Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Simulación dinámica de flota de drones

Autor: Joaquín Romero Cabeza

Tutor: D. Eduardo Fernández Camacho

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

Sevilla, 2022

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Simulación dinámica de flota de drones

Autor:
Joaquín Romero Cabeza

Tutor:
D. Eduardo Fernández Camacho
Profesor Titular

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

Sevilla, 2022

Trabajo Fin de Grado: Simulación dinámica de flota de drones

Autor: Joaquín Romero Cabeza
Tutor: D. Eduardo Fernández Camacho

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Me gustaría agradecer a todas esas personas, principalmente a mis padres, que me han oído el alma, aunque fuera por teléfono, y a las personas tan maravillosas que conocía y que he conocido durante este periodo, la compañía y apoyo tan esencial que he recibido por parte de ellos en los peores momentos. A todas ellas, con las que asimismo he celebrado éxitos, gracias, de corazón. Han sido momentos de esfuerzos titánicos, en el que se han tenido que sacrificar infinitos planes, incluso alguno tan básico como volver a tu propia casa en otra localidad, por estudiar unas horas más. Momentos muy complicados, de sudor y lágrimas, pero al final todo llega a su fin, si de verdad quieres y te lo propones.

Dar las gracias también a mi tutor Eduardo Fernández Camacho, con el cual me he sentido muy cómodo y valorado a la hora de hacer este trabajo. Desearía que hubiese más personas como él en este mundo.

Aprovecho para terminar añadiendo: Gracias al Grado en Ingeniería Aeroespacial, en general a varios profesores, por cambiarme la forma de pensar, que aunque un gran número de veces a lo largo de estos años lo haya hecho de forma totalmente contraria, cuando estás a punto de terminar, entiendes que la ingeniería, más que enseñarte algoritmos de resolución de problemas, te enseña una actitud, una forma de ser disciplinado y de no rendirte.

Joaquín Romero Cabeza
Sevilla, 2022

Resumen

El objetivo del presente proyecto es la elaboración de un modelo dinámico de una flota de 3 drones, capaz de ser controlada para cumplir unas determinadas misiones de medición, teniendo en cuenta las posibles colisiones entre ellos, particularizado especialmente para un proyecto real europeo llamado *OCONTSOLAR*, en el cual participa la Universidad de Sevilla. Además de la simulación del modelo ya comentado, se ha implementado una representación en tiempo real, bastante realista, de la flota junto a la planta de energía solar *Solnova 4*, elección que se justificará posteriormente. El proyecto consta de un sistema supervisor, que es el que organiza las misiones y evita colisiones de los drones principalmente, y de un modelo individual de dron, que, aunque con menor inteligencia que el sistema supervisor, es capaz de controlar su dirección hacia los *waypoints* ordenados.

La programación del modelo dinámico ha sido llevada a cabo en el lenguaje de programación *Python*, mientras que el modelado del escenario se ha materializado en *Unity*, por lo que ha sido necesario el lenguaje *C Sharp* para determinados scripts, y *SketchUp Pro* para edición del modelo 3D del colector.

Abstract

The purpose of the project is the development of a dynamic model, which consists of 3 drones with the capability of being controlled and get the drones'missions accomplished. This model also prevents collisions between the three drones, and it has been done particularly for a real European project, named *OCONTSOLAR*, in which the University of Seville takes part. Moreover, a real-time representation has been implemented, with enough realism, that simulates the fleet around a solar energy plant called *Solnova 4*. The project roughly contains a supervisor system that mainly manage the missions and avoid collisions, and a basic drone model which is less smart, but guides the drone to the ordered *waypoints*.
The implementation of the dynamic model has been done in *Python*, however the scenery has been made in Unity, so *C Sharp* has been useful in the making of some scripts, and *SketchUp Pro* for editing the 3D collector model.

Índice Abreviado

Índice

1 Introducción y objetivos

En este primer capítulo de la memoria se va a poner en contexto el por qué de este trabajo, y los objetivos que se han pretendido conseguir a la hora de la realización.

1.1 Introducción y contextualización

Sin lugar a dudas, la energía solar hoy en día se está convirtiendo poco a poco en la líder de las renovables debido a sus innumerables ventajas [9]:

- Energía 100 % inagotable, renovable y gratuita
- No emite sustancias tóxicas ni contaminantes del aire
- Reduce el uso de combustibles fósiles
- Medida contra el cambio climático

Aun así, conlleva ciertas desventajas respecto a las otras renovables:

- Energía intermitente
- Localización de la vivienda

Por tanto, a pesar de estar algo perjudicada a nivel particular por las limitaciones de las viviendas, sigue siendo una muy buena fuente a explotar a gran escala si se estudia con rigor.
Por ello, hay cada vez más plantas que persiguen generar energía de esta forma. Sobre todo, las termosolares están llevando a cabo un proceso amplio de expansión pues, ya sea un campo de helióstatos apuntando a una torre, o bien sea una central de canales parabólicos, tienen la principal ventaja de almacenar energía [16] en fluidos especiales para un posterior aprovechamiento en caso de carencias en la generación de potencia.
El almacenamiento de energía en una central fotovoltaica actualmente sería inviable, debido a la baja densidad energética de las baterías a fecha de hoy, y su alto coste.

Lo anterior nos lleva a elegir mejorar las termosolares aún más. Sin embargo, se deberían emplear multitud de sensores costosos que miden la intensidad de irradiación solar, de unos 25.000€ por cada unidad.

Figura 1.1 Optimal Control of Thermal Solar Energy Systems [8].

Es aquí donde surge la idea del proyecto *OCONTSOLAR*, que pretende montar en drones y en vehículos terrestres no tripulados estos tipos de sensores y con estos vehículos recorrer el área requerida.

Una vez se tenga el modelo de control de estos vehículos para una planta solar de referencia con el fin de ahorrar el colocar un sensor por cada colector, se podrá extrapolar a otras aplicaciones, tales como:

- Control en el tráfico
- Control de energía en edificios
- Agricultura
- Inundaciones

Figura 1.2 Planta de energía Solnova 4 [13].

Por simpleza geométrica, se va a tomar en este proyecto la planta de energía termosolar *Solnova 4*, visible en la parte inferior de la Figura 1.2, ubicada en Sanlúcar la Mayor, Sevilla.
Esta planta apuesta por tecnología cilindroparabólica, por lo que los colectores de los que disponen van a rotar alrededor de un eje, teniendo asi un solo grado de libertad para orientarse hacia el sol, y conseguir la mayor radiación posible.

La forma en la que los colectores captan la energía solar es tal que, una vez que los reflectores cilindroparabólicos son óptimamente posicionados respecto al sol, estos reflejan la radiación solar que les llega a un tubo por el cual circula una corriente de un fluido con alto calor específico. Por ello, la reorientación de los colectores casi continua es clave en la maximización de la eficiencia de la planta en general.

1.2 Objetivos del trabajo

A continuación se abordan los objetivos principales del trabajo que tienen que ver con el modo de funcionamiento y con los sensores anteriormente comentados.

1.2.1 Control de una flota de drones

El principal propósito del trabajo es el control, y posterior simulación, de una flota de drones para medir la radiación solar en distintos puntos de la planta.
Esta idea permitiría una reducción notable de costos, ya que aunque esta mejora implica un nuevo coste en este tipo de sensores de alrededor de 75.000€, mucho menos que los 9.300.000€ que costarían los 372 sensores de los 372 colectores que tiene el modelo de planta solar ~~que~~ se ha utilizado.

El modelo dinámico consta de varias partes: el sistema supervisor, y el dron individual. Se ha pretendido que cada uno tenga las siguientes funciones. El supervisor contiene:

- Reparto inicial de las tareas dadas con capacidad de elección del usuario
- Vigilancia de posibles colisiones
- Reasignación de tareas una vez se vayan finalizando
- Gestión de movimientos del vehículo con batería baja

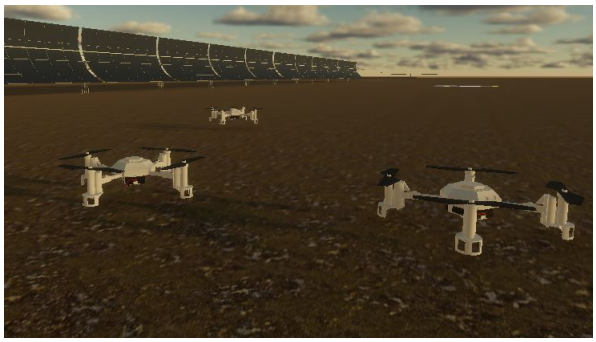


Figura 1.3 Flota de 3 drones individuales.

El dron individualmente tiene las siguientes funciones:

- Maniobra de aceleración y desaceleración
- Función simuladora de batería
- Función de espera para el sensor
- Asignación de *waypoints* en caso de ser alertado por el supervisor ante colisión

1.2.2 Simulación del entorno en tiempo real

Por otro lado, se le ha dado más valor al trabajo añadiendo una representación en tiempo real de la flota cumpliendo dichas tareas de medición de radiación. Con ello, se puede comprobar de una manera más cercana que es lo que pasaría si el control de la flota se llevará a plasmar en la realidad.
Se han seguido los siguientes pasos:

- Edición del modelo 3D de dron
- Edición del modelo 3D de colector
- Creación del terreno sobre el que operar
- Adición de nubes y sol artificiales

Ha sido necesario edición de algunos elementos 3D para su posterior adaptación al entorno de forma adecuada, incluso para la optimización gráfica de fotográmas por segundo. Todo ello se explicará con mayor profundida y detalle en los siguientes capítulos.

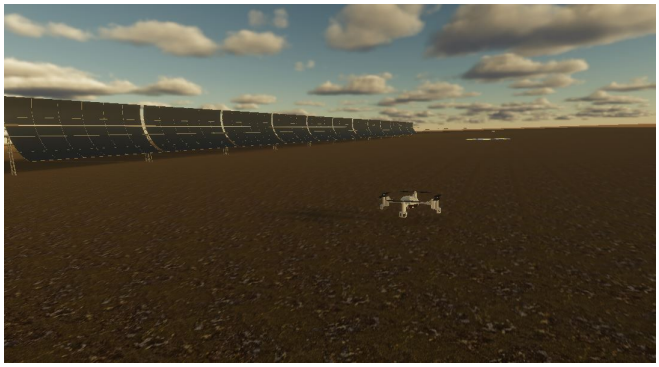


Figura 1.4 Escenario tipo, que contiene dron y colector.

2 Modelo dinámico

Es este el capítulo que pretende detallar el desarrollo que se ha concluido para gestionar la movilidad de la comentada flota a lo largo y ancho del entorno creado en Unity. Comprende todos los tramos, desde la organización inicial del sistema supervisor, el sistema anticolisión, hasta la vuelta a la base del propio dron.

2.1 Algoritmo de organización y asignación de misiones

La primera línea seguida consistió en la optimización de la operación de forma global.

Esa línea se desestimó debido al tiempo de computo necesario ya que el algoritmo evalúa, para los drones disponibles, todas las posibles combinaciones, para que la suma de las distancias que tengan que recorrer los tres drones sea lo mínima posible.

Respecto a ello, el tiempo computacional requerido para un número de misiones bajo, del orden de 10-50 es implementable y funciona bien, pero cuando pasa de 300, ya dependiendo del procesamiento de la computadora, se ralentiza bastante a la hora de pasar por todas las misiones, por lo que es menos viable. Aun así, para 372 tareas es posible utilizarlo, aunque tarde algo más. El único inconveniente de utilizarlo es el que los drones deben seguir trayectorias no tan ordenadas y rectas como de la otra forma.

Vista la ineficiencia de este algoritmo, aunque se haya implementado, se pensó en un segundo método de asignación. Se trata de un método de clusterización jerarquizada.
Esta técnica se suele emplear, entre otros campos, por las aerolíneas, para organizar el tráfico aéreo. Simplemente son agrupaciones de puntos por distancia, la cual puede representar multitud de variables: desde productos que por estar cerca en una gráfica de precios forman parte del mismo grupo o *cluster*, hasta aviones que comparten un mismo destino, y reorganizan su posición por proximidad en aeropuertos.

Con todo, se le dejará como una opción a elegir por el usuario, por defecto a 50 tareas como máximo para el método optimizador de distancia. Se procede a la explicación del funcionamiento de cada algoritmo, ambos ejecutados por el sistema supervisor.

Antes de comenzar, es necesario establecer como se van a introducir los datos iniciales de partida. Se barrerán todas las posiciones centrales de los colectores, que están formados por 12 módulos cada uno. Estas se proporcionarán en formato lista de listas, entendibles por Python, como por ejemplo: [[88,31.3,10], [240.28,31.3,10]], siendo las posiciones en x, y, z de los dos primeros colectores. Este vector será el llamado *mision*.

2.1.1 Selección de tareas por distancias más cercanas

En este caso, para cada tarea dada, se itera la siguiente expresión para cada dron, siendo capaz de evaluar la distancia entre la misión k y el dron j.

$$\sqrt{(x_j - x_{misionk})^2 + (y_j - y_{misionk})^2 + (z_j - z_{misionk})^2} \tag{2.1}$$

Consecuentemente, con las tres distancias a cada misión, estas se reajustan mediante la adición de un pequeño diferencial, por si se diese el caso de repetición, para posteriormente ser ordenadas de menor a mayor.

Una vez ordenadas, si la misión no ha sido asignada anteriormente, y el dron se encuentra libre, entonces es asignada la misión al dron que se encuentre a menor distancia. Si el vehículo ya tuviera una misión en concreto, se pasa al siguiente dron que esté algo más lejos, y se intenta realizar el mismo proceso.

Este proceso se itera n_{tareas} veces, siendo n_{tareas} el número de misiones dadas inicialmente, con el fin de minimizar la suma de las distancias totales obtenidas en cada iteración para cada dron. Se realiza de tal forma que las componentes del vector *mision* van alternándose de lugar, para así no darle preferencia a ninguna misión en especial, y que sean escogidas las más cercanas en distancias.

2.1.2 Selección de tareas por clusterización

La segunda opción a elegir es la clusterización jerarquizada. Primero se debe conocer cómo funcionan las k-medias antes de entrar en más profundidad con la clusterización. Una breve revisión de las k-medias es el siguiente proceso [17]:

- Se decide el número de grupos (k)
- Selección de k puntos aleatorios de los datos como centroides
- Asignación de todos los puntos al centroide más cercano que tengan respectivamente
- Se calculan los centroides de los nuevos grupos de puntos formados
- Repetición de los últimos dos pasos

Es un proceso completamente iterativo. Se mantienen las iteraciones hasta que los centroides de los nuevos grupos formados no cambien, o se llegue a un número máximo de iteraciones.

Pero hay ciertos problemas con las k-medias. Siempre intenta realizar grupos con el mismo tamaño. Entonces, se pretende analizar a continuación qué es lo bueno del agrupamiento o *clusterización* jerarquizada.

Hay dos tipos de clusterización jerarquizada:

- *Agglomerative hierarchical clustering [1]*
- *Divisive hierarchical clustering*

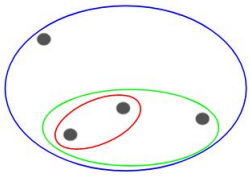


Figura 2.1 Última iteración *Agglomerative hierarchical clustering*.

Mientras que en el primer tipo se parte de cada punto individual y se va agrupando en grupos cada vez mayores, en el segundo tipo es totalmente al contrario, partiendo de un grupo con todos los puntos dividiéndose en subgrupos con cada vez menos en cada iteración.

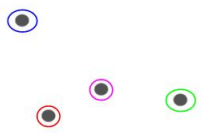


Figura 2.2 Última iteración *Divisive hierarchical clustering*.

Entonces, la pregunta es clara, ¿Cómo se decide qué puntos son similares y cuáles no?

La respuesta: con la distancia entre los centroides de estos grupos. Los puntos que tienen la menor distancia son referidos como puntos similares, y se pueden juntar. Básicamente, se le puede tratar como un algoritmo basado en distancias, por el hecho de que sea vital para los grupos.
Por ello, en clusterización jerarquizada se tiene un concepto llamado *matriz de proximidad*. Esta matriz guarda las distancias entre puntos, y es la cual se sigue el proceso iterativo mencionado.

Para una mayor facilidad, se ha insertado un modelo ya entrenado de *Machine Learning*, encontrado en el repositorio de *UCI Machine Learning* [1], en el que, dado el número de grupos en los que se van a segmentar los datos, el tipo de afinidad, la forma de unión al grupo y las coordenadas, arroja como resultado las etiquetas asignadas a cada misión, o lo que es lo mismo, el grupo al que pertenece.

- Número de grupos: Coincide con el número de drones
- Afinidad: Euclídea
- Forma de unión: Por secciones
- Datos a agrupar: Coordenadas x, y de las tareas a realizar

A continuación, un ejemplo en que se observan tres agrupaciones de puntos de coordenadas aleatorias:

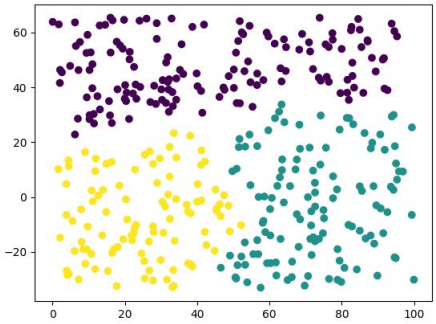


Figura 2.3 Ejemplo uso algoritmo de agrupamiento.

Y en la siguiente figura, ya aplicado al modelo, un grupo para cada dron, con la forma en planta de la central en consideración:

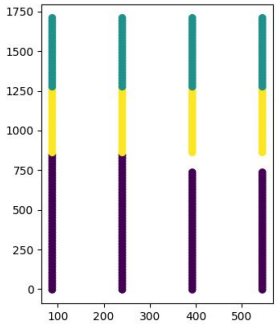


Figura 2.4 Agrupamiento tareas de la central para tres drones.

En este último ejemplo, se puede observar cómo se distribuyen las misiones para cada vehículo aéreo, siendo la zona blanca a la derecha donde ocurre el procesamiento de la energía con turbinas, obteniendo como principal ventaja respecto al tipo de organización anteriormente comentada que la probabilidad de colisión entre drones disminuya mucho más, debido al gran distanciamiento constante entre vehículos durante el proceso. Por comodidad, aunque forme parte del sistema supervisor, se ha implementado en el código que ejecuta la cinemática.

2.2 Algoritmo anticolisión

El problema más importante de la gestión de la flota es el evitar colisiones entre los propios vehículos. Inicialmente, se podría plantear con la instalación de sensores de ultrasonidos, ópticos, incluso de cámaras con reconocimiento de objetos con ayuda de inteligencia artificial.
Pero, cada dron ya lleva montado el sensor de radiación solar, por lo que se optó por otro camino.

Y es que, como se ha explicado, el sistema supervisor tiene más "inteligencia" que el dron propio. Entonces, es interesante que el sistema supervisor conozca las posiciones de toda la flota en cada instante para que se puedan tomar decisiones.
Se va a proceder con esto último.

Se empieza con la siguiente idea tan simple:

$$\vec{x}_i(t) = \vec{v}_i(t) \cdot t + \vec{x}_{i0} \tag{2.2}$$

donde i es el indice identificativo del dron i, x_i la posición del dron i en el instante t, x_{i0} la posición del dron i en el instante inicial y v_i la velocidad del dron i en el instante t, todos estos vectores de tres dimensiones (x,y,z).

Con esta ecuación vectorial, se pueden sacar varias conclusiones. Simplemente, si se toma la velocidad como constante, se puede propagar la trayectoria, y obtenerse la posición para cualquier instante t posterior.
Por ello, haciendo la hipótesis de velocidad constante se puede obtener una trayectoria bastante cercana a la que se produzca en la realidad, puesto que en la transición de misiones interesa una velocidad constante, ya sea por cuestiones mecánicas, para evitar desgastes innecesarios, como por cuestiones de simpleza en el control.

Pero aún se puede mejorar más. Si el sistema supervisor propaga las trayectorias en cada instante, tomando como velocidad la que lleva en ese determinado instante como constante, se convierte en una hipótesis excelente el no considerar aceleraciones, porque la mayor parte del tiempo, o se encontrarán volando a velocidad constante, o parados realizando la medición.

