UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

DISEÑO DE UNA MICROFACTORÍA DE BEBIDAS ALCOHÓLICAS FERMENTADAS

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Girona Anteportamlatinam, Javier

Tutor/a: Tormos Ferrando, Álvaro

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

DISEÑO DE UNA MICROFACTORÍA DE BEBIDAS ALCOHÓLICAS FERMENTADAS

Trabajo Fin de Grado
Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Girona Anteportamlatinam, Javier
Tutor/a: Tormos Ferrando, Álvaro
CURSO ACADÉMICO: 2023/2024

RESUMEN

El siguiente Trabajo de Fin de Grado (TFG) presenta el diseño e implementación de una microfactoría casera semiautomatizada para la elaboración de bebidas alcohólicas fermentadas. A lo largo del documento se exploran los fundamentos de la elaboración de cerveza, incluyendo ingredientes, tipos y procesos clave. Se analizan los sistemas comerciales disponibles para la elaboración de pequeños lotes tanto a nivel casero como a nivel prototipado, justificando la elección de un diseño "todo en uno" por su flexibilidad, adaptabilidad y versatilidad. Se detalla y argumenta la selección de componentes, el desarrollo de una interfaz gráfica de usuario y la implementación del control automatizado mediante Arduino. Se discuten posibles mejoras, como la integración de sensores adicionales y comunicación inalámbrica. El TFG concluye validando la viabilidad de una micro factoría casera automatizada, accesible y personalizable, que fomenta la colaboración entre comunidades y abre nuevas posibilidades para la innovación en la cerveza artesanal.

ABSTRACT

The following Final Degree Project (FDP) presents the design and implementation of an semiautomated home microbrewery for the production of fermented alcoholic beverages. Throughout the document, the fundamentals of beer brewing are explored, including ingredients, types, and key processes. The available commercial systems for small-batch brewing, both at home and prototype levels, are analyzed, justifying the choice of an "all-in-one" design for its flexibility, adaptability, and versatility. The selection of components is detailed and justified, as well as the development of a graphical user interface and the implementation of automated control using Arduino. Possible improvements are discussed, such as the integration of additional sensors and wireless communication. The FDP concludes by validating the feasibility of an automated, accessible, and customizable home microbrewery, which fosters collaboration between communities and opens new possibilities for innovation in craft beer.

RESUM

El següent Treball de Fi de Grau (TFG) presenta el disseny i implementació d'una microfàbrica casolana semiautomatitzada per a l'elaboració de begudes alcohòliques fermentades. Al llarg del document s'exploren els fonaments de l'elaboració de cervesa, incloent-hi ingredients, tipus i processos clau. S'analitzen els sistemes comercials disponibles per a l'elaboració de lots xicotets tant a escala casolana com en l'àmbit de prototipat, justificant l'elecció d'un disseny "tot en un" per la seua flexibilitat, adaptabilitat i versatilitat. Es detalla i argumenta la selecció de components, el desenvolupament d'una interfície gràfica d'usuari i la implementació del control automatitzat mitjançant Arduino.

Es discutixen possibles millores, com la integració de sensors addicionals i comunicació sense fils. El TFG conclou validant la viabilitat d'una microfàbrica casolana automatitzada, accessible i personalitzable, que fomenta la col·laboració entre comunitats i obri noves possibilitats per a la innovació en la cervesa artesana.

PALABRAS CLAVE

Electrónica, Arduino, IoT, Cerveza, Makers, Brewers, PID

KEYWORDS

Electronics, Arduino, IoT, Beer, Makers, Brewers, PID

PARAULES CLAU

Electrònica, Arduino, IoT, Cervesa, Makers, Brewers, PID

AGRADECIMIENTOS

Quiero agradecer primeramente a mis compañeros de universidad que me ayudaron mucho más de lo que creen y sin ellos no hubiese podido acabar, *¡Gracias chiquis!*.
Quiero agradecer también a mis compañeros de departamento que asumieron parte de mi carga de trabajo para que pudiese asistir a clase. Especialmente a mi antiguo jefe Mariano que siempre creyó en mí y me ayudó a la transición de fábrica a oficina y muy especialmente a mi compañero y amigo Carlos, que me ha ayudado y dado ánimos en momentos buenos y en momentos malos, llamando casi a diario para que le diese actualizaciones, ahora me toca a mí para que entregues el proyecto del master.
Finalmente y no por ello menos importante quiero agradecer también a mis progenitores, que con sus consejos (buenos o malos) he acabado donde estoy y decirles "*Charito ya he acabado la carrera, ¿ya estás contenta?"*

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. OBJETIVOS

El objetivo de este proyecto es juntar lo mejor de los mundos “maker” y “brewer” mediante el diseño de una microfactoría casera para la realización de bebidas fermentadas. La microfactoría ha de poder adaptarse a casi cualquier presupuesto y ser mejorable por cualquier persona. Ha de garantizar un mínimo de repetibilidad del proceso mediante un cierto grado de automatización, especialmente en el control de temperaturas y tiempos, aunque también se mantendrá una opción manual, especialmente para aquellos usuarios más “castizos” o artesanales que se guían más por su instinto que por una determinada receta.

Durante el proceso de diseño, se irán valorando diferentes alternativas, así como mejoras a la opción elegida y, si es posible, opciones todavía más básicas para aquellos que deseen realizar una inversión más contenida.

Finalmente se proporcionarán todas las instrucciones necesarias para el ensamblaje de la microfactoría a nivel casero, así como detalles del software utilizado para poder realizar mejoras o cambios que se adecuen más a las necesidades individuales.

2. INTRODUCCIÓN

Desde hace tiempo y gracias a la popularización de tecnologías tales cómo Arduino y la impresión 3D ha surgido el movimiento “maker”, que aboga por el diseño y fabricación de prácticamente cualquier cosa por parte de uno mismo y el compartir con el resto del mundo, normalmente sin ánimo de lucro y de forma altruista, las ideas y diseños que uno es capaz de realizar. Este movimiento también aboga por el rediseño y mejora de productos ya existentes. Cabe destacar que “los makers” experimentaron un crecimiento de forma exponencial durante la pandemia cuando individuos que no se conocían empezaron a coordinarse para la fabricación de pantallas de protección y acoples de mascarilla con el fin de ayudar a la sociedad.

Por otro lado y sin tener ningún tipo de relación, desde hace muchos años, el mercado de la alimentación ha estado y está dominado por unas pocas marcas que un pequeño catálogo de productos a los que los consumidores nos hemos de amoldar. Para luchar contra este pequeño catálogo y recuperar la variedad, durante los años 70, un pequeño grupo de insurgentes se plantearon si no sería posible realizar ellos mismos uno de sus productos favoritos para adecuarlos más a sus gustos personales y no tener que elegir siempre entre las limitadas opciones comerciales disponibles, este producto es la cerveza. A partir de aquellos pensamientos nació el movimiento de los “brewers”, gente que empezó a fabricar su propia cerveza artesana y a compartir sus recetas entre amigos y familiares.

Aunque a priori estos dos grupos tan distintos no parezca que tenga mucha relación, nada más lejos de la realidad. Ambos grupos luchan por ser autosuficientes, luchan por hacer algo que les gusta y compartirlo con los demás. De la intersección de ambos grupos nació la idea de este proyecto, una microfactoría casera para la elaboración de bebidas fermentadas (principalmente cerveza), que sea libre, fácilmente modificable por cualquier persona y que permita la repetibilidad de las recetas realizadas.

2.1. BEBIDAS FERMENTADAS

Las bebidas fermentadas son aquellas que han sido sometidas a un proceso de fermentación. En este proceso, las levaduras convierten los azúcares presentes en los alimentos en alcohol y CO_2. Este proceso, que se lleva a cabo de forma natural, ha sido utilizado por el ser humano desde tiempos remotos, dando lugar a una amplia variedad de bebidas que han tenido un importante papel en la cultura y la gastronomía de muchas civilizaciones.

2.1.1. BREVE CONTEXTO HISTÓRICO

Aunque son todo especulaciones, se cree que los orígenes de las bebidas fermentadas se remontan a la prehistoria, cuando los humanos comenzaron a practicar la agricultura ya que la fermentación de cereales, frutas y otros alimentos, era un método eficaz

para conservarlos. Las primeras evidencias arqueológicas de la producción de bebidas fermentadas se han encontrado en China, Mesopotamia y Egipto. En China, se han descubierto vasijas de cerámica que datan del 7000 a.C. que contenían residuos de arroz fermentado, miel y frutas. En Mesopotamia, se han encontrado tablillas de arcilla que datan del 3000 a.C. que mencionan la cerveza como forma de pago. Y en Egipto, se han encontrado cientos de tinajas de vino en tumbas.

Figura 1-Tablilla "Beer o'clock" del año 3000 AC donde se menciona la cerveza como forma de pago, custiodiada por el British Museum

En el mundo clásico grecorromano, el vino fue la bebida fermentada estrella suponiendo un elemento central de su cultura, tanto es así que hasta tenía su propio dios Dionisio o Baco para los romanos, del cual proceden las famosas fiestas conocidas como bacanales. Los romanos expandieron y perfeccionaron el cultivo de vid por toda su área de influencia. Mientras que en la cultura nórdica, debido a su clima principalmente, la bebida más consumida fue la hidromiel, muy apreciada también por sus propiedades curativas y su asociación con los dioses.

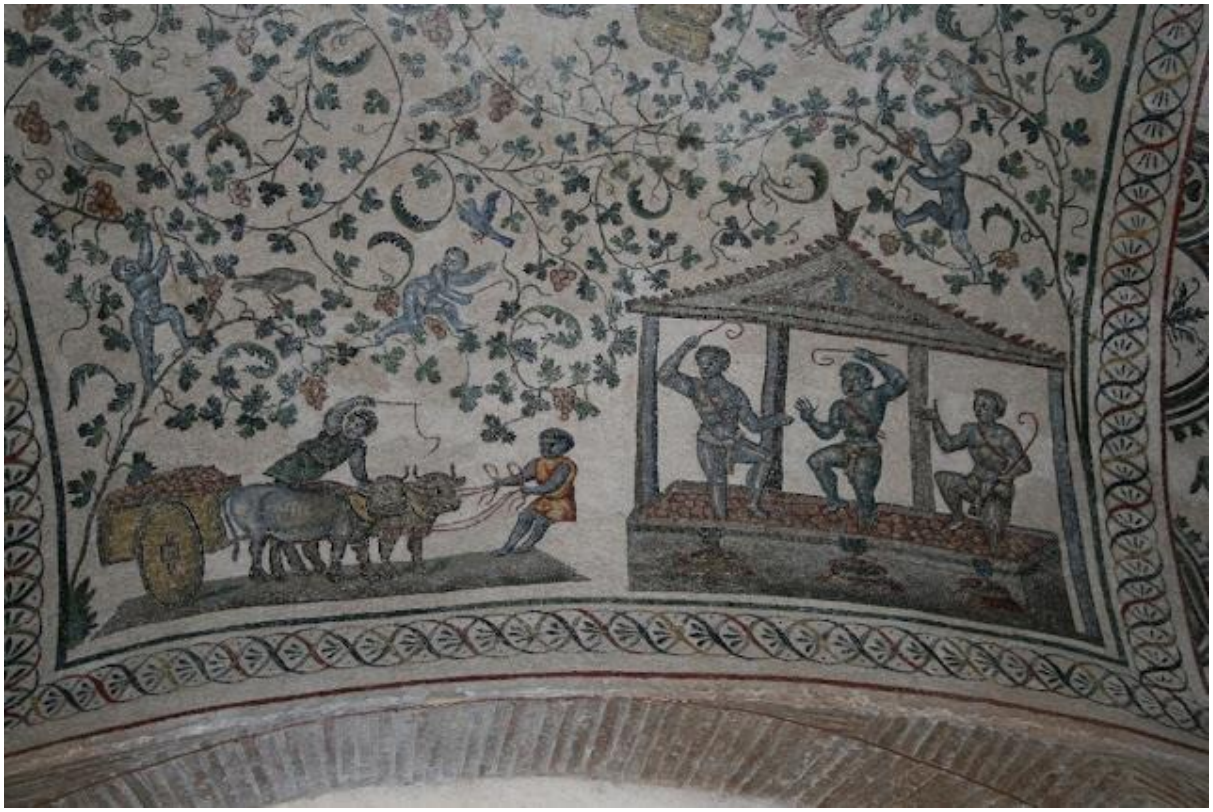


Figura 2-Mural romano donde se aprecia la elaboración de vino ubicado en el mausoleo de Santa Constanza en Roma

Durante la Edad Media, la producción de bebidas fermentadas mantuvo su importancia. La cerveza se convirtió en la bebida nacional de muchos países europeos, se crearon leyes en torno a ella, como “la ley de pureza alemana” y su consumo se incrementó y expandió especialmente gracias a la religión, ya que muchos monasterios la elaboraban tanto para consumo propio como para la venta a peregrinos y viajeros, mientras que el vino empezó a considerarse sagrado, siendo la práctica que sigue vigente hoy en día como “la sangre de Cristo”.

En el Renacimiento otras bebidas empezaron a ganar popularidad como la sidra, elaborada a base de manzanas, especialmente en Asturias y en la zona sur de Francia. Gracias a la colonización de América se abrió un nuevo mundo con el descubrimiento de nuevos ingredientes tales como el maíz y nuevas bebidas elaboradas por las culturas prehispánicas como el pulque, elaborado a base de pulpa de agave

Figura 3-Deidad de pulque soplando en tubo sobre un tarro de pulque y hombres bebiendo pulque, ilustración del Codex Magliabechiano

Cuando llegó la Edad Moderna y la Revolución Industrial el mundo sufrió grandes cambios gracias al descubrimiento de nuevas técnicas, estas también se desarrollaron para la producción de bebidas fermentadas. Se descubrió la pasteurización que permite la eliminación de los microorganismos causante del deterioro de los alimentos. Se inventó el proceso de destilación, que permitió producir bebidas alcohólicas de mayor graduación alcohólica y durante esta época también se industrializó la fabricación de cerveza, lo que permitió abaratar costes y democratizarla aún más si cabe.

En la actualidad, las bebidas fermentadas siguen siendo una parte importante de la cultura y la gastronomía de todo el mundo. Al contrario que en épocas pasadas, ha surgido una nueva apreciación por la fabricación artesanal o a pequeña escala en la que los consumidores buscan una mejor calidad del producto y un producto elaborado con ingredientes locales. También ha surgido un renacimiento de bebidas que eran más locales o tradicionales como la kombucha (que se cree que proviene de Asia), el kéfir (también de orígenes inciertos, aunque se cree que proviene de tribus nómadas del Cáucaso) o la cerveza de jengibre (o “ginger bug”, tan de moda ahora en redes sociales).

Figura 4-Cervezas artesanales elaboradas en la Comunitat Valenciana

2.1.2. CERVEZA

Dentro de las bebidas alcohólicas fermentadas, la cerveza, junto con el vino, es de las de mayor importancia en la cultura europea, tanto por su historia, su consumo y ser la puerta de entrada a la fabricación artesanal o casera.

2.1.2.1. INGREDIENTES

La elaboración de la cerveza es un proceso muy estudiado y afianzado, pero ello no quita que existan miles de vertientes, prácticamente se podría decir que una por cada productor. Existen distintas reglas o leyes como “La ley de pureza alemana”, que indica cuales son los ingredientes que han de utilizarse para fabricar cerveza. Actualmente cada productor utiliza aquellos ingredientes que cree que le darán mejor resultado, le resulta más fácil obtener o simplemente por el mero afán de experimentar. Esto permite la creación de cervezas tan extravagantes como “Bilk”, cerveza elaborada con leche o “Mamma Mia!” la cerveza con sabor a pizza.

Figura 5-Imagen promocional de Mamma Mia! Cerveza con sabor a pizza

Aunque de experimentos extraños está el mundo, los ingredientes para elaborar cerveza se clasifican en dos grandes grupos:

- Ingredientes básicos:
  - Agua, el ingrediente principal.
  - Malta, que proporciona los azucares requeridos para la fermentación.
  - Lúpulo, que aporta amargor, frescor y permite una mejor conservación.
  - Levadura, responsable de la fermentación.

- Ingredientes adjuntos:
  - Cereales no malteados, para dar cuerpo y sabor.
  - Azúcares, para incrementar el contenido alcohólico.
  - Aromatizantes, tales como la fruta, especias, cortezas de árbol, …
  - Otros, aquí la imaginación es el límite…

2.1.2.2. TIPOS DE CERVEZA

Con la combinación de los ingredientes y los métodos de fabricación utilizados, las combinaciones de cerveza son casi infinitas, aun así se han clasificado en función de diversos factores tales como procedencia, sabores y graduación alcohólica en el siguiente diagrama artístico.

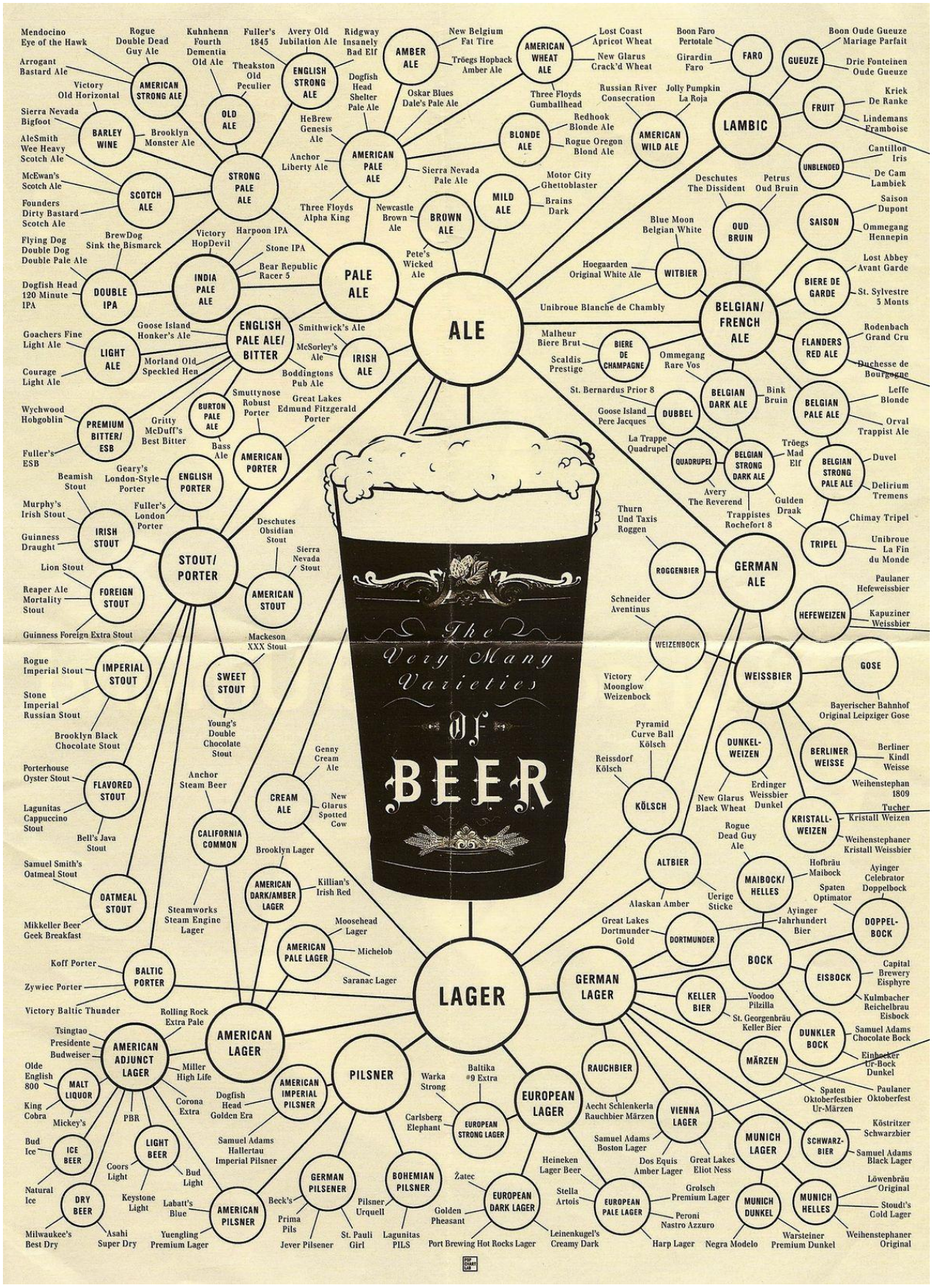


Figura 6-Póster con distintos estilos de cerveza

2.2. SISTEMAS COMERCIALES DE FABRICACIÓN DE PEQUEÑOS LOTES

Actualmente para la elaboración de cerveza en casa existen varios tipos de sistemas comerciales accesibles al gran público, estos son:

- Equipos “todo en uno”.
- Equipos “de iniciación”.
- Otros equipos (“RIMS” y “HERMS”).

2.2.1. EQUIPOS “TODO EN UNO”.

Los sistemas de elaboración de cerveza todo en uno combinan todos los componentes necesarios para elaborar cerveza en un solo aparato, lo que los hace fáciles de usar y configurar. Como todos los equipos amateurs, suelen tener una capacidad de unos 20 litros, aunque lógicamente dependerá del modelo elegido.

Sus componentes básicos son un hervidor, un macerador o cubeta para el grano, un grifo y un sistema de control de temperatura. Aunque también se pueden encontrar con características adicionales tales como filtros de lúpulo, sistema de enfriamiento mediante líquido y bombas de recirculación.

Los componentes básicos de un sistema de elaboración de cerveza todo en uno son los siguientes:

- **Macerador o cubeta para el grano:** El macerador es el recipiente donde se vierte la malta y el agua para que se produzca la conversión de los almidones en azúcares fermentables. Al tratarse de un producto para uso alimenticio, el material utilizado es acero inoxidable. Se trata de un recipiente extraíble con filtros tanto en su parte superior e inferior para evitar que se queden restos de malta en el mosto ya que podría perjudicar la calidad final del producto. Su uso sería similar al de una bolsita de una infusión.
- **Hervidor:** El hervidor es el recipiente donde se hierve el mosto y donde se realiza la maceración junto con el macerador. Está fabricado con acero inoxidable y constituye la gran parte de la maquinaria.
- **Grifo:** El grifo es el dispositivo que se utiliza para sacar el mosto del hervidor y almacenar para su fermentación.
- **Sistema de control de temperatura:** El sistema de control de temperatura permite mantener la temperatura del mosto a un nivel constante durante el proceso de maceración y para incrementar la temperatura para la ebullición.

Características adicionales:

- **Filtro de lúpulo:** El filtro de lúpulo se utiliza para eliminar los restos de lúpulo del mosto después de la ebullición. Debido a la gran cantidad de opciones disponibles, no se puede generalizar.
- **Bomba:** Se trata de una bomba peristáltica o de uso alimentario que permite la recirculación del agua-mosto para mejorar el rendimiento de conversión en azúcares, así como para la homogeneización del mosto.
- **Sistema de enfriamiento mediante líquido:** El sistema de enfriamiento es el dispositivo que se utiliza para enfriar el mosto después de la ebullición. Habitualmente se trata de un serpentín que se introduce mientras el mosto está en ebullición y se hace circular un líquido refrigerante por su interior. El sistema de enfriamiento suele estar hecho de cobre o acero inoxidable. Dependiendo del modelo existen distintas alternativas, desde que esté integrado hasta tener que introducirlo manualmente.

Dentro de la gran variedad de ejemplos comerciales se muestran algunas de las marcas más conocidas:

- Grainfather G30 (PVP 898.95€).

Figura 7-Grainfather G30

- Brew Monk All-in-one (PVP 440€).

Figura 8-Brew Monk All-in-one

Las principales ventajas de los sistemas de elaboración de cerveza todo en uno son su facilidad de uso y configuración y el reducido espacio que ocupan.

Mientras que sus principales desventajas son su elevado precio, especialmente para aquellas personas que empiezan y dificultad de reparación en comparación con los sistemas

2.2.2. EQUIPOS “DE INICIACIÓN”.

Los equipos de iniciación o “starter kits” son equipos pensados para aquellas personas que quieren empezar con la creación de sus propias bebidas fermentadas. Se trata de equipos muy baratos (empezando desde los 20€), fabricados normalmente con plástico alimenticio. Este tipo de kits suele estar compuesto de uno o dos cubos de 30 litros, utilizados para la fermentación y distinto aparataje necesario tales cómo paletas, airlocks (válvulas antirretorno), desinfectante, … siendo necesario que el usuario disponga de aparamenta para realizar tanto la maceración cómo la ebullición.

Los componentes que suelen incluir estos equipos son:

- **Barril de fermentación:** Barril de plástico alimenticio de color usualmente blanco para filtrar los rayos del sol y permitir ver la cantidad de líquido en su interior. Suelen llevar un grifo para facilitar su vaciado y una tapa especial que permite la colocación de un airlock.
- **Airlock:** Se trata de una válvula antirretorno que se llena con líquido, usualmente agua. Permite la fermentación anaeróbica del mosto. Su función es la de dejar escapar el CO_2 que se produce durante la fermentación. También impide que entren cuerpos extraños que contaminen el mosto.
- **Paleta:** Paleta de plástico que se utiliza para remover el mosto.
- **Desinfectante:** Desinfectante utilizado para higienizar todos los instrumentos.

Como material adicional, se suele incluir probetas y densímetros para comprobar el grado alcohólico del mosto, sifones para realizar el traspaso del mosto, útiles de limpieza, … la lista es interminable.

Dentro de los equipos de iniciación, cada vendedor elabora sus propios kits, aunque todos tienen en común los elementos básicos. A continuación, se muestran algunos ejemplos:

- Equipo básico de www.tucervezacasera.com, que incluye un cubo fermentador con grifo y airlock, paleta, limpia botellas, desinfectante y un manual. (PVP 23€).

Figura 9-Equipo básico

- Kit deluxe de www.lacabanadelcervecero.com, que incluye lo mismo que el kit básico, mas aparte otro cubo, chapadora y chapas, densímetro y probeta, sifón y termómetro. (PVP 74€).

Figura 10-Kit deluxe

Las principales ventajas de los kits de iniciación son su bajo precio y mantenimiento y que se tratan de herramientas y útiles que (casi)siempre van a ser necesarios en la elaboración de bebidas fermentadas.

Su mayor inconveniente es que solamente son equipos aptos para la fase final de la elaboración, es decir para la fermentación y el envasado, por lo que siempre es necesario equipo adicional para la maceración cómo para el hervido.

2.2.3. OTROS EQUIPOS (RIMS Y HERMS)

Existen también otros sistemas utilizados en la elaboración artesana de cerveza y bebidas fermentadas, de hecho, se podría decir que cada “brewer” tiene su propio sistema, dos de los más significativos son los “RIMS” y “HERMS”.

Los equipos RIMS (Recirculating Infusion Mash System / Sistema de infusión de calor al macerado por recirculación) es un sistema para controlar la temperatura del macerado por el cual el mosto recircula a través de una fuente de calor directa, como puede ser un elemento eléctrico o un tubo al que se aplica calor.

Los equipos HERMS (Heat Exchanged Recirculating Mash System / Sistema de transferencia de calor para el macerado por recirculación): sistema por el cual se mantiene la temperatura del macerado haciendo pasar el mosto por un serpentín que se encuentra dentro de un recipiente con agua (HLT) a temperatura controlada.

Ambos tienen en común que se tratan de equipos de múltiples recipientes, voluminosos y relativamente caros pensados más para pequeños negocios tales como micro cervecerías o sitios de restauración que sirvan sus propias bebidas caseras.

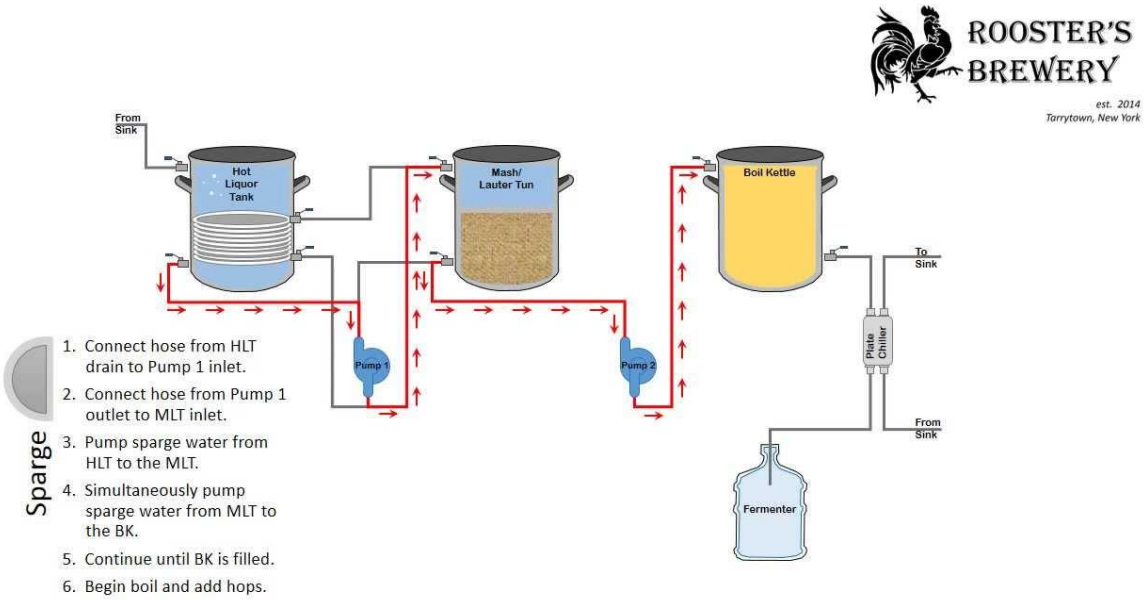


Figura 11-Esquema sistema HERMS

3. PROCESO DE FABRICACIÓN

En esta sección se va a realizar una breve explicación del proceso de fabricación de bebidas fermentadas alcohólicas tomando como ejemplo la cerveza ya que se trata de una de las más completas y que más procesos involucra. Otro tipo de bebidas alcohólicas fermentadas tales como el vino, el sake o la hidromiel comparten también la mayoría de estos procesos.

3.1. ELABORACIÓN

La elaboración de cerveza requiere de distintos pasos bien diferenciados y estudiados, a nivel casero no es posible realizar algunos de ellos ya que requieren de grandes instalaciones, habitualmente costosas. Estos pasos se explicarán, pero no se tendrán en cuenta a la hora de diseñar la micro factoría ya que existen soluciones comerciales baratas que ahorrarán tiempo y dinero al igual que proporcionarán siempre la misma calidad, lo que permitirá eliminar variabilidad. No se entrará en excesiva profundidad en la explicación de cada proceso, ya que no es el objetivo de este proyecto y de cada proceso se podrían sacar un sinfín de estudios.

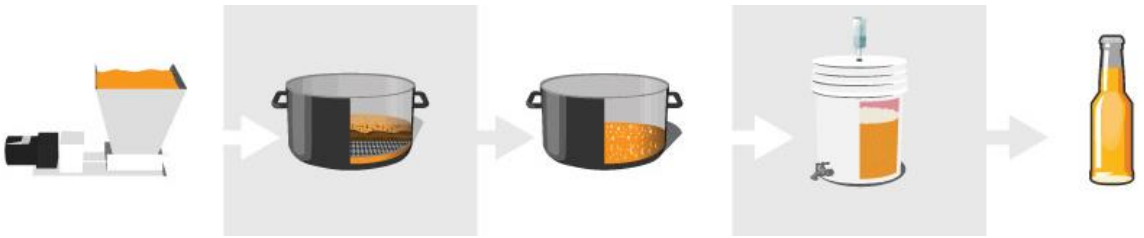


Figura 12-Breve esquemático del proceso de fabricación de cerveza

3.1.1. MALTEADO

El malteado, que provine del verbo “maltear”, de acuerdo con la RAE se define cómo: “Forzar la germinación de las semillas de los cereales, con el fin de mejorar la palatabilidad de líquidos fermentados, como la cerveza.” Es un proceso que, debido a sus características, queda fuera de este diseño. El proceso para realizar un malteado del grano consiste a “grosso modo” en tener en remojo el grano, extenderlo en una serie de camas, ubicadas habitualmente en cámaras con una elevada humedad ambiental y temperatura, durante un determinado tiempo hasta que empiece a germinar el grano (las condiciones específicas dependen del tipo de grano) y acto seguido proceder a su tostado a baja temperatura para detener la germinación.

El malteado tiene principalmente tres funciones de cara a la elaboración de la cerveza. Por una parte, prepara la reserva nutritiva, por otra genera las enzimas degradadoras del almidón y las proteínas, y finalmente da color, aroma y sabor al cereal. Es un procedimiento indispensable ya que transforma los almidones de los cereales en azúcares que, gracias a la

acción de las levaduras, fermentarán y se convertirán en alcohol y CO_2. En suma, a esto, la malta otorga muchas de las características de una cerveza como cuerpo, estabilidad de la espuma, aromas y color.

Figura 13-Infografía del proceso de malteado

Figura 14-Distintos tipos de malteado para el mismo tipo de grano

3.1.2. TRATAMIENTO DEL AGUA

El ingrediente mayoritario de la cerveza es el agua ya que supone aproximadamente el 85%, por tanto, es incuestionable que esta ha de tratarse previamente. Dependiendo del tipo de cerveza que se quiera realizar, esta habrá de tener unas características específicas, de hecho, en las grandes cervecerías industriales se suelen agregar sales y otros productos para homogeneizar el sabor del agua en todos sus centros de fabricación.

Los factores que más influyen en el sabor del agua son el pH, la dureza de esta (cantidad de sales y minerales disueltos en el agua) y la cantidad de cloro presente. El pH es un valor relativamente fácil de alterar mediante la adición de sulfato de calcio para reducirlo o carbonato cálcico para aumentar, mientras que para la eliminación del cloro las técnicas caseras más habituales son hervir el agua, dejarla en reposo (el cloro se evapora a temperatura ambiente) o utilizar filtros de carbón activo. Otra técnica habitual es la de realizar un proceso de osmosis inversa al agua.

Aunque existen las técnicas anteriormente citadas, a menos que se quiera realizar algún tipo muy específico de cerveza y se disponga de un equipo de laboratorio completo, lo habitual y más económico es utilizar agua embotellada.

3.1.3. MACERACIÓN

La maceración es el proceso durante el cual se deja reposar la malta ya molida en agua para extraer todos aquellos azúcares que servirán para la posterior fermentación. Este producto obtiene el nombre de “empaste” y no deja de ser la relación de la mezcla entre malta y agua. El empaste es uno de los factores clave, ya que los almidones se disuelven en agua y si la relación es baja (2 l/kg) estos no se disolverán, por tanto, se desperdiciará mucha malta y el rendimiento será bajo, mientras que por otro lado si la relación es muy alta (6 l/kg) las enzimas presentes (α-amilasa y β-amilasa) en la malta no realizarán su acción de convertir el almidón en azúcares fermentables.

Dentro de la maceración habitualmente se utilizan 3 técnicas distintas, la maceración por infusión simple, la maceración por infusión escalonada y la maceración por decocción.

3.1.3.1. Maceración por infusión simple

Es la forma más sencilla de maceración y normalmente la más utilizada por los principiantes en la elaboración de cerveza artesanal. Es exactamente igual a cuando se realiza cualquier tipo de infusión (te, manzanilla, …) simplemente consiste en poner la malta en agua precalentada y aislar la mezcla lo máximo posible para evitar bajadas de temperatura. Dependiendo del tipo de receta tanto la temperatura a la que debe llegar el empaste cómo el tiempo de reposo para la acción de las enzimas será distinto. Como normal habitual, la temperatura ideal para la maceración es entre los 65ºC y 70 ºC ya que en ese rango de temperaturas están activas las enzimas amilasas.

