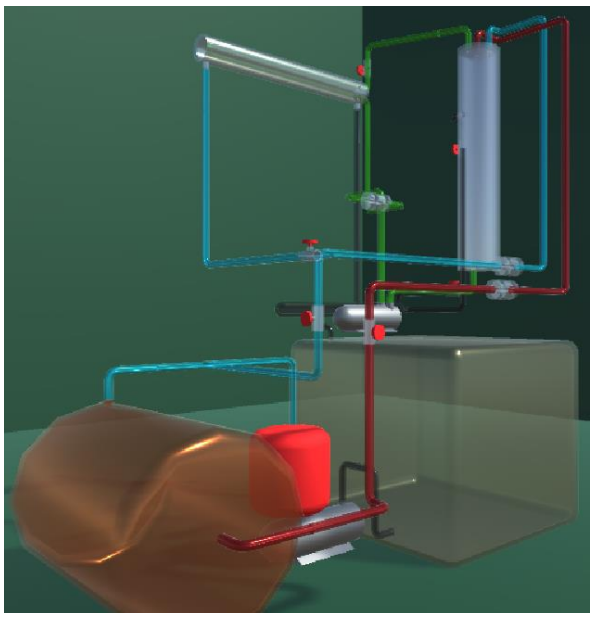
Proyecto Fin de Máster

Máster en Ingeniería Industrial



Diseño y desarrollo de un gemelo digital en Unity 3D de una planta de reacción exotérmica y simulación de su depósito

Autor: Jose Antonio Sáinz-Cantero Paredes

Tutores:

Juan Manuel Escaño González

Paula Chanfreut Palacio

Dpto. de ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Proyecto Fin de Máster
Máster en Ingeniería Industrial

Diseño y desarrollo de un gemelo digital en Unity 3D de una planta de reacción exotérmica y simulación de su depósito

Autor:
Jose Antonio Sáinz-Cantero Paredes

Tutores:
Juan Manuel Escaño González
Paula Chanfreut Palacio

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Proyecto Fin de Máster: Diseño y desarrollo de un gemelo digital en Unity 3D de una planta de reacción exotérmica y simulación de su depósito

Autor: Jose Antonio Sáinz-Cantero Paredes

Tutor: Juan Manuel Escaño González

Paula Chanfreut Palacio

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

A mis padres, por su apoyo incondicional.

A mi hermana, por escucharme y darme un punto de vista distinto.

Agradecimientos

Me gustaría dedicar estas palabras a mi familia, durante todo el proceso que ha sido el desarrollo de este trabajo han demostrado ser un apoyo indispensable para mí: alentándome, dándome fuerzas en los momentos más difíciles, y sirviéndome de inspiración, no solo por aportar un punto de vista distinto al mío, sino por escucharme y permitirme organizar mis ideas. Gracias.

Jose Antonio Sáinz-Cantero Paredes

Sevilla, 2024

Resumen

La industria 4.0 y el desarrollo de los gemelos digitales ha supuesto un gran avance en los campos de la ingeniería industrial, permitiendo no solo facilitar la visualización de un proyecto del mundo real en un entorno digital, sino también realizar modificaciones en el entorno digital para su posterior implementación en la realidad. Con esta filosofía, este trabajo desarrolla un gemelo digital de una planta piloto de reacción exotérmica por medio de Unity 3D y un software de simulación de fluidos llamado Zibra Liquids. Este último se utiliza para visualizar los cambios de la altura del agua dentro de un depósito a partir de los valores que tomen los flujos de entrada de agua caliente y fría. La finalidad de este trabajo es la servir como base de futuros proyectos que impliquen tanto la experimentación con la planta en el laboratorio como su uso en docencia.

Abstract

Industry 4.0 and the development of digital twins have been a great advance in the industrial engineering field, allowing not only to facilitate the visualization of a real-world project in a digital environment, but also to make modifications in the digital environment. With this philosophy, this work develops a digital twin of an exothermic reaction pilot plant using Unity 3D and a fluid simulation software called Zibra Liquids, the latter is used to visualize changes in the watert height level inside a tank based on inlet flows of hot and cold water. The purpose of this work is to serve as a basis for future projects that involve both experimentation with the plant in the laboratory and its use in teaching.

Índice

Índice de figuras

Índice de tablas

1 INTRODUCCIÓN

En este capítulo se realizará una descripción del estado del arte sobre la tecnología de los gemelos digitales y su aplicación en la industria y se expondrá la motivación y objetivo de este trabajo.

1.1. Contexto y estado del arte

En los últimos años se han producido una gran cantidad de avances en distintos campos tecnológicos como la robótica o la inteligencia artificial, que ha influido en todos los aspectos de la vida de las personas, desembocando en el nacimiento de una nueva revolución industrial y permitiendo el desarrollo de los gemelos digitales.

En las siguientes subsecciones se realizará una descripción de los avances mencionados anteriormente, comenzando con la presentación de la 4ª Revolución Industrial y sus orígenes, y terminando con una con una introducción del concepto de gemelo digital y de algunos de ejemplos del uso de esta tecnología.

1.1.1 Industria 4.0

Los recientes cambios y avances en la tecnología han tenido un fuerte impacto en la industria, pudiendo afirmarse que está en proceso de gestación la 4ª Revolución Industrial, también conocida como Industria 4.0 que va a cambiar la forma en la que se desarrolla y fabrica un producto.

El término "Industria 4.0" fue utilizado por primera vez en 2016 [1] por Klaus Schwab, aunque se estima que esta nueva revolución industrial se inició en 2014 debido a la propagación de las fábricas inteligentes (consideradas así porque utilizan medios virtuales y físicos para funcionar) y al uso de la gestión online. En definitiva, lo que se pretende conseguir con esta nueva revolución es la automatización de los procesos, obteniendo una mayor rapidez y eficiencia gracias a la digitalización, produciéndose así una comunicación constante entre el mundo físico y el digital.

Para poder lograr ese objetivo, la 4ª Revolución industrial está basada en los pilares tecnológicos [2] que se muestran en la Figura 1-1 y que se describen a continuación:

- Internet de las Cosas: también conocida como IoT, permite la conexión entre el medio físico y el digital utilizando Internet, permitiendo así una comunicación constante no solo entre el operario y la máquina, sino también entre las propias máquinas.
- La Nube: debido a la cantidad de datos que se generan es necesaria una tecnología para almacenarlos y poder acceder a ellos en cualquier momento. Por lo tanto, el servicio de almacenamiento digital de la Nube se hace imprescindible.
- *Big Data*: este término hace referencia al almacenamiento, procesamiento y análisis de forma eficiente de todos los datos generados.
- Impresión 3D o fabricación aditiva: permite un rápido desarrollo de prototipos o de productos personalizados, pudiendo además realizarse una descentralización del proceso de fabricación al no necesitar estar el diseñador en el mismo sitio que la impresora.
- Robótica: el uso de robots cooperativos va en aumento por lo que no resulta raro encontrar fábricas donde humanos y máquinas interactúan mejorando las capacidades de ambos.

- Inteligencia Artificial: los avances en la inteligencia artificial permiten, gracias a la obtención de numerosos datos, la predicción de situaciones futuras dentro de los procesos industriales, o el desarrollo de nuevos procedimientos más eficientes.
- Realidad aumentada: permitirá en un futuro a los trabajadores ver el producto o la maquinaria, en tiempo real y comprobar los efectos de posibles modificaciones o de tareas de mantenimiento.
- Seguridad: debido al constante intercambio de información la seguridad deberá mejorar también, evitando perdidas de datos en las comunicaciones y limitar el acceso de personas ajenas a las fabrica en los sistemas informáticos o de la maquinaria.
- Plataformas sociales: las redes sociales tienen una gran influencia en el día a día y resultan imprescindibles para comunicarse con los clientes, ya que la interacción con el cliente es de suma importancia para la industria.

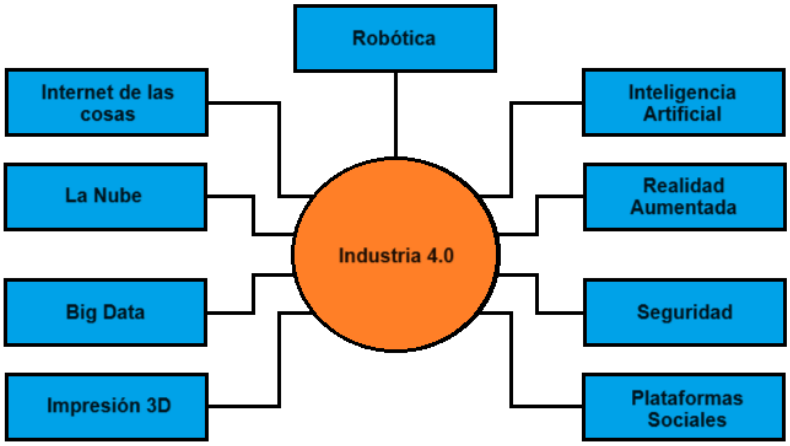


Figura 1- 1: Pilares tecnológicos principales de la Industria 4.0

1.1.2 Gemelos digitales

Dos de los principales pilares de la Industria 4.0 son la realidad aumentada y la comunicación entre el mundo físico y el digital, y fruto de la relación de estos pilares surgieron los gemelos digitales.

Los gemelos digitales o *digital twins* en inglés (abreviado como DT) son representaciones virtuales de elementos o procesos reales y por medio de datos obtenidos en el pasado o en tiempo real, consigue ser un reflejo fiel con el cual es posible llegar a predecir comportamientos futuros, bajo distintas condiciones [3]. Esto último permitiría comprobar la viabilidad de futuros cambios en dichos elementos o procesos, además de comprobar posibles limitaciones.

Se considera que el desarrollo de los gemelos digitales ha pasado por tres fases principales en base a las publicaciones respecto a este tema, las cuales se pueden observar en la Figura 1-2 y se definen a continuación [4]:

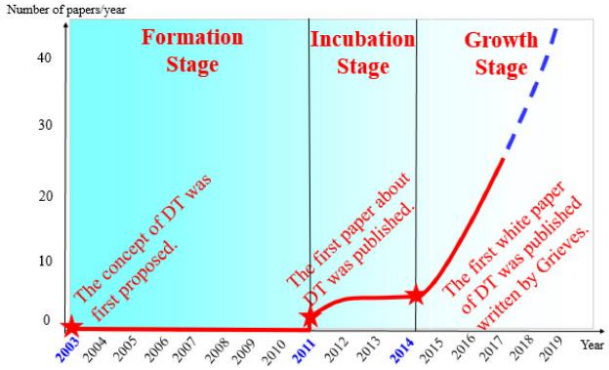


Figura 1- 2: Fases históricas del desarrollo de los gemelos digitales [4]

- Fase de formación: se inició en 2003 cuando se introdujo el concepto de gemelo digital, considerándose que estaba compuesto por un elemento físico y otro digital, además de un medio de comunicación entre dichos elementos [5]. En la Figura 1-3 se muestra un esquema simplificando dichos elementos. En estos momentos el desarrollo de los gemelos digitales es más teórico que práctico (ya que se realizaron pocas publicaciones debido a las limitaciones en las tecnologías de la información y las comunicaciones).

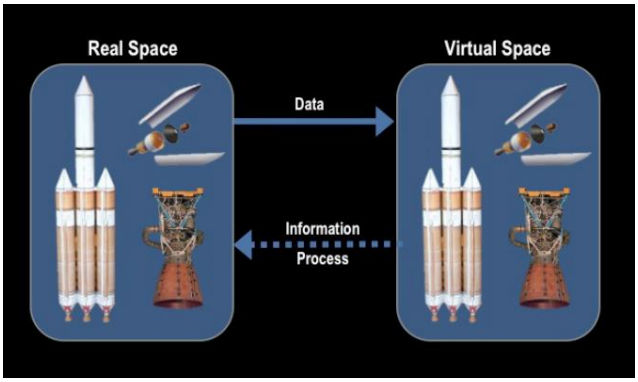


Figura 1- 3: Esquema simplificado de un gemelo digital [5]

- Fase de incubación: a partir de 2011 se demostró la utilidad de los gemelos digitales en diferentes ámbitos, como el de la aeronáutica, donde sirvieron para predecir el comportamiento de diferentes elementos estructurales de un avión durante sus ciclos de vida [6]. Posteriormente, en 2012, la NASA consolidaría la definición de los gemelos digitales, planteando su uso para realizar pruebas en los transbordadores espaciales en un entorno digital, para posteriormente realizar cambios en ellos en el mundo físico [7]. Esto además permitió anticiparse a situaciones que no se habían podido predecir durante el proceso de diseño de las estructuras.
- Fase de crecimiento: en esta última fase, que surge a partir de 2014 se puede apreciar un aumento en el interés en esta tecnología, ya que el número de publicaciones es mayor y se ha experimentado con gemelos digitales en distintos ámbitos de la industria con resultados satisfactorios. Además, en 2017 se propone una actualización del modelo de gemelo digital constando ahora de dos nuevos elementos que son los datos necesarios para el funcionamiento del gemelo digital, y los servicios que comprenden las funciones esperadas tanto del entorno físico como del entorno digital [8].

  En la Figura 1-4 se muestra un esquema de esta estructura de cinco elementos de un gemelo

digital aplicado a una planta de montaje. En el esquema el motor principal de la comunicación son los datos recopilados de por el gemelo digital (*Shop-floor Digital Twin Data*) del entorno físico (*Physical Shop-floor*) que comprende la maquinaria, operarios y materiales; el entorno digital (*Virtual Shop-floor*) y de los servicios de la empresa (*Shop-floor Service System*) los cuales son las funciones que la empresa espera que el entorno físico y digital desempeñen. Los datos del gemelo digital permiten optimizar las funciones de los demás elementos gracias a la comunicación constante de información entre ellos.

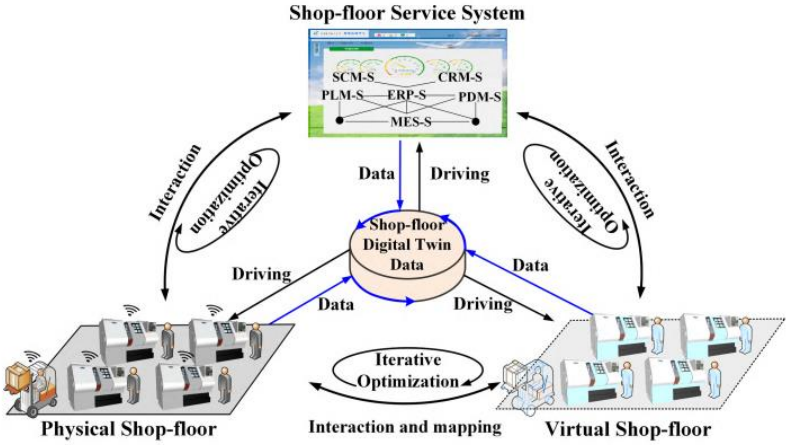


Figura 1- 4:Esquema actualizado de un gemelo digital [8]

Como se ha mencionado, el uso de gemelos virtuales se ha extendido a muchas las industrias, gracias a su flexibilidad y utilidad, por lo que es posible encontrar múltiples ejemplos de su uso como se indica a continuación:

- Industria aeronáutica [9]: se utiliza esta tecnología para desarrollar modelos de los diferentes componentes de sus aviones permitiendo probarlos en diferentes condiciones, mejorar su calidad, y reducir el tiempo y coste de fabricación.
- Industria petrolífera [10]: en este tipo de industria los gemelos virtuales se utilizan para simular los diferentes estados de los pozos de extracción y realizar ensayos con los taladros. Además, durante la operación de taladrado se capta información real para posteriormente estudiarla.
- Industria ganadera [11]: por medio del uso de diferentes sensores aplicados al ganado es posible realizar simulaciones y analizar el estado del ganado, así como la temperatura y la humedad de las instalaciones antes de construirlas para que mejore la comodidad de los animales y predecir posibles conductas peligrosas. En el esquema de la Figura 1-5 se representa la comunicación de los sensores y del gemelo digital con la base de datos de los diferentes análisis realizados.

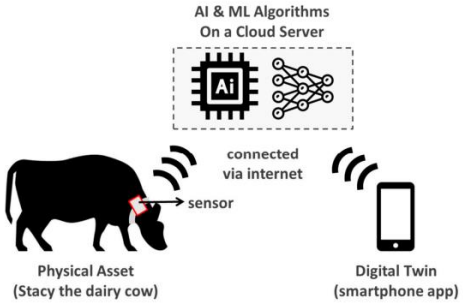


Figura 1- 5: Esquema de la comunicación entre los sensores colocados en un animal y el gemelo digital

- Ciudades inteligentes (*smart cities* en inglés) [12]: combinando los gemelos digitales con las ciudades inteligentes es posible desarrollar un entorno virtual donde se puede experimentar con el crecimiento de la ciudad en diferentes ámbitos (nuevas construcciones, sistemas para prevenir inundaciones…), permitiendo también la simulación del comportamiento de la población. El gemelo podría también dar la oportunidad de conocer la opinión de los ciudadanos al facilitar que estos visualicen los futuros proyectos o cambios que se quieren introducir. En la Figura 1-6 se muestra una comparativa entre una calle real y la misma calle modificada con nuevas construcciones en el gemelo digital.

(a) Current Street View

(b) Current Digital Twin

Figura 1- 6: Comparativa de la ciudad en la realidad (imagen superior) con la ciudad creada en el gemelo digital (imagen inferior)

- Medicina [13]: los gemelos digitales pueden aplicarse de muchas formas en el campo de la medicina, desde la simulación para la administración de hospitales como en el desarrollo de pacientes digitales, para conocer la viabilidad de las operaciones. Un ejemplo de paciente digital se puede apreciar en la Figura 1-7 donde se muestra un gemelo digital de un corazón.

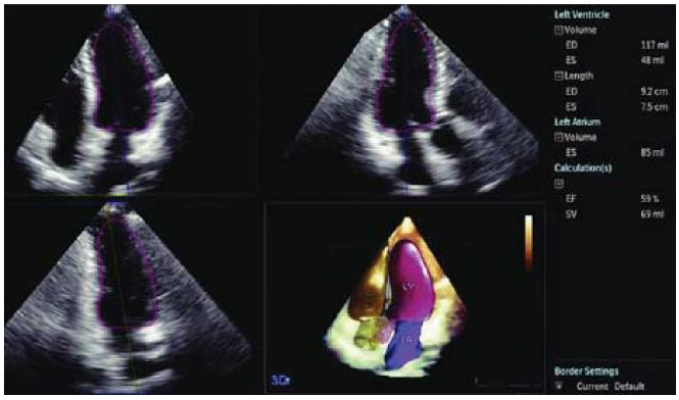


Figura 1- 7: Gemelo digital de un corazón humano

Para poder diseñar gemelos virtuales son necesarios programas de simulación que permitan tanto desarrollar modelos en 3D de los productos o instalaciones a desarrollar, así como tener la capacidad de tratar numerosos datos para ser una réplica lo más parecida al original. Algunos ejemplos de gemelos digitales realizados con distintos programas se observan en la Figura 1-8.

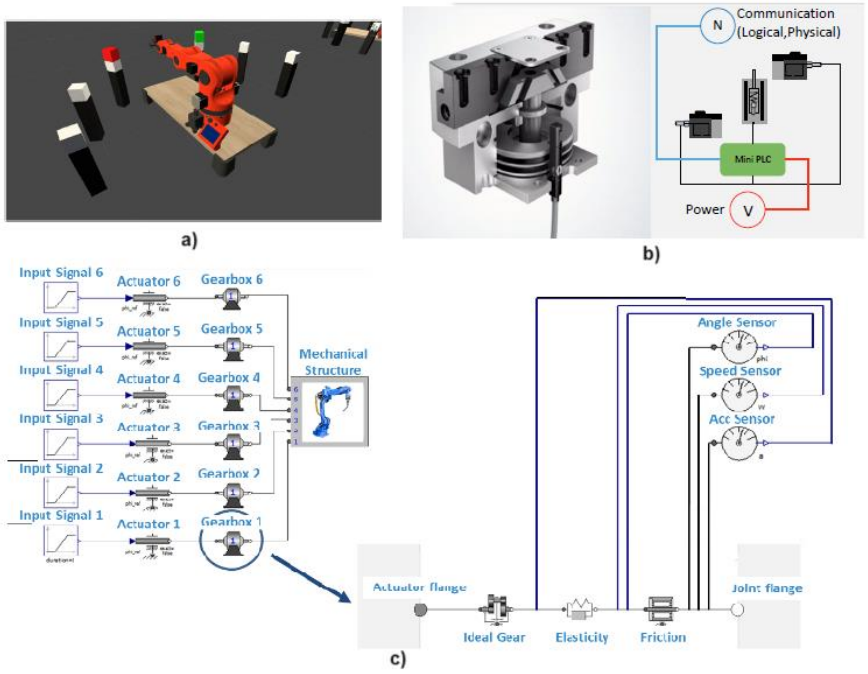


Figura 1- 8:Ejemplos de gemelos digitales hechos con a) Unity [18]. b) Modelica [14] y c) Matlab [16]

Algunos de los programas más utilizados para el desarrollo y diseño de los gemelos digitales son:

- Modelica [14]: es un entorno de programación que permite modelar objetos de forma virtual, siendo estos controlados por ecuaciones matemáticas. Permite por ejemplo desarrollar un gemelo digital de una pieza industrial para comprobar la efectividad de las tareas de mantenimiento correctivo en ella. Dada la versatilidad de este programa, es muy utilizado en conjunción con otros programas, algunos de los cuales se describen más abajo.

- ANSYS: es un programa de simulación para diferentes campos de la ingeniería, como pueden ser la fluidomecánica o la ingeniería estructural. Se puede utilizar junto a un modelo diseñado con Modelica para desarrollar un gemelo digital de un sistema de frenos [15] del cual se quiere monitorizar los cambios de temperatura y poder definir tareas de mantenimiento predictivo.
- MATLAB: este programa pese a ser principalmente de cálculo, permite desarrollar modelos y algoritmos útiles para desarrollar gemelos digitales. Por ejemplo, en [16] se describe un ejemplo de un sistema predictivo de mantenimiento, donde más que simular el modelo físico en sí, se utiliza Matlab para simular sus comportamientos, sensores y parámetros, para poder actuar luego sobre el objeto real.
- Siemens Plant Simulator [17]: esta herramienta desarrollada por Siemens que permite desarrollar gemelos digitales de plantas industriales, permitiendo realizar esas simulaciones a distintos niveles: fábrica, línea de montaje o maquinaria.
- UNITY: es una plataforma de desarrollo principalmente de videojuegos, pero al tener un motor físico permite desarrollar simulaciones de sistemas reales. Una de sus posibles aplicaciones es el desarrollo de un gemelo digital de un brazo robótico para entrenar su inteligencia artificial en diferentes escenarios colocando las piezas en distintos lugares [18].

1.2. Motivación y objetivos del trabajo

Como se ha visto anteriormente, los avances en la tecnología de los gemelos digitales están desarrollando nuevas oportunidades para la predicción del comportamiento de un sistema ante posibles modificaciones de su funcionamiento, aumentando así la capacidad de reacción frente a dichas modificaciones.

El objetivo de este proyecto será el de diseñar un gemelo digital de la planta química presente en el laboratorio de Robótica y Automática de la Escuela Técnica Superior de Ingeniería de Sevilla, y simular el cambio de altura del agua dentro de su depósito en función de los caudales de entrada de agua caliente y fría y del caudal de salida por acción de la gravedad.

El desarrollo del gemelo digital se realizara en dos partes: una parte se correspondiente al diseño y montaje de la planta utilizando AutoCAD, y otra parte la programación y simulación por medio del programa Unity 3D y un *asset* específico para la simulación de fluidos llamado Zibra, consiguiendo así un modelo cuya simulación represente un funcionamiento lo más parecido al de la planta real, tras lo cual se validará por medio de una ecuación que represente el comportamiento del depósito utilizando un método numérico.

Para alcanzar los objetivos de este proyecto, en los siguientes capítulos se describirán los pasos que se han seguido. El contenido de cada capítulo se presenta a continuación:

- En el capitulo siguiente se realizará una descripción de la planta del laboratorio para conocer así los componentes principales y su distribución y conexionado.
- La descripción del software utilizado y el diseño preliminar de la planta se describirán en el tercer capítulo.
- En el cuarto capitulo se describirán las pruebas preliminares que se realizaron para comprender el funcionamiento de Zibra, así como las modificaciones que fue necesario realizar al modelo preliminar de la planta, para después proceder con la configuración de los volúmenes de trabajo de cada elemento.
- Tras haber programado los diferentes elementos de la planta, se comprueban los resultados

obtenidos en el capítulo quinto.

- Finalmente, en el sexto capítulo se expondrán las conclusiones a las que se ha llegado a la vista de los resultados y el desarrollo del trabajo, a la vez que se comentarán las posibilidades que este proyecto supone para trabajos futuros.

Adicionalmente al final del trabajo se incluirán los scripts usados durante la programación en cuatro anexos.

2 DESCRIPCIÓN DE LA PLANTA

A lo largo de este capítulo se describirán los componentes de la planta, así como su funcionamiento.

2.1. Utilidad y componentes

La planta piloto de la cual se va a realizar el gemelo digital se encuentra en el Laboratorio de robótica de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla. Dicha planta puede apreciarse en la Figura 2-1.

Figura 2- 1: Planta piloto

Esta planta fue diseñada para el estudio de reacciones químicas exotérmicas. Este tipo de reacciones son aquellas en las que se desprende energía en forma de calor, siendo la reacción más destacada la de combustión [19].

Para la simulación de este tipo de reacciones, la planta consta de un depósito en el que se mezcla agua caliente proveniente de un termo eléctrico cuya temperatura está controlada y agua fría de un tanque de abastecimiento. Esta mezcla se enfría mediante una camisa de refrigeración que transporta agua

enfriada por un intercambiador de calor. El esquema simplificado de esta estructura aparece en la Figura 2-2.

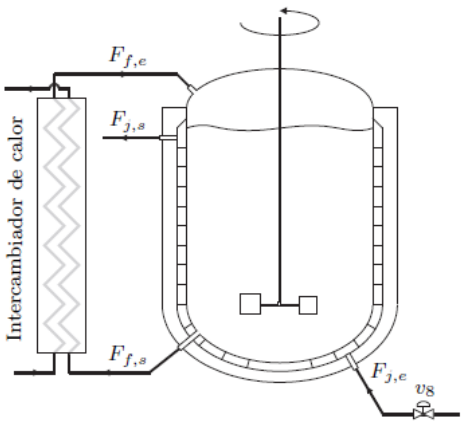


Figura 2- 2: Esquema del depósito, intercambiador de calor y camisa de refrigeración [20]

Para la correcta simulación y el posterior diseño del gemelo digital, se describen a continuación los componentes más importantes de la instalación [21] y en la Figura 2-3 aparecen señalados algunos de estos componentes:

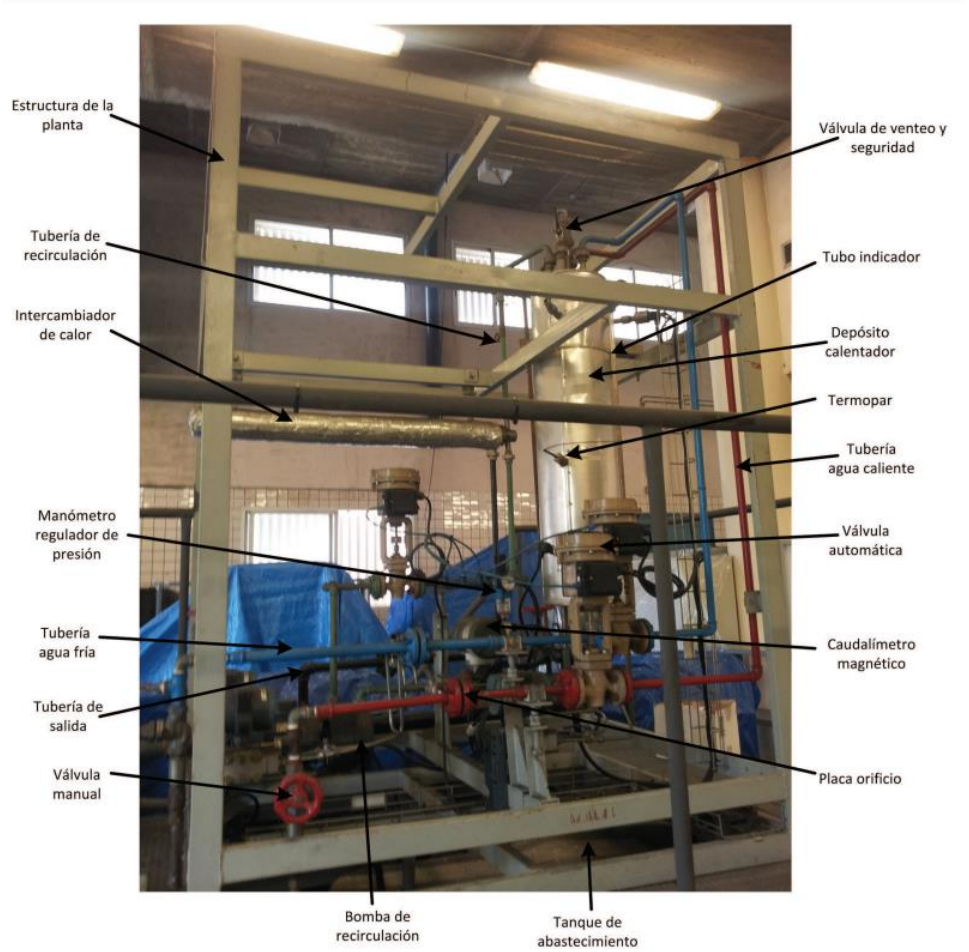


Figura 2- 3: Planta piloto con algunos de sus componentes señalizados [22]

- Depósito: es el elemento principal donde se produce la simulación de la reacción. Para ello consta de una camisa de refrigeración donde se recibe agua fría del intercambiador de calor y que servirá para enfriar el interior del depósito, además de una resistencia interna (que deberá estar cubierta siempre por agua) que permitirá a su vez aumentar la temperatura.

  El agua caliente del interior del depósito proviene de un termo eléctrico que se describirá más adelante, y el agua fría procede del tanque de abastecimiento.

