125300L / result;         // Calcula Vcc (mV); 1125300
= 1.1*1023*1000

  datos[4] = float(result / 1000.0);
}

void LEDs() {
  Serial.println();
  Serial.println("Análisis de luz solar");
  for (int x = 0; x < 5; x++) {
    //Para inicializar las señales
    pinMode(14 + x, OUTPUT);
    digitalWrite(14 + x, LOW);
    delay(3);
    Muestra[0] = analogRead(x);

    //Toma de medidas
    noInterrupts();
    pinMode(14 + x, INPUT);
    Muestra[0] = analogRead(x); //r1
    Muestra[1] = analogRead(x); //r2
    Muestra[2] = analogRead(x);
    Muestra[3] = analogRead(x);
    delay(1);
    Muestra[4] = analogRead(x);
    delay(2);
```

```
Muestra[5] = analogRead(x);
delay(7);
Muestra[6] = analogRead(x);
interrupts();
delay(20);
Muestra[7] = analogRead(x);
delay(70);
Muestra[8] = analogRead(x);
delay(200);
Muestra[9] = analogRead(x);
delay(700);
Muestra[10] = analogRead(x); //r11

if (x == 0)
{
  for (int i = 0; i < 12; i++)
  {
    Rojo[i] = Muestra[i];
  }
  Rojo[11] = 1;
}
if (x == 1)
{
  for (int i = 0; i < 12; i++)
  {
    Verde[i] = Muestra[i];
  }
  Verde[11] = 2;
}
if (x == 2)
{
  for (int i = 0; i < 12; i++)
  {
    Azul[i] = Muestra[i];
  }
  Azul[11] = 3;
}
if (x == 3)
{
  for (int i = 0; i < 12; i++)
  {
    IR[i] = Muestra[i];
  }
  IR[11] = 4;
}
if (x == 4)
{
  for (int i = 0; i < 12; i++)
  {
    UV[i] = Muestra[i];
  }
  UV[11] = 5;
}
```

```
  }
}

void envio_datos() {
  radio.stopListening();
  radio.write(Rojo, sizeof(Rojo));
  radio.write(Verde, sizeof(Verde));
  radio.write(Azul, sizeof(Azul));
  radio.write(IR, sizeof(IR));
  radio.write(UV, sizeof(UV));
  radio.write(datos, sizeof(datos));
  radio.write(datos, sizeof(datos));
  radio.startListening();

}

typedef enum { wdt_16ms = 0, wdt_32ms, wdt_64ms, wdt_128ms,
wdt_250ms,   wdt_500ms,   wdt_1s,   wdt_2s,   wdt_4s,   wdt_8s}
wdt_prescalar_e;

void setup_watchdog(uint8_t prescalar);
void do_sleep(void);

const short sleep_cycles_per_transmission = 7;
volatile         short         sleep_cycles_remaining         =
sleep_cycles_per_transmission;

void setup() {
  radio.begin();                //Inicializamos el NRF24L01
  radio.openWritingPipe(ID[0]); //Abrimos un canal de escritura

  pinMode(5, INPUT);            // Para contar las oscilaciones
  pinMode(4, OUTPUT);          // Para el control de la oscilación
  digitalWrite(4, LOW);         // Para la oscilación

  Serial.begin(57600);
  setup_watchdog(wdt_8s);

  PIN_SAVE_ACTIVE = false;
  pinMode(PIN_SAVE, INPUT_PULLUP);
  pinMode(chipSelect, OUTPUT);
  pinMode(2, OUTPUT); //Buzzer
  save = false;
  SDOK = true;
  if (!SD.begin(chipSelect)) {
    digitalWrite(2, HIGH);
    delay(2000);
    digitalWrite(2, LOW);
    SDOK = false;
    Serial.println("MAL");
    return;
  }
  else {
```

```
    dataFile = SD.open("datalog.csv", FILE_WRITE);
    digitalWrite(2, HIGH);
    delay(500);
    digitalWrite(2, LOW);
    Serial.println("BIEN");

    //Cada vez que reinicie se escribirá esto y no interesa
    /*datosTFG = "Muestras";
    datosTFG += ';';
    for (int j = 0; j < 10; j++) {
      datosTFG += String(j + 1);
      datosTFG += ';';
    }
    datosTFG += "11";
    if (dataFile) dataFile.println(datosTFG);
    else SDOK = false;*/
  }