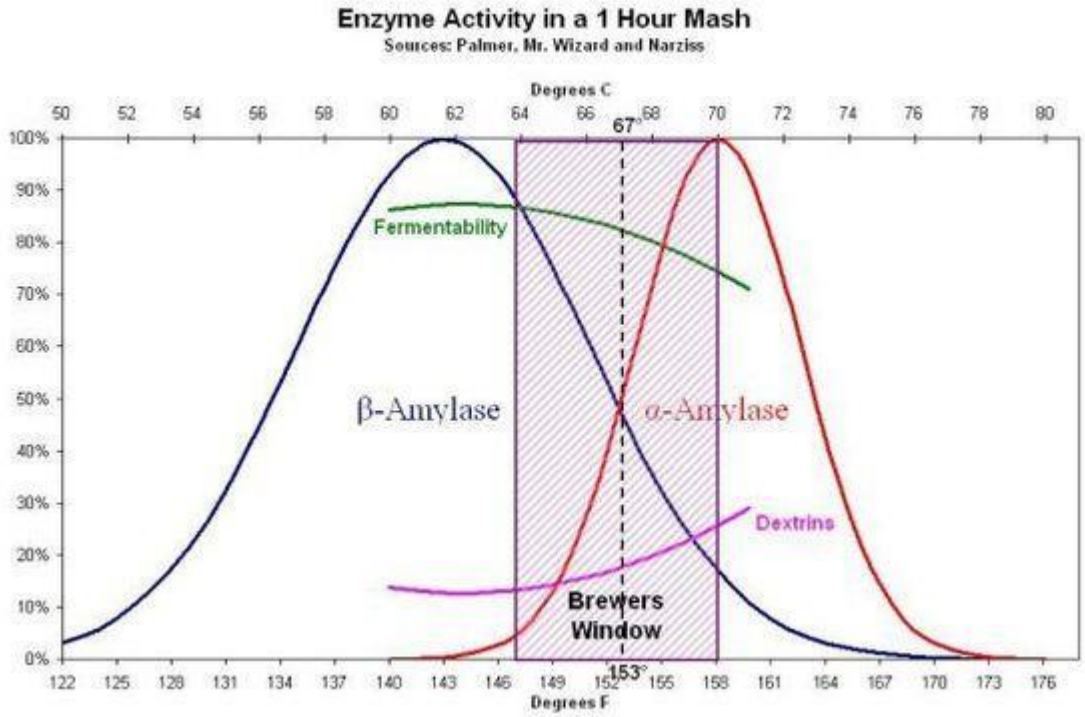


Figura 15-Gráfico acción enzimas temperatura durante el macerdo

3.1.3.2. Maceración por infusión escalonada

La maceración escalonada consiste en la aplicación progresiva de calor al mosto con periodos estancos de temperatura, o reposo térmico, que permitirán la acción de las enzimas para la disolución de los almidones así como la activación de otros compuestos más complejos. Habitualmente esta técnica, aunque más complicada permite obtener un mejor rendimiento de la malta al igual que conseguir cervezas más complejas. Como es habitual, no existe una rampa de temperaturas perfecta, ya que intervienen multitud de factores y cada receta tendrá su propia rampa de acuerdo con las características deseadas.

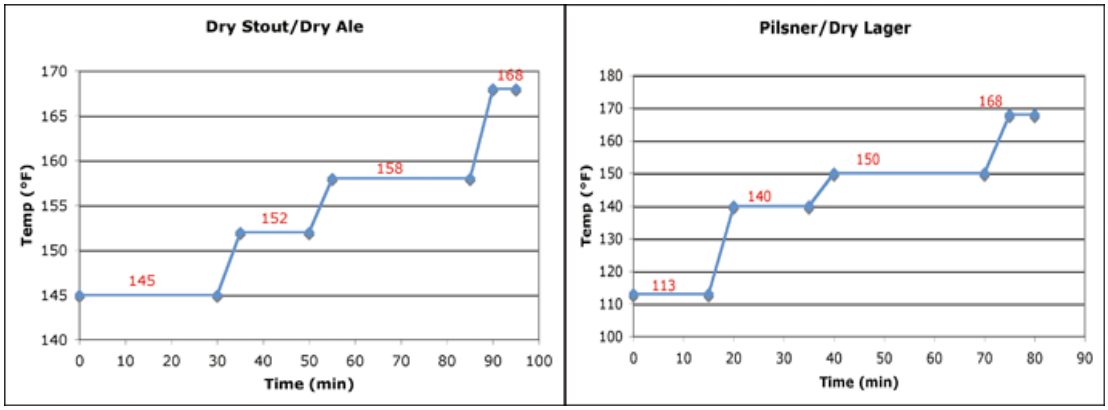


Figura 16-Comparativa rampas de temperatura entre distintas recetas de elaboración de cerveza

3.1.3.3. Maceración por decocción

Este tipo de maceración está habitualmente clasificado como el más complicado de realizar, ya que su proceso consiste en realizar una maceración escalonada tal y como se ha explicado anteriormente con la salvedad que se retira parte del empaste para hervirlo y luego volverlo a juntar con el resto. Esta parte retirada se denomina “temple” y la decocción puede realizarse tantas veces cómo se desee.

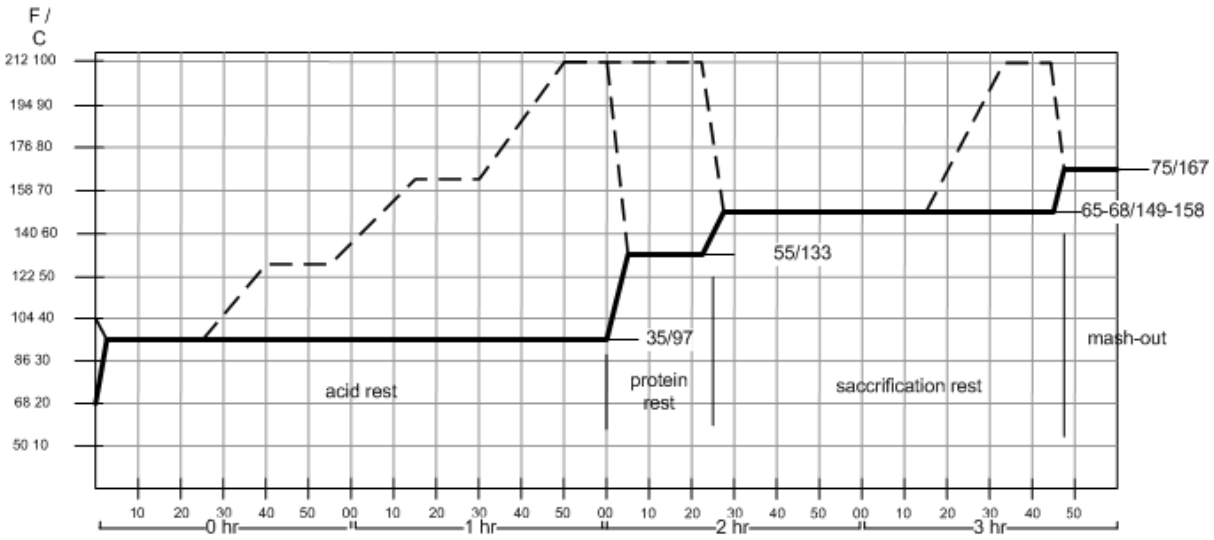


Figura 17-Diagrama decocción

3.1.4. RECIRCULADO,

El recirculado es un proceso optativo que tiene como objetivo mejorar el rendimiento y crear un filtro natural con la malta para eliminar impurezas y turbidez. Este consiste en tomar el mosto, el producto obtenido durante el macerado, de la parte inferior del macerador y volverlo a introducir inmediatamente por la parte superior, evitando la formación de remolinos, para así, por un lado, homogeneizar el mosto y por otro favorecer que todo el líquido pase a través de la malta y que esta actúe de filtro natural filtrando todas aquellas partículas que dan turbidez al producto final. Este proceso puede realizarse mientras dura la maceración, que se conoce como recirculado continuo, a través de una bomba, o al finalizar la maceración.

3.1.5. CENTRIFUGADO, PRENSADO Y LAVADO DEL GRANO

El centrifugado, el prensado y el lavado del grano son también procesos opcionales que buscan aumentar el rendimiento de la malta. Habitualmente el centrifugado y el prensado solamente suelen hacerse en instalaciones industriales, mientras que el lavado puede realizarse indistintamente.

El centrifugado del grano consiste en una vez acabada la maceración, coger todo el empaste y centrifugarlo, para así extraer la mayor cantidad de mosto posible, mientras que el

prensado, como su nombre indica, consiste en prensar el empaste para el mismo fin. Debido a que necesitan maquinaria específica y son procesos aparatosos, no vale la pena el realizarlo para pequeños lotes de cerveza.

El lavado del grano es muy proceso muy similar al recirculado ya que consiste en que una vez acabado el macerado y con todo el mosto ya en otro recipiente, calentar agua y volverla a pasar por la cama de malta para así poder recoger todos aquellos azúcares que se hayan quedado atrapados en esta. Este proceso suele repetirse hasta haber disuelto los azúcares restantes, se realiza comprobación con un densímetro. También es posible realizar este proceso con agua fría, ya que la pérdida de eficiencia es despreciable, pero se modifica el perfil de temperatura del macerado.

3.1.6. COCCIÓN

El proceso de cocción consiste en llevar a ebullición el mosto obtenido durante la maceración. Durante este proceso se agregan los lúpulos y los distintos agregados que darán sabor y personalidad al producto acabado. Este proceso también tiene por objetivo la esterilización del mosto, ya que al ser un líquido con un alto contenido en azúcares fermentables es un caldo de cultivo perfecto para levaduras y bacterias salvajes, además de servir para la eliminación del sulfato de dimetilo (DMS), un subproducto obtenido de la maceración de las maltas a altas temperaturas y eliminado por evaporación mediante la ebullición del mosto.

Los lúpulos son los encargados de darle el amargor y el frescor a la cerveza y dependiendo del tiempo que estén en ebullición, estos adquirirán unos sabores u otros, así cuanto más tiempo estén en ebullición más amargor proporcionarán y cuanto menos, más aromas y sabores. Sin entrar en detalles técnicos esto es debido a que los lúpulos son ricos en aceites y cuanto mayor tiempo de ebullición, más se evaporarán estos y más amargor proporcionarán, como se puede ver en la figura que se muestra a continuación.

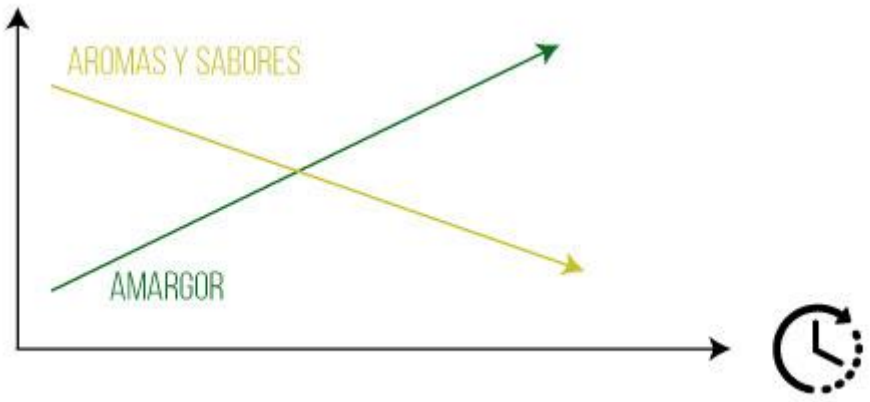


Figura 18-Comportamiento del lúpulo según el tiempo de cocción

La manera de proceder más habitual en este proceso es la adición del lúpulo de amargor nada más empiece la ebullición del mosto. Tras un periodo de ebullición de 30

minutos, se vuelve a agregar más lúpulo, en este caso los lúpulos del sabor y finalmente tras 20 minutos más de ebullición se añaden los lúpulos que darán aroma, en este momento se puede añadir también “Irish Moss”, un alga roja clarificante que permite la precipitación de las proteínas y así conseguir una cerveza más clara. Tras un periodo de 10/15 minutos más de ebullición es el momento de apagar el fuego y pasar al proceso de enfriado. Cabe destacar que en esta fase también se añaden los agregados, tales como miel, astillas de roble, romero, … y que solamente la experiencia indicará cuando se han de agregar. Una vez acabado el proceso es importante retirar los restos sólidos.

3.1.7. ENFRIAMIENTO

El siguiente proceso es el enfriamiento que consiste en bajar la temperatura al mosto hasta llegar a una temperatura de aproximadamente 25 ºC. Es muy importante que este proceso se realice de la manera más rápida posible ya que según explicado anteriormente el mosto es un caldo de cultivo excelente. La temperatura final de enfriamiento está determinada por la levadura que se utilizará para realizar la fermentación.

Para realizar el enfriamiento existen distintas técnicas, desde la más sencilla que es simplemente esperar a que la temperatura baje de manera natural hasta llegar a temperatura ambiente, la de introducir un serpentín en el mosto por el que circule un líquido refrigerante que baje la temperatura o la de bombear el mosto a través de un intercambiador de calor.

Durante este proceso, también se suele realizar la técnica del “Whirlpool” o remolino que simplemente es girar el líquido hasta crear un remolino y dejar reposar el mosto. Con esto se consigue que las partículas en suspensión se precipiten y si se ha utilizado un serpentín o simplemente se ha esperado a que la temperatura baje mediante el reposo, se acelere la transmisión de calor bajando así el tiempo de espera.

Figura 19-Serpentín dentro de olla

3.1.8. FERMENTACIÓN

La fermentación es el proceso más largo durante el proceso de realizar la cerveza ya que puede durar desde varios días hasta varios meses. Durante este proceso el mosto se convierte ya en cerveza gracias a la acción de las levaduras. Este proceso consiste en trasvasar el mosto a un recipiente con cierre hermético, al tratarse de un proceso anaeróbico, e inocular la levadura deseada. En el caso de querer añadir algún agregado, tal como más lúpulo para hacer una cerveza al estilo IPA o de cortezas de limón para la hidromiel, este será el momento. Cerrar el recipiente herméticamente y colocar una válvula de escape de un solo sentido (airlock) que permitirá la expulsión del CO_2 producido por la fermentación e impedirá la entrada de aire, así como la explosión del recipiente.

Para conocer cuando se termina el proceso, se ha de tomar una muestra inicial del mosto y medir su gravedad específica inicial mediante un densímetro, esta medida nos dará una estimación del grado alcohólico final. Regularmente se irán sacando muestras para comprobar que la fermentación sigue su curso o ya ha terminado, para ello se utiliza el valor de 1,010 como valor final de la fermentación.

Para el cálculo de la graduación alcohólico (ABV) se suele utilizar la siguiente fórmula, aunque en internet se pueden encontrar multitud de calculadoras.

$$ABV = (Gravedad\ inicial - Gravedad\ final) \cdot 131.25$$

Debido a que las levaduras son seres vivos, están muy influenciadas por la temperatura, por lo que siempre se ha de dejar el macerador en un lugar con la temperatura lo más estable posible. En las instalaciones industriales, suelen tener salas acondicionadas para que la temperatura sea estable, aunque a nivel casero lo habitual es almacenar los fermentadores en armarios en los que la temperatura es siempre estable y no están expuestos ni a corrientes de aire ni a luz solar directa.

Figura 20-2 fermentadores caseros

3.1.9. CLARIFICADO

El siguiente proceso en la elaboración es el clarificado, se trata también de un proceso opcional que sirve para darle más claridad a la cerveza. Existen distintas técnicas para realizarlo, pero los tres más habituales son el agregar productos clarificantes, que permiten la solidificación y precipitado de proteínas y células de levadura, el “choque frío” que consiste en bajar la temperatura hasta casi la congelación para conseguir el mismo resultado y el simple filtrado. Cabe destacar que el clarificado también puede realizarse durante la fase de cocción mediante la adición de “Irish Moss (Chondrus crispus)”.

3.1.10. ENVASADO Y CARBONATACIÓN

El último paso en la elaboración de cervezas consiste en el envasado y carbonatación. Tras el clarificado, la cerveza aún no está lista para consumir, ya que no tiene gas, para ello existe el proceso de carbonatación. Primeramente, la cerveza ha de ser transferida al recipiente final donde se almacenará, estos pueden ser los típicos botellines o barriles de cerveza. En el caso de los botellines el proceso de carbonatación consistirá en agregar azúcares a cada botella y cerrarla herméticamente. La poca levadura que quede viva después de la fermentación consumirá estos azúcares y producirá CO_2, que quedará disuelto en la cerveza. Este proceso de carbonatación natural suele estar finalizado alrededor de los 3 días, a partir de los cuales ya se podrá consumir la cerveza. En caso de almacenar la cerveza en barriles, la carbonatación se realizará mediante la adición de CO_2 de manera artificial en el momento de servir (el mismo proceso que se emplea en el sector de la restauración), aunque existen soluciones más pequeñas para el elaborados casero.

Figura 21-Distintos barriles para el almacenamiento de cerveza casera y acople para cartuchos de CO_2

4. DISEÑO DE LA MICRO FACTORÍA

4.1. CONSIDERACIONES

Antes de empezar a diseñar la microfactoría, hay que determinar una serie de objetivos mínimos, así como la adecuación a un presupuesto. Se debe diseñar un sistema que sea compacto, que permita la realización de pequeños lotes con gran repetibilidad, es decir que la variación provenga de los ingredientes empleados y no del proceso, apto para el consumo personal pero que a la vez tenga bastante versatilidad que pueda servir para la elaboración de recetas prototipo a cualquier nivel de fabricación.

La microfactoría deberá cumplir con todas las normas sanitarias mediante el empleo de material apto para el uso alimenticio (acero inoxidable preferiblemente) o plásticos aptos para la elaboración y manipulación de productos alimenticios. Esta ha de ser fácilmente ampliable y durante el desarrollo se discutirán alternativas. Estas pueden ser más económicas para un presupuesto más ajustado, que aporten funcionalidades extras o simplemente otra manera de realizar el proceso para que los usuarios puedan aprovechar material ya disponible o prefieran utilizar otro método.

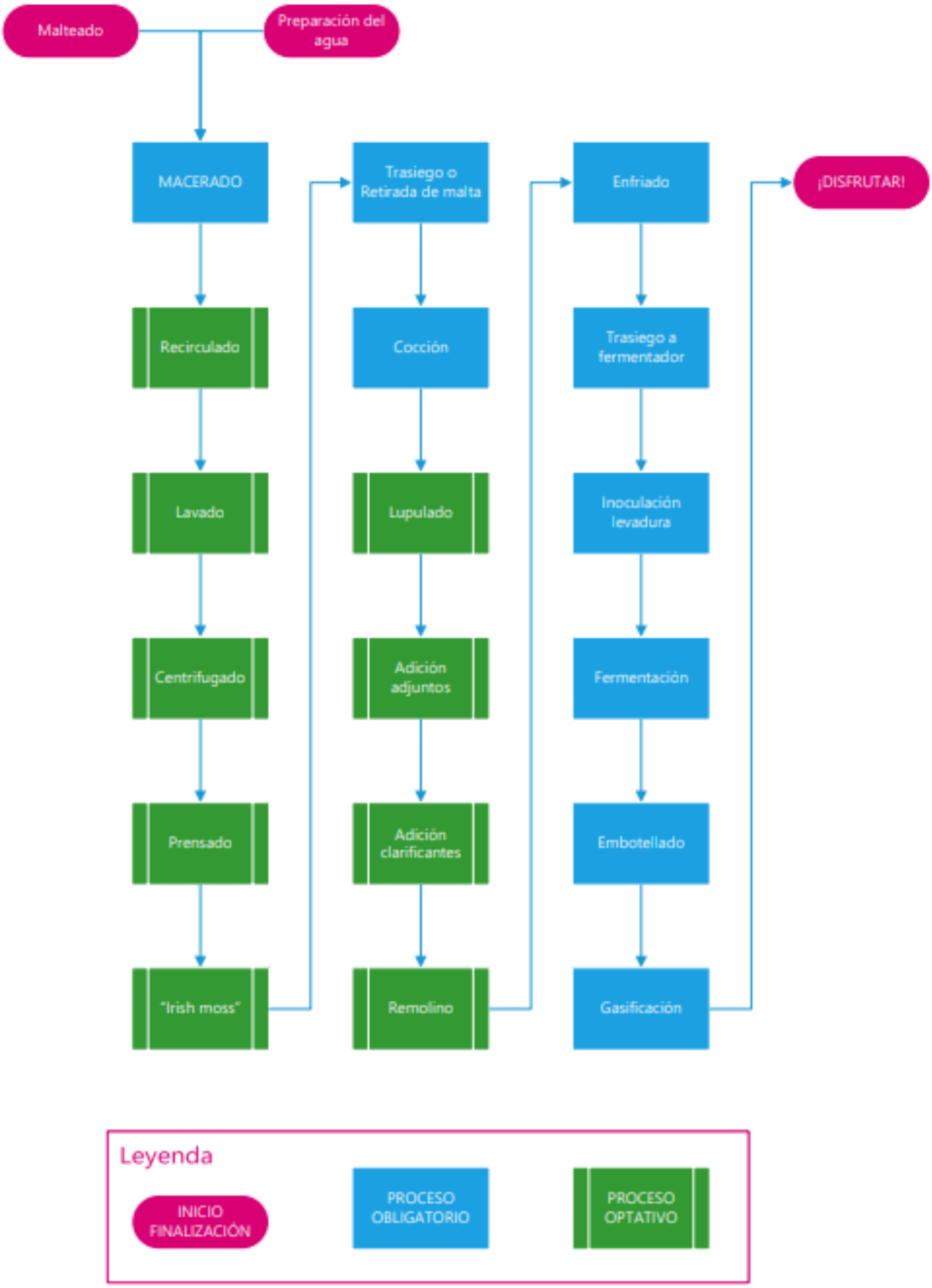


Figura 22-Diagrama de proceso de fabricación de la cerveza

4.2. JUSTIFICACIÓN DEL TIPO ESCOGIDO

A continuación, se muestra una tabla con los pros y contras de los distintos tipos de equipo presentados durante la introducción. Esta servirá de base para la decisión final del equipo a diseñar.

TIPO	PROS	CONTRAS
Básico	Muy económicos. Se suele utilizar utensilios ya disponibles.	Múltiples recipientes. Difícil control de temperaturas. Uso de utensilios de cocina Para lotes medianos (20 l) suele utilizarse gas.
Todo en 1	Precio asequible. Un solo recipiente para la maceración y la ebullición. Fácil recirculado continuo. Posibilidad de BIAB o filtros de inox. Fácil control de temperaturas de calentamiento. Fácil limpieza. Fácil almacenamiento. Poco mantenimiento.	Más caro que los equipos básicos. Suelen ser soluciones comerciales con difícil reparación. Consumo energético alto.
RIMS/HERMS	Gran control de temperatura. Posibilidad de fabricación a gran escala. Se pueden realizar varios lotes de manera simultánea (intercambiando los recipientes). Pueden ser eléctricos o a gas (ebullición).	Caros. Necesitan mucho espacio y no son móviles. Difícil limpieza. Consumo energético alto.

Tabla 1-Comparativa entre tipos de microfactorías de cerveza

Después del análisis de la tabla de la tabla de pros y contras y teniendo en cuenta los objetivos iniciales del proyecto, el equipo a diseñar es del formato “todo en uno”. Este tipo de configuración permite un diseño adaptado a las posibilidades de cada usuario, desde el presupuesto, el tamaño, el grado de automatización, la personalización y un largo etcétera… Además, partiendo de un buen diseño, también es posible el ir añadiendo mejoras o cambios de manera gradual. Es lo suficientemente versátil para poder añadirle o retirarle complementos, tales como añadir un serpentín, sustituir la bolsa del grano por un filtro de inox, … La imaginación es el límite.

Por tanto, el diseño final a realizar será un “todo en 1”, que disponga de un control de temperatura para poder hacer un macerado escalonado, una bomba de recirculado para conseguir unos buenos rendimientos de la malta. Todo irá automatizado mediante un controlador (Arduino, Raspberry, …) con el objetivo de que la repetibilidad y el control del proceso sea el máximo.

4.3. MATERIALES

En esta sección se desglosarán todos los materiales escogidos para el diseño de acuerdo con los procesos de fabricación necesarios. En aquellos casos donde sea posible también se propondrán alternativas, ya que en ingeniería muchas veces no hay una única solución correcta. Se irán justificando todas las elecciones así cómo se proporcionará una base teórica del elemento escogido en aquellos puntos en los que se considere necesario.

Como punto común, todos los materiales escogidos han de ser de grado alimenticio y han de resistir altas temperaturas (100 ºC), se va a favorecer el uso de materiales que no aporten sabor y de fácil limpieza (acero inoxidable) así cómo componentes fáciles de reemplazar, mejorar o reparar.

4.3.1. OLLA

La olla será el recipiente base, el pilar desde el que se empezará a construir todo el diseño. Es importante que sea de acero inoxidable y de buena calidad. Aunque no es obligatorio, si muy recomendable que esta lleve un grifo o una válvula de salida en la parte inferior que permita su vaciado directamente sin tener que volcar o utilizar útiles. Para lotes de 20 litros, lo habitual es tener una olla con una capacidad de mínima de 25/30 litros.

Dentro de la amplísima variedad de ollas que se pueden encontrar, existen un tipo de olla eléctricas fabricadas en acero inoxidable que cuentan con una resistencia integrada dentro de su estructura, un sencillo control de temperatura mediante un selector giratorio, presumiblemente un potenciómetro y un grifo para servir. Este tipo de ollas suele verse en los restaurantes y hoteles y sirven para mantener calientes los líquidos que se encuentran en su interior.

La olla elegida es de este último tipo descrito, en concreto se elige la olla “Klarstein Biggie”, que cuenta con una capacidad de 27 l y una potencia de 2000 W. El precio en su página web es de 119 €. El único inconveniente de esta olla es que habrá que cambiar el grifo de plástico que trae de serie por uno antigoteo apto para uso alimentario. El grifo servirá para poder realizar tanto el recirculado cómo para vaciar el mosto.

Figura 23-Klarstein Biggie de 27 l

Respecto a las alternativas, no se ha podido encontrar una olla eléctrica con mejores prestaciones con el precio tan contenido.

Como alternativas más económicas se puede utilizar una olla normal de acero inoxidable y realizar un macerado simple, apoyándose de un recipiente hermético, alcanzando la temperatura de macerado bien con gas o con electricidad. Las ollas de acero inoxidable de 25 litros con grifo rondan los 100 € y suelen llevar el mismo tipo de grifo que la olla escogida, por lo que también habría que modificarlo

Para la realización de un macerado escalonado habría que estar pendiente de la temperatura todo el rato o bien comprar una resistencia eléctrica sumergible (hervidor de inmersión, con un precio aproximado de 50 € y una potencia de 1500 W) y controlarla de igual forma que se controlará la resistencia integrada.

Figura 24-Hervidor de inmersión

4.3.2. CONTROL DE LA POTENCIA DEL CALEFACTOR

Debido a las altas potencias manejadas por el calentador y a la necesidad de poder limitar la temperatura, será necesario la utilización de algún tipo de dispositivo que sea capaz de ajustar la potencia de la resistencia de la olla y que además sea controlable electrónicamente. Para ello se va a utilizar la modulación por ancho de pulsos (PWM por sus siglas en inglés) junto a un relé de estado sólido (SSR por sus siglas en inglés).

El PWM consiste en modular el ancho de los pulsos de una señal eléctrica para que su potencia promedio sea la deseada en ese momento. Se trata de una técnica ampliamente utilizada en electrónica de potencia para el diseño de fuentes de alimentación. Con esta técnica es posible un control muy preciso y su eficiencia energética es elevada. Los kits de desarrollo de Arduino son capaces de sacar señales PWM en sus pines, por lo que no habrá ningún problema en su implementación.

El control mediante un sistema PID (Proporcional-Integral-Derivativo), es un algoritmo de control de lazo cerrado que permite regular la salida de un sistema a un valor deseado. Es un algoritmo muy conocido y relativamente sencillo de implementar, ya que solamente requiere programación, de hecho, existen librerías ya programadas que permiten la utilización de estos algoritmos y solamente se han de proporcionar los valores de entrada, salida y las constantes PID.

Otra opción barajada es la de un potenciómetro en serie con el resistor que proporcione una caída de tensión a este último y por tanto disminuya su potencia. Este potenciómetro habría de operarse a mano, por lo que sería necesario utilizar un servomotor para su control electrónico.

Después del planteamiento de las distintas alternativas, la opción escogida es la de PWM cuyo valor de salida será proporcionado por un PID que controlará un relé de estado sólido (SSR) y la retroalimentación será proporcionará un sensor de temperatura. Estos relés tienen la misma función que los electromecánicos, es decir, al aplicarles una tensión de control dejarán pasar la corriente por su parte de potencia, pero al no tener la parte mecánica, estos son infinitamente más rápidos aparte de que se evitan todos los ruidos y averías mecánicas, a costa de un precio más elevado.

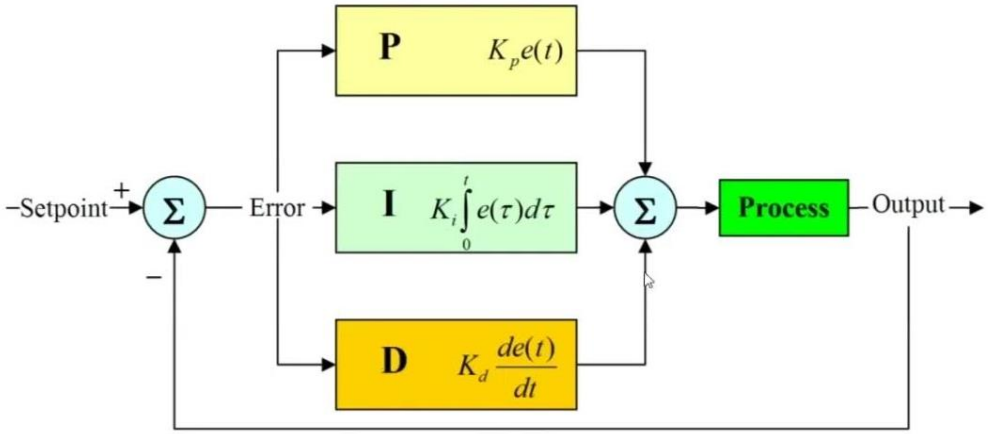


Figura 25-Diagrama de bloques de un PID de una entrada y una salida

4.3.3. GRIFO Y FILTRO

Debido a que el grifo que viene de serie con la olla eléctrica es un grifo de plástico pensado para servir, no es apto para el recirculado, por lo que ha de ser sustituido por otro. Dentro de la gran variedad de grifos, se sustituye por uno de ¾'' que viene con filtro incorporado. Este filtro que no estaba en el diseño inicial servirá como medida de protección adicional contra emboces.

Figura 26-Conjunto grifo y filtro

Como alternativa al grifo se puede mantener el de serie, aunque será necesario realizar un diseño tanto para mantenerlo abierto, ya que tiene un retorno por muelle, como para poder conectarle una manguera para la bomba.

4.3.4. BOLSA DE MACERADO Y LUPULADO

Dentro de las posibles alternativas para realizar el macerado, se opta por BIAB (Brew In A Bag), que consiste en introducir la malta en una bolsa microperforada que facilitará todo el proceso (especialmente la limpieza) a cambio de reducir un poco el rendimiento. La ventaja de estas bolsas es su escaso precio, que son reutilizables y que al contener todo el grano en su interior es más fácil transportarlo.

Para realizar el lupulado, también se utilizarán este tipo de bolsas, pero de un tamaño más pequeño, ya que poseen todas las ventajas anteriormente descritas.

Figura 27-Bolsa macerado para usarla durante la fabricación BIAB

Como alternativas, existen filtros realizados con acero inoxidable, pero estos habitualmente están hechos a medida y como todo lo hecho a medida, son excesivamente caros.

4.3.5. BOMBA DE RECIRCULACIÓN, ACOPLES Y LATIGUILLOS

Para la recirculación continua es necesario que haya una bomba que esté constantemente recirculando el mosto. En un principio la idea era utilizar una bomba de 5 V para así no tener que recurrir a una fuente de alimentación adicional, pero todas las encontradas tenían un caudal aproximado de 1 l/min, lo que resulta insuficiente. También se estuvo barajando la opción de bombas peristálticas, pero estas resultaban caras y muy aparatosas. Finalmente, la opción escogida fue la de una bomba electromagnética fabricada en ABS con motor sin escobillas y apta para uso alimenticio.

Los latiguillos son de silicona de uso alimenticio con un diámetro interior de 10 mm y uno exterior de 14 mm. Se escogen transparentes para así poder ver que el mosto fluye sin

problemas. También se han de añadir unos acoples para poder conectar los latiguillos a la bomba sin problemas.

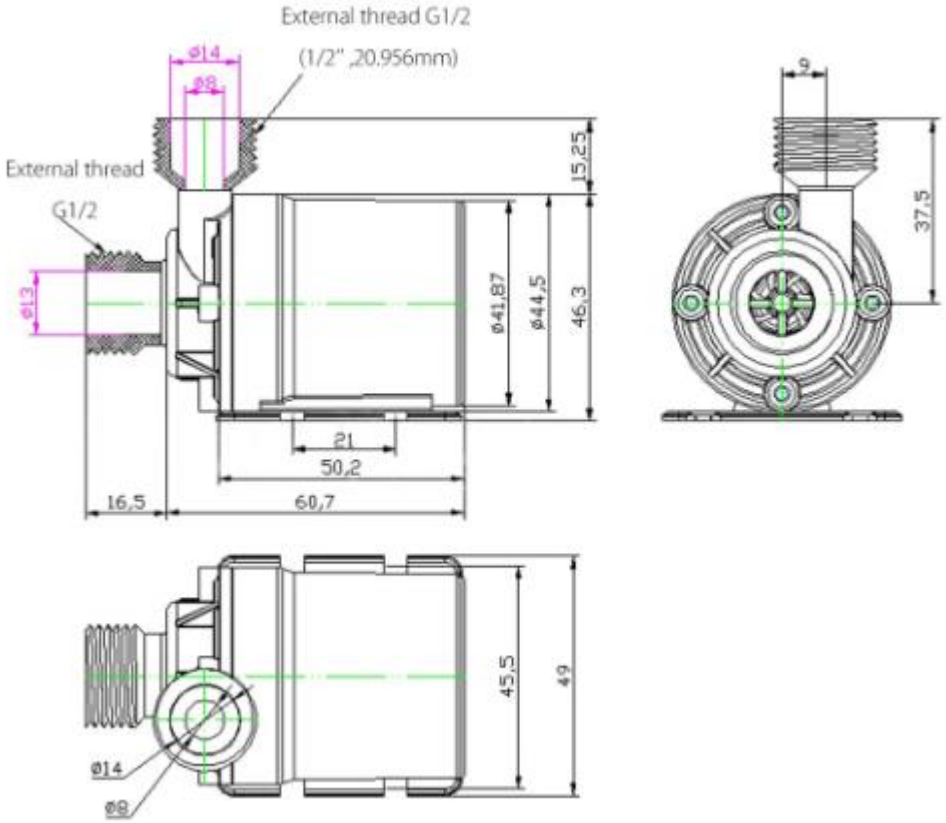


Figura 28-Alzado, planta y perfil acotados de la bomba

4.3.6. DETECTOR DE LÍQUIDOS

Con el objetivo de que la bomba no trabaje en vacío, ya que podría dañarse y quedarse inutilizable, es preciso añadir una medida de seguridad que garantizará que existe líquido en el sistema. Esta comprobación se realizará mediante un sensor de líquidos, o bien de nivel o bien de flujo. Es importante que el sensor sea exterior, para así no afectar a la calidad del producto, sea capaz de soportar altas temperaturas y lógicamente sea compatible con el resto del sistema.

Existen distintas alternativas para ambos tipos, sensores de ultrasonidos colocados en la parte superior del tanque que miden el nivel, boyas ubicadas en el líquido que permiten conocer si el líquido ha llegado a un nivel mínimo, sondas sumergidas en el líquido que permiten conocer el nivel a lo largo de la extensión de la sonda, … De entre todas ellas, existe un tipo de sensor que funciona por capacitancias, la ventaja que tiene este tipo de sensor sobre el resto es que puede servir tanto de detector de nivel, ubicándolo en el exterior del tanque, cómo de detector de flujo, colocándolo en este caso en alguna de las tuberías o latiguillos.

De entre las opciones disponibles que cumplen los requisitos de diseño, el sensor escogido es el XKC-Y28 de la empresa XKC Technology (Bao'an, Shenzhen, China), este sensor es capaz de detectar líquidos a través de una tubería de Ø>10 mm, funciona en un rango de temperaturas entre -20 y 105 ºC y es totalmente compatible con Arduino. Este sensor irá colocado en el latiguillo a la salida del grifo y antes de la entrada a la bomba, lo que permitirá protegerla contra trabajos en vacío y apagarla automáticamente al cerrar el grifo.