Por tanto, queda llevar esta idea al extremo: conociendo el sistema supervisor las posiciones de los drones y sus velocidades en cada instante, es capaz de, en cada diferencial de tiempo, propagar todas las trayectorias de todos los vehículos para detectar posibles colisiones. Esto último se traduce en buscar los mínimos de la función diferencia de posiciones por pares de drones, de la siguiente forma:

$$\vec{x}_i(t) - \vec{x}_j(t) = (\vec{v}_i(t) - \vec{v}_j(t)) \cdot t + \vec{x}_{i0} - \vec{x}_{j0} \tag{2.3}$$

y definiendo la función distancia al cuadrado como:

$$d(t)^2 = (\vec{x}_i(t) - \vec{x}_j(t))^T \cdot (\vec{x}_i(t) - \vec{x}_j(t)) \tag{2.4}$$

y por tanto su derivada:

$$\frac{d(d(t)^2)}{dt} = 2t \cdot (\vec{v}_i(t) - \vec{v}_j(t))^T (\vec{v}_i(t) - \vec{v}_j(t)) + 2(\vec{v}_i(t) - \vec{v}_j(t))^T \cdot (\vec{x}_{i0} - \vec{x}_{j0}) \tag{2.5}$$

e igualando a 0 esta expresión se obtiene un mínimo, puesto que dos drones que se mueven en línea recta, cada uno por la suya respectivamente, la distancia mínima puede ser mayor o igual a cero, dependiendo si colisionan o pasan cerca, y de ahí tiende a infinito, sin lugar a máximos. El otro caso es que siempre se mantengan a la misma distancia, por volar paralelamente, y por ello todos los puntos serían mínimos al mismo tiempo.
Por tanto, se puede obtener una expresión explícita del valor de t que hace mínima la distancia:

$$t_{dmin} = -(\vec{v}_i(t) - \vec{v}_j(t))^T \cdot (\vec{x}_{i0} - \vec{x}_{j0}) \cdot [(\vec{v}_i(t) - \vec{v}_j(t))^T \cdot (\vec{v}_i(t) - \vec{v}_j(t))]^{-1} \tag{2.6}$$

Se comprueba si este instante se encuentra dentro de un intervalo de tiempo razonable, esto es, si t_{dmin} es menor que cero, la colisión se habría producido en el pasado, caso que no tiene sentido; y por otro lado, si t_{dmin} es mayor a un valor $\frac{N\sqrt{2}}{V_{max}}$, que sería el tiempo en recorrer la diagonal de un cuadrado de lado N, por simplificar, a la máxima velocidad soportada por el dron, tampoco tendría sentido este caso, puesto que el punto más cercano de la trayectoria de dos drones estaría fuera del entorno de trabajo.

Además, se le añaden otras condiciones como:

- Distancia entre drones de menos de 60 metros para iniciar cambio de trayectoria, será menor cuanto más lejos de V_{max} esté su V. Con ello, se comprobará a una mayor distancia conforme más rápido se mueva
- El par de vehículos confluyan a una distancia mínima menor o igual a 3 metros en la propagación de la trayectoria
- El dron esté volviendo a base, yendo a misión, o apartándose hacia un lado por un esquivo anterior

Todo este proceso se itera, dejando un dron fijo (índice i) y pasando por el resto (índice j).
A partir de ahora, suponiendo que se ha detectado una posible colisión, se procede a la explicación del desarrollo de la maniobra anticolisión.

Para ello, antes se ha de aclarar la variedad de estados que admite el dron individualmente:

- Estado 0: Vehículo apagado/cargando en base
- Estado 1: Vuelo básico. Es el encargado del control del dron para llevar a cabo la simulación, tanto en aceleraciones, velocidades, como de contabilizar el tiempo que lleva midiendo la radiación. Además, es el encargado de calcular las trayectorias individualmente
- Estado 3.1: Maniobra anticolisión: Tramo de desaceleración
- Estado 3.2: Maniobra anticolisión: Tramo de desplazamiento lateral hacia waypoint
- Estado 3.3: Maniobra anticolisión: Tramo en el que se retoma la misión que iba desarrollando anteriormente
- Estado 4: Vuelta a base

Se ha empleado el siguiente algoritmo:

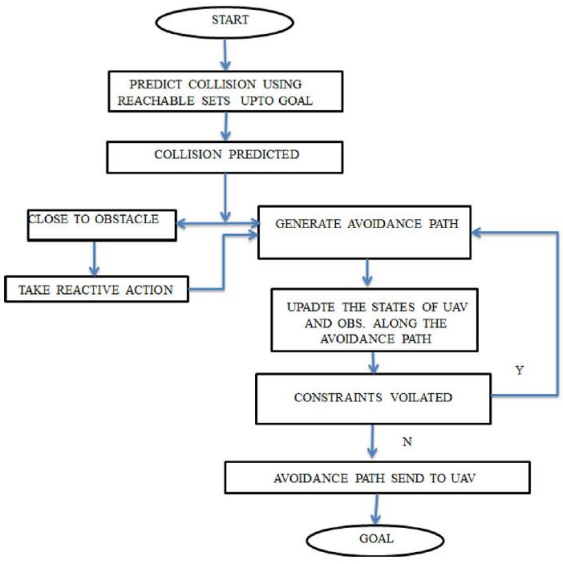


Figura 2.5 Forma de proceder una vez se detecta una posible colisión [4].

que se puede explicar como: se detecta una posible colisión. Si se está muy cerca del obstáculo, en este caso de otro vehículo, toma una acción impulsiva muy rápida, luego genera un *waypoint*, se le cambia de estado si todo es posible, y se consigue evitar.

2.2.1 Maniobra anticolisión: Tramo de desaceleración

Después de varias pruebas, se concluye que lo óptimo es detener el vehículo completamente, para evitar que mediante aceleraciones laterales se pierda totalmente el control del vehículo. De todas formas, el control del vehículo se hace de otra manera en la realidad, solo necesitando así como parámetros de entrada los *waypoints* necesarios, y no el control. El primer *waypoint* se trata de una desaceleración que lo lleve al vehículo a 5 metros en sentido contrario a la velocidad que llevaba cuando se detecta la posible colisión.
Este punto se calcula con la velocidad y su módulo, y se almacena en una lista llamada indic2, que contiene en cada componente la lista correspondiente a cada dron, con las coordenadas del punto a dónde se tiene que desplazar.
Este es el vector con el que trabaja fundamentalmente el estado 1, que acelera y desacelera dependiendo de la distancia al punto indicado.

2.2.2 Maniobra anticolisión: Tramo de desplazamiento lateral hacia *waypoint*

Este es el tramo que consigue cambiar la dirección de la trayectoria inicial. Ello lo consigue mediante un desplazamiento hacia un lado. Particularmente, se ha modelado teniendo en cuenta el funcionamiento del tráfico de vehículos automóviles.

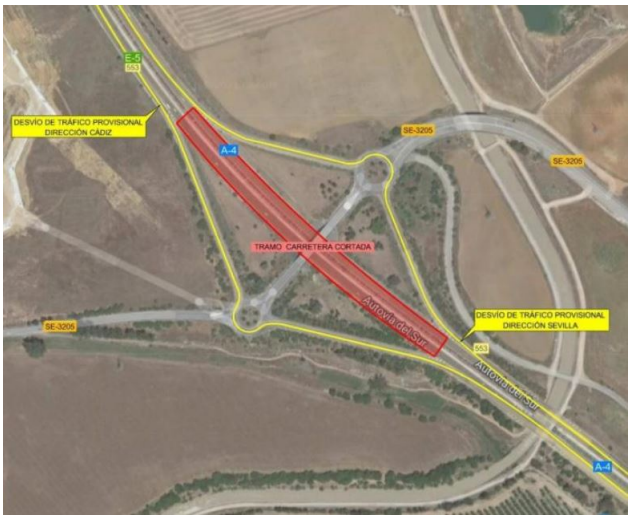


Figura 2.6 Representación visual criterio desplazamiento lateral [6].

La circulación de automóviles está basada en un criterio general por el que se deben regir todas las personas implicadas. Esto es, en una carretera de dos carriles, uno para cada sentido, si un vehículo A quiere tomar alguna salida, tiene que salir por su derecha, para evitar invadir el sentido contrario, mientras que el vehículo B también lo debe hacer por su mismo lado.

Con esto se consigue que no haya posibilidad a la colisión. Es esta la idea que se ha pretendido implementar, por lo que si hubiese dos drones que se encuentran de frente, ambos realizarían el desplazamiento lateral a su derecha de unos 10 metros, para así no poner en peligro la integridad de ningún vehículo, cambiando la trayectoria notoriamente.

2.2.3 Maniobra anticolisión: Tramo en el que se retoma la misión que iba desarrollando anteriormente

Con todo, lo lógico es retomar el desplazamiento que realizaba inicialmente, antes de detectar la posible colisión, hacia la tarea que fuese. Por ello, se recurre al vector misiones, que almacena las coordenadas de los puntos que tenía destinado cada dron respectivamente.
Se comprueba en este tramo, que además la misión no fuera una misión nula, con la componente de lista de mision distinto a [0,0,0], además de que el dron en cuestión tenga la suficientemente batería.
En caso contrario, volvería a la base, y dejaría de lado la tarea.

2.2.4 Sucesión de los estados anteriores

Los estados de la maniobra anterior siguen un determinado orden lógico, siendo el siguiente:

- Primero, el supervisor detecta el posible choque, y cambia el estado del dron en cuestión a 3.1
- El 3.1 dura un diferencial de tiempo, puesto este luego lo actualiza a 1, es decir, a vuelo normal
- Cuando se ha alcanzado el objetivo del tramo 3.1, se cambia al estado 3.2
- Como es de preveer, este estado dura asimismo un diferencial de tiempo, siendo el próximo estado el 1
- Cuando ha logrado el desplazamiento lateral, toma, durante un corto lapso de tiempo, el estado 3.3, para así volver al 1, y ya proseguir en ese estado

2.3 Algoritmo de reposicionamiento de tareas por evolución solar

Un problema que debe ser tenido en cuenta es el del movimiento del sol. Es evidente que a lo largo del día la dirección de procedencia de la radiación solar va cambiando.
Es por ello por lo que, en la central solar, los colectores usados tienen un grado de libertad, puesto que comparado con dos grados de libertad de rotación, al ser de uno ni añade tanta complejidad mecánica, ni necesita espacios grandes para el movimiento de cada colector.

Se va a considerar que el eje de rotación de los colectores es paralelo al eje x, lo que indica en sentido del norte geográfico visto en planta, simplificando la pequeña desviación angular.

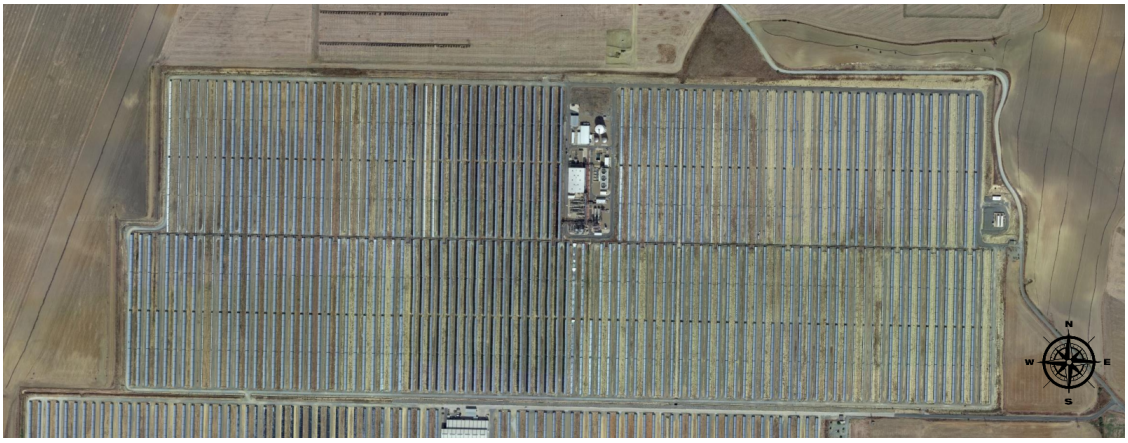


Figura 2.7 Orientación central *Solnova 4* [10].

El objetivo de esta sección es conseguir que los vehículos midan la radiación que de los colectores de interés, y por ejemplo, no se mida la radiación de un colector cuya posición es de 100 metros más alejado. De esta forma, el código de Python que ejecuta la cinemática recibe los ángulos por los que se encuentra definido el sol, θ_x, θ_y y θ_z. El traspaso de datos se explicará posteriormente. Con estos datos, se les hace las siguientes modificaciones oportunas para adaptarlos:

- 1. Si $\theta_z > 90$, entonces se tiene una nueva $\theta_x = 180 - \theta_x$ y una nueva $\theta_y = \theta_y - 180$
- 2. A θ_x se le cambia de signo, puesto en Unity se mide en sentido contrario al que interesa
- 3. Todos los ángulos se pasan a radianes
- 4. Se aplica trigonometría

La trigonometría se basa en la sombra del dron sobre el colector. Se basa en simples triángulos, usando las siguientes relaciones trigonométricas:

$$\begin{aligned} z_1 &= \frac{6}{\tan\theta_x} \\ x_1 &= z_1 \cdot \tan\theta_y \end{aligned} \tag{2.7}$$

Ambas coordenadas son las del plano horizontal en Unity. El 6 hace referencia a la altura que se encuentra el vehículo de dónde se sitúa la sombra, estando el vehículo a 10 metros de altura y el tubo de fluido a 4 metros.

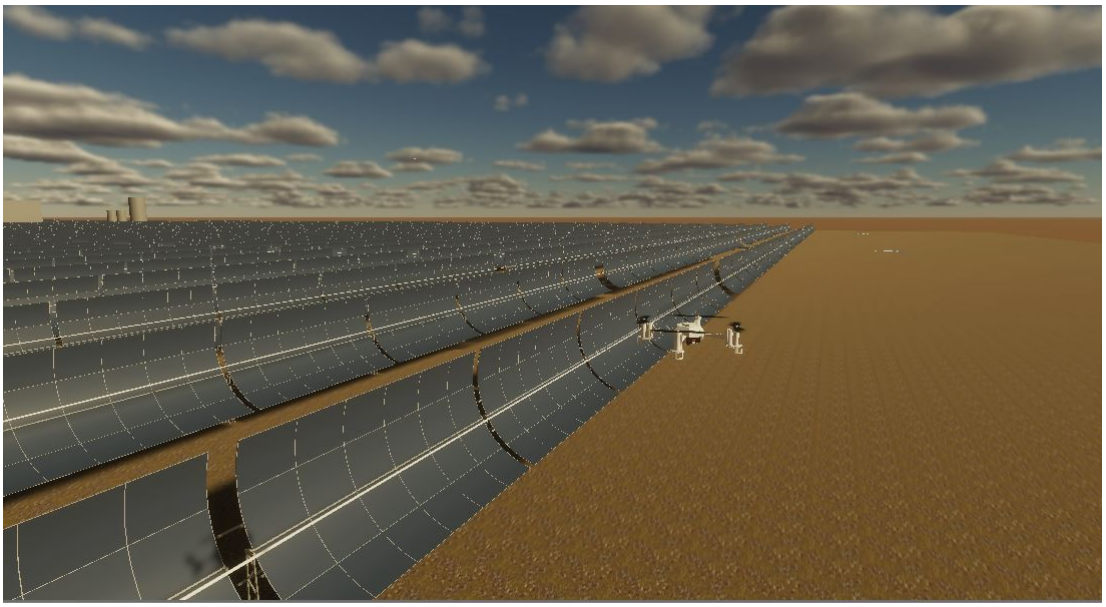


Figura 2.8 Sombra de dron sobre colector.

Como se observa, se ha pretendido que la radiación fuera la que incidiese sobre el tubo del colector.

Se le ha puesto de limitación de unos 2000 metros de deslocalización en cada eje coordenado, por motivo de salida de los drones del mapa, por mucha distancia.

2.4 Zonas de paso bloqueadas

Inicialmente, el modelo no tenía en cuenta la colisión con objetos estáticos perteneciente al mapa. Pero, es importante establecer una serie de restricciones, para que los vehículos no atraviesen objetos. Para ello, se han establecido 3 zonas:

- Zona de colectores
- Zona de turbinas y transformadores
- Zona de cota inferior a 0 metros

2.4.1 Zona de colectores

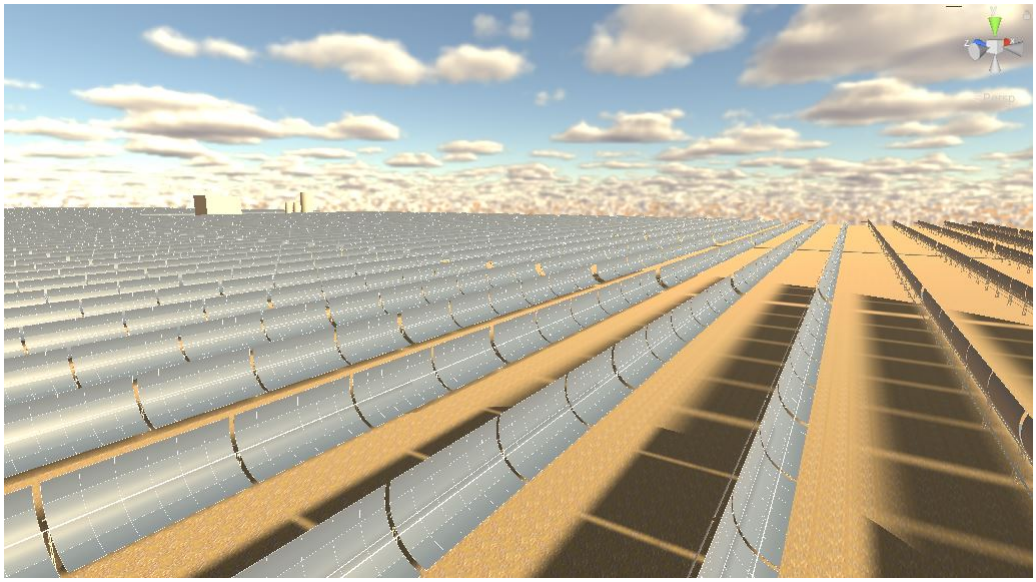


Figura 2.9 Colectores.

Las restricciones en esta parte son claras: el sistema supervisor impulsa hacia arriba si el dron sobrevuela una cota inferior a 8 metros. Es muy útil esta limitación, puesto que el vehículo como intenta desplazarse hacia las tareas en línea recta, hay siempre un momento en el despegue del dron que se tiene que impedir que vuele a tan baja altura.

2.4.2 Zona de turbinas y transformadores

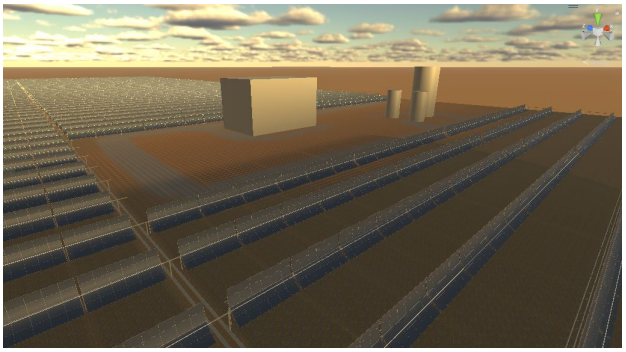


Figura 2.10 Zona central de la planta.

Esta localización es también de las más importantes a evitar, incluso a una altura mayor a unos 45 metros. Es una zona muy importante, puesto que pasar por encima de las torres demasiado cerca podría desestabilizar el dron, como un problema menor, y provocar su rotura en el peor de los casos.

2.4.3 Zona de cota inferior a 0 metros

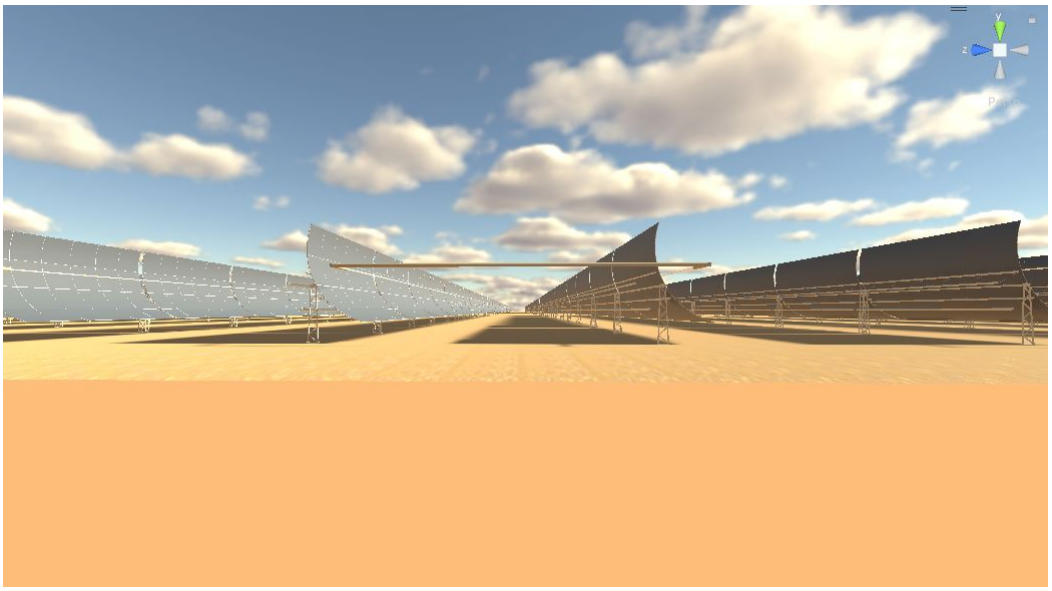


Figura 2.11 Cota 0.

Esta limitación es la que menos sentido físico tiene, pero es necesaria a la hora de modelar, puesto que se ha comprobado que hay veces en las que el dron baja demasiado, y si se descontrola momentáneamente, puede penetrar el terreno, algo que no es lógico. Para ello, si disminuye su cota por debajo de 0 metros en algún momento, es impulsado rápidamente por el supervisor hacia arriba.
Todos los impulsos anteriores se modelan como cambios de la componente vertical de la velocidad.

