CEA Comité Español de Automática

XLV JORNADAS AUTOMÁTICA MÁLAGA 2024

UNIVERSIDADE DA CORUÑA

Jornadas de Automática

Monitorización mediante vehículos aéreos multicámara. Caso de uso de Unreal Engine

López Ruiz, J.F.[a], Rico, J.[a], Alamo, T.[a], Ortega, M.G.[a], Vargas, M.[a,*]

[a]*Dpto. Ing. de Sistemas y Automática. Universidad de Sevilla. Avda. de los Descubrimientos s/n. 41092, Sevilla, España.*

To cite this article: López Ruiz, J.F., Rico, J., Alamo, T., Ortega, M.G., Vargas, M. 2024. Monitoring through multi-camera aerial vehicles: A case study using Unreal Engine.
Jornadas de Automática, 45. https://doi.org/

Resumen

El trabajo propuesto en este artículo parte de la consideración de un vehículo aéreo autónomo, dotado de múltiples cámaras, como agente capaz de proporcionar mayor versatilidad en aplicaciones de monitorización y seguimiento de múltiples objetivos móviles. Dicho agente tendría la capacidad de orientar sus cámaras de forma completamente independiente y éstas estarían dotadas con capacidad de *zoom*, lo que permitiría ajustar la distancia focal de cada una de ellas a conveniencia. Partiendo de este concepto y de una estrategia de optimización desarrollada en trabajos anteriores, se propone una generalización de la misma que permita la colaboración de varios agentes en una misma misión. De forma complementaria, parte del trabajo se centra en explorar las posibilidades que ofrece *Unreal Engine 5* como herramienta de simulación gráfica para la implementación de la propuesta.

Palabras clave: Robots aéreos, Trabajo en entornos reales y virtuales, Soporte al operador humano, Percepción y detección, Sistemas multi-vehículo, Redes de robots y sensores inteligentes.

Monitoring through multi-camera aerial vehicles: A case study using Unreal Engine.

Abstract

The research work presented in this article is based on the concept of an autonomous aerial vehicle equipped with multiple cameras, serving as an agent capable of providing enhanced versatility in monitoring and tracking multiple mobile targets. The onboard cameras can be independently oriented and feature zoom functionality. This feature would enable adjusting the focal length of each camera as needed, optimizing the visual coverage of the targets. Building upon this concept and leveraging an optimization strategy developed in previous works, this study proposes a generalization of the approach to facilitate collaboration among multiple agents in a single mission scenario. Additionally, a significant component of the research focuses on exploring the potential of *Unreal Engine 5* as a graphical simulation tool for implementing the proposal and validating its effectiveness.

Keywords: Flying robots, Work in real and virtual environments, Human operator support, Perception and sensing, Multi-vehicle systems, Networks of robots and intelligent sensors.

1. Introducción

Hoy en día resulta bastante familiar la imagen de un vehículo aéreo no tripulado (UAV) dotado de una cámara orientable mediante un balancín motorizado (*gimbal*), que permita la monitorización visual del entorno de dicho vehículo. Si bien las primeras aplicaciones se limitaban esencialmente al ámbito de la inteligencia militar, en la última década se ha ampliado el espectro de uso a muchos otros dominios, incluyendo inspección de infraestructuras (Jordan et al. (2018); Salahat et al. (2019)), misiones de búsqueda y rescate (Sun et al. (2016); Burke et al. (2019)), vigilancia del tráfico (Khan et al. (2017); Kumar et al. (2020)), seguimiento de objetivos en general (Ahmed et al. (2021); Sun et al. (2023)), sin olvidar los usos recreativos, cada vez más extendidos.

Cuando el propósito es realizar el seguimiento simultáneo de varios objetivos móviles que puedan estar distantes entre

*Autor para correspondencia: mvargas@us.es.

sí, la solución convencional pasa bien por desplegar varios vehículos con capacidades similares, cada uno dedicado a un objetivo específico o grupo de objetivos cercanos, como en Robin and Lacroix (2016); Senanayake et al. (2016), bien por recurrir al concepto de atención compartida. En este último caso, un único vehículo puede ajustar periódicamente la dirección de apuntamiento de su cámara entre diferentes objetivos, intentando no perder la pista de ninguno de ellos (Sharma and Pack (2013); Zhao et al. (2019); Baek and York (2020)).

