Trabajo de Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Implementación gemelo digital de planta de cuatro tanques en Unity 3D: simulación de fluidos en tiempo real y conexión con el sistema físico

Autor: Antonio Tallón Zurita

Tutores:

Juan Manuel Escaño González

Ramón Andrés García Rodríguez

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

Sevilla, 2023

Trabajo Fin de Grado
Ingeniería de Tecnologías Industriales

Implementación gemelo digital de planta de cuatro tanques en Unity 3D: simulación de fluidos en tiempo real y conexión con el sistema físico

Autor:
Antonio Tallón Zurita

Tutores:
Juan Manuel Escaño González
Ramón Andrés García Rodríguez

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Proyecto Fin de Grado: Implementación gemelo digital de planta de cuatro tanques en Unity 3D: simulación de fluidos en tiempo real y conexión con el sistema físico

Autor: Antonio Tallón Zurita

Tutores: Juan Manuel Escaño González,
Ramón Andrés García Rodríguez

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2013

El Secretario del Tribunal

A mis padres y familia, por darme la oportunidad de llegar hasta aquí.

A los dos Marcos, Carlos, Garik, Paco, Julio y el resto de amigos, por todos los momentos vividos en esta etapa.

A Isa, por creer en mí y darme impulso en los últimos 100 metros.

A la música, por mantenerme a flote.

Agradecimientos

Quiero dar las gracias a mi familia, por apoyarme incondicionalmente y creer en mí. A todos los amigos que han ido apareciendo durante esta etapa, por aportar tanta riqueza a mi forma de ver el mundo. A todos aquellos profesores que me inspiraron y me empujaron a llegar más lejos de lo que parecía posible. A Isa, mi mejor apoyo. A todo aquel con quien haya compartido algún tramo del camino.

Antonio Tallón Zurita

Sevilla, 2023

Resumen

El desarrollo de gemelos digitales se ha convertido en una herramienta fundamental en la ingeniería moderna para la optimización de procesos y la toma de decisiones en tiempo real. En este Trabajo de Fin de Grado se propone la implementación de un gemelo digital utilizando la simulación de fluidos en tiempo real en Unity y su conexión con una planta real basada en la 'Four Coupled Tank' propuesta por Karl Henrik Johansson para el control de sistemas multivariables acoplados. [1] El objetivo es desarrollar un modelo virtual preciso del comportamiento de los fluidos en los tanques, integrarlo en un entorno de simulación en el motor físico de Unity y establecer una conexión bidireccional en tiempo real entre la simulación y la planta real. Esta implementación permitirá simular y controlar el comportamiento de la planta en diferentes condiciones operativas, lo que facilitará el análisis y el control del sistema en un entorno virtual, antes de aplicar los resultados en la planta real. Además, el modelo puede emplearse como laboratorio virtual para realizar prácticas relacionadas con el control multivariable sin necesidad de usar el laboratorio físico. El presente trabajo combina los campos de la simulación de fluidos, la realidad virtual, el control de sistemas y la automatización, y se espera que los resultados obtenidos contribuyan a la mejora del aprendizaje del control de procesos industriales y la operación de sistemas de control en tiempo real.

Abstract

The development of digital twins has become a fundamental tool in modern engineering for process optimization and real-time decision-making. This Final Degree Project proposes the implementation of a digital twin using real-time fluid simulation in Unity and its connection with a real plant based on the 'Four Coupled Tank' proposed by Karl Henrik Johansson to study the control of coupled multivariable systems. [1] The goal is to develop an accurate virtual model of the behavior of fluids in tanks, integrate it into a simulation environment in Unity's physics engine, and establish a two-way connection in real time between the simulation and the real plant. This implementation will allow simulating and controlling the behavior of the plant in different operating conditions, which will facilitate the analysis and control of the system in a virtual environment, in order to applying the results in the real plant. In addition, the model can be used as a virtual laboratory to carry out practices related to multivariable control without the need to use the physical laboratory. The present work combines the fields of fluid simulation, virtual reality, systems control and automation, and it is expected that the results obtained contribute to the improvement of learning of industrial process control and the operation of control systems in real time.

Keywords: Digital Twin, Automation, Process Control, Computational Fluid Dynamics, Simulation, Real Time, Virtual Reality, Unity, C#, TCP, Matlab, Four Tanks Plant

Índice

Índice de Tablas

Índice de Figuras

1 INTRODUCCIÓN

"La simulación nos permite hacer descubrimientos en un mundo más controlado y menos costoso que el mundo real, y eso es lo que la hace tan poderosa."

- Barry L. Nelson-

La evolución de las tecnologías de computación ha permitido un avance sin precedentes en el campo del modelado de sistemas. En particular, está en auge el desarrollo de nuevas técnicas que permitan simular el comportamiento de sistemas complejos en tiempo real con elevada precisión. Gracias a la capacidad de procesamiento de las computadoras modernas, ahora pueden simularse sistemas que antes eran imposibles de modelar, incluyendo en su dinámica efectos más complejos que no se contemplan en el modelo analítico.

Estas técnicas de modelado están cambiando la forma en que entendemos y diseñamos los sistemas complejos y están siendo aplicadas en diferentes campos, como la ingeniería, la biología o la medicina, lo que abre mundo de posibilidades para la investigación, el desarrollo de sistemas de control, el diseño y testeo de productos y la docencia.

En concreto, la incorporación de gemelos digitales en el control de sistemas y procesos permite explorar nuevas variantes del control basado en modelo, ampliamente utilizado en la industria.

En el campo de la docencia, el concepto de laboratorio virtual permite a los estudiantes trabajar remotamente y en cualquier momento sobre el objeto de la práctica, sin los riesgos, costes ni limitaciones temporales asociadas al laboratorio tradicional. En combinación con este, se trata de una herramienta potente para el aprendizaje de materias de carácter práctico.

1.1 Gemelos Digitales

El término 'gemelo digital' hace referencia a una representación virtual de entidades y procesos del mundo real, sincronizados a una frecuencia y fidelidad específicas [2]. Para ilustrar este concepto, en la figura 1-1 se muestra el Gemelo Digital aplicado a una planta petrolífera, cuyos distintos sistemas (de posicionamiento frente al oleaje, de seguridad, de control de temperaturas, suministro eléctrico y un largo etcétera) pueden modelarse digitalmente e integrarse en una copia virtual de la misma. Esta copia digital puede emplearse para anticipar el comportamiento de la planta en diferentes escenarios de funcionamiento, como pueden ser cambios en la dirección y velocidad del oleaje o el viento. [3]

Figura 1-1. Gemelo Digital de una plataforma petrolífera offshore

En la actualidad existe una norma que regula y define las características que debe poseer un gemelo digital a nivel industrial, la ISO 23247. Según la misma:

"Un gemelo digital en la fabricación es una representación digital adaptada a un fin, de un elemento de fabricación observable con sincronización entre el elemento y su representación digital." [4]

Es importante señalar el aspecto de conexión y sincronización con el sistema real, ya que es lo que incluye una simulación en la categoría de gemelo digital. Gracias a esta conexión pueden recogerse datos (empleando técnicas de Big Data, Machine Learning etc) que permitan al modelo digital replicar el comportamiento del sistema real con precisión, incluyendo efectos difíciles de modelar analíticamente como pueden ser el desgaste o suciedad de las válvulas, la influencia del clima u otros fenómenos que afecten a su comportamiento.

Es por ello que en este TFG, una vez realizado el modelo virtual de la planta de estudio, se implementará una conexión TCP que permita lanzar experimentos simultáneos sobre ambas plantas y recoger los datos obtenidos para reconciliar las respuestas.

Previamente, tras implementar modelo en el motor físico y una vez definidos sus parámetros, realizaremos un ajuste preliminar de la dinámica mediante técnicas experimentales, dejando como posibilidad de trabajo futuro emplear técnicas más avanzadas para refinar el comportamiento del modelo en base a las desviaciones que arrojen los datos de respuesta obtenidos a través la conexión TCP.

1.2 Realidad Virtual

Partiendo de la definición dada por Diego Levi [5] entendemos la Realidad Virtual como:

''Una base de datos interactivos capaz de crear una simulación que implique a todos los sentidos, generada por un ordenador, explorable, visualizable y manipulable en "tiempo real" bajo la forma de imágenes y sonidos digitales, dando la sensación de presencia en el entorno informático.''

Figura 1-2. Ilustración del concepto de Realidad Virtual

El entorno virtual que pretendemos construir en este TFG tiene como principal objetivo representar la experiencia de manipulación de la planta real de la forma más realista posible, así como replicar la dinámica del sistema físico de forma que puedan diseñarse controladores intercambiables entre la planta real y la simulada. Para ello nos familiarizaremos con el entorno de modelado físico Unity 3D, y exploraremos las alternativas que ofrece para incorporar fluidos a las simulaciones en tiempo real.

Asimismo, para realizar el modelo de la planta emplearemos herramientas de modelado tridimensional, en particular, se realizarán las piezas necesarias en Solid Edge para tener un buen control dimensional que nos permita aproximar correctamente la geometría de la planta, dada la influencia que esta tiene sobre su comportamiento dinámico.

Por último, abordaremos la conexión de la planta simulada con la real a través del protocolo TCP, con el fin de ensayar controladores sobre la planta y su gemelo digital.

1.3 Laboratorios Virtuales y Laboratorios Remotos

1.3.1 Laboratorios virtuales

La docencia mediante prácticas de laboratorio es imprescindible para el aprendizaje en materias con un alto contenido práctico como la medicina o la ingeniería, ya que sirven como complemento y demostración de los conocimientos teóricos. Además, permiten al alumno ponerse en contacto con las dificultades que se presentan al trabajar con un sistema real frente a uno teórico, permitiendo que desarrolle habilidades de resolución de problemas imposibles de transmitir mediante la docencia 'sobre el papel'.

No obstante, no siempre es posible acceder a laboratorios, o la experiencia en ellos se realiza de forma muy limitada y controlada, debido a las razones obvias de coste, seguridad o incompatibilidad horaria con otras actividades docentes.

Es por ello que resulta interesante introducir en la docencia el concepto de 'Laboratorio Virtual' como complemento al laboratorio tradicional. En un laboratorio virtual no existen limitaciones de tiempo, por lo que el alumno puede realizar libremente todas las experiencias que necesite para asimilar los conocimientos de la asignatura sin incurrir en costes ni dañar equipos reales.

En un campo como la ingeniería de control y la robótica, donde los equipos suelen tener un coste y nivel de sofisticación elevados, y la curva de aprendizaje para su manejo suele ser dura, esto supone una gran ventaja ya que permite comprobar y verificar los resultados de la programación sobre una planta simulada antes de aplicarlos al sistema real.

Otro aspecto atractivo es que dichas simulaciones pueden tomar mucho menos tiempo en ejecutarse que un ensayo sobre un sistema real, lo que permite comprobar los efectos de variar un determinado parámetro de forma inmediata, dando pie a realizar un mayor número de pruebas y variantes de la experiencia, reforzando y ampliando el conocimiento adquirido.

A modo de ejemplo, se listan algunos de Laboratorios Virtuales citados en el artículo 'Los Laboratorios Virtuales y Laboratorios Remotos en la enseñanza de la Ingeniería', contenido en el volumen 4 de la Revista Internacional de Educación en Ingeniería [6] :

- PID Controller Laboratory, http://www.pidlab.com con Appplets de LV, para el diseño, sintonía y análisis de sistemas con controladores Porporcional-Integral-Derivativo (PID)

- Web-Based Control System Design and Analysis http://www.softintegration.com/webservices/control/ que permite en análisis y diseño de sistemas de control, de forma interactiva y en el dominio del tiempo o de la frecuencia

- Departamento de Ingeniería de Sistemas y Automática, de la Facultad de Ciencias de la Universidad de Valladolid http://www.isa.cie.uva.es/~jesusm/investiga/labo_virtuales.html, con una serie de aplicaciones en línea de LV

- Control Web, Universidad de la Laguna, España, http://controlweb.isaatc.ull.es/web_eng/eng_index.htm, enfocado al análisis y diseño de sistemas de control que incluye tanto controladores convencionales como control PID.

- El sitio de Easy Java http://www.um.es/fem/EjsWiki/?userlang=es, que ofrece un LV que es un sistema de autoría escrito en Java para desarrollar simulación vía WEB.

• CHERIC Process Control Aplet Series http://www.cheric.org/education/control, con experimentos con applets sobre aspectos básicos de control de procesos (introducción, bucle cerrado, polos y ceros, PID, etc.

1.3.2 Laboratorios Remotos

Derivado del concepto de laboratorio virtual, aparece el de laboratorio remoto, que supone un nexo entre el laboratorio tradicional y el digital.

Un laboratorio remoto es una herramienta tecnológica, compuesta por software y hardware, que permite a docentes y estudiantes realizar, a través de internet, sus prácticas como si estuvieran en un laboratorio tradicional. Los usuarios utilizan y controlan los recursos disponibles en un laboratorio mediante el uso de sensores e instrumentación que permiten interactuar con equipos reales en vez de utilizar simulaciones y sin requerir la presencia física en el laboratorio. [7]

A través de la conexión TCP que implementaremos para conectar la planta con su gemelo digital, sería posible realizar experimentos sobre la planta real, lo que abre la posibilidad de usarla como parte de un laboratorio remoto.

El modelo virtual que se desarrolla en este trabajo podría emplearse como visualización de lo que está ocurriendo físicamente en el laboratorio. Para ello se requeriría un alto grado de realismo del modelo, implementando funciones de control adaptativo por las que el sistema virtual realizase el ajuste de sus parámetros en tiempo real para acomodar el comportamiento de la planta física en cada momento.

Para ello, tendrían que cumplirse una serie de requisitos:

- La red del laboratorio debería configurarse, abriendo los puertos para permitir el acceso al equipo que controla la planta. Esta cuestión no es trivial debido a los sistemas de seguridad de la Universidad de Sevilla. En su defecto, podría emplearse la red LAN para lanzar ensayos simultáneamente en la planta real y el gemelo digital.
- Las válvulas físicas deberían encontrarse en la posición adecuada para el tipo de acople sobre el que queramos trabajar, ya que no es posible manipularlas remotamente.
- La planta debería estar calibrada previamente
- El equipo físico de la planta debería someterse a revisión para asegurar su correcto funcionamiento, llenando los depósitos de agua, reparando averías en los sensores etc

2 UNITY 3D

"Unity es la plataforma de desarrollo más poderosa del mundo para la creación de experiencias interactivas y en tiempo real en 2D, 3D, realidad virtual y aumentada." - John Riccitiello, CEO de Unity Technologies.

2.1 El entorno de Unity

Unity3D es un entorno de desarrollo de videojuegos y aplicaciones en 2D/3D, que permite la creación de experiencias interactivas para una amplia variedad de plataformas, como dispositivos móviles, consolas de videojuegos, computadoras, entre otros.

Este entorno (ver figura 2-1) proporciona herramientas para la creación de gráficos, sonidos, animaciones, efectos visuales, inteligencia artificial y física, así como para el desarrollo de scripts en C# y otros lenguajes de programación.

Además, Unity3D cuenta con una amplia comunidad de desarrolladores y una tienda de activos (Asset Store) con una gran variedad de recursos para facilitar el desarrollo de proyectos, como paquetes de texturas, modelos 3D, scripts, plugins, entre otros.

Uno de los aspectos más destacados de Unity3D es su motor de renderizado en tiempo real, que permite la visualización de gráficos en alta calidad y la optimización del rendimiento en diferentes plataformas. Además, este entorno es compatible con una gran cantidad de herramientas y tecnologías, como la realidad virtual (VR) y aumentada (AR), lo que permite la creación de experiencias inmersivas para los usuarios.

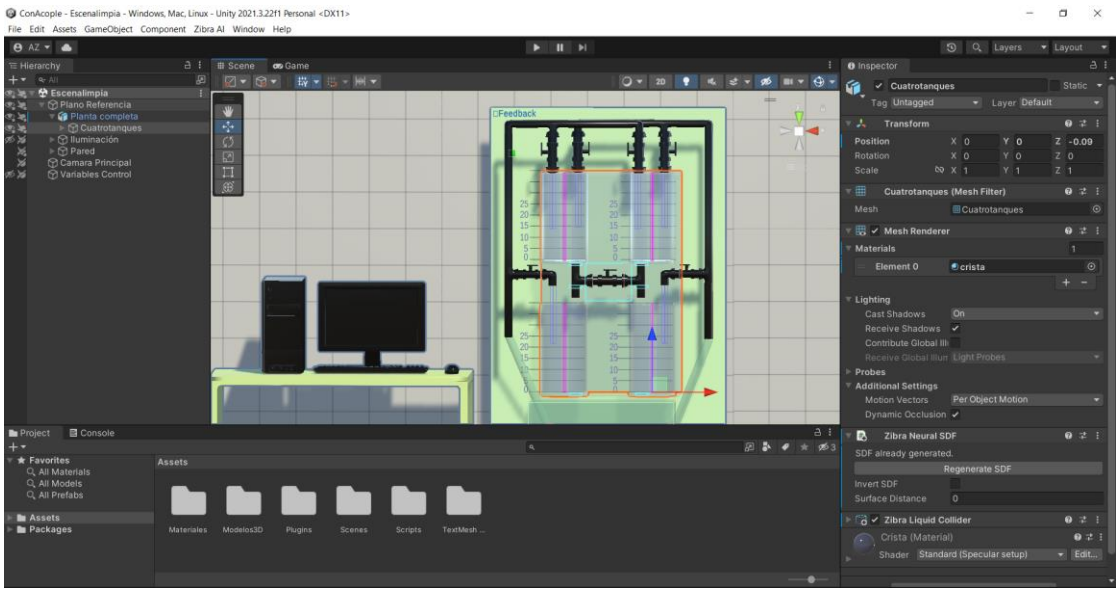


Figura 2-1. Interfaz de Unity 3D

2.1.1 Descripción de la interfaz de trabajo

La interfaz de Unity se compone de varios paneles que permiten al usuario realizar las diferentes tareas, como la creación y gestión de objetos, la importación y gestión de activos, la programación y la edición de escenas.

A continuación, se describe cada uno de los paneles de la misma:

- **Escena:** El panel de escena (Figura 2-2) muestra la vista de la escena actual del juego, permitiendo al usuario editar la posición, rotación y escala de los objetos en la escena. También permite la creación y edición de luces y cámaras. Puede alternar entre la vista en perspectiva o Isométrica, característica que nos será muy útil en el desarrollo de la planta.

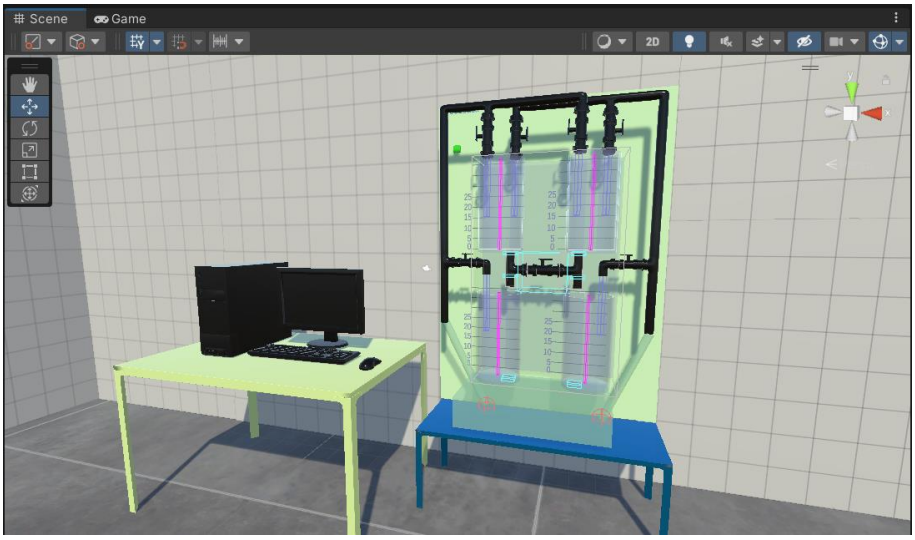


Figura 2-2. Panel de escena

- **Hierarchy:** El panel de jerarquía (Figura 2-3) muestra una lista de todos los objetos que existen en la escena actual, organizados en una estructura jerárquica. Este panel es útil para seleccionar y manipular objetos en la escena, así como establecer las relaciones Padre-Hijo entre ellos, de forma que la escala y posición ligados al objeto 'Transform' del hijo se definan en referencia a los del padre.

Figura 2-3. Panel de Jerarquía del Proyecto

- **Inspector:** El panel de inspector (Figura 2-4) muestra las propiedades y componentes de un objeto seleccionado en la escena o en el panel de jerarquía. Permite la edición de las propiedades de los objetos y la adición o eliminación de componentes, entre los que se encuentran los componentes fundamentales Transform, Mesh, Material, Lighting.. que gobiernan la apariencia y renderizado del objeto, y los asociados a la física del mismo: Mesh Collider, PhysicObject. En particular, aquí aparecerán los nuevos componentes, externos a Unity, que introduce el plugin ZibraLiquids.

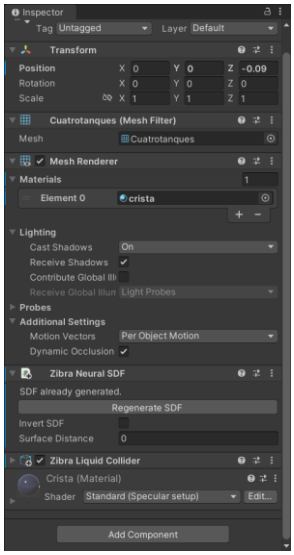


Figura 2-4. Panel de Inspector del objeto asociado a los tanques

En el inspector del objeto 'Cuatrotanques' pueden observarse los componentes asociados al Neural SFD y Zibra Liquid Collider necesarios para simular la interacción del fluido con la superficie del objeto.

- **Proyecto:** El panel de proyecto (Figura 2-5) muestra todos los archivos y carpetas que se han importado al proyecto de Unity. Permite la navegación y búsqueda de los activos, así como la creación y eliminación de carpetas y archivos.