3 Modelado 3D en Unity

En el presente capítulo se explica todo lo relacionado con el entorno gráfico donde se realiza la simulación del modelo dinámico, exponiéndose así desde la modelización de cada elemento tridimensional, hasta el movimiento de ciertos de ellos.

3.1 Edición y modelo de colector

Como se comentó en la introducción hay varios tipos de colectores:

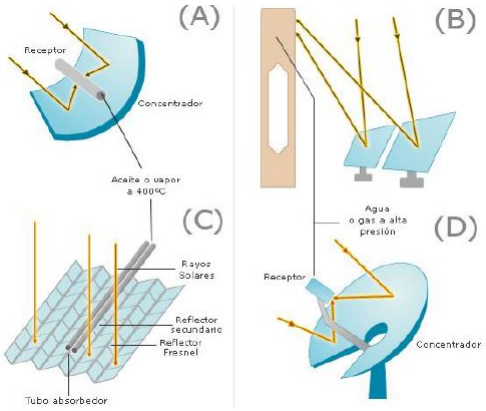


Figura 3.1 (A) son cilindros parabólicos, (B) de receptor central, (C) sistemas lineales fresnel y (D) discos parabólicos [18].

Se emplea el tipo A, de cilindros parabólicos. Este modelo 3D, tras una intensa búsqueda, se encuentra y se descarga. Pero tiene un principal inconveniente.

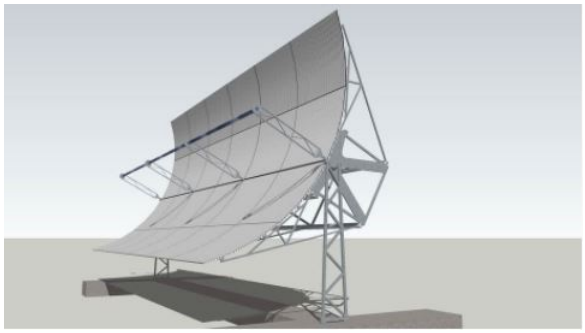


Figura 3.2 Módulo individual de colector [2].

El modelo es de uno de los doce módulos de un colector. Por ello, ha sido necesario una primera edición en el programa *SketchUp Pro 2022*, en el que se han acoplado los 12 módulos que conforman un colector.

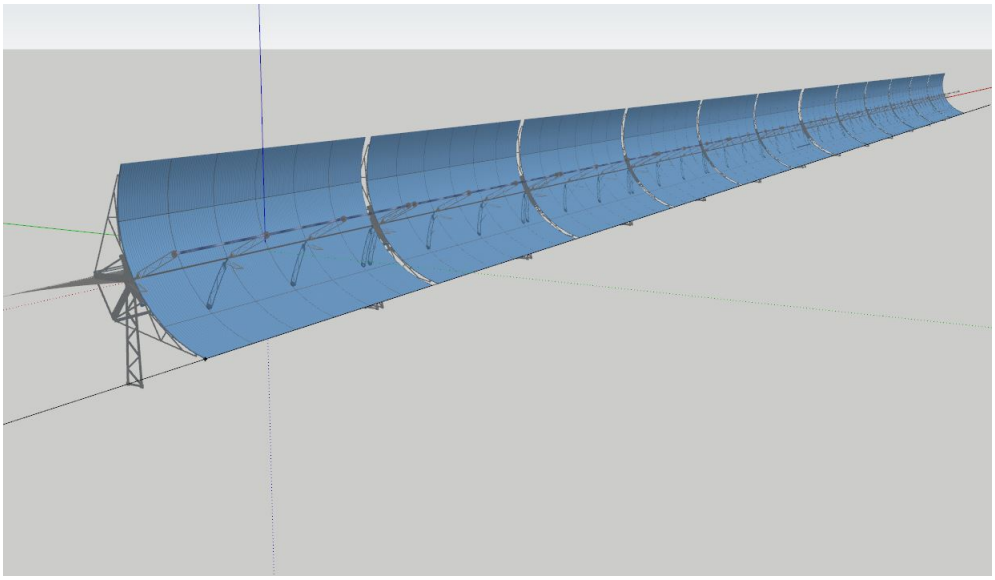


Figura 3.3 Modulos acoplados conformando un colector.

Por conseguir una mejora en los gráficos, se ha suprimido la base cimentada del objeto. Es un elemento que no añadía ningún valor especial, y en vista a la cantidad de colectores que se van a situar, era preferible no exceder las capacidades de la computadora en esta parte. Es más, se ha tenido en cuenta una separación similar a la real, que se ha medido desde Google Earth, entre módulos y en la parte central, que hay una separación especial.
Las dimensiones son las siguientes:

- Apertura de colector: 5,7 metros
- Longitud colector: 148,5 metros
- Separación entre módulos individuales: 0,383 metros
- Separación parte central del colector: 0,847 metros

Estas dos últimas dimensiones son las obtenidas midiendo directamente sobre la planta de *Solnova 4*.
Una vez se tiene modelado el colector, con sus doce módulos, la forma en las que estos se organiza a lo largo del terreno se llama de forma coloquial en "lazos", puesto que vistos en planta, hay una parte en la que el fluido viaja, tubería fría, y otra por la que vuelve, tubería caliente.

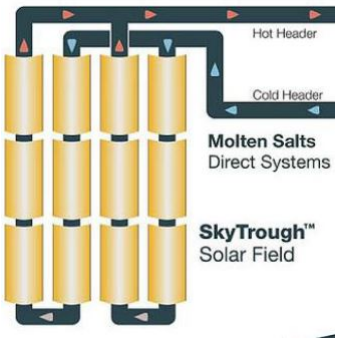


Figura 3.4 Organización de un lazo [18].

Estos lazos están formados por 4 colectores, 2 de ida y 2 de vuelta. El funcionamiento de estos lazos es sencillo: por la rama de colectores de la derecha entra fluido a menor temperatura, siendo bombeado desde la parte central de la planta; este, al pasar por los colectores que reflejan continuamente los rayos que llegan al cilindro parabólico, es calentado, subiendo de temperatura y presión, con el fin de que cuando salga del lazo por el colector 4 lleve más energía con la que entró en el lazo.
Se explicará con posterioridad la procedencia y el destino de este fluido al entrar por un determinado lazo.
En la siguiente figura, se observan los de ida, 1 y 2, y los de vuelta, 3 y 4.

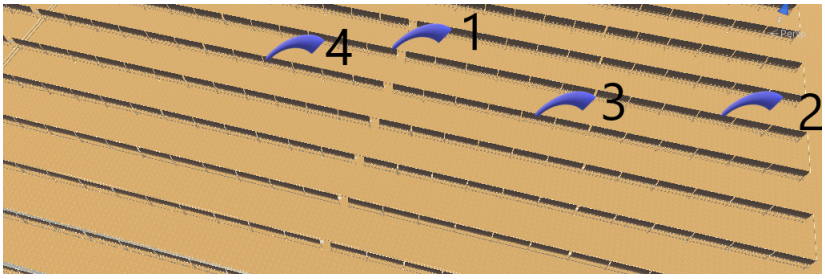


Figura 3.5 Lazo completo.

Estos lazos, se encuentran cerrados por tuberías que se ha tenido que colocar manualmente. Las que unen el tramo 1-2 son cilindros rectos, y las del tramo 2-3, son dos codos con un tubo de mayor longitud que los unifica. Posteriormente, se enseñarán los modelos de tuberías utilizados con mayor cercanía.
Se ha optado por añadir dos modos de visualización: uno en el que se renderiza la estructura que tiene detrás cada colector, que es más detallado pero que baja los fotogramas por segundo con los que se proyecta la simulación, y otro modo sin esta estructura que mejora el rendimiento.

Por último, se comenta un problema asociado a la renderización de los objetos en general. Cuando se probó a activar las estadísticas de rendimiento del mapa en Unity y se ejecuta la simulación, se apreció un problema. Y es que los llamados batches o lotes cargados, llegaban a ser del orden de 50.000, por lo que se procedió a ver qué era lo que ocurría.
Este número se correspondía con el número de entidades cargadas simultáneamente en el entorno Unity, y por ello se continuó a su posterior optimización.

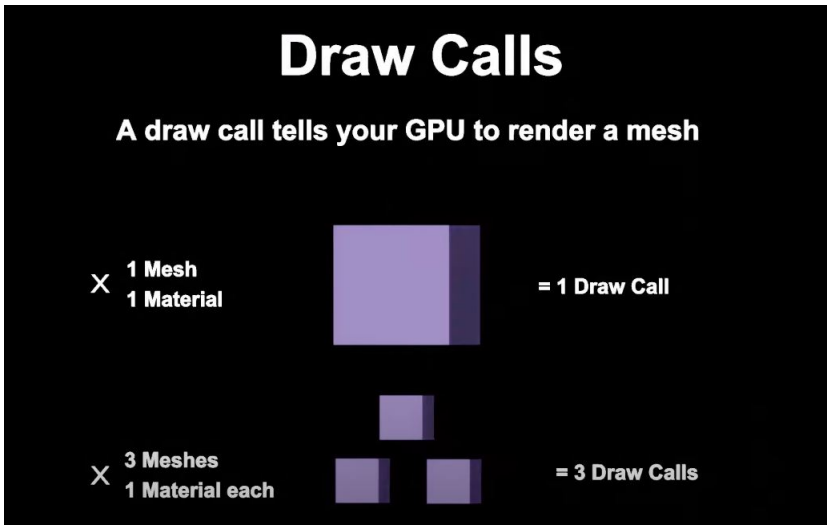


Figura 3.6 Explicación del número de entidades [7].

Unity carga las llamadas *Draw Calls*, por lo que el número de estas son el resultado de multiplicar el número de mallas por los materiales del objeto. Si este número es muy alto, se forma un cuello de botella en la CPU, procesándola a cuentagotas, tardando la GPU más de lo necesario por recibirla más lentamente.
Como solución a este problema, se volvió a entrar en *SketchUp Pro 2022* para reorganizar las mallas ya creadas, y fusionar las que se pudieran.
Resultado final de batches cargados: 3845, siendo una reducción aproximada de un 92 % de entidades cargadas.

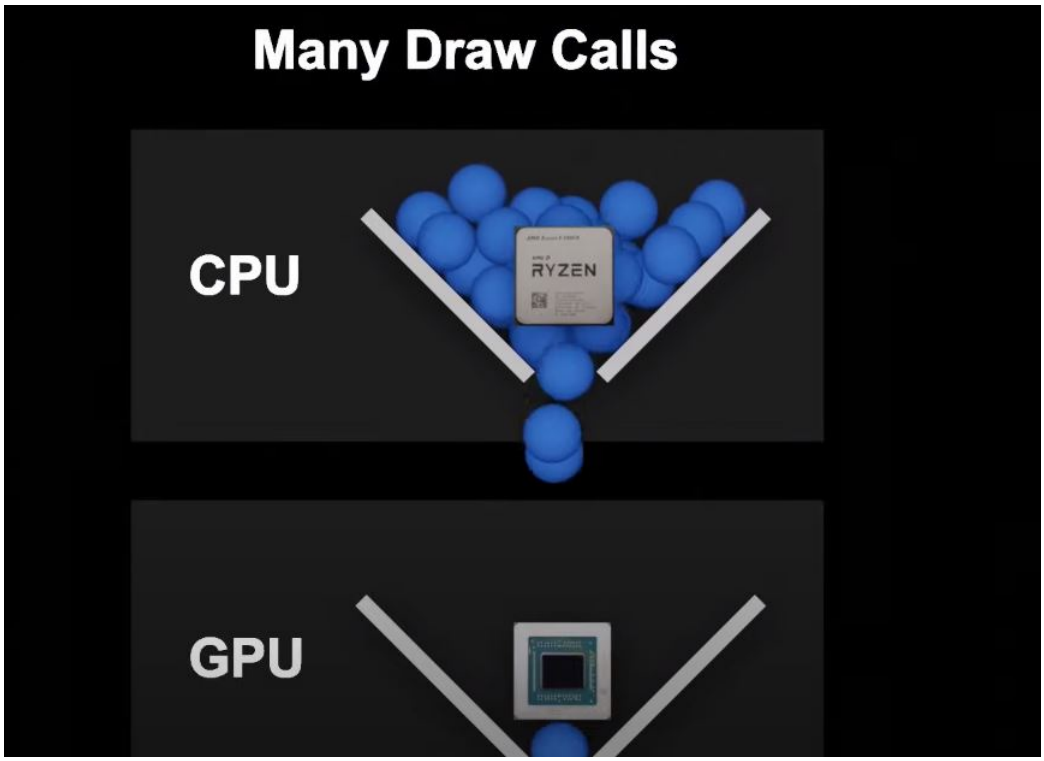


Figura 3.7 Cuello de botella en la CPU [7].

3.2 Edición y modelo de dron

Se ha intentado utilizar el vehículo aéreo no tripulado ya diseñado por *OCONTSOLAR*, del que se disponen de 3 unidades. A continuación, se exponen fotografías del vehículo en la realidad:

Figura 3.8 Dron real utilizado en el proyecto y su correspondiente modelo 3D [3].

Este vehículo tiene unas dimensiones de 45,8 cm de rotor a rotor. El modelo en sí ha sido descargado de la *Unity Asset Store*, en la que se encuentra modelos 3D ya preparados, también llamados *Prefabs*, que se importan rápidamente en la escena Unity. Se le han hecho varios retoques al descargado por defecto:

- Cambio de color
- Inhabilitación de los scripts incorporados por defecto

Primeramente, para que se pareciese al modelo del proyecto, se le cambia el color del cuerpo principal a blanco, y se dejan los rotores en negro.

Y por último, el *prefab* trae consigo una serie de códigos para darle realismo al vehículo, tanto en física de movimientos, como en seguimiento de *waypoints*. Pero no son interesantes, puesto que, al actuarse sobre ellos, a través de Python, mediante traslaciones, estos efectos no pueden ser percibidos, y ralentizarían aún más la simulación de la escena.

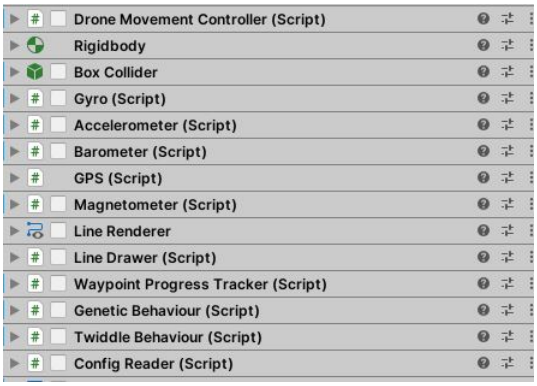


Figura 3.9 Scripts del modelo por defecto.

La única función que sí se utiliza de los scripts que trae por defecto es la activación de la rotación de las hélices, que se le aplica directamente a los 4 rotores, y cuya velocidad de rotación se puede ajustar internamente en el código en función de la potencia disponible del dron. Se han usado, como se ha explicado anteriormente, 3 drones, que se llaman:

- Drone0
- Drone1
- Drone2

Por simplificar, se le han añadido los scripts extras que han sido necesarios al Drone0, por lo que la ausencia de este provocaría un mal funcionamiento de la simulación.
Los scripts citados son: el que recibe continuamente los datos de las posiciones de los vehículos y se encarga de transformarla (Change Pos); el que duplica a los colectores, y los coloca en su determinada posición por todo el escenario (Solar Panel Orig); y el que se encarga de guardar en un archivo .txt la rotación del sol (Save Pos).

3.3 Modelo de la planta solar

Hay varios elementos que constituyen el escenario de la simulación:

- Zona de procesamiento del fluido
- Bases de carga y estacionamiento de los UAV
- Conexiones entre colectores
- Terreno

En las siguientes líneas, se describen con algo más de detalle estas zonas/objetos

3.3.1 Zona de procesamiento del fluido

En estas centrales, el fluido utilizado son las sales fundidas, que almacenan bastante bien la energía. Es un proceso cíclico, en el que el fluido va circulando por los lazos, y se vierten estas sales ya calentadas a una red de tuberías "calientes", que llegan hasta una parte central de la planta, en la que se le extraen la mayor energía posible mediante una transferencia de calor a una gran cantidad de agua, que se evapora, y luego es pasada por una o varias turbinas, que mediante su giro, es convertida dicha energía en energía eléctrica.

Por no saturar demasiado el escenario de objetos inanimados, se ha tomado la decisión de colocar los objetos más voluminosos de *Solnova 4*:

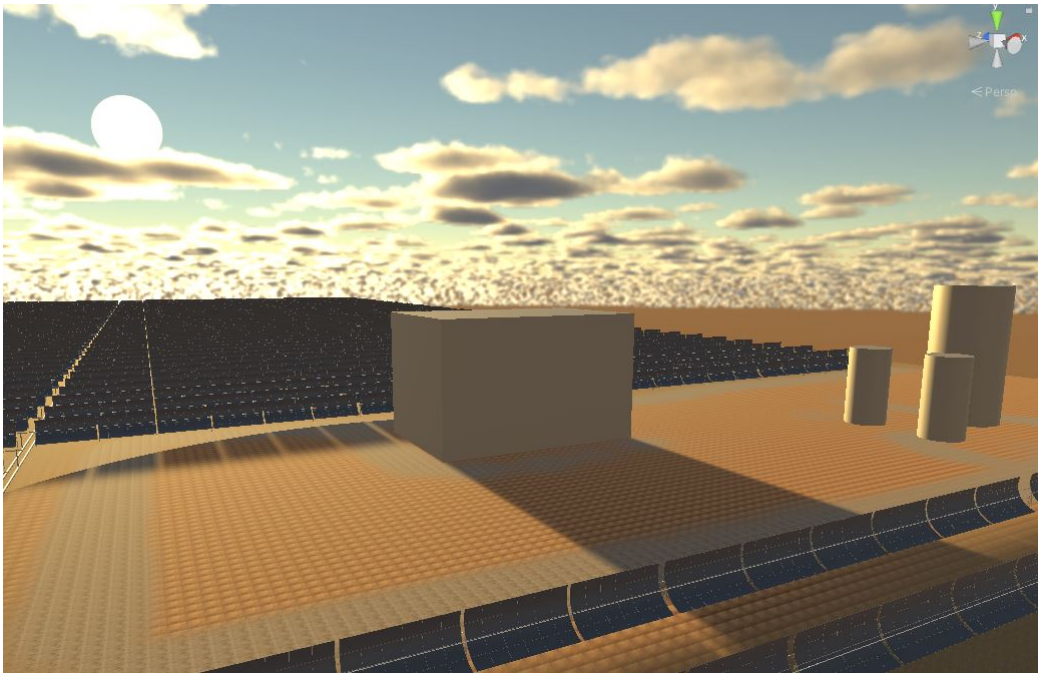


Figura 3.10 Objetos de la zona de procesamiento.

viéndose en la anterior figura el bloque con forma de paralelepípedo que contiene las turbinas y algunos depósitos.

3.3.2 Bases de carga y estacionamiento de los UAV

Otro elemento muy importante que no puede faltar es la base. Se ha seleccionado el típico para los helicópteros, por la similitud de funcionamiento al llevar rotor. Hay tres de ellas en todo el terreno, y de estas parten y es hacia donde se dirigen los vehículos cuando se da alguna de las siguientes condiciones:

- Se terminan todos los objetivos previstos
- Batería del dron inferior a un 15 %
- Se produce el ocaso del sol durante el día

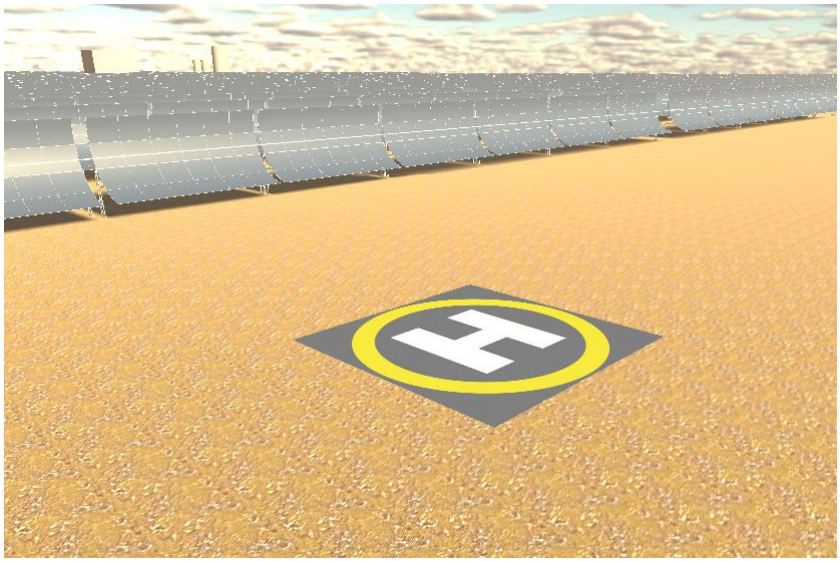


Figura 3.11 Base de carga/estacionamiento [14].

3.3.3 Conexiones entre colectores

Como se ha introducido antes, los colectores forman lazos, por lo que son necesarias tuberías que unan los 4 colectores de un lazo, y otras que unan los lazos a la red central, que son las que abastecen a cada uno de ellos.
Se ha realizado una hipótesis simplificatoria, por la cual todos estos tubos son rígidos y no tienen elementos flexibles, lo que difiere de la realidad. En futuras líneas se comentará más en profundidad.