}

void loop() {

  //Para saber que no se ha apagado
  /*digitalWrite(2, HIGH);
  delay(300);
  digitalWrite(2, LOW);*/

  //Toma de datos
  seq = seq + 1;
  datos[0] = seq;
  temperatura();
  FDR();
  get_vcc();
  LEDs();

  envio_datos();

  //HACER STRING
  /*
  datosTFG = "Rojo";
  datosTFG += ';';
  datosTFG += String(Rojo[0]);
  datosTFG += ';';
  for (int i = 1; i < 10; i++) {
    datosTFG += String(Rojo[i]);
    datosTFG += ';';
  }
  datosTFG += String(Rojo[10]);
  datosTFG += '\n';

  datosTFG += "Verde";
  datosTFG += ';';
  for (int i = 0; i < 10; i++) {
```

```
  datosTFG += String(Verde[i]);
  datosTFG += ';';
}
datosTFG += String(Verde[10]);
datosTFG += '\n';

datosTFG += "Azul";
datosTFG += ';';
for (int i = 0; i < 10; i++) {
  datosTFG += String(Azul[i]);
  datosTFG += ';';
}
datosTFG += String(Azul[10]);
datosTFG += '\n';

datosTFG += "IR";
datosTFG += ';';
for (int i = 0; i < 10; i++) {
  datosTFG += String(IR[i]);
  datosTFG += ';';
}
datosTFG += String(IR[10]);
datosTFG += '\n';

datosTFG += "UV";
datosTFG += ';';
for (int i = 0; i < 10; i++) {
  datosTFG += String(UV[i]);
  datosTFG += ';';
}
datosTFG += String(UV[10]);
datosTFG += '\n';

datosTFG += "Datos";
datosTFG += ';';
for (int i = 0; i < 4; i++) {
  datosTFG += String(datos[i]);
  datosTFG += ';';
}
datosTFG += String(datos[4]);
*/

datosTFG = String(datos[0]);    //Secuencia
datosTFG += ';';
datosTFG += String(Rojo[0]);
datosTFG += ';';
datosTFG += String(Rojo[3]);
datosTFG += ';';
datosTFG += String(Rojo[10]);
datosTFG += ';';
datosTFG += String(Verde[3]);
datosTFG += ';';
datosTFG += String(Verde[6]);
```

```
  datosTFG += ';';
  datosTFG += String(Verde[10]);
  datosTFG += ';';
  datosTFG += String(Azul[3]);
  datosTFG += ';';
  datosTFG += String(Azul[10]);
  datosTFG += ';';
  datosTFG += String(IR[0]);
  datosTFG += ';';
  datosTFG += String(IR[10]);
  datosTFG += ';';
  datosTFG += String(UV[6]);
  datosTFG += ';';

  datosTFG += String(datos[1]);
  datosTFG += ';';
  datosTFG += String(datos[2]);
  datosTFG += ';';
  datosTFG += String(datos[3]);
  datosTFG += ';';
  datosTFG += String(datos[4]);
  datosTFG += ';';




  if (dataFile) dataFile.println(datosTFG);
  else SDOK = false;

  dataFile.close();                              //Si muestreo cada
mucho que guarde
  dataFile = SD.open("datalog.csv", FILE_WRITE);

  while (sleep_cycles_remaining > 0) {
    do_sleep();


    if (digitalRead(PIN_SAVE) == PIN_SAVE_ACTIVE) {
      dataFile.close();
      digitalWrite(2, HIGH);
      delay(1000);
      digitalWrite(2, LOW);
      save = true;
    }
  }
  sleep_cycles_remaining = sleep_cycles_per_transmission;

}

void setup_watchdog(uint8_t prescalar)
{
  prescalar = min(9, prescalar);
  uint8_t wdtcsr = prescalar & 7;
```

```
  if ( prescalar & 8 )
    wdtcsr |= _BV(WDP3);

