UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Escuela Técnica Superior de Ingeniería del Diseño

DISEÑO, DESARROLLO Y VALIDACIÓN DE LA ETAPA DE POTENCIA DE UN SISTEMA DE ESTIMULACIÓN OPTOQUIMIOGENÉTICA DE ALTA INTENSIDAD

TRABAJO FINAL DEL
Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR
Coronel Montesinos, Juan Diego

TUTORIZADO POR
Ibáñez Civera, Francisco Javier

COTUTORIZADO POR
Monreal Trigo, Javier

CURSO ACADÉMICO 2020/2021

ESTRUCTURA

I – MEMORIA DEL PROYECTO

II – PLANOS

III – PLIEGO DE CONDICIONES

IV – PRESUPUESTO

I – MEMORIA DEL PROYECTO

Tabla de contenido

Tabla de figuras

1. INTRODUCCIÓN

En esta memoria se detalla el proceso de diseño, desarrollo y validación de una etapa de potencia para un sistema de alta intensidad dedicado a la estimulación optoquimiogenética, así como la interfaz de usuario dedicada a su control en tiempo real.

La finalidad es contribuir al desarrollo del dispositivo OSIVE HI, del Instituto de Reconocimiento Molecular y Desarrollo Tecnológico (IDM), que será usado para estimular determinadas células presentes en un sistema biológico gracias a dos punteros lumínicos comunicados mediante protocolo CAN.

1.1 Contextualización de la disciplina

Si bien las técnicas relativas a los procesos de manipulación celular son diversas, una que augura resultados prometedores pese a su corta vida (Joshi, Rubart, & Zhu, 2020) es la optogenética, elegida en 2010 como "Método del Año" por la revista de investigación interdisciplinar *Science* (Method of the Year 2010, 2011).

Este método de neuromodulación permite, entre otras, controlar el comportamiento de determinadas células presentes en un sistema con un nivel de precisión nunca visto en métodos más tradicionales, como es el caso de la estimulación eléctrica.

Figura 1. De izquierda a derecha: Estimulación eléctrica en neuronas, estimulación optogenética sobre una neurona con opsina que responde a la luz azul, inhibición optogenética sobre una neurona con opsina que responde a la luz amarilla. Fuente: Sociedad Española de Bioquímica y Biología Molecular.

Para conseguirlo, se utiliza un tipo específico de proteína de membrana, denominada opsina, que se introduce en las *células diana* (aquellas que quieren manipularse o monitorizarse). Alojada la opsina en la célula, a esta última se proyecta una estimulación lumínica con una potencia y longitud de onda determinadas, lográndose una alteración en su potencial de membrana. Estas diferencias de potencial provocan los comportamientos requeridos en la célula.

A fin de cuentas, puede establecerse una analogía entre este proceso y el hecho de abrir y cerrar un interruptor. Por ejemplo, aplicándose luz azul en la opsina canalrodopsina-2 (ChR-2, descrita originalmente en algas) se consigue su activación; lo que permite el flujo de cationes a través suyo, despolarizando la membrana de la célula diana y generando potenciales de acción equivalentes a una señal de encendido.

Por otro lado, aplicándose luz amarilla a la proteína halorodopsina, se desencadena la afluencia de iones de cloro cargados negativamente a través suyo, hiperpolarizando la

membrana de la célula diana y reduciendo las probabilidades de que esta genere potenciales de acción (equivalente a una señal de apagado) (Perea, 2016).

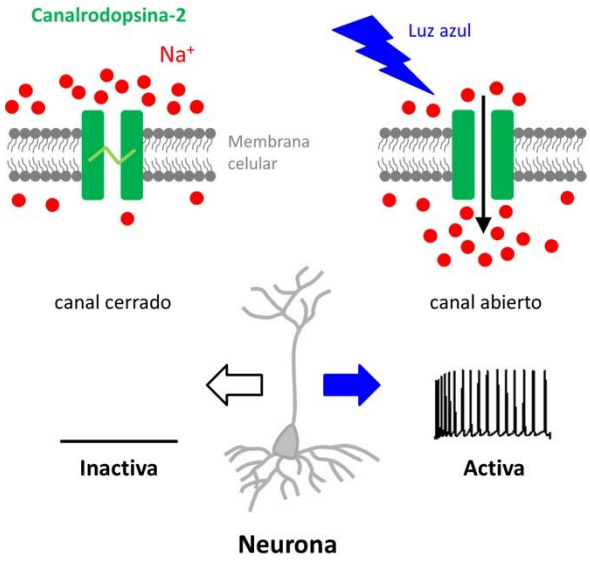


Figura 2. Foto-activación de canalrodopsina-2 y generación de potenciales de acción en neuronas. Fuente: Sociedad Española de Bioquímica y Biología Molecular.

Introduciéndose ambas opsinas simultáneamente en la célula, por ejemplo una neurona, se permite un control rápido y de gran precisión gracias al intercambio de las diferentes luces, eludiendo la necesidad de fármacos o productos químicos adicionales. Se consigue así una serie de ventajas destacables:

- Precisión espacial.
- Precisión temporal.
- Selectividad celular.

A unos niveles que, con otros métodos, eran impensables. En particular, para la aplicación que abarca este proyecto se obtendrán grandes mejoras en precisión temporal.

El hacer más efectivos y eficientes procesos complejos como la reparación del tejido nervioso tras un traumatismo, o el tratamiento de diversas enfermedades neurológicas, son algunas de las aplicaciones que este método brinda.

1.2 Contextualización del proyecto

Como se ha comentado anteriormente, el presente proyecto se encuentra enmancarcado en el desarrollo del dispositivo OSIVE HI (*Optochemogenetic Stimulation for In-Vitro Experimentation, High Irradiance*). Este se encuentra integrado en el plan nacional de investigación RTI2018-100910-B-C43-AR concedido por la Agencia Estatal de Investigación al IDM, titulado *Desarrollo de Plataformas de Detección y Terapeuticas para Aplicaciones Biomedicas Basadas en Dispositivos Electrónicos.*

OSIVE (Monreal Trigo, 2021) es una familia de dispositivos desarrollados *ad hoc* para satisfacer necesidades de estimulación, en lo que refiere a la investigación en el área de la optoquimiogenética, como su nombre indica, procedente de las siglas en inglés para Estimulación Optoquimiogenética para la Experimentación In-Vitro. Hasta la fecha se han desarrollado los equipos:

- OSIVE (Terrés-Haro & Monreal Trigo, 2021).
- OSIVE Slim (pendiente de publicar), de estimulación de baja irradiancia (<20 mW/cm^2).
- OSIVE IMS (Beltrán-Morte, 2021), que permite la medida en tiempo real de la estimulación lumínica monocromática con gran precisión.

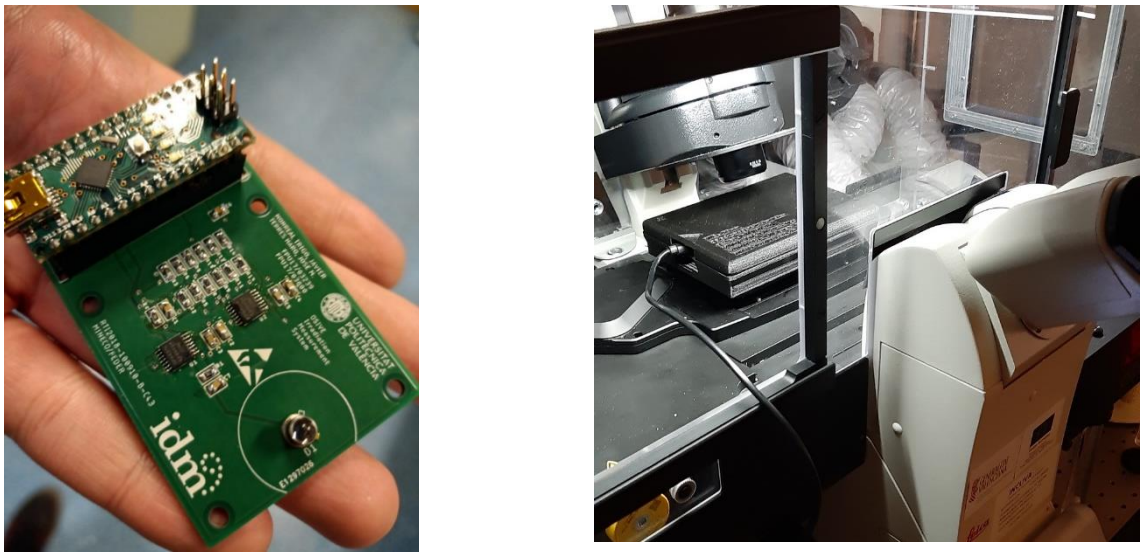


Figura 3. A la izquierda, PCB de OSIVE. A la derecha, configuración para experimento con OSIVE. Fuente: Javier Monreal, IWOSMOR XIV.

Figura 4. Prototipado de OSIVE HI Power Station (100 W, 20 A, 5 V). Fuente: Propia

El último de los requerimientos del desarrollo de los experimentos necesita de un aumento de dos órdenes de magnitud de la irradiación lumínica suministrada por los equipos, por lo que se ha hecho patente la necesidad de añadir a la familia un nuevo dispositivo que cumpla tales requerimientos de alta potencia (10 mW/mm^2). Este es OSIVE HI. El susodicho se encuentra conformado por las siguientes partes, algunas visibles en la Figura 4:

1. Una interfaz gráfica de usuario para comandar el sistema desde un ordenador personal.
2. Una unidad de control central, cuya función es interpretar los comandos transmitidos desde el ordenador personal a los punteros inteligentes descritos posteriormente.
3. Una unidad de potencia para suministrar tensión y corriente contínua a susodichos punteros.
4. Punteros inteligentes, que integran un microcontrolador para interpretar la orden emitida por la unidad de control central y aplicar el patrón de estimulación indicado a los fotoemisores. Cada puntero cuenta con cuatro LEDs de alta potencia, con un consumo nominal de 2 A, y un sistema óptico de colimación.

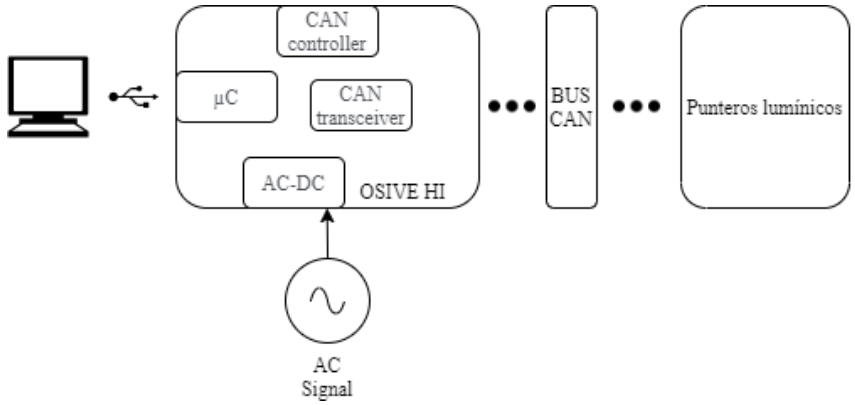


Figura 5. Diagrama general del sistema. Fuente: Propia.

Los tres primeros puntos de la enumeración anterior son desarrollados en el presente proyecto. El cuarto, correspondiente al desarrollo de los punteros inteligentes, queda fuera del alcance de este trabajo final de grado.

1.3 Motivación

Se exponen en el actual punto los alicientes que han provocado el interés para llevar a cabo este proyecto; tanto desde un punto de vista técnico o tecnológico, como desde una perspectiva enfocado a lo social.

1.3.1 Motivación de evolución científica

Como se ha mencionado en el anterior apartado de contextualización, la disciplina se encuentra en vías de desarrollo dado su corto tiempo de vida y la poca investigación dedicada hasta el momento. Contextos y situaciones como esta y similares son indicadas para la propuesta de soluciones y métodos que ayuden a su avance y evolución.

Dada su existente demanda de un equipo capaz de estimular determinadas células usando como estímulo luz con determinadas longitudes de onda, se propone al equipo de investigación especializado en optoquimiogenética del *Centro de Investigación Príncipe Felipe* la solución contenida en este proyecto. Como resultado, el equipo podría estudiar de manera aislada el comportamiento de las células excitadas, extrapolando los resultados al desarrollo de aplicaciones en el ámbito de la biomedicina.

1.3.2 Motivación Médica y Social

Los avances de la medicina durante las últimas décadas son innegables. No obstante, sigue existiendo ese grupo de las "enfermedades raras", conformado por entre 5000 y 8000 tipos de enfermedad (Kaplan, y otros, 2013), y que afectan alrededor de al 7% de la población mundial, algo más de 490 millones de personas (Organización de las Naciones Unidas (ONU), 2011).

Si bien conforme el tiempo pasa se consigue llegar al origen de este tipo de enfermedades, a veces resulta complicado lograr una solución que potencialmente pueda generar algún tipo de tratamiento.

Gracias a los esfuerzos de la comunidad científica, la optoquimiogenética se ha abierto camino como una de las protagonistas a la hora de resolver esta problemática; ofreciéndose tratamientos para estas enfermedades a través de la fotoestimulación de células (por ejemplo, neuronas, en el caso de una enfermedad neurológica) y la modificación genética de estas mismas.

Para el autor de este documento, existe tanto una curiosidad por la disciplina, como una necesidad de aportar a su evolución. Entre otros motivos, por intentar lograrse cierta flexibilización en cuanto oportunidades de tratamiento para la población afectada. Si ya el número de personas que las sufren es considerable ajustándonos a la definición de Enfermedad Rara por parte de la OMS, no es necesario quedarse solo en este tipo de enfermedades para detectar problemáticas actuales en cuanto a calidad de tratamientos. Pueden revisarse indicadores cuantitativos de enfermedades más conocidas como la esquizofrenia, la epilepsia o la esclerosis múltiple (Merck KGaA, 2019), cuyos valores no permiten tratarlas como tal.

2. OBJETIVOS

Los objetivos que tratan de alcanzarse con este proyecto serán expuestos en el presente punto, haciéndose distinción entre cada una de las principales partes del sistema o dispositivo.

2.1 Etapa de potencia

La etapa de potencia otorgará energía eléctrica a los punteros lumínicos. Los valores necesarios para la estación OSIVE HI son, de acuerdo a las especificaciones propuestas para el correcto funcionamiento de los punteros, una tensión de salida de 5 V y una corriente de 20 A, ya que cada puntero necesitará 8 A y se ha escogido un valor correspondiente al sobredimensionamiento de los requisitos en un 20% (valor comercial inmediatamente superior). Este será, por tanto, el objetivo principal: otorgar una señal de alimentación de corriente continua 5 V y 20 A.

Las señales lumínicas de estimulación aplicadas por los punteros deben cumplir una serie de características en cuanto a potencia y luminosidad, entre otras. La señal de salida de la etapa de potencia, por tanto, deberá ser lo suficientemente estable para alimentar los componentes de etapas posteriores de manera adecuada, causando a las salidas de estos las señales requeridas.

Por otro lado, con el fin de que puedan atenderse las principales necesidades de mantenimiento, reparación o ampliación, la sencillez en las soluciones adoptadas es un criterio que se mantendrá presente durante todo el proceso de diseño y desarrollo. Entran

dentro de estas tareas, por ejemplo, el reemplazo de componentes dañados. Así pues, deberá atenderse a necesidades como el uso de pocos componentes, que estén estandarizados y tengan valores normalizados, y que estos sean del menor coste posible sin poner en riesgo las especificaciones a conseguir. También se incorporará el objetivo de hacer la etapa de potencia modular, pudiéndose conectar en paralelo varias unidades comandadas por la misma unidad central e interfaz gráfica si su uso lo precisa.

Inicialmente, no existen unos requisitos mínimos en cuanto a tamaño o portabilidad. Pese a ello, se pretenderá que toda la etapa de potencia se mantenga correctamente acondicionada y protegida.

2.2 Interfaz gráfica

Una interfaz gráfica ejecutada en un ordenador personal (PC) servirá al usuario para acondicionar, a los valores requeridos y de manera directa y sencilla, la señal a aplicar por cada uno de los punteros. El objetivo principal, por tanto, será la fácil configuración de la señal de salida de los fotoemisores LED.

Esta deberá ser fácil de usar, y por tanto autoexplicativa de lo que se pretende. El diseño también se ejecutará para que, sean cuales sean las habilidades del usuario con un PC, este pueda servirse de la propia aplicación. Además, el código de esta debe ser legible, con las suficientes indicaciones de los métodos y variables, y a ser posible escrito en un lenguaje que no requiera de una curva de dificultad excesivamente elevada. Así, cualquier modificación o corrección necesaria posteriormente podrá realizarse sin demasiado esfuerzo.

2.3 Protocolo de comunicaciones

El protocolo de comunicaciones será el encargado de, como su propio nombre indica, comunicar a la unidad central del sistema los requisitos introducidos por el usuario en la interfaz gráfica.

Siendo este el objetivo principal, deberá optarse por un método lo suficientemente robusto como para que los mensajes transmitidos y recibidos sean fiables, garantizando así la integridad de la información en intercambio. Sin abandonar las líneas generales del sistema, la implementación debe ser a su vez sencilla y procurándose un bajo impacto en el coste de este desarrollo.

3. MÉTODO Y MATERIALES

A continuación, se detallará el método de desarrollo usado para este proyecto dadas sus diferentes partes principales, además de presentarse una mejor relación entre los principios de funcionamiento y las soluciones aportadas.

3.1 Método de desarrollo

Para el desarrollo de este proyecto, se ha utilizado el método de desarrollo iterativo e incremental. Este método subdivide el trabajo en unidades denominadas etapas, que se organizan conforme a las diferentes funciones principales, ya sea el diseño y la construcción física o el desarrollo del software para la conectividad de elementos.

Realizando el desarrollo de esta manera, puede lograrse una organización más eficiente, incorporándose al resultado final cada etapa una vez concluida y haciendo cambios en las decisiones que afectan al desempeño, en consecuencia del análisis de ejecución de cada parte. Esto se resume en mejoras de organización.

3.2 Principios de funcionamiento

3.2.1 Fuentes conmutadas AC/DC

El convertidor AC/DC se encargará de extraer energía eléctrica de la red y entregarla acorde a los valores necesarios para la estación OSIVE HI, siendo estos una tensión de salida de 5V y una corriente de 20A.

Siguiendo el diagrama de bloques funcionales típico para una fuente conmutada AC/DC, se dispondrá información sobre las diferentes etapas que la conforman.

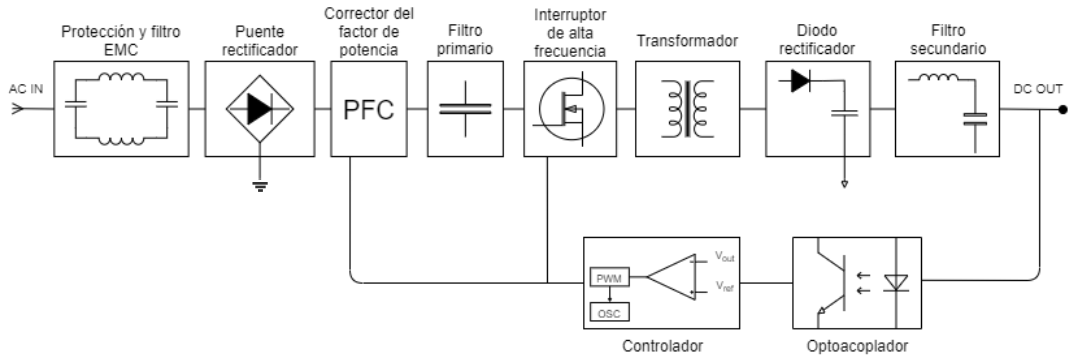


Figura 6. Ejemplo de diagrama de bloques funcionales para una fuente conmutada AC/DC (SMPS). Fuente: Propia

3.2.1.a Filtro EMC de entrada

De sus siglas en inglés *Electromagnetic Compatibility*, es una etapa que debe estar introducida por legislación industrial en los dispositivos electrónicos. Esta etapa procura que las interferencias y sobretensiones (ya sean permanentes o momentáneas) presentes en la red eléctrica no pasen hacia la fuente. Del mismo modo, también previene de que las señales problemáticas para la red, causadas por averías en los componentes, no pasen a la red eléctrica y provoque daños.

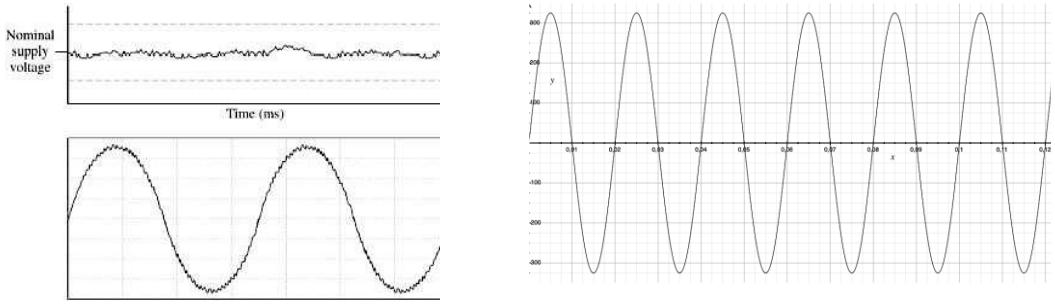


Figura 7. A la izquierda, representación de la señal de ruido (arriba) y representación de la señal AC con ruido (abajo). Fuente: EE Times. A la derecha, señal de tensión alterna teórica de 50 Hz. Fuente: Propia

3.2.1.b Etapa de rectificación

Convierte los semiciclos negativos de la señal de red en semiciclos positivos. Esto se traduce en transformar la entrada de alterna en una tensión continua y pulsante.

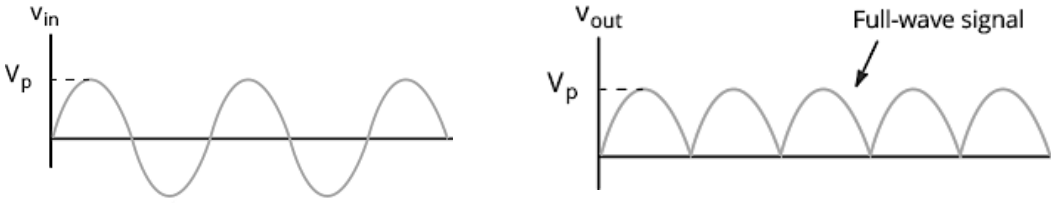


Figura 8. A la izquierda, representación de la señal de tensión alterna filtrada. A la derecha, representación de la misma señal a la salida de la etapa de rectificación. Fuente: https://www.monolithicpower.com

3.2.1.c Filtro primario

Amortigua la tensión continua pulsante que entrega el rectificador. Esto es, disminuye su rizado. Normalmente el filtro se compone de uno o varios condensadores en paralelo y a veces, se introduce una inductancia o bobina.

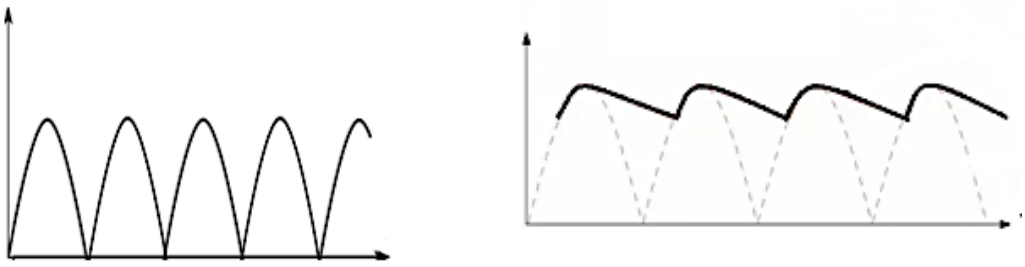


Figura 9. A la izquierda, representación de la señal a la salida de la etapa de rectificación. A la derecha, representación de la señal a la salida del filtro primario. Fuente: https://www.monolithicpower.com

3.2.1.d Interruptor de alta frecuencia

En esta etapa, un conjunto de transistores conmutan a alta velocidad para trocear (la etapa también es comúnmente conocida por su anglicismo "*chopper*") la señal. Lo que está sucediendo es una transformación de la señal de continua (CC) a una señal de tensión alterna cuadrada. La ventaja de usar este tipo de señales es que al estar trabajando en altas frecuencias, la disipación de potencia es considerablemente menor a otros métodos tradicionales de diseño de fuentes de alimentación, como pueden ser las fuentes lineales.

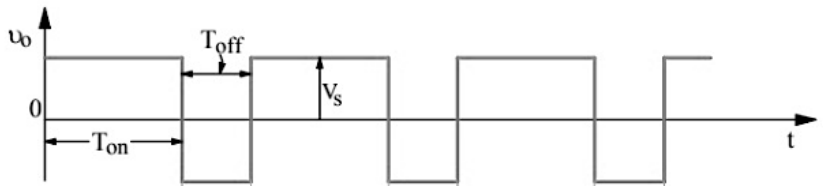


Figura 10. Representación de la señal a la salida de la etapa de switch. Fuente: https://electricalbaba.com

3.2.1.e Transformador

Modifica, en general reduciendo, el valor de voltaje pico-pico en la señal cuadrada de alterna generada. La eficiencia de la transmisión energética entre devanados es óptima con señales senoidales a la entrada. Al tratarse de un convertidor aislado, la señal de entrada en el devanado primario es cuadrada, pagando el precio del aislamiento con una reducción de la eficiencia de entorno al 10%. Una solución de compromiso ampliamente utilizada en las fuentes conmutadas de consumo, industriales y médicas.

3.2.1.f Diodo rectificador

Normalmente conformada por un diodo Schottky (o diodo de conmutación), recorta los semiciclos negativos de la señal alterna, consiguiéndose una señal de cuadrada continua respecto a la referencia (en las fuentes aisladas puede ser distinta a la tierra de la señal AC de entrada).

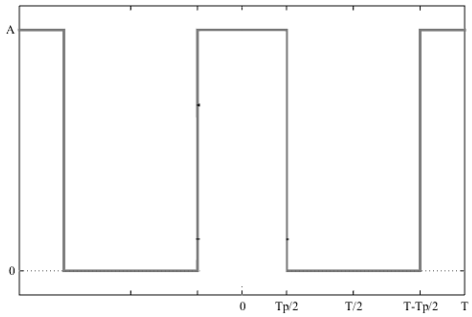


Figura 11. Representación de la señal de tensión rectificada. Fuente: https://lpsa.swarthmore.edu

3.2.1.g Filtro secundario

Conformado habitualmente por un conjunto de bobina y condensador de poca capacidad, puesto que al trabajar en altas frecuencias, no se necesita demasiada capacidad para dar resultado a una impedancia significativa. Cumple con la misma función que aquel filtro primario, que disminuía el rizado de la señal.

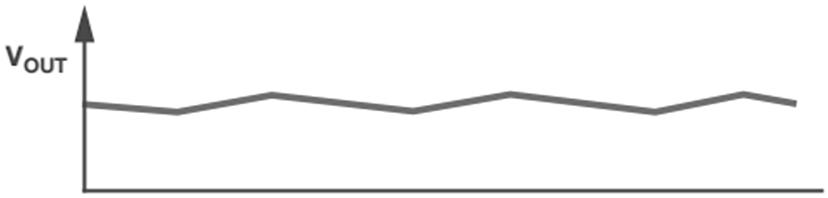


Figura 12. Representación de la señal de tensión de salida de la fuente de alimentación. Fuente: Analog Devices

3.2.1.h Optoacoplador

Se llega de esta manera al bloque de aislamiento, encargado de la realimentación del sistema. Por un lado, sus etapas, y en concreto la etapa del optoacoplador, son las encargadas de tomar parte de la señal de salida y mandarla al controlador para que este sea consciente de la misma.

Por otro lado, el mismo optoacoplador se encargará de mantener las dos partes del circuito aisladas. Existen otras opciones para obtener tal retroalimentación.

3.2.1.i Controlador

En esta etapa se produce la corrección de los niveles del voltaje de salida, para así mantener una señal de valor constante.

En primer lugar, el voltaje de salida se lleva a un comparador, que como su propia nombre indica, comparará la tensión de salida con una señal de referencia (normalmente referenciada como V_{ref}). Esta señal, usualmente, es un diente de sierra:

Una vez hecha la comparación, la señal de salida se manda a un transistor, normalmente un NMOS (de sus siglas en inglés *Negative-channel Metal-Oxide Semiconductor*) que sea capaz de conmutar a altas velocidades.