En la tercera figura se aprecia el detalle de la red de tuberías con sales fundidas frías (la de la izquierda), y las que transportan sales fundidas calientes (la de la derecha), conectándose la primera parte del lazo a la fría y la parte final de este a la caliente. Todas ellas se han colocado a mano, por ahorrar programación, puesto que se tendría que analizar muy bien las posiciones y se tardaría más tiempo.
Volviendo a lo anterior, se pueden dividir los tipos de tuberías en tres:

- Tuberías de continuación
- Tuberías de vuelta
- Red de abastecimiento de lazos

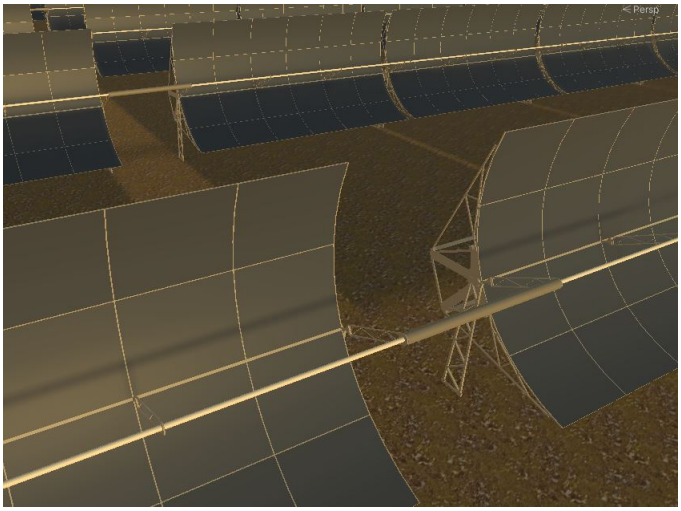


Figura 3.12 Tuberías de continuación.

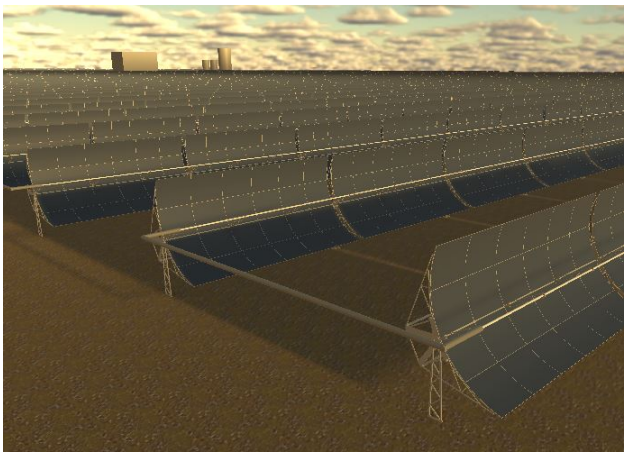


Figura 3.13 Tuberías de vuelta.

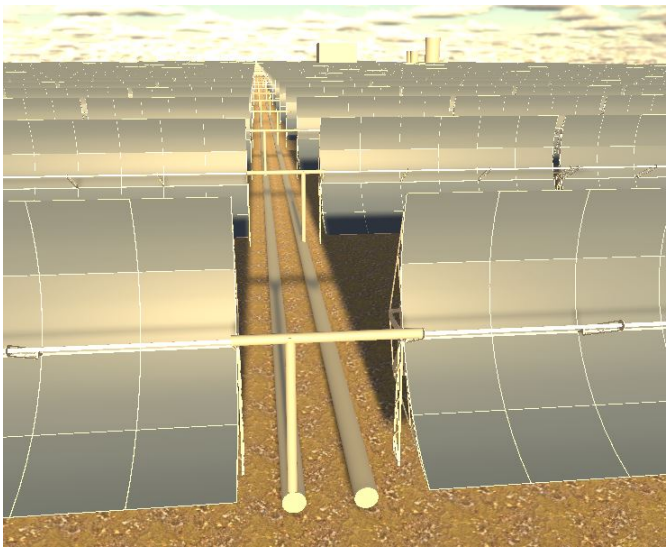


Figura 3.14 Red de abastecimiento de lazos.

3.3.4 Terreno

Se ha seleccionado un terreno rectangular, de 1850 metros de longitud y 640 de ancho, por simplificar. En la realidad, hay unos lazos que se encuentran fuera del rectángulo, visto en la figura de la sección 2.3, por ello que no se han considerado los lazos fuera de este, y se ha realizado una central rectangular. Como resultado, se obtiene la siguiente vista en planta de la central aproximada:

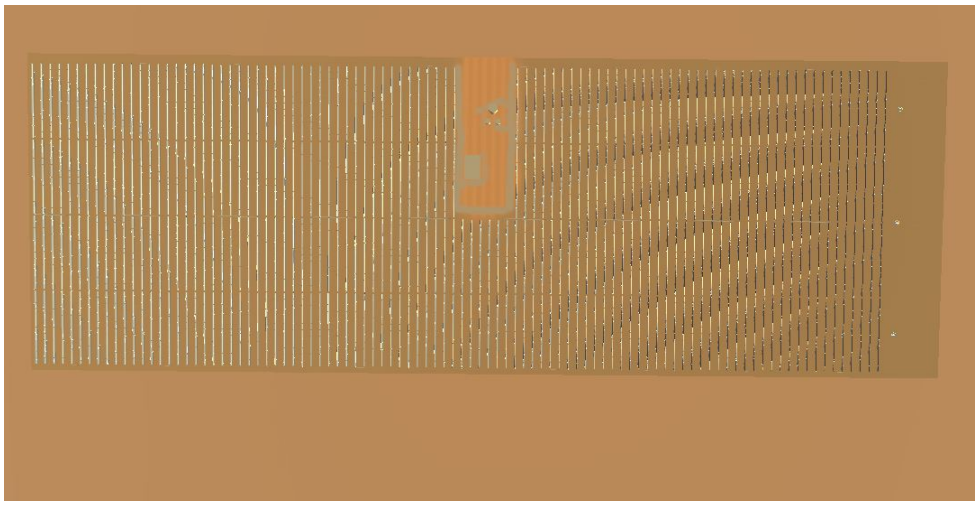


Figura 3.15 Vista en planta de la central simulada.

Para las texturas, he hecho falta la descarga de paquetes de texturas que contuviesen las apropiadas para representar con la mayor similitud el terreno real. Este se ha encontrado, de nuevo, en la *Unity Asset Store*, importándose a la escena:

Figura 3.16 Texturas utilizadas en el terreno.

siendo las dos de la izquierda las utilizadas en la parte central, y la de la derecha la que predomina en el terreno.

En total, hay presentes 93 lazos en la simulación. Sin embargo, *Solnova 4* está basada en 90 lazos. De todos modos, se ha colocado de esta forma por simplificar como se ha explicado, más que nada por evitar añadir demasiada complejidad en el código por una diferencia casi inapreciable de colectores a vista de pájaro.

3.4 Nubes y sol artificiales

Inicialmente, se comenzó el escenario de Unity como uno básico y se trabajó en él creyendo que tendría todas las funcionalidades disponibles. Casi al final de la elaboración del entorno 3D, fue necesario la importación de nubes.
Entonces, se cayó en la necesidad de transformarlo a un proyecto con alta calidad de gráficos (HDRP), para el cual había que convertir todas las texturas a alta resolución, incluidas las de los propios objetos.

Se tuvo que recurrir a esta opción para importar las nubes volumétricas, que es una función que incluye este módulo de Unity. De otra forma, importándolas directamente desde la *Unity Asset Store* no se podrían haber conseguido los posteriores resultados debido a la nula tridimensionalidad.

Estas disponen de múltiples opciones, desde velocidad y dirección hasta densidad. Incluso, existe la posibilidad de añadir mediante un dibujo nubes modificadas de la forma que se quiera. El funcionamiento es perfecto, puesto que cuando se ejecuta la simulación aparecen nubes aleatorizadas que van desplazándose sobre el terreno, con volumen propio, función imprescindible para los posteriores mapas de radiación.

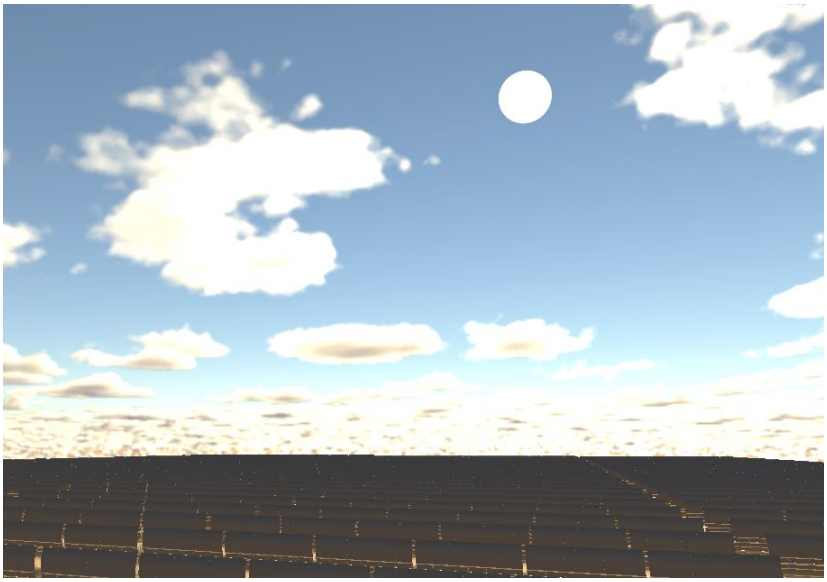


Figura 3.17 Modelo de nubes y sol.

Respecto al modelo solar, se ha incluido la "Luz Direccional" que Unity trae por defecto al instalarse, que es simplemente un objeto situado a la distancia real del sol, con tres parámetros para su manejo: los ángulos alrededor del eje x, y, z ya anteriormente usados. La posición en estos ejes no tiene gran efecto, debido a la distancia a la que se sitúa. Tras cambiar el escenario a tipo HDRP, surgieron nuevas opciones en la renderización solar, que se ajustaron de la mejor forma posible.

Queda por ver como se mueve el sol sobre el "cielo" visto desde el plano de trabajo.

Se encontró en la página web de *GitHub*, que es muy útil para encontrar ciertas soluciones a un código de Python, una combinación de 3 scripts. Estos se encargan de rotar una luz direccional de Unity, basado en la localización deseada, a través de longitud y latitud, y el tiempo.

Estos tres códigos, implementados todos en el lenguaje que entiende Unity (*C Sharp*), son los siguientes [11]:

- *SetDate.cs*
- *SetSunLocation.cs*
- *Sun.cs*

Todos ellos se aplican al objeto *Sun* en Unity, para que se ejecuten una vez comienza la simulación.

El primero se encarga de establecer la fecha y hora actual. El segundo ejecuta la localización solar en tiempo real.
Y por último, el script más relevante, que tiene implementadas las ecuaciones de la mecánica orbital, puesto que dados una latitud y longitud es capaz de calcular el acimut, que es el ángulo respecto al norte geográfico, y la elevación, que es el ángulo que se mide tomando como referencia el horizonte, hasta llegar a 90° en la parte más alta.

Con la época y los elementos orbitales ya calculados, el script es capaz de proporcionar θ_x, θ_y, θ_z del sol como variables de salida.

En particular, se ha considerado la dirección y sentido positivo del eje x en Unity como norte geográfico y $\phi = 37{,}42°$ y $\lambda = -6{,}28°$, latitud y longitud respectivamente.

Un gran problema encontrado al pasar a HDRP el entorno fue el siguiente: el sol producía sombra al ejecutar el script debido al refresco continuo de las sombras. Para ello, se puso la tasa de refresco del movimiento solar cada 10000 fotogramas, y se introdujo una modificación en uno de los códigos en el cual se actualiza continuamente la luminosidad solar con el fin de que se actualicé el cuerpo, y no se quede el entorno sin luminosidad.

Lo único que habría que hacer es desactivar y activar estos scripts al iniciar la animación, para así tenerse en cuenta.

3.5 Rotación de colectores

Uno de los efectos más importante que se pretendía ver en la simulación 3D es la del movimiento de los colectores, aparte del movimiento de los vehículos aéreos. Sin embargo, se ha obviado el transitorio en el movimiento debido a la forma en la que está diseñado, y se dejará para futuras líneas de trabajo. Para ello, se ha usado el mismo script que mueve los drones *ChangePos.cs*, en el que posteriormente se entrará en más detalle.

El proceso que se sigue es bastante sencillo: si un dron concluye con éxito una tarea, se manda el índice del colector medido a Unity desde Python. Al detectarse en el script que se ha finalizado una misión, se selecciona el colector en cuestión como objeto, se miran la componente x de rotación del sol, que si fuese mayor a 90° se operaría con su complementario, y finalmente se saca el ángulo a rotar el panel.

Este ángulo debe ser la resta al ángulo solar del ángulo que tiene actualmente el colector. Además, se le restan 28° extras.
Dicho desfase es debido a la forma en la que está modelado el colector, por lo que es corregido con bastante precisión de esta forma.

Este proceso se realiza para cada uno de los *Child* del objeto, que son subobjetos que componen el objeto principal. Esto se realiza de esta forma puesto que si se rotara el objeto principal al completo, el eje no estaría centrado y la estructura metálica que soporta el peso del colector también rotaría.

Terminada la rotación, se guarda en una variable auxiliar el índice del colector que se ha rotado, para evitar rotaciones de nuevo, problema que surgió durante la implementación de esta función.

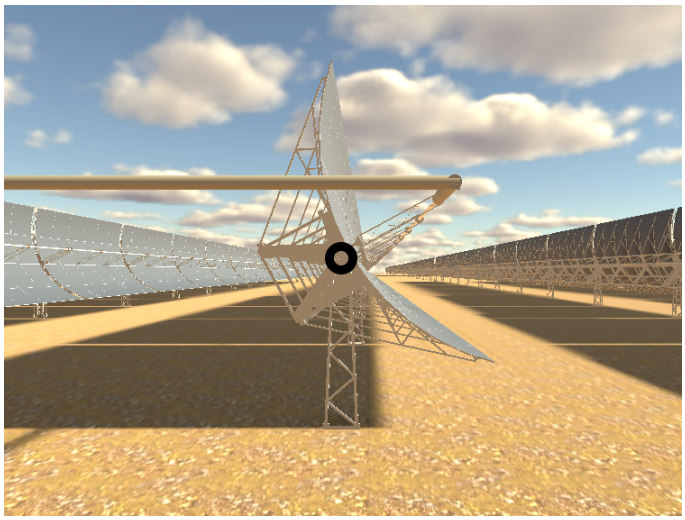


Figura 3.18 Localización eje de rotación.

4 Conexión Python-Unity

A partir de ahora es interesante poner en relación algunas de las funcionalidades anteriormente explicadas con su funcionamiento e implementación.

El propio control de la flota, como ejemplo, se hace desde Python, pero es necesario un puente hasta Unity de alguna forma, para que sea posible la obtención de datos de posiciones por parte de Unity.
Se proceden a explicar las dos casuísticas encontradas en la transmisión de datos.

4.1 Flujo de datos de Python a Unity

El principal problema del trabajo, en general, ha sido la conexión entre estas dos plataformas. Para ello, se encontró una forma, los *Sockets*.
Socket es un concepto abstracto, orientado a la conexión, para que haya flujo de datos. Transmite por octetos, garantizándose que se hace sin errores ni omisiones, además de asegurarse de que todo octeto de información llegará a su destino en el mismo orden en el que fue enviado [19].

La comunicación en este sentido se realiza para:

- Transmitir a Unity la posición de los drones
- Transmitir a Unity el índice de los colectores a los cuales se le haya realizado la medición

Es importante la conexión a internet durante el desarrollo de la simulación, puesto los datos se transmiten a través del puerto 1755 de la IPv4, para establecer la red privada en la computadora.
Los datos son codificados en forma de cadenas de caracteres, para posteriormente ser enviados por la función *sendPos*, que se encuentra en el archivo de Python que ejecuta la cinemática. Esta es llamada después de calcularse las posiciones de todos los drones en un instante de tiempo posterior. Al final la cadena se le añade el índice del colector.

4.1.1 Posición de los UAV

Para la posición de los drones, se utiliza una cadena en el que los caracteres están separados por comas, e implican las posiciones de los drones de la siguiente forma:

$$string1 = [x_0,z_0,y_0,x_1,z_1,y_1,x_2,z_2,y_2] \tag{4.1}$$

indicando el número al dron al que pertenecen las coordenadas. Como se percibe, están cambiadas las z e y de posición. Esto es porque en Python se ha considerado la z como la cota sobre el suelo, a diferencia de Unity, que considera la y como la cota sobre el suelo.

Cuando se llama a la función sendPos, esta abre un socket a través del ya explicado puerto permitiendo la conexión, se embarcan los datos en la cadena, se envían y se cierra la conexión.
Este proceso se repite cada vez que se quieren enviar nuevos datos a Unity.

Luego, existe otra función que recibe estos datos, que se encarga de interpretarlos y suministrarlos a las casillas de los drones. Primeramente, se optó por únicamente convertir los caracteres de la cadena a números con formato *float* (este almacenamiento admite decimales).
Pero, al probarse, los valores que adquirían los cajetines de las posiciones de los drones no tenían sentido. Entonces, se comprobó que el número en sí era el correcto, pero que Unity interpretaba mal el tener tantos decimales por los que les quitaba la coma decimal, y lo interpretaba como un valor muy grande.

Para resolverlo se llevó a cabo una modificación en el código de *ChangePos*, que es el que recibe la información en Unity, de tal forma que, además de convertir las cadenas de texto a flotante, se busca la posición del punto decimal, y teniendo en cuenta la longitud del número en particular como cadena, se realizan unos ajustes para introducirlo correctamente con formato de notación científica, leyéndolo al fin bien por el lenguaje de *C sharp*, y teniendo coherencia con la transmitida y calculada en Python.

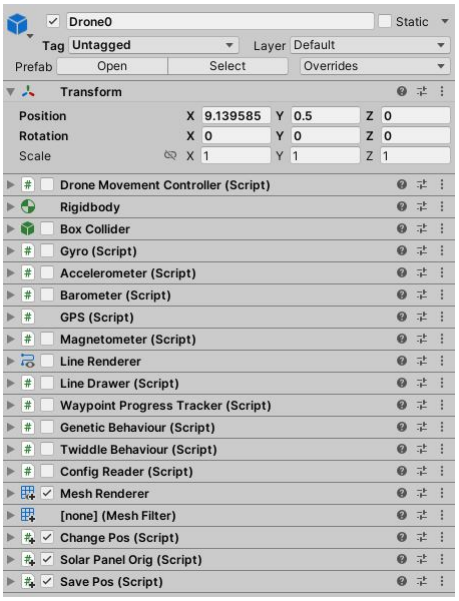


Figura 4.1 Valores modificables del UAV.

4.1.2 Índice de los colectores

Por otro lado, esta función *sendPos* se aprovecha para cuando se cumple una tarea determinada, es decir, la medición sobre un colector. Se le añade un carácter a la cadena anterior, j, que suele tomar un valor arbitrario en cada iteración, -1, pero que toma el valor del índice del colector medido en caso de completarse la tarea.

$$string2 = [j] \tag{4.2}$$

quedando como resultado de juntar ambas funcionalidades en la misma cadena de la siguiente forma:

$$string = [x_0,z_0,y_0,x_1,z_1,y_1,x_2,z_2,y_2,j] \tag{4.3}$$

Este valor j se compara con la ayuda de una variable auxiliar, porque hay veces en el que se repite el proceso, pasándose varias veces consecutivas el índice del mismo colector.

Una vez se tiene, se sigue el procedimiento descrito en *Rotación de colectores*

4.2 Flujo de datos de Unity a Python

Tan importante es la transmisión de datos de Python a Unity, como de Unity a Python para que haya establecida una comunicación plena, teniéndose en cuenta todas las posibles variaciones en el modelo.

Sin embargo, en esta ocasión no se realizará mediante socket, para no ralentizar aún más el proceso por diversas conexiones simultáneas. El método será mucho más simple: los datos que se quieran transmitir se pasarán como cadena de texto a un archivo con formato *.txt*, y será Python el que lea directamente de ahí los datos requeridos, con la frecuencia que se indique. Hay dos causas para llevarlo a cabo:

- Almacenar intensidad de radiación del mapa
- Almacenar la posición solar

Python intenta acceder a la intensidad de radiación solar cada 10 minutos, suponiendo que se obtengan todos los valores en un tiempo menor a este. El fichero que contiene esta información se llama *lightdetectordata.txt* y esta funcionalidad se explicará con posterioridad en mapas de radiación.

Por otro lado, ya se ha comentado el uso de la posición solar (rotación de colectores), guardándose estos valores angulares en el fichero *positionFile.txt* cada dos minutos y medio, para que en el archivo de la cinemática de Python se actualice cada dicho tiempo fijado.
Por simplicidad, se ha elegido almacenar y leer los datos por líneas, para evitar separarlos por comas e introducir más complejidad de la que merece.

