GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

SIMULACIÓN DEL VUELO DE UN DRON EN UN ENTORNO VIRTUAL

Autor: Daniel González Rodríguez

Director: Juan Luis Zamora Macho

Co-Director: María Ana Saenz Nuño

Madrid

Julio de 2022

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Simulación del Vuelo de un Dron en un Entorno Virtual

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2021-2022 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Daniel González Rodríguez Fecha: 13/ 07/ 2022

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.:  Fecha: 18/ 07/ 2022

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

SIMULACIÓN DEL VUELO DE UN DRON EN UN ENTORNO VIRTUAL

Autor: Daniel González Rodríguez

Director: Juan Luis Zamora Macho

Co-Director: María Ana Saenz Nuño

Madrid

Julio de 2022

Agradecimientos

Este trabajo de fin de grado es el final de una etapa en la universidad, así como el comienzo de una nueva. Los cursos no han sido fáciles, y me han parecido una montaña rusa de emociones que da verdadero vértigo. He vivido momentos muy buenos y felices, pero también momentos duros y que costaba digerir, aunque a la larga todos ellos me han servido tanto como para crecer como persona, como para acercarme al objetivo de este grado, convertirme en ingeniero.

Si a alguien he de agradecer haber llegado hasta aquí es a mis padres, por haberme brindado la oportunidad de estudiar en el lugar donde quería aprender y ser formado. También a mi hermana, por hacerme sonreír en los momentos más complicados.

Debo dar las gracias a todos los amigos que he hecho a lo largo de estos años en la universidad. Haber compartido con ellos tantas batallas ganadas, perdidas, revanchas, y celebraciones, nos ha unido y me ha llevado a sentirme parte de una gran familia. También a Pablo, Andrés, y Lucía, por ser mi apoyo siempre y conseguir que no me diese por vencido.

Quiero dar las gracias a mis directores de proyecto, Juan Luis Zamora Macho y María Ana Saenz Nuño, por haber seguido semanalmente y con interés el trabajo. Han sido muy buenos guías, con paciencia y comprendiendo las dificultades que iban surgiendo.

Por último, quiero dar las gracias a todas las personas que han trabajado en partes de este proyecto antes de que yo comenzase con él. Sobre todo, a Javier José Carrera Fresneda, por trabajar conmigo y ayudarme cuando lo he necesitado.

Resumen del Proyecto:

Simulación del Vuelo de un Dron en un Entorno Virtual

Autor:	González Rodríguez, Daniel
Directores:	Zamora Macho, Juan Luis
	Saenz Nuño, María Ana
Entidad Colaboradora:	ICAI – Universidad Pontificia Comillas

1. Introducción

Los drones son vehículos aéreos que se crearon con fines militares, aunque hoy en día se utilizan en gran cantidad de distintos sectores profesionales y de entretenimiento. Algunos ejemplos profesionales son la industria de la agricultura, de la construcción, o del transporte de mercancías. [1]

Un simulador del vuelo de un dron es una herramienta que permite a un usuario aprender a controlar un dron en un entorno virtual visualizado a través de un ordenador, como si se tratase de un videojuego. De esta manera, el usuario se prepara para controlar el dron en un ámbito profesional sin producir daños a su alrededor o al propio dron, con los costes que ello conllevaría. [2]

El simulador debe ser lo más fiel posible a la realidad. Esta premisa se divide en dos ramas que deben acercarse a la experiencia real. Por un lado, los modelos tridimensionales de la simulación, como el dron o el entorno de vuelo, deben ser realistas. Por otra parte, el modelo dinámico del vuelo debe ser exactamente igual al del dron real, atendiendo al peso y estructura del dron, así como al empuje de los motores o las condiciones ambientales.

Otras aplicaciones para las que se utilizan los simuladores de vuelo son las relacionadas con la preparación de vuelos autónomos mediante parámetros de control, o con el perfeccionamiento de redes de drones que trabajan como enjambres. [3] [4]

2. Definición del Proyecto

En el laboratorio de ICAI se dispone de un dron, pero no de un simulador de vuelo. El objetivo es realizar una simulación en la que se controlará en el ordenador un dron igual al del laboratorio, tratando que tanto el modelo dinámico como la visualización del vuelo sean lo más cercanos posible a la realidad. El entorno virtual en el que volará el dron en la simulación será el propio laboratorio de ICAI. El modelo estructural del dron será el que se encuentra en ese laboratorio.

De esta manera, la meta es conseguir controlar mediante el mando del dron, que tiene radiotransmisión, el movimiento realista del modelo 3D del dron en el entorno

virtual del laboratorio. Para ello, se debe integrar el modelo de realidad virtual que reproduzca con la máxima fidelidad posible el entorno físico. Después añadir el modelo estructural del dron a dicho entorno virtual. Integrar también el modelo dinámico del dron, atendiendo a que se deben simular los vuelos tal y como serían en la realidad. Los vuelos se podrán simular tanto en modo autónomo, siguiendo rutas preestablecidas, como en modo manual, mediante un mando con radiotransmisión.

3. Descripción del Trabajo

El proyecto consta de tres bloques de trabajo:

A. El primer bloque es el trabajo es en UnReal, un programa de desarrollo de videojuegos y simulaciones tridimensionales. [5] En UnReal es donde se importa el entorno virtual del laboratorio, desarrollado previamente en SolidWorks y Sketchup por María Ana Saenz Nuño, codirectora de este proyecto. También integra AirSim, que es un simulador de vuelo de drones desarrollado por Microsoft como un añadido a Unreal. [6] AirSim permite simular el vuelo de un dron preestablecido. Por ello, en este bloque también se cambia el modelo estructural del dron preestablecido por AirSim, por el modelo estructural del dron del laboratorio diseñado en SolidWorks, resultado del proyecto *Diseño Mecánico de un Dron para Inventario Automático de Almacenes*, de Diego Cubillo Llanes. [7]
B. El segundo bloque de trabajo es el código de Matlab y Simulink, para que el vuelo del dron tenga el mismo modelo dinámico que el del laboratorio. Este código ha sido facilitado por Juan Luis Zamora Macho, codirector de este proyecto. El dron debe poder controlarse mediante un mando con radiotransmisión conectado a Simulink, así como seguir las rutas de vuelo autónomo programadas en el código de Matlab. [8] [9]
C. El último bloque de trabajo es la programación en Python necesaria para conseguir la comunicación entre los dos bloques anteriores. El modelo dinámico tendrá como salidas las posiciones y orientaciones que debe adoptar el dron en la simulación. Estas posiciones y orientaciones se enviarán al simulador AirSim utilizando funciones de Python. [10] [11]

Por último, también se desarrollan colisiones del dron con otros objetos virtuales, lo que necesita para su correcto funcionamiento trabajo en los tres bloques mencionados. Cuando el dron colisione, la simulación debe terminar. Si AirSim detecta una colisión entre el dron y otro objeto en UnReal, se envía un booleano afirmativo al Simulink del modelo dinámico del dron, para que este pueda detener la simulación.

Alguno de estos bloques de trabajo había sido comenzado en las becas de colaboración con ICAI de Claudia Valero de la Flor, en verano de 2021, y de Javier José Carrera Fresneda, en el curso 2021/2022. Aún así, el proceso ha sido repetido desde el principio para poder ser debidamente documentado en esta memoria. [12] [13]

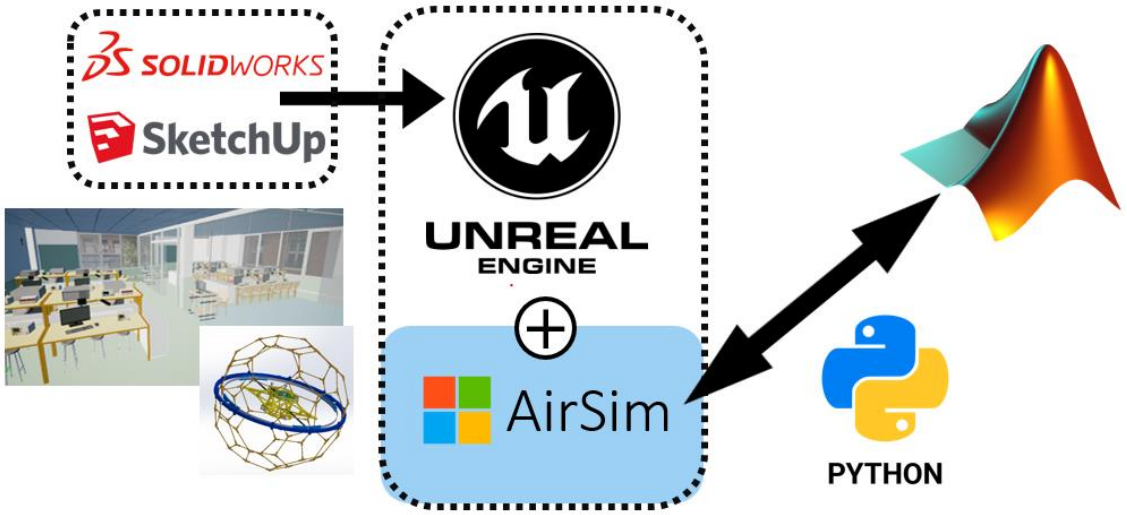

Figura 1: Esquema de conexión entre los bloques de trabajo

4. Resultados

La simulación es un éxito, siendo correcta la comunicación entre el mando a través de Simulink con AirSim. El dron se muestra por pantalla y se mueve acorde a las instrucciones enviadas, deteniéndose la simulación cuando el dron colisiona con una de las paredes de la zona de vuelo.

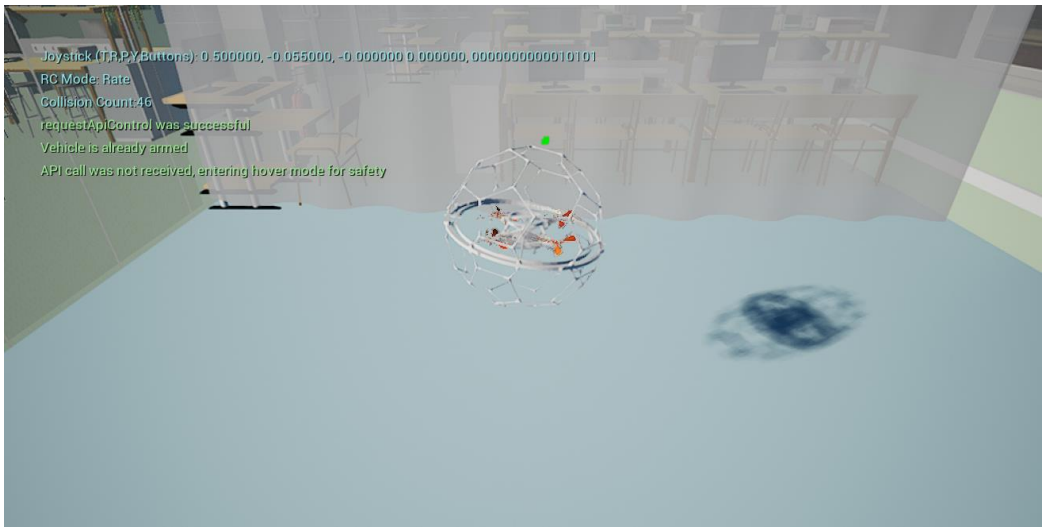

Figura 2: Visualización de la simulación

Como el dron se mueve de una forma demasiado rápida y brusca, es complicado volarlo en el entorno limitado para ello sin que colisione con las paredes. Para solucionar esto, se opta por suavizar la pendiente de las rectas de los distintos canales, tanto el canal del throttle como los de los giros de guiñada, alabeo y cabeceo; para que el movimiento del dron no sea tan sensible al desplazar levemente los joysticks.

5. Conclusiones

Este proyecto es el resultado de la unión de distintos trabajos que se han realizado en proyectos previos, juntando distintas piezas para formar el simulador completo. Esta memoria funciona como guía para poder unir estas partes, formando el simulador desde cero cuando sea necesario, siguiendo y comprendiendo cada paso.

Los objetivos cumplidos en este proyecto coinciden con los mencionados en la descripción del trabajo:

- Creación del proyecto de UnReal, importación del laboratorio y del modelo estructural del dron a AirSim.
- Programación en Python y Simulink para enviar posiciones y orientaciones desde Simulink al simulador AirSim.
- Integración del Simulink creado al Simulink del modelo dinámico del dron.
- Finalización de la simulación al producirse una colisión del dron con otro objeto.
- Descripción detallada de todo el proceso de estos cuatro pasos en esta memoria.

Las ideas para perfeccionar el trabajo en futuros proyectos son amplias, aunque son verdaderamente interesantes:

- Mejorar las colisiones del dron con otros objetos virtuales. En vez de finalizarse la simulación, que el dron se choque y rebote.
- Estudio de los sensores lidar que pueden añadirse a AirSim, para programar vuelos autónomos del dron siguiendo las paredes virtuales por proximidad.
- Añadir en UnReal un modo de vuelo en primera persona. Conectando a UnReal unas gafas de realidad virtual, se podría ver el vuelo de una forma inmersiva, como en los entrenamientos de vuelo profesional.

6. Referencias

[1] Revista Empresarial & Laboral, «7 aplicaciones profesionales de un drone,» 2017.

[2] USS, «Drone Services,» 31 03 2020. [En línea]. Available: https://www.droneservices.com.ar/industria-4-0/simulador-de-drones/. [Último acceso: 22 03 2022].

[3] S. Wang, J. Chen, Z. Zhang, G. Wang, Y. Tan and Y. Zheng, "Construction of a virtual reality platform for UAV deep learning," 2017 Chinese Automation Congress (CAC), 2017, pp. 3912-3916, doi: 10.1109/CAC.2017.8243463.

[4] A. Devos, E. Ebeid and P. Manoonpong, "Development of Autonomous Drones for Adaptive Obstacle Avoidance in Real World Environments," 2018 21st Euromicro Conference on Digital System Design (DSD), 2018, pp. 707-710, doi: 10.1109/DSD.2018.00009.

[5] X. Chen, M. Wang and Q. Wu, "Research and development of virtual reality game based on unreal engine 4," 2017 4th International Conference on Systems and Informatics (ICSAI), 2017, pp. 1388-1393, doi: 10.1109/ICSAI.2017.8248503.

[6] Microsoft Research, «AirSim APIs,» 2021. [En línea]. Available: https://microsoft.github.io/AirSim/apis/. [Último acceso: 23 03 2022].

[7] D. Cubillo Llanes, «Diseño Mecánico de un Dron para Inventario Automático de Almacenes,» 2019. Universidad Pontificia de Comillas, Escuela Técnica Superior de Ingeniería (ICAI) Identificador: http://hdl.handle.net/11531/40919

[8] The MathWorks, Inc., «MathWorks, Matlab,» [En línea]. Available: https://es.mathworks.com/products/matlab.html. [Último acceso: 24 03 2022].

[9] The MathWorks, Inc., «MathWorks, Simulink,» [En línea]. Available: https://es.mathworks.com/products/simulink.html. [Último acceso: 24 03 2022].

[10] L. Garber, «The Lowly API is Ready to Step Front and Center,» 08 2013. [En línea]. Available: https://www.computer.org/csdl/magazine/co/2013/08/mco2013080014/13rRUzpQPJ9. [Último acceso: 24 03 2022].

[11] Fangohr, H. (2004). A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds) Computational Science - ICCS 2004. ICCS 2004. Lecture Notes in Computer Science, vol 3039. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-25944-2_157

[12] C. Valero de la Flor, *Beca de colaboración con la Universidad Pontificia Comillas, ICAI,* 2021.

[13] J. J. Carrera Fresneda, *Beca de colaboración con la Univeridad Pontifica Comillas, ICAI,* 2021.

Project Abstract:

Drone Flight Simulation in a Virtual Environment

Author:	González Rodríguez, Daniel
Supervisors:	Zamora Macho, Juan Luis
	Saenz Nuño, María Ana
Collaborating Entity:	ICAI – Universidad Pontificia Comillas

1. Introduction

Drones are aerial vehicles that were created for military purposes, although nowadays they are used in many different professional sectors and in entertainment industries. Some professional examples are the agriculture, construction, or freight transport industries. [1]

A drone flight simulator is a tool that allows to learn how to control a drone in a virtual environment displayed on a computer, as if it was a video game. Therefore, a user can get prepared to control a drone in a professional environment without causing damages around him or to the drone itself, which could lead to several costs. [2]

The simulator must be as faithful as possible to reality. This premise can be divided in two ways in which the simulation has to resemble the real experience. In one hand, the three-dimensional models of the simulation, such as the drone or the flight environment, must be realistic. On the other hand, the dynamic model of the flight must be the same as the one the real drone has, considering the weight and structure of the drone, as well as the thrust of the motors or the environmental conditions.

Flight simulators are used for other applications as the ones related with the preparation of autonomous flights through control parameters, or with the improvement of drones' networks that operate like swarms. [3] [4]

2. Project Definition

There is a drone in the ICAI laboratory, but there is not a flight simulator. The objective is to carry out a simulation in which a drone alike to the one in the laboratory will be controlled on the computer. Both the dynamic model and the visualization of the flight must be as close as possible to reality. The virtual environment in which the simulation will take place be the ICAI laboratory itself. The structural model of the drone will be the one in the laboratory.

The goal is to control the 3D model of the drone in the laboratory virtual environment using the radio remote control of the drone, and it should be as realistic as possible. To do this, the virtual reality model that reproduces the physical environment

must be incorporated. Later the structural model of the drone shall be added to this virtual environment. Also, the dynamic model of the drone shall be incorporated, considering that the flights must be as accurate to reality as possible. The flights will be simulated both in autonomous mode, following pre-established routes, and in manual mode, by the radio remote control of the drone.

3. Work Description

The project consists in three work blocks:

A. The first block is the work in UnReal, which is a videogame and three-dimensional simulations development program. [5] UnReal is where the laboratory virtual environment is imported. It was previously developed in SolidWorks and Sketchup by María Ana Saenz Nuño, co-director of this project. UnReal also must integrate AirSim, which is a drone flight simulator developed by Microsoft as a plugin to Unreal. [6] AirSim allows you to simulate the flight of a preset drone. Because of this, the pre-established drone structural model must be changed into the laboratory drone structural model. The drone structural model was designed in SolidWorks, and it is the result of the project *Mechanical Design of a Drone for Automatic Warehouse Inventory*, by Diego Cubillo Llanes. [7]
B. The second block of work is the Matlab and Simulink code, which purpose is that the drone flight has the same dynamic model as the one in the laboratory. This code has been provided by Juan Luis Zamora Macho, co-director of this project. The drone must be able to be controlled by a radio remote control transmitter connected to Simulink It may as well follow autonomous flight paths programmed in the Matlab code. [8] [9]
C. The last block of work is the Python programming, which is necessary to achieve the communication between the two previous blocks. The outputs of the dynamic model will be the positions and orientations that the drone must adopt in the simulation. These positions and orientations will be sent to the AirSim simulator by using Python functions. [10] [11]

Finally, the collisions of the drone with other virtual objects are developed. Working in the three mentioned blocks is necessary for it. When the drone collides, the simulation shall end. If AirSim detects a collision between the drone and another object in UnReal, an affirmative Boolean is sent to the Simulink that contains the drones' dynamic model. With that affirmative Boolean, Simulink can stop the simulation.

Some of these work blocks begun in two collaboration studentships with ICAI. The collaborations where with Claudia Valero de la Flor, in the summer of 2021, and with Javier José Carrera Fresneda, in the 2021/2022 academic year. Even so, the process has been repeated from the beginning to be fully documented in this report. [12] [13]

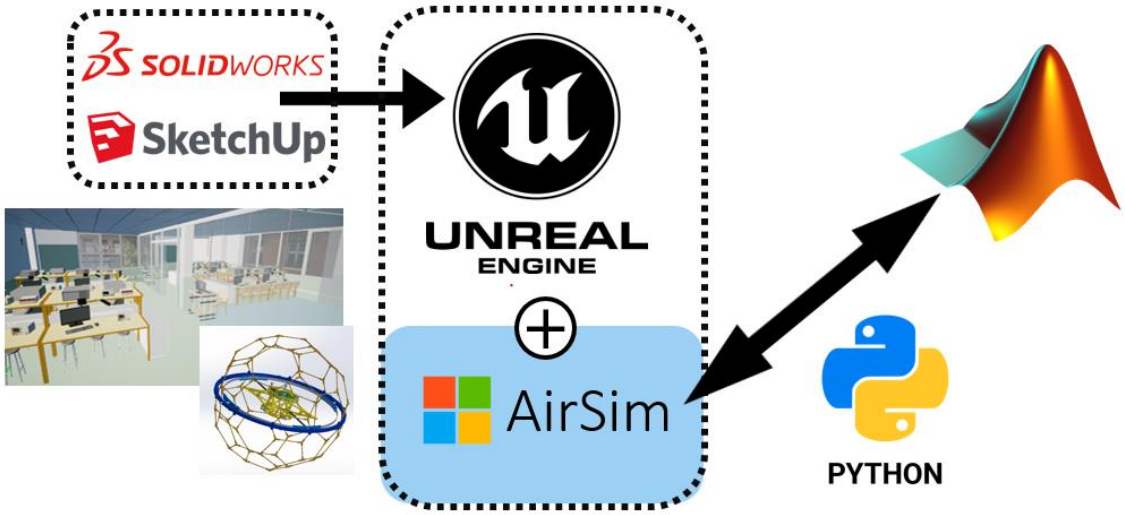

Figure 1: Connection scheme between the working blocks

4. Results

The simulation is successful. The communication between the information of the radio controller that Simulink sends and the visualization of AirSim in UnReal works properly. The drone is shown on the screen and moves according to the instructions sent. The simulation stops when the drone collides with one of the flight zone walls.