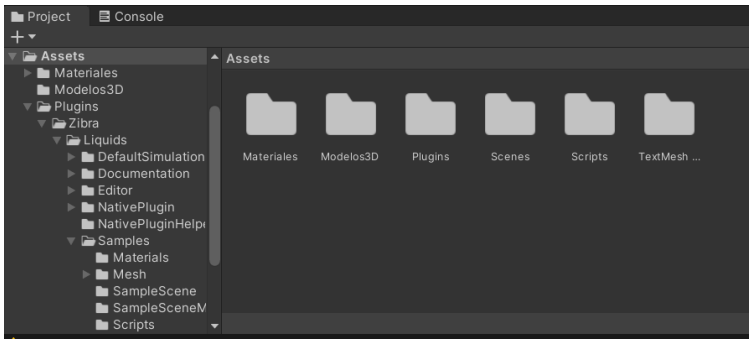


Figura 2-5. Panel de Proyecto

- **Consola:** La consola de Unity (Figura 2-6) muestra información y mensajes de depuración de la aplicación. Esto es útil para encontrar errores y problemas en el proyecto.

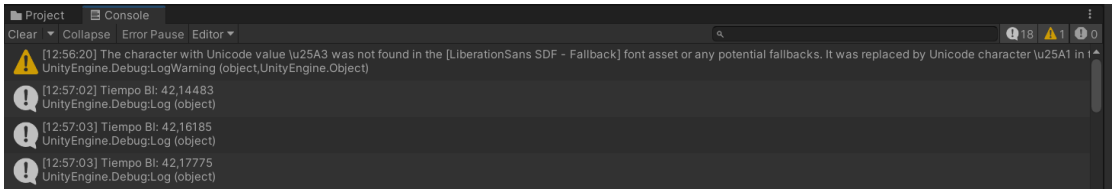


Figura 2-6. Panel de la Consola

2.2 Tratamiento de fluidos en Unity

2.2.1 Estado del arte y alternativas

En el desarrollo de aplicaciones con Unity3D, el tratamiento de fluidos se lleva a cabo a través del motor físico de sistemas de partículas incluido en el software. Para ello, se emplean scripts que codifican el comportamiento global del sistema de acuerdo con los parámetros del líquido y su interacción física con objetos, representados por colliders. Estos scripts se integran en plugins completos y funcionales, pensados para integrarse en la interfaz de Unity y desarrollados por programadores independientes.

En la Assets Store [8], se encuentran disponibles varios de estos plugins para el modelado de fluidos, cada uno con diferentes características y niveles de realismo en su implementación. Entre los más destacados se encuentran ObiFluids y ZibraLiquids:

ObiFluids

ObiFluids es un plugin para Unity3D desarrollado por VirtualMethodStudio, empresa con sede en Madrid que también cuenta en su catálogo con plugins para el tratamiento de SoftBodies, cuerdas y tejidos en el motor físico de Unity.

En concreto, permite el modelado de fluidos en 3D/2D, y ofrece características avanzadas para la manipulación de los parámetros de la simulación, como la viscosidad, dureza, tensión superficial, máxima velocidad de partícula, flotabilidad, densidad de partículas y generación de espuma.

http://obi.virtualmethodstudio.com/ [9]

ZibraLiquids

Por otro lado, ZibraLiquids es un plugin centrado exclusivamente en la simulación en 3D. A diferencia de Obi, ofrece una prueba gratuita de 14 días que nos hizo decantarnos por él para hacer pruebas con la simulación de fluidos. Presenta características similares a Obi en cuanto a los parámetros de fluido que podemos manipular.

Entre sus funciones de pago, se encuentra la generación de Neural SFD, que emplea algoritmos de inteligencia artificial para generar las superficies de contacto del fluido con objetos de geometría compleja. Esta característica a priori resulta atractiva para la aplicación que pensamos darle, ya que el modelado de la planta implica simular la interacción con tanques y tuberías cuya geometría puede ser compleja de modelar mediante objetos geométricos básicos. En capítulos posteriores se comprobará que esta característica no funciona tan bien como aparenta.

https://zibra.ai/zibra-liquids/ [10]

2.2.2 Introducción al uso de Zibra Liquids

ZibraLiquids ofrece gran cantidad de características para desarrollar simulaciones de fluidos (distintas especies de líquido, campos de fuerza, opciones de renderizado avanzadas), por lo que hacer una descripción exhaustiva de todas sus opciones resulta excesivo para el objeto de este trabajo. Sin embargo, haremos una breve introducción de las características que se han empleado para el modelo virtual de la planta, a fin de clarificar el funcionamiento del mismo.

Para ampliar esta información:

https://zibra.notion.site/Zibra-Liquids-Unity-documentation-archive-f945f86fb7494e3c8b6f348622ffcb0c [11]

2.2.2.1 El objeto ZibraLiquid

El plugin Zibra (figura 2-7), una vez instalado y activada su licencia, añade varias clases de objeto a los menús de creación de la jerarquía. En concreto, el tipo 'ZibraLiquid' es el objeto fundamental para la simulación de fluidos, ya que define la región en la que queda confinada, y en sus componentes pueden modificarse los parámetros que definen su comportamiento.

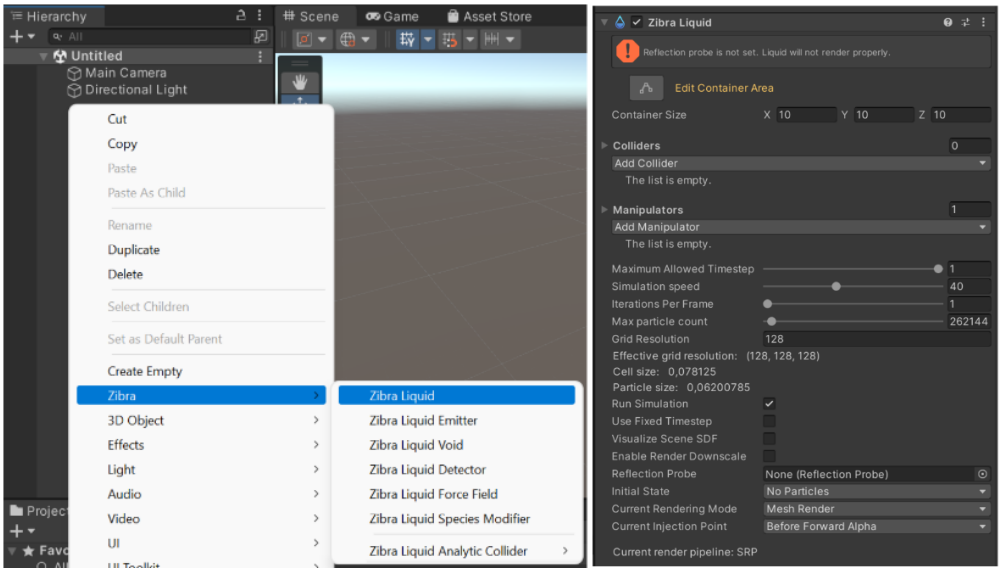


Figura 2-7. Tipos de objeto de Zibra y parámetros de la simulación

Otro componente fundamental del objeto ZibraLiquid es el llamado 'Zibra Liquid Solver Parameters', ya que en él se definen las características del líquido a simular. Modificando estos valores pueden obtenerse gran variedad de comportamientos del líquido en función de las necesidades de nuestra simulación. Por defecto, los parámetros se corresponden con los del agua en la superficie terrestre. Se muestra en la figura 2-8:

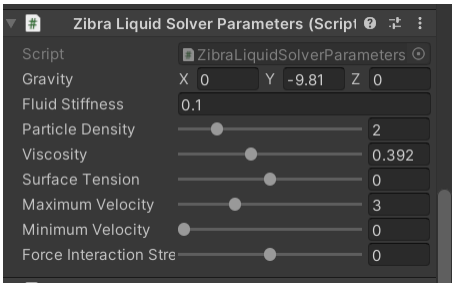


Figura 2-8. Parámetros del Solver

2.2.2.2 Manipuladores del Líquido

Al objeto ZibraLiquid se añaden instancias de manipuladores de diferente tipo, que constituyen el resto de los elementos del menú de Zibra. En concreto, en el modelo de la planta hemos usado:

2.2.2.2.1 Emitters

Este manipulador básico es la fuente de la que mana el fluido para la simulación. Lo emplearemos en cada uno de los grifos de la planta, controlando el caudal emitido mediante un script. Puede modificarse su forma, velocidad inicial, y volumen generado por unidad de tiempo. Este parámetro será el que modificaremos para variar el caudal. Ver figura 2-9.

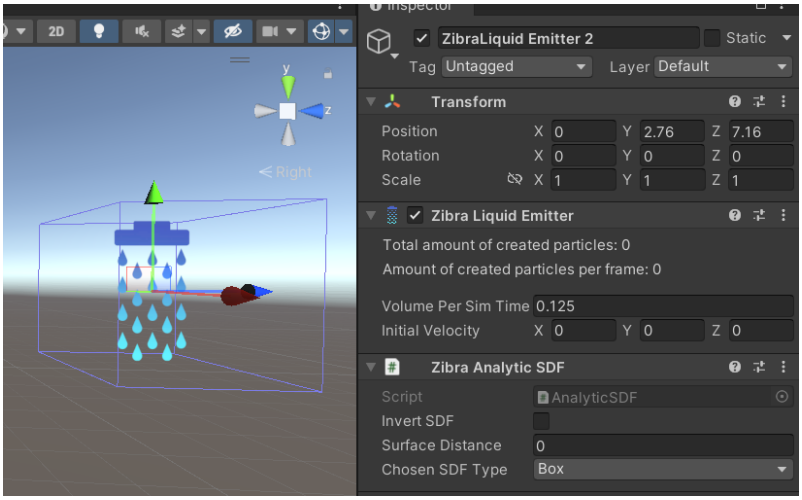


Figura 2-9. Emitter y sus componentes

2.2.2.2.2 Liquid Voids

Otro manipulador básico es el Void, que permite eliminar las partículas de la simulación al hacer contacto con la superficie de su SDF. En el modelo de la planta se emplea para limitar el número de partículas y mantener la simulación estable, ya que un número excesivo provoca la ralentización debido al elevado coste computacional. Se pondrá un Void por cada tubo de rebose, así como dos en la bandeja inferior para simular la succión de las bombas. Ver figura 2-10.

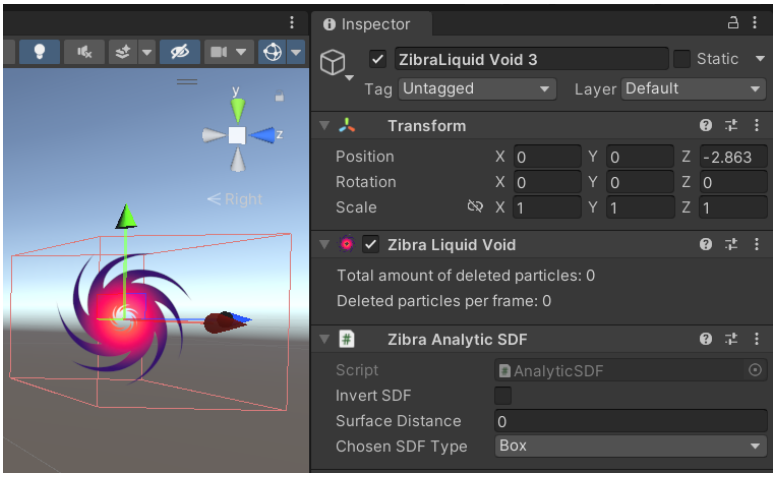


Figura 2-10. Liquid Void y sus componentes

2.2.2.2.3 Liquid Detector

Este manipulador es el que nos permitirá implementar los sensores de altura de los tanques. Lleva un registro del número de partículas contenidas en el interior de su SFD (Detector.ParticlesInside), además de registrar las coordenadas globales de las posiciones más extremas del SFD donde se detectó liquido (Detector.BoundingBoxMin/Max).

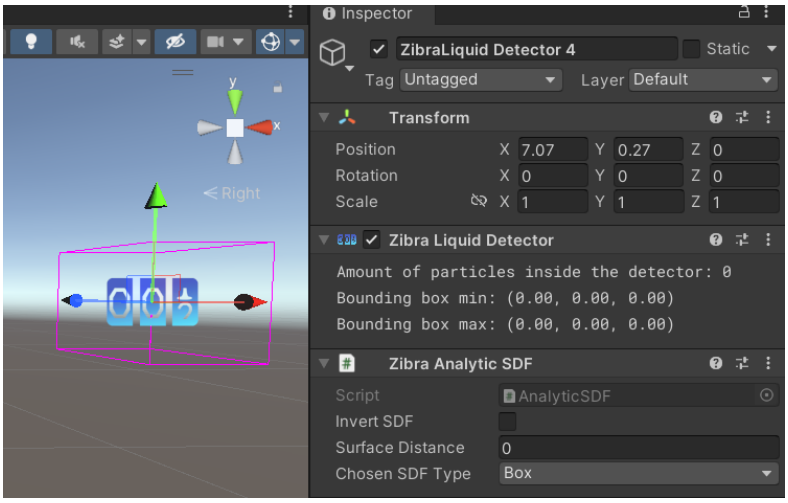


Figura 2-11. Liquid Detector y sus componentes

2.2.2.2.4 Liquid Colliders

Los Zibra Liquid Colliders representan la superficie con la que interacciona el líquido simulado. Son análogos a los colliders asociados a los RigidBody nativos de Unity para las colisiones de objetos rígidos.Se distinguen dos tipos:

2.2.2.2.4.1 Analytical Colliders

Son objetos de geometría básica, con formas predefinidas como esferas, cilindros, cubos, o cápsulas elípticas, pensados para modelar la interacción de forma simplificada con objetos de geometría conocida sin incurrir en demasiado coste computacional. Ver figura 2-12.

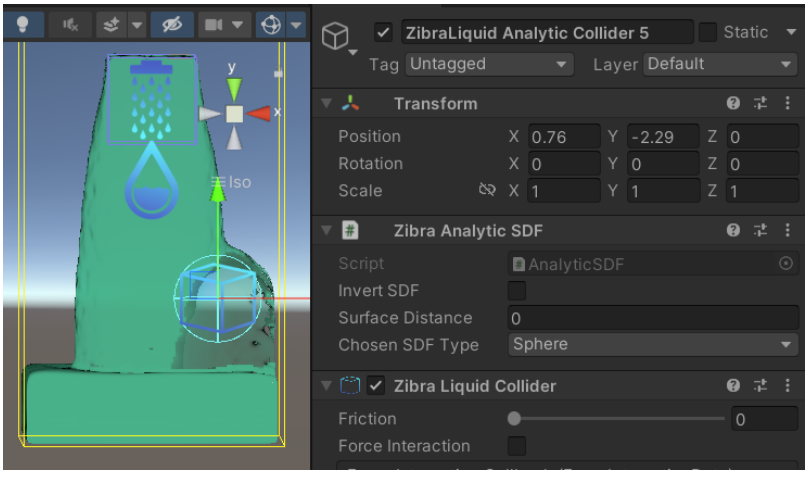


Figura 2-12. Collider Analítico Esférico

Emplearemos este tipo de collider para modelar las válvulas de descarga, colocándolas en superposición al collider asociado al conducto.

2.2.2.2.4.2 *Neural Colliders*

Este tipo de collider avanzado permite modelar la interacción del fluido con geometrías más complejas.

Para ello, hay que añadir un componente de tipo Liquid Collider al objeto cuya superficie queramos modelar, y marcar la opción Neural Collider.

Una vez hecho esto, aparecerá una nueva componente Neural SFD, en la que al seleccionar 'Generate Neural SFD' emplea un algoritmo basado en redes neuronales, que se ejecuta de forma remota en los servidores de Zibra para reconocer el mesh asociado al objeto y optimizar la forma en la que el fluido interaccionará con él. Una vez generado, esta información queda recogida en la componente de tipo Neural SFD.

Ejecutando la simulación y marcando la opción 'Visualize All Scene SFD' en los parámetros del objeto ZibraLiquid, podemos observar la superficie generada en la figura 2-13:

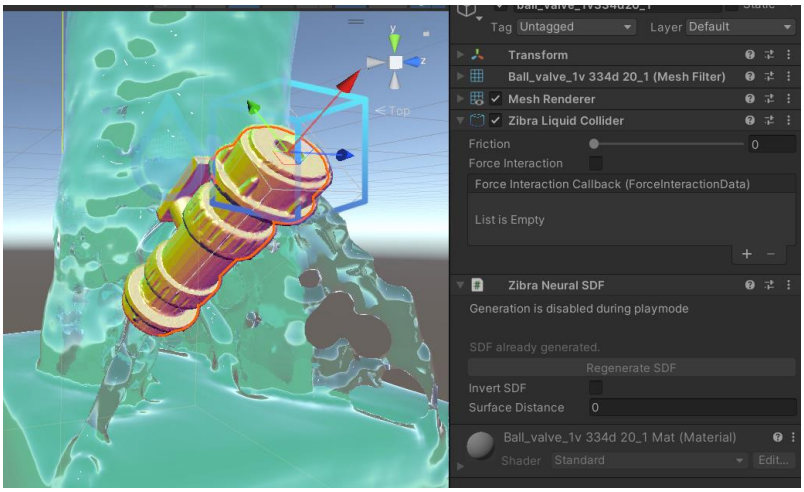


Figura 2-13. Neural Collider de una válvula

2.3 Programación en Unity

En Unity, el comportamiento de los distintos objetos de la escena se implementa por medio de Scripts codificados en C#.

Muchas de las características de este lenguaje son similares a las de C que se estudia en la asignatura de Informática de 1º de GITI, pero al tratarse de un lenguaje orientado a objetos, presenta algunas particularidades con las que hemos tenido que familiarizarnos para implementar el comportamiento de la planta virtual. Para ello, han sido de ayuda consultar algunos libros de referencia [12] y recursos en línea como esta serie de vídeos [13] dedicada a la programación en Unity.

Se incluye un subcapítulo introductorio a este paradigma de programación, y otro en el que se exponen las particularidades de emplearla a la hora de realizar los códigos en C#.

2.3.1 Programación orientada a objetos y C#

La programación orientada a objetos (POO) es un paradigma de programación que se basa en el concepto de "objetos", entidades que combinan datos (atributos) y funciones relacionadas (métodos) en una sola unidad. La idea central de la POO es modelar el mundo real mediante la creación de objetos que interactúan entre sí.

En la POO, un objeto es una instancia de una clase. Una clase es una plantilla que define las caraterísticas comunes que tendrán todos los objetos creados (instanciados) a partir de ella. Estas características pueden ser:

- Atributos: representan los datos que pertenecen al objeto, por ejemplo, la propiedad 'BoundingBox' de la clase de objeto ZibraDetector.
- Métodos: representan los comportamientos del objeto, son funciones que pueden realizar acciones y manipular los datos del objeto. Por ejemplo, el método 'RecibeDatos', de la clase de objeto ServerTCP.

Existen cuatro conceptos fundamentales en la POO, que definen la manera en la que se programa:

- Encapsulación: Es el proceso de ocultar los detalles internos (metodos y atributos) de un objeto. Permite proteger los datos, ya que solo se pueden acceder a ellos a través de los métodos definidos en el objeto.
- Herencia: Es el mecanismo que permite crear nuevas clases a partir de clases existentes. La herencia permite la reutilización de código y establece una relación de jerarquía entre las clases. Una clase heredera (subclase) hereda atributos y métodos de la clase base (superclase) y puede agregar nuevos atributos y métodos, o modificar los existentes.
- Polimorfismo: Capacidad de los objetos de diferentes clases de responder a un mismo mensaje o llamada a un mismo método de manera diferente. Permite tratar diferentes objetos de manera uniforme.
- Abstracción: Es el proceso de identificar las características esenciales de un objeto y representarlas de manera simplificada en una clase. La abstracción permite centrarse en los aspectos relevantes de un objeto y ocultar los detalles innecesarios.

2.3.2 Aplicación de la POO en los Scripts:

En C#, los objetos se representan de una forma parecida a las estructuras de C, de forma que sus métodos y atributos están contenidos en campos de la estructura asociada. Por tanto, para acceder a ellos, se emplean llamadas del tipo: objeto.atributo=…

Los scripts de Unity se asocian a un 'GameObject' siguiendo el mismo esquema de objeto/método, de manera que para asignar un determinado comportamiento codificado en un script a un objeto de la simulación, hay que incluirlo como Component del mismo en la pantalla Hierarchy.

Para que los scripts de Actuadores y LeeSensores puedan compartir datos, se declaran las variables de medidas (AI,BD..) y voltajes (VI,VD) como globales, mientras que otras variables internas, como las asociadas al cálculo de los caudales, se declaran como privadas, de forma que cumplen la encapsulación de datos.

3 PLANTA DE ESTUDIO Y MODELADO

3.1 Descripción de la planta real

La planta que vamos a estudiar se trata de la conocida como 'Cuatro Tanques', mostrada en la figura 3-1, un modelo de planta didáctica fabricada por Feedback que tiene por objetivo el estudio de diferentes estrategias de control sobre la dinámica de un sistema con acoplamiento. Está basada en la propuesta por Karl Henrik Johansson en el año 2000 como 'The quadruple-tank process' [1]. En la figura 3-1 se muestra el esquema de la planta original propuesta por dicho autor.