  MCUSR &= ~_BV(WDRF);
  WDTCSR = _BV(WDCE) | _BV(WDE);
  WDTCSR = _BV(WDCE) | wdtcsr | _BV(WDIE);
}

ISR(WDT_vect)
{
  --sleep_cycles_remaining;
}

void do_sleep(void)
{
  set_sleep_mode(SLEEP_MODE_PWR_DOWN); //Elegimos el "sleep mode"
que deseemos
  sleep_enable();

  sleep_mode();                         //En esta función entra el
sistema en modo durmiente

  sleep_disable();                     //Lo primero que se ejecuta
después del desbordamiento del Watchdog
}
```

7.4. Código Receptor con SD

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <SD.h>

//SD
#define PIN_SAVE  3
#define chipSelect  5
File dataFile;

//nRF24L01
RF24 radio(9, 10); //RF24 radio(CE_PIN, CSN_PIN)
const  uint64_t  ID[2]  =  {  0xF0F0F0F0E1LL,  0xF0F0F0F0D2LL  };
//Variable con la dirección del canal por donde se va a transmitir

//Variables
int Rojo[12], Verde[12], Azul[12], IR[12], UV[12];
float datos[6], secuencia_OLD;
int OLD_IR , OLD_Verde;
boolean    Sol_directo    =    false,    Cielo_azul    =    false,
Ligera_contaminacion = false, Amanecer = false, Anochecer = false;
boolean Nubes_ligeras = false, Nubes_densas = false, Sombra_pobre
= false, Noche = false, Neblina = false, flag = false;
String datosTFG;
bool save, SDOK, PIN_SAVE_ACTIVE;

void setup()
{
  radio.begin();                     //Inicializamos el NRF24L01
  radio.openReadingPipe(1, ID[0]);  //Leo informacion del emisor
  radio.startListening();

  Serial.begin(57600);

  PIN_SAVE_ACTIVE = false;
  pinMode(PIN_SAVE, INPUT_PULLUP);
  pinMode(chipSelect, OUTPUT);
  pinMode(2, OUTPUT); //Buzzer
  save = false;
  SDOK = true;
  if (!SD.begin(chipSelect)) {
    digitalWrite(2, HIGH);
    delay(1500);
    digitalWrite(2, LOW);
    SDOK = false;
    Serial.print("MAL");
    return;
  }
  else {
    dataFile = SD.open("datalog.csv", FILE_WRITE);
```

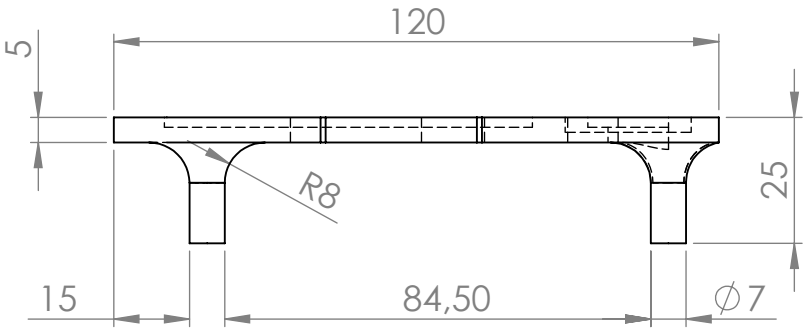
```
    digitalWrite(2, HIGH);
    delay(500);
    digitalWrite(2, LOW);
    Serial.print("BIEN");
  }
}

void loop() {


  if ( radio.available() ) {

    radio.read(Rojo, sizeof(Rojo));
    if (Rojo[11] == 1) {
      datosTFG = String(Rojo[0]);
      datosTFG += '\t';
      for (int i = 1; i < 10; i++) {
        datosTFG += String(Rojo[i]);
        datosTFG += '\t';
      }
      datosTFG += String(Rojo[10]);
      datosTFG += '\n';
    }
    delay(10);

    radio.read(Verde, sizeof(Verde));
    if (Verde[11] == 2) {
      for (int i = 0; i < 10; i++) {
        datosTFG += String(Verde[i]);
        datosTFG += '\t';
      }
      datosTFG += String(Verde[10]);
      datosTFG += '\n';
    }
    delay(10);

    radio.read(Azul, sizeof(Azul));
    if (Azul[11] == 3) {
      for (int i = 0; i < 10; i++) {
        datosTFG += String(Azul[i]);
        datosTFG += '\t';
      }
      datosTFG += String(Azul[10]);
      datosTFG += '\n';
    }
    delay(10);

    radio.read(IR, sizeof(IR));
    if (IR[11] == 4) {
      for (int i = 0; i < 10; i++) {
        datosTFG += String(IR[i]);
        datosTFG += '\t';
      }
```

