Departamento de Ingeniería de Sistemas y Automática
Escuela Superior de Ingenieros
Universidad de Sevilla

Proyecto Fin de Carrera

‘‘Sistema automático para la ejecución de misiones coordinadas con múltiples UAVs’’

Titulación: Ingeniería Superior de Telecomunicación

AUTOR: **Sergio Romero Ossorio**
TUTOR: **Jesús Iván Maza Alcañiz**
FECHA: Diciembre, año 2003

AGRADECIMIENTOS

Llegado este momento es hora de mirar atrás y comprobar lo lejos que queda aquel día en que comencé mi formación en esta Escuela de Ingenieros de Sevilla. Ahora que me dispongo a entregar este documento me vienen a la cabeza las personas que han estado junto a mí a lo largo de esta etapa en mi vida.

En primer lugar, y como no puede ser de otra manera, mi familia. Estudiar en una ciudad alejada de casa me ha resultado duro en algunos momentos, pero siempre he podido contar con vosotros cuando os he necesitado. Gracias por todo.

Mis padres, que me han apoyado todos estos años con una incondicionalidad que agradezco. Mis hermanos Javier, que tanto me ha echado de menos y Marta, que al comienzo de esta etapa era una niña de siete años y de la que tengo la sensación de haberme perdido unos años preciosos de su vida ahora que es una chica de quince. A mis abuelos por su cariño, me acuerdo de los que ya no estáis.

Sin lugar a duda lo más interesante de estos años sois las personas que he encontrado en esta ciudad: Virginia, Manuel, Juan Pablo, Rafa, Dolores, María, Lourdes,... De vosotros guardo de un recuerdo entrañable. Espero que ni el paso de los años ni el lugar de residencia nos distancie.

Por último a mi tutor de proyecto fin de carrera, Iván, por su ayuda inestimable en la elaboración del mismo y su amabilidad en el trato.

A todos, GRACIAS.

Sevilla, diciembre de 2003.

ÍNDICE

Capítulo 1: Introducción.

1.1. La coordinación de múltiples UAVs.

El vehículo aéreo no tripulado se conoce por las siglas UAV (unmanned air vehicle). Un UAV es un vehículo aéreo controlado a distancia de manera automática o semi-automática (con alguna intervención humana). El UAV se destina al cumplimiento de misiones muy diversas que pueden agruparse en dos grandes aplicaciones:

- ✓ Aplicaciones militares: reconocimiento del terreno, ataque masivo de objetivos, detección de agentes enemigos.

- ✓ Aplicaciones civiles: lucha contra incendios (autónoma), agrícola (tratamiento químico de grandes superficies), etcétera.

El sistema automático de coordinación de UAVs exige la cooperación entre los diferentes vehículos aéreos no tripulados para la obtención de una solución global que les permita afrontar determinadas misiones sobre el escenario introduciendo diferentes exigencias temporales para el conjunto de múltiples UAVs.

Las especificaciones temporales son variadas y dependen de la aplicación a que se destine el sistema de coordinación de múltiples UAVs. Se puede optar por la sincronización global de todos los ataques previstos o local para los vehículos que ataquen un determinado blanco. Otro tipo de planificación temporal hace referencia al conjunto de objetivos que ataca cada UAV y la secuenciación de dichas acciones en coordinación con el resto de vehículos.

La coordinación de múltiples UAVs exige la adopción de un esquema de interacción entre módulos, que puede ser de alto o bajo nivel. Normalmente se dispondrá de un sistema central encargado de ejecutar las acciones que implican la coordinación global de los vehículos que puede situarse en tierra o a bordo del UAV si se otorga a uno de ellos la capacidad de liderazgo (formaciones de vuelo).

En el proyecto se presenta una estrategia para la coordinación de las misiones asignadas a un conjunto de vehículos aéreos no tripulados. A partir de ciertas variables y funciones de coordinación se realizan operaciones y comunicaciones de bajo nivel que permiten la consecución de la misión. Se va a hacer hincapié en los aspectos relacionados con la planificación de las trayectorias para los UAVs con limitación del tiempo de la misión debido a los intereses del conjunto.

Tradicionalmente la coordinación de un conjunto de UAVs ha correspondido a una gran variedad de aplicaciones todas ellas militares, sobre todo aquellas que requerían una secuencialidad de operación, simultaneidad de los ataques o supresión de defensas aéreas enemigas. La aplicación de la coordinación de múltiples UAVs llega en la actualidad a otros sectores civiles tales como el control aéreo, la lucha contra incendios, el reconocimiento topográfico o el tratamiento agrícola de superficies.

Existen diversas formas de afrontar el reto de la coordinación de un conjunto de vehículos aéreos no tripulados partiendo de la complejidad que supone la planificación de las trayectorias para todos los UAVs en un escenario con intereses a veces contrapuestos en el que tiene que definirse un objetivo superior del conjunto que habrá que atender. La generación de un conjunto de puntos que definan las trayectorias entre cada UAV y su objetivo requiere el intercambio de una importante cantidad de información en tiempo real.

Otro hecho que aumenta la complejidad de la solución resulta ser la posibilidad de variaciones sustanciales en el escenario de trabajo. Este hecho resulta crítico en las aplicaciones de tipo militar que exigen al sistema automático de coordinación la garantía de poder responder a la aparición de amenazas dinámicas. Todas estas exigencias deben atenderse en el sistema automático de coordinación que tiene limitadas las capacidades de comunicación entre vehículos aéreos no tripulados.

La planificación de las trayectorias de manera individualizada se ha estudiado profusamente a lo largo de décadas, no ocurre lo mismo con la planificación coordinada de un conjunto de UAVs. Por todo lo explicado anteriormente supone un problema de tal complejidad que se aborda aprovechando los resultados que optimizan la decisión individual de cada UAV e integrando estas decisiones en una más amplia que afecte al conjunto maximizando los intereses del mismo.

Los sistemas dinámicos de coordinación de UAVs aplicados en los sectores civiles demandan cada vez mayores capacidades y características más sofisticadas. Esto constituye uno de los mayores retos para los diseñadores de los sistemas automáticos de coordinación, ya que la tecnología tradicional no se adecúa a estas nuevas necesidades. La nuevas características que han de presentarse en estas nuevas plataformas de control se pueden resumir en los siguientes puntos:

- ✓ Adaptabilidad o Reconfiguración dinámica: estos sistemas automáticos deben ser capaces de permitir una rápida y satisfactoria respuesta a los cambios motivados por nuevas expectativas o necesidades. Deben permitir las modificaciones sin violar la integridad del sistema.

- ✓ Modificación de los algoritmos de coordinación: el sistema debe ser capaz de soportar los cambios de los algoritmos de coordinación del sistema. Se tiene que insertar la nueva tecnología en la arquitectura del sistema sin la necesidad de rediseñar los componentes que ya forman parte de dicha arquitectura.

- ✓ Interoperatividad: los sistemas automáticos actuales operan en un escenario heterogéneo, con componentes distribuidos. La coordinación debe llegar y ejecutarse en diferentes procesadores, los cuales pueden llegar a usar diferentes lenguajes de programación o situarse sobre diversas plataformas. Debe de asegurarse una comunicación en tiempo real entre estos componentes susceptibles de ser controlados, satisfaciendo las necesidades de ancho de banda, tiempo de respuesta o seguridad.

- ✓ Carácter abierto: cualquier reconfiguración o intercambio de nuevos componentes requiere de una arquitectura de sistema flexible capaz de permitir la incorporación de algoritmos procedentes de diversas fuentes.

Para llegar a la concepción de sistema automático para múltiples UAVs, desde la aparición de los mismos se han pasado por etapas en las que el control de los vehículos dependía exclusivamente de la acción humana a través del control remoto. Más tarde se fueron introduciendo elementos que aumentaron su carácter autónomo. En la actualidad se opta por sistemas autónomos con las características anteriormente expuestas que facilitan el mantenimiento del sistema y la mejora de éste.

1.2. Objeto del proyecto.

El presente proyecto introduce una aproximación general para el control automático de la cooperación entre los múltiples UAVs del sistema, centrando el objeto en la coordinación temporal de los vehículos aéreos no tripulados. La coordinación entre los UAVs exige la comunicación de información entre ellos. La forma de alcanzar la información no se estudia en este documento, pero puede provenir de sensores que detecten la posición de los diferentes vehículos, objetivos y amenazas. Tampoco se aborda la técnica de comunicación empleada y que permite a los UAVs enviar y recibir información de los demás, de manera inalámbrica.

El objeto del proyecto es la elaboración de un conjunto de algoritmos que permitan ejercer una adecuada coordinación sobre un conjunto de vehículos aéreos no tripulados. Los aspectos relacionados con la coordinación se han expuesto en el apartado anterior y también las características que requiere un sistema automático de coordinación moderno. Los algoritmos se incluirán en dicho sistema, permitiendo el mantenimiento de los programas. En la programación de los procedimientos encargados de la coordinación existen dos aspectos fundamentales.

- ✓ Determinación de las variables de coordinación: se trata de la información que necesariamente ha de compartirse para permitir efectuar la cooperación. Son imprescindibles, pero tiene que procurarse el mínimo uso de variables de coordinación, ya que ese uso implica intercambio de información entre UAVs. Se define por tanto la variable de coordinación como la mínima cantidad de información necesaria para dar efecto a una función objetivo de coordinación.

- ✓ Establecimiento de una estrategia de cooperación entre los UAVs. En este punto y a partir de las variables de coordinación se fijan las funciones de coordinación. El cometido de éstas es evaluar el efecto que produce sobre determinados objetivos del conjunto el valor que toman determinadas variables de coordinación. A través de las funciones de coordinación se fija la estrategia de control de los vehículos aéreos no tripulados del sistema. La coordinación efectuada sobre los UAVs se va a centrar en el ámbito temporal. Existen variadas especificaciones temporales que aplicar a los vehículos, como la simultaneidad de las llegadas o la secuencialidad de las mismas.