  La salida de la camisa de refrigeración permite recircular el agua hacia el intercambiador de calor y la tubería de salida del depósito devuelve el agua enfriada de su interior al sistema. Como salida adicional el depósito cuenta con una tubería de vaciado controlada por una válvula, que en caso de ser necesario puede abrirse para vaciar el interior en el tanque de abastecimiento.

  El depósito mide 1m de altura y tiene 20cm de diámetro, poseyendo una capacidad de 31 litros. Además, consta de diversos sensores para el control de la temperatura, nivel del agua y presión del interior del depósito, así como un rebosadero que impide que el depósito acumule una determinada cantidad de agua, permitiendo recircularla al tanque de abastecimiento.

  El depósito puede apreciarse en la siguiente Figura 2-4:

Figura 2- 4: Depósito de la instalación

- Intercambiador: en él se produce el enfriamiento del agua proveniente de la camisa de refrigeración utilizando para ello el agua del tanque de abastecimiento, para ello utiliza el sistema de flujos cruzados. El agua de la camisa llega al intercambiador gracias a una bomba de recirculación.
- Tuberías: se distinguen cuatro tipos de tuberías según su color (en las Figuras 2-5, 2-6 y 2-7 se pueden observar en más detalle las tuberías y el intercambiador de calor):

- Azul: por ella circula el agua fría que se introduce en el depósito y en el intercambiador actuando como refrigerante.
- Roja: contiene el agua caliente que sale del termo eléctrico y que se introduce a su vez en el depósito.
- Verde: es la tubería de recirculación que se encarga de alimentar la camisa de refrigeración y de extraer su agua para reconducirla a través del intercambiador de calor.
- Negra: conecta la salida del depósito y del intercambiador al tanque de abastecimiento, cerrando así el ciclo.

Figura 2- 5: Detalle del intercambiador y de las tuberías de agua fría y caliente

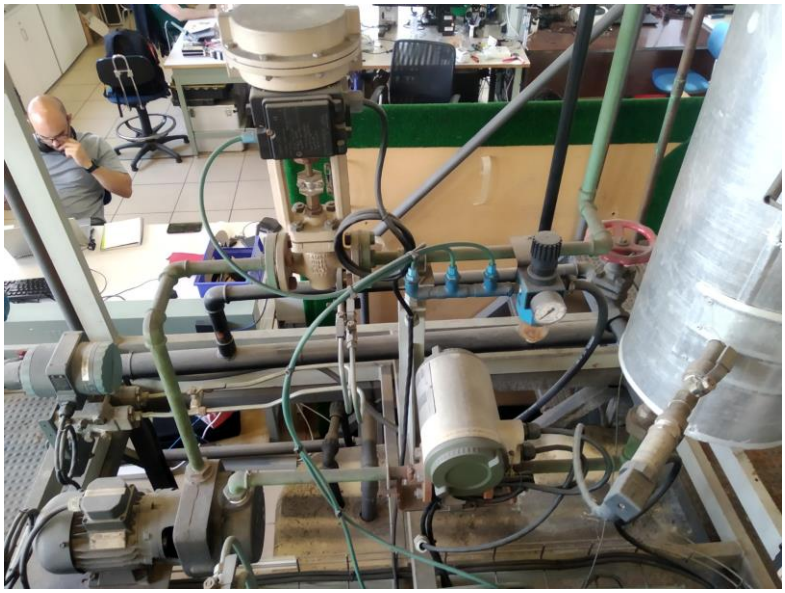


Figura 2- 6: Detalle de la tubería de realimentación

Figura 2- 7: Detalle de la tubería de salida

- Termo eléctrico: permite el calentamiento del agua proveniente del tanque de abastecimiento la cual se introducirá en el depósito. Se ubica en la parte inferior de la planta, tiene una capacidad de 300L y posee una potencia nominal de 12000W. El valor máximo de temperatura que se ha fijado es de 80ºC.
- Grupo de presión: está conformado por una bomba de alimentación, un calderín y un presostato. Este conjunto de elementos es el encargado de introducir el agua del tanque en el sistema a través de las tuberías de agua fría y caliente. La bomba de alimentación succiona el agua del depósito y la impulsa al sistema, donde el calderín la mantiene a una presión de 4bar.
- Tanque de abastecimiento: se sitúa debajo de la planta y tiene una capacidad de $1m^3$. Actúa como suministro de agua del sistema o como sumidero de las diferentes salidas del sistema. El tanque es alimentado a su vez por la red de abastecimiento del laboratorio.

  La temperatura del depósito está controlada por un dispositivo externo que mantiene la temperatura a 17ºC.

  En la Figura 2-8 se muestran los componentes situados en la parte inferior de la planta (el termo eléctrico, el grupo de presión y el tanque de abastecimiento).

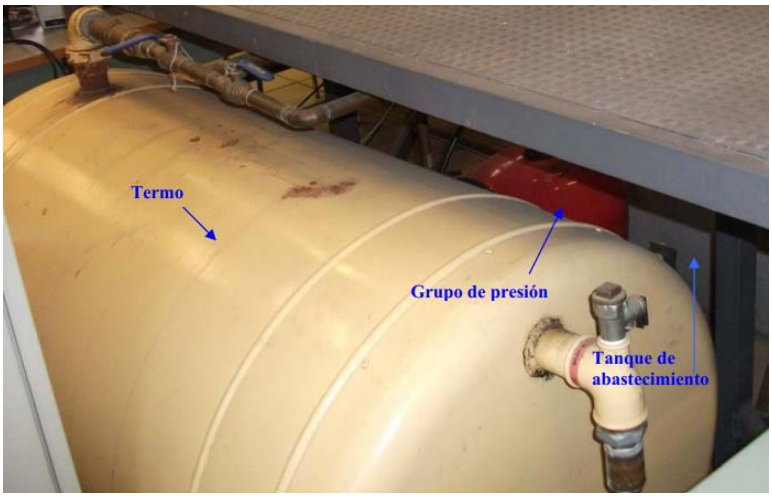


Figura 2- 8: Detalle del termo eléctrico, tanque de abastecimiento y grupo de presión

- Otros elementos: la planta consta de diversas válvulas y sensores (principalmente caudalímetros y transductores) para el correcto funcionamiento de esta. Las válvulas se utilizan para el control del caudal y de la presión dentro del sistema, pudiendo ser de accionamiento manual o automáticas. En la Tabla 2-1 se puede ver un listado de las diferentes válvulas presentes en la planta clasificándolas por tipo y explicando su función, y en la Figura 2-9 un modelo de la planta donde se pueden ubicar estas válvulas.

Tabla 2- 1: Listado de válvulas manuales y automáticas presentes en la planta

Válvula	**Tipo**	**Función**
V1	Manual	Permite el flujo de agua fría hacia el depósito y el intercambiador
V2	Manual	Permite el flujo de agua caliente desde el termo al depósito
V3	Manual	Permite el flujo de agua fría hacia el intercambiador
V4	Automática	Control del flujo en la tubería de agua caliente
V5	Automática	Control del flujo en la tubería de agua fría
V6	Manual	Abre o cierra el circuito de salida del depósito
V7	Manual	Abre o cierra el rebosadero
V8	Automática	Control del flujo en la tubería de recirculación
V9	Manual	Permite el paso del agua de recirculación al deposito

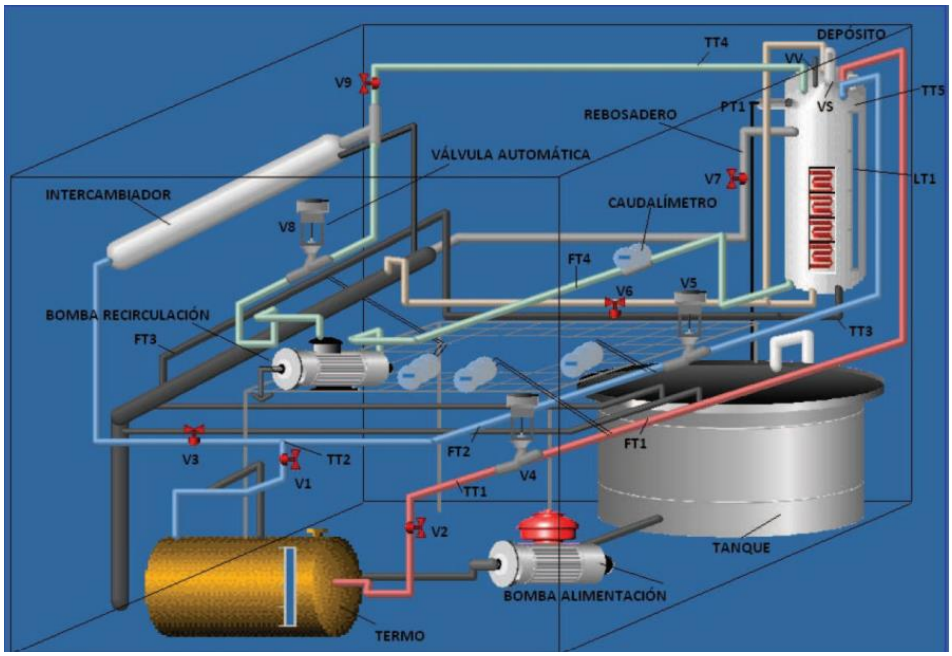


Figura 2- 9: Modelo de la planta donde puede apreciarse la posición de las válvulas [22]

3 HERRAMIENTAS Y DISEÑO DEL GEMELO DIGITAL

En este capítulo se describirán las diferentes herramientas y elementos que se han utilizado para el montaje y desarrollo del gemelo digital de la planta piloto.

3.1. Unity

Unity 3D es un motor de desarrollo multiplataforma de videojuegos muy utilizado en la industria, que fue creado por Unity Technologies en 2004. La popularidad de Unity no solo ha hecho que su uso se limite a la industria de los videojuegos, sino que poco a poco se ha ido extendiendo a otras industrias y aplicaciones [23].

La razón del éxito de Unity es que ofrece una interfaz de fácil comprensión como se muestra en la Figura 3-1, que permite el diseño de diferentes aplicaciones tanto en 3D como en 2D, destacando además el uso de herramientas propias que permiten la edición de gráficos, sonidos y animaciones, siendo C# su principal lenguaje de programación. Esa fácil comprensión del programa y su accesibilidad ha hecho que crezca a su alrededor una gran comunidad de usuarios, los cuales contribuyen también a esta expansión creando nuevos objetos y algoritmos con otros programas compatibles, consiguiendo así suplir las carencias de Unity.

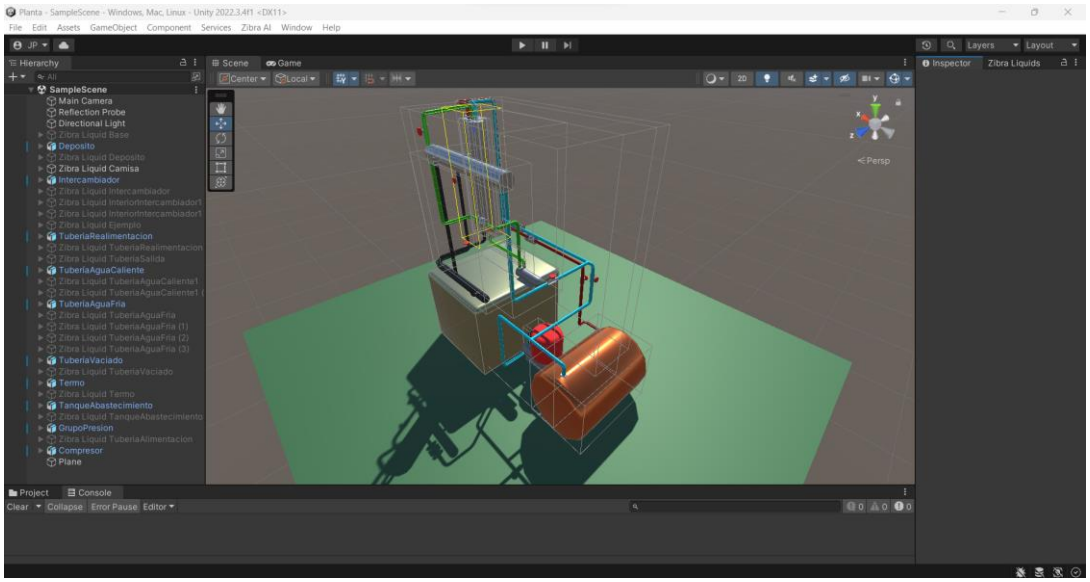


Figura 3- 1: Interfaz de Unity 3D

Como se ha mencionado anteriormente, la interfaz de Unity permite tener un mayor control sobre las aplicaciones que se están desarrollando, y consta de cinco pestañas principales [24]:

- Escena (Figura 3-2): es la pestaña principal de la interfaz, donde se muestran los objetos disponibles en la aplicación, pudiéndose editar la posición, orientación y tamaño de estos. En esta pantalla pueden además añadirse las cámaras, luces y otros objetos o elementos que

son necesarios para la ejecución de las aplicaciones. En la pestaña *Game* se muestra una vista previa de cómo se verían los objetos en caso de iniciar la aplicación.

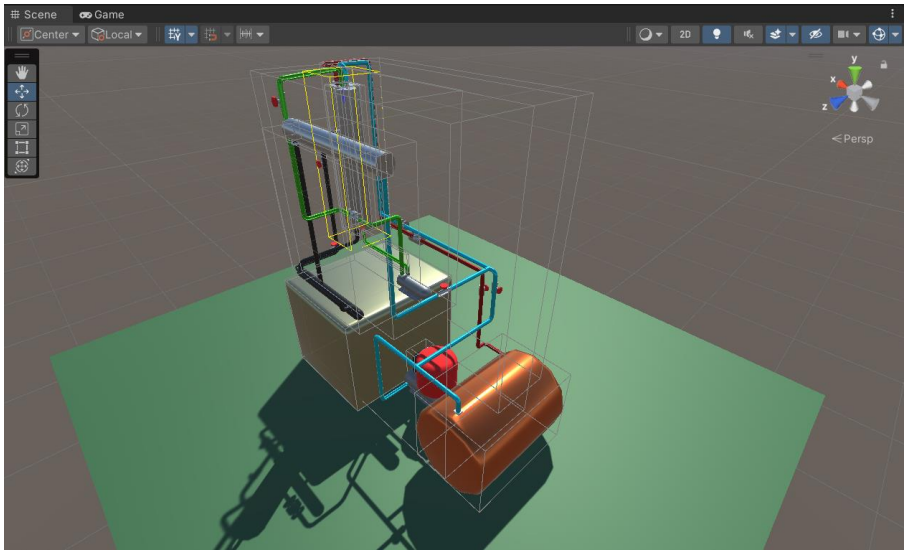


Figura 3- 2: Pestaña escena

- Jerarquía (Figura 3-3): en ella se listan los diferentes objetos que aparecen en la escena. Estos objetos pueden incluir otros objetos a modo de componentes que están ligados al elemento superior en la jerarquía, de forma que, si se cambia la posición o escala de este, los elementos ligados también sufren ese cambio en la misma medida.

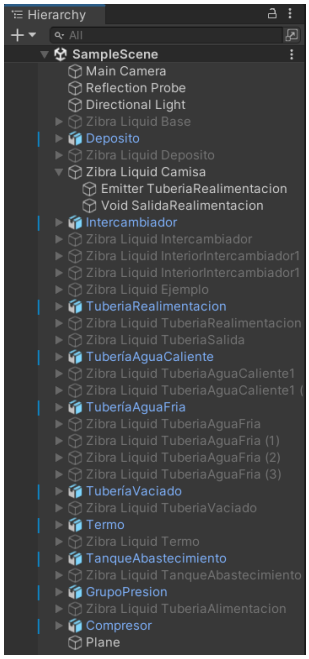


Figura 3- 3: Pestaña Jerarquía

- Inspector (Figura 3-4): en la pestaña inspector aparecen las características del objeto ubicado en la escena que se seleccione, permitiendo su modificación. Además, en esta pestaña se pueden añadir nuevos componentes a los objetos como los *colliders*

(componente que define la forma de un objeto para futuras interacciones físicas en el programa), *scripts* (sistema de programación mediante código o gráficos) y *RigidBodies* (definen al objeto como un sólido rígido al que afecta el motor físico de Unity)

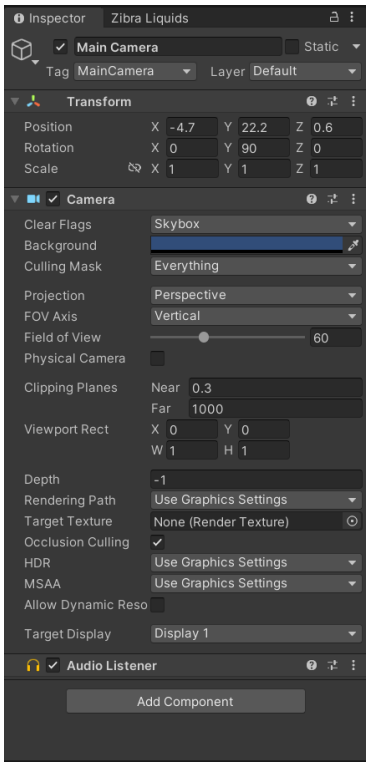


Figura 3- 4: Pestaña inspector

- Proyecto (Figura 3-5): todos los archivos o *assets* con los que se trabaja en el proyecto aparecen listados en la pestaña Proyecto. En esta pestaña se pueden crear carpetas que permiten compartimentar y localizar los *assets* que se quieran usar.

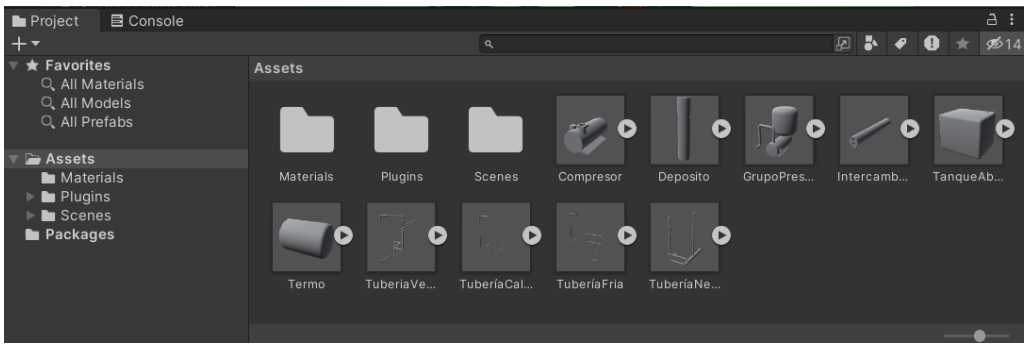


Figura 3- 5: Pestaña Proyecto

- Consola (Figura 3-6): en esta pestaña aparecen los posibles errores que surgen durante la ejecución de la aplicación, permitiendo su corrección.

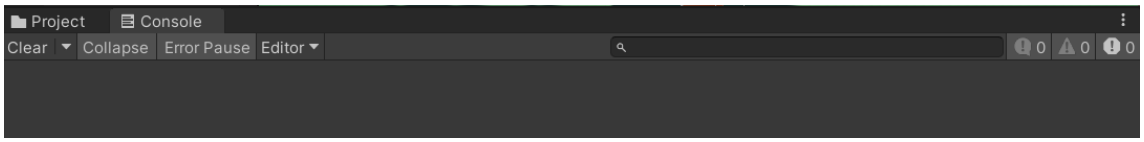


Figura 3- 6: Pestaña Consola

Tras comprobar la versatilidad de este programa a la hora de realizar animaciones y su potente motor físico, se decidió utilizarlo para el desarrollo del gemelo digital, como se ha utilizado en otros proyectos similares [18]. Otra de sus grandes ventajas es que al tener una comunidad de usuarios bastante grande es posible encontrar *assets* que permitan ajustar el gemelo de forma que sea más realista, por lo que el siguiente paso fue localizar una herramienta que permita la simulación de fluidos.

3.2. Zibra Liquids

A la hora de seleccionar un *asset* que permitiera emular un fluido de una forma que fuera lo más parecida a la realidad, se decidió utilizar Zibra Liquids, creado por Zibra AI en 2010 [25]. Este *asset* consigue mediante el uso de tecnología basada en inteligencia artificial, emular el comportamiento de un fluido (ya sea líquido o gas) así como editar las diferentes características del líquido como la viscosidad o su color. Otra funcionalidad que destaca en esta herramienta es la posibilidad de definir objetos mediante una red neural para indicar que se consideran como obstáculos para el movimiento del fluido.

Zibra Liquids dispone de múltiples elementos para la simulación de un fluido. Los que se han utilizado en este trabajo son los que se describen a continuación [26]:

- *Liquid Simulation Volumes*. Para empezar a simular el fluido es necesaria primero definir el volumen en el cuál este va a moverse y para ello se genera un volumen de simulación (*Liquid Simulation Volume*) como el que aparece en la Figura 3-7. La dimensión de este volumen de trabajo es de especial importancia, ya que cuanto mayor es el volumen, mayor es la resolución necesaria para la simulación, lo cual repercute en la necesidad de recursos del ordenador para poder ejecutarla.

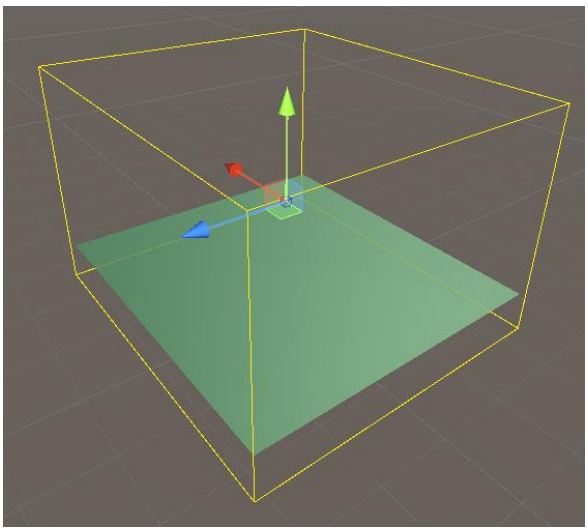


Figura 3- 7: Liquid Simulation Volume en Unity

- *Colliders*. Los *colliders* son unas estructuras en forma de red que en Unity suelen estar ubicados sobre las superficies de los objetos y que permiten la interacción entre ellos. En Zibra se definen dos tipos de *colliders* para su interacción con los fluidos: *Analytic Colliders* (*colliders* con una forma específica, por ejemplo, una esfera o un cubo) y *Neural Colliders* (utilizados para objetos con una geometría compleja y generados mediante inteligencia artificial).

  Para este trabajo se utilizarán los *Neural Colliders*, ya que la estructura de la planta presenta formas complejas, por lo que su aproximación a la realidad es más exacta.

  A la hora de definir un *collider* se debe indicar el SDF (*Signed Distance Field* [27]) y si este es normal o invertido, Esta definición resulta interesante porque indica si el fluido se quedara por fuera del objeto o si por el contrario estará dentro de él.

- *Emitter*: los emisores o *emitters* son los encargados de generar las partículas del fluido. Los parámetros que se pueden modificar son la forma y tamaño del emisor, así como la cantidad de partículas generadas cada segundo de la simulación y su dirección y velocidad inicial. En la Figura 3-8 se observa su símbolo y cómo genera agua.

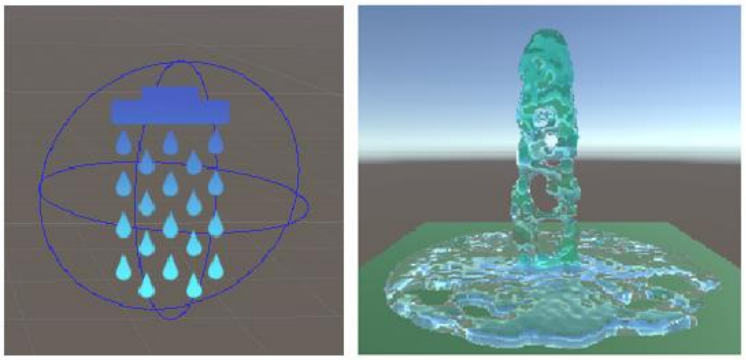


Figura 3- 8: Símbolo de los emitters (izquierda) y simulación (derecha)

- *Void*. Al contrario que los *emitters*, los *voids* eliminan las partículas del sistema, permitiendo que los *emitters* puedan generar partículas de forma continua. En la Figura 3-9 se muestra su símbolo y una imagen durante la simulación en la que se ha colocado un *Void* rectangular en un lateral, observándose que por debajo de este no cae el agua.

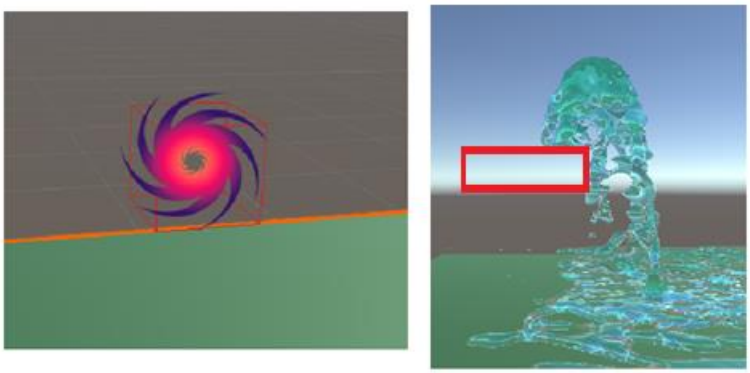


Figura 3- 9: Símbolo de los Voids (izquierda) y simulación (derecha) donde se indica con un rectángulo rojo donde se ha ubicado el void.

- *Detectors*. los detectores se utilizan a modo de sensor, permitiendo saber la cantidad de partículas que circulan a través de ellos. El símbolo de los detectores y la información que obtienen durante la simulación pueden observarse en la Figura 3-10.

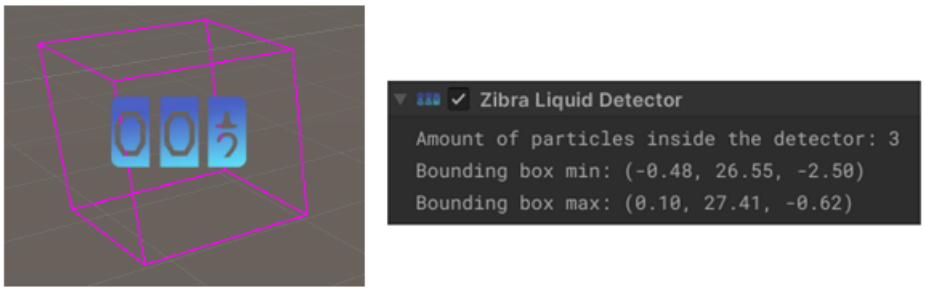


Figura 3- 10: Símbolo de los Detectors (izquierda) y datos obtenidos durante la simulación (derecha)

- *Force Fields*. Los *force fields* son elementos que generan fuerzas que solo actúan sobre los fluidos. Pueden ser de tres tipos: direccionales, que actúan en una dirección definida; radiales, que generan una fuerza que atrae o repele al fluido; y remolino, que provoca un movimiento giratorio simulando el de un torbellino. Los tipos de campos de fuerza y su efecto en la simulación se pueden apreciar en las Figuras 3-11 y 3-12. Los campos de fuerza que se van a utilizar son del tipo radial tanto de atracción como de repulsión, ya que simularán el uso de las bombas.

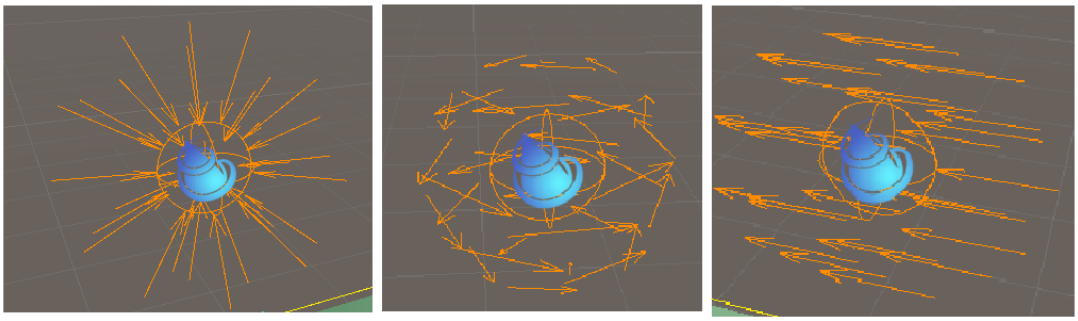


Figura 3- 11: Símbolos de los Force Fields. De izquierda a derecha: Radial, Swirl y Directional

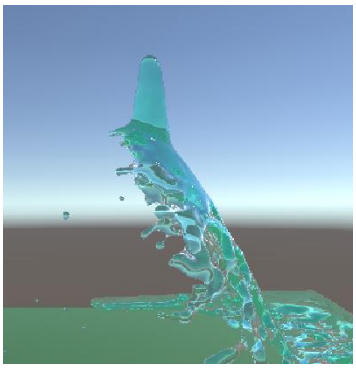


Figura 3- 12: Efecto de un campo de fuerza radial de repulsión en la simulación

3.3. Modelo de la planta

Una de las limitaciones de Unity es su poca versatilidad a la hora de diseñar objetos, por lo que es necesario el uso de programas externos. Unity es compatible con otros programas de diseño y modelado 3D como SolidWorks, Cinema 3D, AutoDesk o Blender, siempre y cuando el objeto a trasladar este en el formato fbx.

Del amplio abanico de programas compatibles se decidió usar los programas de AutoDesk, en concreto AutoCAD para el diseño de cada elemento de la planta y Navisworks para convertir los elementos creados en AutoCAD a formato fbx, ya que la Universidad dispone de una licencia de estos programas para los estudiantes.

En primer lugar, por medio de fotos de la planta, es estimaron las dimensiones de los diferentes elementos, partiendo de las medidas conocidas del depósito. Una vez estimadas las dimensiones se procedió al diseño de la planta. Las medidas se tomaron en milímetros, posteriormente al introducir la planta piloto en Unity se comprobará si es necesario cambiar la escala.

Las principales herramientas del programa que se usaron fueron:

- Barrer: esta herramienta permite que una superficie se extruya a lo largo de una línea, esto permite crear sólidos simples como cilindros o tetraedros de un tamaño concreto con bastante facilidad (en la Figura 3-13 se puede apreciar un ejemplo). Además, en el caso de las tuberías, cuando esta presenta una curvatura, la extrusión continua con el radio de esta.