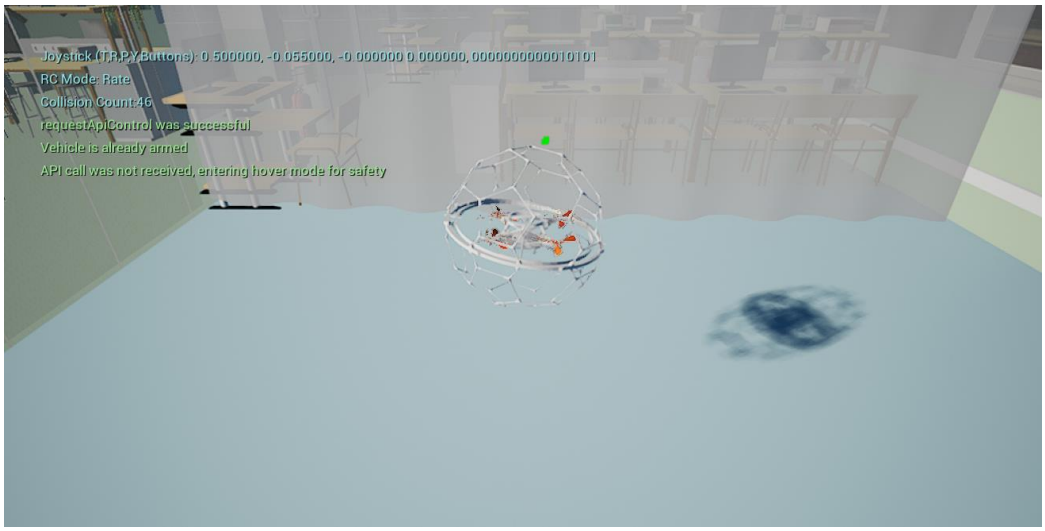

Figure 3: Simulation visualization

Since the drone moves too fast and abruptly, it's difficult to control it in the flight environment without crashing it into the walls. To solve this problem, it is chosen to smooth the slope of the different channel curves, both the throttle channel and the ones of the yaw, roll and pitch angles; so that the movement of the drone is not so sensitive when the joysticks are slightly moved.

5. Conclusions

This project is the result of the union of different works that have been carried out in previous projects. The simulator is created by putting together the different pieces. This memory works as a guide of how to join these parts, so that the simulator can be made from scratch when necessary, following and understanding each step.

The objectives that have been fulfilled in this project coincide with those mentioned in the job description:

- Creation of the UnReal project, importing the laboratory and the structural model of the drone to AirSim.
- Programming in Python and Simulink for sending positions and orientations from Simulink to the AirSim simulator.
- Integration of the created Simulink to the drones' dynamic model Simulink.
- Ending of the simulation when the drone collides with another object.
- Detailed description of the entire process including these four steps in this report.

The ideas to improve this work in future projects are wide, although the most interesting ones could be:

- Improve drone collisions with other virtual objects. Instead of ending the simulation, the drone crashes and bounces while the simulation continues.
- Study of the lidar sensors that can be added to AirSim, to program autonomous flights of the drone, following the virtual walls by proximity.
- Add a first-person flight mode in UnReal. Connecting virtual reality glasses to UnReal, the flight could be seen in an immersive way, like in a professional flight training.

6. References

[1] Revista Empresarial & Laboral, «7 aplicaciones profesionales de un drone,» 2017.

[2] USS, «Drone Services,» 31 03 2020. [Online]. Available: https://www.droneservices.com.ar/industria-4-0/simulador-de-drones/. [Last access: 22 03 2022].

[3] S. Wang, J. Chen, Z. Zhang, G. Wang, Y. Tan and Y. Zheng, "Construction of a virtual reality platform for UAV deep learning," 2017 Chinese Automation Congress (CAC), 2017, pp. 3912-3916, doi: 10.1109/CAC.2017.8243463.

[4] A. Devos, E. Ebeid and P. Manoonpong, "Development of Autonomous Drones for Adaptive Obstacle Avoidance in Real World Environments," 2018 21st Euromicro Conference on Digital System Design (DSD), 2018, pp. 707-710, doi: 10.1109/DSD.2018.00009.

[5] X. Chen, M. Wang and Q. Wu, "Research and development of virtual reality game based on unreal engine 4," 2017 4th International Conference on Systems and Informatics (ICSAI), 2017, pp. 1388-1393, doi: 10.1109/ICSAI.2017.8248503.

[6] Microsoft Research, «AirSim APIs,» 2021. [Online]. Available: https://microsoft.github.io/AirSim/apis/. [Last access: 23 03 2022].

[7] D. Cubillo Llanes, «Diseño Mecánico de un Dron para Inventario Automático de Almacenes,» 2019. Universidad Pontificia de Comillas, Escuela Técnica Superior de Ingeniería (ICAI) Identifier: http://hdl.handle.net/11531/40919

[8] The MathWorks, Inc., «MathWorks, Matlab,» [Online]. Available: https://es.mathworks.com/products/matlab.html. [Last access: 24 03 2022].

[9] The MathWorks, Inc., «MathWorks, Simulink,» [Online]. Available: https://es.mathworks.com/products/simulink.html. [Last access: 24 03 2022].

[10] L. Garber, «The Lowly API is Ready to Step Front and Center,» 08 2013. [Online]. Available: https://www.computer.org/csdl/magazine/co/2013/08/mco2013080014/13rRUzpQPJ9. [Last access: 24 03 2022].

[11] Fangohr, H. (2004). A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds) Computational Science - ICCS 2004. ICCS 2004. Lecture Notes in Computer Science, vol 3039. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-25944-2_157

[12] C. Valero de la Flor, *Studentship collaboration with Universidad Pontificia Comillas, ICAI,* 2021.

[13] J. J. Carrera Fresneda, *Studentship collaboration with Universidad Pontificia Comillas, ICAI,* 2021.

Índice general

Índice de ilustraciones

Índice de códigos

Índice de tablas

MEMORIA DESCRIPTIVA

1. Introducción

En esta introducción se presenta el proyecto de título *Simulación del vuelo de un dron en un entorno virtual*. La sección 1.1 expone los motivos que llevaron a la realización del proyecto. En la 1.2, se plantea en qué consisten los objetivos del proyecto. En la sección 1.3 se detalla el software y hardware que ha sido utilizado a lo largo del proyecto, siendo la parte de mayor relevancia en este caso los diversos programas y aplicaciones utilizadas. Por último, en la sección 1.4 se describen las tareas realizadas para cumplir los objetivos expuestos en la sección 1.2.

1.1. Motivación

En el contexto de una sociedad en la que los drones están cada vez más extendidos en todos los ámbitos, los simuladores de vuelo también serán una pieza importante. En el laboratorio de ICAI se dispone de un dron, pero no de un simulador de vuelo.

El objetivo es realizar una simulación en la que se controlará en el ordenador un dron igual al del laboratorio, tratando que tanto el modelo dinámico como la visualización del vuelo sean lo más cercanos posible a la realidad. El ambiente en el que volará el dron en la simulación será el propio laboratorio de ICAI. Esto tiene sentido ya que es donde volará el dron real.

De esta manera, la meta es conseguir controlar mediante el mando del dron, que tiene radiotransmisión, el movimiento realista del modelo 3D del dron en el entorno virtual del laboratorio.

1.2. Objetivos

1) Integrar el modelo de realidad virtual que reproduzca con la máxima fidelidad posible el entorno físico en el que va a volar el dron real.

2) Añadir el modelo estructural y el sistema dinámico del dron a dicho entorno virtual para poder simular vuelos.

3) Poder simular vuelos tanto en modo autónomo, siguiendo rutas preestablecidas, como en modo manual, mediante un mando con radiotransmisión.

1.3. Recursos

- SolidWorks y Sketchup:

SolidWorks es un software de diseño CAD (Diseño Asistido por Computadora) para modelar piezas y ensamblajes en 3D y planos 2D. Cubre varias etapas del proceso de desarrollo del producto, incluyendo la creación, el diseño, o la gestión de datos del proceso. [1]

SketchUp es un software de modelado 3D de Trimble que es altamente utilizado en la arquitectura y en el diseño de productos. [2]

Tanto los muebles del laboratorio, como las partes del dron, han sido diseñados en SolidWorks. El conjunto del dron ha sido montado en SolidWorks, mientras el ambiente del laboratorio ha sido creado colocando paredes, muebles, y otros detalles con SketchUp.

- UnReal Engine:

Unreal Engine es un motor de juegos de la compañía Epic Games, y es una herramienta muy avanzada para la creación de animaciones fotorealistas en 3D y experiencias inmersivas. Este potente motor es utilizado normalmente en la industria de los videojuegos. Se programa en C++, con algunas simplificaciones a la hora de programar, ya que utiliza diagramas de bloques. [3] [4]

UnReal Engine es el soporte donde se simulará el vuelo del dron. Se le importarán tanto el diseño 3D del laboratorio, como el diseño y la simulación del dron. Deberá comunicarse con Matlab para obtener la información del mando o de las rutas preestablecidas.

La versión utilizada para este proyecto es la 4.25.4, y se descarga de forma gratuita a través de Epic Games Launcher. Epic Games Launcher es una aplicación de descarga de contenido de videojuegos, en la que debemos crear una cuenta y descargar UnReal seleccionando la licencia llamada “Creators Licence”, que es completamente gratuita.

- AirSim:

AirSim es un simulador gratuito de vuelo de drones, y también de conducción de coches y otros vehículos. Está desarrollado por Microsoft como un plugin para UnReal, pudiendo importarse a cualquier escenario desarrollado en este motor. [5]

El objetivo de este simulador es realizar investigación y experimentación relacionada con inteligencia artificial y deep learning para vehículos autónomos. AirSim

es compatible con otros programas mediante APIs (Interfaz de Programación de Aplicaciones), siendo capaz de recibir datos que pueden controlar al vehículo. [6]

La simulación de vuelo de dron de AirSim es una simulación genérica, pero servirá como base para construir la simulación del dron del laboratorio. Está programado en C++.

- Matlab y Simulink:

Matlab es una aplicación de programación y cálculo utilizada mundialmente para analizar datos, desarrollar algoritmos y crear modelos. Su lenguaje de programación utiliza directamente las matemáticas de matrices y vectores. [7]

Simulink es un espacio donde se utilizan diagramas de bloques para diseñar modelos de sistemas. De esta manera, se pueden realizar simulaciones antes de implementar en un hardware. Estas simulaciones en diagramas de bloques hacen en ocasiones innecesaria la escritura de código. [8]

La tarea de Matlab y Simulink en el proyecto será registrar la posición y orientación que tiene que tomar el dron según la información que reciba del mando o de la ruta preestablecida, y prepararlas para enviarlas a AirSim.

La versión utilizada de Matlab es la 2021b, siendo descargado utilizando la licencia de alumno de la universidad.

- Anaconda Python:

Es un paquete que incluye Python y otras librerías y se descarga de forma gratuita. Python es un lenguaje de programación de alto nivel. Es un lenguaje que busca ser claro y conciso a la hora de leerlo, a la vez que puede expresar conceptos en muchas menos líneas de código que otros lenguajes como C++ o Java. [9] [10]

Utilizando Python se establece la comunicación entre Matlab y UnReal, mediante una API (Interfaz de Programación de Aplicaciones). Las API permiten la comunicación entre distintos programas o aplicaciones, sin necesidad de saber cómo están implementados. Sirven para simplificar el diseño de la interfaz de aplicaciones, haciendo la administración del conjunto mucho más flexible y sencilla. [11]

- Visual Studio:

Se trata de un Entorno de Desarrollo Integrado (IDE), es decir, un programa que abarca numerosos aspectos del desarrollo de software. El IDE de Visual Studio se utiliza para editar, depurar y compilar código. Visual Studio incluye herramientas de finalización de código, o diseñadores gráficos, entre otras características que facilitan el proceso de desarrollo de software. [12]

Simplemente es el editor de texto que se ha utilizado para escribir el código de Python, o las modificaciones de la configuración de AirSim.

- Mando FLYSKY FS-i6S:

Mando con radiotransmisor de 2,4 GHz y 10 canales para vuelo de drones.

Ilustración 1: Mando FLYSKY FS-i6S

1.4. Metodología

Los objetivos planteados en la sección 1.2 han sido resueltos acorde a la planificación detallada a continuación. En algunos casos ha sido necesario realizar reajustes según aparecían diversas complicaciones.

El proyecto consta de tres bloques de trabajo. El primer bloque es el trabajo en UnReal y AirSim para disponer del ambiente virtual y del modelo del dron cuando se simule AirSim. El segundo bloque de trabajo es el código de Matlab y Simulink para que el vuelo del dron tenga el mismo modelo dinámico de vuelo que el del laboratorio. El dron debe poder controlarse mediante un mando con radiotransmisión conectado a Simulink, así como seguir las rutas de vuelo autónomo programadas en el código de Matlab. El último bloque de trabajo es en Python para conseguir la comunicación adecuada entre los dos bloques anteriores.

1.4.1. Tareas:

a) Instalación y comprobación de funcionamiento de todos los programas.
b) Familiarización con la interfaz de UnReal.
c) Aprender como importar un entorno ya creado en UnReal.
d) Prueba de la simulación genérica de AirSim. UnReal recibe de Simulink una posición y ángulos aleatorios del dron a través de Python. Importación del simulador AirSim al entorno del laboratorio.

e) Cambio del dron genérico por el diseñado (el del laboratorio). Animación de las hélices.
f) Trabajo en las colisiones con paredes y otros objetos. Simulink debe recibir cuando hay una colisión, para que la simulación termine.
g) Trabajo en Matlab y Simulink para que la información enviada a UnReal no sea una posición y ángulos aleatorios, sino los indicados por el mando.
h) Mejora de las partes que no hayan quedado de una forma del todo satisfactoria.
i) Complementos opcionales. Por ejemplo, una animación del dron cayendo cuando este se choque con una pared, antes de finalizar la simulación; o transmisión de la simulación a través de unas gafas de realidad virtual.
j) Redacción de la memoria del proyecto.

1.4.2. Cronograma:

Objetivos	Enero	Febrero	Marzo	Abril	Mayo	Junio
a						
b						
c						
d						
e						
f						
g						
h						
i						
j						

Tabla 1: Cronograma

2. Estado del Arte

En este capítulo se describe el estado del arte del proyecto. En la sección 2.1 se explica por qué es importante un simulador de vuelo de drones hoy en día, así como algunas de sus aplicaciones. Después, en la sección 2.2, se exponen varios de los simuladores de vuelo de drones disponibles en el mercado y que se utilizan con frecuencia. Finalmente, en la sección 2.3 se detalla información sobre simuladores de vuelo de drones en diferentes ambientes profesionales.

2.1. La importancia de un simulador de vuelo de drones

Los drones son vehículos aéreos que comenzaron a utilizarse en aplicaciones militares. Con la evolución de su tecnología a través de los años, han llegado a abarcar muchos otros sectores, y apuntan a ser un producto disruptivo en un futuro no tan lejano.

El uso de drones ya está extendido en industrias de entretenimiento o información como la cinematográfica o la periodística, así como en cuerpos de seguridad. Los drones son utilizados para mapeo y vigilancia de grandes áreas, así como para operaciones de rescate. En la industria de la agricultura es una herramienta útil para analizar los suelos o realizar aspersiones de forma mucho más rápida. También pueden ayudar en la organización de grandes almacenes o en la industria de la construcción, si están capacitados para transportar objetos. [13]

La promesa de los drones de cara al futuro es el transporte de mercancías en grandes ciudades gracias al desarrollo del IoT. Una red de drones transportistas en las ciudades reducirían el conocido problema de la última milla, que es la enorme dificultad para optimizar el transporte de mercancías en el último tramo antes de la entrega al usuario. [14]

Sabiendo la utilidad que los drones tienen, y el rápido desarrollo en el que se encuentran en estos momentos, aparece la necesidad de personas en la industria que los controlen con destreza. Manejar un dron no es algo inmediato de aprender, y por ello surge la idea de un simulador de vuelo de un dron. Este simulador permite volar el dron en un entorno virtual, en una pantalla, con características prácticamente idénticas a las de la realidad.

Los simuladores de drones están siendo clave en la disminución de accidentes, permitiendo al usuario aprender a conducir uno sin causar daños a su alrededor. Además, una mala maniobra puede dañar el propio dron. Considerando su alto coste, un simulador evita gastar dinero en reparaciones por una conducción inexperta. [15]

Otras aplicaciones para las que se utilizan los simuladores de vuelo pueden ser evaluar diferentes estrategias y parámetros de control para vuelos autónomos, o el entrenamiento de sistemas de navegación basados en inteligencia artificial. [16] [17]

2.2. Simuladores desarrollados y disponibles en el mercado

Los simuladores de vuelo de drones disponibles actualmente en el mercado son muy variados y tienen distintas características, aunque el objetivo en todos ellos es ofrecer la experiencia más realista posible.

La mayoría de ellos son simulaciones FPV (First Person View), es decir, por la pantalla vemos el vuelo del dron en primera persona, como si estuviésemos montados en él. Esto es debido a que los drones reales suelen incorporar una cámara para que su conducción sea sencilla una vez el dron se ha alejado del conductor. De esta manera, controlar el dron contemplando lo que el dron ve en cada momento es la forma más adecuada de acercarse a la conducción de un dron real.

Los simuladores suelen poder manejarse mediante el teclado del ordenador o un mando de consola. Sin embargo, son más interesantes los simuladores que permiten conectarse a un mando mediante radiotransmisión, ya que de esta manera se puede controlar la simulación con el mando que se va a utilizar para volar el dron real. Algunos ejemplos de simuladores disponibles en el mercado son: [18]

- Liftoff Simulator: [19] Es un videojuego diseñado para el vuelo FPV, tanto para veteranos como para principiantes. Incluye transmisión de radio. Tiene gran variedad de escenarios de vuelo y de modalidades de simulación, como vuelo libre o carreras contra otros usuarios.

  Es pionero en incluir opciones que permiten modificar componentes y ajustes, como si se tratase de un dron real. Las réplicas de motores, baterías, y componentes del dron son de alta fidelidad. Con tantas opciones, se puede realizar la customización de tu propio dron al completo eligiendo las partes deseadas.

  Una característica para destacar es que puntúa los movimientos más efectivos y resolutivos, lo que puede ser de gran utilidad si se quieren comparan las conducciones de distintos usuarios.

  Precio: 20 €

- VelociDrone FPV Racing Simulator: [20] Está diseñado para el vuelo FPV. Incluye transmisión de radio. Las simulaciones están centradas en competición de carreras contra otros usuarios o mejorar el tiempo de vuelta en contrarreloj.

  Se puede elegir entre muchos circuitos diferentes, así como pueden customizarse circuitos con diferentes obstáculos fijos o móviles. Los circuitos creados por los usuarios pueden compartirse para que cualquiera pueda probarlos.

  Las físicas son altamente realistas, y permite controlar varios de los modelos de los drones más conocidos. La ventaja de este simulador frente a otros es que se puede ajustar la gravedad, la inercia, o la potencia de los motores del dron, entre otras características físicas.

  Precio: 20 €

- DRL Racing Simulator: [21] Desarrollado por Epic Games. DRL Simulator es el principal simulador de carreras de la Drone Racing League, la liga mundial profesional de carreras de drones. DRL Simulator contiene las pistas en las que compiten los pilotos profesionales en el mundo real, actualizándose cada año. Es importante aclarar que se trata de una herramienta más centrada en el entretenimiento y el realismo gráfico que en el realismo de las físicas del vuelo.

  Precio: 8 €

- DJI Flight Simulator: [22] El simulador de la empresa SZ DJI Technology Co., líder mundial de vehículos no tripulados para fotografía aérea. Simulación de vuelo de drones comercializados por DJI, que permite a parte de controlar el dron, mover el gimbal (movimiento y estabilizador de la cámara). Compatible con los mandos de control de drones de DJI.

  Drones de uso recreativo en la versión gratis, y drones profesionales en la versión de pago. La versión de pago contiene entrenamientos para trabajos específicos como pueden ser inspecciones de líneas de tensión, o misiones de búsqueda y salvamento. Alto realismo en el vuelo y el entorno, con efectos del viento y colisiones de gran acabado.

  Precio: 1500 € (Versión profesional para empresas)

- DroneSim Pro Flight Simulator: [23] Simulador con gráficos básicos pero modelados físicos del vuelo muy exactos. Solo funciona con mandos de conexión por cable.

Precio: Gratis (Versión Pro de pago)

- Freerider: [24] Simulador pionero en drones de carreras, aunque muy básico. Existe una versión totalmente gratuita con un solo circuito. El simulador completo es barato.

  Precio: 4 € (Gratis versión con un solo circuito))

2.3. Simuladores en el mundo profesional

Ingenieros e investigadores pueden utilizar los simuladores para un gran número de aplicaciones diferentes. El diseño gráfico y aerodinámico de la simulación es esencial para conseguir una experiencia técnica satisfactoria. En el mundo profesional, para cada aplicación concreta del vuelo de un dron, se necesitarán unas características concretas en la simulación. Por ello, es necesario modelar el simulador específicamente para la tarea y el ambiente necesario. Actualmente, no es necesario empezar a programar el simulador desde cero, ya que se pueden descargar simuladores base, de código abierto o comerciales, a partir de los cuáles desarrollar un simulador acomodado a las necesidades requeridas. A continuación, se exponen algunos de los simuladores y proyectos en desarrollo que se engloban en el mundo profesional relacionado con el vuelo y el desarrollo de infraestructuras de drones [25]:

- AirSim

  AirSim es un simulador de vuelo de drones de código abierto desarrollado por Microsoft Aerial Informatics and Robotics (AIR), introducido principalmente para el desarrollo de algoritmos de aprendizaje de drones autónomos o generar datos para construir modelos de machine learning. El código de AirSim está modelado para ser un plugin que se añade a UnReal Engine.

  En el vuelo de un dron, el estado deseado son unos nuevos ángulos de precesión, nutación y espín. El controlador estima los ángulos actuales mediante el acelerómetro y el giroscopio, y cambia las señales de los motores para conseguir los ángulos deseados. En el simulador de AirSim el proceso es el mismo, el dron simulado cambia su posición a los nuevos ángulos deseados, siendo el modelo dinámico del vehículo el que genera el desplazamiento.

  Otras fuerzas como la gravedad o la fricción pueden añadirse a la simulación también. Añadidos interesantes pueden ser la identificación de la ruta según del dron se desplaza, así como datos de proximidad mediante sensores o detección de colisiones con otros objetos.

- Banco de pruebas para red inalámbrica de drones

  La Universidad de Colorado ha desarrollado este sistema que recoge datos de red como pueden ser el rango, retraso o conectividad bajo diferentes condiciones de vuelo. Es esencial para evaluar el comportamiento autónomo en una red de drones que se comunican entre ellos.

- Simbeeotic

  Es un simulador desarrollado por la Universidad de Harvard como parte del proyecto RoboBees. Simula la infraestructura de comunicación en un ambiente virtual en 3D para trabajar en la mejora de los puntos débiles del sistema. El proyecto es un conjunto de MAVs (Micro Air Vehicles) que trabajan como un enjambre y deben cumplir la misma función polinizadora que las abejas.

- RAVEN (Real-time indoor Autonomous Vehicle test Environment)

  Es un simulador desarrollado por MIT para estudiar misiones de varios drones de larga duración en un ambiente cerrado y controlado. Se pueden llegar a manejar simultáneamente diez UAVs de forma coordinada mediante algoritmos de control.

