Trabajo Fin de Máster
Ingeniería Electrónica, Robótica y Automática

Metodología para el desarrollo de gemelos digitales. Aplicación a célula de fabricación flexible.

Autor: Javier Gómez Jiménez
Tutor: Juan Manuel Escaño González
Cotutor: Adolfo J. Sánchez del Pozo Fernández

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

Sevilla, 2021

Proyecto Fin de Máster
Ingeniería Electrónica, Robótica y Automática

Metodología para el desarrollo de gemelos digitales. Aplicación a célula de fabricación flexible.

Autor:
Javier Gómez Jiménez

Tutor:
Juan Manuel Escaño González
Cotutor:
Adolfo J. Sánchez del Pozo Fernández

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Proyecto Fin de Carrera: Metodología para el desarrollo de gemelos digitales. Aplicación a célula de fabricación flexible.

Autor: Javier Gómez Jiménez

Tutor: Juan Manuel Escaño González

Cotutor: Adolfo J. Sánchez del Pozo Fernández

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A todos aquellos que nunca
dejaron de creer en mí,
como tú, Abuela

Agradecimientos

A mis Amigos, porque en estos cinco años no ha habido ningún momento que haya estado solo, pasara lo que pasase. Siempre habéis sabido empujarme, soportarme y no soltarme, aunque yo respondiera como vosotros lo habéis hecho.

A mi profesor y compañero Adolfo J., porque sin ti esto no vería la luz, como tantas cosas que hemos sacado juntos.

A mi tutor Juan Manuel Escaño, por hacer posible mis ideas, por muy difíciles que parecieran. Y, sobre todo, por confiar tanto en mi.

Al proyecto DENiM, por dejarme utilizar parte de mi trabajo en éste.

A mi Familia, porque les debo todo.

Y a Dios, por todo.

Javier Gómez Jiménez

Alumno del Máster en Ingeniería Electrónica, Robótica y Automática

Sevilla, 2021

Resumen

En el presente proyecto se propone una metodología para la creación de un gemelo digital. Esta metodología se definirá de manera totalmente abierta para que pueda usarse para realizar el gemelo digital de cualquier sistema real. Para complementar y verificar la exposición del método se expone una implementación de un gemelo digital realizado utilizando la metodología propuesta, con la configuración de comunicaciones, mediante OPC UA.

Abstract

This project proposes a methodology for the creation of a digital twin. This methodology will be defined in a completely open way so that it can be used to realise the digital twin of any real system. To complement and verify the exposition of the method, an implementation of a digital twin made using the proposed methodology, with the communication configuration, by means of OPC UA, is presented.

Índice

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

El siglo XXI se caracteriza, entre otras cosas, como el siglo de la información. La información es casi una moneda de cambio, en nuestros días. Esto se debe a que se ha aumentado la capacidad de procesamiento de la información. Esta capacidad de procesamiento puede verse frenada por una falta de acceso a la información.

Haciendo un cambio brusco en el hilo de la argumentación, se va a introducir un término que lucirá a lo largo de todo este documento. Si alguna persona disfrutara leyendo la RAE, encontraría una definición para un término polisémico, referido a cosas [1]:

Igual que otra con la que normalmente forma pareja.

El término referido es **Gemelo**, del latín *gemellus*. Se podría decir que la palabra clave en esta definición es la palabra **igual**. Esta última palabra también contiene otra definición en la RAE [2]:

Que tiene las mismas características que otra persona o cosa en algún aspecto o en todos.

¿Qué implicación tiene el término *gemelo* en la importancia de la información? La implicación no es inmediata, pero se puede deducir de la capacidad de acceso a la información. Si se consiguiera obtener un gemelo de aquello de lo cual se quieren obtener datos, tendríamos toda la información de aquello, o al menos de aquellos aspectos en los que son idénticos (cfr. Definición de la palabra igual).

Si se piensa con perspectiva, este "gemelo" podría ser la optimalidad de esa capacidad de acceso a la información, siempre y cuando el gemelo permita replicar todas las características que interesen. Esta es la motivación de este proyecto.

Este "gemelo" del que se está hablando tiene nombre propio: Gemelo Digital.

Dentro de las bastantes definiciones encontradas en internet sobre el término gemelo digital, se ha recalcado la siguiente , dada por la Universidad Internacional de Valencia [3]:

Es una "copia" digital de un elemento físico.
Hoy en día se ha digitalizado el proceso para hacer pruebas.

Aparte de que ayuda al lector a entender por qué la introducción sobre la digitalización, esta definición aporta, de manera breve, una definición precisa en el ámbito lingüístico, pero no en el ingenieril, que es donde pretende moverse este documento. Esto hace necesario que se realice una precisión dentro de esas dos comillas puestas a la palabra copia.

Un modelo de un sistema real es un sistema, con o sin memoria, que imita, con una precisión determinada, al sistema real que representa. Si esta precisión se hace tender hacia el infinito, se obtendría un gemelo digital. Es decir, un gemelo digital es un modelo de un sistema real con una precisión infinita.

Obtener esta precisión infinita, de manera general, es imposible, debido al sumatorio de complejidades que lleva asociado un sistema real. Por tanto, se puede dejar esta definición en el plano teórico. Pero, la cuestión es si existen ciertos fenómenos con una influencia tan pequeña que la exclusión de su modelado no repercuta en

la precisión de aquellos fenómenos que se quiere obtener la información. Este trabajo pretende abrir un cauce para responder afirmativamente a esta cuestión. Si fuera posible, tendríamos un nuevo concepto, más refinado, pero ya en un plano práctico:

Un gemelo digital es un modelo de un sistema real con una precisión suficiente que permita predecir cualquier comportamiento del sistema real con relevancia mínima, en un espacio y tiempo determinado.

La relevancia mínima está referida a aquellas características de las cuales se quiere obtener información.

Esta definición ya es más realista, pero surge la duda de que, al ser muy realista, pueda quedarse en los modelos ya existentes, dado que la precisión no se ha cuantificado. Partiendo de la hipótesis de que es imposible alcanzar una precisión infinita, un gemelo digital debe tener una precisión que tienda al infinito con el trascurso temporal. Esta es la gran diferencia que existe con los modelos tradicionales, o al menos en gran parte de ellos, porque un gemelo digital, o a partir de ahora **DT** por sus siglas en inglés (Digital Twin), va *autoaprendiendo* del sistema real con el trascurso del tiempo. Dado que esta es la gran diferencia, el término de autoaprendizaje aparecerá muchas veces en la presente memoria.

Hay un matiz muy importante a lo que se acaba de nombrar. Cuando se dice que un DT se va perfeccionando con el tiempo se quiere decir que todos aquellos relevantes modelados dentro del DT se van clonando de manera más veraz, pero solo esos y solo si éstos no dependen de factores no modelados dentro del DT. De ahí que en la definición se haya introducido la cola "en un espacio y tiempo determinado".

Para ilustrar este matiz se expondrá un ejemplo: un tornillo bien ajustado de una cinta transportadora probablemente no tenga ninguna influencia, a corto plazo, en el modelado de esa cinta transportadora dentro de un DT. Pero es posible que, a largo plazo, debido a las vibraciones constantes existente en una cinta transportadora, crezca la influencia que pueda tener ese mismo tornillo, pero no colocado correctamente.

Esto implica que el autoaprendizaje es capaz de adaptar el DT a lo largo del tiempo, pero solo en aquellos factores modelados. Aunque esto complique aún más la implementación de un DT en su definición teórica, tampoco es un problema irresoluble, dado que, al aumentar la influencia de factores no modelados, el autoaprendizaje puede asumirlos como cambios en los factores modelados, sin saber que son elementos "distintos". Lo importante es que el DT sea robusto al paso del tiempo, e incluso mejore.

Es notable decir que las definiciones aquí mencionadas sirven de contexto para situar al presente trabajo, pero no como definiciones generales. De hecho, existen multitud de proyectos activos investigando sobre los gemelos digitales.

Uno de estos proyectos es el desarrollado para la empresa Maviva S.A. [4] que consiste en desarrollar un entorno de simulación a alto nivel de una o varias de sus fábricas. Este proyecto tiene una pequeña diferencia con el propuesto en el siguiente trabajo: al generar un gemelo digital de un entorno grande (una fábrica) hay modelos de pequeña escala, como puede ser el de una cinta transportadora, que se modelan con funciones estadísticas, o con modelos simples, ya que lo interesante es el modelo a gran escala. Este proyecto, en cierto sentido, se separa bastante de la definición dicha anteriormente. Sus ventajas son la rapidez de montaje del "gemelo digital", a costa de que deja de serlo. Es parecido a softwares desarrollados con Arena Simulation Software, que se centran más en la gestión de plantas industriales que en su modelado. Tiene su función y lo más probable es lo que se busque en el proyecto mencionado.

Otro ejemplo es el proyecto Miraged [5]. El concepto de gemelo digital que introduce este proyecto sí está más alineado con el explicado. Este proyecto trata de generar modelos avanzados (se desconoce si autoaprenden de la realidad) de procesos industriales para poder realizar predicciones posteriores.

Este proyecto se enmarca en las tareas del proyecto Europeo DENiM [6] el cual busca, a partir de gemelos digitales, entre otras cosas, la supervisión y optimización automática de la energía en cualquier proceso industrial. El proyecto se va a ensayar en cinco plantas reales de varios países.

2 EL MÉTODO

Una vez situado el presente proyecto, como aportación al proyecto DENiM, se está ya en disposición de entender la finalidad del proyecto: realizar una metodología para la creación de gemelos digitales (tal y como se han definido de manera práctica) versátil y escalable. En concreto, es necesario que la metodología permita crear gemelos digitales con las siguientes características:

- Precisión: con tendencia al mínimo error. Necesidad de dotar al DT de la capacidad de autoaprendizaje.
- Escalable: si se modifica el sistema real, no haga falta reprogramar por completo al gemelo.
- Versátil: al cambiar el sistema real del cual se ha hecho el gemelo digital, por otro distinto, no sea necesario cambiar un alto porcentaje de la programación del gemelo.

Una vez dicho las características principales que hay que conseguir, es importante resaltar que las dos últimas se refieren más al cómo hay que hacerlo, mientras que la primera más al qué hay que hacer. Por tanto, la única característica intrínseca al método es la primera.

Una vez llegado a este punto, se define un método con seis pasos abiertos para poder crear un gemelo digital:

1. Arquitectura interna.
2. Modelado visual.
3. Modelado funcional.
4. Autoaprendizaje.
5. Conectividad.
6. Servicios.

En un gemelo digital hay que distinguir dos contextos muy distintos: el primero es el gemelo digital en sí; el segundo son los servicios y consecuencias que se derivan del gemelo. Por tanto, el último paso propuesto no es parte del gemelo digital, pero es necesario para darle utilidad al gemelo.

Estos pasos se pueden realizar de manera desordenada, permitiendo realizar cambios futuros en pasos anteriores, sin tener que empezar desde cero. Aunque depende de esas capacidades de escalabilidad y versatilidad que debe dotárselas el creador que esté implementando el método.

A continuación, se irán explicando cada uno de los pasos del método.

2.1. Arquitectura interna

El primer paso del proceso consiste en la definición de una arquitectura para el DT. Este paso es el más abstracto e importante, dado que marcará la programación de la mayoría de módulos a implementar.

Además, la arquitectura interna es la encargada de proporcionar al DT las características de ambliabilidad y

versatilidad.

Antes de continuar, se define el término módulo, en el contexto de este trabajo, como un programa dotado de la suficiente autonomía para funcionar por sí solo, es decir, que no necesita ningún programa externo para su correcto funcionamiento. Esto no implica que los datos que utilice no sean suministrados por un programa externo. Un módulo es similar a un proceso dentro del campo de la Informática Industrial. Unas de las características más importantes de los módulos es su capacidad de ejecución en paralelo y su aislamiento de otros módulos (si no se ejecutan, o fallan, el sistema sigue en funcionamiento) se deben a esa autonomía concedida.

Una posible clasificación de arquitecturas dentro del mundo del software, es mediante la distribución de tareas entre los distintos programas que componen una aplicación. Así se distinguen tres tipos:

1. Centralizada completa: se compone de un único módulo que realiza todas las tareas de aplicación. Su ventaja fundamental es su fácil programación. Esta ventaja se ve quebrada cuando la aplicación tiene una embergadura tal que hace imposible este tipo de arquitectura. Además, no deja margen para ampliar la aplicación de manera fácil.
2. Descentralizada completa: cada tarea la realiza un módulo distinto. Su ventaja fundamental es su escalabilidad, a cambio de una mayor complejidad de programación.
3. Mixta: se trata de una combinación de las dos anteriores. Suele ser la más corriente.

Para un DT se podrían implementar cualquier tipo de arquitectura de las mencionadas o cualquier otra. Dada las características nombradas anteriormente, se propone un tipo de arquitectura de tipo mixta quasidescentralizada: todas las tareas serán realizadas por varios módulos corriendo en paralelo, y un módulo supervisor que los coordinará. Se nombrará a esta arquitectura como la *arquitectura propuesta*. Un esquema simplificado se encuentra en la figura 2.1.

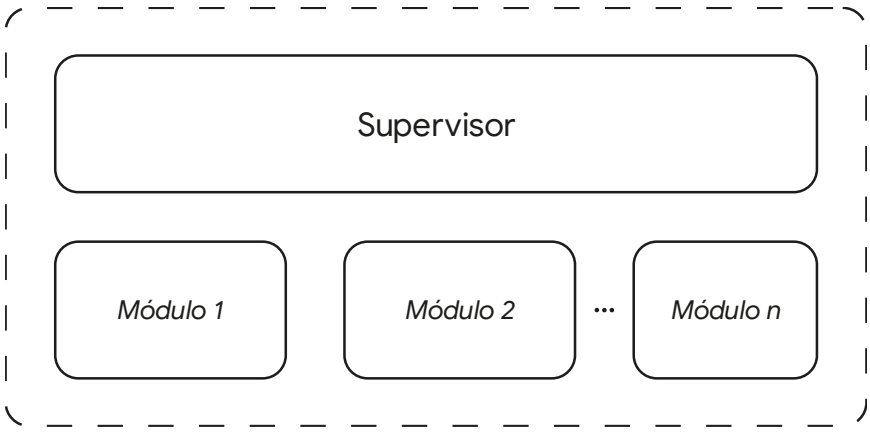


Figura 2.1 - Primer acercamiento a la arquitectura propuesta

Con esta arquitectura se consigue que cuando se requieran más componentes, como por ejemplo un protocolo nuevo de comunicación, se pueda insertar sin modificar ningún componente ya implementado anteriormente, salvo el supervisor. Además, la versatilidad se mejora, dado que para un DT basado en un sistema real distinto, se necesitarán muchos módulos ya implementados, como puede ser el mencionado del protocolo de comunicación.

Es interesante mencionar que realmente la versatilidad viene dada por la cantidad de autoaprendizaje de cada módulo, mientras que la escalabilidad depende de la autonomía dotada a cada módulo. Más adelante se entenderá este detalle.

Antes de continuar definiendo la arquitectura propuesta, volviendo a la definición de un gemelo digital, es importe no perder de vista que el sistema de partida es un sistema real, convirtiéndolo en objeto de multiplicidades de fenómenos dentro del mundo físico. Tener en cuenta esto es imprescindible para definir la arquitectura interna.

Existen multiplicidad de modos para llevar a cabo la implementación de esos fenómenos físicos, desde programar desde cero cada uno de los fenómenos, hasta utilizar módulos ya hechos que los lleven a cabo.

Actualmente los módulos dedicados a implementar los fenómenos físicos reciben el nombre de **motores físicos**. Estos motores proporcionan un entorno digital donde cualquier sólido rígido está bajo el influjo de todas las leyes físicas (o al menos al de las conocidas e implementadas). La utilización de estos motores físicos hace que una parte del gemelo digital, y no pequeña, ya venga implementada. Dadas las facilidades prácticas que suman los motores físicos, se propone incluirlo en la arquitecura nombrada. Para una explicación más detallada [7].

Además, es necesario tener un entorno de ejecución del DT que permita la modularidad requerida por la arquitectura propuesta, es decir, que permita, mediante funciones clave, la ejecución de módulos nuevos sin previo aviso. En la figura 2.2 se puede observar los elementos ya mencionados añadidos a la arquitectura.

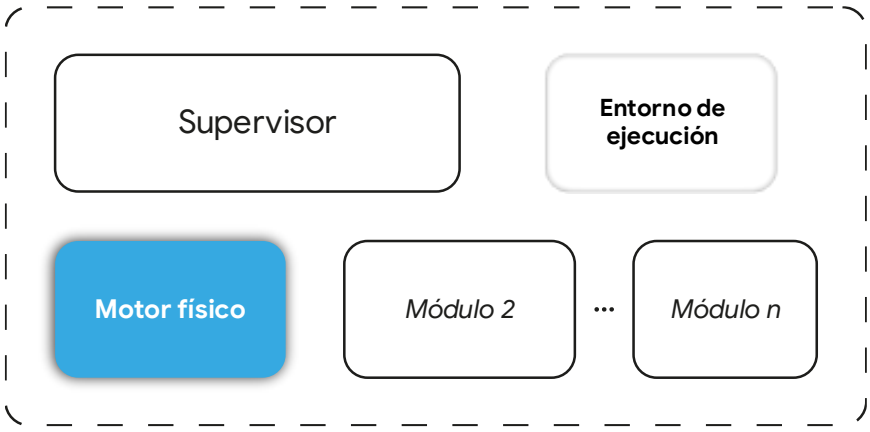


Figura 2.2 - Esquema ampliado con el entorno de ejecución y el motor físico

Al final se tiene un marco de ejecución que permite la ejecución ordenada, llevada a cabo por el supervisor, de diferentes módulos interconectados entre sí, apoyados en el módulo del motor físico (solo aquellos que tengan contacto con los fenómenos reales). La arquitectura se irá ampliando a lo largo de los posteriores pasos del método.

La definición de la arquitectura debe llegar hasta el detalle, dado que la optimización de la ejecución depende de cómo esté diseñada. Más aún cuando la aplicación a ejecutar es un gemelo digital, que como se verá, los recursos requeridos son muy altos.

2.2. Modelo visual

El segundo paso del método consiste en el diseño y creación de un modelo 3d que permita identificar al gemelo digital con el sistema real. Además, puede ayudar a implementar de manera más sencilla los modelos funcionales de cada objeto dentro de nuestro sistema, si se ha dotado a nuestro gemelo digital de un motor físico propio. Esto hace que, dependiendo la parte del sistema real que se esté modelando, pueda ser necesario elevar el nivel de detalle.

Por tanto, hay dos aspectos a tener en cuenta para la calidad del modelado visual:

1. Si existe un motor físico subyacente al DT, los modelos visuales pueden utilizarse como modelos físicos, implicando que existirán ciertas partes del modelo visual que sea necesario modelar de manera exacta a la realidad.
   Ejemplo: una puerta al cerrarse en su marco. Si queremos modelarla mediante ecuaciones matemáticas no importa la calidad del modelo visual, pero si queremos dotar de física al modelo visual, tendremos que realizar en detalle tanto la puerta como el marco, que hará de tope, como ocurre en la realidad. E incluso, si en la realidad la puerta roza con el suelo, habrá que tenerlo en cuenta en el modelo visual.
2. El segundo aspecto es si se requerirán modelos detallados en 3d para permitir al usuario final una mayor facilidad de identificación de los elementos que pertenecen al DT con la realidad para los servicios que se implementarán al final del método.

Ejemplo: un servicio para un DT puede ser el mantenimiento del sistema real mediante realidad aumentada. Para que sea práctico, es necesario que el usuario (en este caso el mantenedor) sea capaz de identificar en el DT los elementos reales. Para ello habrá que marcar un nivel mínimo de detalle para permitirlo.

Actualmente, dentro del campo del diseño tridimensional digital, se han realizado unos avances increíbles, posibilitándose la creación de cualquier diseño 3d que el artista quiera realizar. Para entender estos avances hay que distinguir dos tipos de herramienta dentro del mundo del diseño tridimensional:

1. El software utilizado explícitamente para diseñar. Este software desde hace tiempo permite a los artistas la creación sin restricciones de cualquier cosa.
2. El software que dará "vida" al diseño realizado. Por ejemplo, para realizar un videojuego, primero se diseñan los personajes y los objetos en software distintos, diseñados específicamente para ello (es el caso del punto uno), y posteriormente se integran en programas que permiten programar esos diseños para que realicen lo necesario para el juego.
   Es en estos tipos de programas donde se han realizado los avances más grandes: recientemente, la limitación de los artistas estaba en este segundo software, dado que, si realizaban diseños muy detallados, la renderización de éstos en tiempo real era muy pesada. Pero actualmente, gracias a los avances del hardware y del software, se han conseguido optimizar y realizar la renderización de modelos de más de un billón de vectores en tiempo real [8].

Dentro de un diseño tridimensional, se distinguen dos campos importantes:

- La forma, que está delimitada mediante un conjunto de vectores.
- Los materiales (o texturas) asignados a las superficies de la forma.

Cuando se ha hablado de que en presencia de motores físicos pueda ser necesaria una precisión alta en el modelado, hacía referencia a la forma. Los materiales ayudan a identificar objetos, pero son accidentales. Para que se pueda entender mejor un paralelismo con un cuadro sería que los materiales son los colores, mientras que los límites que contienen a los colores, la forma. Dependiendo de los servicios, los materiales tendrán o no importancia. En la figura 2.3, se puede observar un quadrotor diseñado en 3d sin y con texturas. Lo que queda en la parte izquierda es solo la forma del quadrotor [7].

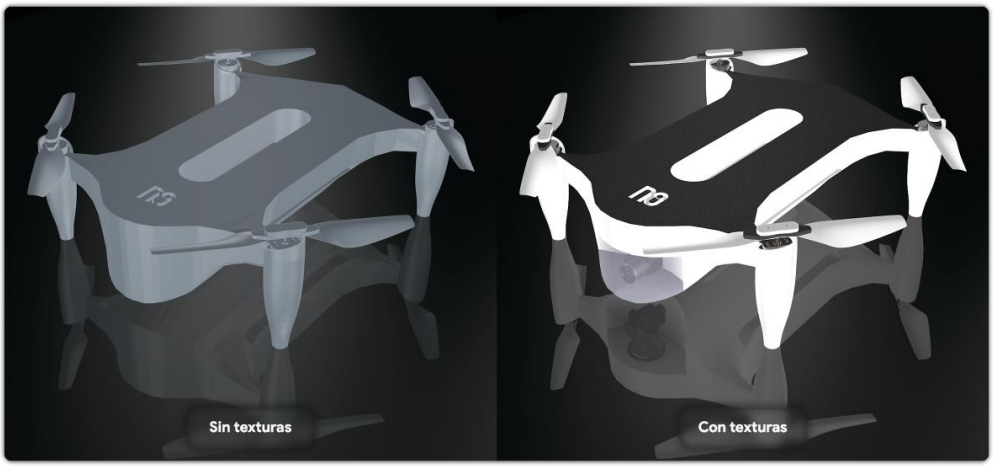


Figura 2.3 - Ejemplo de texturas/materiales en diseño 3d

Aunque se hayan nombrado los avances realizados para posibilitar el renderizado completo de cualquier diseño 3d, no resta que llevar a cabo esta tarea lleva consigo una cantidad alta de recursos. Por esto, además de tener en cuenta las cosas dichas hasta ahora, habrá que decidir la calidad del modelo visual teniendo en cuenta los recursos existentes.

Existen multiplicidad de servicios que no requieren ver los modelos visuales, como puede ser la optimización de la energía de una planta de producción, por lo que en estos casos será recomendable poner en

marcha en DT sin los modelos visuales, o al menos, sin tener en cuenta la renderización en tiempo real, dado que, como se ha dicho anteriormente, puede haber modelos visuales actuando como modelos físicos. Más adelante se expondran varios ejemplos sobre esta problemática, y modos de resolverla.

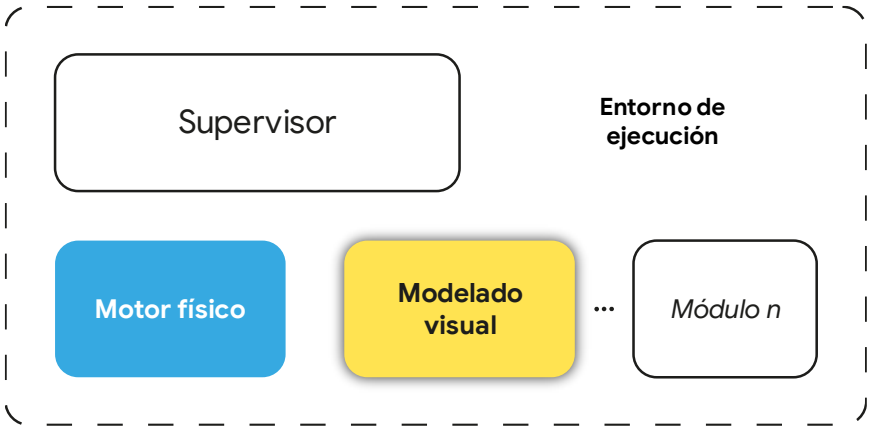


Figura 2.4 - Arquitectura ampliada con el modelado visual

En la figura 2.4 se puede observar el esquema de la arquitectura propuesta ampliada con el modelo visual.

No se ha mencionado que los modelos visuales pueden incluir también sonidos que los hagan más realistas. Igual que antes, dependiendo de los servicios será útil o no implementar sonidos a los modelos visuales.

2.3. Modelo funcional

Así como el modelo visual representa el aspecto externo de un objeto real, lo que se percibe por los sentidos de la vista y el oído, el modelo funcional representa el comportamiento del objeto. Ya se ha visto que, con la inclusión de un motor físico en un DT, los modelos visuales, a veces, pueden comportarse a la vez como modelos funcionales. Este paso del método se centrará más en aquellos aspectos del comportamiento que no se han modelado con el modelo visual (ya sea porque no se ha modelado nada con el visual o porque solo se ha modelado una parte).

En esencia, este paso consiste en el problema de la identificación, dentro del campo de la ingeniería de sistemas. Este problema tiene multitud de soluciones. De hecho, en los últimos años, gracias a las técnicas neuronales, se ha tenido la posibilidad de alcanzar multitud de soluciones que hasta ahora no eran posibles.

Por simplificar, para los modelos funcionales se hará una clasificación en dos grupos:

1. Modelos analíticos: son un conjunto de ecuaciones matemáticas que rigen el comportamiento del objeto. Para poder utilizar este tipo de modelo funcional es necesario conocer las ecuaciones y que sea posible implementarlas.
2. Modelos no analíticos: en caso contrario, ya sea por desconocimiento del objeto o porque no sea posible implementar sus ecuaciones.

Estos dos grupos contienen características que comparten y varias que los diferencias y lo hace, aparte de lo dicho, preferible al otro. Por ello se explicará a continuación cada uno de ellos.

2.1.1 Modelos analíticos

Un modelo analítico es un conjunto de ecuaciones proveniente del análisis de un objeto. Este conjunto puede ser las ecuaciones diferenciales que definen al completo el objeto a modelar o un conjunto de linealizaciones procedente de una función de transferencia obtenida.

Volviendo a la definición de un DT, no importa de donde procedan las ecuaciones, sino su precisión a la hora de modelar al sistema real; es decir, si los aspectos que ignoran las ecuaciones no sea un *comportamiento del sistema real con relevancia mínima*.

En la mayoría de los casos, lo más utilizado suelen ser linealizaciones. Estos modelos son buenos alrededor

del punto y el instante de trabajo donde y cuando se ha linealizado. Esto lo hace muy dependiente, siendo, de forma general, no aceptable para implementar en un DT, dado que uno de los servicios más importantes es la antelación de eventos; eventos que suceden en otros instantes de tiempos distintos a los que se realiza la linealización. Dependiendo de la cercanía de estos instantes se dará por válida la linealización o no para su implementación.