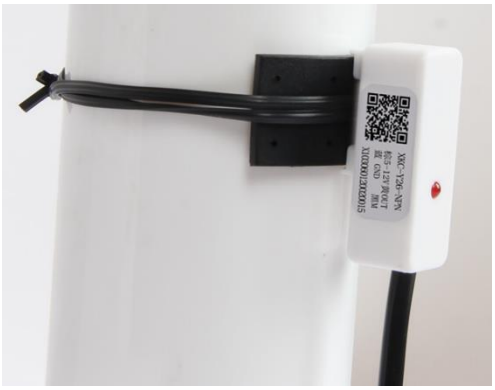


Figura 29-Sensor XKC-Y28 colocado en una tubería

4.3.7. SERPENTÍN

Para acelerar el proceso y evitar posibles contaminaciones en la cerveza después de la cocción, es necesario enfriar el mosto a temperatura ambiente lo más rápido posible. Por ello, se hace necesaria la compra de un serpentín, en este caso el serpentín irá conectado mediante unos latiguillos al grifo de agua corriente, lo que enfriará rápidamente el mosto. Para no desperdiciar el agua, la caliente se almacenará en un recipiente para luego reutilizarla.

Figura 30-Serpentín de cobre

Como alternativas, se puede utilizar un enfriador de placas, que es la versión mejorada del serpentín. Su principal problema es que necesitan de dos bombas para hacerlos funcionar, una para el mosto y otra para el líquido refrigerante (aunque también se podría conectar a la red de saneamiento y utilizar solamente una bomba).

4.3.8. KIT CERVECERO BÁSICO

Para complementar el resto del equipo, se comprará un kit cervecero Deluxe de nivel principiante que contendrá todo el resto de material necesario para la realización de la cerveza. El kit elegido llevará cubos para la fermentación, airlock (o válvula de un solo paso) para garantizar la fermentación anaeróbica, densímetro, paletas, desinfectante, …

La elección de este kit es simplemente por comodidad ya que vienen preparados con todas las herramientas básicas lo que evitará pasar horas buscando en distintas tiendas estas mismas herramientas para ahorrarse unos pocos euros.

Figura 31-Kit cervecero deluxe

Las alternativas, en este caso son muchas y variadas, desde utilizar utensilios de los que ya se disponga, como por ejemplo garrafas de agua para la fermentación o paletas de cocina de las que ya se disponga.

4.3.9. SENSOR DE TEMPERATURA

Para poder controlar con precisión las rampas de temperaturas que se producen durante el macerado y conseguir que el sistema sea automático, será necesario disponer de uno (o varios) sensor de temperatura que recojan el valor real de la temperatura del mosto y se monitorice constantemente. Dentro de las distintas alternativas, el sensor escogido es el

DS18B20 de Maxim Integrated (San José, California, Estados Unidos). Este tipo de sensor de medición de temperatura tiene un rango de medición de -55 ºC a 125 ºC, una exactitud de ±0.5 ºC, salida digital y resolución ajustable. Además, el sensor también utiliza el protocolo de comunicación “1-Wire”, que permite la conexión de varios sensores mediante un puerto, con el consiguiente ahorro de puertos y la posibilidad de incrementar el número de puntos de medición de temperatura de la micro factoría. Otro factor para la elección de este dispositivo es que es posible comprarlo ya con un encapsulado estanco por un precio muy económico. Cabe destacar que para el correcto funcionamiento del sensor es necesario que este disponga de una resistencia de “pull up” de 4.7 kΩ (la resistencia de “pull up” se encarga de mantener un pin en un estado lógico alto), con la compra del sensor es posible comprar un pequeño módulo que hace ya innecesaria esa resistencia y facilita el montaje a la placa de desarrollo.

Figura 32-Kit de sensor DS18B20, módulo de conexión y cables Dupont

Como alternativas al módulo escogido, existen los sensores LM35 de Texas Instruments (Dallas, Texas, Estados Unidos) o el PT100, ambos sensores de salida analógica. El sensor LM35 es relativamente sencillo de usar, ya que el valor de su salida es proporcional a la temperatura que esté midiendo, mientras que, para el PT100, a pesar de ser un sensor más preciso que el DS18B20, no es posible conectarlo directamente a la placa, ya que es necesario diseñar un puente de Wheatsone, lo que dificultaría que los usuarios no acostumbrados a la electrónica pudiesen repararlo o comprobarlo fácilmente.

4.3.10. PROCESADOR

Para controlar todo el proceso es necesario disponer de un dispositivo que sea capaz de leer todas las entradas, analizarlas y activar las salidas necesarias según la situación lo requiera, es decir, es necesario disponer de un procesador. Dentro de esta categoría existen múltiples alternativas desde control manual de cada uno de los instrumentos, hasta el uso de ordenadores (tanto personales como industriales). Para la elección de este componente, es necesario recordar cuáles son los objetivos del proyecto:

- Adaptarse a casi cualquier presupuesto.
- Mejorable por cualquier usuario.

- Automatización y alternativa manual.

Con estos objetivos en mente, se valoran varias alternativas, la primera de ellas es Arduino. Arduino es una plataforma de código abierto, con una gran y activa comunidad, es sencilla de usar y programar, existen varios tipos de placas de desarrollo con distinta cantidad de entradas y salidas y es extremadamente barato. Los inconvenientes de los microprocesadores de Arduino, el ATmega328, es su baja capacidad de procesamiento con respecto al resto de alternativas y que energéticamente no es tan eficiente al estar alimentado a 5 V.

La siguiente alternativa planteada son microprocesadores “ESP32” de Espressif Systems (Zhangjiang Shangai, China), este tipo de microprocesadores son más potentes que la placa estándar de Arduino. Cuentan con conectividad wifi y bluetooth integrada de serie, son compatibles con el entorno de programación de Arduino y además son también muy económicos. Una de sus principales ventajes es que todos sus pines pueden ser configurados como de entrada o salida (tanto analógica como digital) y pueden ser moduladas. Como inconvenientes cabe destacar que, al llevar menos años en el mercado, algunas de las librerías de Arduino no son 100% compatibles, por lo que si se utiliza el entorno de Arduino es posible encontrarse con fallos inesperados, por lo que es necesario mayores conocimientos de informática y electrónica, su alimentación es de 3.3 V en lugar de los 5 V de Arduino, por lo que algunos sensores es posible que necesiten alimentación externa.

La siguiente alternativa son los microprocesadores STM32 de STMicroelectronics (Plan-les-Ouates, Suiza), estos microprocesadores, al igual que el ESP32 son microprocesadores de 32 bits, por lo que tienen una potencia de cálculo mucho mayor que Arduino, además su velocidad de reloj va entre los 24 MHz a los 480 MHz, que permiten una velocidad de procesamiento mayor. Como inconvenientes hay que tener en cuenta su precio, más caro que Arduino. Al tratarse de un procesador de desarrollo más avanzado, no es apto para principiantes y requiere una serie de conocimientos más avanzados, tanto a nivel de electrónica como de programación. La comunidad formada no es tan amplia como en el caso de Arduino, por lo que para la resolución de problemas será difícil conseguir ayuda y la curva de aprendizaje es bastante elevada.

Las siguientes alternativas planteadas son Raspberry Pi, Ordenador personal (PC) y autómata programable (PLC). Estos tres equipos son muy potentes y capaces de realizar todas las tareas sin ningún tipo de problema, será muy fácil poder visualizar los datos y el estado en tiempo real mediante pantalla conectada. Como contra es imprescindible que tengan un sistema operativo para hacerlos funcionar (estos pueden ser gratuitos o de pago), son infinitamente más caros que cualquiera de las opciones planteadas anteriormente y si no es utilizado para más tareas, sería una infra utilización de los recursos. También hay que tener en cuenta que las curvas de aprendizaje serán más pronunciadas con estos sistemas.

Recapitulando, hasta el momento, el sistema de control consta de:

- Calefactor controlado mediante un PID.
- Sensor de temperatura.
- Sensor de presencia en tubería.
- Bomba de recirculación de 12 V.

- Dispositivos de entrada/salida (E/S).

Finalmente, la opción escogida es un procesador ESP32, básicamente por su bajo precio, su simplicidad de uso, su potencia de cálculo y contar con conectividad Wifi y Bluetooth de forma nativa.

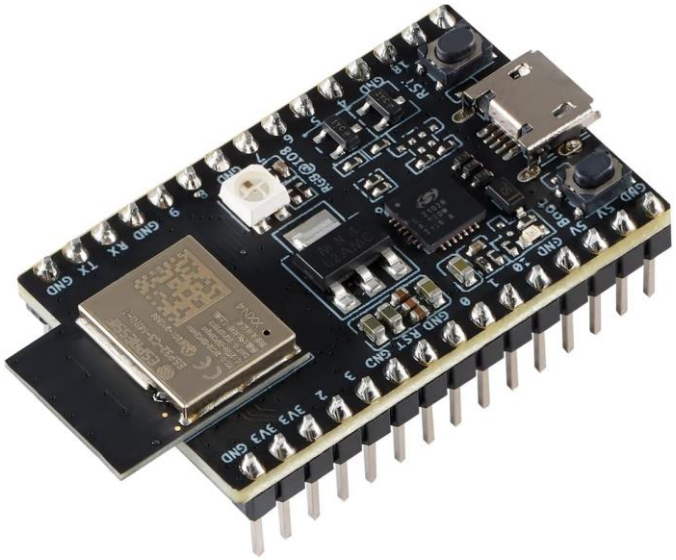


Figura 33-Placa de desarrollo de ESP32

4.3.11. INTERACCIÓN USUARIO/DISPOSITIVO

Como cualquier sistema programable, es necesario una serie de dispositivos para la interacción humano/máquina que permitan la comunicación bidireccional entre ambos. Para ello, y en gran medida dependiendo del tipo de procesador/controlador escogido, existen diversas alternativas, desde las más básicas, unas simples bombillas (o leds) que se iluminasen y apagasen según la parte del proceso a las que estuviesen asignadas a las más avanzadas como unas gafas de realidad aumentada con inteligencia artificial.

Primeramente, se va a dividir la interacción en 2 fases, por un lado, estarán los dispositivos de salida, que serán aquellos que darán información al usuario y por otro lado estarán los dispositivos de entrada, en las que el usuario dará instrucciones a la máquina.

Para los dispositivos de salida, se tienen distintas alternativas, como las 2 mencionadas anteriormente, que irían desde lo más simple a lo más complejo. Pero existen una gran variedad de opciones intermedias, como los displays de 7 segmentos, las pantallas lcd, las pantallas TFT, que no dejan de ser una evolución de las pantallas lcd y las pantallas táctiles, encontradas ampliamente en todo tipo de dispositivos. Cabe destacar que todas son bastante económicas, especialmente los displays lcd, aunque las pantallas táctiles TFT pueden triplicar el precio de las no táctiles.

Dependiendo del tipo de procesador escogido, existen también gran variedad de dispositivos de entrada, si se utiliza Arduino, lo más común es utilizar pulsadores o joysticks

para poder navegar entre las distintas opciones, mientras que, si se utiliza un procesador más potente, tal y como un PC o una Raspberry Pi, lo más habitual sería utilizar teclado y ratón o incluso cámaras de visión artificial con reconocimiento mediante IA.

Para la interacción con el usuario se utilizará una pantalla táctil de 2.8'' que estará controlada por un microcontrolador ESP32. Se utilizará un "shield" con la pantalla integrada, lo que garantizará la buena comunicación entre procesador y la pantalla y en caso de rotura o que haya que cambiar el proceso será mucho más sencillo. La opción escogida es bastante conocida entre la comunidad "maker" bajo el nombre de "Cheap Yellow Display" o CYD.

Figura 34- "Cheap Yellow Display", shield con un ESP32 y pantalla táctil

4.3.12. FUENTE DE ALIMENTACIÓN

Debido a la cantidad de elementos de distintos voltajes que se emplean será necesario la utilización de distintas fuentes de alimentación que sean capaces de soportar las cargas de los elementos. Para ello, la mejor opción será la comprar varias fuentes de alimentación cuyo voltaje de salida sea el requerido por cada dispositivo.

- Fuente de alimentación 3.3 V de corriente continua, para alimentar los procesadores, sensores y dispositivos E/S.
- Fuente de alimentación 12 V de corriente continua, para alimentar bomba de recirculado.
- Fuente de alimentación 220 V de corriente alterna, para alimentar el calefactor, proporcionado directamente por la red eléctrica.

Las alternativas en este caso no son muy variadas ya que, si los elementos no están alimentados a la tensión necesaria, en el mejor de los casos su funcionamiento será nefasto y en el peor pueden llegar a incendiarse, con el peligro que ello conlleva.

4.3.13. COMPONENTES ELECTRÓNICOS DIVERSOS

Finalmente, también es necesario tener en cuenta que harán falta una serie de materiales más “comunes” en la electrónica tales como placas de prototipado (usualmente conocidas como “protoboards”), cables eléctricos para realizar las distintas conexiones, se recomienda que sean del tipo “DuPont” que vienen con los extremos con cabezales que facilitan el conexionado a las diferentes placas, tanto la protoboard cómo a la del procesador. La mejor manera de conseguir todos estos componentes es mediante la adquisición de un kit de principiante de electrónica.

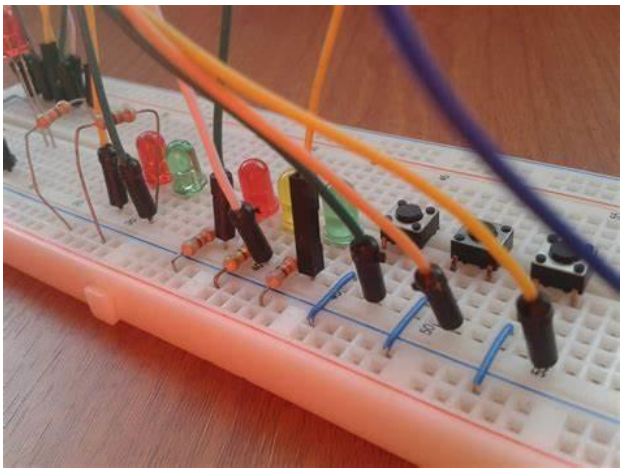


Figura 35-Protoboard con distintos cables y elementos electrónicos

4.3.14. TORNILLERÍA E IMPRESIÓN 3D

Para la instalación final se diseña un par de cajas que servirán para fijar los componentes y protegerlos contra agentes externos. La decisión de diseñar dos cajas, una para cada placa es tanto por seguridad, ya que en caso de tener una incidencia solamente sería necesario reemplazar una de las placas y no se arruinaría todos los componentes y la otra es para aprovechar las capacidades inalámbricas del ESP32. También se hace necesario la compra de tornillería, para ello la mejor opción es la de un juego de tornillos de nailon variados, tanto por su precio como por la cantidad de ellos que vienen.

Figura 36-Caja con tornillos de distintas métricas y longitudes

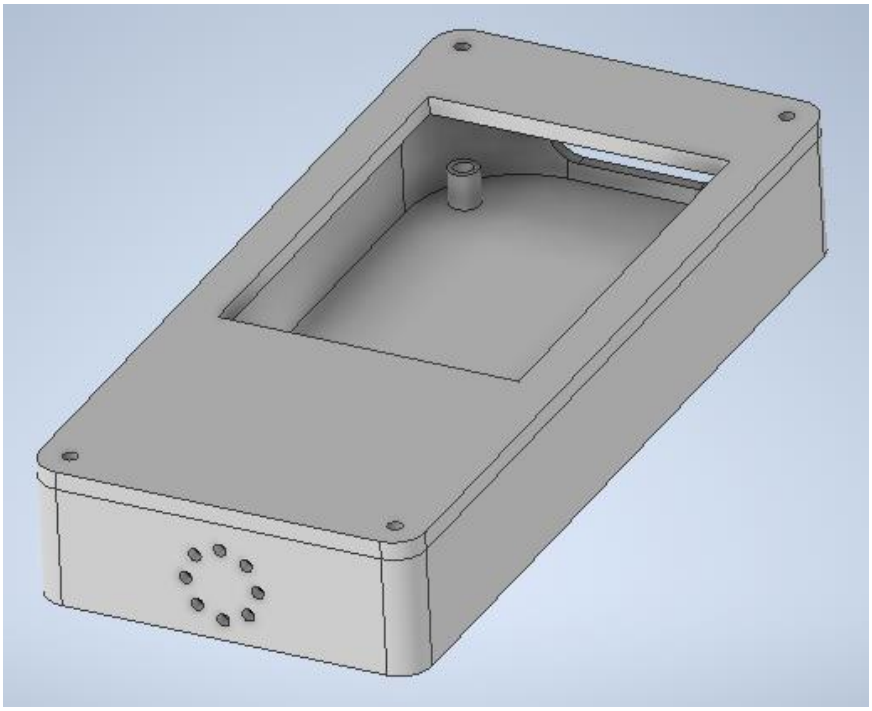


Figura 37-Modelo 3D de la caja que contendrá la pantalla táctil y la CYB

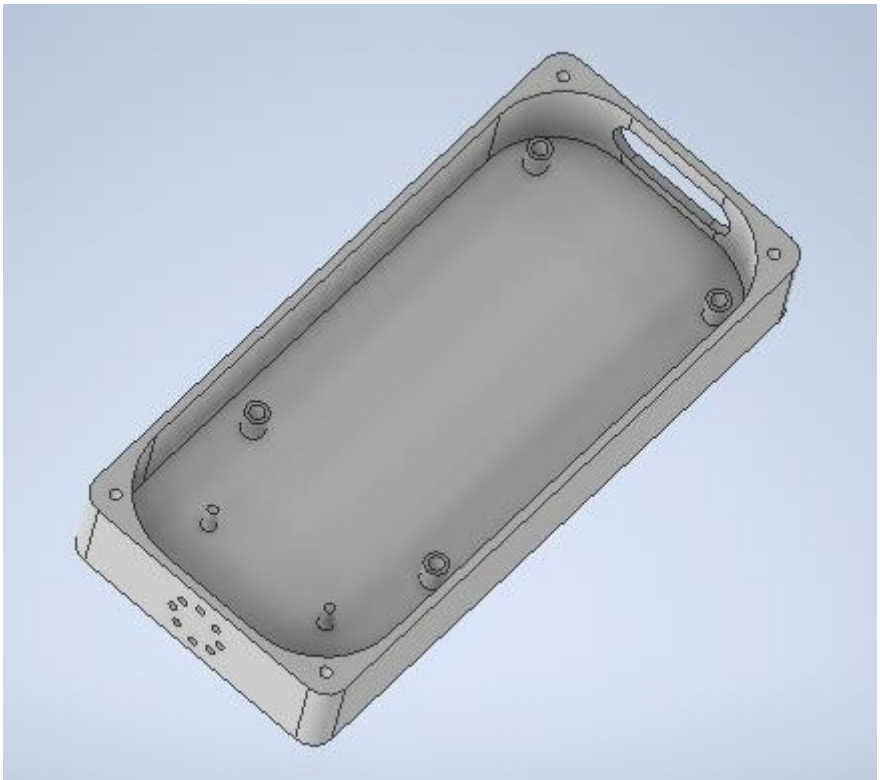


Figura 38-Modelo 3D de la caja que contendrá la pantalla táctil y la CYB sin tapa

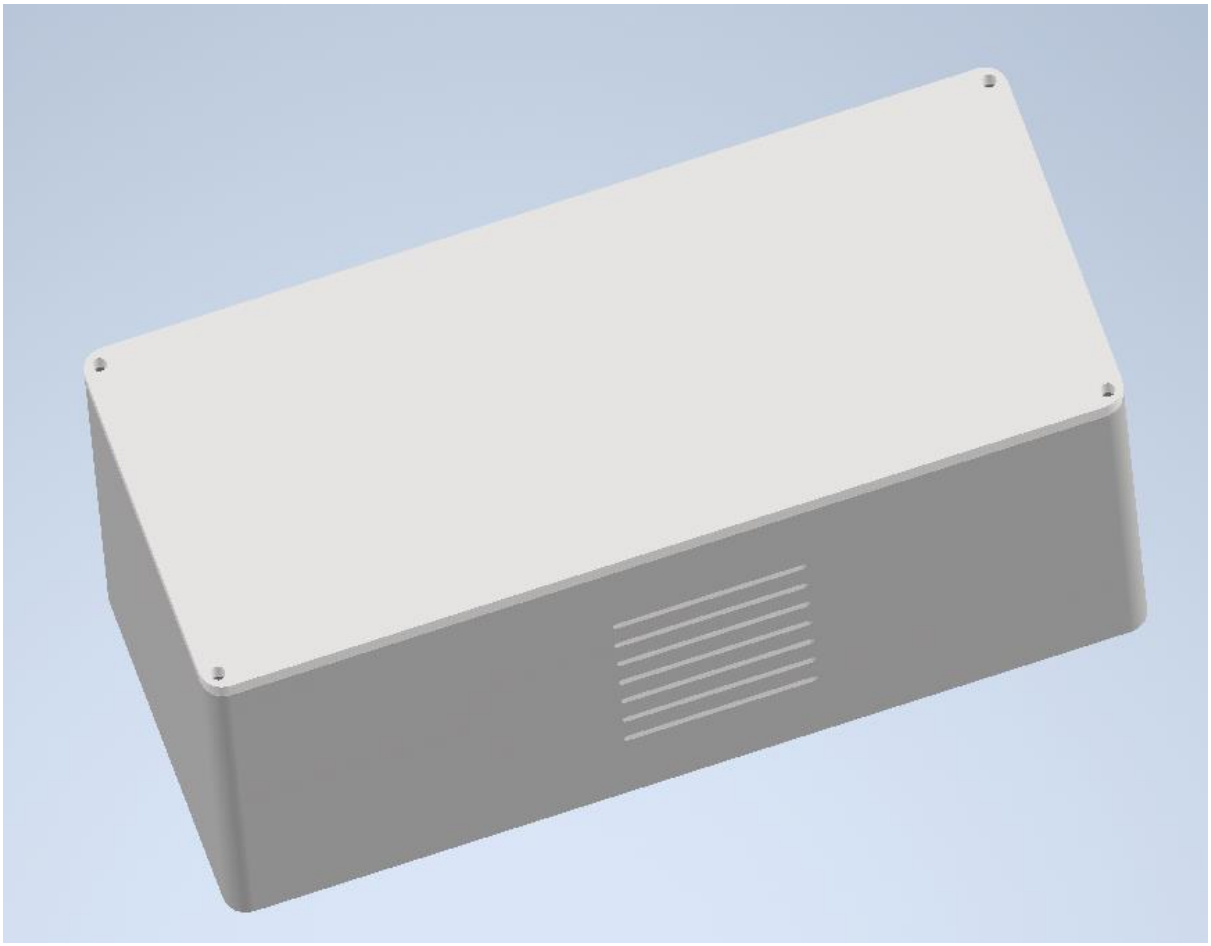


Figura 39-Modelo 3D de la caja que tendrá la PCB y los componentes de control de la PLACA2

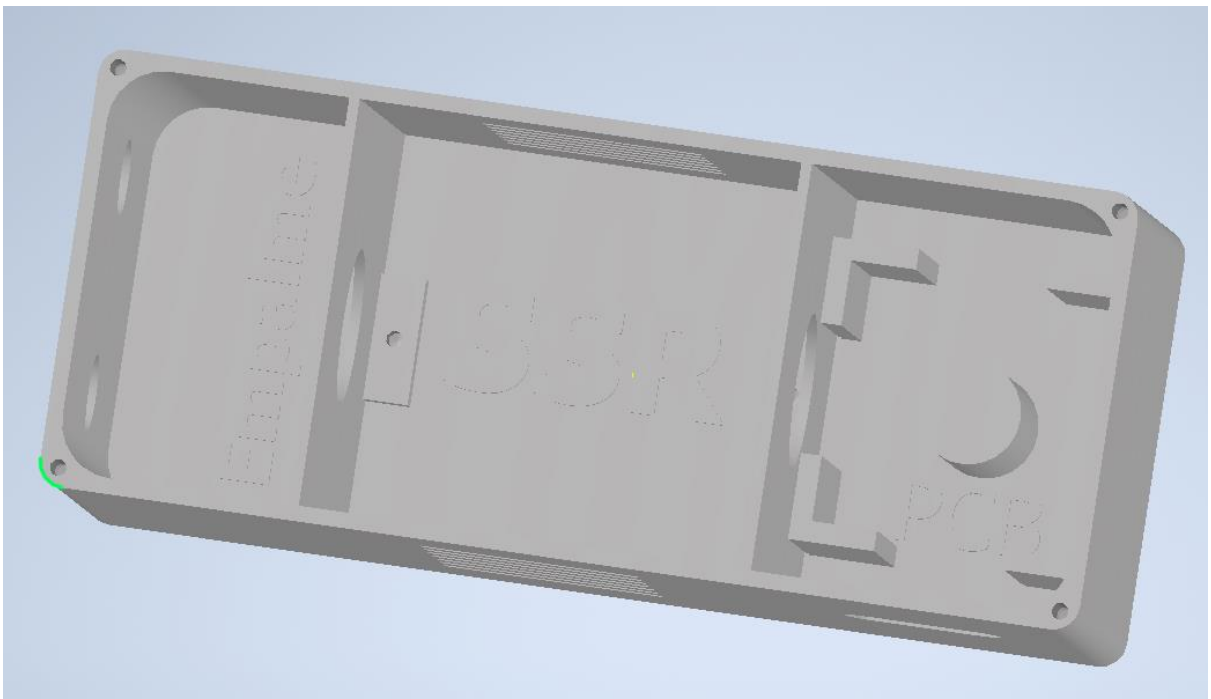


Figura 40-Modelo 3D de la caja que tendrá la PCB y los componentes de control de la PLACA2 sin tapa para poder ver el interior

4.3.15. PCB

Con el fin de integrar la mayor parte de los componentes electrónicos de la placa 2, se hace necesario el diseño de una PCB (Printed Circuit Board o placa de circuito impreso), que permitirá soldar y asegurar todos los componentes. No será necesario para la placa CYD ya que se trata de un shield y ya viene todo integrado.

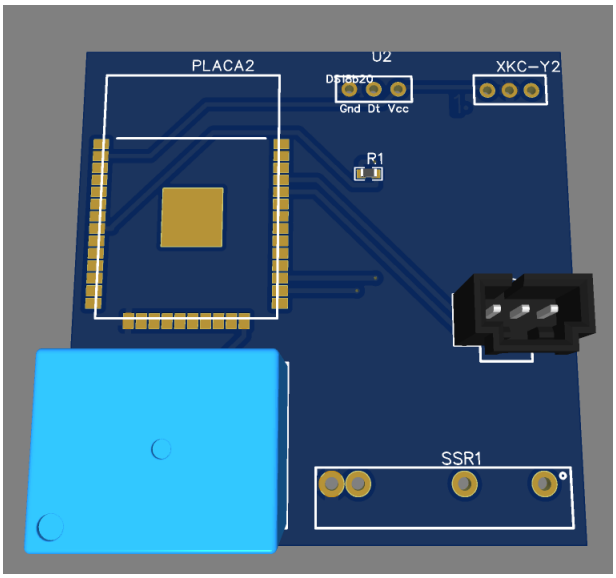


Figura 41-Modelo 3D de la PCB "Placa2"

4.4. PID

Según comentado anteriormente, los controladores PID (Proporcional-Integral-Derivativo) son un tipo de controlador utilizado en sistemas de control automático. Su simplicidad, robustez y efectividad los convierten en una herramienta ampliamente utilizada en prácticamente cualquier escenario que sea necesario el control de cualquier tipo de variable. Los ejemplos más comunes son temperaturas en hornos industriales, nivel de líquido en tanques e incluso posicionamiento de robots.

El controlador PID basa su funcionamiento en la medición de la diferencia entre la variable de referencia (valor deseado) y la variable medida (valor real del sistema. Esta diferencia, denominada error, es la que se utiliza para el cálculo de la acción de control mediante un bucle retroalimentado), ver figura 36. La acción de control se compone de tres términos:

1. **Acción proporcional (Kp)**: Es proporcional al error actual, es decir a mayor error mayor acción de control:

$$Acción\ proporcional = Kp \cdot error$$

2. **Acción integral (Ki)**: Es proporcional a la integral del error en el tiempo. Acumula el error a lo largo del tiempo, lo elimina el error estacionario.

$$Acción\ integral = Ki \cdot \int error(t)\, dt$$

3. **Acción derivativa (Kd)**: Es proporcional a la derivada del error en el tiempo. Anticipa cambios futuros en el error y ayuda a reducir el tiempo de estabilización.

$$Acción\ derivativa = Kd \cdot \frac{d(error)}{dt}$$

Con estos tres cálculos, ya se puede calcular la ecuación de salida del regulador que es de la forma:

$$Salida\ del\ controlado = (Kp \cdot error) + (Ki \cdot error \int \frac{error(t)}{dt}) + (Kd \cdot \frac{d(error)}{dt})$$

En la figura 36, se puede observar el esquema básico de un sistema de control, siendo:

- **r(t):** la referencia o valor de consigna.
- **e(t):** el error, obtenido entre la diferencia del valor real y el valor de consigna.
- **u(t):** la señal de control que proporciona el sistema a la planta-
- **y(t):** el valor real de lo que se está controlando (temperatura, velocidad, …).

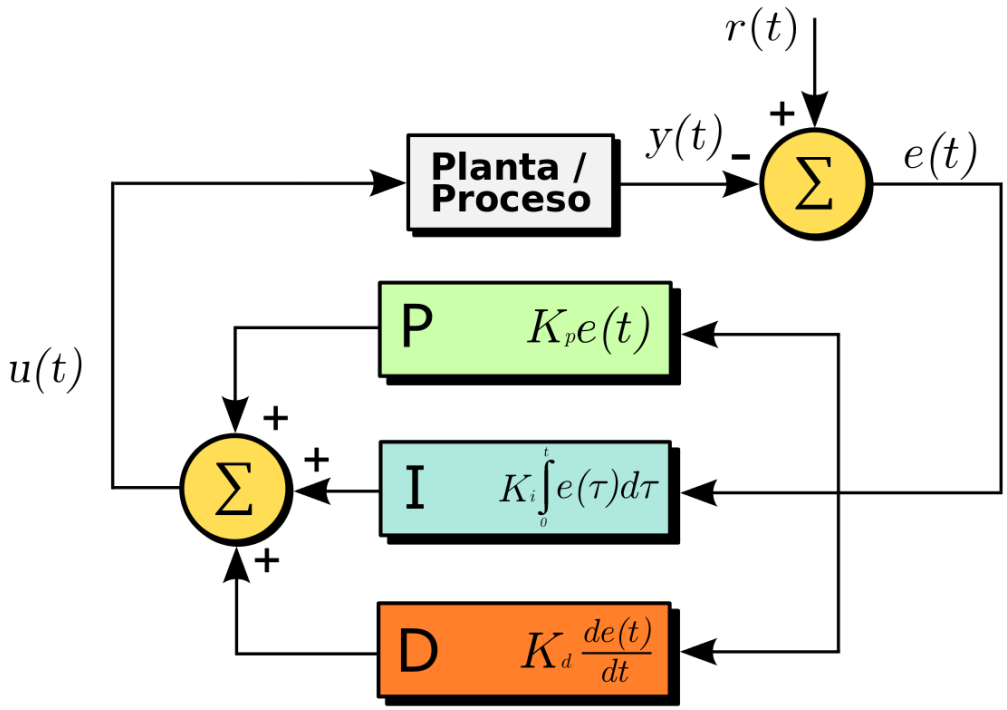


Figura 42-Esquema básico del modelo PID

El rendimiento de los controladores PID está relacionado con los valores de Kp, Ki y Kd. Cada uno de ellos es el responsable de alterar el comportamiento del sistema de distinta forma.

- **Kp:** afecta la rapidez de respuesta del controlador. Un Kp alto produce una respuesta más rápida, pero también puede aumentar la oscilación.
- **Ki:** elimina el error estacionario. Un Ki alto elimina el error más rápido, pero también puede provocar inestabilidad.
- **Kd:** reduce el tiempo de estabilización. Un Kd alto reduce el tiempo de estabilización, pero también puede aumentar la sensibilidad al ruido.

Cabe destacar que el valor de algún valor de K puede ser nulo sin que ello suponga que el regulador esté mal calculado.

Para el cálculo de los distintos parámetros, existen varios métodos, cada uno de ellos con sus pros y sus contras. A continuación, se van a nombrar una serie de ellos junto a una breve descripción:

- **Método de Ziegler-Nichols**: Se trata de un método empírico en el que no es necesario conocer ni las funciones de transferencia de la planta ni del regulador. Mediante una serie de experimentos definidos, es posible ajustar cada uno de los tres valores.
- **Método del lugar de las raíces**: Se trata de un método gráfico en el que si es necesario conocer las funciones de transferencia de la planta. Mediante una simulación del sistema en programas informáticos (tales como MATLAB) se van dibujando las gráficas del sistema hasta conseguir las características deseadas.

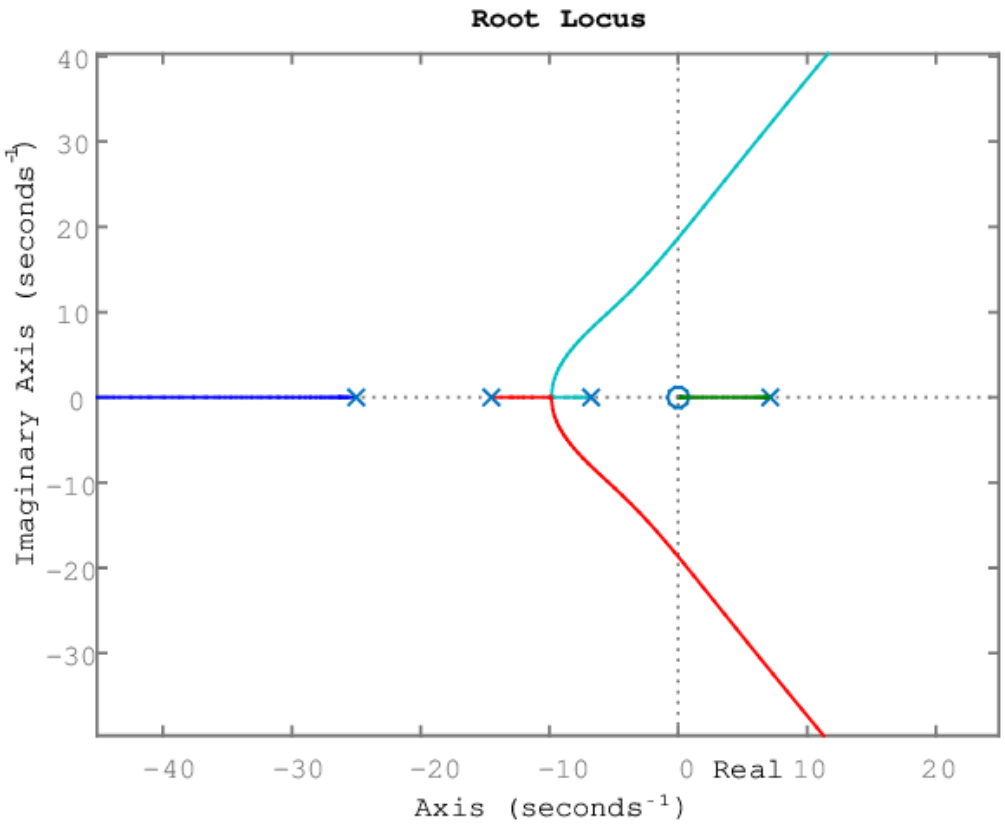


Figura 43-Método del lugar de las raíces en Matlab

- **Método de optimización**: Al igual que el método del lugar de las raíces, se trata de un método informático que requiere el conocimiento de las funciones de planta. Mediante distintas iteraciones se va optimizando el valor del pid.

4.4.1. CÁLCULO DE LA FUNCIÓN DE TRANSFERENCIA

Para realizar el cálculo de los valores Kp, Ki y Kd en este caso en concreto, se va a proceder por el método del lugar de las raíces.

El primer paso es realizar un modelo matemático del sistema (o planta) que será la función de transferencia. Para ello se tomarán los valores de los que se dispone son:

- Potencia del calefactor: 2000 W.
- Temperatura ambiente: 20 ºC.
- Volumen agua a calentar: 20 litros.
- Calor específico agua: 4186 J/kg ºC
- Radio base olla: 18 cm.
- Altura olla: 30 cm.