En un trabajo reciente (Vargas et al. (2022)), se explora la idea de un único agente aéreo equipado con múltiples cámaras orientables de forma independiente, con objeto de rastrear simultáneamente varios objetivos con mayor efectividad, al tiempo que se minimice el despliegue de recursos. En el citado trabajo, se asume como premisa que las cámaras destinadas a las tareas de seguimiento poseen una focal fija. En Vargas et al. (2024), por su parte, se generaliza la propuesta anterior. En primer lugar, se introduce la capacidad de agrupación de los objetivos (*clustering*). Esto permite abordar situaciones en las que el número de objetivos sea sustancialmente superior al número de cámaras a bordo, de forma que cada una de ellas se centre en un grupo de objetivos, en lugar de en uno individual. Por otra parte y más relevante, se introduce la capacidad de *zoom* ajustable en las cámaras. Esto proporciona una flexibilidad muy superior al agente, al permitirle encontrar automáticamente el equilibrio óptimo entre la distancia focal a fijar en cada una de las cámaras y la distancia real al objetivo o *cluster* en cuestión.

El presente trabajo plantea, como primera contribución destacada, una generalización del concepto original, considerando que se dispone de varios agentes multicámara, para desempeñar una misma misión de supervisión y seguimiento.

Por otra parte, cabe destacar que las herramientas de simulación gráfica y física desempeñan un papel fundamental en la investigación y desarrollo de sistemas robóticos aéreos, entre otros. Ello se debe a que proporcionan un entorno ideal para llevar a cabo pruebas de concepto y ensayos preliminares con seguridad, comodidad y bajo coste. En este sentido, la segunda contribución relevante del presente trabajo es la propuesta de una implementación del concepto referido en un entorno de simulación que ofrezca, entre otros aspectos, cualidades de animación foto-realistas.

La estructura del artículo es la siguiente. En primer lugar, en la Sección 2, se describe el caso de uso tratado, detallando los conceptos teóricos necesarios e incluyendo la generalización a múltiples agentes de los trabajos previos. En la Sección 3, se hace un rápido repaso a algunos de los entornos de simulación más aptos para la aplicación de interés, justificando la elección realizada, siendo ésta *Unreal Engine*. La Sección 4, por su parte, describe los aspectos más relevantes de la implementación llevada a cabo en *Unreal Engine* y los detalles de la misión recreada. Finalmente, en la Sección 5, se presentan las conclusiones del trabajo.

2. Caso de uso: Monitorización aérea mediante equipo de agentes multi-cámara

2.1. *Concepto de agente aéreo autónomo multi-cámara*

Tal y como se describe en Vargas et al. (2022, 2024), cada uno de los agentes considerados sería un vehículo aéreo con una cámara central, instalada sobre un *gimbal* que puede ser de dos grados de libertad y un número variable de cámaras de seguimiento, colocadas sobre respectivos *gimbals* de tres grados de libertad. En la Figura 1, se muestra una imagen esquemática de un agente de tipo multirrotor con una cámara central flanqueada por dos cámaras de seguimiento. La cámara central mantendría en todo momento un punto de vista cenital, porporcionando una vista general de la escena. Por su parte, las cámaras de seguimiento se centrarán en un objetivo o *cluster* particular, intentando mantenerlo centrado y encuadrado en la imagen.

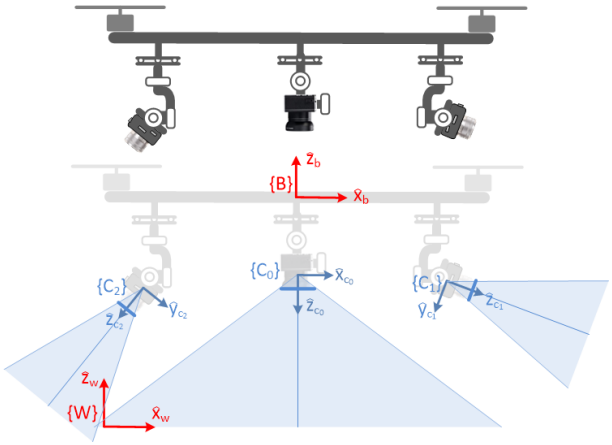

Figura 1: Concepto de agente aéreo multi-cámara empleado.