Por otro lado, las ecuaciones conocidas que caracterizan a un sistema real, en cierta medida suelen estar simplificadas, para ser viable su implementación. Al igual que antes, si los errores conllevados de esta simplificación son aceptables, serían aptas para su implementación.

El problema de la decisión está en saber si de verdad son aceptables o no. Poniendo un ejemplo, si se quiere comprobar que las ecuaciones siguen la degradación de un sistema a lo largo de un ancho periodo de tiempo, para comprobarlo de verdad habría que esperar ese tiempo o haberlo medido con anterioridad. Como siempre, depende de los servicios que se requieran.

La posible falta de implementación de estos modelos puede deberse a muchas causas. Entre ellas se encuentra que estos modelos no sean causales, es decir, que requieran de estados futuros para su ejecución; y su alta tendencia a necesitar altos recursos para ejecutarse, si el sistema es lo suficientemente complejo, como es el caso general de los sistemas reales. Esta segunda causa no imposibilita realmente implementarlos, pero resta mucha funcionalidad al DT si se implementan.

Por último, la gran ventaja de estos modelos, si es posible su implementación, es su inmediata implementación, dado que ya son conocidos.

2.1.2 Modelos no analíticos

Un modelo no analítico es aquel que no se basa en ecuaciones matemáticas.

Actualmente existen muchos tipos. Por ejemplificar, el problema de la identificación se puede resolver mediante técnicas de fuzzyficación, que se trata de crear leyes con lógica difusa para modelar un sistema. La ventaja de este tipo de modelos es que pueden complicarse los suficiente para ir adaptando más el modelo al sistema real.

Además, están en auge otro tipo de modelos basados en redes neuronales, que poseen dos características muy importantes: su ejecución es muy rápida, consumiendo pocos recursos; y, al igual que antes, las redes neuronales son tan dinámicas que se pueden adaptar al sistema real.

Sin juntamos ambos ejemplos, se estaría ante modelos basados en redes neuroborrosas, uniendo los puntos positivos de ambos ejemplos [9]

Viendo las ventajas de ambos grupos de modelos presentados, cabe la posibilidad de implementar varios modelos para un mismo sistema de ambos grupos para llevar a cabo su identificación. Esto conseguiría tener las ventajas de todos los modelos implementados.

Para conseguir esa multiplicidad de modelos es necesario unificarlos. Al proceso de asociación de modelos se le va a denominar **unificación de modelos funcionales**.

Para ilustrar como se podría implementar esta unificación y sus ventajas, se pondrá un ejemplo: se está modelando en paralelo un sistema real mediante sus ecuaciones matemáticas conocidas y una red neuronal, aprendiendo en tiempo real. Las salidas de los modelos van a una abstracción de un filtro de Kalmman, que, con el paso del tiempo, va dando prioridad a la red neuronal, hasta que esté totalmente entrenada, que le dará preferencia máxima, dado que será más precisa que el modelo analítico, porque se le ha dotado de la suficiente dinámica para amoldarse al paso del tiempo. Además, dado que, en este ejemplo, la velocidad de ejecución de la red neuronal es más rápida, tras un cierto período de tiempo se podría anular el modelo analítico y la unificación para mejorar el rendimiento, teniendo solo la red neuronal en funcionamiento.

Otro ejemplo sería el caso de varios modelos analíticos. Cada uno linealizado en puntos distintos, de trabajo o temporales. Con otra abstracción del filtro de Kalmman a la salida se podría crear un modelo mucho más completo que cualquiera de ellos individualmente.

El gran problema de la unificación son los recursos que consume, aunque, como se ha ejemplificado,

pueden implementarse técnicas para optimizar su implementación (dejando la unificación solo un período de tiempo, un tiempo de transición entre dos modelos: uno no completo y otro en entrenamiento).

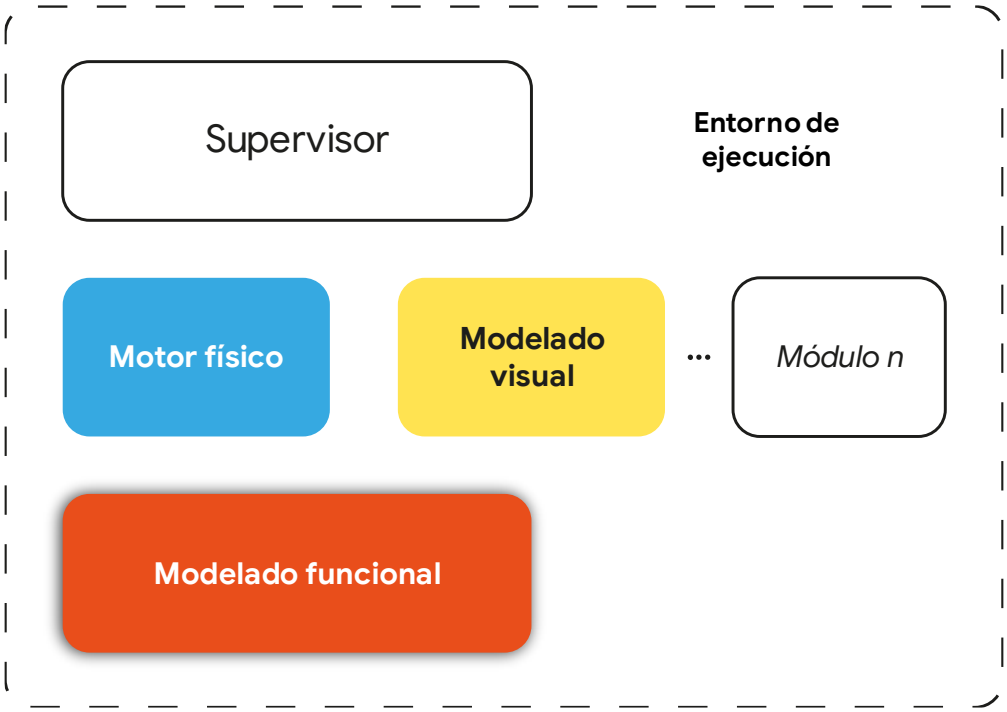


Figura 2.5 - Arquitectura ampliada con el modelado funcional

En la figura 2.5 se puede observar el esquema de la arquitectura propuesta, a la cual se le ha añadido el modelo funcional.

Todos los modelos dichos anteriormente se implementan con unos parámetros para asegurar que ningún modelo es estático. Esto permite adaptarse a las condiciones cambiantes de los sistemas reales. Además, permite que se posibilite la traslación de módulos de modelos funcionales a otros DT sin tener que cambiarlos, solo ajustando sus parámetros, si el sistema real es lo suficientemente parecido al de origen.

Un ejemplo puede ser el modelado de un motor de una cinta transportadora. Ese modelado se podría utilizar también para otro motor en otro sitio. Para ello solo hay que entrenar de nuevo el modelo completo (o solo sus parámetros en el caso del analítico). Este entrenamiento se explicará en el siguiente paso.

2.4. Autoaprendizaje

Todos los modelos funcionales, como ya se ha dicho, se implementan de tal manera que puedan amoldarse al sistema real. Es decir, son dinámicos. Esto es muy importante, no solo por la versatilidad que proporciona utilizar el mismo módulo para varios elementos semejantes, sino porque el sistema real cambia a lo largo del tiempo.

Al final un modelo no deja de ignorar ciertos aspectos, que, con el transcurso del tiempo, van tomando importancia. Si los modelos no fueran dinámicos, el DT estaría obsoleto al instante. Además, para poder predecir, es necesario que los modelos aprendan cómo se comporta el sistema a lo largo de un intervalo de tiempo.

Se pueden distinguir dos tipos de entrenamientos:

1. Entrenamiento offline: el DT está parado y los modelos se entrenan con datos recogidos con anterioridad del sistema real.
2. Entrenamiento online: o en tiempo real. El DT está en funcionamiento y los modelos se están entrenando en paralelo con los datos pasados y actuales del DT y del sistema real.

Para el DT se propone utilizar ambos entrenamientos, pero para momentos distintos:

a. El offline es muy útil cada vez que se inserte un modelo por primera vez aun DT. Este modelo se entrena externamente al DT y, una vez entrenado, se introduce en el DT.
b. El online permite que los modelos se vayan actualizando a los cambios temporales del sistema real.

Para llevar a cabo ambos entrenamientos es necesario unos algoritmos de entrenamiento, dependiendo de cada modelo será necesario uno concreto, pudiendo existir un algoritmo que entrene a varios modelos.

Está en proceso de investigación la creación de un único algoritmo lo suficientemente genérico que sea capaz de entrenar todos los modelos funcionales de un DT. Junto con esta investigación, subyace otra, más profunda: la creación de una red neuronal lo suficientemente versátil como para poder adaptarse a cualquier sistema. Esta es la llave hacia una arquitectura totalamente portable de un sistema real a otro.

Esta "red genérica" apoyada por el algoritmo genérico, trataría de ir configurándose a sí misma desde empezando por formas sencillas (un perceptrón, por ejemplo), hasta otras más complejas. La idea consiste en tres pasos:

1. Adoptar una forma nueva (realizado por la red genérica).
2. Entrenar esa forma nueva (realizado por el algoritmo genérico).
3. Si el error definido está por debajo del umbral, fin de la identificación. En caso contrario, se vuelve al paso 1, complicando la forma, dependiendo del sistema.

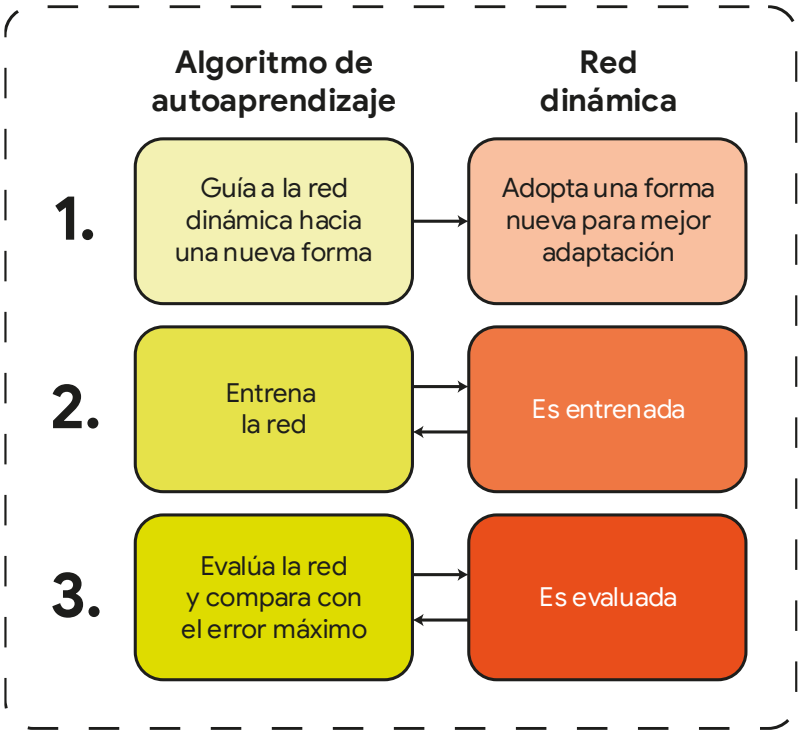


Figura 2.6 - Pasos de la rutina de identificación

En la figura 2.6 se puede observar de manera gráfica los pasos nombrados que se acaban de nombrar. Una nota para el primer paso es que esa búsqueda de una forma nueva más compleja, que consiste en añadir neuronas y/o capas (neuronal, fuzzy o del tipo requerido), debe estar guiada para que el problema de la identificación sea lo más óptimo, por un lado, para que se llegue a una forma adecuada para el sistema a identificar; y, a la vez, para que no consuma todos los recursos existentes.

Al final existen tres problemas que hay que solucionar: identificación, entrenamiento y guiado. Todos ellos deben ser optimizados para que se permita su implementación. Al módulo que lleva a cabo estas tres tareas se le llamará **módulo de autoaprendizaje**.

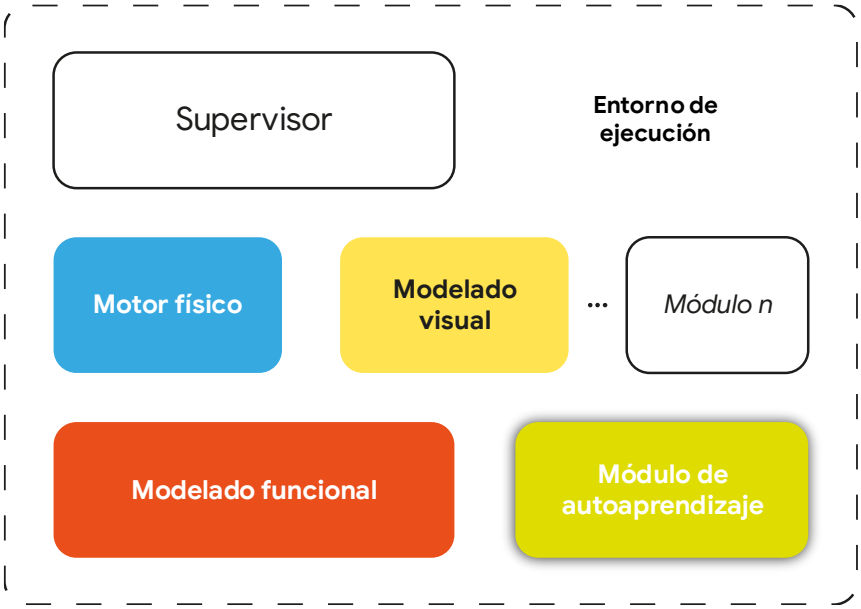


Figura 2.7 - Arquitectura ampliada con el módulo de autoaprendizaje

En la figura 2.7 se puede observar la arquitectura actualizada hasta el momento.

2.5. Conectividad

El gemelo digital no puede estar aislado. Las causas son múltiples: necesita la recepción de datos para los entrenamientos, para cerrar el lazo con los controladores, para la implementación de servicios, etc.

Existen multiplicidad de posibilidades de comunicación, infinidad de protocolos. Se sugieren a continuación tres parámetros para ayudar a la elección:

1. Calidad: fiabilidad necesaria para la comunicación.
2. Rapidez: velocidad requerida para la comunicación.
3. Contexto del sistema real: el ambiente del sistema real marcará la gran mayoría de veces comunicaciones estándar a implementar.

Para ilustrar estas tres características se pondrán dos ejemplos:

a. Se tiene un servicio de un DT compuesta por una red neuronal que recibe información en tiempo real del precio de las acciones en bolsa. El servicio decide si es recomendable o no invertir en algunas acciones. Probablemente, este servicio necesite una comunicación con alta calidad, rapidez media o alta, dependiendo de la volatilidad de las acciones analizadas normalmente. Con alta probabilidad la información proceda de Internet, por tanto el estándar de comunicación a implementar será la norma 802.3 si tiene acceso por cable o la 802.11 si la conexión es inalámbrica.
b. Se tiene un servicio de un DT. Se trata del control de estabilidad de un robot bípedo. Este servicio requiere que la rapidez sea extrema, dado que el sistema a controlar es inestable.

Esto implica que la comunicación es muy dependiente de los servicios que estén implementados, aparte del necesario para el autoaprendizaje del DT.

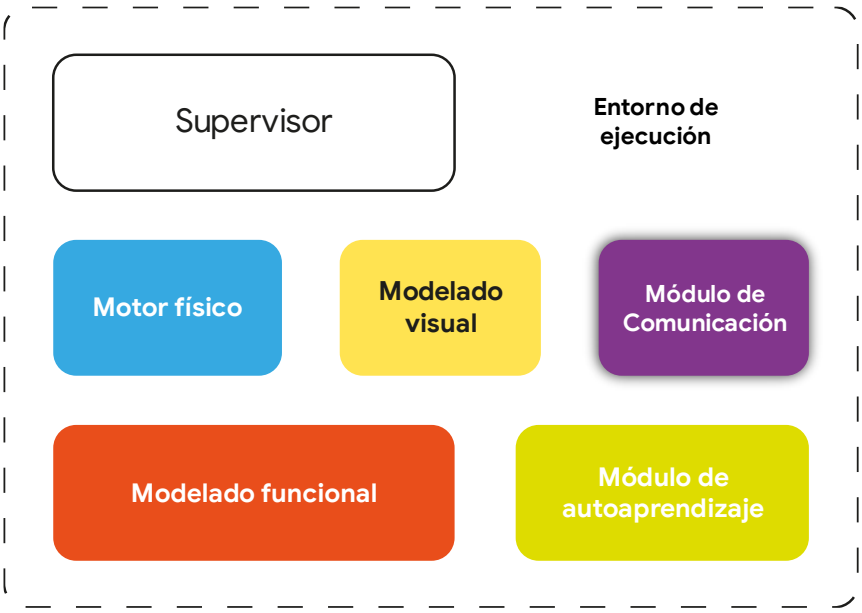


Figura 2.8 - Arquitectura interna del DT completa

En la figura 2.8 se ha mostrado el conjunto de módulos necesarios para el DT.

2.6. Servicios

Este último paso es el único que no pertenece al DT propiamente. Si los anteriores pasos del método se centraban en el cómo, este paso es el para qué. Este paso, aunque parezca el menor, es el que le da utilidad al gemelo digital.

La ventaja de un DT, frente a los modelos convencionales, es su parecido con la realidad (teóricamente perfecto, en la práctica con tendencia perfecta con el tiempo). Esto posibilita predecir el comportamiento del sistema real.

Esa predicción abre un abanico de servicios muy amplio, en múltiples campos como el mundo del control, la optimización de los sistemas (eficiencia energética, productividad…), solución preventiva de errores, vida material de los sitemas, etc.

Además, en el caso de que se haya insertado modelado visualmente el DT, se abren campos como la realidad aumentada y virtual para mejorar la experiencia del usuario: mantenimiento con AR, identificación visual de fallos, etc.

La gran inmensa variedad de servicios posibles para implementar hace que la implementación de un gemelo digital sea recomendable en multitud de situaciones.

En la figura 2.9 se puede ver que se ha añadido un bloque externo al DT para indicar que los módulos de servicios no pertenecen al DT propiamente; esto no implica que no se ejecuten en el mismo entorno de ejecución.

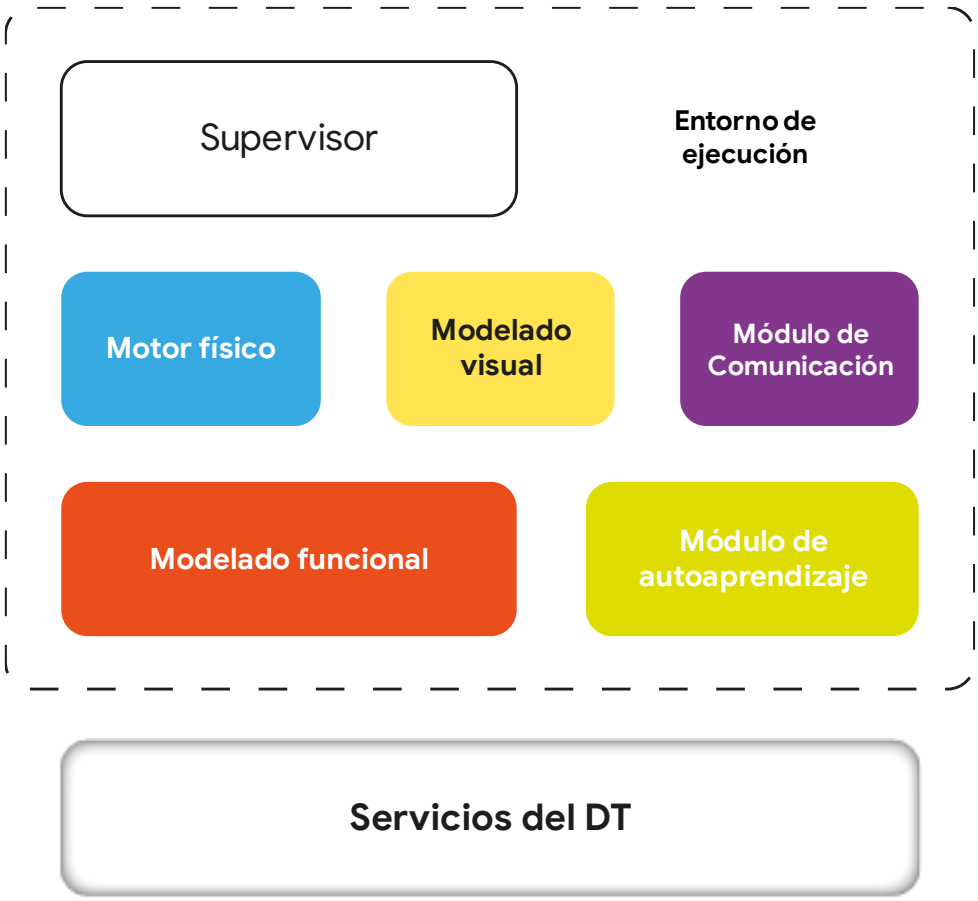


Figura 2.9 - Arquitectura interna del DT completa más los servicios

Se ha explicado el método de manera abierta, con unas pautas para llevar a cabo una implementación, a propósito, para dejar reflejado su versatilidad como su potencial para poder llevar a cabo gemelos digitales sobre multiplicidad de sistemas reales muy distintos entre sí: desde DTs sobre procesos de fabricación hasta DTs sobre mercados financieros o DTs sobre vehículos, barcos, etc.

Para verificar su potencial, se explicará a continuación el estado actual de un gemelo digital en proceso de implementación sobre una célula de fabricación situada en la Escuela Superior de Ingeniería de Sevilla.

3 UNA IMPLEMENTACIÓN

Esta segunda parte trata sobre un caso real de implementación de un gemelo digital, realizado por cuatro personas, entre ellas se encuentra el autor de este trabajo. Para dar una visión completa del caso real, se resumirán las partes realizadas por los tres restantes del equipo, mientras que se entrará en detalle en las pertes realizadas por el autor del presente trabajo. Se dirá en cada parte del método quién es el autor.

Este gemelo es una lanzadera para algunas tareas del proyecto DENiM. Esto implica que se ha querido implementar algunos conceptos de varias formas, para conocer todas las posibilidades existentes. Además, los módulos se han intentado hacer de manera genérica para su posible implantación en otros gemelos. Actualmente, la implementación está en fase de desarrollo.

Lo primero y más importante antes de plantearse realizar un gemelo digital es conocer con detalle el sistema real. Para ello se dará una breve explicación de este.

3.1. Sistema real

El sistema real es una célula de fabricación de pruebas construída en el laboratorio L1 de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla.

Se compone, entre otros, de los siguientes subsistemas:

1. 4x Cintas transportadoras.
2. Pistones.
3. Autómata.
4. Caja de control y seguridad.
5. Almacén.
6. Sensores.
7. Brazo robótico de almacenaje.
8. Almacén de alimentación de bandejas.
9. Bandejas.
10. Robot manipulador Scorbot.
11. Robot manipulador tipo SCARA.

En las figuras 3.1 y 3.2 se muestran fotografías realizadas sobre la célula. Se pueden observar en ellas algunos de estos subsistemas.

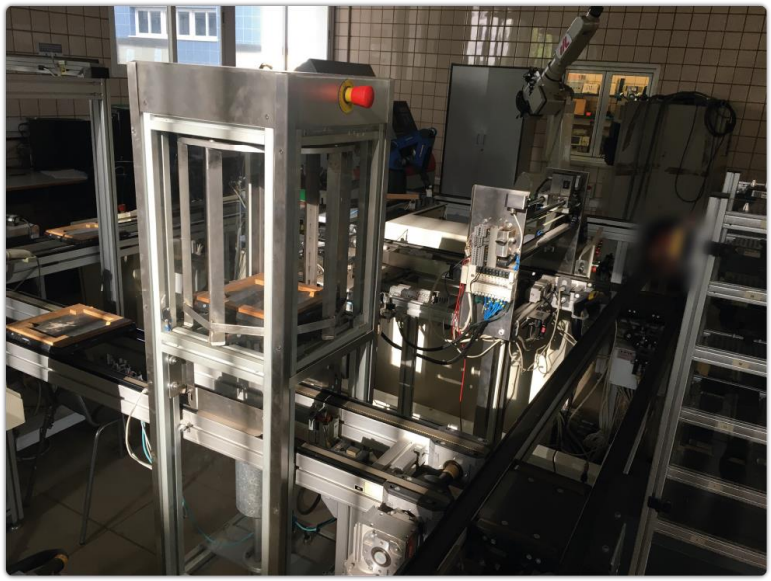


Figura 3.1 - Primera vista de la célula de fabricación

Figura 3.2 - Segunda vista de la célula de fabricación

De todos los elementos instalados en la célula, se ha realizado el gemelo digital de algunos. En la figura 3.3 se puede ver la planta de la célula de manera esquemática, solo con los elementos que se han implementado en el gemelo digital.

Figura 3.3 - Esquema sobre la célula de fabricación

Como se ve en la figura 3.3 el sistema está dividido en muchos subsistemas que habrá que tratar por separado, esto implica que el método propuesto de forma teórica habrá que aplicárselo a cada subsistema por separado (solo los pasos que procedan para cada susbsistema) y unificarlos posteriormente. A continuación, se irá desarrollando el método, paso por paso, tratando de todos los subsistemas en paralelo.

3.2. Arquitectura interna

Lo primero que se ha realizado es una investigación sobre las herramientas existentes en el mercado. El fruto de esa investigación ha sido un trabajo finde grado. La conclusión de éste es que la herramienta idónea para implementar el gemelo digital del sistema real explicado es Unity3d [10].

Unity3d es un software diseñado para la creación y el desarrollo de videojuegos en diferentes plataformas. Existen otros programas muy similares como Unreal Engine o Blender, cada una con sus respectivas ventajas respecto a las otras.

A la hora de crear contenido, Unity combina una interfaz gráfica con programación en lenguaje C#. La rutina básica de desarrollo se basa en crear un nuevo objeto mediante el uso de la barra de herramientas gráficas, y posteriormente desarrollar y asignar al objeto el programa que describe su comportamiento. Esto hace que sea muy intuitivo trabajar con él, pero es necesario conocer a fondo el lenguaje C# para poder desarrollar cualquier aplicación con un mínimo de complejidad. Para una explicación más detallada [11].

Las características que hacen idónea estas herramientas son:

1. Contiene un motor físico muy completo de fábrica con el cual se puede interactuar en cualquier momento.
2. Contiene un entorno de ejecución que permite la modularidad y el paralelismo de manera innata.

Para explicar la ejecución dentro de Unity3d es necesario saber antes cómo funciona el tiempo dentro de Unity3d. En la ejecución de una aplicación creada por este software hay dos intervalos importantes:

- Fixed time: es un intervalo de tiempo fijo (por defecto ~ 0.02s). Además, es el mínimo intervalo se ejecutan los programas implementados, de manera síncrona. Puede modificarse.
- Frame time: es el tiempo que transcurre entre dos fotogramas. Depende del hardware donde se esté ejecutando.

Acoplados a estos tiempos existen cuatro funciones fundamentales propias de Unity3d:

1. Awake y Start: se utilizan para inicializar programas. De manera general, solo se ejecutan una vez en toda la ejecución. La diferencia entre ambas es que la primera se ejecuta nada más lanzarse la aplicación, mientras que la segunda se espera hasta el primer fotograma.
2. Update y FixedUpdate: se ejecutan en bucle durante la ejecución. La primera se ejecuta cada vez que hay un fotograma (Frame time), mientras que la segunda cada fixed time. Esta segunda función es la recomendada para la ejecución de toda la física.

Estas últimas funciones permiten que cualquier módulo con una de ellas funcione en paralelo sin tener que cambiar ninguna del resto; en esencia, permiten la modularidad. Esto no asegura que haya que modificar otros módulos si necesitan comunicarse entre sí; es decir, la escalabilidad buscada no se consigue de fábrica con este software. Para ello es necesario, como ya se verá más adelante, realizar las funciones con esta finalidad, cuando sea posible.