La primera ecuación que se plantea es la de balance de energía:

$$Q_e - Q_s = Q_T$$

Donde:

- Q_e es la energía aportada por la resistencia (2000 W).
- Q_s son las pérdidas de calor.
- Q_T es la energía almacenada.

El cálculo de la energía almacenada se realiza mediante la fórmula de la capacidad calorífica.

$$Q_T = m \cdot c \cdot \frac{dT}{dt}$$

Siendo:

- m: la masa del agua, en este caso 20 kg (1 l = 1 kg).
- c: el calor específico del agua (4186 J/kg ºC).
- dT/dt: la variación de temperatura respecto al tiempo.

El cálculo de las pérdidas de calor se modela de la siguiente manera:

$$Q_s = U \cdot A \cdot (T - Ta)$$

Siendo:

- U: el coeficiente de transferencia de calor.
- A: el área superficial de contacto entre el agua y el ambiente.
- T: temperatura del agua.
- Ta: Temperatura ambiente.

El siguiente paso es sustituir las expresiones para formar la ecuación diferencial y aplicar la transformada de Laplace.

$$2000 - U \cdot A \cdot (T - 20) = 20 \cdot 4186 \cdot \frac{dT}{dt}$$

$$\frac{2000}{s} - U \cdot A \cdot \left(T(s) - \frac{20}{s}\right) = 20 \cdot 4186 \cdot s \cdot T(s)$$

Conociendo que la función de transferencia es la variación de la temperatura del agua en función de la potencia del calefactor y simplificando, la función de transferencia es:

$$G(s) = \frac{T(s)}{P(s)} = \frac{K}{\tau \cdot s + 1}$$

Para:

$$K = \frac{1}{U \cdot A} \; ; \; \tau = \frac{20 \cdot 4186}{U \cdot A}$$

Se introducen los datos en Matlab, para el cálculo del PID.

```
>> % Datos del sistema
rho = 1000; % Densidad del agua (kg/m^3)
c = 4186; % Calor específico del agua (J/kg°C)
V = 20/1000; % Volumen de agua (m^3)
A_base = pi * 0.18^2; % Área de la base de la olla (m^2)
h_olla = 0.3; % Altura de la olla (m)
h_agua = V / A_base; % Altura del agua en la olla (m)
A_lateral = pi * 0.36 * h_agua; % Área lateral en contacto con el agua (m^2)
A_total = A_base + A_lateral; % Área total en contacto con el agua y el ambiente (m^2)
U = 1; % Coeficiente global de transferencia de calor (W/m^2°C), valor típico en este tipo de sistemas
% Cálculo de los parámetros de la función de transferencia
K = 1 / (U * A_total);
tau = rho * c * V / (U * A_total);
% Definición de la función de transferencia de la planta
s = tf('s');
G = K/(tau*s + 1);
>>
```

Figura 44-Datos del sistema en Matlab

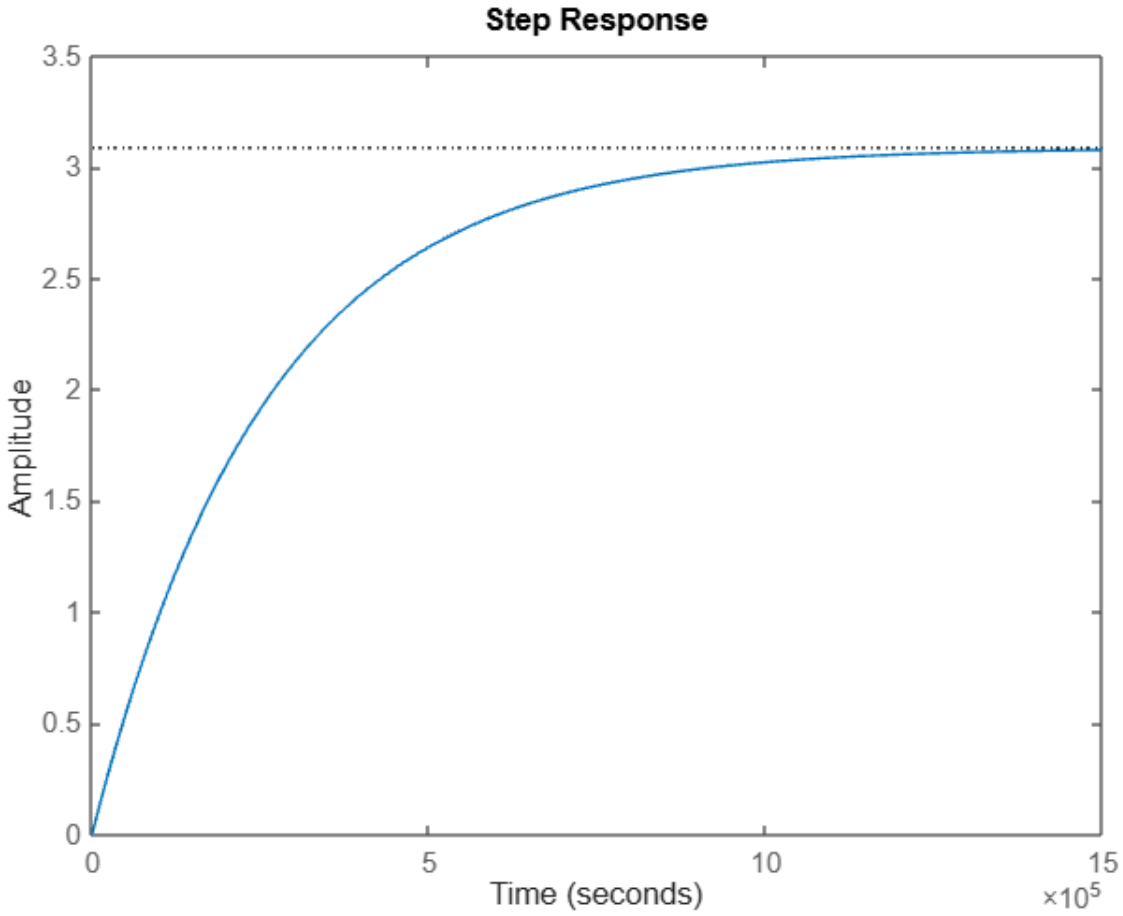


Figura 45-Respuesta ante escalón del sistema

Donde la función de transferencia es:

$$G = \frac{3{,}086}{2{,}548 \cdot 10^5 s + 1}$$

Que se trata de una función de primer grado con un cero en 3,086 y un polo en 254800.

4.4.2. CÁLCULO DEL PID

Debido a la posibilidad de poder montar el sistema y realizar pruebas experimentales, se prepara un experimento en el que se introducirá un escalón de entrada al sistema y se irá registrando la salida (temperatura) en intervalos de 30 segundos. Para ello, aprovechando las capacidades Wifi del ESP32 y la herramienta “ThingSpeak” de Matlab, se enviarán los datos recogidos a un servidor y luego se descargarán en un CSV para su posterior tratamiento y análisis. En los anexos se presenta el código utilizado.

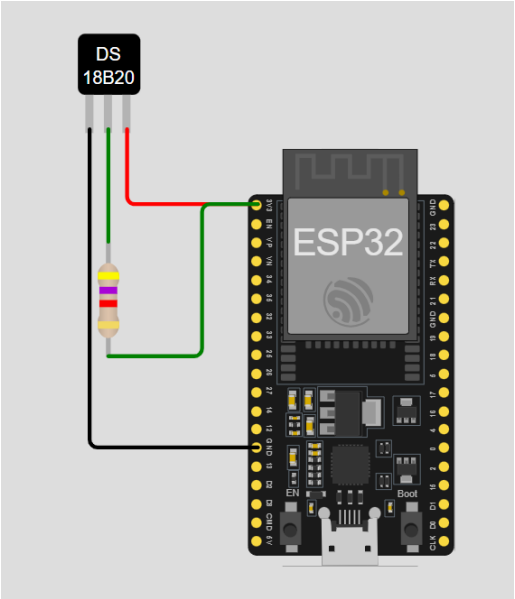


Figura 46-Esquema circuito toma de datos

Figura 47-Montaje experimental toma de datos. La pinza es para que el cable del sensor no esté en contacto directo con la olla.

Tras obtener los datos del servidor, se descargan y grafican para poder trabajar con ellos y obtener una representación visual.

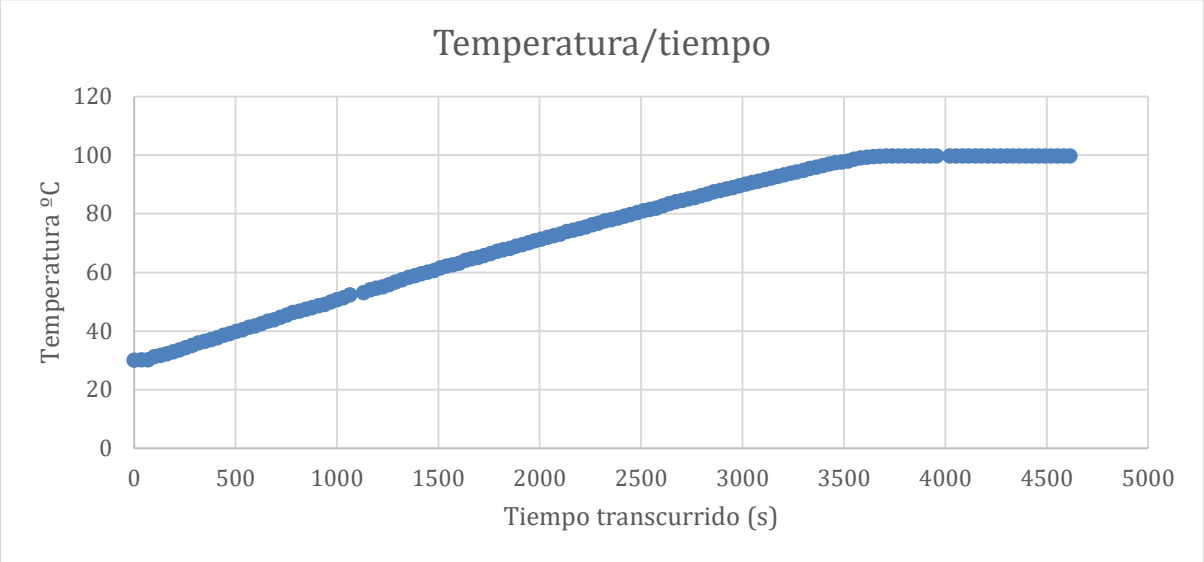


Figura 48-Gráfica obtenida mediante experimentación

Con los datos experimentales, utilizando la herramienta de Matlab pidTuner, se evalúan los datos.

```
t = readtable("/MATLAB Drive/tpo.csv");
t = table2array(t);  % Convertir la tabla 't' en una matriz numérica

y = readtable("/MATLAB Drive/temp.csv");
y = table2array(y);  % Convertir la tabla 'y' en una matriz numérica


datos = iddata(y,t,30);

pidTuner
```

Obteniendo el resultado de planta:

$$G(s) = \frac{104.32}{1 + 1399.3 \cdot s}$$

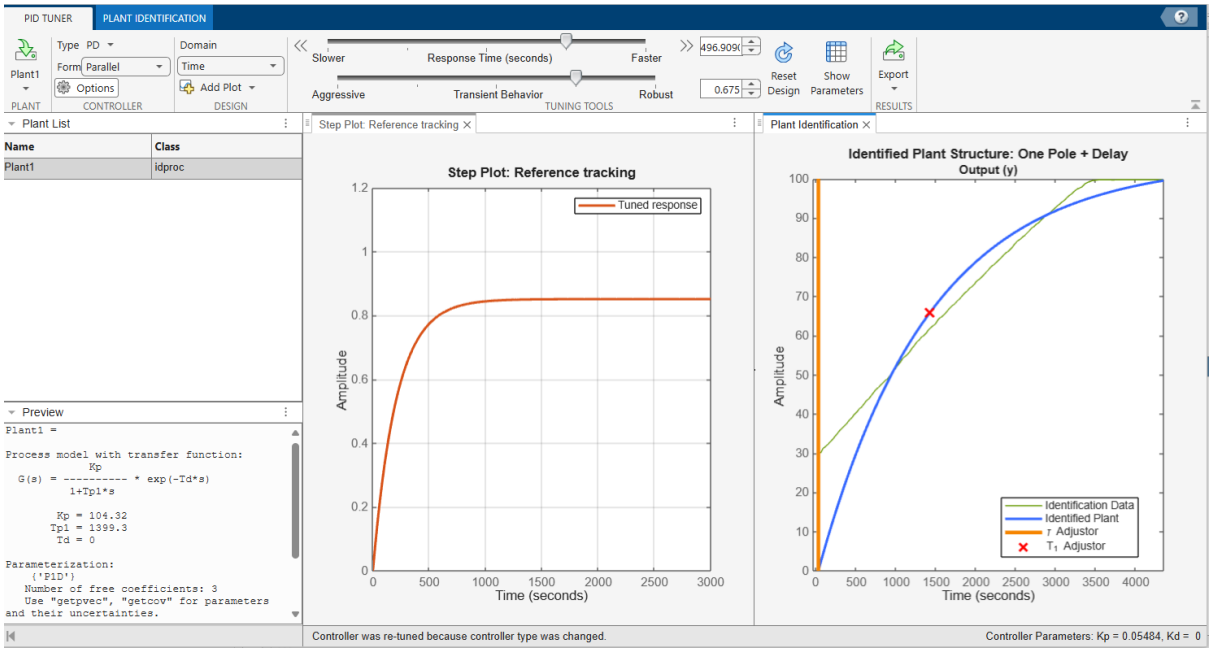


Figura 49-Ajuste de la planta mediante datos experimentales mediante pidTuner de Matlab

Que proporciona un regulador P con un valor de Kp = 0,68.

4.5. ARDUINO

En esta sección se va a realizar la explicación del montaje así como la configuración necesaria de cada uno de los elementos. En cada sección se irán desglosando de manera pormenorizada aquellos puntos que se crean necesarios para facilitar el montaje o la comprensión de los distintos puntos.

4.5.1. CONEXIONES Y PINOUT

Las conexiones del shield, o CYD, proporcionadas por el fabricante son las siguientes:

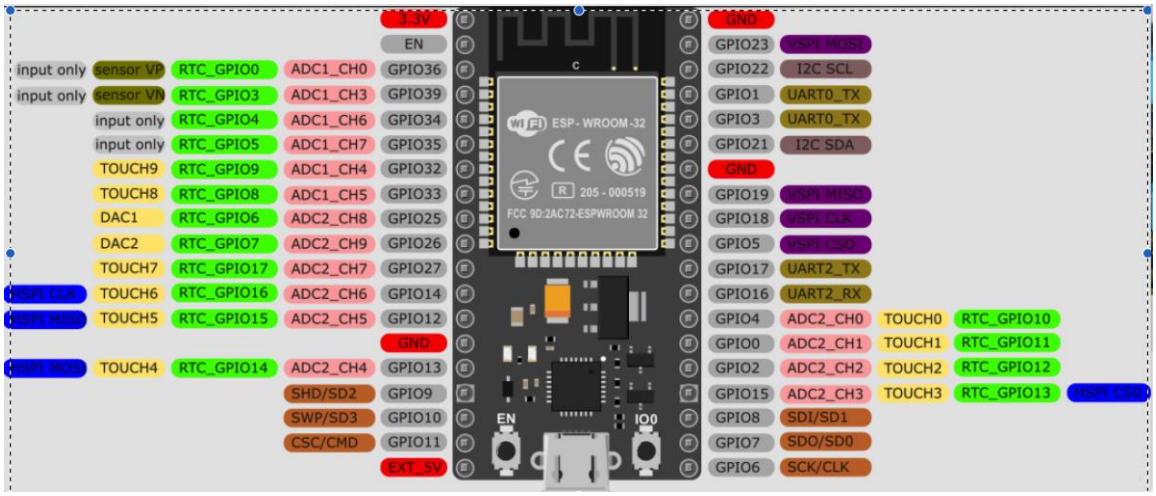


Figura 50-Conexiones ESP32

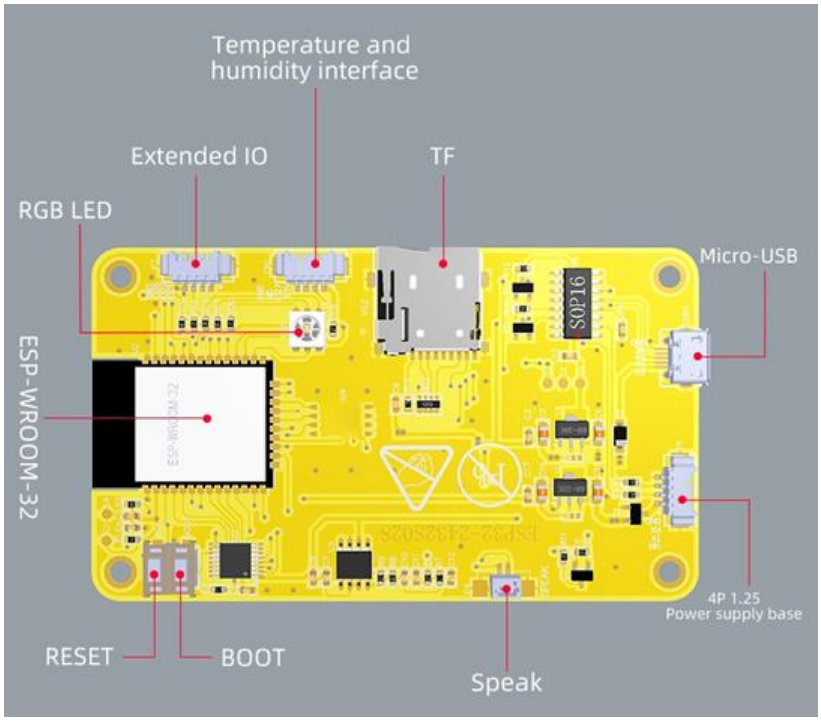


Figura 51-Vista trasera del "shield" CYD, proporcionado por el fabricante

La correlación de pines del shield al que se le denominará CYD, es la siguiente (PINOUT). Aunque no se van a utilizar todos durante el diseño, es importante conocer la configuración para así poder añadir más funcionalidades o utilizar pines alternativos en caso de que alguno falle.

PINES DE PANTALLA	
MISO(TFT_MISO)	GPIO 16
MOSI (TFT_MOSI)	GPIO 13
SCKL (TFT_SCLK)	GPIO 14
CS (TFT_CS)	GPIO 15
DC (TFT_DC)	GPIO 2
RST(TFT_RST)	GPIO 12
Backlight Pin	GPIO 21
PINES DE CONTROL DEL TÁCTIL	
IRQ (XPT2046_IRQ)	GPIO 36
MOSI (XPT2046_MOSI)	GPIO 32
MISO (XPT2046_MISO)	GPIO 39
CLK (XPT2046_CLK)	GPIO 25
CS (XPT2046_CS)	GPIO 33
PINES DE LED RGB TRASERO	
ROJO	GPIO 4
VERDE	GPIO 16
AZUL	GPIO 17
PINES DE ACCESO A TARJETA MICROSD	
MISO	GPIO 19
MOSI	GPIO 23
SCK	GPIO 18
CS	GPIO 5
PINES DEL SENSOR DE LUMINOSIDAD (LDR)	
LDR	GPIO 34
PINES DEL ALTAVOZ	
ALTAVOZ	GPIO 26
BOTONES DE ARRANQUE Y REINICIO	
ARRANQUE	GPIO 0
AMPLIACIÓN DE ENTRADAS Y SALIDAS	
GPIO	GPIO 35
GPIO	GPIO 22
GPIO	GPIO 21
GPIO	GPIO 27
PINES DE COMUNICACIÓN	
TX	GPIO 1
RX	GPIO 3

Tabla 2-Relación de pines CYD

Debido a que los pines de comunicación GPIO 1 y GPIO 3 no funcionan adecuadamente, para la comunicación entre placas se utilizarán los pines:

- GPIO 22 – RX
- GPIO 27 – TX

Las conexiones realizadas en el otro procesador ESP32, al que se denominará “Placa2” son las siguientes:

SENSORES	
SENSOR DS18B20 (TEMPERATURA)	GPIO 32
SENSOR XKC-Y28 (GRIFO ABIERTO)	GPIO 16
ACTUADORES	
ACTIVACIÓN RELÉ BOMBA	GPIO 2
ACTIVACIÓN SSR	GPIO 4
PINES DE COMUNICACIÓN	
TX	GPIO 27
RX	TPIO 14

Tabla 3-Relación de pines "Placa2"

El pin 22 (RX) de la CYD se conecta al pin 27 (TX) de la Placa2.
El pin 27 (TX) de la CYD se conecta al pin 14 (RX) de la Placa2.
Las masas de ambas placas han de estar conectadas.

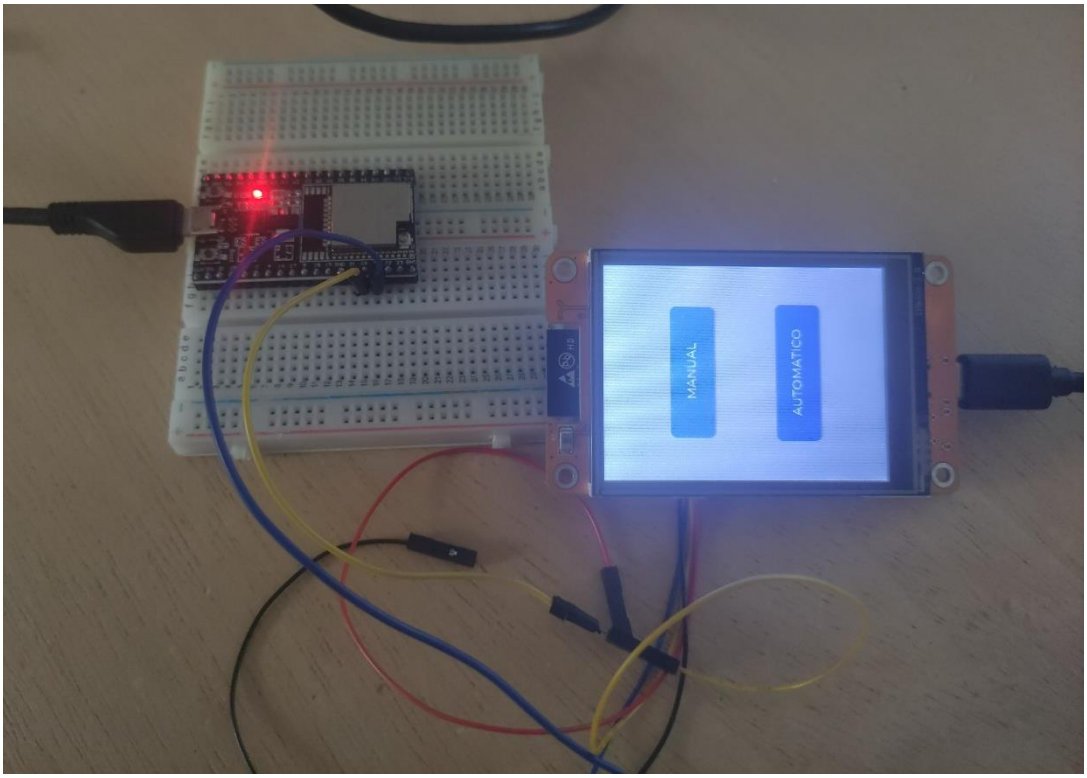


Figura 52-Montaje durante las pruebas de comunicación

Mediante el monitor en serie integrado en el IDE de Arduino se comprueba que la comunicación entre ambas placas es efectiva. Es importante que si se utiliza comunicación directa entre las placas, el conectar las masas ya que si no se conectan, pueden dar problemas.

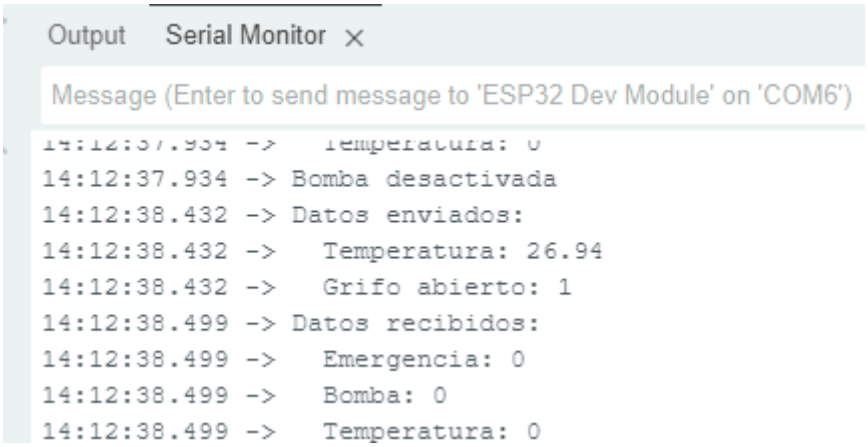


Figura 53-Puerto serie "Placa2" en el que se comprueba que se envían y reciben datos.

Figura 54-Monitor serie de CYD para comprobar la comunicación bidireccional

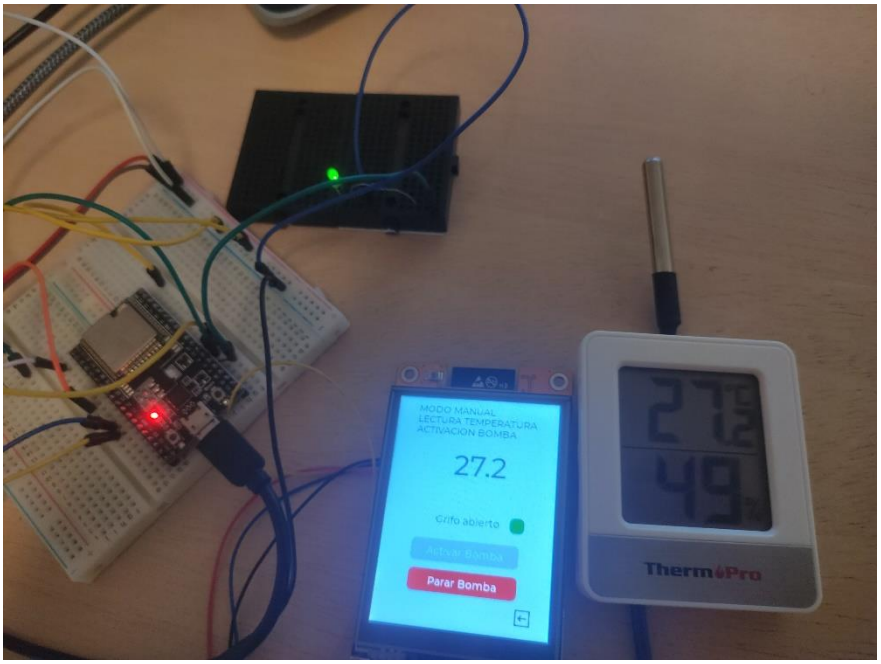


Figura 55-Comprobación de lectura del sensor de temperatura y su correcta visualización en pantalla. El led verde simboliza la activación de la bomba, debido a eso se observa que en la pantalla sólamente está habilitada la opción de "Parar bomba"

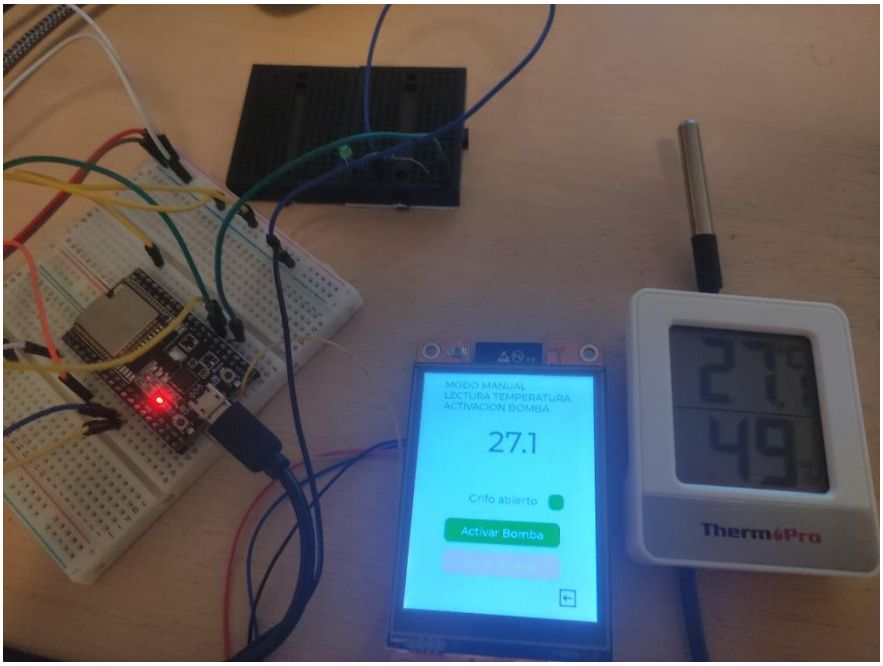


Figura 56-Comprobación de lectura de temperatura y comunicación entre placas. En esta ocasión el led aparece apagado ya que la bomba no está activa.

4.5.2. ESQUEMA

Los esquemas de conexión del shield son los siguientes, hay que tener en cuenta que vienen proporcionados por el fabricante.

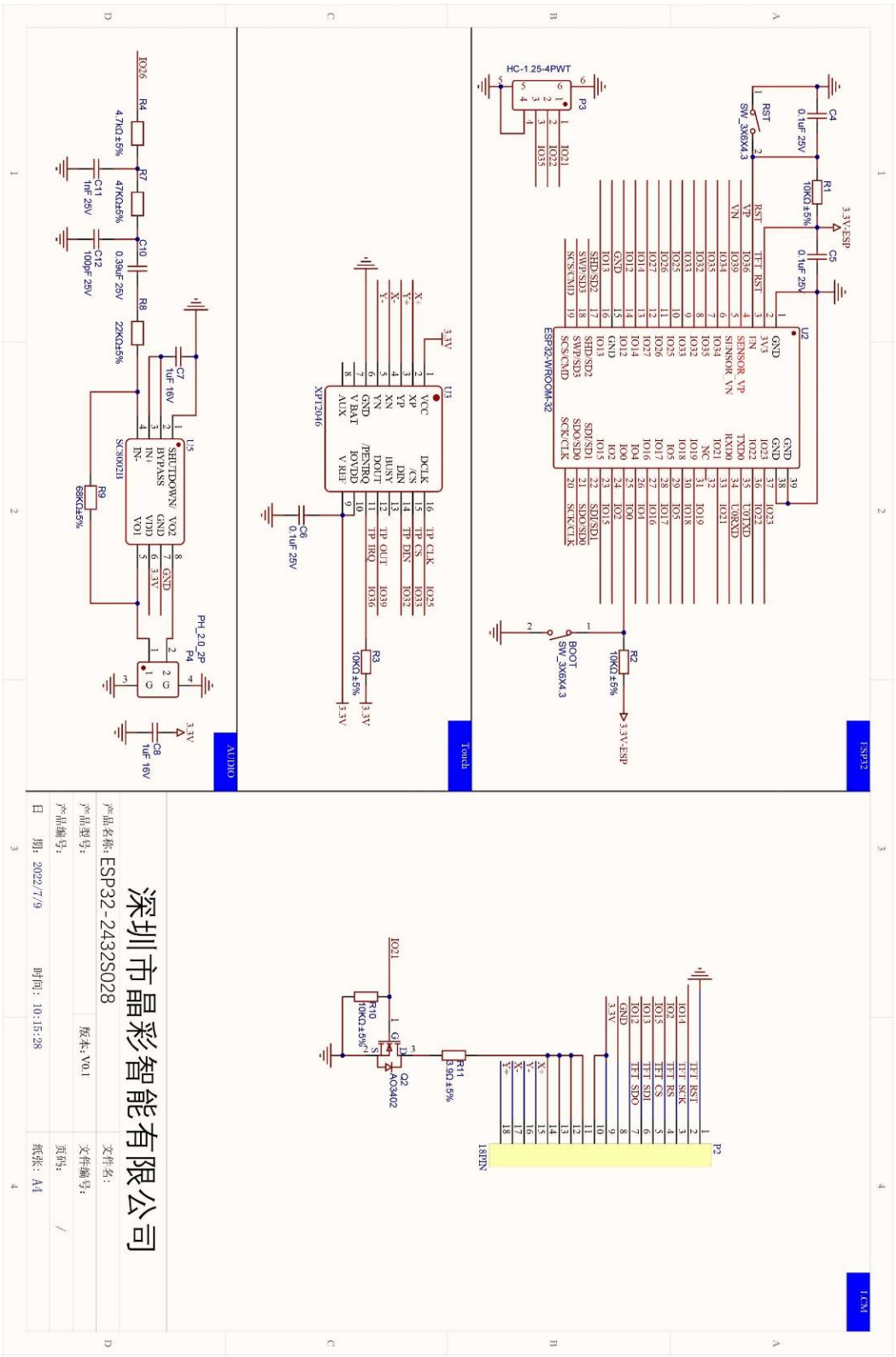


Figura 57-Esquema ESP32-2432S028-LCM proporcionado por el fabricante

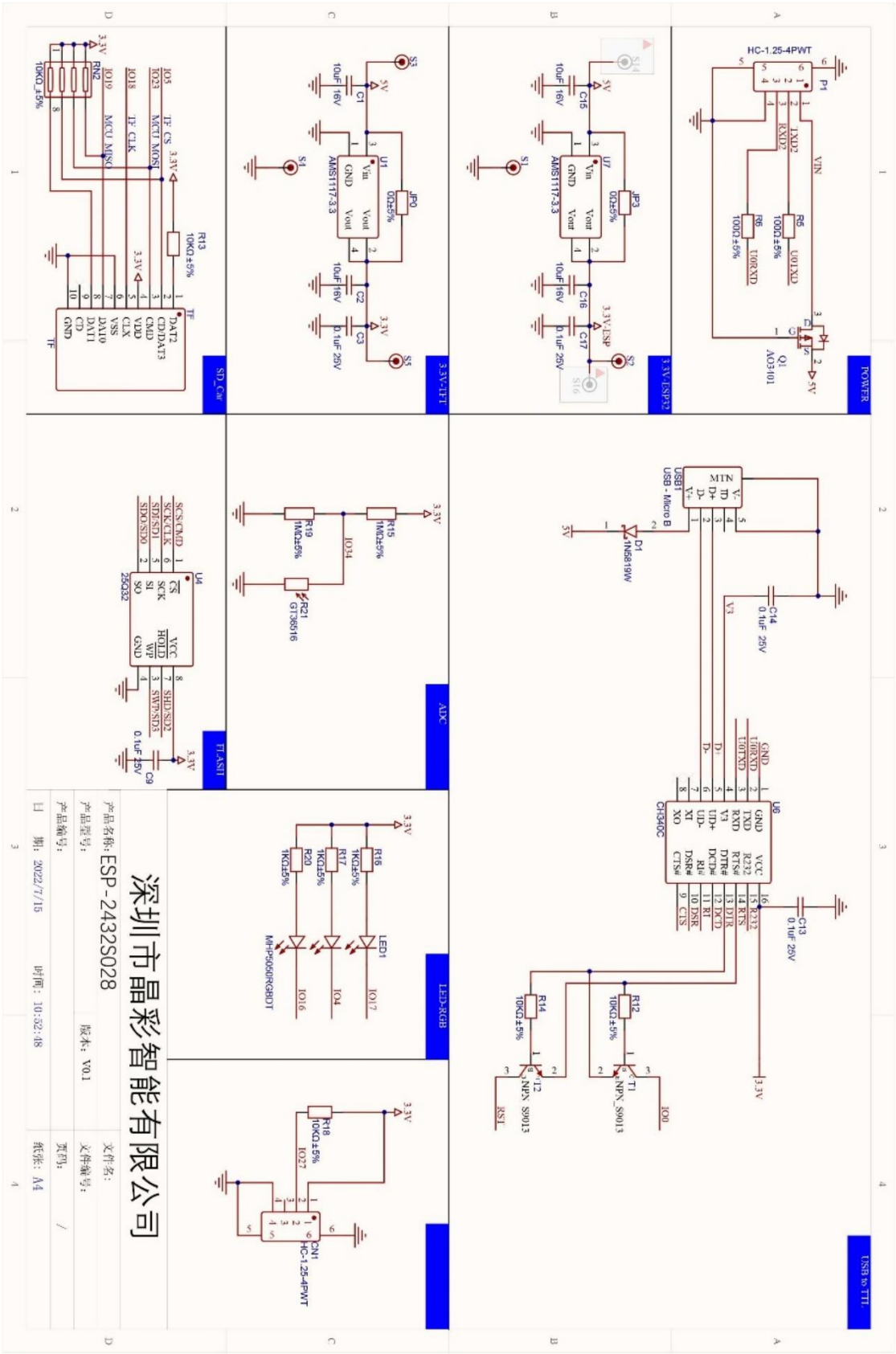


Figura 58-Esquema ESP32-2432S028-MCU proporcionado por el fabricante

4.5.3. CREACIÓN INTERFAZ GRÁFICA

Para la creación de la interfaz gráfica, se va a utilizar el programa “SquareLine Studio”, que aunque se trate de un programa comercial, ofrece la posibilidad de usar el programa con todas las funcionalidades durante 30 días y luego tras registrarse ofrece una licencia personal gratuita con ciertas limitaciones, pero perfecta para proyectos personales y no comerciales.