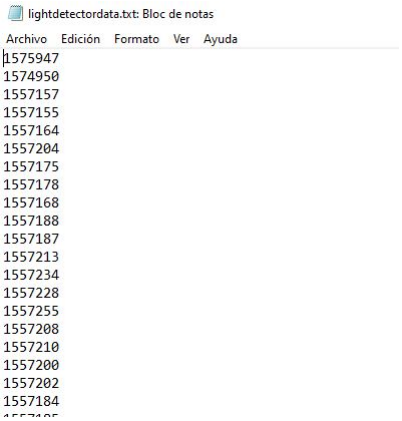


Figura 4.2 Forma de almacenar datos en ficheros *.txt*.

4.3 Extracción de dirección IP automática

Como ya se expuso anteriormente, es necesario conectarse a internet para poderse llevar a cabo la conexión del socket de Python a Unity. Por ello, se requiere el conocimiento de la *IPv4* que se muestra en la consola del ordenador al teclear el comando *ipconfig*.

Pero este proceso es automatizable, para así no tenerse que editar el código cada vez que se cambie de red. Se debe importar en Python la librería *socket*, para así, con una función que trae embebida, obtener el nombre de la red, y con este nombre obtenerse la ip asignada. El script de Unity que recibe la información también debe conocer la ip y puerto (este último fijo: 1755). El proceso se realiza de forma similar al de Python, pero con las respectivas librerías en *C Sharp*.

Por otro lado, se ha automatizado la obtención de la ruta de los archivos, puesto que cada ordenador va a almacenar los códigos en distintas carpetas, y es muy improbable de que coincidan alguna vez.

5 Mapas de radiación del entorno

5.1 Utilidad

Otra funcionalidad bastante importante que se quería reflejar en la simulación era la capacidad de poder sacar unos planos de radiación, que indicarán la disminución de radiación sobre colectores debido a la presencia de nubosidad.
Esta perturbación climatológica actúa muy negativamente sobre el rendimiento de las centrales solares en sí, de cualquier tipo.

Tanto es este efecto, que es necesario llevar a cabo técnicas de mapeado continuamente para saber en que puntos de la central hay mayor eficiencia y en cuales menos.

Con ello, lo que se persigue es poder actuar sobre la planta de alguna forma para disminuir ineficiencias. En el caso de una central fotovoltaica, la presencia de obstáculos en la radiación no se podría solventar con facilidad, no sirviendo posibles reorientaciones de placas.

En cambio, para el tipo de planta en cuestión, puede aumentarse el rendimiento con la ayuda de estos mapas, esto es, es posible actuar sobre los colectores para prevenir un desastre mayor.

Los dos principales problemas en estas centrales son:

- La suciedad sobre los espejos reflectores
- La nubosidad

Ocurre que cuando el fluido atraviesa las zonas de la central con peor rendimiento debido a una de estas dos causas, este se enfría de forma que penaliza la cantidad de energía que llevan los tubos.

No tendría sentido que una parte de la planta aumentara la presión y temperatura de las sales fundidas, y otra parte de esta se encargara de liberar dicha energía asociada a la temperatura y presión ganada. Es por esto por lo que esta central cuenta con una especie de sistema que abre y cierra válvulas, para que el fluido entre o no por lazos que tengan mayor o menor eficiencia, utilizadas para el equilibrado hidráulico.

Con esta solución de redistribución de caudal se pueden aprovechar todos los lazos que funcionen bien durante un determinado instante, y no se pierda demasiada energía.
Es esta la explicación de porque es interesante implementar dicha funcionalidad en una simulación bastante realista.

Figura 5.1 Nubes en una de las plantas de *Solnova*.

5.2 Medición de radiación

Tras las explicación de como se han implementado y modelado las nubes volumétricas, tiene sentido sacarle algún partido. Y este es el de, para unas nubes dadas, obtener el mapa de radiación para ver su influencia sobre los colectores.
Esta idea que a priori parece fácilmente implementable, pero ha requerido llevar a cabo un proceso innovador para conseguirlo.
Aunque es algo lento, y su velocidad depende completamente de los fotogramas por segundo a la que corre la simulación, es bastante efectivo.

En la industria de videojuegos, se suele emplear un método para evaluar la intensidad de luz que le llega al jugador, para determinar ciertas acciones al respecto. Se analiza el cambio de radiación sobre la textura del jugador.

Con esto ya bien desarrollado, surge la idea de, sí se tiene la intensidad de luz sobre un cuerpo, en este caso una cápsula, que es un objeto propio de Unity (categorizado como básico), y se traslada por todo el terreno, midiendo la radiación cada vez que se mueve, se puede obtener como resultado una base de datos con la luminosidad de todos los puntos del entorno gráfico. Es un proceso que puede tardar entre 5 y 10 minutos, dependiendo de los objetos que se estén cargando en determinado momento. Consta de la cápsula, que emula a un jugador, y de un plano sobre el que se proyecta la textura del objeto, con el que se evalúa dicho nivel.

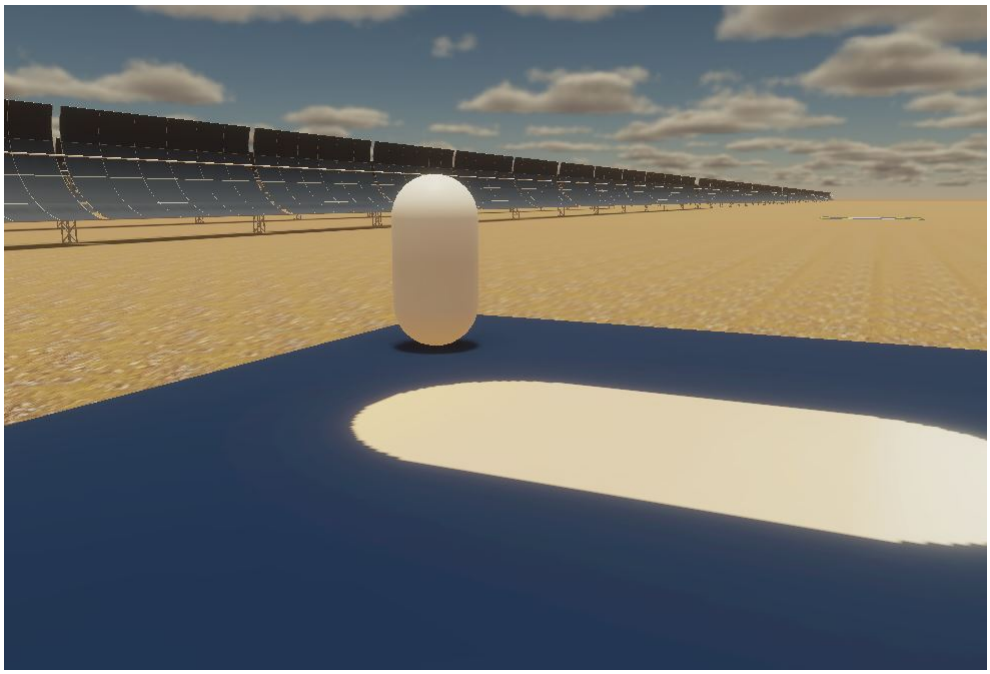


Figura 5.2 Detector de radiación (Cápsula y plano) [15].

Más concretamente, se inicializa un vector de 20.000 componentes en Unity donde se va a almacenar la información en el *.txt*. La posición de la cápsula del plano varían de la siguiente forma:

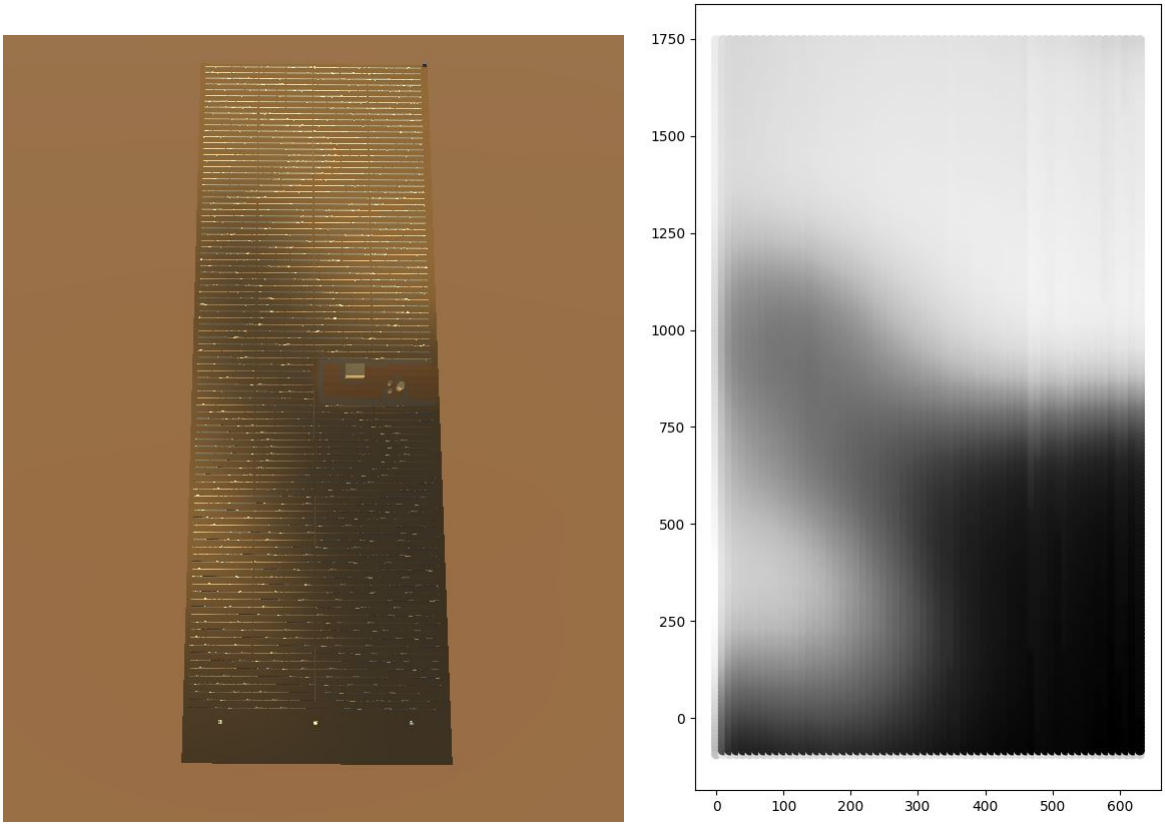
- La x varía entre 0 y 640 metros de 10 en 10
- Para cada x, la z varía entre -93 y 1757 metros de 10 en 10
- Se realiza a una altura de 14 metros

Esta elección se ha hecho de tal forma que se recorra el terreno con la mayor precisión posible, pero no excediendo el tiempo demasiado, puesto que el cuello de botella en este proceso se encuentra en lo que tarda en trasladarse cápsula y plano, y en cargar la textura.

Un dato de interés es que los colores se normalizan según el máximo de radiación en todo el entorno y el mínimo, en una escala de grises, significado el color negro el lugar donde se produce sombra, y blanco el lugar al que la radiación es capaz de llegar sin obstaculizarse. A continuación se exponen varios ejemplos.

5.2.1 Ejemplo mapa de radiación con velocidad despreciable del viento

Para esta prueba, se ha fijado al 0,1 % la velocidad de las nubes, encontrándose el cuerpo celeste a 10 grados de elevación sobre el horizonte ($\theta_x = 10$). El grupo de nubes tiene como altura mínima 2 km.



(a) Vista en planta del terreno con sombras.

(b) Correspondiente mapa de radiación sin viento.

Figura 5.3 Mapa de radiación sin viento.

En estas figura anterior se puede observar como queda plasmado, en una gráfica, la radiación que incide sobre el terreno, incluso con el degradado que supone que la parte central de una nube sea más opaca que en los bordes de estas, que cuando se tiende a este, la radiación tiende a la del sol sin perturbaciones adicionales, en ese momento.

De momento se ha implementado con la normalización entre el mínimo y máximo, pero queda para futuras mejoras un estudio más exhaustivo, puesto que requiere de un análisis de que intensidad se considera mínima, relacionadas seguramente con días de nubes muy densas, y que se considera máxima, que serán los días del año en los que recibamos más potencia irradiada por parte del sol.

5.2.2 Ejemplo mapa de radiación con velocidad no despreciable del viento

Para este caso se ha fijado al 50 % la velocidad de las nubes, los 10 grados se mantienen. En la figura, se

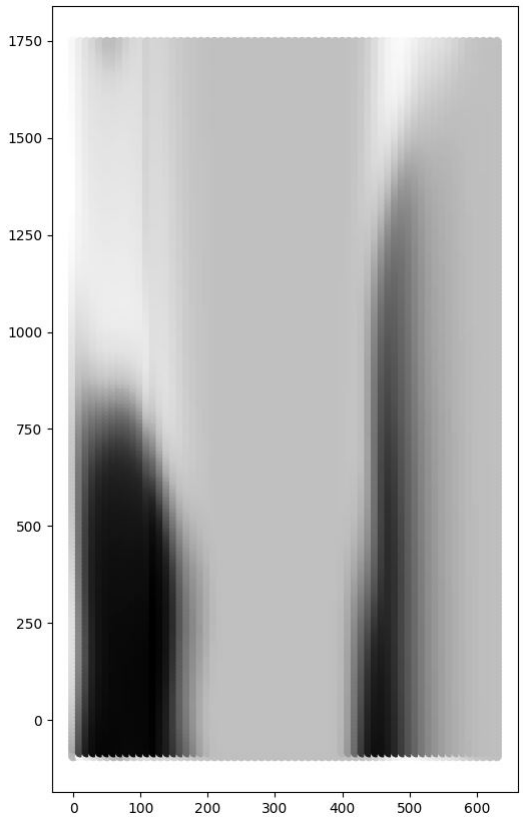


Figura 5.4 Mapa de radiación con viento.

comprueba el efecto del viento sobre las nubes, tomándose un mapa, por así llamarlo, "movido" de la realidad. Es por ello que la forma que tome este depende fundamentalmente de la velocidad del viento, y de su dirección (en este caso tomada en dirección sur, o es lo mismo, con un ángulo de 180°).

6 Limitaciones del modelo dinámico

Aunque sería conveniente que todo funcionara siempre, hay que asumir que a la hora de realizar un modelo sobre cualquier comportamiento, se están cometiendo errores, puesto que el hecho de modelar lleva implícito el realizar hipótesis simplificatorias, ya que la realidad es mucho más compleja e impredecible que lo que llevemos a cabo los seres humanos. Por ello, se va a comentar los extremos encontrados en el modelo en sí.

6.1 Limitación en espacio

Inicialmente, se empezó el modelo con una serie de puntos en Python. Estos se representaron en una espacio tridimensional de 100x100x100 metros para programar las primeras funciones básicas.

Se inició todo con un punto azul, que debía aproximarse hasta otro rojo que hizo de objetivo a cumplir, como un simple *waypoint*. Luego se fue implementando una mayor complejidad en cuanto a posibles casos:

- Un mismo número de drones y misiones
- Mayor número de drones que de misiones
- Menor número de drones que de misiones

Lo complicado fueron los dos últimos puntos, puesto que suponía descartar algunos drones o tareas respectivamente.

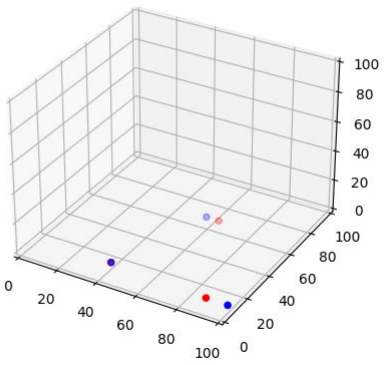


Figura 6.1 Representación a los inicios del modelo.

En la anterior figura se observan 3 vehículos interceptando sus respectivas misiones.

A continuación, este modelo es el que se lleva a Unity, pero antes es aprovechado para ver como de bien es capaz de interceptar las colisiones.

Se pone a prueba y se llega a la cifra de 40 drones/10.000 m^2 (o 100x100 m), esto es, el supervisor consigue evitar las colisiones entre drones en esta superficie partiendo todos de una distancia mínima con otro vehículo no menor a 5 metros.

El resultado mejora en tiempo de finalización de tareas conforme la altura de las misiones es mayor, todas al mismo nivel. Esto se debe a que las trayectorias se encuentran algo más separadas, y no se frenan tanto por evitar posibles choques.

Tabla 6.1 Resultados obtenidos tras varias pruebas con 40 drones a 10 metros de altura.

Ensayo nº	Tiempo, en segundos, en completarse 40 tareas (1 por dron)
1	194
2	146
3	145
4	183
5	144
6	220
7	153

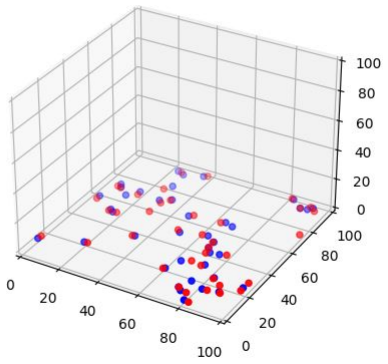


Figura 6.2 40 vehículos para 40 tareas.

6.2 Limitación en tiempo

Respecto al tiempo, se comenzó pasándole a Unity posición a posición de cada vehículo, pero esto tenía un inconveniente muy grande para la simulación: esta iba a saltos, ya que existe un lapso de tiempo entre que establece la conexión, envía los datos y cierra la conexión.

Entonces, la única opción que quedaba era mandar todas las posiciones de todos los drones, calculadas en un instante, a la vez. Con esto se conseguía que tras hacer una pasada por todos los UAVs, todos fueran actualizados en el entorno gráfico, con menor efecto de teletransportación. Por ello, se llegó a la solución, ya explicada antes, de mandar las 9 coordenadas.

Por mucho más que se quiera acelerar el proceso, no es posible porque ya dependería integramente de la velocidad de Python, o bien que realizase los cálculos de propagación de trayectorias más rapido con mayor capacidad de computo, u optimizar el código para disminuir los retrasos.

Un aspecto importante que no se había comentado hasta el momento, es que la simulación se realiza basándose en el tiempo real, es decir, para un nuevo cálculo de posiciones, y demás variables, se tiene en cuenta el instante anterior como referencia, y el difirencial de tiempo que se aplica es simplemente el llamado *timestamp* actual menos el anterior. Estos *timestamps* se usan como una forma de hacer referencia al tiempo, puesto que es resultado de un amplio consenso internacional.

Se logra, con ello, desacoplar la capacidad de computación del ordenador en concreto donde se esté ejecutando dicha cinemática.

Figura 6.3 Representación real de la flota de 3 vehículos.

6.3 Algunos valores de interés

En esta breve sección se aportan valores típicos trabajo:

- Diferencial de tiempo medio: 0,08 segundos
- Velocidad máxima del vehículo: 7,5 m/s con posible +2 m/s de error
- Tiempo medio en completar una tarea: 27,46 seg/tarea (35 min y 15 seg en completar 77 misiones)
- Estimación de tiempo en completar todas las misiones: 2 horas y 50 minutos
- Tiempo medio que se tarda en maniobrar para evitar colisión: 7 segundos

Respecto a la batería, se ha empleado un modelo que tiene en cuenta el *hovering* y el avance.

$$Bat_t = Bat_{t-1} \cdot (1 - \frac{V \cdot \Delta t}{1800 \cdot V_{max}} - 0.00001) \tag{6.1}$$

Se ha ideado este modelo con el fin de que la batería dure unos 30 minutos aproximadamente, a máxima velocidad. Se considera además el efecto que pueda tener el estar sobrevolando un mismo punto, con un término constante, todo ello como consecuencia de estar consumiendo energía aunque no haya desplazamiento.

Estas se recargarían una vez agotadas durante la operación, y durante la noche.

6.4 Instrucciones para operar simulación

Es conveniente explicar la forma en la que se opera la simulación para poder llevarla a cabo, por cualquier persona, y en cualquier computadora. Una vez se tienen todos los archivos descargados y extraídos del *.rar* en una carpeta deseada, se deben instalar las librerías indicadas en el comienzo de cada código de Python:

```
1   import socket
2   import random as rd
3   import time
4   from matplotlib import pyplot as plt
5   from matplotlib.pyplot import figure
6   from mpl_toolkits.mplot3d import Axes3D
7   from dron2 import dron
8   from superv2 import superv
9   import copy
10  import math
11  import os
12  import socket
13  import pathlib
14  from sklearn.cluster import AgglomerativeClustering
```

Figura 6.4 Librerías de Python.

Luego es conveniente abrir el escenario en Unity. Este archivo se encuentra en *simulador-flota-drones/Assets/flota.unity*, y se debe abrir con Unity, versión 2021.2.14.