Para dar lugar a una operatividad coordinada en el sistema de automático de múltiples UAVs tienen que abordarse las diferentes tareas de las que se ocupa el sistema para alcanzar la misión global. El objeto del proyecto aborda la implementación de los métodos y algoritmos correspondientes a dichos cometidos, de manera que se asegure la cooperación entre ellos con el fin de alcanzar el objetivo temporal. Las tareas del sistema automático de coordinación son las que siguen:

1. Planificación de Trayectorias. Se verá afectada por la asignación de objetivos (2) y la interceptación (3). Se establecerá un algoritmo de planificación que genere la solución más ventajosa asegurando la integridad de los vehículos aéreos no tripulados.

2. Asignación de Objetivos. A partir de un escenario de trabajo en el que se pretendan alcanzar ciertos objetivos por los UAVs con la presencia de ciertas localizaciones amenazantes, se decidirá de manera coordinada la asignación de los objetivos a los diferentes vehículos. La información intercambiada (variables de coordinación) provocará la decisión en función del valor de la función objetivo de asignación (función de coordinación).

3. Interceptación simultánea. La coordinación temporal de los UAVs se ha decidido centrar en el hecho de que los UAVs alcancen los objetivos asignados en el mismo instante de tiempo. La ejecución de esta tarea es crítica desde el punto de vista de la coordinación de los diferentes UAVs y afecta de manera significativa a la operatividad de los dos anteriores.

4. Generación de Trayectorias. Los UAVs presentan determinadas restricciones en el movimiento. Se adecuará la trayectoria planificada y seleccionada gracias a la interacción de los módulos anteriores para que pueda ser recorrida por el vehículo aéreo no tripulado. Resulta la trayectoria de referencia para el UAV.

5. Seguimiento de la Trayectoria. A partir de la trayectoria de referencia, y en función de las características funcionales del UAV se dará lugar al cumplimiento de la misión. Se realiza el seguimiento temporal de la referencia.

6. Comunicaciones. Se arbitrará la estrategia que permita el intercambio de las variables de coordinación que tengan que evaluarse en las funciones de coordinación programadas en los diferentes módulos del sistema automático.

1.3. Sistema automático de coordinación.

Se parte de un escenario de trabajo que va a propiciar la elección de diferentes métodos para los distintos cometidos. Se analizará la respuesta que se produzca desde el sistema ante esas especificaciones. El escenario de trabajo en el que efectúa su operación el sistema automático de coordinación de múltiples UAVs cuenta con:

- ✓ N_v vehículos aéreos no tripulados. Son el objeto de control por parte del sistema. Se procurará el cumplimiento coordinado de la misión asignada. (**^**).

- ✓ N_o objetivos. Son las localizaciones dentro del espacio controlado por los UAVs que se pretenden alcanzar por aquellos. Ante una aplicación de tipo militar puede tratarse de la eliminación de una defensa aérea enemiga. Si se dedica al ámbito civil, el objetivo será un punto del espacio que coincida por ejemplo con el foco de un incendio. En cualquiera de los casos el objetivo se considera puntual y será especificado por las coordenadas en el espacio (**+**).

- ✓ N_p situaciones peligrosas. Corresponden a obstáculos en el campo de operación del sistema o a elementos amenazantes que supongan un riesgo para la integridad de los UAVs. Puede tratarse de un sistema de detección radar a partir de la cual se ponga en marcha el ataque del UAV del sistema. Desde el punto de vista civil, las localizaciones amenazantes se asimilan a obstáculos para el UAV. En cualquiera de los casos se van a considerar puntuales, por lo que vendrán descritos por sus coordenadas espaciales (**o**).

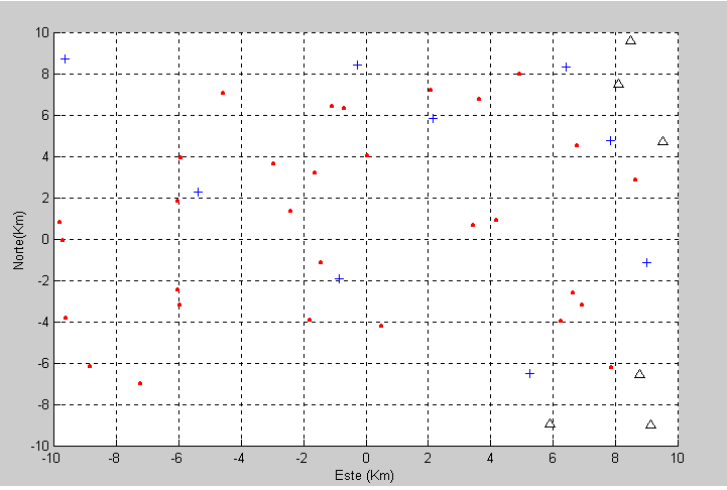

Figura 1: Escenario de trabajo.

La planificación de trayectorias compatible con una cooperación en tiempo real entre los distintos UAVs exige que determinados resultados de dicha planificación se empleen como variables de coordinación para la asignación de objetivos y la interceptación simultánea. El algoritmo de planificación incorporará herramientas con las que se evite el peligro y se obtengan trayectorias de mínima longitud posible. Se opta por el uso del grafo de Voronoi que se forma a partir del conocimiento de la posición de las localizaciones amenazantes para el UAV. Se establecerá un criterio para la búsqueda de las trayectorias más cortas entre dos puntos.

La asignación de objetivos se realizará en función de las rutas planificadas, que discurrirán a través de los segmentos que presenta el diagrama de Voronoi. Esta aproximación permite elaborar una estrategia de cooperación convirtiéndola en un problema con grado de complejidad asumible. De otra manera, tanto la planificación de trayectorias entre dos puntos determinados como la asignación de objetivos a los UAVs resulta de una complejidad intratable con métodos matemáticos.

Una vez resueltas la asignación de objetivos y la planificación de las trayectorias óptimas en un tiempo mínimo se puede abordar la interceptación simultánea de los objetivos. La cuantificación de la longitud y del grado de peligro de los segmentos por los que transcurre una trayectoria formarán parte como variables de coordinación de la función de cooperación que resuelva el objetivo temporal de las llegadas simultaneadas de los diferentes UAVs a sus objetivos espaciales.

Si se opta por el criterio de alcanzar la coordinación óptima, la determinación de la mejor solución temporal (objetivo para cada UAV, ruta seleccionada y tiempo de interceptación del objetivo) exige el conocimiento en cada momento del escenario y cómo este afecta a cada vehículo aéreo. La consecución de la solución óptima, incluso con pocos participantes, requiere de un excesivo intercambio de información que genera un ancho de banda no asumible en el diseño del sistema automático de coordinación. Además la complejidad matemática de los algoritmos de coordinación es excesiva y dificulta la obtención de la solución global en un tiempo admisible.

Se sustituye la búsqueda de la solución óptima por una ‘solución adecuada’ que satisfaga los objetivos de coordinación planteados para el conjunto de múltiples UAVs. La determinación de la solución adecuada se llevará cooperativamente entre los diferentes UAVs y ciertos componentes centralizados.

Una vez coordinadas temporalmente las trayectorias entre cada UAV y su objetivo, resultado de la solución adecuada al problema de la planificación, asignación e interceptación simultánea, cada UAV debe adecuar la ruta seleccionada a sus especificaciones técnicas (apartado 1.4.). El Generador de Trayectoria proporciona una trayectoria de referencia que accionando los mandos el Controlador del Sistema permite el seguimiento fiel de dicha trayectoria que le permite acceder al objetivo en el tiempo de interceptación coordinado por el sistema.

La Arquitectura del Sistema presenta los módulos enumerados en el objeto del proyecto con una estrategia de comunicación descrita a grandes rasgos en el apartado presente y que determina su carácter automático. El diagrama bajo estas líneas describe el ámbito en el que se sitúan los módulos y la comunicación entre ellos.