En este trabajo se hará la suposición de mundo plano, de forma que, partiendo del conocimiento de la altitud del vehículo, y de la posición proyectada en la imagen de un cierto objetivo, pueda deducirse la posición tridimensional del mismo. En la descripción que sigue, se denotará por x_k la posición del agente k-esimo (mediante $\mathrm{x}_k(t)$, en caso de que se requiera explicitar la dependencia con el tiempo, donde t representaría el número de ciclo de ejecución), dotado con n_k cámaras de seguimiento. Por su parte, z_i representa la posición del objetivo i-ésimo (o centro del *cluster* i-èsimo). La variable $d_{(k;i)}$ representa la distancia entre ambos: $d_{(k;i)} = \|\mathrm{x}_k - \mathrm{z}_i\|$.

Basándose en la idea anterior, en Vargas et al. (2024) se plantea un índice de optimización que proporcione referencias de posición en tiempo real, que permitan posicionar el vehículo de forma óptima para la observación del conjunto de objetivos. Para el objetivo o *cluster* i-ésimo, se determina, en primer lugar, la distancia ideal para la observación del mismo, $d_{(k;i);r}$, en función del tamaño equivalente que se estima que tiene en la realidad dicho objetivo o *cluster*, y del tamaño deseado del mismo en la imagen. A partir de esta distancia, se determina que, si efectivamente no hubiera más objetivos, la posición deseada a la que se debería dirigir el vehículo sería una posición en la vertical del objetivo, a una distancia igual a $d_{(k;i);r}$. A esta posición se le denota por $\mathrm{p}_{(k;i);r}$.

Cuando hay más de un objetivo o grupo, salvo en casos límite, no será posible satisfacer la condición anterior para todos los objetivos, de modo que se plantea una segunda condición deseable, en defecto del cumplimiento de la primera. Esta nueva condición pretende que se respete la distancia de observación deseada, aunque no se tenga un punto de vista vertical del objetivo en cuestión.

Cuando hay más de tres objetivos o grupos, en general, no será posible satisfacer la condición anterior para todos ellos. La siguiente condición pretenderá que la distancia al objetivo sea tal que se garantice el tamaño deseado en la imagen, usando un valor de distancia focal en la cámara correspondiente que esté dentro de su rango operativo: $f_{(k;i);min} \leq f_{(k;i)} \leq f_{(k;i);max}$.

Si la condición anterior tampoco es posible, el siguiente paso es intentar que tanto la distancia focal como el tamaño aparente del objeto en la imagen estén dentro del rango aceptable.

Finalmente, en caso de que lo anterior no sea viable, tenemos la situación en la que no se podrá ajustar un valor de distancia focal que permita que el tamaño del objeto en la imagen esté dentro del rango deseado, penalizándose la desviación respecto a los límites de dicho rango.

Como resumen de lo anterior, las condiciones de observación pueden enumerarse, comenzando por la situación ideal hasta llegar a la más desfavorable, del siguiente modo:

1. $x_k = p_{(k;i);r}$: Alineamiento vertical óptimo respecto a un objetivo individual, consiguiéndose al mismo tiempo la distancia focal ideal y el tamaño deseado en la imagen.
2. $d_{(k;i)} = d_{(k;i);r}$: Se viola el alineamiento vertical óptimo, pero la distancia focal y el tamaño del objeto en la imagen son los ideales.
3. $d_{(k;i)} \in [L_{(k;i);1}; U_{(k;i);1}]$: La distancia focal no coincide con el valor óptimo, pero permanece dentro del rango admisible, mientras que se garantiza que el tamaño del objeto en la imagen es el deseado.
4. $d_{(k;i)} \in [L_{(k;i);2}; U_{(k;i);2}]$: La distancia focal y el tamaño aparente del objeto en la imagen no son los ideales, pero sí están dentro del rango deseado.
5. $d_{(k;i)} < [L_{(k;i);2}; U_{(k;i);2}]$: No puede conseguirse mantener el tamaño aparente del objetivo en la imagen dentro del rango deseado, dadas las limitaciones de la distancia focal de la cámara.