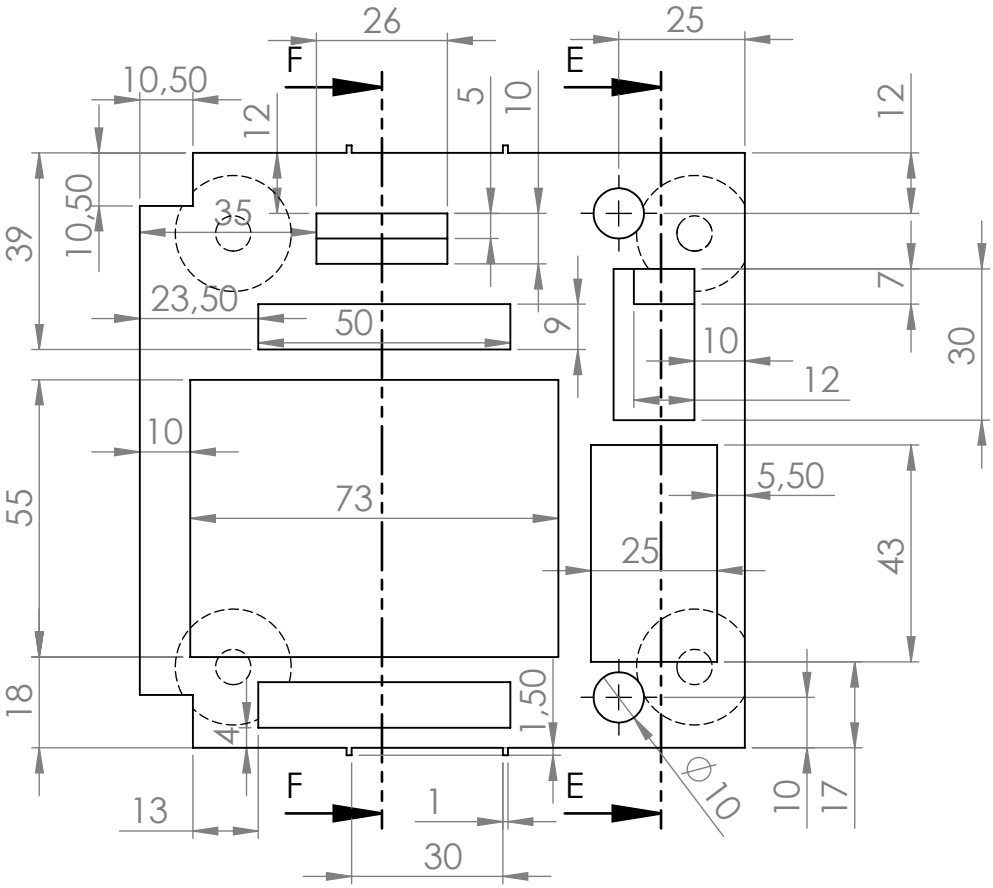
```
      datosTFG += String(IR[10]);
      datosTFG += '\n';
    }
    delay(10);

    radio.read(UV, sizeof(UV));
    if (UV[11] == 5) {
      for (int i = 0; i < 10; i++) {
        datosTFG += String(UV[i]);
        datosTFG += '\t';
      }
      datosTFG += String(UV[10]);
      datosTFG += '\n';
    }
    delay(10);

    radio.read(datos, sizeof(datos));
    if (datos[5] == 6)
    {
      for (int i = 0; i < 5; i++) {
        datosTFG += String(datos[i]);
        datosTFG += '\t';
      }
      datosTFG += String(float(datosTFG.length()));
      datosTFG += '\r';
      Serial.print(datosTFG);
      flag = true;
    }
    delay(10);

  }

  if (flag) {
    Serial.print("GUARDANDO DATOS");
    if (dataFile) dataFile.println(datosTFG);
    else SDOK = false;

    //EVALUACION DE DATOS//

////////////////////////////////////////////////////////////////
////////////////////////////////////
    if (IR[0] > 50) {
      Sol_directo = true;
    }

    if (Rojo[0] >= 15 && Rojo[0] > IR[0] * 0.8)
    {
      Cielo_azul = true;
    }

    if (Rojo[3] < 20)
    {
      if (Rojo[10] > 15 && Verde[3] > 0) {
```