La elección principal de este software es que se apoya en la librería LVGL (**L**ight and **V**ersatile **G**raphics **L**ibrary), una librería diseñada especialmente para realizar interfaces gráficas en sistemas embebidos. Sus grandes cualidades son el bajo consumo de recursos, un rendimiento optimizado, es compatible con una alta gama de dispositivos y posee una gran biblioteca de componentes (widgets) preconstruidos, además de configuración avanzada.

Primeramente se empieza con la creación de un nuevo proyecto en Arduino, es importante definir las dimensiones correctas de la pantalla, la profundidad así, cómo la versión de la librería LVGL, en el momento de la creación del proyecto sólo estaba disponible la versión 8.3.11, pero tras la última actualización ya se puede utilizar la nueva versión 9.11). En caso de equivocación, siempre será posible modificar los valores de la pantalla, tanto desde el propio proyecto como desde los archivos de configuración de Arduino.

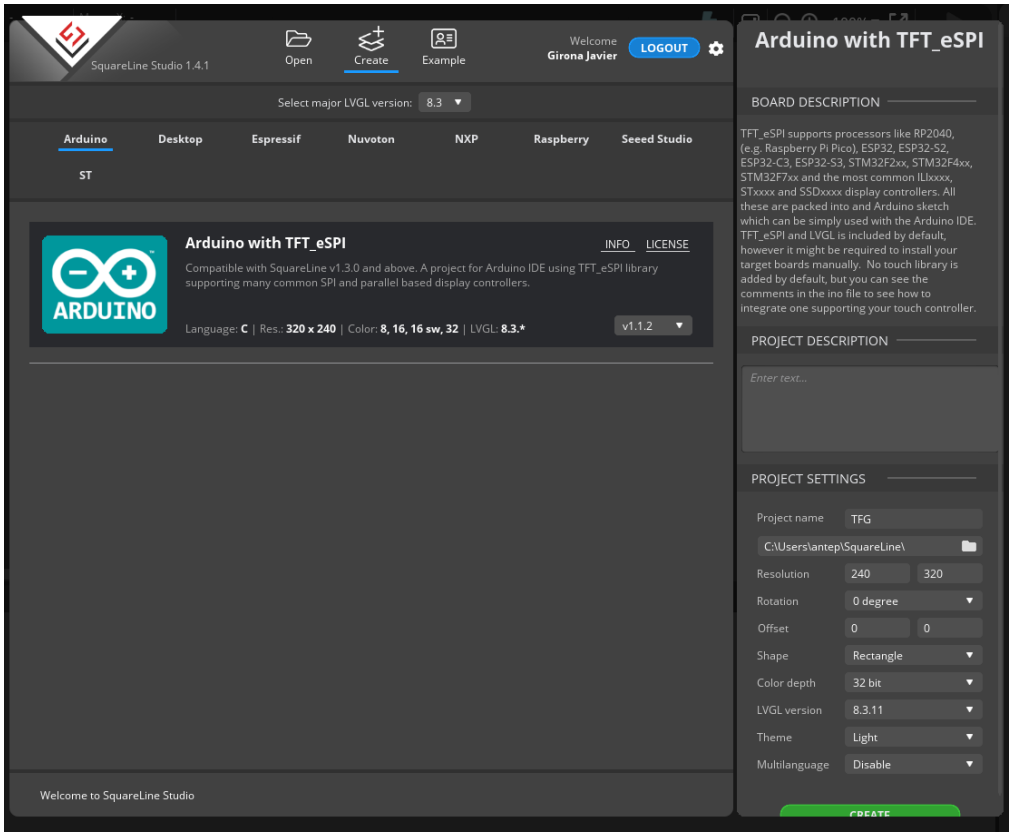


Figura 59-Creación de proyecto en Squareline Studio

Una vez creado el proyecto, se presenta la siguiente interfaz gráfica, una representación gráfica de la pantalla situado en el centro de la imagen, en ella se irán incluyendo los distintos elementos (widgets) con los que irá interactuando el usuario (situados

en la esquina inferior izquierda). El procedimiento es muy sencillo, ya que se trata de un “drag & drop”, es decir soltar y arrastrar.

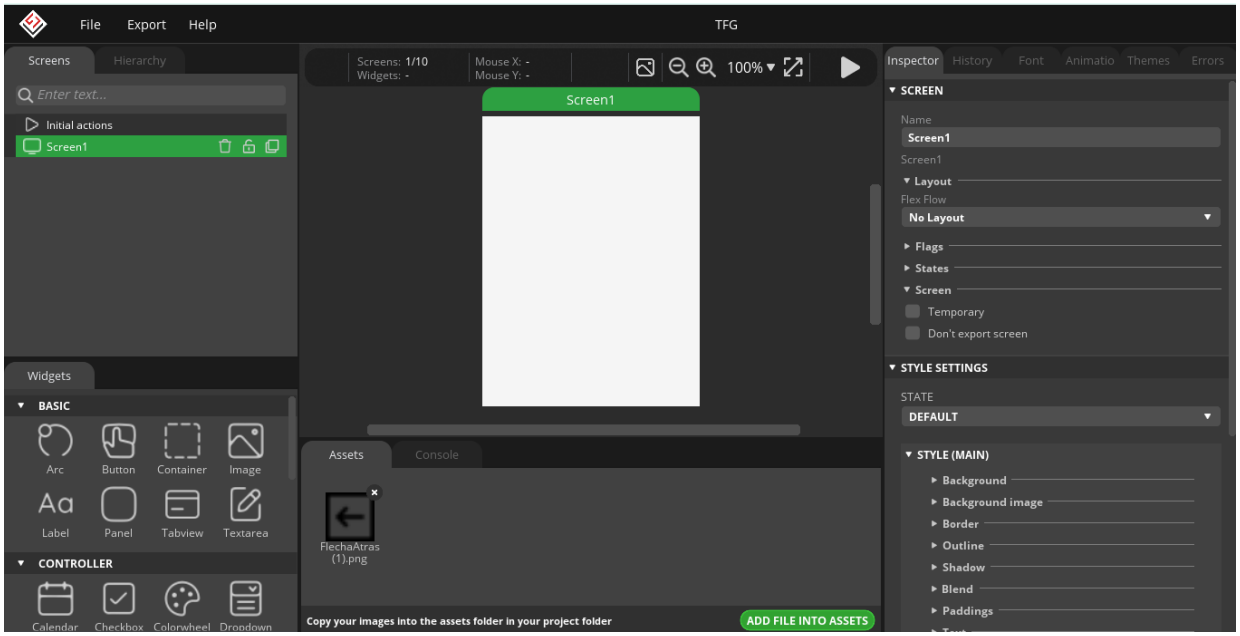


Figura 60-Pantalla de inicio de nuevo proyecto

Una vez incluido el primer elemento (en este caso un botón), en la parte derecha se aprecia que el inspector del elemento cambia para poder modificarlo según las necesidades del diseñador. En la parte izquierda dentro de la pestaña “Hierarchy”, aparece como elemento el botón “Button1” junto al símbolo del botón.

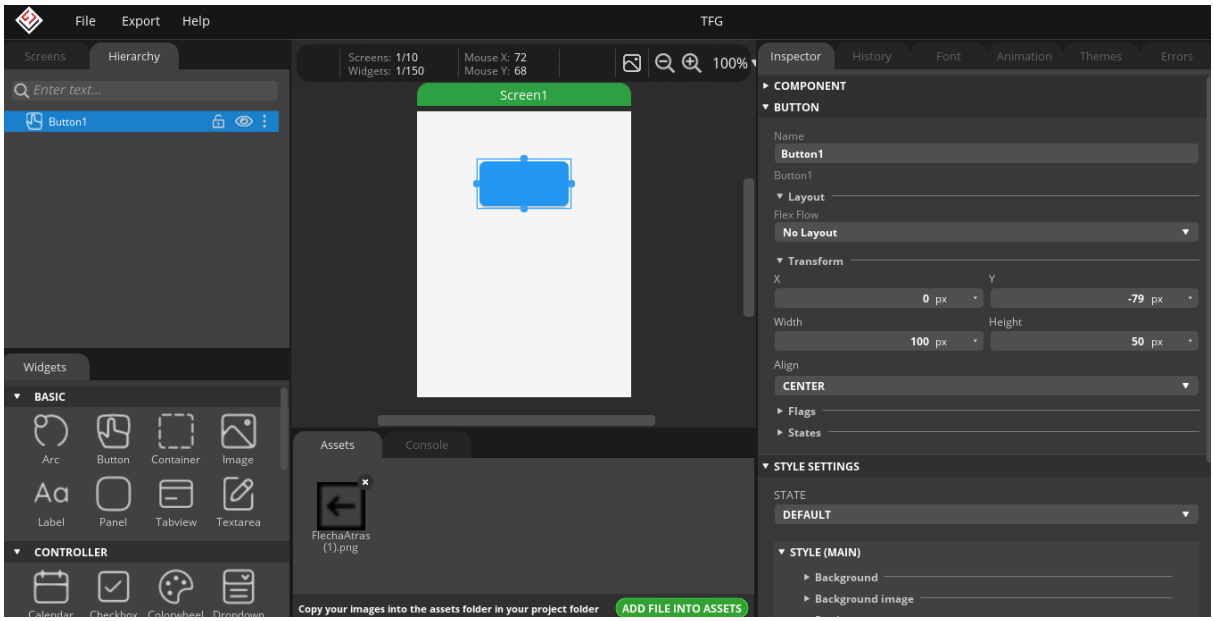


Figura 61-Insertando el primer elemento dentro de Squareline Studio

Para identificar el botón, se arrastra un elemento tipo “Label” y se arrastra dentro del botón, creando así una estructura en la que cualquier movimiento que se realice al botón se aplicará también a la etiqueta. También es importante el renombrar los nombres para que

luego faciliten su identificación tanto durante el resto del diseño de la interfaz gráfica cómo cuando se realice la programación de la lógica del programa. Hay que recordar que este programa solamente sirve para la creación de la interfaz gráfica (es decir el frontend), la lógica del programa ha de ser programada. Durante la programación el nombre de todas las variables de la interfaz gráfica cambiará a ui_[NombreSquareline] así la etiqueta “LManual” será renombrada a “ui_LManual”.

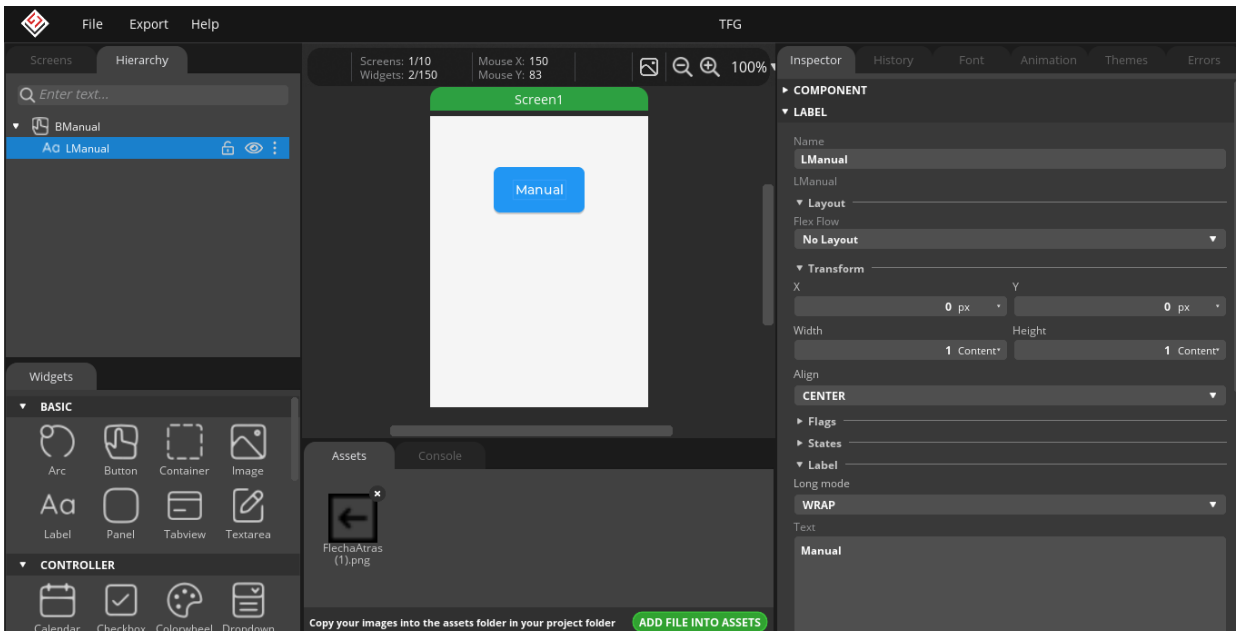


Figura 62-Cambio en la jerarquía y actualización de nombres

El siguiente punto es el de asignar eventos que se activaran según unas acciones realizadas por el usuario, la más común es la de pulsar un botón, seleccionar de un desplegable, … En este caso se selecciona el botón y en el inspector de la parte de elementos, en la sección de “EVENTS”, se pulsa el botón de “ADD EVENT”.

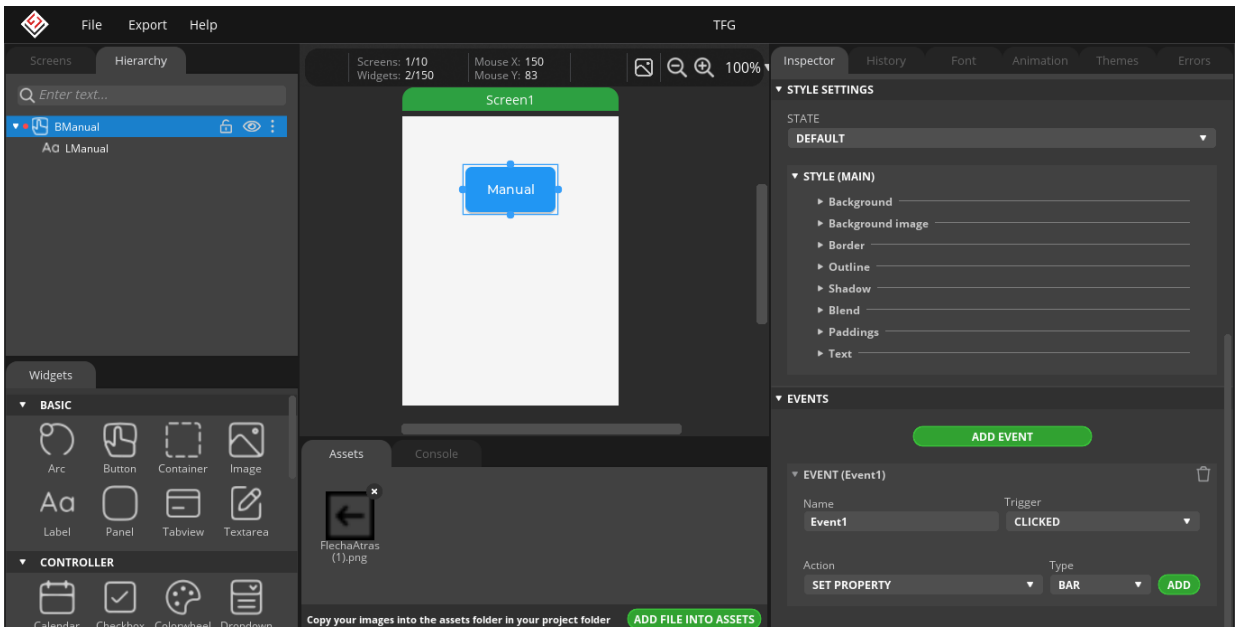


Figura 63-Creación de eventos

Primeramente se nombra el evento, es aconsejable que el nombre sea descriptivo para luego poder identificarlo fácilmente. Se elige un “trigger” o disparador que será el responsable de activar el evento, normalmente suele ser pulsar el botón, aunque también puede accionarse al cambiar el valor, seleccionar, … El evento puede actuar sólo a nivel de la interfaz gráfica, cambiando los estados de los elementos o llamar a las funciones dentro de la lógica del programa.

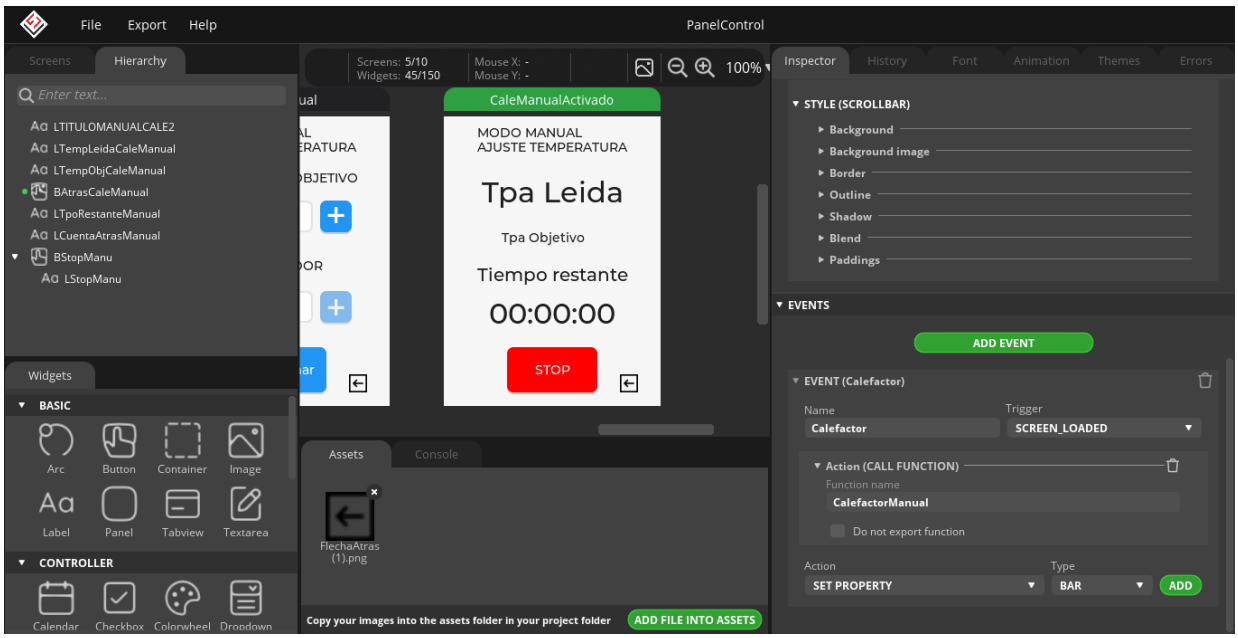


Figura 64-Evento que llama a una función al cargarse una pantalla

En este caso, cuando la pantalla “CaleManualActivado” se carga, el evento “Calefactor” llamará a la función CalefactorManual. A posteriori habrá que definir en el código del programa todas las funciones que se vayan creando. En caso de que no se definan, el código no compilará.

El programa tiene también la posibilidad de ir comprobando los cambios que se van realizando en una simulación, esta opción permite ir confirmando el comportamiento que tendrá la interfaz diseñada antes de cargarla en la pantalla y permite un ahorro de tiempo bastante considerable. Es importante destacar que aquellas variables que se lean directamente desde los sensores no se actualizarán, al igual que aquellas funciones que se empleen posteriormente en el código. Para acceder a esta función basta con presionar el botón con el símbolo de reproducir al lado del inspector de elementos.

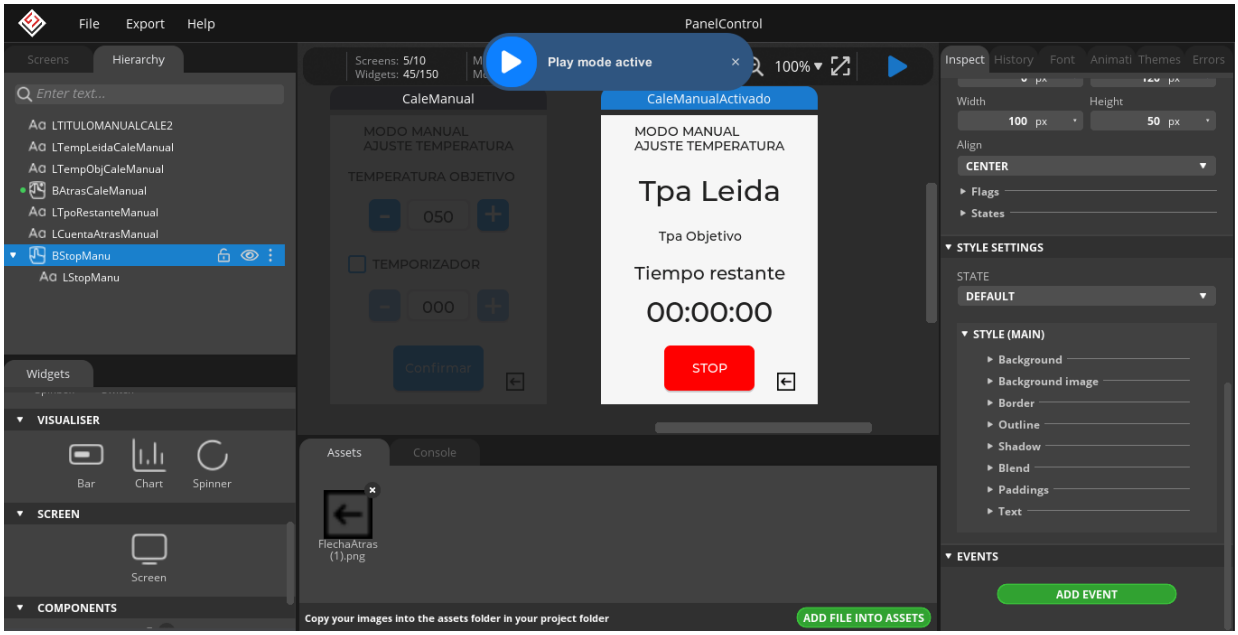


Figura 65- Modo simulación activado

Una vez se tenga la estructura básica del proyecto, habrá que exportarlo, para ello simplemente desde el menú superior se selecciona “Export -> Create Template Project”.

Según se vaya actualizando la interfaz gráfica, será necesario volver a exportar a la carpeta del proyecto los cambios realizados, en esta ocasión en vez de utilizar la opción de “Create Template Project”, se utilizará la opción de “Export UI Files”, que solamente añadirá los cambios realizados a la librería “ui” dentro de la carpeta “libraries”. Esta opción permite que se mantengan todos los cambios realizados en el fichero “ui.ino” así como las configuraciones específicas del proyecto.

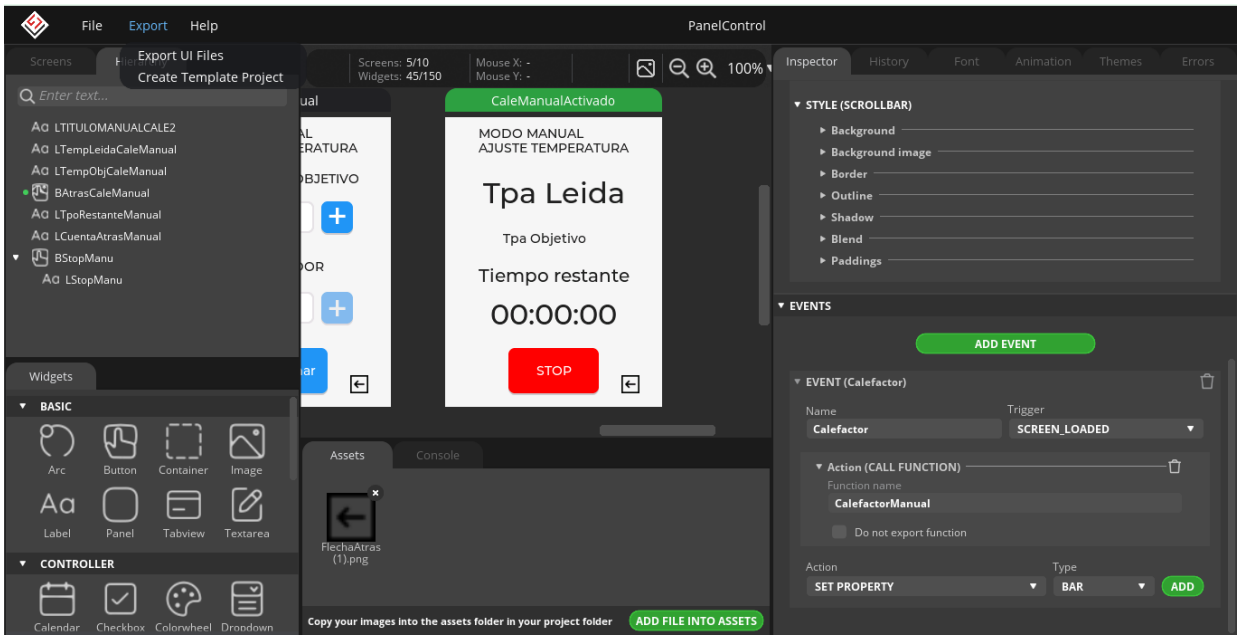


Figura 66-Exportar proyecto

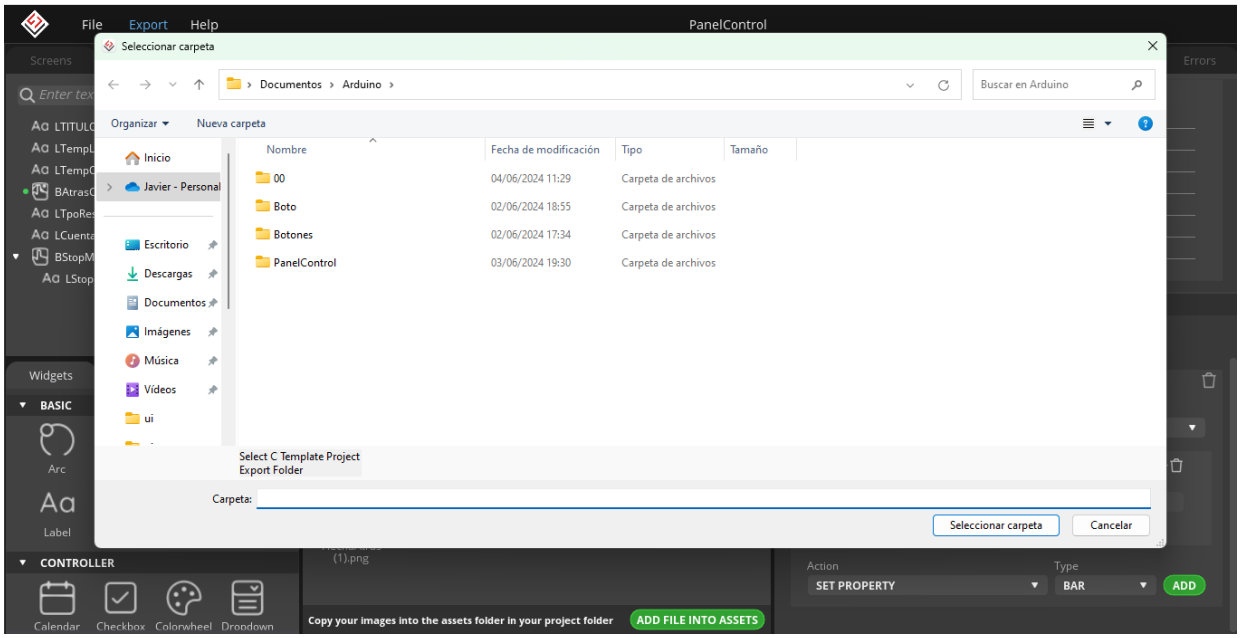


Figura 67-Exportar el proyecto

Una vez el proyecto esté exportado, creará una carpeta en el directorio seleccionado, dentro habrá otras 2 subcarpetas “ui” y “libraries” y un archivo “README.MD” con la explicación de la exportación. El código de Arduino estará dentro de la carpeta “ui” con el nombre de archivo “ui.ino”. En este fichero será donde se programe la lógica del programa.

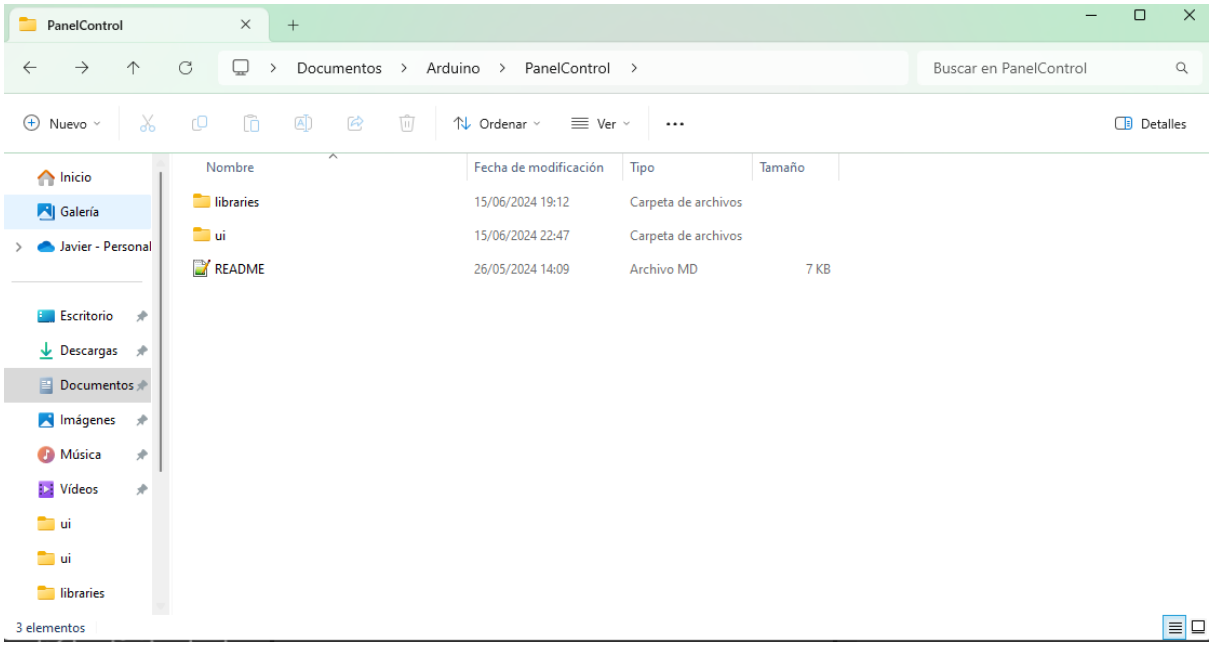


Figura 68-Carpeta creada tras la exportación

Al crear funciones dentro de Squareline Studio, este las incluirá en el fichero “ui_events.H”, dentro de la ruta …/[NombreDelProyecto]/libraries/ui/src”, es importante recordar que habrá que definirlas.

Otro punto importante es que el programa se apoya en la librería “TFT_eSPI” para el correcto funcionamiento de la pantalla táctil. Tal y como aparecen en las instrucciones del archivo “README”, hay que configurar los datos de la pantalla dentro del archivo “User_Setup.H”. En el caso de la placa utilizada el archivo de configuración se incluye en los anexos.

En la documentación de la aplicación, así como en los foros se encuentra la descripción y el funcionamiento de los distintos componentes así como buenas prácticas y dudas resueltas por la comunidad.

4.5.4. CÓDIGO

Todo el sistema ha sido implementado en Arduino por su amplia comunidad y por su accesibilidad. En las siguientes líneas se va a dar una explicación de cómo está escrito y estructurado el código, así como aquellos ajustes específicos que ha habido que hacer. Hay que tener en cuenta que dependiendo de las conexiones y del hardware utilizado, estas variarán. La explicación se divide en 2 partes, en la primera se explicará el código de la CYB y en la segunda parte, el código de la Placa2.

El primer punto es la explicación de las librerías, ya que estas son la base sobre las que se va a trabajar.

```
#include <lvgl.h>
#include <TFT_eSPI.h>
#include <ui.h>
#include <XPT2046_Touchscreen.h>
#include <SPI.h>
#include <HardwareSerial.h>
#include <ArduinoJson.h>
```

- Lvgl.h – Librería gráfica que permite la creación de la interfaz de usuario. Es necesario configurarla en el archivo lv_conf.h
- TFT_eSPI.h – Permite la visualización a través de la pantalla TFT. Requiere configurarla en el archivo User_Setup.h
- ui.h – Librería proveniente de Squareline Studio, contiene todas las declaraciones y objetos que se han diseñado.
- XPT2046_Touchscreen.h – Necesaria para que la pantalla táctil tenga capacidades táctiles.
- SPI.h – Necesaria para la visualización en la pantalla táctil.
- HardwareSerial.h – Permite definir puertos GPIO como puertos de comunicación, necesario para la comunicación entre placas.
- ArduinoJson.h – Facilita la comunicación entre placas al enviar todos los datos en un archivo Json.

El siguiente punto es el prototipado de funciones que ha sido necesario crear para la lógica del programa. Son aquellas funciones que se llaman directamente desde el código, no desde la interfaz de usuario. Las funciones están divididas en 3 categorías, de comunicación, de tiempo y de interacción con el usuario La razón por la que se han definido aquí las funciones es que al crear librerías personalizadas Arduino daba problemas.

```
//Prototipos de funciones no incluidas en el ui
//Funciones de comunicacion entre placas
void enviarDatos(bool emergencia, bool bomba, int temperaturaEnviada);
void recibirDatos(void);
//funciones de tiempo
void iniciar_cuenta_atras(lv_obj_t * label, lv_obj_t * spinbox, int minutos);
// iniciar cuenta atras manual
void timer_callback_manual(lv_timer_t * timer_manual);
void timer_callback_maceracion(lv_timer_t * timer_maceracion);
void timer_callback_ebullicion(lv_timer_t * timer_ebullicion);
//funciones interaccion con usuario
void Alarma(void); // Prototipo de la función Alarma(), para que suene el
buzzer
void reiniciarSpinboxesTiempo(void); //puesta a 0 de todos los spinboxes, al
acabar los procesos o después de la seta
void actualizarLabelsTemperatura(void);
```

- enviarDatos – Envía los datos a la Placa2 en formato Json.
- recibirDatos – Recibe los datos de la Placa2 en formato Json.
- Iniciar_cuenta_atras – Inicia una cuenta atrás proveniente de la interfaz de usuario.
- Timer_callback_manual – Resta 1 segundo del temporizador que esté activo, actualiza el texto de la iu y se sale, es imprescindible ya que si no bloquea la pantalla y el sistema. Se usa en la cuenta atrás manual.
- Timer_callback_maceracion – Misma función que el resto de callbacks, pero para la maceración.
- Timer_callback_ebullicion – Misma función que el resto de callbacks, pero para la ebullición
- Alarma – Crea un aviso sonoro mediante un zumbador
- reiniciarSpinboxesTiempo – Restablece el valor de los “spinbox” de la iu a cero.
- actualizarLabelsTemperatura – Actualiza el valor de las etiquetas de temperatura con los datos recibidos de la Placa2.