A partir de todo lo anterior, el índice se formula como una suma de términos, cada uno correspondiente a una cámara de seguimiento. A su vez, cada uno de estos términos se construye mediante la suma de tres sub-términos que pretenden penalizar que la solución viole la secuencia de condiciones expuestas. En particular, el problema de optimización propuesto determina la referencia óptima para la posición del agente resolviendo:

$$x_k = \arg\min_{x_2 X_k} J_{k;o}(x_k); \qquad (1)$$

siendo X_k la caja que define las restricciones a imponer sobre la posición del agente k-ésimo, de forma que, por ejemplo, pueda obligarse a que dicha posición esté restringida a una cierta franja de altitud. Por su parte, el índice de optimización asociado a la "calidad" en la observación del conjunto de objetivo o clusters vinculados a dicho agente, $J_{k;o}$, se define como:

$$J_{k;o}(x_k) = \sum_{i=1}^{n_k} {}_{(k;i)}(x_k; z_i) + {}_{(k;i)}(x_k; z_i) + {}_{(k;i)}(x_k; z_i) \qquad (2)$$

Los términos ${}_{(k;i)}(x_k; z_i)$ penalizan el incumplimiento de las condiciones 1 y 2. Los términos ${}_{(k;i)}(x_k; z_i)$ penalizan el hecho de que se rebasen los umbrales superiores $U_{(k;i);j}$, mientras que los términos ${}_{(k;i)}(x_k; z_i)$ penalizan el incumplimiento de los umbrales inferiores $L_{(k;i);j}$. Estos dos últimos, por tanto, cuantifican las condiciones 3 a 5. Para una descripción más detallada del desarrollo expuesto, se remite al lector a Vargas et al. (2024).

2.2. Gestión de múltiples agentes

En este trabajo, se propone la generalización de la idea anterior para la inclusión de varios agentes, cuyo cometido sea la monitorización conjunta de un grupo de objetivos. Como se verá, se trata de una solución sub-óptima, donde la coordinación no se plantea mediante la reformulación integral del problema de optimización ampliado. Por contra, se adopta la solución óptima individual existente, y se combina con una estrategia de optimización combinatorial aplicada al mecanismo de clustering.

Como punto de partida, se asume que se dispone de m agentes para una misma misión, pudiendo ser éstos de diversa naturaleza y estar dotados con diferente número de cámaras de seguimiento. Se denotará por a_k al identificador del k-ésimo agente, mientras que $A := \{a_1; a_2; \ldots; a_m\}$ constituirá el conjunto de los identificadores de todos los agentes disponibles. De modo similar, t_i representará el identificador del cluster (u objetivo, target) i-ésimo, siendo el conjunto ordenado de los n clusters $T := \{t_1; t_2; \ldots; t_n\}$. Por otra parte, X representará el conjunto de las posiciones adjudicadas a los agentes en un cierto momento: $X = \{x_1; x_2; \ldots; x_m\}$ mientras que Z hará referencia al conjunto de las posiciones de los clusters en un cierto instante: $Z = \{z_1; z_2; \ldots; z_n\}$. Para cada agente, se definirá un coste compuesto por tres términos: en primer lugar, el asociado a la observación de los objetivos, acorde a la definición (2); en segundo lugar, el coste del desplazamiento del agente desde su posición actual y, por último, el coste asociado a eventuales cambios en las asignaciones cámara-cluster (swaps):

$$J_k(x_k) = {}_o J_{k;o}(x_k) + {}_d \|x_k(t) - x_k(t-1)\|_2 + {}_{sw} S_k: \qquad (3)$$

donde se han introducido como factores de ponderación las constantes ${}_o$, ${}_d$, ${}_{sw}$ y donde S_k representa el número de cambios de asignación cámara-cluster que ha experimentado el agente k-ésimo respecto al ciclo previo. En relación a esto último, cabe destacar el interés de introducir dicha penalización, con idea de reducir la frecuencia de estos intercambios, favoreciendo con ello la tarea de supervisión por parte de operadores humanos.

El primer paso del algoritmo consistirá en agrupar los objetivos en un número n de clusters, coincidente con el número de cámaras de seguimiento disponibles: $n = \sum_{k=1}^{m} n_k$. Para tener en cuenta todas las posibles asignaciones de clusters que se puedan hacer al agente a_k, se obtendrán las permutaciones de los n clusters, tomados de n_k en n_k, $P(T; n_k)$. Las permutaciones resultantes se denotarán por $T_{n_k;j}$; $j = 1; \ldots; n_k! = (n \; n_k)!$. Dado un subconjunto ordenado de identificadores de cluster, $T_{n_k;j}$, tentativamente asignados al agente a_k, se asume que el primero se está asignando a la cámara 1 del agente, el segundo a la cámara 2, etc. El coste de cada asignación completa, considerando todos los agentes, se obtendrá a partir de (3), del siguiente modo:

$$J(X) = \sum_{k=1}^{m} J_k(x_k):$$

En un principio, cabría pensar en calcular, para cada una de las posibles asignaciones de clusters a cámaras, el correspondiente coste y simplemente elegir aquella que proporcione el valor mínimo del mismo. Esto, sin embargo, supondría resolver, en cada ciclo de ejecución, un problema de optimización relativamente costoso un número de veces igual a $n!$. Para evitar esta gravosa estrategia exhaustiva, se propone una alternativa branch & bound, que evite el cálculo del índice para aquellas permutaciones parciales que superen o igualen el coste mínimo, obtenido hasta el momento a partir de las permutaciones que ya han sido completamente desarrolladas. El procedimiento propuesto se detalla mediante los Algoritmos 1 y 2.

Algorithm 1: Cálculo del coste para un agente

```
1  Function {J_k, x_k} = CosteAgente(a_k, T_{n_k;j}):
2      x_k(t-1) = a_k.PosActual();
3      T_{n_k;j}(t-1) = a_k.AsignacionActual();
4      if a_k.CosteYaCalculado(T_{n_k;j}) then
5          J_k = a_k.LeeCosteCalculado(T_{n_k;j});
6      else
7          Z = {z : z = t_i.PosActual(); ∀t_i ∈ T_{n_k;j}}
8          Resolver x_k(t) = arg min_{x∈X_k} J_{k;o}(x; Z);
9          S_k = NumSwaps(T_{n_k;j}(t-1); T_{n_k;j});
10         J_k = ω_o J_{k;o}(x_k(t); Z_j) + ω_d ||x_k(t) - x_k(t-1)|| + ω_sw S_k;
11         a_k.RegistraCosteCalculado(T_{n_k;j}, J_k);
12     end if
```

Algorithm 2: Cálculo del coste global

```
1  Function {J*, X*} =
     CosteGlobal(A, T, J, X, J*, X*):
2      if A == ∅ then
3          if J < J* then
4              J* = J;  X* = X;
5          end if
6          return;
7      end if
8      for cada a_k ∈ A do
9          n_k = a_k.NumCamaras()
10         Obtener P(T, n_k);
11         for cada T_{n_k;j} ∈ P(T, n_k) do
12             {J_k, x_k} = CosteAgente(a_k, T_{n_k;j});
13             if J + J_k < J* then
14                 {J*, X*} = CosteGlobal(A \ {a_k}, T \
                   T_{n_k;j}, J + J_k, {X, x_k}, J*, X*);
15             end if
16         end for
17     end for
```

El Algoritmo 1 calcula, de acuerdo con (3), el coste para un agente individual, supuesta una asignación de clusters $T_{n_k;j}$, proporcionando tanto el valor óptimo del índice, como la posición correspondiente. El segundo algoritmo, por su parte, realiza el cálculo del coste global óptimo de forma recursiva, desarrollando el árbol de combinaciones agente-clusters necesarias, mediante la estrategia de branch & bound citada. Este segundo algoritmo recibe un conjunto o subconjunto de identificadores de agentes y clusters, $\{A, T\}$, el coste acumulado hasta el momento y el conjunto de posiciones calculadas hasta el momento al ir descendiendo por las ramas del árbol, $\{J, X\}$, y, finalmente, el coste mínimo encontrado hasta el momento y la solución correspondiente, $\{J^*, X^*\}$. Por otra parte, devuelve como salida la actualización de este último par de parámetros. El caso base de la estrategia recursiva en el Algoritmo 2 se alcanza cuando el conjunto A recibido es el conjunto vacío, en cuyo caso, se habrá alcanzado el nivel de mayor profundidad del árbol, habiéndose hecho, por tanto, una asignación completa de todos los clusters, procede entonces comprobar si el coste correspondiente al camino desarrollado es inferior al mínimo que se tenía hasta el momento, en cuyo caso, este mínimo es actualizado. El operador $\setminus$ usado en la llamada recursiva del algoritmo representa sustracción de conjuntos. La llamada principal al Algoritmo 2 será del siguiente modo: CosteGlobal (A ; T; 0; ; 1 ;), donde A y T representan el conjunto completo de agentes y clusters.

3. Entornos de simulación para aplicaciones de robótica aérea

Esta sección ofrece una breve enumeración de algunas de las herramientas de simulación gráfica más utilizadas en el contexto de vehículos aéreos no tripulados. Una interesante revisión reciente a este respecto puede encontrarse en Collins et al. (2021), donde se dedica un apartado específico a los sistemas de simulación para robótica aérea.