Una vez entendido cómo es posible que se ejecuten varios programas en paralelo, se explicará la arquitectura completa del gemelo digital implementado.

Nota: es difícil distinguir entre modelos visuales y funcionales cuando algún modelo use el motor físico.

En el Anexo I se muestra un esquema de todos los elementos que componen la arquitectura interna del gemelo digital.

El elemento más carácterístico de esta arquitecura es el módulo *Gate Components* del esquema del Anexo I. Este módulo es el que separa los modelos funcionales de todos los elementos del gemelo digital. Se compone de dos tipos de componentes: unos utilizados por los actuadores de la planta (los pistones, las cintas transportadoras y el compresor de aire) y los otros utilizados por los sensores (en este caso solo hay sensores inductivos).

La causa de la implementación de este módulo es conseguir el desacople que proporcione esa escalabilidad buscada. Para entender como se puede conseguir se explicará el código de cada uno de los dos tipos.

En primer lugar, tenemos el código del *ActuatorComponent.cs*:

```
/*
////////////////////////////////////////////////////////
    PROPERTY OF THE UNIVERSITY OF SEVILLE,
    PARTNER OF THE DENiM PROJECT

    CONTEXT:
    1. Variables that define the actuator at each instant
////////////////////////////////////////////////////////
*/

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

////////////////////////////////
// VARIABLES FOR THE ACTUATOR //
////////////////////////////////
public class ActuatorComponent : MonoBehaviour
{
    public string Name;
    public int Number;
    public string Type;
    public bool[] flag;
}
////////////////////////////////
```

Este clase contiene todas las variables que caracterizan a cualquier actuador: el nombre del actuador, su número de identificación, el tipo, y el valor del actuador. Nótese que todos los actuadores implementados son digitales (los valores posibles son booleanos).

El programa *SensorComponent.cs* es similar:

```
/*
////////////////////////////////////////////////////
    PROPERTY OF THE UNIVERSITY OF SEVILLE,
    PARTNER OF THE DENiM PROJECT

    CONTEXT:
    1. Variables that define the sensor at each instant
////////////////////////////////////////////////////
*/

using System;
using UnityEngine;

//////////////////////////////
// VARIABLES FOR THE SENSOR //
//////////////////////////////
public class SensorComponent : MonoBehaviour
{
    public string Name;
    public int Number;

    // When change the sensor value, it launchs any functions
    public bool[] flag
    {
        get { return flagValue; }
        set
        {
            flagValue = value;

            if (AutomationParameters.internalAutomation)
                InternalAutomation.AutomationProcess(ref Automation.Memory);
        }
    }
    private bool[] flagValue;
}
//////////////////////////////
```

La diferencia entre las dos clases está en la definición del valor del sensor: cuando se escribe el valor de un sensor, se lanza una función. En este caso es la automatización interna, si procede. Solo decir que esto hace que la automatización interna se ejecute solo cuando un sensor cambia. Este es uno de los puntos de optimización realizados.

Estos códigos por sí solos no hacen nada, necesitan de otros que los instancien: cada actuador/sensor, nada más empezar (con la función *awake*), instancia su clase correspondiente y le asigna su nombre y valor. Externamente, con buscar todas las copias de estas clases se puede saber perfectamente cuántos sensores/actuadores están en la planta. De hecho, si se introduce otro sensor/actuador, no existiría ningún problema, dado que se le buscaría de la misma manera que a los demás. En esto consiste la amploiabilidad de esta arquitectura.

Esa búsqueda que se acaba de nombrar se explicará más adelante cuando se vean los programas que la realizan.

El supervisor en esta arquitectura, en el estado actual de la implementación, se encarga, actualmente, en orden, de lanzar la inicialización de toda la planta, controlar la automatización de la misma e iniciar la comunicación. El código del supervisor se llama *Main.cs*. Se adjunta a continuación.

```
/*
////////////////////////////////////////////////
    PROPERTY OF THE UNIVERSITY OF SEVILLE,
    PARTNER OF THE DENiM PROJECT
```

```
      CONTEXT:
      1. Main control of the Digital Twin
  ////////////////////////////////////////////////
  */

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

  ////////////////////////////////////////////////////////////////
  // CLASS THAT CONTROLS THE ENTIRE OPERATION OF THE DIGITAL TWIN //
  ////////////////////////////////////////////////////////////////
  public class Main : MonoBehaviour
  {
      // DECLARATION OF GLOBAL VARIABLES
      private Automation automation;
      private Communication communication;

      // FUNCTION THAT IS ACTIVATED IMMEDIATELY AFTER ALL FUNCTIONS AWAKE WERE
 ENDED
      public void Start()
      {
          // Initialize all components of Digital Twin
          Initialization.GlobalInitialization(transform);

          // Add the class that control the automation of the plant
          automation = gameObject.AddComponent<Automation>() as Automation;

          // Add the class that control the communication
          communication = gameObject.AddComponent<Communication>() as Communic
ation;
      }
  }
  ////////////////////////////////////////////////////////////////
```

El primer elemento que lanza es la inicialización de la planta. Ésta inicializa todos los componentes necesarios: los elementos puente para el filtrado de los sensores (se explicará más adelante), que son los ficheros *JSON*; y la dirección de los pistones.

Un elemento a resaltar dentro de la inicialización de los pistones, que ocurre de otras maneras en otros subsistemas, es que se programa un pistón únicamente, y se instancia el mismo todas las veces que exija la realidad. Para diferenciarlos entre sí, solo hace falta inicializarlos (en este caso solo se distinguen por el sentido de la cinta).

```
  /*
  ////////////////////////////////////////////////////
      PROPERTY OF THE UNIVERSITY OF SEVILLE,
      PARTNER OF THE DENiM PROJECT

      CONTEXT:
      1. Functions that initialize all the Digital Twin
  ////////////////////////////////////////////////////
  */

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

  ////////////////////////////////////////////
  // FUNCTIONS THAT INITIALIZE ALL COMPONENTS //
```

```
//////////////////////////////////////////
public class Initialization : MonoBehaviour
{
    // FUNCTIONS THAT INITIALIZE ALL THE PLANT
    public static void GlobalInitialization(Transform transform)
    {
        Models(transform);
        FilteredFiles(transform);
    }

    // FUNCTIONS THAT INITIALIZE THE FILTERED FILES
    private static void FilteredFiles(Transform transform)
    {
        bool[] InitialValues = new bool[transform.GetComponentsInChildren<Se
nsorComponent>().Length];
        for (int i = 0; i < InitialValues.Length; i++)
            InitialValues[i] = false;
        FilteringFunctions.FileInitialization(FilteringParameters.ReadSensor
sPath, InitialValues);
        FilteringFunctions.FileInitialization(FilteringParameters.FilteredSe
nsorsPath, InitialValues);
    }

    // FUNCTIONS THAT INITIALIZE ALL MODELS
    private static void Models(Transform transform)
    {
        InitializeChangePistonsDirection(transform);
    }

    // FUNCTION THAT INITIALIZE THE DIRECTION OF EACH CHANGE PISTONS
    private static void InitializeChangePistonsDirection(Transform transform
)
    {
        int j = 1;
        foreach (Transform child in transform)
        {
            foreach (ChangePistonModule CPM in child.GetComponentsInChildren
<ChangePistonModule>())
            {
                CPM.sentidoCinta = j;
                j *= -1;
            }
        }
    }
}
//////////////////////////////////////////
```

Los métodos restantes que lanza el supervisor son la automatización y la comunicación, que se explicarán en el paso correspondiente.

En proceso de desarrollo están la mayoría de servicios que se implementarán. Todos estos servicios se manejarán desde el supervisor.

3.3. Modelo visual

Del modelo visual se mostrarán algunos detalles, dado que no ha sido desarrollado por el autor del presente documento.

Para realizar todos los modelos visuales se ha utilizado el software Cinema4d. Para más información sobre Cinema4d y su compatibilidad con Unity3d [9].

En primer lugar, se mostrará algunas capturas del modelo visual en general.

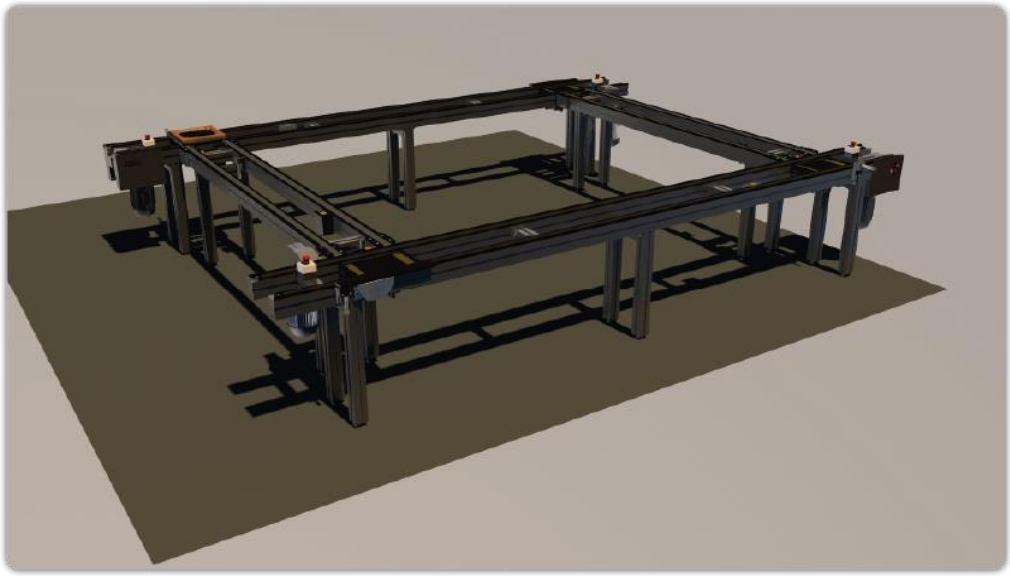


Figura 3.4 - Vista con perspectiva del modelo visual

En la figura 3.4 se puede observar un vistazo del modelo visual completo, es decir, de la unión de todos los modelos visuales implementados.

Algunos de ellos utilizan el motor físico para implementar su modelo funcional. Un ejemplo es el de la bandeja y los bordes de la cinta. La caja es arrastrada por la cinta transportadora porque tiene un material físico que hace que al contacto con la cinta exista rozamiento. Además, no se sale porque los bordes de las cintas son físicos, al igual que la caja.

En la figura 3.5 se muestra el modelo visual completo visto desde arriba.

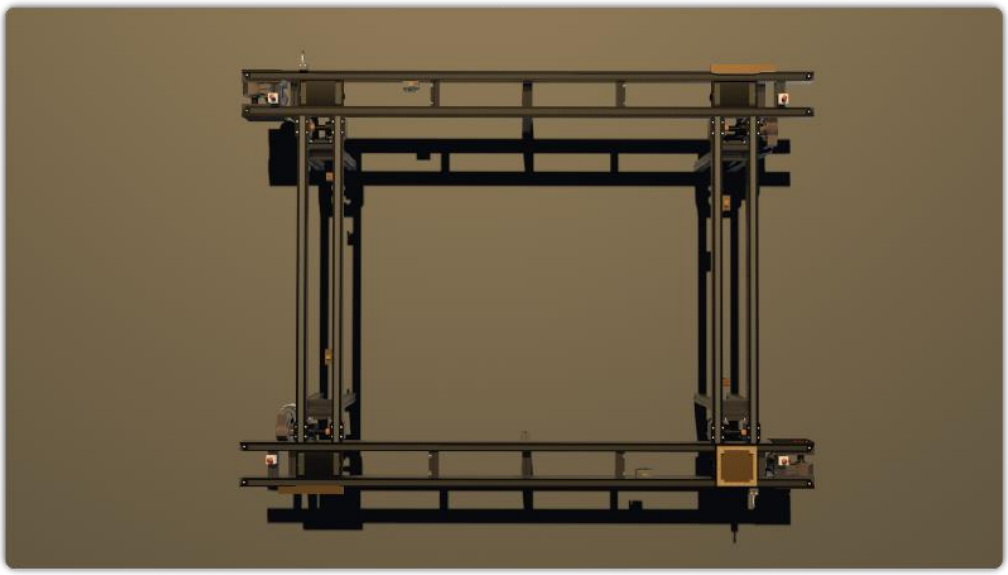


Figura 3.5 - Vista desde arriba del modelo visual

Cinema4d permite modelar cualquier cosa en 3d, con un realismo increíble. Para que se vean algunas de sus funciones, se mostrarán dos comparativas con la realidad, para que el lector pueda decidir por sí mismo.

En la figura 3.6 se puede ver la primera de ellas. Se trata de un zoom al sensor de presión. Este sensor tiene un sensor inductivo mirando a una chapa que existe debajo del tablón de madera superior. Una vez que llega a una plataforma desde la cinta chica e intercepta al sensor, la chapa metálica deja de estar enfrentada con el

sensor y salta.

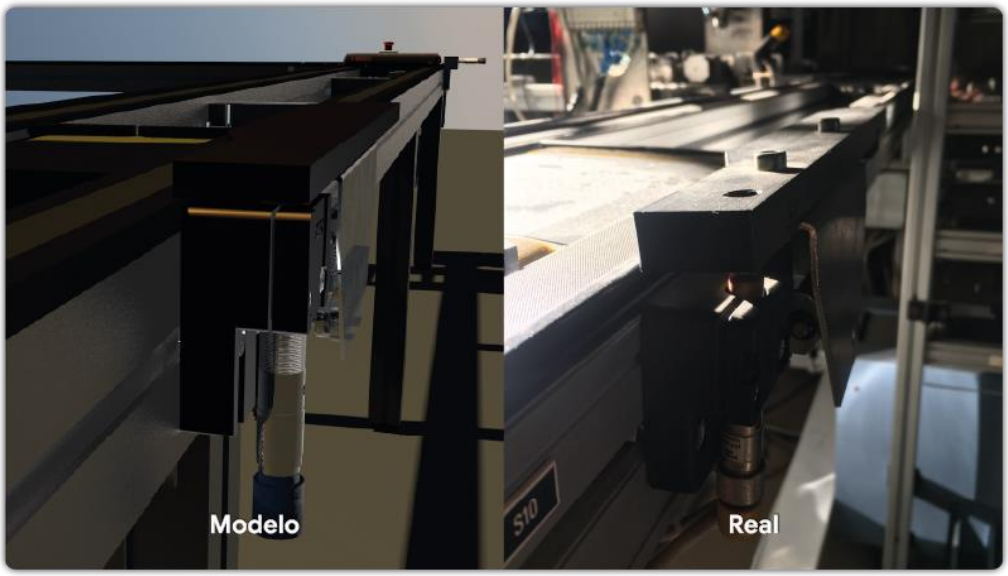


Figura 3.6 - Zoom sensor de presión

Este es un buen ejemplo de cómo se puede utilizar el motor físico. todo lo demás se ha calcado de la realidad: se ha utilizado un componente interno de Unity3d que simula el muelle de retroceso, se ha puesto un sensor inductivo frente a una placa metálica (el mismo sensor que hay en los otros lados) y el giro respecto a un eje (también se ha utilizado otro elemento de Unity3d para modelarlo). Se podría haber modelado con fuerzas elásticas y calcular giros y desplazamientos, pero dos elementos ya incluidos en Unity3d, que utilizan en motor físico, se hace más sencillo.

La siguiente comparación, mostrada en la figura 3.7, es una ampliación del fijador que hay entre las dos barras de la cinta larga.

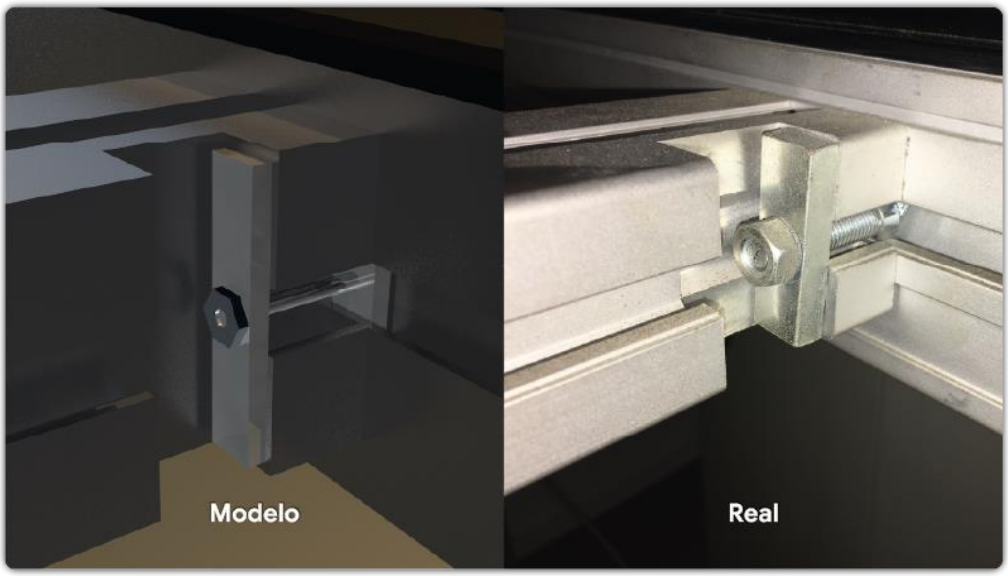


Figura 3.7 - Tornillo de ajuste entre barras

Por último, se muestra, en las figuras 3.8 y 3.9 el modelo de las plataformas desde varias vistas. Se han agrupado las fichas metálicas en una etiqueta o *tag* dentro de Unity3d, de nombre *metal*, para que puedan ser detectadas por los sensores inductivos. En total, en cada plataforma hay 8 fichas metálicas como se ve en las figuras 3.8 y 3.9.

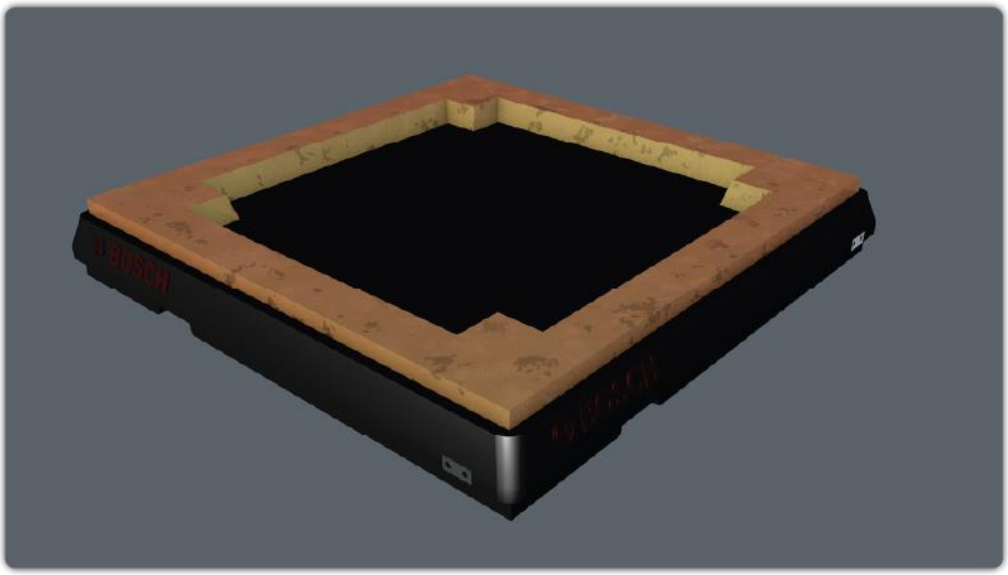


Figura 3.8 - Vista superior de una plataforma

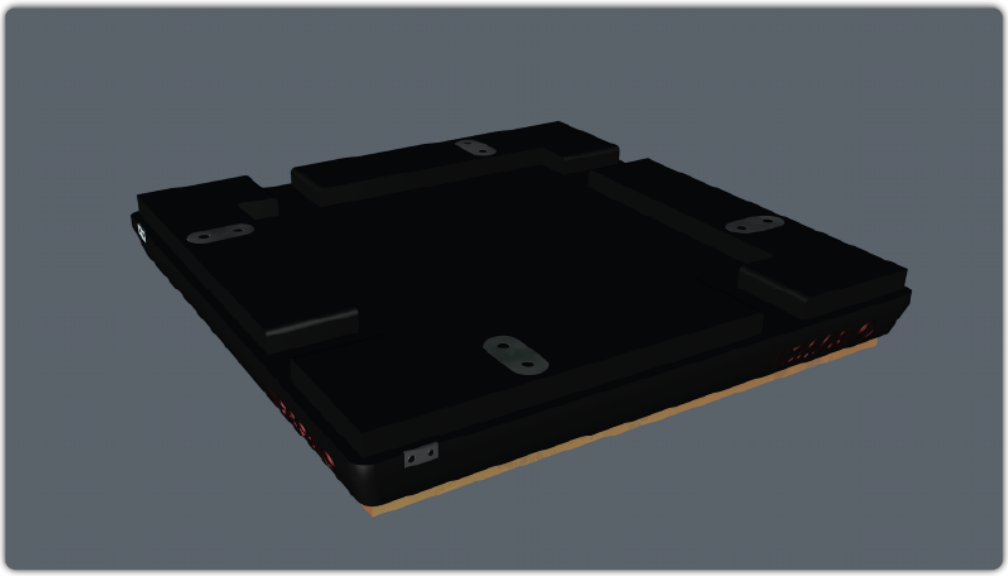


Figura 3.9 - Vista inferior de la plataforma

Como se puede observar, se pueden conseguir modelos idénticos a la realidad. El problema es el mismo: a mayor detalle, mayor número de recursos se necesitanpara renderizarse en tiempo real.

Unity3d permite, mediante plugins, la ejecución de las aplicaciones sin modelos visuales (pero manteniendo sus propiedades de modelos funcionales) para permitir simulaciones con un rendimiento muy alto. Depende el uso es muy útil, por ejemplo, para el autoaprendizaje, o para aquellas simulaciones que se realizan para predecir. Por esto mismo, el modelo visual depende de los servicios que se quieran implementar, pero puede diseñarse de manera detallada sin que reste rendimiento a servicios que no lo utilicen.

3.4. Modelo funcional

Parte del modelo funcional también ha sido implementado por otro miembro del equipo, que no es el autor. Como antes, se mostrarán pinceladas de lo que se ha hecho en general, pero solo se entrará en detalle por lo desarrollado por el autor.

En primer lugar, decir que se han implementado modelos desarrollados directamente o realizados con anterioridad en MATLAB, como es el caso del diseñado para el compresor de aire del tanque.

Un ejemplo de los desarrollados directamente ha sido el modelo de los sensores inductivos. Un sensor inductivo consiste en un sensor digital que detecta si un metal está en su zona sensible. Esa zona se ha modelado en Unity3d mediante un *collider*. A continuación, se expone el código del sensor:

```
/*
////////////////////////////////////////////////
    PROPERTY OF THE UNIVERSITY OF SEVILLE,
    PARTNER OF THE DENiM PROJECT

    CONTEXT:
    1. The Inductive Sensor class.
////////////////////////////////////////////////
*/

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

//////////////////////////////
// PARAMETER FOR ALL MODELS //
//////////////////////////////
public class InductiveSensor : MonoBehaviour
{
    // DECLARATION OF GLOBAL VARIABLES
    SensorComponent sensorComponent;

    void Awake()
    {
        sensorComponent = gameObject.AddComponent<SensorComponent>() as Sens
orComponent;
        sensorComponent.flag = new bool[1];
        sensorComponent.Name = transform.name;
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "metal")
            sensorComponent.flag = new bool[]{true};
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.tag == "metal")
            sensorComponent.flag = new bool[]{false};
    }
}
```

Dos detalles para resaltar: el primero es que su funcionamiento está basado en las dos funciones últimas; si el objeto que ha entrado en su zona de sensibilidad es de metal, el sensor se pone a *true*. En caso contrario toma el valor de *false*. La segunda cosa importante es la primera dentro de la función *Awake*. Esa línea es la que realiza la instancia del código *SensorComponent.cs* que se explicó anteriormente.

Al modelo funcional se le podrían añadir detalles como el retraso que experimenta la señal desde que el sensor detecta el objeto metálico, hasta que llega hasta el autómata. Pero es importante que no se pierda de vista que esta implemetación es un prototipo, con la finalidad de poner en marcha el método.

Otro ejemplo de esto, realizado por el presente autor, es el implementado para el autómata. Se han implementado dos perspectivas: si el autómata está dentro del gemelo digital (automatización interna) o si, por el contrario, está afuera (automatización externa).

La decisión de qué tipo de automatización se ejecuta (interna o externa) corresponde al usuario, mediante un servicio en proceso de implementación, que se explicará más adelante. De momento la elección se hace de

manera manual como se verá un poco más adelante.

El autómata está regido por cuatro programas: la rutina de la automatización interna (*InternalAutomation.cs*), la rutina de la externa (*ExternalAutomation.cs*), los parámetros y funciones del autómata (*AutomationSupport.cs*) y el programa principal del autómata (*Automation.cs*).

En primer lugar, se explicará *AutomationSupport.cs*. Este programa consta de los siguientes elementos:

- Dos parámetros: ExternalAutomationInterval es un intervalo que configura la comunicación de la automatización externa; e internalAutomation que es la variable que decide si se ejecuta la automatización externa o interna.
- Una definición de una estructura (AutomationObject) que define a todos los objetos grandes existente en la célula: cuatro cintas (con sus correspondientes sensores y actuadores) y el sistema de aire comprimido.
- Cuatro funciones que buscan esos módulos *Gate* (*SensorComponent* y *ActuatorComponent*) filtrándolos por nombre o por número de sensor/actuador.

A continuación, se expone el script *AutomationSupport.cs*.

```
/*
////////////////////////////////////////////////
    PROPERTY OF THE UNIVERSITY OF SEVILLE,
    PARTNER OF THE DENiM PROJECT

    CONTEXT:
    1. Parameters of automation.
    2. Functions for the automation.
////////////////////////////////////////////////
*/

// LIBRARIES
using UnityEngine;
using System.Collections;

////////////////////////////////////////////
// PARAMETERS OF ALL MODELS FOR AUTOMATION //
////////////////////////////////////////////
public class AutomationParameters : MonoBehaviour
{
    // EXTERNAL AUTOMATION PARAMETERS
    public const int ExternalAutomationInterval = 100;

    // INTERNAL AUTOMATION PARAMETERS
    public static bool internalAutomation = false;

    // STRUCT THAT CONTAIN A GENERIC OBJECT
    public struct AutomationObject
    {
        public string Name;
        public SensorComponent[] Sensors;
        public ActuatorComponent[] Actuators;
    }
}
////////////////////////////////////////////



////////////////////////////////////////////
// FUNCTIONS OF ALL MODELS FOR AUTOMATION //
////////////////////////////////////////////
public class AutomationFunctions : MonoBehaviour
{
```

```
      // FUNCTIONS THAT RETURN THE SENSOR COMPONENT
      public static SensorComponent ReturnSensorComponentByName(SensorComponen
t[] Sensors, string SensorName)
      {
          foreach(SensorComponent Sensor in Sensors)
          {
              if (Sensor.Name == SensorName)
                  return Sensor;
          }
          return null;
      }

      public static SensorComponent ReturnSensorComponentByNumber(AutomationPa
rameters.AutomationObject[] Memory, int SensorNumber)
      {
          foreach (AutomationParameters.AutomationObject memory in Memory)
              foreach(SensorComponent Sensor in memory.Sensors)
                  if (Sensor.Number == SensorNumber)
                      return Sensor;
          return null;
      }

      // FUNCTIONS THAT RETURN THE ACTUATOR COMPONENT
      public static ActuatorComponent ReturnActuatorComponentByName(ActuatorCo
mponent[] Actuators, string ActuatorName)
      {
          foreach(ActuatorComponent Actuator in Actuators)
          {
              if (Actuator.Name == ActuatorName)
                  return Actuator;
          }
          return null;
      }

      public static ActuatorComponent ReturnActuatorComponentByNumber(Automati
onParameters.AutomationObject[] Memory, int ActuatorNumber)
      {
          foreach (AutomationParameters.AutomationObject memory in Memory)
              foreach(ActuatorComponent Actuator in memory.Actuators)
                  if (Actuator.Number == ActuatorNumber)
                      return Actuator;
          return null;
      }
  }
  ////////////////////////////////////////
```

En segundo lugar, se entrará en el programa principal realizado por el autómata. Se implementan dos elementos:

- La función Start que llama a la rutina de inicialización de la memoria y, si se ha seleccionado, se añade una instancia de la automatización externa. Tal como está puesto, para selaccionarla habría que cambiar el *true* por un *false* para que se llevara acabo la automatización externa.
- La rutina de inicialización de la memoria. La memoria está compuesta por objetos tipo *AutomationObject* explicados anteriormente. Tal como está hecha la rutina, de ante mano no se sabe cuántos objetos existen, ni sensores ni actuadores dentro de los mismos. Esto es lo que permite el uso de los elementos *Gates*: cuando se requiera se introducen nuevos objetos y automáaticamente son añadidos en la memoria.