Antes de ejecutar ningún script, es necesario preguntarse si se quiere mayor velocidad de fotogramas o menos. Para ello, se debe habilitar o deshabilitar el *Group 3* del objeto *Parabollic+Collector*. Habilitado, provocaría más caídas en los fotogramas por segundo, pero una visión más compleja y realista de los colectores:

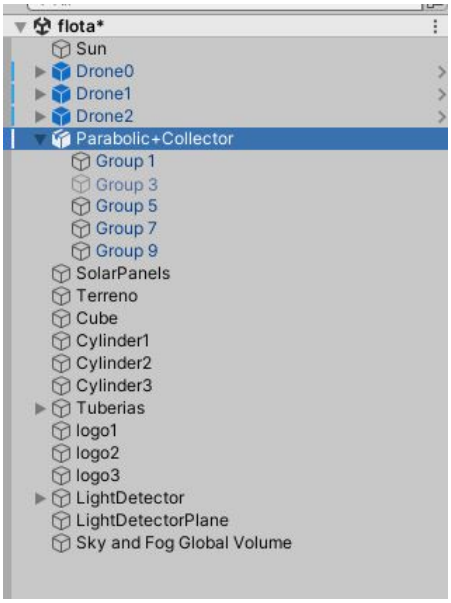


Figura 6.5 Opción de activación del Grupo 3 del colector.

Luego, el siguiente paso es ejecutar la simulación en Unity, que es la que pone en funcionamiento los scripts en lenguaje *C Sharp*, que interactúan con los objetos del escenario. Entrando en el escenario de Unity (abrir *flota.unity*), se debe clickear en el siguiente botón azul de reproducir de la parte superior:

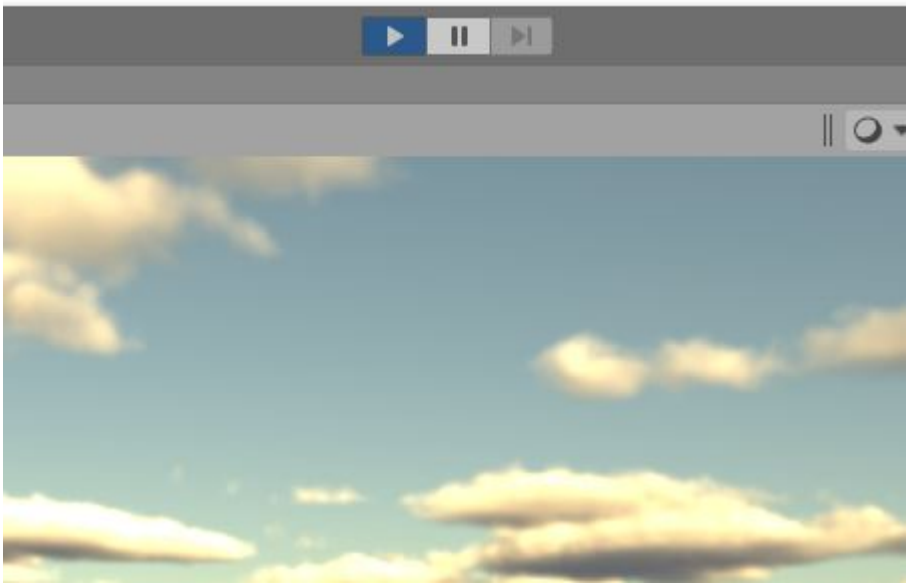


Figura 6.6 Activación del escenario Unity.

Por otro lado, se ponen en marcha los scripts de Python: está el *cinemat.py*, *dron.py* y *supervisor.py*. El único que se debe ejecutar es el *cinemat.py*, porque los otros dos son llamados por este último.

Hay dos formas para poner en marcha los archivos *.py*:

- Ejecutarlo directamente desde la consola
- Tener un editor de texto como *Visual Studio Code* que lo ejecute sin tener que pensar en comandos

En el primer caso, se expone a continuación como se realizaría:

```
Microsoft Windows [Versión 10.0.19044.1586]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\usuario>cd C:\Users\usuario\OneDrive - UNIVERSIDAD DE SEVILLA\4º GIA\TFG

C:\Users\usuario\OneDrive - UNIVERSIDAD DE SEVILLA\4º GIA\TFG>py cinemat.py
```

Figura 6.7 Comandos utilizados para correr la simulación.

que, generalmente, siguen el siguiente orden:

- Se cambia el directorio con: cd (ruta de la carpeta que contiene los 3 scripts de Python)
- Se ejecuta el código en cuestión: py cinemat.py

Una vez se pone en marcha, saldran tres opciones a elegir con *True* o *False* que permiten la visualización de ciertas funciones, u ocultarlas. Es la consecución de estos pasos lo que ya debería permitir a los vehículos ir moviéndose por el terreno e ir realizando las misiones. Faltaría una opción, por activar o desactivar según se desee:

hay una pestaña en el objeto *Sun*, que es el sol, llamada Activar Mapeado. Esta cajetilla se debe marcar durante la simulación si se quiere un barrido por todo el terreno para almacenar los valores de intensidad de radiación. Una vez activada, dejará de estarlo una vez se haga una pasada.

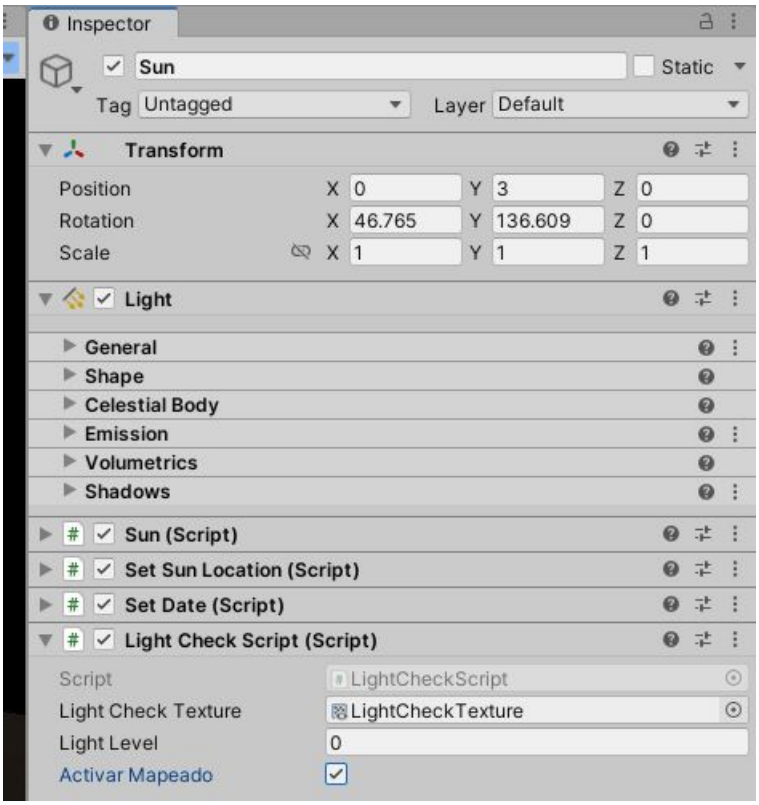


Figura 6.8 Casilla a marcar para activar recogida de datos de radiación.

Si se quiere frenar la simulación en cualquier instante, es suficiente con desmarcar el botón anteriormente pulsado para reproducir la simulación. Con ello, el socket dejará de funcionar automáticamente y los vehículos se detendrán donde estuvieren.

7 Líneas futuras de trabajo y conclusiones

Revisando el trabajo ya realizado, en general, se encuentran varias posibles mejoras, tanto a nivel gráfico, como de código, pero se escapan de los límites buscados en esta primera aproximación de una *Simulación dinámica de una flota de drones*. Aun así, se proponen para futuras líneas de investigación.

7.1 Optimización gráfica en Unity

Uno de los aspectos más importantes a mejorar es el rendimiento gráfico de la simulación. A este le queda un gran trabajo, pero, para ello, se debería asimilar más conocimientos sobre videojuegos, que es el campo que tiene como principal objetivo este tipo de optimización.

Posibles mejoras encontradas a simple vista:

- Mejoras en el modelado de los colectores, pues estos son los elementos principales en el escenario y consumen bastantes recursos
- Depurado del código de transmisión de datos, o incluso, convertir todos los códigos de Python a C#

Con estos cambios, se podría avanzar en la mejora de fotogramas por segundo, y consecuentemente, el rendimiento general de la simulación.

7.2 Transitorio en seguimiento solar

A la hora de que los colectores deban rotar, estos lo hacen de forma casi instantánea. Se intentó conseguir un transitorio, pero en C# no ha sido posible, ya que se probó con bucles, definiendo movimientos incrementales, pero, una vez se entraba en el bucle, también se producía muy rápidamente, no sirviendo en este caso.

Se deja para futuras modificaciones, siendo esta la más probable de solucionar, a priori, sin demasiada dificultad.

7.3 Simulación con velocidad incrementada

Otro reto es la posibilidad de fijar una velocidad a la que se desarrolla la simulación. Este es más complicado, puesto que habría que sincronizar a la velocidad dada tanto el código de la simulación, como en los elementos externos, tales como el sol y las nubes.

En el momento de redactar esta parte, se piensa que una posible solución podría ser el hecho de dividir el tiempo, esto es, todos los tiempos usados en la simulación quedan divididos por una constante, excepto el diferencial de tiempo, para así, aunque vaya a saltos, pero es como si se pusiera a cámara rápida.
Igualmente habría que ajustar el diferencial de tiempo, puesto que, si la constante que divide fuese muy alta, se saltaría pasos intermedios, y no completaría los estados.

Habría que comprobar también que la capacidad de Python es suficiente para ello.

7.4 Elementos flexibles en el seguimiento solar

Por el simple hecho de no obtener tan buen rendimiento gráfico en este primer disparo, no se ha optado por emular los elementos flexibles de los que constan los tubos reales, que conectan un colector con otro, permitiendo la reorientación de estos sin uniones rígidas.

Por un momento, se pensó en colocar elementos parecidos a una manguera, pero ni Unity lo incluye por defecto en su lista de objetos genéricos, ni era viable su modelado, debido a su alta complejidad.

7.5 Modificación método anticolisión por uno más sofisticado

En el presente trabajo, no ha sido necesaria la implementación de un algoritmo que evite las colisiones con demasiada precisión, ya que el proyecto real está formado por solamente 3 vehículos aéreos, y estos vuelan en zonas muy amplias, donde la probabilidad de simplemente encontrarse dos de estos es muy pequeña.

Sin embargo, si este estuviese formado por una flota de 200 drones, por ejemplo, entonces sí que habría que implementar planificadores de trayectorias más complejos, pudiéndose implementar desde algoritmos algo más complejos, como pueden ser algoritmos que tengan en cuenta todas las posiciones de todos los vehículos al mismo tiempo, hasta algoritmos que sean fruto de la inteligencia artificial, tales como los algoritmos evolutivos, basados en la evolución biológica, en el que mediante recompensas y penalizaciones se entrene a una red.

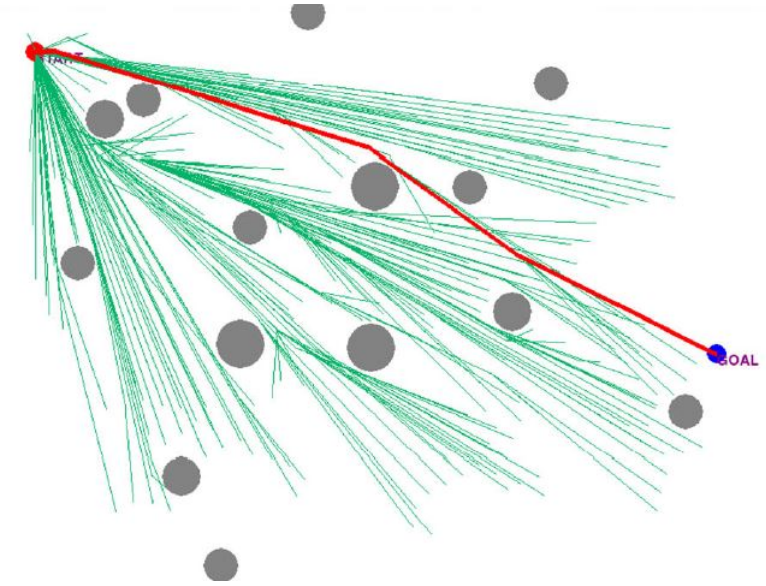


Figura 7.1 Caminos generados por un algoritmo RRT modificado [4].

7.6 Conclusiones

En primer lugar, el objeto del proyecto ha sido culminado con éxito, de tal forma que se es capaz de controlar una flota de 3 vehículos aéreos no tripulados, teniendo en cuenta sus posibles choques, entre ellos, y con el terreno. Se ha llegado a este objetivo buscando siempre la mayor simpleza posible, puesto que es así como se pueden encontrar y solucionar errores rápidamente.

Por otro lado, también se ha abordado con éxito un entorno en el que poder visualizar con claridad los resultados obtenidos. Esta representación, que se ha llevado una gran parte del tiempo dedicado en el trabajo, puede ser empleada en mostrar la realidad del algoritmo, de forma que una persona sin demasiados conocimientos en algoritmia pueda observar la potencia de este proyecto con imágenes.

Cabe decir que todos los parámetros utilizados son una primera aproximación de lo que pueden llegar a ser, y que se trata de un modelo bastante flexible para rápidas modificaciones. Se puede explicar como que este modelo constituye una base bastante robusta con la que, a través de dichos cambios, se puede ajustar con precisión a lo que se requiera.

Por último, aunque parezca que sea un detalle a pasar por alto, el hecho de conectar Python y Unity abre las puertas a futuros desarrollos de cualquier tipo, ya que reúne la capacidad de Python a usar cualquier tipo de librerías, como por ejemplo, la de *TensorFlow*, que incorpora la estructura básica para crear redes neuronales, siendo algoritmos avanzados y funcionales, con la capacidad de Unity, que es capaz de representarlo en tiempo real todo lo que adecuadamente se le pida, modificando las partes pertinentes del código.

En el siguiente enlace se pueden descargar los archivos: https://drive.google.com/file/d/1eyjbvQIIpwXTOVN8zQ1v9lLQfQezBaA5/view?usp=sharing

8 Apéndice I. Códigos *Python*

8.1 Códigos personalmente implementados casi en su totalidad

Código 8.1 Cinemática en general.

```
import socket
import random as rd
import time
from matplotlib import pyplot as plt
from matplotlib.pyplot import figure
from mpl_toolkits.mplot3d import Axes3D
from dron import dron
from superv import superv
import copy
import math
import os
import socket
import pathlib
from sklearn.cluster import AgglomerativeClustering


basedir=os.path.dirname(os.path.abspath(__file__))
basedir=pathlib.PureWindowsPath(basedir)
basedir=str(basedir.as_posix())
name = socket.gethostname()
ip = socket.gethostbyname(name)

def sendPos(x,y,z,j):
    s = socket.socket()
    s.connect((str(ip), 1755))
    string1=str()
    for i in range(0,len(x)-1):
        string1=string1+str(x[i])+","+str(z[i])+","+str(y[i])+","
    string1=string1+str(x[len(x)-1])+","+str(z[len(x)-1])+","+str(y[len(x)-1])
        +","+str(int(j))
    s.send((string1).encode())
    s.close()
mapa=input("Mostrar mapa de radiación (True o False): ")
grafica=input("Mostrar simulación en tiempo real con puntos (True o False): ")
graficaclusters=input("Mostrar mapa por grupos de tareas (True o False): ")
Nx=4;Nz=96
```

```
f = open(basedir+"/simulador_flota_drones/Assets/positionFile.txt", "r")
thetax=f.readline();thetay=f.readline();thetaz=f.readline()#Posición Solar
    Inicial
if thetax.count(",")>0:thetax=float(thetax.replace(",",".",1))
else:thetax=float(thetax)
if thetay.count(",")>0:thetay=float(thetay.replace(",",".",1))
else:thetay=float(thetay)
if thetaz.count(",")>0:thetaz=float(thetaz.replace(",",".",1))
else:thetaz=float(thetaz)
f.close()
if thetaz>90:
    thetax=180-thetax
    thetay=thetay-180
thetax=-thetax*3.1415/180
thetay=thetay*3.1415/180
cant=3
posinic=list([]);posinic2=list([]);mision=list([]);bat=list([])
dmin=5;d=1
for j in range(0,Nz):
    for i in range(0,Nx):
        if j<42 or j>47 or (j>=42 and j<=47 and i<2):
            mision.append([i*152.28+88,j*18.02+31.3])

posinic=copy.deepcopy(mision)
ndrones=cant;ntareas=len(mision)
cluster = AgglomerativeClustering(n_clusters=ndrones, affinity='euclidean',
    linkage='ward')
cluster.fit_predict(mision)
clusterlabels=list(cluster.labels_)

for k in range(0,len(mision)):
    if mision[k]!=[0,0,0]:
        if 6/math.tan(thetax+0.00001)>2000:
            aux1=2000
        else: aux1=6/math.tan(thetax)
        if aux1*math.tan(thetay+0.00001)>2000:
            aux=2000
        else: aux=aux1*math.tan(thetay)
        if thetay>3.1415/2:
            aux1=-aux1;aux=-aux
        mision[k][1]=posinic[k][1]+aux1
        mision[k][0]=posinic[k][0]+aux
        mision[k].append(10)

if graficaclusters=="True":
    plt.scatter([item[0] for item in mision],[item[1] for item in mision], c=
        cluster.labels_)
    plt.show()


mision1=list([])
estadossup=0 #Estado inicial del supervisor: Organizar el incio
t1=list([]);xy1=list([]);xy1p=list([]);xy1pp=list([])
t2=list([]);xy2=list([]);xy2p=list([]);xy2pp=list([])
estado=list([]);t_estado=list([]);indic2=list([]);estadomax=list([]);base=list
    ([])
ts=time.time()
```

```
for k in range(0,ndrones):
    base.append([k*228+89,0,0.3])
    t1.append(time.time());xy1.append(base[k]);xy1p.append([0,0,0]);xy1pp.
        append([0,0,0])
    xy2.append(base[k]);xy2p.append([0,0,0]);xy2pp.append([0,0,0])
    estado.append(0)
    t_estado.append(0)
    indic2.append([0,0,0])
    estadomax.append(0)
    bat.append(100)


posinic2=copy.deepcopy(base)
[t2,_,_,mision1,estado,mision,posinic2]=superv(t1,xy1,xy1p,xy1pp,mision,
    estadossup,estado,mision1,indic2,posinic2,estadomax,base,bat,clusterlabels)
for i in range(0,ntareas):
    if mision1[i]!=(ndrones+1):
        [_,xy2[mision1[i]],xy2p[mision1[i]],xy2pp[mision1[i]],identif,indic,
            estado[mision1[i]],t_estado[mision1[i]],mision1,estadomax,mision,bat
            [mision1[i]]]=dron(t1[mision1[i]],xy1[mision1[i]],xy1p[mision1[i]],
            xy1pp[mision1[i]],mision1[i],mision[i],1,t_estado[mision1[i]],
            posinic2[mision1[i]],mision1,mision,estadomax,ts,base,bat[mision1[i
            ]],thetax)
for i in range(0,ndrones):
    if i in mision1:
        indic2[i]=mision[mision1.index(i)]
if grafica=="True":
    plt.ion()
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')


p=1;filenames=list([])
while sum(estado)!=0:

    if (time.time()-ts)%600<=1 and mapa=="True":
        cloud=list([]);var1=list([]);var2=list([]);var3=list([])
        f = open(basedir+"/simulador_flota_drones/Assets/lightdetectordata.txt",
             "r")
        i=0;j=-93;m=0
        while i<640:
            var1.append(i);var2.append(j)
            var4=f.readline()
            if var4.count(",")>0:var4=float(var4.replace(",",".",1))
            else:var4=float(var4)
            var3.append(var4)
            j=j+10
            m=m+1
            if j>=1757:j=-93;i=i+10

        colormap = plt.cm.Greys_r
        normalize = plt.Normalize(vmin=0, vmax=1)
        var3 = [(num-min(var3))/(max(var3)-min(var3)) for num in var3]
        plt.figure(figsize=(6, 12), dpi=100)
        plt.scatter(var1, var2, c=var3, s=40, cmap=colormap, norm=normalize)
        f.close()
```

```
        plt.show()


    if (time.time()-ts)%150<=1:
        f = open(basedir+"/simulador_flota_drones/Assets/positionFile.txt", "r")

        thetax=f.readline();thetay=f.readline();thetaz=f.readline()
        if thetax.count(",")>0:thetax=float(thetax.replace(",",".",1))
        else:thetax=float(thetax)
        if thetay.count(",")>0:thetay=float(thetay.replace(",",".",1))
        else:thetay=float(thetay)
        if thetaz.count(",")>0:thetaz=float(thetaz.replace(",",".",1))
        else:thetaz=float(thetaz)
        f.close()
        if thetaz>90:
            thetax=180-thetax
            thetay=thetay-180
        thetax=-thetax*3.1415/180
        thetay=thetay*3.1415/180
        for k in range(0,len(mision)):
            if mision[k]!=[0,0,0]:
                if 6/math.tan(thetax+0.00001)>2000:
                    aux1=2000
                else: aux1=6/math.tan(thetax)
                if aux1*math.tan(thetay+0.00001)>2000:
                    aux=2000
                else: aux=aux1*math.tan(thetay)
                if thetay>3.1415/2:
                    aux1=-aux1;aux=-aux
                mision[k][1]=posinic[k][1]+aux1
                mision[k][0]=posinic[k][0]+aux

    for identif in range(0,ndrones):
        if identif in mision1:
            indic=indic2[identif]
            [t2[identif],xy2[identif],xy2p[identif],xy2pp[identif],identif,indic
                ,estado[identif],t_estado[identif],mision1,estadomax,mision,bat[
                identif]]=dron(t2[identif],xy2[identif],xy2p[identif],xy2pp[
                identif],identif,indic,estado[identif],t_estado[identif],
                posinic2[identif],mision1,mision,estadomax,ts,base,bat[identif],
                thetax)
            indic2[identif]=indic
            t2[identif]=time.time()
        x=list([]);y=list([]);z=list([])
        for h in range(0,ndrones):
            x.append(xy2[h][0])
            y.append(xy2[h][1])
            z.append(xy2[h][2])
    j=-1
    if estado.count(4)>0:
        for i in range(0,ndrones):
            if estado[i]==4:
                j=mision1.index(i)+1
    for _ in range(0,ndrones):
        sendPos(x,y,z,j)
        time.sleep(0.0001)
```

```
    [t2,xy2p,indic2,mision1,estado,mision,posinic2]=superv(t2,xy2,xy2p,xy2pp,
        mision,1,estado,mision1,indic2,posinic2,estadomax,base,bat,
        clusterlabels)
    if grafica=="True":
        ax.scatter(x,y,z, marker='o',c='blue')
    misx=list([]);misy=list([]);misz=list([])
    if grafica=="True":
        for h in range(0,ntareas):
            if mision1[h]!=ndrones+1:
                misx.append(mision[h][0])
                misy.append(mision[h][1])
                misz.append(mision[h][2])
        ax.scatter(misx, misy, misz, marker='o',c='red')
        ax.set_xlim3d(min([item[0] for item in mision]), max([item[0] for item
            in mision]))
        ax.set_ylim3d(min([item[1] for item in mision]), max([item[1] for item
            in mision]))
        ax.set_zlim3d(0, 100)
        plt.draw()

    if grafica=="True":
        plt.pause(0.001)
        ax.cla()
    if time.time()-ts>3000:
        break
print(time.time()-ts)
```