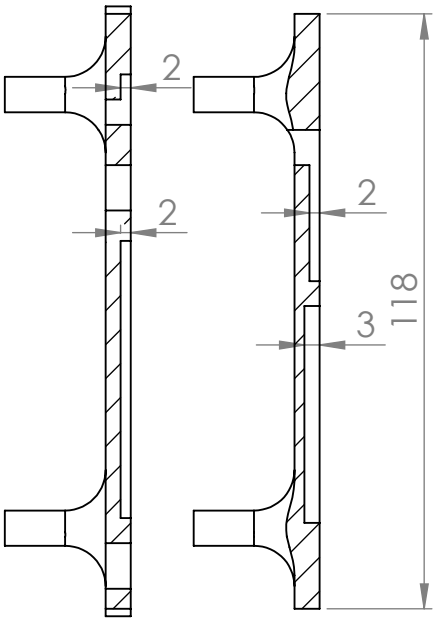
```
        Ligera_contaminacion = true;
      }
      if (Rojo[10] > 5 && Rojo[10] * 3 > IR[10]) {
        Ligera_contaminacion = true;
      }
      if (IR[10] < 12 || OLD_IR < 12) {
        Noche = true;
      }
    }

    if (Rojo[3] < 20 && Verde[10] > 30) {
      if (IR[10] < 55 && IR[10] > OLD_IR) {
        if (Verde[6] > OLD_Verde) {
          Amanecer = true;
        }
      }
      if (Verde[6] > OLD_Verde) {
        Anochecer = true;
      }
    }

    if (IR[0] > Rojo[0] && Rojo[0] > 4)
    {
      if (Verde[3] > 35 && IR[0] < 100) {
        Neblina = true;
      }
      if (Verde[3] > 15 && IR[0] < 100) {
        Nubes_ligeras = true;
        if (Azul[3] > IR[0] && Azul[3] > 10) {
          Nubes_ligeras = false;
          Nubes_densas = true;
        }
      }
    }

    if (Azul[3] > IR[0] && Azul[3] < 30)
    {
      Sombra_pobre = true;
    }

    OLD_IR = IR[10];
    OLD_Verde = Verde[6];

////////////////////////////////////////////////////////////////
////////////////////////////////////

    if (digitalRead(PIN_SAVE) == PIN_SAVE_ACTIVE) {
      Serial.println("GUARDANDO");
      dataFile.close();
      save = true;
    }
```

```
    flag = false;
  }
}
```