- RotorS Simulator

  Simulador desarrollado por el Instituto Federal de Tecnología de Zurich. Incluye sensores de cámara y también cambios de la inercia del dron según la carga con la que vuele. Su función es mejorar los caminos de los vuelos de una manera más sencilla, reduciendo los accidentes de drones reales.

- VAMPIRE Unmanned Air System

  Es un simulador para entrenamiento de pilotos de drones. Recrea distintos entrenamientos para operadores con distintas funciones o emergencias. Los ambientes son altamente detallados, incluyendo los urbanos movimientos de personas y vehículos.

- Zephyr Drone Simulator

  Diseñado para mejorar las habilidades de conducción de los pilotos profesionales de la industria. Es un simulador disponible para Windows y Mac. Informa sobre

el progreso del piloto, y la experiencia es tan afín a al de un vuelo real que el cambio de la simulación a este apenas se aprecia.

- Computational Multicopter Design

  Investigadores del MIT han desarrollado este software de simulación con el que se puede diseñar la posición, tamaño y tipo de componentes de un dron desde cero. El software tiene en cuenta todos estos parámetros a la hora de simular el vuelo, decidiendo si el diseño será capaz de volar o no, y en caso de hacerlo mostrando si volará correctamente. Se determina si el diseño es práctico basándose en el desplazamiento y los giros, tratando de mostrar qué partes deben cambiarse para alcanzar la mejor optimización posible. Los parámetros pueden cambiarse en tiempo real durante el vuelo.

- D-MUNS (Distributed Multiple UAV Net Simulator)

  Incluye tres componentes que utilizan simuladores ya desarrollados para conseguir una simulación más completa. Se utiliza RotorS en un monitor para simular distintos vuelos de UAVs, enviando información como la posición actual de cada uno de ellos a un simulador de la infraestructura de una red de drones que se encuentra en un segundo monitor. Por último, un tercer monitor muestra el estado de la infraestructura completa.

3. Desarrollo del entorno virtual y del modelo estructural del dron

En este capítulo se desarrolla cómo implementar el simulador AirSim en un proyecto de Unreal, al cual se le importará un entorno tridimensional desarrollado anteriormente. También se explica cómo cambiar el modelo estructural genérico del dron que AirSim tiene por defecto, para que se utilice un modelo estructural diseñado previamente. La sección 3.1 trata cómo ha sido el diseño previo del modelo estructural del dron y del entorno virtual. La sección 3.2 trata la importación de este entorno virtual a un proyecto de UnReal. La sección 3.3 desarrolla cómo se implementa el simulador AirSim al proyecto de Unreal, mientras la 3.4 explica cómo cambiar el dron genérico del simulador por el modelo estructural deseado.

3.1. Diseño 3D del entorno virtual y del dron

La estructura del dron ha sido diseñada previamente en SolidWorks. Las partes han sido ensambladas generando el modelo tridimensional que se utilizará para la simulación. El archivo es un objeto 3D (.obj) que debe exportarse con suficiente resolución superficial para poder ser importado adecuadamente al entorno de UnReal posteriormente.

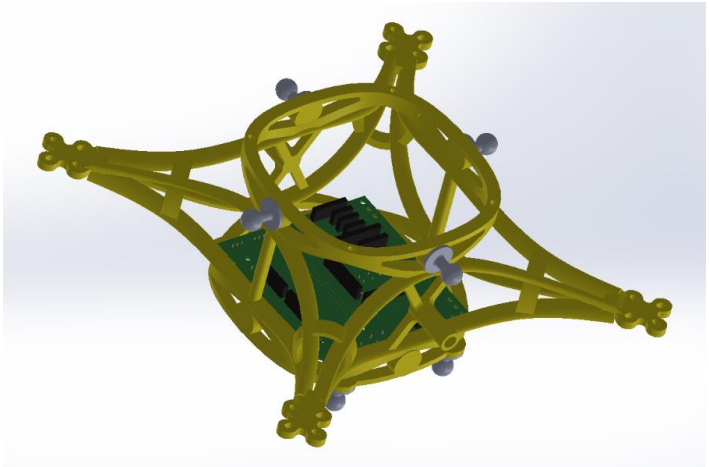

Ilustración 2: Diseño 3D de la estructura del dron en SolidWorks

La estructura diseñada incluye tanto la base para los componentes del dron, como una carcasa esférica exterior. La finalidad de esta carcasa es proteger tanto al dron como a los objetos y personas que se encuentren en el entorno de vuelo, en caso de que se produzca un accidente.

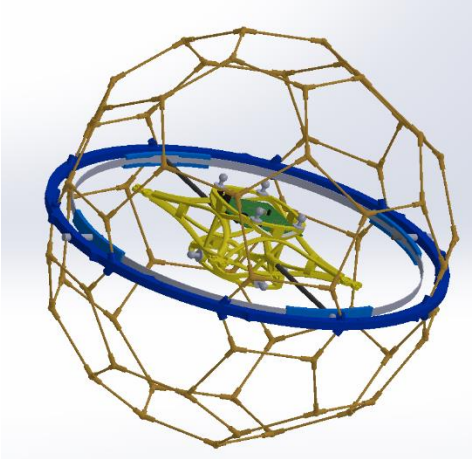

Ilustración 3: Diseño 3D de la estructura del dron en SolidWorks incluyendo carcasa

Las hélices han quedado sin ensamblar, siendo otro objeto 3D (.obj) que se importará por separado al programa. Esto es debido a que de esta manera se podrá generar la animación de giro de las hélices, que no sería posible si formasen parte de un ensamblaje total junto al resto del dron.

El entorno virtual ha sido diseñado mediante SkecthUp, recreando con la mayor exactitud posible el laboratorio de regulación automática y de control digital de la universidad. Para ello, también han sido diseñados los muebles, ordenadores, osciloscopios, y demás objetos de este ambiente, utilizando SolidWorks. Los objetos 3D (.obj) de estos muebles han sido importados y colocados en sus respectivas posiciones en el entorno de SketchUp mencionado.

El entorno es exportado como un archivo filmbox (.fbx), que se importará también al proyecto de UnReal posteriormente. Al exportar el archivo deben seleccionarse las opciones avanzadas de triangular caras, y de exportación de las dos caras. Esto es para para evitar problemas posteriores en la importación a UnReal, como pueden ser que la resolución de las caras no sea adecuada, o que solo podamos ver una de ellas en el caso de objetos planos.

Este material previo, tanto el diseño de la estructura del dron y de las hélices, como el entorno virtual del laboratorio, han sido facilitados por María Ana Saenz Nuño, codirectora de este proyecto. La estructura del dron es el resultado del trabajo de fin de grado *Diseño Mecánico de un Dron para Inventario Automático de Almacenes*, de Diego Cubillo Llanes [26].

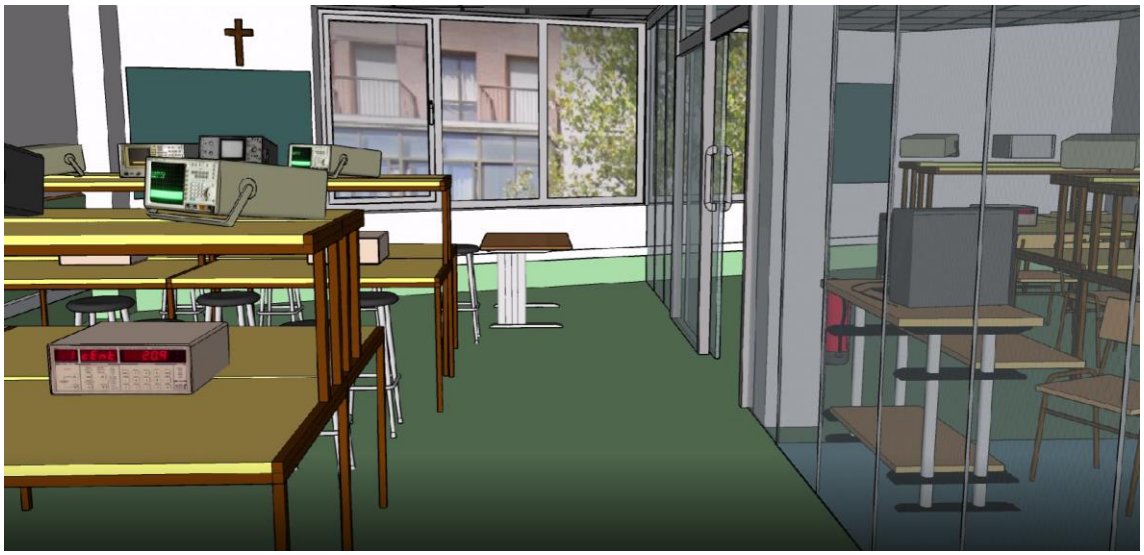

Ilustración 4: Diseño del entorno virtual en SketchUp, perspectiva 1

Ilustración 5: Diseño del entorno virtual en SketchUp, perspectiva 2

3.2. Importación del entorno virtual a un proyecto de UnReal

Para empezar, se debe crear un nuevo proyecto de UnReal Engine. En este proyecto es en el cual se van a integrar el entorno virtual, la estructura del dron, y el simulador de vuelo; siendo esta plataforma a través de la cual se visualizará la simulación.

La descarga de UnReal Engine se realiza a través de la aplicación Epic Games Launcher, siendo la versión 4.25.4 de UnReal la utilizada en este proyecto.

[27] Para crear un nuevo proyecto abrimos la aplicación UnReal Engine. En New Project Categories se selecciona la categoría Games, y se abrirá una ventana para seleccionar una plantilla con la que comenzar a diseñar el proyecto (Select Template). Aquí interesa elegir la opción Blank, para que cree un proyecto completamente vacío y sin código.

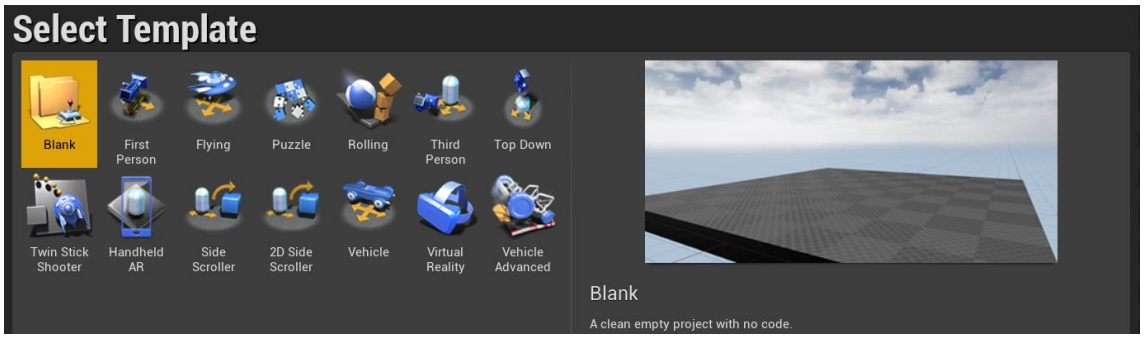

Ilustración 6: Plantilla en blanco de un nuevo proyecto UnReal

Con el proyecto abierto, en la barra de herramientas superior se encuentra la opción Import Unreal Datasmith File. Datasmith es un plugin cuya función es convertir e importar archivos a Unreal en un formato que este programa sea capaz de procesar.

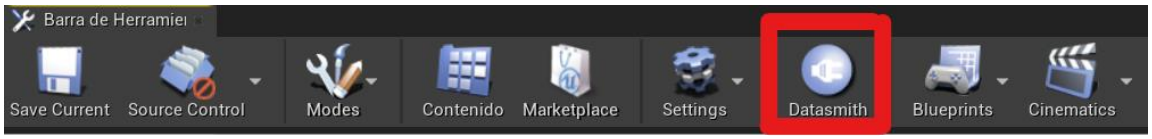

Ilustración 7: Icono de Datasmith en la barra de herramientas de UnReal

Se importa archivo filmbox (.fbx) del entorno del laboratorio al proyecto. En las opciones de importación se ajusta la escala a x100, para que los diseños de Sketchup no se vean muy pequeños. También se debe desactivar la casilla “Auto Generate Collision”, que genera automáticamente colisiones entre los objetos importados y el jugador, que en nuestro caso será el dron. Estas colisiones autogeneradas deben desactivarse ya que con ellas el dron no podrá volar por dentro de la habitación diseñada, que sería interpretada por el programa como un volumen sólido en vez de hueco.

Ilustración 8: Opciones de importación de archivo FBX en UnReal

Ilustración 9: Entorno del laboratorio importado al proyecto de UnReal

El archivo se importa dividido en varios Static Mesh. Este es el nombre que llevan todos los objetos del proyecto que podrían tener propiedades físicas o interacciones con otros objetos. El actor u objeto que controlará el jugador, en este caso el dron, también será un Static Mesh.

Si se quiere cambiar la posición de algún Static Mesh u objeto que no haya quedado correctamente colocado, tan solo se pulsa sobre él y aparecerá un pivote con unos ejes de coordenadas en las tres dimensiones. Pulsando sobre los distintos ejes y arrastrando, podemos moverlo para colocarlo debidamente. Pulsando E en el teclado, los ejes del pivote se cambian por ángulos para poder inclinar el objeto. Pulsando R, el pivote pasa a tener la utilidad de cambiar la escala del objeto en las distintas direcciones. Para volver al pivote de posicionamiento, se pulsa W.

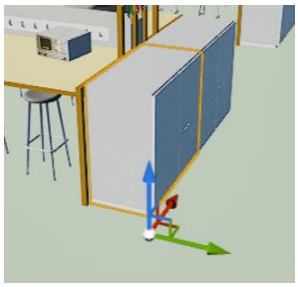

Ilustración 10: Pivote de posicionamiento

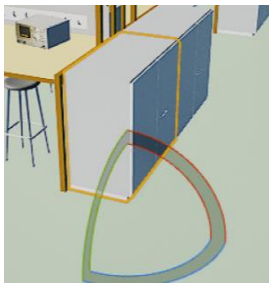

Ilustración 11: Pivote de orientación angular

Ilustración 12: Pivote de escala

Si se desea que el pivote de un objeto coincida con algún punto en concreto de este, como podría ser su centro de gravedad, se mantiene pulsado ALT y el botón central del ratón mientras lo desplazamos al punto deseado. Una vez situado donde nos interesa, se pulsa botón derecho, y luego *Pivote*. En el desplegable seleccionamos *Set as Pivot Offset*.

Como la cantidad de objetos del laboratorio es muy alta, se generan muchas sombras indeseadas que hacen que los objetos dejen de verse con claridad. Para solucionarlo, haciendo doble clic en el Static Mesh de la habitación, se abrirá una ventana para ajustar diferentes propiedades del objeto. En *Lighting*, se deselecciona la opción *Sombra proyectada*, por lo que no se proyectarán sombras sobre el suelo y paredes del laboratorio de todos los objetos que se encuentran sobre él.

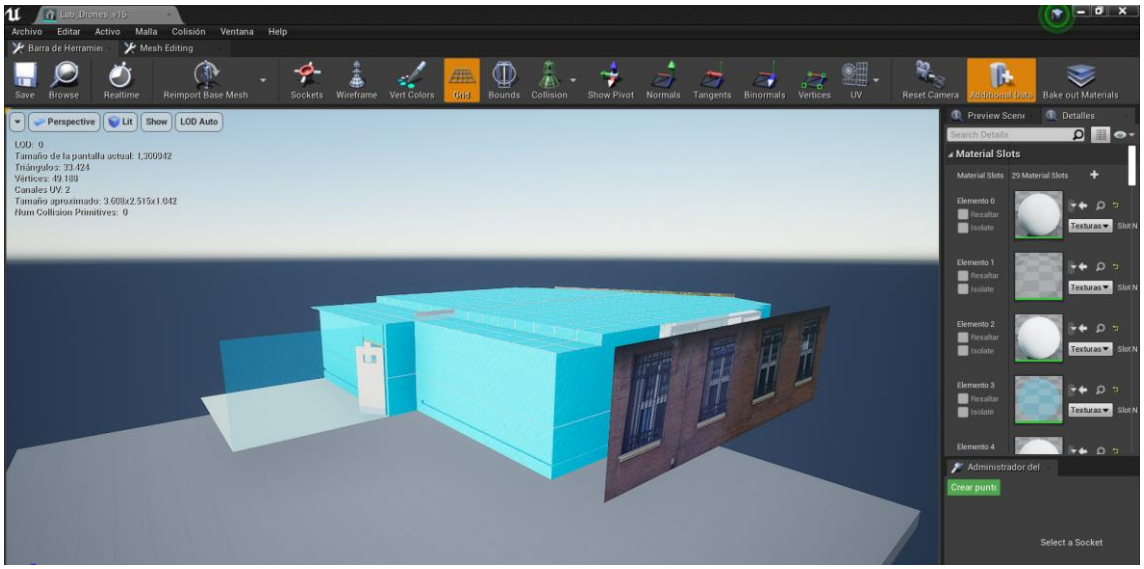

Ilustración 13: Ventana de ajuste de propiedades del Static Mesh de la habitación

Entre las características que pueden ajustarse en esta ventana también se encuentra el tipo de colisión. Con colisión se refiere a la propiedad que hace que un determinado objeto no pueda ser atravesado por otros objetos. La forma en la que UnReal dota a un objeto de esta propiedad es mediante una malla invisible alrededor de la superficie del objeto, que no atravesará las mallas de otros objetos.

Existen dos tipos de mallas, las simples y las complejas. Las mallas complejas cubren mediante una gran cantidad de puntos y uniones entre ellos toda la superficie del objeto, y están pensadas para objetos de formas irregulares. Las mallas simples

siguen patrones regulares y más sencillos que intentan cubrir toda la superficie del objeto, y aunque pueden dejar pequeñas esquinas o recovecos sin cubrir, suelen cumplir su función correctamente. El archivo final será de un tamaño menor si se utilizan colisiones simples que si se utilizan colisiones complejas en todos los objetos.

Pulsando en *Colisión* en la parte superior izquierda de la ventana de propiedades del objeto importado, se despliegan varias opciones con las que añadir una malla simple al objeto. En este caso se opta por *26DOP Simplified Collision*, aunque otras opciones que cubriesen correctamente la superficie también cumplirían bien su función.

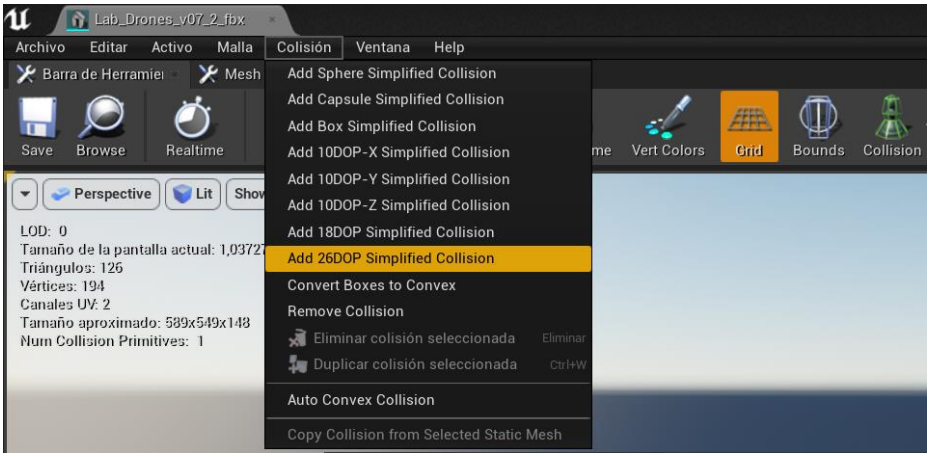

Ilustración 14: Añadir malla simple para colisiones a la habitación

En la parte derecha de esa misma ventana, se busca el apartado *Collision* que se encuentra en *Detalles*, donde se puede elegir qué tipos de colisiones queremos que sean utilizadas en el objeto. Se selecciona *Usar la colisión compleja como colisión simple* para que la malla utilizada cuando haya colisiones complejas con otros objetos sea la simple que se acaba de crear en vez de una compleja generada por el programa.

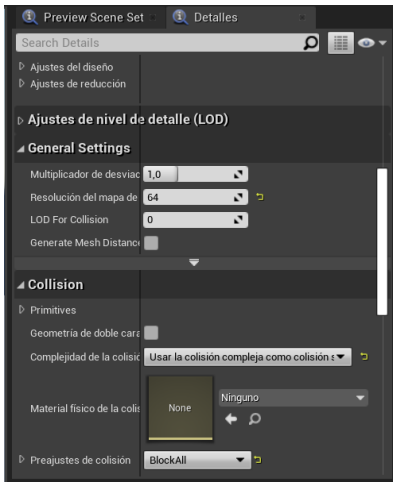

Ilustración 15: Elegir complejidad de la colisión de la habitación

Para visualizar como ha quedado la malla cubriendo la superficie del objeto, se puede seleccionar *Show Collision* en la parte superior de la ventana. Se nos permite visualizar tanto la malla simple, como la compleja, o ambas a la vez.

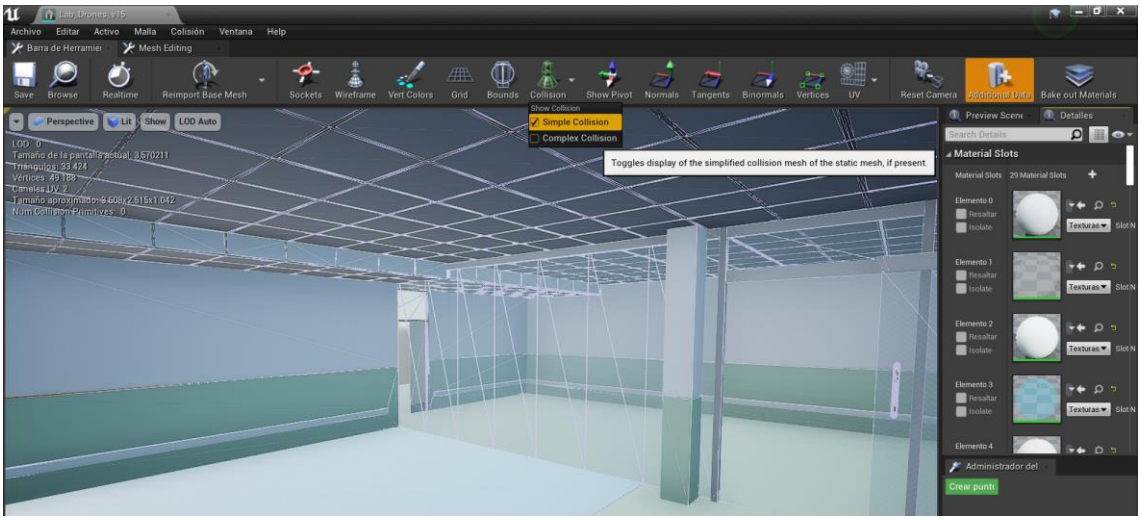

Ilustración 16: Visualización de la malla simple de la habitación

Algunos materiales deben ser parcialmente transparentes y pueden haber perdido esta propiedad en la importación. Para cambiar la transparencia de un material se busca el material en la parte derecha de la ventana de propiedades, y haciendo doble clic sobre él se abre una ventana con un esquema con conectores sobre propiedades de este material. Estos conectores son un sistema de blueprints. Un sistema de blueprints es una simplificación de la programación C++ de UnReal, en la que el usuario puede utilizar un sistema de nodos y conectores en vez de programar escribiendo código.