Código 8.2 Modelo individual de dron (*dron.py*).

```
import math
import time
import random as rd
def dron(t1,xy1,xy1p,xy1pp,identif,indic,estado,t_estado,posinic2,mision1,
    mision,estadomax,ts,base,bat,thetax):
    tope=5 #Segundos de medición
    tdesv=rd.random()+0.1 #Segundos de desviación
    ndrones=len(base)
    paramacel=5;paramvelmin=0.5;paramvelmax=7.5;rblanco=20#Parámetros de diseño
    paramacelmax=20
    batlvl=15
    t2=time.time()-t1
    t2=1*t2
    v=math.sqrt((xy1p[0])**2+(xy1p[1])**2+(xy1p[2])**2)
    d=math.sqrt((indic[0]-xy1[0])**2+(indic[1]-xy1[1])**2+(indic[2]-xy1[2])**2)
    if v==0: v=1
    if d==0: d=1
    if v<=paramvelmin and d<=rblanco/20:
        xy2pp=[0,0,0]
        xy1p=[xy1p[0]/v/10,xy1p[1]/v/10,xy1p[2]/v/10]
    if identif in mision1 and estado!=0 and d>rblanco/40 and (time.time()-ts)
        %(10)<=0.5: #En caso de descontrol por aceleraciones que descolocan
        xy1p=[xy1p[0]/v/10,xy1p[1]/v/10,xy1p[2]/v/10]
        xy2pp=[-4*(mision[mision1.index(identif)][0]-xy1[0])/d*paramvelmin,-4*(
            mision[mision1.index(identif)][1]-xy1[1])/d*paramvelmin,-4*(mision[
            mision1.index(identif)][2]-xy1[2])/d*paramvelmin]
        estado=1
    if identif in mision1 and estado!=0 and d>rblanco/40 and (time.time()-ts)
        %(100)<=0.3:
        xy1p=[xy1p[0]/v/100,xy1p[1]/v/100,xy1p[2]/v/100]
        xy2pp=[4*(identif*10-xy1[0])/d*paramvelmin,4*(identif*10-xy1[1])/d*
            paramvelmin,4*(identif*10-xy1[2])/d*paramvelmin]
        estado=1
    if estado>estadomax[identif] and estado!=4:
        estadomax[identif]=estado
    bat=bat*(1-v*t2/1800/paramvelmax-0.00001) #Duración de 30 min para el
        Phantom 4 Pro (Contiene el hovering)
    if bat<batlvl and d<=rblanco/20 and indic==posinic2:
        estado=0
    if -thetax*180/3.1415<=5 or -thetax*180/3.1415>=175:#De noche: como si
        estuviera sin bateria
        bat=batlvl*0.95
    if -thetax*180/3.1415<=6 and -thetax*180/3.1415>=5:#Amanecer: recargado
        bat=100

    if estado==0:#Apagado/Cargando
        t2=t1;xy2=xy1;xy2p=[0,0,0];xy2pp=[0,0,0];indic=[1,1,1]
    elif estado==1:#Vuelo normal
        v=math.sqrt((xy1p[0])**2+(xy1p[1])**2+(xy1p[2])**2)
        if v==0: v=1
        s=(indic[0]-xy1[0])*xy1p[0]+(indic[1]-xy1[1])*xy1p[1]+(indic[2]-xy1[2])*
            xy1p[2]
        if s==0:   s=1
        s=s/abs(s)
```

```
        xy2pp=[(indic[0]-xy1[0])/d*paramacel,(indic[1]-xy1[1])/d*paramacel,(
            indic[2]-xy1[2])/d*paramacel]
        if v>=paramvelmax:
            xy2pp=[0,0,0]

        #Detección de aproximación para desaceleración
        if d<=rblanco:
            xy2pp=[-v*s*(indic[0]-xy1[0])/d,-v*s*(indic[1]-xy1[1])/d,-v*s*(indic
                [2]-xy1[2])/d]
            if v<=paramvelmin:
                xy2pp=[(indic[0]-xy1[0])*paramvelmin*8,(indic[1]-xy1[1])*
                    paramvelmin*8,(indic[2]-xy1[2])*paramvelmin*8]

        #Operación sensor
        if v<=paramvelmin and v>=0 and d<=rblanco/20 and t_estado==0:
            xy1pp=[0,0,0]
            xy2=xy1;xy2p=xy1p;xy2pp=xy1pp
            t_estado=tope
            if indic not in mision and indic not in posinic2:
                t_estado=tdesv
            if estadomax[identif]==3.3:
                t_estado=tdesv
        if v<=paramvelmin and d<=rblanco/20:
            t_estado=t_estado-t2
        if t_estado<0:
            if indic in mision:
                estado=4
                xy2p=[0,0,0]
                xy2pp=[0,0,0]
            if indic not in mision and indic not in posinic2:
                estado=3.2
                if estadomax[identif]==3.2:
                    estado=3.3
                if estadomax[identif]==3.3:
                    estadomax[identif]=0

            xy1p=[xy1p[0]/v/100,xy1p[1]/v/100,xy1p[2]/v/100]

            t_estado=0
        if indic==posinic2 and d<=rblanco/10:
            estado=0
            mision1[mision1.index(identif)]=ndrones+1

        #Propagación trayectoria
        a=math.sqrt((xy2pp[0])**2+(xy2pp[1])**2+(xy2pp[2])**2)
        if a==0: a=1
        if a>=paramacelmax:
            xy2pp=[xy2pp[0]/a*paramacelmax,xy2pp[1]/a*paramacelmax,xy2pp[2]/a*
                paramacelmax]
        xy2p=[xy1p[0]+xy2pp[0]*t2,xy1p[1]+xy2pp[1]*t2,xy1p[2]+xy2pp[2]*t2]
        xy2=[xy1[0]+xy2p[0]*t2,xy1[1]+xy2p[1]*t2,xy1[2]+xy2p[2]*t2]
    elif estado==3.1:#Maniobra para esquivar: Desaceleración
        indic=[xy1[0]-5*xy1p[0]/v,xy1[1]-5*xy1p[1]/v,xy1[2]-5*xy1p[2]/v]
        t2=t1;xy2=xy1;xy2p=xy1p;xy2pp=xy1pp
        estado=1
    elif estado==3.2:#Maniobra para esquivar: Aceleración hacia waypoint
        indic=[xy1[0]-xy1p[1]/v*10,xy1[1]+xy1p[0]/v*10,xy1[2]-xy1p[2]/v*10]
```

```
        t2=t1;xy2=xy1;xy2p=xy1p;xy2pp=xy1pp
        estado=1
    elif estado==3.3:#Maniobra para esquivar: Retoma mision
        if mision[mision1.index(identif)]!=[0,0,0]:
            indic=mision[mision1.index(identif)]
        else:
            indic=base[identif]
        if bat<batlvl:
            indic=base[identif]

        t2=t1;xy2=xy1;xy2p=xy1p;xy2pp=xy1pp
        estado=1
    elif estado==4:#Viaje de vuelta
        indic=base[identif]
        t2=t1;xy2=xy1;xy2p=[0,0,0];xy2pp=[0,0,0]
        mision[mision1.index(identif)]=[0,0,0]
        estado=1
    else:
        t2=t1;xy2=xy1;xy2p=xy1p;xy2pp=xy1pp


    return t2,xy2,xy2p,xy2pp,identif,indic,estado,t_estado,mision1,estadomax,
        mision,bat
```

Código 8.3 Sistema supervisor (*superv.py*).

```
import math
import time
def superv(t1,xy1,xy1p,xy1pp,mision,estadossup,estado,mision1,indic2,posinic2,
    estadomax,base,bat,clusterlabels):
    ndrones=len(xy1)
    ntareas=len(mision)
    tareasmax=1
    rblanco=3
    paramvelmax=7.5
    batlvl=15
    N=1900
    if estadossup==0:#Reparto incial de tareas
        if len(mision)<tareasmax:
            t2=list([]);dtotal=list([])
            #Optimizador de misiones
            for h in range(0,ntareas):
                mision1=list([])
                misionaux = mision[h:] + mision[:h]
                #Organización de tareas por distancia
                for k in range(0,ntareas):
                    d=list([])
                    aux2=list([])
                    for j in range(0,ndrones):
                        aux=math.sqrt((xy1[j][0]-misionaux[k][0])**2+(xy1[j][1]-
                            misionaux[k][1])**2+(xy1[j][2]-misionaux[k][2])**2)
                        d.append(aux)
                        aux2.append(aux)
                    for n in range(0,len(d)):
                        if d.count(d[n])>1:
                            d[n]=d[n]+0.01*n
                            aux2[n]=aux2[n]+0.01*n
                    aux2.sort()
                    mision1.append(ndrones+1)
                    if k==0:
                        mision1[0]=d.index(aux2[0])
                    else:

                        for m in range(0,k):
                            if m>=ndrones:
                                break
                            if d.index(aux2[m]) not in mision1:
                                mision1[k]=d.index(aux2[m])
                                break
                        if mision1[k]==ndrones+1 and k<ndrones:
                            mision1[k]=d.index(aux2[m+1])
                d=list([])
                for j in range(0,ndrones):
                    if j in mision1:
                        aux=math.sqrt((xy1[j][0]-misionaux[mision1.index(j)][0])
                            **2+(xy1[j][1]-misionaux[mision1.index(j)][1])**2+(
                            xy1[j][2]-misionaux[mision1.index(j)][2])**2)
                        d.append(aux)
                dtotal.append(sum(d))
            #Organización óptima
```

```
            mision=mision[dtotal.index(min(dtotal)):] + mision[:dtotal.index(min
                (dtotal))]
            mision1=list([])
            for k in range(0,ntareas):
                d=list([])
                aux2=list([])
                for j in range(0,ndrones):
                    aux=math.sqrt((xy1[j][0]-misionaux[k][0])**2+(xy1[j][1]-
                        misionaux[k][1])**2+(xy1[j][2]-misionaux[k][2])**2)
                    d.append(aux)
                    aux2.append(aux)
                for n in range(0,len(d)):
                    if d.count(d[n])>1:
                        d[n]=d[n]+0.01*n
                        aux2[n]=aux2[n]+0.01*n
                aux2.sort()
                mision1.append(ndrones+1)
                if k==0:
                    mision1[0]=d.index(aux2[0])
                else:

                    for m in range(0,k):
                        if m>=ndrones:
                            break
                        if d.index(aux2[m]) not in mision1:
                            mision1[k]=d.index(aux2[m])
                            break
                    if mision1[k]==ndrones+1 and k<ndrones:
                        mision1[k]=d.index(aux2[m+1])
        else:
            aux4=list([1000000000000])
            mision1=list([]);t2=list([]);
            for h in range(0,ntareas):
                mision1.append(ndrones+1)
                aux4.append(clusterlabels[h])
                if aux4.count(clusterlabels[h])==1:
                    mision1[h]=clusterlabels[h]


        for k in range(0,ndrones):
                t2.append(time.time())

    elif estadossup==1 and ndrones>1:#Comprobación mecanismo anticolisión y
        reorganización de misiones
        t2=t1
        for i in range(0,ndrones):
            for j in range(0,ndrones):
                if i!=j:
                    t=-((xy1p[i][0]-xy1p[j][0])*(xy1[i][0]-xy1[j][0])+(xy1p[i
                        ][1]-xy1p[j][1])*(xy1[i][1]-xy1[j][1])+(xy1p[i][2]-xy1p[j
                        ][2])*(xy1[i][2]-xy1[j][2]))
                    if ((xy1p[i][0]-xy1p[j][0])*(xy1[i][0]-xy1[j][0])+(xy1p[i
                        ][1]-xy1p[j][1])*(xy1[i][1]-xy1[j][1])+(xy1p[i][2]-xy1p[j
                        ][2])*(xy1[i][2]-xy1[j][2]))==0:
                        t=-1
                    else:
```

```
                    t=t/((xy1p[i][0]-xy1p[j][0])*(xy1p[i][0]-xy1p[j][0])+(
                        xy1p[i][1]-xy1p[j][1])*(xy1p[i][1]-xy1p[j][1])+(xy1p[
                        i][2]-xy1p[j][2])*(xy1p[i][2]-xy1p[j][2]))
                d=((xy1p[i][0]-xy1p[j][0])*(xy1p[i][0]-xy1p[j][0])+(xy1p[i
                    ][1]-xy1p[j][1])*(xy1p[i][1]-xy1p[j][1])+(xy1p[i][2]-xy1p
                    [j][2])*(xy1p[i][2]-xy1p[j][2]))*t**2
                d=d+2*((xy1p[i][0]-xy1p[j][0])*(xy1[i][0]-xy1[j][0])+(xy1p[i
                    ][1]-xy1p[j][1])*(xy1[i][1]-xy1[j][1])+(xy1p[i][2]-xy1p[j
                    ][2])*(xy1[i][2]-xy1[j][2]))*t
                d=d+((xy1[i][0]-xy1[j][0])*(xy1[i][0]-xy1[j][0])+(xy1[i][1]-
                    xy1[j][1])*(xy1[i][1]-xy1[j][1])+(xy1[i][2]-xy1[j][2])*(
                    xy1[i][2]-xy1[j][2]))
                if d<0: d=0
                d=math.sqrt(d)
                v=math.sqrt((xy1p[i][0])**2+(xy1p[i][1])**2+(xy1p[i][2])**2)
                d2=math.sqrt((xy1[i][0]-xy1[j][0])**2+(xy1[i][1]-xy1[j][1])
                    **2+(xy1[i][2]-xy1[j][2])**2)
                if t>0 and t<=N*math.sqrt(2)/paramvelmax and (d2<=rblanco*20*
                    v/paramvelmax or d2<=rblanco*2) and d<=rblanco and (
                    indic2[i] in mision or indic2[i] in posinic2):
                    estado[i]=3.1
                elif t>0 and t<=N*math.sqrt(2)/paramvelmax and (d2<=rblanco
                    *20*v/paramvelmax or d2<=rblanco*2) and d<=rblanco and
                    estadomax[i]==3.2:
                    estado[i]=3.1
                d2=math.sqrt(((xy1[i][0]-xy1[j][0])*(xy1[i][0]-xy1[j][0])+(
                    xy1[i][1]-xy1[j][1])*(xy1[i][1]-xy1[j][1])+(xy1[i][2]-xy1
                    [j][2])*(xy1[i][2]-xy1[j][2])))
                if d2<=5:
                    xy1p[i]=[(xy1[i][0]-xy1[j][0]),(xy1[i][1]-xy1[j][1]),(xy1
                        [i][2]-xy1[j][2])]
                if d2<=1:
                    print('COLISION',d2)
        if (xy1[i][0]>330 and xy1[i][0]<640 and xy1[i][1]>775 and xy1[i
            ][1]<890 and xy1[i][2]<45) or (xy1[i][1]>8 and xy1[i][2]<8) or (
            xy1[i][2]<0):
            xy1p[i]=[xy1p[i][0],xy1p[i][1],5]


if len(mision)<tareasmax:
    if estadossup==1 and estado.count(4)>0 and mision.count([0,0,0])!=
        ntareas:
        #Organización óptima
        aux3=list([])
        for k in range(0,ntareas):
            d=list([])
            aux2=list([])
            for j in range(0,ndrones):
                    aux=math.sqrt((xy1[j][0]-mision[k][0])**2+(xy1[j][1]-
                        mision[k][1])**2+(xy1[j][2]-mision[k][2])**2)
                    d.append(aux)
                    aux2.append(aux)

            for n in range(0,len(d)):
                if d.count(d[n])>1:
                    d[n]=d[n]+0.01*n
                    aux2[n]=aux2[n]+0.01*n
```

```
                aux2.sort()
                m=0
                for _ in range(0,ndrones):
                    if mision1[k]==ndrones+1 and mision[k]!=[0,0,0] and estado[d.
                        index(aux2[m])]==4 and d.index(aux2[m]) not in aux3:
                        mision[mision1.index(d.index(aux2[m]))]=[0,0,0]
                        mision1[mision1.index(d.index(aux2[m]))]=ndrones+1
                        mision1[k]=d.index(aux2[m])
                        estado[d.index(aux2[m])]=1
                        indic2[d.index(aux2[m])]=mision[k]
                        posinic2[d.index(aux2[m])]=xy1[d.index(aux2[m])]
                        aux3.append(d.index(aux2[m]))
                        break
                    m=m+1
        if estado.count(4)>0 and estadossup==1:
            for identif in range(0,ndrones):
                if estado[identif]==4:
                    posinic2[identif]=base[identif]
                    indic2[identif]=base[identif]
        for identif in range(0,ndrones):
            if identif in mision1:
                if bat[identif]<batlvl and indic2[identif]!=base[identif] and
                    indic2[identif]==mision[mision1.index(identif)]:
                    xy1p[identif]=[xy1p[identif][0]/1.5,xy1p[identif][1]/1.5,xy1p
                        [identif][2]/1.5]
                    estado[identif]=1
                    indic2[identif]=base[identif]
                    posinic2[identif]=base[identif]
    else:
        if estadossup==1 and estado.count(4)>0 and mision.count([0,0,0])!=
            ntareas:
            #Organización óptima
            aux3=list([]);aux5=[0]*ndrones
            for i in range(0,ntareas):
                for h in range(0,ndrones):
                    if mision[i]!=[0,0,0] and clusterlabels[i]==h:
                        aux5[h]=aux5[h]+1
            for k in range(0,ntareas):
                d=list([])
                aux2=list([])
                for j in range(0,ndrones):
                    aux=math.sqrt((xy1[j][0]-mision[k][0])**2+(xy1[j][1]-mision[k
                        ][1])**2+(xy1[j][2]-mision[k][2])**2)
                    d.append(aux)
                    if d.count(d[j])>1:
                        d[j]=d[j]+0.01*j
                    aux2.append(d[j])
                aux2.sort()
                m=0

                for h in range(0,ndrones):
                    if mision1[k]==ndrones+1 and mision[k]!=[0,0,0] and estado[d.
                        index(aux2[m])]==4 and d.index(aux2[m]) not in aux3 and
                        clusterlabels[k]==d.index(aux2[m]) and aux5[d.index(aux2[
                        m])]>=1:
                        mision[mision1.index(d.index(aux2[m]))]=[0,0,0]
                        mision1[mision1.index(d.index(aux2[m]))]=ndrones+1
```

```
                    mision1[k]=d.index(aux2[m])
                    estado[d.index(aux2[m])]=1
                    indic2[d.index(aux2[m])]=mision[k]
                    posinic2[d.index(aux2[m])]=xy1[d.index(aux2[m])]
                    aux3.append(d.index(aux2[m]))
                    break
                m=m+1
    if estado.count(4)>0 and estadossup==1:
        for identif in range(0,ndrones):
            if estado[identif]==4:
                posinic2[identif]=base[identif]
                indic2[identif]=base[identif]
    for identif in range(0,ndrones):
        if identif in mision1:
            if bat[identif]<batlvl and indic2[identif]!=base[identif] and
                indic2[identif]==mision[mision1.index(identif)]:
                xy1p[identif]=[xy1p[identif][0]/1.5,xy1p[identif][1]/1.5,xy1p
                    [identif][2]/1.5]
                estado[identif]=1
                indic2[identif]=base[identif]
                posinic2[identif]=base[identif]



t2=t1


return t2,xy1p,indic2,mision1,estado,mision,posinic2
```

9 Apéndice II. Códigos *C#*

9.1 Códigos adaptados a la aplicación en cuestión

Código 9.1 Guarda los ángulos del sol (*savePos.cs*) [5].