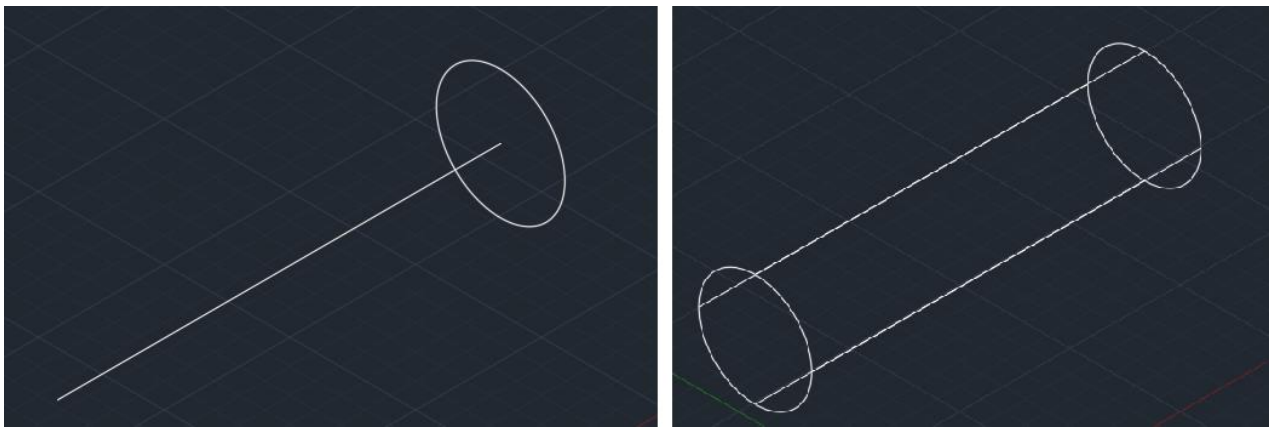


Figura 3- 13: Ejemplo del uso de la herramienta Barrer para crear un cilindro a partir de una línea y un círculo

- Empalme: para unir las tuberías con un radio concreto se utiliza la herramienta "empalme" en la cual se define el radio de unión.
- Vaciado: esta herramienta se utiliza para crear un hueco dentro de un sólido, indicándole la distancia a la superficie exterior. Puede utilizarse de dos formas, que pueden distinguirse en la Figura 3-14, indicando las superficies de los extremos, lo cual crea un hueco que atraviesa el sólido (esto se utilizó para crear las tuberías) o sin indicar las superficies para crear una oquedad proporcional a la superficie exterior (se utilizó para el depósito, intercambiador y tanque).

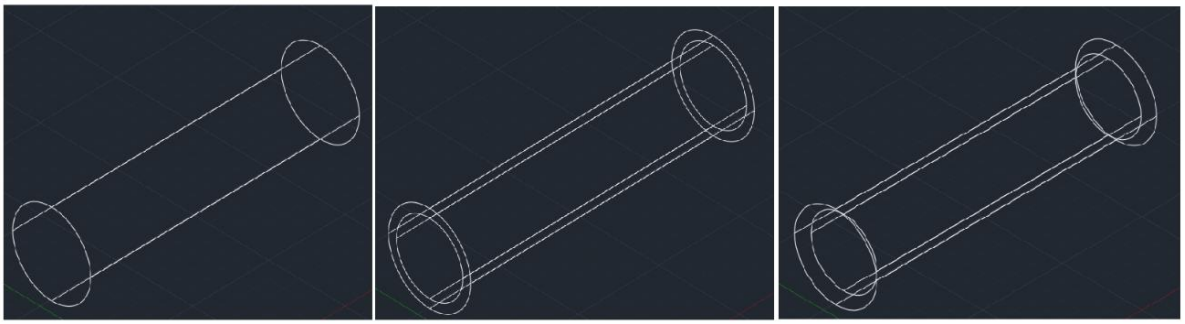


Figura 3- 14: Ejemplo del uso de la herramienta Vaciado. A la izquierda se observa el sólido original, en el centro el sólido con un hueco que lo atraviesa y a la derecha el sólido hueco

- Sólido, unión: la herramienta "Solido, unión" permite construir un sólido a partir de dos o más sólidos. En el ejemplo de la Figura 3-15 se realiza la unión de dos cilindros.

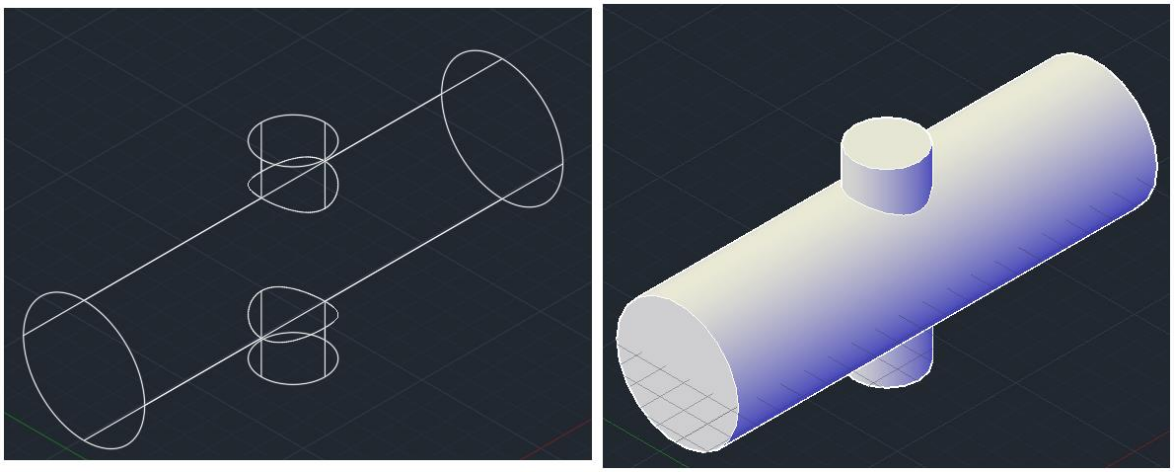


Figura 3- 15: Ejemplo del uso de la herramienta Solido, unión

- Sólido, diferencia: al contrario que "Solido, unión", "Solido, diferencia" resta un sólido a otro, creando un hueco o hendidura. En la Figura 3-16 se muestra un ejemplo.

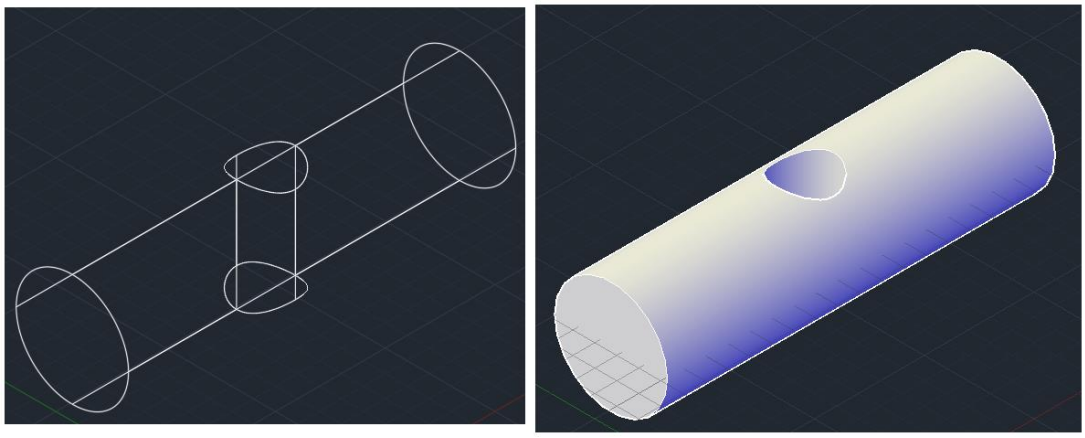


Figura 3- 16: Ejemplo del uso de la herramienta Solido, diferencia

Una vez descritas las herramientas utilizadas, a continuación, se describe como se han realizado los diferentes componentes de la planta:

- Tuberías. En primer lugar, se realizó la estructura alámbrica de las tuberías utilizando las medidas estimadas, tras lo cual, haciendo uso de la herramienta "Barrer" se extruyó una circunferencia de 3cm de diámetro. Una vez se obtuvo la estructura de las tuberías como un conjunto de cilindros sólidos, se usó "Sólido, Unión" para crear un único sólido compuesto de esos cilindros. A este solido final se le aplica un vaciado con una distancia a la superficie 1,5mm (este radio se comprobó en un catálogo de tuberías de acero inoxidable [28]). En las Figuras 3-17, 3-18 y 3-19 se muestra el resultado final de las tuberías principales de la planta.

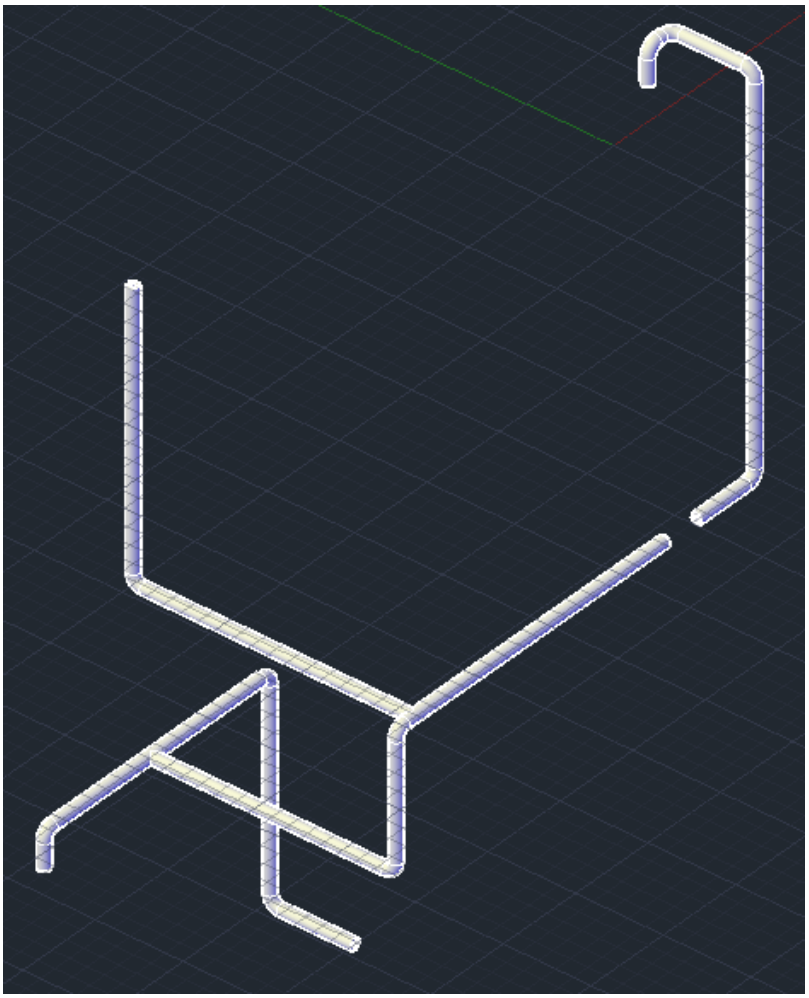


Figura 3- 17: Detalle de la tubería de agua fría

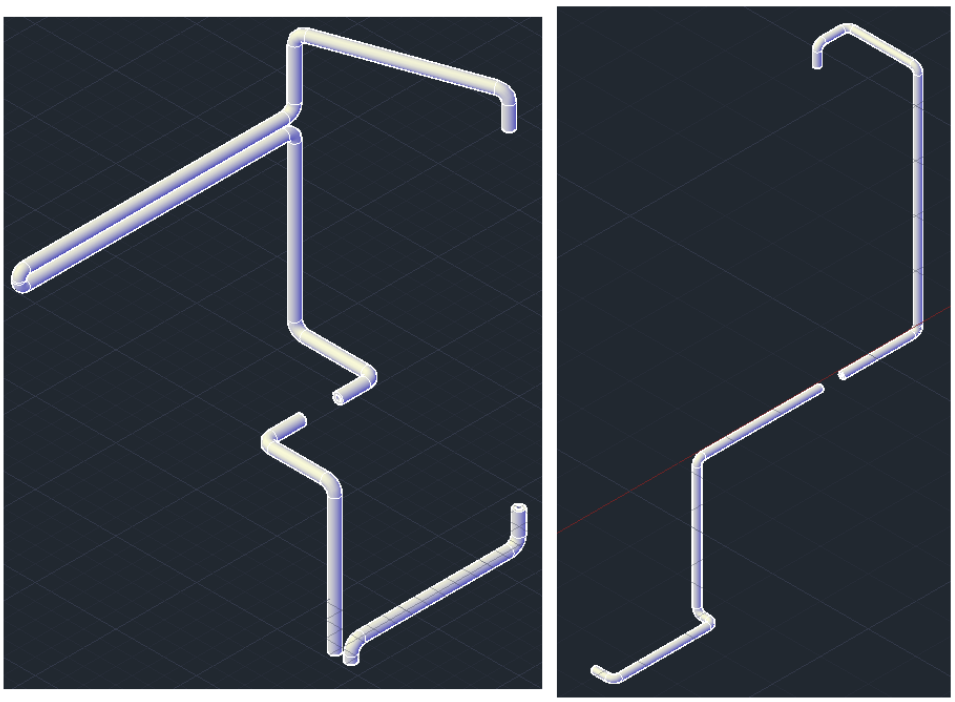


Figura 3- 18: Detalle de la tubería de recirculación (izquierda) y agua caliente (derecha)

Figura 3- 19: Detalle de las tuberías de salida

Para la tubería de salida del intercambiador y del depósito se creó una tubería colectora de mayor grosor (7 cm de diámetro). Para ello, primero se diseñó un cilindro con esas medidas y se les unieron dos esferas a los extremos para darle el aspecto de una capsula. Tras esto,

se crearon unos puntos de unión cilíndricos para poder unir las tuberías del rebosadero y salida del depósito y la entrada al tanque de abastecimiento, siendo el siguiente paso realizar el vaciado con la misma distancia establecida en el resto de las tuberías. En la Figura 3-20 se ve un más en detalle esta sección de la tubería.

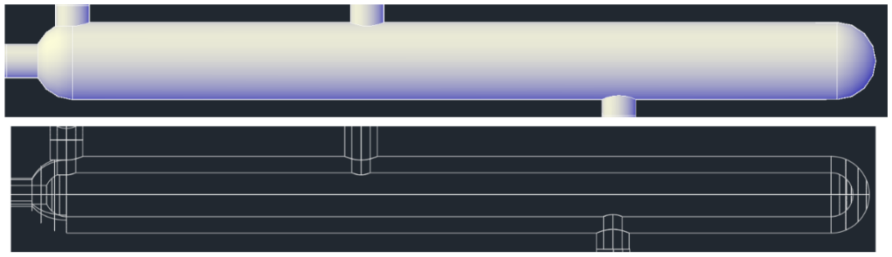


Figura 3- 20: Detalle de la tubería colectora

- Depósito. El depósito está compuesto de dos componentes, una carcasa exterior y un depósito interior, de forma que el hueco entre la carcasa exterior y el depósito interior forma la camisa de refrigeración. El depósito según otros proyectos tiene una forma cilíndrica con 1m de altura y 20 cm de diámetro, aunque no se tienen datos de las medidas del grosor de la carcasa y del depósito interior, por ello se han estimado de forma que a la hora de ejecutar la simulación sea posible apreciar el fluido moviéndose.

  Las medidas de la carcasa exterior son las que se han indicado al principio. Tras crear el cilindro con esas medidas, se utiliza la herramienta "Vaciado" para crear un sólido hueco, al que luego con la herramienta "Solido, diferencia" se le practican tres agujeros de entrada de agua y dos de salida del depósito, los cinco con diámetro de 29 mm, correspondientes al interior de las tuberías. Para la salida de la tubería del rebosadero se practica en el lateral un agujero con un diámetro de 3 cm, por el cual esta pasara. En la Figura 3-21 se muestra la carcasa exterior del depósito.

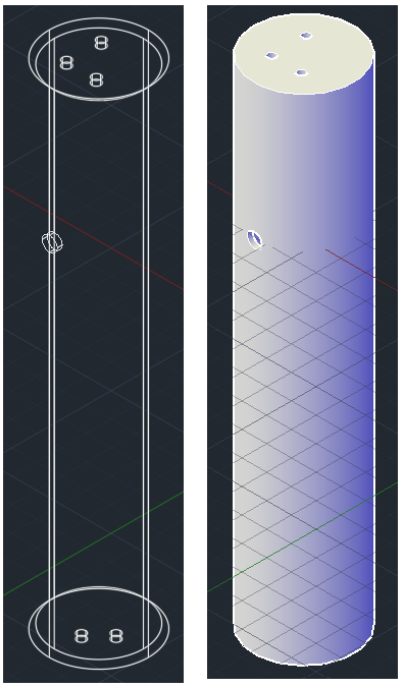


Figura 3- 21: Detalle de la carcasa exterior del depósito

El depósito interior tiene un diámetro de 15 cm y una altura de 94 cm, el grosor de las paredes es igual que el de la carcasa exterior y se consigue mediante un vaciado. Para la entrada de agua caliente y fría, así como para la salida del depósito se crea unos cilindros huecos de entrada para aislarlas de la camisa de refrigeración. Esto mismo, se realiza para la tubería de rebosadero, aunque esta atraviesa la carcasa. El sólido final se observa en la Figura 3-22

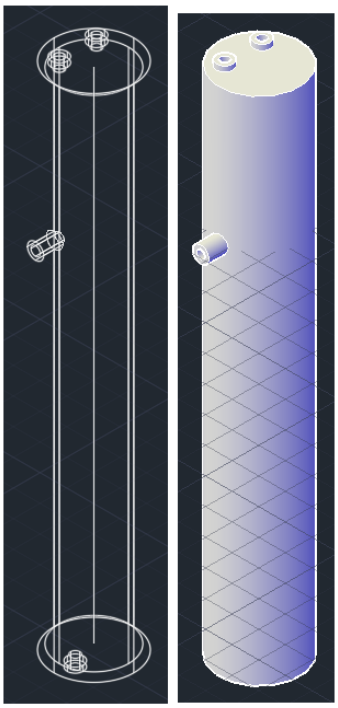


Figura 3- 22: Detalle del depósito interior

- Intercambiador. El intercambiador de calor parte de un cilindro de 85 cm de largo y un diámetro de 11 cm. El interior se realiza mediante un vaciado y se practican dos agujeros en el extremo por donde se introducen la tubería de realimentación. Para la unión con las tuberías de entrada y salida del líquido refrigerante se unen sendos cilindros huecos para unirlos a las tuberías anteriormente mencionadas. El intercambiador ya diseñado se muestra en la Figura 3-23.

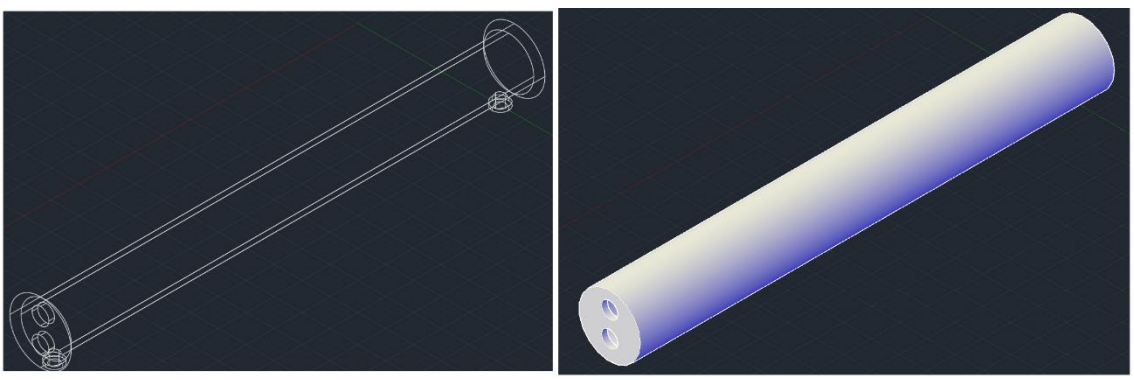


Figura 3- 23: Detalle del intercambiador de calor

- Tanque de abastecimiento. Para el tanque de abastecimiento se tomó un sólido cúbico de 100x85x80 metros al cual se le redondearon las aristas por medio de la herramienta "Empalme". Se le practico un agujero donde el agua se reintroduce en el tanque tras haber pasado por todo el sistema. Una vez hecho esto, se usó la herramienta "Vaciado" para dejar hueco el tanque. El tanque final se observa en la Figura 3-24.

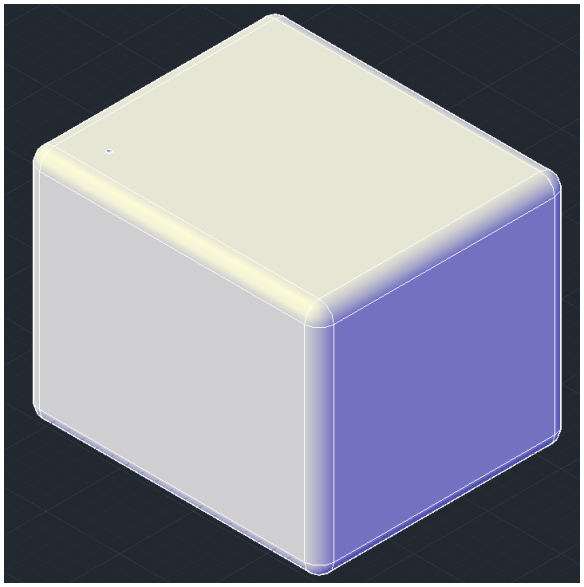


Figura 3- 24: Detalle del tanque de abastecimiento

- Termo eléctrico. El procedimiento para realizar el termo eléctrico es el mismo que para el tanque de abastecimiento, aunque en este caso se parte de un cilindro de 60 cm de radio y 85 cm de largo. En la Figura 3-25 se aprecia el termo eléctrico final.

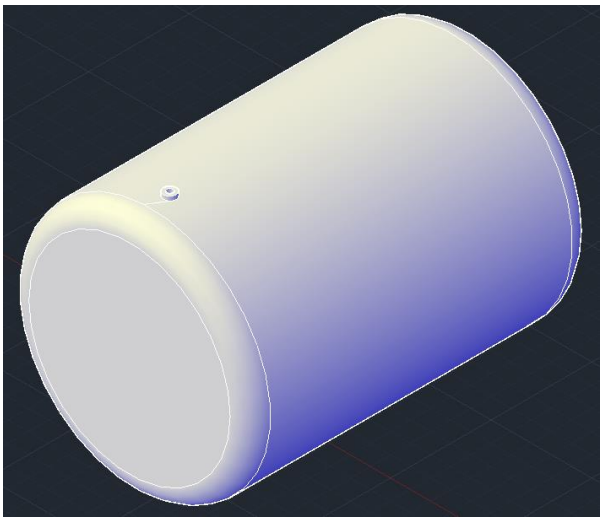


Figura 3- 25: Detalle del termo eléctrico

- Grupo de presión y compresor. Estos elementos de la planta siguen una estructura similar. salvo la tubería de alimentación de la planta, todos los elementos van a ser sólidos sin

huecos, ya que se considera que simulando el fluido en las tuberías no es necesario diseñar el interior de los compresores. Los compresores están compuestos por una pieza cilíndrica central en cuyo extremo se le ha unido una esfera del mismo radio y una pieza triangular que actúa a modo de base. En el caso del grupo de presión, este consta de un tanque hidroneumático que se ha diseñado de la misma forma que el termo eléctrico, pero a menor escala y sin el vaciado. Finalmente se diseña una tubería de alimentación que lleva el agua del interior del tanque de abastecimiento al compresor. El grupo de presión y el compresor se observan en la Figura 3-26.

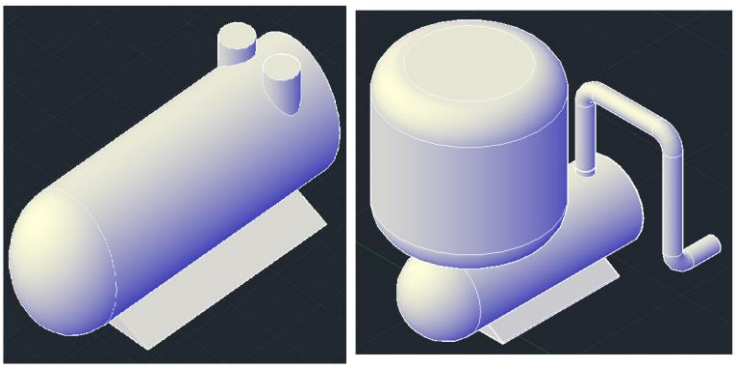


Figura 3- 26: Detalle del compresor (izquierda) y grupo de presión (derecha)

- Válvulas automáticas. Para este tipo de válvulas se creó una estructura de varios cilindros para que se parecieran lo máximo posible a la realidad. Esta estructura es hueca para que el agua circule por dentro ella, estando pensada para que actúe a modo de unión entre las tuberías y para que posteriormente, durante la programación del gemelo digital, pueda controlarse el flujo de agua que circula por ellas.
- válvulas manuales, se consideró que estas fueran un elemento estético para indicar los puntos en los cuales se corta o abre el flujo de agua, al ser consideradas como válvulas tod o nada. Esta acción de apertura o cierre se llevará a cabo únicamente por medio de la simulación. Este elemento está compuesto por dos piezas: una manivela y un cuerpo cilíndrico hueco, por cuyo interior pasa la tubería en la que se ubica.

  En la Figura 3-27 pueden apreciarse los dos tipos de válvulas en detalle.

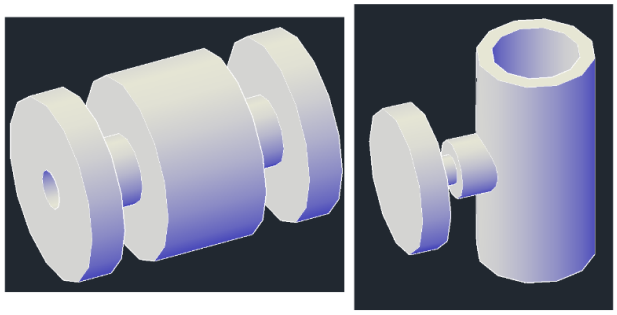


Figura 3- 27: Detalle de las válvulas automáticas (izquierda) y manuales (derecha)

Una vez se han creado los diferentes elementos de la planta, se unen todos en un único archivo formando así la planta piloto en su totalidad como se muestra en la Figura 3-28.

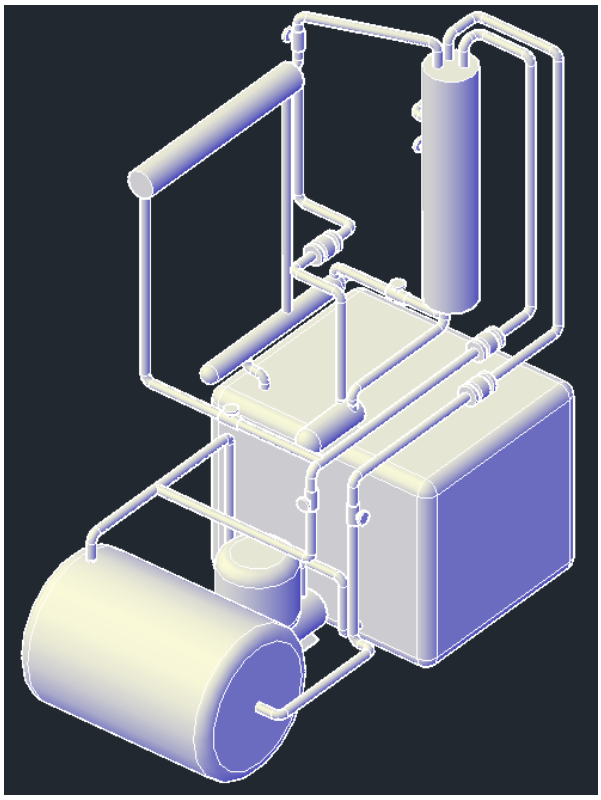


Figura 3- 28: Detalle de la planta piloto creada con AutoCAD

Una vez montada la planta, esta se guarda en un archivo en formato dwg ya que AutoCad no puede generar archivos fbx, para eso es necesario otro programa de AutoDesk llamado Naviswork.

Al abrir el archivo en Naviswork es posible convertirlo en formato fbx (en la Figura 3-29 puede apreciarse la pestaña para convertir el archivo). Para ello, es necesario definir a que unidades se va a convertir la escala, esto debe hacerse porque las unidades no se conservan al pasar de AutoCAD a Naviswork, solo el valor numérico. En este caso, aunque se tomaron las medidas en milímetros, al diseñar la planta se hizo en metros, por lo que se seleccionará esta opción en Naviswork.

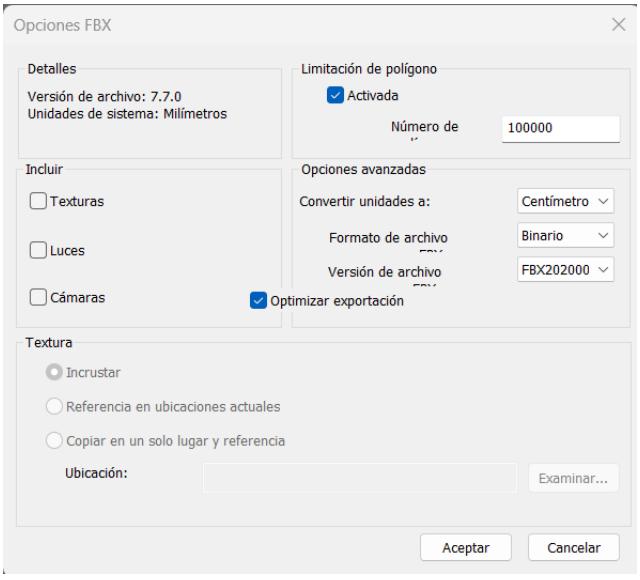


Figura 3- 29: Pestaña para exportar archivos dwg a formato fbx

La planta final en Unity queda como se muestra en la Figura 3-30.