Se expone el código de *Automation.cs*:

```
/*
```

```
/////////////////////////////////////////////////
    PROPERTY OF THE UNIVERSITY OF SEVILLE,
    PARTNER OF THE DENiM PROJECT

    CONTEXT:
    1. This program contains the automation
    of the manufacturing process.
/////////////////////////////////////////////////
*/

// LIBRARIES
using System;
using UnityEngine;

//////////////////////////////
// ROUTINE OF THE AUTOMATION //
//////////////////////////////
public class Automation : MonoBehaviour
{
    // DECLARATION OF GLOBAL VARIABLES
    public static AutomationParameters.AutomationObject[] Memory;
    private ExternalAutomation externalAutomation;

    // FUNCTION THAT IS ACTIVATED IMMEDIATELY AFTER ALL FUNCTIONS AWAKE WERE
 ENDED
    void Start()
    {
        MemoryInitialization();

        // Change the flag value for start the internal automation
        AutomationParameters.internalAutomation = true;

        // Add the class that contain the protocol for the external automati
on
        if (!AutomationParameters.internalAutomation)
            externalAutomation = gameObject.AddComponent<ExternalAutomation>
() as ExternalAutomation;
    }

    // FUNCTION THAT INITIALIZES ALL THE VARIABLES OF THE AUTOMATIC PROGRAM
    void MemoryInitialization()
    {
        Memory = new AutomationParameters.AutomationObject[transform.childCo
unt];
        int numActuator = 0, numSensor = 0;
        for (int i = 0; i < transform.childCount; i++)
        {
            Memory[i].Name = transform.GetChild(i).name;

            Memory[i].Sensors = transform.GetChild(i).GetComponentsInChildre
n<SensorComponent>();
            foreach (SensorComponent sensor in Memory[i].Sensors)
            {
                sensor.Number = numSensor;
                numSensor++;
            }

            Memory[i].Actuators = transform.GetChild(i).GetComponentsInChild
ren<ActuatorComponent>();
            foreach (ActuatorComponent actuator in Memory[i].Actuators)
            {
                actuator.Number = numActuator;
```

```
                    numActuator++;
                }
            }
        }
    }
    ////////////////////////////////
```

Nótese que solo se ejecuta una vez, dado que solo tiene la función clave Start. Las automatizaciones internas y externas se autogestionarán la ejecución, como se verá más adelante.

La rutina de la automatización interna solo implementa la automatización que realizaría el autómata, no su dinámica interna, que se realizará más adelante. Esta rutina implementa tres subrutinas de automatización. En la figura 3.10 se puede observar que las tres subrutinas hacen referencia al compresor y a las cuatro esquinas. Dado que las esquinas tienen un comportamiento similar, dos a dos, para las cuatro solo hace falta dos subrutinas. Se podrían poner cuatro si se quisiera que alguna esquina se comportara de manera distinta a las otras.

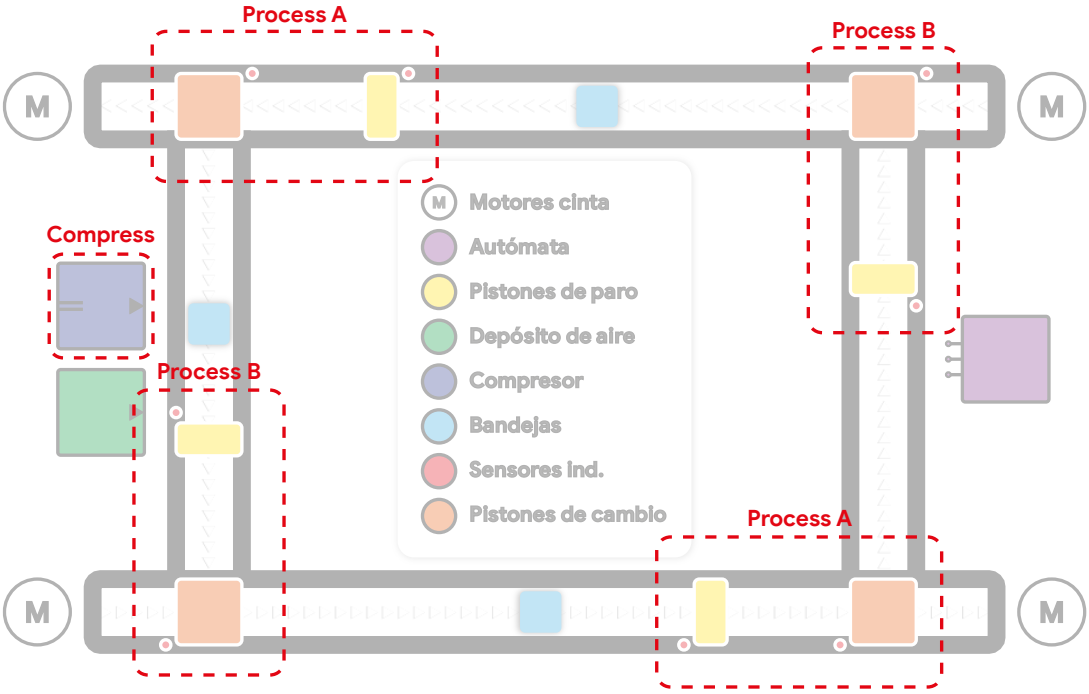


Figura 3.10 - Esquema sobre la célula de fabricación

En la figura 3.11 se muestra la red implementada para automatizar el compresor de aire. Analizándola, se obtiene que el depósito de aire tiene instalado dos sensores: uno que es el nivel máximo de presión (SC1) y otro que marca el nivel mínimo de presión que debe haber en el depósito (SC0). El compresor se activa o se desactiva, no se controla de manera analógica, en este caso. El nivel mínimo que marca el SC0 está por encima del nivel mínimo que requieren los pistones para funcionar, por seguridad.

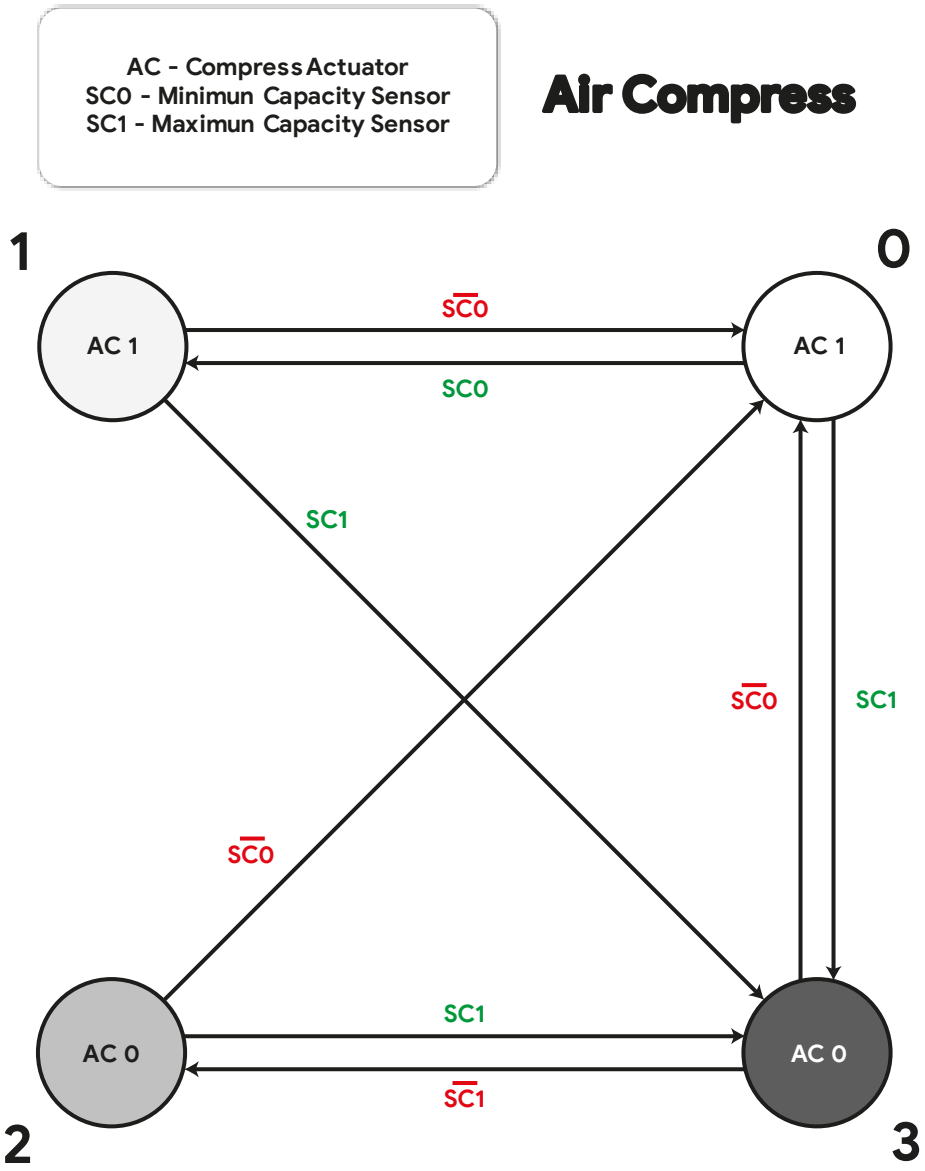


Figura 3.11 - Rutina de automatización del compresor de aire

Los estados son los siguientes:

0. Arranque del sistema, depósito por debajo del nivel de presión mínimo. Compresor encendido.
1. Compresor aumentando la presión del depósito. Presión entre la máxima y mínima.
2. Disminuyendo la presión del depósito. Presión entre la máxima y mínima. Compresor apagado.
3. Compresor apagado. Presión por encima del nivel máximo.

En la figura 3.12 se muestra la rutina implementada para la esquina tipo A, es decir, la plataforma está en la cinta larga y pasa a la cinta corta. En esa rutina intervienen tres sensores inductivos y dos actuadores.

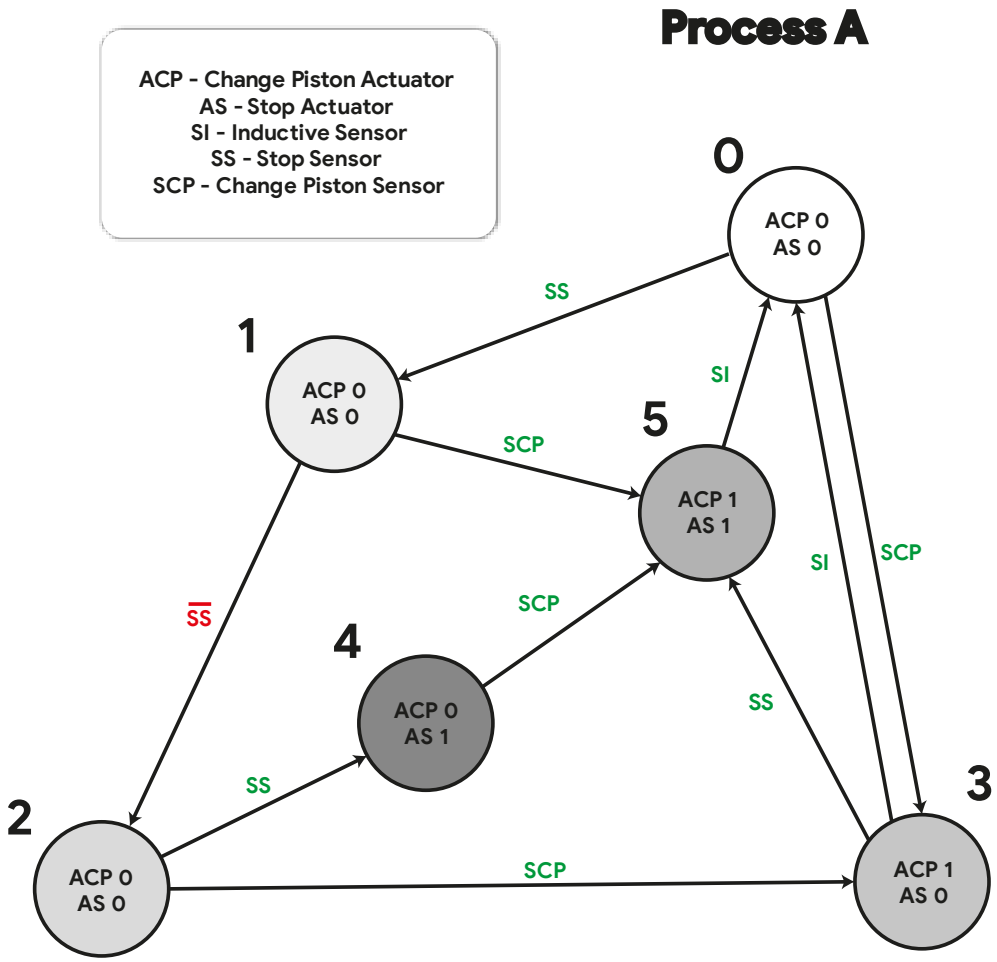


Figura 3.12 - Rutina de automatización tipo A

Los estados son los siguientes:

0. No hay ninguna plataforma en el pistón de cambio ni saliendo de éste.
1. La plataforma está justo encima del pistón de paro.
2. La plataforma está entre el pistón de paro y el pistón de cambio.
3. La primera plataforma llega al pistón de cambio. Éste se activa.
4. Ha llegado otra plataforma y la primera todavía está entre el pistón de paro y el pistón de cambio. El actuador de paro se activa para dejar margen a la primera plataforma.
5. La primera plataforma está en el pistón de cambio, la segunda espera en el pistón de paro. Ambos actuadores están activados.

En la figura 3.13 se muestra la rutina implementada para la esquina tipo B, es decir, la plataforma está en la cinta corta y pasa a la cinta larga. En esa rutina intervienen dos sensores inductivos y dos actuadores.

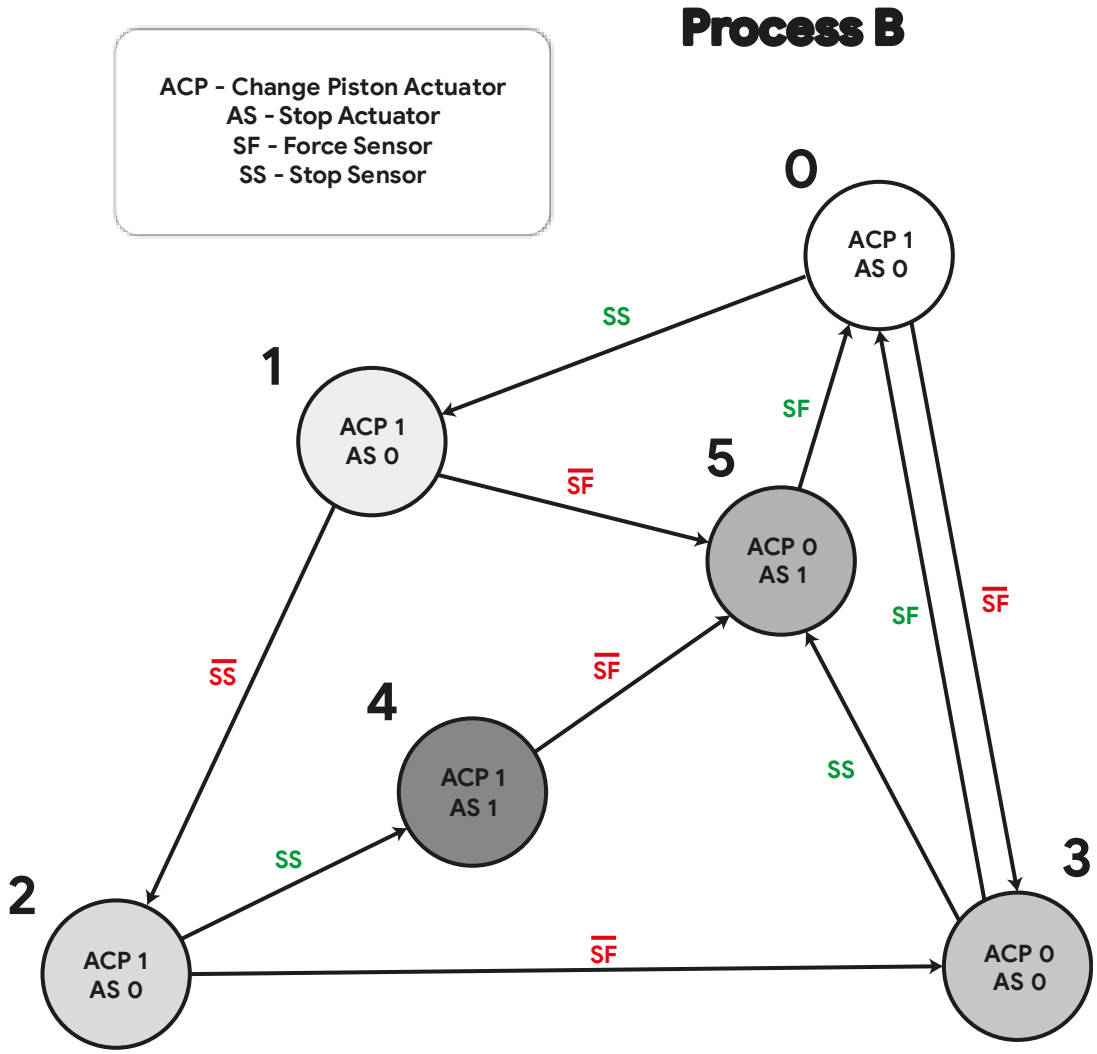


Figura 3.13 - Rutina de automatización tipo B

Los estados son similares a la rutina de automatización de tipo A.

El código de la implementación de la automatización interna se muestra a continuación.

```
/*
////////////////////////////////////////////////
    PROPERTY OF THE UNIVERSITY OF SEVILLE,
    PARTNER OF THE DENiM PROJECT

    CONTEXT:
    1. The routine of internal automation.
////////////////////////////////////////////////
*/

// LIBRARIES
using System;
using UnityEngine;

//////////////////////////////////////
// ROUTINE OF THE INTERNAL AUTOMATION //
//////////////////////////////////////
public class InternalAutomation : MonoBehaviour
{
    // DECLARATION OF GLOBAL VARIABLES
    private static int[] statevector = {0, 0, 0, 0, 0};
```

```
      // FUNCTIONS FOR THE CONTROL
      public static void AutomationProcess(ref AutomationParameters.Automation
Object[] Memory)
      {
          ControlProcessA( ref Memory[2], ref Memory[1], ref statevector[0]);
          ControlProcessA( ref Memory[3], ref Memory[0], ref statevector[1]);
          ControlProcessB( ref Memory[2], ref Memory[0], ref statevector[2]);
          ControlProcessB( ref Memory[3], ref Memory[1], ref statevector[3]);

          ControlCompress( ref Memory[4], ref statevector[4]);
      }

      // FUNCTIONS FOR THE CONTROL OF THE AIR COMPRESS
      private static void ControlCompress (ref AutomationParameters.Automation
Object AirTank, ref int state)
      {
          // Declaration and assignment of sensor variables
          bool[] Capacity = AutomationFunctions.ReturnSensorComponentByName(Ai
rTank.Sensors, "AirTank").flag;

          // Declaration of actuator variables
          bool Compress = false;

          // Previous status statement to optimize execution
          int beforeState = state;

          switch(state)
          {
              // The capacity is behind of the minimal value
              case 0:
              Compress = true;

              if (Capacity[1])
                  state = 3;
              else if (Capacity[0])
                  state = 1;
              break;

              // The capacity is between of the minimal value and the maximal
value filling up
              case 1:
              Compress = true;

              if (!Capacity[0])
                  state = 0;
              else if (Capacity[1])
                  state = 3;
              break;

              // The capacity is between of the minimal value and the maximal
value emptying
              case 2:
              Compress = false;

              if (!Capacity[0])
                  state = 0;
              else if (Capacity[1])
                  state = 3;
              break;

              // The capacity is in the maximal value
              case 3:
              Compress = false;
```

```
                if (!Capacity[0])
                    state = 0;
                else if (!Capacity[1])
                    state = 2;
                break;
            }

            // Assignment of values to actuators
            AutomationFunctions.ReturnActuatorComponentByName(AirTank.Actuators,
 "AirCompressor").flag[0] = Compress;

            // Recursive call with previous state
            if (beforeState != state)
                ControlCompress(ref AirTank, ref state);

        }

        // FUNCTION FOR THE CONTROL OF THIS BOX MOVEMENT: LONG CONVEYOR -
> SHORT CONVEYOR
        private static void ControlProcessA (ref AutomationParameters.Automation
Object LongBeltConveyorObject,
                                             ref AutomationParameters.Automation
Object ShortBeltConveyorObject,
                                             ref int state)

        {
            // Declaration and assignment of sensor variables
            bool StopSensor = AutomationFunctions.ReturnSensorComponentByName(Lo
ngBeltConveyorObject.Sensors, "InductiveSensor0").flag[0];
            bool ChangePistonSensor = AutomationFunctions.ReturnSensorComponentB
yName(LongBeltConveyorObject.Sensors, "InductiveSensor2").flag[0];
            bool InductiveSensor = AutomationFunctions.ReturnSensorComponentByNa
me(ShortBeltConveyorObject.Sensors, "InductiveSensor1").flag[0];

            // Declaration of actuator variables
            bool ChangePiston = false, StopActuator = false;

            // Previous status statement to optimize execution
            int beforeState = state;

            switch(state)
            {
                case 0:
                ChangePiston = false;
                StopActuator = false;
                if (StopSensor)
                    state = 1;
                else if (ChangePistonSensor)
                    state = 3;
                break;

                case 1:
                ChangePiston = false;
                StopActuator = false;
                if (ChangePistonSensor)
                    state = 5;
                else if (!StopSensor)
                    state = 2;
                break;

                case 2:
```

```
                ChangePiston = false;
                StopActuator = false;
                if (StopSensor)
                    state = 4;
                else if (ChangePistonSensor)
                    state = 3;
                break;

                case 3:
                ChangePiston = true;
                StopActuator = false;
                if (StopSensor)
                    state = 5;
                else if (InductiveSensor)
                    state = 0;
                break;

                case 4:
                ChangePiston = false;
                StopActuator = true;
                if (ChangePistonSensor)
                    state = 5;
                break;

                case 5:
                ChangePiston = true;
                StopActuator = true;
                if (InductiveSensor)
                    state = 0;
                break;
            }

            // Assignment of values to actuators
            AutomationFunctions.ReturnActuatorComponentByName(LongBeltConveyorOb
ject.Actuators, "ChangePistonModule1").flag[1] = ChangePiston;
            AutomationFunctions.ReturnActuatorComponentByName(LongBeltConveyorOb
ject.Actuators, "DynamicStopActuatorModule").flag[0] = StopActuator;

            // Recursive call with previous state
            if (beforeState != state)
                ControlProcessA(ref LongBeltConveyorObject, ref ShortBeltConveyo
rObject, ref state);
        }

        // FUNCTION FOR THE CONTROL OF THIS BOX MOVEMENT: SHORT CONVEYOR -
> LONG CONVEYOR
        private static void ControlProcessB (ref AutomationParameters.Automation
Object LongBeltConveyorObject,
                                             ref AutomationParameters.Automation
Object ShortBeltConveyorObject,
                                             ref int state)
        {
            // Declaration and assignment of sensor variables
            bool StopSensor = AutomationFunctions.ReturnSensorComponentByName(Sh
ortBeltConveyorObject.Sensors, "InductiveSensor0").flag[0];
            bool ForceSensor = AutomationFunctions.ReturnSensorComponentByName(L
ongBeltConveyorObject.Sensors, "InductiveSensor1").flag[0];

            // Declaration of actuator variables
            bool ChangePiston = false, StopActuator = false;

            // Previous status statement to optimize execution
            int beforeState = state;
```

```
            switch(state)
            {
                case 0:
                ChangePiston = true;
                StopActuator = false;
                if (StopSensor)
                    state = 1;
                else if (!ForceSensor)
                    state = 3;
                break;

                case 1:
                ChangePiston = true;
                StopActuator = false;
                if (!StopSensor)
                    state = 2;
                else if (!ForceSensor)
                    state = 5;
                break;

                case 2:
                ChangePiston = true;
                StopActuator = false;
                if (!ForceSensor)
                    state = 3;
                else if (StopSensor)
                    state = 4;
                break;

                case 3:
                ChangePiston = false;
                StopActuator = false;
                if (ForceSensor)
                    state = 0;
                else if (StopSensor)
                    state = 5;
                break;

                case 4:
                ChangePiston = true;
                StopActuator = true;
                if (!ForceSensor)
                    state = 5;
                break;

                case 5:
                ChangePiston = false;
                StopActuator = true;
                if (ForceSensor)
                    state = 0;
                break;
            }

            // Assignment of values to actuators
            AutomationFunctions.ReturnActuatorComponentByName(LongBeltConveyorOb
ject.Actuators, "ChangePistonModule0").flag[1] = ChangePiston;
            AutomationFunctions.ReturnActuatorComponentByName(ShortBeltConveyorO
bject.Actuators, "DynamicStopActuatorModule").flag[0] = StopActuator;

            // Recursive call with previous state
            if (beforeState != state)
```

```
            ControlProcessB(ref LongBeltConveyorObject, ref ShortBeltConveyo
rObject, ref state);
        }
    }
    ////////////////////////////////////////
```

Dos matices para resaltar en el código:

1. El método *AutomationProcess()* es el que se llamaba desde la asignación de un sensor. De hecho, esta clase no contiene ningún método de autoejecución (Update o FixedUpdate). Como se dijo anteriormente, se ha implementado de esta manera para que solo se ejecute cuando sea necesario. Queda pendiente en la optimización, no solo ejecutarse cuando se cambie el valor de un sensor, sino lanzar solo la rutina que se vea afectado por ese sensor.
2. Para traer el valor de los sensores y grabar los de los actuadores, se utilizan los métodos vistos anteriormente de traer de la memoria (inicializada en el programa principal del autómata) un sensor o un actuador por el nombre.

La rutina de automatización externa se encarga de, mediante la comunicación abierta, traer los valores de los actuadores y guardarlos en la memoria. Esos valores son calculados y enviados, en tiempo real, por el autómata de la planta. La gran diferencia con la interna es que las rutinas de automatización son ejecutadas externamente por el autómata.