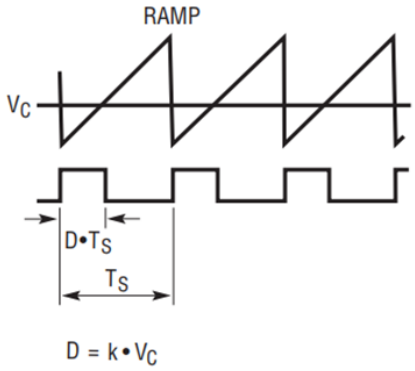


Figura 13. Representación del principio de generación de la señal de modulación de ancho de pulso (PWM, pulse-width modulation). Fuente: Analog Devices

La señal PWM se envía de vuelta a la etapa del interruptor a través de la realimentación y, si el convertidor dispusiese de ella, también a la etapa de corrección del factor de potencia. Así se conseguirá la linealidad y constancia del valor del voltaje a la salida de la fuente y los factores de potencia característicos de este tipo de fuentes de alimentación, respectivamente.

3.2.2 Protocolo de comunicaciones CAN (Controller Area Network)

El protocolo CAN, ISO-11898:2003, orquestará las comunicaciones entre la interfaz gráfica de usuario disponible en el ordenador personal, y la etapa de potencia del dispositivo OSIVE HI.

La información es distribuida mediante un bus, hacia nodos. Estos nodos son también conocidos en algunas áreas como Unidad de Control Electrónico (*Electronic Control Unit*, ECU).

Un estándar de mensaje bajo el protocolo CAN se estructura de la siguiente manera:

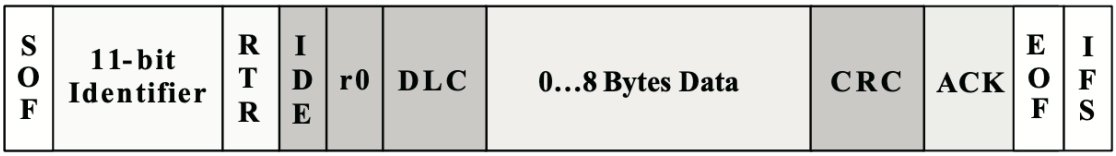


Figura 14. Diagrama que indica las partes de un mensaje estándar CAN con identificador de 11 bits. Fuente: Texas Instruments

3.2.2.a SOF (Start of Frame)

Marca el inicio de un mensaje, sincronizando los diferentes nodos para ponerles en sobreaviso de que un mensaje les va a llegar.

3.2.2.b Identificador CAN (CAN-ID)

Esta parte del mensaje contiene la prioridad del mismo, así como información funcional. Por ejemplo, en un sistema electrónico para un vehículo (que a su vez es el caso más frecuente de aplicación de este protocolo), la velocidad del mismo es información funcional. El CAN-ID puede variar de longitud, siendo hasta 29 los bits máximos. No obstante, en un CAN estándar tendrá un tamaño de 11 bits. Cuanto más bajo sea su valor en binario, mayor es la prioridad.

*3.2.2.c Petición de transmisión remota (**RTR**, Remote Transmission Request)*

Bit de carácter dominante cuando la información es requerida por otro nodo. Todos los nodos reciben la petición, pero el ID determina de qué nodo en específico se trata. Los datos enviados en la consecuente respuesta también son recuperados por todos los nodos, y usados por los que los necesitan. De esta manera se consigue la uniformidad de toda la información en el sistema.

*3.2.2.d Bit de extensión de identificador (**IDE**, Identifier Extension)*

Cuando este bit se encuentra en la trama, significa que se está transmitiendo un CAN-ID sin extensión.

*3.2.2.e Bit reservado (**r0**)*

Bit de reserva para su posible uso si en el futuro se producen correcciones em el estándar CAN.

*3.2.2.f Código de longitud de datos (**DLC**, data length code)*

Estos 4 bits de información contienen el número de bytes de datos que serán transmitidos.

3.2.2.g Campo de datos

Con una longitud de hasta 64 bits, representa la información transmitida.

*3.2.2.h **CRC** (Cyclic Redundancy Check)*

Esta parte del mensaje está conformada por 16 bits (15 bits más 1 carácter de sepación), y contendrá el valor de *checksum* de los datos transmitidos, con objetivo de detectar posibles errores. El valor de *checksum* asegura que la información que se está enviando es correcta, y en el caso de un mensaje estándar CAN, simboliza el número de bits transmitidos.

*3.2.2.i Hueco de acuse de recibo (**ACK**, Acknowledge slot)*

2 bits (1 bit para el reconocimiento y 1 bit para el carácter de separación) que se envían como recesivo (valor lógico alto) por defecto a todos los nodos. Cada nodo que detecta un mensaje satisfactorio lo sobreescribe por un bit dominante (valor lógico bajo). En cambio, si un error es detectado, el nodo receptor deja el bit a su valor original, descartándose el mensaje. De este modo, cada nodo reconoce la integridad de sus datos.

*3.2.2.j Fin de trama (**EOF**, End of Frame)*

Este campo, de 7 bits de longitud, marca el final de una trama o mensaje CAN.

*3.2.2.k **IFS** (Inter-frame spacing)*

Este espacio, también de 7 bits, contiene el tiempo requerido por el controlador para mover una trama correctamente recibida a su posición adecuada en un buffer de mensajes.

3.3 Software empleado en el desarrollo

Para el desarrollo de este proyecto, se ha dispuesto de una serie de herramientas *software* acorde con la persecución de los objetivos expuestos.

3.3.1 EAGLE

Desde su origen en 1988, EAGLE sigue siendo un estándar en cuanto a diseño de diagramas y PCBs (EDA, *Electronic Design Automation*). El popular software contiene soluciones para diseño de esquemáticos y PCB (*Printed Circuit Board*), ofreciendo algunas características como simulación SPICE (*Simulation Program with Integrated Circuits Emphasis*), autocomprobación de reglas electrónicas o renderizados 3D de los modelos PCB diseñados.

Si bien EAGLE es un software distribuido por Autodesk bajo licencias de suscripción, se dispone de una versión gratuita y limitada que incluye 2 hojas para esquemáticos, 2 capas de señal y diseño de placas de hasta 80 cm^2. Además, también se dispone de una licencia educativa con prestaciones superiores a la gratuita, aunque inferior a las prestaciones de la licencia profesional. Estas limitaciones se tendrán en cuenta para las fases de diseño, con el objetivo de impactar positivamente y de manera drástica al desarrollo del proyecto.

3.3.2 SolidWorks

Programa de diseño asistido por computador (CAD, *Computer-aidded design*) para modelado mecánico en 2D y 3D, que permite el modelado de piezas y la extracción posterior de planos técnicos y otra información necesaria para su producción. Este *software* se utilizará para el diseño y obtención de la información necesaria para producir la caja que protegerá y dotará de mayor portabilidad a la OSIVE HI.

El software está desarrollado por Dassault Systèmes y se distribuye bajo distintos tipos de licencia. A pesar de no existir una versión gratuita, se dispone de versiones de prueba de 15 a 30 días de duración, que dan acceso a la versión premium del producto. Además, la compañía creadora ofrece de manera gratuita recursos educativos para manejar los básicos del programa. Cabe añadir también las soluciones ofrecidas para emprendedores y *startups* a coste cero.

3.3.3 Ultimaker Cura

Una vez extraída la información necesaria del diseño de *SolidWorks* en un archivo portable 3D (STL, acrónimo de *STereoLithography*), este *software* permitirá crear una secuencia de instrucciones que una impresora 3D podrá aprovechar, materializando así la pieza. Se conseguirá una solución lo suficientemente válida y profesional sin recurrir a tecnologías de mayor coste.

3.3.4 Python

Python es un lenguaje de programación multiparadigma. Aunque la gran mayoría de usos profesionales se dan en base a la programación orientada a objetos, es perfectamente funcional para programación imperativa e incluso programación orientada a procesos.

Con una filosofía que hace hincapié en la legilibilidad de su código, un formato de distribución de código abierto, y el hecho de que se convierte, con el paso de tiempo, en el lenguaje más popular buscado en sitios de referencia como Stack Overflow; Python se convierte en un candidato ideal para adoptarlo a la solución de este proyecto a la hora de

crear una interfaz gráfica de usuario sencilla para el usuario final, fácil para hacer sencillas las futuras modificaciones, y lo suficientemente versátil para que pueda ejecutarse en gran cantidad de equipos.

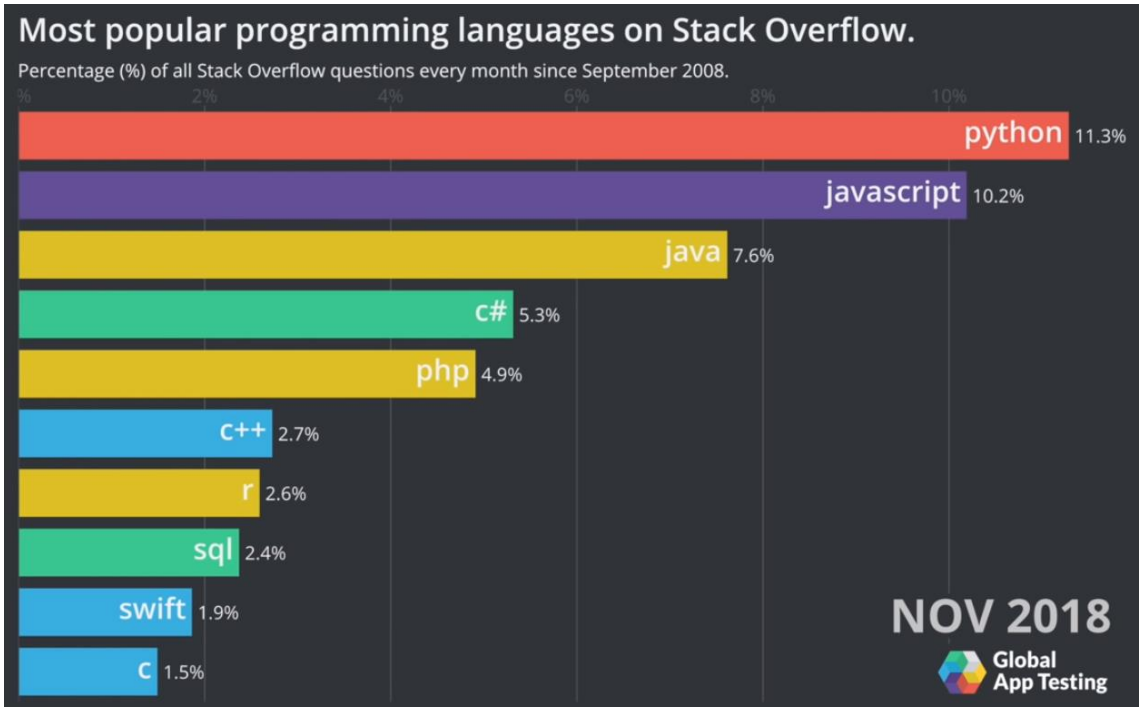


Figura 15. Diagrama de comparación de lenguajes de programación más populares en Stack Overflow. Fuente: Global App Testing.

3.3.5 Arduino IDE

Cuando se trata de proyectos de electrónica de carácter "hazlo tú mismo" (DIY, en inglés *Do It Yourself*) o de bajo coste, Arduino es ampliamente reconocida como una de las compañías líderes que ofrece soluciones de *hardware* y *software* libre.

Su propio entorno de desarrollo, disponible bajo descarga gratuita, servirá para desarrollar parte del protocolo de comunicaciones de este proyecto. En concreto, el código que albergará la Arduino Nano (una de sus soluciones hardware), que se encargará de que el microcontrolador sea capaz de interpretar los mensajes transmitidos por la interfaz gráfica.

3.3.6 Microsoft Office

Para el desarrollo de la documentación de este proyecto y diapositivas para su futura presentación ante tribunal, se ha hecho uso de la archiconocida suite de ofimática de Microsoft. También cabe destacar el uso de su solución de almacenamiento de archivos en la nube, la cual ha sido de utilidad para tener acceso en todo momento a la información necesario, sin importar el dispositivo que estuviera usándose en ese momento.

Si bien el software no se distribuye libre o gratuitamente, no tiene un coste excesivamente elevado y se encuentra presente en gran cantidad de instituciones gubernamentales y entornos empresariales, como es el caso de la UPV y su convenio educativo con Microsoft, que da acceso a la comunidad universitaria a los servicios de Office 365. Por no mencionar que, si se quisiera llegar un paso más allá reduciendo costes, podría usarse su alternativa de código abierto OpenOffice, desarrollado por Apache.

3.4 Hardware empleado en el desarrollo

A continuación se enumeran las herramientas hardware y componentes utilizados para el desarrollo del presente proyecto.

3.4.1 Ordenador personal

Dada la estandarización en el uso de sistemas operativos actualmente presente, el proyecto se ha desarrollado en diversos equipos en función de las necesidades presentes en el momento. Principalmente, se ha usado un ordenador personal con sistema operativo Windows, donde se han desarrollado la gran mayoría de tareas dedicadas a desarrollo de la interfaz y generación de la documentación requerida.

Por otro lado, al ser necesario en momentos puntuales del desarrollo del proyecto desplazarse al correspondiente laboratorio, se ha hecho uso de un ordenador personal portátil Macbook Pro para continuar allí con el trabajo.

En lo que a impresión de piezas se refiere, se han usado los equipos disponibles en el laboratorio para dicha función

3.4.2 Mean Well IRM-60-5

Este modulo de alimentación, del fabricante Mean Well y conformado por un convertidor AC-DC, será el encargado de proporcionar las señales de salida requeridas en la etapa de potencia. No dispone de unas dimensiones aptas para realizarse el diseño del esquemático en una versión gratuita de EAGLE (87 mm x 52 mm x 29,5 mm), pero sí dispone de las características suficientes para cumplir con los requisitos de la aplicación a un precio reducido. La potencia de salida es de 50 W (5V, 10 A) y, en contraparte, solo se dispone de un canal de salida, siendo necesario el uso de dos módulos.

Figura 16. Imagen de producto IRM-60-5. Fuente: Mouser.

3.4.3 Microchip MCP22515-E/SO

Este circuito integrado (CI) actuará como controlador en las comunicaciones en base a protocolo CAN. Es capaz de transmitir y recibir información dada en *estandard CAN* y *extended CAN*, además de comunicarse con microcontroladores vía SPI (*Serial Peripheral Interface*), lo cual resulta de gran utilidad para la comunicación con la Arduino Nano.

Figura 17. Imagen de producto MCP2515-E/SO. Fuente: Mouser.

3.4.4 muRata Electronics CSTLS16M0X53-A0

Resonador cerámico que se empleará para generar la señal de reloj para controlar dispositivos de lógica digital. En caso concreto de esta aplicación, proveerá las señales de entrada para los pines reservados a señales de oscilador del MCP22515-E/SO.

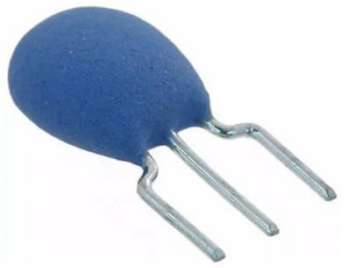


Figura 18. Imagen de producto CSTLS16M0X53-A0. Fuente: RS Components.

3.4.5 Microchip MCP2551T-I/SN

Este CI actúa de transceptor (emisor y receptor) de mensajes CAN. Adapta los niveles de señal procedentes del bus a niveles con los que el controlador CAN pueda trabajar, además de poseer circuitería de protección que protege al controlador CAN. Añadido a esto, convierte las señales del controlador CAN a señales que se envían al bus y que este puede interpretar.

Figura 19. Imagen de producto MCP2551T-I/SN. Fuente: Mouser.

3.4.6 Molex PicoBlade Connector 0532611071

Estos cabezales de 10 pines permitirán la conexión con un bus CAN. Dos pines están reservados al propio bus, mientras que 4 se dedican al voltaje de alimentación (Vcc) y los 4 restantes, a tierra (GND). Previamente se comprobó, mediante una prueba empírica, que los pines podían ser utilizados durante largas sesiones sin aumentos significativos en la temperatura, consistiendo la prueba en un sometimiento de los pines a una señal de 5 V y 8 A durante 24 horas.

Figura 20. Imagen de producto Molex PicoBlade Connector 10 pines. Fuente: Arrow.

3.4.7 Molex PicoBlade Connector 0530480210 (requiere desarrollo)

Estos cabezales de 4 pines permitirán la expansión del módulo OSIVE HI, permitiendo la conexión de más unidades de la etapa de potencia.

Figura 21. Imagen de producto Molex PicoBlade Connector 4 pines. Fuente: Arrow

3.4.8 Arduino Nano

Este microcontrolador, que forma parte de la unidad central del sistema, será más que suficiente para comunicarse con la interfaz gráfica y, a su vez, proporcionar las señales necesarias a otros componentes de OSIVE HI. Desde la interfaz, el usuario introducirá datos relevantes para la configuración de la señal de salida que debe ofrecer el dispositivo, y estos llegarán a la Arduino en un formato apto para su interpretación.

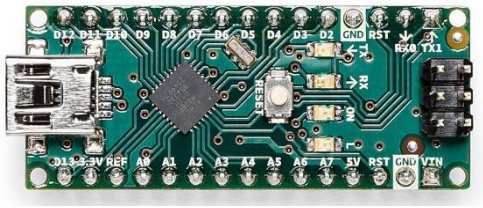


Figura 22. Imagen de producto microcontrolador Arduino Nano. Fuente: Arduino.

Posteriormente, se efectuará una conversión SPI – CAN para otorgar los datos, de manera que el controlador CAN (apartado 3.4.3) y el transceptor (apartado 3.4.5) sea capaces de gestionar la información.

3.4.9 Impresora 3D

Para producir la caja de la etapa de potencia OSIVE HI, se ha usado la impresora comercial Ultimaker 2 Extended disponible en el laboratorio.

Figura 23. Imagen de producto de la impresora 3D Ultimaker 2 Extended. Fuente: TR3SDLAND

3.4.10 Filamento de ácido poliláctico (PLA)

Para la impresión de la caja de OSIVE HI, se ha utilizado el material estándar en este tipo de aplicaciones. En concreto, un plástico biodegradable color negro de 2.85 mm del distribuidor RS Pro.

3.4.11 Resistencias y condensadores

Componentes pasivos que acondicionarán la señal a medida que la corriente circule por la PCB. La solución adoptada en este proyecto incluye resistencias de valores típicos, condensadores sin polaridad y un condensador electrolítico.

3.4.12 Soldador Weller WXR3

Gracias a esta estación de soldadura de estaño con potencia nominal de 600W disponible en el banco de trabajo del laboratorio, se soldarán los componentes a la PCB una vez se reciba el conjunto. Además de soldado, es capaz de desoldar componentes y posee características interesantes como la gestión de la potencia, autoregulándose según el uso.

Figura 24. Imagen de producto Weller WXR3. Fuente:

3.4.13 Fuente de alimentación PROMAX FAC-662B

Con una salida regulable de 30 V / 1 A y una salida auxiliar de 5 V, esta fuente de alimentación, también disponible en el banco de trabajo del laboratorio, es de utilidad para la alimentación de prototipos cuyo funcionamiento aún no ha sido testado, impidiendo alterar el funcionamiento de la red eléctrica.

Figura 25. PROMAX FAC-662B. Fuente: Media UPV.

4. ESPECIFICACIÓN DE REQUISITOS

En este apartado se definen como requisitos una serie de condiciones que el sistema ha de cumplir para que los objetivos del proyecto se lleven a cabo. Para esta aplicación, serán aquellos que sirvan para desarrollar una etapa de potencia de coste reducido, con protocolo CAN incorporado para comandar los punteros lumínicos, configuración de la señal de salida vía interfaz gráfica, y envío y recibo de datos entre esta y el microcontrolador Arduino Nano.

Pueden clasificarse estos requisitos entre funcionales, que definen las características relacionadas con el comportamiento principal, y no funcionales, que atienden a cómo el sistema realiza las distintas acciones.

4.1 Requisitos técnicos o funcionales

- La etapa de potencia debe ser capaz de proporcionar una señal de salida de 5 V y 20 A.
- La señal de salida de la etapa debe ser configurable por el usuario en términos de irradiancia, tiempo de encendido (duración del estímulo), tiempo de apagado (tiempo durante el que el estímulo cesa), número de pulsos, repeticiones y tiempo de reposo.
- La etapa de potencia debe presentar posibilidades de expansión, pudiendo conectarse varias unidades de esta entre sí.
- Debe asegurarse la accesibilidad total al software y al hardware para futuras correcciones, modificaciones y mantenimientos.

4.2 Requisitos comerciales o no funcionales

- Estructura modular del software, facilitándose su futura modificación y mejora.
- El diseño de la etapa debe albergar únicamente los componentes necesarios, garantizándose su sencillez.
- Sistema orientado a una fácil producción, garantizándose el uso de componentes comerciales y fácilmente accesibles.
- La etapa de potencia debe estar correctamente acondicionada y protegida.
- La solución de control de la etapa debe ser, a nivel de usuario: clara, intuitiva y sencilla de usar.
- Software ejecutable desde distintos sistemas operativos: Windows, macOS, GNU/Linux.

5. PLANIFICACIÓN

En este punto se procede a la definición de cada una de las fases de desarrollo del proyecto, con el objetivo de lograr una distribución del trabajo en el tiempo.

n	Fase de desarrollo	Tiempo estimado (h)	Tiempo invertido (h)
1	Planificación y replanificación (intermitente).	15	20
2	Investigación y formación en optoquimiogenética.	5	4
3	Análisis del proyecto y solución adoptada.	15	10
4	Búsqueda y adquisición de componentes para prototipado.	2	1
5	Formación en herramientas de diseño EDA.	4	2
6	Diseño de esquemáticos y PCB.	1	1
7	Soldado de componentes. Prototipado de la placa.	3	4
8	Diseño de la caja de la etapa de potencia.	2	2
9	Impresión de la pieza y ensamblado de la etapa.	5	4
10	Formación en Python (intermitente).	30	27
11	Elaboración de la interfaz gráfica de usuario.	50	60
12	Pruebas de funcionamiento de la interfaz.	10	10
13	Elaboración del protocolo de comunicaciones interfaz gráfica - Arduino Nano.	20	15
14	Validación de las tramas de comunicación.	10	6
15	Redacción de la documentación (intermitente).	70	60
16	Preparación de la defensa oral.	60	55
17	TOTAL	302	281

Tabla 1. Fases de desarrollo del proyecto. Fuente: Propia.

6. ANÁLISIS

Con el objetivo de realizar el análisis del proyecto, se determinarán los casos de uso principales, ante los que el sistema deberá funcionar de una forma determinada. Tras ello, un estudio de soluciones alternativas es realizado para escoger la solución adoptada y mostrar el diagrama de flujo final del sistema.

6.1 Casos de uso

Siguiendo la definición de *caso de uso* correspondiente a el conjunto de actividades que se pueden o deben realizar para el correcto funcionamiento del sistema, se considerarán los siguientes.

El principal cometido del dispositivo es otorgar una señal eléctrica determinada, interpretándose la información introducida por un usuario en una interfaz gráfica. Dicho caso de uso se encuentra representado en la *Figura 26*, expuesta a continuación.

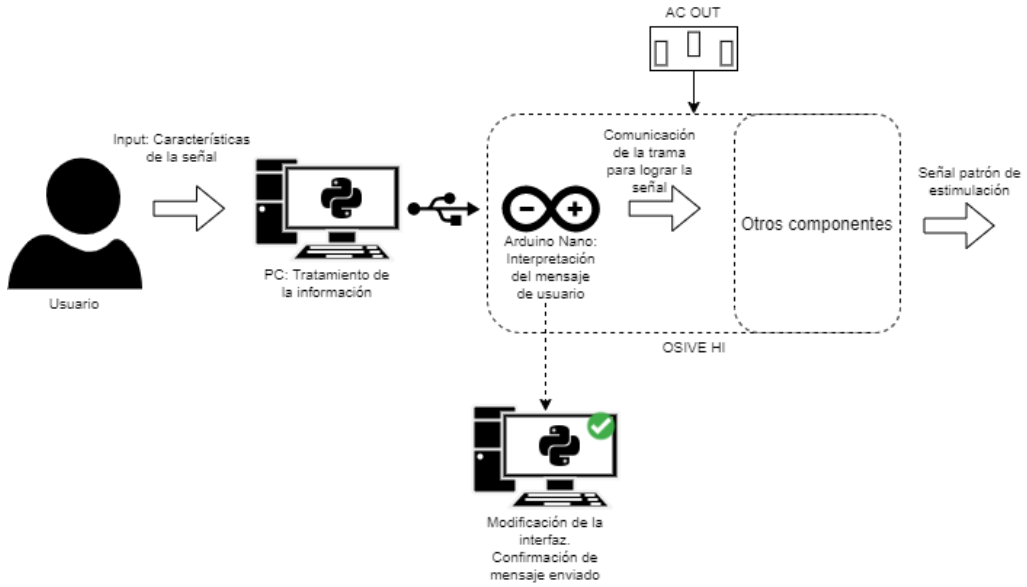


Figura 26.Caso de uso principal del sistema. Fuente: Propia.

Las características de la señal estarán previamente limitadas por la interfaz, impidiendo al usuario introducir los siguientes valores:

- Valores de irradiancia comprendidos fuera del intervalo [0, 10] mW/mm2 para cualquiera de los dos punteros.
- Valores menores a 0.01 ms para los tiempos de encendido (t_On).
- Valores menores a 0.01 ms para los tiempos de apagado (t_Off).
- Valores decimales para los parámetros número de pulsos, tiempo de reposo (horas, minutos y segundos) y número de repeticiones.

En caso de que el usuario intente evadir estas limitaciones, la propia interfaz inhabilitará el envío de la trama de comunicación hacia la Arduino Nano.

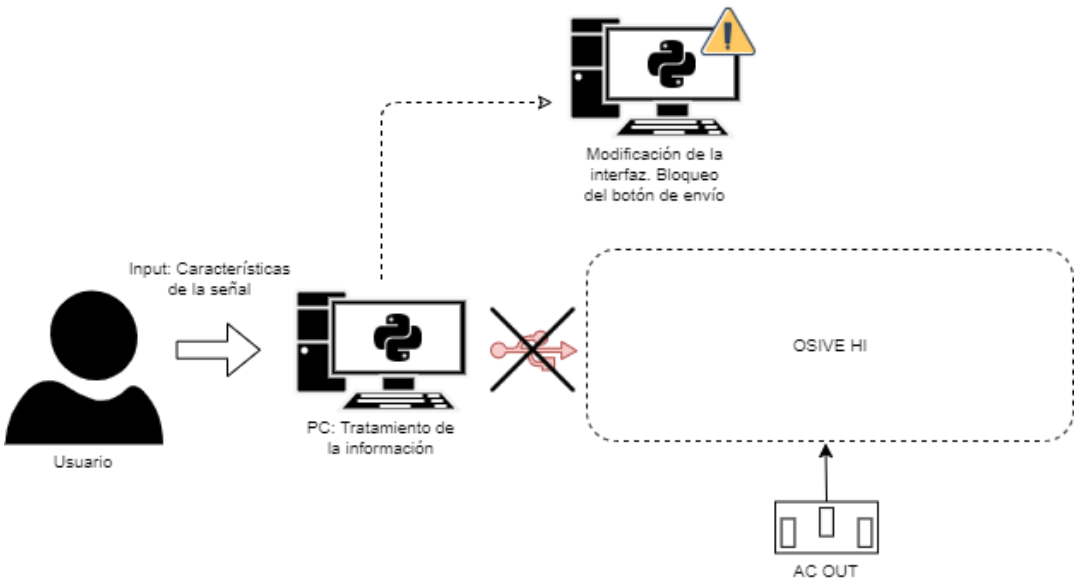


Figura 27. Caso de uso: Bloqueo de la interfaz por limitación. Fuente: Propia

Finalmente, cuando un usuario decida terminar el experimento, podrá interactuar con un widget en pantalla para ejecutar esta tarea.