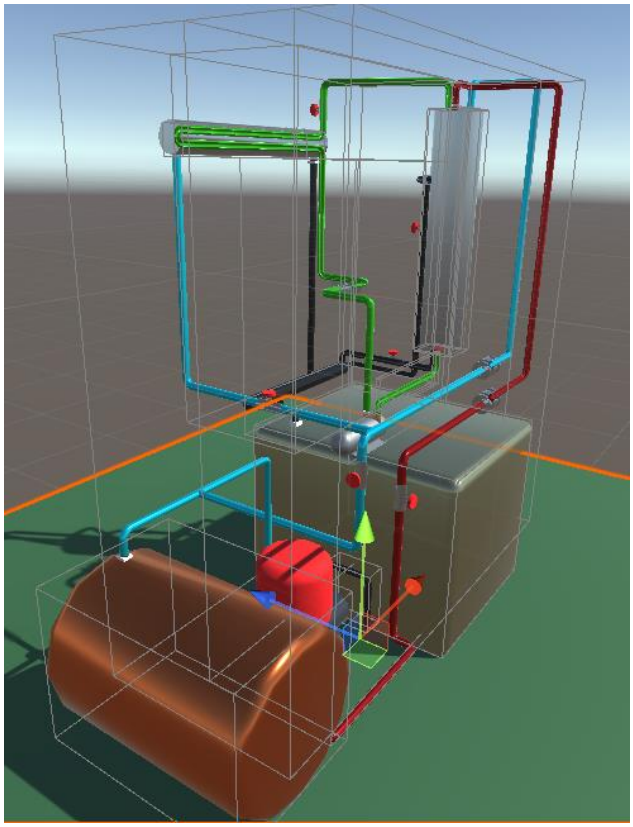


Figura 3- 30: Planta piloto en Unity

4 DESARROLLO DEL GEMELO DIGITAL

A lo largo de este capítulo se va a explicar en detalle los pasos que se siguieron para la programación del gemelo digital.

4.1. Pruebas y ajustes preliminares

Antes de importar la planta piloto en su totalidad a Unity 3D, se decidió trasladar primero una de las tuberías para comprobar el funcionamiento de Zibra y si había que realizar cambios al modelo de la planta. La tubería que se decidió importar es la de salida del tanque de abastecimiento por donde circula el agua hacia el grupo de presión (esta tubería se muestra en la Figura 4-1). El motivo por el cual se eligió esta tubería es porque es de corta longitud y con tres giros en su trayectoria, esto permitiría comprobar si el agua generada por los emisores tiene la suficiente presión para ascender por la tubería y realizar todo el recorrido.

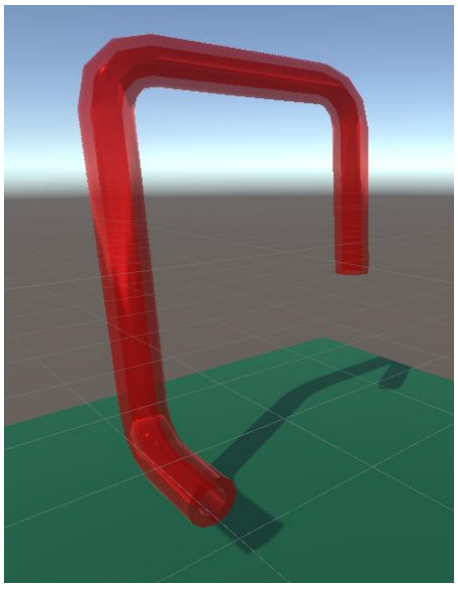


Figura 4- 1: Tubería del grupo de presión

Al realizar la importación a Unity se encontraron una serie de dificultades a las que se tuvo que poner solución. Dichas dificultades fueron:

- Escalado de la planta y volumen de trabajo de Zibra Liquids. Al iniciar la simulación de un fluido con Zibra Liquids, es necesario definir un volumen en el cual se generará el líquido y que a su vez actuará como espacio de trabajo. El problema que se observó es que cuanto mayor es el volumen de trabajo, mayor es la resolución necesaria para simular el fluido provocando que la ejecución del programa exige más recursos del ordenador. Esto hace que la simulación se ralentice, no se genere bien fluido o que este tenga comportamientos extraños.

  Tomando como base lo anteriormente explicado, se realizaron experimentos para

comprobar cuál era el máximo volumen que el ordenador permitía ejecutar para más tarde escalar la planta entera para que se ajuste a ese volumen. El escalado de la planta puede realizarse de dos formas: cambiando el tamaño de sus elementos en AutoCAD o al convertirlos en formato sbx en Naviswork indicando las unidades en las que deben estar. Tras comprobar diferentes escalas se comprobó que la mejor opción era multiplicar la escala en Autocad por 1000 y en Naviswork establecer las unidades en centímetros, consiguiendo así un tamaño de planta que se ajusta al máximo volumen que el ordenador es capaz de simular.

- *Neural colliders*. Para que el agua se desplazara por dentro de la tubería se añadió a esta un *collider* específico de Zibra. La característica principal de estos *Neural Colliders* es que se adaptan a la forma del objeto siguiendo una red neural. El problema surgió al iniciar la simulación, el flujo del agua se bloqueaba al inicio de la tubería, esto se debía a que la red generada por el *collider* se enlaza por dentro del hueco de la tubería creando una pared que bloquea el agua como se puede apreciar en la Figura 4-2.

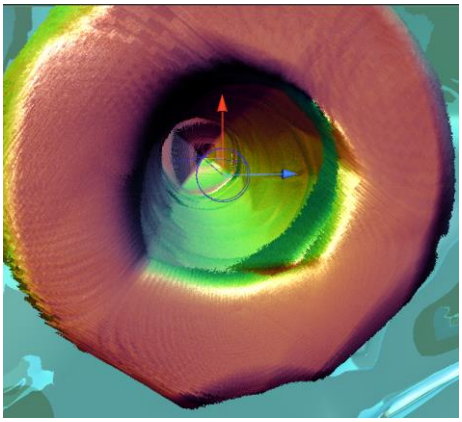


Figura 4- 2: Detalle del collider de la tubería donde se muestra el bloqueo

Como solución al bloqueo del agua, se utilizó una de las características de los *colliders* de Zibra y es que puedes definir si su red SDF es normal o invertida, la diferencia entre estas dos opciones es que las redes normales están pensadas para que el fluido se encuentre fuera del objeto, y en las invertidas se ubique por dentro de los objetos. Sabiendo esto, se decidió crear sólidos que actuaran a modo de relleno de los diferentes elementos de la planta, definiendo sus *colliders* como redes invertidas, de esta forma al ser estructuras más simples de analizar por el programa, el agua no encuentra ningún tipo de bloqueo en su camino. Estos nuevos sólidos posteriormente se harán invisibles durante la ejecución del programa, pareciendo así que el agua se mueve por dentro de las tuberías. En la Figura 4-3 se muestra la tubería con agua generada en su interior.

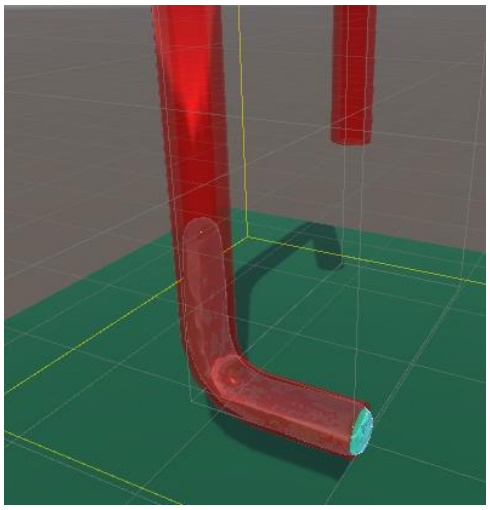


Figura 4- 3: Detalle de la tubería con agua en su interior

- Presión en las tuberías. Una vez configurados los *colliders* y el volumen de trabajo, se procedió a configurar la generación del fluido. En este proceso se observó los siguiente:
    - Los *emitters*, elementos necesarios para la generación del fluido no tiene la suficiente velocidad para hacer que este ascienda por una tubería. Esto provoca que el agua se, acumule en la tubería. Además, el número de partículas que generan es limitado, por lo que resulta necesario encontrar la forma de que los *emitters* funcionen de forma constante.
    - El comportamiento del fluido es similar a un líquido, pero no se puede considerar como tal. El "fluido" realmente son una serie de partículas minúsculas que en conjunto emulan el movimiento de un líquido, pero presentan comportamientos que no son propios de estos, por ejemplo, al acumularse en las tuberías verticales, lo lógico sería que las partículas que se generan empujen a las ya creadas, pero esta acción no se observó, simplemente seguían acumulándose en el mismo lugar sin haber desplazamientos.
    - Debido a que las paredes de las tuberías son demasiado finas, algunas partículas sobresalen, haciendo que la propia tubería deje de apreciarse. Para evitar esto se aumentó el grosor de las tuberías.

  Para poner solución a estos problemas se usan otros elementos de Zibra: los *Force Fields* para atraer o repeler las partículas de fluido, simulando así la presión dentro de las tuberías; y los *Voids*, que permiten eliminar las partículas que llegan al final de la tubería evitando así acumulaciones y que el sistema siga generando partículas de forma constante.

Tras los resultados obtenidos en las pruebas con una tubería, se realizaron las comprobaciones de los cambios sugeridos anteriormente y al observar su viabilidad, se procedió a realizar dichos cambios en el modelo de la planta.

4.2. Rediseño y montaje de la planta en Unity

La mayoría de los cambios que se consideraron necesarios deben aplicarse en la etapa de diseño de AutoCAD. Los cambios más sencillos fueron el aumento del grosor de las tuberías que se aumentó el radio exterior de las mismas en 5mm y el cambio de escala de la planta, que se realizó con la herramienta Escala, la cual permite aumentar o disminuir el tamaño de un objeto.

El siguiente paso tras estas pequeñas modificaciones, es el diseño del relleno interior de los diferentes elementos de la planta. Adicionalmente en este momento, se planteó en fragmentar la planta en sus diferentes elementos e importarlos uno a uno a Unity en vez de trasladar la planta entera. Esto facilitará más tarde la programación y ayudará en proyectos futuros a modificar la planta añadiendo nuevos elementos a la misma o cambiando los ya existentes.

El diseño de los rellenos de cada elemento de la planta se describe a continuación:

- Depósito. Al diseño original del depósito se le han añadido parte del trayecto final de las tuberías de entrada y salida de este, esto facilitará como se explicará en el siguiente apartado la simulación. Además, se van a diseñar dos rellenos: uno para la camisa de refrigeración y otro para el depósito interior. En la Figura 4-4 se muestran los dos elementos del depósito con sus rellenos.

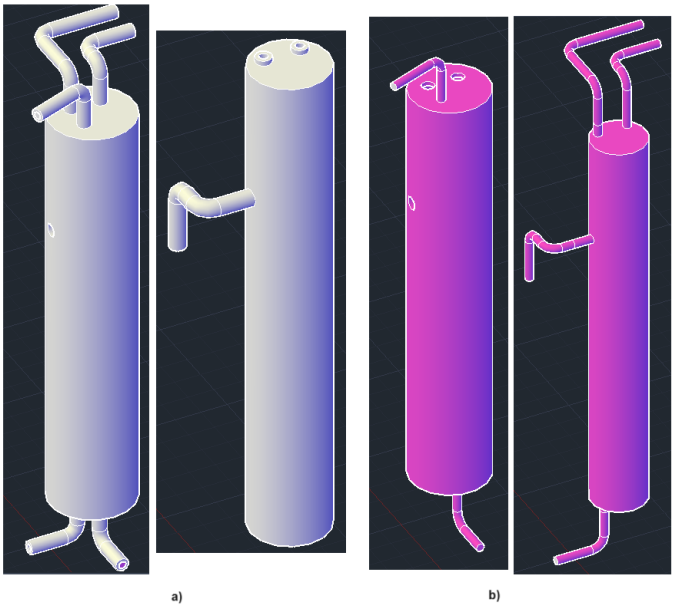


Figura 4- 4: Detalle de los componentes del depósito siendo a) el exterior y b) el relleno de cada uno

- Intercambiador de calor. Al igual que con el depósito se añaden al modelo original los trayectos finales de la tubería, siendo su aspecto final el de la Figura 4-5. Se diseña el relleno del intercambiado como se muestra en la Figura 4-6.

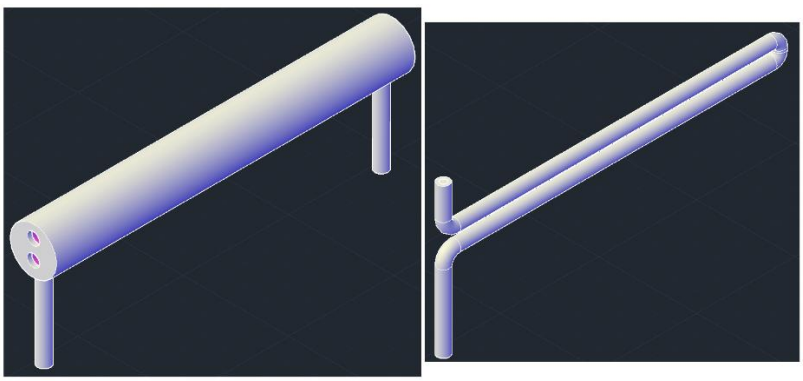


Figura 4- 5: Detalle de los elementos exteriores del intercambiador

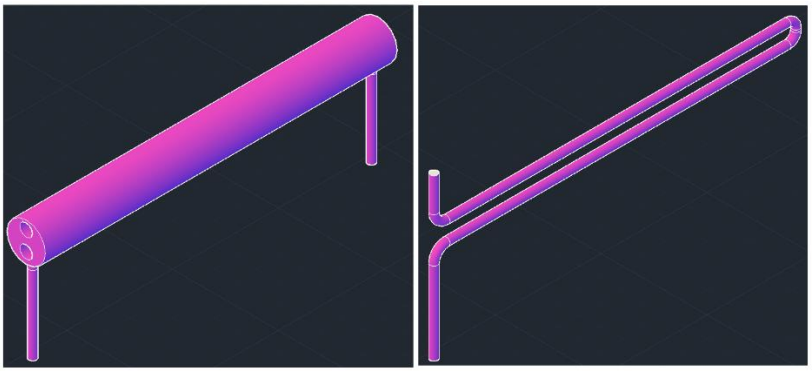


Figura 4- 6: Detalle del relleno de los elementos del intercambiador

- Tanque de abastecimiento y grupo de presión. Se tomó la decisión de unir estos dos objetos en uno, de forma que el tanque de abastecimiento fuera el origen del agua y esta se introdujera al sistema a través de la tubería de alimentación. En la Figura 4-7 se observa el objeto final y su relleno.

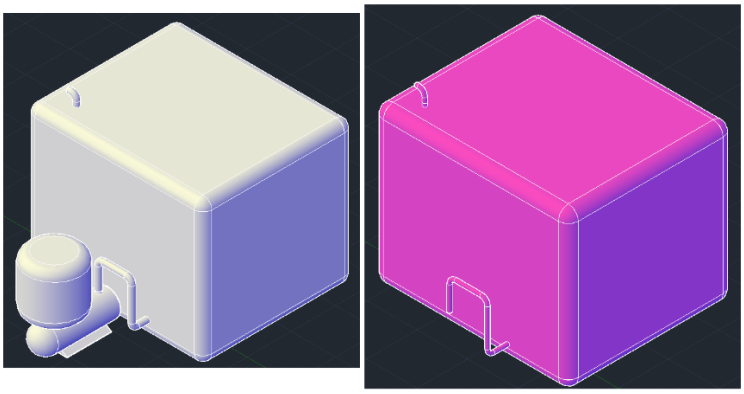


Figura 4- 7: Detalle del conjunto del tanque de abastecimiento y grupo de presión (izquierda) y su relleno (derecha)

- Termo eléctrico. Para el termo eléctrico se ha seguido el mismo procedimiento que con el tanque de abastecimiento. En la Figura 4-8 se muestra el elemento rediseñado.

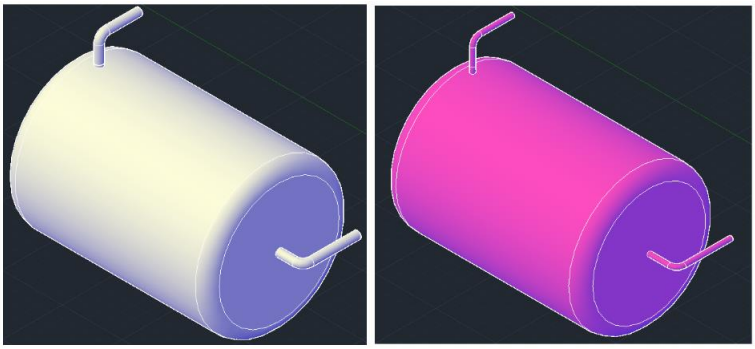


Figura 4- 8: Detalle del termo eléctrico (izquierda) y su relleno (derecha)

- Tuberías. Las únicas modificaciones que se realizan a las tuberías es el cambio de grosor, el diseño de sus rellenos y añadirles las válvulas en sus recorridos. En el caso de la tubería de realimentación, se le ha añadido también el compresor. En las Figuras 4-9, 4-10, 4-11, 4-12 se observan las nuevas tuberías.

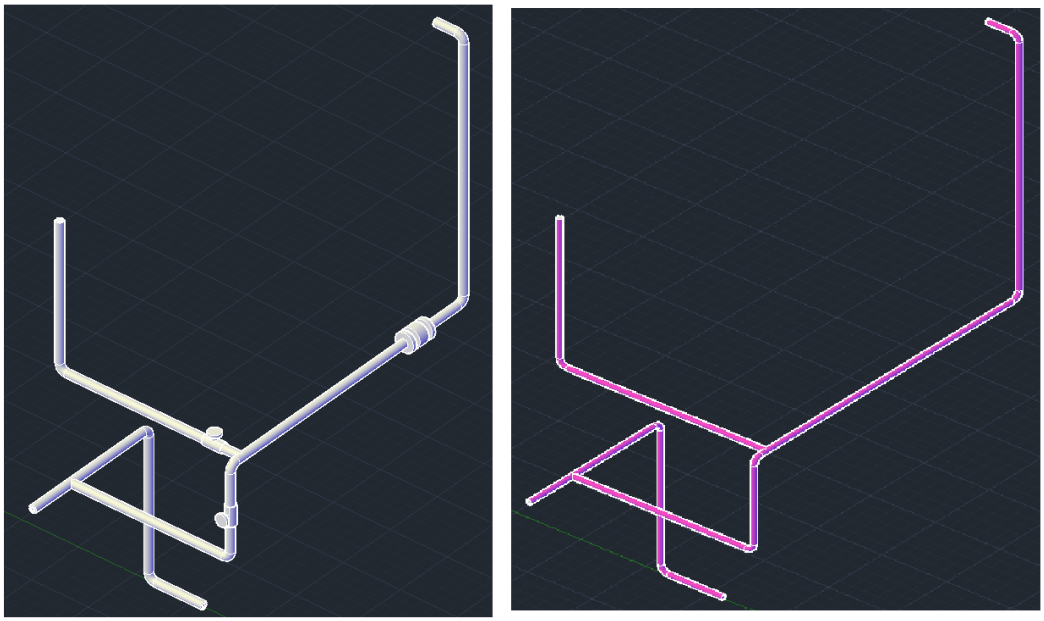


Figura 4- 9: Detalle de la tubería de agua fría (izquierda) y su relleno (derecha)

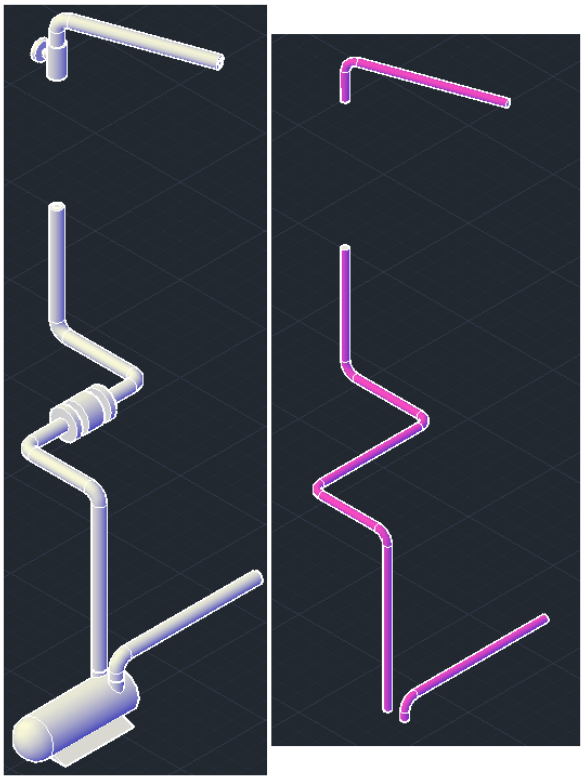


Figura 4- 10: Detalle de la tubería de recirculación (izquierda) y su relleno (derecha)

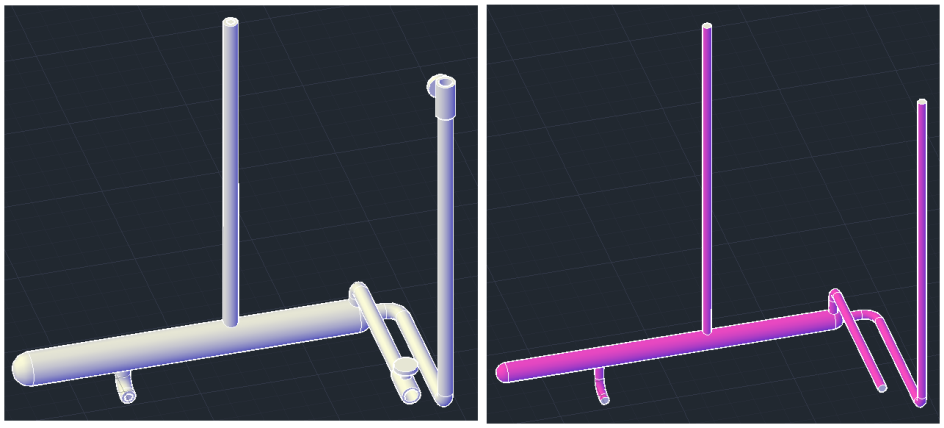


Figura 4- 11: Detalle de la tubería de salida (izquierda) y su relleno (derecha)

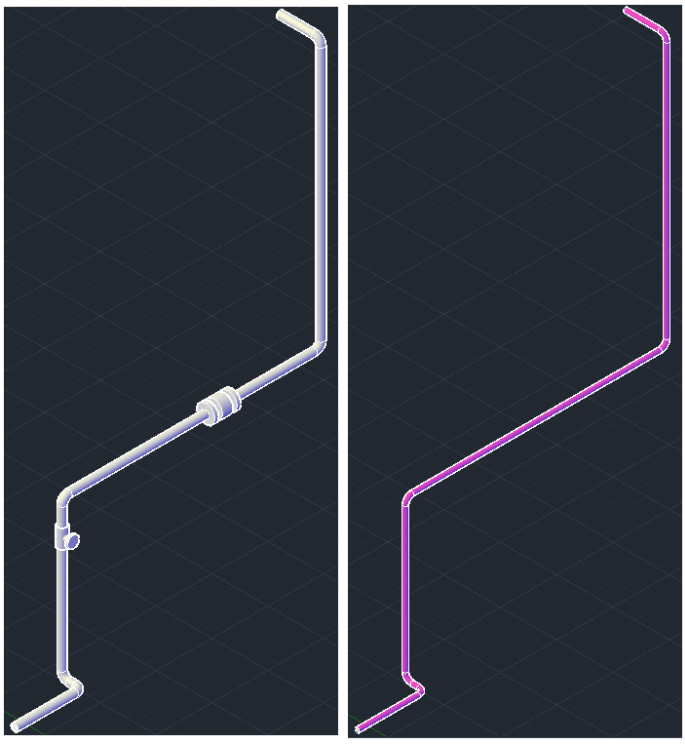


Figura 4- 12:Detalle de la tubería de agua caliente (izquierda) y su relleno (derecha)

Una vez se finaliza el rediseño de la planta, se trasladan cada uno de sus elementos de forma individual a Unity, de esta forma si en el futuro se realizan cambios a la planta, es posible agregarlos con facilidad. Para hacer que sea más cómoda la ubicación de cada componente de la planta en Unity se ha añadido al diseño una pareja de sólidos rectangulares a modo de empalme (ver Figura 4-13), de esta forma es posible colocar con mayor precisión cada elemento.

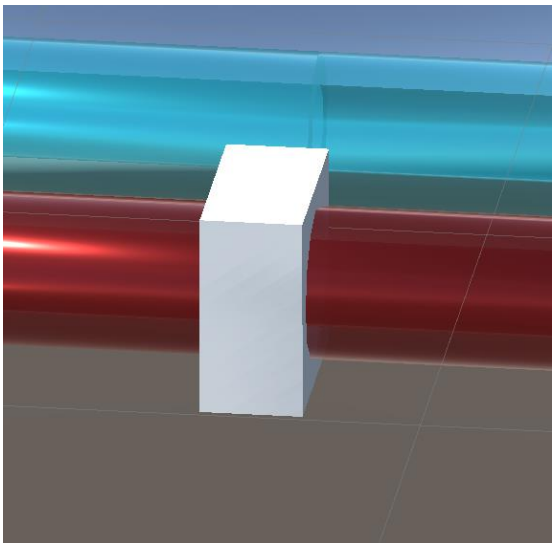


Figura 4- 13: Unión de la tubería de agua fría con el depósito

Una vez la planta ha sido montada en Unity, se añade en la pestaña *Scene* tres planos para actúen a modo de paredes y suelo, siendo la vista que se observa en la pantalla la que se muestra en la Figura 4-14.

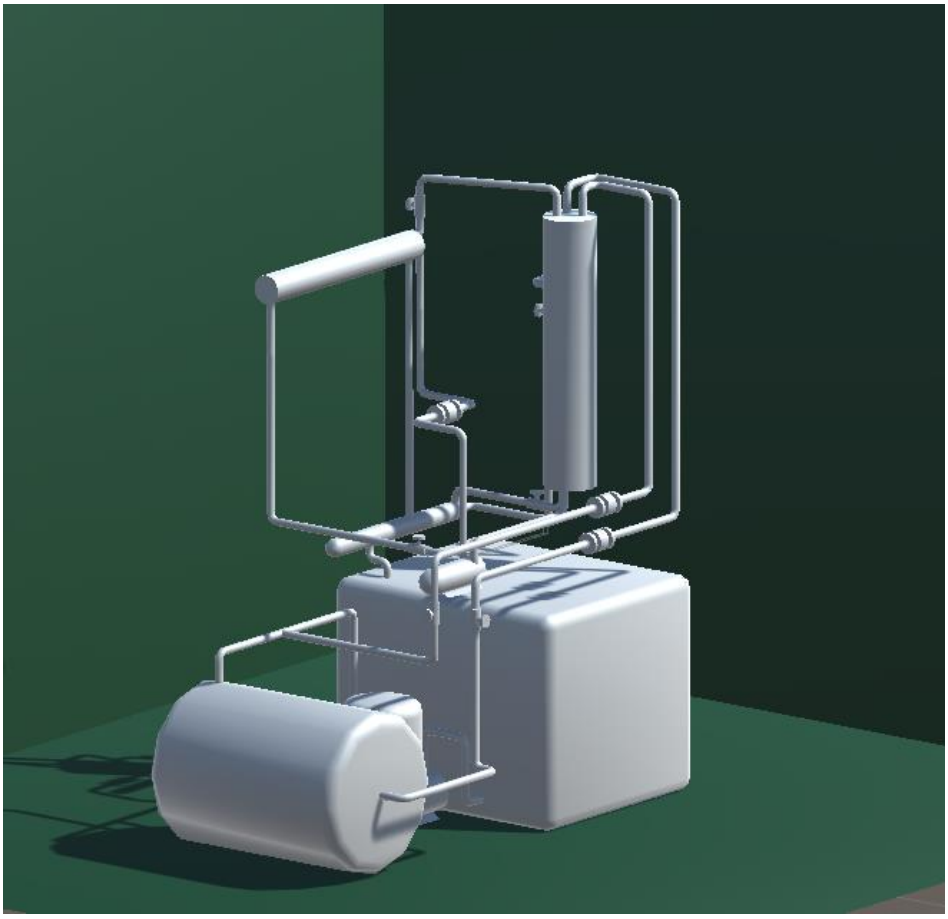


Figura 4- 14: Planta importada a Unity 3D

El siguiente paso es asignar a cada elemento un material, este no solo proporcionará color, sino que además añadirá una textura. Se han distinguido dos tipos de materiales:

- Opacos: se utiliza este material para los elementos por los cuales no circulará el agua en la simulación. Estos son las manivelas de las válvulas, el suelo, las paredes, los compresores y el tanque hidroneumático del grupo de presión.
- Traslucidos: para el resto de los elementos se utilizan materiales de tipo *Fade*, estos son materiales traslucidos que permiten ver el flujo de agua moverse por dentro de ellos.

Los materiales creados se muestran en la Figura 4-15.

Figura 4- 15: Materiales creados en Unity

La planta, con los materiales asignados a sus elementos se muestra en la Figura 4-16.