A continuación, se muestra el script *ExternalAutomation.cs* que implementa la rutina de la automatización externa.

```
/*
////////////////////////////////////////////////
    PROPERTY OF THE UNIVERSITY OF SEVILLE,
    PARTNER OF THE DENiM PROJECT

    CONTEXT:
    1. The routine of external automation.
////////////////////////////////////////////////
*/

// LIBRARIES
using System;
using UnityEngine;

////////////////////////////////////////
// ROUTINE OF THE EXTERNAL AUTOMATION //
////////////////////////////////////////
public class ExternalAutomation : MonoBehaviour
{
    // FUNCTION THAT IS ACTIVATED IMMEDIATELY AFTER ALL FUNCTIONS AWAKE WERE
 ENDED
    void Start()
    {
        Opc.Ua.Client.Implementation.CreateSubscriptionForChildren
        (
            CommunicationParameters.ActuatorFolderName,
            AutomationParameters.ExternalAutomationInterval,
            AutomationProcess
        );
    }

    // FUNCTION FOR THE CONTROL THAT EXECUTE WHEN A NOTIFICATION ARRIVED
    public static void AutomationProcess(Opc.Ua.Client.MonitoredItem monitor
edItem, Opc.Ua.Client.MonitoredItemNotificationEventArgs e)
    {
        try
```

```
            {
                Opc.Ua.MonitoredItemNotification notification = e.NotificationVa
lue as Opc.Ua.MonitoredItemNotification;

                // Change the memory value with the notification
                AutomationFunctions.ReturnActuatorComponentByNumber(
                    Automation.Memory,
                    Convert.ToInt32(monitoredItem.DisplayName.TrimEnd(Communicat
ionParameters.Nums).TrimEnd('.'))
                ).flag[Convert.ToInt32(monitoredItem.DisplayName.TrimStart(Commu
nicationParameters.Nums).TrimStart('.'))] = notification.Value.GetValue<bool>
(false);
            }

            catch (Exception ex)
            {
                Debug.Log("OnMonitoredItemNotification error: " + ex.Message);
            }
        }
    }
    ////////////////////////////////////////
```

El proceso que sigue la rutina comienza creando una subscripción (se explicará más adelante) a cada variable del servidor que contiene el valor de un actuador calculado por el autómata de la planta. Esto se realiza en el método *Start()*.

El método *AutomationProcess()* se ejecuta cada vez que algún valor es modificado en el servidor. El servidor, mediante la subscripción creada, avisa al gemelo y le pasa el valor nuevo. Con esto se consigue, al igual que en la automatización interna, que solo se ejecute cuando sea necesario.

Nótese que este método, al hacer referencia a una comunicación con posibles fallos se ejecuta mediante la sentencia *try*, para evitar que el sistema se colapse si la comunicación falle. Además, se permite crear manejadores de fallos, aunque actualmente solo está implementado un aviso.

Es difícil que el lector pueda deducir, sin ver todos los códigos, por qué se utilizan las funciones start y awake en distintas ocasiones. La causa es dejar la función awake (no siempre) solo para los modelos funcionales, para que puedan implementar ellos primero los componentes *Gates*, para que, posteriormente se les pueda identificar por esos componentes.

3.5. Autoaprendizaje

Todos los módulos implementados en el gemelo digital, salvo el supervisor, son dinámicos, es decir, tienen unos parámetros variables que los caracterizan y los adaptan al sistema real (en el caso de los módulos de modelos funcionales) o a las circunstancias queridas en ese momento.

Para ello, todos los módulos cuentan con una clase llamada ModuleName*Support.cs* que contiene todos los parámetros que caracterizan a ese módulo (como se ha visto en el caso de la automatización, aunque tuviese pocos parámetros).

Para entrenar los parámetros de los modelos funcionales hace falta, entre otras cosas, los valores de los sensores incorporados en la planta. Cualquier sensor tiene una incertidumbre asociada. Para solucionar errores en el entrenamiento debidos a esta incertidumbre, se está desarrollando un filtro para todos los sensores. Éste está siendo desarrollado por otros miembros del equipo, que no son el autor. Decir simplemente que se está implementando en python, y que la comunicación con python ya está implementada, y se realiza mediante archivos JSON, como se puede ver en la arquitectura del Anexo I.

Parte de la información del autoaprendizaje llega mediante la comunicación implementada. Ésta se verá en el siguiente paso.

El último paso del autoaprendizaje es implementar dos tipos de algoritmos de autoaprendizaje:

1. El primer tipo se trata de un conjunto de algoritmos explícitos para cada modelo. Un ejemplo es el utilizado para ajustar la velocidad de la cinta transportadora: el sistema real y el gemelo digital se lanzan en paralelo. El gemelo funcionando con la automatización externa. El algoritmo recibe, mediante la comunicación implementada, el momento que deben activarse los pistones de cambio. Mediante análisis matemático se puede deducir la velocidad de la cinta.
   Este conjunto de algoritmos tiene la ventaja de que los algoritmos son sencillos de diseñar e implementar, siempre y cuando los sensores puestos en el sistema lo permitan.
2. La desventaja más grande del primer conjunto es su falta de generalización. Esto va en contra de la escalabilidad requerida para el gemelo. Este segundo tipo se trata de un solo algoritmo que sea capaz de ajustar todos los parámetros de todos los módulos. Para poder realizarlo es imprescindible unificar los modelos para que el algoritmo pueda trabajar con ellos. Esto resolvería el problema de la escalabilidad, pero todavía se está investigando para llegar a una posible implementación.

El autoaprendizaje estará en todo momento posible funcionando, es decir, en cualquier momento que esté funcionando síncronamente con el sistema real y esté comunicado con él; o en parada, es decir, con datos históricos. Para esa segunda posibilidad, el usuario deberá activarlo y proporcionar, mediante la comunicación, la información requerida.

Para acabar este paso, nombrar que la implementación de la solución al problema de la identificación mediante un algoritmo adaptativo a todos los subsistemas está en proceso de investigación. En la explicación teórica de este paso se puede encontrar más información. Hasta que no se implemente el segundo tipo de algoritmo de autoaprendizaje, no será posible aplicar esta solución a la identificación.

3.6. Conectividad

En este paso se nombrará toda la comunicación implementada en el modelo. Toda la conectividad implementada hasta ahora ha sido desarrollada por el presente autor.

El contexto del sistema real es un entorno industrial. Dentro de este entorno existen muchos protocolos estándar. Entre ellos se encuentran PROFINET, Industrial Ethernet, PROFIBUS, etc. Entre ellos se encuentra un protocolo llamado OPC UA.

Citando textualmente de [12], “OPC UA es la evolución de la tecnología OPC Clásica: es una tecnología de comunicación industrial multiplataforma, abierta, orientada a servicios, segura, y con ricos modelos de información. Como el OPC Clásico, se trata de un protocolo de comunicación pensado para comunicar datos de equipos industriales, pero, su principal diferencia con éste es que no se limita sólo a comunicar datos entre aplicaciones SCADA y sensores, sino que su objetivo es ir más allá y que pueda comunicarse con todas las aplicaciones de la empresa y a través de todas las capas empresariales.”

Las ventajas más relevantes de utilizar este protocolo son:

1. Es escalable, como se requiere en la metodología que se está aplicando.
2. Es una tecnología independiente de los proveedores. Esto hace que no se esté atado a ninguno.
3. El mantenimiento de una comunicación mediante este protocolo es muy sencillo, comparado con otros protocolos de comunicación industrial, como los mencionados anteriormente.

Por estos beneficios se ha decidido implementar este protocolo en este gemelo digital. La implementación se ha realizado de forma genérica, para que esté módulo se pueda utilizar en otros gemelos.

El protocolo se ha implementado solo desde la parte del cliente, de manera que el servidor siempre esté afuera. No tiene sentido implementar la tecnología OPC en el gemelo si antes no existía en la realidad. Por tanto, siempre existirá un servidor externo al que accederá el gemelo.

Para implementar el cliente Opc Ua son necesarios unos plugins gratuitos que se encuentran en el repositorio específico para Opc Ua de la .NETStandard [13]. Estos pligins se integran de manera automática en Unity3d. Estos son:

- *BouncyCastle.Crypto.dll*
- *Newtonsoft.Json.dll*
- *Opc.Ua.Client.dll*
- *Opc.Ua.Configuration.dll*
- *Opc.Ua.Core.dll*

En primer lugar, se mostrará el script de comunicación. Este script intermedio está pensado para gobernar toda la comunicación, para que se implementen en él futuros protocolos. Así se consigue aislar todos los protocolos de comunicación y poder elegir fácilmente el protocolo que se requiera. Este script se llama *Communication.cs* y se puede ver en las siguientes líneas.

```
/*
////////////////////////////////////////////////
    PROPERTY OF THE UNIVERSITY OF SEVILLE,
    PARTNER OF THE DENiM PROJECT

    CONTEXT:
    1. This program contains the communication process.
////////////////////////////////////////////////
*/

// LIBRARIES
using System;
using UnityEngine;

//////////////////////////
// COMMUNICATION PROCESS //
//////////////////////////
public class Communication : MonoBehaviour
{
    // FUNCTION THAT IS INIT CONNECTION WITH SERVER
    void Awake()
    {
        try
        {
            Opc.Ua.Client.Implementation.InitConnectionWithServer
            (
                CommunicationParameters.Name,
                CommunicationParameters.Identity,
                CommunicationParameters.ServerURL,
                CommunicationParameters.OperationTimeout,
                CommunicationParameters.SessionTimeout
            );
        }
        catch (Exception ex)
        {
            Console.WriteLine("Connection error: {0}", ex.Message);
        }
    }
}
//////////////////////////
```

Como se ve en el código, actualmente solo inicia la comunicación con el servidor. Como se ha mencionado anteriormente, para tratar la comunicación se utilizan métodos precedidos de *try*. Los parámetros que necesita el protocolo Opc Ua implementado se describirán cuando se explique el protocolo.

Este módulo también consta de sus parámetros correspondientes, alojados en el script *CommunicationSupport.cs*. Se muestra el código de este script a continuación.

```
/*
////////////////////////////////////////////////
```

```
        PROPERTY OF THE UNIVERSITY OF SEVILLE,
        PARTNER OF THE DENiM PROJECT

        CONTEXT:
        1. All of the parameters that defines the communication.
        2. Functions for the communication.
    ////////////////////////////////////////////////
    */

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

    ////////////////////////////////
    // PARAMETER FOR COMMUNICATION //
    ////////////////////////////////
    public class CommunicationParameters : MonoBehaviour
    {
        // IDENTIFICATION AND CONNECTION PARAMETERS
        public static string[] Identity = new string[] {"DigitalTwinUser", "DENi
M"};
        public const int OperationTimeout = 15000, SessionTimeout = 60000;
        public const string ServerURL = "opc.tcp://PC-
JGJ:53530/OPCUA/SimulationServer";
        public const string Name = "DigitalTwin";

        // SERVER PARAMETERS
        public const string ActuatorFolderName = "Actuators";
        public const string SensorFolderName = "Sensors";

        // USEFUL VECTORS FOR REMOVE NUM OF SENSOR/ACTUATOR NAMES
        public static char[] Nums = {'0','1','2','3','4','5','6','7','8','9'};
    }
    ////////////////////////////////
```

Todos los parámetros que aparecen hacen referencia al protocolo Opc Ua. En orden, significan:

- Identity: usuario y contraseña dado por el servidor requeridos para establecer la comunicación con éste.
- OperationTimeout: intervalo de tiempo sin respuesta que se espera entre operaciones para abortar.
- SessionTimeout: intervalo de tiempo sin transmisión alguna que se espera para abortar la comunicación.
- ServerURL: dirección del servidor.
- ActuatorFolderName y SensorFolderName: nombres de las carpetas donde se encuentran los valores de los sensores y actuadores.
- Nums: es un vector útil para algunos métodos que se han implementado que funcionan con los nombres de las variables del servidor.

A continuación, se explicará el código que implementa el protocolo Opc Ua. El script se llama *ClientOpcUa.cs*. Se muestra a continuación.

```
    /*
    ////////////////////////////////////////////////
        PROPERTY OF THE UNIVERSITY OF SEVILLE,
        PARTNER OF THE DENiM PROJECT

        CONTEXT:
        1. The implementation of the protocol OPC UA.
    ////////////////////////////////////////////////
```

```
  */

  using UnityEngine;
  using System;

  ////////////////////////////////////////////////////
  // NAMESPACE FOR ALL THE LIBRARIES OF OPC UA PROTOCOL //
  ////////////////////////////////////////////////////
  namespace Opc.Ua.Client
  {
      ////////////////////
      // PROTOCOL OPCUA //
      ////////////////////
      public class Implementation : MonoBehaviour
      {
          // DECLARATION OF GLOBAL VARIABLES
          private static Session session;

          // FUNCTION THAT INITIALIZES ALL THE VARIABLES OF THE AUTOMATIC PROG
RAM
          public static async void InitConnectionWithServer(string Application
Name, string[] Identity, string ServerURL, int OperationTimeout, int SessionT
imeout)
          {
              ////////////////////// Step 1 -
 Create a config ////////////////////
              ApplicationConfiguration Config = new ApplicationConfiguration()
              {
                  ApplicationName = ApplicationName,
                  ApplicationUri = Utils.Format(@"urn:{0}:" + ApplicationName,
 System.Net.Dns.GetHostName()),
                  ApplicationType = ApplicationType.Client,
                  SecurityConfiguration = new SecurityConfiguration {
                      ApplicationCertificate = new CertificateIdentifier { Sto
reType = @"Directory", StorePath = @"%CommonApplicationData%\DENiMProject\Cer
tificateStores\MachineDefault", SubjectName = "DigitalTwin"},
                      TrustedIssuerCertificates = new CertificateTrustList { S
toreType = @"Directory", StorePath = @"%CommonApplicationData%\DENiMProject\C
ertificateStores\UA Certificate Authorities" },
                      TrustedPeerCertificates = new CertificateTrustList { Sto
reType = @"Directory", StorePath = @"%CommonApplicationData%\DENiMProject\Cer
tificateStores\UA Applications" },
                      RejectedCertificateStore = new CertificateTrustList { St
oreType = @"Directory", StorePath = @"%CommonApplicationData%\DENiMProject\Ce
rtificateStores\RejectedCertificates" },
                      AutoAcceptUntrustedCertificates = true
                  },
                  TransportConfigurations = new TransportConfigurationCollecti
on(),
                  TransportQuotas = new TransportQuotas { OperationTimeout = O
perationTimeout },
                  ClientConfiguration = new ClientConfiguration { DefaultSessi
onTimeout = SessionTimeout },
                  TraceConfiguration = new TraceConfiguration()
              };
              await Config.Validate(ApplicationType.Client);
              if (Config.SecurityConfiguration.AutoAcceptUntrustedCertificates
)
              {
                  Config.CertificateValidator.CertificateValidation += (s, e)
=> { e.Accept = (e.Error.StatusCode == StatusCodes.BadCertificateUntrusted);}
;
```

```
            }

            ////////////////////// Step 2 -
 Create a certificate for this application and create a user ////////////////
////
            CreateClientApplicationCertificate(Config.ApplicationName, Confi
g, 2048);
            UserIdentity userIdentity = new UserIdentity(Identity[0], Identi
ty[1]);

            ////////////////////// Step 3 -
 Configure the endpoint ////////////////////
            EndpointDescription selectedEndpoint = CoreClientUtils.SelectEnd
point(ServerURL, useSecurity : true, SessionTimeout);
            selectedEndpoint.SecurityPolicyUri = Opc.Ua.SecurityPolicies.Bas
ic256Sha256;
            ConfiguredEndpoint EndPoint = new ConfiguredEndpoint(null, selec
tedEndpoint, EndpointConfiguration.Create(Config));

            ////////////////////// Step 4 -
 Create a session with your server ////////////////////
            session = await Session.Create(Config, EndPoint, true, Applicati
onName + "Session", Convert.ToUInt32(SessionTimeout), userIdentity, null);
        }

        // FUNCTION THAT CREATE A CLIENT CERTIFICATE FOR THIS APPLICATION IF
 NOT EXIST
        private static async void CreateClientApplicationCertificate(string
 AplicationName, ApplicationConfiguration Config, ushort MinKeySize)
        {
            Configuration.ApplicationInstance application = new Configuratio
n.ApplicationInstance
            {
                ApplicationName = AplicationName,
                ApplicationType = ApplicationType.Client,
                ApplicationConfiguration = Config
            };
            await application.CheckApplicationInstanceCertificate(false, Min
KeySize); // Create the certificate
        }

        // FUNCTION THAT CREATE A SUBSCRIPTION FOR CHILDREN OF A FATHER OBJE
CT
        public static void CreateSubscriptionForChildren(string FatherName,
 int Interval, MonitoredItemNotificationEventHandler OnNotificationFunction)
        {
            try
            {
                ReferenceDescriptionCollection refs, ChildRefs = null;
                byte[] cp, ChildCp;
                session.Browse(null, null, ObjectIds.ObjectsFolder, 0u, Brow
seDirection.Forward, ReferenceTypeIds.HierarchicalReferences, true, (uint)Nod
eClass.Variable | (uint)NodeClass.Object | (uint)NodeClass.Method, out cp, ou
t refs);

                foreach (var rd in refs)
                {
                    if (rd.DisplayName == FatherName)
                    {
                        session.Browse(
                            null,
                            null,
```

```
                                ExpandedNodeId.ToNodeId(rd.NodeId, session.Names
paceUris),
                                0u,
                                BrowseDirection.Forward,
                                ReferenceTypeIds.HierarchicalReferences,
                                true,
                                (uint)NodeClass.Variable | (uint)NodeClass.Objec
t | (uint)NodeClass.Method,
                                out ChildCp,
                                out ChildRefs);

                            SubscribeToDataChanges(
                                FatherName + "Subscription",
                                Interval,
                                ChildRefs,
                                OnNotificationFunction);

                            break;
                        }
                    }
                }

                catch (Exception ex)
                {
                    Console.WriteLine("Subscribe error: {0}", ex.Message);
                }
            }

            // FUNCTION THAT CREATE A SUBSCRIPTION
            private static void SubscribeToDataChanges(string SubscriptionName,
int Interval, ReferenceDescriptionCollection ChildRefs, MonitoredItemNotifica
tionEventHandler OnNotificationFunction)
            {
                if (session == null || session.Connected == false)
                {
                    Console.WriteLine("Session not connected!");
                    return;
                }

                try
                {
                    // Define Subscription parameters and created it on the Serv
er side
                    Subscription subscription = new Subscription(session.Default
Subscription);
                    subscription.DisplayName = SubscriptionName;
                    subscription.PublishingEnabled = true;
                    subscription.PublishingInterval = Interval;
                    session.AddSubscription(subscription);
                    subscription.Create();

                    // Create MonitoredItems for data changes
                    foreach (var child in ChildRefs)
                    {
                        MonitoredItem monitoredItem = new MonitoredItem(subscrip
tion.DefaultItem);
                        monitoredItem.StartNodeId = ExpandedNodeId.ToNodeId(chil
d.NodeId, null);
                        monitoredItem.AttributeId = Attributes.Value;
                        monitoredItem.DisplayName = child.DisplayName.ToString()
;
                        monitoredItem.SamplingInterval = Interval;
```

```
                    monitoredItem.Notification += OnNotificationFunction;
                    subscription.AddItem(monitoredItem);
                }

                // Create the monitored items on Server side
                subscription.ApplyChanges();
            }

            catch (Exception ex)
            {
                Console.WriteLine("Subscribe error: {0}", ex.Message);
            }
        }
    }
    ////////////////////
}
////////////////////////////////////////////////////////
```

Lo primero que nos encontramos es un marco nominal o un espacio de nombres. Esto es útil si dentro de ese marco se utilizan métodos y variables de otros programas que se encuentran en esa dirección. Como es el caso, dado que todos los plugins instalados se encuentran en esa dirección, se ha utilizado el espacio de nombres *Opc.Ua.Client*.

Dentro de este marco se encuentran cuatro métodos:

- El primero de ellos es el método que inicia la comunicación. Es un método asíncrono, lo que implica que se ejecutará sin tener en cuenta el FixedTime. En este caso, en el script *Automation.cs*, se llama a este método de manera que se espera a que termine para continuar. Los parámetros que necesitan son: el nombre de la aplicación, el usuario y contraseña, la dirección del seridor y los intervalos de tiempo límite. La comunicación se establece en cuatro pasos:
    1. Se crea y se valida la configuración para la aplicación del gemelo digital. Dentro de la configuración se encuentra la configuración de la seguridad. Se ha configurado para que tenga la máxima posible: se necesita un usuario y contraseña, aparte de un certificado de seguridad de aplicación. Los certificados se guardan en una carpeta común en el sistema operativo Windows: *CommonApplicationData*. Dentro de este directorio se crea una carpeta para identificar correctamente los certificados creados por el gemelo digital.
    2. Se crea el certificado de seguridad y la identidad del usuario con los datos suministrados.
    3. Se configura y se crea el *endpoint* con la configuración creada en el primer paso.
    4. Por último, se crea la sesión con el servidor.
- El segundo método crea el certificado de la aplicación.
- El tercer método crea, para un objeto dado, una subscripción única para todos sus hijos y una monitorización para cada uno de ellos. Una subscripción es una manera que existe dentro de este protocolo para avisar al servidor de que un cliente está interesado en unas variables concretas. Acoplado a una subscripción hay un método que se ejecuta en el cliente cada vez que se recibe una notificación de esa subscripción. Una subscripción, se compone de monitorizaciones. Una monitorización está asociada a una variable concreta. El servidor notifica al cliente cuando una variable monitorizada se modifica. Dentro de esa notificación se encuentra el nombre de la variable que se ha modificado y su valor nuevo. Esto es muy útil, dado que permite que el cliente no esté continuamente preguntando al servidor si ha cambiado una variable.
  Los parámetros de este método son: el nombre del padre (el nombre de la carpeta de la cual se va a crear una subscripción), el intervalo de tiempo mínimo entre notificaciones y el método acoplado a esta subscripción.

  Los pasos que se realizan son: primero explora el servidor y al encontrar la carpeta con el nombre del padre llama al último método.

- El último método, embebido en el anterior, crea la subscripción mencionada y las monitorizaciones correspondientes.

De todos los módulos explicados hasta ahora, hay uno que llama al tercer método: la automatización externa. Todo lo dicho sobre las subscripciones y monitorizaciones es muy útil para la automatización externa, dado que solo se ejecuta la rutina de automatización (guardar los valores de los actuadores en la memoria) cada vez que cambia una variable.

Tal y como se ha implementado el protocolo Opc Ua, solo hace falta llamar al tercer método para que se creen automáticamente todas las herramientas necesarias para implementar la comunicación mediante este protocolo.

Para probar el funcionamiento correcto de la comunicación y de la automatización externa se ha simulado un servidor Opc Ua utlizando el software Prosys OPC UA Simulation Server.

A continuación, se muestran varias capturas de este software en el momento de la conexión con el gemelo digital.

La figura 3.14 muestra la pestaña inicial del software donde se puede ver el estado del servidor y su dirección (en este caso solo se ha configurado la conexión por TCP).

Figura 3.14 - Pestaña Status de Prosys OpcUa

En el servidor se han creado unas variables para simular el comportamiento de los actuadores y sensores de la planta real. Todas esas variables se pueden observar en la pestaña *Objects* (en la figura 3.15 se visualiza esta pestaña). La foto captura el momento en que el actuador 2 está a *true*. Los números corresponden a la variable number que se encuentra en el script *Gate actuator.cs* instanciado por cada actuador. Como se puede observar, hay algunos actuadores que tienen dos objetos asociados. Es el caso de los pistones de cambio: el primero corresponde al movimiento de su propia cinta (realmente es dependiente de las cintas cortas, dado que el motor que las mueve es el mismo que las cintas cortas; se ha querido mantener este grado de libertad para su posible optimización de la energía) y el segundo a la subida del pistón.

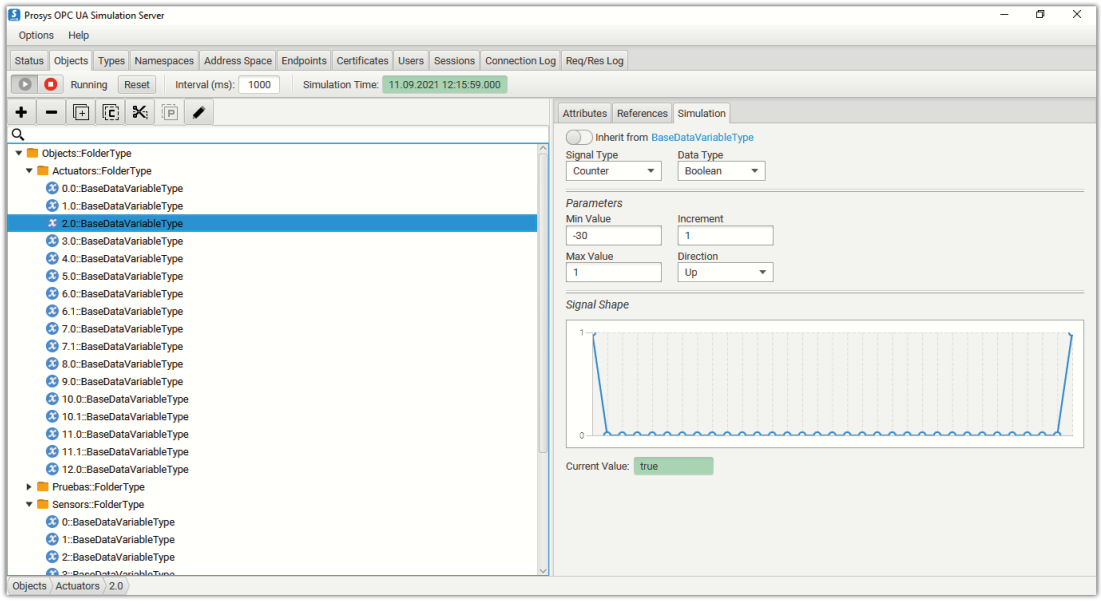


Figura 3.15 - Pestaña Objects de Prosys OpcUa

La figura 3.16 muestra los certificados de seguridad aceptados por el servidor hasta el momento. Seleccionado se encuentra el certificado que utiliza el gemelo digital como cliente.

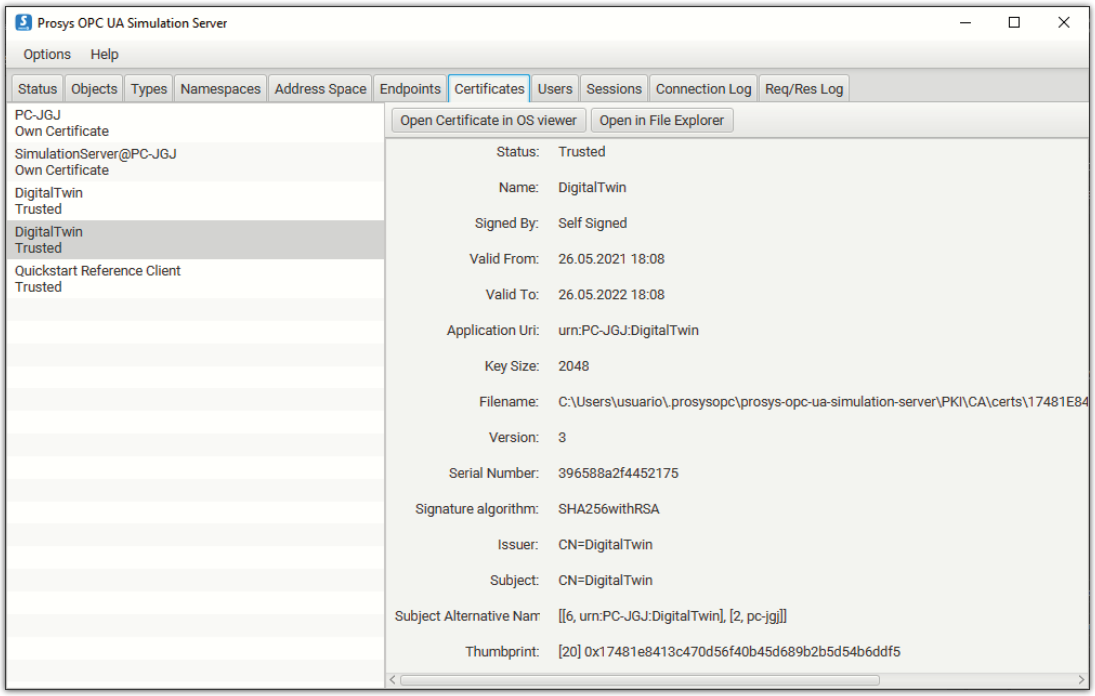


Figura 3.16 - Pestaña Certificates de Prosys OpcUa

La pestaña que contiene las sesiones abiertas en ese instante se muestra en la figura 3.17. Dentro de la sesión se puede observar la configuración que se explicó anteriormente.