Se crea un nuevo bloque que sea una constante, y se conecta a la propiedad *Opacity*. El valor 0 en esta constante indica que el material será completamente transparente, y su opacidad se incrementa linealmente hasta el valor 1, que indica que es completamente opaco. En este esquema de conectores se podrían cambiar otras propiedades como pueden ser el color de los objetos o la forma en la que la luz se refracta al pasar a través de ellos.

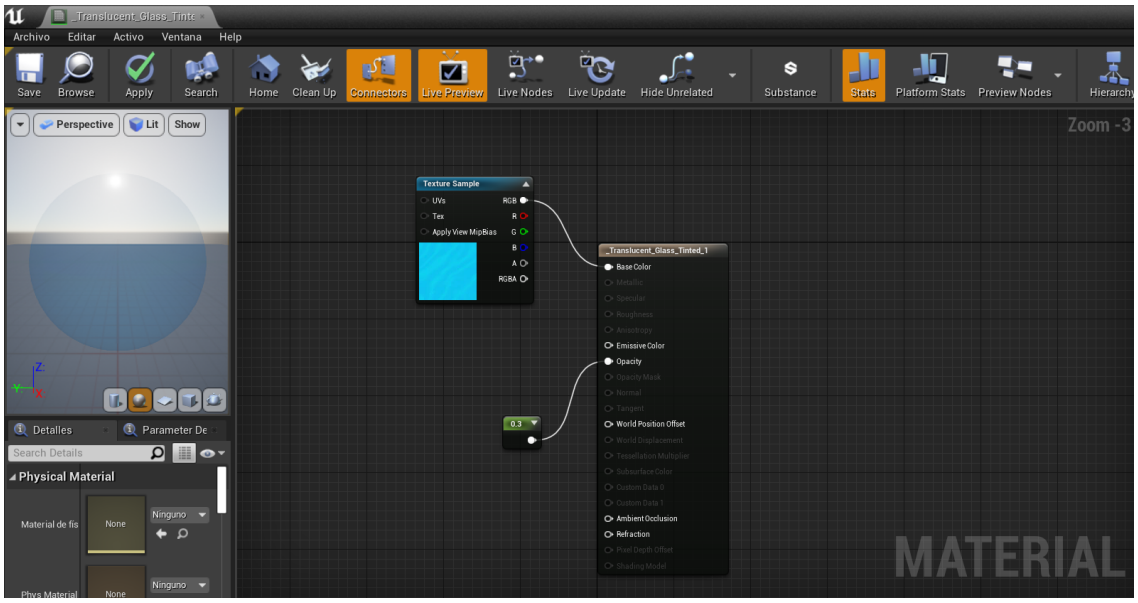

Ilustración 17: Cambio de transparencia de un material mediante su blueprint

3.3. Implantación del simulador AirSim a UnReal

[28] El proyecto de UnReal creado contiene un documento programado en C++ dentro de la carpeta *Config*, con configuraciones iniciales del proyecto en blanco y añadidos posteriores. El documento tiene la extensión de los proyectos de UnReal (*.uproject*). Se abre este documento mediante el editor de texto Visual Studio, siendo el código creado por defecto en el proyecto en blanco el mostrado en el Código 1.

Código 1: ProyectoLAB4_6.uproject, Código inicial por defecto en proyecto en blanco

```
{
	"FileVersion": 3,
	"EngineAssociation": "4.25",
	"Category": "",
	"Description": "",

	"Modules": [
		{
		"Name": "ProyectoLAB4_6",
		"Type": "Runtime",
		"LoadingPhase": "Default",
		}

	]

}
```

Para implantar AirSim en el proyecto, se ejecuta como administrador *x64 Native Tools Command Prompt for VS 2019*. A través de esta ventana de comandos, se accede a la carpeta donde se haya instalado AirSim, la carpeta se encuentra dentro de *Unreal* y se llama *AirSim-master*. Mediante el comando *build.cmd* (*Build Command Script)*, se crea la carpeta *Plugins* dentro de *AirSim-master* Esta carpeta contendrá los plugins que pueden añadirse a cualquier proyecto de UnReal para utilizar el simulador de AirSim en él.

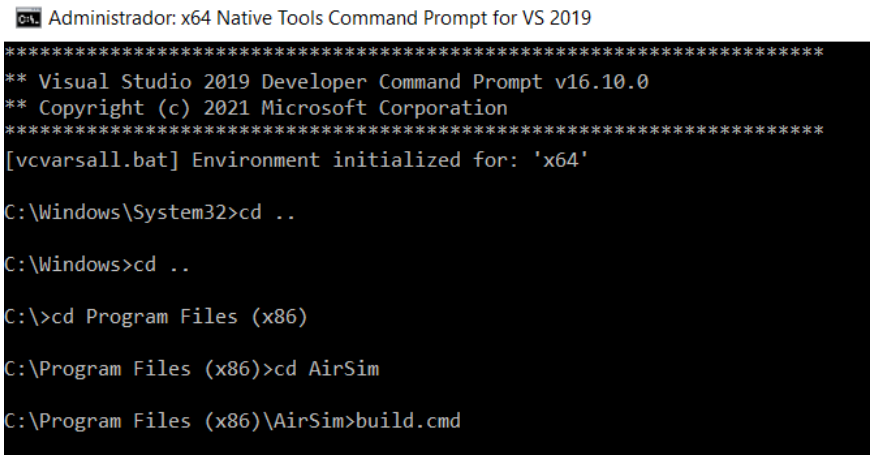

Ilustración 18: Creación de la carpeta de plugins de AirSim mediante build.cmd

Para añadir estos plugins al proyecto, debe copiarse la carpeta *Plugins* a la del proyecto, y añadir al Código 1 la dependencia del proyecto con AirSim y los nuevos plugins. El código que debe añadirse se muestra en el Código 2.

Código 2: ProyectoLAB4_6.uproject, Código con plugins de AirSim

```
{
	"FileVersion": 3,
	"EngineAssociation": "4.25",
	"Category": "",
	"Description": "",
	"Modules": [
		{
		"Name": "ProyectoLAB4_6",
		"Type": "Runtime",
		"LoadingPhase": "Default",
		"AdditionalDependencies": [
				"AirSim"
			]
		}
	],
	"Plugins": [
		{
			"Name": "AirSim",
			"Enabled": true
		}

	]
}
```

Al abrir de nuevo UnReal puede aparecer una ventana informando de que el módulo de AirSim ha sido construido con una versión del programa diferente o previa a la que se está utilizando. Esta ventana pregunta si se quiere reconstruir el módulo de AirSim, y debe responderse que sí o no funcionará el simulador.

Una vez abierto el proyecto de UnReal, actualizar el código en *Archivo*, pulsando en el desplegable *Actualizar Visual Studio proyecto*. En *Ventana*, *Ajustes del mundo*, se encuentra la pestaña *GameMode Override*, donde se puede seleccionar el tipo de juego o reproducción que va a ocurrir cuando simulemos el proyecto. Gracia a los plugins añadidos, puede seleccionarse la opción *AirSimGameMode*, con lo que ya se simulará el vuelo del dron de AirSim cuando pulsemos en el botón de simulación. Se coloca el objeto *PlayerStart* donde se desea que aparezca el dron cuando se inicie la simulación.

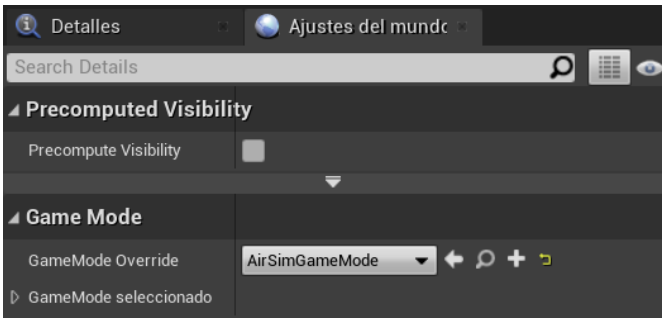

Ilustración 19: Modo de juego

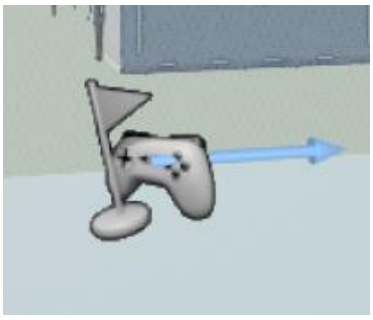

Ilustración 20: PlayerStart

3.4. Cambio del modelo estructural del dron del simulador por el diseño 3D del dron deseado

[29] En la carpeta *Unreal\Plugins\AirSim\Content\Blueprints* se encuentra *BP_FlyingPawn.uasset*, que es el dron que utiliza el simulador de AirSim por defecto. El objetivo es cambiar el modelo estructural de este dron por el modelo estructural del dron del laboratorio. Se copia el modelo de la carpeta mencionada a la carpeta *Content*, que tiene un acceso más sencillo desde la interfaz de UnReal.

Ilustración 21: Modelo estructural por defecto del dron del simulador AirSim

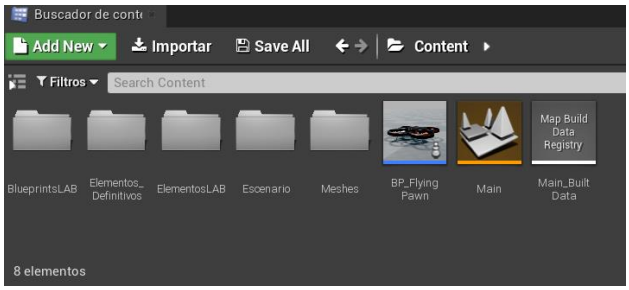

Ilustración 22: Carpeta Content con BP_FlyingPawn.uasset

Abriendo el BP_FlyingPawn.uasset desde el proyecto de UnReal, se muestran todos los detalles del dron. Pulsando sobre la carcasa se muestran los detalles de la carcasa, incluyendo el diseño 3D del modelo estático (Static Mesh). A la derecha de la imagen que muestra el modelo estático, hay una pequeña lupa. Al ser pulsada, la carpeta en la que se encuentra el Static Mesh utilizado para el dron se abre directamente en el buscador de contenido de UnReal, tal y como se muestra en la Ilustración 23 y en la Ilustración 24.

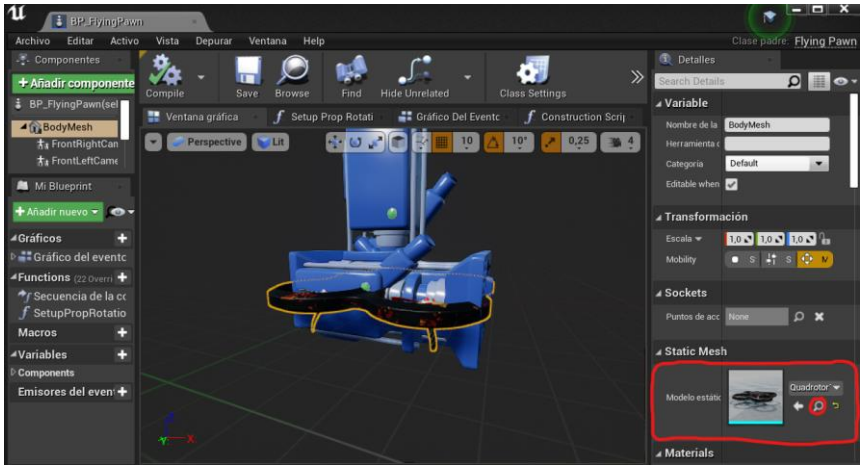

Ilustración 23: Abrir la ubicación del Static Mesh del dron

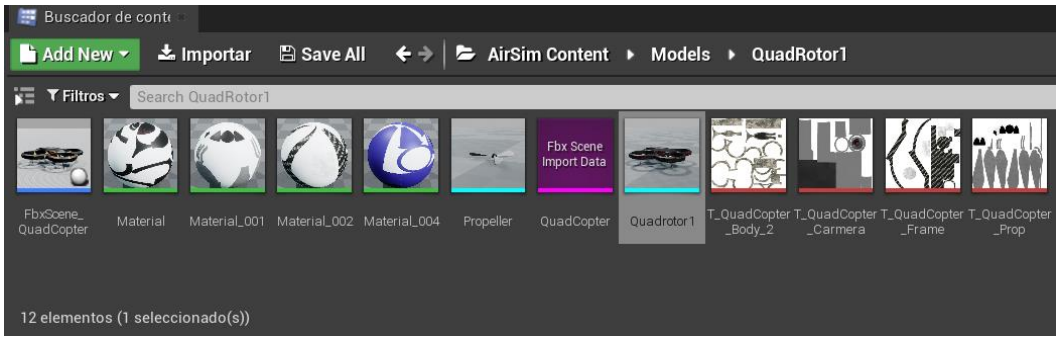

Ilustración 24: Ubicación del Static Mesh del dron

Ahora la carpeta *Blueprints* de *AirSimContent* tiene un acceso directo desde el buscador de contenido. Visualizar el archivo *BP_FlyingPawn.uasset* accediendo a su carpeta de origen desde UnReal ya es inmediato, como muestra la Ilustración 25. Se duplica el archivo para para tener un dron nuevo y no modificar el anterior. Un ejemplo para el nombre al archivo duplicado y así evitar confusiones es *BP_MyPawn.uasset*.

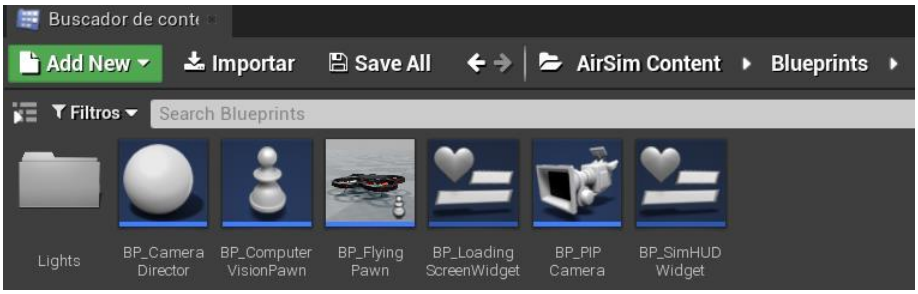

Ilustración 25: Ubicación original de BP_FlyingPawn.uasset

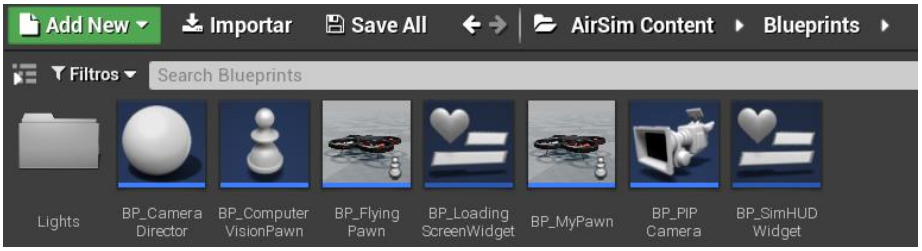

Ilustración 26: Duplicado del dron, llamado BP_MyPawn.uasset

En la misma carpeta *Blueprints* se importan los archivos del ensamblaje del dron y las hélices. Se utiliza una escala 200 en las opciones de importación para que el tamaño sea adecuado.

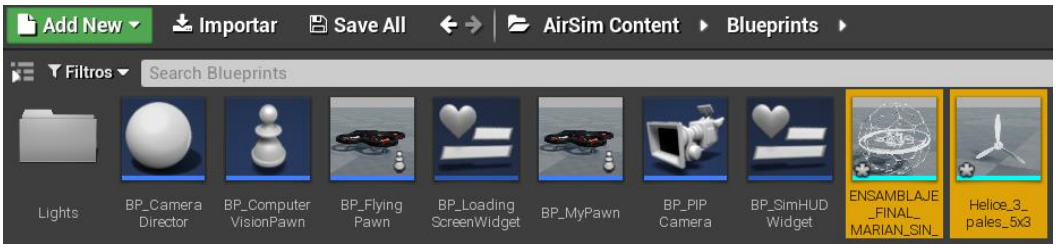

Ilustración 27: Ensamblaje del dron y hélices en la carpeta Blueprints

Abriendo *BP_MyPawn.uasset*, se puede pulsar sobre el ensamblaje para abrir a la derecha la ventana *Detalles* del ensamblaje. En el desplegable de *Static Mesh*, se cambia su modelo estático por el nuevo ensamblaje importado, ajustando su posición para centrarlo y que el plano que simboliza el suelo coincida con el punto más bajo de la estructura del dron. Pulsando sobre cada una de las hélices, se repite el proceso para cambiar su modelo estático por el de la hélice diseñada. Es necesario colocar cada una de las hélices en la posición correcta del ensamblaje como muestran la Ilustración 30 y la Ilustración 31.

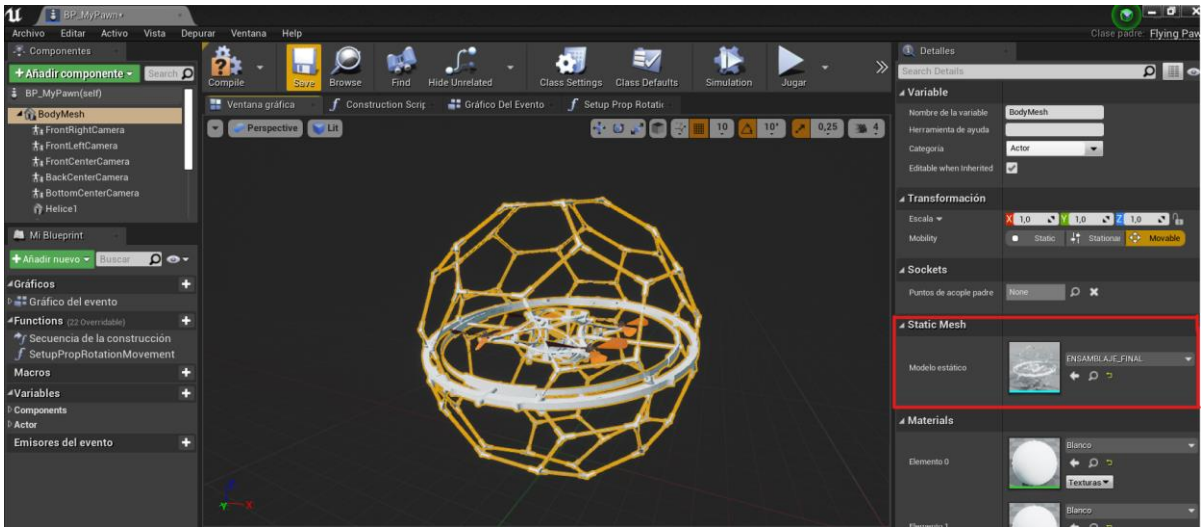

Ilustración 28: Cambio del modelo estático del ensamblaje

También en *Detalles*, en *Transformación*, se debe comprobar que tanto en el ensamblaje como en las hélices que la opción *Movable* está marcada, es decir, que se trata de objetos que en la simulación se van a mover. En *Variable*, es importante poner a cada hélice un nombre distinto y reconocible, esto será necesario para la animación de giro de las hélices.

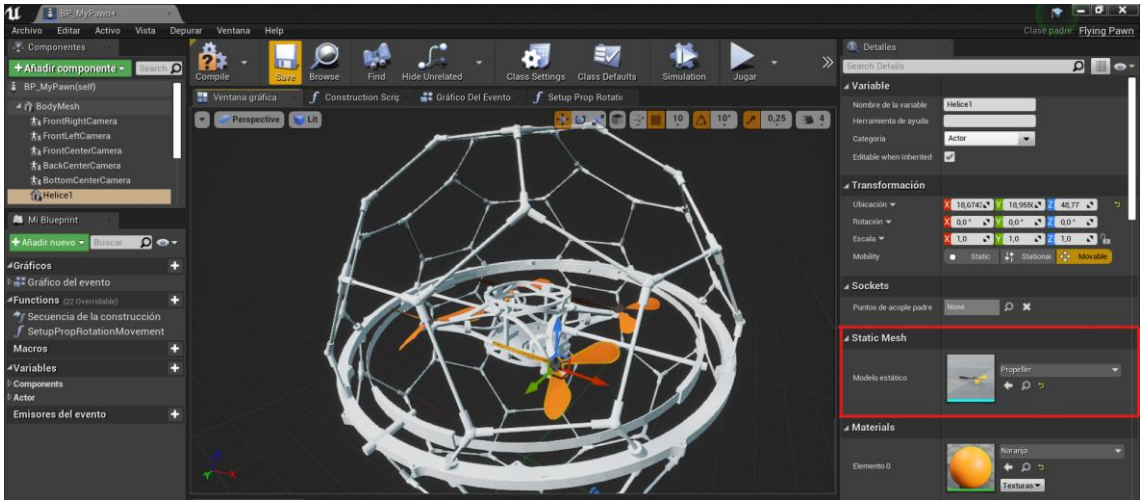

Ilustración 29: Cambio del modelo estático de una hélice

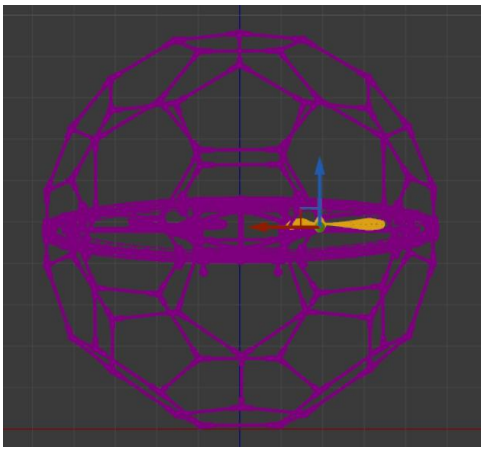

Ilustración 30: Colocación de una hélice vista frontal

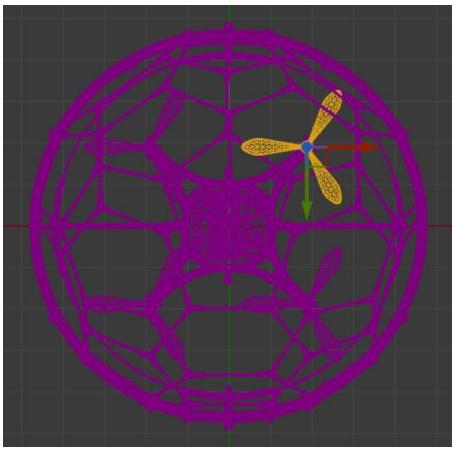

Ilustración 31: Colocación de una hélice vista cenital

Para que las hélices giren, se comprueba que la función *SetupPropRotationMovement* está creada dentro de *BP_MyPawn.uasset* en la ventana *Mi Blueprint*, ya que era la encargada de que girasen en el modelo antes de modificarlo. Si no es así, porque la función haya sido eliminada automáticamente tras las modificaciones, debe crearse la función desde cero como muestra la Ilustración 32. Dentro de esta función se encuentra el esquema blueprint del código que permite que las hélices giren. Si la función acaba de ser creada, el esquema blueprint puede ser copiado desde la misma función que se encuentra en *BP_FlyingPawn.uasset*, para insertarse en la función nueva.