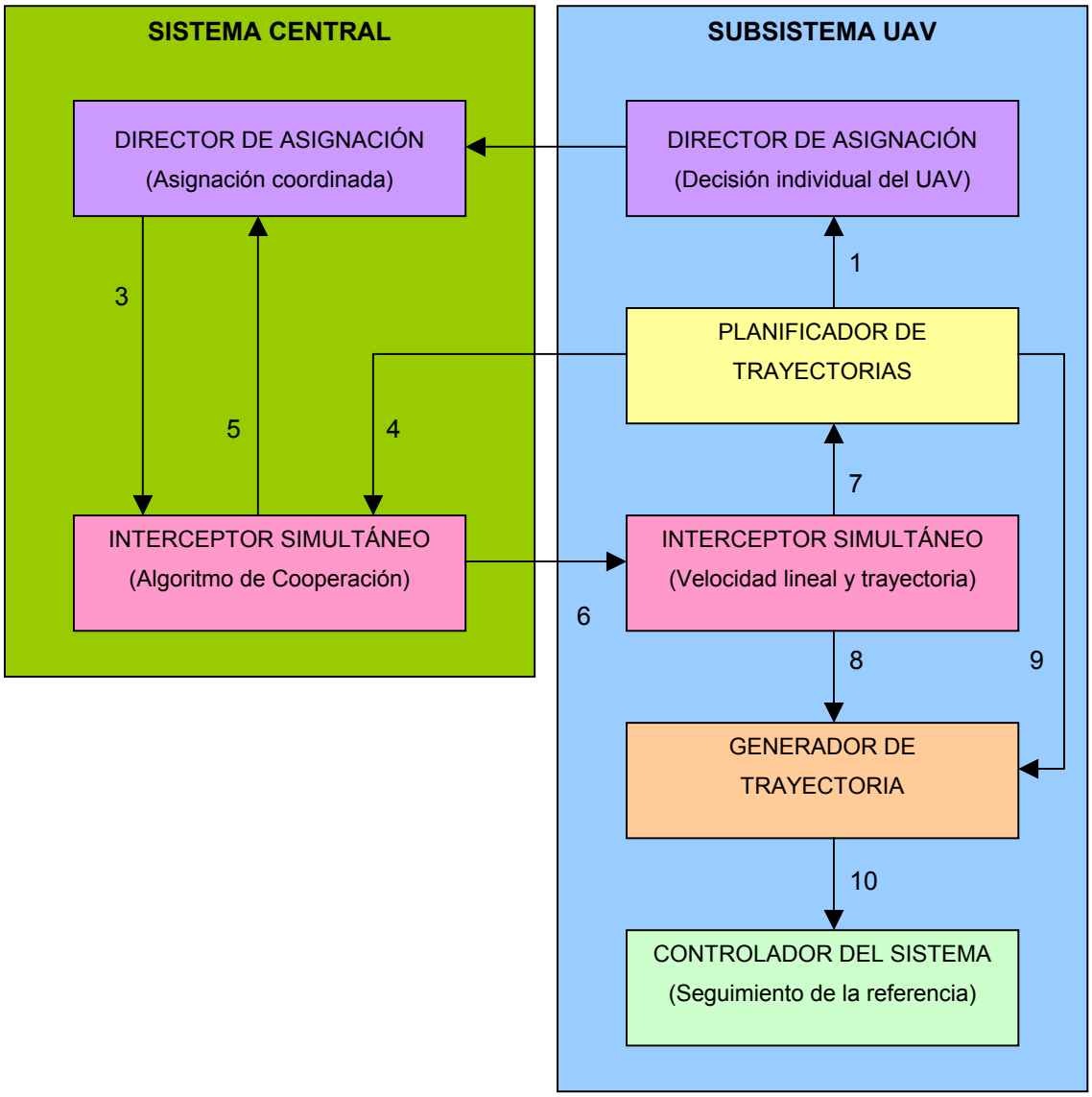

Figura 2: Arquitectura del Sistema automático de coordinación de múltiples UAVs.

Como aparece en el diagrama existen cinco módulos exceptuando al Director de Comunicaciones cuya presencia se intuye a través de las flechas (diálogo entre los módulos). El Planificador de Trayectorias, el Director de Asignación, el Interceptor Simultáneo, el Generador de Trayectoria y el Controlador del Sistema se sitúan bien en el sistema central o en los subsistemas UAVs y se relacionan en diferentes momentos de la ejecución del modelo. Se enumeran estos instantes:

1. Planificador→D.Asignación-UAV: se comunican los costes de longitud y de exposición al peligro para tomar la decisión individual de cada vehículo.

2. D.AsignaciónUAV→D.Asignación-Central: decisiones individuales de todos los UAVs para que el Director de Asignación coordine la decisión coloctiva.

3. D.Asignación-Central→Interceptor-Central: se comunica la asignación coordinada (objetivos para los UAVs).

4. Planificador→Interceptor-Central: comunicación de los costes de longitud y exposición al peligro para aplicar el algoritmo de cooperación.

5. Interceptor-Central→D.Asignación-Central: en el caso de imposibilidad de sincronización temporal con la asignación en curso se requiere otra (3).

6. Interceptor-Central→Interceptor-UAV: información para determinar la velocidad lineal y el número de ruta de las determinadas por Planificador de Trayectorias.

7. Interceptor-UAV→Planificador: número de ruta de las planificadas por el UAV al objetivo finalmente asignado.

8. Interceptor-UAV→Generador: velocidad lineal para cumplir el tiempo de interceptación al objetivo.

9. Planificador→Generador: colección de segmentos a partir de los cuales se determina la trayectoria de referencia.

10. Generador→Controlador: se comunica la trayectoria de referencia para que el controlador del sistema la siga con el mayor grado de fiabilidad.

1.4. La dinámica interna del UAV.

Para dar cumplimiento con la solución adecuada que resuelve la coordinación de las misiones de los UAVs en su camino hacia los objetivos, se tienen en cuenta las especificaciones técnicas de los vehículos. Se considera que éstos vienen equipados con pilotos automáticos capaces de desarrollar el seguimiento de la trayectoria. Dejando a un lado el seguimiento de un determinado perfil de altitud, la acción coordinada afecta sólo al control de la trayectoria en el plano horizontal.

El vector del estado del UAV i viene determinado por las coordenadas X_i e Y_i, el ángulo de dirección respecto del eje *x* Ψ_i y la velocidad transversal V_i, así como la altitud del UAV y la primera derivada de la misma:

$$UAV \leftarrow \left[X_i \quad Y_i \quad V_i \quad \Psi_i \quad H_i \quad \overset{\circ}{H}_i \right]$$

La dinámica interna del UAV asumiendo altitud constante viene dada por las expresiones que siguen (variables que afectan al plano horizontal de movimiento):

$$\begin{aligned} X_i &= V_i \cdot \cos(\Psi_i) \\ Y_i &= V_i \cdot sen(\Psi_i) \\ \overset{\circ}{\Psi} &= \alpha_\Psi \cdot (\Psi_i^c - \Psi_i) \\ \overset{\circ}{V}_i &= \alpha_V \cdot (V_i^c - V_i) \end{aligned}$$

Se asume la presencia de pilotos automáticos para el control de la velocidad y del ángulo de dirección, V_c y Ψ_c, respectivamente. Asociado a esos pilotos automáticos aparecen dos parámetros, α_v y α_ψ cuyos valores son constantes, pero dependen de las prestaciones de vehículo aéreo no tripulado. De las especificaciones técnicas de éste también depende las limitaciones de la velocidad y de la variación del ángulo de dirección respecto al eje *x* en el tiempo (parámetros *c*, V_{mn} y V_{mx}).

$$\begin{aligned} -c \le \overset{\circ}{\Psi}_i \le c \\ V_{mn} \le V_i \le V_{mx} \end{aligned}$$

El control de la altitud del UAV se ha desacoplado del resto de variables. Se tendrá en cuenta sólo en el caso en que se detecten posibles colisiones entre los UAVs. En ese caso se asignarán altitudes diferentes a los vehículos implicados. Existe un piloto automático para el control de la altura, H_c, los parámetros constantes dependientes del vehículo en cuestión α_H y α_H' La dinámica viene dada por la expresión:

$$\ddot{H}_i = -\alpha_{\dot{H}} \cdot \dot{H}_i + \alpha_H \cdot \left(H_i^c - H_i\right)$$

Tanto las variables como las funciones de coordinación tienen que tener en cuenta todos los aspectos relativos a la dinámica interna de los UAVs. Cuando sean codificados los procedimientos encargados de la coordinación temporal de las misiones encomendadas a los vehículos aéreos no tripulados se tiene que permitir que los parámetros sean configurados según las prestaciones de funcionamiento de los UAVs.

El valor de los parámetros que aparecen en las distintas expresiones depende del tipo de UAV que se utilice en el sistema. Sin pérdida de generalidad se toman los datos correspondientes a una aeronave *Boeing X-45A*. En la siguiente tabla aparecen los valores que se van a considerar a lo largo del documento.

V_{mn}(km/s)	0.121
V_{mx}(Km/s)	0.148
Ψ_{mx}'(deg/s)	20
α_v	0.2
α_Ψ	1.33
α_H	0.3364
α_H'	1.4680

Tabla 1: Boeing X-45A

1.5. Organización de la memoria.

A lo largo del *Capítulo 1* se han descrito los aspectos fundamentales que permiten comprender el desarrollo de este proyecto fin de carrera: la definición de vehículo aéreo no tripulado y las características de un sistema automático de coordinación moderno, las exigencias del sistema automático de coordinación de múltiples UAVs especificado en el objeto del proyecto y la arquitectura funcional de la solución adoptada. Por último la situación de partida para el diseño y la dinámica interna del UAV considerado.

En el *Capítulo 2* se describen los principios de funcionamiento del Planificador de Trayectorias. Se detallan los procesos que permiten la elaboración del diagrama de Voronoi y la búsqueda eficiente de las K trayectorias. Se fijan los criterios que rigen la búsqueda a partir del escenario de partida. Por último se analiza la estructura del software del módulo que implementa los procedimientos que lo desarrollan.

El *Capítulo 3* se destina al análisis de la asignación de objetivos. Se estudia la intervención del UAV en el proceso, a través de la decisión individual en sus dos versiones (método de las distancias o de evaluación de las rutas). La decisión colectiva tiene en cuenta la necesidad de coordinación global del conjunto de UAVs motivo por el cual interactúa con el anterior y siguiente módulo. Por último se detallan las funciones que implementan el Director de Asignación.

En el *Capítulo 4* se estudian las acciones encargadas de la coordinación temporal del conjunto de UAVs. Es el Interceptor Simultáneo el encargado de dicha tarea, para la que implementa el algoritmo de cooperación. Se proponen dos métodos diferenciados, el método de velocidades lineales y el de tiempos de interceptación. Para cada uno de estos métodos se detalla la codificación realizada.

El *Capítulo 5* describe la generación de trayectorias que consiste en la adecuación de las mismas que exige la dinámica interna de los UAVs, que presentan limitaciones en la variación máxima del ángulo de dirección y en la velocidad máxima que pueden desarrollar. Se afronta desde el Generador de Trayectorias la parametrización en el tiempo de la trayectoria de referencia gracias a las funciones y modelos en Simulink que se lo codifican.

El *Capítulo 6* se destina al estudio del seguimiento de la trayectoria de referencia. A partir de la solución teórica al problema se describe el Controlador del Sistema, que se ha programado gracias a Simulink. Se ha integrado en un mismo modelo los controladores de los planos horizontal y vertical, cuyo funcionamiento se encuentra desacoplado. Se recuerda que la coordinación de los múltiples UAVs se calcula en el plano horizontal de funcionamiento, asumiendo altitud constante. La altura de los UAVs sólo se utiliza en el caso en que dos vehículos tengan trayectorias concurrentes en determinados puntos

El *Capítulo 7* corresponde a la descripción de la estrategia de comunicación que permite al sistema operar de forma automática para coordinar al conjunto de múltiples UAVs. Se establece un sistema de ficheros que emula la comunicación entre los módulos gracias a operaciones de lectura y escritura. No se analiza la comunicación inalámbrica entre vehículos. Se tiene en cuenta que la cooperación entre módulos se produce entre los dos ámbitos de funcionamiento, el sistema central y los múltiples subsistemas UAV. Se detalla la información contenida en el sistema de ficheros de cada uno de ellos y el intercambio de información que arbitra el Director de Comunicaciones.