A continuación se definen los parámetros que servirán para poder hacer uso de la pantalla táctil. Estos parámetros vienen marcados por el hardware como el ancho y alto de la pantalla o los pines utilizados para la capacidad táctil, estos se pueden consultar en el apartado “pinout”. También se crean los timer que permiten realizar funciones de tiempo sin bloquear la pantalla.

```
/*Change to your screen resolution*/
static const uint16_t screenWidth = 240;
```

```
static const uint16_t screenHeight = 320;

static lv_disp_draw_buf_t draw_buf;
static lv_color_t buf[ screenWidth * screenHeight / 10 ];

TFT_eSPI tft = TFT_eSPI(screenWidth, screenHeight); /* TFT instance */

// Touchscreen pins
#define XPT2046_IRQ 36 // T_IRQ
#define XPT2046_MOSI 32 // T_DIN
#define XPT2046_MISO 39 // T_OUT
#define XPT2046_CLK 25 // T_CLK
#define XPT2046_CS 33 // T_CS

SPIClass tsSPI = SPIClass(VSPI);
XPT2046_Touchscreen touchscreen(XPT2046_CS, XPT2046_IRQ);

lv_timer_t * timer_manual; // Define timer_manual para el modo manual
lv_timer_t * timer_maceracion; // Temporizador para la maceración
lv_timer_t * timer_ebullicion; // Temporizador para la ebullición
```

A continuación se definen los pines que se van a conectar manualmente, 2 para comunicación ya que por algún motivo los pines RX y TX dan problemas y otro para el zumbado (buzzer) que emitirá los avisos sonoros.

```
//DEFINICION DE PINES LIBRES///
// Comunicación entre placas
#define RX_PIN 22       // Pin RX (GPIO 22)
#define TX_PIN 27       // Pin TX (GPIO 27)
//El pin de la alarma
#define PIN_BUZZER 26 // Pin GPIO al que está conectado el buzzer
```

El siguiente punto es la definición de variables globales. Estas variables se utilizan en varias funciones y se van actualizando y consultando en cada ciclo del programa. Cabe destacar la importancia de inicializarlas a 0 para evitar comportamientos no deseados.

```
/DEFINICION DE VARIABLES GLOBALES///
//variables globales para enviar, se van a usar en varias funciones
bool bomba = 0; // Variable para enviar si se activa la bomba o no
int temperaturaEnviada = 0; //Variable para enviar la temperatura manual
bool emergencia = 0;

//variables globales para recibir
float temperaturaRecibida = 0.0;
bool grifoAbierto = 0;

//Para los procesos
int temperaturasAuto[4] = {0,0,0,0};
```

```
int tiempoMaceracion[4] = {0,0,0,0};
int tiempoAvisoEbull[4] = {0,0,0,0};
bool bombaMaceracion[4] = {0,0,0,0};

// Variables para la cuenta atrás
static int segundos_restantes = 0; //Segundos restantes manual
static int segundosRestantesMaceracion = 0; // Segundos restantes para la
maceración
static int segundosRestantesEbullicion = 0; // Segundos restantes para la
ebullición
static int segundosRestantesPeriodoEbullicion = 0; // Nuevo contador para el
periodo en curso
int tiempoTotalMaceracion = 0;
bool maceradofin=0;
int indiceMaceracion = 0;
int indiceEbullicion = 0;
```

Como el sistema es capaz de almacenar hasta 4 temperaturas distintas y 4 tiempos distintos, estos se almacenan en vectores.

El siguiente punto conflictivo, es la lectura de la pulsación en la pantalla táctil, para ello hay que calibrarla. Esta acción hay que realizarla antes de la programación de la iu ya que si la pantalla no está bien calibrada se registrarán pulsaciones incorrectas.

```
/*Read the touchpad*/
void my_touchpad_read( lv_indev_drv_t * indev_driver, lv_indev_data_t * data )
{
    uint16_t touchX = 0, touchY = 0;
    TS_Point p = touchscreen.getPoint(); //yo
    /*Serial.print("Raw X: "); Serial.print(p.x);
    Serial.print(", Raw Y: "); Serial.print(p.y);
    Serial.print(", Pressure: "); Serial.println(p.z);*/

    data->point.x = map(p.x, 271, 3880, 0, screenWidth); //valores sacados
mediante calibración
    data->point.y = map(p.y, 183, 3796, 0, screenHeight); //valores sacados
mediante calibración

    if( p.z>60 ) //se está apretando la pantalla.
    {
        data->state = LV_INDEV_STATE_PR; //Pressed
        /*Serial.print("Raw X: "); Serial.print(p.x);
        Serial.print(", Raw Y: "); Serial.print(p.y);
        Serial.print(", Pressure: "); Serial.println(p.z);*/
    }
    else
    {
        data->state = LV_INDEV_STATE_REL; //Released
```

```
    }
}
```

Se puede observar que al pasar los datos mediante la función map, se aprecian 2 valores, estos son los valores “crudos” mínimos y máximos de cada eje que recoge la pantalla. Si hubiese problemas de calibrado solamente hay que descomentar las líneas de serial.print, pulsar en las 4 esquinas y actualizar los valores.

El siguiente apartado del código es el setup, donde se van inicializando los diversos componentes.

```
void setup(){
    Serial.begin( 115200 ); /* prepare for possible serial debug */

    String LVGL_Arduino = "Hello Arduino! ";
    LVGL_Arduino += String('V') + lv_version_major() + "." +
lv_version_minor() + "." + lv_version_patch();

    Serial.println( LVGL_Arduino );
    Serial.println( "I am LVGL_Arduino" );

    lv_init();

#if LV_USE_LOG != 0
    lv_log_register_print_cb( my_print ); /* register print function for
debugging */
#endif

//código inicializar touchscreen, los parámetros están definidos arriba, se
usa la libreria XPT2046
    tsSPI.begin(XPT2046_CLK, XPT2046_MISO, XPT2046_MOSI, XPT2046_CS);
    touchscreen.begin(tsSPI);
    touchscreen.setRotation(0); // Rotación del táctil a 0

    tft.begin();          /* TFT init */
    tft.setRotation(0); // Rotación pantalla a 0, debe coincidir con la táctil
    tft.invertDisplay(1); //Para que los colores no salgan invertidos

    lv_disp_draw_buf_init( &draw_buf, buf, NULL, screenWidth * screenHeight /
10 );

    /*Initialize the display*/
    static lv_disp_drv_t disp_drv;
    lv_disp_drv_init( &disp_drv );
    /*Change the following line to your display resolution*/
    disp_drv.hor_res = screenWidth;
    disp_drv.ver_res = screenHeight;
    disp_drv.flush_cb = my_disp_flush;
```

```
    disp_drv.draw_buf = &draw_buf;
    lv_disp_drv_register( &disp_drv );

    /*Initialize the (dummy) input device driver*/
    static lv_indev_drv_t indev_drv;
    lv_indev_drv_init( &indev_drv );
    indev_drv.type = LV_INDEV_TYPE_POINTER;
    indev_drv.read_cb = my_touchpad_read;
    lv_indev_drv_register( &indev_drv );

    ui_init();
    Serial.println( "Setup done" );

///Timers
  timer_manual = lv_timer_create(timer_callback_manual, 1000, NULL); // Crear
el temporizador (1 segundo de intervalo) para que no se bloquee la pantalla en
la cuenta atrás
  timer_maceracion = lv_timer_create(timer_callback_maceracion, 1000, NULL);
// Temporizador maceración
  timer_ebullicion = lv_timer_create(timer_callback_ebullicion, 1000, NULL);
// Temporizador ebullición
  //timer_verificar_grifo = lv_timer_create(verificar_grifo_callback, 100,
NULL);

  lv_timer_pause(timer_manual); // Pausar el temporizador al inicio para que
no vaya restando
  lv_timer_pause(timer_maceracion);
  lv_timer_pause(timer_ebullicion);
 // lv_timer_pause(timer_verificar_grifo); // Pausar el temporizador al inicio

//Monitores serie
  Serial.begin( 115200 ); /* prepare for possible serial debug */
  Serial2.begin(115200, SERIAL_8N1, RX_PIN, TX_PIN);//recibir y enviar datos
por el serial 2, comunicación entre placas
}
```

Por orden primeramente se inicializa el monitor serie para ir mostrando por la pantalla del ordenador los diversos mensajes que están programados, tales como errores para depuración. Luego se inicializa la librería LVGL, la pantalla (primero el táctil y luego la visualización), es importante que la rotación de ambas coincida. En este caso es necesario también invertir los colores, ya que por algún motivo el driver los muestra al revés. Lo siguiente es continuar con la interfaz de usuario y finalmente se inicializa el temporizador que acto seguido se pausa y la comunicación serie 2 para poder comunicarse con la placa2.

A continuación viene el programa principal “loop”, que se va repitiendo hasta el infinito o hasta que se apague la placa, lo que suceda antes.

```
void loop(){
```

```
  recibirDatos();
  enviarDatos(emergencia,bomba,temperaturaEnviada);
  lv_timer_handler(); /* let the GUI do its work */
  delay(5);
}
```

Este bucle recibe los datos de la placa 2, revisa si ha habido alguna interrupción por parte de la interfaz de usuario y para durante 5 ms.

La siguiente parte del código contiene el desarrollo de las funciones que se pueden consultar en los anexos.

Configuración de User_Setup.h de la librería TFT_eSPI.h

```
#define ILI9341_2_DRIVER
#define TFT_WIDTH  240
#define TFT_HEIGHT 320 // ST7789 240 x 320
#define TFT_BL   21            // LED back-light control pin
#define TFT_BACKLIGHT_ON HIGH  // Level to turn ON back-light (HIGH or LOW)
#define TFT_MISO 16
#define TFT_MOSI 13
#define TFT_SCLK 14
#define TFT_CS   15  // Chip select control pin
#define TFT_DC    2  // Data Command control pin
//#define TFT_RST   4  // Reset pin (could connect to RST pin)
#define TFT_RST  12
#define TOUCH_CS 33     // Chip select pin (T_CS) of touch screen
#define LOAD_GLCD   // Font 1. Original Adafruit 8 pixel font needs ~1820
bytes in FLASH
#define LOAD_FONT2
#define LOAD_FONT4
#define LOAD_FONT6
#define LOAD_FONT7
#define LOAD_FONT8
#define LOAD_GFXFF  // FreeFonts. Include access to the 48 Adafruit_GFX free
fonts FF1 to FF48 and custom fonts
#define SMOOTH_FONT
#define SPI_FREQUENCY  65000000
#define SPI_READ_FREQUENCY  20000000
 #define SPI_TOUCH_FREQUENCY  2500000
#define USE_HSPI_PORT
```

A continuación se describe el código de la Placa2, este código se encarga de recoger los valores del sensor de temperatura y del sensor de flujo, recibir si hay que realizar una parada de emergencia y los valores de temperatura consigna y activar/desactivar la bomba.

Al igual que antes, lo primero de todo es definir las librerías utilizadas que son la base de todo el código. Solamente se mencionan aquellas que no han sido mencionadas antes.

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <HardwareSerial.h>
#include <ArduinoJson.h>
#include <PID_v1.h>
```

- OneWire.h – permite la utilización de un solo pin GPIO para recibir datos de distintos dispositivos compatibles.
- DallasTemperature.h – la librería que permite obtener los datos del sensor de temperatura DS18B20.
- PID_V1.h – crea un PID virtual, que permite la salida modulada mediante un pin PWM para el control de sistemas, en este caso para el calefactor.

Siguiendo la estructura anterior, lo siguiente es el prototipado de las funciones, que permite al IDE reconocerlas y no dar erratas.

```
//Prototipos de funciones para evitar errores
// Funciones de comunicación
void recibirDatos(void);  // Recibe datos (JSON) de la placa 1
void enviarDatosPlaca2(void); // Envía datos (JSON) a la placa 1
// Funciones de sensores
void leerTemperaturaSensor(void); // Lee la temperatura del sensor DS18B20
void leerEstadoGrifo(void); // Lee el estado del grifo (abierto o cerrado)
// Funciones de actuadores
void ActivarBomba(void); // Activa o desactiva la bomba según las condiciones
void activarCalentador(void); // Controla el calentador mediante PID
```

El siguiente punto es la definición de pines y variables globales. Siguiendo la estela de la CYD, la decisión de hacerlas globales en vez de locales es para facilitar su actualización y consulta.

```
//////////////DEFINICION DE PINES/////////////
//Comunicacion entre placas
#define ONE_WIRE_BUS 32 // Pin de datos del DS18B20
#define TX_PIN 1 // Pin TX (GPIO 1)
#define RX_PIN 3 // Pin RX (GPIO 3)
//Actuadores y sensores
#define PIN_BOMBA 2 //Pin activar relé bomba
#define PIN_CALENTADOR 4 //Pin activar SSR, salida del PID
#define PIN_GRIFO 16 //Pin sensor XKC-Y28, que indica el grifo abierto


//////////////Variables globales recibidas de la otra placa////////////////
bool emergenciaRecibida = false;
bool bombaRecibida = false;
```

```
int temperaturaRecibida = 0;
int indicePeriodoRecibido = 0;
//Por si falla el JSON
bool emergenciaAnterior = false;
bool bombaAnterior = false;
float temperaturaAnterior = 0.0;

//////Variables globales enviadas///////
bool grifoAbiertoEnviado = 0;
float temperaturaLeida = 0.0;

// Variables para el PID
double Setpoint, Input, Output;
double Kp = 5, Ki = 10, Kd = 1;
int OutputMin = 0, OutputMax = 255; // Límites de salida (0-255 para PWM)
```

Luego, han de crearse las instancias de las distintas librerías.

```
/Crear instancias
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

Una vez ya están todo definido, es el momento de cargar el setup y hacer la inicialización de los distintos componentes.

```
void setup() {
  Serial.begin(115200);
  Serial2.begin(115200, SERIAL_8N1, RX_PIN, TX_PIN
  sensors.begin();

  pinMode(PIN_GRIFO, INPUT);
  pinMode(PIN_BOMBA, OUTPUT);

  myPID.SetMode(AUTOMATIC);
  myPID.SetOutputLimits(OutputMin, OutputMax);
  myPID.SetSampleTime(100)
}
```

Se inicializan 2 puertos series para la comunicación entre placas y se inicializan los sensores que se comunican por OneWire. Se definen los puertos, el PIN_GRIFO se inicializa como entrada, así cuando reciba un valor de 3,3 V, la placa lo identificará como activo, mientras que el PIN_BOMBA es el que activará el relé de control de la bomba de recirculado de 12V. Finalmente dentro de este bloque se inicializa la librería PID, en modo automático, irá ajustando la salida del actuador según la temperatura recibida, se marcan los límites de salida definidos junto al resto de variables (0 sin salida, 255 salida continua) y finalmente el tiempo de muestreo.

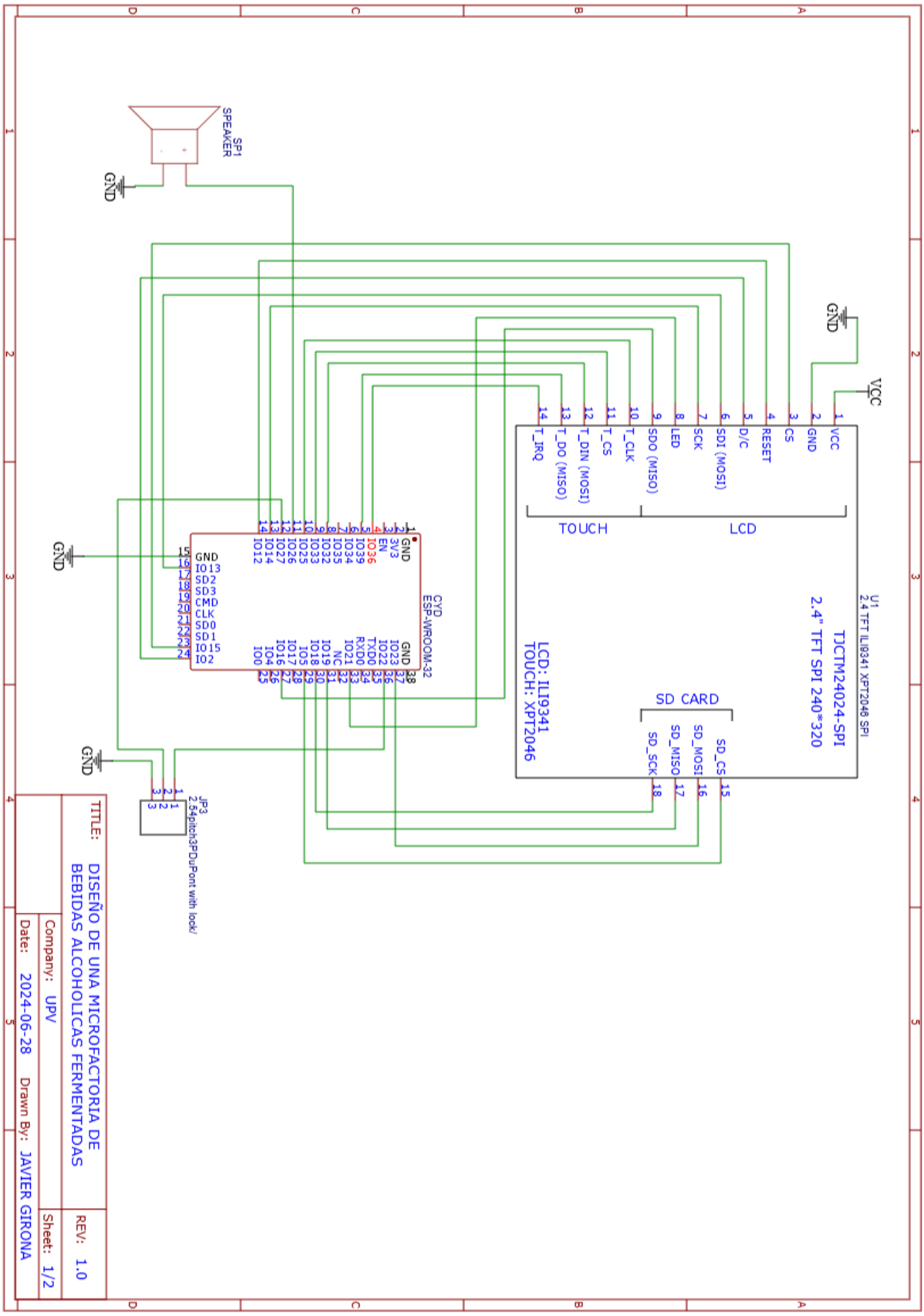
Para finalizar la explicación del código, se llega al bucle principal (loop).

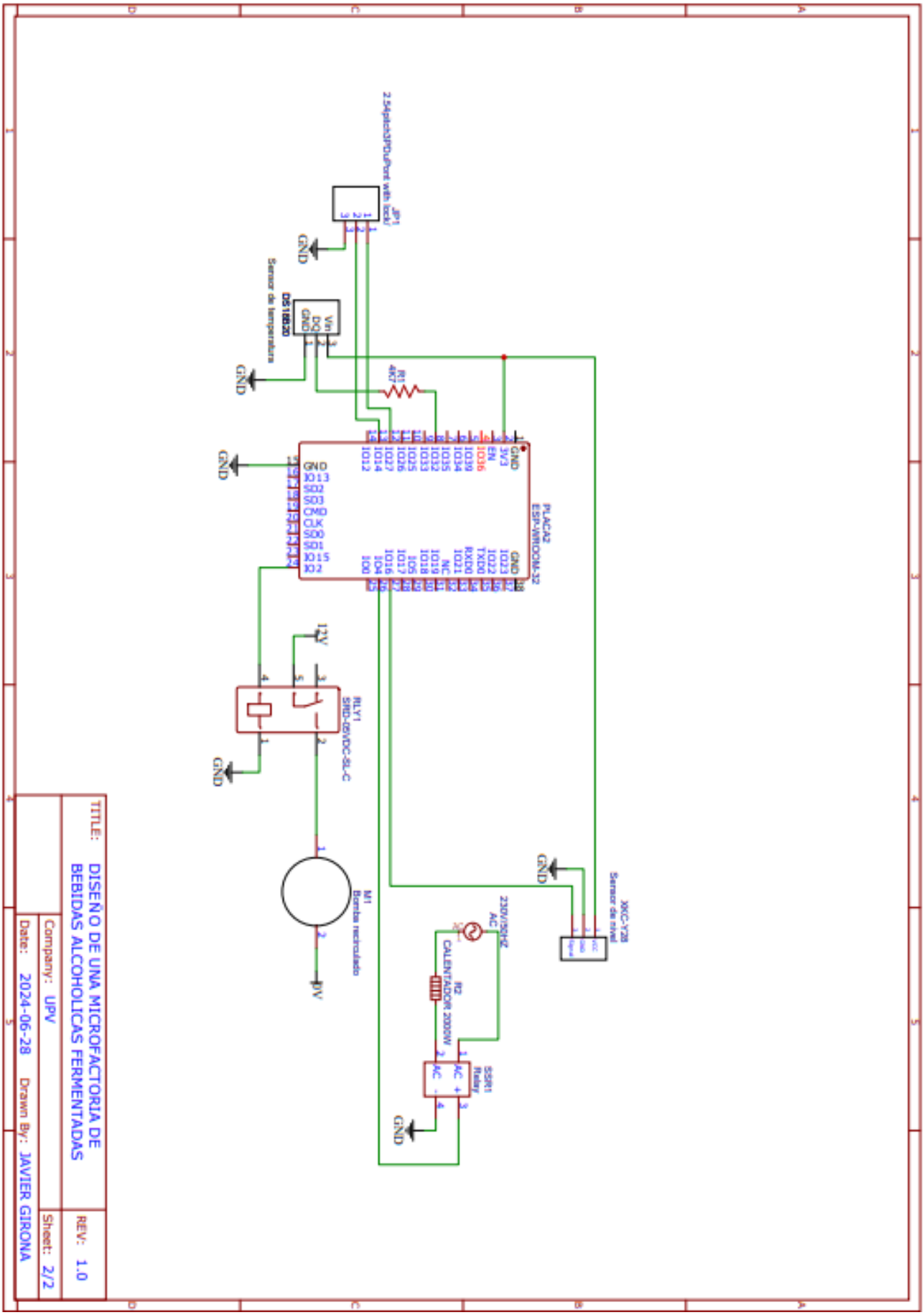
```
void loop() {
  leerTemperaturaSensor();
  leerEstadoGrifo();
  enviarDatosPlaca2(); // Enviar datos a la placa 1
  recibirDatos(); // Recibir datos de la placa 1
  if (emergenciaRecibida) {
    // Parada de emergencia: apagar bomba y calentador
    digitalWrite(PIN_BOMBA, LOW);
    digitalWrite(PIN_CALENTADOR, LOW);
  }
  else{
    ActivarBomba();
    activarCalentador();
  }

  delay(5); // Retardo igual al de la placa CYD
}
```

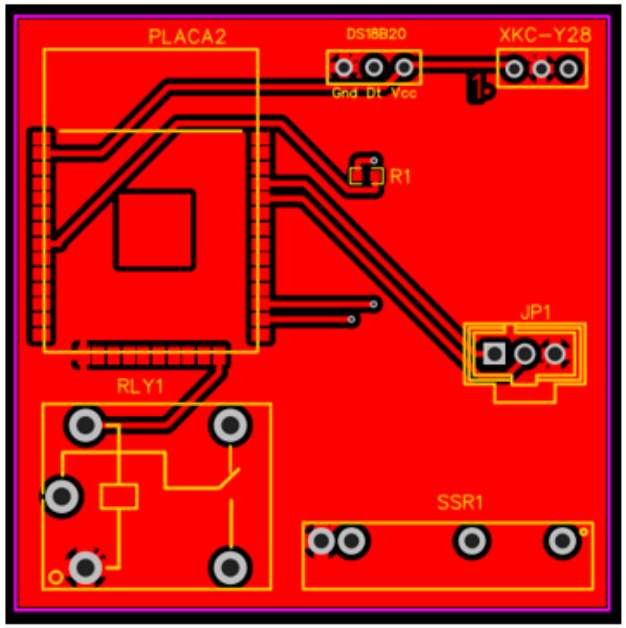
Primeramente lee la temperatura del sensor, a continuación si el grifo de la bomba de recirculado está abierto y envía estos datos a la placa CYD. Su siguiente instrucción es la de recibir los datos de la otra placa y decidir cómo actuar, si hay una emergencia para inmediatamente la bomba y el calefactor, en caso contrario llama a las funciones de activación para que actúe según las órdenes recibidas. Finalmente realiza un parada de 5 ms igual que la otra placa y se reinicia el bucle.

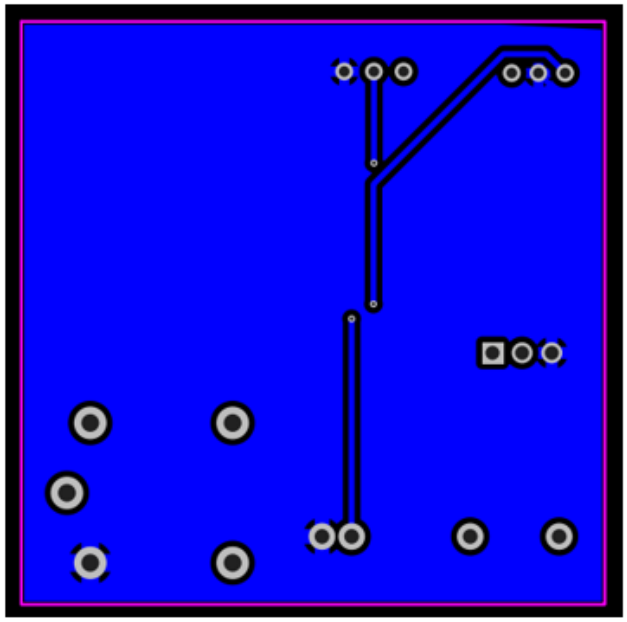
El resto del código puede consultarse en los anexos.

5. PLANOS

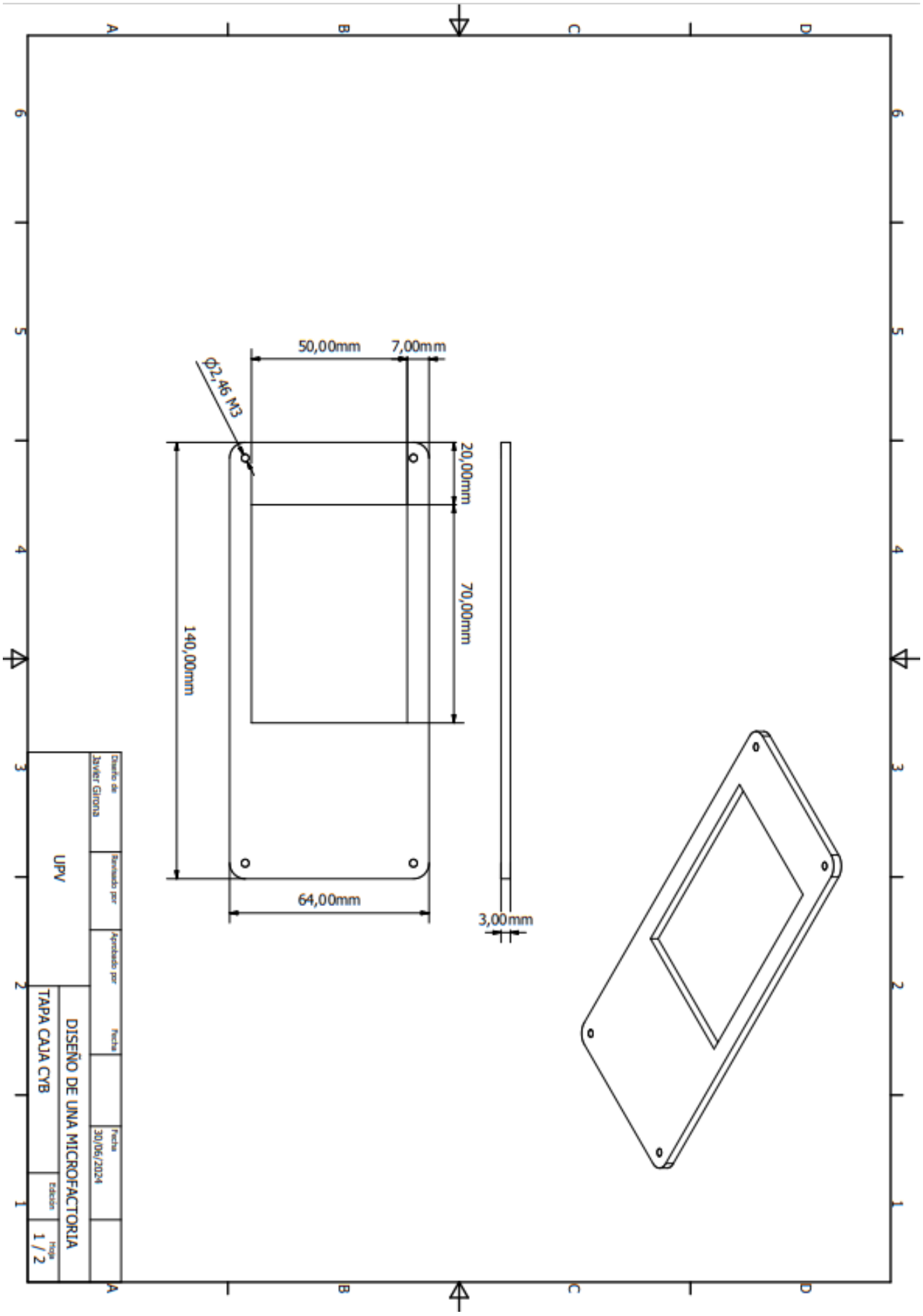

SP1
SPEAKER
GND
VCC
U1
2.4 TFT ILI9341 XPT2046 SPI
TJCTM24024-SPI
2.4" TFT SPI 240*320
1 VCC
2 GND
3 CS
4 RESET
5 D/C
6 SDI (MOSI)
7 SCK
8 LED
9 SDO (MISO)
10 T_CLK
11 T_CS
12 T_DIN (MOSI)
13 T_DO (MISO)
14 T_IRQ
TOUCH
LCD
LCD: ILI9341
TOUCH: XPT2046
SD CARD
SD_CS 15
SD_MOSI 16
SD_MISO 17
SD_SCK 18
CYD
ESP-WROOM-32
JP3
2.54pitch3PDuPont with lock/
TITLE:
DISEÑO DE UNA MICROFACTORIA DE BEBIDAS ALCOHOLICAS FERMENTADAS
REV: 1.0
Company: UPV
Sheet: 1/2
Date: 2024-06-28
Drawn By: JAVIER GIRONA

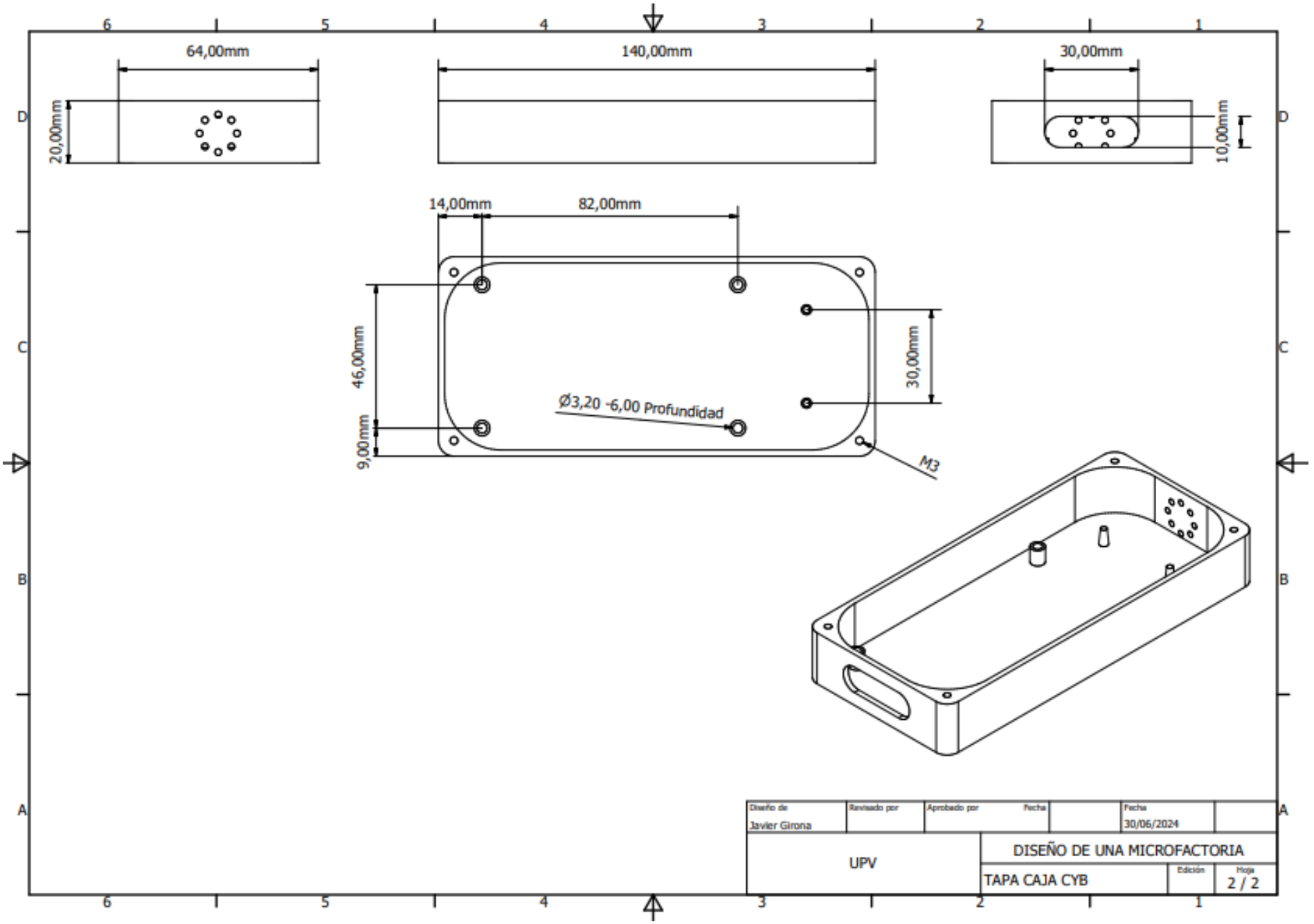

TITLE: DISEÑO DE UNA MICROFACTORIA DE BEBIDAS ALCOHOLICAS FERMENTADAS
REV: 1.0
Company: UPV
Sheet: 2/2
Date: 2024-06-28
Drawn By: JAVIER GIRONA
PLACA2 ESP-WROOM-32
RLY1 SRD-05VDC-SL-C
M1 Bomba recirculado
R2 CALENTADOR 2000W
SSR1 Relay
230V/50HZ AC
XKC-Y28 Sensor de nivel
R1 4K7
DS18B20 Sensor de temperatura
GND
12V