El funcionamiento de este blueprint es sencillo. Al inicializarse la función, cuatro bloques de componentes de movimiento (opción *Movable* activada anteriormente), se actualizan con una animación de movimiento. Los cuatro componentes de movimiento son las cuatro hélices, que van conectadas cada una al nodo *New Updated Component* de cada bloque. Como tienen el nombre de las hélices del modelo previo, cambiar estos nombres por los que se han asignado a cada una de las hélices, asegurando que se sustituyen los nombres correctos para evitar que alguna de las hélices gire en un sentido antinatural. Los cuatro movimientos son de rotación, y estaban desarrollados previamente para el modelo predeterminado, estando conectados al nodo *Meta* de cada bloque. El esquema blueprint de la función se muestra en la Ilustración 33.

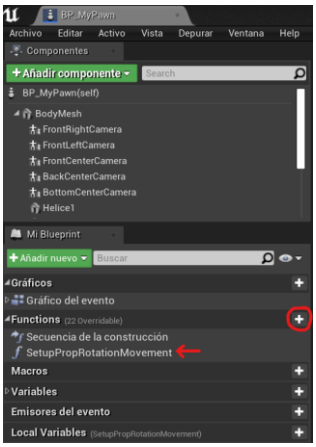

Ilustración 32: Creación de una nueva función en BP_MyPawn

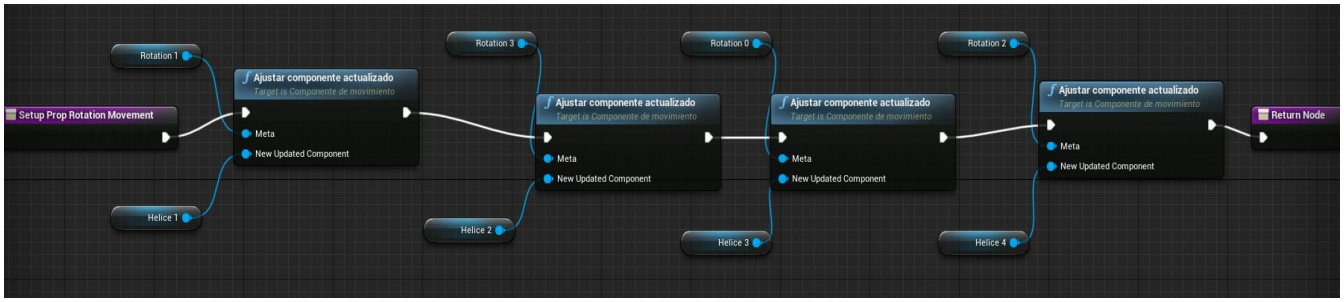

Ilustración 33: Blueprint de la función SetupPropRotationMovement

También en *Mi Blueprint*, en *Gráfico del evento*, debe encontrarse el esquema blueprint que inicializa la función para que las hélices giren cuando comienza la simulación. Se comprueba que el blueprint es el mostrado en la Ilustración 34.

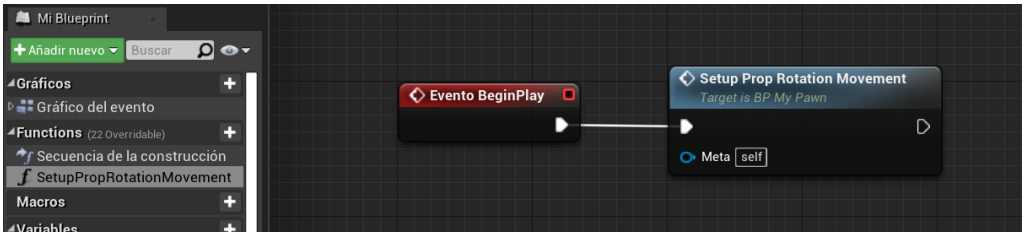

Ilustración 34: Blueprint del gráfico del evento

Todavía queda un pequeño detalle por modificar. Si se simula en este momento, el dron atravesaría el suelo y caería al vacío, esto es porque el modelo estático del ensamblaje nuevo no tiene una malla generada para detectar colisiones con las mallas de otros objetos, como puede ser la malla de la habitación que ha sido generada previamente. El proceso es idéntico al explicado detalladamente en la sección 3.2.

Se crea en el modelo estático del ensamblaje una colisión simple con forma de esfera, *Sphere Simplified Collision*, que se selecciona en el desplegable *Colisión* en la ventana de propiedades del modelo estático del ensamblaje. Esta vez también se elige la opción *Usar la colisión compleja como colisión simple*.

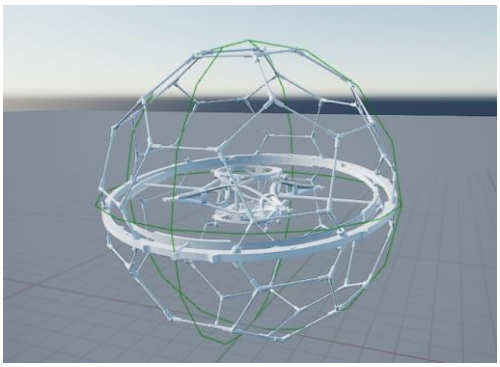

Ilustración 35: Visualización de la malla simple del ensamblaje del dron

Los cambios se han realizado en el archivo *BP_MyPawn.uasset*, pero el simulador seguirá utilizando *BP_FlyingPawn.uasset* por defecto si no se indica que el nuevo archivo es el que AirSim tiene que utilizar. Para ello, se cambia el código *settings.json*, que se encuentra en la carpeta *AirSim*, añadiendo la parte subrayada en gris que se muestra en el Código 3Ilustración 1.

Código 3: settings.json, Añadido para que en la simulación aparezca el nuevo modelo

```
{
  "SeeDocsAt":
"https://github.com/Microsoft/AirSim/blob/master/docs/settings.md",
  "SettingsVersion": 1.2,
  "PawnPaths": {
    "DefaultQuadrotor": { "PawnBP":
"Class'/AirSim/Blueprints/BP_MyPawn.BP_MyPawn_C'" }
      }
}
```

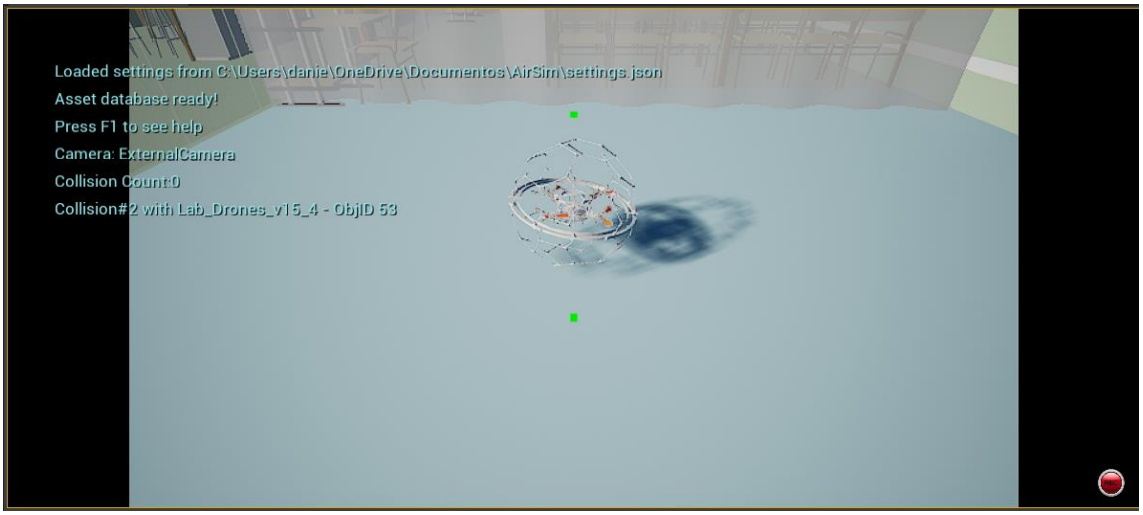

Ilustración 36: Simulación AirSim con el modelo estructural del dron del laboratorio

4. Programación en Python de la comunicación entre AirSim y Simulink

En este capítulo se desarrolla como se va a comunicar Matlab con AirSim a través de Python. En la sección 4.1 se explica qué información necesita recibir AirSim, que va a generarse en Simulink. En la sección 4.2 se expone el código de Python necesario para la comunicación entre los programas. En la sección 4.3 se explica el esquema de Simulink necesario para enviar la información a AirSim. Por último, en la sección 4.4 se desarrolla cómo se ha gestionado el problema de las colisiones del dron con las paredes; mientras en la sección 4.5 se ha delimitado la zona de vuelo del dron.

4.1. Funcionamiento de la simulación: Qué recibe AirSim

El simulador de vuelo AirSim funciona a través de cualquier mando de videoconsola o con radiotransmisor que sea compatible con UnReal. Si conectamos un mando al ordenador en este momento, podremos volar el dron, aunque las físicas del vuelo serán las preestablecidas por el simulador de AirSim.

El simulador recibe la información de la posición de los joysticks del mando, y la transforma en el desplazamiento del dron en las tres direcciones del espacio, así como en variación de la orientación angular del mismo según los tres ángulos de Euler; precesión, nutación, y espín.

El problema es que esta simulación transforma la información de los joysticks en desplazamiento y ángulos del dron según el peso, tipo de estructura y condiciones ambientales que han sido pensados y desarrollados en un modelo dinámico para que el vuelo de AirSim sea lo más parecido posible a cómo sería el vuelo del dron del simulador en la realidad. Esto es aplicable al dron predeterminado de AirSim, pero no al dron del laboratorio, que tiene estructura y peso diferentes al de AirSim.

Para solucionar este problema, se utilizará el modelo dinámico del dron del laboratorio, desarrollado previamente en Matlab y Simulink. Con un radiotransmisor conectado a Simulink, será este modelo dinámico el que transformará la información de los joysticks en el desplazamiento en las tres direcciones del espacio y la variación de ángulos de Euler. Estos seis datos serán los que tendrán que enviarse desde Simulink a AirSim a través de Python, para que el simulador los utilice en el movimiento del dron

en vez de utilizar los generados en el modelo dinámico predeterminado en el simulador inicial.

Los desplazamientos en las tres direcciones del espacio de un dron están representados en la Ilustración 37. Se compone de los movimientos en los ejes x e y para el desplazamiento del dron en un plano paralelo al suelo, y el throttle, que es como se llama al desplazamiento en el eje vertical z para cambiar la altura del vuelo. Los ángulos de Euler en la orientación en el vuelo de un dron están representados en la Ilustración 38, y se componen del alabeo o roll en el eje longitudinal al dron, el cabeceo o pitch en el eje lateral del dron, y la guiñada o yaw en el eje vertical. En el vuelo de un dron, el cabeceo es el que genera el desplazamiento en x, mientras que el alabeo es el que genera el desplazamiento en y. El movimiento de guiñada sirve para orientar el dron, mientras que el throttle o desplazamiento vertical se provoca aumentando o disminuyendo la potencia de los motores de las cuatro hélices por igual.

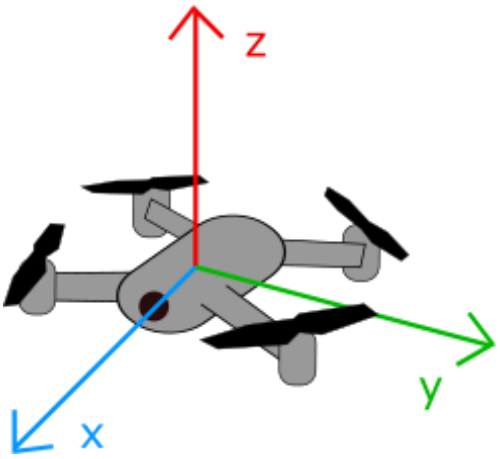

Ilustración 37: Desplazamientos de un dron

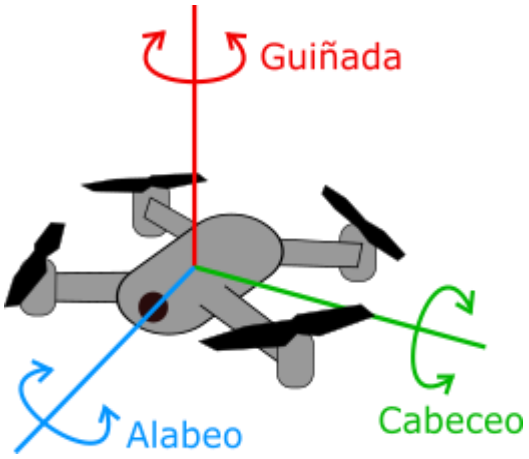

Ilustración 38: Ángulos de Euler en el vuelo de un dron

Partiendo de esta idea, se crean en Simulink seis señales que simbolizan los tres desplazamientos y las tres variaciones de ángulos, para generar el código que envíe estas seis señales a AirSim. De esta manera, se podrá trabajar de una manera más sencilla el enviar los seis datos de movimiento y ángulos, quedando pendiente para más adelante unir esta parte con las salidas del modelo dinámico del dron en lugar de con las seis señales.

Las tres señales del desplazamiento son de tipo escalón para que el dron se desplace de forma lineal y podamos comprobar fácilmente que se están enviando correctamente los datos.

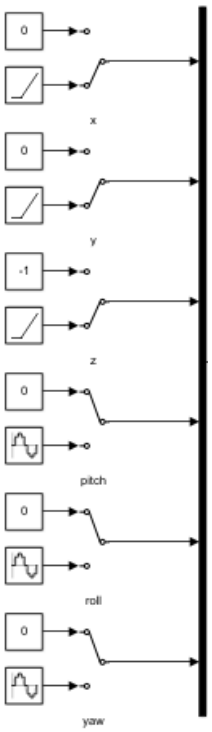

Ilustración 39: Señales de desplazamientos y variación de ángulos en Simulink

4.2. Archivo de programación Python que establece las funciones de la comunicación

Se crea un archivo para programar en Python las funciones que permitan la comunicación entre Simulink y AirSim. Estas funciones podrán ser llamadas más adelante desde el código de Simulink, como se mostrará en la sección 4.3. Este archivo puede programarse en Visual Studio, con extensión de archivo de Python (*.py*), y debe guardarse en la misma carpeta del Simulink del simulador, para que éste pueda acceder fácilmente a él. El nombre elegido para el archivo es *drone.py*.

Para empezar, se importa *AirSim* para poder crear objetos de la clase *airsim* y utilizar sus funciones. También se importa la librería *NumPy*, que sirve para poder crear vectores y matrices, así como para utilizar funciones matemáticas avanzadas. Importar *NumPy* como *np* tiene la utilidad de que se puede escribir *np* en lugar de *numpy* cuando queramos llamar a las funciones de esta librería.

Código 4: drone.py, Importaciones iniciales

```
import airsim
import numpy as np
```

Se comienza programando la función inicializadora *setup*, que crea un objeto de la clase *airsim*, objeto que engloba en sí la simulación del vuelo del dron de AirSim. Al objeto se le ha dado el nombre *client*, y será devuelto por la función. La función llama a tres funciones de la clase *airsim* para modificar el objeto *client* antes de devolverlo. Estas tres funciones utilizadas son las siguientes:

- *confirmConnection*. Para comprobar la conexión cada segundo y reportarlo a la consola, pudiendo ver el usuario de esta manera el estado de la conexión.
- *enableApiControl*. Enviándole como parámetro *True*, activa el control Application Programming Interface (API), para establecer la comunicación del código de Python con AirSim.
- *armDisarm*. Enviándole como parámetro *True*, activa el dron en la simulación.

Código 5: drone.py, Función inicializadora

```
def setup():

    client = airsim.MultirotorClient()

    client.confirmConnection()
    client.enableApiControl(True)
    client.armDisarm(True)

    return client
```

Después, se realiza la función encargada de establecer la nueva posición y orientación del dron. La función se inicializa con dos parámetros, el primero será un objeto *client* de la clase *airsim*, mientras el segundo será un vector *pose* de seis componentes que incluye las tres coordenadas nuevas de posición y los tres ángulos nuevos de orientación.

Se guardan las seis componentes del vector en seis variables, y se utiliza la función *simGetVehiclePose* de la clase *airsim* para obtener la posición y orientación actuales del vehículo *client* tal y como se encuentran en la programación del simulador, y guardarlas en *pose*. Se sobreescriben las posiciones y orientaciones actuales de *pose* utilizando los valores de posición y orientación nuevos guardados en las variables. Al sobreescribir los ángulos hay que pasarlos de formato Euler a formato quaternion, que es el que utiliza AirSim internamente. Por último, se envía la información de posición y orientación que han sido sobrescritas, utilizando la función *simSetVehiclePose* de la clase *airsim* sobre *client*.

Código 6: drone.py, Función que establece la posición y orientación del dron

```
def set_pose(client, pose):

    x, y, z, pitch, roll, yaw = pose

    pose = client.simGetVehiclePose()

    # Position
```

```
        pose.position.x_val = x
        pose.position.y_val = y
        pose.position.z_val = z

        # Orientation
        pose.orientation = airsim.to_quaternion(pitch, roll,
    yaw)

        client.simSetVehiclePose(pose, True)
```

La última función, servirá para obtener a cada momento la posición y orientación del dron en la simulación. Como parámetro tendrá un objeto *client* de la clase *airsim*. La función se encargará de guardar en dos variables la posición y la orientación del dron tal y como se encuentran en la programación del simulador. Para ello se utiliza la función s*imGetGroundTruthKinematics* de la clase *airsim* sobre *client*. Los ángulos se transforman a formato Euler ya que AirSim los envía en formato quaternion. Después se guardan los seis valores de las tres coordenadas de posición y los tres ángulos de Euler en seis variables, que la función devolverá en un vector.

Código 7: drone.py, Función que obtiene la posición y orientación del dron

```
def get_pose(client):

    position =
client.simGetGroundTruthKinematics().position
    orientation =
airsim.to_eularian_angles(client.simGetGroundTruthKinematic
s().orientation)

    x = position.x_val
    y = position.y_val
    z = position.z_val

    pitch = orientation[0]
    roll = orientation[1]
    yaw = orientation[2]

    return np.array([x, y, z, pitch, roll, yaw])
```

4.3. Utilización las funciones de comunicación desde el modelo de Simulink

Para que Matlab pueda comunicarse con Python, es necesario buscar desde Matlab la ubicación o path del ejecutable de Python que se tiene instalado. Después se especifica este path para que sea el que utilice al usar Python. Por último, se puede

ejecutar *drone.py* para asegurar que la comunicación funcionará correctamente y no hay errores. El proceso se muestra en el Código 8Código 1.

Código 8: Especificación del path de Python desde Matlab

```
>> pyversion
      version: '3.9'
   executable: 'C:\Users\danie\Documents\Programas\Anaconda\python.exe'
      library: 'C:\Users\danie\Documents\Programas\Anaconda\python39.dll'
         home: 'C:\Users\danie\Documents\Programas\Anaconda'
     isloaded: 0
>> pyversion('C:\Users\danie\Documents\Programas\Anaconda\python.exe')
>> pyrunfile('drone.py')
```

Se crea una función en Matlab llamada *convert_pose* que se encarga de llamar a la función *get_pose* de *drone.py*, función que ha sido explicada en la sección 4.2 y se puede ver en el Código 7. La finalidad es devolver la posición y orientación del dron del simulador con una función de Matlab que devuelva valores que el programa reconozca fácilmente, y se programa como se expone en el Código 9. El archivo de esta función deberá guardarse en la carpeta del Simulink del simulador.

Código 9: convert_pose.m, Función de Matlab que obtiene la posición del dron del simulador

```
function matrix = convert_pose(client)

    matrix = py.drone.get_pose(client);

end
```

Tras ello, se construye el diagrama de Simulink a partir de las señales de la Ilustración 39. Las seis señales se introducen en un vector llamado *pose*, que será el parámetro de entrada del bloque función *pose_setting*. Este bloque, que será desarrollado más adelante, será el encargado de enviar los datos que se encuentran en el vector a AirSim, utilizando las funciones que hemos construido a lo largo de este capítulo. De esta manera, el dron de AirSim cambiará su posición y orientación a las que han llegado al bloque de Simulink.

Además, el bloque *pose_setting* tendrá también una salida llamada *pose*, que tendrá la posición y orientación del dron recibidas desde AirSim. De esta manera,

mostrando los valores en scopes o guardándolos en variables, se podrá comprobar si la posición y orientación que tiene el dron en cada momento es acorde a la que debería adoptar con la información que se le está enviando.