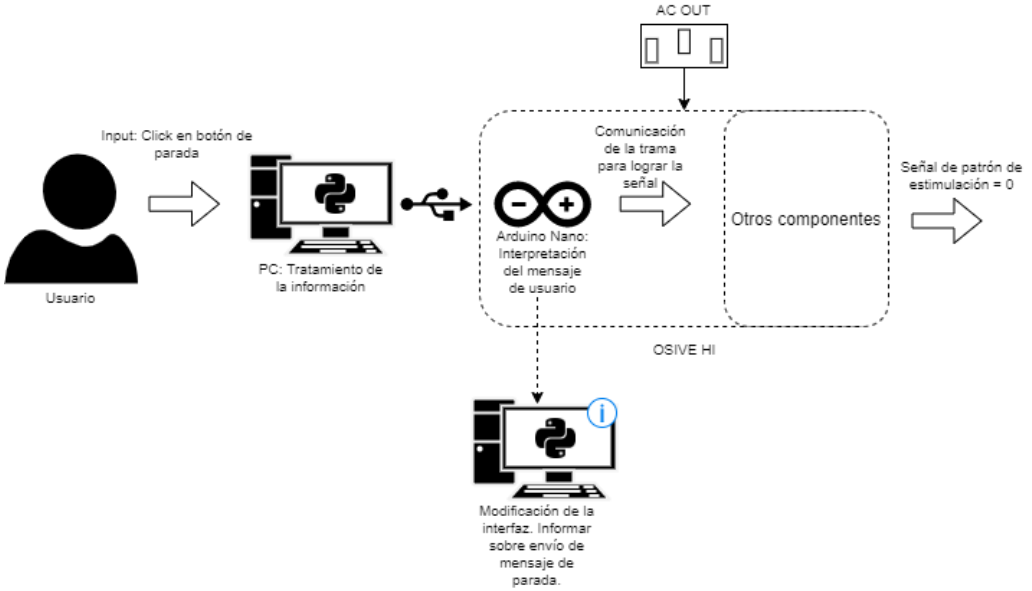


Figura 28. Caso de uso: Finalizado de la comunicación. Fuente: Propia

6.2 Estudio de alternativas

Se presentarán en este punto las diferentes opciones y alternativas que se plantearon para llevar a cabo la resolución de este proyecto.

6.2.1 Fuente de alimentación lineal

En el caso de este tipo de fuente de alimentación, el diseño es considerablemente más sencillo que el de las fuentes conmutadas. Presentan una serie de ventajas en términos de velocidad y compatibilidad electromagnética, aunque desgraciadamente el tamaño y el peso son alarmantemente mayores. Aunque podrían resultar interesantes para alguna aplicación específica que requiere de niveles de ruido ultrabajos, este proyecto dista de aquel campo, lo cual la convierte en una opción no demasiado atractiva, especialmente teniendo en cuenta su principal deventaja: el rendimiento.

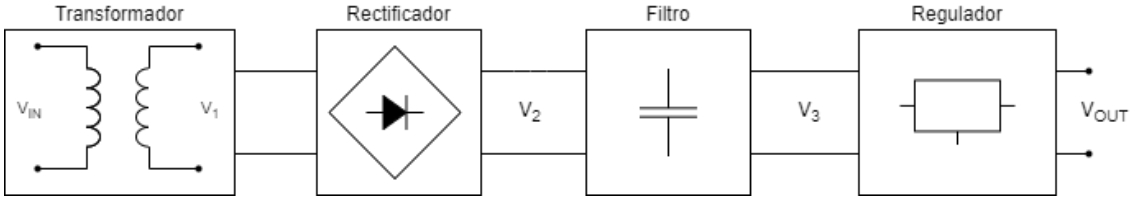


Figura 29. Ejemplo de diagrama de bloques funcionales para una fuente lineal. Fuente: Propia

La diferencia fundamental de una fuente lineal es que la regulación del voltaje se consigue con un componente disipativo, adecuando el punto de funcionamiento de un transistor en su zona lineal mediante un circuito de realimentación. Dicho de otro modo, el acondicionamiento de la señal se consigue con la disipación de calor.

El equipo resultado de este proyecto se ubicará en laboratorios para su uso en ensayos, pero es posible que se requiera de su portabilidad. Sumado a esto hay que considerar la duración de los ensayos, que puede extenderse a varias horas en el tiempo. Al no ser una solución precisamente amigable en términos de peso, tamaño y eficiencia, esta queda descartada.

6.2.2 Fuente de alimentación conmutada

Pese al aumento de complejidad del diseño (véase *Figura 6*) en comparación con la solución presentada en el punto anterior, las decisiones tomadas en este convierte a las fuentes conmutadas en candidatas muy aptas para el desarrollo de este proyecto.

A fecha de la redacción de este documento, son la elección mayoritaria para aplicaciones donde se necesita estabilidad, tamaño y pesos reducidos, y buenos niveles de eficiencia, donde el caso paradigmático se encuentra en los cargadores móviles. Los valores típicos indican entre un 75 y 85% para diseños estándar, mientras que algunos pueden llegar a valores como el 95%, por tanto se estará aprovechando gran parte de la energía en la conversión.

No obstante, para lograr esta serie de indicadores es necesario renunciar a otros. Los inconvenientes que pueden presentar este tipo de fuentes son tales como una mayor generación de ruido eléctrico susceptible a generar interferencias, ya que los componentes trabajan a alta frecuencia para garantizar estos factores de conversión. Por tanto, filtros EMI y blindajes de radiofrecuencia suelen ser necesarios en este tipo de diseños, impactando en la complejidad del diseño. También existen otras interferencias, como el ruido eléctrico causado por la conmutación o incluso ruido acústico, que con el deterioro de los componentes puede causar, con el tiempo, efectos audibles.

Pese a ello, el precio para una solución comercial de este tipo puede ser muy contenido; lo que, sumado a los factores discutidos anteriormente, la convierten en una de las soluciones adoptadas.

6.2.3 Configuración de la señal

Puesto que una de las especificaciones de la organización contratante consistía en el control por GUI (de sus siglas en inglés, *Graphical User Interface*) del dispositivo, no se han analizado soluciones alternativas en cuanto a cómo el usuario logrará configurar determinados valores de la señal estímulo.Por ello, se ha considerado directamente esta solución software.

6.2.4 Protocolo de comunicaciones

Existen diferentes protocolos de comunicaciones que podrían implementarse en la solución de este proyecto. Protocolos como LIN (*Local Interconnect Network*), FlexRay, CAN o MOST (*Media Oriented Systems Transport*) son principales competidores entre sí, en áreas de aplicación similares. Cualquiera es una solución lo suficientemente robusta y rápida como para usarse, no obstante, predecesores de esta OSIVE HI venían implementando un bus CAN para comunicaciones, que decidió mantenerse en pos de la integridad y trazabilidad del trabajo.

Podrían haberse estudiado alternativas como Ethernet, o incluso comunicaciones inalámbricas, pero decidieron descartarse por un aumento en la complejidad global de la solución.

6.3 Solución adoptada

Se pretende en este punto hacer una relación entre las soluciones escogidas y el cómo o de qué manera los objetivos se satisfacen en consecuencia.

- En cuanto a cómo recoger energía de la red eléctrica para llevarla a la fuente de alimentación, se seleccionó un cable de alimentación con conector IEC (*International Electrotechnical Commission*) 60320 C14, el cual es un estándar y tiene un uso ampliamente extendido (por ejemplos, en fuentes de alimentación para ordenadores personales).
- Para la etapa de alimentación, se seleccionó un modelo comercial de fuente de alimentación conmutada que tuviera un costo reducido y ofreciera posibilidad de soldado en PCB. Con esto, se seguirían manteniendo los objetivos de peso y dimesiones reducidas, y eficiencia notable. Además de resolver de manera positiva el diseño de la etapa en su conjunto, haciéndose uso de una solución comercial en la que el trabajo de diseño de los propios convertidores ya está previamente realizado.
- Para brindar la posibilidad al usuario de configurar la señal de la etapa de potencia, se decidió por desarrollar una solución software específica para este proyecto, ofreciendo accesibilidad total al código fuente para futuras modificaciones y mejoras. En añadido, las herramientas de código abierto usadas para este cometido impactan positivamente en el coste total del proyecto.
- Para las comunicaciones entre la interfaz gráfica de usuario y la etapa de potencia, se desarrolló un protocolo siguiendo las directrices de CAN, comunicándose una trama de valores que el microcontrolador de Arduino recibiría, confirmaría e interpretaría. Al tratarse de la solución más económica de la compañía, de nuevo se impacta positivamente en el coste.

- Para las comunicaciones entre la etapa de potencia OSIVE HI y etapas posteriores del sistema OSIVE, se incorporaron los componentes electrónicos necesarios al diseño, para dotarlo de compatibilidad con el protocolo CAN. De este modo, la integridad del proyecto del *Centro de Investigación Príncipe Felipe* no se vería afectada.

6.4 Diagramas de flujo

Se adjuntan en el Anexo 1 de este documento los principales flujogramas que el sistema ha de seguir para garantizar el correcto funcionamiento de la etapa de potencia OSIVE HI. También se incluye información detallada sobre aspectos como el orden de generación de variables, lógica de los métodos usados, asociación de métodos a botones de la interfaz, etcétera.

7. DISEÑO

Durante el presente punto se expone el desarrollo del dispositivo objeto de este proyecto.

7.1 Protocolo de comunicaciones

En el desarrollo de OSIVE HI se han tenido en cuenta varios protocolos de comunicación. Una diferenciación rápida se ofrece a continuación:

- Comunicaciones UART (por sus siglas del término inglés, *Universal Asynchronous Receiver-Transmitter*): Son las comunicaciones establecidas entre el ordenador personal y la Arduino Nano. En lo referente al dispositivo desarrollado, el controlador UART es un periférico del microcontrolador que controla los puertos serie, y la tarjeta de evaluación Arduino Nano cuenta con un convertidor UART-USB para la comunicación con el ordenador.
- Comunicaciones SPI (por sus siglas del término inglés, *Serial Peripheral Interface*): Se dan desde el periférico integrado en el microcontrolador del Arduino Nano hasta el controlador CAN, circuito integrado de la unidad central de OSIVE HI comandado por el microcontrolador por SPI.
- Comunicaciones CAN: Se dan entre el bus CAN de OSIVE HI, siendo su nodo máster el correspondiente a la unidad central, y los nodos esclavos aquellos asociados a los punteros lumínicos. Todos los nodos se conectan al bus a través de transceptores comandados por controladores, a su vez estos controlados por SPI desde el microcontrolador correspondiente (véase el apartado 3.4.3 para controlador CAN y el apartado 3.4.5 para el transceptor CAN).

Diferenciadas las comunicaciones, se adjuntan los siguientes diagramas para una mejor distinción visual entre etapas del dispositivo:

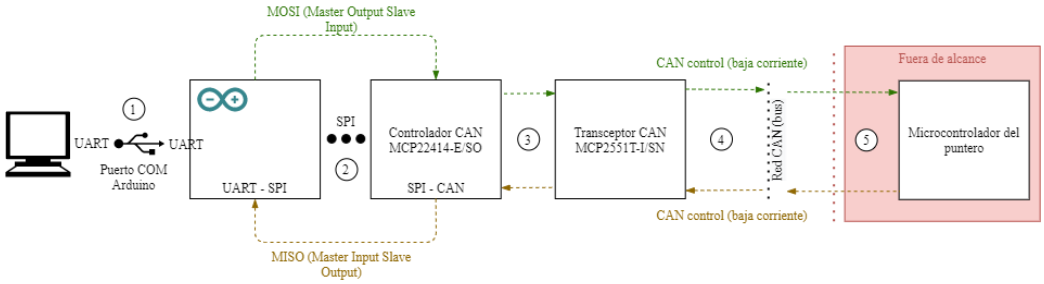


Figura 30. Diagrama general de comunicaciones en OSIVE HI. Fuente: Propia

Se pueden diferenciar cinco partes esenciales, señaladas en la Figura 30:

1. Comunicaciones interfaz gráfica-Arduino Nano: Tramas enviadas por USB que el microcontrolador interpretará para su posterior conversión a SPI.

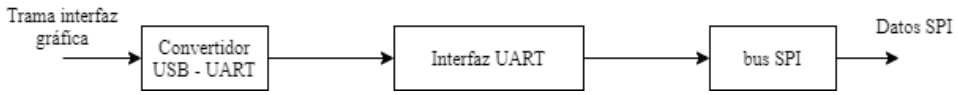


Figura 31. Detalle de la conversión UART – SPI dentro de Arduino. Fuente: Propia

2. Nano – Controlador CAN: Además de que el microcontrolador de Arduino convierte los datos, gracias a su interfaz SPI integrada, también se producen las conexiones con el controlador CAN MCP22414-E/SO. Por un lado, por la pista

de MOSI circulan señales de la Arduino al controlador, mientras que por la pista de MISO circulan señales en sentido contrario.

3. Controlador CAN – Transceptor CAN: Para la información que viaja hacia el bus CAN, el controlador envía las señales al transceptor en formato serie full-duplex, que este convertirá en diferenciales half-duplex.
4. Transceptor CAN – bus CAN: Si la señal viaja hacia el bus, el transceptor ajusta la señal CAN a una señal de control diferencial. Hecho esto, las envía a la red distribuida. Si por el contrario la señal viene del bus, el transceptor la recibe para su acondicionamiento y posterior envío al controlador.
5. Bus CAN – Punteros lumínicos: Antes de llegar la señal a los punteros lumínicos, la señal CAN se somete a dos procesos:
    a. La señal que viene del bus se convierte a una señal CAN de comunicaciones.
    b. Esta señal resultante se convierte a una señal SPI para la correcta interpretación del microcontrolador disponible en los punteros lumínicos.

Estas últimas etapas 5.a y 5.b quedan fuera del alcance de este trabajo final de grado, así como el desarrollo del firmware necesario para la Arduino Nano, que necesitaría para la interpretación y correcto envío de las señales SPI al controlador CAN.

En la siguiente figura del plano de diseño de la PCB se destacan los componentes dedicados a este cometido:

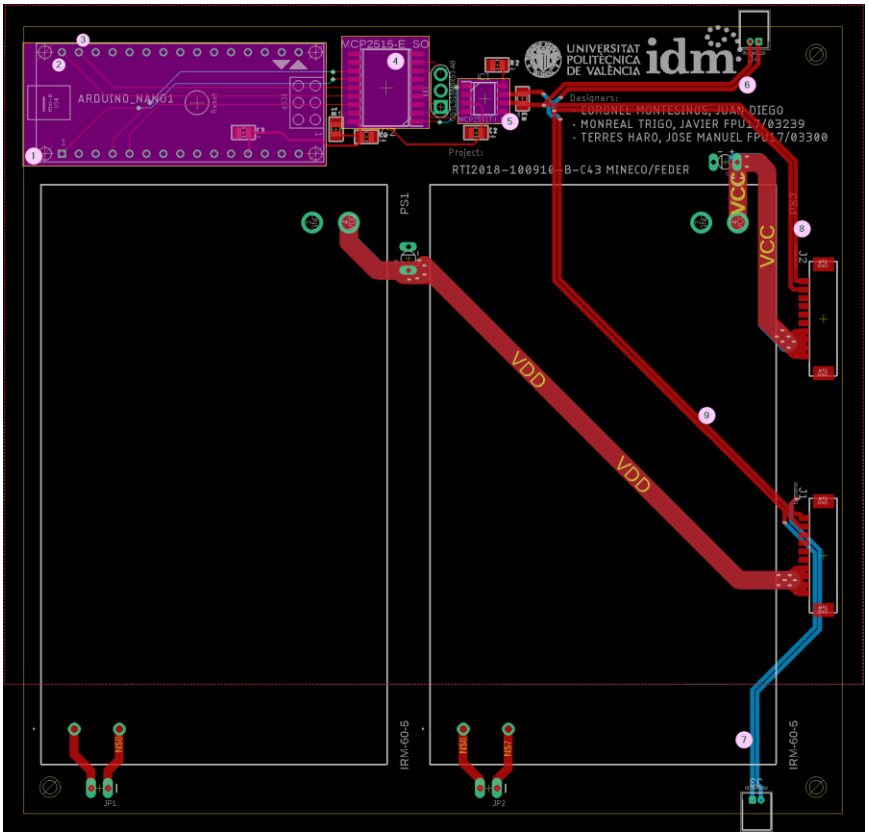


Figura 32. Exportado del diseño de la PCB, con resalte de los componentes de comunicación en morado. Fuente: Propia

1. Arduino Nano: Microcontrolador encargado de recibir e interpretar las comunicaciones llegadas de la interfaz gráfica.
2. Pista MISO: Como se ha mencionado anteriormente (véase Figura 32), por esta pista se envían las señales desde el controlador CAN hasta la Arduino.
3. Pista MOSI: Como se ha mencionado anteriormente (véase Figura 32), por esta pista se envían las señales desde la Arduino hasta el controlador CAN.
4. Controlador CAN: Recibe señales SPI, las interpreta, convierte y envía al transceptor CAN en el correspondiente formato serie. También recibe señales del transceptor, provenientes del bus CAN.
5. Transceptor CAN: Envía señales al bus CAN previamente acondicionadas para ello, y a su vez recibe señales del propio bus. Antes de enviarlas de vuelta al controlador, las acondiciona para este. A su lado puede observarse la resistencia de terminación de 120 Ω para evitar reflexiones.
6. Bifurcación de las pistas CAN diferencial para expansión modular.
7. Segunda bifurcación de las pistas CAN diferencial para expansión modular.
8. Bifurcación de las pistas CAN diferencial para comunicación con el bus a través de los conectores de 10 pines.
9. Segunda bifurcación de las pistas CAN diferencial para el mismo cometido narrado anteriormente.

7.1.1 Script Interfaz gráfica – Arduino Nano

Como se mencionó al comienzo del Apartado 7.1, las comunicaciones UART tienden el puente entre la interfaz gráfica albergada en el ordenador personal y el microncontrolador Arduino. Pueden consultarse los flujogramas de la lógica implementada en los módulos diseñados para este cometido en el Anexo 2 de este mismo documento. No obstante, en este apartado se ofrece un resumen de diseño de la solución:

- Función de callback: Son un tipo de método asociado a un widget, que se ejecutarán al interactuar con el susodicho. Con estos métodos se han implementado funciones para escribir tramas en el puerto COM de la Arduino, abrir y cerrar este puerto, o parar el sistema.

Las funciones dedicadas a comunicaciones a nivel de interfaz gráfica son las siguientes:

7.1.1.a connectCOM(selPort)

Este método se usa para abrir el puerto COM seleccionado (el usuario debe previamente seleccionar el puerto de una lista dada):

```
def connectCom(selPort):
    strComVar = selPort.split(' ')[0] #Split para quedarnos unicamente con la
parte del nombre del puerto COM que nos interesa
    global arduino #llamada a la variable global arduino dentro de este método
para modificarla
    arduino = serial.Serial(strComVar, 9600, timeout=30) #Almacenado del
objeto Serial en la variable arduino. EL objeto es el puerto COM seleccionado
    btnConnectCOM.configure(state=tk.DISABLED) #Se desactiva el botón de
CONNECT mientras que el puerto COM esté abierto
    return arduino #El método devuelve el valor de arduino para que pueda
trabajarse con él en otros scopes
```

Este método se apoya en un método auxiliar, `connectComOnBtn()`, para asociar el método a un botón. Es una limitación de la propia librería tkinter no poder asociar a botones métodos que dependen de argumentos.

7.1.1.b disconnectCOM(selPort)

Método usado para cerrar el puerto COM con el que se está estableciendo comunicaciones en ese momento:

```
def disconnectCom(selPort):
    selPort.close()
    btnConnectCOM.configure(state=tk.NORMAL)
```

Este método se apoya en un método auxiliar, `disconnectComOnBtn()`, para asociar el método a un botón, por limitaciones de la propia librería tkinter.

7.1.1.c sendValues()

Cuando se pulsa el correspondiente botón, este método se ejecuta, desencadenando las siguientes acciones:

```
def sendValues():
    #reescribir variables globales por lo que haya en ese momento en los
cuadros de texto
    global amp1
    strAmp1 = amp1.get()
    global amp2
    strAmp2 = amp2.get()
    global tOn1
    strTOn1 = tOn1.get()
    global tOn2
    strTOn2 = tOn2.get()
    global tOff1
    strTOff1 = tOff1.get()
    global tOff2
    strTOff2 = tOff2.get()
    global nPulse
    strNPulse = nPulse.get()
    global hRest
    strHRest = hRest.get()
    global mRest
    strMRest = mRest.get()
    global sRest
    strSRest = sRest.get()
    global nRep
    strNRep = nRep.get()
    global headerCom
    headerCom = 'j'
    headerComAsciiValues = [ord(c) for c in headerCom] #Esto convierte cada
caracter en su valor ASCII (ints)
    headerAsciiCheckSum = float(headerComAsciiValues[0]) #Esto es un cast de
int a float. Para la j vale 106.0

    '''Se escogió que el valor de checksum fuera la suma en float de todos los
valores enviados.
    Posteriormente se obtiene el módulo de 95 de ese valor y se le suma 32,
con el objetivo de que el checkSum quede
    dentro del rango de valores perteneciente a un caracter ASCII imprimible.
    Esto nos ayuda a verificar por consola comunicaciones cuando testeamos el
software.'''
    checkSumFloat =
(headerAsciiCheckSum+float(strAmp1)+float(strAmp2)+float(strTOn1)+float(strTOn
2)+float(strTOff1)+float(strTOff2)+float(strNPulse)+float(strHRest)+float(strM
Rest)+float(strSRest)+float(strNRep))
    checkSum = (int(checkSumFloat)%95)+32
    checkSumAscii = chr(checkSum)


    sentMessage =
(headerCom+','+strAmp1+','+strAmp2+','+strTOn1+','+strTOn2+','+strTOff1+','+st
rTOff2+','+strNPulse+','+strHRest+','+strMRest+','+strSRest+','+strNRep+','+ch
eckSumAscii+'\r\n')
    arduino.write(sentMessage.encode()) #Envío del mensaje. Escritura en
puerto COM
```

```
    time.sleep(1)
    msgSent() #Informar de que la comunicación se realizó correctamente


    time.sleep(0.4) #Esperar 400 milisegundos
    if arduino.inWaiting() > 0:
        print(arduino.read(arduino.inWaiting())) #Devolver por consola de la
interfaz gráfica el valor que ha sido enviado a la arduino
```

7.1.1.d sendStopMessage()

Cuando se pulsa el correspondiente botón, este método se ejecuta, desencadenando las siguientes acciones;

```
def sendStopMessage():
    global amp1
    strAmp1 = '0'
    global amp2
    strAmp2 = '0'
    global tOn1
    strTOn1 = '0'
    global tOn2
    strTOn2 = '0'
    global tOff1
    strTOff1 = '0'
    global tOff2
    strTOff2 = '0'
    global nPulse
    strNPulse = '0'
    global hRest
    strHRest = '0'
    global mRest
    strMRest = '0'
    global sRest
    strSRest = '0'
    global nRep
    strNRep = '0'
    global headerCom
    headerCom = 's'
    headerComAsciiValues = [ord(c) for c in headerCom]
    headerAsciiCheckSum = float(headerComAsciiValues[0])  # Esto es un
cast de int a float. Para la s vale 115.0

checkSumFloat=(headerAsciiCheckSum+float(strAmp1)+float(strAmp2)+float
(strTOn1)+float(strTOn2)+float(strTOff1)+float(strTOff2)+float(strNPul
se)+float(strHRest)+float(strMRest)+float(strSRest)+float(strNRep))
    checkSum = (int(checkSumFloat) % 95) + 32
    checkSumAscii = chr(checkSum)


sentMessage=(headerCom+','+strAmp1+','+strAmp2+','+strTOn1+','+strTOn2
+','+strTOff1+','+strTOff2+','+strNPulse+','+strHRest+','+strMRest+','
+strSRest+','+strNRep+','+checkSumAscii+'\r\n')
    arduino.write(sentMessage.encode())
    stopMsgSent() #Informar de que la comunicación se realizó
correctamente

    time.sleep(0.4)
    if arduino.inWaiting() > 0:
        print(arduino.read(arduino.inWaiting()))
```

Puede observarse que es la misma lógica que para el método `sendValues()`, solo que aquí se fuerza el envío de una comunicación errónea para que el microcontrolador efectúe una parada.

7.1.2 Script Arduino Nano

Si bien desde la interfaz gráfica se formulaba una trama en formato de String y se escribía en el puerto COM conectado en ese momento, la Arduino Nano sigue el mismo protocolo como "*listener*". En términos generales, el script .ino se ejecuta de la siguiente forma:

Se definen las variables para almacenar los datos. Al ser C el lenguaje en que se basa Arduino, y ser este un lenguaje fuertemente tipado, se deberá definir cada tipo de variable en función del dato a almacenar.

- Función de *setup*: Configuración del puerto COM con el número de baudios escogido para trabajar.
- Función *loop()*: Función de bucle infinito característica de Arduino. En ella se comprobará si en el puerto hay datos para leer. De ser así, se almacenarán los datos en un *array* de datos tipo *char* y se irá subdividiendo el array para almacenar los distintos variables, en función de la detección de comas en la cadena. Tras esto, se calculará el valor de checksum con la misma lógica seguida en el lado de la interfaz, con el fin de comparar este valor con el leído en el puerto. Si son similares, la comunicación se da por satisfactoria. Si no lo son, se tomará como una comunicación fallida.

Puede consultarse tanto los diagramas de flujo del script desarrollado (véase Anexo 1), como el código fuente en su totalidad (véase Anexo 3).

7.2 Unidad central del sistema

OSIVE HI no es solamente un aparato capaz de proporcionar alimentación para los punteros lumínicos que generan los estímulos lumínicos necesarios en los experimentos comentados anteriormente (apartado 1.2). También es un dispositivo capaz de comandar los mencionados punteros LED.

Es por esto que el proyecto tiene diversos alcances, y se ha considerado oportuno dividir el proceso de diseño en función de una serie de actividades principales:

- La etapa de potencia, que proporcionará la señal de alimentación necesaria para los punteros.
- La unidad de control, que englobará el diseño de las soluciones otorgadas en materia de comunicaciones. Tanto desde la interfaz gráfica al microcontrolador, como de este a los componentes que trabajan bajo CAN (incluyendo los propios punteros).
- El diseño de la placa de circuito impreso, que alberga todo el hardware necesario.
- El diseño del encapsulado para OSIVE HI.

7.2.1 Etapa de potencia

A expensas del propio título de este proyecto, será incluido en la definición de etapa de potencia para este proyecto todo elemento necesario para generar las señales de alimentación que los punteros lumínicos necesitan.

Si bien podría haberse escogido la opción de hacer un diseño propio de fuente de alimentación escogiendo entre distintas topologías de diseño y demás, se escogió para este cometido una solución comercial ya encapsulada y con posibilidad de soldado a la PCB. Para localizar el componente exacto, se consideró recurrir al distribuidor RS Components, pues existen convenios entre la compañía y el centro para el que se realiza el presente proyecto.

Echando un primer vistazo a las fuentes de alimentación capaces de generar 5 V (DC) y unos valores de corriente continua comprendidos entre mínimo 10 A y máximo 24 A, el intervalo de precios es muy poco acotado, hallándose soluciones desde los 10,28 € hasta incluso 1.189,20 € (precios consultados en la web de RS Components a día 5 de septiembre de 2021).

Ya que la solución completa al proyecto consta de una placa de circuito impresa de diseño propio, filtrando por tipo de montaje (montaje en PCB) se disminuye considerablemente el rango de opciones:

Producto	Fabricante	Tensión de salida (V)	Corriente de salida (A)	Potencia nominal (W)	Número de salidas	PVP unidad (€)
RPS-65-5	Mean Well	5	10	50	1	14,81
IRM-60-5	Mean Well	5	10	50	1	15,90
ECE60US05	XP Power	5	10	50	1	70,34
SNTUNS50F05	Cosel	5	10	50	1	183,71
SNTUNS100F05	Cosel	5	20	100	1	231,58

Tabla 2. Fuentes de alimentación SMPS que cumplen con los requisitos de diseño. Fuente: RS Components.

Cada puntero lumínico LED requiere como señal de alimentación 5 V y 8 A para su correcto funcionamiento, Tomando un sobredimensionamiento del 20%, las cinco opciones cumplirían con las especificaciones.

Finalmente, se optó por la solución IRM-60-D (véase apartado 3.4.2) por ser la más económica, utilizándose 2 unidades, una por puntero. Gracias a sus señales otorgadas a la salida, se otorgará potencia suficiente a los punteros lumínicos. Las unidades de control y comunicaciones se encuentran alimentadas por el ordenador y aisladas por tanto de la parte de potencia. La expansión modular exclusivamente comparte el bus de comunicaciones y no la alimentación, puesto que sería contradictorio al fin de conseguir que el dispositivo fuera modular (con 20 A no es posible alimentar más de dos punteros, los punteros adicionales necesitan de su propia alimentación adicional).