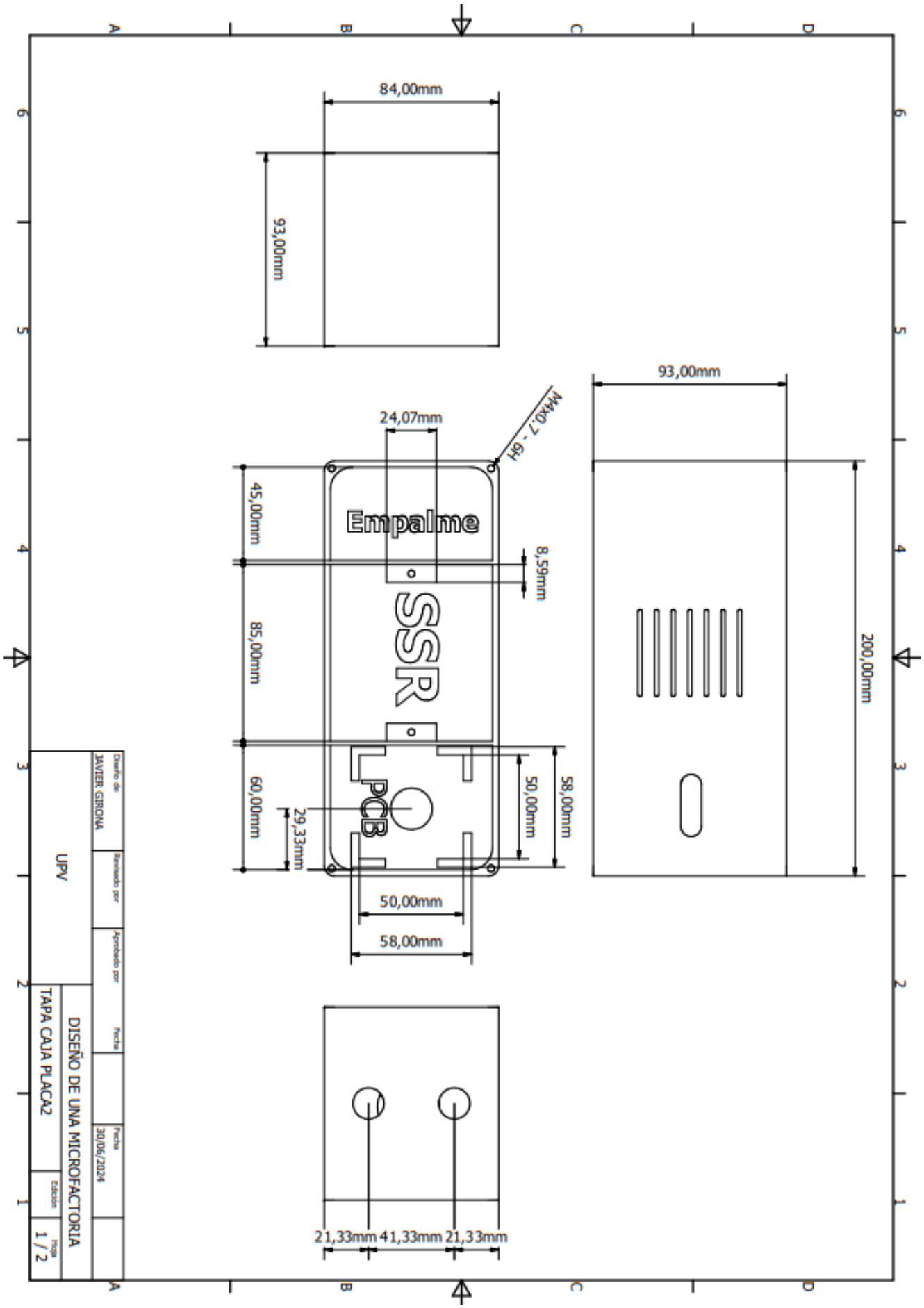


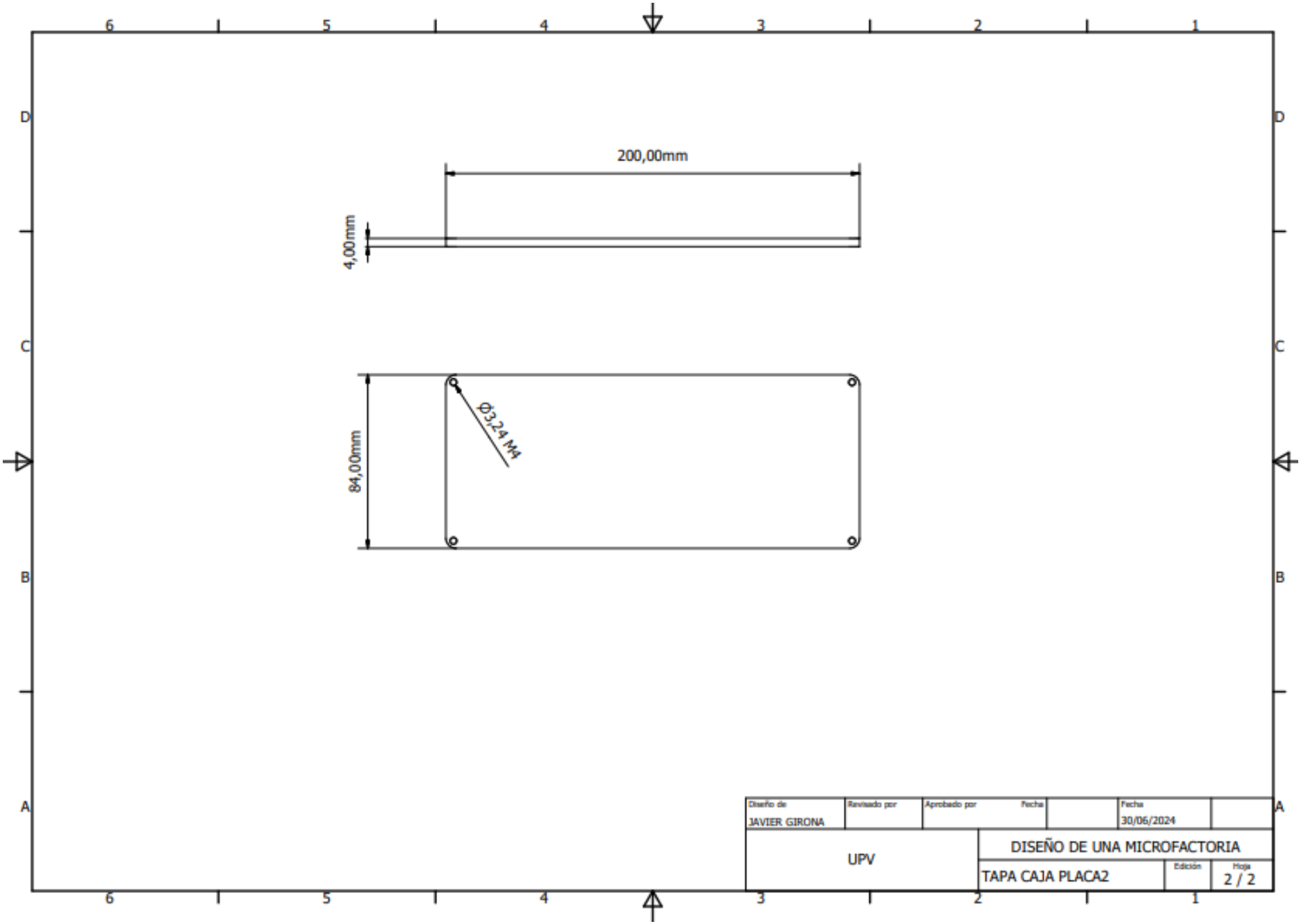
TITLE:	DISEÑO DE UNA MICROFACTORIA DE BEBIDAS ALCOHOLICAS FERMENTADAS PCB	REV: 1.0
	Company: UPV	Sheet: 1/1
	Date: 2024-06-28 Drawn By: JAVIER GIRONA	


50,00mm
7,00mm
ϕ2,46 M3
20,00mm
70,00mm
140,00mm
64,00mm
3,00mm
UPV
Diseño de
Javier Girona
Revisado por
Aprobado por
Fecha
Fecha
30/06/2024
Edición
Hoja
1 / 2
TAPA CAJA CYB
DISEÑO DE UNA MICROFACTORIA



Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
Javier Girona					30/06/2024	
UPV				DISEÑO DE UNA MICROFACTORIA		
				TAPA CAJA CYB	Edición	Hoja 2 / 2


84,00mm
93,00mm
93,00mm
200,00mm
24,07mm
M4x0.7 - 6H
45,00mm
Empalme
8,59mm
SSR
85,00mm
PCB
60,00mm
29,33mm
50,00mm
58,00mm
50,00mm
58,00mm
21,33mm
41,33mm
21,33mm
UPV
Diseño de
JAVIER GIRONA
Revisado por
Aprobado por
Fecha
Fecha
30/06/2024
Edición
Hoja
1 / 2
DISEÑO DE UNA MICROFACTORIA
TAPA CAJA PLACA2



Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
JAVIER GIRONA					30/06/2024	

UPV	DISEÑO DE UNA MICROFACTORIA		
	TAPA CAJA PLACA2	Edición	Hoja 2 / 2

6. PLIEGO DE CONDICIONES

6.1. OBJETIVO

El objetivo principal de este proyecto es el diseño de una microfactoría de bebidas alcohólicas fermentadas que permita la elaboración de pequeños lotes de hasta 20 litros y que cumpla con los siguientes requisitos:

- **Modularidad y adaptabilidad:** El diseño ha de ser flexible y debe permitir la modificación y mejora por parte del usuario, este se ha de adaptarse a diferentes presupuestos y necesidades.
- **Repetibilidad:** El sistema debe garantizar un alto grado de repetibilidad en el proceso de elaboración, minimizando la variabilidad entre lotes.
- **Automatización y control:** Se implementará un sistema de control automatizado para regular temperaturas, tiempos y otros parámetros críticos del proceso, ofreciendo también la opción de control manual para mayor flexibilidad.
- **Facilidad de uso:** La interfaz de usuario debe ser intuitiva y fácil de usar, permitiendo a cualquier persona operar la microfactoría sin dificultad.
- **Seguridad y cumplimiento normativo:** El diseño debe cumplir con todas las normas sanitarias y de seguridad aplicables, utilizando materiales de grado alimenticio y componentes seguros.
-

6.2. ALCANCE DEL PROYECTO

El proyecto abarcará las siguientes fases:

- **Diseño conceptual:** Definición de los requisitos funcionales y técnicos, selección de componentes y propuesta de diferentes alternativas.
- **Diseño detallado:** Desarrollo de planos, especificaciones de componentes, diseño de la interfaz gráfica de usuario e implementación en Arduino.
- **Documentación:** Elaboración de un manual de usuario que permita el uso del sistema.

6.3. REQUISITOS FUNCIONALES

La micro factoría deberá ser capaz de realizar las siguientes funciones:

- **Maceración:** Control preciso de la temperatura y el tiempo de maceración, se podrá realizar tanto de manera manual como de manera automática.

- **Recirculado:** Incorporar un sistema de recirculado continuo para mejorar el rendimiento de la maceración y filtrar el mosto. El sistema irá protegido contra trabajo en vacío y se podrá activar.
- **Cocción:** Control preciso del tiempo de cocción, con distintos avisadores programables que permitirán añadir los ingredientes en diferentes etapas.
- **Fermentación y maduración:** No se incluye en el alcance de este proyecto, pero el diseño debe facilitar la transferencia del mosto a un fermentador externo.
- **Control y monitorización:** El sistema de control debe permitir la monitorización en tiempo real de la temperatura, el tiempo y otros parámetros relevantes, así como el ajuste de los parámetros de control.

6.4. REQUISITOS TÉCNICOS

- **Materiales:** Todos los componentes en contacto con el mosto deben ser de acero inoxidable de grado alimenticio o de plástico apto para uso alimentario.
- **Capacidad:** La micro factoría debe tener una capacidad mínima de 5 litros Y máxima de 20 litros.
- **Sistema de control:** Se utilizará un “shield” con pantalla táctil y un microcontrolador ESP32 para el mando y monitorización del proceso. Mediante una interfaz gráfica. Se utilizará otro microcontrolador ESP32 para la lectura de datos y el mando de los actuadores.
- **Sensores:** Se utilizarán sensores de temperatura y un sensor de líquidos para monitorizar el sistema y garantizar la seguridad y su correcto funcionamiento.
- **Actuadores:** Se utilizarán distintos actuadores tales como una bomba de recirculación de 12V, un relé de estado sólido (SSR) para controlar la potencia del calefactor y un zumbador para alertas.

6.5. NORMATIVA

La normativa que deberá cumplir el proyecto está dividida en dos bloques. Uno correspondiente a la parte alimentaria y otro correspondiente a la parte eléctrica.:

6.5.1. SEGURIDAD ALIMENTARIA

- **Real Decreto 678/2016,** de 16 de diciembre, por el que se aprueba la norma de calidad de la cerveza y de las bebidas de malta.
- **Reglamento (CE) 1935/2004** del Parlamento Europeo y del Consejo, de 27 de octubre de 2004, sobre los materiales y objetos destinados a entrar en contacto con alimentos y por el que se derogan las Directivas 80/590/CEE y 89/109/CEE.

- **Reglamento (CE) 2023/2006** de la Comisión, de 22 de diciembre de 2006 , sobre buenas prácticas de fabricación de materiales y objetos destinados a entrar en contacto con alimentos (Texto pertinente a efectos del EEE).
- **Real Decreto 2060/2008,** de 12 de diciembre, por el que se aprueba el Reglamento de equipos a presión y sus instrucciones técnicas complementarias.
- **Real Decreto 769/1999**, de 7 de mayo, por el que se dictan las disposiciones de aplicación de la Directiva del Parlamento Europeo y del Consejo, 97/23/CE, relativa a los equipos de presión y se modifica el Real Decreto 1244/1979, de 4 de abril, que aprobó el Reglamento de aparatos a presión.

6.5.2. NORMATIVA ELÉCTRICA

- **Real Decreto 842/2002** , de 2 de agosto, por el que se aprueba el Reglamento electrotécnico para baja tensión.
- **Real Decreto 1644/2008**, de 10 de octubre, por el que se establecen las normas para la comercialización y puesta en servicio de las máquinas.
- **UNE-EN 60204-1:** Seguridad de las máquinas. Equipo eléctrico de las máquinas. Parte 1: Requisitos generales.
- **UNE-EN ISO 12100:** Seguridad de las máquinas. Principios generales para el diseño. Evaluación del riesgo y reducción del riesgo. (ISO 12100:2010)
- **UNE-EN 60335-1:** Aparatos electrodomésticos y análogos. Seguridad. Parte 1: Requisitos generales.

7. PRESUPUESTO

A continuación, se detalla el presupuesto del proyecto. Está separado en distintas partes, la primera es donde se detalla el coste de material y el coste de los programas informáticos, en la segunda se detalla el tiempo en mano de obra, en la tercera se detallan los costes indirectos. Esta decisión de separar el presupuesto está justificada por que, si se trata de un proyecto personal, solamente se tendrá en cuenta los materiales, mientras que si se trata de obtener un beneficio del diseño se han de tener en cuenta los costes de personal.

7.1. COSTE DIRECTO DE MATERIAL

MATERIALES					
Concepto	Ctd	Precio sin IVA	IVA (21 %)	Coste (€)	Proveedor
Olla eléctrica	1	94,00	24,99	118,99	Klarstein
Grifo y filtro	1	22,40	5,96	28,36	MasMalta.com
Bomba DC 12V	1	6,78	1,80	8,58	Aliexpress.com

Latiguillos silicona 10x14 mm (precio por metro)	5 m	2,32	0,62	14,70	Aliexpress.com
Serpentín	1	47,36	12,59	59,95	Tucervezacasera.com
Kit cervecero Deluxe	1	55,26	14,69	69,95	Tucervezacasera.com
Sensor de temperatura DS18B20	1	1,06	0,28	1,34	Aliexpress.com
Sensor de líquido XKC-Y28	1	5,72	1,52	7,24	Aliexpress.com
Shield pantalla táctil	1	2,80	0,74	3,54	Aliexpress.com
Procesador ESP32	1	0,78	0,21	0,99	Aliexpress.com
Zumbador	1	0,96	0,25	1,21	Aliexpress.com
Fuente de alimentación de 5 V y 5 A	1	3,03	0,81	3,84	Aliexpress.com
Fuente de alimentación de 12 V y 5 A	1	4,47	1,19	5,66	Aliexpress.com
Conjunto componentes electrónicos	1	8,33	2,22	10,55	Aliexpress.com
Kit tornillos fijación tapas	1	0,78	0,21	0,99	Aliexpress.com
Impresión 3D caja CYD	1	22,89	4,80	27,70	Tecnobro3d.com
Impresión 3D caja PLACA2		116,57	24,48	141,05	Tecnobro3d.com
COSTE TOTAL		**396,06**	**29,49**	**425,55**	

Tabla 4-Presupuesto de materiales

Para el coste del software se toma en consideración la licencia “Business”, por ser la más habitual. En caso de existir una licencia de por vida, se incluirá esa, en lugar de las licencias por suscripción.

SOFTWARE			
USUARIO			
Software	Licencia sin IVA (€)	IVA (21%)	Coste total (€)
IDE Arduino	0,00	0,00	0,00
Squareline Studio	0,00	0,00	0,00
Matlab	119,00	24,99	143,99
Coste total	119,00	24,99	143,99
EMPRESA			
Software	Licencia sin IVA (€)	IVA (21%)	Coste total (€)
IDE Arduino	0,00	0,00	0,00
Squareline Studio	3927,00	824,67	4751,67
Matlab	3650,00	766,50	4416,50

TOTAL	7577,00	1591,17	9168,17

Tabla 5-Coste de software tanto a usuario cómo a empresa

7.2. COSTE MANO DE OBRA

A continuación se mostrará el coste relativo a la mano de obra, este será relevante para aquellas empresas que quieran contratar a un ingeniero electrónico que prepare el proyecto o para aquellos ingenieros que quieran vender el proyecto a una empresa.

MANO DE OBRA				
	Horas	**Precio/hora (€/h)**	**Iva (21%)**	**Total (€)**
Ingeniero	120	19,75	5,25	3000,00
DESGLOSE MANO DE OBRA				
Tarea	**Tiempo (h)**	**Precio/hora (€/h)**	**Iva (21%)**	**Coste (€)**
Investigación proceso	24	25,00	5,25	600,00
Diseño microfactoría	22	25,00	5,25	550,00
Preparación materiales	2	25,00	5,25	50,00
Diseño interfaz gráfica	24	25,00	5,25	600,00
Programación código	40	25,00	5,25	1000,00
Montaje y puesta a punto	8	25,00	5,25	200,00
TOTAL	120	25,00	5,25	3000,00

Tabla 6-Coste de mano de obra

7.3. COSTES INDIRECTOS

En esta sección se detallan los costes indirectos, estos aplican tanto a las empresas como a los particulares.

GASTOS INDIRECTOS				
Concepto	Precio sin IVA (€)	Iva (21%)	Unidades	Coste total (€)
Ordenador*	237,00	63,00	1	300,00
Luz (KW/h)**	8,81	2,34	4	45,00
TOTAL	245,81	65,34		345,00

Tabla 7-Costes indirectos

7.4. COSTE TOTAL DEL PROYECTO

De acuerdo a la filosofía del proyecto y según comentado al inicio del presupuesto, el coste total del proyecto se va a calcular de 2 maneras distintas. La primera será el coste que deberá asumir un miembro de la comunidad “brewer” o “maker” que realice el proyecto para su uso personal, que podrá acceder al código en plataformas como github o arduino, mientras que la otra será el coste que debería asumir una empresa que realice el proyecto desde cero.

7.4.1. COSTE TOTAL PARA LA COMUNIDAD

Para el cálculo de este coste no se va a tener en cuenta el coste de mano de obra ya que se trata de un proyecto personal, para el software utilizado no se va a tener en cuenta el uso de Matlab ni el de impresión 3D.

APARTADOS	COSTE (€)
MATERIAL	334,90
SOFTWARE	0,00
MANO DE OBRA	0,00
INDIRECTOS	345,00
TOTAL	679,90

Tabla 8-Coste total para la comunidad

7.4.2. COSTE TOTAL PARA LA EMPRESA

Para el cálculo de este coste se tienen en cuenta todos los costes calculados anteriormente.

APARTADOS	COSTE (€)
MATERIAL	425,55
SOFTWARE	9168,17
MANO DE OBRA	3000,00
INDIRECTOS	345,00
TOTAL	12938,72

Tabla 9-Coste total para la empresa

8. MANUAL DE USUARIO

El siguiente manual recoge el funcionamiento de la interfaz de usuario, en adelante iu, de la microfactoría de bebidas fermentadas.

Tras arrancar, lo primero que se puede observar es la selección de modo, estos son:

- MANUAL
- AUTOMATICO.

Figura 69-Pantalla inicial de selección de modo

8.1. MODO MANUAL

Primeramente se va a explicar el modo MANUAL y luego el modo AUTOMATICO. Al pulsar sobre el botón MANUAL, la ui cambia de pantalla y muestra tres opciones más.

- LEER ESTADO / ACTIVAR BOMBA
- ACTIVAR CALEFACTOR
- ICONO DE VOLVER ATRÁS
    - representado con un cuadrado y una flecha en su interior apuntando hacia la izquierda. Este botón aparecerá en la mayoría de las pantallas y su función es siempre volver a la pantalla anterior, por lo que no se volverá a mencionar.

Figura 70-Botón "vuelta atrás"

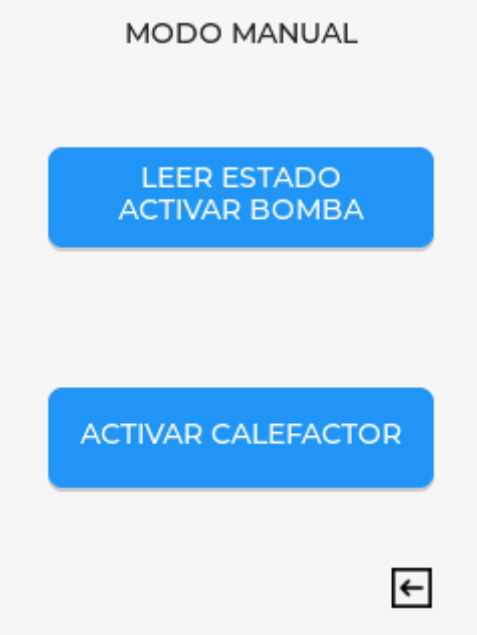


Figura 71-Dentro del modo Manual las opciones disponibles

Al pulsar el primer botón LEER ESTADO/ACTIVAR BOMBA, la pantalla siguiente mostrará la temperatura actual del sensor, así como tres opciones más.

- El señalizador “Grifo abierto” con un cuadro de color a su lado.
  - Si el cuadro está rojo, el grifo está cerrado y por tanto no se puede activar la bomba
  - Si el cuadro está verde, el grifo está abierto y es posible activar la bomba.
- Activar Bomba.
  - Este botón estará deshabilitado hasta que se detecte que el grifo está abierto. Una vez se detecte que está abierto se habilitará.
  - Una vez habilitado al pulsarlo, se encenderá la bomba de recirculación, se habilitará el botón “Parar Bomba” y se deshabilitará el botón Activar Bomba. Por seguridad la bomba se apagará automáticamente al cerrar el grifo.
- Parar Bomba.
  - Este botón estará deshabilitado de primeras y solamente se habilitará cuando la bomba esté en funcionamiento. Al pulsarlo parará la bomba y volverá a habilitar el botón de activar bomba.

Como recordatorio, el botón “Activar bomba” sólo estará habilitado si la bomba está parada y si el grifo está abierto. Al volver la bomba NO cambiará de estado.

Figura 72-Pantalla de estado de la bomba. Presentación de variaciones: con grifo abierto, con grifo cerrado y bomba parada y encendida

Volviendo atrás y seleccionando la opción de “ACTIVAR CALEFACTOR”, se presentan otra vez 3 opciones:

- TEMPERATURA OBJETIVO
  - Que se deberá ajustar en ºC mediante los botones +/- situados a los extremos.
- TEMPORIZADOR
  - Activará o desactivará el uso del temporizador. Al activarlo, se habilitarán los botones +/- ubicados debajo.
  - Con los botones se ajustará el tiempo en MINUTOS que se desea que funcione el calefactor. Si no se activa ningún tiempo el calefactor funcionará de manera indefinida. Esta opción es útil para macerar o fermentar bebidas que requieren una temperatura relativamente baja.
- Confirmar.
  - Confirma las opciones seleccionadas.

Figura 73-Pantalla modo manual, con temperatura y temporizador activado

En la siguiente pantalla aparece la cuenta atrás y el tiempo restante en formato hh:mm:ss. El botón STOP es una parada de emergencia, restablecerá la temperatura y el tiempo a 0 y apagará la bomba. El botón atrás NO detendrá ni el calefactor ni el tiempo.

Figura 74-Pantalla de temperatura y tiempo

8.2. MODO AUTOMATICO

Al seleccionar el modo AUTOMATICO, lo primero se muestra es la pantalla de selección de las rampas de macerado. El sistema permite 4 rampas distintas, cada una con una temperatura, un tiempo y la opción de recirculado. Como de costumbre la temperatura se pondrá en ºC y el tiempo en minutos.

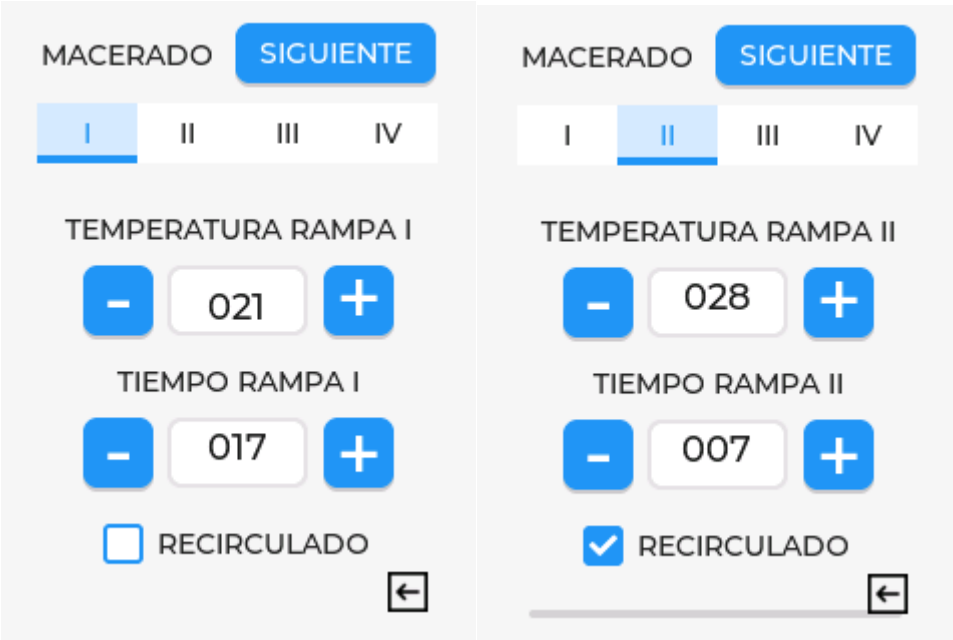


Figura 75-Distintas opciones de las rampas de maceración

Como buena práctica, las rampas de temperatura siempre han de ir en ascenso. En caso de que el tiempo o la temperatura sea 0, no se tendrá en cuenta esa rampa.

Al presionar SIGUIENTE, la iu mostrará 4 contadores de tiempo para los avisos sonoros que tendrán lugar durante el hervido. El tiempo es acumulativo y sirve para calcular también el tiempo total que estará hirviendo el mosto. Es decir, si se pone un aviso a los 6 minutos y otro a los 15, el tiempo total de hervido será de 21 minutos con 2 avisos sonoros. Es importante informar que los temporizadores solamente empezarán a funcionar cuando se haya alcanzado la temperatura objetivo, +/- 3 ºC para el macerado y una temperatura superior a 98 ºC para el hervido. Al presionar el botón EMPEZAR, empezará el proceso automático de la microfactoría.

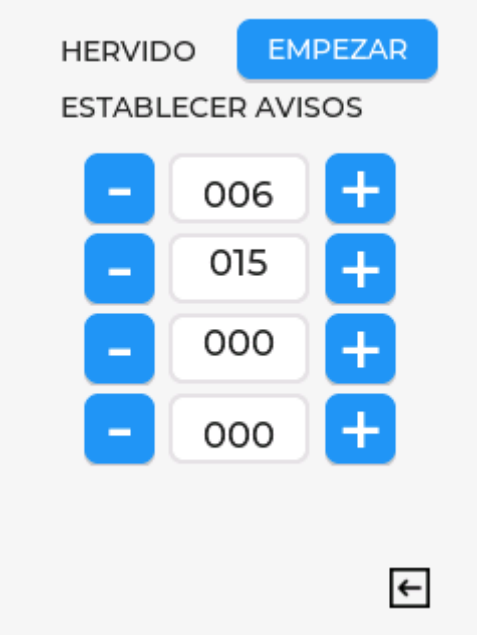


Figura 76-Avisos de hervido

En la siguiente pantalla, se mostrará la temperatura la temperatura actual que está leyendo el sensor, el tiempo restante de macerado, que es la suma de las distintas rampas de tiempo, y dos botones, EBULLICION y STOP.

- EBULLICION.
  - Este botón se habilitará cuando haya acabado el tiempo de macerado y el grifo esté cerrado.
  - STOP
    - Parada de emergencia.

Figura 77-Pantalla maceración, la primera durante el proceso y la otra con el proceso finalizado y la otra con el proceso finalizado y listo para empezar la ebullición

Al presionar EBULLICION, el calentador empezará a aumentar la temperatura hasta que el líquido empiece a hervir. La cuenta atrás de los avisos empezará cuando la temperatura sea superior a 98 ºC. En esta pantalla se mostrará la temperatura actual, el tiempo restante hasta el siguiente aviso y el tiempo total hasta la finalización del hervido. Una vez finalizado el proceso se habilitará el botón FIN PROCESO que permitirá volver a la selección de modo.

Figura 78-Pantalla de ebullición en marcha y proceso finalizado

Con esto se da por finalizado el manual de usuario de la iu.

Figura 79-A disfrutar, imagen generada por IA

9. BIBLIOGRAFÍA

9.1. LIBROS Y REVISTAS

- Gayre, R. (1948). *Wassail! In Mazers of Mead*. Phillimore & Co Ltd.
- McGovern, P. E. (2009). *Uncorking the Past: The Quest for Wine, Beer, and Other Alcoholic Beverages*. Princeton University Press.
- Merker, G. P., Hanbury, W. T., & Rautenbach, W. L. (1992). Experimental investigation of the overall heat transfer coefficient in a domestic electric kettle. *International Journal of Heat and Mass Transfer*, 35(6), 1473-1477.
- Parsons, J. R., & Parsons, M. H. (1990). *Maguey Utilization in Highland Central Mexico: An Archaeological Ethnography*. Anthropological Papers of the Museum of Anthropology, University of Michigan.
- Sanchis, P., Orive, M., & Ramos, J. (2000). *La cerveza: elaboración, composición y propiedades*. AMV Ediciones.
- Unwin, T. (1991). *Wine and the Vine: An Historical Geography of Viticulture and the Wine Trade*. Routledge.

9.2. RECURSOS WEB

- Sadurní, J. M. (2023, 6 de octubre). Descubren cientos de tinajas de vino intactas en la tumba de la reina Merneith en Abydos. *National Geographic*. https://historia.nationalgeographic.com.es/a/descubren-cientos-tinajas-vino-tumba-reina-merneith-abydos_20268
- Llamas, L. (2018, 15 de octubre). Medir temperatura de líquidos y gases con Arduino y DS18B20. luisllamas.es. https://luisllamas.es/medir-temperatura-de-liquidos-y-gases-con-arduino-y-ds18b20/
- Random Nerd Tutorials. (s. f.). ESP32 Cheap Yellow Display Board (ESP32-2432S028R). https://randomnerdtutorials.com/esp32-cheap-yellow-display-board-esp32-2432s028r/
- The Science of Step Mashing. Brew Your Own. https://byo.com/article/the-science-of-step-mashing/
- George, A. (2016, 22 de junio). The world's oldest paycheck was cashed in beer. *New Scientist*. The world's oldest paycheck was cashed in beer | New Scientist
- witnessmenow. (s. f.). ESP32-Cheap-Yellow-Display: Building a community around a cheap ESP32 Display with a touch screen. GitHub. https://github.com/witnessmenow/ESP32-Cheap-Yellow-Display
- Cerveza monástica. (2019, 30 de mayo). Holyblog.es. https://www.holyart.es/blog/monasterios/cerveza-monastica/
- Pardo Martín, C. F. (2018, 10 de diciembre). *Método de Ziegler-Nichols*. Control Automático - Picuino. https://picuino.com/blog/control-ziegler-nichols/
- Foro ACCE - Página principal (cerveceros-caseros.com)

9.3. LIBRERÍAS ARDUINO

- **OneWire:** Paul Stoffregen. (2023). *OneWire* (2.3.7) [Código fuente]. https://github.com/PaulStoffregen/OneWire
- **DallasTemperature:** Miles Burton. (2023). *DallasTemperature* (3.9.1) [Código fuente]. https://github.com/milesburton/Arduino-Temperature-Control-Library
- **HardwareSerial:** Arduino. (2023). *HardwareSerial* (1.2.0) [Librería incorporada en el núcleo de Arduino].
- **ArduinoJson:** Benoit Blanchon. (2023). *ArduinoJson* (6.21.3) [Código fuente]. https://github.com/bblanchon/ArduinoJson
- **PID_v1:** Brett Beauregard. (2019). *PID_v1* (1.2.0) [Código fuente]. https://github.com/br3ttb/Arduino-PID-Library
- **lvgl:** LittlevGL. (2023). *lvgl* (8.3.11) [Código fuente]. https://github.com/lvgl/lvgl
- **TFT_eSPI:** Bodmer. (2023). *TFT_eSPI* (2.5.10) [Código fuente]. https://github.com/Bodmer/TFT_eSPI
- **XPT2046_Touchscreen:** Paul Stoffregen. (2021). *XPT2046_Touchscreen* (1.4.1) [Código fuente]. https://github.com/PaulStoffregen/XPT2046_Touchscreen
- **SPI:** Arduino. (2023). *SPI* (1.0.0) [Librería incorporada en el núcleo de Arduino].

10. CONCLUSIONES

A lo largo del presente documento se ha ido desarrollando el diseño de una microfactoría de bebidas alcohólicas fermentadas, centrándose especialmente en las partes que requieren una mayor atención al detalle como son la fase de maceración y ebullición.

Se ha empezado con una breve explicación histórica de la importancia de las bebidas alcohólicas fermentadas para a continuación y tomando la cerveza como ejemplo, enumerar los distintos sistemas comerciales que actualmente es posible encontrar en el mercado. Tras ese pequeño acercamiento, se han revisado los procesos que intervienen en la fabricación de cerveza y con todo el contexto obtenido, ya ha sido posible decidir el tipo de microfactoría a diseñar.

El siguiente punto ha sido la elección del alcance del proyecto, debido a que se va a elaborar bebidas, siempre se parte de algún alimento y se ha considerado que creaba demasiada complejidad el tratamiento de las materias primas, siendo mejor centrarse en el proceso de su transformación. Los procesos seleccionados para la microfactoría han sido los de maceración y ebullición, que son aquellos en los que hay que llevar un mayor control.

Cuando ya estaba claro que se iba a diseñar, se ha profundizado en la selección y funcionamiento de los componentes clave, como la olla eléctrica, el microcontrolador ESP32, … así como se han ido planteando alternativas que permitiesen añadir funcionalidades o abaratar costos, decidiendo finalmente el uso de Arduino para implementar el proyecto.

Durante la implementación se han utilizado tecnologías de código abierto y componentes fácilmente disponibles en el mercado, lo que permite que sea un proyecto que facilite y fomente la colaboración entre la comunidad “maker” y la comunidad “brewer”.

En conclusión, este TFG demuestra la viabilidad de diseñar una microfactoría casera de bebidas alcohólicas fermentadas, con un grado bastante alto de automatización, que sea accesible y personalizable. El proyecto abre un abanico de posibilidades para la innovación y la experimentación en el mundo de la cerveza artesanal, poniendo al alcance de los aficionados herramientas que antes estaban reservadas a los profesionales.

Con el objetivo también de contribuir a los mundos “maker” y “brewer” se crea un github (Ctrl-Alt-Brew) que contiene todos los ficheros desarrollados hasta el momento. El objetivo es mantener el proyecto vivo y actualizado, implementar mejoras y corregir errores.

10.1. POSIBLES MEJORAS

El proyecto cumple con las expectativas que podría tener cualquier amante de la fabricación artesanal de cerveza y de otro tipo de bebidas alcohólicas fermentedas, ya que dispone de flexibilidad suficiente para poder ir implementándolo “a fascículos” o todo de una vez. Si se quisiera ir más allá, podrían integrarse sensores de pH del agua así como sus actuadores que permitan el tratamiento del agua, otra mejora seria la creación de una app

para dispositivos móviles que aprovechasen las capacidades de comunicación inalámbricas del ESP32. En la misma línea, se podrían implementar dosificadores de lúpulo o adjuntos que se vertiesen al mosto en el momento adecuado en vez de introducirlos manualmente.

Las posibles mejoras van todas encaminadas a una mayor automatización y menor dependencia del ser humano, lo que para algunas comunidades de “brewers” sería un paso atrás, ya que se volvería a una cerveza industrializada.

10.2. OBJETIVOS DE DESARROLLO SOSTENIBLE

De acuerdo con la agenda 2030, el siguiente proyecto está ligado con los objetivos:

- 8 – TRABAJO DECENTE Y CRECIMIENTO ECONÓMICO.
  - Al permitir a la gente crear sus propias bebidas, es posible llegar a comercializarlas permitiéndoles ser sus propios jefes y alcanzar la independencia económica.
- 9 -INDUSTRIA, INNOVACIÓN E INFRAESTRUCTURA
  - Al fomentar la innovación y la experimentación en la elaboración artesanal, se podrían descubrir e impulsar nuevas técnicas y tecnologías.
- 12 – PRODUCCIÓN Y CONSUMO RESPONSABLE
  - Al fomentar la producción a pequeña escala y con productos locales, se podría contribuir a un consumo más responsable y a la reducción del desperdicio en alimentos.