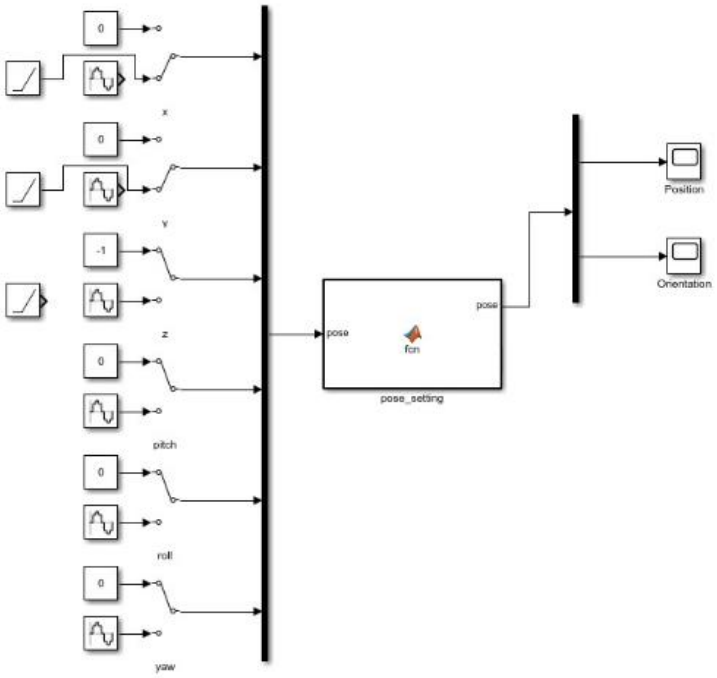

Ilustración 40: Diagrama de Simulink de la comunicación con AirSim

La programación del bloque *pose_setting*, que tiene de parámetro entrada el vector *pose*, puede verse al completo en el Código 10. El código comienza con la declaración de las funciones que vamos a utilizar y han sido desarrolladas previamente. Se utilizarán todas las funciones de *drone.py* explicadas en la sección 4.2, y también *convert_pose.m*, que se puede ver en el Código 9.

Después inicializamos el simulador. Para ello, se crea una variable persistente, es decir, que no varíe entre llamadas a la función, de nombre *client*. En esta variable guardaremos el resultado de la función *setup* de *drone.py*, que es el simulador inicializado tal y como mostraba el Código 5. Esto ocurrirá sólo si la variable *client* está vacía, es decir, la simulación acaba de comenzar y la variable aún no tiene nada guardado en ella.

Luego se llama a la función *set_pose* de *drone.py*, con parámetros la variable persistente *client* y el parámetro *pose* que contiene la posición y ángulos deseados. El dron del simulador entonces cambiará su posición y orientación, según la función que fue explicada en el Código 6.

Por último, se crea un nuevo vector *pose* de seis componentes, que será la salida del bloque, y se guardan en él la posición y orientación del dron obtenidas desde AirSim.

Para ello se utiliza la función *convert_pose*, con parámetro la variable persistente *client*. Según lo explicado en el Código 9, esta función llama a su vez a la función *get_pose* de *drone.py*, detallada en el Código 7.

Código 10: pose_setting, Bloque de comunicación de Simulink con AirSim

```
function [pose] = fcn(pose)

    coder.extrinsic('py.drone.setup')
    coder.extrinsic('py.drone.set_pose')
    coder.extrinsic('py.drone.get_pose')
    coder.extrinsic('convert_pose')

    % Initialization
    persistent client;
    if isempty(client)
        client = py.drone.setup();
    end

    % Set Pose
    py.drone.set_pose(client, pose);

    % Get Pose
    pose = zeros([6, 1]);
    pose = double(convert_pose(client));

end
```

4.4. Colisiones: Final de simulación

Se presenta un problema en la simulación. Las colisiones del dron con otros objetos en el entorno virtual son ignoradas mientras la simulación del vuelo, y el dron los atraviesa como si no estuviesen ahí. Esto es debido a que, aunque UnReal sí detecte las colisiones, el modelo de Simulink le está enviando una información a través de Python que obliga al dron a desplazarse a la posición hacia la que se dirige, obviando cualquier otra información que UnReal contenga.

La solución que se ha optado para este problema ha sido aprovechar que las colisiones sí son detectadas por AirSim, aunque sean ignoradas debido a la comunicación con Simulink. En el momento en el que una colisión sea detectada en AirSim, se enviará un booleano afirmativo al Simulink, de tal forma que la simulación se detenga. Una vez la simulación se ha detenido, como la comunicación entre Simulink y AirSim ha terminado, el dron caerá hasta el suelo colisionando con el resto de los objetos. Esto es debido a que en ese momento solo continuará vigente la parte de la simulación de UnReal y no la de Simulink.

Para conseguir el booleano afirmativo en caso de que se produzca una colisión, se crea una nueva función en *drone.py* de nombre *collision_info*, y que utiliza como parámetro un objeto *client* de la clase *airsim*. Esta función pregunta si ha habido una colisión en la simulación mediante la función *simGetCollisionInfo* de la clase *airsim* sobre *client*. En caso afirmativo, una variable llamada *stop* se pone a uno, siendo la variable que la función devolverá.

Código 11: drone.py, Función que obtiene si el dron ha colisionado

```
def collision_info(client):

    if client.simGetCollisionInfo().has_collided:
        stop=1
    else: stop =0

    return np.array([stop])
```

De la misma manera que en *convert_pose.m*, expuesta en el Código 9, se crea una nueva función en Matlab llamada *convert_collision.m*, para que devuelva el booleano en un formato que el programa reconozca fácilmente, y se programa como se expone en el Código 12. El archivo de esta función deberá guardarse en la carpeta del Simulink del simulador.

Código 12: convert_collision.m, Función de Matlab que obtiene si el dron ha colisionado

```
function matrix = convert_collision(client)

    matrix = py.drone.collision_info(client);

end
```

Se añade al bloque *pose_setting*, mostrado anteriormente en el Código 10, la declaración de la función de Matlab *convert_collision.m*, y de la función *collision_info* de *drone.py*. Después se utiliza una variable *stop_simulation*, a la que se le dará el valor que devuelva *convert_collision.m* con parámetro *client*. De esta manera, la simulación tendrá que detenerse en caso de que *stop_simulation* tenga valor uno, y continuar con normalidad en caso de que su valor sea cero. El código completo de *pose_setting* se expone en el Código 13.

Código 13: pose_setting, con variable de finalizar la simulación si hay una colisión

```
function [pose,stop_simulation]  = fcn(pose)
```

```
    coder.extrinsic('py.drone.setup')
    coder.extrinsic('py.drone.set_pose')
    coder.extrinsic('py.drone.get_pose')
    coder.extrinsic('py.drone.collision_info')
    coder.extrinsic('convert_pose')
    coder.extrinsic('convert_collision')

    % Initialization
    persistent client;
    if isempty(client)
        client = py.drone.setup();
    end

    % Set Pose
    py.drone.set_pose(client, pose);

    % Get Pose
    pose = zeros([6, 1]);
    pose = double(convert_pose(client));

    % Stop Simulation when there is a Collision
    stop_simulation = zeros([1, 1]);
    stop_simulation =
           double(convert_collision(client));

end
```

La variable *stop_simulation* será una salida del bloque *pose_setting*. Esta salida estará conectada a un bloque *STOP*. El funcionamiento de este bloque es sencillo, la simulación se detendrá cuando reciba un uno. De esta manera, ya se ha completado la programación del Simulink, pudiendo verse el diagrama completo en la Ilustración 41.

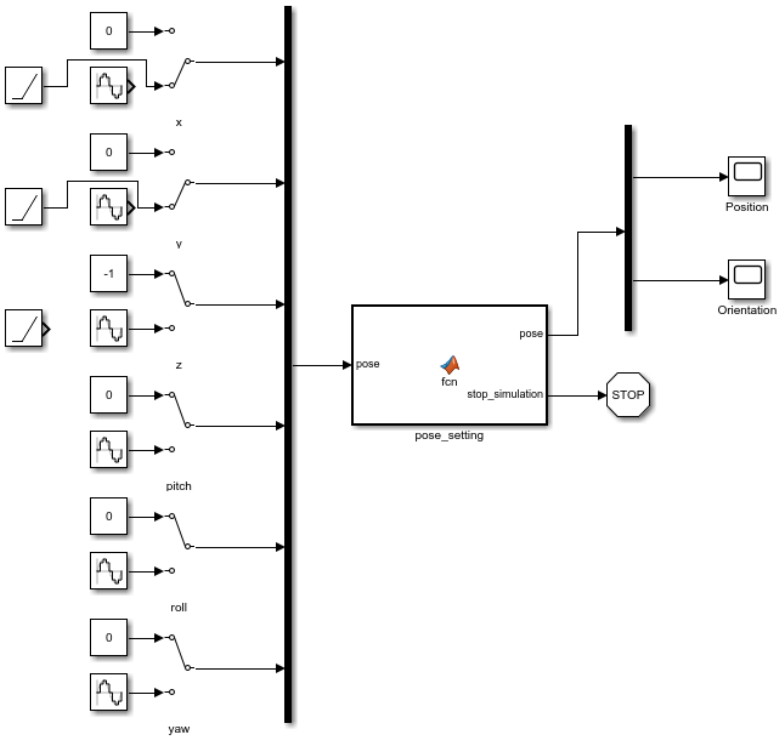

Ilustración 41: Diagrama de Simulink de la comunicación con AirSim completado con colisiones

4.5. Colisiones: Delimitar zona de vuelo

Como el entorno del laboratorio tiene una gran cantidad de objetos, se ha decidido delimitar la zona de simulación de vuelo a la zona en la que el dron se va a volar en la realidad. De esta manera, no tendrán que añadirse propiedades de colisiones a todos los objetos del laboratorio. Esta zona se encuentra entre tres paredes y una red, como muestra la Ilustración 42.

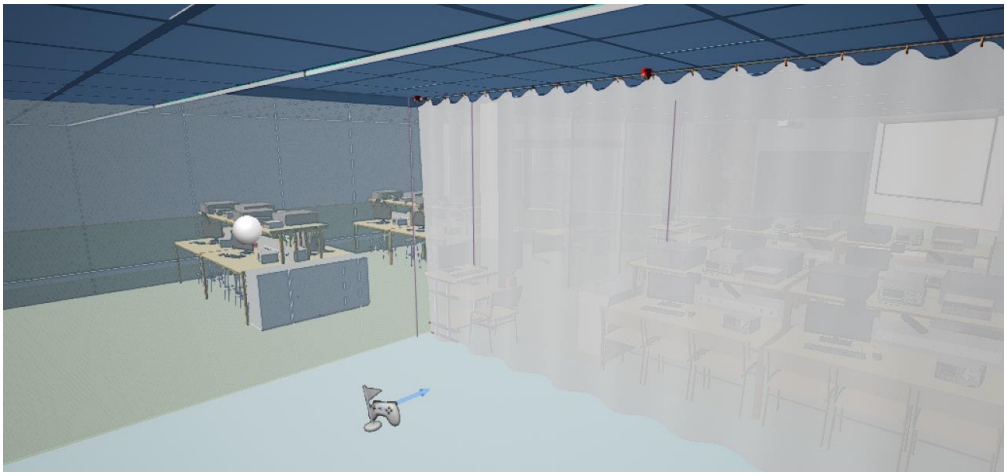

Ilustración 42: Zona delimitada del laboratorio para vuelo del dron

Para ello, seguimos el mismo proceso al explicado en la sección 3.2, para crear una malla de colisiones entorno a la red. Se añade al modelo estático de la red una colisión simple con forma de caja, *Box Simplified Collision*, que se selecciona en el desplegable *Colisión* en la ventana de propiedades del modelo estático del ensamblaje. Esta vez también se elige la opción *Usar la colisión compleja como colisión simple*.

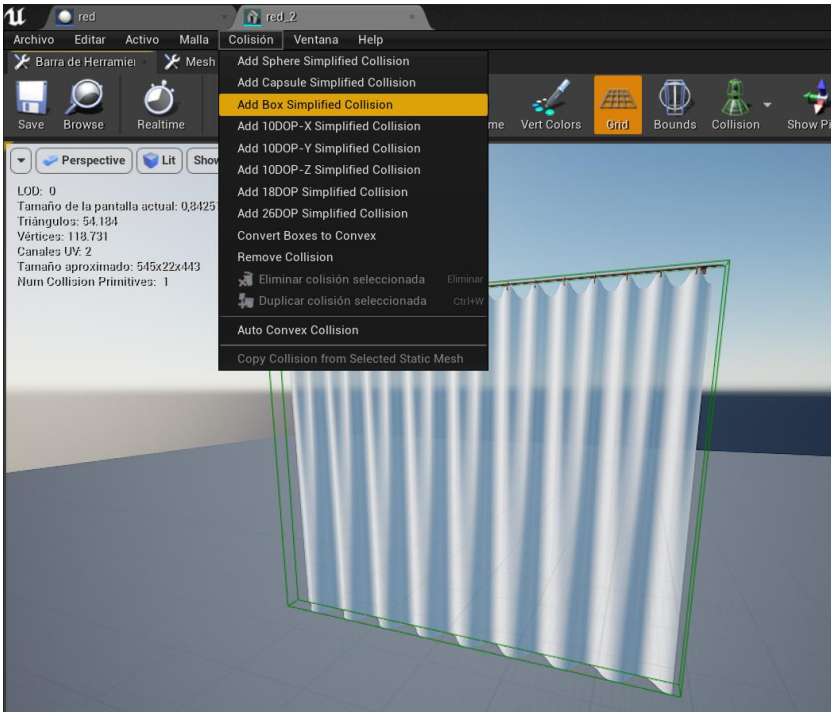

Ilustración 43: Colisión simple con forma de caja en la red

Se crea un nuevo blueprint al que se le llama *red* dentro de una carpeta nueva del proyecto que se puede llamar *MyBlueprints*. A este blueprint le añadimos la malla estática de la red como muestra la Ilustración 44.

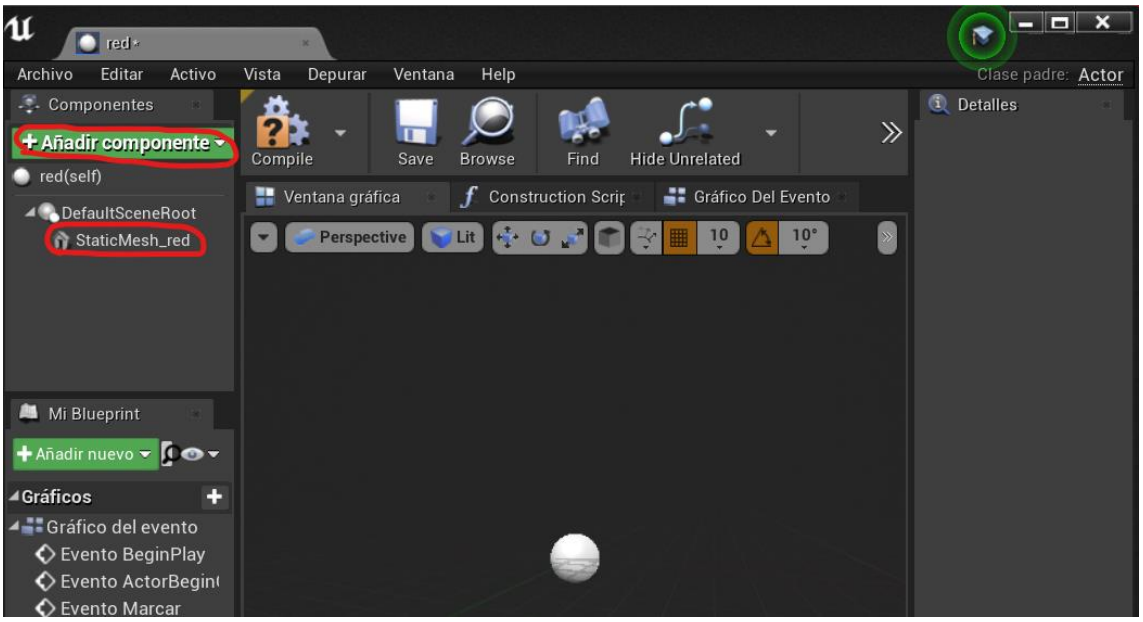

Ilustración 44: Añadir malla estática de la red al nuevo blueprint

En el *Construction Scrip*t de *red*, se crea una nueva variable local eligiendo en tipo *Modelo estático*, *Object Reference*. Luego, en la pestaña *Valor predeterminado* de la nueva variable local, se selecciona el modelo estático de la red.

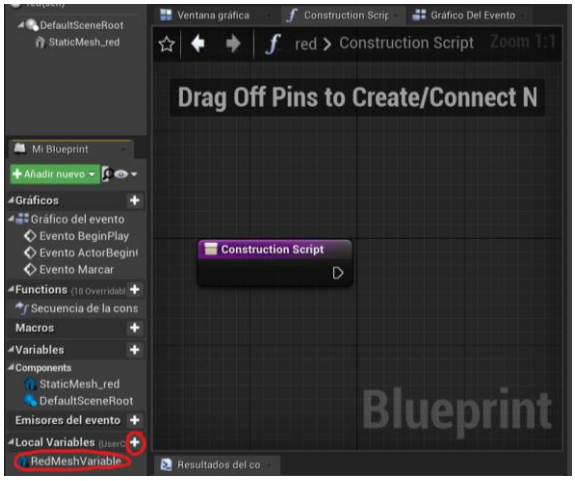

Ilustración 45: Creación de variable local en blueprint red

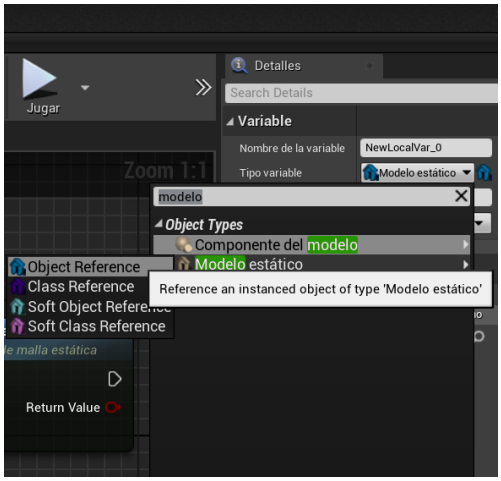

Ilustración 46: Tipo de variable local en blueprint red

Ilustración 47: Modelo estático de la red en el blueprint de la red

Se construye el esquema del blueprint *red* con un bloque de *Ajustar malla estática*, que sirve para unir el modelo estático de la red con la variable local creada que indica que es un objeto.

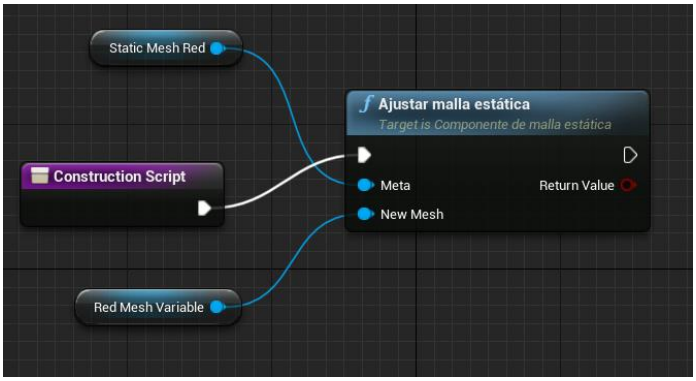

Ilustración 48: Esquema del blueprint de la red

Se añade el elemento a la escena, añadiendo el Blueprint creado, nunca el objeto original. El objeto es colocado en la posición correcta. Para elegir las propiedades físicas, se busca el blueprint en el *World Outliner* y se pulsa en *Editar*. Pulsando sobre la variable, pueden cambiarse las propiedades físicas en el desplegable *Physics*, como muestra la Ilustración 49.

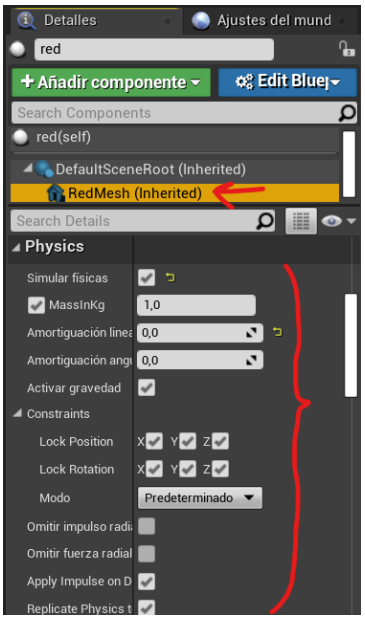

Ilustración 49: Propiedades físicas de la red

Las colisiones con las paredes del entorno virtual fallan a veces. Para asegurar que las colisiones ocurren correctamente, se realiza el mismo proceso que con la red con cinco objetos prismáticos invisibles que ocupan el lugar del techo, suelo, y paredes de la zona de vuelo. De esta manera, aunque no sean visibles los objetos, el dron detectará la colisión con ellos.

5. Modelo dinámico de vuelo del dron

En este capítulo se muestran las partes más significantes del modelo dinámico del vuelo del dron desarrollado en Matlab y Simulink. En la sección 4.1 se explica el funcionamiento de los códigos de Matlab indispensables para la puesta en marcha del simulador. En la sección 4.2 se explica cómo realizar un vuelo manual mediante el radiotransmisor, incluyendo cómo ajustar los recorridos de los joysticks.

5.1. Matlab y Simulink con el modelo dinámico de vuelo

El modelo dinámico de vuelo del dron ha sido desarrollado previamente y consta de una carpeta *UAV_PROJECT*, que contiene varias subcarpetas con distintos códigos desarrollados en Matlab y una con los esquemas de Simulink que constituyen simulador. El modelo ha sido facilitado por Juan Luis Zamora Macho, codirector de este proyecto.

Dentro de la carpeta *CONFIGURATION*, hay un archivo de Matlab llamado *CONFIG_UAV.m*. Este código es el que será ejecutado para establecer los parámetros y condiciones del modelo dinámico que se van a utilizar cuando se ejecute el simulador de Simulink. En el apartado *GENERAL CONFIGURATION*, mostrado en el Código 14, se debe comprobar que la variable *RUN_MODE* tiene asignada valor cero. Esto indica que la simulación ocurrirá en tiempo real, aunque no es tiempo real exacto ya que hay una pequeña desviación.

Código 14: CONFIG_UAV.m, GENERAL CONFIGURATION

```
%% GENERAL CONFIGURATION
%-------------------------------------------------------------
% IP for SSH and TCP/IP communications
UAV_IP = '192.168.1.20';
% RUN_MODE DEFINITION
% / 0. REAL-TIME SIMULATION / 1. FAST SIMULATION / 2. TEST
VERIFICATION
% / 3. IMPLEMETATION / 4. ALREADY DEPLOYED / 5. RPI
SIMULATION
RUN_MODE = 0;
```

```
% TEST FOR MODEL IDENTIFICATION
IDENT_TEST = false;
% Sampling time (s)
SAMPLING_TIME = 10e-3;
```

Dentro de la carpeta *SOFTWARE_COMPONENTS*, hay una carpeta llamada *CONTROL*. En ella se encuentra *CONFIG_CONTROL.m*, que permite elegir entre distintos modos de vuelo y modificar ciertos parámetros.