En la imagen adjunta puede apreciarse la zona delimitada, dedicada a estos componentes:

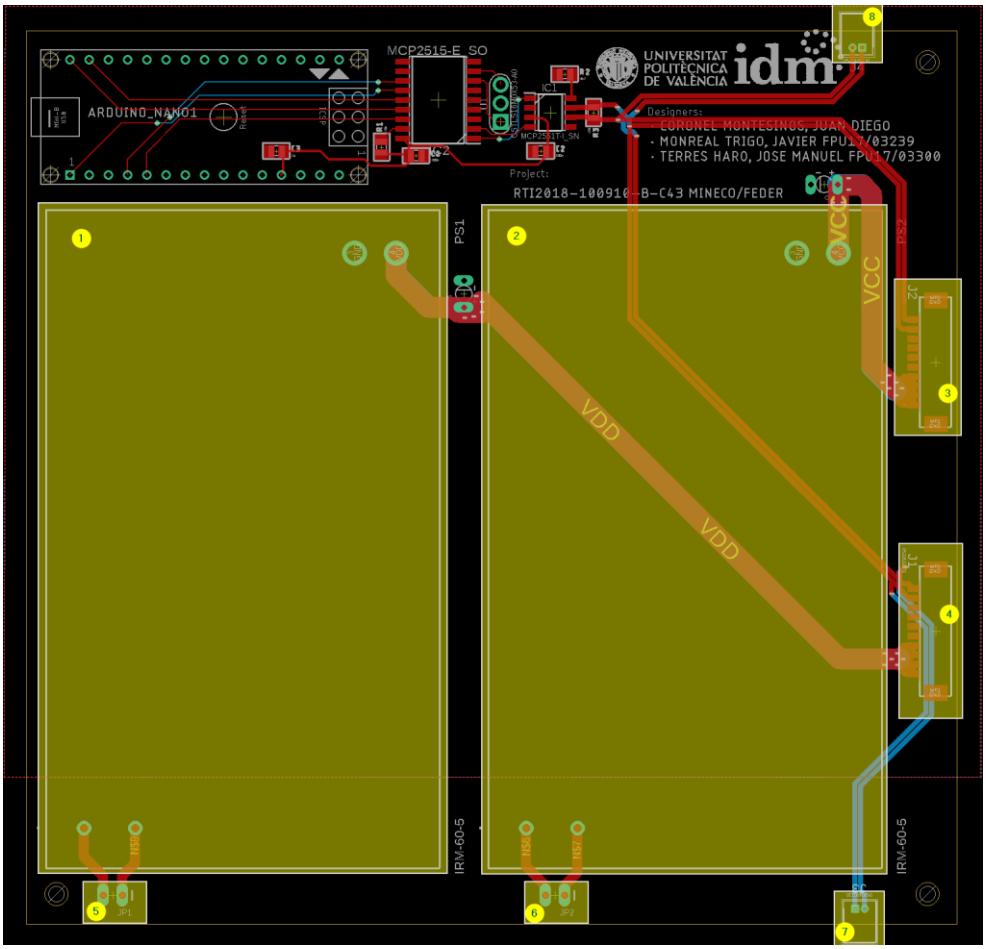


Figura 33. Exportado del diseño de la PCB, con resalte de las funciones de alimentación y conectores CAN, en amarillo. Fuente: Propia.

1. Espacio dedicado a albergar una unidad Mean Well IRM-60-5.
2. Espacio dedicado a albergar la segunda unidad Mean Well IRM-60-5.
3. Uno de los conectores 0532611071, alimentado por la IRM-60-5 y que servirá para las comunicaciones CAN con los punteros.
4. Segundo conector 0532611071, también alimentado por la IRM-60-5, con el mismo cometido.
5. Conexión del conector IEC, que llevará la señal de la red eléctrica a la entrada de la IRM-60-5.
6. Conexión del conector IEC, que llevará la señal de la red eléctrica en paralelo, con el mismo conector, a la entrada de la IRM-60-5.
7. Módulo de expansión para conectar otra unidad de OSIVE HI en paralelo, permitiendo la comanda de más punteros desde la misma interfaz gráfica y con el mismo microcontrolador Arduino Nano.
8. Segundo módulo de expansión lateral para conectar otra unidad de OSIVE HI.

7.2.2 Unidad de control

Se define como la unidad de control de OSIVE HI todo *hardware* necesario para establecer las comunicaciones que dotarán al sistema de capacidad para comunicar unos elementos y otros.

Si bien existen numerosos microcontroladores en el mercado con periféricos similares, se ha escogido para el desarrollo de la unidad de control una solución basada en el microcontrolador Arduino Nano, que comanda por SPI un controlador CAN.

Arduino es una familia de microcontroladores popularizada por la comunidad educativa y de entusiastas de la electrónica, que cuenta con un gran respaldo de documentación. Si bien requiere utilizar tarjetas de evaluación que tienen unas dimensiones superiores a utilizar sólo su microcontrolador, Arduino Nano es la más reducida de la familia, y el uso de esta plataforma tiene un impacto significativo en el desarrollo del prototipo, acelerándolo por las facilidades que presta.

Esquemas detallados sobre cómo los distintos elementos están conectados entre sí pueden verse en la Figura 30 del apartado 7.1 de esta memoria, pues allí se detalló acerca de estas cuestiones en relación a los protocolos de comunicaciones implementados en este proyecto.

7.2.3 Diseño de la PCB

La placa de circuito impreso de este dispositivo se ha diseñado con el software EAGLE. Tanto el esquemático como el plano de la PCB pueden consultarse en el apartado Planos de este documento.

Posterior al diseño, el archivo correspondiente (extensión .brd en EAGLE) se ha enviado a un fabricante para su impresión y revelado. Finalmente la placa ostenta unas dimensiones de 122,96 mm x 118,08 mm.

7.2.4 Diseño del encapsulado

El encapsulado para el la PCB se ha diseñado con el software SolidWorks. El plano puede consultarse en el apartado Planos de este documento.

7.3 Interfaz gráfica

Para el diseño de la interfaz gráfica, se ha usado el lenguaje de programación Python por su accesibilidad, poca dificultad y herramientas facilitadas. Python cuenta con librerías dedicadas a la comunicación con puertos serie (*third party*), que deben ser instaladas en el sistema previamente antes de ejecutar el código fuente. No obstante, la librería usada para la creación de la interfaz, tkinter, viene integrada al instalar el lenguaje. Lo que significa que cualquier máquina que tenga Python instalada, será capaz de ejecutar el código (siempre y cuando tenga las librerías de terceros usadas para este proyecto instaladas).

En el diseño de esta interfaz se han usado tres librerías principales:

- Tkinter: Opción integrada en Python para el desarrollo de interfaces gráficas.
- pySerial: Dedicada a la comunicación serie con los puertos COM habilitados en el sistema.
- Time: Librería integrada en Python. Se utilizó para dejar el sistema en *stand by* en ciertos momentos.

A continuación se describirá la disposición de elementos en la interfaz y qué metodología se siguió para cumplir con el mayor éxito los objetivos propuestos.

7.3.1 Disposición de elementos

En tkinter, un set de objetos predeterminados pueden ser rápidamente creados, configurados, estilizados y empaquetados. La definición de empaquetar un *widget* en tkinter atiende a asociar el susodicho a un *frame* existente. Un frame puede definirse como un lienzo en blanco. Un marco que al principio está vacío, y en el cual pueden ir albergándose objetos.

Son diferentes métodos los que tkinter propone para estructurar una interfaz. Pueden usarse coordenadas cartesianas para especificar posiciones absolutas donde los widgets se colocarán. También pueden usarse métodos que relacionan el widget a un frame, siendo estos `.pack()` y `.grid()`. Ambos se han utilizado en este proyecto.

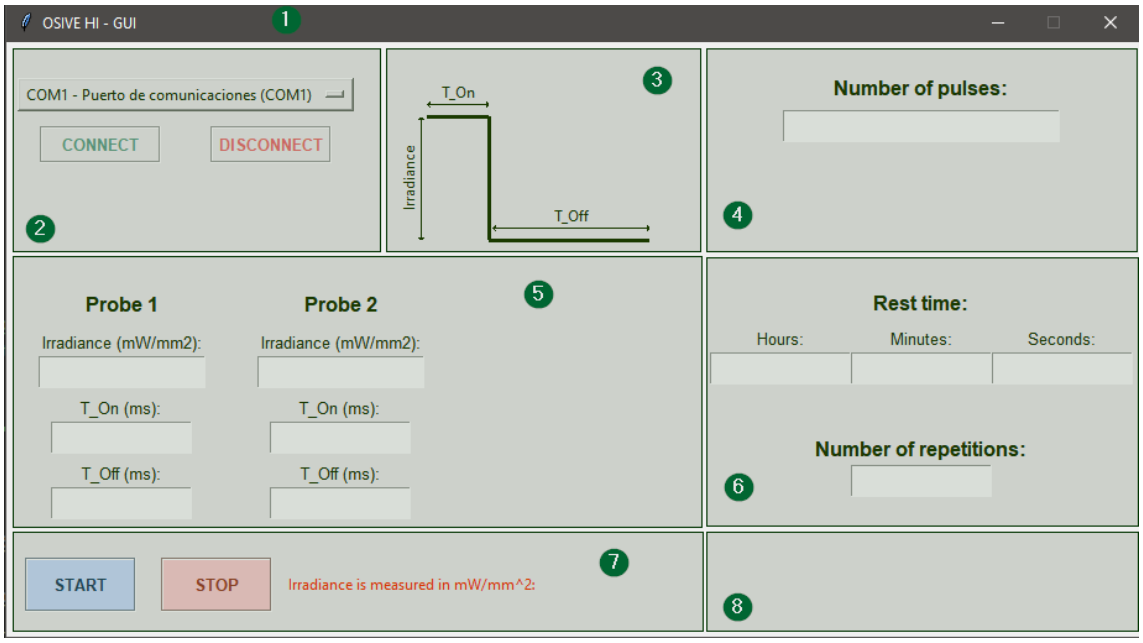


Figura 34. Distribución de los frame de primer nivel en la interfaz gráfica. Fuente: Propia.

La siguiente clasificación se basa en la Figura 34:

1. Ventana: objeto *Window* de tkinter. El objeto principal para generar la ventana y que a su vez sirve como el primer contenedor. La ventana se ha subdividido en una matriz 3x3 para garantizar sencillez en el diseño, legibilidad en el código y sencillez para futuras modificaciones; persiguiendo un diseño modular. Si por ejemplo quisiera cambiarse la distribución de los ítems, al estar todo empaquetado en sus correspondientes *frames*, solo habría que cambiar la fila y la columna donde se quieren localizar.
2. Frame COM: En este frame conviven la lista de puertos COM disponibles, el botón para abrir el puerto COM y el botón para cerrarlo.
3. Frame Canvas: En este frame reside un objeto Canvas, predeterminado de tkinter para dibujo de gráficos. Se pretende con el gráfico de la señal dar una idea general sobre el tipo de señales que van a enviarse vía USB.
4. Frame de pulsos: Este frame alberga todo lo necesario para informar el cometido del campo de texto incorporado en él. Si el usuario intenta introducir un valor que no sea un número entero, inmediatamente el botón de START,

dedicado a escribir el mensaje en el puerto COM abierto, se deshabilita, informándose en la parte posterior derecha el motivo.

5. Frame de Probes: Se define "*Probe*" como traducción de puntero lumínico, es decir, cada uno de los experimentos que pueden realizarse simultáneamente con OSIVE HI. Aquí, el usuario configurará los valores de irradiancia de la señal, tiempo de encendido (nivel alto) y tiempo de apagado (nivel bajo). Con el fin de aplicar una validación individual a cada campo de escritura, se ha habilitado la siguiente subestructura de frames:

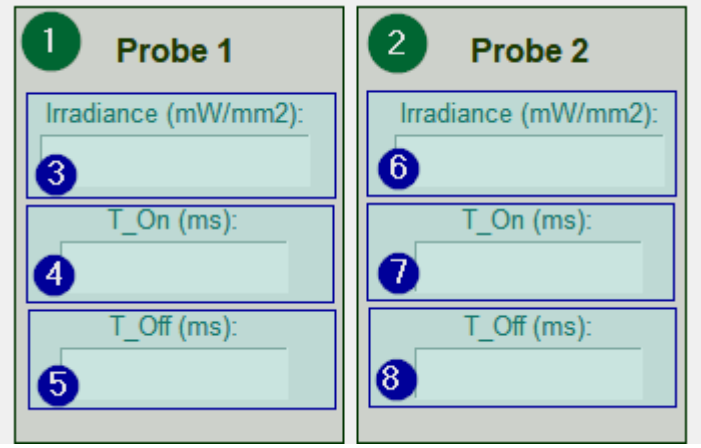


Figura 35. Subdivisión de un frame maestro de puntero. Fuente: Propia.

En primer lugar, se tiene un frame maestro para cada puntero (1 y 2). Dentro de cada uno de estos, se encuentran otros 3 frames. Uno para irradiancia (que contiene tanto la label de texto como el campo de escritura), y otros dos para tiempo de encendido y tiempo de apagado, respectivamente y siguiendo el mismo esquema. Con esto se consigue robustez en el código y en la propia interfaz, además de poder aplicar validaciones a nivel de *entry*.

6. Frame de Tiempo de Reposo: Aquí se controlan los tiempos de reposo y número de repeticiones de la señal. Al igual que en el frame definido en el punto 3, se restringe el envío si se introduce un número decimal.
7. Frame botones principales: En este frame se albergan los botones principales para enviar mensajes a la Arduino Nano. Además, una label informativa indica la medida de la irradiancia.
8. Frame de avisos: Último de los principales frames. Alberga objetos de texto previamente creados que se harán visibles cuando alguna de las validaciones en tiempo real se activen, con el fin de que el usuario sepa la causa de por qué no puede efectuar comunicaciones.

7.3.2 Métodos de los elementos

Se definirán aquí los métodos desarrollados para el funcionamiento requerido de la interfaz.

7.3.2.a Métodos de restricción

Se denomina método de restricción a aquel método que sirve para restringir el acceso a los puertos COM- Existen tres tipos principales:

- El valor introducido está fuera del rango [0, 10].
- El valor introducido es menor que 0,01.
- El valor introducido no es un número entero.

Como ejemplo de implementación, se tomará la validación del valor de irradiancia en el puntero 1. Si quieren revisarse los demás métodos, estos están disponibles en el Anexo 4 de este documento, que contiene el código fuente de la interfaz gráfica:

```
#Diccionario usado para validar las entries
fieldsVal = {'entAmp1':True, 'entTOn1':True, 'entTOff1':True,
'entAmp2':True, 'entTOn2':True, 'entTOff2':True}


#Definición de StringVar, variable de control que tkinter permite
asociar a una entry como su variable de texto
amp1 = tk.StringVar()

#Creación del campo de texto
entAmp1 = tk.Entry(
    master=frmCurrent1,
    width=10,
    insertwidth=1,
    justify="center",
    font=entryFnt,
    textvariable=amp1 #Pasar StringVar a la entry
)
#Nombrar a la entry para que el tracking pueda identificar el widget:
entAmp1.name = "entAmp1"

#Variable para trackear la entry. regAmp1 es un registro de lo que se
introduce en el frame frmCurrent1 en base a ampVal:
regAmp1 = frmCurrent1.register(ampVal)

#En base al nombre de la entry, se configura para que se registre el
valor con la instrucción de tracking definida arriba:
entAmp1.config(validate="key", validatecommand=(regAmp1, '%P'))

#Definición del método ampVal, que en base al valor del booleano
ampInp, se cumple una u otra condición:
def ampVal(ampInp):
    try:
        focusedWidget = window.focus_get() #Se obtiene el widget que
tiene la atención del usuario en ese momento
        if ampInp:
            y = float(ampInp)  # cast de string a float
            if y > 10 or y < 0:
                raise ValueError()  # creación y lanzamiento del
objeto "ValueError", de tipo ValueErrorException
            else:
                hideLabel(lblAmpOutOfRange)
                fieldsVal[focusedWidget.name] = True
                allFlagsVal()
    except ValueError as ex: #Ejecución del objeto ValueError
        showLabel(lblAmpOutOfRange) #Se muestra la label
correspondiente al error de irradiancia
        fieldsVal[focusedWidget.name] = False #Se setea el
correspondiente value del diccionario a False
        allFlagsVal() #Se llama al método para que compruebe los
values del diccionario y se desactive el botón
    return True
```

En términos generales, se puede extraer lo siguiente:

- Un diccionario de Python alberga un booleano para cada *widget* de tipo Input (campo de escritura). Se inicializan a True para que los botones estén

disponibles desde el momento en que la interfaz se ejecuta, y en el momento en que uno de los booleanos cambia a False, los botones destinados a comunicarse con la Arduino son deshabilitados.

- Con ayuda de las variables de control (StringVar), se puede obtener el valor del widget Input si esta se le relaciona.
- Es necesario nombrar el widget Input, de manera que sea posible pasarlo a las instrucciones de tracking.
- El booleano de validación (ampInp en el ejemplo expuesto anteriormente) vale False cuando lo introducido en el widget Input es un string vacío o nada (*None*). Por tanto devuelve True. En el momento en que algo se escribe, también se cuenta como valor True. Esta lógica asegura la validación del flujo de escritura completo del usuario.
- El string se actualiza constantemente. Tanto si el usuario borra caracteres como si los introduce. Por ende, la validación se hace también constantemente.

En la figura 36, expuesta a continuación puede verse el comportamiento de la interfaz una vez ha saltado el error:

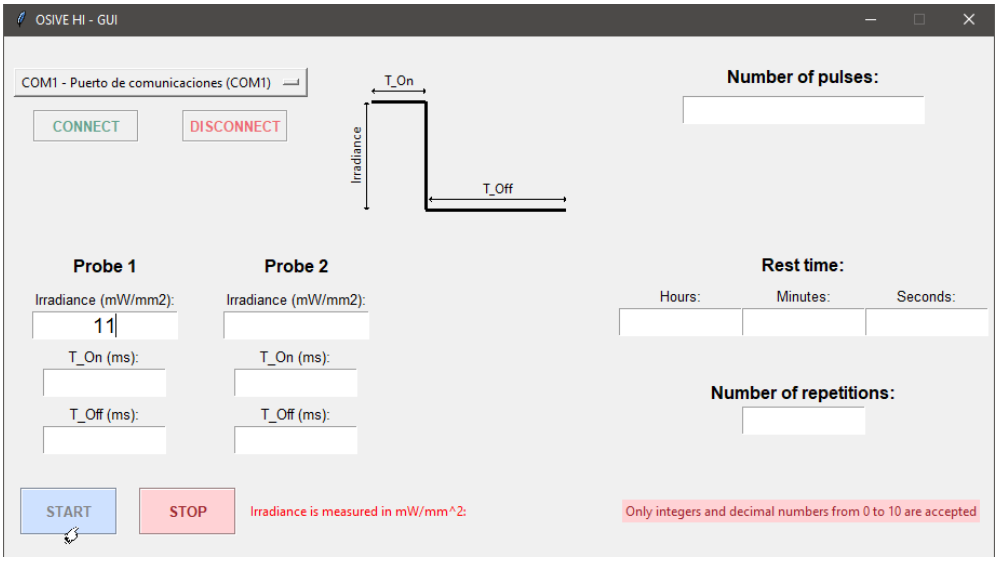


Figura 36. Comportamiento de la interfaz gráfica cuando una validación levanta el objeto de error. Fuente: Propia.

7.3.2.b Métodos callback

En Tkinter, se denomina método callback a aquel método que es asociado a un objeto de tipo botón (Button). A continuación se listan los usados en la interfaz gráfica de OSIVE HI, relacionando cada uno con su botón. Si quiere consultarse información detallada de su función, pueden recurrirse a los métodos desarrollados en los subapartados de la Sección 7.1.1, que exponía los métodos de callback en vista a las comunicaciones, pues es para lo que se han diseñado. También puede consultarse el código correspondiente en el Anexo 4 de este documento.

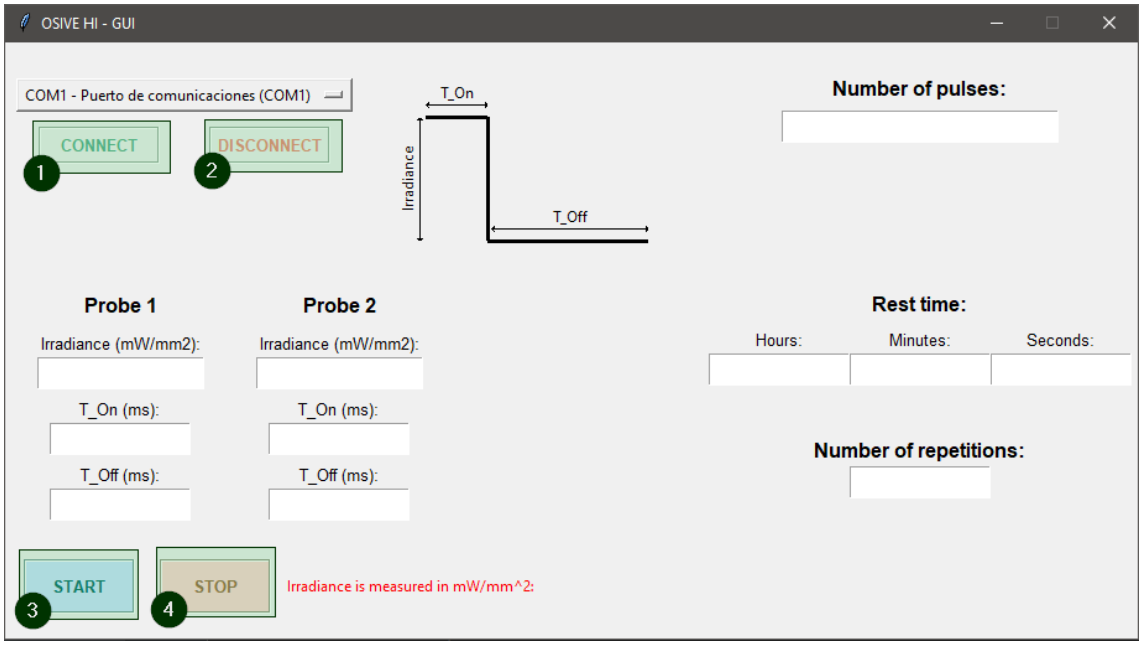


Figura 37. Vista de la interfaz gráfica OSIVE HI en relación con sus métodos de callback. Fuente: Propia.

1. ConnectComOnBtn(): Se conecta al puerto COM seleccionado en la lista desplegable y deshabilita el botón CONNECT.

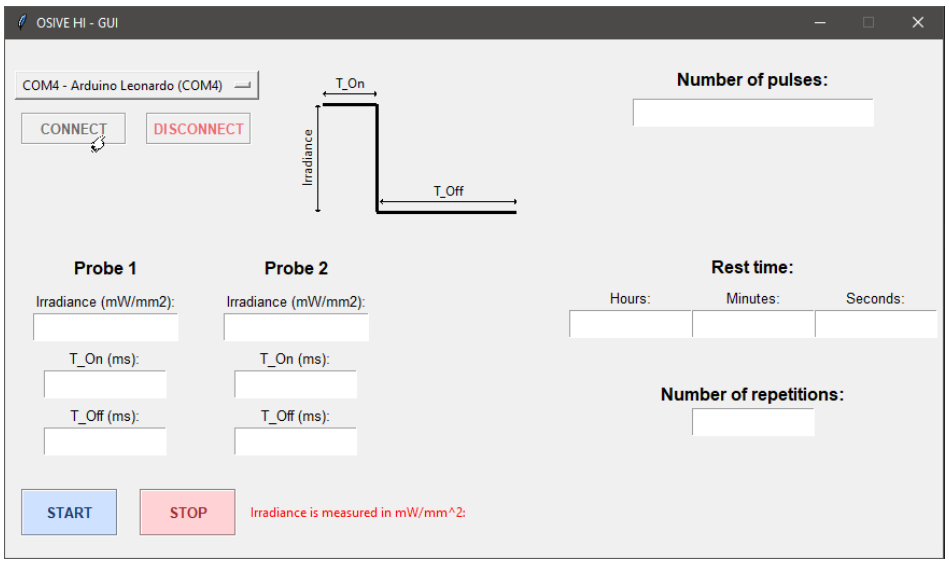


Figura 38. Comportamiento de la interfaz gráfica cuando se pulsa el botón CONNECT. Fuente: Propia.

2. disconnectComOnBtn(): Se desconecta del puerto COM con el que se están produciendo comunicaciones y vuelve a habilitar el botón CONNECT.

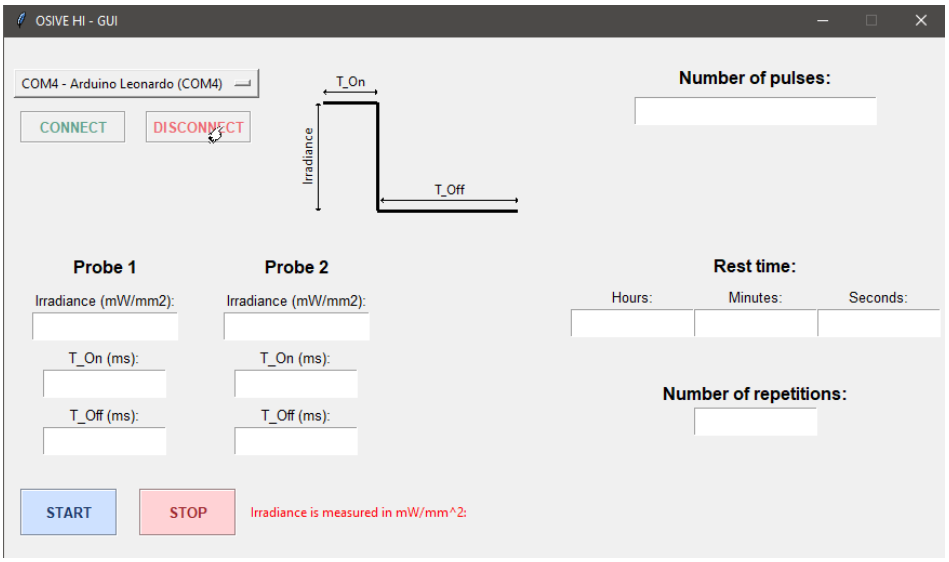


Figura 39. Comportamiento de la interfaz gráfica cuando se pulsa el botón DISCONNECT. Fuente: Propia.

3. sendValues(): Una vez introducidos todos los valores en sus correspondientes widgets Input, al pulsar el botón START se llamará al método sendValues() para generar el mensaje y escribirlo en el puerto COM. Tras efectuar la comunicación, se informa al usuario.

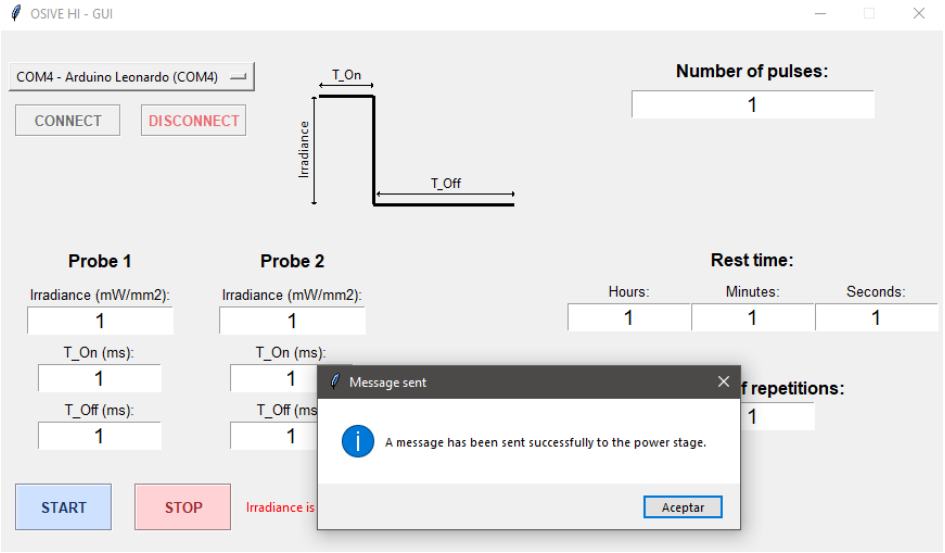


Figura 40. Comportamiento de la interfaz gráfica cuando se envía un mensaje a Arduino Nano. Fuente: Propia.

4. sendStopMessage(): Si se está produciendo un experimento y quiere abortarse, al pulsar el botón STOP se sigue la misma lógica que en el método sendValues(), solo que en esta ocasión se envía una trama de ceros al microcontrolador. Tras efectuarse la comunicación, se informa al usuario.