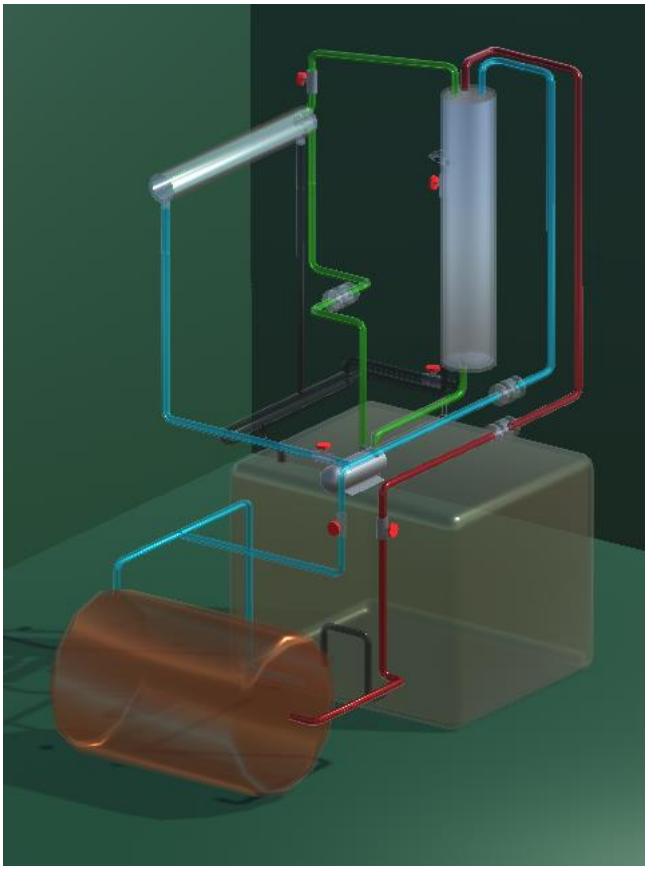


Figura 4- 16: Planta con los materiales asignados a cada elemento

4.3. Configuración de Zibra Liquids

Para la simulación de la planta en Unity en un principio se consideró utilizar un único volumen de trabajo que comprendiera la totalidad de los elementos de la planta, pero debido a la complejidad de la forma de los *colliders* y a la necesidad de un gran número de partículas para la simulación del fluido en todo el recorrido, se descartó esta opción. Como solución se decidió realizar la simulación de forma escalonada, mostrando uno a uno los elementos de la planta. La principal ventaja esta simulación es que se reducen los recursos del ordenador necesarios para realizarlos, permitiendo que la visualización del agua sea más nítida, que no se produzcan ralentizaciones y que no aparezcan comportamientos extraños.

Para realizar esta simulación se van a diseñar diferentes volúmenes de trabajo para cada uno de los elementos de la planta, además de definir los *colliders* para los distintos rellenos de los objetos. Para definir los *colliders* en primer lugar se añade el componente *Zibra Collider* al objeto en la pestaña Inspector (ver Figura 4-17) y se indica que se desea utilizar una Neural SDF, la cual la genera Unity de forma automática y se selecciona que va a ser del tipo invertido. Adicionalmente se desactiva el *Mesh Renderer* para que el relleno no sea visible.

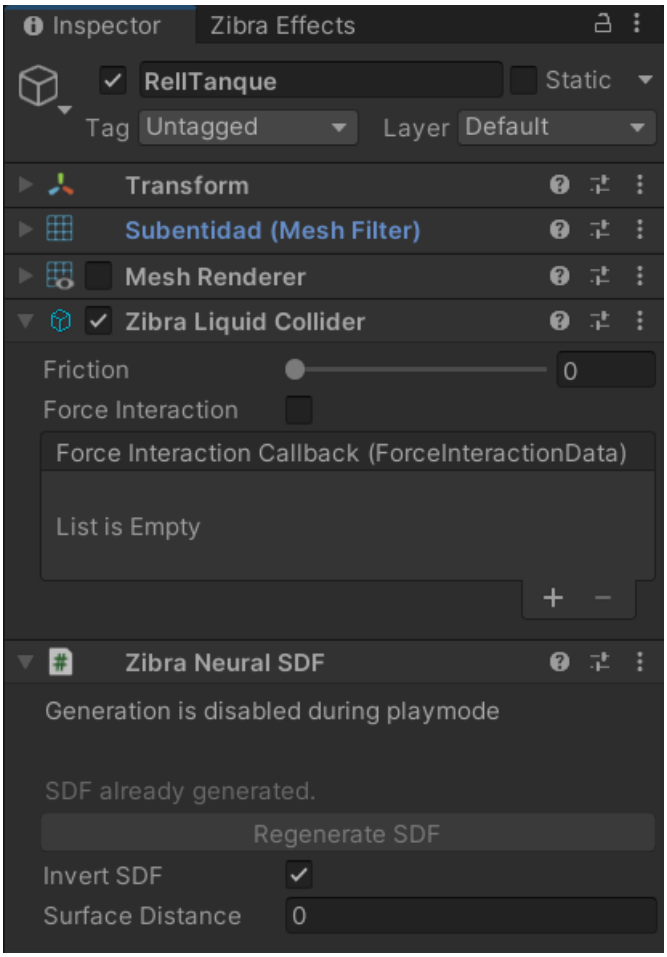


Figura 4- 17: Pestaña inspector del relleno del tanque de abastecimiento

Una vez se han creado todos los *colliders*, se procede con el diseño de los volúmenes de trabajo, lo cuales se describen a continuación:

- Tanque de abastecimiento. El tanque de abastecimiento va a constar de tres volúmenes: el del propio tanque, la tubería de salida que transmite el agua a la planta y la tubería de entrada al

tanque que recibe el agua de la planta.

El volumen del tanque de abastecimiento constará de un *void* para simular el vaciado cuando el agua entra en la tubería de salida y un *emitter* que actuará a modo de entrada. Una de las funcionalidades de Zibra es que permite iniciar la simulación existiendo ya líquido en la escena, por lo que se decide utilizar esta herramienta para que el tanque este parcialmente lleno de agua cuando se inicializa la simulación. Cabe destacar que el tanque no se llena del todo debido a que son necesarias más partículas de las que el ordenador puede simular. En la Figura 4-18 se puede visualizar el estado inicial del tanque de abastecimiento, junto con los elementos de su volumen de trabajo.

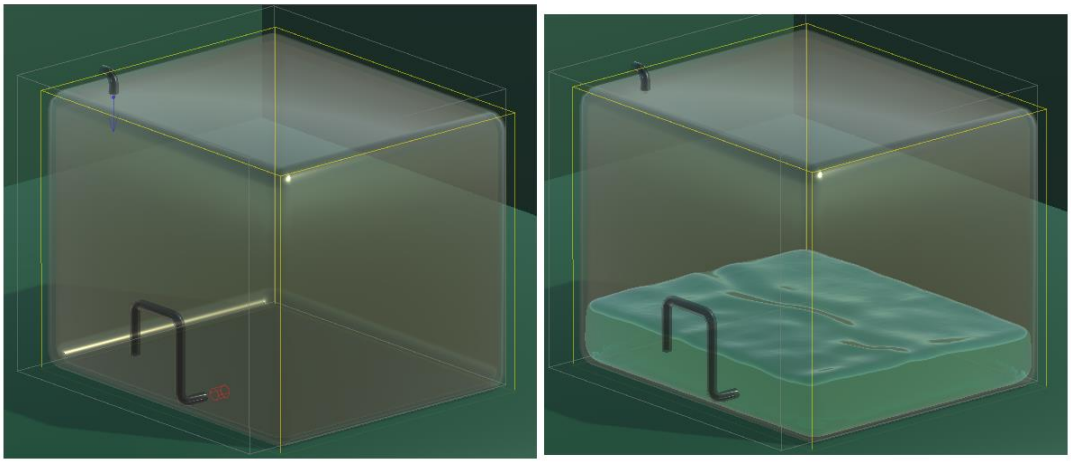


Figura 4- 18: Volumen de trabajo del tanque de abastecimiento (izquierda) y simulación de este elemento (derecha)

Para el volumen de las tuberías de entrada y salida se define un *emitter* y un *void* para cada una. En el caso de la tubería de salida, al tener esta un recorrido con dos giros es necesario definir también dos *force fields*: uno que ayude a impulsar el agua y otro que ejerza una fuerza de atracción, emulando así la acción del compresor. En la Figura 4-19 puede observarse los dos volúmenes de trabajo y en la Figura 4-20 cada volumen trabajando con el tanque de abastecimiento. Se añaden también un detector al final del recorrido de cada tubería para detectar cuando el agua llega a ese punto.

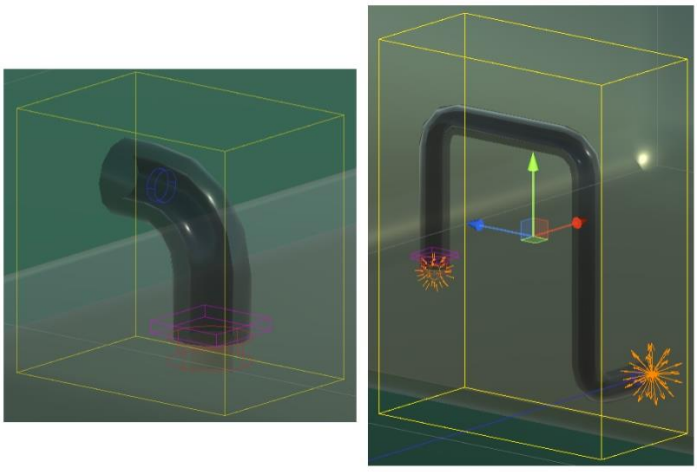


Figura 4- 19: Volumen de trabajo de la entrada al tanque de abastecimiento (izquierda) y la salida

del tanque (derecha)

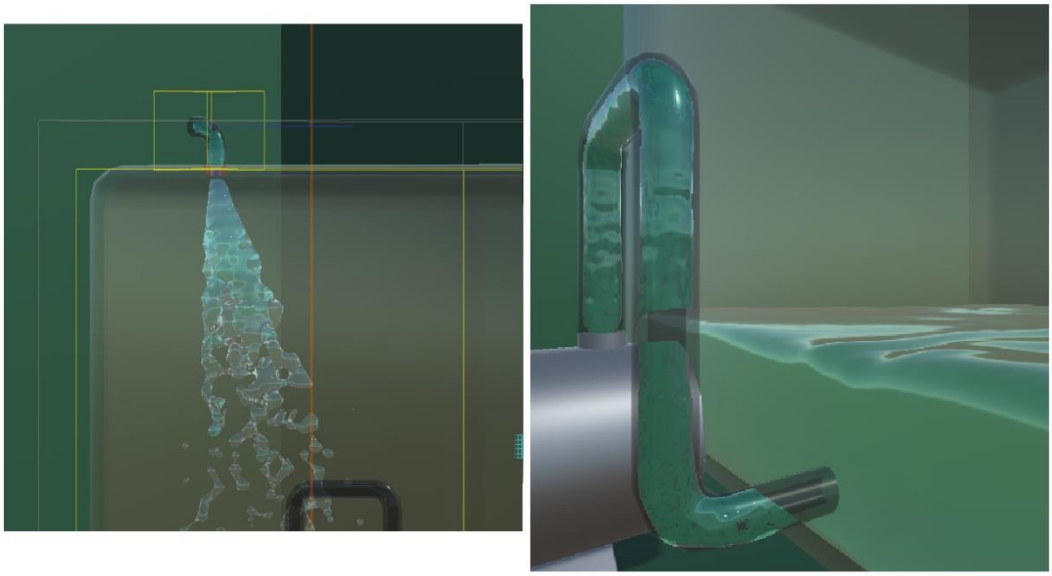


Figura 4- 20: Simulación de la entrada al tanque de abastecimiento (izquierda) y la salida del tanque (derecha)

- Termo eléctrico: se procede igual que con el tanque de abastecimiento, definiendo tres volúmenes de trabajo. En este caso no es necesario el uso de *force fields* en las tuberías ya que estas son muy cortas. En la Figura 4-21 se presenta el termo inicializado con agua en su interior y su volumen de trabajo, en él se ha creado un emisor en la entrada de agua fría y un *void* en el inicio de la tubería de agua caliente. En la Figura 4-22 se muestran los volúmenes de las dos tuberías con sus respectivos emisores, *voids* y detetectores. El conjunto en funcionamiento se muestra en la Figura 4-23.

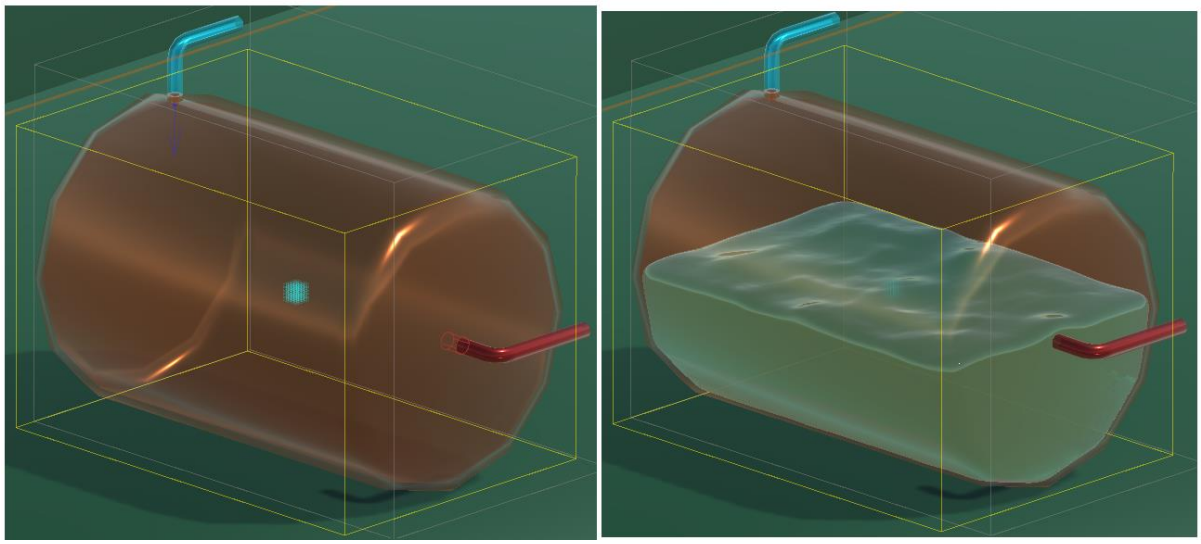


Figura 4- 21: Volumen de trabajo del termo (izquierda) y simulación de este elemento (derecha)

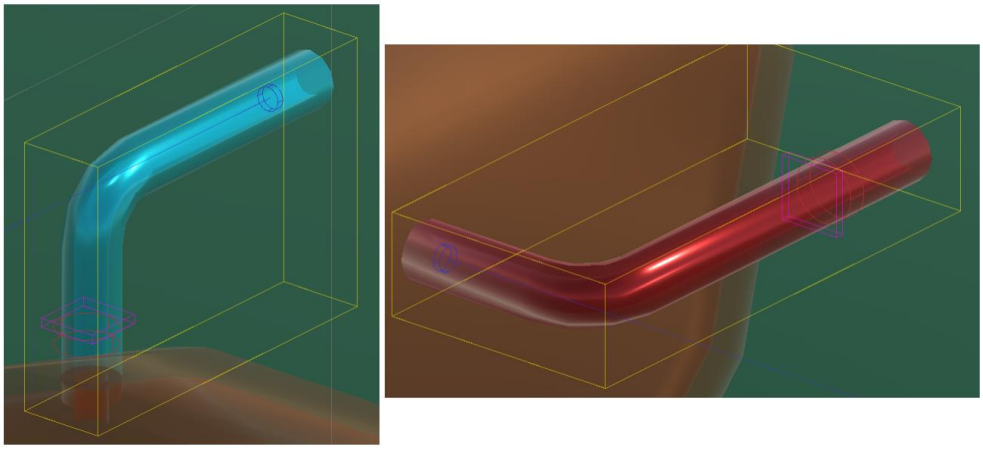


Figura 4- 22: Volumen de trabajo de la entrada al termo (izquierda) y la salida del termo (derecha)

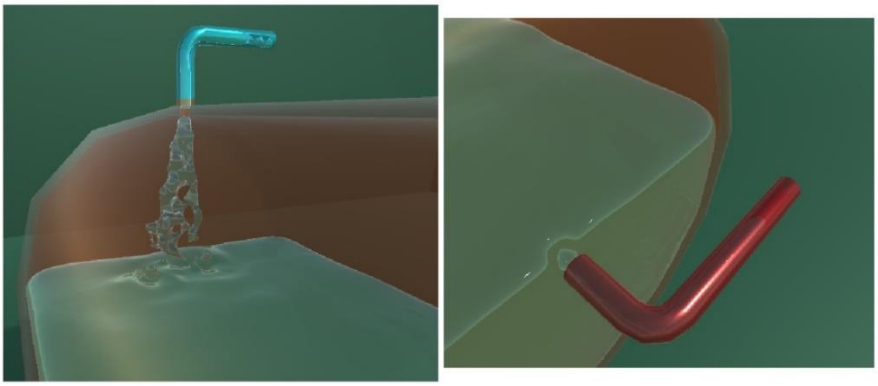


Figura 4- 23:Simulación de la entrada al termo (izquierda) y la salida del termo (derecha)

- Intercambiador. El intercambiador se va a separar en tres volúmenes distintos. El primero se corresponde con la carcasa exterior y las tuberías por las que circula el agua de refrigeración. Debido a que el sistema tarda mucho en llenar el intercambiador, se establece un estado inicial en el cuál este ya está lleno y se crea un emisor en la tubería de entrada, además de un *void* y un detector en la tubería de salida. En la Figura 4-24 se observa el volumen de trabajo y el intercambiador en funcionamiento).

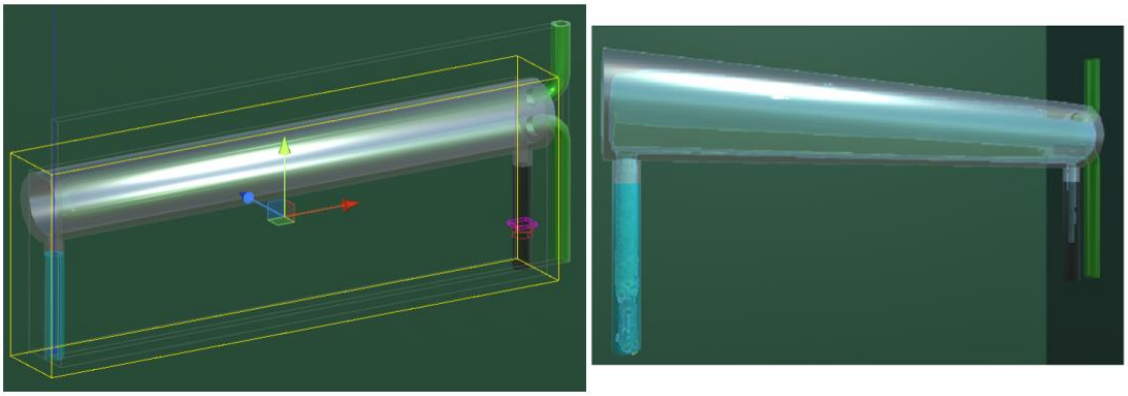


Figura 4- 24: Volumen de trabajo del exterior del intercambiador (izquierda) y simulación de este elemento (derecha)

Para la tubería interior del intercambiador ha sido necesario definir dos volúmenes, ya que al no tener los emisores fuerza suficiente para que el agua circule por tuberías ascendentes, es necesario el uso de *force fields*, y en este caso uno tan fuerte, que impide que el agua alcance el final de recorrido de este tramo de tubería. Para ambas secciones de las tuberías se crean un emisor al inicio y un *void* junto a un detector al final, junto con dos *force fields* uno para impulsar el agua al inicio y otro para atraerla al final. En las Figuras 4-25 y 4-26 se muestra el detalle de la tubería y su simulación respectivamente.

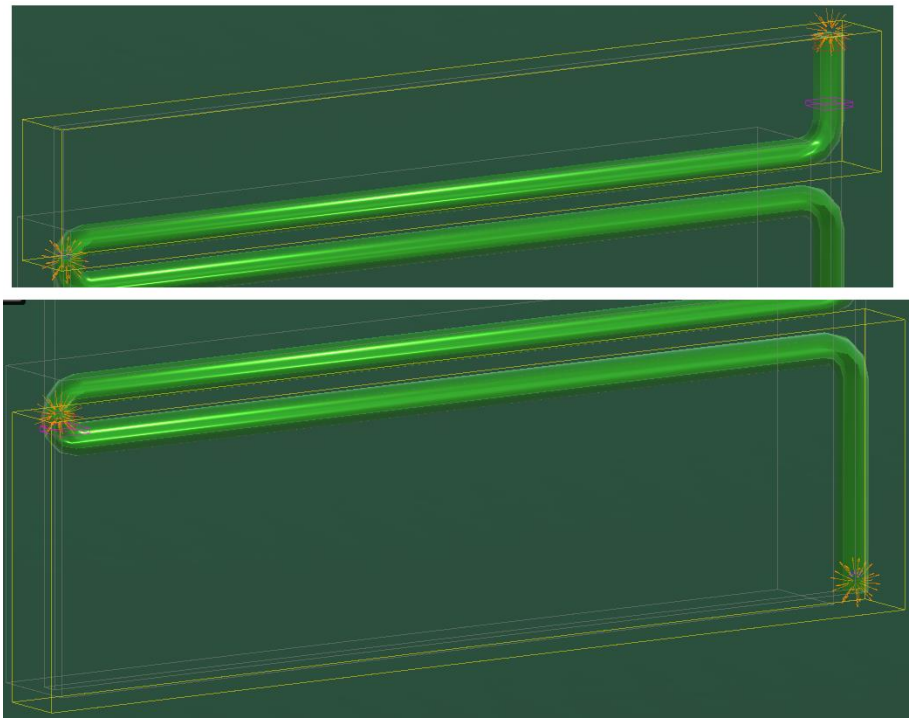


Figura 4- 25: Volúmenes de trabajo de la tubería interior del intercambiador

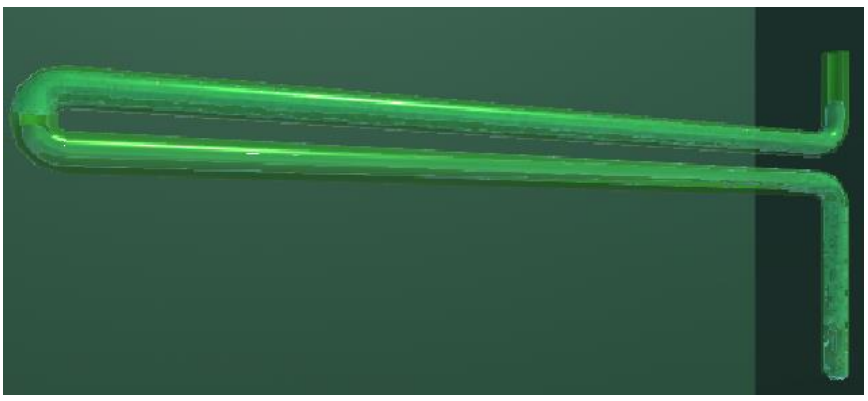


Figura 4- 26: Simulación de la tubería interior del intercambiador

- Depósito. Se han creado dos volúmenes de trabajo para el depósito.

  El volumen correspondiente a la camisa de refrigeración consta de un emisor en la tubería de entrada de recirculación para simular el paso de agua por ella y un void con un detector al final de esta. Cuando el agua llega al detector se activará mediante un script un emisor más

grande ubicado en la parte superior del tanque simulando que el agua circula por la camisa. Finalmente se colocará un void en la tubería de salida de recirculación. El volumen de trabajo de la camisa se muestra en la Figura 4-27.

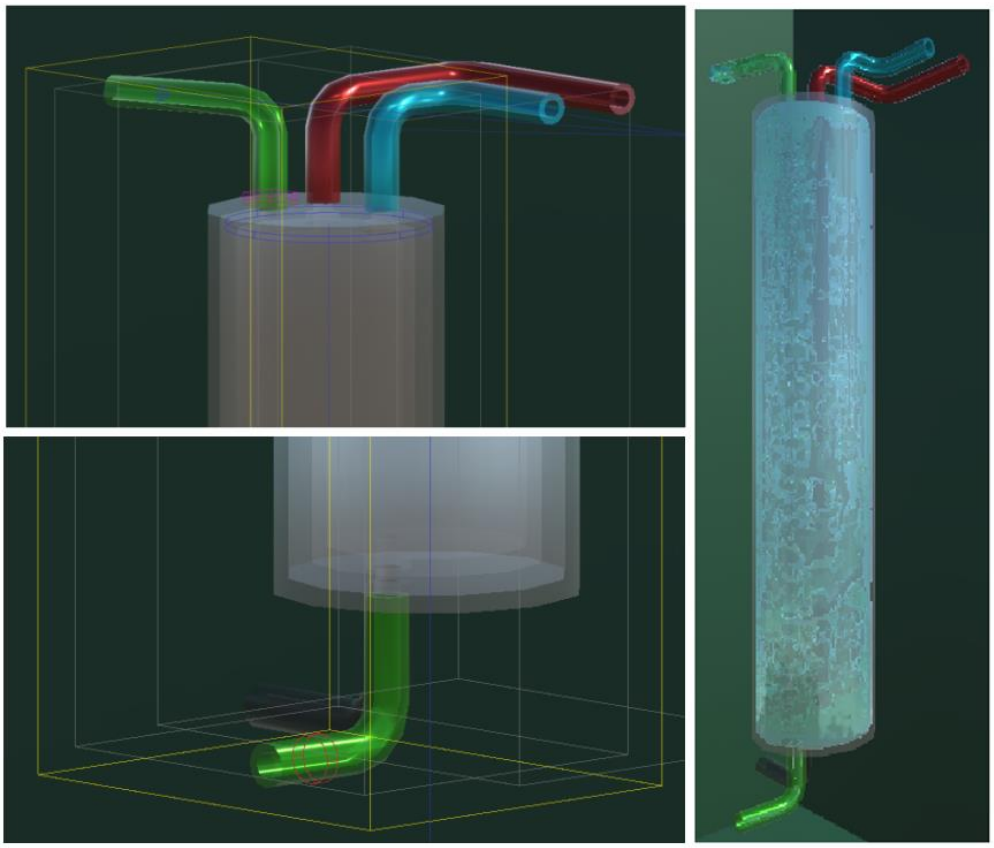


Figura 4- 27: Volumen de trabajo de la camisa de refrigeración (izquierda) separado en parte superior e inferior, y simulación de este elemento (derecha)

El segundo volumen se corresponde con el depósito interior, donde se mezcla el agua fría con la caliente. En este caso habrá dos emisores, uno en cada tubería de entrada y dos voids (en la tubería del rebosadero y en la salida). A la hora de simular se ha observado que cuando el agua llega a la entrada del depósito esta es bloqueada por la forma del collider, por lo que se decidió añadir un detector para poder activar otros dos emisores dentro del depósito que simule el agua cayendo y un void rectangular que bloquee el paso de los emisores de las tuberías. La simulación del interior del depósito se inicializa con agua en su interior.

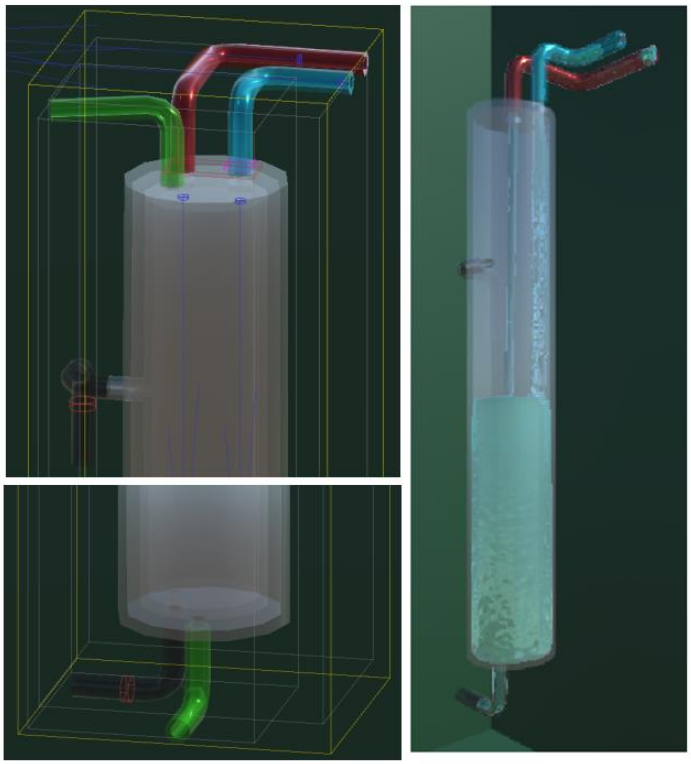


Figura 4- 28: Volumen de trabajo del interior del depósito (izquierda) separado en parte superior e inferior, y simulación de este elemento (derecha)

- Tubería de agua fría. Dada la complejidad de su trayectoria se va a dividir en tres tramos distintos. Los dos primeros volúmenes de trabajo van a contener en su interior las dos bifurcaciones de la tubería. Se ubicarán sensores al final de cada tramo y un void en las válvulas para simular el cierre de estas. Por medio de la programación este void se activará o desactivará en función de las necesidades del usuario. En las Figuras 4-29, 4-30, 4-31 y 4-32 se muestra cada tramo de tubería con su volumen de Zibra. Los detectores ubicados en la mitad del trayecto son utilizados para cambiar el ángulo de la cámara en el modo vídeo.