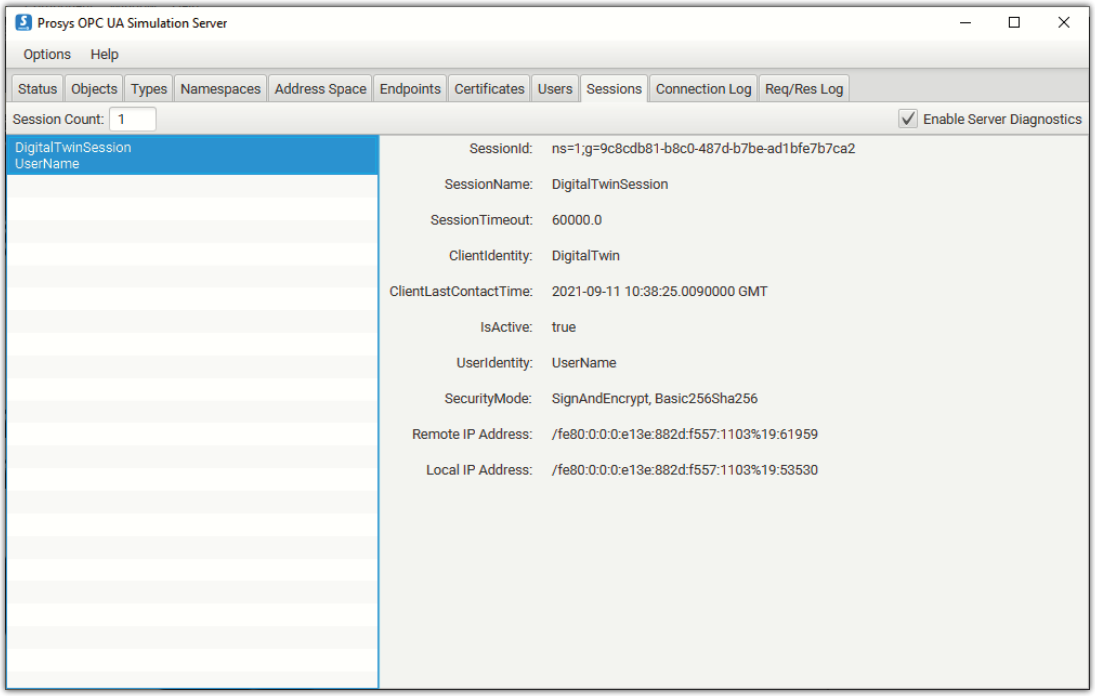


Figura 3.17 - Pestaña Sessions de Prosys OpcUa

Hay una pestaña que muestra el registro de todas las conexiones hechas desde que se lanzó el servidor. En la figura 3.18 se puede observar los cuatro pasos típicos que ocurren en cualquier conexión: creación de la sesión, activando la sesión, sesión activada (en este momento la conexión está perfectamente establecida y comienza la transacción de datos) y el cierre de sesión (cuando se ha concluido la conexión o se ha abortado).

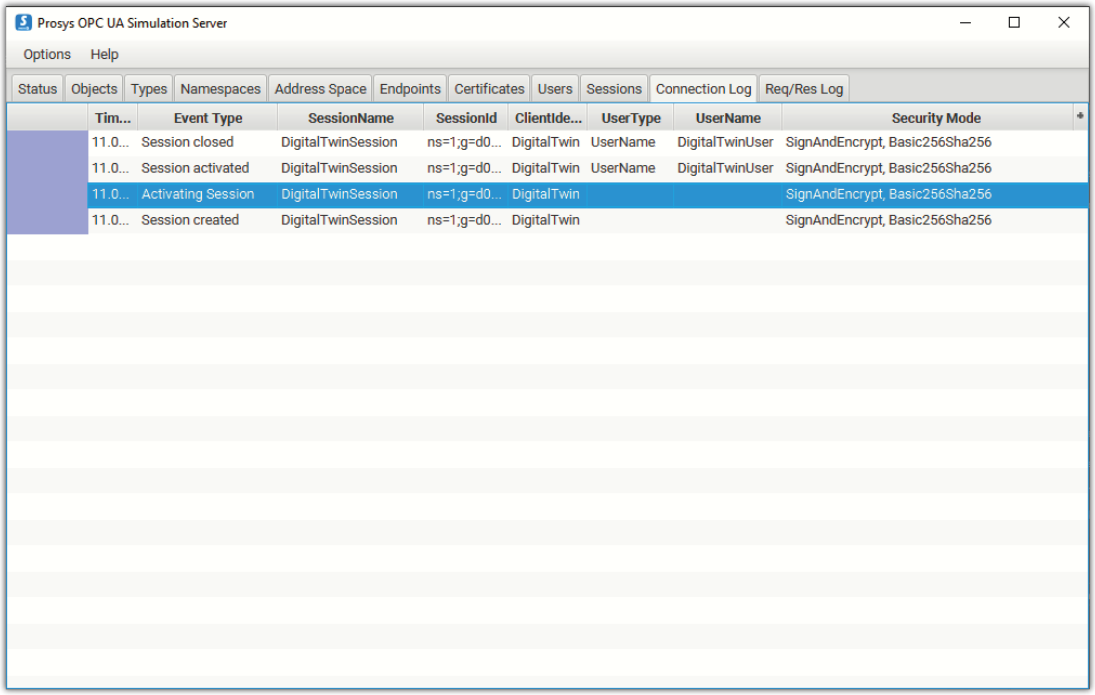


Figura 3.18 - Pestaña Connection Log de Prosys OpcUa

Por último, en la figura 3.19 se muestra el registro de todos los mensajes existentes en la conexión. Se puede ver cómo se han ido enviando los valores de los sensores y actuadores al gemelo digital.

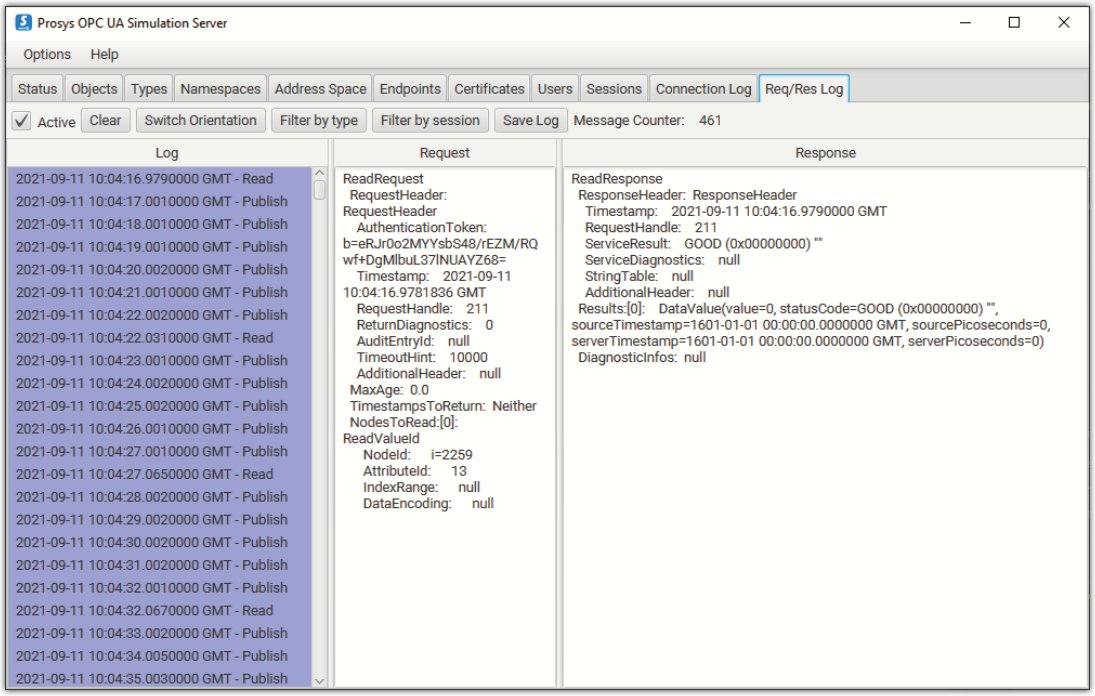


Figura 3.19 - Pestaña Req/Res Log de Prosys OpcUa

Se ha comprobado mediante el visor de Unity3d que los valores de los objetos en el servidor afectaban correctamente a las variables del gemelo digital.

3.7. Servicios

Este paso del método es el más abiertos de todos, debido que es el paso más dependiente del objetivo de la implentación del gemelo digital. Además, es el paso que nunca se da por terminado, dado que una vez construido el gemelo, los servicios se van añadiendo cuando se van necesitando. Actualmente hay dos campos de servicios que se están implementando en este gemelo digital.

El primer campo consiste en todo lo referido a la interfaz del usuario. El usuario debe tomar múltiples decisiones, aparte de poder tener acceso al gemelo digital, para futuros servicios, como es el mantenimiento de la planta guiado por el modelo visual. Algunas de estas decisiones han ido saliendo a lo largo del presente proyecto. La interfaz de usuario se está implementando para dos tipos de visualizaciones:

1. En el propio ordenador donde se ejecuta el gemelo digital.
2. Mediante unas gafas de realidad virtual.

Esta segunda visualización abre las puertas a multitud de servicios (como el dicho sobre el mantenimiento). Se está implementando en las gafas de realidad virtual Oculus Quest, tanto internamente como mediante el uso de Oculus Link [14].

Entre otras cosas, el usuario, mediante esta interfaz, podrá elegir qué servicio utilizar, tomar decisiones sobre el autoaprendizaje y la comunicación, lanzar la automatización externa o interna, etc.

El presente autor no está involucrado en este primer campo de servicios, pero sí en el segundo: la optimización de la energía de la planta.

Uno de los objetivos fundamentales de la implementación de este gemelo digital es conseguir reducir la energía consumida en todo el proceso. Para ello, el primer paso ha sido crear un módulo sobre la energía dentro del gemelo digital. Este módulo contiene los modelos energéticos de todos los elementos de la planta y las variables que guardan la energía consumida hasta el momento por cada elemento.

El módulo contiene dos scripts. El primero, que se muestra a continuación, contiene las variables globales que almacenan la energía. Este script se denomina *PowerConsumption.cs*.

```
/*
////////////////////////////////////////////////
    PROPERTY OF THE UNIVERSITY OF SEVILLE,
    PARTNER OF THE DENiM PROJECT

    CONTEXT:
    1. All the consumption powers
////////////////////////////////////////////////
*/

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

//////////////////////////////////////////
// VARIABLES THAT STORE THE POWER CONSUMED //
//////////////////////////////////////////
public class PowerConsumption : MonoBehaviour
{
    public static double AirCompressor;
    public static double BeltMotors;
}
//////////////////////////////////////////
```

Nótese que de todos los elementos que se han modelado hasta ahora solo dos consumen potencia eléctrica: los motores de las cintas transportadoras y el compresor de aire.

El segundo script contiene los modelos energéticos. Actualmente, en el gemelo, están implementados unos modelos lineales para verificar la arquitectura, pero los modelos reales están en proceso de implementación, por otro componente del equipo de desarrollo de DENiM.

```
/*
////////////////////////////////////////////////
    PROPERTY OF THE UNIVERSITY OF SEVILLE,
    PARTNER OF THE DENiM PROJECT

    CONTEXT:
    1. All the consumption powers
////////////////////////////////////////////////
*/

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

////////////////////////////////////////////////
// MODELS THAT REALIZE THE CONSUMPTION DYNAMICS //
////////////////////////////////////////////////
public class ConsumptionModels : MonoBehaviour
{
    public static void AirCompressor()
    {
        PowerConsumption.AirCompressor += 0.1;

    }

    public static void BeltMotors()
    {
        PowerConsumption.BeltMotors += 0.01;
    }
}
////////////////////////////////////////////////
```

Estas funciones son ejecutadas desde los modelos funcionales al ejecutarse.

Otro servicio en proceso de investigación, llevado a cabo por el autor, es la optimización del control del compresor de aire.

Los pistones funcionan mediante la compresión de aire en un depósito, que se va vaciando con la acción de éstos. Además, el depósito tiene una fuga permanente pequeña comparada con el anterior consumo, aunque los pistones estén inactivos. Existe una presión mínima necesaria para el funcionamiento correcto de los pistones.

Para mantener la presión adecuada es necesaria la acción continua de un compresor, en los momentos adecuados. Hay que recordar que el control llevado a cabo por el gemelo digital es de tipo todo o nada, llevado a cabo por la automatización interna.

El compresor es una fuente de consumo importante en este proceso de fabricación, por eso se ha decidido optimizar el control de éste, consistiendo esto en el desarrollo del proyecto que describe esta memoria.

En primer lugar, se ha desarrollado un modelo del compresor. El modelo se ha basado en la información proporcionada por un artículo científico [14]. En resumen, este artículo trata, en primer lugar, del modelo analítico de un compresor que, mediante una serie de suposiciones, se obtiene un modelo linealizado. Posteriormente se implementa un Fuzzy para su control y se muestran los resultados.

Solo se ha obtenido del artículo científico el modelo lineal del compresor.

El modelo del compresor consiste en un conjunto de funciones de transferencia dependientes de la carga del compresor, que se van unificando. En la figura 3.20 que se muestra a continuación se puede ver el conjunto.

LOAD	1/K	Transfer function of the generalized objects
37%	0.04	$\frac{0.023}{s^2+0.071s+0.022} \bullet \frac{1.1}{(1+56.6s)^5}$
50%	0.081	$\frac{0.033}{s^2+0.08s+0.034} \bullet \frac{1.12}{(1+42.1s)^4}$
75%	0.15	$\frac{0.05115}{s^2+0.1s+0.05365} \bullet \frac{1.2}{(1+27.1s)^4}$
100%	0.3	$\frac{0.063}{s^2+0.11s+0.067} \bullet \frac{1.276}{(1+18.4s)^3}$

Figura 3.20 - Tabla con el conjunto de funciones de transferencia

Este conjunto se ha implementado en Simulink para poder simular. Además, para el desarrollo de los controladores, se ha decidido trabajar en torno a un punto de trabajo del 50% de la carga.

En la figura 3.21 se muestra el esquema del modelo completo implementado en Simulink.

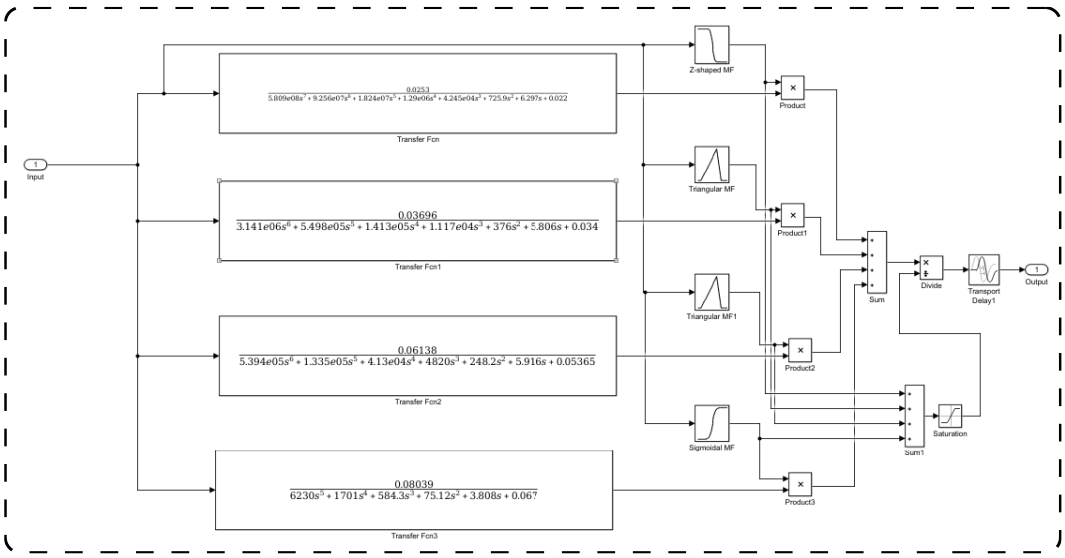


Figura 3.21 - Esquema completo del compresor

Un detalle para resaltar es el delay puesto a la salida. Este delay es casi cero. Se debe a que para que Simulink pueda trabajar es necesario que todos los sistemas implementados en él sean causales. El delay proporciona un valor inicial que, en bucle cerrado con el controlador, consigue que el sistema sea causal.

Para el desarrollo de los controladores se han probado primero con la función de transferencia de la carga al 50%, y si el resultado era aceptable, se ha pasado a simular con el modelo completo del compresor.

A continuación, se muestra el código implementado para el modelo lineal.

```
%% Modelo del compresor de aire linealizado en un punto de equilibrio de la
carga
function G = LinMod(in)
    if in <= 37
        % 37% de la capacidad
        numLA = 0.023;
        denLA = [1 0.071 0.022];
        numIZ = 1.1;
        denIZp = [55.6 1];
        aux = 5;

        denIZ = 1;
        for i = 1:aux
            denIZ = conv(denIZ, denIZp);
        end

        GLA = tf(numLA, denLA);
        GIZ = tf(numIZ, denIZ);
        G = GLA * GIZ;
        % ---------------------------------

    elseif in <= 50
        % 50% de la capacidad
        numLA = 0.033;
        denLA = [1 0.08 0.034];
        numIZ = 1.12;
        denIZp = [42.1 1];
        aux = 4;

        denIZ = 1;
        for i = 1:aux
            denIZ = conv(denIZ, denIZp);
        end
```

```
            GLA = tf(numLA, denLA);
            GIZ = tf(numIZ, denIZ);
            G = GLA * GIZ;
            % ---------------------------------

        elseif in <= 75
            % 75% de la capacidad
            numLA = 0.05115;
            denLA = [1 0.1 0.05365];
            numIZ = 1.2;
            denIZp = [27.1 1];
            aux = 4;

            denIZ = 1;
            for i = 1:aux
                denIZ = conv(denIZ, denIZp);
            end

            GLA = tf(numLA, denLA);
            GIZ = tf(numIZ, denIZ);
            G = GLA * GIZ;
            % ---------------------------------

        else
            % 100% de la capacidad
            numLA = 0.063;
            denLA = [1 0.11 0.067];
            numIZ = 1.276;
            denIZp = [18.4 1];
            aux = 3;

            denIZ = 1;
            for i = 1:aux
                denIZ = conv(denIZ, denIZp);
            end

            GLA = tf(numLA, denLA);
            GIZ = tf(numIZ, denIZ);
            G = GLA * GIZ;
            % ---------------------------------
        end
    end
% ------------------------------------------------------------------
```

Es una función que devuelve la función de transferencia linealizada en el punto de carga pedido. Una vez finalizado la caracterización del modelo, se ha pasado al desarrollo de los controladores.

3.7.1 Control PID

El primer controlador desarrollado ha sido un PID. La implementación se ha llevado a cabo mediante el bloque PID de Simulink, con su respectivo *tuning* para el ajuste de parámetros.

El esquema utilizado se muestra en la figura 3.22.

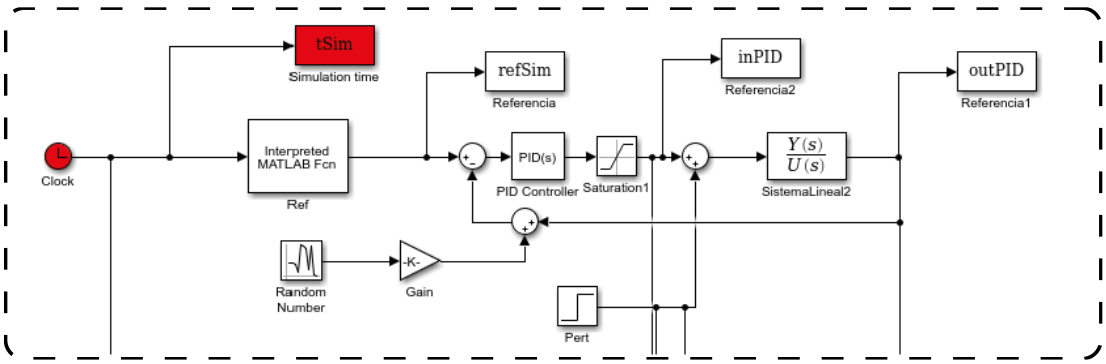


Figura 3.22 - Esquema del PID en Simulink

Como se puede observar, se le ha añadido la posibilidad de introducir un ruido a la entrada, además de una saturación [-100, 100] a la señal de control. Para poder simular perturbaciones, se ha escogido introducirlas a la entrada, mediante un escalón de valor 5, correspondiendo a un 10% aprox. de la entrada.

Los resultados obtenidos por el PID con el modelo simplificado del compresor (la función de transferencia con la carga al 50%) se muestran a continuación en la figura 3.23.

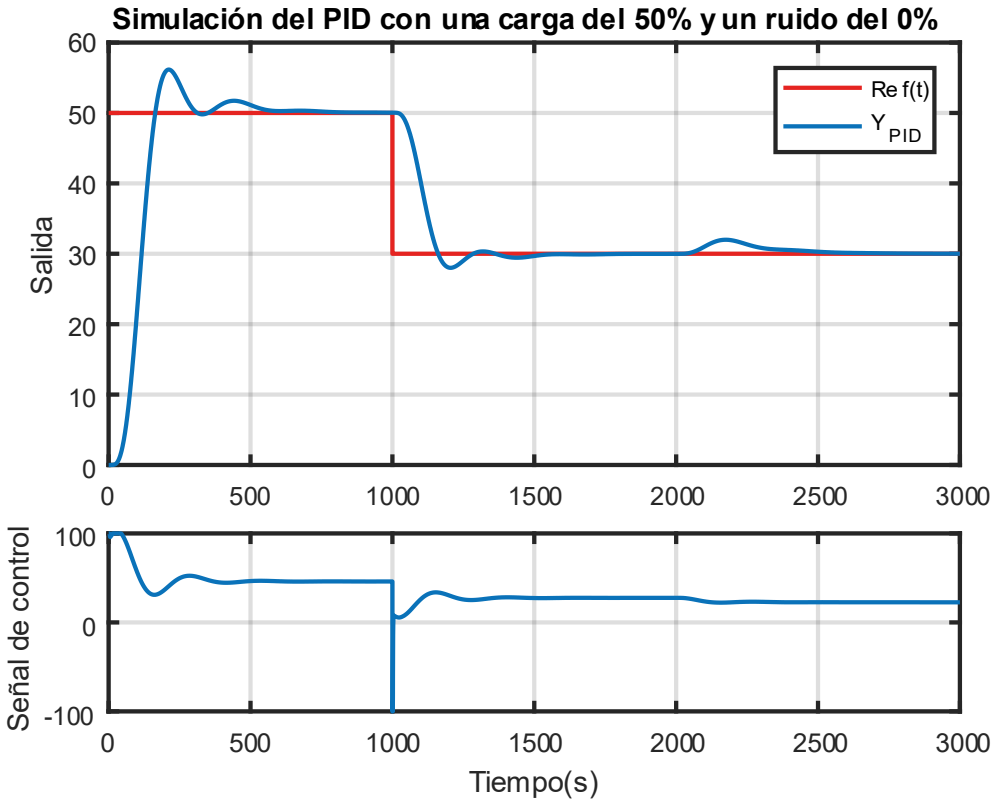


Figura 3.23 - Resultados del PID

Una vez verificado que los resultados eran buenos se ha introducido el modelo completo del compresor. Este primer controlador se vuelve inestable.

3.7.2 Controles basados en FUZZY

La siguiente topología de diseño es la basada en FUZZY. Se han desarrollado varios controladores, comparándolos con el PID, antes de probarlos con el sistema completo.

El control FUZZY se basa en lógica difusa, que se trata de tomar decisiones más o menos intensas dependiendo del grado de cumplimiento de ciertas premisas. Este método se adapta mejor a la realidad, ya que

normalmente se da la presencia de grados de complimiento de condiciones.

Un control FUZZY consta de una *fuzzificación*, evaluación de las reglas y *defuzzificación*.

El proceso de *fuzzificación* consiste en evaluar la entrada del control con respecto a las regiones de pertenencia de entrada. Las regiones de pertenencia son funciones que clasifican la entrada respecto a varias cualidades referentes al control.

Estas regiones son las encargadas de cuantizar la entrada para posteriormente calcular la salida correspondiente. Como se ha comentado antes, la entrada puede estar incluida en varias regiones de pertenencia, con grados de pertenencia de mayor o menor intensidad.

Una vez cuantizada la entrada con respecto a las regiones de pertenencia, es el momento de evaluarla respecto a las reglas del control. Estas reglas son las encargadas de asociar una salida para una entrada determinada, dependiendo del grado de pertenencia con las regiones. Esta evaluación de reglas se desarrolla mediante uniones e intersecciones de las regiones de pertenencia, dando como resultado un conjunto a partir de estas regiones.

Por último, se encuentra el proceso de *defuzzificación*, que es el encargado de pasar el conjunto resultante a valores de salida.

A continuación, se va a explicar los controladores modelados, que son dos: un controlador FUZZY incremental, y un FUZZY PD con integrador.

El desarrollo del primer controlador se basa en un controlador PI convencional, del cual se obtienen sus constantes de control y se transforman en las constantes de control FUZZY.

El primer paso es modelar el controlador PI en Simulink:

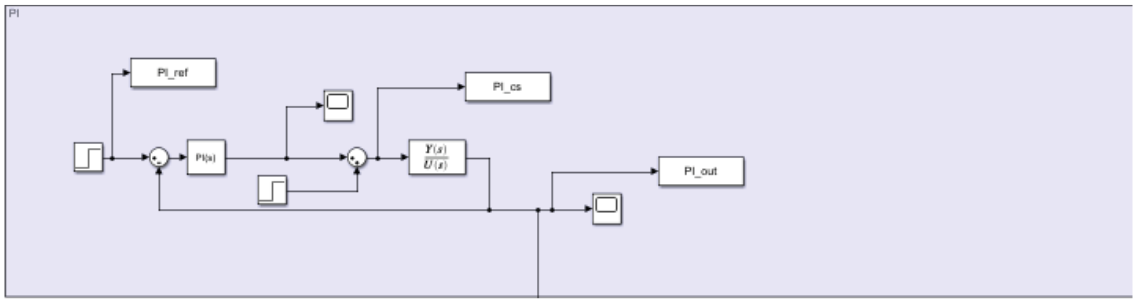


Figura 3.24 - Esquema PI en Simulink

A continuación, en la figura 3.25, se muestra el diagrama de bloques del controlador FUZZY integral. Como se puede apreciar, las entradas del controlador son el error y la derivada del error.