El *Capítulo 8* resume las conclusiones a las que lleva el presente proyecto y las líneas de posibles desarrollos futuros. En este punto acaba la descripción de los componentes que integran la solución propuesta excluyendo el detalle del código fuente que la desarrolla.

En el *Anexo A* se adjunta el código fuente de los algoritmos y procedimientos que implementan el proyecto fin de carrera. El anexo se ordena por módulos precedido por los programas encargados de la gestión de los distintos subsistemas (el sistema central y los subsistemas UAV). La descripción de cada una de estas funciones se aborda desde la exposición que cada módulo ha realizado del software codificado.

Capítulo 2: Planificación de Trayectorias.

Todos los procedimientos encargados de resolver el problema de la planificación de trayectorias se integran en el módulo conocido como '*Planificador de Trayectorias*', que se sitúa a bordo de los diferentes UAVs.

En el escenario en el que desarrolla su actividad el Planificador de Trayectorias se cuenta con los N_v vehículos aéreos no tripulados del sistema, N_o objetivos registrados y N_p amenazas que el sistema tendrá que evitar. El UAV debe llegar a su objetivo reduciendo al máximo la exposición a las amenazas y procurando el mínimo consumo de combustible. Para ello debe pasar lo más alejado posible de las amenazas y escoger entre los posibles recorridos los que presenten una menor distancia.

El problema de la planificación de trayectorias se formula en el plano horizontal del espacio, descartando cualquier orden que tenga que ver con la altura que más tarde desarrolle el UAV. El escenario de trabajo queda descrito mediante las componentes *x* e *y* de las posiciones de los distintos agentes. El Planificador de Trayectoria dispondrá a la entrada de los datos referidos a la posición de todos los elementos registrados (UAVs, objetivos y amenazas). La misión encomendada al módulo es la entrega de una serie de trayectorias que faciliten al vehículo aéreo no tripulado el alcance del objetivo que le haya sido asignado por el Director de Asignación.

No hay que perder de vista que el Planificador de Trayectorias es un módulo equipado a bordo del UAV. Este aspecto permite a cada uno manejar la información de forma independiente. El primer hecho es la interpretación del espacio. Está claro que todos los obstáculos constituyen una amenaza para todos los UAVs. También convienen ser evitadas las ubicaciones de los diferentes objetivos, que serán alcanzadas únicamente por los vehículos involucrados por el sistema en su ataque.

2.1. Fases operativas del planificador.

El Planificador de Trayectorias juega un papel muy importante en la determinación de la solución adecuada por parte del sistema automático de coordinación de múltiples UAVs. Para cada UAV dicha solución está compuesta por la velocidad lineal y la trayectoria.

$$\mathrm{E}_i(x_i) = [V_i \quad T_i]$$

El Planificador de Trayectorias genera un conjunto de posibles rutas entre la posición del UAV y un determinado objetivo, del que se seleccionará la trayectoria final T_i, compuesta por un conjunto de puntos

$$T_i = \{t_{i1}, t_{i2}, ..., t_{ip}\}$$

En la planificación de las trayectorias se tienen en cuenta la posición de partida del UAV en el escenario, la localización del objetivo asignado (destino del vehículo) y el conjunto de elementos amenazantes para el vehículo aéreo no tripulado.

$$x_i = [z_{i0} \quad z_{if} \quad P]$$

El Planificador de Trayectorias es el primer procedimiento ejecutado cuando se desea cumplir con una determinada misión. A partir del conocimiento de las localizaciones amenazantes y de los objetivos detectados el Planificador realiza las siguientes fases:

- ✓ Generación del diagrama de Voronoi que permita la planificación de trayectorias a partir de dicho grafo.

- ✓ Asignación de costes a los segmentos del diagrama para hacer posible la búsqueda de trayectorias de acuerdo con ciertos criterios.

- ✓ Búsqueda de las K rutas más cortas entre el UAV y el objetivo en cuestión. Se puede desarrollar gracias a las operaciones anteriores.

2.1.1. Generación del diagrama de Voronoi.

De las infinitas posibilidades que ofrece el escenario a cada UAV para transitar hacia su objetivo, el Planificador de Trayectorias debe seleccionar aquellas que no pongan en peligro la integridad del sistema compuesto por todos y cada uno de los vehículos, y que además no impidan la eficacia de la misión encomendada al sistema.

El aspecto crítico es la presencia de una serie de agentes peligrosos para el vehículo aéreo no tripulado (los obstáculos y objetivos). Las rutas seleccionadas por el Planificador de Trayectorias, evitarán el paso por las cercanías de estas localizaciones, pues representan un riesgo muy elevado de choque con un elemento físico, detección no deseada o colisión con otro vehículo. La primera tarea consiste en establecer el mecanismo de seguridad ante estos obstáculos.

La presencia de localizaciones peligrosas para el UAV exige el tratamiento del escenario de trabajo. En el marco general en que se encuadra el objeto del presente proyecto, no se ocupa del estudio pormenorizado de los obstáculos que afectan al sistema, por ello las ubicaciones peligrosas para el UAV serán consideradas puntuales, aunque algunas de ellas puedan tener formas y extensiones muy diferentes. Es una hipótesis simplificativa que permite el diseño del planificador de trayectorias atendiendo a otras necesidades, algunas impuestas por el resto de los módulos del sistema.

Los obstáculos se registran en el sistema en tiempo real, luego se descartan planificaciones de trayectorias basadas en estudios estadísticos de la presencia de agentes peligrosos. Estos no son necesarios ya que la información es real y actualizada. El sistema dispone de los elementos que le permiten disponer de esta información, y evitar el tratamiento del espacio en base a registros de experiencia previa que den lugar a consideraciones estadísticas de diferentes regiones que dividan el escenario.

Los elementos amenazantes para cada uno de los vehículos aéreos no tripulados (obstáculos y objetivos) se consideran puntuales y ocurren en el instante de creación del diagrama. No existe medida del daño que puedan producir, razón que lleva a tratarlas de la misma manera. El diagrama de Voronoi permite a cada vehículo planificar por separado las trayectorias factibles para él.

Existe una herramienta matemática muy potente, aplicada en campos muy diversos de la ciencia y la técnica (biología molecular, organización, topografía), el diagrama de Voronoi. Consiste en la división del plano en regiones, tantas como elementos amenazantes concurran en el momento de elaboración del diagrama. Cada UAV genera su propio diagrama de Voronoi, con los datos registrados en el sistema.

Las regiones las limitan las aristas del diagrama. Algunas regiones de Voronoi son cerradas pero otras, las correspondientes a las amenazas cercanas a la frontera del escenario, son ilimitadas. Las líneas del diagrama se denominan segmentos de Voronoi cuando están limitadas por la intersección con otras líneas, o rayos de Voronoi en el caso de intersectar con otras líneas sólo en un extremo. Una línea, ya sea segmento o rayo, está compuesto por un conjunto de puntos que equidistan por lo menos de dos de las amenazas vecinas. Con toda seguridad la equidistancia corresponderá a las dos localizaciones peligrosas más próximas.

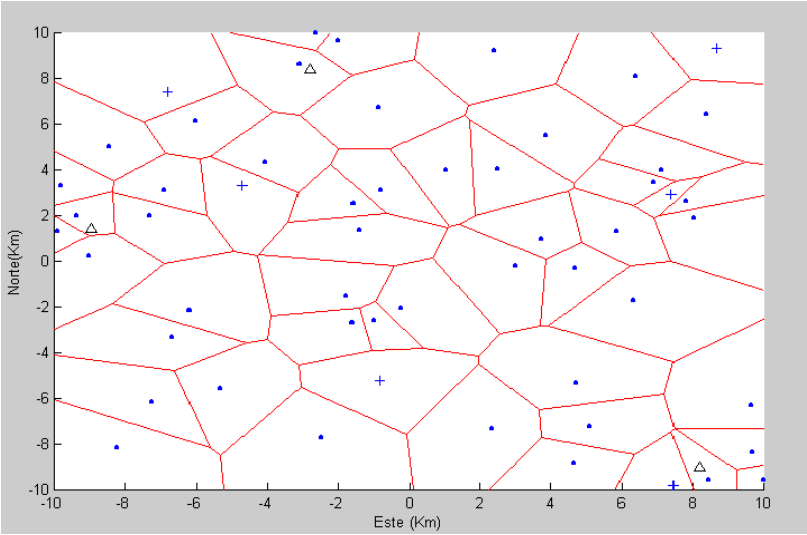

Figura 3: Diagrama de Voronoi de un escenario concreto.

En el ejemplo anterior se parte de un escenario de trabajo limitado a un cuadro espacial de lado 20 Km y en el que se encuentran situados los siguientes elementos que determinan el diagrama de Voronoi.

- ✓ *3 UAVs* confinados en la región que rodea el escenario a 2 Km (**^**).
- ✓ *6 objetivos* o misiones (**+**).
- ✓ *48 localizaciones amenazantes* (**o**)

Si se exige a los UAVs transitar a través de los puntos incluidos en el conjunto de puntos que conforman el diagrama de Voroni, se consigue simplificar muy significativamente el problema, al pasar de un número infinito de posibles trayectorias entre UAV y objetivo a un conjunto finito. Además se hace máxima la distancia con respecto a las dos localizaciones amenazantes más próximas. Una trayectoria se forma a partir de una sucesión de líneas de Voronoi (segmentos o rayos), que compartan los extremos, de manera que permita la llegada hasta la región correspondiente al objetivo asignado.

Habrá que modificar el diagrama de Voronoi de cada UAV introduciendo segmentos adicionales desde el UAV hacia los vértices de la región de Voronoi en la que se encuentre, y entre el objetivo planificado y los vértices de su región. De esta manera se hacen accesibles estos elementos desde el diagrama. El Planificador de Trayectorias seleccionará las colecciones de segmentos y rayos que permitan al UAV llegar al objetivo de la forma más satisfactoria para las especificaciones del sistema.