En el apartado *CONTROL MODE*, mostrado en el Código 15, se encuentra la línea *CONTROL.STATE.CONTROL_MODE=uint8(0)*. Esta línea permite elegir el modo de vuelo del dron:

- 0 indica vuelo manual con radiotransmisor, en el que hay que controlar manualmente la altura para mantenerla constante.
- 1 indica control de altitud, en el que la altura está controlada en lazo cerrado, por lo que el dron permanece a una altura determinada constantemente durante el vuelo sin necesidad de controlarla manualmente.
- 2 indica control de navegación, que se basa en un control de las velocidades en los ejes x e y del sistema de referencia de tierra.

Independientemente del modo que se utilice, el modelo dinámico tendrá como salidas los desplazamientos y ángulos necesarios para enviar a AirSim a través del Simulink desarrollado en la sección 4.3.

Código 15: CONFIG_CONTROL.m, CONTROL MODE

```
%% CONTROL MODE
%--------------------------------------------------------------
% CONTROL MODE
% / 0. RC FLIGHT / 1. ALTITUDE CONTROL / 2. NAVIGATION
CONTROL
CONTROL.STATE.CONTROL_MODE = uint8(0);
```

También en *CONFIG_CONTROL.m*, en el apartado *REFERENCE DEFINITION* hay un subapartado llamado *REFERENCE SOURCE*, en el que se encuentra la línea *CONTROL.STATE.ATT_TARGET_SOURCE = uint8(2)*. Aquí se puede especificar de dónde obtiene el simulador las referencias que tiene que seguir para el vuelo:

- 0 indica vuelo local. Este vuelo sigue referencias que hay que especificar en el código, en el mismo *REFERENCE DEFINITION*.
- 1 indica PC, referencias que no son de interés en este proyecto.
- 2 indica referencias de vuelo controladas por el radiotransmisor.

Código 16: CONFIG_CONTROL.m, REFERENCE DEFINITION, REFERENCE SOURCE

```
% REFERENCE SOURCE
% / 0. LOCAL / 1. PC / 2. RC
CONTROL.STATE.ATT_TARGET_SOURCE = uint8(2);
% / 0. LOCAL / 1. PC / 2. RC / 3. MISSION
CONTROL.STATE.NAV_TARGET_SOURCE = uint8(3);
```

Dentro de la carpeta *SIMULINK* se encuentra el modelo de simulink, llamado *UAV_CONTROL_SYSTEM*. El simulink muestra varios bloques como se muestra en la Ilustración 50.

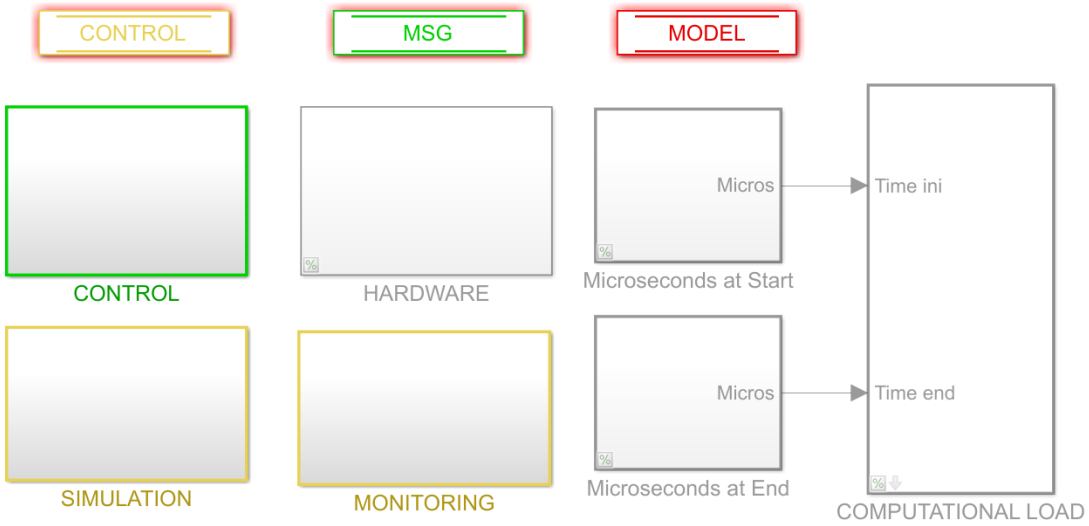

Ilustración 50: UAV_CONTROL_SYSTEM

En el bloque *SIMULATION* se muestra en la Ilustración 51. En él se encuentra el hardware real del propio dron representado matemáticamente. El bloque *RC TRANSMITER*, mostrado en la Ilustración 52, es el que interactúa con la emisora, y muestra a la derecha las lecturas de los canales; estos canales se alimentan al sistema de control, que se encarga de cambiar las coordenadas y ángulos según los cambios de referencia de la emisora.

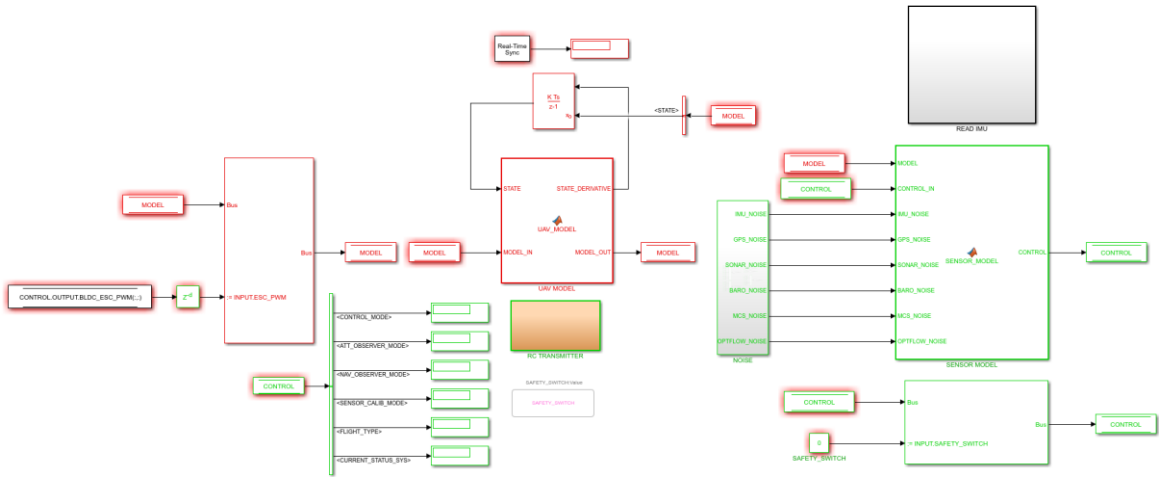

Ilustración 51: UAV_CONTROL_SYSTEM, SIMULATION

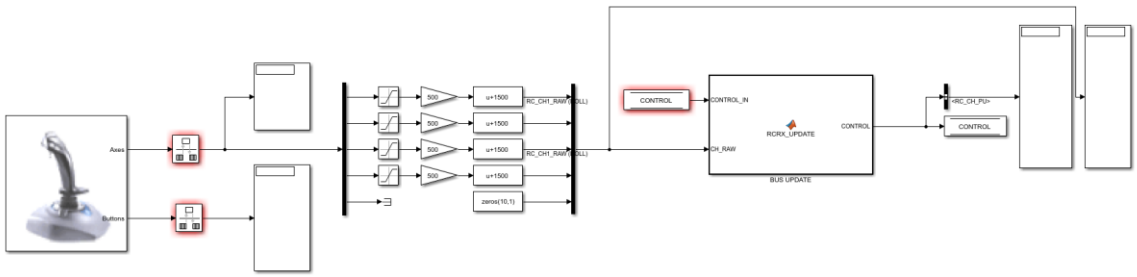

Ilustración 52: UAV_CONTROL_SYSTEM, SIMULATION, RC TRANSMITER

En el bloque *CONTROL* se encuentran todos los lazos de control para que las variables principales estén controladas y sigan a sus valores de referencia; así como la máquina de estados que determina por qué fases hay que pasar desde que se enciende el equipo hasta que se apaga.

Dentro del bloque *MONITORING*, hay otro bloque llamado *VR SIMULATOR*. En este bloque se recibe la información de las tres coordenadas de deslazamiento en los ejes x, y, z; las tres velocidades en esos ejes; los tres ángulos de Euler; las tres velocidades angulares; y por último los cuatro PWM que determinan las velocidades de los motores. Aquí es donde debe añadirse el simulink desarrollado en la sección 4.3 y expuesto en la Ilustración 41, utilizando como las seis entradas las tres posiciones y los tres ángulos de Euler que son salidas del modelo dinámico. El acoplamiento de ambos códigos se realiza como muestra la Ilustración 53.

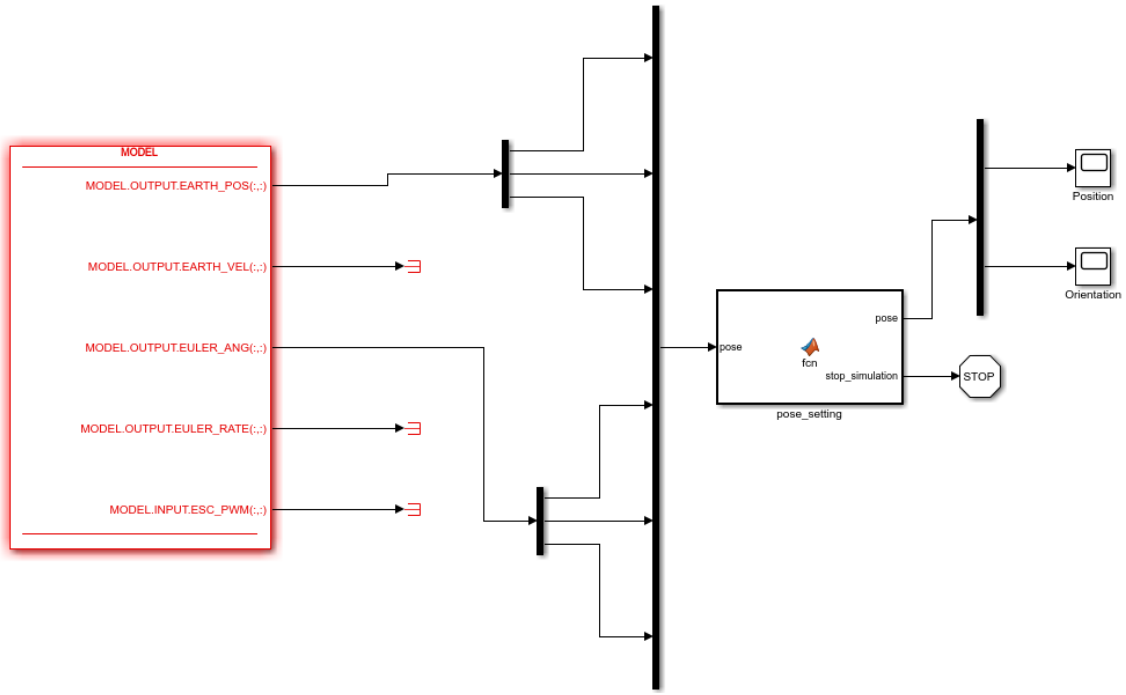

Ilustración 53: UAV_CONTROL_SYSTEM, MONITORING, VR SIMULATOR

5.2. Vuelo manual mediante radiotransmisor

El control que se elige para controlar la simulación con el mando será el de vuelo manual con radiotransmisor, indicándose con un 0 en el Código 15. Para que el simulador tome como referencias las de la emisora, se indica con un 2 en el Código 16. Se ejecuta *CONFIG_UAV.m*.

Para realizar un vuelo manual mediante el radiotransmisor, primero debe conectarse el mando al ordenador y calibrar la información recibida del movimiento de los joysticks correctamente. Para ello, se ejecuta el simulink y se observa la información recibida por los joysticks en el bloque *RC TRANSMITER* de la Ilustración 52. A la hora de ejecutar el simulink, es importante tener activo AirSim en el proyecto de UnReal, de otra manera la comunicación entre ambos programas no podrá establecerse, y el simulink se pausará por este error.

La información recibida por los joysticks son los cuatro datos mostrados en la Ilustración 54, y están ordenados de canal 1 a canal 4. Se mira qué canal corresponde a cada movimiento del dron:

- Canal 1: Roll o alabeo, que se controla mediante el joystick derecho con un recorrido de izquierda a derecha.
- Canal 2: Pitch o cabeceo, que se controla mediante el joystick derecho con un recorrido de abajo a arriba.

- Canal 3: Throttle o cambio de altura, que se controla mediante el joystick izquierdo con un recorrido de abajo a arriba.
- Canal 4: Rudder o yaw o guiñada, que se controla mediante el joystick izquierdo con un recorrido de izquierda a derecha.

En la carpeta *HARDWARE_COMPONENTS*, se encuentra el archivo *CONFIG_RCRX.m*. En este archivo se puede definir a qué movimiento corresponde la información de cada canal que acabamos de comprobar. Para ello escribimos los números de los canales en cada movimiento como se muestra en el Código 17.

Código 17: CONFIG_RCRX.m, Definición de canales

```
% Channel definition : [ROLL PITCH RUDDER THROTTLE]
RCRX.CH_DEF = [1 2 4 3];
```

En ese mismo archivo deben calibrarse los recorridos exactos de los joysticks. Para ello se deben ajustar los valores de los máximos y mínimos, así como el punto de reposo de cada uno de los joysticks. Los valores máximos, mínimos y de reposo de los canales se observan en *RC TRANSMITER*, ya expuesto en la Ilustración 52. La Ilustración 54 señala el lugar exacto donde se encuentran estos valores.

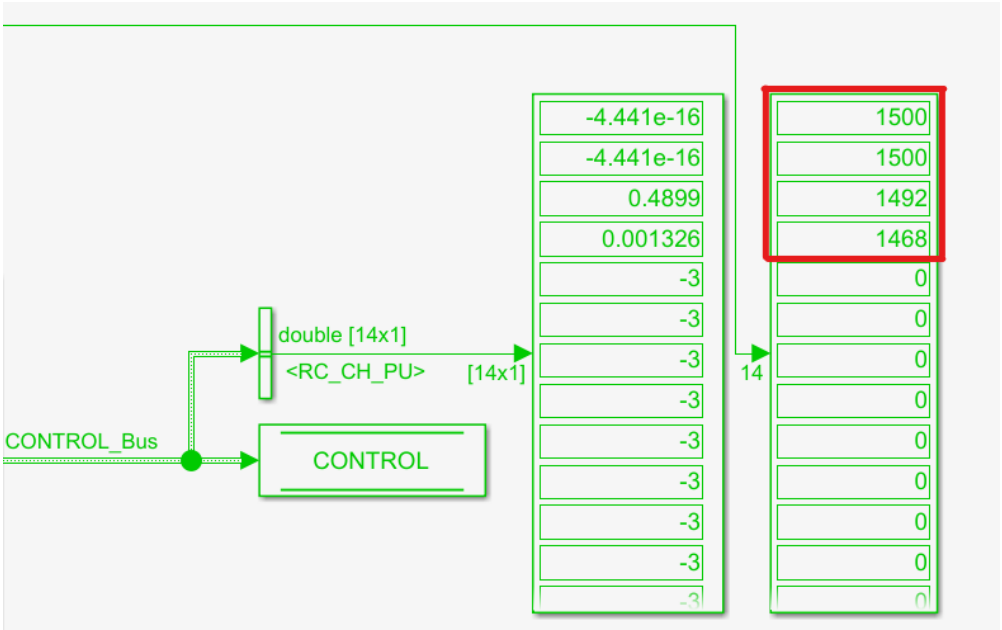

Ilustración 54: UAV_CONTROL_SYSTEM, SIMULATION, RC TRANSMITER información de recorrido de los joysticks

Se apuntan los valores que se reciben en los canales en los puntos de reposo, máximos, y mínimos de cada uno de los joysticks del radiotransmisor. Después, se establecen en el código de *CONFIG_RCRX.m*, tal y como muestra el Código 18.

	Mínimo	Reposo	Máximo
Throttle (Altura)	1110	1488	1877
Roll (Giro)	1098	1468	1849
Pitch (Cabeceo)	1091	1500	1893
Yaw (Alabeo)	1130	1500	1885

Tabla 2: Tabla de calibración de recorrido de joysticks

Código 18: CONFIG_RCRX.m, Ajuste de los recorridos de los joysticks

```
%% RCRX PARAMETER STRUCT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Minimum value for channel range
% CH_MIN = [1000 1000 1000 1000 ...
%           1000 1000 1000 1000 ...
%           1000 1000 1000 1000 ...
%           1000 1000];
CH_MIN = [1126 1094 1110 1102 ...
          1000 1000 1000 1000 ...
          1000 1000 1000 1000 ...
          1000 1000];
% Maximum for channel range
% CH_MAX = [2000 2000 2000 2000 ...
%           2000 2000 2000 2000 ...
%           2000 2000 2000 2000 ...
%           2000 2000];
CH_MAX = [1881 1893 1885 1845 ...
          2000 2000 2000 2000 ...
          2000 2000 2000 2000 ...
          2000 2000];
% Trim or central point
% RCRX.CH_TRIM = [1500 1500 1500 1500 ...
%                 1500 1500 1500 1500 ...
%                 1500 1500 1500 1500 ...
%                 1500 1500];
RCRX.CH_TRIM = [1500 1500 1500 1468 ...
                1500 1500 1500 1500 ...
                1500 1500 1500 1500 ...
                1500 1500];
% Channel definition: [ROLL PITCH RUDDER THROTTLE]
RCRX.CH_DEF = [1 2 4 3];
```

Tras calibrar la emisora, el simulador ya está listo para utilizarse siempre y cuando UnReal tenga el proyecto con AirSim activo. Una vez ejecutado el simulink comienza la simulación, y se puede ver cuánto se desvía el tiempo de la simulación del tiempo real en la parte superior del bloque *SIMULATION*, como se muestra en la Ilustración 55.

Ilustración 55: UAV_CONTROL_SYSTEM, SIMULATION, Diferencia entre el tiempo real y el de simulación

La simulación durará indefinidamente a no ser que la paremos voluntariamente o se pare porque haya una colisión. El simulador pasa por diversos estados que se muestran en el bloque *SIMULATION*, en *CURRENT_STATUS_SYS*, indicado en la Ilustración 56.

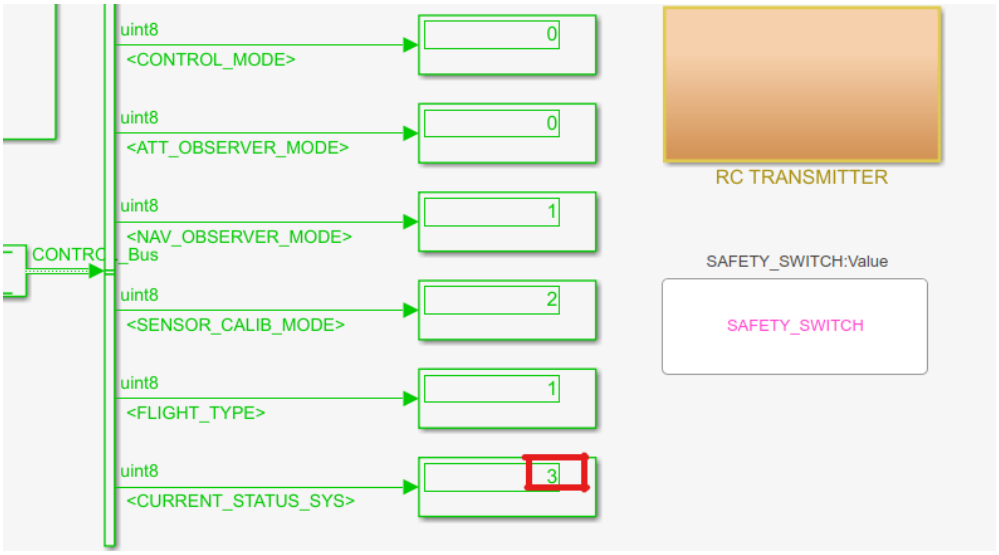

Ilustración 56: UAV_CONTROL_SYSTEM, SIMULATION, Estado en el que se encuentra la simulación

La simulación comienza en el estado 0 durante los tres segundos iniciales. Después pasa al estado 1 durante diez segundos, el estado de calibración de sensores. Una vez pasado este tiempo, entra en el estado 2, estado preparado.

En la Ilustración 56 también puede verse el botón de seguridad, SAFETY_SWITCH. Este botón deberá pulsarse para pasar del estado 2 al estado 3, estado de preparación para vuelo.

Moviendo el joystick izquierdo abajo y a la derecha durante dos segundos, se pasa al estado 5, armando motores. Directamente pasa al estado 4, en el que ya se puede volar la simulación.

Existe la posibilidad de empezar en modo manual y luego pasar a control de altura, o comenzar en control de altura directamente. Para el cambio hay un conmutador en el propio mando, una de las palancas de encima de los joysticks, que corresponde al canal 5. De la misma manera, se puede empezar en modo manual y pasar a modo navegación, o comenzar en modo navegación directamente, mismo interruptor que para el anterior.

Otro conmutador, el que corresponde al canal 6, es para detener los motores. Una vez aterrizado, se detienen los motores para parar así el modo de vuelo, que vuelve

al estado 2. Sin embargo, este conmutador no será útil en la simulación, ya que al aterrizar el dron en el suelo se detectará una colisión que detendrá la simulación, por lo que habrá que ejecutar de nuevo el simulador para volver a volar.

5.3. Vuelo automático: misión de ruta de puntos

Para realizar un vuelo automático, se tiene que ajustar el modo de control de *CONFIG_CONTROL.m*, mostrado en el Código 15. En la línea de código *CONTROL.STATE.CONTROL_MODE=uint8(2)* se le pasa de parámetro un dos, que indica modo de vuelo de control de navegación.

También, en el apartado *REFERENCE DEFINITION* se encuentra el subapartado *REFERENCE SOURCE*. Pasamos como parámetro a la línea *CONTROL.STATE.ATT_TARGET_SOURCE = uint8(0)* un cero. Esto significa que las referencias de vuelo son las especificadas en el apartado *MISSION WAYPOINTS*.

El vuelo funciona mediante diversos puntos a los que se va moviendo el dron durante su vuelo autónomo. Estos puntos contienen el momento del tiempo de simulación en el que el dron se desplaza a este punto, las coordenadas x, y, y z del punto, y la orientación del ángulo de guiñada del dron en grados. La programación de una misión de diversos puntos se muestra en el Código 19.