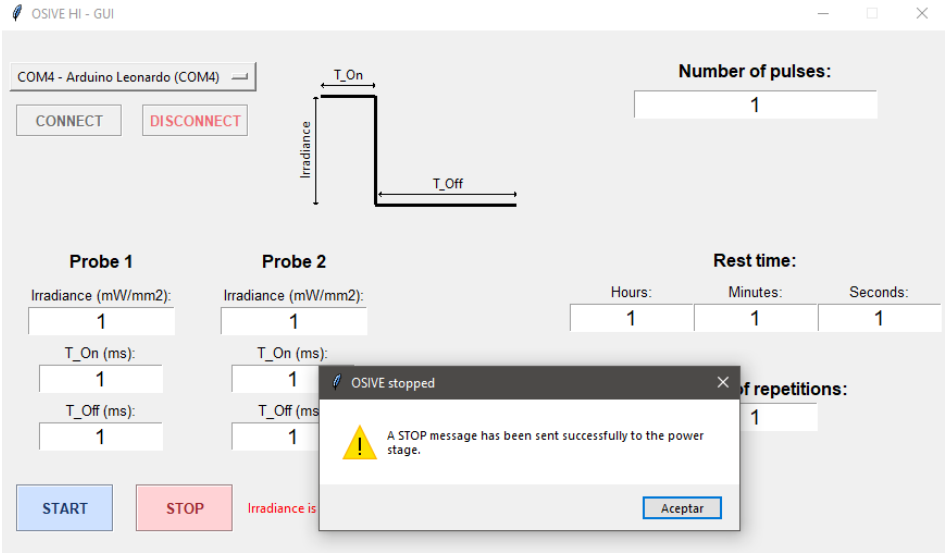


Figura 41. Comportamiento de la interfaz gráfica cuando se envía un mensaje de parada a Arduino Nano. Fuente: Propia.

8. PROTOTIPAJE Y VALIDACIÓN

Se recoge en este apartado la información pertinente al modelo desarrollado para este proyecto y su implementación. Finalmente, el prototipo del dispositivo OSIVE HI dispone de una interfaz gráfica dedicada para comunicarse con el microcontrolador Arduino, además de una unidad central que alimenta y comanda los punteros lumínicos, siendo necesario diferenciar dos tareas principales: la propia alimentación y los protocolos de comunicaciones usados.

8.1 Unidad central

La unidad central está compuesta de una placa de circuito impreso y el hardware soldado a ella. El servicio de impresión y revelado de la PCB fue subcontratado y, cuando esta llegó junto con los componentes, se procedió al soldado de toda la unidad en el banco de trabajo del laboratorio, usando las herramientas destinadas a este cometido (véase Apartados 3.4.12 y 3.4.13).

Una vez se soldaron los componentes y se comprobó con el multímetro la correcta conductividad esperada en todo el circuito, la unidad central se conectó a la red eléctrica para observar su comportamiento. No se produjeron sobrecalentamientos, dándose así por finalizada esta fase del desarrollo del presente trabajo de fin de grado.

Figura 42. PCB de OSIVE HI sin componentes, asegurada en el banco de trabajo. Fuente: Propia.

Figura 43. Unidad central de OSIVE HI, después de soldar todos los componentes a la PCB. Fuente: Propia.

En la Figura 43, adjunta arriba, puede diferenciarse entre los módulos de alimentación conmutados IRM-60-5 (Apartado 3.4.2 para más detalles), la Arduino Nano, con un característico conector USB tipo micro B, y los integrados dedicados a las comunicaciones CAN.

También pueden diferenciarse los conectores de 10 pines para la conexión del bus CAN y alimentación de un puntero lumínico, y los conectores de 4 pines para la expansión de OSIVE HI, de esta forma se permite la ampliación en la alimentación y control de punteros lumínicos adicionales mediante el uso de la misma PCB diseñada pero sin la necesidad de una segunda unidad de microcontrolador Arduino. Además puede apreciarse el conector IEC 60320 C13 del que se obtendrá la señal de la red eléctrica.

Posteriormente se desarrolló con la impresora 3D del laboratorio (véase Apartado 3.4.9) el encapsulado de la unidad central. En la figura expuesta abajo, puede verse cómo quedó finalmente tras ensamblar la PCB en el encapsulado:

Figura 44. Unidad central de OSIVE HI ensamblada en encapsulado. Fuente: Propia.

8.2 Interfaz gráfica

Al acabarse el desarrollo del código, la interfaz gráfica fue validada junto con la Arduino Nano, comprobando el funcionamiento (práctica conocida como *debug*) línea a línea en los sectores críticos para observar si el comportamiento cumplía con lo esperado.

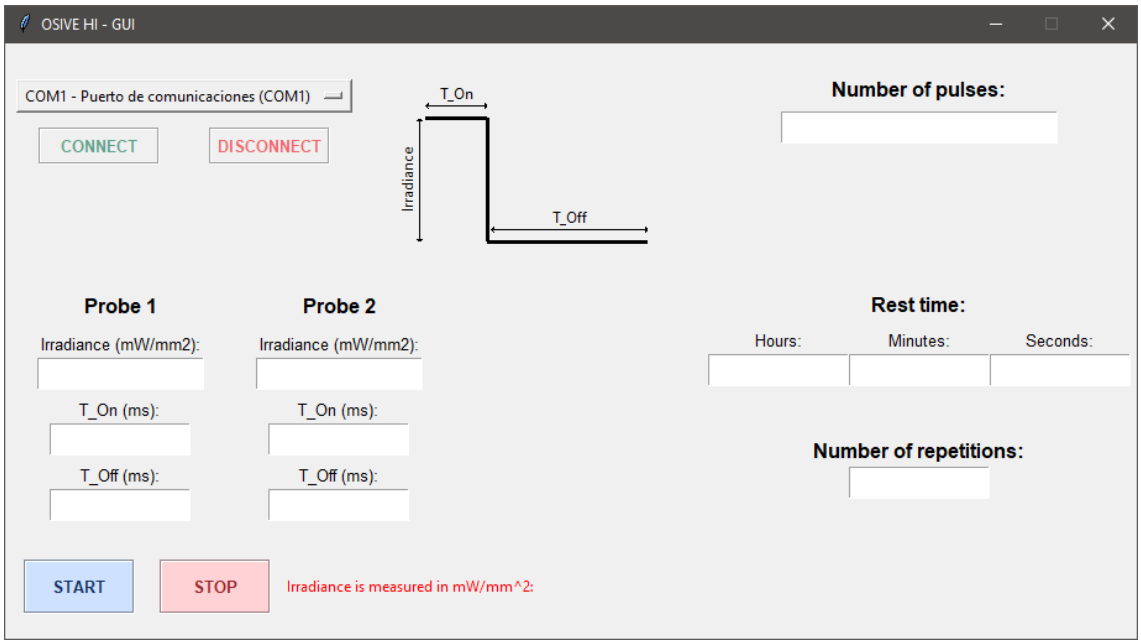


Figura 45. Interfaz gráfica de OSIVE HI ejecutada en Windows 10. Fuente: Propia.

Se encontraron algunos problemas posteriores en la lectura de la trama por parte de la Arduino Nano, que se solucionaron con la modificación del script desarrollado. Finalmente, se ejecutaron varios *journey* de usuario para verificar el correcto funcionamiento del sistema. A continuación se expone uno de ellos a modo de ejemplo, consistente en los siguientes pasos:

n	Descripción	Resultado
1	Ejecución de la interfaz.	La interfaz gráfica se lanza correctamente.
2	Click en la lista de puertos COM.	Aparecen los puertos COM disponibles para su interacción, incluyendo la Arduino.
3	Selección del puerto Arduino.	El puerto COM al que está conectada la Arduino es seleccionado.
4	Click en botón CONNECT.	Se establece la conexión con la Arduino Nano. El botón CONNECT se desactiva.
5	Introducción de valor: 5 en campo Irradiancia 1.	El valor queda reflejado correctamente.
6	Introducción de valor: 3 en campo Irradiancia 2.	El valor queda reflejado correctamente.
7	Introducción de valor: 1.5 en campo tOn1	El valor queda reflejado correctamente.
8	Introducción de valor: 4 en campo tOn2.	El valor queda reflejado correctamente.
9	Introducción de valor: 10 en campo tOff1.	El valor queda reflejado correctamente.
10	Introducción de valor: 20 en campo tOff2.	El valor queda reflejado correctamente.
11	Introducción de valor: 1.5 en campo Número de pulsos.	Bloqueo de botón START. Se muestra la label “Only integers are accepted”.
12	Introducción de valor: 1 en campo Número de pulsos.	El valor queda reflejado correctamente.
13	Introducción de valor: 1 en campo Horas tRest.	El valor queda reflejado correctamente.
14	Introducción de valor: 0 en campo Minutos tRest.	El valor queda reflejado correctamente.
15	Introducción de valor: 0 en campo Segundos tRest.	El valor queda reflejado correctamente.
16	Introducción de valor: 3 en campo nRep.	El valor queda reflejado correctamente.
17	Click en botón START.	La trama ‘j,5,3,1.5,4,10,20,1,1,0,0,3,!’ se envía a la Arduino. Se informa de la recepción de la trama.
18	Click en botón STOP.	La trama ‘s,0,0,0,0,0,0,0,0,0,0,0,!’ se envía a la Arduino. Se informa de la recepción de la trama.
19	Click en botón DISCONNECT.	Se cierra el puerto COM de la Arduino.
20	Click en botón de cerrar ventana.	Termina la ejecución del programa.

Tabla 3. Descripción de un journey de usuario en la interfaz gráfica. Fuente: Propia.

8.3 Protocolo de comunicaciones

La implementación del protocolo de comunicaciones se ha trabajado solamente hasta el punto en que el microcontrolador de Arduino recibe las tramas y las subdivide en distintas variables para su posterior envío.

La validación de estas comunicaciones se realizó con la Arduino Nano de OSIVE HI y un ordenador personal en el laboratorio que ejecutaba en ese momento el código de la interfaz gráfica. Similarmente, se usó una herramienta de depuración del IDE de Arduino para ver cómo el sistema se comportaba ante distintas tramas y, a continuación, se resume una de ellas:

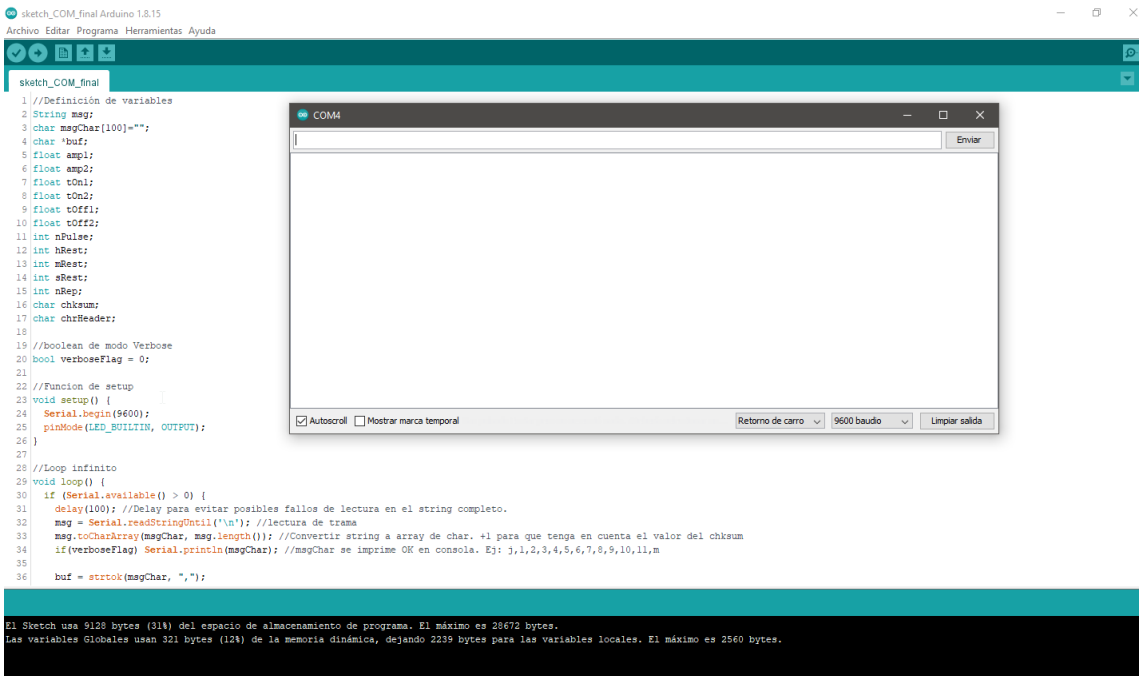


Figura 46. Entorno de desarrollo de Arduino mostrando la herramienta Monitor Serie. Fuente: Propia.

n	Descripción	Resultado
1	Carga de firmware.	Se muestran los mensajes informativos correspondientes. Carga completada.
2	Escribir en Monitor Serie: ‘j,1,1,1,1,1,1,1,1,1,1,1,!’	Arduino devuelve la trama enviada al Monitor Serie. Se imprime el valor de cada variable almacenada para comprobar la correcta subdivisión de la trama.

Tabla 4. Descripción de un journey de usuario en el IDE de Arduino para validación. Fuente: Propia.

9. CONCLUSIONES

El objetivo del proyecto consistía en el desarrollo e implementación de una unidad central, que sirviera a su vez de etapa de potencia a un dispositivo utilizado para ensayos clínicos de optoquimiogenética.

Finalmente se ha conseguido un dispositivo contenido en costes, y lo suficientemente robusto, trazable y sencillo para que sea de utilidad al equipo de investigación especializado en optoquimiogenética del *Centro de Investigación Príncipe Felipe*, haciendo uso de los conocimientos adquiridos en el Grado en Ingeniería Electrónica Industrial y Automática, y realizando un desempeño de investigación en disciplinas que quedan, en ocasiones, fuera de alcance del programa de estudios del propio grado.

Se desarrolló, por una parte, una placa de circuito impreso que albergara una solución para la alimentación de los punteros lumínicos usados para estimulación en los ensayos. Tras ello, se implementó la interfaz gráfica en base al contexto del proyecto y los requisitos generales presentados. Finalmente, se inició el protocolo de comunicaciones, dotando a la unidad central del sistema de la capacidad de interpretar los mensajes del usuario, enviados a través de un sistema sencillo de comprender por este.

Pese a que el hecho de finalizar el desarrollo cumpliendo los objetivos planteados y pasando las validaciones es una muy agradable noticia, OSIVE HI presenta algunos inconvenientes que podrían corregirse a posteriori. Por ejemplo, puede dotarse de más robustez a la interfaz gráfica, preparándola para manejar excepciones resultados de posibles *corner cases* en los que el usuario no siga el curso establecido.

Además, también cabe mencionar que, para utilizarse el sistema en experimentos, deberá implementarse lo relativo a la comunicación con los punteros lumínicos, ahora en fase de diseño.

10. REFERENCIAS

[1] Joshi, J., Rubart, M., & Zhu, W. (2020). Optogenetics: Background, Methodological Advances and Potential Applications for Cardiovascular Research and Medicine. Frontiers in Bioengineering and Biotechnology.

[2] Method of the Year 2010. (2011). Nat Methods 8, 1.

[3] Perea, G. (2016). Luz para desvelar los misterios del cerebro. Recuperado el 30 de julio de 2021, de Página Web de la Sociedad Española de Bioquímica y Biología Molecular (SEBBM): web2020.sebbm.es

[4] Monreal Trigo, J. et al. (2021), Development and Perspectives of Optochemogenetic Devices at IDM, IWOSMOR XIV (8-9 Julio, València,).

[5] Terrés-Haro, J. M., et al. OSIVE Prototype For Optochemogenetic Stimulation, IWOSMOR XIV (8-9 Julio, València, 2021).

[6] Beltrán Morte, et al. High Irradiance Real-Time Measurement System For In-Vitro Optochemogenetic Stimulation, IWOSMOR XIV (8-9 Julio, València, 2021).

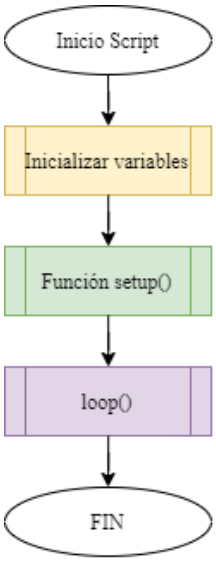
[7] Kaplan, W., J. Wirtz, V., Mantel-Teeuwisse, A., Stolk, P., Duthey, B., & Laing, R. (9 de julio de 2013). Recuperado el 29 de agosto de 2021, de www.who.int/medicines/areas/priority_medicines/MasterDocJune28_FINAL_Web.pdf

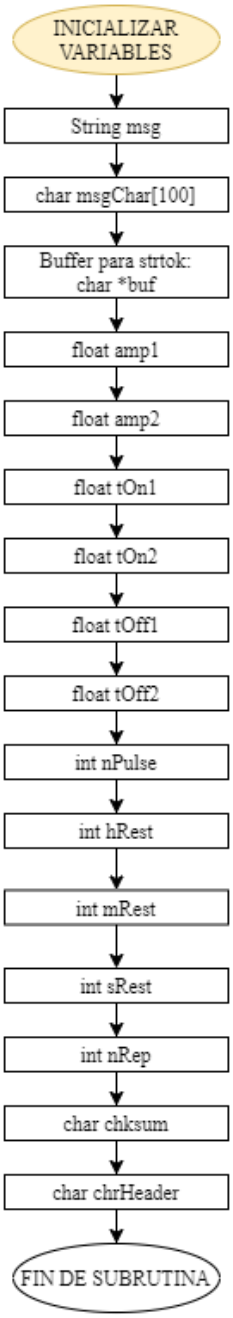
[8] Organización de las Naciones Unidas (ONU). (2011). Desafiíos, oportunidades y acciones en un mundo de 7 mil millones. Nueva York

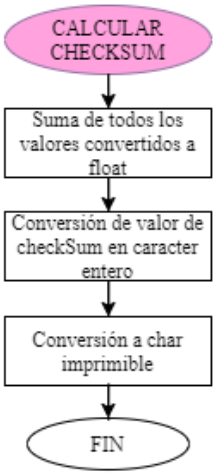
[9] Merck KGaA. (16 de julio de 2019). ¿Está considerada la Esclerosis Múltiple como una enfermedad rara? Obtenido de www.conlaem.es/actualidad/esclerosis-multiple-enfermedad-rara

ANEXO 1. FLUJOGRAMAS DEL CÓDIGO DE ARDUINO

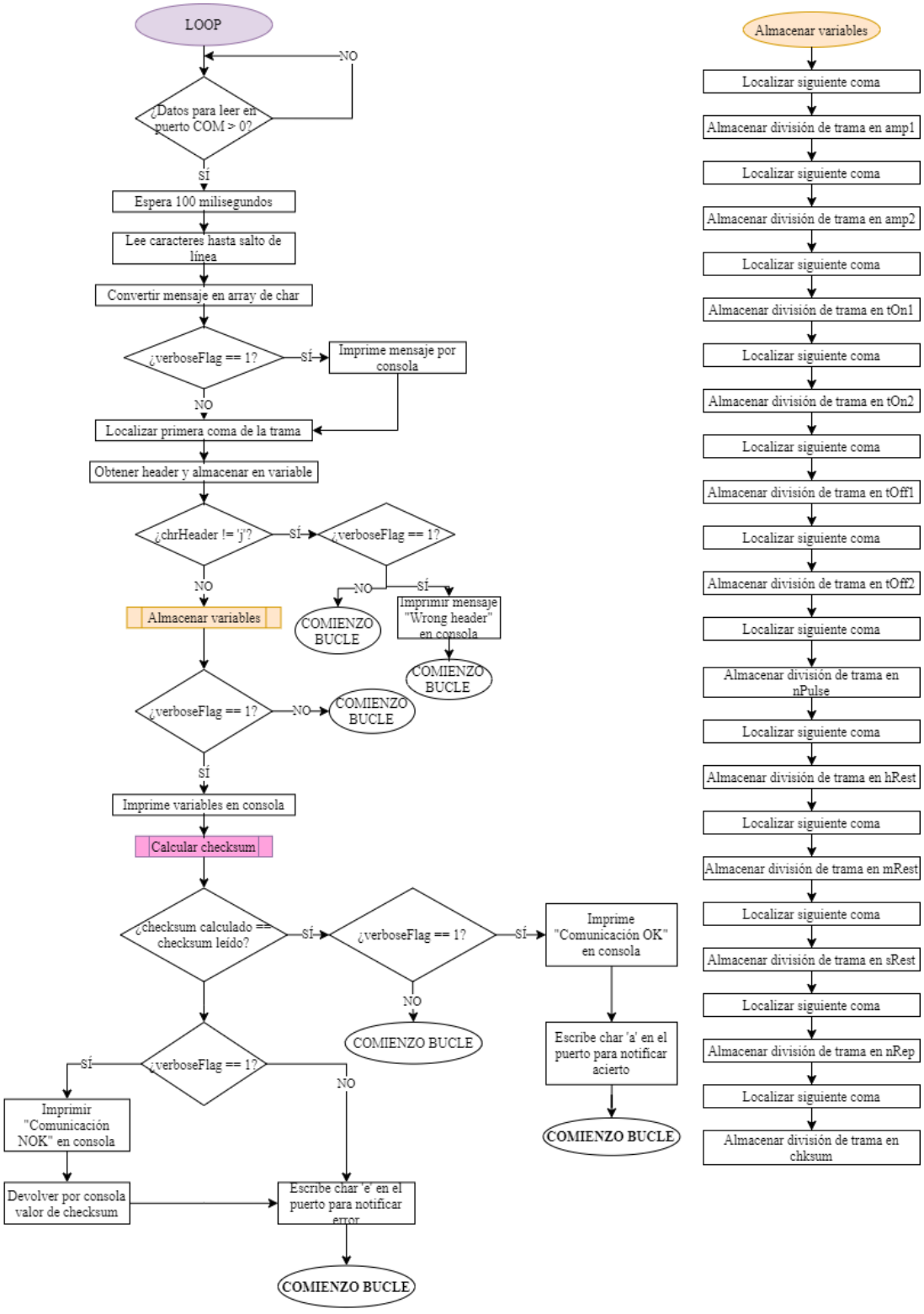
Se presentan en este Anexo los flujogramas detallados en cuanto al código cargado en el microcontrolador de la OSIVE HI que lee e interpreta los mensajes recibidos desde la interfaz gráfica, en cuyo caso se trata de un Arduino Nano. Se han usado colores para identificar más rápidamente las principales rutinas:







La lógica del bucle infinito principal se muestra en la siguiente página:


LOOP
¿Datos para leer en puerto COM > 0?
NO
SÍ
Espera 100 milisegundos
Lee caracteres hasta salto de línea
Convertir mensaje en array de char
¿verboseFlag == 1?
SÍ
Imprime mensaje por consola
NO
Localizar primera coma de la trama
Obtener header y almacenar en variable
¿chrHeader != 'j'?
SÍ
¿verboseFlag == 1?
NO
SÍ
Imprimir mensaje "Wrong header" en consola
COMIENZO BUCLE
NO
Almacenar variables
COMIENZO BUCLE
¿verboseFlag == 1?
NO
COMIENZO BUCLE
SÍ
Imprime variables en consola
Calcular checksum
¿checksum calculado == checksum leído?
SÍ
¿verboseFlag == 1?
SÍ
Imprime "Comunicación OK" en consola
NO
COMIENZO BUCLE
Escribe char 'a' en el puerto para notificar acierto
COMIENZO BUCLE
¿verboseFlag == 1?
SÍ
NO
Imprimir "Comunicación NOK" en consola
Devolver por consola valor de checksum
Escribe char 'e' en el puerto para notificar error
COMIENZO BUCLE
Almacenar variables
Localizar siguiente coma
Almacenar división de trama en amp1
Localizar siguiente coma
Almacenar división de trama en amp2
Localizar siguiente coma
Almacenar división de trama en tOn1
Localizar siguiente coma
Almacenar división de trama en tOn2
Localizar siguiente coma
Almacenar división de trama en tOff1
Localizar siguiente coma
Almacenar división de trama en tOff2
Localizar siguiente coma
Almacenar división de trama en nPulse
Localizar siguiente coma
Almacenar división de trama en hRest
Localizar siguiente coma
Almacenar división de trama en mRest
Localizar siguiente coma
Almacenar división de trama en sRest
Localizar siguiente coma
Almacenar división de trama en nRep
Localizar siguiente coma
Almacenar división de trama en chksum

ANEXO 2. FLUJOGRAMAS DE LA INTERFAZ GRÁFICA

Se presentan en este Anexo los flujogramas detallados en cuanto a la interfaz gráfica con la que el usuario controla OSIVE HI. Para facilitar la comprensión de los diagramas, se dispone de la siguiente leyenda de colores:

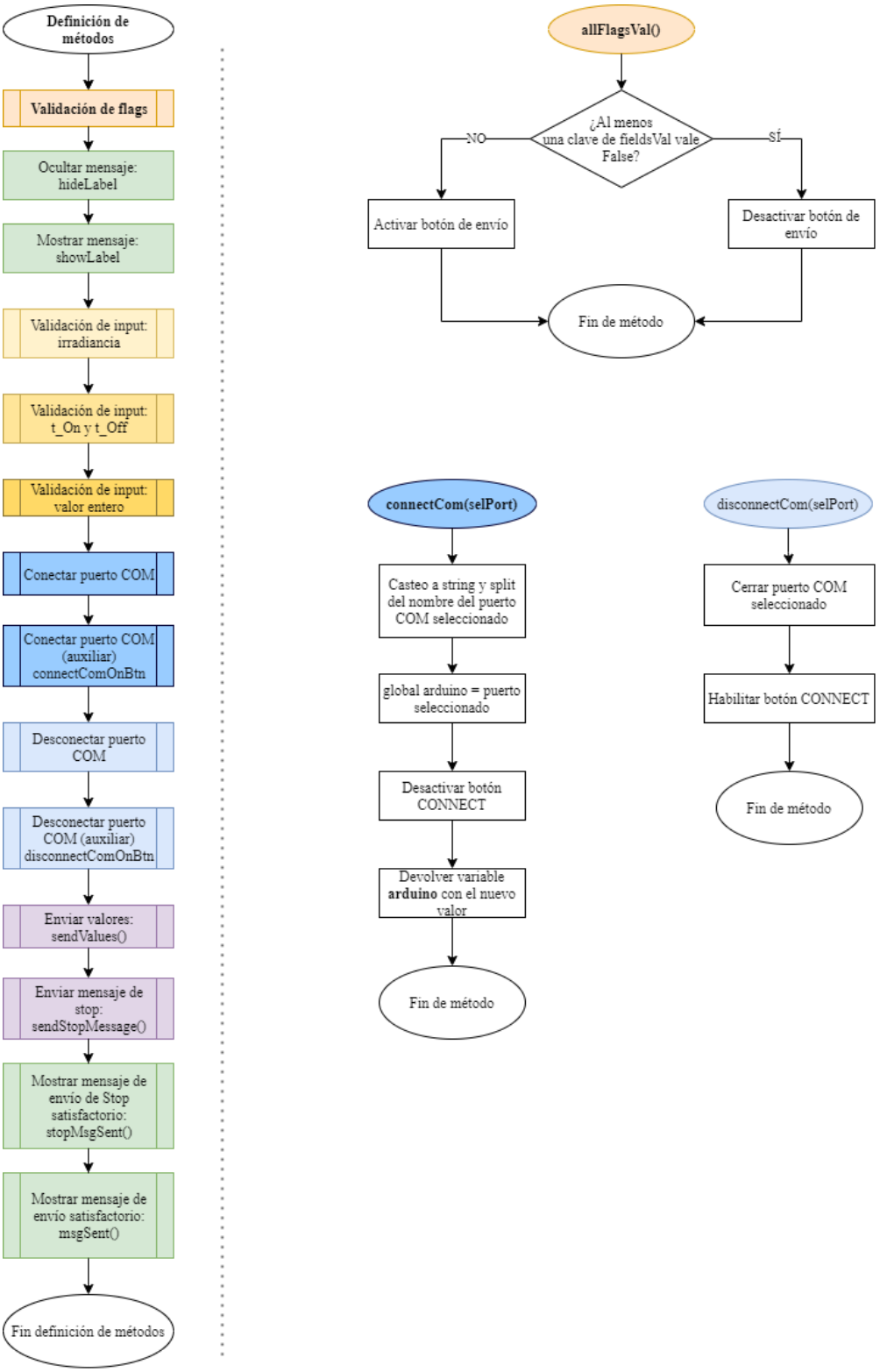
Color (HEX)	**Descripción**
Naranja (FFE6CC)	Lo relativo a la validación en tiempo real de las inputs del usuario: diccionario de flags (fieldsVal), variables de control de tkinter (amp1, amp2, tOn1, tOn2…), métodos de revisión del diccionario (allFlagsVal()), etc.
Verde (D5E8D4)	Lo relativo a mostrar al usuario mensajes relevantes para asegurar el funcionamiento del sistema: Mensajes de restricción, mensajes de comunicación exitosa, etc.
Amarillo (FFF2CC)	Reservado para identificar la lógica de validación de irradiancia (el valor introducido no puede estar fuera del intervalo [0, 10]).
Amarillo (FFE599)	Reservado para identificar la lógica de validación de los tiempos de encendido y apagado (el valor introducido no puede ser menor a 0,01).
Amarillo (FFD966)	Reservado para identificar la lógica de validación de número entero (el valor introducido no puede ser un número decimal).
Azul (99CCFF)	Lo relativo a comunicaciones UART entre el puerto COM del PC y la Arduino Nano.
Azul (DAE8FC)	Reservado para lo relativo a comunicaciones UART, solamente referente a rutinas de desconexión y parada.
Morado (E1D5E7)	Reservado para identificar la lógica de los métodos sendValues() (envío de trama) y sendStopMessage() (envío de mensaje de parada).