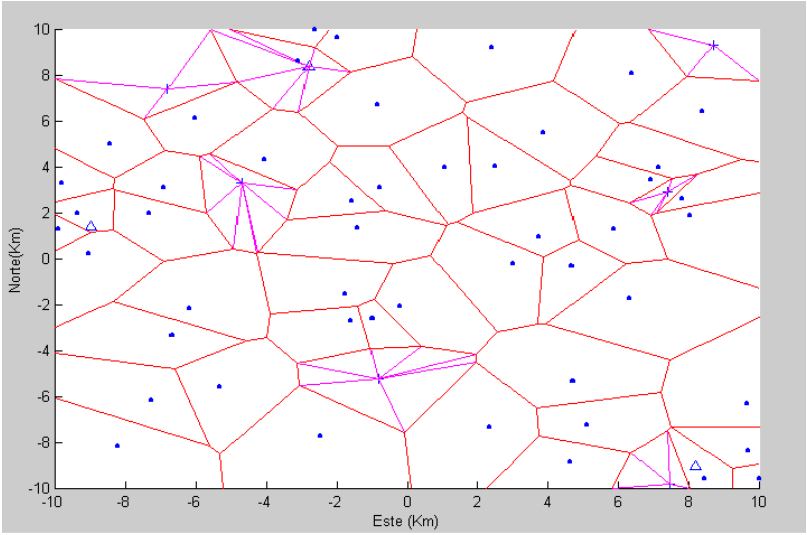

Figura 4: Diagrama de Voronoi para un UAV.

Resumiendo, el UAV parte de su localización inicial hasta un vértice de Voronoi de la región del diagrama en la que se encuentra, a través de una sucesión de segmentos o rayos de Voronoi llega hasta un punto perteneciente a los lados que definen la región que envuelve al objetivo, al que accede a través de otro segmento. En su camino hacia el objetivo se garantiza que el UAV se encuentre en cada momento a igual distancia de las dos localizaciones amenazantes más próximas. Queda por establecer el criterio de selección de estos segmentos de Voronoi.

2.1.2. Asignación de costes a los segmentos de Voronoi.

Una vez definidos los puntos que puedan formar parte de las trayectorias de salida habrá que seleccionar los más ventajosos para los UAVs desde el punto de vista individual y colectiva. La selección de trayectorias consistirá en la elección de la sucesión de lados de Voronoi que permitan la llegada del UAV al objetivo.

El lado de Voronoi es el objeto de la selección. Se etiquetan asignándoles un coste a cada uno de ellos. El coste total de cada lado de Voronoi viene determinado por dos costes, que miden los dos aspectos que preocupan en el sistema, el gasto de combustible para el UAV y el grado de exposición al peligro que sufre a su paso por un determinado lado de Voronoi:

- ✓ Gasto de combustible: El gasto en combustible que sufre un UAV al recorrer un lado depende en primera aproximación de la longitud de este (relación directamente). A partir de este punto, el gasto de combustible se denomina Coste de Longitud.

- ✓ Grado de exposición al peligro: Depende de la distancia desde cada segmento a los diferentes elementos amenazantes (obstáculos y objetivos) para el vehículo, tratándose de una relación inversa (a mayor distancia menor exposición a peligro).

Los Costes de Longitud y de Exposición al Peligro de cada segmento S se expresan en las siguientes expresiones.:

$$C_L^S = L_S$$

$$C_P^S = L_S \cdot \sum_{p=1}^{P} \left(\frac{1}{D_{1/6,S,p}^4} + \frac{1}{D_{1/2,S,p}^4} + \frac{1}{D_{5/6,S,p}^4} \right)$$

El coste de longitud corresponde a la longitud del segmento. Para el coste de exposición al peligro se evalúa la influencia de todas las amenazas. El orden cuarto de esta relación inversa con la distancia amenaza-segmento describe la 'firma radar' (amenaza). En lugar de integrar la influencia de cada amenaza para todos los puntos del segmento, se evalúa la influencia en los puntos 1/6, 1/2 y 5/6 del segmento (puntos significativos del arco), reduciendo sustancialmente la complejidad de cálculo.

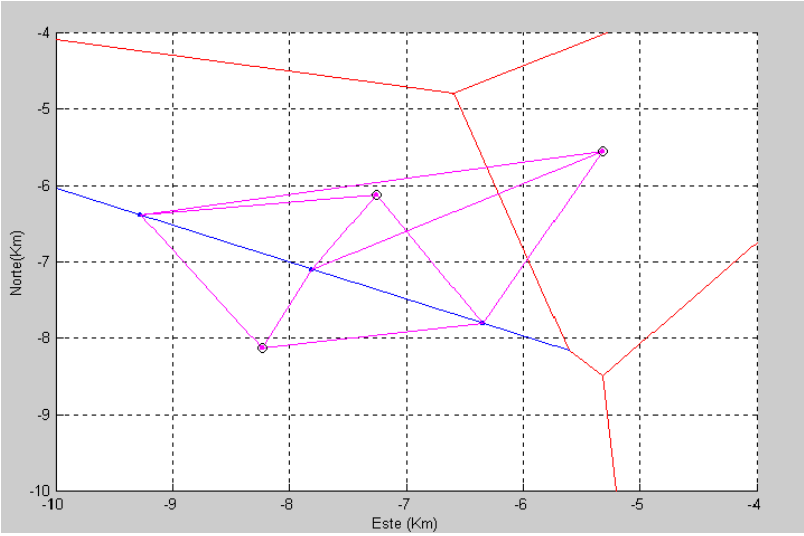

Figura 5: Cálculo de los costes del segmento de Voronoi.

En este ejemplo se comprueba como las tres localizaciones presentes en el diagrama afectan al segmento que se está computando (azul). El coste de longitud del segmento corresponderá al módulo mientras que el coste de exposición al peligro lo determinan las distancias entre las localizaciones peligrosas y los puntos intermedio por un lado y los situados a un sexto de los extremos.

El Coste Total del segmento es la suma ponderada de los dos anteriores, pudiéndose dar mayor importancia a uno que a otro. El coste total se calcula mediante la suma de del coste de longitud multiplicado por este parámetro y del coste de exposición al peligro multiplicado por la diferencia hasta la unidad del parámetro. Si *c* es inferior a *0.5* se prima el segundo y en caso contrario se valora más el de longitud.

$$C_S = c \cdot C_L^S + (c-1) \cdot C_P^S$$

El Planificador de Trayectorias cuenta con la forma de evaluar las posibles trayectorias. El coste que experimenta un UAV en su camino hacia el objetivo será la suma de los costes de los lados del diagrama de Voronoi modificado (con las uniones del UAV y objetivo) que le permitan alcanzar el objetivo. Si S_T es el conjunto de segmentos que forman parte de una trayectoria *T*, el coste de la trayectoria es la suma de los costes individuales de los segmentos del conjunto *S*:

$$C_T = \sum_{S=1}^{|S_T|} C_S, \forall s \in S_T$$

2.1.3. La búsqueda de las trayectorias menos costosas.

A partir del diagrama de Voronoi, con todos sus lados etiquetados con el coste, se aplica una búsqueda de trayectorias entre el origen (posición del UAV) y destino (el objetivo). Se aplica un algoritmo de búsqueda de los caminos más cortos en el grafo. Entre todos los métodos el más conocido es el algoritmo de *Dijkstra*, eficiente si se quiere conocer la ruta más corta entre el origen y destino pero computacionalmente inaceptable para la determinación de las *K* trayectorias más cortas.

La búsqueda de caminos es una tarea ardua, pues el número de vértices del grafo de Voronoi es alto. Se aplicarán técnicas que permitan la búsqueda de las *K* rutas más cortas entre el origen y el destino de una manera eficiente, sin implicar un coste computacional demasiado alto. El Método de Búsqueda de Eppstein plantea una solución muy apropiada y rápida a este problema. En el *apartado 2.3*. se analizan los fundamentos teóricos de este método para la determinación de las *K* rutas más cortas.

A la salida del Planificador de Trayectorias se obtiene un conjunto de *K* rutas entre el vehículo aéreo no tripulado y el objetivo, compuestas por sucesiones de lados del diagrama de Voronoi. Las K trayectorias se almacenan en orden descendente según el coste total de trayectoria. Para el UAV_i, el resultado puede expresarse así:

$$T(x_i) = \{T_{i1}, T_{i2}, ..., T_{ik}\}$$

A partir del resultado anterior se forman determinadas variables de coordinación que se van a emplear en otros módulos del sistema con el fin de tomar las decisiones del conjunto. La intervención del Planificador afecta a los siguientes:

- ✓ Asignación de objetivos: el Planificador de Trayectorias calcula las *K* rutas más cortas entre el correspondiente UAV y cada uno de los objetivos en el sistema. Los costes de longitud y de peligro de cada una de las rutas trazadas en el planificador son variables de coordinación para la asignación de objetivos.

- ✓ Interceptación Simultánea: las longitudes y los costes totales de las *K* trayectorias entre el UAV y el objetivo asignado en principio son las variables de coordinación hacia el Interceptor Simultánea que aplicará la función de coordinación temporal para todos los múltiples UAVs del sistema.

2.2. El diagrama de Voronoi.

2.2.1. Un número finito de posibles trayectorias para el UAV.

El Diagrama de Voronoi divide el escenario de trabajo en N_p+N_o celdas trazadas de manera que el centro de estas regiones poligonales corresponde a la localización del agente amenazante (obstáculo u objetivo). La primera virtud del grafo de Voronoi es la reducción del problema de la determinación de las rutas, pasando de infinitas posibilidades a un conjunto discreto de posibilidades.

Los segmentos que describen estos polígonos se caracterizan porque equidistan al menos de las dos amenazas más próximas. Al exigir a los vehículos aéreos no tripulados recorrer el camino hacia el objetivo a través de una sucesión de segmentos de los polígonos de Voronoi se garantiza en cada momento el paso del vehículo lo más alejado posible de los peligros registrados en el escenario de trabajo.