Esta misión consiste en que el dron sube un metro, y avanza desde del centro de un cuadrado imaginario de lado dos metros al punto central de uno de sus lados. Después, recorre todo el perímetro del cuadrado, y por último vuelve al punto central del cuadrado.

Código 19: CONFIG_CONTROL.m, REFERENCE DEFINITION, Puntos de misión de vuelo autónomo

```
% MISSION WAYPOINTS [wait time(s) X(m) Y(m) Z(m) YAW(deg)]
MISSION_WAYPOINTS = [
    0      0       0       0       0
    0      0       0      -1       0
    1      1       0      -1       0
    2      1      -1      -1     -90
    3     -1      -1      -1    -180
    4     -1       1      -1    -270
    5      1       1      -1    -360
    6      1       0      -1    -360
    7      0       0      -1    -360
    ];
```

6. Resultados

En este capítulo se muestra el resultado de la simulación, así como una solución para mejorar la eficacia de esta.

La simulación en vuelo autónomo es un éxito, siendo correcta la comunicación entre Simulink y AirSim. El dron se muestra por pantalla y se mueve acorde a las instrucciones enviadas, deteniéndose la simulación cuando el dron colisiona con una de las paredes de la zona de vuelo.

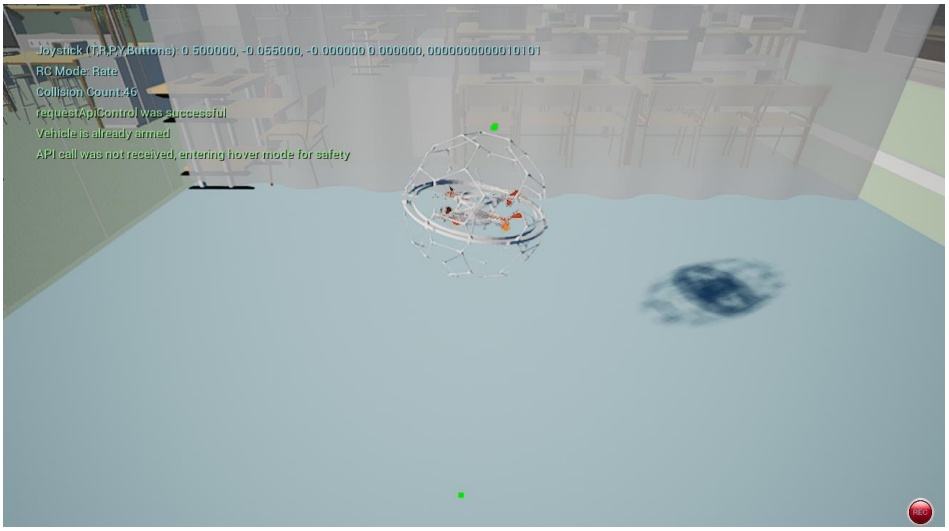

Ilustración 57: Visualización de la simulación

Sin embargo, el control del dron en modo manual con el radio transmisor es prácticamente imposible de manejar, ya que los movimientos de los joysticks del mando generan que el dron se mueva de una forma demasiado rápida y brusca. El entorno que se había determinado es demasiado pequeño para volar el dron, y se debe buscar una solución a esto.

Se opta por suavizar las curvas de los joysticks, que van aproximadamente de 1000 a 2000 con punto intermedio 1500, para que el movimiento del dron no sea tan sensible al cambiar los joysticks en las zonas que no interesen para volar el dron cómodamente.

Suavizaremos las curvas en cada canal en el Simulink, en *SIMULATION*, *RC TRANSMITTER*. Para suavizar la curva de un canal se utiliza un bloque denominado *1-D Lookup Table*. Como se ha especificado en la sección 5.2, el canal del throttle es el canal 3, mientras los canales de los giros de guiñada, cabeceo y alabeo son el 4, 2, y 1, respectivamente.

El throttle crece desde 1000 hasta 2000. Comienza subiendo sin que el dron despegue, hasta que se alcanza la fuerza de sustentación, momento en el cual empieza a subir. El problema es que tarda mucho recorrido de joystick en despegar, y una vez lo hace el movimiento hacia arriba es tan rápido que llega inmediatamente hasta el techo, ya que la pendiente es muy brusca en este tramo.

La nueva recta del throttle debe tener mucha pendiente al principio del recorrido del joystick, para que la fuerza de sustentación se alcance en cuanto se mueva un poco el joystick. Después, la pendiente debe ser lo más suave posible, para que el dron no suba tan bruscamente hasta el techo. La forma de la recta se puede ver en la Ilustración 59. Su *1-D Lookup Table* se ajusta cambiando los *Breakpoints* según lo mostrado en la Ilustración 58.

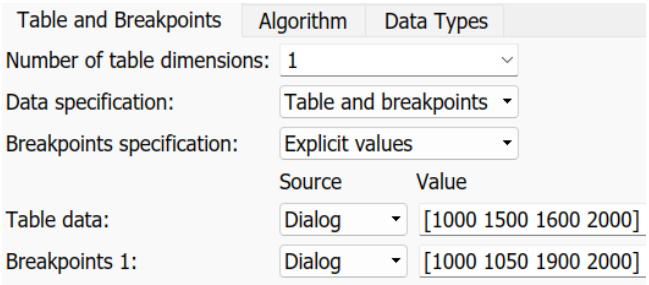

Ilustración 58: Breakpoints de la 1-D Lookup Table del Throttle

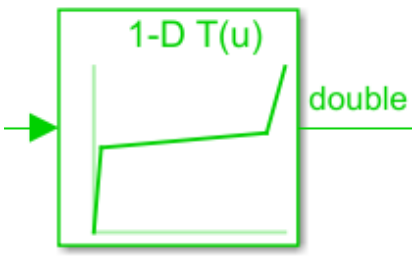

Ilustración 59: 1-D Lookup Table del Throttle

Las rectas del resto de canales, los de los giros, comienzan en el punto central o de reposo, y desde ahí decrecen o crecen según hacia donde se mueva el joystick. Por ejemplo, el cabeceo que es el canal 2, es el recorrido del joystick derecho en vertical. En el punto central el cabeceo es nulo, al mover el joystick hacia arriba, la curva crece y el dron gira hacia delante generando un movimiento hacia delante. Al contrario, si se gira el joystick hacia abajo, el dron gira hacia detrás haciendo que el dron se desplace hacia detrás. El problema es que estos giros y desplazamientos son demasiado bruscos y rápidos tanto en el cabeceo como en el alabeo y la guiñada, y ocurren de forma muy abrupta, aunque el joystick se esté moviendo solo un poco.

Para solucionar esto, las nuevas rectas deben tener la pendiente lo más suave posible en la zona central. De esta manera, al mover los joysticks en cualquiera de las dos direcciones, el dron se desplaza poco a poco y no tan bruscamente. La forma de las rectas se puede ver

en la Ilustración 61Ilustración 59. Sus *1-D Lookup Table* se ajustan cambiando los *Breakpoints* según lo mostrado en la Ilustración 60.

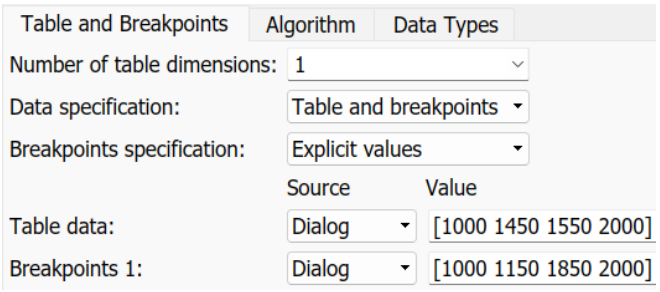

Ilustración 60: Breakpoints de la 1-D Lookup Table del resto de canales

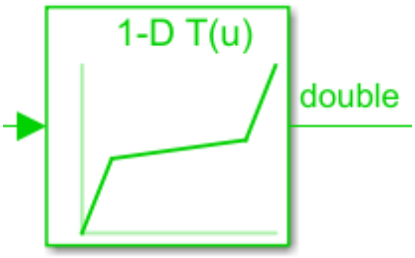

Ilustración 61: 1-D Lookup Table del resto de canales

Las *1-D Lookup Table* se colocan en cada canal en *RC TRANSMITER* según lo mostrado en la Ilustración 62.

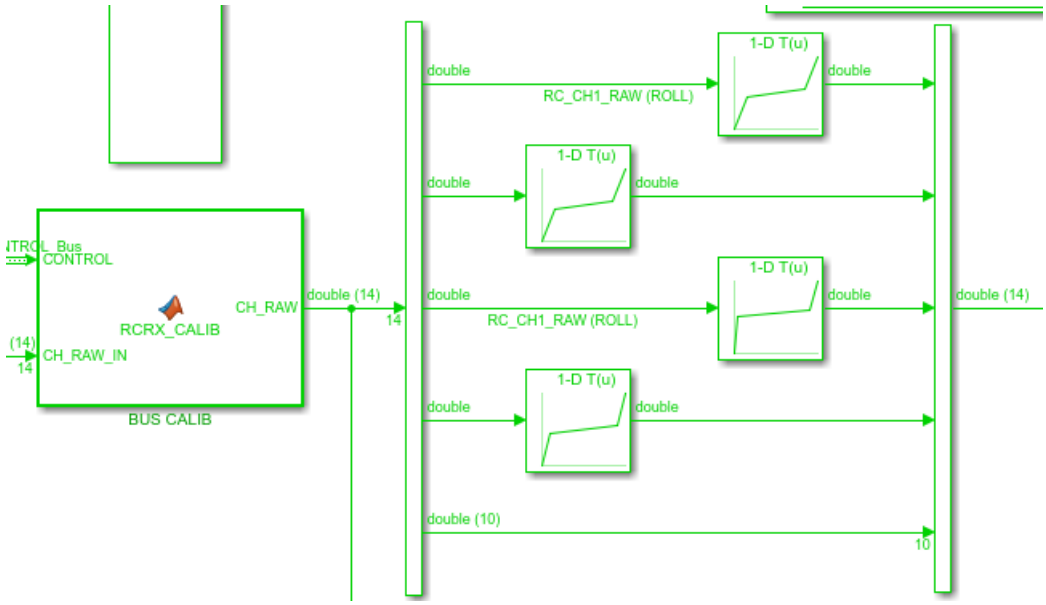

Ilustración 62: UAV_CONTROL_SYSTEM, SIMULATION, RC TRANSMITER suavización de cada canal

Como resultado, el vuelo manual tiene movimientos mucho más suaves, y ya es posible de realizar.

7. Conclusiones

En este capítulo se concluye el proyecto con dos secciones. En la sección 7.1 se explican las complicaciones y los objetivos que se han cumplido. En la sección 7.2 se aportan algunas ideas en las que se puede trabajar en futuros proyectos, a partir de los resultados de este.

7.1. Objetivos cumplidos y dificultades

Este proyecto es el resultado de la unión de distintos trabajos que se han realizado en proyectos previos, juntando distintas piezas para formar el simulador completo. Los trabajos previos han sido:

- El diseño del entorno del laboratorio en SolidWorks y SketchUp, por María Ana Saenz Nuño, codirectora de este proyecto.
- El diseño de la estructura del dron del laboratorio en SolidWorks, resultado del trabajo de fin de grado *Diseño Mecánico de un Dron para Inventario Automático de Almacenes*, de Diego Cubillo Llanes [26].
- El modelado del control dinámico del vuelo del dron en Matlab y Simulink, por Juan Luis Zamora Macho, codirector de este proyecto.
- El trabajo en UnReal y Python de las becas de colaboración de Claudia Valero de la Flor, en el verano de 2021, y de Javier José Carrera Fresneda, en el curso 2021/2022. [30] [31]

El proyecto ha constado de la comprensión del trabajo de estas becas, que ha sido repetido desde cero y ha quedado detallado en esta memoria, así como de todo el trabajo que faltaba para llegar a que el simulador funcionase correctamente. Esta memoria funciona como guía para repetir el proceso de la unión de las distintas partes que forman el simulador. De esta manera, puede montarse de nuevo el simulador desde el principio, siguiendo y comprendiendo cada paso. Los pasos para unir las distintas partes son los detallados anteriormente en esta memoria, y coinciden con los objetivos cumplidos en este trabajo:

- Creación del proyecto de UnReal desde cero, importando en él el entorno del laboratorio y el simulador AirSim, y cambiando el modelo estructural del dron por el del laboratorio.

- Creación de las funciones de Python necesarias para comunicar AirSim con Simulink, así como los archivos de Matlab y Simulink que envían orientaciones y posiciones necesarias a AirSim.
- Integración de los archivos de Matlab y Simulink creados al Simulink del modelo dinámico del dron, y ajuste de parámetros y características para su correcto funcionamiento.
- Trabajo en UnReal, Python, y Simulink, para conseguir que se detecte una colisión del dron con otro objeto virtual, y que cuando esto ocurra, se finalice la simulación.
- Descripción detallada de todo el proceso de estos cuatro pasos en esta memoria.

7.2. Ideas para futuros proyectos

Aunque el resultado del simulador es bueno, no es perfecto, dejando libre el camino para que otros añadan complementos en futuros proyectos. Algunas ideas que podrían perfeccionar el trabajo pueden ser:

- Estudio de UnReal y AirSim en mayor profundidad, así como la creación de nuevas funciones en Python, para mejorar al máximo las colisiones del dron con otros objetos virtuales. En vez de finalizarse la simulación, que el dron se choque y rebote o caiga, pero que la simulación siga funcionando.
- Estudio de los sensores de proximidad en ciertas direcciones o sensores lidar que pueden añadirse a AirSim, y creación de nuevas funciones en Python, para programar vuelos autónomos del dron que consten en recorridos siguiendo las paredes virtuales por proximidad.
- Estudio y mejora del modelo dinámico del dron en Matlab y Simulink. Mejora del modelo y de los parámetros mediante repetición de recorridos y machine learning.
- Ajuste del tamaño del dron a un tamaño exactamente igual a su tamaño real, comparándolo respecto al resto de objetos virtuales del laboratorio. En este momento el dron tiene un tamaño prácticamente idéntico al real al compararlo con el laboratorio virtual, pero no guarda una proporción exacta.
- Añadir en UnReal un modo de vuelo FPV, es decir, en primera persona. De esta manera, se podrá volar el dron como si se viese a través de una cámara que lleve el propio dron. Conectando a UnReal unas gafas de realidad virtual, se podría ver el vuelo de una forma inmersiva, como en los entrenamientos de vuelo profesional.

8. Concordancia con los objetivos de desarrollo sostenible de la ONU

[32] El proyecto concuerda con algunos de los objetivos sostenibles de la ONU, y su consecución no perjudica ni frena ninguno de los objetivos. Los objetivos que ayuda a cumplir son la producción y el consumo responsables, así como la acción por el clima.

El consumo y la producción son impulsores de la economía mundial, y necesitan del uso medio ambiente y los recursos, provocando en muchas ocasiones efectos dañinos sobre el planeta. El consumo y la producción sostenibles se resume en conseguir hacer más y mejor con menos; es decir, en utilizar menos recursos para conseguir cumplir las máximas tareas posibles.

Mediante el uso de un simulador de vuelo de drones para probar tanto los vuelos autónomos como los manuales, se reducen los accidentes que ocurrirían al probarse los vuelos del modelo dinámico directamente sobre el dron real. De esta manera, se evita que piezas del dron se destruyan en accidentes y deban ser reemplazadas, reduciendo en gran medida la sobreutilización de recursos que supondría reponer estas partes.

Por otro lado, el objetivo de la acción por el clima incluye la reducción del uso abusivo de baterías o pilas. Las baterías están compuestas mediante sustancias como mercurio, cadmio, litio o plomo, que son sumamente tóxicas para la salud y el ambiente. Estas baterías terminarán convirtiéndose en un residuo tóxico, y sus componentes químicos se modificarán con el ambiente, volviéndose en algunos casos incluso más tóxicos.

El uso de un simulador en vez de un dron real para pruebas de vuelo elimina el desgaste innecesario de baterías de drones, o su posible destrucción en un accidente. Los drones suelen utilizar baterías de litio polímero, las cuales si se incendian por un accidente explotan, emitiendo gases tóxicos y corrosivos. El contacto de estas baterías con la humedad, el agua, o cualquier tipo de líquido o vapor puede provocar un cortocircuito o un deterioro químico, lo que derivaría también en que la batería se incendie, conllevando a su explosión y la emisión de gases.

9. Bibliografía:

[1] Talentum Industrias Avanzadas, «Talentum Digital,» 20 01 2020. [En línea]. Available: https://talentumdigital.cl/2020/01/20/por-que-elegir-solidworks/. [Último acceso: 23 03 2022].

[2] Trimble Inc., «SketchUp,» 2022. [En línea]. Available: https://www.sketchup.com/es. [Último acceso: 23 03 2022].

[3] X. Chen, M. Wang and Q. Wu, "Research and development of virtual reality game based on unreal engine 4," 2017 4th International Conference on Systems and Informatics (ICSAI), 2017, pp. 1388-1393, doi: 10.1109/ICSAI.2017.8248503.

[4] Epic Games, Inc., «Epic Games Unreal Engine,» [En línea]. Available: https://www.unrealengine.com/en-US. [Último acceso: 22 03 2022].

[5] Microsoft Research, «AirSim,» 2021. [En línea]. Available: https://microsoft.github.io/AirSim/. [Último acceso: 23 03 2022].

[6] Microsoft Research, «AirSim APIs,» 2021. [En línea]. Available: https://microsoft.github.io/AirSim/apis/. [Último acceso: 23 03 2022].

[7] The MathWorks, Inc., «MathWorks, Matlab,» [En línea]. Available: https://es.mathworks.com/products/matlab.html. [Último acceso: 24 03 2022].

[8] The MathWorks, Inc., «MathWorks, Simulink,» [En línea]. Available: https://es.mathworks.com/products/simulink.html. [Último acceso: 24 03 2022].

[9] Fangohr, H. (2004). A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds) Computational Science - ICCS 2004. ICCS 2004. Lecture Notes in Computer Science, vol 3039. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-25944-2_157

[10] Anaconda, Inc., «Anaconda.org,» 2022. [En línea]. Available: https://anaconda.org/anaconda/python. [Último acceso: 24 03 2022].

[11] L. Garber, «The Lowly API is Ready to Step Front and Center,» 08 2013. [En línea]. Available: https://www.computer.org/csdl/magazine/co/2013/08/mco2013080014/13rRUzpQPJ9. [Último acceso: 24 03 2022].

[12] Microsoft, «Microsoft, Visual Studio,» [En línea]. Available: https://visualstudio.microsoft.com/es/. [Último acceso: 24 03 2022].

[13] Revista Empresarial & Laboral, «7 aplicaciones profesionales de un drone,» 2017.

[14] G. Brunner, B. Szebedy, T. Simon y R. Wattenhofer, «The Urban Last Mile Problem: Autonomous Drone Delivery to Your Balcony,» 2019. https://doi.org/10.48550/arXiv.1809.08022

[15] USS, «Drone Services,» 31 03 2020. [En línea]. Available: https://www.droneservices.com.ar/industria-4-0/simulador-de-drones/. [Último acceso: 22 03 2022].

[16] A. Devos, E. Ebeid and P. Manoonpong, "Development of Autonomous Drones for Adaptive Obstacle Avoidance in Real World Environments," 2018 21st Euromicro Conference on Digital System Design (DSD), 2018, pp. 707-710, doi: 10.1109/DSD.2018.00009.

[17] S. Wang, J. Chen, Z. Zhang, G. Wang, Y. Tan and Y. Zheng, "Construction of a virtual reality platform for UAV deep learning," 2017 Chinese Automation Congress (CAC), 2017, pp. 3912-3916, doi: 10.1109/CAC.2017.8243463.

[18] Drones España, «Drones España,» [En línea]. Available: https://xn--drones-espaa-khb.eu/los-mejores-simuladores-de-drones/. [Último acceso: 16 05 2022].

[19] Liftoff, «Liftoff Drone Simulations,» [En línea]. Available: https://www.liftoff-game.com/liftoff-fpv-drone-racing. [Último acceso: 24 03 2022].

[20] VelociDrone, Bat Cave Games, «Velocidrone,» [En línea]. Available: https://www.velocidrone.com/. [Último acceso: 24 03 2022].

[21] Epic Games, Inc., «Epic Games, The Drone Racing League Simulator,» [En línea]. Available: https://store.epicgames.com/es-ES/p/the-drone-racing-league-simulator. [Último acceso: 24 03 2022].

[22] DJI, «DJI Flight Simulator,» [En línea]. Available: https://www.dji.com/es/simulator. [Último acceso: 16 05 2022].

[23] droneSim Pro., «droneSim Pro Drone Simulator,» [En línea]. Available: https://www.dronesimpro.com/. [Último acceso: 16 05 2022].

[24] FPV FreeRider, «FPV FreeRider,» [En línea]. Available: https://fpv-freerider.itch.io/fpv-freerider. [Último acceso: 16 05 2022].

[25] A. Mairaj, A. I. Baba y A. Y. Javaid, «Application specific drone simulators: Recent advances and challenges,» Elsevier B. V., 2019, https://doi.org/10.1016/j.simpat.2019.01.004.

[26] D. Cubillo Llanes, «Diseño Mecánico de un Dron para Inventario Automático de Almacenes,» 2019. Universidad Pontificia de Comillas, Escuela Técnica Superior de Ingeniería (ICAI) Identificador: http://hdl.handle.net/11531/40919

[27] TheSkethUpEssentials, «Import SketchUp Files to Unreal Engine with Datasmith,» 2018. [En línea]. Available: https://youtu.be/uhw2DwcoXr8. [Último acceso: 04 02 2022].

[28] C. Lovett, «Unreal AirSim Setup from scratch,» 2017. [En línea]. Available: https://www.youtube.com/watch?v=1oY8Qu5maQQ&t=2s. [Último acceso: 05 02 2022].

[29] A. Elsaharti, «How to use a custom drone frame mesh in AirSim,» 13 04 2021. [En línea]. Available: https://www.youtube.com/watch?v=Bp86WiLUC80. [Último acceso: 05 02 2022].

[30] C. Valero de la Flor, *Beca de colaboración con la Universidad Pontificia Comillas, ICAI,* 2021.

[31] J. J. Carrera Fresneda, *Beca de colaboración con la Univeridad Pontifica Comillas, ICAI,* 2021.

[32] ONU, «Objetivos de desarrollo sostenible,» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/. [Último acceso: 19 07 2022].