Tabla 5. Leyenda de colores de diagramas de flujo. Fuente: Propia.

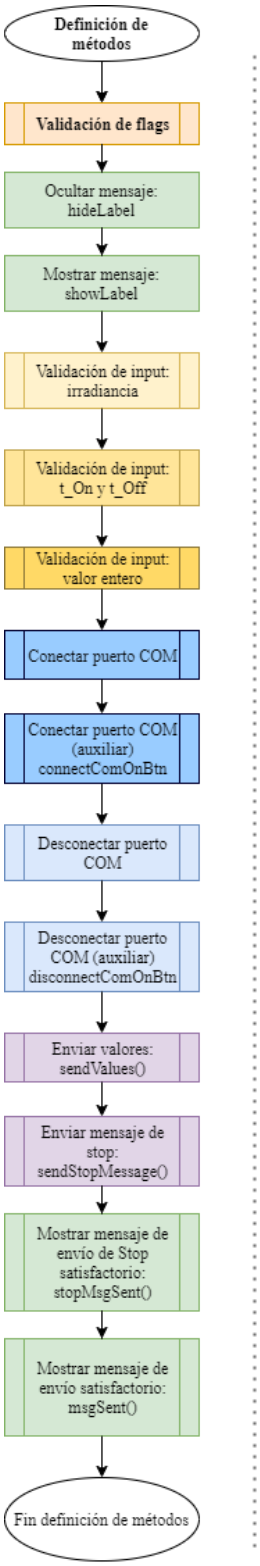
Anexo 1.1 Resumen e inicialización de variables

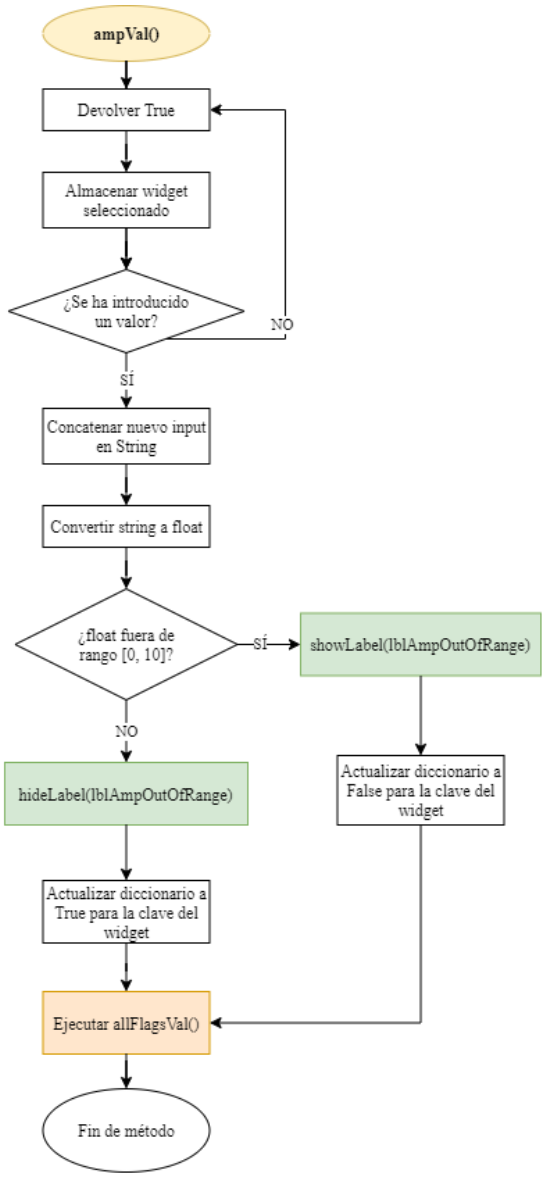
A la izquierda, resumen de un "journey" completo de usuario que garantiza el correcto funcionamiento de la interfaz. A la derecha, importado de librerías, generación de variables y generación de la ventana de la aplicación.

Anexo 1.2 Definición de métodos

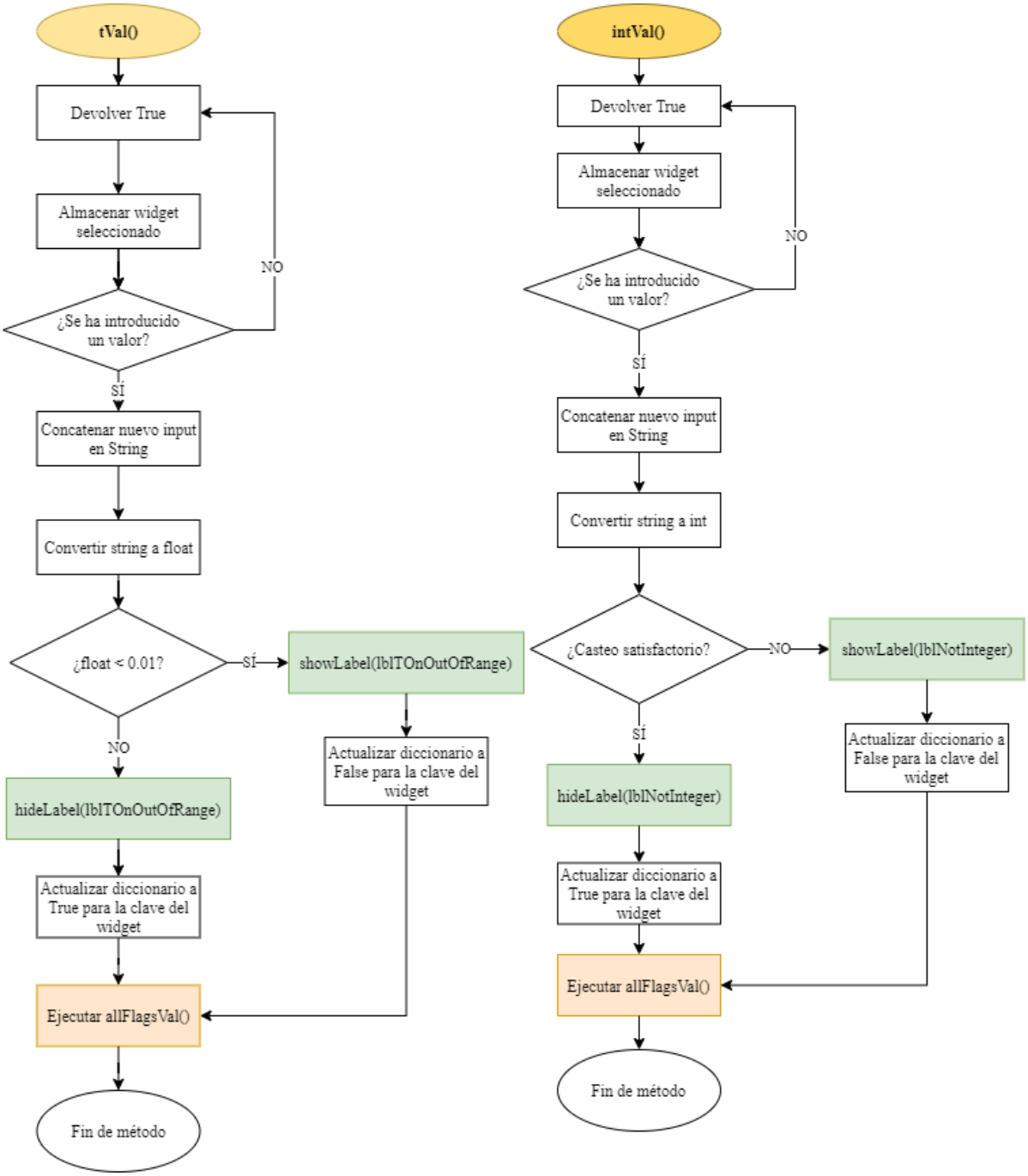


Mientras que a la izquierda se muestra un resumen de en qué orden son generados los métodos, a la derecha se van mostrando las distintas lógicas implementadas para cada uno. En este caso, el método de validación del diccionario de flags para decidir si deshabilitar o no el envío de mensajes, y la conexión y desconexión del puerto COM.

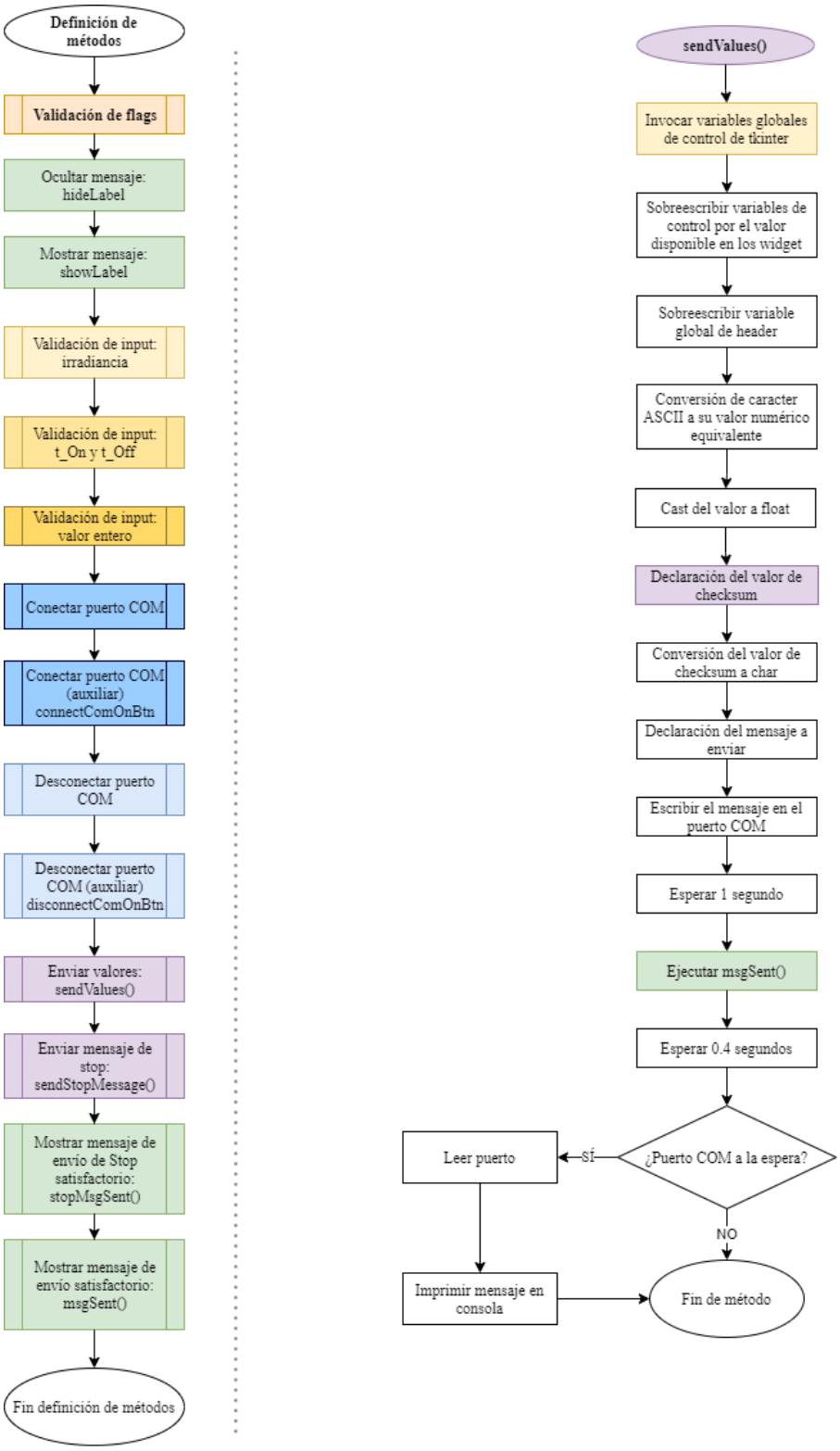




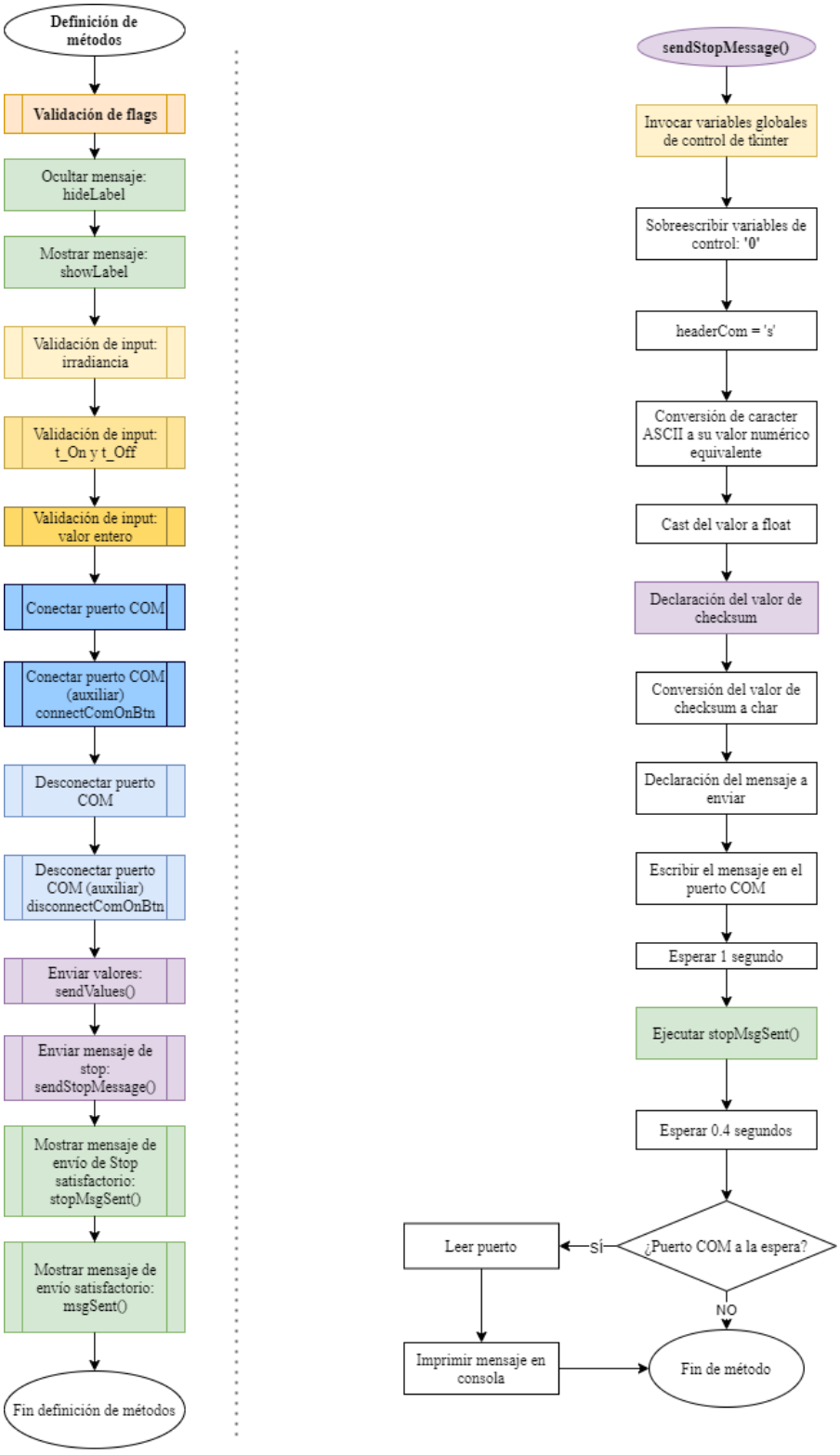
Método de validación de valores de irradiancia. La comprobación se hace en tiempo real, registrando cada valor que el usuario introduce en el campo correspondiente.



Método de validación de valores para los valores de t_{On} y t_{Off}. La comprobación se hace en tiempo real, registrando cada valor que el usuario introduce en el campo correspondiente.

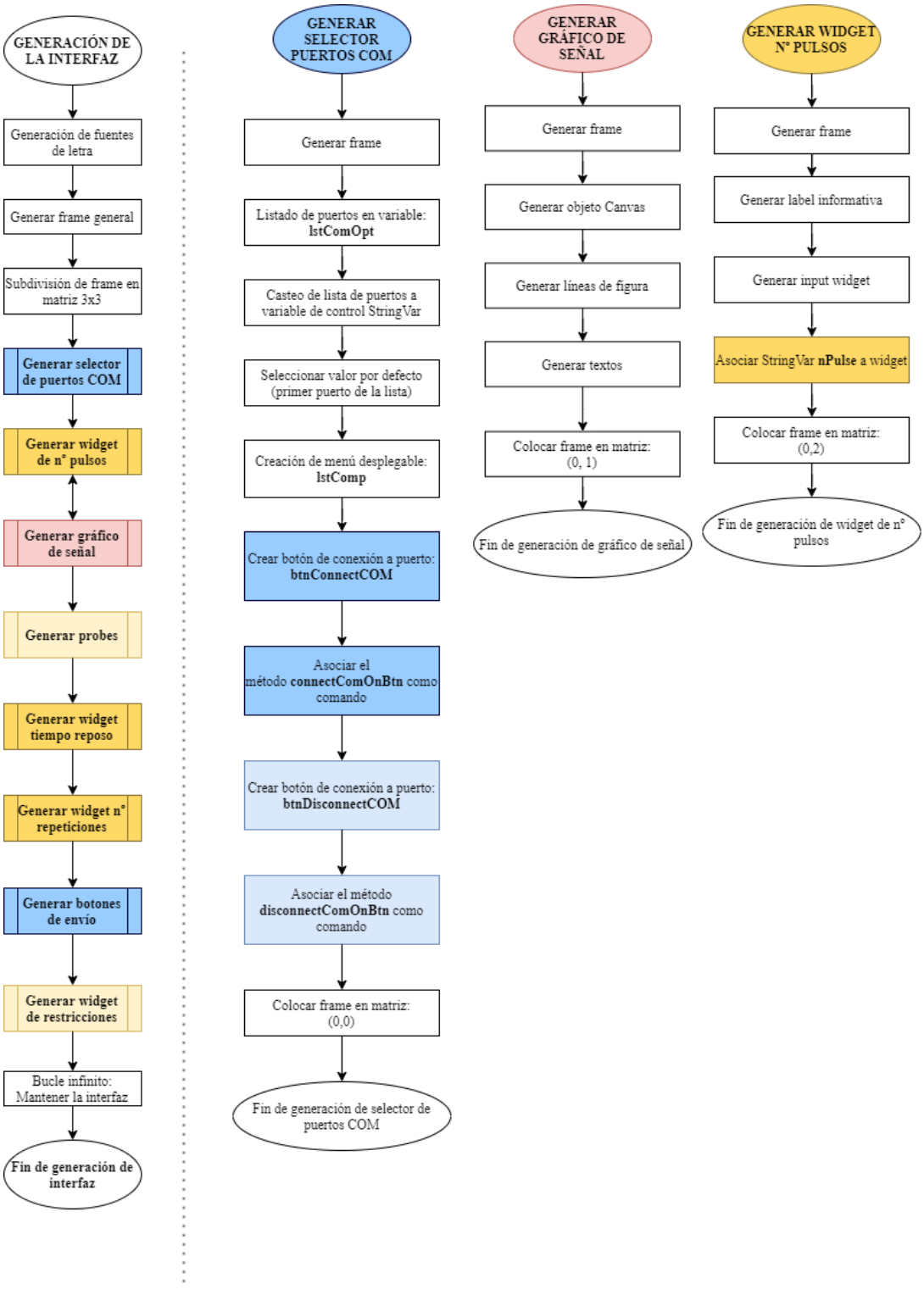


Método que extrae los inputs de cada campo, forma el mensaje y lo envío a la Arduino Nano. Cuando la comunicación acaba de forma satisfactoria, se imprime un mensaje informativo en pantalla y se obtiene la trama enviada para imprimirla en consola, con objetivo de validar de una manera adicional la información enviada.

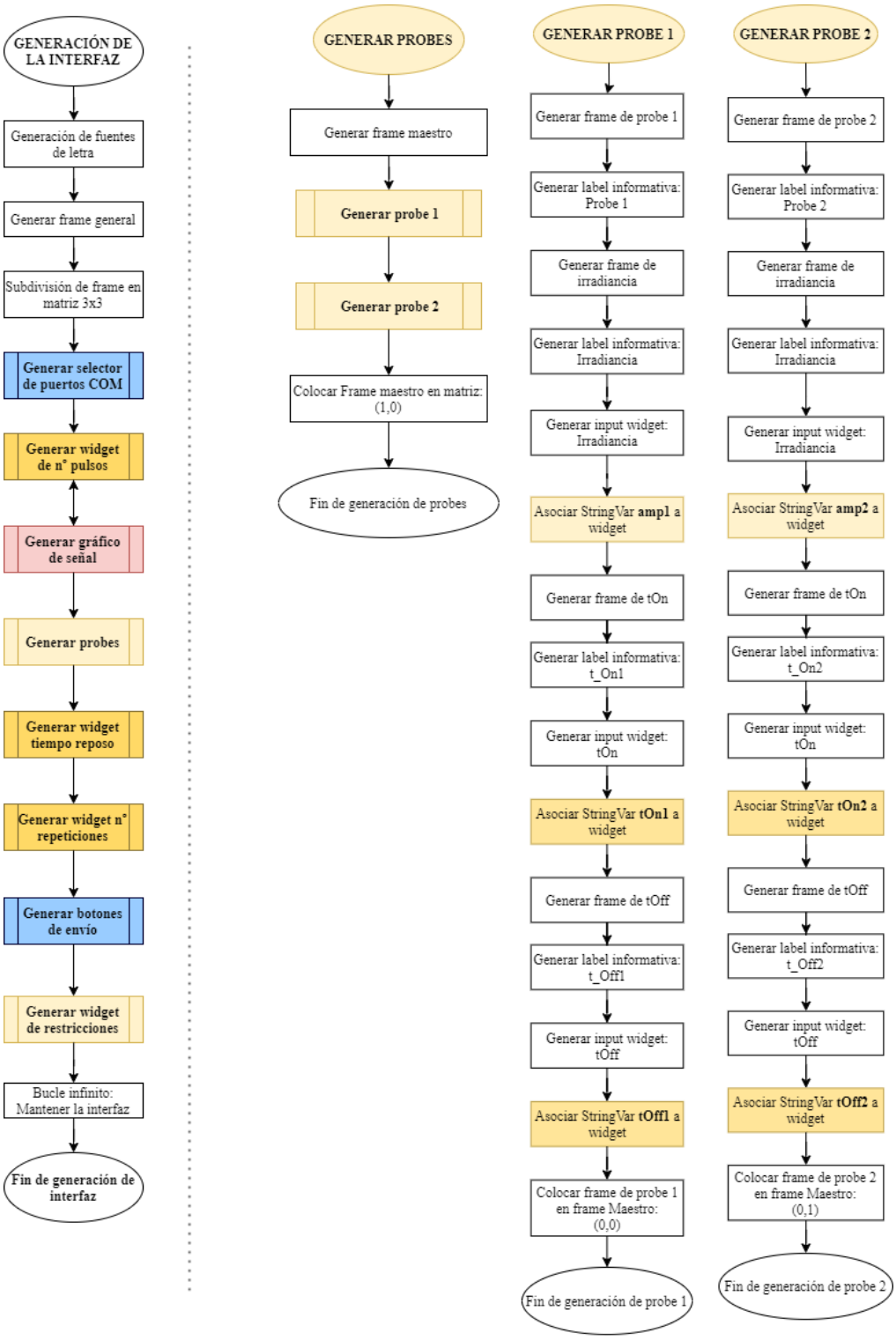


Método que se ejecuta cada vez que se pulsa el botón de STOP (detalles de la generación del botón en el Anexo 1.3).

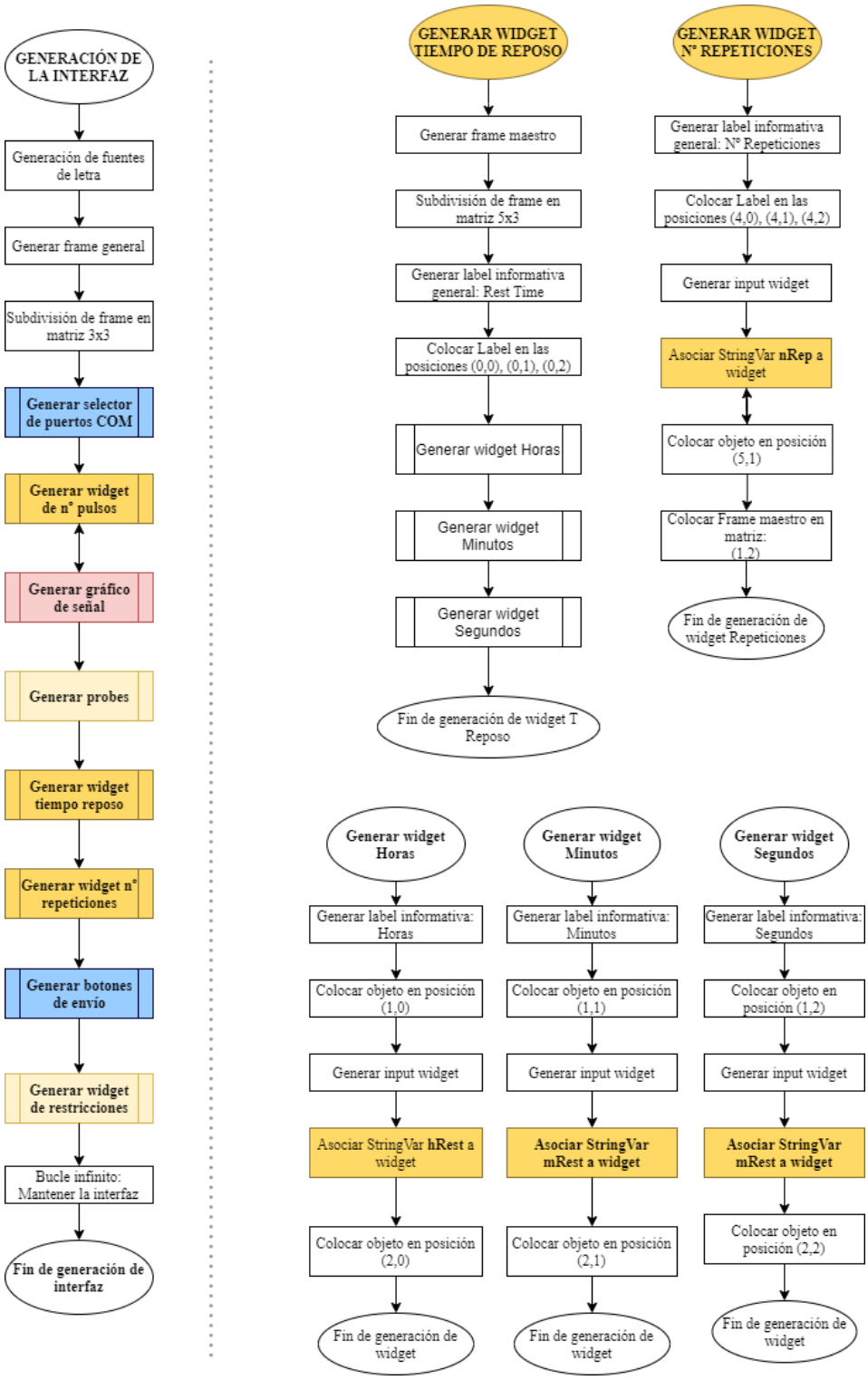
Anexo 1.3 Generación de la interfaz



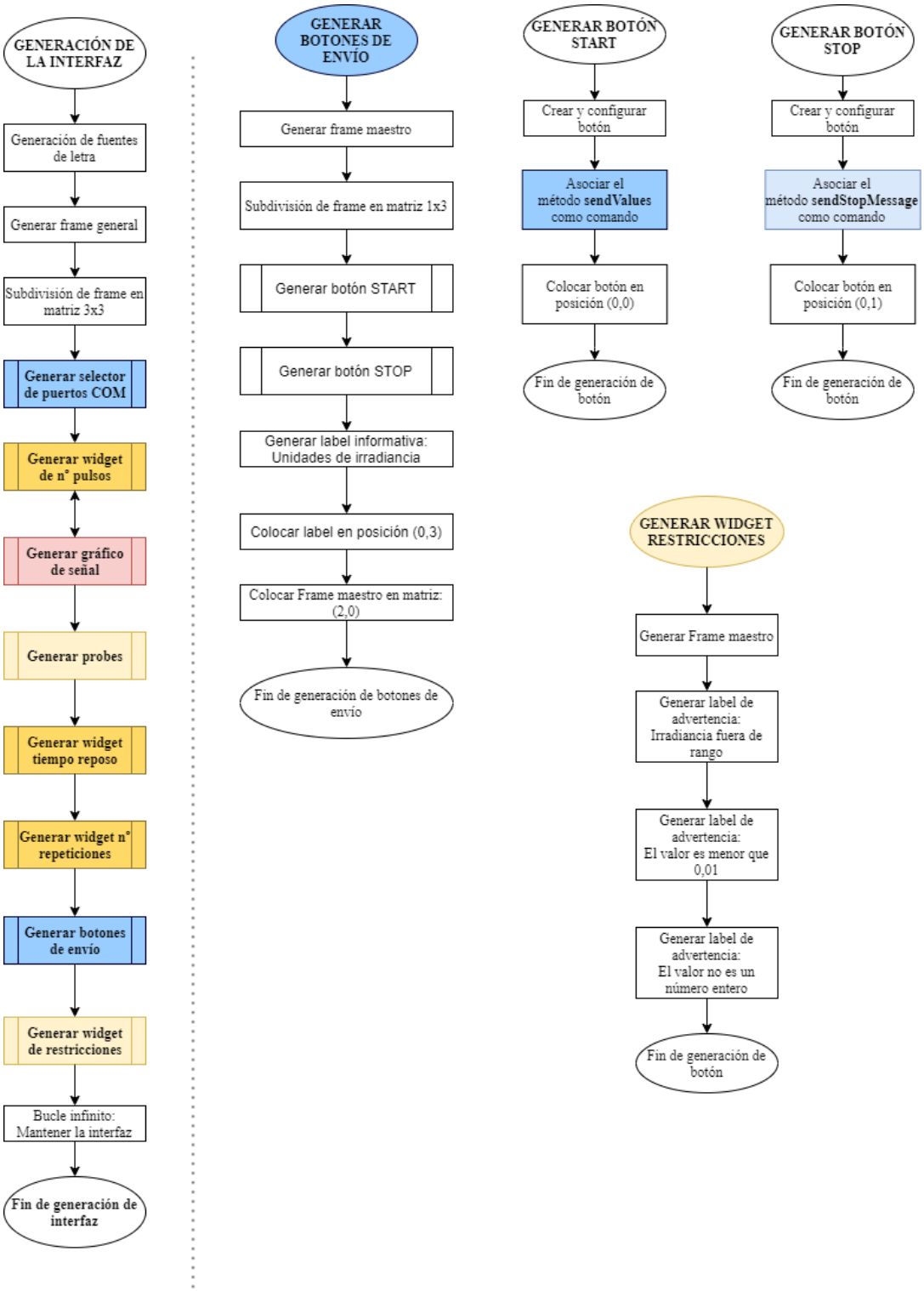
Mientras que a la izquierda se muestra un resumen de en qué orden son generados los elementos de la interfaz, a la derecha se van mostrando las distintas lógicas implementadas para la generación de cada widget. Generar un widget incluye la asociación de métodos a este, su asociación a un frame, su estilo y formato, etcétera.