En primer lugar, cabe citar a Gazebo (Koenig and Howard (2004)), como el simulador multi-robot de código abierto más ampliamente utilizado en combinación con ROS. Posee una capacidad de renderizado relativamente limitada, pero ofrece, por el contrario, un amplio soporte para modelos UAV y controladores hardware. Como segunda alternativa, se puede hacer referencia a la ofrecida por el ecosistema Matlab, incluyendo Simulink y el UAV Toolbox, que posibilitan la simulación de vehículos aéreos basadas en Unreal Engine (The MathWorks (2024)). Sin embargo, al sustentarse esta alternativa en Matlab, podría ser menos deseable a la hora de trasladar de forma ágil el desarrollo a un entorno real.

Por otra parte, podemos mencionar a Webots (Michel (2004)). Se trata de un software de simulación de robots móviles 3D de código abierto que ofrece un conjunto diverso de robots y sensores predefinidos, pero carece de renderización foto-realista del entorno. AirSim (Shah et al. (2018)) es otro simulador de código abierto, desarrollado por Microsoft y basado en Unreal Engine 4, que está más específicamente orientado a vehículos aéreos. Si bien Airsim ofrece renderizado realista y detección de colisiones, se trata de una herramienta muy exigente en términos computacionales.

Dentro de la categoría de motores de videojuegos empleados para simulación en aplicaciones robóticas, destacan Unity y Unreal Engine. Unity ofrece renderizado realista y permite simulación en tiempo real mediante la integración de ROS. Basado en Unity, puede mencionarse también a Flightmare (Song et al. (2021)), que es un simulador de vehículos aéreos modular de código abierto que combina el motor de renderizado Unity con un motor de física flexible, adecuado para simulaciones de alta fidelidad.

En este trabajo se ha optado, sin embargo, por Unreal Engine (Epic Games (2024)). La elección se fundamenta en que se trata una herramienta permanentemente en la vanguardia,

gratuita para desarrolladores, respaldada por una fuerte comunidad y que ofrece por sí misma las capacidades de modelado y renderizado realista deseadas, con una carga computacional configurable según las limitaciones del hardware. En particular, se ha optado por Unreal Engine 5 (UE5), la cual introdujo importantes mejoras de rendimiento y un mayor foto-realismo de los diseños.

4. Implementación en Unreal Engine 5

Como se ha mencionado anteriormente, el motor de simulación empleado es UE5, concretamente su versión 5.2.1. UE5 ofrece numerosas herramientas para la programación, como Python Scripting, Blueprint Visual Scripting[1] y C++ Coding. Es esta última herramienta en la que se basa gran parte de la implementación llevada a cabo. En esta primera aproximación a UE5, sin embargo, no se simulan de forma totalmente realista la dinámica de objetos ni las colisiones, sino que el desarrollo se ha centrado mayoritariamente en los elementos cinemáticos y en los aspectos gráficos. Hay diversos conceptos, propios de la terminología UE5, que resulta relevante referir. En primer lugar el concepto de actor, referido a cualquier objeto existente en la escena. En segundo lugar, static mesh es cualquier actor compuesto por un grupo estático de polígonos, mientras que scene component es un actor no visible que proporciona un marco de referencia.

4.1. Creación del escenario

Se ha optado por recrear, en el contexto de una misión de salvamento y rescate marítimo tras accidente aéreo, el despliegue de un equipo de 4 agentes multirrotor. La Figura 4 permite apreciar como fondo de la misma el escenario simulado, cuyos elementos más destacados se pasan a comentar a continuación. En primer lugar, aparecen objetos estáticos creados a través de modelos 3D de libre uso e introducidos como static meshes. Otro elemento relevante es el agua con física realista (olas, refracción de la luz, etc.), creada con el complemento Water de UE5. El paisaje ha sido implementado con la herramienta Landscape. Finalmente, el sistema de iluminación, compuesto por varios actores, entre los cuales destacan Sun Sky, que recrea la luz solar junto a la atmósfera terrestre.