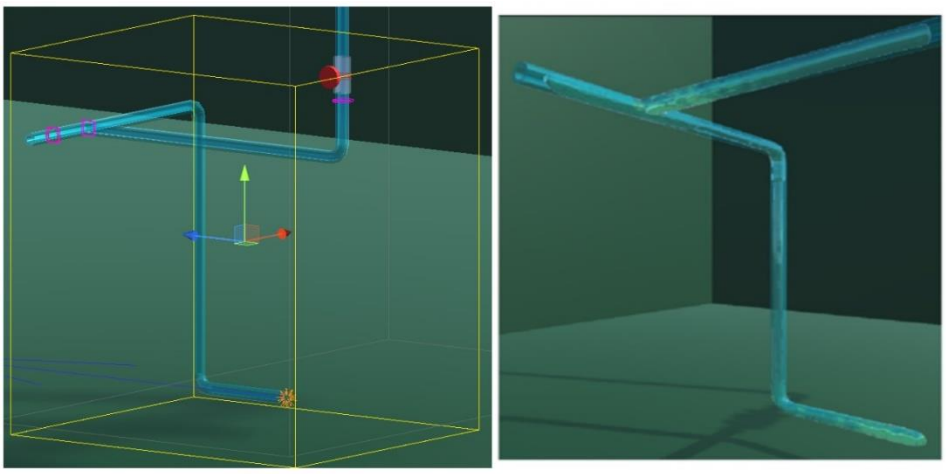


Figura 4- 29: Volumen de trabajo del primer tramo de la tubería de agua fría (izquierda) y simulación de este elemento (derecha)

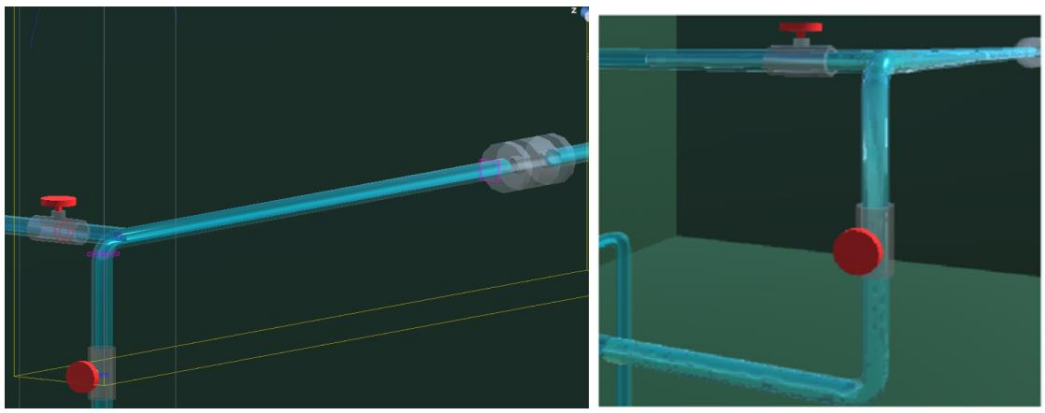


Figura 4- 30: Detalle de la primera parte del volumen de trabajo del segundo tramo de la tubería de agua fría (izquierda) y simulación de este elemento (derecha)

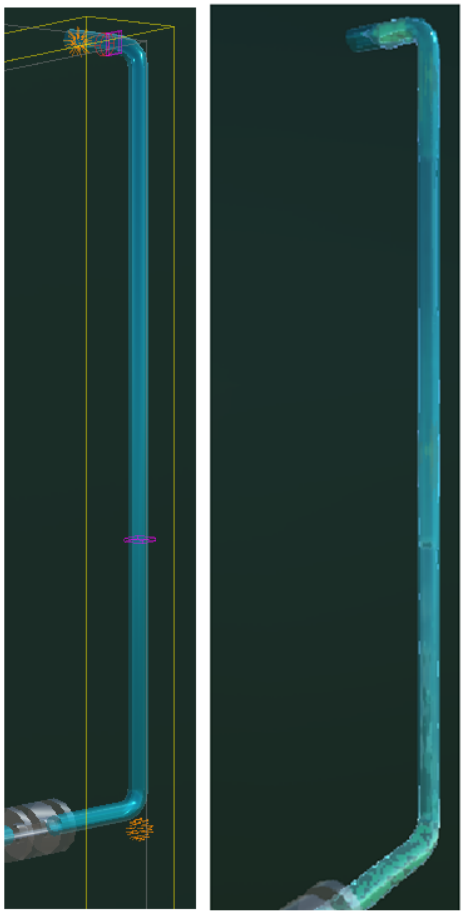


Figura 4- 31: Detalle de la segunda parte del volumen de trabajo del segundo tramo de la tubería de agua fría (izquierda) y simulación de este elemento (derecha)

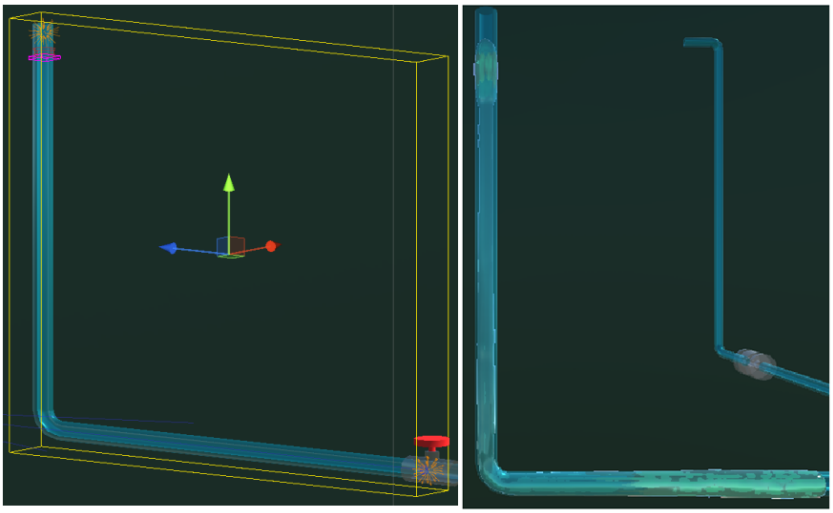


Figura 4- 32: Volumen de trabajo del tercer tramo de la tubería de agua fría (izquierda) y simulación de este elemento (derecha)

- Tubería de agua caliente. La tubería de agua caliente se va a dividir en dos tramos con sus correspondientes volúmenes de trabajo: un primer tramo que llega hasta la mitad de la válvula automática y un segundo tramo que se corresponde con el resto de la tubería. Al igual que en la tubería de agua fría, la válvula manual se simula con un void. En las Figuras 4-33 y 4-34 se pueden observar los dos volúmenes de trabajo.

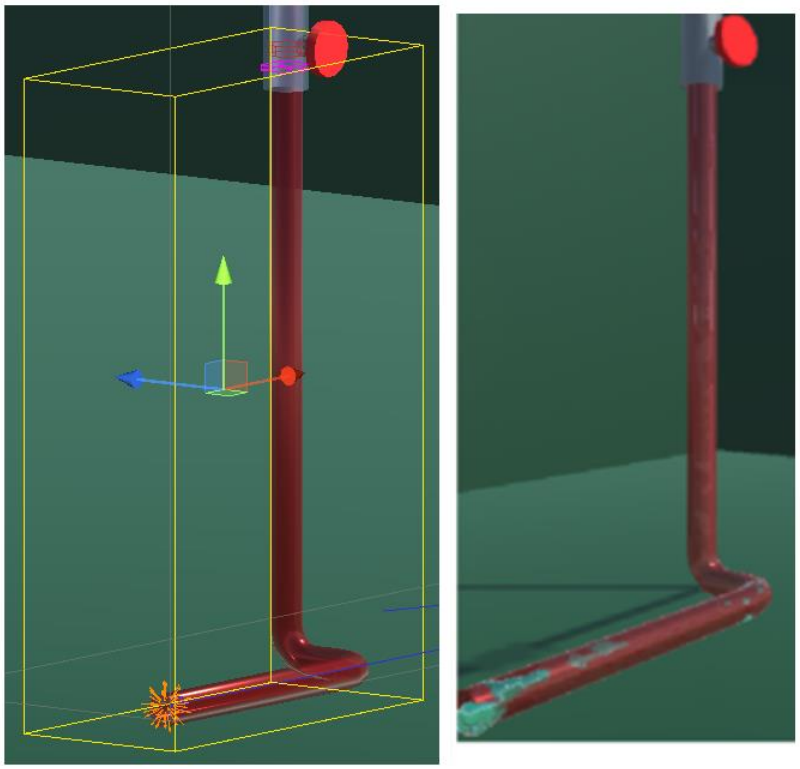


Figura 4- 33:Volumen de trabajo del primer tramo de la tubería de agua caliente (izquierda) y simulación de este elemento (derecha)

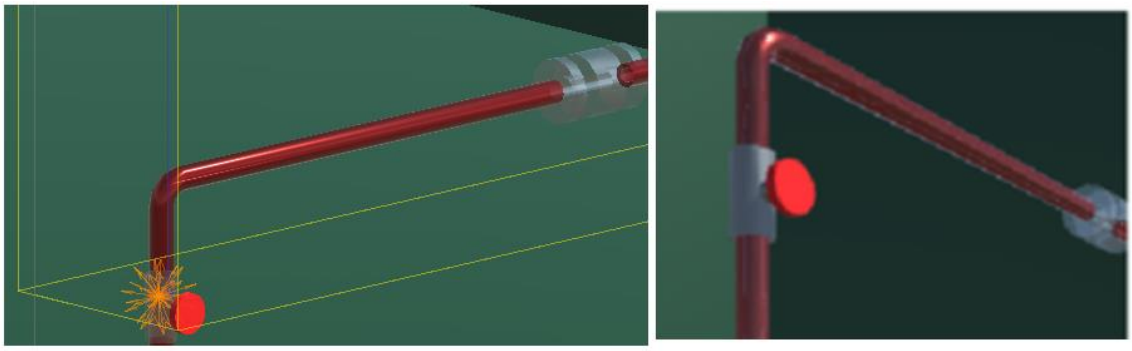


Figura 4- 34: Detalle de la primera parte del volumen de trabajo del segundo tramo de la tubería de agua caliente (izquierda) y simulación de este elemento (derecha)

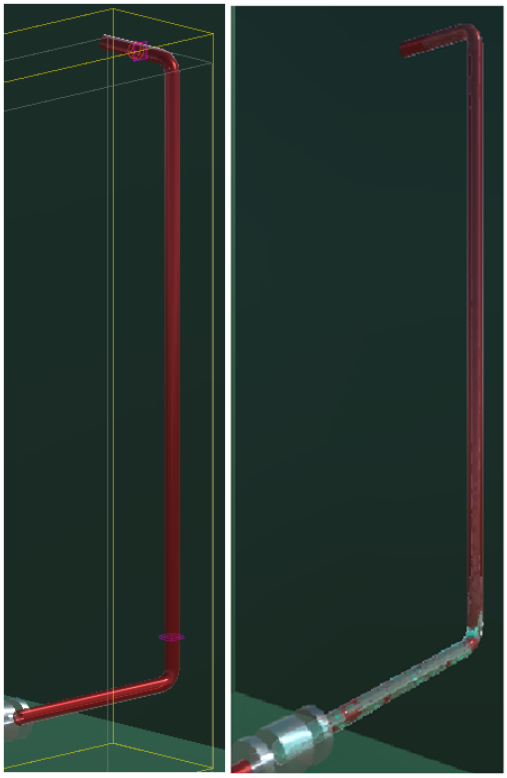


Figura 4- 35: Detalle de la segunda parte del volumen de trabajo del segundo tramo de la tubería de agua caliente (izquierda) y simulación de este elemento (derecha)

- Tubería de recirculación. Al estar separada en tres tramos, se crea para cada uno un volumen de trabajo. En el caso del segundo tramo, es necesario añadir *force fields* para impulsar el agua hacia arriba. En las Figuras 4-36, 4-37 y 4-38 se representan los volúmenes de trabajo y sus simulaciones. En el tercer tramo fue necesario utilizar un void con forma toroidal para evitar que algunas partículas salieran de la tubería, esto se produjo por la estructura del collider de este tramo.

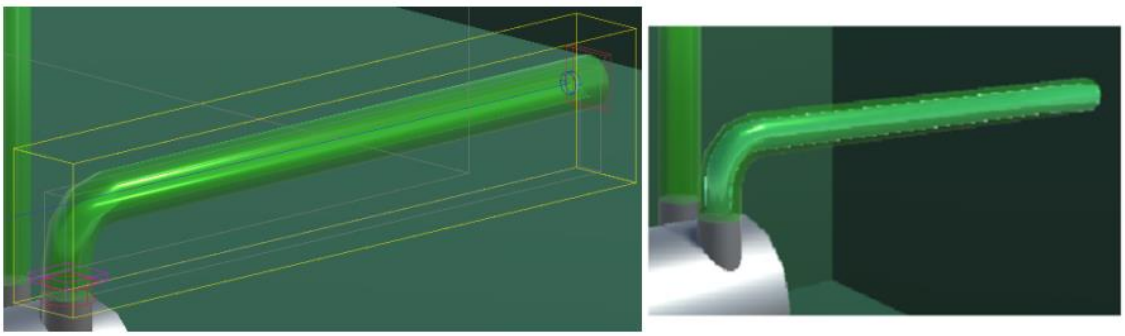


Figura 4- 36: Volumen de trabajo del primer tramo de la tubería de recirculación (izquierda) y simulación de este elemento (derecha)

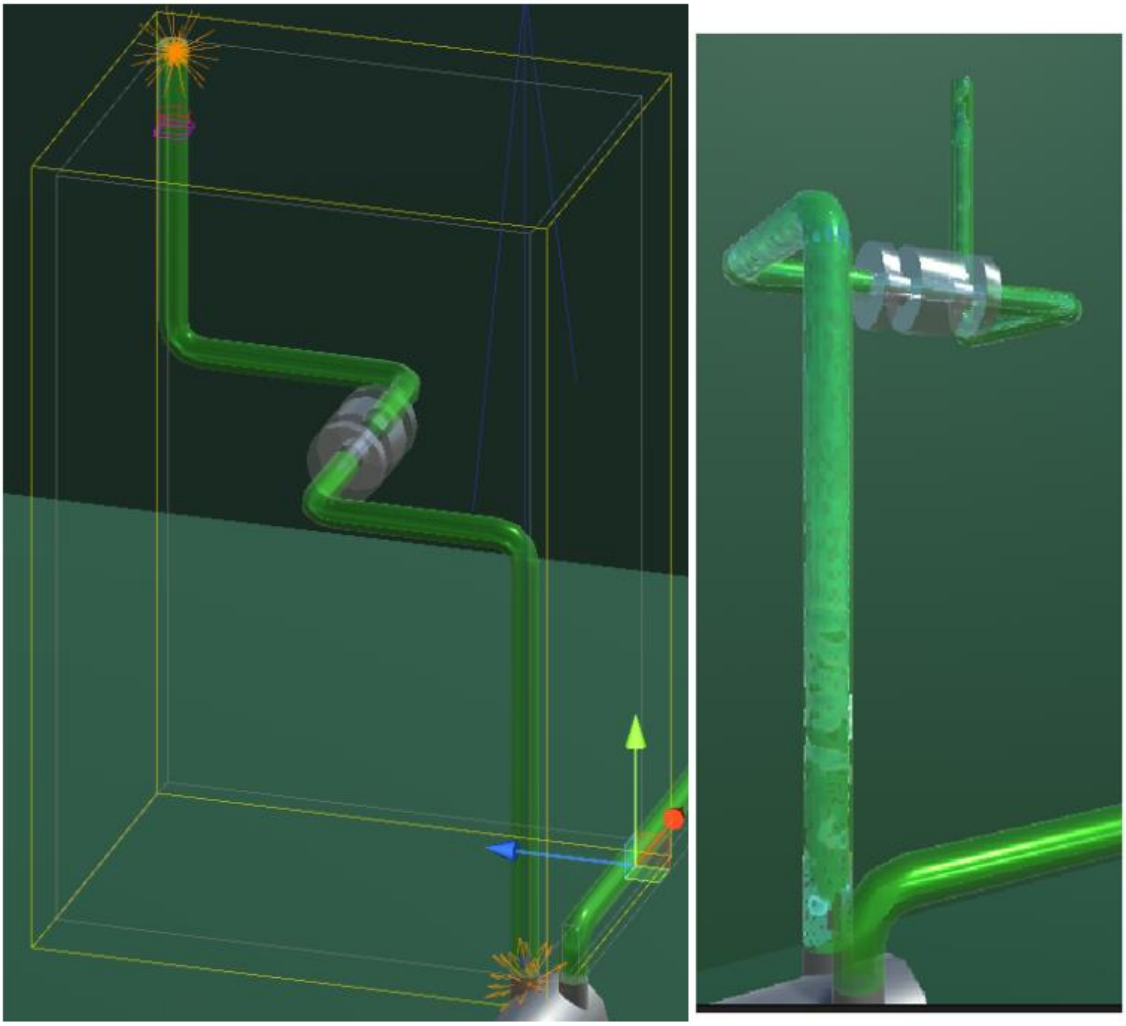


Figura 4- 37:Volumen de trabajo del segundo tramo de la tubería de recirculación (izquierda) y simulación de este elemento (derecha)

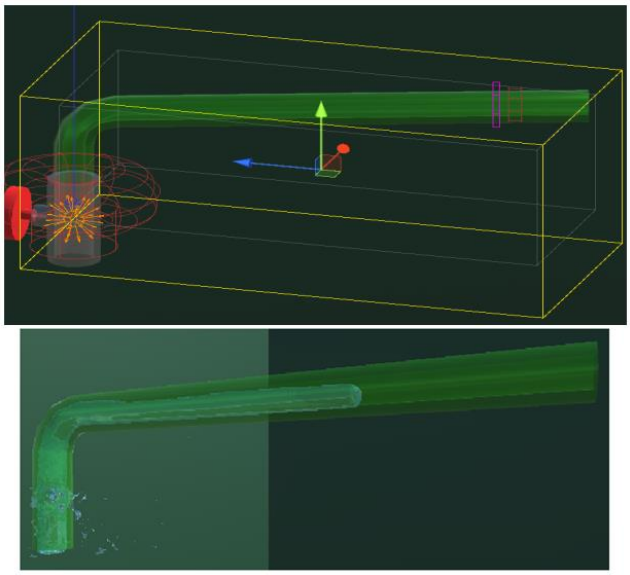


Figura 4- 38: Volumen de trabajo del tercer tramo de la tubería de recirculación (izquierda) y simulación de este elemento (derecha)

- Tubería de salida. Para la tubería de salida solo es necesario definir un único volumen de trabajo con cuatro emisores, uno para cada tubería que la compone. En la Figura 4-39 se muestra la tubería.

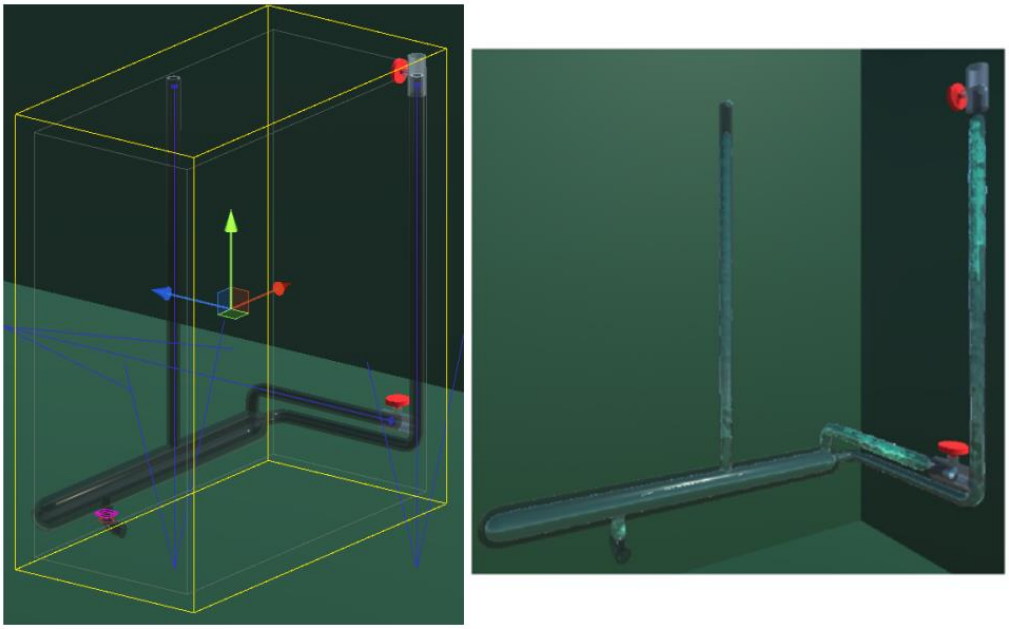


Figura 4- 39: Volumen de trabajo de la tubería de salida (izquierda) y simulación de este elemento (derecha)

Tras haber creado los volúmenes de trabajo de cada elemento se puede proceder a la programación de la planta y a la validación del funcionamiento del depósito.

4.4. Programación del gemelo digital

La programación del gemelo digital se va a dividir en tres apartados:

- Desarrollo de un programa que permita ver cada elemento de la planta en funcionamiento y que pueda visualizarse como un video.
- Desarrollo y validación de un programa que calcule el nivel de agua dentro del depósito, usando como datos de entrada los flujos de agua caliente, fría y de salida, de forma que pueda apreciarse en el gemelo.
- Diseño de una interfaz que permita abrir o cerrar las válvulas manuales, cambiar entre los dos programas de los puntos anteriores y la introducción y visionado de los datos de entrada y salida de la planta.

4.1.1 Desarrollo de la interfaz

Para poder alternar entre los diferentes modos de visualización y funcionamiento de la planta, se decidió crear una interfaz que permitiera esto, además de controlar las válvulas manuales y realizar la lectura de parámetros de entrada. Para ello se utilizarán las herramientas de interfaz de usuario (UI Toolkit) de Unity.

En primer lugar, para diferenciar los diferentes modos de funcionamiento de la planta se utilizará la herramienta *Dropdown*, que genera un menú desplegable donde se presentaran tres opciones que se describen a continuación.

- Inicio: representa el estado inicial de la planta. En él se muestra una imagen de toda la planta, el dropdown y una columna de toggles que representan el estado de las válvulas manuales. Estos toggles pueden marcarse para abrir las válvulas manuales. En la Figura 4-39 se muestra la vista de la simulación al seleccionar el modo Inicio.

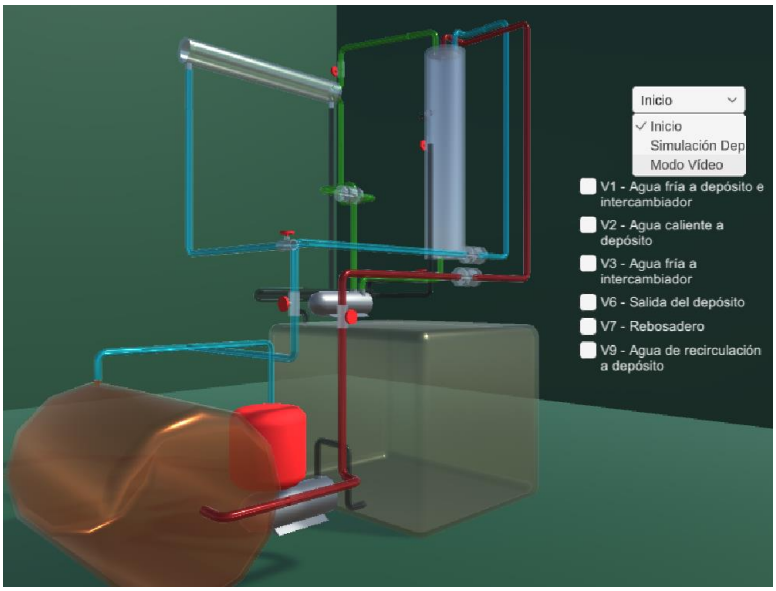


Figura 4- 40: Vista del programa en al seleccionar Inicio

- Simulación depósito: en este modo se visualizará el depósito y se realizarán los cálculos necesarios para obtener la altura en cada instante. La interfaz de esta modalidad consta de tres toggles: los correspondientes a las válvulas que controlan el flujo de agua caliente y fría, y

uno que permite inicializar el cálculo de la altura. También se crean dos sliders para la introducción de los datos de entrada de flujo con sus respectivos textos para mostrar el valor según la posición del slider y un display que muestra la altura de agua dentro del depósito. La vista de este modo se muestra en la Figura 4-40.

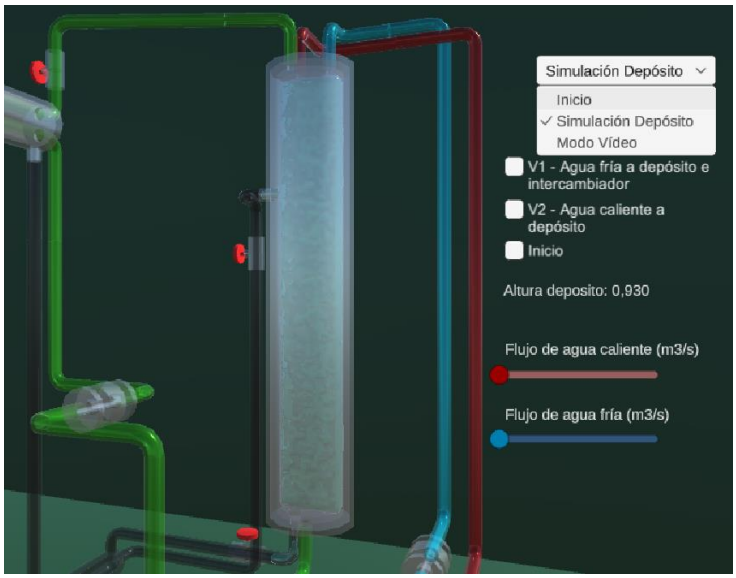


Figura 4- 41: Vista modo depósito

- Modo vídeo: al seleccionarse esta opción se ejecutará un modo en el que se verá el desplazamiento del agua por dentro de las tuberías. En caso de que algunas de las válvulas no estén abiertas, el agua se detendrá hasta que se abra la válvula. En la Figura 4-41 se muestra uno de los tramos del recorrido del agua en el modo vídeo.

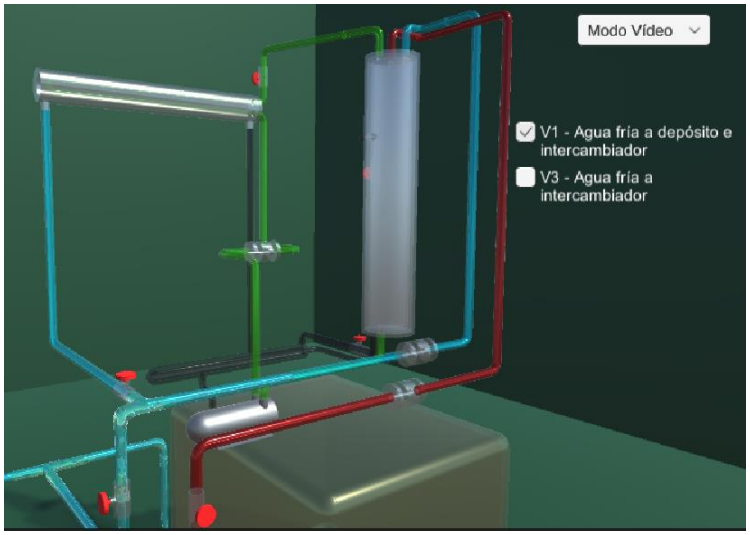


Figura 4- 42: Vista de uno de los tramos de la tubería de agua fría estando las válvulas V1 abierta y V3 cerrada

Los scripts relacionados con la interfaz se pueden comprobar en el Anexo I.

4.1.2 Desarrollo del modo vídeo

El modo vídeo va a consistir en un proceso escalonado en el cual, cada sección de la planta se activará mostrando el fluido moviéndose por dentro de los diferentes elementos que la componen.

Para este proceso, en primer lugar, se van a guardar en distintos GameObjects las posiciones que la cámara va a tomar en cada instante de la simulación. Estos objetos estarán guardados en la jerarquía dentro de un objeto padre, llamado PosicionesCámara. Durante la ejecución del modo vídeo, el programa leerá propiedad Transform donde se guardan las coordenadas y ángulos del objeto y actualizará los de la cámara.

Una vez definidas las posiciones de las cámaras se desarrolla el programa para cada elemento de la planta. Estos programas siguen una estructura similar que se describe a continuación:

- En primer lugar, se definen las variables públicas y privadas que se van a usar. En el caso de que una variable pública sea del tipo GameObject, permite que se le indique los objetos que se va a utilizar en el programa. Estos objetos pueden ser desde los volúmenes de trabajo, emisores, detectores o directamente propiedades de los objetos. En el Ejemplo 4-1 se observan las variables de uno de los programas.

Ejemplo 4–1. *variables en un programa, en concreto del primer tramo de la tubería de agua fría:*

```
public Video Vid;
public GameObject TubFria1, EmitterAux, Emitter;
public GameObject Camara;
public Transform  PosCamara1, PosCamara2;
public ZibraLiquidDetector Detector, DetectorAux, DetectorAux2;
```

- En la sección del Void Start() (esta parte del programa se ejecuta una sola vez, cuando se inicializa) se establecen los valores originales de las variables y se asegura que algunos elementos estén desactivados, para lo que se utiliza el comando *SetActive(false)*. Un ejemplo de esto se muestra en el Ejemplo 4-2, en este caso se asegura de que el emisor auxiliar del primer tramo de la tubería esté apagado. Estas secciones del programa pueden resultar más complejas dependiendo de si el mismo elemento se accederá en distintos momentos de la planta, lo que hace necesario añadir variables auxiliares.

Ejemplo 4–2. *Void Start() del primer tramo de la tubería de agua fría:*

```
void Start(){
    EmitterAux.SetActive(false);
}
```

- El Void Update() del programa es el que se va a encargar de inicializar el volumen de trabajo de Zibra (mediante el comando *SetActive(true)*, cambiar la posición de la cámara y comprobar si se ha alcanzado el sensor o no. Una vez detecta que las partículas han alcanzado los sensores, se actualiza la variable que permite saltar a otro elemento de la planta y se desactivan

los GameObjects que se han utilizado, esto se puede observar en el Ejemplo 4-3, en donde el programa espera primero a recibir la señal de un primer sensor para activar el emisor auxiliar, y luego la señal de otro sensor para saltar al siguiente elemento del vídeo. En el caso del depósito, tanque de abastecimiento y termo, el salto de elemento se realiza por medio de un temporizador como se muestra en el Ejemplo 4-4. La desactivación de los volúmenes de Zibra se producen cuando el elemento deja de estar presente en la vista de la cámara.

Ejemplo 4–3. *Void Update() del primer tramo de la tubería de agua fría:*

```
void Update(){
    if (Vid.Index == 1 && Vid.Activar){
      Camara.transform.SetPositionAndRotation(PosCamara1.position,PosCamara1.rotation);
      TubFria1.SetActive(true);
      if (DetectorAux.ParticlesInside != 0){
        EmitterAux.SetActive(true);
      }
      if (Detector.ParticlesInside != 0 && Vid.Index == 1){
        Vid.Index = 2;
      }
    }
    else if (Vid.Index >= 1 && Vid.Index <= 2){
      TubFria1.SetActive(false);
      Emitter.SetActive(false);
    }
    else if (!Vid.Activar)      {
      TubFria1.SetActive(false);
    }
    if (Vid.Index == 3 && Vid.Activar){
      Camara.transform.SetPositionAndRotation(PosCamara2.position,PosCamara2.rotation);
      TubFria1.SetActive(true);
      EmitterAux.SetActive(true);
      if (DetectorAux2.ParticlesInside != 0 && Vid.Index == 3){
        Vid.Index = 4;
      }
    }
    else if (Vid.Index >= 5){
      TubFria1.SetActive(false);
    }
  }
```

Ejemplo 4–4. *Void Update() con temporizador, correspondiente a la salida del termo:*

```
void Update(){
    if (Vid.Index == 5 && Vid.Activar){
        Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
        Termo.SetActive(true);
        TubSalTermo.SetActive(true);
        if (Detector.ParticlesInside >= 10 && Vid.Index == 5){
            Vid.Index = 6;
        }
    }
    else if (Vid.Index >= 7){
        Termo.SetActive(false);
        TubSalTermo.SetActive(false);
        Vid.Aux1 = true;
    }
    else if (!Vid.Activar){
        TubSalTermo.SetActive(false);
        Termo.SetActive(false);
        }
  }
```

Al estar la programación de Unity orientada a objetos, los scripts deben de estar asociados a un GameObject. Cada uno de los scripts diseñados para el modo vídeo se asignan a los objetos creados con Autocad, para facilitar así su localización. Tras la asignación de los scripts en la pestaña Inspector, es necesario definir las variables publicas correspondientes a objetos o propiedades, para ello solo se deben arrastrar los elementos correspondientes a las casillas del script, como se muestra en la Figura 4-43.