Aquí se define la generación de los *probes*. Cada *probe* representa un *ensayo*, pudiéndose introducir en cada uno los valores para su correspondiente puntero.



Diagramas de flujo que muestran la generación del widget para introducir horas, minutos y segundos de reposo. Nótese que quedan asociadas las variables de control a cada uno de los campos, siendo posible así validar los valores introducidos.



Finalmente, se muestra el flujo seguido para crear los botones de envío de mensaje (START), envío de mensaje de parada (STOP), y la parte de la interfaz dedicada a mostrar mensajes de error cuando salten las validaciones.

ANEXO 3. CÓDIGO ARDUINO

Se presenta en el siguiente anexo el código de Arduino utilizado para la interpretación de los mensajes enviados por el usuario desde la interfaz gráfica.

```
//Definición de variables
String msg;
char msgChar[100]="";
char *buf;
float amp1;
float amp2;
float tOn1;
float tOn2;
float tOff1;
float tOff2;
int nPulse;
int hRest;
int mRest;
int sRest;
int nRep;
char chksum;
char chrHeader;

//boolean de modo Verbose
bool verboseFlag = 0;

//Función de setup
void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
}

//Loop infinito
void loop() {
  if (Serial.available() > 0) {
    delay(100); //Delay para evitar posibles fallos de lectura en el string
completo.
    msg = Serial.readStringUntil('\n'); //lectura de trama
    msg.toCharArray(msgChar, msg.length()); //Convertir string a array de
char. +1 para que tenga en cuenta el valor del chksum
    if(verboseFlag) Serial.println(msgChar); //msgChar se imprime OK en
consola. Ej: j,1,2,3,4,5,6,7,8,9,10,11,m

    buf = strtok(msgChar, ",");
    chrHeader = buf[0]; //Guardar valor del header para usarlo en el checksum

    if(buf[0] != 'j') {
      if(verboseFlag) Serial.println("Wrong header"); //Imprime en consola que
el header es incorrecto
    }
    else { //Si el header es correcto, almacena las variables
      buf = strtok(NULL, ",");
      amp1 = atof(buf);

      buf = strtok(NULL, ",");
      amp2 = atof(buf);

      buf = strtok(NULL, ",");
      tOn1 = atof(buf);

      buf = strtok(NULL, ",");
      tOn2 = atof(buf);

      buf = strtok(NULL, ",");
      tOff1 = atof(buf);
```

```
      buf = strtok(NULL, ",");
      tOff2 = atof(buf);

      buf = strtok(NULL, ",");
      nPulse = atoi(buf);

      buf = strtok(NULL, ",");
      hRest = atoi(buf);

      buf = strtok(NULL, ",");
      mRest = atoi(buf);

      buf = strtok(NULL, ",");
      sRest = atoi(buf);

      buf = strtok(NULL, ",");
      nRep = atoi(buf);

      buf = strtok(NULL, ",");
      chksum = buf[0];

      if(verboseFlag) { //Imprime en consola los valores de las variables para
validación
        Serial.println(amp1);
        Serial.println(amp2);
        Serial.println(tOn1);
        Serial.println(tOn2);
        Serial.println(tOff1);
        Serial.println(tOff2);
        Serial.println(nPulse);
        Serial.println(hRest);
        Serial.println(mRest);
        Serial.println(sRest);
        Serial.println(nRep);
        Serial.println(chksum);
      }
    }

    //Cálculo de valor de Checksum
    float checkSumFloat = (float)chrHeader + amp1 + amp2 + tOn1 + tOn2 + tOff1
+ tOff2 + (float)nPulse + (float)hRest + (float)mRest + (float)sRest +
(float)nRep;
    //la suma en float coincide con la suma de Python
    int checkSum = (int(checkSumFloat)%95)+32;
    char checkSumAscii = char(checkSum);

    //Si el valor del checkSum calculado coincide con el valor de checkSum
leído, se considera la comunicación exitosa
    if(chksum == checkSumAscii) {
      if(verboseFlag) Serial.println("Comunicación correcta");
      Serial.write('a');
    } else {
      if(verboseFlag)  {
        Serial.println("Comunicación incorrecta");
        Serial.println(checkSumAscii);
      }
      Serial.write('e');
    }
  }
}
```

ANEXO 4. CÓDIGO PYTHON

Se presenta en el siguiente anexo el código de Python utilizado para la creación de la interfaz gráfica:

```
#!/usr/bin/env python
"""Ejecuta una interfaz gráfica para que un usuario
controle el módulo OSIVE HI."""

try:  # Python 2
    import Tkinter as tk
    from Tkinter import messagebox
    import tkFont
    import serial
    import serial.tools.list_ports
    import time
except ModuleNotFoundError:  # Python 3
    import tkinter as tk
    from tkinter import messagebox
    import tkinter.font as tkFont
    import serial
    import serial.tools.list_ports
    import time

__author__ = "Juan Diego Coronel Montesinos"
__credits__ = ["Juan Diego Coronel Montesinos", "Javier Monreal
Trigo"]
__email__ = "juacomo4@etsid.upv.es"


window = tk.Tk()
window.minsize(720, 480)
window.resizable(False, False)
window.title("OSIVE HI - GUI")


'''------------------- Variables globales: --------------------'''
fieldsVal = {'entAmp1':True, 'entTOn1':True, 'entTOff1':True,
'entAmp2':True, 'entTOn2':True, 'entTOff2':True}

arduino = 0 #Variable para almacenar el puerto de la arduino micro a
la que se conectará

amp1 = tk.StringVar()
amp2 = tk.StringVar()
tOn1 = tk.StringVar()
tOn2 = tk.StringVar()
tOff1 = tk.StringVar()
tOff2 = tk.StringVar()
nPulse = tk.StringVar()
hRest = tk.StringVar()
mRest = tk.StringVar()
sRest = tk.StringVar()
nRep = tk.StringVar()

headerCom = ''


'''------------------- Métodos: --------------------'''
def allFlagsVal():
```

```
    if False in fieldsVal.values():
        btnStart.configure(state=tk.DISABLED)
    else:
        btnStart.configure(state=tk.NORMAL)

def hideLabel(label):
    label.pack_forget()

def showLabel(label):
    label.pack()

def ampVal(ampInp):
    try:
        focusedWidget = window.focus_get()
        if ampInp:
            y = float(ampInp)
            if y > 10 or y < 0:
                raise ValueError()
            else:
                hideLabel(lblAmpOutOfRange)
                fieldsVal[focusedWidget.name] = True
                allFlagsVal()
    except ValueError as ex:
        showLabel(lblAmpOutOfRange)
        fieldsVal[focusedWidget.name] = False
        allFlagsVal()
    return True

def tVal(tInp):
    try:
        focusedWidget = window.focus_get()
        if tInp:
            y = float(tInp)
            if y < 0.01:
                raise ValueError()
            else:
                hideLabel(lblTOnOutOfRange)
                fieldsVal[focusedWidget.name] = True
                allFlagsVal()
    except ValueError as ex:
        showLabel(lblTOnOutOfRange)
        fieldsVal[focusedWidget.name] = False
        allFlagsVal()
    return True

def intVal(intInp):
    try:
        focusedWidget = window.focus_get()
        if intInp:
            y = int(intInp)
            if not isinstance(y, int):
                raise ValueError()
            else:
                hideLabel(lblNotInteger)
                fieldsVal[focusedWidget.name] = True
                allFlagsVal()
    except ValueError as ex:
        showLabel(lblNotInteger)
        fieldsVal[focusedWidget.name] = False
        allFlagsVal()
    return True
```

```
def connectCom(selPort):
    strComVar = selPort.split(' ')[0]
    global arduino
    arduino = serial.Serial(strComVar, 9600, timeout=30)
    btnConnectCOM.configure(state=tk.DISABLED)
    return arduino

def connectComOnBtn():
    connectCom(lstComVar.get())

def disconnectCom(selPort):
    selPort.close()
    btnConnectCOM.configure(state=tk.NORMAL)

def disconnectComOnBtn():
    disconnectCom(arduino)

def sendValues():
    global amp1
    strAmp1 = amp1.get()
    global amp2
    strAmp2 = amp2.get()
    global tOn1
    strTOn1 = tOn1.get()
    global tOn2
    strTOn2 = tOn2.get()
    global tOff1
    strTOff1 = tOff1.get()
    global tOff2
    strTOff2 = tOff2.get()
    global nPulse
    strNPulse = nPulse.get()
    global hRest
    strHRest = hRest.get()
    global mRest
    strMRest = mRest.get()
    global sRest
    strSRest = sRest.get()
    global nRep
    strNRep = nRep.get()
    global headerCom
    headerCom = 'j'
    headerComAsciiValues = [ord(c) for c in headerCom]
    headerAsciiCheckSum = float(headerComAsciiValues[0])

    checkSumFloat =
(headerAsciiCheckSum+float(strAmp1)+float(strAmp2)+float(strTOn1)+floa
t(strTOn2)+

float(strTOff1)+float(strTOff2)+float(strNPulse)+float(strHRest)+float
(strMRest)+float(strSRest)+float(strNRep))
    checkSum = (int(checkSumFloat)%95)+32
    checkSumAscii = chr(checkSum)

    sentMessage =
(headerCom+','+strAmp1+','+strAmp2+','+strTOn1+','+strTOn2+','+

strTOff1+','+strTOff2+','+strNPulse+','+strHRest+','+strMRest+','+strS
Rest+','+strNRep+','+checkSumAscii+'\r\n')
    arduino.write(sentMessage.encode()) #Envío del mensaje. Escritura
```

```
en puerto COM

    time.sleep(1)
    msgSent()

    time.sleep(0.4)
    if arduino.inWaiting() > 0:
        print(arduino.read(arduino.inWaiting()))

def sendStopMessage():
    global amp1
    strAmp1 = '0'
    global amp2
    strAmp2 = '0'
    global tOn1
    strTOn1 = '0'
    global tOn2
    strTOn2 = '0'
    global tOff1
    strTOff1 = '0'
    global tOff2
    strTOff2 = '0'
    global nPulse
    strNPulse = '0'
    global hRest
    strHRest = '0'
    global mRest
    strMRest = '0'
    global sRest
    strSRest = '0'
    global nRep
    strNRep = '0'
    global headerCom
    headerCom = 's'
    headerComAsciiValues = [ord(c) for c in headerCom]
    headerAsciiCheckSum = float(headerComAsciiValues[0])

    checkSumFloat = (headerAsciiCheckSum + float(strAmp1) +
float(strAmp2) + float(strTOn1) + float(strTOn2) +
                     float(strTOff1) + float(strTOff2) +
float(strNPulse) + float(strHRest) + float(strMRest) + float(
                strSRest) + float(strNRep))
    checkSum = (int(checkSumFloat) % 95) + 32
    checkSumAscii = chr(checkSum)

    sentMessage = (headerCom + ',' + strAmp1 + ',' + strAmp2 + ',' +
strTOn1 + ',' + strTOn2 + ',' +
                   strTOff1 + ',' + strTOff2 + ',' + strNPulse + ',' +
strHRest + ',' + strMRest + ',' + strSRest + ',' + strNRep + ',' +
checkSumAscii + '\r\n')
    arduino.write(sentMessage.encode())
    stopMsgSent()

    time.sleep(0.4)
    if arduino.inWaiting() > 0:
        print(arduino.read(arduino.inWaiting()))

def stopMsgSent():
    messagebox.showwarning("OSIVE stopped", "A STOP message has been
sent successfully to the power stage.")
```

```
def msgSent():
    messagebox.showinfo("Message sent", "A message has been sent
successfully to the power stage.")


'''------------------- Creación de la Interfaz: -------------------'''
btnBoldFnt = tkFont.Font(weight="bold", size=10)
normalFnt = tkFont.Font(size=10)
entryFnt = tkFont.Font(size=14)
TitleFnt = tkFont.Font(size=12, weight="bold")

'''Crear el COM port selector'''
frmCOM = tk.Frame(
    master=window,
    bd=2,
    highlightbackground="blue")
frmCOM.grid(
    in_=window,
    row=0,
    column=0,
    padx=5,
    pady=(10, 0),
    sticky="ns")

# Lista de puertos COM
lstComOpt = serial.tools.list_ports.comports()
lstComVar = tk.StringVar(frmCOM)
lstComVar.set(lstComOpt[0])

lstComp = tk.OptionMenu(frmCOM, lstComVar, *lstComOpt)
lstComp.grid(
    in_=frmCOM,
    row=0,
    column=0,
    pady=(15, 0),
    columnspan=2)

# Botón de conexión COM
btnConnectCOM = tk.Button(
    master=frmCOM,
    text="CONNECT",
    font=btnBoldFnt,
    fg="#62A28D",
    activeforeground="#62A28D",
    width=11,
    height=1,
    cursor="hand1",
    relief="groove",
    pady=2,
    command=connectComOnBtn
)
btnConnectCOM.grid(
    in_=frmCOM,
    row=1,
    column=0,
    padx=3,
    pady=10)

# Botón de desconexión COM
btnDisconnectCOM = tk.Button(
    master=frmCOM,
```

```
    text="DISCONNECT",
    font=btnBoldFnt,
    fg="#EE6B72",
    activeforeground="#EE6B72",
    width=11,
    height=1,
    cursor="hand1",
    relief="groove",
    pady=2,
    command=disconnectComOnBtn
)
btnDisconnectCOM.grid(
    in_=frmCOM,
    row=1,
    column=1,
    padx=3,
    pady=10)


'''Crear el apartado N° de pulsos'''
frmPulse = tk.Frame(
    master=window
)
frmPulse.grid(
    in_=window,
    row=0,
    column=2,
    padx=5,
    pady=(10, 0),
    sticky="nsew"
)

lblPulseNum = tk.Label(
    master=frmPulse,
    text="Number of pulses:",
    font=TitleFnt)
lblPulseNum.pack(pady=(15, 5))
entPulseNum = tk.Entry(
    master=frmPulse,
    insertwidth=1,
    justify="center",
    font=entryFnt,
    textvariable=nPulse
)
entPulseNum.name="entPulseNum"
entPulseNum.pack()


'''Crear la gráfica de la señal'''
frmGraph = tk.Frame(
    master=window
)
frmGraph.grid(in_=window, row=0, column=1, pady=(10, 0))

cnvGraph = tk.Canvas(
    master=frmGraph,
    width=270,
    height=180)
cnvGraph.pack(pady=0)
# Líneas principales
cnvGraph.create_line(50, 50, 100, 50, fill="black", width=3,
```

```
                    joinstyle="miter", smooth=True)
cnvGraph.create_line(100, 50, 100, 150, fill="black", width=3,
                    joinstyle="miter", smooth=True)
cnvGraph.create_line(100, 150, 230, 150, fill="black",
                    width=3, joinstyle="miter", smooth=True)
# Líneas de medidas
cnvGraph.create_line(50, 40, 100, 40, arrow="both", arrowshape="2 3
3")
cnvGraph.create_line(103, 140, 230, 140, arrow="both", arrowshape="2 3
3")
cnvGraph.create_line(45, 150, 45, 50, arrow="both", arrowshape="2 3
3")
# Textos
cnvGraph.create_text(75, 30, text="T_On")
cnvGraph.create_text(167, 130, text="T_Off")
cnvGraph.create_text(35, 100, text="Irradiance", angle=90)

'''Crear los probes'''
frmMasterProbes = tk.Frame(
    master=window)
frmMasterProbes.grid(
    in_=window,
    row=1,
    column=0,
    columnspan=2,
    padx=(5, 0),
    sticky="ewsn"
    )

frmProbe1 = tk.Frame(
    master=frmMasterProbes,
    borderwidth=1,
)
frmProbe1.grid(in_=frmMasterProbes, row=0, column=0,
               padx=10, ipadx=10, ipady=10, sticky="nsw")

lblProbeHeader1 = tk.Label(
    master=frmProbe1,
    text="Probe 1",
    font=TitleFnt)
lblProbeHeader1.pack(pady=(5, 2), fill="x")

frmCurrent1 = tk.Frame(master=frmProbe1)
frmCurrent1.pack()
lblAmpHeader1 = tk.Label(
    master=frmCurrent1,
    text="Irradiance (mW/mm2):",
    font=normalFnt)
lblAmpHeader1.pack(pady=(5, 0), fill="x")
entAmp1 = tk.Entry(
    master=frmCurrent1,
    width=10,
    insertwidth=1,
    justify="center",
    font=entryFnt,
    textvariable=amp1
)
entAmp1.name = "entAmp1"
entAmp1.pack(fill="x")
```

```
frmTOn1 = tk.Frame(master=frmProbe1)
frmTOn1.pack()
lblTOn1 = tk.Label(master=frmTOn1, text="T_On (ms):", font=normalFnt)
lblTOn1.pack(pady=(5, 0), fill="x")
entTOn1 = tk.Entry(
    master=frmTOn1,
    width=10,
    insertwidth=1,
    justify="center",
    font=entryFnt,
    textvariable=tOn1
)
entTOn1.name = "entTOn1"
entTOn1.pack(fill="x")

frmTOff1 = tk.Frame(master=frmProbe1)
frmTOff1.pack()
lblTOff1 = tk.Label(master=frmTOff1, text="T_Off (ms):",
font=normalFnt)
lblTOff1.pack(pady=(5, 0), fill="x")
entTOff1 = tk.Entry(
    master=frmTOff1,
    width=10,
    insertwidth=1,
    justify="center",
    font=entryFnt,
    textvariable=tOff1)
entTOff1.name = "entTOff1"
entTOff1.pack(fill="x")

frmProbe2 = tk.Frame(
    master=frmMasterProbes,
    borderwidth=1
)
frmProbe2.grid(in_=frmMasterProbes, row=0, column=1,
               padx=10, ipadx=10, ipady=10, sticky="nsw")

lblProbeHeader2 = tk.Label(
    master=frmProbe2,
    text="Probe 2",
    font=TitleFnt)
lblProbeHeader2.pack(pady=(5, 2), fill="x")

frmCurrent2 = tk.Frame(master=frmProbe2)
frmCurrent2.pack()
lblAmpHeader2 = tk.Label(
    master=frmCurrent2,
    text="Irradiance (mW/mm2):",
    font=normalFnt)
lblAmpHeader2.pack(pady=(5, 0), fill="x")
entAmp2 = tk.Entry(
    master=frmCurrent2,
    width=10,
    insertwidth=1,
    justify="center",
    font=entryFnt,
    textvariable=amp2
)
entAmp2.name = "entAmp2"
entAmp2.pack(fill="x")
```

```
frmTOn2 = tk.Frame(master=frmProbe2)
frmTOn2.pack()
lblTOn2 = tk.Label(master=frmTOn2, text="T_On (ms):", font=normalFnt)
lblTOn2.pack(pady=(5, 0), fill="x")
entTOn2 = tk.Entry(
    master=frmTOn2,
    width=10,
    insertwidth=1,
    justify="center",
    font=entryFnt,
    textvariable=tOn2
)
entTOn2.name="entTOn2"
entTOn2.pack(fill="x")

frmTOff2 = tk.Frame(master=frmProbe2)
frmTOff2.pack()
lblTOff2 = tk.Label(master=frmTOff2, text="T_Off (ms):",
font=normalFnt)
lblTOff2.pack(pady=(5, 0), fill="x")
entTOff2 = tk.Entry(
    master=frmTOff2,
    width=10,
    insertwidth=1,
    justify="center",
    font=entryFnt,
    textvariable=tOff2)
entTOff2.name="entTOff2"
entTOff2.pack(fill="x")



'''Crear el apartado para tiempos de reposo y número de
repeticiones'''
frmReposo = tk.Frame(master=window)
frmReposo.grid(
    in_=window,
    row=1,
    column=2,
    padx=5,
    sticky="ns")

lblTReposo = tk.Label(master=frmReposo, text="Rest time:",
font=TitleFnt)
lblTReposo.grid(in_=frmReposo, row=0, column=0, columnspan=3, pady=5,
sticky="ew")

lblHReposo = tk.Label(master=frmReposo, text="Hours:", font=normalFnt)
lblHReposo.grid(in_=frmReposo, row=1, column=0, sticky="ew")
entHorasReposo = tk.Entry(
    master=frmReposo,
    width=10,
    insertwidth=1,
    justify="center",
    font=entryFnt,
    textvariable=hRest)
entHorasReposo.name="entHorasReposo"
entHorasReposo.grid(in_=frmReposo, row=2, column=0, sticky="w")


lblMinReposo = tk.Label(master=frmReposo, text="Minutes:",
```

```
font=normalFnt)
lblMinReposo.grid(in_=frmReposo, row=1, column=1, sticky="ew")
entMinReposo = tk.Entry(
    master=frmReposo,
    width=10,
    insertwidth=1,
    justify="center",
    font=entryFnt,
    textvariable=mRest
)

entMinReposo.name= "entMinutosReposo"
entMinReposo.grid(in_=frmReposo, row=2, column=1)


lblSegReposo = tk.Label(master=frmReposo, text="Seconds:",
font=normalFnt)
lblSegReposo.grid(in_=frmReposo, row=1, column=2, sticky="ew")
entSegReposo = tk.Entry(
    master=frmReposo,
    width=10,
    insertwidth=1,
    justify="center",
    font=entryFnt,
    textvariable=sRest
)
entSegReposo.name="entSegReposo"
entSegReposo.grid(in_=frmReposo, row=2, column=2, sticky="e")



lblRepNum = tk.Label(master=frmReposo, text="Number of repetitions:",
font=TitleFnt)
lblRepNum.grid(in_=frmReposo, row=3, column=0, columnspan=3, pady=(40,
0), sticky="ew")
entRepNum = tk.Entry(
    master=frmReposo,
    width=10,
    insertwidth=1,
    justify="center",
    font=entryFnt,
    textvariable=nRep
)
entRepNum.name="entRepNum"
entRepNum.grid(in_=frmReposo, row=4, column=1, sticky="ew")


'''Validar lo introducido en las Entries'''
regAmp1 = frmCurrent1.register(ampVal)
entAmp1.config(validate="key", validatecommand=(regAmp1, '%P'))

regAmp2 = frmCurrent2.register(ampVal)
entAmp2.config(validate="key", validatecommand=(regAmp2, '%P'))

regTOn1 = frmTOn1.register(tVal)
entTOn1.config(validate="key", validatecommand=(regTOn1, '%P'))

regTOn2 = frmTOn2.register(tVal)
entTOn2.config(validate="key", validatecommand=(regTOn2, '%P'))

regTOff1 = frmTOff1.register(tVal)
```

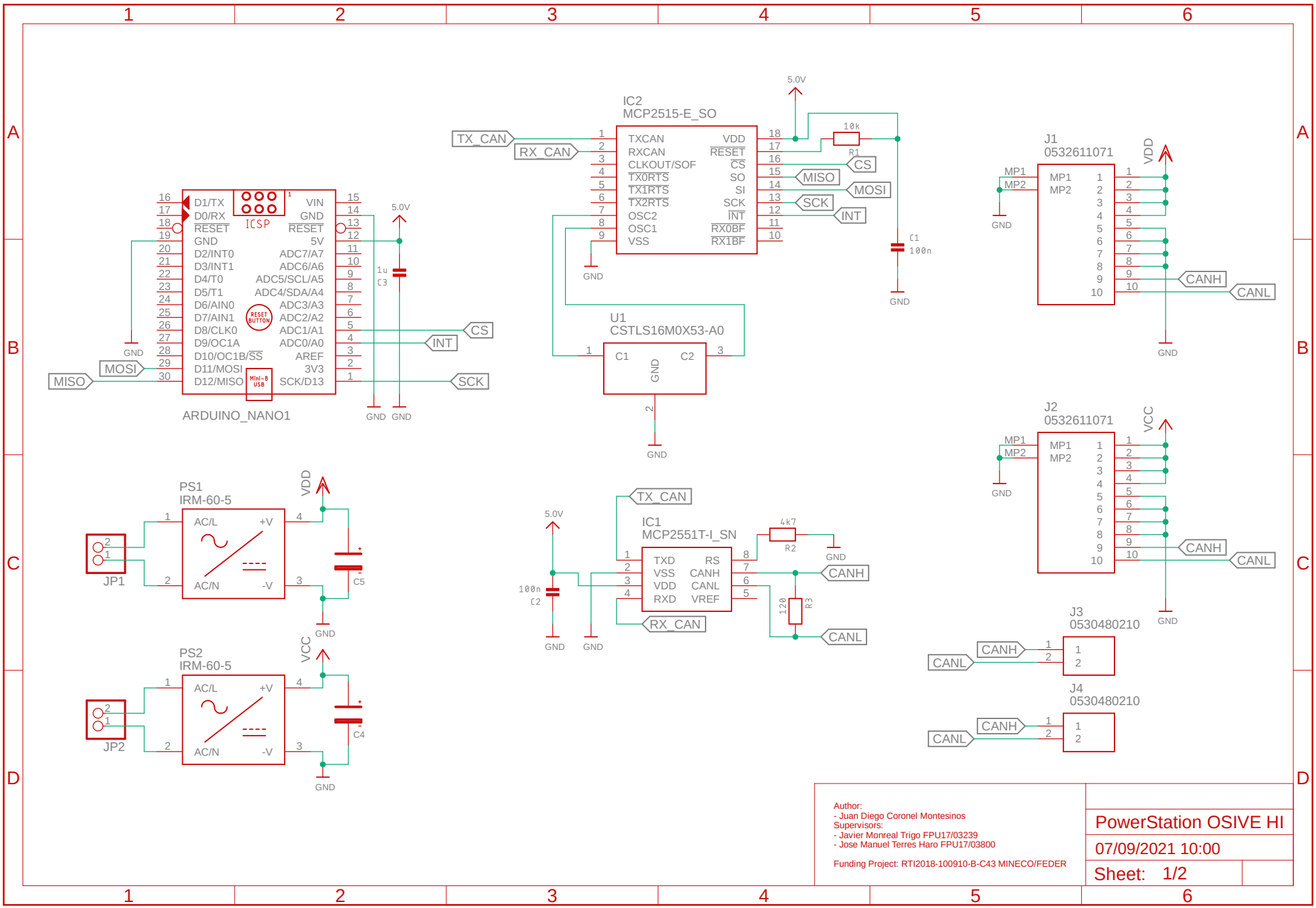
```
entTOff1.config(validate="key", validatecommand=(regTOff1, '%P'))

regTOff2 = frmTOff2.register(tVal)
entTOff2.config(validate="key", validatecommand=(regTOff2, '%P'))

regPulseNum = frmPulse.register(intVal)
entPulseNum.config(validate="key", validatecommand=(regPulseNum,
'%P'))

regHReposo = frmReposo.register(intVal)
entHorasReposo.config(validate="key", validatecommand=(regHReposo,
'%P'))

regMinReposo = frmReposo.register(intVal)
entMinReposo.config(validate="key", validatecommand=(regMinReposo,
'%P'))

regSegReposo = frmReposo.register(intVal)
entSegReposo.config(validate="key", validatecommand=(regSegReposo,
'%P'))

regRepNum = frmReposo.register(intVal)
entRepNum.config(validate="key", validatecommand=(regRepNum, '%P'))

'''Crear botones de START y STOP'''
frmMainBtns = tk.Frame(master=window)
frmMainBtns.grid(
    in_=window,
    row=2,
    column=0,
    columnspan=2,
    padx=5,
    pady=(0, 10),
    sticky="ew")

btnStart = tk.Button(
    master=frmMainBtns,
    text="START",
    font=btnBoldFnt,
    width=10,
    height=2,
    cursor="hand1",
    relief="groove",
    fg="#153262",
    activeforeground="#153262",
    disabledforeground="grey",
    bg="#CEE1FF",
    activebackground="#CEE1FF",
    command=sendValues
)
btnStart.grid(
    in_=frmMainBtns,
    row=0,
    column=0,
    padx=10,
    pady=10)


btnStop = tk.Button(
    master=frmMainBtns,
    text="STOP",
```