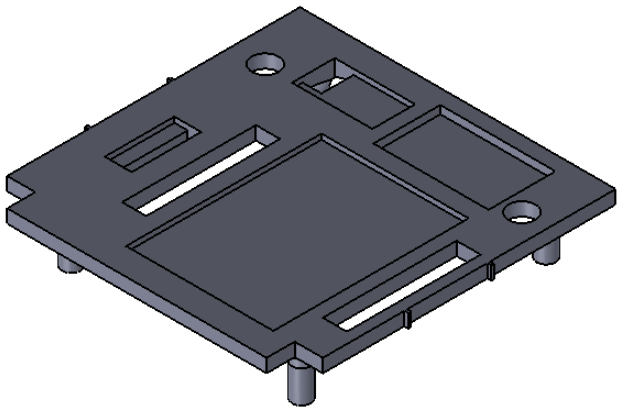
7.5. Plano del soporte del emisor





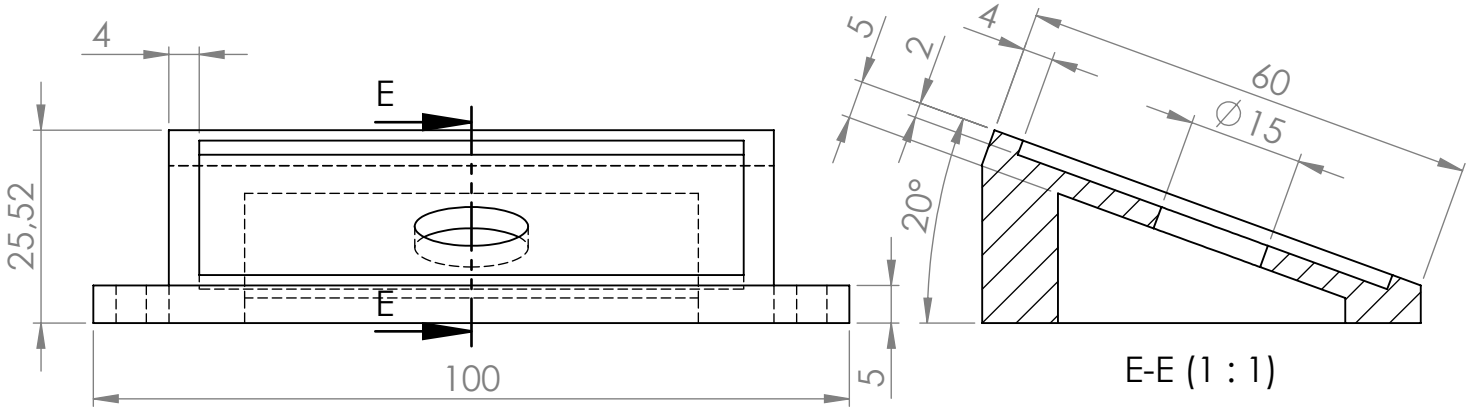


F-F (2 : 3) E-E (2 : 3)

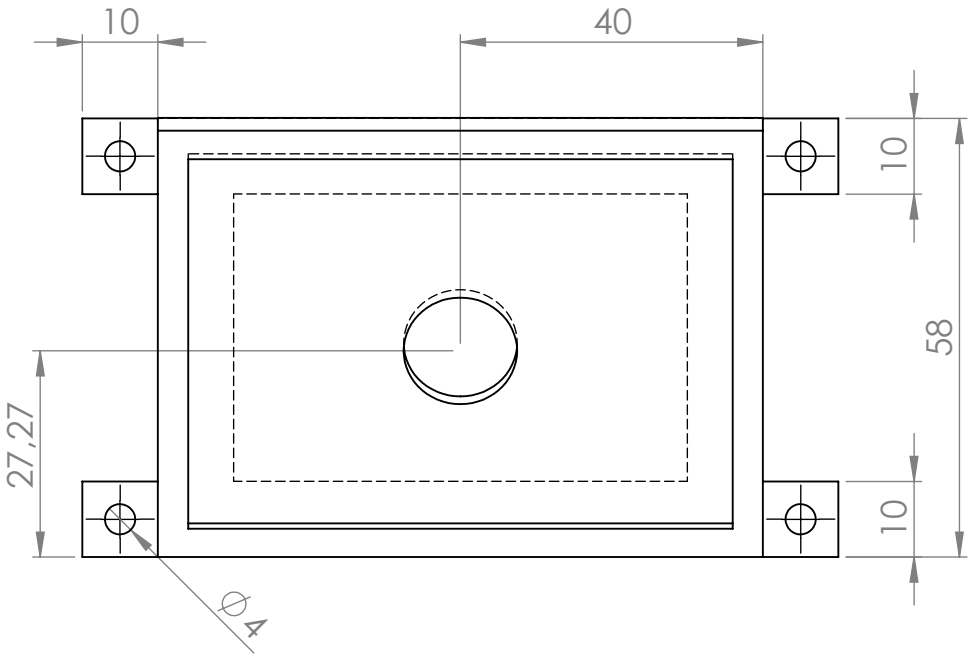


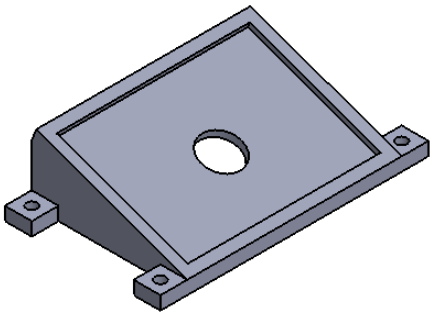
	UNIVERSIDAD POLITECNICA DE CARTAGENA	TRABAJO FINAL DE GRADO		
		TÍTULO: SOPORTE PARA EMISOR		
PLANTILLA A4V	FIRMA	NOMBRE: FRANCISCO FRANCO MARTÍNEZ	NUMERO PLANO	Nº 2
		FECHA: 11/05/2017	ESCALA: 2:3	GRADO GITI

7.6. Plano del panel solar



E-E (1 : 1)





UNIVERSIDAD POLITECNICA DE CARTAGENA	TRABAJO FINAL DE GRADO			
	TÍTULO: SOPORTE PARA PANEL DE LUZ SOLAR			
PLANTILLA A4V	FIRMA	NOMBRE: FRANCISCO FRANCO MARTÍNEZ	NUMERO PLANO	Nº 1
		FECHA: 03/04/2017	ESCALA: 1:1	GRADO GITI