11. ANEXOS

11.1. CÓDIGO INTEGRO UI

```
#include <lvgl.h>
#include <TFT_eSPI.h>
#include <ui.h>
#include <XPT2046_Touchscreen.h>
#include <SPI.h>
#include <HardwareSerial.h> // Incluir la librería HardwareSerial, para enviar los datos con los pines que quiero
#include <ArduinoJson.h> // Para enviar los datos en formato JSON, más robusto


//Prototipos de funciones no incluidas en el ui
//Funciones de comunicacion entre placas
void enviarDatos(bool emergencia, bool bomba, int temperaturaEnviada);
void recibirDatos(void);
//funciones de tiempo
void iniciar_cuenta_atras(lv_obj_t * label, lv_obj_t * spinbox, int minutos);
// iniciar cuenta atras manual
void timer_callback_manual(lv_timer_t * timer_manual);
void timer_callback_maceracion(lv_timer_t * timer_maceracion);
void timer_callback_ebullicion(lv_timer_t * timer_ebullicion);
//funciones interaccion con usuario
void Alarma(void); // Prototipo de la función Alarma(), para que suene el buzzer
void reiniciarSpinboxesTiempo(void); //puesta a 0 de todos los spinboxes, al acabar los procesos o después de la seta
void actualizarLabelsTemperatura(void);


/*Don't forget to set Sketchbook location in File/Preferences to the path of your UI project (the parent foder of this INO file)*/
/*Change to your screen resolution*/
static const uint16_t screenWidth  = 240;
static const uint16_t screenHeight = 320;

static lv_disp_draw_buf_t draw_buf;
static lv_color_t buf[ screenWidth * screenHeight / 10 ];

TFT_eSPI tft = TFT_eSPI(screenWidth, screenHeight); /* TFT instance */

// Touchscreen pins
#define XPT2046_IRQ 36  // T_IRQ
#define XPT2046_MOSI 32 // T_DIN
```

```
#define XPT2046_MISO 39 // T_OUT
#define XPT2046_CLK 25  // T_CLK
#define XPT2046_CS 33  // T_CS

SPIClass tsSPI = SPIClass(VSPI);
XPT2046_Touchscreen touchscreen(XPT2046_CS, XPT2046_IRQ);

lv_timer_t * timer_manual; // Define timer_manual para el modo manual
lv_timer_t * timer_maceracion; // Temporizador para la maceración
lv_timer_t * timer_ebullicion; // Temporizador para la ebullición


//DEFINICION DE PINES LIBRES///
// Comunicación entre placas
#define RX_PIN 22       // Pin RX (GPIO 22)
#define TX_PIN 27       // Pin TX (GPIO 27)
//El pin de la alarma
#define PIN_BUZZER 26 // Pin GPIO al que está conectado el buzzer


//DEFINICION DE VARIABLES GLOBALES///
//variables globales para enviar, se van a usar en varias funciones
bool bomba = 0; // Variable para enviar si se activa la bomba o no
int temperaturaEnviada = 0; //Variable para enviar la temperatura manual
bool emergencia = 0;

//variables globales para recibir
float temperaturaRecibida = 0.0;
bool grifoAbierto = 0;

//Para los procesos
int temperaturasAuto[4] = {0,0,0,0};
int tiempoMaceracion[4] = {0,0,0,0};
int tiempoAvisoEbull[4] = {0,0,0,0};
bool bombaMaceracion[4] = {0,0,0,0};

// Variables para la cuenta atrás
static int segundos_restantes = 0; //Segundos restantes manual
static int segundosRestantesMaceracion = 0; // Segundos restantes para la
maceración
static int segundosRestantesEbullicion = 0; // Segundos restantes para la
ebullición
static int segundosRestantesPeriodoEbullicion = 0; // Nuevo contador para el
periodo en curso
int tiempoTotalMaceracion = 0;
bool maceradofin=0;
int indiceMaceracion = 0;
int indiceEbullicion = 0;
```

```
#if LV_USE_LOG != 0
/* Serial debugging */
void my_print(const char * buf)
{
    Serial.printf(buf);
    Serial.flush();
}
#endif

/* Display flushing */
void my_disp_flush( lv_disp_drv_t *disp, const lv_area_t *area, lv_color_t
*color_p )
{
    uint32_t w = ( area->x2 - area->x1 + 1 );
    uint32_t h = ( area->y2 - area->y1 + 1 );

    tft.startWrite();
    tft.setAddrWindow( area->x1, area->y1, w, h );
    tft.pushColors( ( uint16_t * )&color_p->full, w * h, true );
    tft.endWrite();

    lv_disp_flush_ready( disp );
}

/*Read the touchpad*/
void my_touchpad_read( lv_indev_drv_t * indev_driver, lv_indev_data_t * data )
{
    uint16_t touchX = 0, touchY = 0;
    TS_Point p = touchscreen.getPoint(); //yo
    /*Serial.print("Raw X: "); Serial.print(p.x);
    Serial.print(", Raw Y: "); Serial.print(p.y);
    Serial.print(", Pressure: "); Serial.println(p.z);*/

    data->point.x = map(p.x, 271, 3880, 0, screenWidth); //valores sacados
mediante calibración
    data->point.y = map(p.y, 183, 3796, 0, screenHeight); //valores sacados
mediante calibración

    if( p.z>60 ) //se está apretando la pantalla.
    {
        data->state = LV_INDEV_STATE_PR; //Pressed
        /*Serial.print("Raw X: "); Serial.print(p.x);
        Serial.print(", Raw Y: "); Serial.print(p.y);
        Serial.print(", Pressure: "); Serial.println(p.z);*/
    }
    else
    {
        data->state = LV_INDEV_STATE_REL; //Released
```

```
    }
}

void setup(){
    //Serial.begin( 115200 ); /* prepare for possible serial debug */

    String LVGL_Arduino = "Hello Arduino! ";
    LVGL_Arduino += String('V') + lv_version_major() + "." +
lv_version_minor() + "." + lv_version_patch();

    Serial.println( LVGL_Arduino );
    Serial.println( "I am LVGL_Arduino" );

    lv_init();

#if LV_USE_LOG != 0
    lv_log_register_print_cb( my_print ); /* register print function for
debugging */
#endif

//código inicializar touchscreen, los parámetros están definidos arriba, se
usa la libreria XPT2046
    tsSPI.begin(XPT2046_CLK, XPT2046_MISO, XPT2046_MOSI, XPT2046_CS);
    touchscreen.begin(tsSPI);
    touchscreen.setRotation(0); // Rotación del táctil a 0

    tft.begin();          /* TFT init */
    tft.setRotation(0); // Rotación pantalla a 0, debe coincidir con la táctil
    tft.invertDisplay(1); //Para que los colores no salgan invertidos

    lv_disp_draw_buf_init( &draw_buf, buf, NULL, screenWidth * screenHeight /
10 );

    /*Initialize the display*/
    static lv_disp_drv_t disp_drv;
    lv_disp_drv_init( &disp_drv );
    /*Change the following line to your display resolution*/
    disp_drv.hor_res = screenWidth;
    disp_drv.ver_res = screenHeight;
    disp_drv.flush_cb = my_disp_flush;
    disp_drv.draw_buf = &draw_buf;
    lv_disp_drv_register( &disp_drv );

    /*Initialize the (dummy) input device driver*/
    static lv_indev_drv_t indev_drv;
    lv_indev_drv_init( &indev_drv );
    indev_drv.type = LV_INDEV_TYPE_POINTER;
    indev_drv.read_cb = my_touchpad_read;
```

```
    lv_indev_drv_register( &indev_drv );

    ui_init();
    Serial.println( "Setup done" );

///Timers
  timer_manual = lv_timer_create(timer_callback_manual, 1000, NULL); // Crear
el temporizador (1 segundo de intervalo) para que no se bloquee la pantalla en
la cuenta atrás
  timer_maceracion = lv_timer_create(timer_callback_maceracion, 1000, NULL);
// Temporizador maceración
  timer_ebullicion = lv_timer_create(timer_callback_ebullicion, 1000, NULL);
// Temporizador ebullición
  //timer_verificar_grifo = lv_timer_create(verificar_grifo_callback, 100,
NULL);

  lv_timer_pause(timer_manual); // Pausar el temporizador al inicio para que
no vaya restando
  lv_timer_pause(timer_maceracion);
  lv_timer_pause(timer_ebullicion);
 // lv_timer_pause(timer_verificar_grifo); // Pausar el temporizador al inicio

//Monitores serie
  Serial.begin( 115200 ); /* prepare for possible serial debug */
  Serial2.begin(115200, SERIAL_8N1, RX_PIN, TX_PIN);//recibir y enviar datos
por el serial 2, comunicación entre placas
}



void loop(){
 // Serial.println("Inicio del bucle loop()");
  recibirDatos();
  enviarDatos(emergencia,bomba,temperaturaEnviada);
  lv_timer_handler(); /* let the GUI do its work */
  delay(5);

}



//////////////////////////////////////////////////FUNCIONES
AÑADIDAS/////////////////////////////////////////////////////////////////////
//////////////////////////

/////////////////////////////////////////////FUNCIONES DE CONTROL DE
TIEMPO/////////////////////////////////////////////////////
// Función para iniciar la cuenta atrás
```

```
void iniciar_cuenta_atras(lv_obj_t * label, lv_obj_t * spinbox, int minutos) {
    segundos_restantes = minutos * 60;
    lv_timer_resume(timer_manual); // Reanudar el temporizador de LVGL
}


void timer_callback_manual(lv_timer_t * timer_manual) {
    if (segundos_restantes > 0) {
        segundos_restantes--;
        // Actualizar el label de cuenta atrás manual
        int horas = segundos_restantes / 3600;
        int minutos = (segundos_restantes % 3600) / 60;
        int segundos = segundos_restantes % 60;
        lv_label_set_text_fmt(ui_LCuentaAtrasManual, "%02d:%02d:%02d", horas,
minutos, segundos);
        //Serial.print("timer_callback: ");Serial.println(segundos_restantes);
    } else {
        // Cuenta atrás manual terminada
        reiniciarSpinboxesTiempo();
        Serial.println("fin");
        lv_timer_pause(timer_manual);
        enviarDatos(false, false, 0);
        //Serial.println("cuenta atras terminada");
    }
}

void timer_callback_maceracion(lv_timer_t * timer_maceracion) {
    // Verificar si la maceración ha finalizado
    if (maceradofin == 1) {
        lv_label_set_text_fmt(ui_LTiempoRestante, "MACERACION FINALIZADA,\n
CIERRA EL GRIFO");
        lv_obj_add_state(ui_BHervirAuto1, LV_STATE_DISABLED);
        bomba=0;
        if (!grifoAbierto) {
            lv_obj_clear_state(ui_BHervirAuto1, LV_STATE_DISABLED); //
Habilitar botón
            lv_label_set_text_fmt(ui_LTiempoRestante, "GRIFO CERRADO");
            return; // Salir del callback si el grifo está cerrado
        }
    }
    else {
        // Verificar si la temperatura está en rango o si
tiempoMaceracion[indiceMaceracion] es cero
        if ((temperaturaRecibida >= temperaturaEnviada - 3 &&
temperaturaRecibida <= temperaturaEnviada + 3) ||
tiempoMaceracion[indiceMaceracion] == 0) {

            // Restar tiempo al periodo actual (solo si
tiempoMaceracion[indiceMaceracion] no es cero)
```

```
            if (tiempoMaceracion[indiceMaceracion] != 0) {
                segundosRestantesMaceracion--;
                tiempoTotalMaceracion--;
            }

            // Actualizar los labels de tiempo (solo si
tiempoMaceracion[indiceMaceracion] no es cero)
            if (tiempoMaceracion[indiceMaceracion] != 0) {
                lv_label_set_text_fmt(ui_LCuentaAtrasAuto, "%02d:%02d:%02d",
segundosRestantesMaceracion / 3600, (segundosRestantesMaceracion % 3600) / 60,
segundosRestantesMaceracion % 60);
                lv_label_set_text_fmt(ui_LTiempoRestante, "%02d:%02d:%02d",
tiempoTotalMaceracion / 3600, (tiempoTotalMaceracion % 3600) / 60,
tiempoTotalMaceracion % 60);
            }

            // Verificar si el periodo actual ha terminado o si
tiempoMaceracion[indiceMaceracion] es cero
            if (segundosRestantesMaceracion == 0 ||
tiempoMaceracion[indiceMaceracion] == 0) {
                // Incrementar el índice del periodo
                indiceMaceracion++;

                // Verificar si quedan periodos
                if (indiceMaceracion < 4) {
                    // Iniciar el siguiente periodo
                    segundosRestantesMaceracion =
tiempoMaceracion[indiceMaceracion] * 60;
                    bomba = bombaMaceracion[indiceMaceracion];
                    temperaturaEnviada = temperaturasAuto[indiceMaceracion];
                    enviarDatos(emergencia, bomba, temperaturaEnviada);
                } else {
                    // Finalizar la maceración
                    maceradofin = 1;
                    enviarDatos(false, false, 0); // Apagar bomba y calentador
                    lv_label_set_text_fmt(ui_LCuentaAtrasAuto, "COMPLETADO");
                }
            }
        } else {
            // La temperatura no está en rango, mostrar mensaje
            lv_label_set_text_fmt(ui_LTiempoRestante, "MODFICANDO
TEMPERATURA");
        }
    }
}


void timer_callback_ebullicion(lv_timer_t * timer_ebullicion) {
```

```
    // Restar tiempo al periodo actual y al tiempo total si no se desfasa
    if (segundosRestantesEbullicion > 0) {
        segundosRestantesEbullicion--;
    }
    if (segundosRestantesPeriodoEbullicion > 0) {
        segundosRestantesPeriodoEbullicion--;
    }

    // Actualizar los labels de tiempo
    lv_label_set_text_fmt(ui_LCuentaAtrasAutoEbuTotal, "%02d:%02d:%02d",
segundosRestantesEbullicion / 3600, (segundosRestantesEbullicion % 3600) / 60,
segundosRestantesEbullicion % 60);
    lv_label_set_text_fmt(ui_LCuentaAtrasAutoEbuPeriodo, "%02d:%02d:%02d",
segundosRestantesPeriodoEbullicion / 3600, (segundosRestantesPeriodoEbullicion
% 3600) / 60, segundosRestantesPeriodoEbullicion % 60);

    // Verificar si el periodo actual ha terminado
    if (segundosRestantesPeriodoEbullicion == 0) {
        // Activar la alarma
        Alarma();

        // Incrementar el índice del periodo
        indiceEbullicion++;

        // Verificar si quedan periodos
        if (indiceEbullicion < 4 && tiempoAvisoEbull[indiceEbullicion] > 0) {
// <-- Verificar si el tiempo del siguiente periodo es mayor que cero
            // Iniciar el siguiente periodo
            segundosRestantesPeriodoEbullicion =
tiempoAvisoEbull[indiceEbullicion] * 60;
        } else {
            // Finalizar la ebullición
            lv_timer_pause(timer_ebullicion); // Detener el temporizador
            lv_label_set_text_fmt(ui_LCuentaAtrasAutoEbuTotal, "COMPLETADO");
            lv_label_set_text_fmt(ui_LCuentaAtrasAutoEbuPeriodo,
"COMPLETADO");
            lv_obj_clear_state(ui_BFin, LV_STATE_DISABLED);
        }
    }
}


//////////////////////////////////////////////////FUNCIONES DE ACTUALIZACIÓN
IU Y
AVISO//////////////////////////////////////////////////////////////////////////
///////
void reiniciarSpinboxesTiempo() {
  // Spinboxes de maceración
  lv_spinbox_set_value(ui_STpoAuto1, 0);
```

```
  lv_spinbox_set_value(ui_STpoAuto2, 0);
  lv_spinbox_set_value(ui_STpoAuto3, 0);
  lv_spinbox_set_value(ui_STpoAuto4, 0);

  // Spinboxes de ebullición
  lv_spinbox_set_value(ui_STpoAutoAviso1, 0);
  lv_spinbox_set_value(ui_STpoAutoAviso2, 0);
  lv_spinbox_set_value(ui_STpoAutoAviso3, 0);
  lv_spinbox_set_value(ui_STpoAutoAviso4, 0);

  // Spinbox manual
  lv_spinbox_set_value(ui_STpo, 0);
}


void actualizarLabelsTemperatura(void) {
    // Labels en modo manual
  char tempStr[10]; // Buffer para almacenar la cadena de texto, que si no no
actualiza los labels..
  dtostrf(temperaturaRecibida, 6, 1, tempStr);

  lv_label_set_text(ui_LTempLeida, tempStr);
    lv_label_set_text(ui_LTempLeidaCaleManual, tempStr);
    lv_label_set_text(ui_LTempLeidaMaceAuto, tempStr);
    lv_label_set_text(ui_LTempLeidaEbullAuto, tempStr);
}

void Alarma() {
  tone(PIN_BUZZER, 1000); // Emitir un tono continuo de 1000 Hz
  delay(1000);            // Duración del tono (1 segundo)
  noTone(PIN_BUZZER);     // Detener el tono
}


////////////////////////////////////FUNCIONES AÑADIDAS EN SQUARELINE SE
INVOCAN DESDE LA
IU////////////////////////////////////////////////////////////////////////////
/////////////////

//Funcion de pantalla CaleManualActivado, se activa al acceder a la pantalla,
mira a ver si hay temporizador, lo activa y activa el calefactor a la
temperatura solicitada.
void CalefactorManual (lv_event_t * e){

    lv_label_set_text_fmt(ui_LTempObjCaleManual, "%d",
lv_spinbox_get_value(ui_STemp));
    int tempoManu = lv_obj_has_state(ui_CTemporizador,
LV_STATE_CHECKED);//checkbox de temporizador
    temperaturaEnviada = lv_spinbox_get_value(ui_STemp);
```

```
    if (tempoManu){
        int tpo_manu = lv_spinbox_get_value(ui_STpo);
        iniciar_cuenta_atras(ui_LCuentaAtrasManual, ui_STpo, tpo_manu); //
Iniciar la cuenta atrás si no se declara así peta, brujería!!
        enviarDatos(emergencia,bomba,temperaturaEnviada);
    }
    else{
      enviarDatos(emergencia,bomba,temperaturaEnviada);
    }
}

//Función de parada de emergencia, en los distintos botones de STOP
void ParadaEmergencia(lv_event_t * e) {
  emergencia = 1;
  reiniciarSpinboxesTiempo();
  lv_timer_pause(timer_manual);
  lv_timer_pause(timer_maceracion);
  lv_timer_pause(timer_ebullicion);
  enviarDatos(emergencia,bomba,0);
}


//Funcion para resetear la emergencia, se reseta al volver a la pantalla de
inicio
void resetEmergencia(lv_event_t * e) {
  emergencia = 0;
}


//Boton activar bomba en pantalla TempManual, si grifoAbierto=0, no debería de
estar habilitado el botón, pero se hace un doble check
void FactivarBomba(lv_event_t * e){
    if(grifoAbierto==1){
    bomba=1;
    }
}


//Boton parar bomba en pantalla TempManual, solamente se habilita al pulsr el
de activar bomba
void FpararBomba(lv_event_t * e){
  bomba=0;
}


//Funcion de tomar datos, se activa al darle al botón Empezar en la pantalla
"AutoHervido"
void TomarDatosAuto (lv_event_t * e){
// Obtener valores de los spinboxes de temperatura y tiempo
```

```
  temperaturasAuto[0] = lv_spinbox_get_value(ui_STempAuto1);
  tiempoMaceracion[0] = lv_spinbox_get_value(ui_STpoAuto1);
  //si no hay tiempo de maceración, ponemos la temperatura a 0 para no activar
el ssr
  if(tiempoMaceracion[0]==0){
    temperaturasAuto[0]=0;
  }
  else if (temperaturasAuto[0]==0){
    tiempoMaceracion[0]=0;
  }


  temperaturasAuto[1] = lv_spinbox_get_value(ui_STempAuto2);
  tiempoMaceracion[1] = lv_spinbox_get_value(ui_STpoAuto2);
  if(tiempoMaceracion[1]==0){
    temperaturasAuto[1]=0;
  }
  else if (temperaturasAuto[1]==0){
    tiempoMaceracion[1]=0;
  }

  temperaturasAuto[2] = lv_spinbox_get_value(ui_STempAuto3);
  tiempoMaceracion[2] = lv_spinbox_get_value(ui_STpoAuto3);
    if(tiempoMaceracion[2]==0){
    temperaturasAuto[2]=0;
  }
  else if (temperaturasAuto[2]==0){
    tiempoMaceracion[2]=0;
  }

  temperaturasAuto[3] = lv_spinbox_get_value(ui_STempAuto4);
  tiempoMaceracion[3] = lv_spinbox_get_value(ui_STpoAuto4);
    if(tiempoMaceracion[3]==0){
    temperaturasAuto[3]=0;
  }
  else if (temperaturasAuto[3]==0){
    tiempoMaceracion[3]=0;
  }

   //recirculado si o no
  bombaMaceracion[0] = lv_obj_has_state(ui_CRecirculadoAuto1,
LV_STATE_CHECKED);
  bombaMaceracion[1] = lv_obj_has_state(ui_CRecirculadoAuto2,
LV_STATE_CHECKED);
  bombaMaceracion[2] = lv_obj_has_state(ui_CRecirculadoAuto3,
LV_STATE_CHECKED);
  bombaMaceracion[3] = lv_obj_has_state(ui_CRecirculadoAuto4,
LV_STATE_CHECKED);
```

```
  //cada cuanto avisa durante la ebullicion
  tiempoAvisoEbull[0] = lv_spinbox_get_value(ui_STpoAutoAviso1);
  tiempoAvisoEbull[1] = lv_spinbox_get_value(ui_STpoAutoAviso2);
  tiempoAvisoEbull[2] = lv_spinbox_get_value(ui_STpoAutoAviso3);
  tiempoAvisoEbull[3] = lv_spinbox_get_value(ui_STpoAutoAviso4);

  tiempoTotalMaceracion = 0;
  for (int i = 0; i < 4; i++) {
      tiempoTotalMaceracion += tiempoMaceracion[i] * 60;
  }
  lv_obj_add_state(ui_BFin, LV_STATE_DISABLED);
}


//Función de iniciar la maceración, se acitva al cargar la pantalla
AutoMaceradoStatus
void IniciarMaceracion(lv_event_t * e) {
    maceradofin=0;
    indiceMaceracion = 0; // Índice del periodo actual (0 al inicio)
    segundosRestantesMaceracion = tiempoMaceracion[indiceMaceracion] * 60; //
Inicializar el tiempo del primer periodo
    bomba=bombaMaceracion[indiceMaceracion];
    temperaturaEnviada=temperaturasAuto[indiceMaceracion];
    // Enviar datos del primer periodo
    enviarDatos(emergencia, bomba, temperaturaEnviada);
    //Serial.print(emergencia);Serial.print(bombaMaceracion[indiceMaceracion])
;Serial.println(temperaturasAuto[indiceMaceracion]);
    // Iniciar el temporizador
    lv_timer_resume(timer_maceracion);
}


void Ebullicion(lv_event_t * e) {
    static int indiceEbullicion = 0; // Índice del periodo actual
    // Calcular el tiempo total de ebullición y reiniciar el contador
    segundosRestantesEbullicion = 0;

    for (int i = 0; i < 4; i++) {
        segundosRestantesEbullicion += tiempoAvisoEbull[i] * 60;
    }
    // Inicializar el tiempo restante del primer periodo
    segundosRestantesPeriodoEbullicion = tiempoAvisoEbull[indiceEbullicion] *
60;

    // Verificar temperatura antes de iniciar
    if (temperaturaRecibida >= 97.0) {
        // Enviar datos a la otra placa (sin emergencia, sin bomba,
temperatura objetivo alta)
        enviarDatos(0, 0, 100);
```

```
        lv_timer_resume(timer_ebullicion);
    } else {
        // Mostrar mensaje si la temperatura no es suficiente
        lv_label_set_text_fmt(ui_LCuentaAtrasAutoEbuTotal, "MODIFICANDO\n
TEMPERATURA");
    }
}



////////////////////////////FUNCIONES DE COMUNICACION ENTRE
PLACAS/////////////////////////////////////////////
void enviarDatos(bool emergencia, bool bomba, int temperaturaEnviada){
  StaticJsonDocument<128> datosEnviar; // Buffer de 256 bytes, más vale que
sobre

  datosEnviar["ParadaEmergencia"] = emergencia;// Para el calefactor y corta
los reles
  datosEnviar["ActivarBomba"] = bomba; // Activación manual bomba
  datosEnviar["temperaturaEnviada"] = temperaturaEnviada;// Temperatura a
enviar

  String jsonEnviar;
  serializeJson( datosEnviar, jsonEnviar);
  Serial2.println(jsonEnviar);

  //Imprimir los datos enviados para depuración
  /*Serial.println("Datos enviados:");
  Serial.print("  Parada Emergencia: "); Serial.println(emergencia);
  Serial.print("  Activar Bomba: "); Serial.println(bomba);
  Serial.print("  temperaturaEnviada: ");
Serial.println(temperaturaEnviada);*/
}


void recibirDatos() {//
  if (Serial2.available()) {
    String jsonRecibido = Serial2.readStringUntil('\n');

    StaticJsonDocument<128> doc; // Hay que ajustar para que no de muchos
fallos
    DeserializationError error = deserializeJson(doc, jsonRecibido);

    if (error) {
      Serial.print(F("Error al deserializar JSON: "));
      Serial.println(error.f_str());
      return;
    }
    else{
```

```
      temperaturaRecibida = doc["temperaturaLeida"];
      grifoAbierto = doc["grifoAbierto"];
      /*Serial.println("Datos recibidos:");
      Serial.print("  Temperatura: "); Serial.println(temperaturaRecibida);
      Serial.print("  grifoAbierto: "); Serial.println(grifoAbierto);*/
      actualizarLabelsTemperatura();
      if (grifoAbierto==1){
        lv_obj_set_style_bg_color(ui_BGrifoLeer, lv_color_hex(0x007400), 0);
        lv_obj_clear_state(ui_BActivarBombaManual, LV_STATE_DISABLED);
        if(bomba==1){
          lv_obj_add_state(ui_BActivarBombaManual, LV_STATE_DISABLED);
          lv_obj_clear_state(ui_BPararBombaManual, LV_STATE_DISABLED);

        }
      }
      else{
        lv_obj_set_style_bg_color(ui_BGrifoLeer, lv_color_hex(0xFC0000), 0);
        lv_obj_add_state(ui_BActivarBombaManual, LV_STATE_DISABLED);
      }

    }
  }
}
```

11.2. CÓDIGO INTEGRO PLACA2

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <HardwareSerial.h>
#include <ArduinoJson.h>
#include <PID_v1.h>

//Prototipos de funciones para evitar errores
// Funciones de comunicación
void recibirDatos(void);  // Recibe datos (JSON) de la placa 1
void enviarDatosPlaca2(void); // Envía datos (JSON) a la placa 1
// Funciones de sensores
void leerTemperaturaSensor(void); // Lee la temperatura del sensor DS18B20
void leerEstadoGrifo(void); // Lee el estado del grifo (abierto o cerrado)
// Funciones de actuadores
void ActivarBomba(void); // Activa o desactiva la bomba según las condiciones
void activarCalentador(void); // Controla el calentador mediante PID



//////////////DEFINICION DE PINES/////////////
//Comunicacion entre placas
#define ONE_WIRE_BUS 32 // Pin de datos del DS18B20
#define TX_PIN 27 // Pin TX (GPIO 1)
#define RX_PIN 14 // Pin RX (GPIO 3)
//Actuadores y sensores
#define PIN_BOMBA 2 //Pin activar relé bomba
#define PIN_CALENTADOR 4 //Pin activar SSR, salida del PID
#define PIN_GRIFO 16 //Pin sensor XKC-Y28, que indica el grifo abierto


//////////////Variables globales recibidas de la otra placa///////////////
bool emergenciaRecibida = false;
bool bombaRecibida = false;
int temperaturaRecibida = 0;
//Por si falla el JSON
bool emergenciaAnterior = false;
bool bombaAnterior = false;
float temperaturaAnterior = 0.0;

//////Variables globales enviadas///////
bool grifoAbiertoEnviado = 0;
float temperaturaLeida = 0.0;

////// Variables para el PID*///////
double Setpoint, Input, Output;
double Kp = 5, Ki = 10, Kd = 1;
```

```
int OutputMin = 0, OutputMax = 255; // Límites de salida (0-255 para PWM)


//Crear instancias
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);



void setup() {
  Serial.begin(115200);
  Serial2.begin(115200, SERIAL_8N1, RX_PIN, TX_PIN);
  sensors.begin();

  pinMode(PIN_GRIFO, INPUT);
  pinMode(PIN_BOMBA, OUTPUT);
  pinMode(PIN_CALENTADOR, OUTPUT);


  digitalWrite(PIN_BOMBA, LOW);
  digitalWrite(PIN_CALENTADOR, LOW);

  myPID.SetMode(AUTOMATIC);
  myPID.SetOutputLimits(OutputMin, OutputMax);
  myPID.SetSampleTime(100);
}


void loop() {
  leerTemperaturaSensor();
  leerEstadoGrifo();
  enviarDatosPlaca2(); // Enviar datos a la placa 1
  recibirDatos(); // Recibir datos de la placa 1
  if (emergenciaRecibida) {
    // Parada de emergencia: apagar bomba y calentador
    digitalWrite(PIN_BOMBA, LOW);
    digitalWrite(PIN_CALENTADOR, LOW);
  }
  else{
    ActivarBomba();
    activarCalentador();
  }

  delay(5); // Retardo igual al de la placa 1
}
```

```
////////////////////////////////////FUNCIONES
AÑADIDAS//////////////////////////////////////
////////FUNCIONES DE COMUNICACION////////////////////

//Función para recibir
/*void recibirDatos() {
  if (Serial2.available()) {
    String jsonRecibido = Serial2.readStringUntil('\n');

    StaticJsonDocument<128> doc;
    DeserializationError error = deserializeJson(doc, jsonRecibido);
    //Serial.println(jsonRecibido); //ver que llega por si hay errores

    if (error) {
      Serial.print(F("Error al deserializar JSON: "));
      Serial.println(error.f_str());
      return;
    }

    // Leer los valores del JSON
    emergenciaRecibida = doc["ParadaEmergencia"];
    bombaRecibida = doc["ActivarBomba"];
    temperaturaRecibida = doc["temperaturaEnviada"];

    // Imprimir los datos recibidos
    Serial.println("Datos recibidos:");
    Serial.print("  Emergencia: "); Serial.println(emergenciaRecibida);
    Serial.print("  Bomba: "); Serial.println(bombaRecibida);
    Serial.print("  Temperatura: "); Serial.println(temperaturaRecibida);
  }
}*/

void recibirDatos() {
    if (Serial2.available()) {
        String jsonRecibido = Serial2.readStringUntil('\n');

        StaticJsonDocument<128> doc;
        DeserializationError error = deserializeJson(doc, jsonRecibido);

        if (!error) {
            // Solo actualizar si la deserialización fue exitosa
            emergenciaAnterior = doc["ParadaEmergencia"];
            bombaAnterior = doc["ActivarBomba"];
            temperaturaAnterior = doc["temperaturaEnviada"];

            // Imprimir los datos recibidos
            Serial.println("Datos recibidos:");
            Serial.print(" Emergencia: "); Serial.println(emergenciaAnterior);
```

```
            Serial.print(" Bomba: "); Serial.println(bombaAnterior);
            Serial.print(" Temperatura: ");
Serial.println(temperaturaAnterior);
        } else {
            // Usar los valores anteriores en caso de error
            emergenciaRecibida = emergenciaAnterior;
            bombaRecibida = bombaAnterior;
            temperaturaRecibida = temperaturaAnterior;

            Serial.println("Error en el JSON, usando valores anteriores:");
            Serial.print(" Emergencia: "); Serial.println(emergenciaRecibida);
            Serial.print(" Bomba: "); Serial.println(bombaRecibida);
            Serial.print(" Temperatura: ");
Serial.println(temperaturaRecibida);
        }
    }
}


//Funcion para enviar datos desde la placa 2
void enviarDatosPlaca2() {
  StaticJsonDocument<128> datosEnviar; // Igual que lo recibido

  datosEnviar["temperaturaLeida"] = temperaturaLeida; //hay que poner lo que
la otra placa espera recibir
  datosEnviar["grifoAbierto"] = grifoAbiertoEnviado;

  String jsonEnviar;
  serializeJson(datosEnviar, jsonEnviar);
  Serial2.println(jsonEnviar);

  //Imprimir los datos enviados para depuración
  Serial.println("Datos enviados:");
  Serial.print("  Temperatura: "); Serial.println(temperaturaLeida);
  Serial.print("  Grifo abierto: "); Serial.println(grifoAbiertoEnviado);
}




//////////////////////////FUNCIONES DE LOS SENSORES//////////////
//Leer la temperatura
void leerTemperaturaSensor() {
  sensors.requestTemperatures(); // Solicitar la medición de temperatura
  temperaturaLeida = sensors.getTempCByIndex(0); // Leer la temperatura del
primer sensor


      /* Mostrar la temperatura en el Monitor Serie
```

```
    char tempString[10];
    dtostrf(temperaturaLeida, 6, 2, tempString);
    Serial2.print(tempString);
    Serial2.println(" C");*/
}

//Leer el estado del grifo
void leerEstadoGrifo() {
  grifoAbiertoEnviado = digitalRead(PIN_GRIFO) == 1; //Grifo abierto = 1
}


////////////////////////////////FUNCIONES DE LOS
ACTUADORES////////////////////////////////
//Activar la bomba de recirculado
void ActivarBomba() {
  if (bombaRecibida && grifoAbiertoEnviado==1) {
    digitalWrite(PIN_BOMBA, HIGH); // Activar la bomba
    //Serial.println("Bomba activada"); // Mensaje de depuración
  }
  else {
    digitalWrite(PIN_BOMBA, LOW);  // Desactivar la bomba
    //Serial.println("Bomba desactivada"); // Mensaje de depuración
  }
}

//Activar el calentador//
void activarCalentador() {
  Input = temperaturaLeida;
  Setpoint = temperaturaRecibida;
  myPID.Compute();              // Calcular la salida del PID
  analogWrite(PIN_CALENTADOR, Output); // Controlar el calentador con PWM
}
```

11.3. CÓDIGO TOMA DE DATOS EXPERIMENTO PID

```
#include <WiFi.h>           // Biblioteca WiFi para ESP32
#include <ThingSpeak.h>    // Biblioteca ThingSpeak
#include <OneWire.h>
#include <DallasTemperature.h>

// Configuración WiFi
const char* ssid = "Internet";
const char* password = "contraseñadeinternet";
WiFiClient client;

// Configuración ThingSpeak
unsigned long myChannelNumber = 2582421; // Channel ID
const char* myWriteAPIKey = "YM1QOS1MYHTRZU68";

// Pin del sensor
#define ONE_WIRE_BUS 14

// Objetos
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

void setup() {
    Serial.begin(115200);

    // Conexión a WiFi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nConectado a WiFi!");

    // Inicializar ThingSpeak
    ThingSpeak.begin(client);
    sensors.begin();
}

void loop() {
    // Leer la temperatura
    sensors.requestTemperatures();
    float temperatureC = sensors.getTempCByIndex(0);
    Serial.println(temperatureC); //Que la muestre en el monitor para
asegurarme que funciona.

    // Enviar datos a ThingSpeak
    ThingSpeak.setField(1, temperatureC);
```

```
    int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
    if (x == 200) {
        Serial.println("Datos enviados a ThingSpeak correctamente.");
    } else {
        Serial.println("Error al enviar datos a ThingSpeak (" + String(x) +
").");
    }

    // Esperar 30 segundos
    delay(30000);
}
```

11.4. DATASHEETS

XKC-Y28 DATASHEET
1686719313.pdf (xkc-sensor.com)

DS18B20 DATASHEET
DS18B20 Datasheet(PDF) - Dallas Semiconductor (alldatasheet.com)

ESP32-WROOM-32
ESP32-WROOM-32.PDF