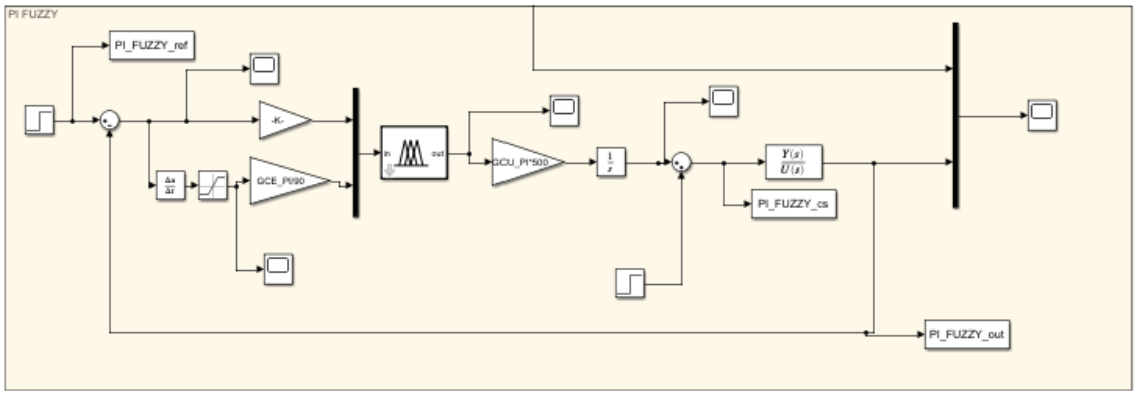


Figura 3.25 - Esquema Fuzzy integral en Simulink

A la hora de diseñar el controlador, se han definido dos regiones de pertenencia para cada una de las entradas, y tres regiones de salida. Las regiones de pertenencia de ambas entradas son dos funciones triangulares cuyo rango está ajustado a cada entrada, siendo el rango del error [-100, 100] y el rango de la derivada [-1, 1]. Se ha observado que, si la entrada se sale de dicho rango de las regiones de pertenencia, el

bloque asigna como valor de entrada el valor medio del rango, detalle que no parece muy adecuado, siendo mejor tomar el valor límite de dicho rango. En las figuras 3.26 y 3.27 se muestran las regiones de pertenencia de las entradas.

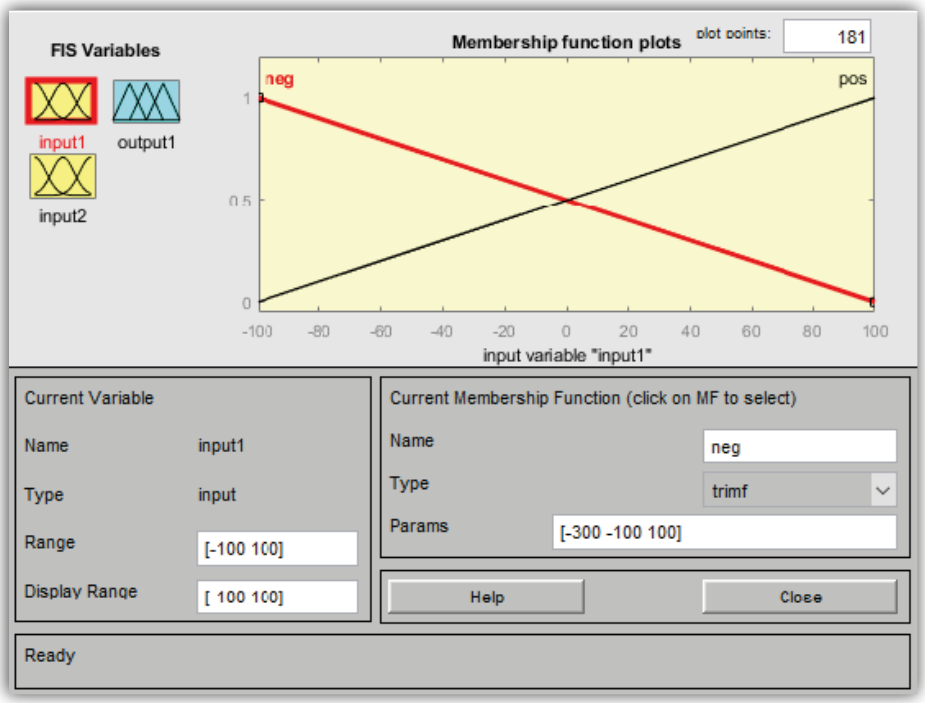


Figura 3.26 - Regiones de pertenencia para la entrada 1

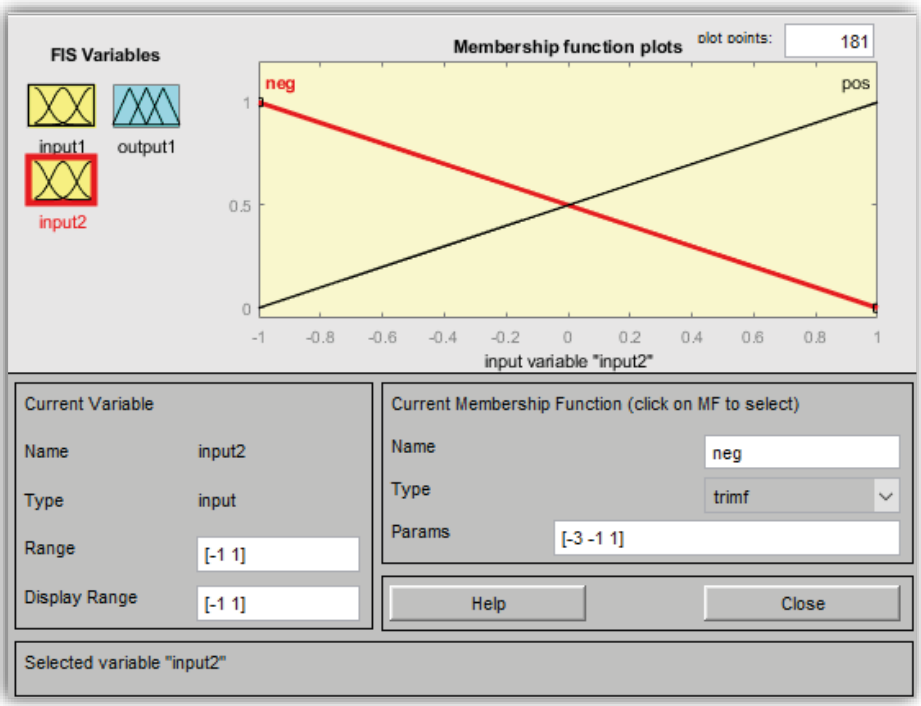


Figura 3.27 - Regiones de pertenencia para la entrada 2

En cuanto a las reglas del control, se han definido 3 funciones de salida, que son **neg, zero y pos.** Se tratan de tres funciones triangulares, y se ha establecido un rango de salida de -1 a 1. Se ha definido este rango unitario, ya que la salida del control es incremental y posteriormente se integra. En la figura 3.28 se puede observar los rangos de pertenencia de la salida.

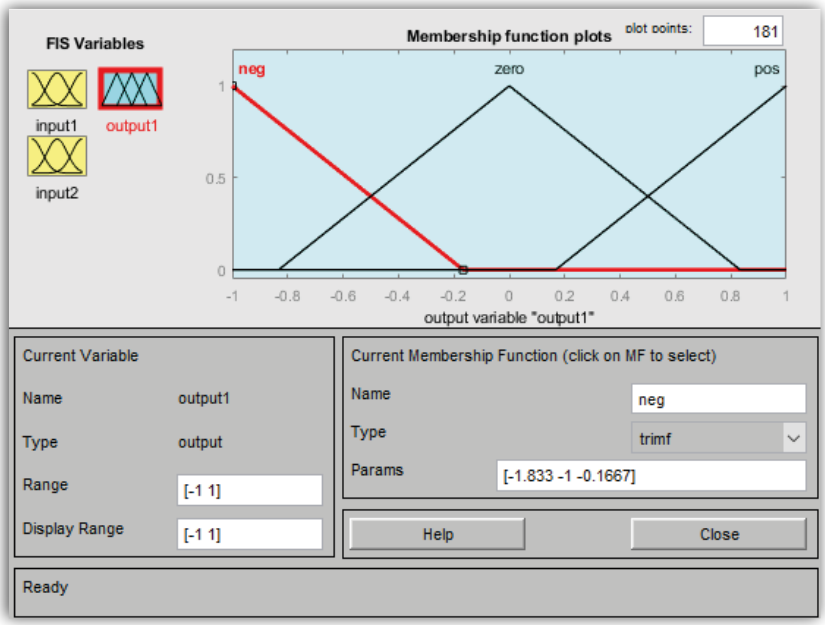


Figura 3.28 - Regiones de pertenencia de la salida

Por último, se muestran a continuación las reglas definidas en el controlador:

```
If (input1 is neg) AND (input2 is neg) THEN (output is neg)
If (input1 is neg) AND (input2 is pos) THEN (output is zero)
If (input1 is pos) AND (input2 is neg) THEN (output is zero)
If (input1 is pos) AND (input2 is pos) THEN (output is pos)
```

Para terminar el controlador, se calculan las constantes de ganancia, que en este caso son GE, GCE y GCU, a partir de los parámetros del PI Después de su cálculo, se han ajustado para lograr un mejor control.

```
% Parámetros PI tras realizar el proceso de PI tunning para G50
Kp_PI = 1.12555747324198;
Ki_PI = 0.00753326810088655;

Ti_PI = Kp_PI / Ki_PI;

% Parametros FUZZY para PI
GE_PI = 1;
GCE_PI = GE_PI * Ti_PI;
GCU_PI = Kp_PI / GCE_PI;
```

Es necesario añadir que se ha añadido un ajuste final a cada parámetro para lograr un mejor control:

```
GCU_PI = GCU_PI * 500
GCE_PI = GCE_PI / 90
GE_PI = GE_PI * 1.4
```

A continuación, en las figuras 3.29 y 3.30, se muestran los resultados para el control PI y el control PI FUZZY.

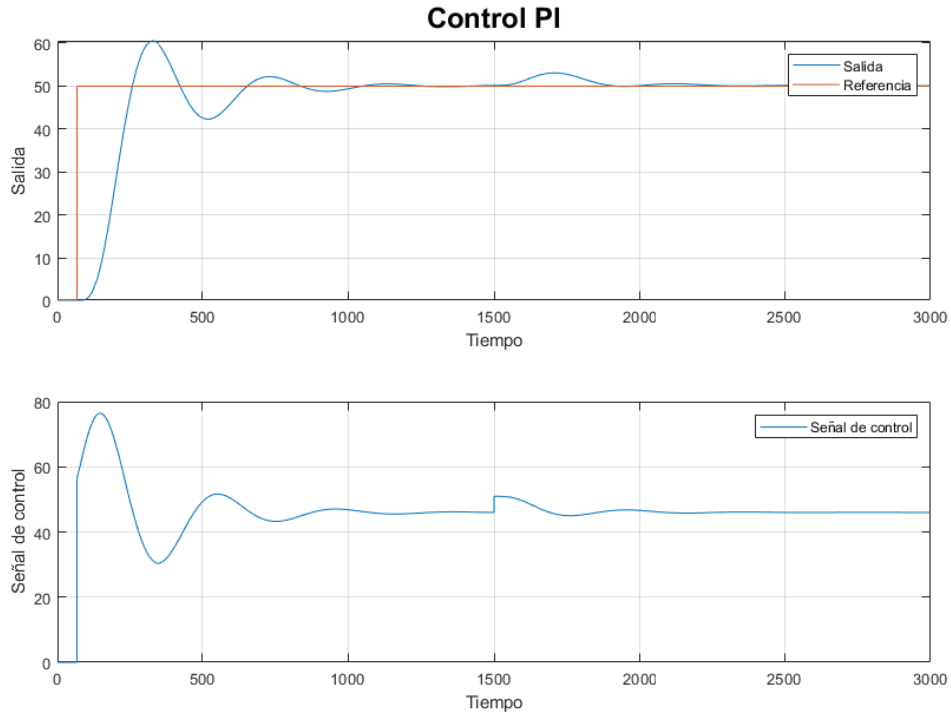


Figura 3.29 - Resultados del controlador PI

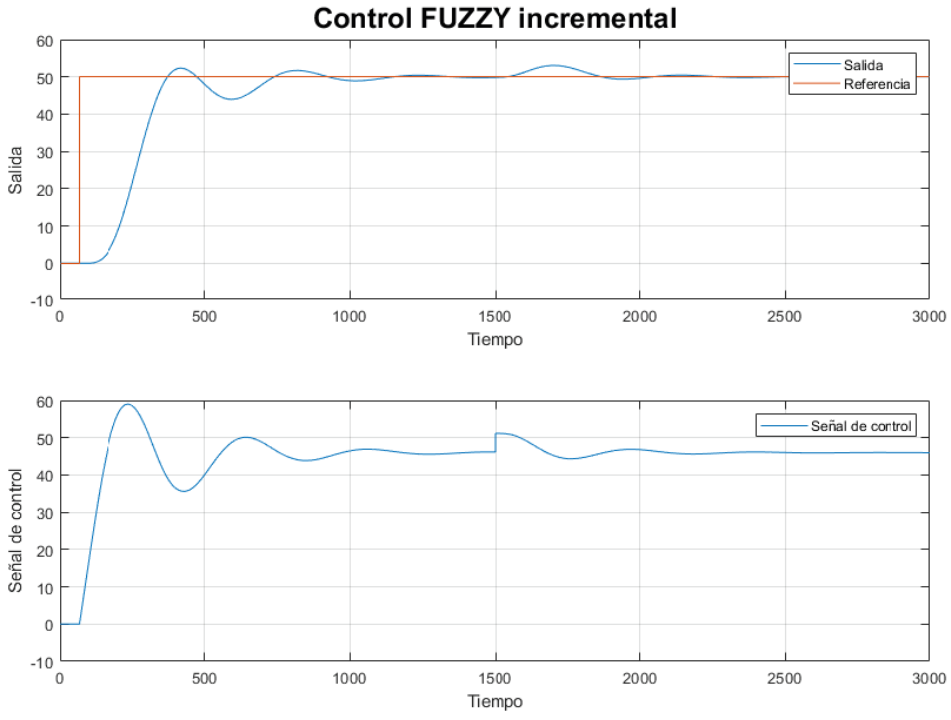


Figura 3.30 - Resultados del controlador FUZZY incremental

A continuación, se desarrollará el segundo controlador de esta topología: un controlador FUZZY PD con término integrador. Este controlador hay que manejarlo con cautela, ya que el término integral no pasa por el controlador FUZZY y si no se ajusta bien puede dar lugar a confusiones. Como experiencia en el desarrollo de la práctica, primeramente, se diseñó el bloque FUZZY del controlador con unos rangos de entrada unitarios, que estaban produciendo un mal funcionamiento de dicha rutina. Sin embargo, el efecto del integrador estaba tapando este fallo y daba la apariencia de un buen ajuste de este controlador.

A continuación, en las figuras 3.31 y 3.32 se muestran los diseños en Simulink de los controladores PID y PID FUZZY:

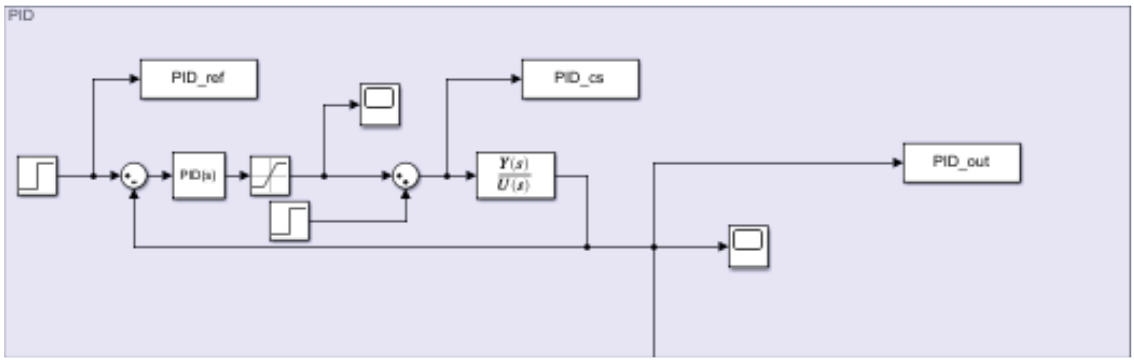


Figura 3.31 - Esquema PID en Simulink

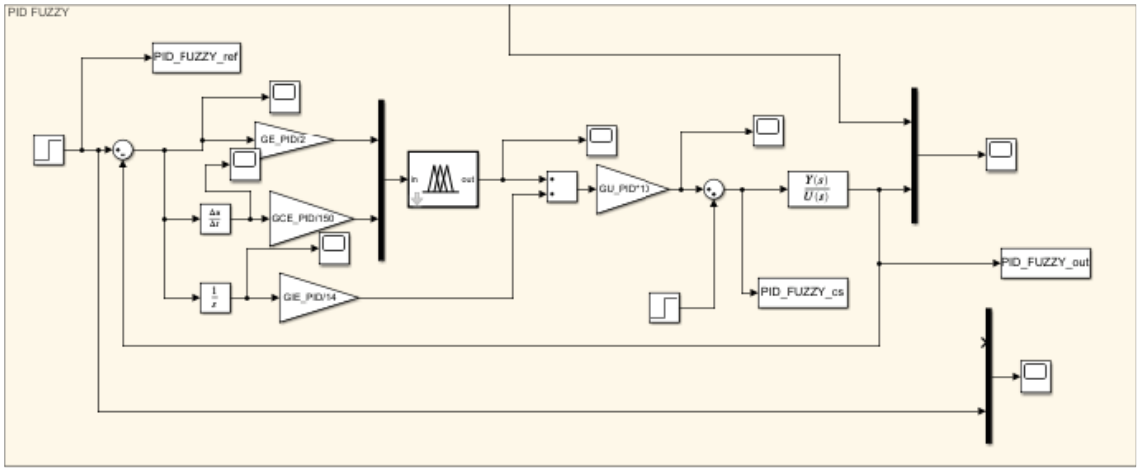


Figura 3.32 - Esquema PID Fuzzy en Simulink

Para el diseño del controlador FUZZY PID no se va a hacer mucho hincapié, ya que es bastante similar al controlador del FUZZY incremental, con las mismas reglas de evaluación y mismas regiones de pertenencia de entrada. Sin embargo, el rango de salida de este control se ha definido tal que [-100, 100], ya que la salida de este control es absoluta, a diferencia del control FUZZY incremental.

El cálculo de los parámetros es el siguiente:

```
Kp_PID = 1.85757919712897;
Ki_PID = 0.0103731480313567;
Kd_PID = 82.5325058054696;

Ti_PID = Kp_PID / Ki_PID;
Td_PID = Kd_PID / Kp_PID;

% Parámetros FUZZY para PID
GE_PID = 1;
GCE_PID = GE_PID * Td_PID;
GIE_PID = GE_PID / Ti_PID;
GU_PID = Kp_PID / GE_PID;
```

Tras el ajuste, los parámetros se quedan tal que así:

```
GCE_PID = GCE_PID / 150
```

```
GIE_PID = GIE_PID / 14
GU_PID = GU_PID *10
```

Las figuras 3.33 y 3.34 muestran los resultados del PID convencional y del FUZZY PID.

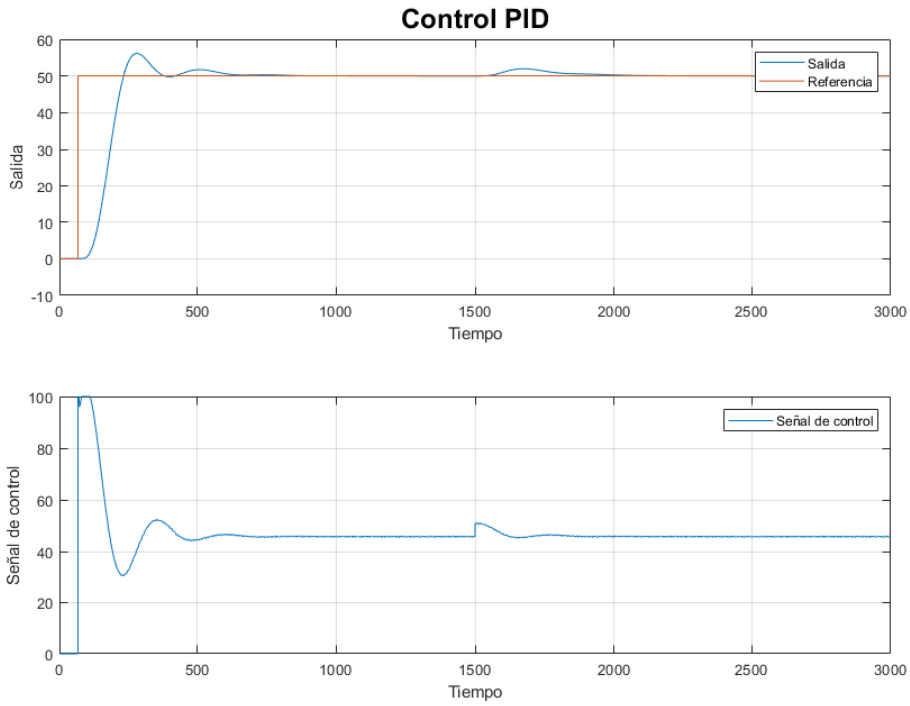


Figura 3.33 - Resultados del controlador PID

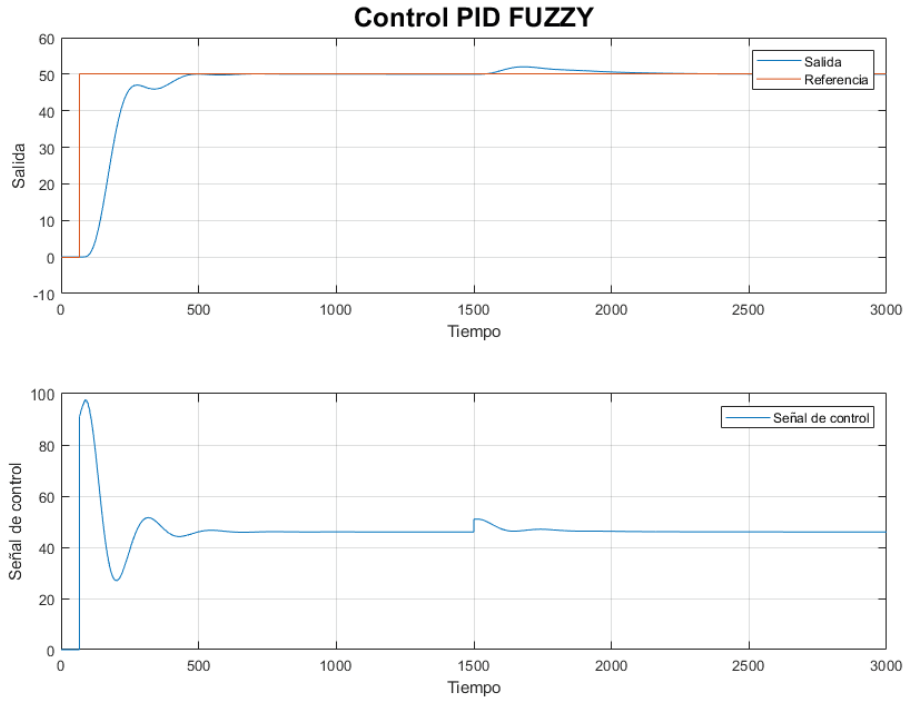


Figura 3.34 - Resultados del controlador FUZZY PID

Por último, se adjunta una gráfica que compara la acción de los 4 controladores explicados anteriormente.

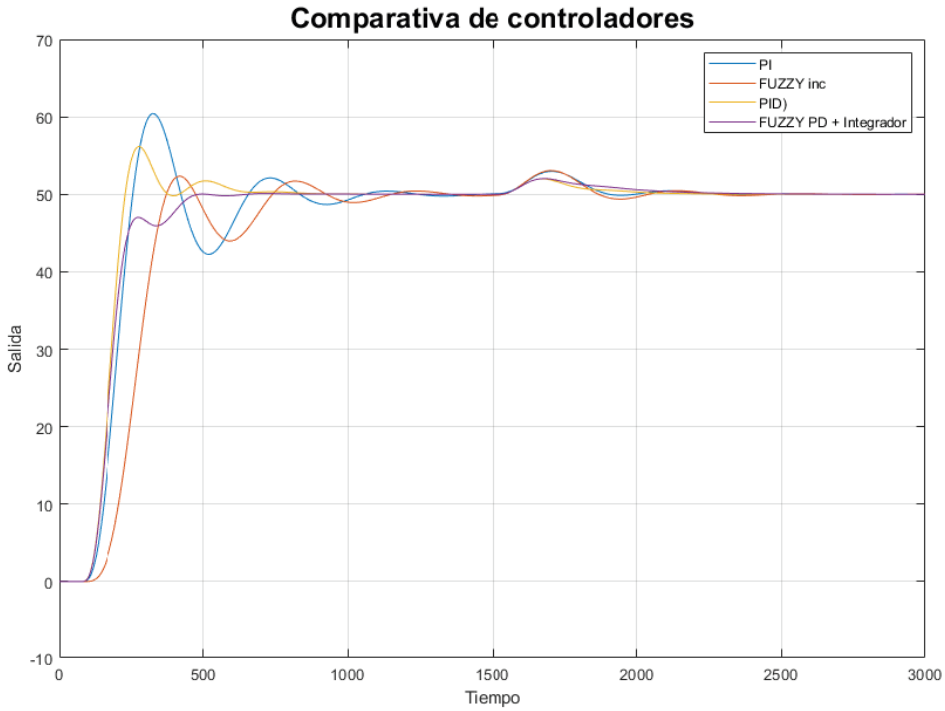


Figura 3.35 - Resultado de los 4 controladores explicados anteriormente

Analizando un poco esta gráfica, puede observarse que el controlador que obtiene una respuesta más rápida es el FUZZY PD con efecto integrador, que alcanza el objetivo en torno a los 600 segundos. Por otro lado, el FUZZY incremental resulta ser el más lento de todos.

Dado que eran estables con la función de transferencia en el punto de trabajo, se probaron con el sistema completo. El resulto coincidió con el PID. Esto ha llevado a una profundización en el modelo desarrollado. El resultado ha sido que el modelo contiene unos retrasos que son difíciles de estabilizar con los controladores propuestos.

3.7.3 Controles predictivos

Acto seguido se cambió la topología de controladores por predictivos. Estos controladores permiten, mediante la integración con un modelo interno del sistema, adelantarse a los acontecimientos y reaccionar en consecuencia.

El primer controlador de esta topología fue el MPC. Se desarrollaron tres modelos para el propio controlador MPC: dos obtenidos mediante la identificación, cambiando el orden de éstos, y el propio simplificado del compresor con una carga del 50%. El resultado fue que tampoco se consiguió controlar.

Acto seguido se modificó la estructura genérica del MPC para llegar a otro controlador de la misma topología: un DMC, por sus siglas en inglés de *Dynamic Matrix Control.*

Un DMC es un tipo de controlador predictivo, caracterizado porque el modelo que utiliza para predecir el comportamiento futuro del sistema es su respuesta ante escalón.

El DMC se compone de dos partes: offline y online. La primera de ellas se trata de la preparación de este, antes de ejecutarse con el sistema. A continuación, se muestra el código de la parte offline:

```
%% DMC (Dynamic Matrix Control)
% ---------------------------------


%% Declaración de variables
global refDMC g KDMC
```

```
%% Calculamos la respuesta en escalón
% N => Tamaño del horizonte de predicción
N = 10;
p = 200;
% Nu => Tamaño del horizonte de control
Nu = 5;

% Penalización de los movimientos del control
theta = 300;
lambda = 150;

% Se calcula la respuesta entera y se escala al intervalo deseado
auxg = step(GL);

L = length(auxg);
j = 1;
for i = 1: round(L/(N+p+1)) : L
    if j <= N + p + 1
        g(1, j) = auxg(i);
        j = j + 1;
    end
end

for i = length(g) + 1 : 1 : N + p + 1
    g(1, i) = g(1, i-1);
end

% Acto seguido construimos la matriz G
G = g(1, 2 : p+1)';
aux = G;
for j = 2 : Nu%p
    aux = [0; aux];
    aux = aux(1 : p);
    G = [G, aux];
end
% ---------------------------------

%% Referencia
for t = 1 : (Tsim + p)
    refDMC(t) = Ref(t);
end
% ---------------------------------

% Calculamos la salida óptima y nos quedamos con la primera fila
M = inv(theta*(G' * G) + lambda * eye(Nu)) * (G');
KDMC = M(1, :);
% ---------------------------------
```

```
g = g';
% --------------------------------
```

El código mostrado contiene cuatro pasos:

1. Ajuste de los horizontes (de control y de predicción) y de las penalizaciones para el control: se ha añadido una doble penalización, para optimizar el transitorio. Esta pequeña optimización se puede ver en el cálculo de la matriz M (salida óptima).
2. Cálculo de la respuesta ante escalón del sistema linealizado.
3. Por último, se calculan las matrices G y M necesarias para el DMC. También se almacena la referencia de la simulación en una variable propia al DMC para poder simular varios controladores en paralelo.