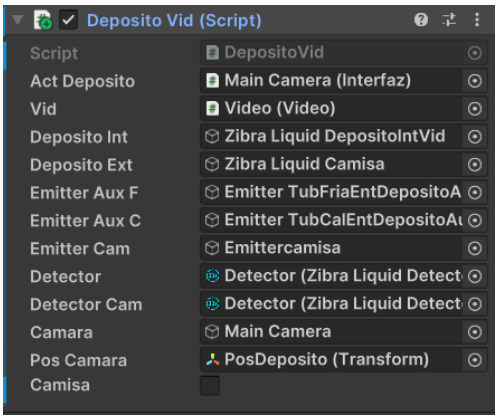


Figura 4- 43: Ejemplo de objetos asociados al script del depósito interior.

Todos scripts del Modo Vídeo se muestran en el Anexo II.

4.1.3 Desarrollo del programa de simulación del deposito

El objetivo de este script será el de mostrar la altura que debe alcanzar el agua en base a los flujos de entrada de agua caliente y fría en cada instante y que serán introducidos mediante sliders.

Para poder iniciar la programación en primer lugar debe de obtenerse un modelo matemático que represente el comportamiento del depósito. Dicho modelo es el que se muestra a continuación:

$$A \cdot \frac{dh(t)}{dt} = q_F(t) + q_C(t) - a \cdot \sqrt{2 \cdot g \cdot h(t)} \qquad (1)$$

Siendo $h(t)$ la altura en m, $q_F(t)$ y $q_C(t)$ el flujo de agua fría y caliente respectivamente, en m^3/s, "a" el área del orificio de salida del depósito, "A" el área del depósito y "g" la gravedad. Este modelo es una ecuación diferencial, por lo que debe aproximarse con el objetivo poder trabajar con ella en el programa. Para ello, la derivada temporal se aproxima utilizando el método de Euler hacia delante, como se indica en la ecuación (2) y se obtiene al despejar la ecuación (3).

$$\frac{dh(t)}{dt} \approx \frac{h(k+1) - h(k)}{T_s} \qquad (2)$$

$$h(k+1) = h(k) + \frac{T_s}{A} \cdot \left[q_F(k) + q_C(k) - a \cdot \sqrt{2 \cdot g \cdot h(k)}\right] \qquad (3)$$

Siendo Ts el tiempo de muestreo.

Una vez se ha obtenido el modelo matemático es posible proceder con la programación de la simulación del depósito. Esta simulación va a estar compuesta por cuatro scripts, donde cada uno controlara un aspecto distinto:

- Uno de los scripts se encargará de la visualización y obtención del valor de los sliders. Se estableció que el rango de las sliders fuera de 0 a $5.45 \cdot 10^{-3}$ m^3/s de esta forma si se los dos flujos estaban en su valor máximo se compensaba la velocidad de vaciado del depósito. En el momento en el que se mueva uno de los sliders, se guardará el valor que este tome y lo muestra en un cuadro de texto al lado. En el Código 4-1 se muestran las funciones principales del script.

Código 4–1. *Scripts de control de sliders:*

```
public void sliderQc(float valorQc){
    FlujoQc.text = valorQc.ToString("f5");
    qc = valorQc;
  }
```

```
public void sliderQf(float valorQf){
    FlujoQf.text = valorQf.ToString("f5");
    qf = valorQf;
}
```

- Para la visualización de la altura del depósito se utiliza el script Textos, que lo que hace es la diferencia entre la altura máxima y mínima en la que el sensor detecta partículas de fluido y muestra esa altura en un cuadro de texto. En el Código 4-2 se muestra el script.

Código 4–2. *Script de visualización de altura:*

```
void Update()
  {
    Altura = SensorNivel.BoundingBoxMax - SensorNivel.BoundingBoxMin;
    h = Altura.y;
    Nivel.text = "Altura deposito: " + h.ToString("f3");
  }
```

- El Script de cálculo va a ser el principal de la simulación. En un principio se inicializan las variables con los valores que se muestran en la Tabla 4-1, y tras esto el programa espera a que se pulse le toggle de inicio.

Tabla 4- 1: Variables iniciales.

Variable	Nombre	Valor	Unidades
hmax	Altura máxima del depósito	0,94	[m]
hmin	Altura mínima del depósito	0	[m]
R	Radio del interior del depósito	0.1	[m]
r	Radio interior de las tuberías	0.0285	[m]
Ts	Tiempo de muestreo	0.1	[s]

En el momento en el que toggle de inicio es pulsado se inicia la corrutina Inicialización, (esta parte del script se muestra en el Código 4-3), las corrutinas son unas funciones de Unity que permiten ejecutar partes de un script pausando la ejecución y reiniciándola en un tiempo definido o en el siguiente frame, esto evita que se pierda información, ya que normalmente, Unity ejecuta varias veces las funciones en un mismo frame. Mientras el toggle de inicio este pulsado el script comprobará si las válvulas V1 y V2 están abiertas, en el caso de estar cerradas los flujos de entrada tendrán un valor de 0 m^3/s, a partir de que estén abiertas, esos

flujos tomaran el valor que se indique con los sliders y activaran unos emisores que simularan el agua cayendo en el depósito, por medio de la función ComprobarEmitters (Código 4-4).

Código 4–3. *Corrutina del script de cálculo:*

```
public void Inicializar()
  {
    StartCoroutine(Espera());
  }

  IEnumerator Espera(){
    while(Botones.Inicio){
      yield return new WaitForSeconds (Ts);
      if (Botones.V1){
        Qf = Flujos.qf;
      }
      else{
        Qf = 0f;
      }
      if (Botones.V2){
        Qc = Flujos.qc;
      }
      else{
        Qc = 0f;
      }
      ComprobarEmitters();
      Ecuacion();
      VisualizarAltura();
      texto.EscribirDatos();
      t=t+Ts;
      if (!Botones.Inicio){
        break;
      }
    }
  }
```

Código 4–4. *Función ComprobarEmitters() del script de cálculo:*

```
void ComprobarEmitters(){
    if (Qc==0f){
       EmitterC.SetActive(false);
    }
    if (Qf==0f){
       EmitterF.SetActive(false);
    }
    if (Qc!=0f){
       EmitterC.SetActive(true);
    }
    if (Qf!=0f){
       EmitterF.SetActive(true);
    }
  }
```

Una vez obtenidos los valores de los flujos, se procede al cálculo de la altura con la función Ecuación, que se muestra en el Código 4-5. En esta función se calcula el valor que la altura va a tener en el instante siguiente (este valor se guarda en la variable "hk") y se asegura de que no se superen los valores máximos y mínimos de la altura, los cuales se pueden establecer al principio del script. Una vez se ha calculado "hk" se guarda su valor en h, para poder calcular el nuevo valor en la siguiente iteración.

Código 4–5. *Función Ecuación() del script de cálculo:*

```
void Ecuacion(){
    hk = h + (Ts/A)*(Qf+Qc-a*Mathf.Sqrt(2f*9.81f*h));
    if (hk < 0){
       hk = 0;
    }
    if (hk > hmax){
       hk = hmax;
    }
    h = hk;
  }
```

Tras calcular la altura, se procede a la visualización de esta altura en el entorno de Unity, para ello se ejecuta la función VisualizarAltura (que se muestra en el Código 4-6). Para visualización se ha creado otro volumen de trabajo de Zibra específicamente para esta simulación. Este volumen se coloca de forma que solo se trabaje con la mitad del depósito, ya que permite contar con un mayor número de partículas sin comprometer la velocidad de ejecución del programa. En la Figura 4-44 se muestra el nuevo volumen de trabajo.

Código 4–6. *Función VisualizarAltura() del script de cálculo:*

```
void VisualizarAltura()
  {
    yVacioq = Vacio2.transform.position.y;
    yVacioh = Vacio.transform.localScale.y;
    d2 = h+14.88f;
    d1 = 0.94f-h;
    if (yVacioh>d1){
      dyv=yVacioh-d1;
      Vacio.transform.localScale += new Vector3(0, -dyv, 0);
    }
    if (yVacioh<d1){
      dyv=d1-yVacioh;
      Vacio.transform.localScale += new Vector3(0, dyv, 0);
    }
    if(yVacioh==d1){
      Vacio.transform.localScale += new Vector3(0, 0, 0);
    }
    if(yVacioq > d2){
      dyq = yVacioq - d2;
      Vacio2.transform.Translate(0,-dyq,0);
    }
    if (yVacioq < d2){
      dyq = d2 - yVacioq;
      Vacio2.transform.Translate(0,dyq,0);
    }
    if (yVacioq == d2){
      Vacio2.transform.Translate(0,0,0);
    }
  }
```

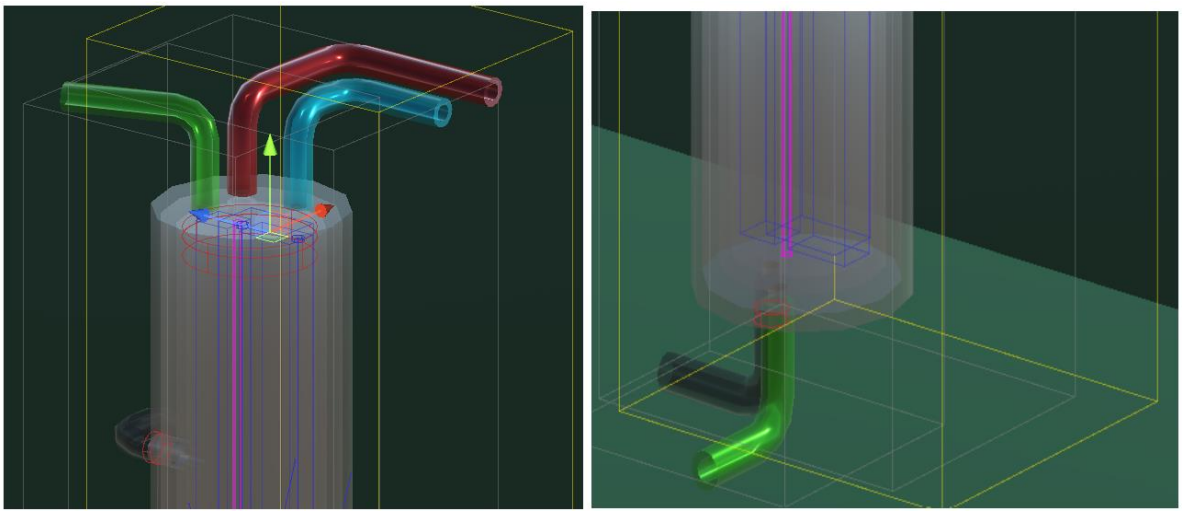


Figura 4- 44:Volumen de trabajo para la simulación del depósito, siendo la imagen de la izquierda la parte superior y la derecha la inferior.

El proceso de visualización consiste en que unos emitters (colocados de forma que no entren en contacto con el sensor de altura) generan partículas de agua de forma continua, y un void cilíndrico, ubicado en la parte más alta del depósito será el encargado de establecer la altura del agua. Para ello en primer lugar se comprobará si la altura del cilindro esta por encimo o por debajo de la altura deseada y en base a esta comparación, la altura del cilindro se modificará para alcanzar la altura calculada. De esta forma todas partículas generadas en este punto se eliminan, visualizándose únicamente las que se generan por debajo del void.

Por otro lado, para simular el agua cayendo desde las tuberías se colocan dos pequeños emisores cilíndricos justo en la entrada de agua del depósito. Estos emisores generaran unas partículas distintas, pero con las mismas características que el agua, de forma que el void encargado de mostrar la altura no las elimine, evitando así que parezca que el depósito se llena por su parte inferior. Para que las partículas de estos emisores aparente caer sobre la superficie del agua dentro del depósito se crea un void cuya posición cambiará según la altura calculada, colocándose justo sobre esa superficie, eliminando del sistema las partículas que caen.

Cuando se han realizado todos los cálculos se llama a la función EscribirDatos, perteneciente a otro script y que se encarga de almacenar los datos en un documento txt.

- El script EscrituraTxt es el encargado de guardar los datos obtenidos por el script cálculo en un documento txt para su visualización en forma de gráfica en Matlab. En primer lugar, crea un nuevo documento con la fecha y la hora en la que se inició la simulación y en cada instante guarda las distintas variables usadas. Las variables guardadas son: los flujos de agua caliente y fría, la altura calculada, la altura medida por el sensor y el tiempo de ejecución. Este script se muestra en el Código 4-7.

Código 4–7. *Script de escritura de texto:*

```
public class EscrituraTxt : MonoBehaviour{
   string ruta, datos, hora, minuto, dia, mes, year,fecha;
```

```
    public Calculo variables;
    public Texto hsensor;
    // Start is called before the first frame update
    void Start()
    {
        hora = System.DateTime.Now.Hour.ToString();
        minuto = System.DateTime.Now.Minute.ToString();
        dia = System.DateTime.Now.Day.ToString();
        mes = System.DateTime.Now.Month.ToString();
        year = System.DateTime.Now.Year.ToString();
        fecha = dia+"-"+mes+"-"+year+"-"+hora+minuto;
        ruta = Application.dataPath + "/DatosMatlab/"+fecha+".txt";
        if (!File.Exists(ruta)){
            File.WriteAllText(ruta,"");
        }
    }
    public void EscribirDatos(){
        datos = (variables.t) + " " + variables.Qf + " " + variables.Qc + " " + variables.h + " " +
hsensor.h + "\n";
        File.AppendAllText(ruta, datos);
    }
}
```

Todos los scripts relacionados con la simulación del depósito se encuentran recogidos en el Anexo III.

5 RESULTADOS

A continuación, se presentarán los resultados obtenidos al ejecutar la simulación del depósito. Para ello se va a realizar una comparación entre los resultados obtenidos del comportamiento del gemelo con los obtenidos de la dinámica del sistema modelada por las ecuaciones del capítulo anterior.

5.1. Resultados obtenidos tras la simulación del depósito

Tras lo expuesto en el capítulo anterior, se procede a ejecutar un script de Matlab para mostrar de forma gráfica los resultados de la simulación. Este script, que se encuentra recogido en el Anexo IV, se ha creado a partir del propio código que utiliza Matlab para lectura de los documentos en formato txt y que permite transformarlos en una tabla. Los elementos de esta tabla serán representados por separado:

- Por un lado, se representa en una única gráfica la altura calculada y los flujos de agua caliente y fría de entrada. Un ejemplo se muestra la Figura 5-1, donde se han ido seleccionando distintos valores de flujo de entrada con los sliders, en este caso es posible simular escalones si en lugar de arrastrar el slider, se pulsa en puntos en concretos.

  En la gráfica se puede observar el comportamiento esperado cuando solo uno de los flujos alcanza su valor máximo, alcanzándose una altura de 0.23 [m] aproximadamente, y al estar los dos flujos en el valor máximo establecido se alcanza una altura de 0.92 [m] muy cercano al valor de altura máxima del depósito.

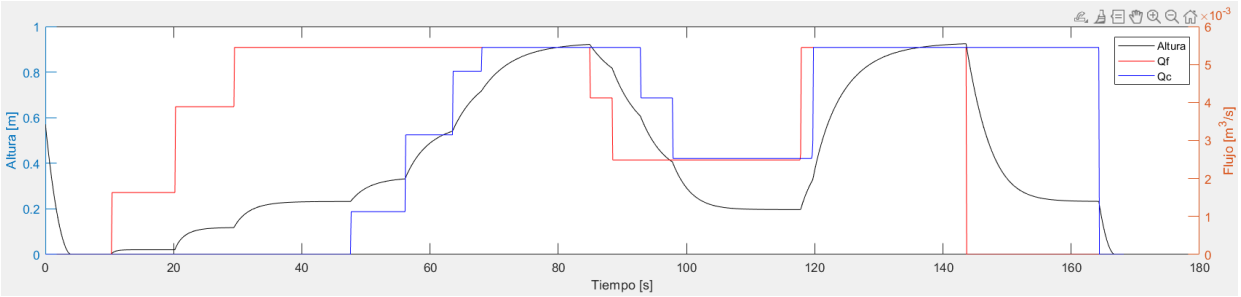


Figura 5- 1: Gráfica donde se representan los flujos de entrada y la altura frente al tiempo

- También se representa la altura calculada con la leída por el sensor. Tomando como ejemplo la Figura 5-2, se observa que la lectura del sensor tiene un pequeño retraso con respecto a la calculada, esto se debe a que el agua tarda en bajar un poco más de lo que el programa calcula la altura que se debe alcanzar.

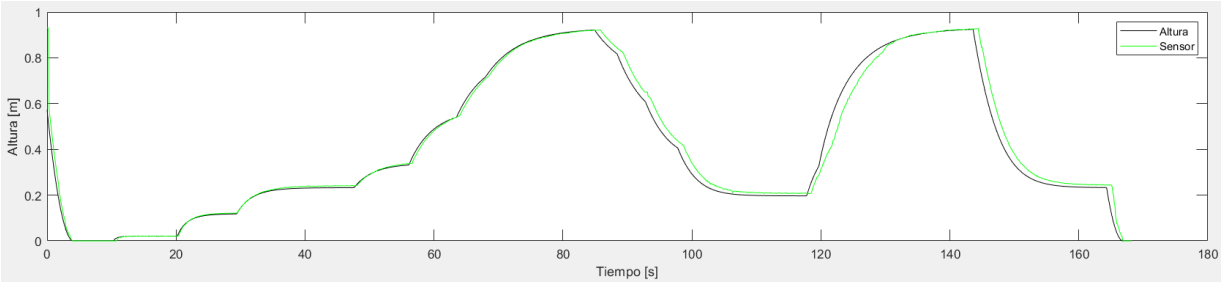


Figura 5- 2: Gráfica donde se representan la altura calculada y la medida frente al tiempo

A la vista de estos resultados se obtiene un comportamiento parecido al esperado de la planta real, aun así, se observa lo siguiente:

- Cuando los cálculos se aproximan a la altura máxima, se puede apreciar que el progreso de aumento de altura se ralentiza llegando a tardar más tiempo en estabilizarse. Este se debe principalmente al método elegido de aproximación.
- En el momento en el que se produce una disminución brusca en la altura, en la animación se puede observar cómo algunas partículas de agua se separan de la superficie, esto no debería ser posible ya que el depósito se vacía por efecto de la gravedad, pero se considera que se debe a como el algoritmo de Zibra simula el comportamiento del fluido.

6 CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo final se expondrán las conclusiones a las que se ha llegado tras realizar este trabajo y se comentaran posibles trabajos futuros que pueden tomar como base este proyecto.

6.1. Conclusiones

Respecto la simulación del depósito, a la vista de los resultados mostrados en el quinto capítulos, se puede considerar que esta aproximación es parecida al comportamiento del depósito en la realidad, si bien no ha podido compararse con los datos reales de la planta, se consideró que, dada la filosofía de los gemelos digitales, el uso de esta ecuación sería suficiente.

Una de las ventajas que ofrece el gemelo digital diseñado y que se considera uno de los principales objetivos, es que permite alterar fácilmente los parámetros con los que se trabaja. Durante la realización de este trabajo, ha sido necesario estimar los valores de algunas de las variables utilizadas y se ha diseñado el gemelo de forma que dichas variables sean fácilmente modificables en un futuro, ya que cuando se ponga en funcionamiento la planta podrán tomarse datos de funcionamiento más precisos que permitan ajustar aún más el modelo matemático del comportamiento.

Durante las configuraciones de los volúmenes de trabajo de cada elemento de la planta y durante la ejecución del *Modo Vídeo* se observó que los colliders de algunos tramos de las tuberías no cubrían la totalidad del recorrido de la tubería (notándose más en los inicios y finales de los tramos), o se estrechaban tanto que impedían el paso de las partículas de fluido o parece que existen cuellos de botella dentro de las tuberías. Esto se debe a como Zibra calcula las mallas de los colliders y la única solución posible fue colocar los emisores mas adelantados en la tubería en lugar de justo al principio de estas, y realizar la misma acción con los voids y detectores en los finales de recorrido. En la Figura 6-1 se muestra un ejemplo de este problema con la malla de los colliders, donde se puede observar tanto el bloqueo al inicio de la tubería como un cuello de botella.

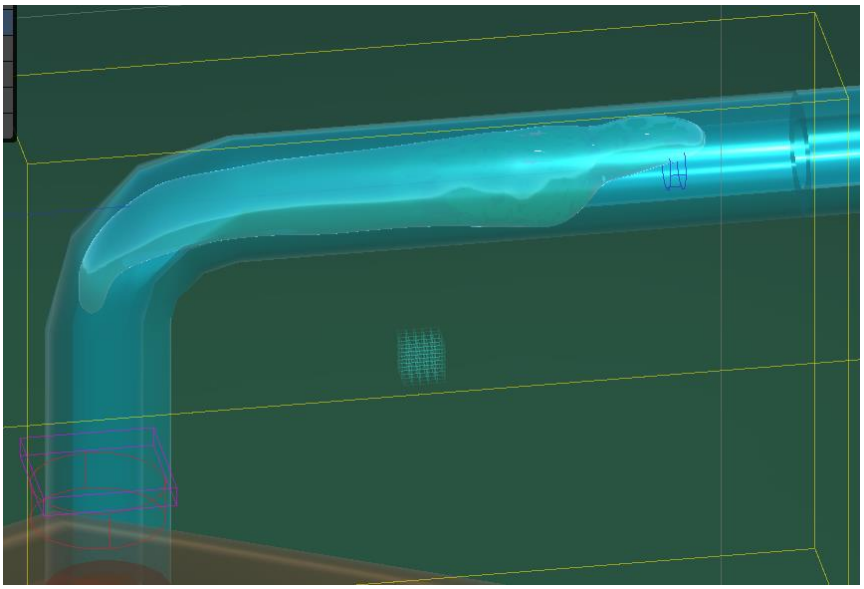


Figura 6- 1: Ejemplo de fallo en la optimización en la malla de un collider.

Pese al problema anteriormente expuesto, y considerando que en futuras versiones de Zibra se corregirá y optimizará el algoritmo de creación de colliders, el *Modo Vídeo* permite observar el comportamiento del agua en cada elemento de la planta, pudiendo servir de base para futuras simulaciones.

6.1 Trabajos futuros

Este proyecto se realizó con la idea de que sirviera de base para futuros proyectos ya sea para realizar experimentos en el propio laboratorio o para que se pudiera usar en docencia para aplicar técnicas de control. En relación con esto, resultaría interesante en el momento en el que la planta estuviera operativa poder programar un entorno de trabajo donde el gemelo digital y la planta real puedan comunicarse entre sí, de esta forma podría comprobarse el estado en el que se encuentra la planta real durante su operación por medio de su visualización en el entorno digital o una vez obtenidos los datos de la simulación comprobarlos en tiempo real en la planta, permitiendo así ajustar con más precisión el modelo a la realidad.

Una de las principales dificultades encontradas durante el desarrollo del gemelo, ha sido la simulación del comportamiento del agua. Zibra Liquid ha resultado ser una herramienta muy útil a la hora imitar el comportamiento de un fluido, aun así, tiene sus limitaciones:

- Los colliders de estructuras complejas (concretamente de las tuberías) no terminan de ajustarse del todo a las formas deseadas, produciendo bloqueos en las tuberías o esquinas donde se acumulan grandes cantidades de partículas. Estas partículas normalmente se acumulan en esos puntos sin avanzar o simplemente son eliminadas por el algoritmo de Zibra.
- La resolución afecta en gran medida a la calidad de la visualización y tiende a consumir muchos recursos del ordenador, provocando una ralentización de la simulación apreciable.

Muchos de los problemas de Zibra se han podido solucionar optimizando el tamaño y resolución de los volúmenes de trabajo, así como ajustando el número de partículas o el tiempo de interacción entre ellas, pero considerando que Zibra Liquids es una herramienta en constante evolución es muy probable que estos problemas acaben siendo solucionados en futuras actualizaciones. También los problemas de resolución podrían solucionarse con ordenadores de una mayor potencia, permitiendo así poder simular toda la planta en un único volumen de trabajo y permitiendo poder observar el comportamiento de todos los elementos de la planta a la vez y apreciar el movimiento del fluido por su interior.

conociéndose las distintas especificaciones de los componentes de la planta y tomándose los datos reales cuando esta esté en funcionamiento, se podrá ajustar con más precisión el modelo actual del depósito y se podrían añadir más sistemas:

- Conociendo el tiempo que tarda el agua en alcanzar el depósito desde las válvulas automáticas se podría experimentar con retardos y distintos sistemas de control de los compresores.
- La camisa de refrigeración podría ser implementada, permitiendo añadir la temperatura como nuevo parámetro a simular, haciendo necesario calcular nuevos modelos matemáticos.

La versatilidad del gemelo digital permitiría también probar el comportamiento de la planta con nuevos dispositivos o elementos, ya sea por sustitución de los elementos antiguos por mal funcionamiento o si se desea realizar cambios de diseño en los mismos. Solo serían necesarios pequeños ajustes en el gemelo y diseñar los nuevos elementos.

BIBLIOGRAFÍA

[1] C. Sáinz de la Flor. (2020). Manual para entender la Cuarta Revolución Industrial. Available: https://www.wearemarketing.com/es/blog/que-es-la-cuarta-revolucion-industrial.html#:~:text=El%20concepto%20de%20la%20cuarta,gesti%C3%B3n%20online%20de%20la%20producci%C3%B3n.

[2] J.L. del Val Román. "Industria 4.0: La transformación digital de la industria". In Conferencia de Directores y Decanos de Ingeniería Informática. Deusto. 2016

[3] Digital Twin Consortium. (2022, 4 mayo). The Definition of a Digital Twin - Digital Twin Consortium. https://www.digitaltwinconsortium.org/hot-topics/the-definition-of-a-digital-twin/

[4] Boschert, S., Rosen, R. (2016). Digital Twin—The Simulation Aspect. In: Hehenberger, P., Bradley, D. (eds) Mechatronic Futures. Springer, Cham. https://doi.org/10.1007/978-3-319-32156-1_5

[5] M. Grieves, "Digital twin: Manufacturing excellence through virtual factory replication," White paper, 2015.

[6] Eric J. Tuegel, Anthony R. Ingraffea, Thomas G. Eason, S. Michael Spottswood, "Reengineering Aircraft Structural Life Prediction Using a Digital Twin", *International Journal of Aerospace Engineering*, vol. 2011, Article ID 154798, 14 pages, 2011. https://doi.org/10.1155/2011/154798

[7] Edward Glaessgen and David Stargel. "The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles," AIAA 2012-1818. *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*. April 2012.

[8] F. Tao and M. Zhang, "Digital Twin Shop-Floor: A New Shop-Floor Paradigm Towards Smart Manufacturing," in *IEEE Access*, vol. 5, pp. 20418-20427, 2017, doi: 10.1109/ACCESS.2017.2756069.

[9] W. Bellamy. (2018) Boeing CEO Talks 'Digital Twin' Era of Aviation. Available: de https://www.aviationtoday.com/2018/09/14/boeing-ceo-talks-digital-twin-era-aviation/

[10] Mayani, M. Gholami, Svendsen, M.. , and S. I. Oedegaard. "Drilling Digital Twin Success Stories the Last 10 Years." Paper presented at the SPE Norway One Day Seminar, Bergen, Norway, April 2018. doi: https://doi.org/10.2118/191336-MS

[11] Neethirajan, S.; Kemp, B. Digital Twins in Livestock Farming. *Animals* **2021**, *11*, 1008. https://doi.org/10.3390/ani11041008

[12] Gary White, Anna Zink, Lara Codecá, Siobhán Clarke, A digital twin smart city for citizen feedback, Cities, Volume 110, 2021, 103064, ISSN 0264-2751, https://doi.org/10.1016/j.cities.2020.103064.

[13] T. Erol, A. F. Mendi and D. Doğan, "The Digital Twin Revolution in Healthcare," *2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Istanbul, Turkey, 2020, pp. 1-7, doi: 10.1109/ISMSIT50672.2020.9255249.

[14] Milan Vathoopan, Maria Johny, Alois Zoitl, Alois Knoll, Modular Fault Ascription and Corrective Maintenance Using a Digital Twin, IFAC-PapersOnLine, Volume 51, Issue 11, 2018, Pages 1041-1046, ISSN 2405-8963, https://doi.org/10.1016/j.ifacol.2018.08.470.

[15] Magargle, Ryan S. et al. "A Simulation-Based Digital Twin for Model-Driven Health Monitoring and Predictive Maintenance of an Automotive Braking System." *International Modelica Conference* (2017).

[16] P. Aivaliotis, K. Georgoulias, Z. Arkouli, S. Makris, Methodology for enabling Digital Twin using advanced physics-based modelling in predictive maintenance, Procedia CIRP, Volume 81, 2019, Pages 417-422, ISSN 2212-8271, https://doi.org/10.1016/j.procir.2019.03.072.

[17] S. M. Jeon and S. Schuesslbauer, "Digital Twin Application for Production Optimization," *2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Singapore, Singapore, 2020, pp. 542-545, doi: 10.1109/IEEM45057.2020.9309874.

[18] Marius Matulis, Carlo Harvey, A robot arm digital twin utilising reinforcement learning, Computers & Graphics, Volume 95, 2021, Pages 106-114, ISSN 0097-8493, https://doi.org/10.1016/j.cag.2021.01.011.

[19] *Reacción exotérmica - concepto y ejemplos*. (s. f.). Concepto. https://concepto.de/reaccion-exotermica/

[20] Jorn K. Gruber, Carlos Bordons, Control predictivo no lineal basado en modelos de volterra. aplicación a una planta piloto, Revista Iberoamericana de Automática e Informática Industrial RIAI, Volume 4, Issue 3, 2007, Pages 34-45, ISSN 1697-7912, https://doi.org/10.1016/S1697-7912(07)70223-6.