El diagrama de Voronoi tiene su dual en la Triangulación de Delaunay, correspondiente a la unión de localizaciones peligrosas más próximas. A lo largo del apartado se explica el procedimiento que permite la determinación del diagrama de Voronoi. Se toma como referencia el método de Fortune. A partir de este método, introduciendo algunas modificaciones se consigue mejorar el rendimiento temporal de la determinación del diagrama.

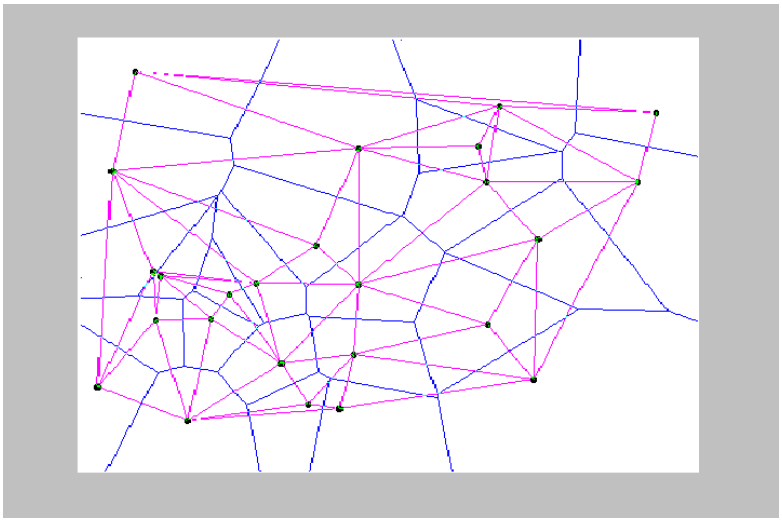

Figura 6: Diagrama de Voronoi y su dual (Triangulación de Delaunay).

2.2.2. El método de Fortune.

El diagrama de Voronoi es una herramienta matemática fundamental para codificar la información más próxima. Fortune desarrolla una implementación en C muy apropiada para la planificación de trayectorias basada en la determinación de los diagramas de Voronoi y Delaunay que proporcionan al UAV un conjunto de puntos a partir del cuales se puede generar una trayectoria. Las funciones del procedimiento son la determinación de los vecinos más próximos, el par más cercano y la celda convexa.

El diagrama de Voronoi es una herramienta matemática muy empleada en diversos campos de la ciencia y de la técnica. Se trata de un diagrama planar hecho a partir de un conjunto de localizaciones en el plano, que son envueltas por su correspondiente región que contiene los puntos que se encuentren más próximos a esa localización que al resto. La frontera de estas regiones son los llamados lados de Voronoi, formados por los puntos que equidistan respecto a las dos localizaciones vecinas. Los lados intersectan en los vértices de Voronoi, caracterizados por equidistar de tres o más localizaciones próximas.

El diagrama, por tanto, codifica la información más cercana en el espacio y es de gran utilidad en problemas geométricos tales como la búsqueda del elemento más cercano para cada localización, la determinación del par más próximo para cada localización o la elaboración de la celda convexa. El diagrama dual al de Voronoi es la triangulación de Dalaunay que encuentra sus aplicaciones en campos tan diversos como la topografía o la organización de ventas.

De entre todos los métodos, el de Fortune tiene frente a otros la ventaja de hacer de la premisa ‘divide y vencerás’ la filosofía de la implementación, lo que permite para *N* localizaciones llevar a cabo el diagrama en un tiempo *~O(NlogN)* y facilitar la mejora de los algoritmos de forma rápida y sencilla.

El principio de funcionamiento se basa en barridos a lo largo de una línea recta y en la exploración a lo largo de un conjunto de localizaciones. El movimiento que genera este barrido es el que permite la construcción del diagrama, técnica conocida como ‘Planificación por Barrido’, muy presente en otras aplicaciones geométricas.

Dos son las estructuras de datos en la planificación por barrido, la cola de eventos (también conocida por cola de prioridad) y la tabla de barrido (o de entradas):

- ✓ Cola de eventos: contiene una secuencia de caracteres lexicográficos que ordenan los eventos que indican el momento en que el barrido debe detenerse.

- ✓ Tabla de barrido: dispone de una sección transversal que registra la manera en que la línea de barrido reconoce las localizaciones y genera las regiones y los lados del diagrama de Voronoi. La tabla se actualiza con la aparición de cada evento.

El algoritmo de Fortune desarrolla transformaciones geométricas que modifican los diagramas de partida. El diagrama de Voronoi y su correspondiente de Delaunay se calculan simultáneamente gracias a las primitivas del método y a los cálculos que hacen uso de la aritmética en punto flotante. Este método genera el diagrama de Voronoi de un conjunto de 10000 localizaciones en un tiempo aproximado de 4 seg en una estación SPARC y utilizando aritmética en punto flotante de doble precisión.

La capacidad de computación del método es lo que ha llevado a su elección. Los requerimientos del sistema nunca serán tan elevados como los de la situación anterior, y el método permitirá la determinación del diagrama de forma rápida que compensará la pérdida de tiempo que se produce en otras tareas en la planificación, como es el caso de la selección de las trayectorias más cortas.

Partiendo de la ordenación de las localizaciones y su separación de los vértices en la cola de eventos, introduciendo diferencias en la forma de representar ciertas estructuras como la tabla de barrido o la cola de eventos se logra reducir el tiempo de cómputo del diagrama. Fortune considera listas apuntadas con un esquema de búsqueda bastante complicado, cuyo rendimiento es sensible al tamaño de la lista.

Se presentará el Método Tess que hace uso de una estructura y un árbol binario de localizaciones que logra una mayor simplicidad sin renunciar a la efectividad del método primitivo. Además se utiliza una representación compacta de la lista de lados del esquema, que tiene en cuenta las conexiones entre ellos de manera que quedan reflejadas todas las relaciones entre lados y vértices.

2.2.2.1. Aspectos preeliminares del método.

Se denota por *R* le conjunto de los números reales y R^2 los correspondientes al plano. Por tanto, un punto *p* de R^2 queda representado por sus coordenadas cartesianas x e y (*px , py*). Si *p* y *q* pertenecen al plano R^2 se dice que *p* < *q* si se verifica que *px* < *qx* y *py* <= *qy*.

Las localizaciones que darán lugar al diagrama son puntos del conjunto R^2. El método emplea conjuntos S de tamaño fijo N. Si una localización pertenece a S, $p \in S$ la distancia euclidiana entre un punto $z \in \Re^2$ a la localización *p* se denota por $d_p(z)$ y a la más cercana de todas *d(z).*

Se define el bisector B_{pq} perpendicular al segmento que une las localizaciones *p* y *q* del conjunto *S*, a la línea $\{z \in \Re^2 : dp(z) = dq(z)\}$ que divide el plano en dos mitades, H_{pq} que contiene el punto *p* y H_{qp} que contiene a la localización *q*. Todos los puntos de H_{pq} se encuentran más cerca de *p* que de *q*, y al contrario ocurre en H_{qp}. Dadas las localizaciones *p* y *q* de *S*, el semiplano H_{pq} se define por $\{z \in \Re^2 : dp(z) < dq(z)\}$, y constituye un conjunto abierto y convexo.

Dada una localización $p \in S$, la región de Voronoi R_p la constituye el conjunto de los puntos de R^2 estrictamente más cercanos a *p* que a cualquier otra de las *N-1* localizaciones de *S*, $Rp = \bigcap_{q \in S, q \neq p} H_{pq}$. R_p es convexa, poligonal e incluye a *p* es su interior, cerrado o abierto (si se define sólo por dos segmentos).

Dos localizaciones se denominan vecinas cuando comparten algún punto los bordes de sus regiones de Voronoi. El diagrama de Voronoi V se define por:

$$\Re^2 - \left(\bigcup_{p \in S} Rp\right) = \{z \in \Re^2 : \exists p, q \subset S, p \neq q, dp(z) = dq(z) = d(z)\}.$$

La expresión matemática que describe el diagrama de Voronoi exige que las líneas que agrupan a los puntos del diagrama equidisten de dos o más localizaciones vecinas. V es la unión de estos segmentos que forman parte de los diferentes bisectores. Un vértice de Voroni es un punto del diagrama V que equidista de tres o más localizaciones vecinas y en el que inciden tres o más segmentos o rayos del diagrama de Voronoi.

Si *p*, *q* y *r* son tres localizaciones cualquiera no colineales, la intersección de las líneas B_{qr}, B_{pq} y B_{pr} es un punto conocido por *vértice candidato*, centro del *círculo candidato* definido para las localizaciones *p*, *q* y *r*. Un círculo candidato que no contenga en su interior a otras localizaciones recibe el nombre de *círculo de Voronoi*, correspondiendo el centro con un *vértice de Voronoi*.

Un lado de Voronoi es un subconjunto de puntos conexos de *V* que corresponde a una de las siguientes circunstancias: línea entre dos vértices de Voronoi, rayo que nace en uno de ellos. Todos los lados de Voronoi son parte de un bisector, por lo tanto hacen referencia a dos localizaciones, las más próximas de las que equidista cualquier punto del lado. El lado de Voronoi se define para dos localizaciones, *p* y *q*, y se denota por ε_{pq} o ε_{qp}. Si el lado corresponde a un rayo, los puntos *p* y *q* pertenecerán a celdas convexas abiertas y las regiones de Voronoi R_p y R_q serán no limitadas.

Es compleja la aplicación directa de la técnica de la planificación por barrido para encontrar el diagrama de Voronoi. Si la línea de barrido intersecta la región de Voronoi de cierta localización, hecho que se registraría en la sección transversal de la tabla de barrido, se encontraría antes la región que la localización correspondiente. Por tanto no se puede saber qué región necesita ser grabada en la tabla de barrido sin el conocimiento previo de su localización.