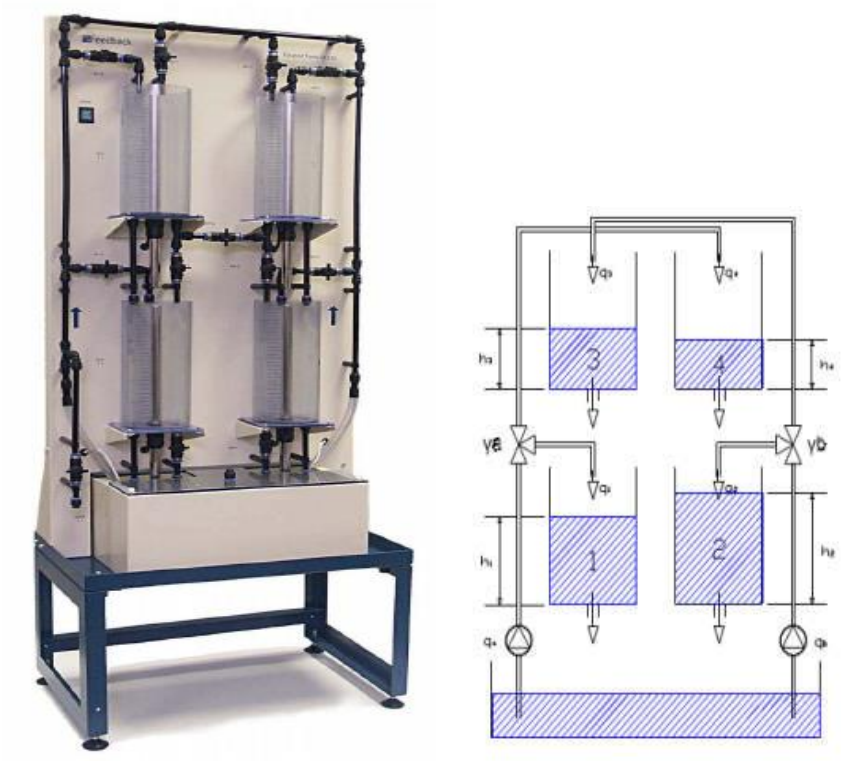


Figura 3-1. Planta Real y esquema de la planta de Johnson

3.1.1 Elementos de la planta

La planta consta de los siguientes elementos, que nombraremos según su situación (I/D para izquierda/derecha, A/B para Alto/Bajo):

- Cuatro tanques de 13.5 cm de diámetro y 25 cm de altura máxima, cada uno de ellos cuenta con un tubo por el que rebosa el agua si se supera la altura máxima, en torno a 27 cm.
- Un sensor de altura de la columna de líquido en cada tanque, que realiza la medida mediante un transductor de presión.

- Conductos de descarga: Cada tanque cuenta con dos perforaciones en la base para su descarga. Una de ellas puede abrirse y cerrarse mediante una llave de paso, mientras que la otra sólo puede taparse o destaparse con un tapón o un reductor de flujo. En la configuración en la que vamos a modelar la planta, se anulan los conductos sin válvula y se ponen los reductores de caudal en los regulables. Con ello se consigue una dinámica de descarga más lenta y uniforme de los tanques.

- Dos bombas eléctricas, cuya presión se regula variando el voltaje de entrada entre 0 y 5 V. Cada una de ellas está conectada mediante un montaje de tuberías a 3 grifos con sus respectivas llaves de apertura y cierre, las cuales denominaremos I1,I2,I3 para la rama izquierda y D1, D2, D3 para la derecha.
- Una válvula central que permite conectar las descargas de los tanques superiores entre sí, haciendo posibles configuraciones con descarga cruzada de los tanques, en las que, por ejemplo, el tanque IA descarga sobre el DB, afectando al control de la altura de este como una perturbación.

3.1.2 Descripción de la dinámica

Al estar pensada la planta original como un laboratorio para ensayar distintas estrategias de control, su dinámica varía en función de las multiples configuraciones de las llaves de descarga, lo que abre una casuística de posibles sistemas a los que aplicar control.

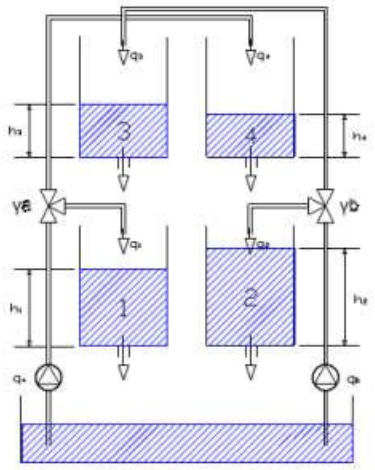
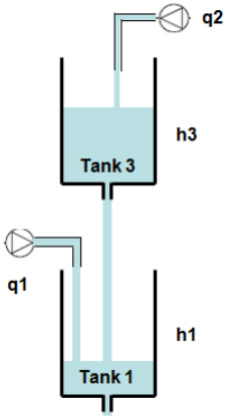


Figura 3-2. Esquema de la planta de Johansson

El objetivo de este trabajo no pasa por describir mediante ecuaciones el proceso de llenado/descarga de varios tanques en cascada, puesto que será la propia simulación virtual del fluido, si está bien implementada, la que exhibirá este comportamiento. No obstante, se incluye una breve descripción matemática de un caso sencillo (dos tanques acoplados verticalmente, figura 3-3) para ilustrar mejor la naturaleza del sistema.

3.1.2.1 Descripción analítica

La descarga de dos tanques acoplados en serie puede modelarse utilizando un conjunto de ecuaciones diferenciales no lineales [14]:



$$\frac{dh_3(t)}{dt} = -\frac{a_3}{A_3}\sqrt{2gh_3(t)} + \frac{q_2(t)}{A_3}$$
$$\frac{dh_1(t)}{dt} = -\frac{a_1}{A_1}\sqrt{2gh_1(t)} + \frac{a_3}{A_1}\sqrt{2gh_3(t)} + \frac{q_1(t)}{A_1}$$

Figura 3-3. Modelo analítico de la descarga de dos tanques en cascada

Donde h1(t), h3(t) son las alturas de los tanques 1 y 3; q1(t), q2(t) son los caudales de las bombas de entrada; A1, A3 son el área de los tanques; a1, a3 el área de la sección de descarga de cada tanque, y g es la gravedad.

Linealizando las ecuaciones anteriores en torno a un punto de funcionamiento:

$$\frac{d\bar{h}_3(t)}{dt} = -\frac{a_3}{A_3}\sqrt{\frac{g}{2h_3^o}}\bar{h}_3(t) + \frac{\bar{q}_2(t)}{A_3}$$

$$\frac{d\bar{h}_1(t)}{dt} = -\frac{a_1}{A_1}\sqrt{\frac{g}{2h_1^o}}\bar{h}_1(t) + \frac{a_3}{A_1}\sqrt{\frac{g}{2h_3^o}}\bar{h}_3(t) + \frac{\bar{q}_1(t)}{A_1}$$

Y tomando el cambio de variables siguiente para poner la ganancia estática y la constante de tiempo en función de la geometría de cada tanque:

$$T_i = \frac{A_i}{a_i}\sqrt{\frac{2h_i^o}{g}}\ , c_i = \frac{T_i}{A_i}, \quad i = 1,3$$

Podemos identificar un sistema lineal de primer orden para el tanque superior, cuya función de transferencia viene dada por:

$$\frac{\bar{H}_3(s)}{\bar{Q}_2(s)} = \frac{c_3}{T_3 s + 1}$$

Donde la ganancia estática (c3), y la constante de tiempo (T3) están determinadas según el cambio de variable por la geometría del propio tanque.

Asimismo, el tanque inferior presenta una dinámica con dependencia múltiple, considerando los caudales q1 (inferior) y q2 (superior) como entradas, con función de transferencia:

$$\bar{H}_1(s) = \frac{c_1}{T_1 s + 1}\bar{Q}_1(s) + \frac{c_1}{(T_1 s + 1)(T_3 s + 1)}\bar{Q}_2(s)$$

- El primer término representa la dependencia con el caudal de entrada q1, que es de tipo primer orden y análoga al tanque superior con q2.
- El segundo término recoge la dependencia de la altura inferior con el caudal superior q2, que viene dada por una dinámica de segundo orden en función de las constantes de tiempo de ambos tanques.

Del estudio analítico de la dinámica podemos concluir que nuestro sistema alcanzará el punto de equilibrio para cada valor de caudal de entrada en función de la geometría del tanque, por lo que a la hora de hacer el diseño 3D de las piezas que lo componen, habrá que poner especial cuidado en las dimensiones del tanque y el orificio de descarga. Es por ello que se decidió realizar el modelado en Solid Edge para tener un buen control dimensional.

3.2 Modelado de la planta virtual

3.2.1 Modelado 3D con Solid Edge

El modelado 3D de la planta se realiza en Solid Edge, programa desarrollado por Samsung enfocado en el diseño de piezas industriales. Aprovechando su función de simulación de piezas, podemos generar la malla superficial (mesh) de la geometría de los tanques, y con ella generar el collider para el fluido a través de la herramienta Neural Collider de Zibra. Para ello, Solid exige que las piezas estén conectadas entre sí, por lo que se añade el panel trasero.

El modelo básico de la planta (figura 3-4) son los tanques cilíndricos con un agujero en el fondo. El tamaño de este agujero serán determinante en el comportamiento dinámico del sistema, como hemos visto en el apartado anterior.

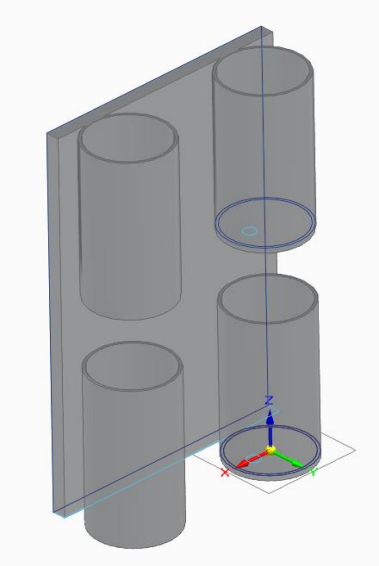


Figura 3-4. Modelo de los tanques en Solid

Las dimensiones de partida son las del sistema real:

- 13.5 cm de diámetro del tanque
- 1 cm de diámetro del orificio de descarga

Aunque posteriormente, tuvieron que irse ajustando para mantener las constantes de tiempo a medida que variabamos los parámetros de la simulación.

Se probaron distintos tamaños de orificio de descarga, debido a que la resolución del fluido y la malla están limitados por el coste computacional y conviene bajar su valor al mínimo admisible para que la simulación sea estable. A partir de 2.5 cm el fluido pasa por los orificios y la simulación es fluida, para una resolución del fluido de 200 divisiones. El mallado (Figura 3-5) se hizo lo más tosco que permite la herramienta de Solid, ya que se comprobó que una resolución de malla más alta aumentaba considerablemente el coste computacional sin aportar ventajas al objetivo de la simulación.

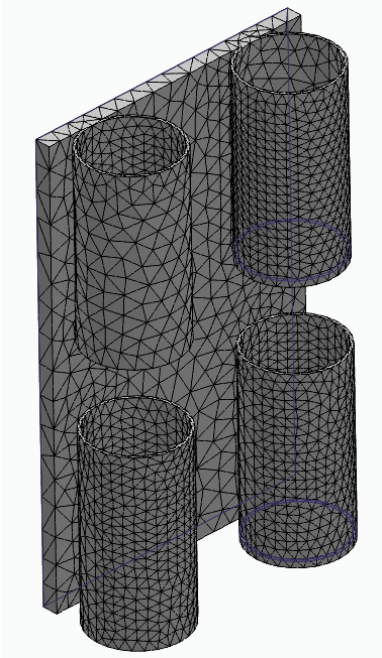


Figura 3-5. Mallado de los tanques en Solid

Una vez importadas las piezas, se realizaron varias correcciones al diseño original a medida que se iba probando su funcionamiento integradas en la planta virtual para asemejar la dinámica a la real.

El diámetro de los orificios (figura 3-6) de descarga se ajustó posteriormente por ensayo-error, asegurando que el tiempo de descarga se correspondiese con los 2 min 21 s de la planta real con reductor de caudal para la resolución de fluido (130) y el tiempo de iteración (0.8s) fijados.

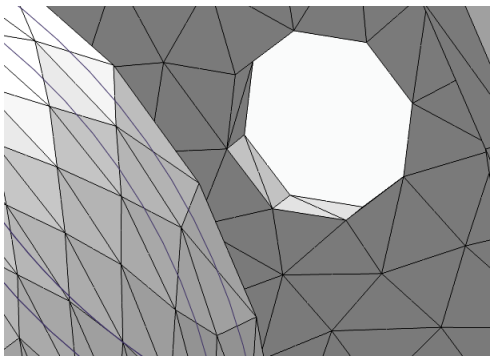


Figura 3-6. Detalle del mallado en los orificios de descarga

De igual forma se modelaron piezas para las tuberías, como se muestra en la figura 3-7:

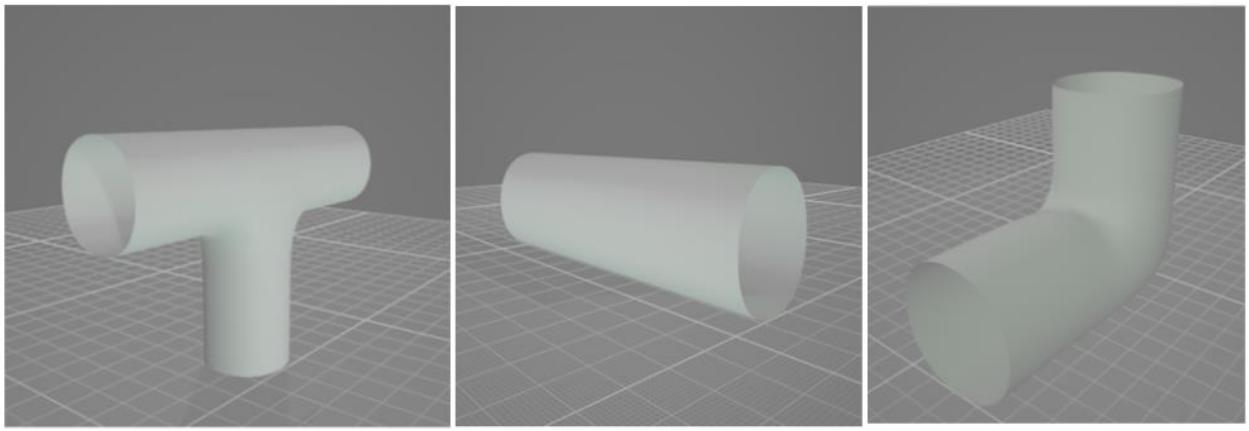


Figura 3-7. Modelado de tuberías

El resto de los objetos 3D usados para la estética del modelo (Ordenador, mesa…) se consiguieron de repositorios gratuitos en la red.

3.2.1.1 Importación a Unity

La extension de archivo predeterminada en Unity es .OBJ, por lo que los modelos anteriores se exportaron en dicho formato de Solid. Una vez importados a Unity, lo hacen como archivo de tipo paquete, que agrupan en su interior la malla, el material por defecto y el objeto tridimensional. En la figura 3-8 se muestran estos objetos)

Para incluir un archivo de este tipo en la escena hay que tener en cuenta una serie de consideraciones:

- Hay que ajustar el factor de escala de importación. En nuestro caso, 0.01.
- Se produce un error con los Neural SFD de Zibra generados a partir de un archivo de tipo paquete o los objetos que contiene, si se modifica el objeto 'Transform' de alguno de ellos en relación al paquete u objeto padre. Es recomendable extraer el objeto 3D copiando y pegando fuera de la jerarquía del paquete, y borrando este. Una vez hecho esto, asociar a la copia el collider y correspondiente Neural SFD generado.
- Los vectores normales a las superficies pueden importarse incorrectamente, Unity ofrece una opción para recalcularlas.

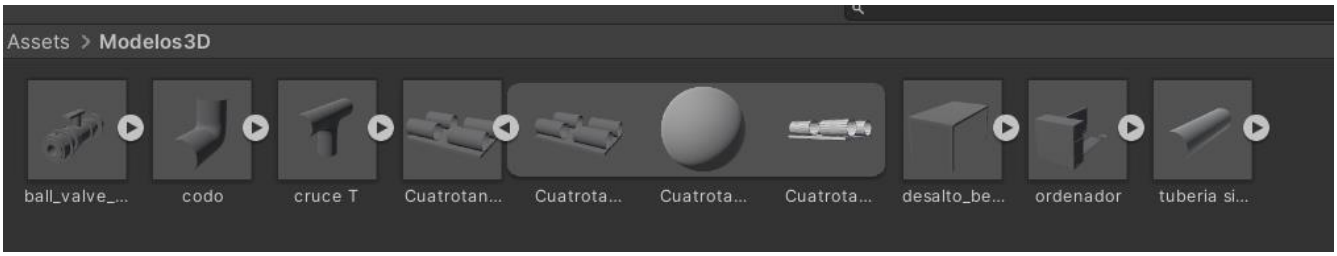


Figura 3-8. Detalle de los assets tipo paquete del proyecto

3.2.1.1.1 Generación de los SFD

Una vez importados los objetos 3D, procedimos a generar los Neural Colliders asociados a cada elemento de la planta virtual. Los tanques (figura 3-9), una vez ajustado el tamaño de los orificios de descarga funcionaron correctamente, aunque no ocurrió lo mismo con los tubos.

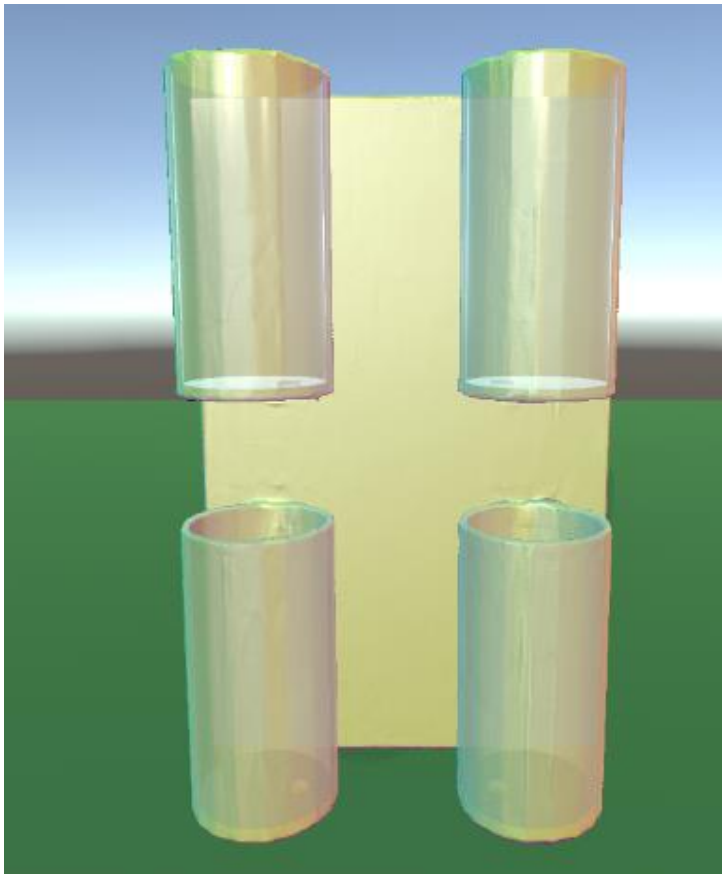


Figura 3-9. Neural SFD asociado a los tanques

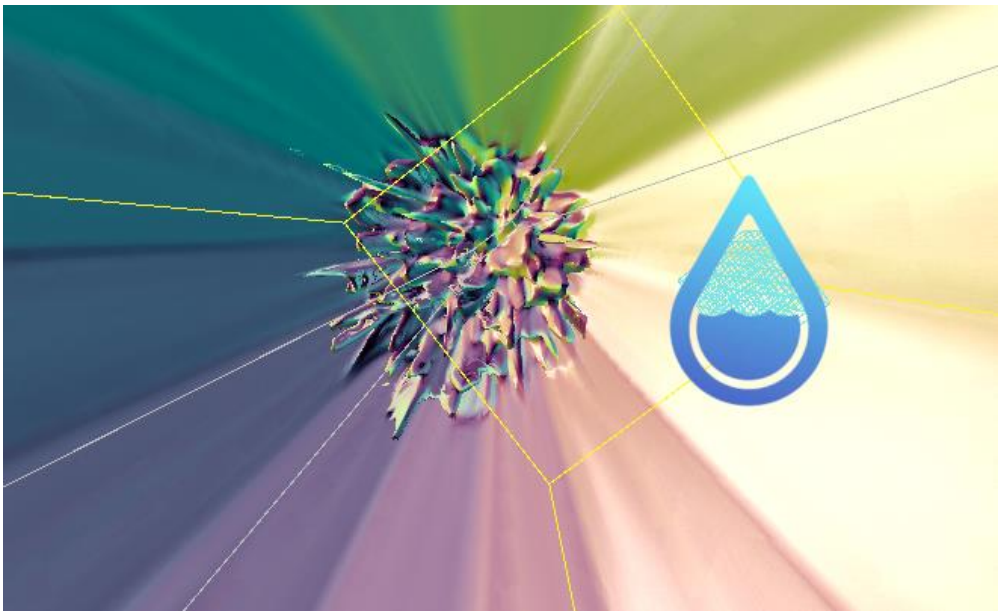


Figura 3-10. Fallo en el Neural SFD del interior de un tubo

Al generar los colliders para los tubos, se comprobó que la red neuronal interpreta incorrectamente la geometría, produciéndose efectos extraños en el interior (figura 3-10):

Se probaron diferentes modelos de tubo de distintas dimensiones, con espesor y sin él, pero los errores seguían apareciendo, lo que hacía imposible simular el flujo por el interior de los tubos. Se muestra en la figura 3-11:

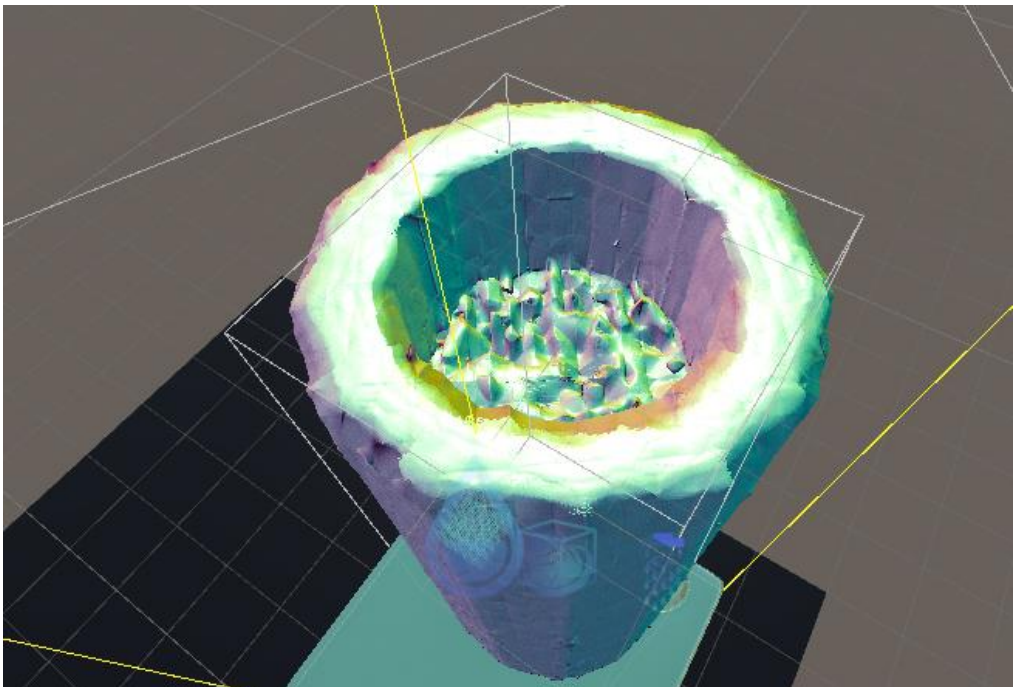


Figura 3-11. Fallo en el Neural SFD del interior de un tubo con espesor

En concreto, es prácticamente imposible simular el efecto de succión de una bomba para hacer subir el líquido por una tubería vertical, ya que se producen espontáneamente comportamientos extraños, como que el líquido suba por la cara exterior del tubo donde está situado el campo de fuerza. Se muestra en la figura 3-12:

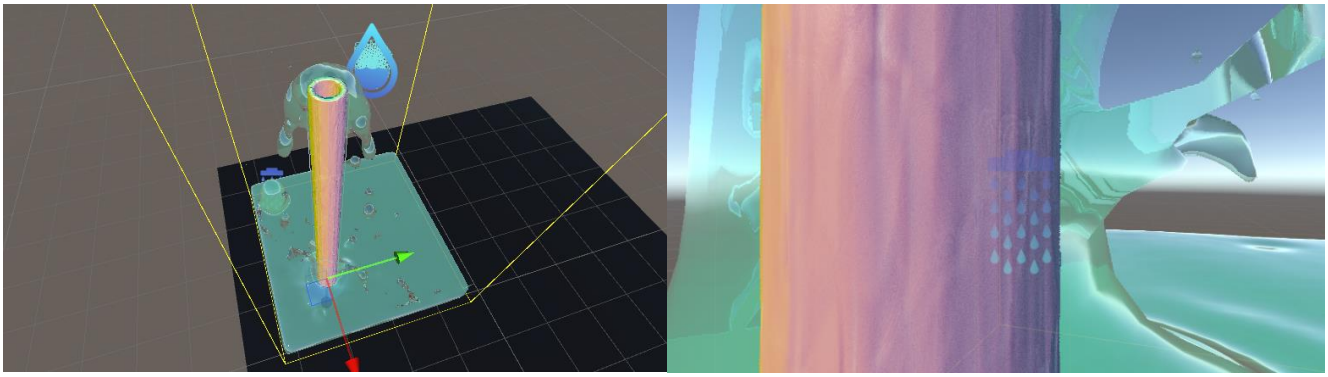


Figura 3-12. Fallo en la simulación debido a los errores en SFD de tubos

Por tanto, abandonamos la idea inicial de simular el circuito hidráulico de la planta desde las bombas, limitándonos a usar los colliders de tubos solo en la parte central que permite la descarga cruzada, donde estos comportamientos extraños del plugin no son tan evidentes.

Los grifos por tanto se modelarán con emiters cuyo volumen generado esté ligado al voltaje de las bombas de forma que sigan el comportamiento de la planta real. Para ello, codificaremos en un script los comportamientos observados en los experimentos.

3.2.2 Descripción de los elementos de la planta virtual

Se muestra en la figura 3-13 un esquema general de la estructura de la planta virtual ya consolidadada en la que se identifican los elementos funcionales de la simulación. En los sucesivos apartados se irá detallando su funcionamiento.

3.2.2.1 Sensores

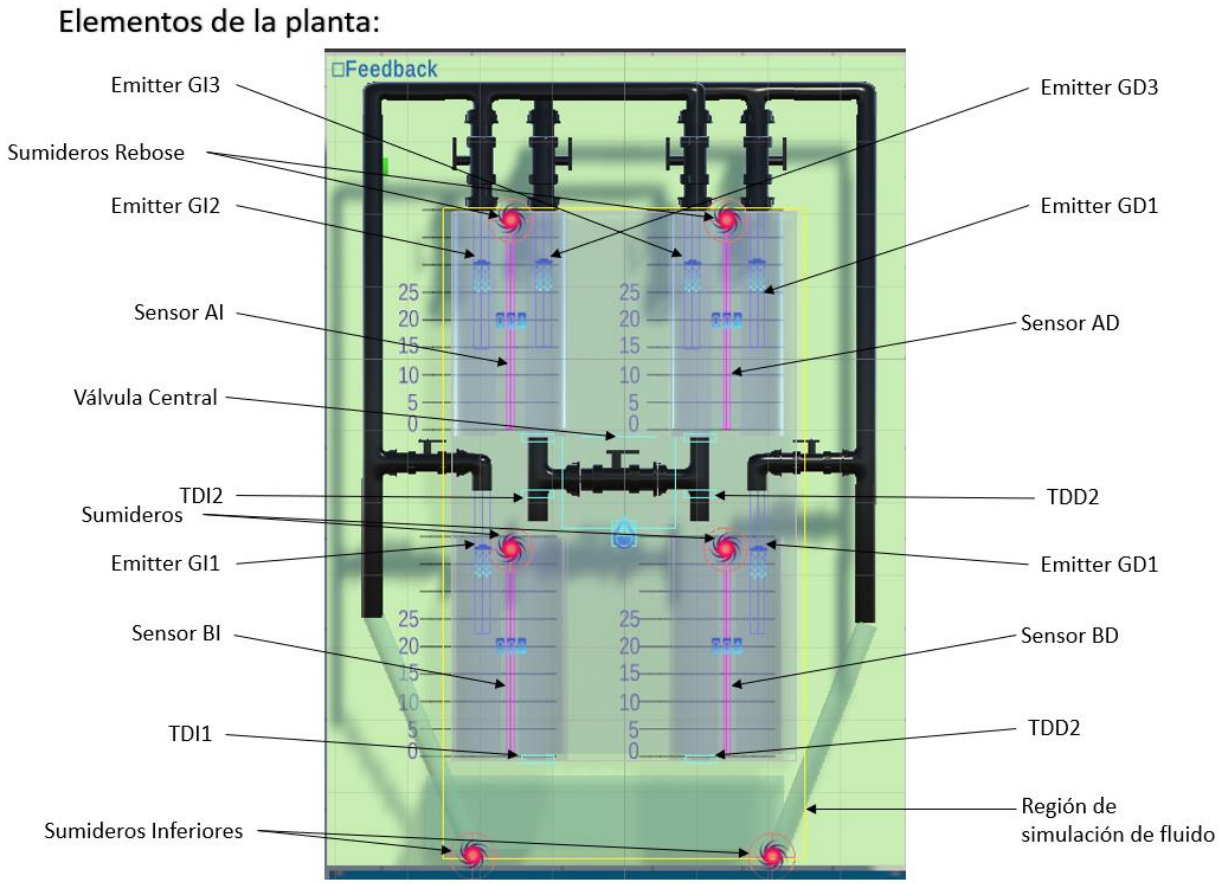


Figura 3-13. Esquema de los elementos de la planta virtual

Los sensores se han implementado como Liquid Detectors de geometría cilíndrica, empleando la propiedad 'BoundingBox' asociados al objeto de Zibra para estimar la altura mediante un ajuste lineal. El proceso por el que se llegó a esta solución detalla en el apartado 3.3.2.

3.2.2.2 Actuadores

Los actuadores se han implementado haciendo uso del objeto Emitter de Zibra, modificando mediante scripts la propiedad 'VolumePerSimTime' para hacer que el caudal generado se corresponda con el que manaría del grifo real en función del voltaje de alimentación de la bomba y la posición de las llaves. Para ello se han tenido en cuenta los umbrales de presión para cada configuración, realizando un ajuste entre el caudal mínimo y el máximo por medio de una recta. En el apartado 3.3.3 se explica con más detalle.

3.2.2.3 Válvulas

Las válvulas de configuración se modelan mediante colliders, que interaccionan con el fluido interrumpiendo el paso por los conductos. Estos también se han modelado para la parte central, pero debido a las limitaciones de la generación de colliders de Zibra, se ha evitado dar demasiado protagonismo a esta característica para asegurar el correcto funcionamiento de la simulación.

3.3 Ajuste del gemelo digital

3.3.1 Experimentación sobre la planta real

Realizamos una serie de experimentos sobre la planta real con el fin de obtener datos para ajustar la respuesta del modelo virtual a lo observado. En principio las experiencias se han realizado para la rama izquierda de la planta, ya que los sensores de el lado derecho presentan un deterioro que impide calibrarlos correctamente, y las desviaciones observadas entre una y otra rama no son apenas significativas, mostrando ambas bombas un comportamiento muy similar.

No obstante, en el código se declaran las variables necesarias para definir el comportamiento de cada grifo de manera independiente para las dos ramas, por lo que incluir las desviaciones de pérdida de presión en el modelo digital es sencillo.

Esto se ha hecho así en previsión de incorporar funciones de autoajuste del gemelo digital, de forma que puedan adecuarse los parámetros de la simulación de la planta a los comportamientos observados en la planta real.

3.3.1.1 Determinación de umbrales de presión

Al variar el voltaje de entrada de las bombas, se observa que existen intervalos en los que la presión generada no es suficiente para alcanzar la altura necesaria para que comience a manar líquido por los grifos. Esto ocurre de forma distinta para cada configuración de las llaves, por lo que hay que establecer una casuística y registrar los valores de voltaje a partir de los que el líquido empieza a fluir para cada una. En la tabla 3-1 se muestran los datos recogidos a este respecto:

	Posición Válvulas			Umbrales de Voltaje (V)		
	Llave 1	Llave 2	Llave 3	Grifo 1	Grifo 2	Grifo 3
Situación de máximo caudal	A	C	C	1.15	-	-
Sólo la vertical	A	A	C	1.15	2.6	-
Situación cruzada	A	C	A	1.15	-	2.6
Situación de mínimo caudal	A	A	A	1.15	2.4	3.25
Situación horizontal	C	A	A	-	1.5	2
Situación sólo g3	C	C	A	-	-	1.6
Situación sólo g2	C	A	C	-	1.6	-

Tabla 3–1. Umbrales de presión (Voltaje)

3.3.1.2 Determinación de caudales máximos

Igual que ocurre con los umbrales donde el caudal de cada grifo pasa de cero a empezar a fluir, existe un caudal máximo que puede manar por cada grifo en cada configuración de las llaves. Para determinarlo, se realiza un experimento en el que se cierran los tapones de descarga de todos los tanques y se pone la bomba a 5V durante un tiempo fijado de 25 s, tras ello, registramos la altura que alcanza cada tanque. De esta manera podemos conocer indirectamente el caudal que hay que fijar como máximo en cada emitter. En la tabla 3-2 se recogen los resultados.

	Posición Válvulas			Altura alcanzada (cm)		
	Llave 1	Llave 2	Llave 3	Grifo 1	Grifo 2	Grifo 3
Situación de máximo caudal	A	C	C	*13.3 en 10s	-	-
Sólo la vertical	A	A	C	17	13	-
Situación cruzada	A	C	A	18	-	12
Situación de mínimo caudal	A	A	A	19.5	11	11
Situación horizontal	C	A	A	-	17	18
Situación sólo g2	C	C	A	-	* 12.5 en 10s	-
Situación sólo g3	C	A	C	-	-	* 12 en 10s

Tabla 3–2. Caudales Máximos (Altura)

*En 25s se produce el desborde, por lo que se repite el experimento en 10s, considerando que la altura sube de forma lineal para un caudal de entrada dado.

3.3.1.3 Determinación del tiempo de descarga

Se realiza un experimento sobre el tanque AI para determinar el tiempo de descarga desde 25 cm. Este valor está íntimamente ligado al diámetro efectivo de los orificios de descarga, por lo que en base a él se dimensionan los orificios del modelo en Solid, teniendo en cuenta que los cambios en la resolución y demás parámetros de la simulación de fluido hacen inútil ajustarnos a las dimensiones reales si damos prioridad a la exactitud del comportamiento dinámico de la planta virtual.

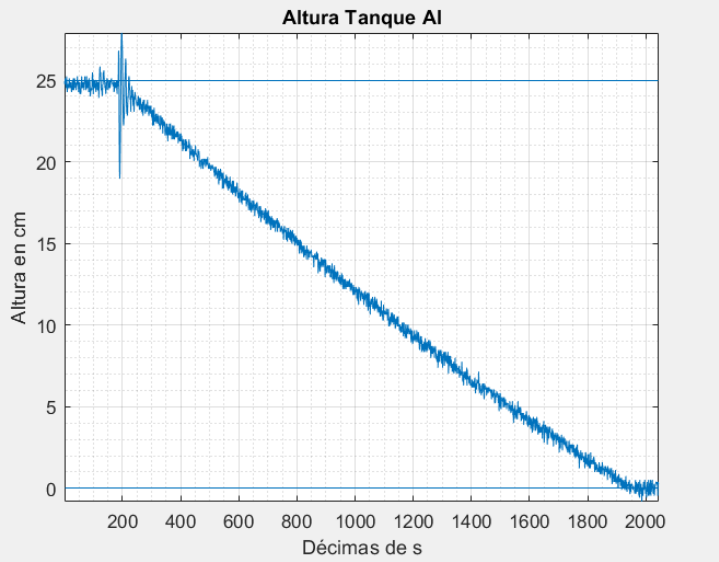


Figura 3-14. Descarga de uno de los tanques con reductor

A la vista de los datos de la figura 3-14 se establece que el tiempo de descarga con reducción es de aproximadamente 2 min 20 s con un comportamiento prácticamente lineal durante todo el proceso.

Tras ajustar la planta virtual, y una vez hecha la conexión, se repitió este experimento sobre ella y se, se obtuvo una respuesta similar en tiempo, aunque menos lineal, como se observa en la figura 3-15:

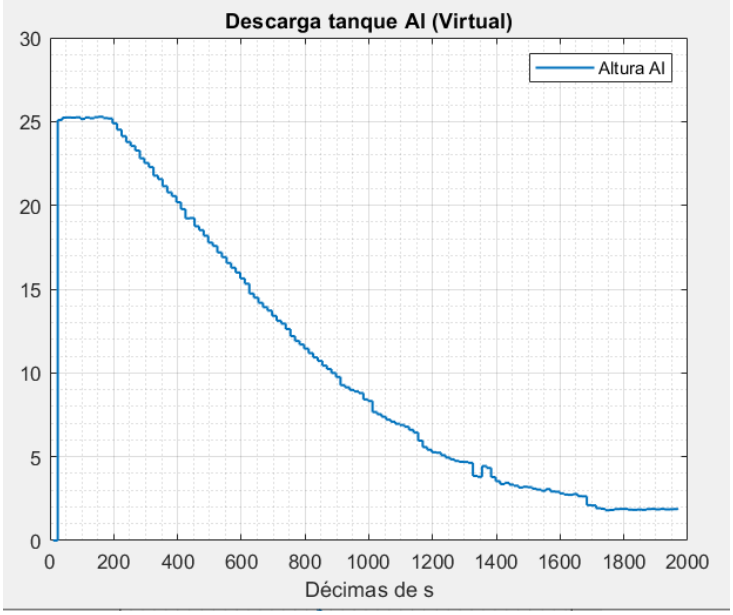


Figura 3-15. Descarga de tanque virtual

Si realizamos el mismo experimento sin los tapones reductores de caudal (ver figura 3-16), se comprueba que el sistema tiene una dinámica mucho más rápida (tiempo de descarga 64 s), y en ocasiones se aleja de la linealidad (se producen remolinos aleatoriamente al final de la descarga que modifican la curva)), por lo que hacemos caso a la recomendación del manual de la planta y realizamos el modelo virtual para la configuración con reductores.

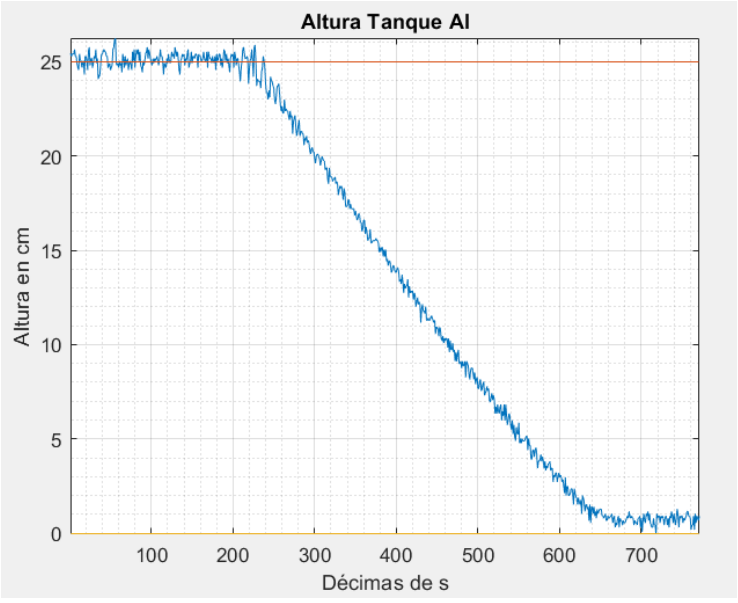


Figura 3-16. Descarga de tanque sin reductor

3.3.2 Script de Sensores

Para simular la medida de altura en la planta virtual, se programa un script en C# que accede a las propiedades del objeto detector de Zibra y las transforma en medidas en escala real.

A continuación se detallan las dos estrategias por las que se trató de estimar la medida, comentando las dificultades encontradas y las soluciones que se fueron aplicando para solventarlas.

3.3.2.1 Estrategia de estimación de la medida

Las características del Solver de Zibra complicaron en buena medida la correcta estimación de las alturas de líquido presentes en cada tanque. A continuación se detalla el proceso por el que se llegó a una solución aceptable.

3.3.2.1.1 Estimación por recuento de partículas en el detector

En primer lugar, se planteó usar la propiedad de ParticlesInside del 'liquid detector' para estimar la altura, escalando linealmente la cantidad de partículas, pero tras muchas pruebas y intentos de calibrar se comprobó que este método era inviable. El Solver de Zibra elimina partículas y crea otras en las zonas en las que la simulación se vuelve inestable (Interacción con neural colliders principalmente), por lo que el número de partículas no se mantiene constante en estado estacionario. En aplicaciones donde importa más la dinámica del fluido que sus características estáticas, esto no supone demasiado problema, pero en nuestro caso, el error resultante en la altura de la columna era demasiado alto, como se observa en la figura 3-17.

Además, el método del recuento de partículas presentaba el inconveniente de tener que reajustar el factor de escala cada vez que se cambiaba alguna característica de la simulación que afectase al tamaño de las mismas (resolución de rejilla, stiffness, densidad de partículas…), por lo que se decidió buscar otra forma de estimar la altura.

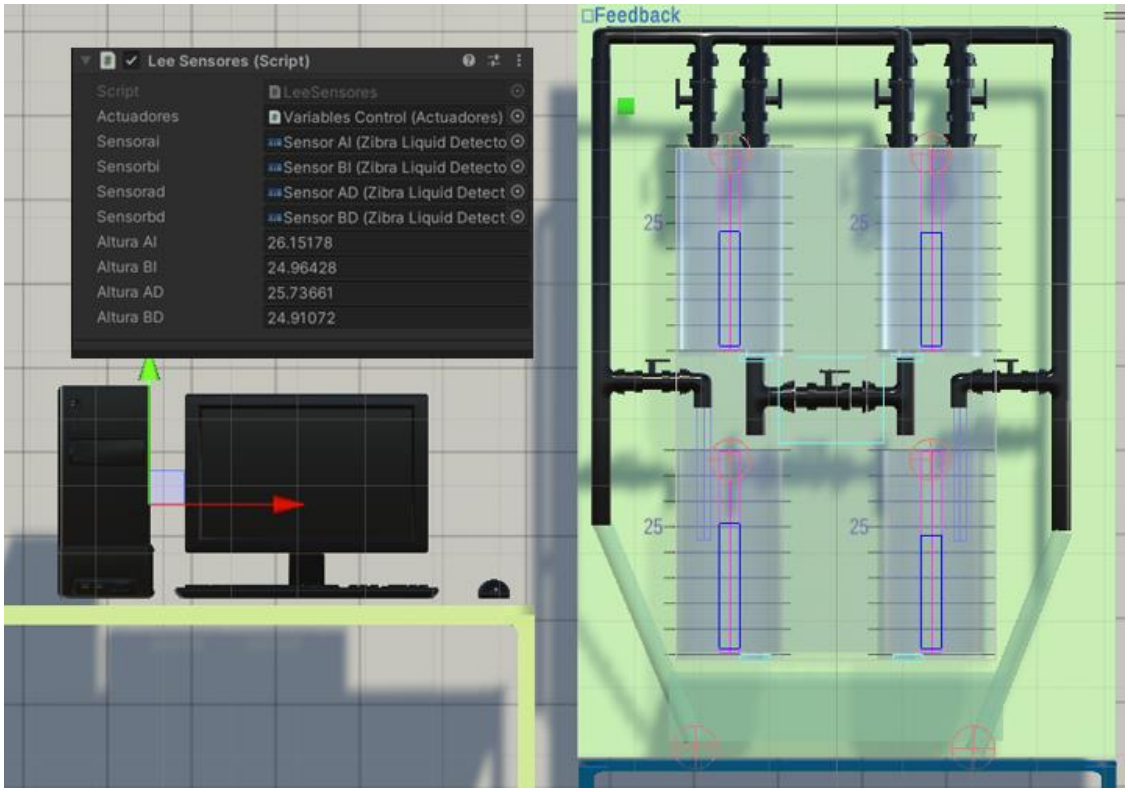


Figura 3-17. Poca precisión de las medidas

3.3.2.1.2 Estimación mediante BoundingBox

Un segundo método consiste en usar las propiedades 'BoundingBoxMax' y 'BoundingBoxMin' asociadas al objeto del detector, que proporcionan respectivamente las coordenadas de los puntos más extremos en los que se detecta líquido dentro del SFD asociado. Escalando la diferencia entre ellos y escalando mediante el factor 'escalasens', fijado para graduar la escala entre 0-25 cm, se obtiene una mejor medida de altura.