La parte online consiste en el control en sí, funcionando en cada tiempo de muestreo. A continuación, se muestra el código de esta segunda parte:

```
%% DMC (Dynamic Matrix Control)
% --------------------------------
function u = DMC(t, ym, N, p)

    % Declaración de variables globales y estáticas
    global refDMC g KDMC Tsim ruido
    persistent f U dU

    if t < 1
        f = zeros(Tsim, 1);
        U = zeros(Tsim, 1);
        dU = zeros(Tsim, 1);
    end

    if t < N + 1
        u = 0;

    else
        t = round(t);

        % Respuesta libre + forzada
        for k = 1 : p
            f(t+k, 1) = (ym + ruido * randn(1,1));
            for i = 1 : N
                f(t+k, 1) = f(t+k, 1) + (g(k+i, 1) - g(i, 1)) * dU(t-i);
            end
        end

        % Cálculo de la referencia en los próximos instantes de tiempo
        w = refDMC(t+1 : t+p)';

        % Cálculo de la señal de control
        dU(t) = KDMC * (w - f(t+1 : t+p));

        % Actualizamos la señal de control
```

```
        U(t) = U(t-1) + dU(t);
        u = U(t);
    end
end
% -------------------------------------------------------------------
```

Este programa se compone de los siguientes eventos:

1. Inicialización de las variables.
2. Antes del horizonte de predicción, se aplica una señal de control nula.
3. Si el tiempo ha superado el horizonte de predicción, se calcula la señal de control a aplicar de la siguiente manera: primero se calcula la respuesta libre y forzada (se incluye el ruido en la medida de la salida del sistema); se actualiza la referencia desde ese instante hasta el horizonte de control; se calcula la señal de control como la ganancia *KDMC* multiplicada por la diferencia de la referencia, la respuesta libre y la forzada; y al final se actualiza la señal de control.

La clave de los controladores predictivos es que se anticipan gracias a que conocen la referencia y un modelo del sistema de ante mano. El modelo redunda en la respuesta libre y forzada. Estas dos respuestas son la manifestación de la predicción del control. Se caracterizan por lo siguiente:

- Respuesta libre: se trata de la evolución del proceso debido a su estado actual (incluido por tanto el efecto de acciones pasadas).
- Respuesta forzada: es la debida a las acciones de control futuras.

En la figura 3.36 se puede ver una explicación gráfica de estas dos respuestas.

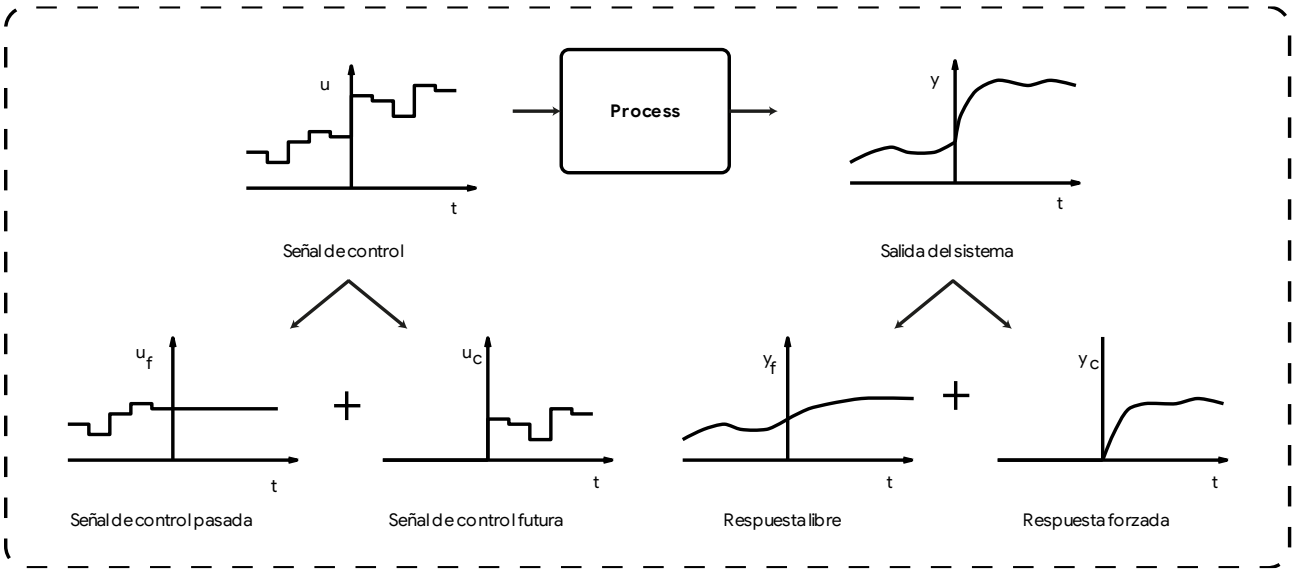


Figura 3.36 - Explicación de la respuesta libre y forzada

El esquema escogido para la implementación en Simulink ha sido el siguiente:

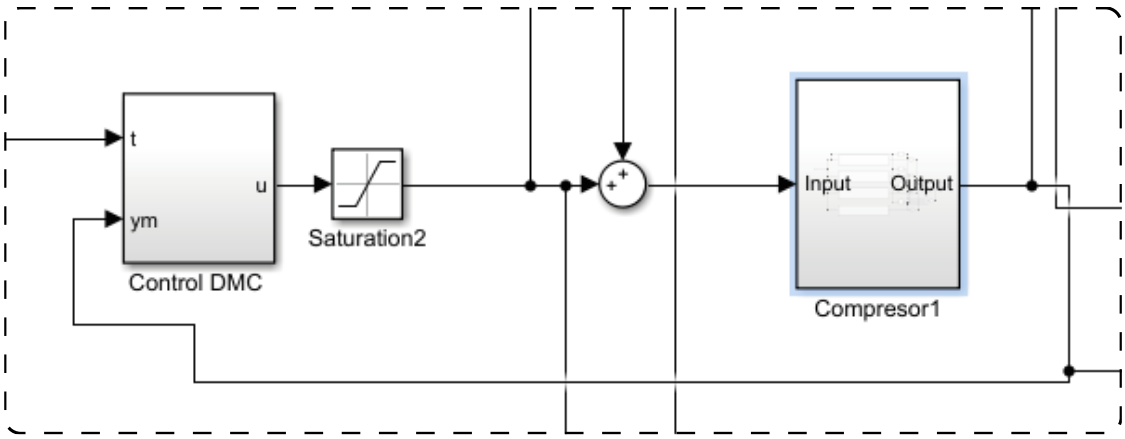


Figura 3.37 - Esquema de control para el DMC con el sistema completo

Se le ha añadido una saturación a la señal de control idéntica a la implementada en el PID [-100, 100]. El ruido se ha implementado dentro del controlador.

Los resultados obtenidos con el modelo se adjuntan en la figura 3.38.

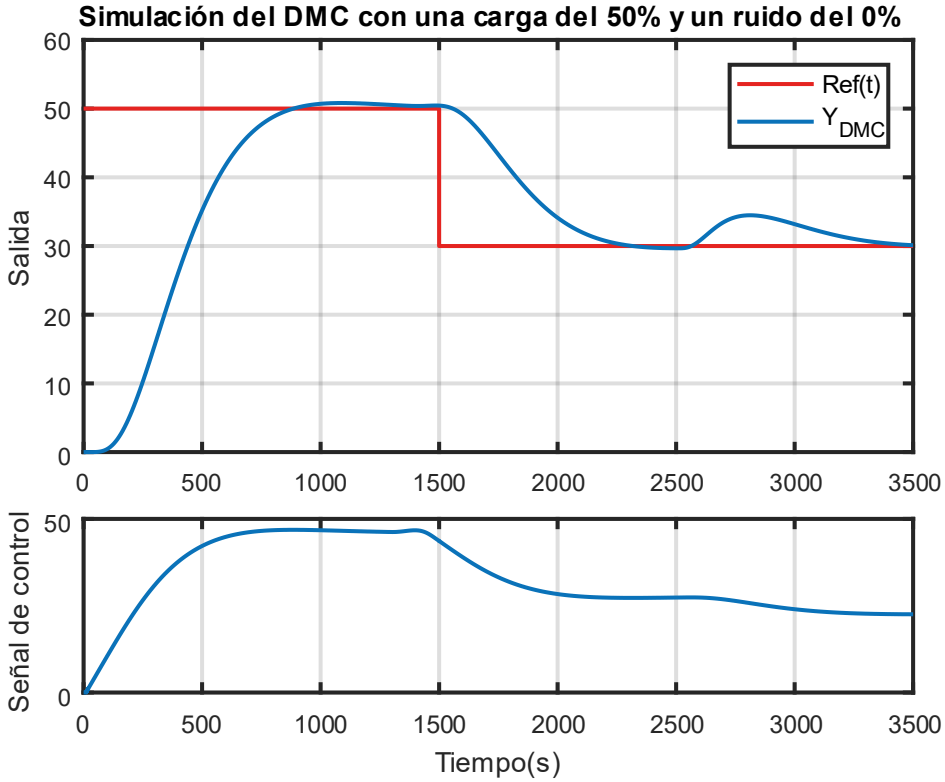


Figura 3.38 - Resultado del DMC con modelo simplificado

A continuación, se ha puesto a controlar el modelo completo del compresor, y se han ajustado los parámetros para hacerlo más rápido. El resultado ha sido el mostrado en la figura 3.39.

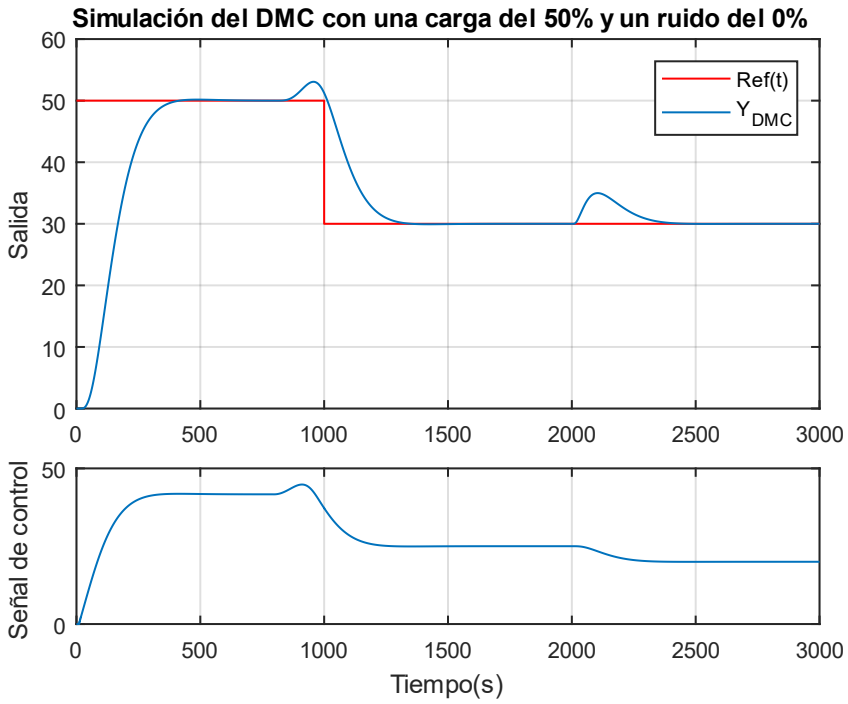


*Figura 3.*39 - Resultado del DMC con el modelo completo

Dados los buenos resultados se ha comparado con el PID. Esta comparación realmente no es justa, dado que el DMC tiene que controlar el sistema no lineal completo y el PID tiene que controlar el modelo simplificado, pero da luces sobre la bondad del controlador DMC, y de su robustez.

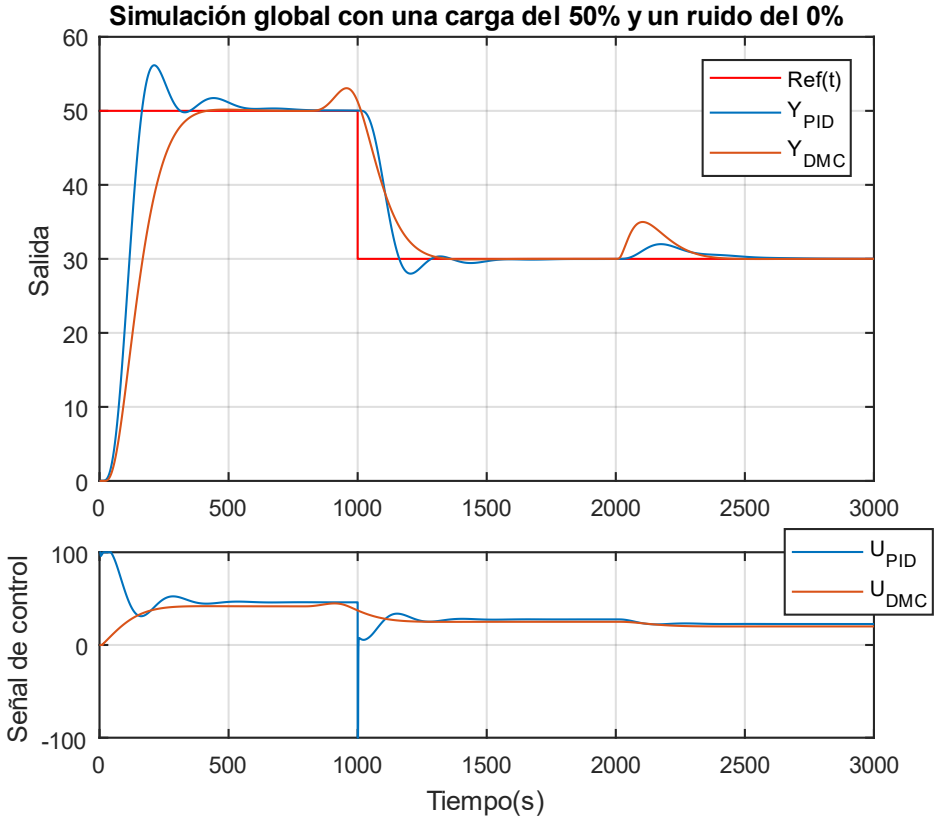


Figura 3.40 - Comparativa sin ruido

Como se puede observar en la figura 3.40, el DMC, aunque teniendo desventaja, es capaz de, con menor señal de control (menor energía gastada) controlar mejor. A continuación, se ha introducido ruido aleatorio a la entrada, de valor máximo igual a cinco. El resultado es el siguiente.

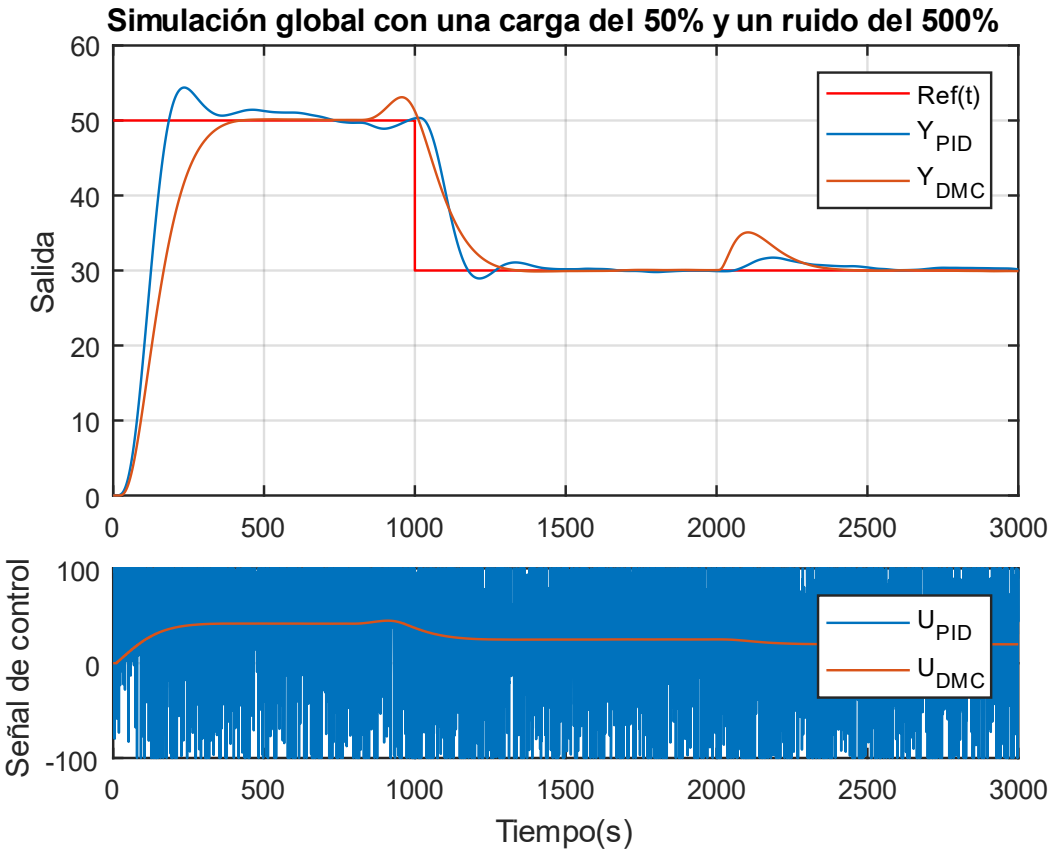


Figura 3.41 - Comparativa con ruido

Como se puede observar, el DMC es muy robusto frente el ruido en la medida de los sensores.

Dados los resultados de todas las simulaciones para el modelo en el punto de trabajo, la conclusión es que el mejor controlador, dentro de la investigación realizada, para optimizar el control del compresor es el DMC, debido a su robustez y bajo consumo. Además, es el único capaz de funcionar con el sistema completo.

Esta conclusión está a expensas de continuar la investigación.

Estos son todos los servicios en proceso de implementación en el gemelo digital.

4 CONCLUSIÓN

Se ha expuesto un método con varios pasos para la realización de un gemelo digital. La exposición de cada paso del método se ha hecho de manera generalizada, dado que la concreción de cada paso es parte del desarrollador/es del proyecto que lo aplique.

A continuación, se ha expuesto un ejemplo aplicado en la realidad para que sirva para dos cuestiones: de validez del método y de guiado para llevarlo a cabo.

Después del ejemplo expuesto se puede intuir que se requieren unos conocimientos amplios interdisciplinares para hacer un gemelo digital con este método. Además, el tiempo requerido para llevarlo a cabo es exponencial con la cantidad de subsistemas que se implementen.

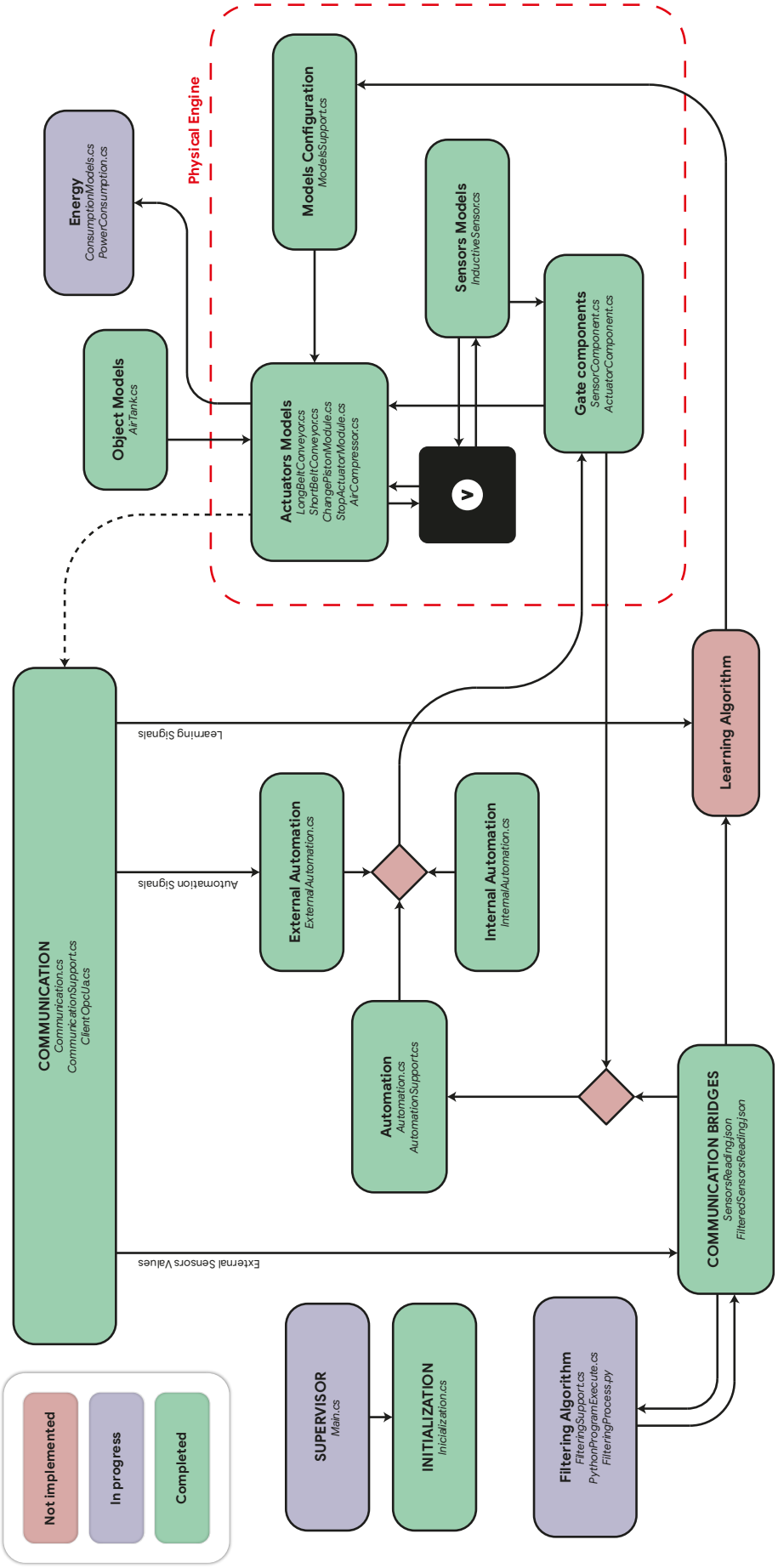
Otro detalle importante para resaltar, visto en el ejemplo, es la reutilización de los módulos para otros gemelos digitales. Para ello es necesario que el módulo esté bien encapsulado y no tenga dependencias externas. Además, existen softwares que traen módulos de modelos funcionales de sistemas comerciales que se pueden utilizar. Todo esto disminuye el tiempo de la implementación.

Por último, hay que decir que el desarrollo de gemelos digitales sigue en proceso de investigación. Este trabajo quiere presentar un camino para guiar futuras investigaciones, por los resultados obtenidos, aunque teniendo conscientes todos los defectos que presenta este método.

5 ANEXO I

El anexo uno está formado por el esquema de la arquitectura interna del gemelo digital implementado.

Nota aclaratoria: hay dos símbolos que no se muestra su significado. El primero es un rombo (no implementado = relleno rojo) que consiste en un análogo de un multiplexador en electrónica. El segundo es el cuadrado con un **V** en el centro: representa el visor donde se encuentran todos los modelos visuales.


Physical Engine
Energy
ConsumptionModels.cs
PowerConsumption.cs
Models Configuration
ModelsSupport.cs
Sensors Models
InductiveSensor.cs
Gate components
SensorComponent.cs
ActuatorComponent.cs
Object Models
AirTank.cs
Actuators Models
LongBeltConveyor.cs
ShortBeltConveyor.cs
ChangePistonModule.cs
StopActuatorModule.cs
AirCompressor.cs
Learning Signals
Automation Signals
External Automation
ExternalAutomation.cs
Internal Automation
InternalAutomation.cs
Learning Algorithm
COMMUNICATION
Communication.cs
CommunicationSupport.cs
ClientOpcUa.cs
Automation
Automation.cs
AutomationSupport.cs
COMMUNICATION BRIDGES
SensorsReading.json
FilteredSensorsReading.json
External Sensors Values
SUPERVISOR
Main.cs
INITIALIZATION
Inicialization.cs
Filtering Algorithm
FilteringSupport.cs
PythonProgramExecute.cs
FilteringProcess.py
Not implemented
In progress
Completed

6 BIBLIOGRAFÍA

[1] Real Academia Española.
URL: https://dle.rae.es/gemelo

[2] Real Academia Española.
URL: https://dle.rae.es/igual

[3] Equipo de expertos de la Universidad Internacional de Valencia, 2021. ¿Qué es un gemelo digital?
URL: https://www.universidadviu.com/es/actualidad/nuestros-expertos/gemelo-digital-definicion-funcionamiento-y-ejemplos

[4] Ferrovial servicios, 2020. Proyectos de innovación con gemelos digitales.
URL: https://www.ferrovialservicios.com/es/proyecto/gemelos-digitales/

[5] Interempresas, 2020. Proyecto Miraged.
URL: https://www.interempresas.net/Robotica/Articulos/306575-El-proyecto-Miraged-desarrollara-gemelos-digitales-para-la-mejora-de-la-Industria-40.html

[6] Comisión Europea, Cordis, 2020. Proyecto DENiM.
URL: https://cordis.europa.eu/project/id/958339/es

[7] Javier Gómez Jiménez, Trabajo Fin de Grado, Universidad de Sevilla, 2020. "Modelado y control de quadrotors en la plataforma UNITY 3D".
Fuente: http://bibing.us.es/proyectos/abreproy/92985/fichero/TFG-2985+G%C3%93MEZ+JIM%C3%89NEZ%2C+JAVIER.pdf

[8] Just An Idea Studio, 2021. Avances en el renderizado en tiempo real.
URL: https://www.youtube.com/watch?v=EgB4C1M6T-c

[9] Adolfo Sánchez del Pozo, Juan Gómez Jiménez, Javier Gómez Jiménez, 2021. "Simulación de sistemas mecatrónicos". Ed. Paraninfo.

[10] Pablo González Camacho, Trabajo Fin de Grado, Universidad de Sevilla, 2021. "Evaluación de herramientas industriales para Gemelos Digitales".

[11] Unity Technologies. Página oficial del software Unity3d.
URL: https://unity.com/es

[12] Opiron. Ventajas del uso del protocolo OPC UA en sistemas de fabricación.
URL: https://www.opiron.com/5-razones-por-las-que-elegir-opc-ua/

[13] OPC Foundation. Repositorio oficial OPC UA .NET de la Fundación OPC.
URL: https://github.com/OPCFoundation/UA-.NETStandard

[14] Oculus. Sitio oficial del artículo Oculus Link.
URL: https://www.oculus.com/accessories/oculus-link/?locale=es_ES

[15] Keping Liu, Xuesheng Feng, Min Yang, Chonghe Tang, Changhong Jiang, 2010. “Fuzzy Predictive Control of Air Compressor System”.