4.2. Simulación de objetivos de interés

Para la recreación de las víctimas del incidente, que serán los objetivos de interés para la aplicación, se puede usar la tecnología Meta Human, que permite la creación de réplicas de humanos de calidad foto-realista, de forma simple y rápida. Esta será una opción seleccionable, de forma opcional, en la aplicación, dado que supone un aumento significativo del coste computacional de la simulación. Se simularán trayectorias elípticas de distinta amplitud, excentricidad y velocidad para estos objetivos, desarrollándose todas ellas a nivel ligeramente inferior al de la superficie del agua. En la simulación realizada, se ha simplificado la inicialización del problema, asumiendo que, a la hora de poner en marcha la estrategia de monitorización y seguimiento de los objetivos, los agentes están situados de forma que aquellos queden encuadrados por las cámaras centrales de éstos.

4.3. Creación de los agentes

Como se ha comentado con anterioridad, se ha optado por un equipo de cuatro agentes $\mathcal{A} = \{a_1; a_2; a_3; a_4\}$. El número de cámaras de los agentes será diferente, poseyendo 4, 3, 2 y 1 cámaras de seguimiento, respectivamente, además de las respectivas cámaras centrales. La implementación de los agentes se ha llevado a cabo principalmente mediante funciones en C++ diseñadas para integrarse con Blueprint. De este modo, se configura el flujo de información entre dichas funciones, dando lugar a la lógica completa de cada agente. Asimismo, se ha modelado cada componente del agente (cuerpo, gimbals, cámaras, etc) mediante CAD y se han creado los static meshes correspondientes, lo que permite una visualización precisa del movimiento de los mismos. La Figura 2 muestra un agente con cuatro cámaras de seguimiento.

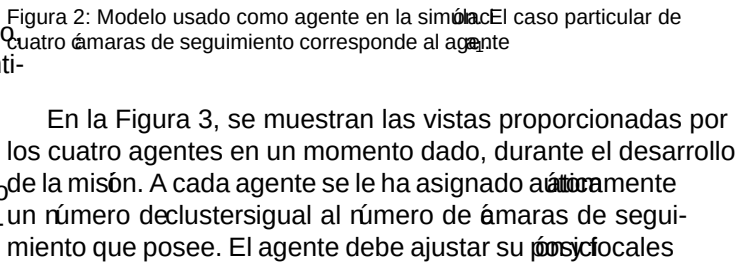

Figura 2: Modelo usado como agente en la simulación. El caso particular de cuatro cámaras de seguimiento corresponde al agente a_1.

En la Figura 3, se muestran las vistas proporcionadas por los cuatro agentes en un momento dado, durante el desarrollo de la misión. A cada agente se le ha asignado automáticamente un número de clusters igual al número de cámaras de seguimiento que posee. El agente debe ajustar su pose y focales de forma óptima, acorde al índice de minimización descrito, para el adecuado encuadre de los clusters asignados.

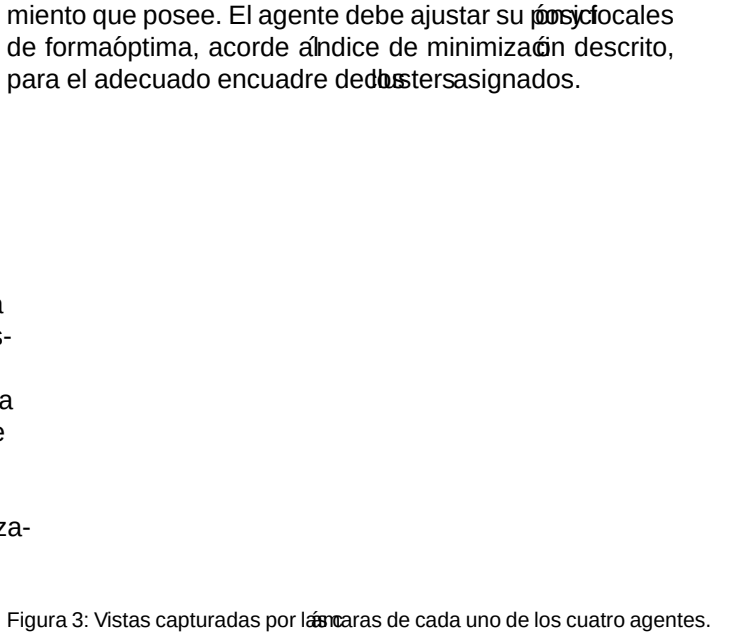

Figura 3: Vistas capturadas por las cámaras de cada uno de los cuatro agentes.

[1] Blueprint es un sistema de scripting visual nativo de UE, que está basado en la interconexión de nodos.