Emplear estos valores para estimar la medida supuso una notable mejoría respecto de la estimación por recuento de partículas, aunque aún se observaba una deriva temporal, esta vez hacia abajo, debido a que en la simulación de Zibra, el líquido no es totalmente incompresible. El Solver presenta un parámetro ajustable llamado 'Stiffness', que permite regular en cierta medida este efecto, pero tiene por contrapartida una ralentización del movimiento del fluido que haría imposible acercarnos a la dinámica del sistema real.

3.3.3 Script de Sensores

Para simular la medida de altura en la planta virtual, se programa un script en C# que accede a las propiedades del objeto detector de Zibra y las transforma en medidas en escala real.

A continuación se detallan las dos estrategias por las que se trató de estimar la medida, comentando las dificultades encontradas y las soluciones que se fueron aplicando para solventarlas.

3.3.3.1 Estrategia de estimación de la medida

Las características del Solver de Tras consultar de nuevo con el equipo de soporte de Zibra, probamos a variar los parámetros de Gravedad, Tiempo máximo de iteración y Número de iteraciones, observando los resultados. El mejor ajuste en dinámica y reducción de las derivas se produjo al bajar el tiempo máximo de iteración a 0.8.

Con los ajustes anteriores se obtuvo una medida estable, aunque todavía ruidosa.

Para evitar el ruido asociado a los valores altos de caudal se redujo el diámetro del cilindro SFD asociado al sensor a 1 cm en unidades de Unity, como se muestra en la figura 3-18.

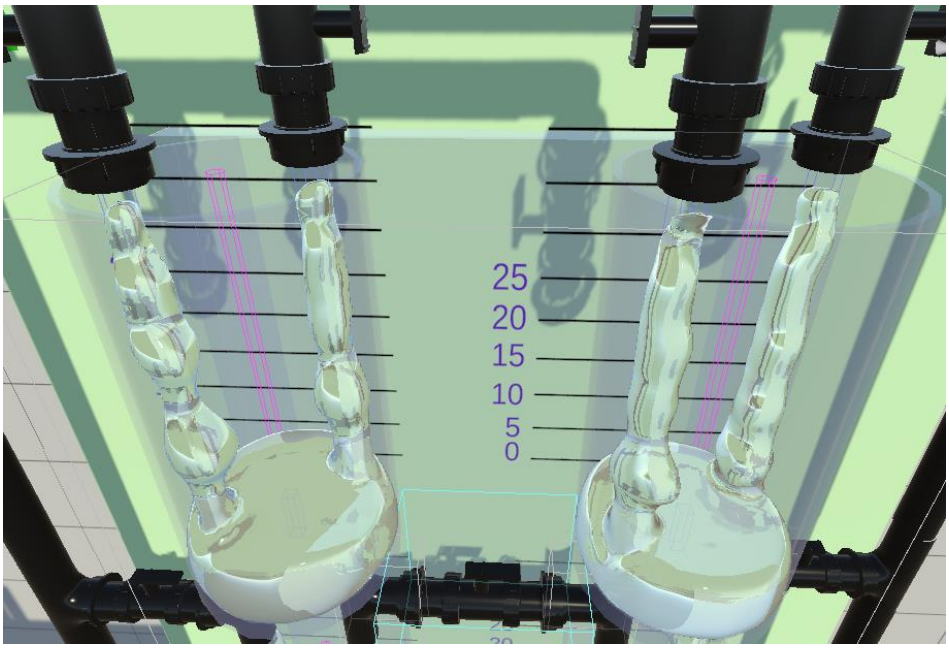


Figura 3-18. Eliminación del ruido por salpicaduras

Para terminar de tratar la señal, optamos por implementar un filtro paso bajo que limita las variaciones bruscas de la señal asociadas al ruido, aumentando el peso dado al valor anterior hasta comprobar que la dinámica de la medida empezaba a ser demasiado lenta. En la figura 3-19 se muestra la implementación.

```
47          escalasens = 25f/(10.91f - 9.09f); // para res =130
48          alfa = 0.1f; // constante del filtro
49      }
50
51      // Update is called once per frame
        Mensaje de Unity | 0 referencias
52      void Update()
53      {
54          // Cuenta de Tiempo Transcurrido
55          if (Time.frameCount == 1) {tiempoinicio = Time.time;}
56          // Lectura Sensores
57          alturasfAI = (sensorai.BoundingBoxMax.y- sensorai.BoundingBoxMin.y) *escalasens;
58          alturasfAD = (sensorad.BoundingBoxMax.y - sensorad.BoundingBoxMin.y) * escalasens;
59          alturasfBI = (sensorbi.BoundingBoxMax.y - sensorbi.BoundingBoxMin.y) * escalasens;
60          alturasfBD = (sensorbd.BoundingBoxMax.y - sensorbd.BoundingBoxMin.y) * escalasens;
61          // Filtrado lecturas:
62          alturaAI = (alturasfAI+0.01f) * alfa + alturaAI * (1 - alfa);
63          alturaBI = alturasfBI * alfa + alturaBI * (1 - alfa);
64          alturaAD = alturasfAD * alfa + alturaAD * (1 - alfa);
65          alturaBD = alturasfBD * alfa + alturaBD * (1 - alfa);
```

Figura 3-19. Filtrado de las medidas de los sensores

3.3.4 Script de Actuadores

Para ajustar el comportamiento de los actuadores de la planta virtual a los observados en los experimentos, se programa un script en C# en el que se activan y desactivan los colliders asociados a las válvulas de descarga, se calculan los caudales de cada grifo en función del voltaje de las bombas y la casuística de apertura o cierre de estas, y se consideran las situaciones de interconexión entre las dos columnas de tanques.

En una primera aproximación, se replican los datos experimentales de la rama izquierda en la derecha, ya que los sensores de esta última en la planta real presentan mayor variabilidad en las medidas, y las desviaciones entre una y otra rama son prácticamente despreciables.

Se explicará el funcionamento del script comentando algunos fragmentos relevantes del código:

- En primer lugar, se fija la escala de actuación (V/VolPerSimTime) en base al caudal máximo simulado, como se muestra en la figura 3-20. Este se escoge de manera que el tiempo de subida en la situación de máximo caudal coincida con el de la planta real, es decir, 18.75 s en subir hasta los 25 cm.

```
void Start()
{ // Inicialización

    qmax = 0.028f;
    escalactuadores = 5 / qmax; // Se ajusta para fijar el caudal máximo a 5V (0.028 VolPerSimtime - 18.75s hasta 25cm)

}
```

Figura 3-20. Escala de actuación

- A continuación (ver figura 3-21), ya dentro del bloque Update, que se ejecuta en cada 'frame' de la simulación, se asegura la saturación de la entrada de voltaje para las bombas, se comprueba el estado de las variables globales asociadas a la activación o desactivación de los colliders de la descarga, distinguiendo dos ramas de comportamiento en función de la posición de la llave central:

  - En caso de estar cerrada, los colliders activados son los superiores (TDD2_,TDI2_), y la descarga se realiza con normalidad.

  - En el caso de estar abierta (es decir, situación de descarga cruzada), se emplean los colliders inferiores (TDD2, TDI2) y se desactivan los colliders de los tubos en T y los sumideros de rebose de la rama contraria para permitir que el fluido caiga al otro tanque sin chocar o destruirse.

```
void Update()
{
    // Saturación Bombas:
    if (BombaI > 5) { BombaI = 5f; }
        if (BombaI <= 0) { BombaI = 0f; }
        if (BombaD > 5) { BombaD = 5f; }
        if (BombaD <= 0) { BombaD = 0f; }

    // Activacion Colliders Tapones Descarga

    //Boton ----------- Acción//
    if (LlaveCentral)
    {
        DD2.enabled = false;
        DI2.enabled = false;
        Llavecentral.enabled = true;
        if (TDD1) { DD1.enabled = true; } else { DD1.enabled = false; };
        if (TDD2) { DD2_1.enabled = true; } else { DD2_1.enabled = false; };
        if (TDI1) { DI1.enabled = true; } else { DI1.enabled = false; };
        if (TDI2) { DI2_1.enabled = true; } else { DI2_1.enabled = false; };
    }

    if (!LlaveCentral)  // configuraciones cruzadas
    {
        DD2_1.enabled = false;
        DI2_1.enabled = false;

        Llavecentral.enabled = false;
        if (TDD1) { DD1.enabled = true; } else { DD1.enabled = false; };
        if (TDD2 ) { DD2.enabled = true; ReboseBI.enabled = false; TI.enabled = false; } else { DD2.enabled = false; TI.enabled = true; ReboseBI.enabled = true; }; // para permitir la descarga cruzada

        if (TDI1) { DI1.enabled = true; } else { DI1.enabled = false; };
        if (TDI2 ) { DI2.enabled = true; TD.enabled = false; ReboseBD.enabled = false; } else { DI2.enabled = false; TD.enabled = true; ReboseBD.enabled = true; }; // para permitir la descarga cruzada

        if(TDD2 && TDI2) { Llavecentral.enabled = true; DD2.enabled = true; TI.enabled = true; ReboseBI.enabled = true; DI2.enabled = true; TD.enabled = true; ReboseBD.enabled = true; }
    }
```

Figura 3-21. Activación de Colliders

- Ahora (ver figura 3-22), en función de la posición de las llaves de los grifos, se activan o desactivan los emitters correspondientes y se establece la casuística en la que se introducen los datos experimentales relativos a los umbrales de presión y los caudales máximos. Dentro de cada uno de estos 'ifs' es donde se calcula el caudal de cada grifo. Lo vemos con más detalle en el siguiente punto.

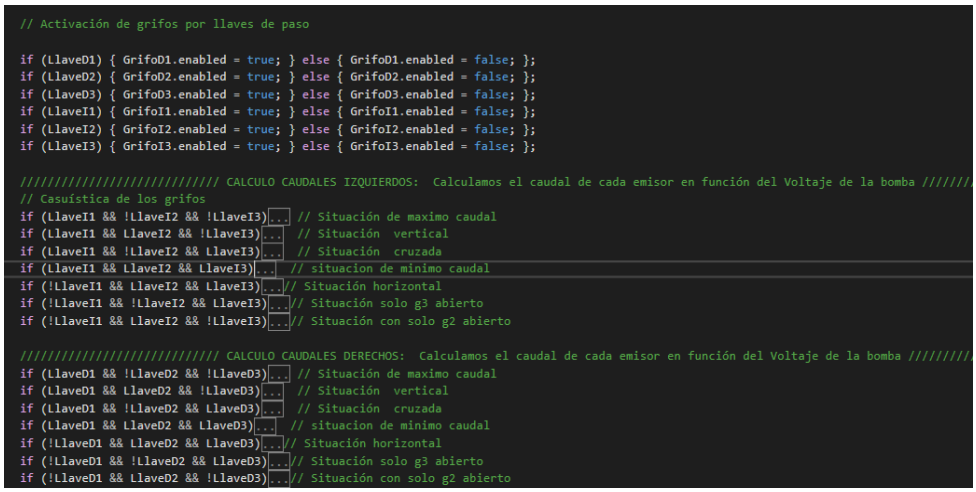


Figura 3-22. Casuística de los grifos

- Para cada situación de la casuística y cada grifo, el cálculo del caudal en función del voltaje se ha planteado de manera que este se mantiene a cero hasta que se supera su umbral de voltaje, a partir de ahí varía de forma lineal hasta alcanzar el caudal máximo.

  La pendiente de la recta de actuación de cada emitter se calcula por tanto en función de los umbrales y caudales máximos para ajustar el comportamiento observado en los experimentos. Se muestra en la figura 3-23 la sección del cálculo de la situación de mínimo caudal:

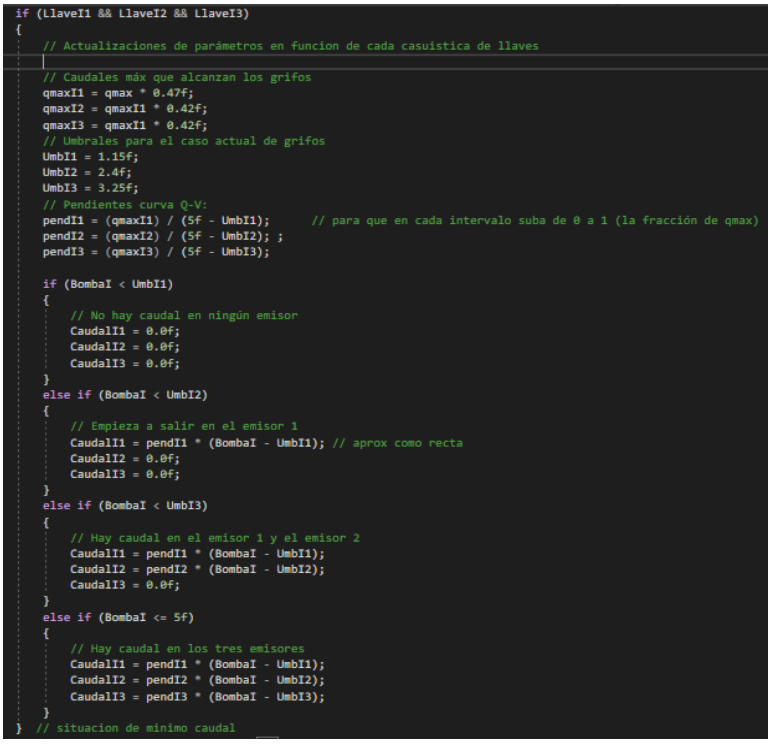


Figura 3-23. Cálculo de los caudales en función del Voltaje

- Por último (ver figura 3-24), se escalan los caudales y se actualiza la propiedad 'VolumePerSimTime' de cada emitter para modular el caudal de salida de cada grifo de la planta virtual.

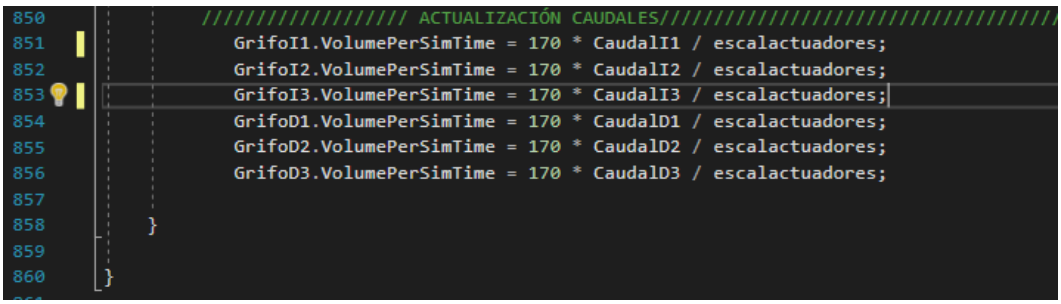


Figura 3-24. Actualización de los Emitters

4 CONEXIÓN CON LA PLANTA REAL

En este capítulo se aborda la conexión entre la planta virtual y la planta real, de manera que pueda establecerse un flujo bidireccional de datos que permita realizar ensayos de controladores sobre ambas.

El protocolo sobre el que realizaremos la conexión es el TCP/IP, al cual dedicamos un primer subcapítulo introductorio para recordar los conceptos asociados al mismo. La información incluida en este apartado proviene en su mayoría de los apuntes de clase [15] de la asignatura Informática Industrial, impartida en el cuarto curso del Grado en Ingeniería en Tecnologías Industriales.

En una primera fase los datos intercambiados serán las medidas de los sensores y los voltajes de cada bomba, con lo que podrá establecerse un bucle de control donde las actuaciones se calculan de forma remota.

Una vez realizada y depurada esta implementación, se habrán sentado las bases para añadir funciones de autoajuste al gemelo digital, que permitirían adecuar el comportamiento de la simulación a las desviaciones que se producen en la planta física, mediante técnicas cercanas al campo del control adaptativo.

4.1 Protocolo TCP

El Protocolo de Control de Transmisión (TCP, por sus siglas en inglés) es uno de los protocolos fundamentales en la arquitectura de Internet. Diseñado originalmente por Vinton G. Cerf y Robert E. Kahn [16] en la década de 1970, TCP se ha convertido en el protocolo de transporte más ampliamente utilizado en la comunicación de redes de datos. Su función principal es proporcionar una comunicación confiable y orientada a la conexión entre aplicaciones ubicadas en diferentes equipos en red.

4.1.1 El Modelo de Referencia TCP/IP

El protocolo TCP es una parte integral del Modelo de Referencia TCP/IP, que se ha convertido en el estándar de facto para la interconexión de redes en todo el mundo.

El modelo TCP/IP se divide en cuatro capas (capa de aplicación, capa de transporte, capa de internet y capa de acceso a la red) a diferencia del modelo de referencia OSI del que deriva, estructurado en 7 capas. Como puede observarse en la figura 4-1, existe una correspondencia entre las capas de ambos.

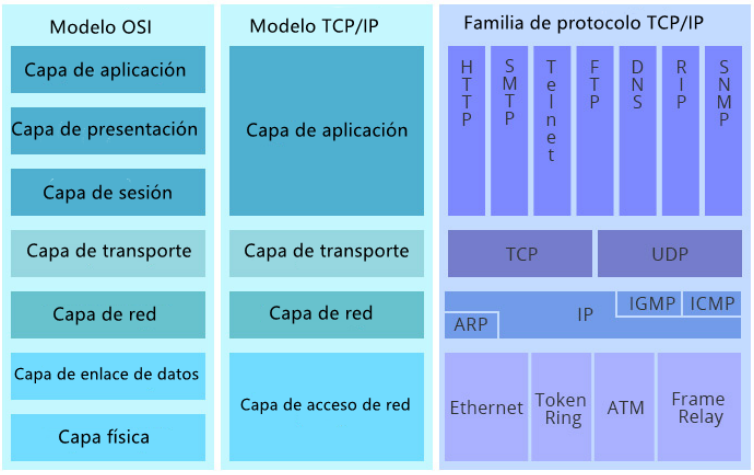


Figura 4-1. Correspondencia entre capas OSI-TCP/IP

4.1.1.1 Principios del protocolo TCP

TCP se basa en varios principios clave para garantizar una comunicación fiable y eficiente:

- Orientado a la conexión: Antes de que se pueda establecer una comunicación, TCP requiere establecer una conexión entre el emisor y el receptor. Esto implica un intercambio de mensajes de control conocido como un "apretón de manos" (handshake) para establecer y cerrar la conexión de manera ordenada.
- Fiable: TCP garantiza que los datos enviados sean recibidos correctamente y en el orden correcto. Para lograr esto, utiliza números de secuencia para identificar cada segmento de datos y utiliza confirmaciones (ACKs) para informar al remitente sobre los datos recibidos con éxito.
- Control de flujo: TCP tiene mecanismos para controlar la cantidad de datos que se envían entre el remitente y el receptor, evitando así que el receptor se vea abrumado por una sobrecarga de datos.
- Control de congestión: TCP monitorea la congestión en la red y ajusta la tasa de envío de datos para evitar la sobrecarga de la red. Utiliza algoritmos de control de congestión como TCP Reno o TCP Cubic para regular el flujo de datos y garantizar un rendimiento óptimo.
- Segmentación y reensamblaje: TCP divide los datos en segmentos más pequeños antes de enviarlos y los reensambla en el destino. Esto permite una transmisión eficiente de grandes volúmenes de datos y garantiza una entrega ordenada y confiable.

4.2 Implementación del Servidor TCP en Unity

Como se comenta en el punto anterior, TCP es un protocolo orientado a conexión, por la que la comunicación se basa en el modelo cliente-servidor entre dos entidades: el sevidor actuará como elemento pasivo atendiendo a las peticiones que se le hagan desde el otro extremo, el cliente.

Se presentará en los siguientes subcapítulos la implementacion del modelo realizada, explicando por separado cada extremo. En este caso, el servidor residirá en Unity, implementado mediante un conjunto de scripts en C#, y el cliente residirá en Matlab, desde donde puede controlarse la planta real.