La solución aportada por Fortune genera una transformación geométrica o mapeo. En ese caso se elabora un diagrama de Voronoi transformado en el que el punto más cercano a una región intersectada por la línea de barrido sea la propia localización. El resto de la región quedará oscurecida por el punto significativo que prevalecerá sobre los demás. El diagrama primal se computa simultáneamente de manera eficiente. La transformación propuesta por Fortune genera continuos mapeos de $\Re^2 \to \Re^2$ definidos de la siguiente manera: $*(z) = \left(z_x, z_y + d(z)\right), \forall z \in \Re^2$.

El mapeo se aplicará sólo determinados bisectores, tras el cual, un lado de *V* se convierte en la rama de una parábola a no ser que sea vertical. Las localizaciones permanecen fijas tras el mapeo. Si ε_{pq} es un lado de Voronoi, al mapearlo $*(\varepsilon_{pq})$ se convierte en una sección de una hipérbola o de una línea vertical. El único punto significativo de la región mapeada $*(R_p)$ será *p*.

El mapeo preserva las relaciones entre lados y vértices de Voronoi del diagrama transformado. Las ramas de la hipérbola correspondientes a diferentes bisectores intersectarán como mucho una vez porque eso es lo que pueden intersectar dos bisectores. En el diagrama transformado **(V)* se hace referencia a los vértices, lados y regiones aunque propiamente sean las imágenes de aquellas. Se analiza el efecto del mapeo sobre el bisector de las localizaciones vecinas *p* y *q*, B_{pq}:

- Si $py \neq qy$ y B_{pq} no es vertical: el mapeo $*(B_{pq})$ corresponderá con la rama superior de la parábola cuyo mínimo se sitúa precisamente en *p*. La rama $*(B_{pq})$ se bifurca hacia los rayos izquierdo o derecho de la parábola conocidos como límites negativos o positivos. C^-_{pq} decrece monotónicamente hacia la izquierda de *p* mientras que C^+_{pq} crece monotónicamente hacia la derecha de *p*. Este esquema de mapeo bifurca B_{pq} en los rayos izquierdo y derecho. El punto mínimo de los bordes C^-_{pq} y C^+_{pq} corresponde a la localización *p* (si es la mayor de las dos).

- Si $py = qy$ y le bisector B_{pq} es vertical, entonces el mapeo $*(B_{pq})$ es un rayo vertical abierto dirigido hacia arriba que comienza en el punto medio de la línea que une *p* y *q* ($(p_x+q_x)/2$, p_y) y que se denota por C^o_{pq}. El punto medio se ha convertido en el mínimo. Lo importante no es la dirección de los bisectores mapeados sino la posición del punto mínimo, por eso no se distingue el supraíndice y se denotan por C_{pq}.

El rayo se ha convertido en un nuevo rayo no limitado, subconjunto del original. El punto más bajo del rayo mapeado se conoce por base, que puede o no corresponder con el mínimo. El segmento se convierte en un subconjunto del borde. El punto más alto se denomina cumbre. Se utiliza la misma notación que la utilizada en los bisectores. A partir de ahora el bisector transformado se va a nombrar 'borde'.

Se comprueba como una línea de barrido horizontal intersecta con el borde correspondiente a un rayo o segmento. La tabla de barrido del algoritmo de Fortune contiene una copia de las regiones transformadas y los bordes de los rayos intersectados por la línea de barrido horizontal. Una vez descritas las estructuras de datos y los elementos que intervienen en el método se detallará el algoritmo.

2.2.2.2. El Algoritmo del método de Fortune.

El algoritmo actúa sobre dos espacios diferentes (el original y el transformado), que contienen ambos los mismos conceptos (localizaciones, regiones, vértices o lados), pero hay que recordar que pertenecen a ámbitos diferentes. Aún así se recuerda que existe relación entre el diagrama original y el transformado. Por lo tanto será importante no perder de vista el contexto que envuelva a cada operación. Cuando se trate de entradas al algoritmo, éstas se referirán al conjunto de localizaciones del conjunto S. Por el contrario las salidas estarán referidas al diagrama de Voronoi transformado, es decir, corresponderán a vértices y lados mapeados.

El Algoritmo de Fortune puede generar rayos semi-infinitos que etiqueta con los vértices transformados de Voronoi que los forman. Los segmentos y los rayos se han mapeado para conectar los subconjuntos de lados de Voronoi del espacio primal, resultado que se conoce como bordes del diagrama transformado.

Las dos estructuras de datos principales son la cola de eventos y la tabla de barrido. La actuación de ambas se produce en estos términos:

- ✓ Cola de eventos: se organiza según la prioridad, y se denota por *Q*. Afecta a las localizaciones transformadas y vértices candidatos a ser procesados en el orden lexicográfico. El evento lexicográfico más pequeño es entregado de manera que la línea de barrido avanza hasta el punto correspondiente.

- ✓ Tabla de barrido: El barrido se detiene momentáneamente en la localización o vértice candidato para introducirlo en la tabla de barrido, una lista ordenada compuesta por bordes y regiones intersectadas por la línea de barrido. El orden lo marca la coordenada *x* de la intersección con la línea de barrido.

El algoritmo de Fortune asegura que el evento rescatado de *Q* corresponde a un vértice de Voronoi si se seleccionan los vértices candidatos de Q cuyos círculos contengan una localización. Fortune detecta el vértice candidato en el diagrama mapeado gracias a cualquier intersección de bordes adyacentes. Si la intersección existe será detectada por la línea de barrido debido al carácter monótono de estos rayos transformados. Al detectar el círculo candidato con el vértice candidato en el centro, las tres localizaciones *p*, *q* y *s* que lo definen se consideran barridas.

El algoritmo proporciona resultados en un tiempo *~O(NlogN)* con operaciones aritméticas exactas, incluso en el caso de presentarse degeneraciones en la ubicación de alguna localización. La degeneración consiste en cuatro o más localizaciones dentro de un mismo círculo, tres o más localizaciones co-lineales o una localización situada sobre un bisector. Las degeneraciones producen lados de longitud cero. Se detectan si coinciden con vértices candidatos, procesándose en cualquier orden.

Para la elaboración paralela del diagrama primal *V* se genera la imagen inversa del borde (rayo o segmento). La unión de todas las imágenes inversas conforma el grafo primal V. La imagen inversa de un borde C_{pq} corresponderá a un subconjunto conexo del bisector B_{pq}. Sin pérdida de generalidad se asume *p* más alto que *q*, denotando la base del rayo C_{pq} por λ, y por μ a la mínima coordenada *y* de las localizaciones. El conocimiento de λ^{-1} se produce a la vez que se crea el borde, y permite discriminar una de las siguientes circunstancias mutuamente excluyentes:

- Si λ corresponde a un vértice transformado, la imagen inversa coincidirá con un vértice del diagrama primal. Por lo tanto se alcanza algún vértice candidato.

- Si C_{pq} representa a un borde no vertical y λ es su punto medio entonces la imagen inversa corresponderá al punto del bisector B_{pq} justo por debajo de *p*. Se considera alcanzada la localización *p*.

- Si C_{pq} es un rayo vertical, λ es el punto ($(p_x+q_x)/2$, μ), su imagen inversa coincidirá con el punto $\left((px+qx)/2,-\infty\right)$. En este caso aparece más de una localización como resultado del barrido.

Al generar C_{pq} se le asigna el nivel σ correspondiente a la localización en el espacio transformado, que tiene su imagen en σ^{-1} en una localización del espacio primal. El subconjunto de B_{pq} que formará parte del diagrama de Voronoi es el formado por la porción entre λ^{-1} y σ^{-1}, generando un segmento de Voronoi (o rayo si se da el tercer caso).

La determinación de este resultado inverso se produce paralelamente a la de C_{pq}, dependiendo de las diferentes situaciones. Si el punto correspondiente a λ^{-1} se encuentra en B_{pq} por debajo de *p*, no hace falta calcularlo.

Para determinar el diagrama de Voronoi a partir del transformado en el mismo tiempo que se evalúa el segundo, se tendrán en cuenta estas circunstancias:

- Si C_{pq} se registra, y $p_y \neq q_y$, el subconjunto de B_{pq} si se trata de un segmento, o el rayo a izquierda o derecha de λ^{-1}, depende de la dirección y el sentido positivo o negativo de C_{pq}.

- Si se registra C_{pq}, y $p_y = q_y = \mu$, el subconjunto de B_{pq} no es tal subconjunto, porque se seleccionan todos los puntos de B_{pq}.

- Si C_{pq} se registra, y $p_y = q_y \neq \mu$, se selecciona el subconjunto de B_{pq} formado por el rayo que parta de λ^{-1}.

El lado de Voronoi ε_{pq}, lo forman dos subconjuntos de B_{pq} con un punto en común λ^{-1}. Se puede emplear un sólo subconjunto, soslayando el cálculo de las coordenadas de λ^{-1}, en el caso de alcanzar una sóla localización con dos bordes C^{-}_{pq} y C^{+}_{pq}. Si se conectan ambos, cualquier cambio que afecte al primero se detecta en el segundo, y viceversa. El vértice de Voronoi asociado a la localización correspondiente al primer borde se considera un atributo que describe al segundo. Si se denota por *t* al vértice en el espacio primal, y C_{pq} al segundo borde en el espacio transformado:

- Si C_{pq} ha sido limitado por el vértice transformado asociado a *v* del espacio primal, el subconjunto de B_{pq} es el segmento limitado por los vértices *t* y *v*.

- Si C_{pq} no ha sido limitado, el subconjunto corresponde al rayo sobre el bisector B_{pq} que parte de *t* y se dirige hacia la izquierda o derecha dependiendo de la dirección de C_{pq} (negativa o positiva).

- Si ninguno de los dos bordes han sido limitados, se seleccionará todo el bisector B_{pq}. Serán los rayos los últimos en introducir en el diagrama a partir de la tabla de barrido, donde se encuentran registrados.

El algoritmo trabaja sólo con los bisectores y bordes del espacio transformado. Esto supone una ventaja ya que se dispone de una memoria limitada. Cuando se van generando partes del diagrama se registran en la memoria.