[21] González Urbano, Samara. *Modificacion, puesta en marcha y programacion de una planta para el control e instrumentacion industrial*. Dirigido por J. Manuel Escaño Gonzalez. Proyecto Fin de Carrera defendido en la Escuela Técnica Superior de Ingeniería, Universidad de Sevilla, 2009. [Consulta:04 de octubre de 2023] Disponible en: https://biblus.us.es/bibing/proyectos/abreproy/11812/

[22] Pajarón Pérez, Rafael. *Control borroso industrial: uso de la iec 1131-7 para el control de plantas industriales*. Dirigido por Juan Manuel Escaño Gonzalez. Proyecto Fin de Carrera defendido en la Escuela Técnica Superior de Ingeniería, Universidad de Sevilla, 2012. [Consulta:04 de octubre de 2023] Disponible en: https://biblus.us.es/bibing/proyectos/abreproy/12070/

[23] González, A. C. (2022). Unity 3D, que es y para qué se utiliza. *Profesional Review*. https://www.profesionalreview.com/2022/02/20/unity-3d-que-es-y-para-que-se-utiliza/

[24] *¿Cómo usar la interfaz de Unity? - Unity Learn*. (s. f.). Unity Learn. https://learn.unity.com/tutorial/como-usar-la-interfaz-de-unity?language=es#

[25] *Zibra AI - creating AI-generated assets for any virtual world or experience.* (s. f.). https://effects.zibra.ai/#team

[26] *Zibra Effects 2.0.0 User Guide.pdf*. (s. f.). Google Docs. https://drive.google.com/file/d/1ynhRtx8jb3HQ8E9_KbctzWoLd9oiICxU/view

[27] S*igned distance fields in the Visual Effect Graph | Visual Effect Graph | 12.0.0*. (s. f.). https://docs.unity3d.com/Packages/com.unity.visualeffectgraph@12.0/manual/sdf-in-vfx-graph.html

[28] *Tubos.* (s.f.) https://www.bonnet.es/Productos/Catalogos/Inoxidable/tubos.pdf

ANEXO I: SCRIPTS DE LA INTERFAZ

Script del dropdown de la interfaz

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Interfaz : MonoBehaviour
{
   public GameObject Camara;
   public Transform  PosCamara;
   public int GlobalIndex;
   public bool Deposito, Video;

   void Start()
   {
      GlobalIndex = 0;
      Deposito = false;
      Video = false;
   }

   void Update()
   {
   }

   public void InterfazFunction (int index)
   {
      GlobalIndex = index;
      switch (index)
      {
         case 0:
            Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
            Deposito = false;
```

```
            Video = false;
        break;

        case 1:
            Deposito = true;
            Video = false;
        break;

         case 2:
            Deposito = false;
            Video = true;
        break;
      }
   }
}
```

Script del texto del sensor de nivel

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using com.zibra.liquid.Manipulators;

public class Texto : MonoBehaviour
{
   public ZibraLiquidDetector SensorNivel;
   public TextMeshProUGUI Nivel;
   public Vector3 Altura;
   public float h;

   void Start()
   {
   }

   void Update()
   {
```

```
        Altura = SensorNivel.BoundingBoxMax - SensorNivel.BoundingBoxMin;
        h = Altura.y;
        Nivel.text = "Altura deposito: " + h.ToString("f3");
    }
}
```

Script inicializador de válvulas

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ValManuales : MonoBehaviour
{
    public bool V1, V2, V3, V6, V7, V9;

    void Start()
    {
    }

    void Update()
    {
    }

    public void ToggleV1()
    {
        V1=!V1;
    }
    public void ToggleV2()
    {
        V2=!V2;
    }
    public void ToggleV3()
    {
        V3=!V3;
    }
```

```
    public void ToggleV6()
    {
        V6=!V6;
    }
    public void ToggleV7()
    {
        V7=!V7;
    }
    public void ToggleV9()
    {
        V9=!V9;
    }
}
```

ANEXO II: SCRIPTS DEL MODO VÍDEO

Script de inicio del vídeo

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Video : MonoBehaviour
{
   public int Index;
   public bool Aux1, Aux2, Activar, Panel;
   public Interfaz ActVideo;
   public GameObject V1, V2, V3, V6, V7, V9;

   void Start()
   {
      Index = 0;
      Aux1 = true;
      Aux2 = true;
      Activar = false;
      Panel = true;
   }

   void Update()
   {
      if(ActVideo.Video)
      {
         Activar = true;
         DesactPanel();
      }
      else
      {
         Activar = false;
```

```
            ActPanel();
        }
    }

    void DesactPanel()
    {
        if (Panel)
        {
            V1.SetActive(false);
            V2.SetActive(false);
            V3.SetActive(false);
            V6.SetActive(false);
            V7.SetActive(false);
            V9.SetActive(false);
            Panel = false;
        }
    }

    void ActPanel()
    {
        if (!Panel)
        {
            V1.SetActive(true);
            V2.SetActive(true);
            V3.SetActive(true);
            V6.SetActive(true);
            V7.SetActive(true);
            V9.SetActive(true);
            Panel = true;
        }
    }
}
```

Script de salida del tanque de abastecimiento

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;
using com.zibra.liquid.Manipulators;

public class SalidaTanque : MonoBehaviour
{
   public Video Vid;
   public GameObject TubSalTanque, Tanque;
   public GameObject Camara;
   public Transform  PosCamara;
   public ZibraLiquidDetector Detector;
   float delay = 5;
   float timer;

   void Start()
   {
   }

   void Update()
   {
      if (Vid.Index == 0 && Vid.Activar)
      {
         Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
         Tanque.SetActive(true);

         timer += Time.deltaTime;
         if (timer>delay){
            TubSalTanque.SetActive(true);
         }

         if (Detector.ParticlesInside >= 10 && Vid.Index == 0)
         {
            Vid.Index = 1;
         }

         }
      else if (Vid.Index >= 1 && Vid.Aux2)
      {
```

```
            TubSalTanque.SetActive(false);
            Tanque.SetActive(false);
            }
        else if (!Vid.Activar)
        {
            TubSalTanque.SetActive(false);
            Tanque.SetActive(false);
            }
    }
}
```

Script de entrada del tanque de abastecimiento

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;

public class EntradaTanque : MonoBehaviour
{
    public Video Vid;
    public GameObject Tanque, TubEntTanque, Emitter;
    public GameObject Camara;
    public Transform  PosCamara;
    float delay = 10;
    float timer;
    public ZibraLiquidDetector Detector;

    void Start()
    {
        Emitter.SetActive(false);
    }

    void Update()
    {
        if (Vid.Index == 14 && Vid.Activar)
        {
            Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
```

```
            Tanque.SetActive(true);
            TubEntTanque.SetActive(true);
            if (Detector.ParticlesInside != 0)
            {
               Emitter.SetActive(true);
            }
            timer += Time.deltaTime;
            if (timer>delay)
            {
               Vid.Index = 15;
               Emitter.SetActive(false);
            }

         }
         else if (Vid.Index >= 15)
         {
            Tanque.SetActive(false);
            TubEntTanque.SetActive(false);
            Vid.Aux2 = true;
         }
         else if (!Vid.Activar)
         {
            TubEntTanque.SetActive(false);
            Tanque.SetActive(false);
            }
      }
}
```

Script del primer tramo de la tubería de agua fría

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;

public class Tramo1Fria : MonoBehaviour
```

```
{
    public Video Vid;
    public GameObject TubFria1, EmitterAux;
    public GameObject Camara;
    public Transform  PosCamara;
    public ZibraLiquidDetector Detector, DetectorAux;

    void Start()
    {
        EmitterAux.SetActive(false);
    }

    void Update()
    {
        if (Vid.Index == 1 && Vid.Activar)
        {
            Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
            TubFria1.SetActive(true);
            if (DetectorAux.ParticlesInside != 0)
            {
                EmitterAux.SetActive(true);
            }
            if (Detector.ParticlesInside != 0 && Vid.Index == 1)
            {
                Vid.Index = 2;
            }
            }
        else if (Vid.Index >= 4)
        {
            TubFria1.SetActive(false);
            }
        else if (!Vid.Activar)
        {
            TubFria1.SetActive(false);
            }
    }
}
```

Script del segundo tramo de la tubería de agua fría

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;
using UnityEngine.UI;

public class Tramo2Fria : MonoBehaviour
{
    public Video Vid;
    public ValManuales Val13;
    public GameObject TubFria2, EmitterAux;
    public GameObject Camara, V1, V3;
    public Transform  PosCamara;
    public RectTransform PosV3;
    public ZibraLiquidDetector Detector, DetectorAux;

    void Start()
    {
        EmitterAux.SetActive(false);
    }

    void Update()
    {
        if (Vid.Index == 3 && Vid.Activar)
        {
            Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
            V1.SetActive(true);
            PosV3.anchoredPosition = new Vector2(310.3f,66.5f);
            V3.SetActive(true);
            if (Val13.V1)
            {
                TubFria2.SetActive(true);
                if (DetectorAux.ParticlesInside != 0)
```

```
            {
                EmitterAux.SetActive(true);
            }
            if (Detector.ParticlesInside != 0 && Vid.Index == 3)
            {
                Vid.Index = 4;
                V1.SetActive(false);
                V3.SetActive(false);
                PosV3.anchoredPosition = new Vector2(310.3f,26.5f);
            }
        }
    }
    else if (Vid.Index >= 4)
    {
        TubFria2.SetActive(false);
        Vid.Aux1 = false;
        }
    else if (!Vid.Activar)
    {
        TubFria2.SetActive(false);
        }
  }
}
```

Script del tercer tramo de la tubería de agua fría

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;

public class Tramo3Fria : MonoBehaviour
{
  public Video Vid;
  public ValManuales Val3;
  public GameObject TubFria3;
  public GameObject Camara, V3;
```

```
    public Transform  PosCamara;
    public RectTransform PosV3;
    public ZibraLiquidDetector Detector;

    void Start()
    {
    }

    void Update()
    {
      if (Vid.Index == 10 && Vid.Activar)
      {
        Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
        PosV3.anchoredPosition = new Vector2(310.3f,106.5f);
        V3.SetActive(true);
        if (Val3.V3)
        {
          TubFria3.SetActive(true);
          if (Detector.ParticlesInside != 0 && Vid.Index == 10)
          {
            Vid.Index = 11;
            V3.SetActive(false);
            PosV3.anchoredPosition = new Vector2(310.3f,26.5f);
          }
        }
        }
      else if (Vid.Index >= 11)
      {
        TubFria3.SetActive(false);
        }
      else if (!Vid.Activar)
      {
        TubFria3.SetActive(false);
        }
    }
}
```

Script de entrada del termo

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;

public class EntradaTermo : MonoBehaviour
{
    public Video Vid;
    public GameObject Termo, TubEntTermo, Emitter;
    public ZibraLiquidDetector Detector;
    float delay = 20;
    float timer;

    void Start()
    {
        Emitter.SetActive(false);
    }

    void Update()
    {
        if (Vid.Index == 2 && Vid.Activar)
        {
            Termo.SetActive(true);
            TubEntTermo.SetActive(true);
            if (Detector.ParticlesInside != 0)
            {
                Emitter.SetActive(true);
            }
            timer += Time.deltaTime;
            if (timer>delay)
            {
                Vid.Index = 3;
                Emitter.SetActive(false);
            }
```

```
        }
        else if (Vid.Index >= 3 && Vid.Aux1)
        {
            Termo.SetActive(false);
            TubEntTermo.SetActive(false);
        }
        else if (!Vid.Activar)
        {
            TubEntTermo.SetActive(false);
            Termo.SetActive(false);
            }
    }
}
```

Script de salida del termo

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;

public class SalidaTermo : MonoBehaviour
{
    public Video Vid;
    public GameObject Termo, TubSalTermo;
    public GameObject Camara;
    public Transform  PosCamara;
    public ZibraLiquidDetector Detector;

    void Start()
    {
    }

    void Update()
    {
        if (Vid.Index == 4 && Vid.Activar)
```

```
        {
            Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
            Termo.SetActive(true);
            TubSalTermo.SetActive(true);
            if (Detector.ParticlesInside >= 10 && Vid.Index == 4)
            {
                Vid.Index = 5;
            }
        }
        else if (Vid.Index >= 6)
        {
            Termo.SetActive(false);
            TubSalTermo.SetActive(false);
            Vid.Aux1 = true;
        }
        else if (!Vid.Activar)
        {
            TubSalTermo.SetActive(false);
            Termo.SetActive(false);
            }
    }
}
```

Script del primer tramo de la tubería de agua caliente

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;

public class Tramo1Cal : MonoBehaviour
{
    public Video Vid;
    public GameObject TubCal1;
    public GameObject Camara, V2;
    public Transform  PosCamara;
    public RectTransform PosV2, PosDrop;
```

```
    public ZibraLiquidDetector Detector;

    void Start()
    {
    }

    void Update()
    {
      if (Vid.Index == 5 && Vid.Activar)
      {
        Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
        TubCal1.SetActive(true);
        PosV2.anchoredPosition = new Vector2(-324.7f,106.5f);
        PosDrop.anchoredPosition = new Vector2(-310f,204f);
        V2.SetActive(true);
        if (Detector.ParticlesInside != 0 && Vid.Index == 5)
        {
          Vid.Index = 6;
          V2.SetActive(false);
          PosDrop.anchoredPosition = new Vector2(325f,204f);
          PosV2.anchoredPosition = new Vector2(-310.3f,66.5f);
        }
      }
      else if (Vid.Index >= 7)
      {
        TubCal1.SetActive(false);
        }
      else if (!Vid.Activar)
      {
        TubCal1.SetActive(false);
        }
    }
}
```

Script del segundo tramo de la tubería de agua caliente

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;

public class Tramo2Cal : MonoBehaviour
{
   public Video Vid;
   public ValManuales Val2;
   public GameObject TubCal2;
   public GameObject Camara;
   public Transform  PosCamara;
   public ZibraLiquidDetector Detector;

   void Start()
   {
   }

   void Update()
   {
      if (Vid.Index == 6 && Val2.V2 && Vid.Activar)
      {
         Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
         TubCal2.SetActive(true);
         if (Detector.ParticlesInside != 0 && Vid.Index == 6)
         {
            Vid.Index = 7;
         }
      }
      else if (Vid.Index >= 7)
      {
         TubCal2.SetActive(false);
         }
      else if (!Vid.Activar)
      {
         TubCal2.SetActive(false);
```

```
        }
    }
}
```

Script del depósito

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DepositoVid : MonoBehaviour
{
    public Interfaz ActDeposito;
    public Video Vid;
    public GameObject DepositoInt, DepositoExt;
    public GameObject Camara;
    public Transform  PosCamara;
    public bool Camisa;
    float delay1 = 10;
    float delay2 = 10;
    float timer1, timer2;
    int c;

    void Start()
    {
        Camisa = false;
        c=0;
    }

    void Update()
    {
        if (Vid.Index == 7 && Vid.Activar)
        {
            Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
            if (c==0)
            {
```

```
            DepositoInt.SetActive(true);
            timer1 += Time.deltaTime;
            if (timer1>delay1 && Camisa == false)
            {
               Camisa = true;
               c=1;
               DepositoInt.SetActive(false);
               }
         }
         if (Camisa == true)
         {
            DepositoExt.SetActive(true);
            timer2 += Time.deltaTime;
            if (timer2>delay2)
            {
               Vid.Index = 8;
               DepositoExt.SetActive(false);
               Camisa = false;
            }
         }
         }
      else if (!Vid.Activar && !ActDeposito.Deposito)
      {
         DepositoExt.SetActive(false);
         DepositoInt.SetActive(false);
         }
   }
}
```

Script del primer tramo de la tubería de recirculación

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;
```

```
public class Tramo1Rec : MonoBehaviour
{
   public Video Vid;
   public GameObject TubRec1;
   public GameObject Camara;
   public Transform  PosCamara;
   public ZibraLiquidDetector Detector;

   void Start()
   {
   }

   void Update()
   {
      if (Vid.Index == 8 && Vid.Activar)
      {
         Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
         TubRec1.SetActive(true);
         if (Detector.ParticlesInside != 0 && Vid.Index == 8)
         {
            Vid.Index = 9;
         }
         }
      else if (Vid.Index >= 10)
      {
         TubRec1.SetActive(false);
         }
      else if (!Vid.Activar)
      {
         TubRec1.SetActive(false);
         }
   }
}
```

Script del segundo tramo de la tubería de recirculación

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;

public class Tramo2Rec : MonoBehaviour
{
   public Video Vid;
   public GameObject TubRec2;
   public GameObject Camara;
   public Transform  PosCamara;
   public ZibraLiquidDetector Detector;

   void Start()
   {
   }

   void Update()
   {
      if (Vid.Index == 9 && Vid.Activar)
      {
         Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
         TubRec2.SetActive(true);
         if (Detector.ParticlesInside != 0 && Vid.Index == 9)
         {
            Vid.Index = 10;
         }
      }
      else if (Vid.Index >= 10)
      {
         TubRec2.SetActive(false);
         }
      else if (!Vid.Activar)
      {
         TubRec2.SetActive(false);
         }
```

```
    }
}
```

Script del tercer tramo de la tubería de recirculación

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;

public class Tramo3Rec : MonoBehaviour
{
    public Video Vid;
    public ValManuales Val9;
    public GameObject TubRec3;
    public GameObject Camara, V9;
    public Transform  PosCamara;
    public RectTransform PosV9, PosDrop;
    public ZibraLiquidDetector Detector;

    void Start()
    {

    }

    void Update()
    {
        if (Vid.Index == 12  && Vid.Activar)
        {
            Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
            PosV9.anchoredPosition = new Vector2(-324.7f,106.5f);
            PosDrop.anchoredPosition = new Vector2(-310f,204f);
            V9.SetActive(true);
            if(Val9.V9)
            {
                TubRec3.SetActive(true);
```

```
            if (Detector.ParticlesInside != 0 && Vid.Index == 12)
            {
               Vid.Index = 13;
               V9.SetActive(false);
               PosDrop.anchoredPosition = new Vector2(325f,204f);
               PosV9.anchoredPosition = new Vector2(-310.3f,66.5f);
            }
         }
      }
      else if (Vid.Index >= 13)
      {
         TubRec3.SetActive(false);
         }
      else if (!Vid.Activar)
      {
         TubRec3.SetActive(false);
         }
   }
}
```

Script del intercambiador

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;

public class Intercambiador : MonoBehaviour
{
   public Video Vid;
   int c, b;
   public GameObject Inter, Tub1, Tub2;
   public GameObject Camara;
   public Transform  PosCamara;
   public ZibraLiquidDetector Detector1, Detector2, Detector3;
```

```
void Start()
{
   c = 0;
   b = 0;
}

void Update()
{
   if (Vid.Index == 11 && Vid.Activar)
   {
      Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
      if(b==0)
      {
         Inter.SetActive(true);
         b = 1;
      }
      if (Detector1.ParticlesInside != 0 && c == 0 )
      {
         Inter.SetActive(false);
         Tub1.SetActive(true);
         c = 1;
      }
      if (Detector2.ParticlesInside != 0 && c == 1)
      {
         Tub2.SetActive(true);
         c = 2;
      }
      if (Detector3.ParticlesInside != 0)
      {
         Vid.Index = 12;
         Tub1.SetActive(false);
         c = 0;
      }
   else if (Vid.Index >= 13)
   {
      Tub2.SetActive(false);
      }
```

```
        else if (!Vid.Activar)
        {
            Tub2.SetActive(false);
            Tub1.SetActive(false);
            Inter.SetActive(false);
            }
    }
}
}
```

Script de la tubería de salida

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using com.zibra.liquid.Manipulators;

public class Salida: MonoBehaviour
{
    public Video Vid;
    public ValManuales Val67;
    public GameObject TubSalida, EmitterAux, EmitterV6, EmitterV7;
    public GameObject Camara, V6, V7;
    public Transform  PosCamara;
    public RectTransform PosV6, PosV7, PosDrop;
    public ZibraLiquidDetector Detector, DetectorAux;

    void Start()
    {
        EmitterAux.SetActive(false);
    }

    void Update()
    {
        if (Vid.Index == 13 && Vid.Activar)
        {
            Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
```

```
    TubSalida.SetActive(true);
    V6.SetActive(true);
    V7.SetActive(true);
    PosDrop.anchoredPosition = new Vector2(118.3f,204f);
    PosV6.anchoredPosition = new Vector2(103.6f,106.5f);
    PosV7.anchoredPosition = new Vector2(103.6f,66.5f);
    if (!Val67.V6)
    {
        EmitterV6.SetActive(true);
    }
    else{
        EmitterV6.SetActive(false);
    }
    if (!Val67.V7)
    {
        EmitterV7.SetActive(true);
    }
    else{
        EmitterV7.SetActive(false);
    }
    if (DetectorAux.ParticlesInside != 0)
    {
        EmitterAux.SetActive(true);
    }
    if (Detector.ParticlesInside != 0 && Vid.Index == 13)
    {
        Vid.Index = 14;
        Vid.Aux2 = false;
        V6.SetActive(false);
        V7.SetActive(false);
        PosDrop.anchoredPosition = new Vector2(325f,204f);
        PosV6.anchoredPosition = new Vector2(310.3f,-3.5f);
        PosV7.anchoredPosition = new Vector2(103.6f,-31.5f);
    }
    }
else if (Vid.Index >= 14)
{
```

```
            TubSalida.SetActive(false);
            }
        else if (!Vid.Activar)
        {
            TubSalida.SetActive(false);
            }
    }
}
```

ANEXO III: SCRIPTS DE LA SIMULACIÓN DEL DEPÓSITO

Script de inicialización del depósito para su simulación

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class DepositoInterior : MonoBehaviour
{
   public Interfaz ActDeposito;
   public Video Vid;
   public GameObject DepositoInt;
   public GameObject Camara;
   public Transform  PosCamara;
   public bool Camisa, Panel, ModoVideo;
   public GameObject V1, V2, V3, V6, V7, V9, Input, SliderQc, SliderQf, QcText, QfText, ValorQc,
ValorQf, MFuncion;


   void Start(){
      Camisa = false;
      Panel = true;
   }

   void Update(){
      if (ActDeposito.Deposito){
         DesactPanel();
         Camara.transform.SetPositionAndRotation(PosCamara.position,PosCamara.rotation);
         DepositoInt.SetActive(true);
         }
      else if (ActDeposito.Deposito == false && Vid.Index != 7 && ActDeposito.Video == true){
```

```
        DepositoInt.SetActive(false);
        Input.SetActive(false);
        DesactValvulas();
        }

    else if (ActDeposito.Deposito == false){
        ActPanel();
    }
}

void DesactPanel(){
    if (Panel)
    {
        Input.SetActive(true);
        V1.SetActive(true);
        V2.SetActive(true);
        V3.SetActive(false);
        V6.SetActive(false);
        V7.SetActive(false);
        V9.SetActive(false);
        SliderQc.SetActive(true);
        SliderQf.SetActive(true);
        ValorQc.SetActive(true);
        ValorQf.SetActive(true);
        QcText.SetActive(true);
        QfText.SetActive(true);
        MFuncion.SetActive(true);
        Panel = false;
    }
}

void ActPanel(){
    if (!Panel)
    {
        Input.SetActive(false);
        V1.SetActive(true);
        V2.SetActive(true);
```

```
            V3.SetActive(true);
            V6.SetActive(true);
            V7.SetActive(true);
            V9.SetActive(true);
            SliderQc.SetActive(false);
            SliderQf.SetActive(false);
            ValorQc.SetActive(false);
            ValorQf.SetActive(false);
            QcText.SetActive(false);
            QfText.SetActive(false);
            MFuncion.SetActive(false);
            Panel = true;
        }
    }

    void DesactValvulas(){
        if (!Panel)
        {
            V1.SetActive(false);
            V2.SetActive(false);
            SliderQc.SetActive(false);
            SliderQf.SetActive(false);
            ValorQc.SetActive(false);
            ValorQf.SetActive(false);
            QcText.SetActive(false);
            QfText.SetActive(false);
            MFuncion.SetActive(false);
            Panel = true;
        }
    }
}
```

Script de control de los sliders

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
using TMPro;

public class ControlSliders : MonoBehaviour
{
    public TextMeshProUGUI FlujoQc, FlujoQf;
    public float qc, qf;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {}

    public void sliderQc(float valorQc){
        FlujoQc.text = valorQc.ToString("f5");
        qc = valorQc;
    }

    public void sliderQf(float valorQf){
        FlujoQf.text = valorQf.ToString("f5");
        qf = valorQf;
    }
}
```

Script para el cálculo y visualización de la altura dentro del depósito

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using com.zibra.liquid.Manipulators;
```

```
public class Calculo : MonoBehaviour
{
    float hmin, hmax, R, r, A, a, d1, d2, yVacioh, dyv, dyq, yVacioq;
    public float hk, h, Qf, Qc, t, Ts;
    public GameObject Vacio, Vacio2, EmitterF, EmitterC;
    public EscrituraTxt texto;
    public ControlSliders Flujos;
    public ValManuales Botones;

    // Start is called before the first frame update
    void Start()
    {
        hmax = 0.94f;
        hmin = 0f;
        R = 0.1f;
        r = 0.0285f;
        A = Mathf.PI*(R*R);
        a = Mathf.PI*(r*r);
        h = hmin;
        t = 0f;
        Ts = 0.1f;
    }

    // Update is called once per frame
    public void Inicializar()
    {
        StartCoroutine(Espera());
    }

    IEnumerator Espera(){
        while(Botones.Inicio){
            yield return new WaitForSeconds (Ts);
            if (Botones.V1){
                Qf = Flujos.qf;
            }
            else{
                Qf = 0f;
```

```
        }
        if (Botones.V2){
            Qc = Flujos.qc;
        }
        else{
            Qc = 0f;
        }
        ComprobarEmitters();
        Ecuacion();
        VisualizarAltura();
        texto.EscribirDatos();
        t=t+Ts;
        if (!Botones.Inicio){
            break;
        }
    }
}

void Ecuacion(){
    hk = h + (Ts/A)*(Qf+Qc-a*Mathf.Sqrt(2f*9.81f*h));
    if (hk < 0){
        hk = 0;
    }
    if (hk > hmax){
        hk = hmax;
    }
    h = hk;
}

void VisualizarAltura()
{
    yVacioq = Vacio2.transform.position.y;
    yVacioh = Vacio.transform.localScale.y;
    d2 = h+14.88f;
    d1 = 0.94f-h;
    if (yVacioh>d1){
        dyv=yVacioh-d1;
```

```
            Vacio.transform.localScale += new Vector3(0, -dyv, 0);
        }
        if (yVacioh<d1){
            dyv=d1-yVacioh;
            Vacio.transform.localScale += new Vector3(0, dyv, 0);
        }
        if(yVacioh==d1){
            Vacio.transform.localScale += new Vector3(0, 0, 0);
        }
        if(yVacioq > d2){
            dyq = yVacioq - d2;
            Vacio2.transform.Translate(0,-dyq,0);
        }
        if (yVacioq < d2){
            dyq = d2 - yVacioq;
            Vacio2.transform.Translate(0,dyq,0);
        }
        if (yVacioq == d2){
            Vacio2.transform.Translate(0,0,0);
        }
    }

    void ComprobarEmitters(){
        if (Qc==0f){
            EmitterC.SetActive(false);
        }
        if (Qf==0f){
            EmitterF.SetActive(false);
        }
        if (Qc!=0f){
            EmitterC.SetActive(true);
        }
        if (Qf!=0f){
            EmitterF.SetActive(true);
        }
    }
}
```

Script de escritura de datos en document txt

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.IO;
using TMPro;

public class EscrituraTxt : MonoBehaviour
{
   string ruta, datos, hora, minuto, dia, mes, year,fecha;
   public Calculo variables;
   public Texto hsensor;
   // Start is called before the first frame update
   void Start()
   {
      hora = System.DateTime.Now.Hour.ToString();
      minuto = System.DateTime.Now.Minute.ToString();
      dia = System.DateTime.Now.Day.ToString();
      mes = System.DateTime.Now.Month.ToString();
      year = System.DateTime.Now.Year.ToString();
      fecha = dia+"-"+mes+"-"+year+"-"+hora+minuto;
      ruta = Application.dataPath + "/DatosMatlab/"+fecha+".txt";
      if (!File.Exists(ruta)){
         File.WriteAllText(ruta,"");
      }
   }
   public void EscribirDatos(){
      datos = (variables.t) + " " + variables.Qf + " " + variables.Qc + " " + variables.h + " " + hsensor.h
+ "\n";
      File.AppendAllText(ruta, datos);
   }
}
```

ANEXO IV: SCRIPT DE MATLAB

```
clear;
%% Set up the Import Options and import the data
opts = delimitedTextImportOptions("NumVariables", 5);

% Specify range and delimiter
opts.DataLines = [1, Inf];
opts.Delimiter = " ";

% Specify column names and types
opts.VariableNames = ["Tiempo", "FlujoF", "FlujoC", "Altura", "Sensor"];
opts.VariableTypes = ["double", "double", "double", "double","double"];

% Specify file level properties
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";
opts.ConsecutiveDelimitersRule = "join";
opts.LeadingDelimitersRule = "ignore";

% Specify variable properties
opts = setvaropts(opts, ["Tiempo", "FlujoF", "FlujoC", "Altura", "Sensor"], "DecimalSeparator", ",");
opts = setvaropts(opts, ["Tiempo", "FlujoF", "FlujoC", "Altura", "Sensor"], "ThousandsSeparator",
".");

% Import the data
Datos = readtable("27-1-2024-1934.txt", opts);


%% Clear temporary variables
clear opts


t=Datos.Tiempo;
Qf=Datos.FlujoF;
```

```
Qc=Datos.FlujoC;
h=Datos.Altura;
hs=Datos.Sensor;

tiledlayout(2,1);

nexttile
yyaxis left
plot(t, h,'-k')
ylabel('Altura [m]')

yyaxis right
plot (t, Qc,'-r',t, Qf,'-b')
ylabel('Flujo [m^3/s]')
xlabel("Tiempo [s]")
legend("Altura","Qf","Qc")

nexttile
plot(t, h,'-k',t,hs,'-g')
ylabel('Altura [m]')
xlabel("Tiempo [s]")
legend("Altura","Sensor")
```