Resumidamente, la conexión se establece como sigue:

- El servidor de Unity esperará a recibir la petición de conexión del cliente asociado a Matlab
- Desde Matlab, el cliente envía la petición de conexión.
- Una vez recibe la petición, el cliente creará un socket de red por el que se establece el flujo de datos. A través de él se reciben los voltajes de las bombas que envía el cliente y se responde con las medidas de altura proporcionadas por los sensores.
- El cliente de Matlab recibe de vuelta las medidas de los sensores virtuales

4.2.1 Script 'TCPServer.cs'

En él declaramos la clase pública TCPServer, en la que se incluyen como métodos las operaciones asociadas a la inicialización, envío, recepción de datos y cierre de la conexión. Se muestran dichos métodos en las figura 4-2 y 4-3:

```
// METODO PARA INICIALIZAR EL SERVIDOR TCP:
1 referencia
public async Task StartServerAsync(string ipAddress, int port)
{
    IPAddress localAddress = IPAddress.Parse(ipAddress);
    server = new TcpListener(localAddress, port);
    server.Start();
    Debug.Log("Servidor TCP iniciado. Esperando conexiones...");
    client = await server.AcceptTcpClientAsync();
    stream = client.GetStream();
    buffer = new byte[1024];
}
```

Figura 4-2. Método de inicialización del servidor y escucha de peticiones

```
// METODO PARA RECIBIR DATOS DEL CLIENTE
1 referencia
public string RecibeDatos()
{
    if (client.Connected )
    {
        int bytesRead = stream.Read(buffer, 0, buffer.Length);
        string data = Encoding.ASCII.GetString(buffer, 0, bytesRead);
        return data;
    }
    else
    {
        StopServer();
        return "0 0";
    }
}
// METODO PARA ENVIAR DATOS AL CLIENTE
2 referencias
public void EnviaDatos(string data)
{
    byte[] buffer = Encoding.ASCII.GetBytes(data);
    stream.Write(buffer, 0, buffer.Length);
}
// METODO PARA CERRAR EL SERVER
2 referencias
public void StopServer()
{
    stream.Close();
    client.Close();
    server.Stop();
    Debug.Log("Servidor TCP detenido.");
}
```

Figura 4-3. Métodos de Recepción, Envío, y Cierre de la conexión

4.2.2 Script 'LanzadorTCP.cs'

En él creamos una instancia de la clase TCPServer descrita anteriormente, y se emplean sus métodos para realizar la conexión propiamente dicha. Se muestra en la figura 4-4:

```
private TCPServer miserver;

Mensaje de Unity | 0 referencias
private async void Start()
{
    // INICIALIZO EL SERVER
    Tconex = 1f;
    miserver = new TCPServer();
    leesens = GetComponent<LeeSensores>(); // para poder acceder a las alturas leidas y enviarlas
    actua = GetComponent<Actuadores>(); // para poder acceder a las boombas

    await miserver.StartServerAsync("127.0.0.1", 5555); // Ejemplo: IP: 127.0.0.1, Puerto: 5555
    miserver.EnviaDatos("Conexion Aceptada");
    Debug.Log("Conexion establecida");

    InvokeRepeating("ConexionTCP", 0.0f, Tconex); // Envío y recepcion de datos cada Tconex segundos
}
```

Figura 4-4. Creación de la instancia 'miserver' y escucha de la petición de conexión del cliente

En el método Start asociado (Figura 4-4) se establece la conexión por medio de una espera asíncrona de la petición de conexión del cliente. Una vez es recibida, se devuelve un mensaje al cliente a modo de confirmación.

Tras ello, se invoca al método de LanzadorTCP 'ConexionTCP' (Figura 4-5) mediante la llamada 'InvokeRepeating', que permite realizar una tarea de forma periódica. En este método es donde se realizan síncronamente el envío y recepción de los datos, así como la codificación/decodificación en string de los mismos. Se muestra en la figura 4-5:

```
private void ConexionTCP()
{
    // CONVERSIÓN DE LAS LECTURAS A STRING
    string datosenvio = leesens.alturaBI.ToString("F3") + " " + leesens.alturaAI.ToString("F3") + " " + leesens.alturaAD.ToString("F3") + " " + leesens.alturaBD.ToString("F3");
   // ENVÍO DE ALTURAS
    if (enviando) { miserver.EnviaDatos(datosenvio); }

    // GESTIÓN DE LA DESCONEXIÓN (evitar cuelgue de Unity)
    if (!miserver.client.Connected)
    {
        Debug.Log("El cliente está desconectado");
        return ;
    }

    // RECEPCIÓN DE LOS NUEVOS VOLTAJES DE BOMBA
    else if (Recibiendo && miserver.client.Connected)
    {
        datosrec=miserver.RecibeDatos();
        Debug.Log("Voltaje Bombas Recibido: " + datosrec);
        // EXTRACCIÓN DE LOS DATOS DEL STRING:
        string[] floatStrings = datosrec.Split(" "); // Dividir el string en substrings al estar separadas por el espacio
        VI = Convert.ToSingle(floatStrings[0], CultureInfo.InvariantCulture.NumberFormat); // Convertir el  substring a float. Hay que incluir CultureInfo para interpretar el
        VD = Convert.ToSingle(floatStrings[1], CultureInfo.InvariantCulture.NumberFormat);

        // Actualización de las bombas
        actua.BombaD=VD;
        actua.BombaI=VI;
    }
}
```

Figura 4-5. Envío y recepción síncronos desde Unity

4.3 Cliente TCP en Matlab

El extremo de Matlab de la conexión actúa como parte activa, solicitando establecer la conexión con el server de Unity, recibiendo las lecturas de los sensores, calculando la acción de control y enviando de vuelta los voltajes de las bombas. A continuación se mostrarán los fragmentos de código más representativos del funcionamiento.

4.3.1 Inicialización

En la Figura 4-6 se muestra el fragmento de código en que se inicia el protocolo de conexión desde Matlab con el servidor previamente descrito, que ya se supone en escucha de nuevas peticiones de conexión. Una vez lanzada la petición, espera a recibir una confirmación por parte del servidor, que si todo va bien, llegará en forma de string con el mensaje 'Conexion Aceptada', como programamos previamente en el script de C#. Se muestra por pantalla dicha confirmación y se continúa el curso del programa.

Además, en dicho código se incluye una etiqueta de función asociada al campo 'ErrorOcurredFcn' de la estructura asociada al cliente, cuyo contenido es simplemente un manejador para los posibles errores que puedan terminar con la conexión de forma abrupta.

```
try % para el tratamiento de errores

%% Conexion al servidor remoto de Unity como cliente:
Tconex=1;
client = tcpclient("127.0.0.1",5555)
client.ErrorOccurredFcn = @TerminaConex
disp("Peticion de conexión")
pause(1)
confirm=read(client,client.NumBytesAvailable,'string');
disp(confirm)
```

Figura 4-6. Conexión al servidor desde el cliente en Matlab

4.3.2 Bucle de envío y recepción

En la figura 4-7 se muestra el proceso de envío y recepción de datos, obviando por mayor claridad el espacio en el que se implementa el cálculo de la acción de control.

```
%% BUCLE DE ENVÍO Y RECEPCIÓN DE DATOS
while(1)
pause(Tconex)  %% Muestreo de la conexión

   %% RECEPCIÓN DE DATOS SENSORES:
   rec= read(client,client.NumBytesAvailable,'string');
   % Conversion de los datos de string a formato double
   if(isstring(rec))
   str = strrep(rec, ',', '.'); % Reemplazar comas por puntos
   datosnum = str2num(str); % Convertir el string en array de números
   if(not(isempty(datosnum))) % cuando la cadena viene superpuesta, st2num devuelve vacío, así se des
   datosnum=datosnum(1:4);
   disp(datosnum)
     Alturas=[Alturas;datosnum]; %% Almaceno las medidas: [BI AI AD BD]
   end
   end

    %% ((((((((((((((CÁLCULO DE ACCIÓN DE CONTROL: ))))))))))))))))))))))

     %% ENVÍO DEL VOLTAJE CALCULADO:
   i=i+1;
   Bombas(i,1)=control;
   env=string([num2str(Bombas(i,1),4) ' ' num2str(Bombas(i,2),4)]); % Conversión a string en formato
   writeline(client,env)
end
```

Figura 4-7. Bucle de envío y recepción desde el cliente en Matlab

El método de conversión de los datos de String a float en el caso de Matlab es bastante robusto de la forma en la que se ha implementado, por lo que apenas se producen fallos en la lectura de los sensores. En el extremo de Unity, en cambio, sí que se producían fallos al convertir el string en float, provocando que los voltajes de las bombas tomasen valores extraños al alcanzar la saturación de los actuadores. Se corrigen este y otros fallos según lo mostrado en el subcapítulo siguiente.

4.3.3 Gestión de errores

Para asegurar la estabilidad del programa frente a posibles fallos en la conexión, se han realizado algunos cambios y mejoras a medida que se han ido produciendo errores. Entre ellas se incluyen:

- Encapsulado del código de Matlab dentro de una llamada 'try', que permite capturar mediante la orden'catch' el error que se produce al terminar la conexión desde el servidor. En ese caso, se ejecuta el código que lanza los objetos gráficos asociados a los resultados del experimento.
- En algunos casos las tramas de datos de los sensores llegaban superpuestas, lo que provocaba valores erróneos que era preciso descartar. De la forma en la que se ha implementado la conversión de los datos recibidos a float, se descartan los datos espúreos de manera implícita.
- En la fase de inicialización se incluye el manejador asociado al fallo en la petición de conexión del cliente, que simplemente interrumpe la ejecución.
- Cambio en la forma en la que se realiza la conversión de string a float en el lado de Unity, pasando de usar la función 'float.parse(..)' a emplear 'Convert', incluyendo la librería 'System.globalization' que permite interpretar correctamente el punto decimal.

4.4 Prueba de la conexión. Control de la planta virtual

En este último apartado se implementa y ajusta un controlador PID básico a modo de testeo de la conexión. El tiempo de muestreo para esta prueba se ha fijado en 1s, para no comprometer la estabilidad de la simulación, que se ve afectada por el ciclo de envío y recepción de datos.

4.4.1 Implementación

La implementación se ha realizado en el espacio reservado anteriormente en el Script de comunicación de Matlab, como se muestra en la figura 4-8.

```
% %% CÁLCULO DE ACCIÓN DE CONTROL: PID

 error = referencia - datosnum(1);  % Se controla la altura del tanque 1 en este ejempl

 %Acción Proporcional
 proporcional = Kp * error;

 % Acción Integral con Antiwindup
 if(control<=5 && control>=0)
 integral = integral + Ki * error;
 end

 % Acción derivativa:
 derivativo = Kd * (error - error_previo);

 % Calcular la señal de control
 control = proporcional + integral +derivativo;

 % Saturación:
 if control>=5 control=5; end
 if control<=0 control=0; end

 % Actualizar el error previo
 error_previo = error;
```

Figura 4-8. Implementación del controlador PID

4.4.2 Ajuste del controlador

Se ajusta el controlador para regular la altura del tanque inferior izquierdo (BI), de manera que siga una referencia de 12.5 cm. Empleando sólo el término proporcional (Figura 4-9) se observa una acción muy agresiva sobre los caudales, que provoca oscilaciones en el sistema. Para corregirlo se añaden los términos Integral y Derivativo, partiendo de un ajuste conservador (Figura 4-10). Finalmente, se ajustan más agresivamente los parámetros para conseguir una respuesta que minimice el tiempo de establecimiento, como la mostrada en la figura 4-11.

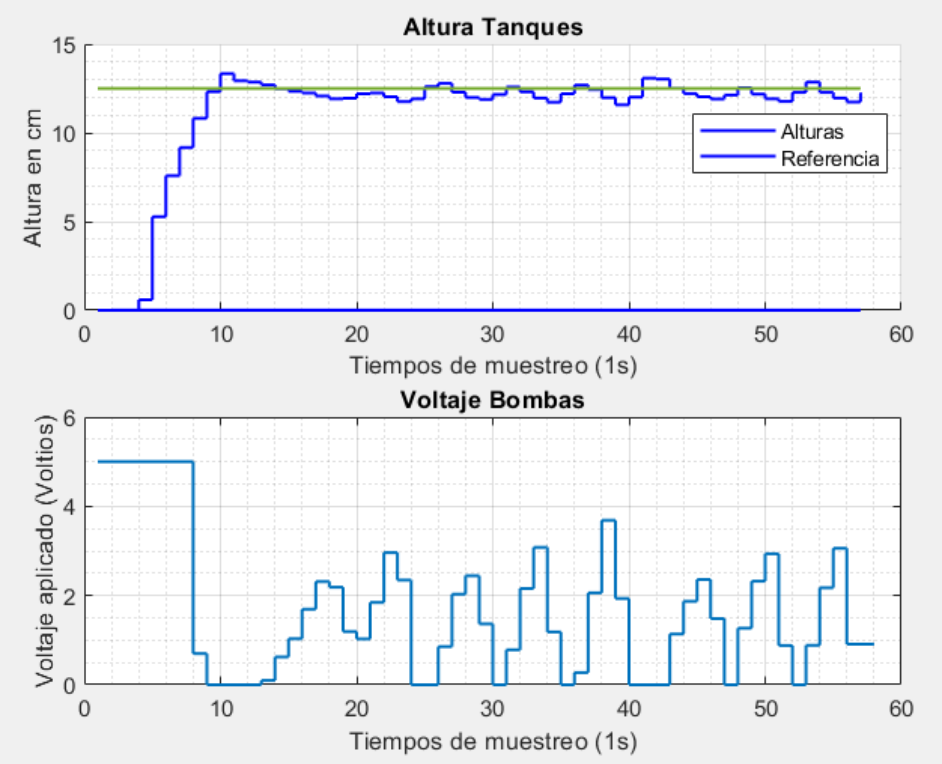


Figura 4-9. Controlador Proporcional (Kp=4)

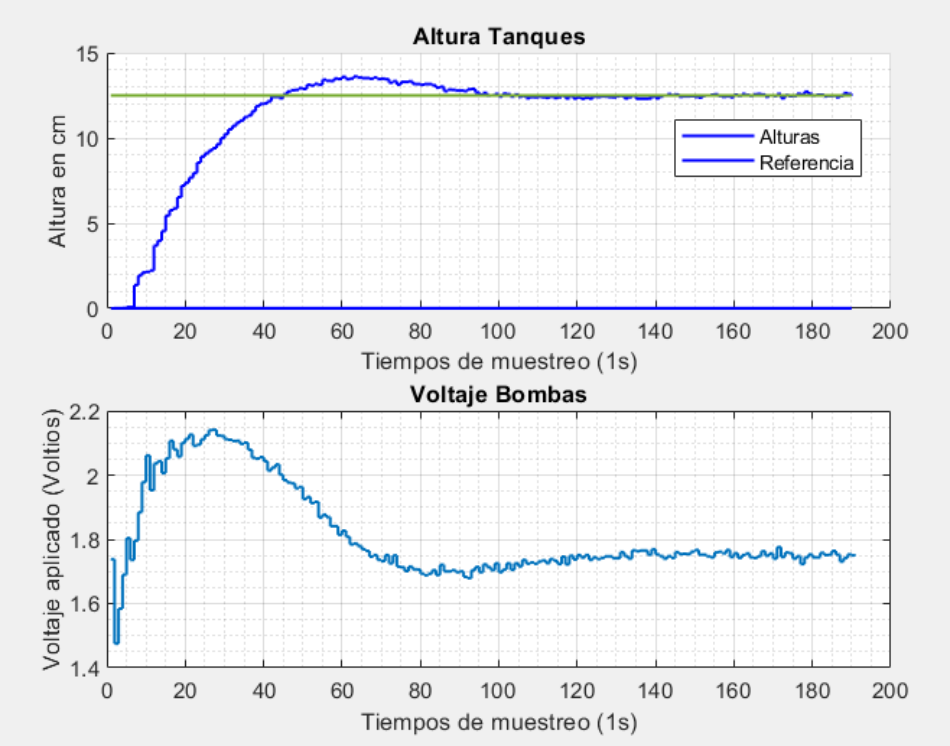


Figura 4-10. Controlador PID (Kp = 0.1 Ki = 0.009 Kd = 0.03)

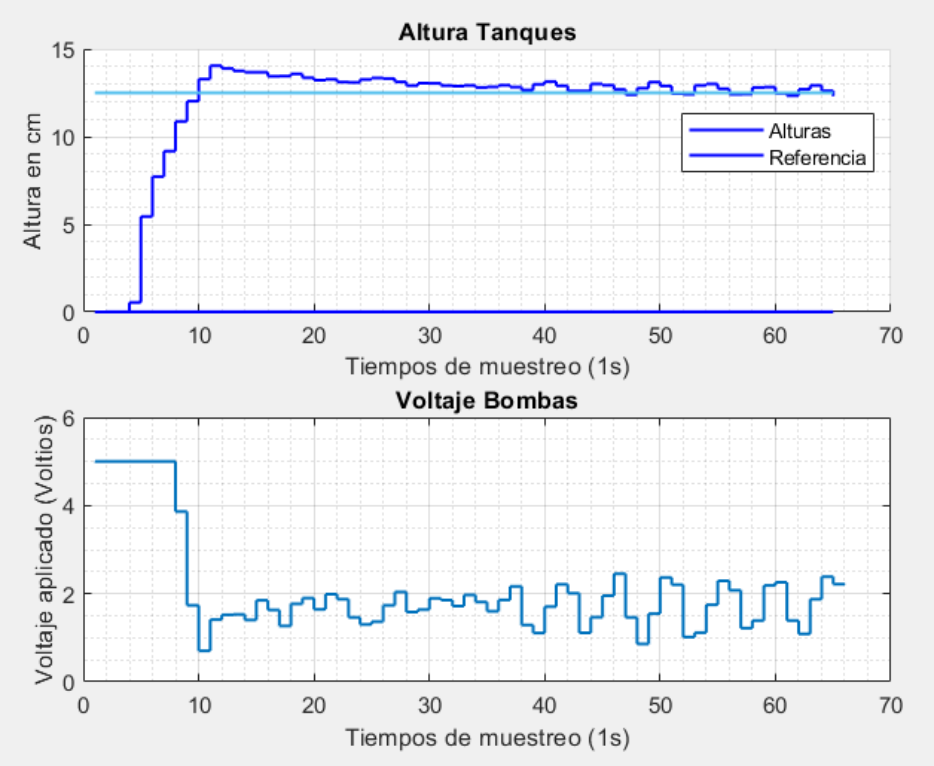


Figura 4-11. Controlador PID (Kp = 1.6 Ki = 0.07 Kd = 0.6)

5 CONCLUSIONES Y TRABAJOS FUTUROS

Durante la realización de este trabajo nos hemos familiarizado con los distintos softwares (SolidEdge, Unity3D, VisualStudio, Matlab), métodos experimentales y técnicas que se emplean en el desarrollo de un gemelo digital, buscando soluciones para los problemas prácticos que han ido surgiendo en el proceso.

En concreto, descubrir la programación orientada a objetos ha resultado de gran interés por suplir una carencia en mis conocimiento de programación.

Por otro lado, se ha realizado un trabajo de investigación en lo relativo a la simulación de fluidos en Unity, campo novedoso que puede tener aplicación en nuevas líneas de investigación del Departamento de Sistemas y Automática.

Todo ello ha supuesto un aprendizaje bastante diverso en técnicas y metodologías de trabajo que pueden ser de interés en el desarrollo de mi carrera profesional.

En cuanto a los resultados obtenidos, el balance es bastante positivo, ya que en la medida de lo posible, y teniendo en cuenta las limitaciones de la simulación de fluidos, se han alcanzado los objetivos de implementar un gemelo digital de la planta ajustado al comportamiento dinámico real, plenamente funcional y conectado satisfactoriamente con el equipo real.

En lo relativo a trabajos futuros, la planta virtual, aunque ya funcional, podría seguir desarrollándose para incorporar características más avanzadas de autoajuste:

- Sería viable, haciendo uso de la vinculación con Matlab implementada para la obtención de datos del modelo y la planta original, realizar ensayos de respuesta ante escalón unitario o impulso para identificar la dinámica de ambos sistemas (haciendo uso del System Identification Toolbox), obteniendo sendos modelos de función de transferencia o espacio de estados que podrían compararse para reconciliar las dinámicas.

  En base a esto, podrían recalcularse los parámetros que rigen el comportamiento de la planta virtual (umbrales de presión, pendientes Voltaje-Caudal, fricción en la descarga, etc), para cada sesión de experimentación.

- Otro punto a desarrollar sería la integración del código de conexión en un bloque de Simulink, que permita ejecutar la simulación en paralelo con el experimento real, lanzando el proceso desde el mismo equipo. Esto permitiría realizar las funciones de autoajuste en tiempo real, ya que podrían compararse las respuestas a medida que van ocurriendo.