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System.Threading.Tasks;

    public class savePos : MonoBehaviour {
        string path = Directory.GetCurrentDirectory();
        string FILE_NAME = "c:/Users/usuario/OneDrive - UNIVERSIDAD DE SEVILLA
            /4º GIA/TFG/simulador_flota_drones/Assets/positionFile.txt";
        private GameObject Sun;
        private float time=0;
        void Start()
        {
        FILE_NAME=path+@"\Assets\positionFile.txt";
        FILE_NAME=FILE_NAME.Replace(@"\","/");
        }
        void Update()
        {
        time=time+Time.deltaTime;
        if (time%150<=Time.deltaTime*3){
        Sun=GameObject.Find("Sun");
        StreamWriter sw = new StreamWriter(FILE_NAME);
        float x = Sun.transform.rotation.eulerAngles.x;
        float y = Sun.transform.rotation.eulerAngles.y;
        float z = Sun.transform.rotation.eulerAngles.z;
        sw.WriteLine(x);
        sw.WriteLine(y);
        sw.WriteLine(z);
        sw.Close();
        }

        }
    }
```

Código 9.2 Almacena la intensidad de radiación del entorno *LightCheckScript.cs*.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System.Threading.Tasks;

public class LightCheckScript : MonoBehaviour
{
    public RenderTexture lightCheckTexture;
    public float LightLevel;
    private GameObject Capsule;
    private GameObject Plane;
    private int i=0;
    private int j =-93;
    private int m=0;
    public bool ActivarMapeado=false;
    string path = Directory.GetCurrentDirectory();
    string FILE_NAME = "c:/Users/usuario/OneDrive - UNIVERSIDAD DE SEVILLA/4º
        GIA/TFG/simulador_flota_drones/Assets/lightdetectordata.txt";
    float[] lightlist = new float[20000];


    void Start()
    {
        FILE_NAME=path+@"\Assets\lightdetectordata.txt";
        FILE_NAME=FILE_NAME.Replace(@"\","/");
        Capsule=GameObject.Find("LightDetector");
        Plane=GameObject.Find("LightDetectorPlane");


    }
    void Update()
    {

        if (ActivarMapeado)
        {
            Capsule.transform.position=new Vector3(i*1f,14*1f,j*1f);
            Plane.transform.position=new Vector3(i*1f,10*1f,j*1f);
            RenderTexture tmpTexture=RenderTexture.GetTemporary(
                lightCheckTexture.width,lightCheckTexture.height,0,
                RenderTextureFormat.Default,RenderTextureReadWrite.Linear);
            Graphics.Blit(lightCheckTexture,tmpTexture);
            RenderTexture previous=RenderTexture.active;
            RenderTexture.active=tmpTexture;

            Texture2D temp2DTexture=new Texture2D(lightCheckTexture.width,
                lightCheckTexture.height);
            temp2DTexture.ReadPixels(new Rect(0,0,tmpTexture.width,tmpTexture.
                height),0,0);
            temp2DTexture.Apply();

            RenderTexture.active=previous;
            RenderTexture.ReleaseTemporary(tmpTexture);
            Color32[] colors=temp2DTexture.GetPixels32();
```

```
            LightLevel=0;
            for(int n=0;n<colors.Length;n++)
            {
                LightLevel += (0.2126f*colors[n].r)+(0.7152f*colors[n].g)
                    +(0.0722f*colors[n].b);
                lightlist[m]=LightLevel;

            };
            j=j+10;
            m=m+1;
            if(j>=1757)
            {
                j=-93;
                i=i+10;

            };
            if(i>=640)
            {
                i=0;
                j=-93;
                ActivarMapeado=false;
                StreamWriter sw = new StreamWriter(FILE_NAME);
                for (int k=0;k<20000;k++)
                {
                    sw.WriteLine(lightlist[k]);
                }

                sw.Close();
            }
        }
    }
}
```

Código 9.3 Traslada los vehículos y rota los colectores (*ChangePos.cs*) [12].

```
using System;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using UnityEngine;

public class ChangePos : MonoBehaviour
{
    static Socket listener;
    private CancellationTokenSource source;
    public ManualResetEvent allDone;
    public Renderer objectRenderer;
    private Vector3 positions0;
    private Vector3 positions1;
    private Vector3 positions2;

    public GameObject Drone0;
    public GameObject Drone1;
    public GameObject Drone2;
    public GameObject Sun;
    public GameObject SolarPanels;
    private float thetax;
    private GameObject SolarPanelClone;
    int j;
    float aux;


    public static readonly int PORT = 1755;
    public static readonly int WAITTIME = 1;

    ChangePos()
    {
        source = new CancellationTokenSource();
        allDone = new ManualResetEvent(false);
    }


    async void Start()
    {
        Drone0=GameObject.Find("Drone0");
        Drone1=GameObject.Find("Drone1");
        Drone2=GameObject.Find("Drone2");
        Sun=GameObject.Find("Sun");
        objectRenderer = GetComponent<Renderer>();
        await Task.Run(() => ListenEvents(source.Token));
    }

    void Update()
    {

        Drone0.transform.position=positions0;
        Drone1.transform.position=positions1;
```

```
        Drone2.transform.position=positions2;
        SolarPanelClone=GameObject.Find("SolarPanel"+(j));


        if (j!=-1 && aux!=j)
        {

            thetax=Sun.transform.rotation.eulerAngles.x;
            Debug.Log(Sun.transform.rotation.eulerAngles);
            if (Sun.transform.rotation.eulerAngles.z>=90)
            {
                thetax=180-Sun.transform.rotation.eulerAngles.x;

            }
            if (Sun.transform.rotation.eulerAngles.y>=90)
            {
                Debug.Log(thetax);
                thetax=180-thetax;
            }
            if (Sun.transform.rotation.eulerAngles.y-Sun.transform.rotation.
                eulerAngles.z<90 && Sun.transform.rotation.eulerAngles.z>=90)
            {
                thetax=180-thetax;
            }
            Debug.Log(thetax);
            thetax=thetax-SolarPanelClone.transform.GetChild(1).rotation.
                eulerAngles.x-28;

            SolarPanelClone.transform.GetChild(1).RotateAround(SolarPanelClone.
                transform.position+new Vector3(0f,0f,0f),new Vector3(1f,0f,0f),
                thetax);
            SolarPanelClone.transform.GetChild(2).RotateAround(SolarPanelClone.
                transform.position+new Vector3(0f,0f,0f),new Vector3(1f,0f,0f),
                thetax);
            SolarPanelClone.transform.GetChild(3).RotateAround(SolarPanelClone.
                transform.position+new Vector3(0f,0f,0f),new Vector3(1f,0f,0f),
                thetax);
            SolarPanelClone.transform.GetChild(4).RotateAround(SolarPanelClone.
                transform.position+new Vector3(0f,0f,0f),new Vector3(1f,0f,0f),
                thetax);

            aux=j;
        }

    }

    private void ListenEvents(CancellationToken token)
    {


        IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
        IPAddress ipAddress = ipHostInfo.AddressList.FirstOrDefault(ip => ip.
            AddressFamily == AddressFamily.InterNetwork);
        IPEndPoint localEndPoint = new IPEndPoint(ipAddress, PORT);
```

```
        listener = new Socket(ipAddress.AddressFamily, SocketType.Stream,
            ProtocolType.Tcp);


        try
        {
            listener.Bind(localEndPoint);
            listener.Listen(10);


            while (!token.IsCancellationRequested)
            {
                allDone.Reset();
                listener.BeginAccept(new AsyncCallback(AcceptCallback),listener)
                    ;

                while(!token.IsCancellationRequested)
                {
                    if (allDone.WaitOne(WAITTIME))
                    {
                        break;
                    }
                }

            }

        }
        catch (Exception e)
        {
            //print(e.ToString());
        }
    }

    void AcceptCallback(IAsyncResult ar)
    {
        Socket listener = (Socket)ar.AsyncState;
        Socket handler = listener.EndAccept(ar);

        allDone.Set();

        StateObject state = new StateObject();
        state.workSocket = handler;
        handler.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0, new
            AsyncCallback(ReadCallback), state);
    }

    void ReadCallback(IAsyncResult ar)
    {
        StateObject state = (StateObject)ar.AsyncState;
        Socket handler = state.workSocket;

        int read = handler.EndReceive(ar);

        if (read > 0)
        {
            state.colorCode.Append(Encoding.ASCII.GetString(state.buffer, 0,
                read));
```

```
            handler.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0, new
                 AsyncCallback(ReadCallback), state);
        }
        else
        {
            if (state.colorCode.Length > 1)
            {
                string content = state.colorCode.ToString();
                SetPos(content);
            }
            handler.Close();
        }
    }

    //Set color to the Material
    private void SetPos (string data)
    {
        string[] colors = data.Split(',');
        int i=0;
        for(;;){
            string a =colors[i];
            string b =colors[i+1];
            string c =colors[i+2];
            j=int.Parse(colors[9]);
            int index1 = a.IndexOf('.');
            int index2 = b.IndexOf('.');
            int index3 = c.IndexOf('.');
            index1=a.Length-index1-1;
            index2=b.Length-index2-1;
            index3=c.Length-index3-1;
            float x=float.Parse(colors[i]);
            x=x/(float)Math.Pow(10.00, index1);
            float y=float.Parse(colors[i+1]);
            y=y/(float)Math.Pow(10.00, index2);
            float z=float.Parse(colors[i+2]);
            z=z/(float)Math.Pow(10.00, index3);
            if (i==0){
            positions0= new Vector3 (x, y, z) ;
            };
            if (i==3){
            positions1= new Vector3 (x, y, z) ;
            };
            if (i==6){
            positions2= new Vector3 (x, y, z) ;
            };
            i=i+3;

        }

    }

    private void OnDestroy()
    {
        source.Cancel();
    }

    public class StateObject
```

```
    {
        public Socket workSocket = null;
        public const int BufferSize = 1024;
        public byte[] buffer = new byte[BufferSize];
        public StringBuilder colorCode = new StringBuilder();
    }

}
```

9.2 Códigos realizados integramente por otros desarrolladores

Código 9.4 *SetSunLocation.cs* [11].

```
using UnityEngine;
using System.Collections;

namespace Entropedia {
    public class SetSunLocation : MonoBehaviour
    {
    [SerializeField]
    Sun sun;

    public void Start()
    {
        StartCoroutine(SetLocation());
    }

    public IEnumerator SetLocation()
    {
        if (!Input.location.isEnabledByUser){
            Debug.LogWarning("location disabled by user");
            yield break;
        }
        Input.location.Start();

        while (Input.location.status == LocationServiceStatus.Initializing){
        yield return new WaitForSeconds(0.5f);
        }

        if (Input.location.status == LocationServiceStatus.Failed){
            Debug.LogWarning("Unable to determine device location");
            yield break;
        }

        if(Input.location.status==LocationServiceStatus.Running){
        var locInfo = Input.location.lastData;
        Debug.LogFormat("long={0} lat={1}",locInfo.longitude,locInfo.latitude);
        sun.SetLocation( locInfo.longitude, locInfo.latitude );
        }

        Input.location.Stop();
    }
    }
}
```

Código 9.5 *Sun.cs* [11].

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

namespace Entropedia
{

    [RequireComponent(typeof(Light))]
    [ExecuteInEditMode]
    public class Sun : MonoBehaviour
    {
        [SerializeField]
        float longitude;

        [SerializeField]
        float latitude;

        [SerializeField]
        [Range(0, 24)]
        int hour;

        [SerializeField]
        [Range(0, 60)]
        int minutes;

        DateTime time;
        Light light;

        [SerializeField]
        float timeSpeed = 1;

        [SerializeField]
        int frameSteps = 1;
        int frameStep;

        [SerializeField]
        DateTime date;

        public void SetTime(int hour, int minutes) {
            this.hour = hour;
            this.minutes = minutes;
            OnValidate();
        }

        public void SetLocation(float longitude, float latitude){
            this.longitude = longitude;
            this.latitude = latitude;
        }

        public void SetDate(DateTime dateTime){
            this.hour = dateTime.Hour;
            this.minutes = dateTime.Minute;
            this.date = dateTime.Date;
            OnValidate();
```

```
        }

        public void SetUpdateSteps(int i) {
            frameSteps = i;
        }

        public void SetTimeSpeed(float speed) {
            timeSpeed = speed;
        }

        private void Awake()
        {

            light = GetComponent<Light>();
            time = DateTime.Now;
            hour = time.Hour;
            minutes = time.Minute;
            date = time.Date;
        }

        private void OnValidate()
        {
            time = date + new TimeSpan(hour, minutes, 0);

            Debug.Log(time);
        }

        private void Update()
        {
            time = time.AddSeconds(timeSpeed * Time.deltaTime);
            if (frameStep==0) {
                SetPosition();
            }
            frameStep = (frameStep + 1) % frameSteps;


        }

        void SetPosition()
        {

            Vector3 angles = new Vector3();
            double alt;
            double azi;
            SunPosition.CalculateSunPosition(time, (double)latitude, (double)
                longitude, out azi, out alt);
            angles.x = (float)alt * Mathf.Rad2Deg;
            angles.y = (float)azi * Mathf.Rad2Deg-90;
            //UnityEngine.Debug.Log(angles);
            transform.localRotation = Quaternion.Euler(angles);
            light.intensity = Mathf.InverseLerp(-12, 0, angles.x);
        }


    }

    /*
```

```
        * The following source came from this blog:
        * http://guideving.blogspot.co.uk/2010/08/sun-position-in-c.html
        */
    public static class SunPosition
    {
        private const double Deg2Rad = Math.PI / 180.0;
        private const double Rad2Deg = 180.0 / Math.PI;

        /*!
            * \brief Calculates the sun light.
            *
            * CalcSunPosition calculates the suns "position" based on a
            * given date and time in local time, latitude and longitude
            * expressed in decimal degrees. It is based on the method
            * found here:
            * http://www.astro.uio.no/~bgranslo/aares/calculate.html
            * The calculation is only satisfiably correct for dates in
            * the range March 1 1900 to February 28 2100.
            * \param dateTime Time and date in local time.
            * \param latitude Latitude expressed in decimal degrees.
            * \param longitude Longitude expressed in decimal degrees.
            */
        public static void CalculateSunPosition(
            DateTime dateTime, double latitude, double longitude, out double
                outAzimuth, out double outAltitude)
        {
            // Convert to UTC
            dateTime = dateTime.ToUniversalTime();

            // Number of days from J2000.0.
            double julianDate = 367 * dateTime.Year -
                (int)((7.0 / 4.0) * (dateTime.Year +
                (int)((dateTime.Month + 9.0) / 12.0))) +
                (int)((275.0 * dateTime.Month) / 9.0) +
                dateTime.Day - 730531.5;

            double julianCenturies = julianDate / 36525.0;

            // Sidereal Time
            double siderealTimeHours = 6.6974 + 2400.0513 * julianCenturies;

            double siderealTimeUT = siderealTimeHours +
                (366.2422 / 365.2422) * (double)dateTime.TimeOfDay.TotalHours;

            double siderealTime = siderealTimeUT * 15 + longitude;

            // Refine to number of days (fractional) to specific time.
            julianDate += (double)dateTime.TimeOfDay.TotalHours / 24.0;
            julianCenturies = julianDate / 36525.0;

            // Solar Coordinates
            double meanLongitude = CorrectAngle(Deg2Rad *
                (280.466 + 36000.77 * julianCenturies));

            double meanAnomaly = CorrectAngle(Deg2Rad *
                (357.529 + 35999.05 * julianCenturies));
```

```
            double equationOfCenter = Deg2Rad * ((1.915 - 0.005 *
                julianCenturies) *
                Math.Sin(meanAnomaly) + 0.02 * Math.Sin(2 * meanAnomaly));

            double elipticalLongitude =
                CorrectAngle(meanLongitude + equationOfCenter);

            double obliquity = (23.439 - 0.013 * julianCenturies) * Deg2Rad;

            // Right Ascension
            double rightAscension = Math.Atan2(
                Math.Cos(obliquity) * Math.Sin(elipticalLongitude),
                Math.Cos(elipticalLongitude));

            double declination = Math.Asin(
                Math.Sin(rightAscension) * Math.Sin(obliquity));

            // Horizontal Coordinates
            double hourAngle = CorrectAngle(siderealTime * Deg2Rad) -
                rightAscension;

            if (hourAngle > Math.PI)
            {
                hourAngle -= 2 * Math.PI;
            }

            double altitude = Math.Asin(Math.Sin(latitude * Deg2Rad) *
                Math.Sin(declination) + Math.Cos(latitude * Deg2Rad) *
                Math.Cos(declination) * Math.Cos(hourAngle));

            // Nominator and denominator for calculating Azimuth
            // angle. Needed to test which quadrant the angle is in.
            double aziNom = -Math.Sin(hourAngle);
            double aziDenom =
                Math.Tan(declination) * Math.Cos(latitude * Deg2Rad) -
                Math.Sin(latitude * Deg2Rad) * Math.Cos(hourAngle);

            double azimuth = Math.Atan(aziNom / aziDenom);

            if (aziDenom < 0) // In 2nd or 3rd quadrant
            {
                azimuth += Math.PI;
            }
            else if (aziNom < 0) // In 4th quadrant
            {
                azimuth += 2 * Math.PI;
            }

            outAltitude = altitude;
            outAzimuth = azimuth;
        }

        /*!
        * \brief Corrects an angle.
        *
        * \param angleInRadians An angle expressed in radians.
        * \return An angle in the range 0 to 2*PI.
```

```
        */
        private static double CorrectAngle(double angleInRadians)
        {
            if (angleInRadians < 0)
            {
                return 2 * Math.PI - (Math.Abs(angleInRadians) % (2 * Math.PI));
            }
            else if (angleInRadians > 2 * Math.PI)
            {
                return angleInRadians % (2 * Math.PI);
            }
            else
            {
                return angleInRadians;
            }
        }
    }

}
```

Código 9.6 *SetDate.cs* [11].

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

namespace Entropedia
{
    public class SetDate : MonoBehaviour
    {
        public Sun sun;
        private Light Sun1;


        private void OnValidate()
        {
            Sun1 = GetComponent<Light>();
            try{
                DateTime d = new DateTime(DateTime.Now.Year,DateTime.Now.Month,
                    DateTime.Now.Day,DateTime.Now.Hour,DateTime.Now.Minute,0);
                Debug.Log(d);
                if(sun) sun.SetDate(d);
            }
            catch(System.ArgumentOutOfRangeException e){
                Debug.LogWarning("bad date");
            }
        }
        private void Update()
        {
            Sun1.intensity=130001;
            Sun1.intensity=130000;
        }


    }
}
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] *Machine learning repository*, https://archive.ics.uci.edu/ml/index.php.

[2] 3DWarehouse, *Parabolic trough solar collector*, https://3dmdb.com/en/3d-model/parabolic-trough-solar-collector/9993588/, Jul 2021.

[3] AnanasProject, *Realistic drone*, https://assetstore.unity.com/packages/3d/vehicles/air/realistic-drone-66698, Aug 2016.

[4] Kaviyarasu Ayyakannu, *Sampling based path planning algorithm for uav collision avoidance*, Sept 2021.

[5] Clavus, *How do you save the position of clone prefabs in a text file?*, https://answers.unity.com/questions/1468095/how-do-you-save-the-position-of-clone-prefabs-in-a.html, Feb 2018.

[6] Gobierno de España, *Imagen desvío*, https://www.mitma.gob.es/el-ministerio/sala-de-prensa/noticias/lun-26072021-1644.

[7] Lofi Dev, *Unity performance tips: Draw calls*, https://www.youtube.com/watch?v=IrYPkSIvpIw, Dec 2020.

[8] Comisión Europea, *Optimal control of thermal solar energy systems*, https://cordis.europa.eu/project/id/789051/es, Nov 2020.

[9] Caroline Garrett, *Energía solar fotovoltaica y térmica: ventajas y desventajas*, t.ly/y7AMU, Mar 2022.

[10] Google, *Google earth*, earth.google.com.

[11] Paul Hayes, *Códigos sol artificial*, https://gist.github.com/paulhayes/54a7aa2ee3cccad4d37bb65977eb19e2, Jan 2020.

[12] ITTinkerGirl, *Códigos que conectan python a unity*, https://github.com/ITTinkerGirl/UnitySocketExample, Aug 2020.

[13] kallerna, *Solnova solar power station*, t.ly/RAf7, Apr 2021.

[14] LogoDix, *Heliport logo*, https://logodix.com/logos/1684829.

[15] Bospear Programming, *Unity get light intensity on player*, https://youtu.be/NYysvuyivc4, Mar 2018.

[16] PROTERMOSOLAR, *Energía termosolar*, https://www.protermosolar.com/la-energia-termosolar/que-es-tipos-de-plantas-beneficios/.

[17] PulkitS, *A beginners guide to hierarchical clustering and how to perform it in python*, https://www.analyticsvidhya.com/blog/2019/05/beginners-guide-hierarchical-clustering/, May 2019.

[18] Patricia Martín Rodríguez, *Estudio y revisión crítica de diseño de la planta solar termoeléctrica solnova 1*, Universidad Politécnica de Madrid, 2016.

[19] Wikipedia, *Socket de internet*, t.ly/hWLo, Jan 2022.