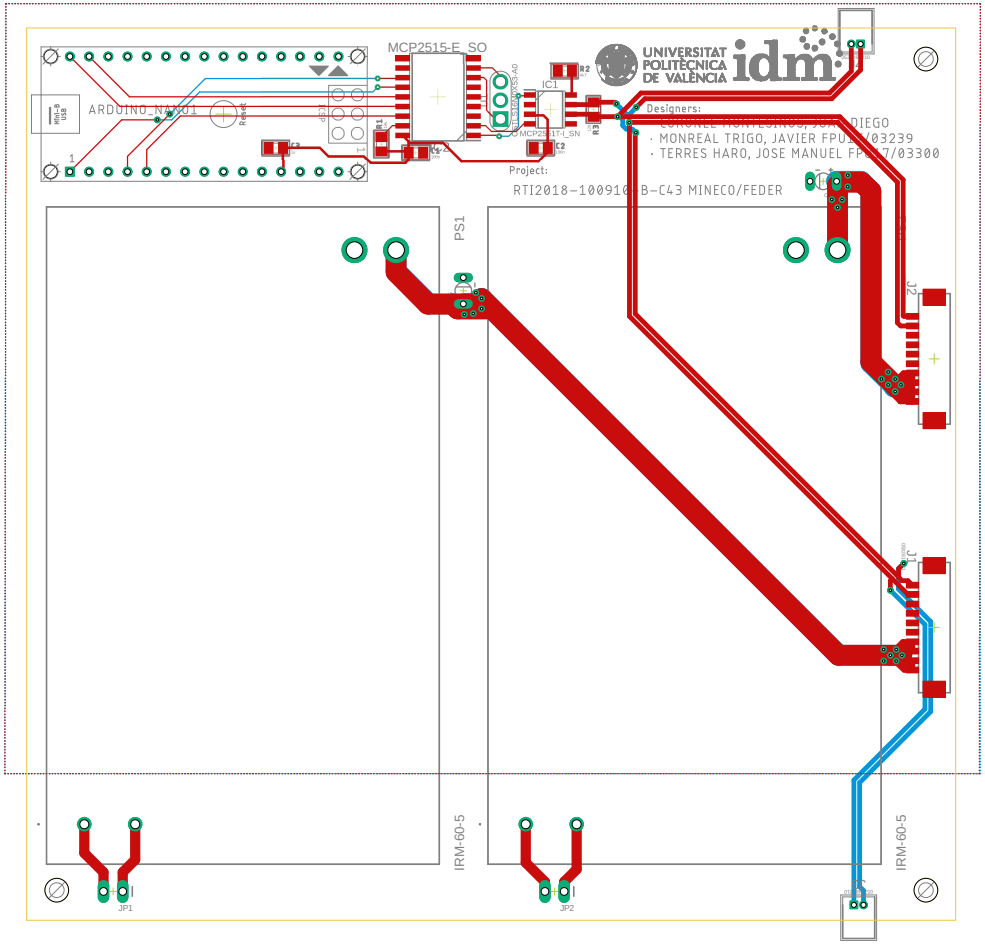
```
    font=btnBoldFnt,
    width=10,
    height=2,
    cursor="hand1",
    relief="groove",
    fg="#9C252C",
    activeforeground="#9C252C",
    bg="#FFD2D5",
    activebackground="#FFD2D5",
    command=sendStopMessage
)
btnStop.grid(
    in_=frmMainBtns,
    row=0,
    column=1,
    padx=10,
    pady=10)

lblAmpUnitInfo = tk.Label(
    master=frmMainBtns,
    text="Irradiance is measured in mW/mm^2:",
    fg="red")
lblAmpUnitInfo.grid(in_=frmMainBtns, row=0, column=3)

# Labels de advertencia/información
frmWarnLabels = tk.Label(master=window)
frmWarnLabels.grid(
    in_=window,
    row=2,
    column=2,
    padx=(0, 5),
    pady=(0, 10),
    sticky="ew")

lblAmpOutOfRange = tk.Label(
    master=frmWarnLabels,
    text="Only integers and decimal numbers from 0 to 10 are
accepted",
    fg="#9C252C",
    bg="#FFD2D5"
)

lblTOnOutOfRange = tk.Label(
    master=frmWarnLabels,
    text="Only integers and decimal numbers greater than 0.01 are
accepted",
    fg="#9C252C",
    bg="#FFD2D5"
)

lblNotInteger = tk.Label(
    master=frmWarnLabels,
    text="Only integers are accepted",
    fg="#9C252C",
    bg="#FFD2D5"
)

window.mainloop()
```

II – PLANOS

Durante el presente apartado se muestran los planos referentes a:

- Esquemáticos de la unidad central.
- Diseño de la PCB de la unidad central.
- Encapsulado de OSIVE HI.


ARDUINO_NANO1
D1/TX
D0/RX
RESET
GND
D2/INT0
D3/INT1
D4/T0
D5/T1
D6/AIN0
D7/AIN1
D8/CLK0
D9/OC1A
D10/OC1B/SS
D11/MOSI
D12/MISO
VIN
GND
RESET
5V
ADC7/A7
ADC6/A6
ADC5/SCL/A5
ADC4/SDA/A4
ADC3/A3
ADC2/A2
ADC1/A1
ADC0/A0
AREF
3V3
SCK/D13
ICSP
RESET BUTTON
Mini-B USB
5.0V
1u
C3
CS
INT
SCK
MISO
MOSI
GND
IC2
MCP2515-E_SO
TXCAN
RXCAN
CLKOUT/SOF
TX0RTS
TX1RTS
TX2RTS
OSC2
OSC1
VSS
VDD
RESET
CS
SO
SI
SCK
INT
RX0BF
RX1BF
TX_CAN
RX_CAN
10k
R1
C1
100n
U1
CSTLS16M0X53-A0
C1
C2
GND
IC1
MCP2551T-I_SN
TXD
VSS
VDD
RXD
RS
CANH
CANL
VREF
4k7
R2
120
R3
C2
100n
J1
0532611071
MP1
MP2
CANH
CANL
VDD
J2
0532611071
VCC
J3
0530480210
J4
0530480210
PS1
IRM-60-5
PS2
IRM-60-5
AC/L
AC/N
+V
-V
JP1
JP2
C5
C4
Author:
- Juan Diego Coronel Montesinos
Supervisors:
- Javier Monreal Trigo FPU17/03239
- Jose Manuel Terres Haro FPU17/03800
Funding Project: RTI2018-100910-B-C43 MINECO/FEDER
PowerStation OSIVE HI
07/09/2021 10:00
Sheet: 1/2



Author:
- Juan Diego Coronel Montesinos
Supervisors:
- Javier MonrealTrigo FPU17/03239
- Jose ManuelTerres Haro FPU17/03800

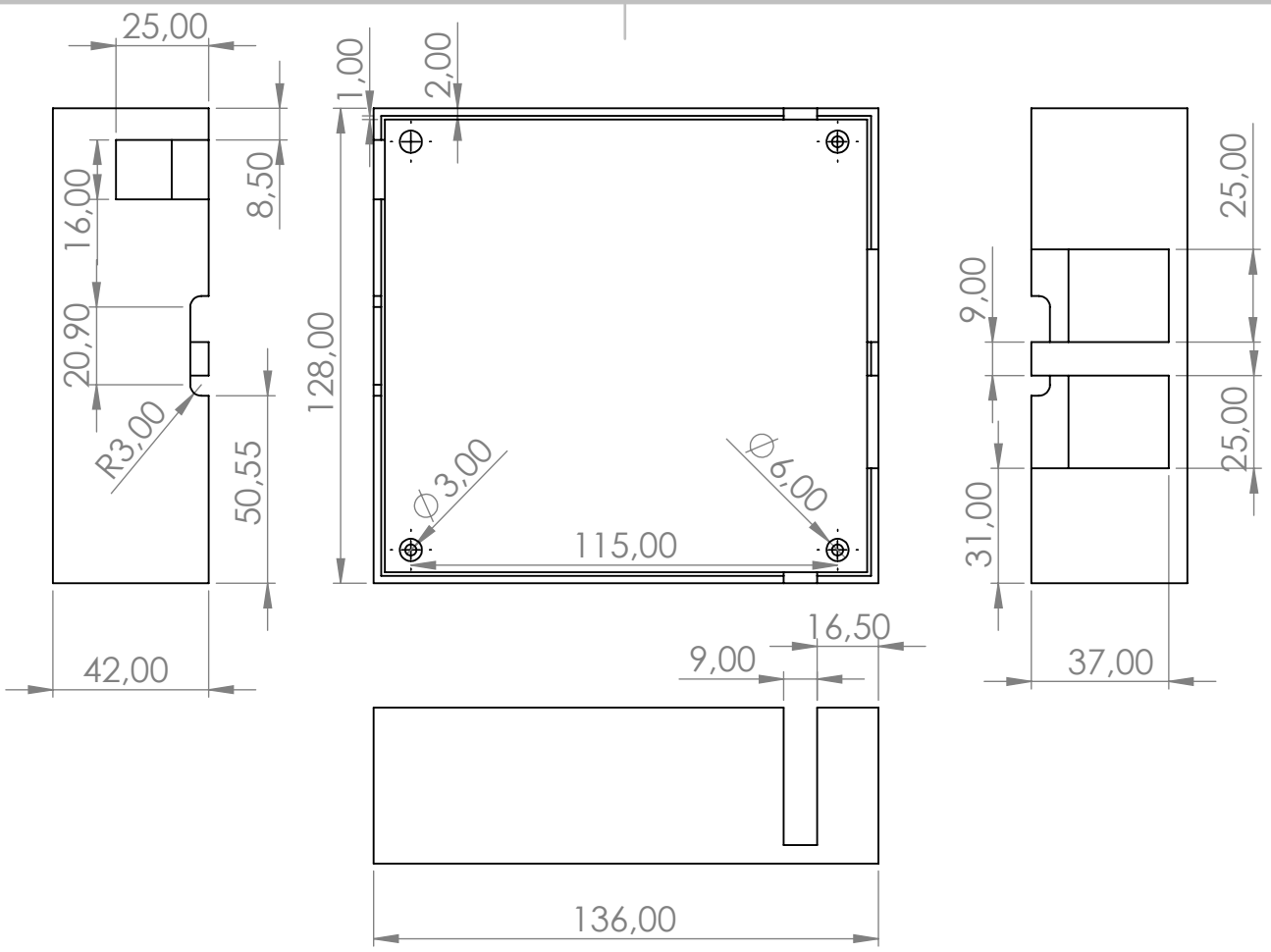
Funding Project: RTI2018-100910-B-C43 MINECO/FEDER

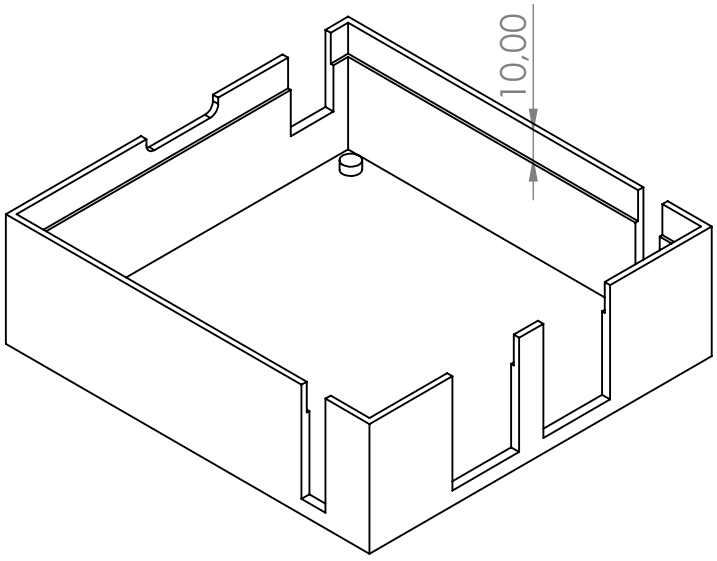
Scale: 1:1

PowerStation OSIVE HI

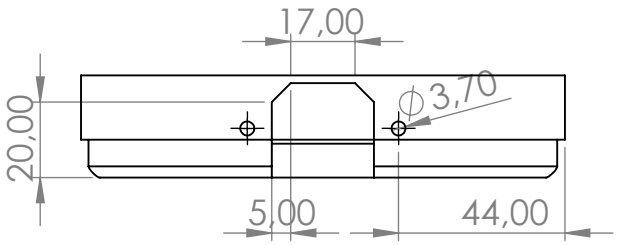
07/09/2021 10:00

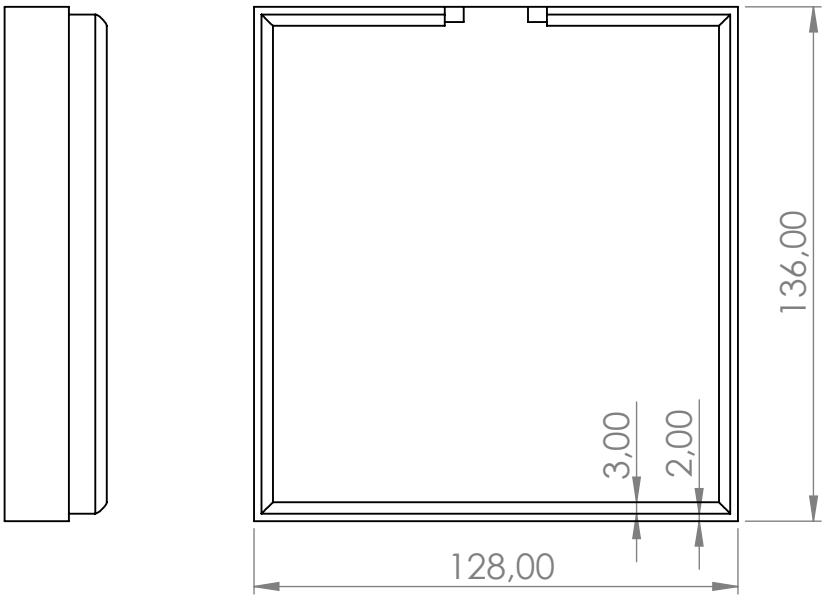
Sheet: 2/2

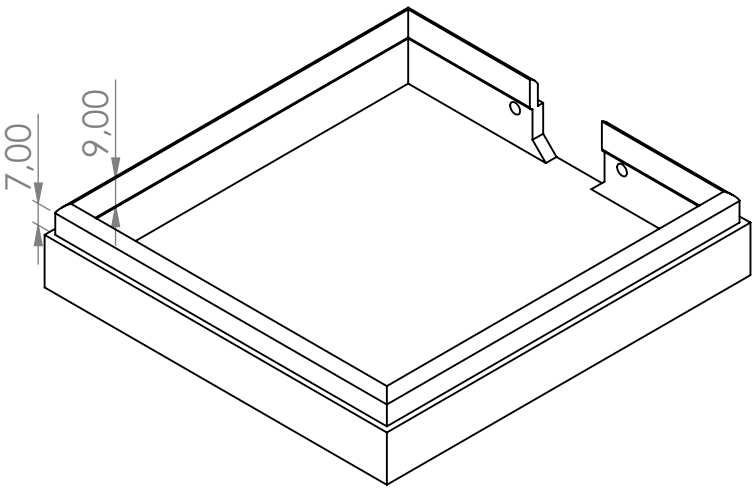




SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:		ACABADO:		REBARBAR Y ROMPER ARISTAS VIVAS		NO CAMBIE LA ESCALA	REVISIÓN
	NOMBRE	FIRMA	FECHA			TÍTULO:	
DIBUJ.	Juan Diego Coronel Montesinos		07/09/2021			Diseño, desarrollo y validación de la etapa de potencia de un sistema de estimulación optoquimiogenética de alta intensidad - Plano de encapsulado	
VERIF.	Javier Monreal Trigo		07/09/2021				
APROB.							
FABR.							
CALID.				MATERIAL: PLA		N.º DE DIBUJO Box	A4
				PESO: 94 gr		ESCALA:1:5	HOJA 1 DE 1







SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:		ACABADO:		REBARBAR Y ROMPER ARISTAS VIVAS		NO CAMBIE LA ESCALA	REVISIÓN
	NOMBRE	FIRMA	FECHA			TÍTULO:	
DIBUJ.	Juan Diego Coronel Montesinos		07/09/2021			Diseño, desarrollo y validación de la etapa de potencia de un sistema de estimulación optoquimiogenética de alta intensidad - Plano de encapsulado	
VERIF.	Javier Monreal Trigo		07/09/2021				
APROB.							
FABR.							
CALID.				MATERIAL: PLA		N.º DE DIBUJO Enclosure	A4
				PESO: 94 gr		ESCALA:1:5	HOJA 1 DE 1

III – PLIEGO DE CONDICIONES

El presente documento tiene como finalidad regular los niveles ténicos y de calidad exigibles, precisando las intervenciones que correspondan de acuerdo al contrato y la legislación aplicable, tanto a la parte contratante como a la parte técnica, así como las relaciones entre ellos y las obligaciones correspondientes en orden al cumplimiento del mismo.

Los procesos y montajes accesorios, entendiendo por este nombre los que no pueden ser previstos con todo detalle, se realizarán conforme aparezca necesidad. Cuando la propia importancia lo exija, se realizarán proyectos adicionales que los definan. En casos de menor importancia, entendiéndose por estos los que suponen menos de un 10% del presupuesto del proyecto, se seguirán las directrices de la parte técnica dentro del presente proyecto.

Se considera parte contratante: al equipo de investigación especializado en optoquimiogenética del *Centro de Investigación Príncipe Felipe,* bajo la responsabilidad del director actual VICTORIA MORENO MANZANO, tal y como se considera en el contrato de empresa, con la obligación de antes de empezar el proyecto e inmediatamente después de de recibir los documentos, verificarlos y ante cualquier discrepancia, contradicción u omisión, hacerlo saber a la parte técnica; además de facilitar y proporcionar todo el material necesario para el desarrollo del proyecto, bien en forma física o mediante el pago de las facturas con un plazo máximo de cinco días laborables, tal y como establece el contrato legal.

Se considera parte técnica: a JUAN DIEGO CORONEL MONTESINOS, tal y como se considera en el contrato de empresa, con las responsabilidades facultativas a la vista del proyecto y el contrato de planificar y establecer las condiciones técnicas, normativa a cumplir, control de calidad y control económico, realizar la planificación temporal, seleccionar los materiales más idonios, desarrollar, ensamblar y comprobar el sistema y redactar los documentos pertinentes.

Integran el contrato los siguientes documentos relacionados por orden de prelación en cuanto a valor de sus especificaciones en caso de omisión o aparente contradicción:

Ante cualquier voluntad de modificación del proyecto por parte de la parte contratante, la propuesta hará falta que se realice a la parte técnica con el tiempo suficiente para realizar la modificación si se aceptara, realizándose la modificación del presente proyecto de mutuo acuerdo.

Ante la voluntad de paro del proyecto por parte de la parte contratante, esta se podrá realizar abonando la inversión en personal correspondiente a 40 días laborales, sin posibilidad de reclamación de los gastos materiales ocasionados.

La fecha límite del proyecto se establece el día 10 de septiembre de 2021, fecha en la cual toda documentación actualizada del proyecto: memoria, planos, presupuestos y pliego de condiciones debe estar entregada a la parte contratante, junto con el prototipo y el software desarrollado. La parte contratante, por esta fecha, debe haber entregado la cuantía económica pactada en el contrato legal.

El incumplimiento de obligaciones supondrá la paralización del proyecto, sin derecho a reclamación alguna y en detrimento económico y de otro cariz de la parte incumplidora, hasta que se solucione la deficiencia. En caso de no ser posible, se realizará una compensación económica.

CONDICIONES TÉCNICAS

Se definen las condiciones técnicas como la serie de puntos que el sistema a desarrollar debe cumplir. El sistema a desarrollar, objetivo del presente proyecto, es una etapa de potencia para aplicaciones optoquimiogenéticas que proporcione alimentación a determinados punteros lumínicos objeto de las estimulaciones necesarias, además de albergar el propio dispositivo una unidad central que permita la configuración del patrón de estímulo producido por dichos punteros. Las siguientes características forman parte del sistema:

- Compacto: volumen adecuado a puesto de laboratorio.
- Ligero: no llegando al kilogramo de peso.
- Alimentación proporcionada de 5 V y 20 A.
- GUI intuitiva y versátil, que permita ser utilizada sin la necesidad de conocimientos informáticos avanzados.
- Modular: sistema con posibilidades de expansión.
- Utilizable bajo distintos sistemas operativos sin modificaciones de software.
- Sistema de fácil reproducción, con un coste económico y materiales inferiores a los 100€.
- Accesibilidad total al hardware y al software.

Ante la condición de expansión, se plantea como posibilidad el desarrollo de un sistema esclavo, con menos componentes, por tanto menor coste, y que sirva para la tarea señalada, con el cual se cumplirá a tiempo en caso de ser posible, no estableciéndose como condición debido a las limitaciones temporales del proyecto.

MATERIALES A UTILIZAR EN EL PROCESO Y NORMATIVAS

En cuanto a las normativas a cumplir:

- Los elementos electrónicos han de cumplir con la normativa de emisiones electromagnéticas como clase 1 ANSI/IPV-D-275.
- Es necesario que los plásticos empleados cumplan con la normativa para ser empleados: ISO 10993 y USP clase VI.

En cuanto a los materiales a utilizar en el proceso, estos son listados a continuación:

- Componentes electrónicos pasivos.
- Circuitos electrónicos integrados u otros componentes que doten de la capacidad de establecer comunicaciones para el envío y el recibimiento de información.
- Circuitos electrónicos integrados u otros componentes que doten de la capacidad de otorgar una señal de corriente continua, que se usará para la alimentación de diversos componentes.
- Herramientas necesarias para tareas de fabricación, soldado y ensamble de los componentes.
- Herramientas software necesarias para el desarrollo del sistema y su documentación del sistema.

VERIFICACIONES A REALIZAR

Las comprobaciones que se señalan a continuación se realizarán de forma secuencial.

- Comprobación de la fuente de alimentación: Verificar que se entrega la señal de alimentación requerida y comprobar que no se dan problemas de sobrecalentamiento en el módulo.
- Comprobación de la interfaz gráfica: Verificar que el usuario pueda completar un uso, desde que abre hasta que cierra la interfaz, ejecutando las acciones principales.
- Comprobación de las comunicaciones: Hacer transparentes las comunicaciones entre la interfaz gráfica que envía la señal de configuración para el patrón lumínico, y OSIVE HI, con el objetivo de verificar el correcto envío y recibimiento de tramas entre los dos sistemas.

IV – Presupuesto y estudio económico

Durante el presente punto se describe el estudio económico del proyecto, tanto para la parte de diseño, como desarrollo y prototipaje del mismo.

PRECIO DEL DESARROLLO DEL PROTOTIPO

Para el desarrollo del prototipo, se han subdividido los gastos en dos apartados: los honorarios del proyectista y la ejecución material.

Honorarios del proyectista

Se estiman las horas invertidas en el proyecto según su función, y con el precio por hora correspondiente, en la siguiente tabla, dando como resultado unos horarios de 2280,00 €.

INVERSIÓN EN PERSONAL EN EL DESARROLLO			
CONCEPTE	**HORES**	**PRECIO(€)/HORA**	**IMPORTE**
Análisis y estudio del proyecto	15	30	450,00 €
Implementación del código	70	20	1400,00 €
Diseño de estructuras 3D	7	20	140,00 €
Soldado/Ensamblaje de sistema	4	15	60,00 €
Validación	10	25	250,00 €
		TOTAL	**2300,00 €**

Presupuesto de ejecución material

Para presupuestar los costes de ejecución material del prototipo, hay que tener en cuenta los gastos relacionados con amortización de equipos y licencias, las cuales se muestran en la siguiente tabla y ascienden a un total de 1550,00 €.

GASTOS EN MATERIALES PARA EL DESARROLLO	
CONCEPTO	**IMPORTE**
Amortización de licencia EAGLE	1000,00 €
Amortización de licencia SolidWorks	300,00 €
Amortizacoción de PC para desarrollo	150 €
Amortización impresora 3D	100 €
TOTAL	**1550,00 €**

Por otro lado, respecto a la ejecución material para el prototipo, se adjunta la siguiente tabla:

PRESUPUESTO DE EJECUCIÓN MATERIAL (1 u.)			
CONCEPTO	**CANTIDAD**	**PRECIO UNITARIO**	**IMPORTE**
Fuente de alimentación encapsulada	2	15,61 €	31,22 €
Conector macho IEC C14	1	3,70 €	3,70 €
Cable de alimentación IEC C13	1	12.13 €	12,13 €
Transceptor CAN	1	0,945 €	0,945 €
Conector macho PCB 10 pines	2	0,474 €	0,948 €
Controlador CAN	1	1,93 €	1,93 €
Latiguillo de cable a placa	1	5,50 €	5,50 €
Resonador cerámico 16 MHz	1	0,305 €	0,305 €
Gastos externos de construcción	1	38,16 €	38,16 €
TOTAL			**94,84 €**

Obteniéndose un total de 94,84 € como precio para el prototipo.

A continuación se presenta una tabla resumen para el precio de una tirada de 100 unidades:

PRESUPUESTO DE EJECUCIÓN MATERIAL (100 u.)			
CONCEPTO	**CANTIDAD**	**PRECIO UNITARIO**	**IMPORTE**
Fuente de alimentación encapsulada	200	12,72 €	2544 €
Conector macho IEC C14	100	3,33 €	333,00 €
Cable de alimentación IEC C13	100	10,67 €	1067,00 €
Transceptor CAN	100	0,85 €	85,00 €
Conector macho PCB 10 pines	200	0,35 €	70,00 €
Controlador CAN	100	1,73 €	173,00 €
Latiguillo de cable a placa	100	4,40 €	440,00 €
Resonador cerámico 16 MHz	100	0,23 €	23,00 €
Gastos externos de construcción	100	20,10 €	2010,00 €
TOTAL		**54,38 €**	**6745,00 €**

Gastos generales, beneficio industrial e impuestos (por PVP)

Finalmente, se presenta el precio unitario de venta al público para 100 unidades:

PRESUPUESTO UNITARIO DE VENTA AL PÚBLICO PARA 100 UNIDADES	
CONCEPTO	**IMPORTE**
Ejecución material unitario	54,38 € €
Inversión en personal	230 €
Gastos en materiales para el desarrollo	155,00 €
Beneficio industrial	81,54 €
IVA (21%)	117,85 €
TOTAL	**638,77 €**