6 ANEXOS

6.1 Códigos en C#

6.1.1 Script 'LeeSensores'

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


public class LeeSensores : MonoBehaviour
{
    public Actuadores actuadores;
    // Asocio los sensores con su variable

    public ZibraLiquidDetector sensorai;
    public ZibraLiquidDetector sensorbi;
    public ZibraLiquidDetector sensorad;
    public ZibraLiquidDetector sensorbd;

    public float alturaAI;
    public float alturaBI;
    public float alturaAD;
    public float alturaBD;

    private float alturasfAI;
    private float alturasfBI;
    private float alturasfAD;
    private float alturasfBD;

    private float alturaAIr;
    private float alturaBIr;
    private float alturaADr;
    private float alturaBDr;

    private float alfa;

    // Escalado para sensores
    private float escalasens;

    private float tiempoinicio;

    // Start is called before the first frame update
    void Start()
    {
        //escalasens = 1000; // se toma como 25 cm el maximo del tanque (resolucion
de fluido 200)
        //escalasens = 16000/25; // 16000/f=25 (para res=160)
        //escalasens = 4500 / 25; // para res =130 y líneas puestas

        escalasens = 25f/(10.91f - 9.09f); // para res =130 y líneas puestas

        actuadores = FindObjectOfType<Actuadores>();

        alfa = 0.1f; // constante del filtro
```

```
    }

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

    {
        // Cuenta de Tiempo Transcurrido
        if (Time.frameCount == 1) {tiempoinicio = Time.time;}


        // Lectura Sensores
        alturasfAI   =   (sensorai.BoundingBoxMax.y-   sensorai.BoundingBoxMin.y)
*escalasens;
        alturasfAD  =  (sensorad.BoundingBoxMax.y  -  sensorad.BoundingBoxMin.y)  *
escalasens;
        alturasfBI  =  (sensorbi.BoundingBoxMax.y  -  sensorbi.BoundingBoxMin.y)  *
escalasens;
        alturasfBD  =  (sensorbd.BoundingBoxMax.y  -  sensorbd.BoundingBoxMin.y)  *
escalasens;

        // Filtrado lecturas:
        alturaAI = (alturasfAI+0.01f) * alfa + alturaAI * (1 - alfa);
        alturaBI = alturasfBI * alfa + alturaBI * (1 - alfa);
        alturaAD = alturasfAD * alfa + alturaAD * (1 - alfa);
        alturaBD = alturasfBD * alfa + alturaBD * (1 - alfa);


        // Calcula el tiempo transcurrido en cada cuadro
        float tiempoTranscurrido = Time.time - tiempoinicio;
        if (alturaBI >= 24.9f && alturaBI <= 25.1f ) { Debug.Log("Tiempo BI: " +
tiempoTranscurrido); }
        if (alturaAI >= 24.9f && alturaAI <= 25.1f) { Debug.Log("Tiempo AI: " +
tiempoTranscurrido); }
        if (alturaBD >= 24.9f && alturaBD <= 25.1f) { Debug.Log("Tiempo BD: " +
tiempoTranscurrido); }
        if (alturaAD >= 24.9f && alturaAD <= 25.1f) { Debug.Log("Tiempo AD: " +
tiempoTranscurrido); }

    }
}
```

6.1.2 Script ‘Actuadores’

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using com.zibra.liquid.Manipulators;
public class Actuadores : MonoBehaviour
{
   // Emisores Rama Izquierda
   public ZibraLiquidEmitter GrifoI1;
   public ZibraLiquidEmitter GrifoI2;
   public ZibraLiquidEmitter GrifoI3;

   // Emisores Rama Derecha
   public ZibraLiquidEmitter GrifoD1;
   public ZibraLiquidEmitter GrifoD2;
   public ZibraLiquidEmitter GrifoD3;

   // Tapones descarga     T descarga planta rreal=2.21.45 con tapón de reduccion // T descarga gemelo digital=2.23.27

   public ZibraLiquidCollider DI1;
   public ZibraLiquidCollider DI2;
   public ZibraLiquidCollider DI2_1;

   public ZibraLiquidCollider DD1;
   public ZibraLiquidCollider DD2;
   public ZibraLiquidCollider DD2_1;
   public ZibraLiquidCollider Llavecentral;

   // Conexion horizontal;
   public ZibraLiquidCollider TD;
   public ZibraLiquidCollider TI;

   // Sumideros rebose
   public ZibraLiquidVoid ReboseBI;
```

```
public ZibraLiquidVoid ReboseBD;

// Voltajes de las bombas (0-5 V)
public float BombaI;
public float BombaD;

// Llaves de configuración: //
public bool LlaveI1;
public bool LlaveI2;
public bool LlaveI3;

public bool LlaveD1;
public bool LlaveD2;
public bool LlaveD3;

// Tapones de descarga
public bool TDI1;
public bool TDI2;
public bool TDD1;
public bool TDD2;
public bool LlaveCentral;


// Caudales de cada grifo
private float CaudalI1;
private float CaudalI2;
private float CaudalI3;

private float CaudalD1;
private float CaudalD2;
private float CaudalD3;


//Caudales máximos a 5V:
private float qmax;

private float qmaxI1;
```

```
    private float qmaxI2;
    private float qmaxI3;

    private float qmaxD1;
    private float qmaxD2;
    private float qmaxD3;

    // Umbrales de voltaje para cada grifo
    private float UmbI1;
    private float UmbI2;
    private float UmbI3;

    private float UmbD1;
    private float UmbD2;
    private float UmbD3;



    // Pendientes Caudal-Voltaje:
    private float pendI1;
    private float pendI2;
    private float pendI3;

    private float pendD1;
    private float pendD2;
    private float pendD3;

    private float escalactuadores;



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


        qmax = 0.028f;
        escalactuadores = 5 / qmax; // Se ajusta para fijar el caudal máximo a 5V (0.028 VolPerSimtime
- 18.75s hasta 25cm)


    }
```

```
  void Update()
  {

    // Saturación Bombas:
    if (BombaI > 5) { BombaI = 5f; }
      if (BombaI <= 0) { BombaI = 0f; }

      if (BombaD > 5) { BombaD = 5f; }
      if (BombaD <= 0) { BombaD = 0f; }



    // Activacion Colliders Tapones Descarga

    //Boton ---------- Acción//
    if (LlaveCentral)
    {
      DD2.enabled = false;
      DI2.enabled = false;

      Llavecentral.enabled = true;
      if (TDD1) { DD1.enabled = true; } else { DD1.enabled = false; };
      if (TDD2) { DD2_1.enabled = true; } else { DD2_1.enabled = false; };
      if (TDI1) { DI1.enabled = true; } else { DI1.enabled = false; };
      if (TDI2) { DI2_1.enabled = true; } else { DI2_1.enabled = false; }; // para permitir la descarga
cruzada

    }

    if (!LlaveCentral)  // configuraciones cruzadas
    {
      DD2_1.enabled = false;
      DI2_1.enabled = false;

      Llavecentral.enabled = false;
      if (TDD1) { DD1.enabled = true; } else { DD1.enabled = false; };
      if (TDD2 ) { DD2.enabled = true; ReboseBI.enabled = false; TI.enabled = false; } else {
DD2.enabled = false; TI.enabled = true; ReboseBI.enabled = true; }; // para permitir la descarga
cruzada
```

```
        if (TDI1) { DI1.enabled = true; } else { DI1.enabled = false; };
        if (TDI2 ) { DI2.enabled = true; TD.enabled = false; ReboseBD.enabled = false; } else { DI2.enabled = false; TD.enabled = true; ReboseBD.enabled = true; }; // para permitir la descarga cruzada

        if(TDD2 && TDI2) { Llavecentral.enabled = true; DD2.enabled = true; TI.enabled = true; ReboseBI.enabled = true; DI2.enabled = true; TD.enabled = true; ReboseBD.enabled = true; }
    }

    // Activación de grifos por llaves de paso

    if (LlaveD1) { GrifoD1.enabled = true; } else { GrifoD1.enabled = false; };
    if (LlaveD2) { GrifoD2.enabled = true; } else { GrifoD2.enabled = false; };
    if (LlaveD3) { GrifoD3.enabled = true; } else { GrifoD3.enabled = false; };

    if (LlaveI1) { GrifoI1.enabled = true; } else { GrifoI1.enabled = false; };
    if (LlaveI2) { GrifoI2.enabled = true; } else { GrifoI2.enabled = false; };
    if (LlaveI3) { GrifoI3.enabled = true; } else { GrifoI3.enabled = false; };



    ////////////////////////////// CALCULO CAUDALES IZQUIERDOS:  Calculamos el caudal de cada emisor en función del Voltaje de la bomba ////////////////////////////////
    // Casuística de los grifos (falta por medir umbrales antes de copiar al lado derecho)

    if (LlaveI1 && !LlaveI2 && !LlaveI3)
    {
        // Actualizaciones de parámetros en funcion de cada casuistica de llaves

        // Caudales máx que alcanzan los grifos
        qmaxI1 = qmax;
        qmaxI2 = 0f;
        qmaxI3 = 0f;

        // Umbrales para el caso actual de grifos

        UmbI1 = 1.15f;
        UmbI2 = 2.5f;
```

```
        UmbI3 = 3.5f;


        // Pendientes curva Q-V:
        pendI1 = (qmaxI1) / (5f - UmbI1);       // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
        pendI2 = (qmaxI2) / (5f - UmbI2); ;
        pendI3 = (qmaxI3) / (5f - UmbI3);


        if (BombaI < UmbI1)
        {
           // No hay caudal en ningún emisor
           CaudalI1 = 0.0f;
           CaudalI2 = 0.0f;
           CaudalI3 = 0.0f;
        }
        else if (BombaI < UmbI2)
        {
           // Empieza a salir en el emisor 1
           CaudalI1 = pendI1 * (BombaI - UmbI1); // aprox como recta
           CaudalI2 = 0.0f;
           CaudalI3 = 0.0f;
        }
        else if (BombaI < UmbI3)
        {
           // Hay caudal en el emisor 1 y el emisor 2
           CaudalI1 = pendI1 * (BombaI - UmbI1);
           CaudalI2 = pendI2 * (BombaI - UmbI2);
           CaudalI3 = 0.0f;
        }
        else if (BombaI <= 5f)
        {
           // Hay caudal en los tres emisores
           CaudalI1 = pendI1 * (BombaI - UmbI1);
           CaudalI2 = pendI2 * (BombaI - UmbI2);
           CaudalI3 = pendI3 * (BombaI - UmbI3);
        }


    } // Situación de maximo caudal
```

```
if (LlaveI1 && LlaveI2 && !LlaveI3)  // Sólo vertical
{
   // Actualizaciones de parámetros en funcion de cada casuistica de llaves

   // Caudales máx que alcanzan los grifos
   qmaxI1 = qmax * 0.37f;
   qmaxI2 = qmaxI1 * 0.57f;
   qmaxI3 = 0;

   // Umbrales para el caso actual de grifos

   UmbI1 = 1.15f;
   UmbI2 = 2.6f;
   UmbI3 = 3.5f;

   // Pendientes curva Q-V:
   pendI1 = (qmaxI1) / (5f - UmbI1);      // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
   pendI2 = (qmaxI2) / (5f - UmbI2); ;
   pendI3 = (qmaxI3) / (5f - UmbI3);

   if (BombaI < UmbI1)
   {
      // No hay caudal en ningún emisor
      CaudalI1 = 0.0f;
      CaudalI2 = 0.0f;
      CaudalI3 = 0.0f;
   }
   else if (BombaI < UmbI2)
   {
      // Empieza a salir en el emisor 1
      CaudalI1 = pendI1 * (BombaI - UmbI1); // aprox como recta
      CaudalI2 = 0.0f;
      CaudalI3 = 0.0f;
   }
   else if (BombaI < UmbI3)
   {
      // Hay caudal en el emisor 1 y el emisor 2
```

```
            CaudalI1 = pendI1 * (BombaI - UmbI1);
            CaudalI2 = pendI2 * (BombaI - UmbI2);
            CaudalI3 = 0.0f;
        }
        else if (BombaI <= 5f)
        {
            // Hay caudal en los tres emisores
            CaudalI1 = pendI1 * (BombaI - UmbI1);
            CaudalI2 = pendI2 * (BombaI - UmbI2);
            CaudalI3 = pendI3 * (BombaI - UmbI3);
        }
    }  // Situación  vertical
    if (LlaveI1 && !LlaveI2 && LlaveI3)  // Sólo vertical
    {
        // Actualizaciones de parámetros en funcion de cada casuistica de llaves

        // Caudales máx que alcanzan los grifos
        qmaxI1 = qmax * 0.41f;
        qmaxI2 = 0;
        qmaxI3 = qmaxI1 * 0.57f;

        // Umbrales para el caso actual de grifos

        UmbI1 = 1.15f;
        UmbI2 = 2.5f;
        UmbI3 = 2.6f;

        // Pendientes curva Q-V:
        pendI1 = (qmaxI1) / (5f - UmbI1);      // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
        pendI2 = (qmaxI2) / (5f - UmbI2); ;
        pendI3 = (qmaxI3) / (5f - UmbI3);

        if (BombaI < UmbI1)
        {
            // No hay caudal en ningún emisor
            CaudalI1 = 0.0f;
            CaudalI2 = 0.0f;
```

```
        CaudalI3 = 0.0f;
    }
    else if (BombaI < UmbI2)
    {
        // Empieza a salir en el emisor 1
        CaudalI1 = pendI1 * (BombaI - UmbI1); // aprox como recta
        CaudalI2 = 0.0f;
        CaudalI3 = 0.0f;
    }
    else if (BombaI < UmbI3)
    {
        // Hay caudal en el emisor 1 y el emisor 2
        CaudalI1 = pendI1 * (BombaI - UmbI1);
        CaudalI2 = pendI2 * (BombaI - UmbI2);
        CaudalI3 = 0.0f;
    }
    else if (BombaI <= 5f)
    {
        // Hay caudal en los tres emisores
        CaudalI1 = pendI1 * (BombaI - UmbI1);
        CaudalI2 = pendI2 * (BombaI - UmbI2);
        CaudalI3 = pendI3 * (BombaI - UmbI3);
    }
} // Situación  cruzada
if (LlaveI1 && LlaveI2 && LlaveI3)
{
    // Actualizaciones de parámetros en funcion de cada casuistica de llaves

    // Caudales máx que alcanzan los grifos
    qmaxI1 = qmax * 0.47f;
    qmaxI2 = qmaxI1 * 0.42f;
    qmaxI3 = qmaxI1 * 0.42f;

    // Umbrales para el caso actual de grifos

    UmbI1 = 1.15f;
    UmbI2 = 2.4f;
```

```
        UmbI3 = 3.25f;


        // Pendientes curva Q-V:
        pendI1 = (qmaxI1) / (5f - UmbI1);       // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
        pendI2 = (qmaxI2) / (5f - UmbI2); ;
        pendI3 = (qmaxI3) / (5f - UmbI3);


        if (BombaI < UmbI1)
        {
           // No hay caudal en ningún emisor
           CaudalI1 = 0.0f;
           CaudalI2 = 0.0f;
           CaudalI3 = 0.0f;
        }
        else if (BombaI < UmbI2)
        {
           // Empieza a salir en el emisor 1
           CaudalI1 = pendI1 * (BombaI - UmbI1); // aprox como recta
           CaudalI2 = 0.0f;
           CaudalI3 = 0.0f;
        }
        else if (BombaI < UmbI3)
        {
           // Hay caudal en el emisor 1 y el emisor 2
           CaudalI1 = pendI1 * (BombaI - UmbI1);
           CaudalI2 = pendI2 * (BombaI - UmbI2);
           CaudalI3 = 0.0f;
        }
        else if (BombaI <= 5f)
        {
           // Hay caudal en los tres emisores
           CaudalI1 = pendI1 * (BombaI - UmbI1);
           CaudalI2 = pendI2 * (BombaI - UmbI2);
           CaudalI3 = pendI3 * (BombaI - UmbI3);
        }
     }  // situacion de minimo caudal
     if (!LlaveI1 && LlaveI2 && LlaveI3)
```

```
    {
        // Actualizaciones de parámetros en funcion de cada casuistica de llaves

        // Caudales máx que alcanzan los grifos
        qmaxI1 = 0;
        qmaxI2 = qmax * 0.4f;
        qmaxI3 = qmax * 0.41f;

        // Umbrales para el caso actual de grifos

        UmbI1 = 1f;
        UmbI2 = 1.5f;
        UmbI3 = 2f;

        // Pendientes curva Q-V:
        pendI1 = (qmaxI1) / (5f - UmbI1);       // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
        pendI2 = (qmaxI2) / (5f - UmbI2); ;
        pendI3 = (qmaxI3) / (5f - UmbI3);

        if (BombaI < UmbI1)
        {
            // No hay caudal en ningún emisor
            CaudalI1 = 0.0f;
            CaudalI2 = 0.0f;
            CaudalI3 = 0.0f;
        }
        else if (BombaI < UmbI2)
        {
            // Empieza a salir en el emisor 1
            CaudalI1 = pendI1 * (BombaI - UmbI1); // aprox como recta
            CaudalI2 = 0.0f;
            CaudalI3 = 0.0f;
        }
        else if (BombaI < UmbI3)
        {
            // Hay caudal en el emisor 1 y el emisor 2
            CaudalI1 = pendI1 * (BombaI - UmbI1);
```

```
            CaudalI2 = pendI2 * (BombaI - UmbI2);
            CaudalI3 = 0.0f;
        }
        else if (BombaI <= 5f)
        {
            // Hay caudal en los tres emisores
            CaudalI1 = pendI1 * (BombaI - UmbI1);
            CaudalI2 = pendI2 * (BombaI - UmbI2);
            CaudalI3 = pendI3 * (BombaI - UmbI3);
        }
    }// Situación horizontal
    if (!LlaveI1 && !LlaveI2 && LlaveI3)
    {
        // Actualizaciones de parámetros en funcion de cada casuistica de llaves

        // Caudales máx que alcanzan los grifos
        qmaxI1 = 0;
        qmaxI2 = qmax * 0.4f;
        qmaxI3 = qmax *0.53f ;

        // Umbrales para el caso actual de grifos

        UmbI1 = 1f;
        UmbI2 = 1.5f;
        UmbI3 = 1.6f;

        // Pendientes curva Q-V:
        pendI1 = (qmaxI1) / (5f - UmbI1);      // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
        pendI2 = (qmaxI2) / (5f - UmbI2); ;
        pendI3 = (qmaxI3) / (5f - UmbI3);

        if (BombaI < UmbI1)
        {
            // No hay caudal en ningún emisor
            CaudalI1 = 0.0f;
            CaudalI2 = 0.0f;
            CaudalI3 = 0.0f;
```

```
    }
    else if (BombaI < UmbI2)
    {
        // Empieza a salir en el emisor 1
        CaudalI1 = pendI1 * (BombaI - UmbI1); // aprox como recta
        CaudalI2 = 0.0f;
        CaudalI3 = 0.0f;
    }
    else if (BombaI < UmbI3)
    {
        // Hay caudal en el emisor 1 y el emisor 2
        CaudalI1 = pendI1 * (BombaI - UmbI1);
        CaudalI2 = pendI2 * (BombaI - UmbI2);
        CaudalI3 = 0.0f;
    }
    else if (BombaI <= 5f)
    {
        // Hay caudal en los tres emisores
        CaudalI1 = pendI1 * (BombaI - UmbI1);
        CaudalI2 = pendI2 * (BombaI - UmbI2);
        CaudalI3 = pendI3 * (BombaI - UmbI3);
    }
}// Situación solo g3 abierto
if (!LlaveI1 && LlaveI2 && !LlaveI3)
{
    // Actualizaciones de parámetros en funcion de cada casuistica de llaves

    // Caudales máx que alcanzan los grifos
    qmaxI1 = 0;
    qmaxI2 = qmax * 0.4f;
    qmaxI3 = qmax * 0.57f;

    // Umbrales para el caso actual de grifos

    UmbI1 = 1f;
    UmbI2 = 1.6f;
    UmbI3 = 2f;
```

```
        // Pendientes curva Q-V:
        pendI1 = (qmaxI1) / (5f - UmbI1);       // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
        pendI2 = (qmaxI2) / (5f - UmbI2); ;
        pendI3 = (qmaxI3) / (5f - UmbI3);


        if (BombaI < UmbI1)
        {
           // No hay caudal en ningún emisor
           CaudalI1 = 0.0f;
           CaudalI2 = 0.0f;
           CaudalI3 = 0.0f;
        }
        else if (BombaI < UmbI2)
        {
           // Empieza a salir en el emisor 1
           CaudalI1 = pendI1 * (BombaI - UmbI1); // aprox como recta
           CaudalI2 = 0.0f;
           CaudalI3 = 0.0f;
        }
        else if (BombaI < UmbI3)
        {
           // Hay caudal en el emisor 1 y el emisor 2
           CaudalI1 = pendI1 * (BombaI - UmbI1);
           CaudalI2 = pendI2 * (BombaI - UmbI2);
           CaudalI3 = 0.0f;
        }
        else if (BombaI <= 5f)
        {
           // Hay caudal en los tres emisores
           CaudalI1 = pendI1 * (BombaI - UmbI1);
           CaudalI2 = pendI2 * (BombaI - UmbI2);
           CaudalI3 = pendI3 * (BombaI - UmbI3);
        }
      }// Situación con solo g2 abierto
```

```
    ////////////////////////// CALCULO CAUDALES DERECHOS:  Calculamos el caudal de cada emisor en función del Voltaje de la bomba ////////////////////////////7777
    if (LlaveD1 && !LlaveD2 && !LlaveD3)
    {
      // Actualizaciones de parámetros en funcion de cada casuistica de llaves

      // Caudales máx que alcanzan los grifos
      qmaxD1 = qmax;
      qmaxD2 = 0f;
      qmaxD3 = 0f;

      // Umbrales para el caso actual de grifos

      UmbD1 = 1.15f;
      UmbD2 = 2.5f;
      UmbD3 = 3.5f;

      // Pendientes curva Q-V:
      pendD1 = (qmaxD1) / (5f - UmbD1);     // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
      pendD2 = (qmaxD2) / (5f - UmbD2); ;
      pendD3 = (qmaxD3) / (5f - UmbD3);

      if (BombaD < UmbD1)
      {
        // No hay caudal en ningún emisor
        CaudalD1 = 0.0f;
        CaudalD2 = 0.0f;
        CaudalD3 = 0.0f;
      }
      else if (BombaD < UmbD2)
      {
        // Empieza a salir en el emisor 1
        CaudalD1 = pendD1 * (BombaD - UmbD1); // aprox como recta
        CaudalD2 = 0.0f;
        CaudalD3 = 0.0f;
      }
      else if (BombaD < UmbD3)
```

```
        {
            // Hay caudal en el emisor 1 y el emisor 2
            CaudalD1 = pendD1 * (BombaD - UmbD1);
            CaudalD2 = pendD2 * (BombaD - UmbD2);
            CaudalD3 = 0.0f;
        }
        else if (BombaD <= 5f)
        {
            // Hay caudal en los tres emisores
            CaudalD1 = pendD1 * (BombaD - UmbD1);
            CaudalD2 = pendD2 * (BombaD - UmbD2);
            CaudalD3 = pendD3 * (BombaD - UmbD3);
        }

    } // Situación de maximo caudal
    if (LlaveD1 && LlaveD2 && !LlaveD3)  // Sólo vertical
    {
        // Actualizaciones de parámetros en funcion de cada casuistica de llaves

        // Caudales máx que alcanzan los grifos
        qmaxD1 = qmax * 0.37f;
        qmaxD2 = qmaxD1 * 0.57f;
        qmaxD3 = 0;

        // Umbrales para el caso actual de grifos

        UmbD1 = 1.15f;
        UmbD2 = 2.6f;
        UmbD3 = 3.5f;

        // Pendientes curva Q-V:
        pendD1 = (qmaxD1) / (5f - UmbD1);     // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
        pendD2 = (qmaxD2) / (5f - UmbD2); ;
        pendD3 = (qmaxD3) / (5f - UmbD3);

        if (BombaD < UmbD1)
        {
```

```
        // No hay caudal en ningún emisor
        CaudalD1 = 0.0f;
        CaudalD2 = 0.0f;
        CaudalD3 = 0.0f;
    }
    else if (BombaD < UmbD2)
    {
        // Empieza a salir en el emisor 1
        CaudalD1 = pendD1 * (BombaD - UmbD1); // aprox como recta
        CaudalD2 = 0.0f;
        CaudalD3 = 0.0f;
    }
    else if (BombaD < UmbD3)
    {
        // Hay caudal en el emisor 1 y el emisor 2
        CaudalD1 = pendD1 * (BombaD - UmbD1);
        CaudalD2 = pendD2 * (BombaD - UmbD2);
        CaudalD3 = 0.0f;
    }
    else if (BombaD <= 5f)
    {
        // Hay caudal en los tres emisores
        CaudalD1 = pendD1 * (BombaD - UmbD1);
        CaudalD2 = pendD2 * (BombaD - UmbD2);
        CaudalD3 = pendD3 * (BombaD - UmbD3);
    }
} // Situación  vertical
if (LlaveD1 && !LlaveD2 && LlaveD3)  // Sólo vertical
{
    // Actualizaciones de parámetros en funcion de cada casuistica de llaves

    // Caudales máx que alcanzan los grifos
    qmaxD1 = qmax * 0.41f;
    qmaxD2 = 0;
    qmaxD3 = qmaxD1 * 0.57f;

    // Umbrales para el caso actual de grifos
```

```
        UmbD1 = 1.15f;
        UmbD2 = 2.5f;
        UmbD3 = 2.6f;

        // Pendientes curva Q-V:
        pendD1 = (qmaxD1) / (5f - UmbD1);     // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
        pendD2 = (qmaxD2) / (5f - UmbD2); ;
        pendD3 = (qmaxD3) / (5f - UmbD3);

        if (BombaD < UmbD1)
        {
           // No hay caudal en ningún emisor
           CaudalD1 = 0.0f;
           CaudalD2 = 0.0f;
           CaudalD3 = 0.0f;
        }
        else if (BombaD < UmbD2)
        {
           // Empieza a salir en el emisor 1
           CaudalD1 = pendD1 * (BombaD - UmbD1); // aprox como recta
           CaudalD2 = 0.0f;
           CaudalD3 = 0.0f;
        }
        else if (BombaD < UmbD3)
        {
           // Hay caudal en el emisor 1 y el emisor 2
           CaudalD1 = pendD1 * (BombaD - UmbD1);
           CaudalD2 = pendD2 * (BombaD - UmbD2);
           CaudalD3 = 0.0f;
        }
        else if (BombaD <= 5f)
        {
           // Hay caudal en los tres emisores
           CaudalD1 = pendD1 * (BombaD - UmbD1);
           CaudalD2 = pendD2 * (BombaD - UmbD2);
           CaudalD3 = pendD3 * (BombaD - UmbD3);
```

```
    }
  } // Situación  cruzada
  if (LlaveD1 && LlaveD2 && LlaveD3)
  {
    // Actualizaciones de parámetros en funcion de cada casuistica de llaves

    // Caudales máx que alcanzan los grifos
    qmaxD1 = qmax * 0.47f;
    qmaxD2 = qmaxD1 * 0.42f;
    qmaxD3 = qmaxD1 * 0.42f;

    // Umbrales para el caso actual de grifos

    UmbD1 = 1.15f;
    UmbD2 = 2.4f;
    UmbD3 = 3.25f;

    // Pendientes curva Q-V:
    pendD1 = (qmaxD1) / (5f - UmbD1);     // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
    pendD2 = (qmaxD2) / (5f - UmbD2); ;
    pendD3 = (qmaxD3) / (5f - UmbD3);

    if (BombaD < UmbD1)
    {
      // No hay caudal en ningún emisor
      CaudalD1 = 0.0f;
      CaudalD2 = 0.0f;
      CaudalD3 = 0.0f;
    }
    else if (BombaD < UmbD2)
    {
      // Empieza a salir en el emisor 1
      CaudalD1 = pendD1 * (BombaD - UmbD1); // aprox como recta
      CaudalD2 = 0.0f;
      CaudalD3 = 0.0f;
    }
    else if (BombaD < UmbD3)
```

```
        {
            // Hay caudal en el emisor 1 y el emisor 2
            CaudalD1 = pendD1 * (BombaD - UmbD1);
            CaudalD2 = pendD2 * (BombaD - UmbD2);
            CaudalD3 = 0.0f;
        }
        else if (BombaD <= 5f)
        {
            // Hay caudal en los tres emisores
            CaudalD1 = pendD1 * (BombaD - UmbD1);
            CaudalD2 = pendD2 * (BombaD - UmbD2);
            CaudalD3 = pendD3 * (BombaD - UmbD3);
        }
    }  // situacion de minimo caudal
    if (!LlaveD1 && LlaveD2 && LlaveD3)
    {
        // Actualizaciones de parámetros en funcion de cada casuistica de llaves

        // Caudales máx que alcanzan los grifos
        qmaxD1 = 0;
        qmaxD2 = qmax * 0.4f;
        qmaxD3 = qmax * 0.41f;

        // Umbrales para el caso actual de grifos

        UmbD1 = 1f;
        UmbD2 = 1.5f;
        UmbD3 = 2f;

        // Pendientes curva Q-V:
        pendD1 = (qmaxD1) / (5f - UmbD1);     // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
        pendD2 = (qmaxD2) / (5f - UmbD2); ;
        pendD3 = (qmaxD3) / (5f - UmbD3);

        if (BombaD < UmbD1)
        {
            // No hay caudal en ningún emisor
```

```
        CaudalD1 = 0.0f;
        CaudalD2 = 0.0f;
        CaudalD3 = 0.0f;
    }
    else if (BombaD < UmbD2)
    {
        // Empieza a salir en el emisor 1
        CaudalD1 = pendD1 * (BombaD - UmbD1); // aprox como recta
        CaudalD2 = 0.0f;
        CaudalD3 = 0.0f;
    }
    else if (BombaD < UmbD3)
    {
        // Hay caudal en el emisor 1 y el emisor 2
        CaudalD1 = pendD1 * (BombaD - UmbD1);
        CaudalD2 = pendD2 * (BombaD - UmbD2);
        CaudalD3 = 0.0f;
    }
    else if (BombaD <= 5f)
    {
        // Hay caudal en los tres emisores
        CaudalD1 = pendD1 * (BombaD - UmbD1);
        CaudalD2 = pendD2 * (BombaD - UmbD2);
        CaudalD3 = pendD3 * (BombaD - UmbD3);
    }
}// Situación horizontal
if (!LlaveD1 && !LlaveD2 && LlaveD3)
{
    // Actualizaciones de parámetros en funcion de cada casuistica de llaves

    // Caudales máx que alcanzan los grifos
    qmaxD1 = 0;
    qmaxD2 = qmax * 0.4f;
    qmaxD3 = qmax * 0.53f;

    // Umbrales para el caso actual de grifos
```

```
        UmbD1 = 1f;
        UmbD2 = 1.5f;
        UmbD3 = 1.6f;

        // Pendientes curva Q-V:
        pendD1 = (qmaxD1) / (5f - UmbD1);     // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
        pendD2 = (qmaxD2) / (5f - UmbD2); ;
        pendD3 = (qmaxD3) / (5f - UmbD3);

        if (BombaD < UmbD1)
        {
           // No hay caudal en ningún emisor
           CaudalD1 = 0.0f;
           CaudalD2 = 0.0f;
           CaudalD3 = 0.0f;
        }
        else if (BombaD < UmbD2)
        {
           // Empieza a salir en el emisor 1
           CaudalD1 = pendD1 * (BombaD - UmbD1); // aprox como recta
           CaudalD2 = 0.0f;
           CaudalD3 = 0.0f;
        }
        else if (BombaD < UmbD3)
        {
           // Hay caudal en el emisor 1 y el emisor 2
           CaudalD1 = pendD1 * (BombaD - UmbD1);
           CaudalD2 = pendD2 * (BombaD - UmbD2);
           CaudalD3 = 0.0f;
        }
        else if (BombaD <= 5f)
        {
           // Hay caudal en los tres emisores
           CaudalD1 = pendD1 * (BombaD - UmbD1);
           CaudalD2 = pendD2 * (BombaD - UmbD2);
           CaudalD3 = pendD3 * (BombaD - UmbD3);
        }
```

```
}// Situación solo g3 abierto
if (!LlaveD1 && LlaveD2 && !LlaveD3)
{
   // Actualizaciones de parámetros en funcion de cada casuistica de llaves

   // Caudales máx que alcanzan los grifos
   qmaxD1 = 0;
   qmaxD2 = qmax * 0.4f;
   qmaxD3 = qmax * 0.57f;

   // Umbrales para el caso actual de grifos

   UmbD1 = 1f;
   UmbD2 = 1.6f;
   UmbD3 = 2f;

   // Pendientes curva Q-V:
   pendD1 = (qmaxD1) / (5f - UmbD1);     // para que en cada intervalo suba de 0 a 1 (la fracción de qmax)
   pendD2 = (qmaxD2) / (5f - UmbD2); ;
   pendD3 = (qmaxD3) / (5f - UmbD3);

   if (BombaD < UmbD1)
   {
      // No hay caudal en ningún emisor
      CaudalD1 = 0.0f;
      CaudalD2 = 0.0f;
      CaudalD3 = 0.0f;
   }
   else if (BombaD < UmbD2)
   {
      // Empieza a salir en el emisor 1
      CaudalD1 = pendD1 * (BombaD - UmbD1); // aprox como recta
      CaudalD2 = 0.0f;
      CaudalD3 = 0.0f;
   }
   else if (BombaD < UmbD3)
   {
```

```
            // Hay caudal en el emisor 1 y el emisor 2
            CaudalD1 = pendD1 * (BombaD - UmbD1);
            CaudalD2 = pendD2 * (BombaD - UmbD2);
            CaudalD3 = 0.0f;
        }
        else if (BombaD <= 5f)
        {
            // Hay caudal en los tres emisores
            CaudalD1 = pendD1 * (BombaD - UmbD1);
            CaudalD2 = pendD2 * (BombaD - UmbD2);
            CaudalD3 = pendD3 * (BombaD - UmbD3);
        }
    }// Situación con solo g2 abierto
     ////////////////// ACTUALIZACIÓN CAUDALES//////////////////////////////////////////////////////////
    GrifoI1.VolumePerSimTime = 170 * CaudalI1 / escalactuadores;
        GrifoI2.VolumePerSimTime = 170 * CaudalI2 / escalactuadores;
        GrifoI3.VolumePerSimTime = 170 * CaudalI3 / escalactuadores;

        GrifoD1.VolumePerSimTime = 170 * CaudalD1 / escalactuadores;
        GrifoD2.VolumePerSimTime = 170 * CaudalD2 / escalactuadores;
        GrifoD3.VolumePerSimTime = 170 * CaudalD3 / escalactuadores;

    }

}
```

6.1.3 Script TCPServer:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;

public class TCPServer : MonoBehaviour
{
    private float VI;
    private float VD;
    private TcpListener server;
    public TcpClient client;
    private NetworkStream stream;
    private byte[] buffer;

    // METODO PARA INICIALIZAR EL SERVIDOR TCP:
    public async Task StartServerAsync(string ipAddress, int port)
    {
        IPAddress localAddress = IPAddress.Parse(ipAddress);
        server = new TcpListener(localAddress, port);
        server.Start();
        Debug.Log("Servidor TCP iniciado. Esperando conexiones...");
        client = await server.AcceptTcpClientAsync();
        stream = client.GetStream();
        buffer = new byte[1024];
    }

    // METODO PARA RECIBIR DATOS DEL CLIENTE
    public string RecibeDatos()
    {
        if (client.Connected )
        {
            int bytesRead = stream.Read(buffer, 0, buffer.Length);
            string data = Encoding.ASCII.GetString(buffer, 0, bytesRead);
            return data;
        }
        else
        {
            StopServer();
            return "0 0";
        }
    }
    // METODO PARA ENVIAR DATOS AL CLIENTE
    public void EnviaDatos(string data)
    {
        byte[] buffer = Encoding.ASCII.GetBytes(data);
        stream.Write(buffer, 0, buffer.Length);
    }
    // METODO PARA CERRAR EL SERVER
    public void StopServer()
    {
        stream.Close();
        client.Close();
        server.Stop();
        Debug.Log("Servidor TCP detenido.");
    }
}
```

6.1.4 Script LanzadorTCP:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;
using System.Globalization;

public class LanzadorTCP : MonoBehaviour
{
    public bool enviando;
    public bool Recibiendo;        // Activa o desactiva el recibo de voltajes por
TCP
    public float Tconex;  // frecuencia de actualización de comunicaciones TCP
    public float VI;
    public float VD;
    private string datosrec;
    private LeeSensores leesens;
    private Actuadores actua;
    private TCPServer miserver;

    private async void Start()
    {
        // INICIALIZO EL SERVER
        Tconex = 1f;
        miserver = new TCPServer();
        leesens = GetComponent<LeeSensores>(); // para poder acceder a las alturas
leidas y enviarlas
        actua = GetComponent<Actuadores>(); // para poder acceder a las boombas

        await  miserver.StartServerAsync("127.0.0.1",  5555);  //  Ejemplo:  IP:
127.0.0.1, Puerto: 5555
        miserver.EnviaDatos("Conexion Aceptada");
        Debug.Log("Conexion establecida");

        InvokeRepeating("ConexionTCP", 0.0f, Tconex); // Envío y recepcion de datos
cada Tconex segundos
    }

    // Desde aqui, se están escuchando los datos del cliente con un hilo separado
en 2do plano para no colgar Unity

    private void ConexionTCP()
    {
        // CONVERSIÓN DE LAS LECTURAS A STRING
        string   datosenvio   =   leesens.alturaBI.ToString("F3")   +   "   "   +
leesens.alturaAI.ToString("F3") + " " + leesens.alturaAD.ToString("F3") + " " +
leesens.alturaBD.ToString("F3");
       // ENVÍO DE ALTURAS
        if (enviando) { miserver.EnviaDatos(datosenvio); }

        // GESTIÓN DE LA DESCONEXIÓN (evitar cuelgue de Unity)
        if (!miserver.client.Connected)
        {
            Debug.Log("El cliente está desconectado");
            return ;
        }
```

```
        // RECEPCIÓN DE LOS NUEVOS VOLTAJES DE BOMBA
        else if (Recibiendo && miserver.client.Connected)
        {
            datosrec=miserver.RecibeDatos();
            Debug.Log("Voltaje Bombas Recibido: " + datosrec);
            // EXTRACCIÓN DE LOS DATOS DEL STRING:
            string[] floatStrings = datosrec.Split(" "); // Dividir el string en
substrings al estar separadas por el espacio

            VI                =                Convert.ToSingle(floatStrings[0],
CultureInfo.InvariantCulture.NumberFormat); // Convertir el  substring a float.
Hay que incluir CultureInfo para interpretar el .
            VD                =                Convert.ToSingle(floatStrings[1],
CultureInfo.InvariantCulture.NumberFormat);


            actua.BombaD=VD;
            actua.BombaI=VI;
        }
    }

    private void OnApplicationQuit()
    {
        miserver.StopServer();

    }
}
```

6.2 Códigos en Matlab:

```
%% COMUNICACIÓN CON LA PLANTA VIRTUAL:
clear
clc
close all

try % para el tratamiento de errores

%% Conexion al servidor remoto de Unity como cliente:
Tconex=1;
client = tcpclient("127.0.0.1",5555)
client.ErrorOccurredFcn = @TerminaConex
disp("Peticion de conexión")
pause(1)
confirm=read(client,17,"string");
disp(confirm)
disp('Lecturas sensores:')
disp('AI     BI      AD      BD')

%% Inicializo Arrays para sensores y voltajes bombas:
Alturas=zeros(1,4); %% máximo 5min*60/0.2s

Bombas(:,1)=zeros(1500,1);
Bombas(:,2)=zeros(1500,1);
i=0;
datosnum=[0 0 0 0];
%% Parámetros Cnntrol (PID)
```

```
% Parámetros del controlador PID
Kp = 1.6;    % Ganancia proporcional
Ki = 0.07;  % Ganancia integral
Kd = 0.6;  % Ganancia derivativa

% Parámetros del sistema
referencia = 12.5;  % Altura deseada
error_previo = 0;     % Error previo para el término derivativo
integral = 0;         % Término integral acumulado
control=0;

%% BUCLE DE ENVÍO Y RECEPCIÓN DE DATOS
while(1)
pause(Tconex)  %% Muestreo de la conexión

   %% RECEPCIÓN DE DATOS SENSORES:
   rec= read(client,client.NumBytesAvailable,'string');
   % Conversion de los datos de string a formato double
   if(isstring(rec))
   str = strrep(rec, ',', '.'); % Reemplazar comas por puntos
   datosnum = str2num(str); % Convertir el string en array de números
   if(not(isempty(datosnum))) % cuando la cadena viene superpuesta, st2num
devuelve vacío, así se descartan errores
   datosnum=datosnum(1:4);
   disp(datosnum)
     Alturas=[Alturas;datosnum]; %% Almaceno las medidas: [BI AI AD BD]
   end
   end
   % %% CÁLCULO DE ACCIÓN DE CONTROL: PID

    error = referencia - datosnum(1);  % Se controla la altura del tanque 1 en
este ejemplo

    %Acción Proporcional
    proporcional = Kp * error;

    % Acción Integral con Antiwindup
    if(control<=5 && control>=0)
    integral = integral + Ki * error;
    end

    % Acción derivativa:
    derivativo = Kd * (error - error_previo);

    % Calcular la señal de control
    control = proporcional + integral +derivativo;

    % Saturación:
    if control>=5 control=5; end
    if control<=0 control=0; end

    % Actualizar el error previo
    error_previo = error;

     %% ENVÍO DEL VOLTAJE CALCULADO:
   i=i+1;
   Bombas(i,1)=round(control,3);
   env=string([num2str(Bombas(i,1)) ' ' num2str(Bombas(i,2))]); % Conversión a
string en formato 3 decimales
```

```
   write(client,env)
end



%% MANEJO DEL FINAL DE LA CONEXION
catch ex
    if strcmp(ex.identifier, 'transportlib:transport:invalidConnectionState')
% Manejo del error al pausar la simul Unity
     disp('Conexion Terminada')

    %% Ploteo de resultados
     close all
     subplot(2,1,1)
     axis([0 60 0 25])
      hold on
     stairs(Alturas,'LineWidth',1.2,'Color','b')
     stairs(referencia*ones(length(Alturas)),'LineWidth',1.2,'Color','r')
      legend('Alturas','Referencia','Location','best')
     xlabel('Tiempos de muestreo (1s)')
     ylabel('Altura en cm')
     grid on
     grid minor

     title('Altura Tanques')
     subplot(2,1,2)

     stairs(Bombas(1:i,1),'LineWidth',1.2)
     hold on
     axis([0 60 0 5.1])
     grid on
     grid minor
     xlabel('Tiempos de muestreo (1s)')
     ylabel('Voltaje aplicado (Voltios)')
     title('Voltaje Bombas')

    end
end
```

REFERENCIAS

[1] K.H.Johansson, "The quadruple-tank process: a multivariable laboratory process with an adjustable zero," in IEEE Transactions on Control Systems Technology, vol. 8, no. 3, pp. 456-465, May 2000, doi: 10.1109/87.845876.

[2] Digital Twin Consortium, «Digital twin consortium defines digital twin.,» [En línea]. Available: https://www.digitaltwinconsortium.org/2020/12/digital-twin-consortium-defines-digital-twin.

[3] [En línea]. Available: https://sectormaritimo.es/gemelo-digital-para-plataformas-offshore.

[4] [International Organization for Standardization, (ISO 23247-1:2021) Automation systems and integration — Digital twin framework for manufacturing — Part 1: Overview and general principles. https://www.iso.org/standard/75066.html]

[5] Levis, Diego. Comunicación & Educación. ¿Qué es la realidad virtual? Copyright © Diego Levis-1997/2006. [En Línea] www.diegolevis.com.ar/secciones/Articulos/Que_es_RV.pdf. NoComercial 2.5 Buenos Aires, Argentina.

[6] Alberto Pedro Lorandi Medina, et al. , «Los Laboratorios Virtuales y Laboratorios Remotos en la Enseñanza de la Ingeniería,» Revista Internacional de Educación en Ingeniería, vol. 4, Pp. 24-30. 2011.

[7] [En línea] https://labsland.com/blog/es/2022/10/13/que-es-un-laboratorio-remoto/.

[8] [En línea] «https://assetstore.unity.com/account/assets,»

[9] VirtualMethod, «ObiFluids,» [En línea]. http://obi.virtualmethodstudio.com/.

[10] Zibra AI, «Zibra.AI,» [En línea] https://effects.zibra.ai/.

[11] [En línea]. Available: https://zibra.notion.site/Zibra-Liquids-Unity-documentation-archive-f945f86fb7494e3c8b6f348622ffcb0c.

[12] J. A. G. Seco, El lenguaje de programación C#, 2001. [En línea]. Available: http://di002.edv.uniovi.es/~cueva/investigacion/lineas/lenguajes/lenguajeCsharp.pdf

[13] La última pregunta, «La última pregunta - Introducción a Unity (youtube),» [En línea]. Available: https://www.youtube.com/watch?v=TP_qrtchjYA&list=PLYzVYgj8PHSUtzTtWVO6bt12lBcd0spfY&ab_channel=La%C3%9AltimaPregunta.

[14] J. L. G. Sanchez, Ejemplo de Linealizacion. Sistema de dos tanques, 2015. [En línea]. Available: https://w3.ual.es/personal/joguzman/virtual_tank/Ejemplo_Tanques.pdf

[15] Joaquín Ferruz Melero (Apuntes de Informática Industrial, 2022) – Departamento de Ingeniería de Sistemas y Automática, Universidad de Sevilla.

[16] V. Cerf and R. Kahn, "A Protocol for Packet Network Intercommunication," in IEEE Transactions on Communications, vol. 22, no. 5, pp. 637-648, May 1974, doi: 10.1109/TCOM.1974.1092259.

[17] Varas Chiquito, M., García Plua, J. C., Bustamante Chong, M., & Bustamante Chong, C. (2020). Gemelos digitales y su evolución en la industria. RECIMUNDO, 4(4), 300-308. https://doi.org/10.26820/recimundo/4.(4).noviembre.2020.300-308

[18] [En línea]. Available: https://zibra.notion.site/Zibra-Liquids-Unity-documentation-archive-f945f86fb7494e3c8b6f348622ffcb0c