2.2.3. El método Tess. Una mejora del método de Fortune.

2.2.3.1. La cola de eventos y la tabla de barrido.

Para implementar la cola de evento, se usa un conjunto implícito, una representación de una estructura en forma de árbol binario donde se mantiene el orden en la estructura. Un evento se representa por un puntero genérico a cada conjunto. La representación no hace mención sobre el tipo de dato ni la clase de eventos que se tratan. El identificador es un puntero que permite el acceso inmediato a la estructura en la que se borran y reemplazan los eventos. No existe limitación en el número de eventos que pueden tratarse. Para ganar en simplicidad se utilizan vectores ordenados y estructuras de árboles.

No se mezcla el tratamiento de las localizaciones y los vértices candidatos a formar parte del diagrama de Voronoi. Mientras las primeras se guardan en el vector ordenado, los vértices candidatos se recogen en la estructura. El número de vértices candidatos en cada momento para un conjunto de N localizaciones distribuidas uniformemente se predice en un número del $O(\sqrt{N})$. El tratamiento separado de estos dos conceptos incrementa la rapidez del proceso en un 15%. El diagrama de Voronoi y la Triangulación de Delaunay se lleva a cabo con las estructuras de datos descritas en el apartado anterior.

N	UNIFORME		NORMAL		EXPONENCIAL	
	TEST	FORTUNE	TEST	FORTUNE	TEST	FORTUNE
2000	0.73	0.95	0.72	0.97	0.75	0.97
4000	1.50	1.97	1.55	1.95	1.55	2.03
6000	2.27	2.97	2.30	3.10	2.32	3.12
8000	3.00	4.05	2.98	4.10	3.10	4.23
10000	3.80	5.07	3.80	5.17	3.82	5.37
12000	4.73	6.33	4.73	6.28	4.80	6.50
14000	5.43	7.17	5.53	7.32	5.40	7.70
20000	6.30	8.35	6.20	8.50	6.43	8.83

Tabla 2: Rendimiento en tiempo de los métodos Tess y Fortune.

Son estructuras de datos abstractas, independientes al algoritmo y que en principio pueden soportar otro tipo de datos diferentes a los vértices, rayos, bordes, etcétera. Este aspecto permite la introducción sencilla de mejoras en el método.

2.2.3.2. Las primitivas del método Tess.

Las primitivas del método Tess se agrupan en dos grandes conjuntos, el primero compuesto por las encargadas de detectar y tratar los vértices candidatos, el segundo formado por aquellas que determinan la región que define cada localización. Como el método Tess se basa en el desarrollado por Fortune, la operación consiste en determinar el lado de referencia del borde relacionado con cada localización.

Un vértice candidato del diagrama mapeado es el resultado de la intersección de dos bordes adyacentes de la tabla de barrido. A veces dos bordes adyacentes no llegan a contactar, pero si lo hacen Tess la detecta gracias a la intersección de las dos parábolas resultado de la transformación de los dos bisectores. Se denota por C_{uv} y C_{vw} a dos bordes adyacentes, que cumplen con las siguientes propiedades:

TEOREMA: Si C_{uv} y C_{vw} son dos bordes negativo y positivo, consecutivos en la tabla de barrido, no intersectarán o se dirigirán hacia el mismo bisector.

Si se analiza el número de eventos (localizaciones y vértices candidatos) procesados, después de procesar *k* eventos, se estudia el comportamiento en el caso de incluir el evento *k+1*. Cuando se alcanza una localización, los dos bordes de diferente signo consecutivos y generados por el mismo bisector se insertarán entre otros dos bordes.

Cuando se alcance un vértice *p*, un nuevo borde reemplazará a otros dos. Se pueden dar dos situaciones, la primera es el caso de insertar un borde positivo tras uno negativo, el segundo corresponde al caso de un borde negativo antes que uno positivo.

Los bordes C_{uv} y C_{vw} intersectan sólo si lo hacen los bisectores correspondientes en el espacio primal. En este caso la intersección se produce en el centro *cx,cy* de un círculo a través de las localizaciones *u*, *v* y *w*. El centro del círculo viene determinado por las ecuaciones:

$$\begin{bmatrix} c_x \\ c_y \end{bmatrix} = \frac{1}{2 \cdot a} \cdot \begin{bmatrix} \left(b_1 \cdot \left(v_y - w_y\right) - b_2 \cdot \left(u_y - v_y\right)\right) \\ \left(b_2 \cdot \left(u_x - v_x\right) - b_1 \cdot \left(v_x - w_x\right)\right) \end{bmatrix}$$

Estos son los parámetros que aparecen en la expresión anterior. El método Tess calcula estos parámetros para ahorrar tiempo, ya que se necesitarán más adelante en futuros cálculos:

$$a = \left(u_x - v_x\right) \cdot \left(v_y - w_y\right) - \left(u_y - v_y\right) \cdot \left(v_x - w_x\right)$$
$$b_1 = \left(u_x - v_x\right) \cdot \left(u_x + v_x\right) + \left(u_y - v_y\right) \cdot \left(u_y + v_y\right)$$
$$b_2 = \left(v_x - w_x\right) \cdot \left(v_x + w_x\right) + \left(v_y - w_y\right) \cdot \left(v_y + w_y\right)$$

TEOREMA: Dos bordes consecutivos correspondientes a bisectores distintos intersectarán si y sólo si las localizaciones *u, v* y *w* están dispuestas formando un giro a izquierda, lo que equivale a la condición:

$$\left(v_x - u_x\right) \cdot \left(w_y - u_y\right) - \left(v_y - u_y\right) \cdot \left(w_x - u_x\right) > 0 \rightarrow a > 0$$

La evaluación de esta condición es eficiente y el valor de *a* se puede emplear para calcular la intersección de los bisectores si los bordes equivalentes se cruzan. El teorema 2 es crítico en la construcción de la lista de lados del diagrama.

Si C_{tu} , C_{uv} , y C_{vw} son tres bordes consecutivos. Si los dos primeros intersectan y los dos últimos también, entonces C_{tu} y C_{uv} intersectan, y lo mismo sucede con C_{tu} y C_{uw}, entonces también lo harán C_{tv} y C_{vw}. Esta propiedad proporciona los vértices candidatos que se pueden el eliminar de la cola de eventos por otros más apropiados.

Una vez detectada la intersección del borde, se añade la distancia hacia las tres localizaciones *u, v, w*. En el caso en que tres bordes consecutivos intersecten en dos vértices candidatos, con tan sólo el vértice más pequeño se dispone de la información necesaria. Los otros se pueden borrar de la cola de eventos. De esta manera Tess reduce el tamaño de la cola de eventos.

Cuando se procesa una localización p, el algoritmo ejecuta una búsqueda en la tabla de barrido para localizar la región en la que se encuentra *p*. Se busca en la tabla de barrido de forma binaria y rechaza las regiones en las que *p* no tiene cabida. El método Tess proporciona los bordes anterior y posterior correspondientes a la región.

2.2.4. Diagrama de Voronoi: un ejemplo práctico.

Para la obtención del diagrama de Voronoi se han descrito los pasos a seguir en los apartados destinados a la formulación de los métodos de Fortune y Tess. Resulta interesante simular el proceso de formación del diagrama de Voronoi. Esta es una tarea complicada, pues como se ha detallado en la descripción del Método de Fortune, la determinación de las aristas del diagrama no sucede en el espacio real, sino en el espacio transformado.

Si se cuenta con una serie de localizaciones (puntos en negro) repartidas en el espacio, los elementos en verde tratan de aproximarse al espacio primal, pero se recuerda que el barrido se ejecuta en el transformado (línea de barrido horizontal que se desplaza de arriba hacia abajo y las parábolas que representan a los bisectores). En rojo se muestran los bisectores que entran en escena y en negro los círculos de Voronoi.

En las siguientes ilustraciones la línea de barrido alcanza los mínimos de las parábolas correspondientes a dos localizaciones consecutivas según la componente *y* de su posición. Se comprueba como algunas aristas finales ya han sido trazadas. Los bisectores corresponden a puntos ya barridos. Entre las dos situaciones se percibe una diferencia, la inserción de un bisector. Esta tarea comienza con la llegada de la línea al punto señalado con un cuadrado en la segunda figura (de las dos localizaciones objeto de la comparación la situada a la izquierda):

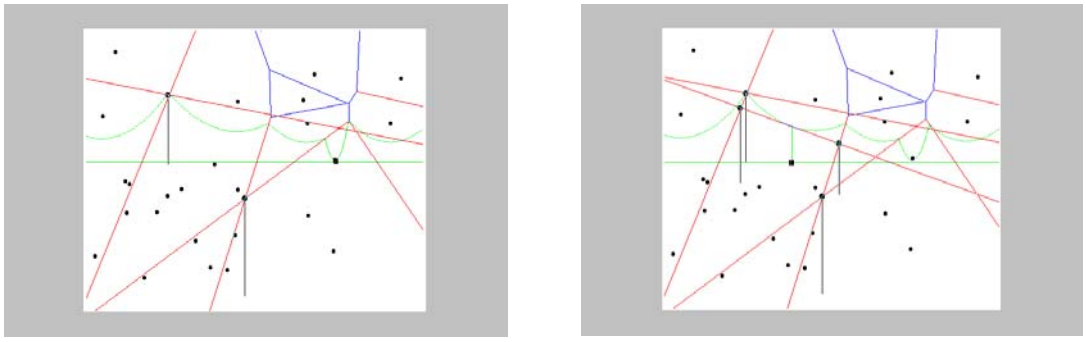

Figura 7: Llegada de la línea de barrido a localizaciones consecutivas.

La llegada de la línea de barrido a la siguiente localización (derecha) provoca la generación de un nuevo bisector, tangente al mínimo de la parábola. En cada caso se han evaluado los bisectores que introduce la detección del evento.

La tarea de escoger los vértices candidatos alcanzado el mínimo de una parábola es complejo. De ella se obtendrán los puntos seleccionados a partir de los cuales se trazan los bisectores correspondientes. Resulta ilustrativo comprobar como determinadas localizaciones (al menos tres en cada caso) constituyen un círculo de Voronoi. Para la primera localización alcanzada se comprueba como forma parte de dos círculos junto con otras localizaciones ya barridas por la línea:

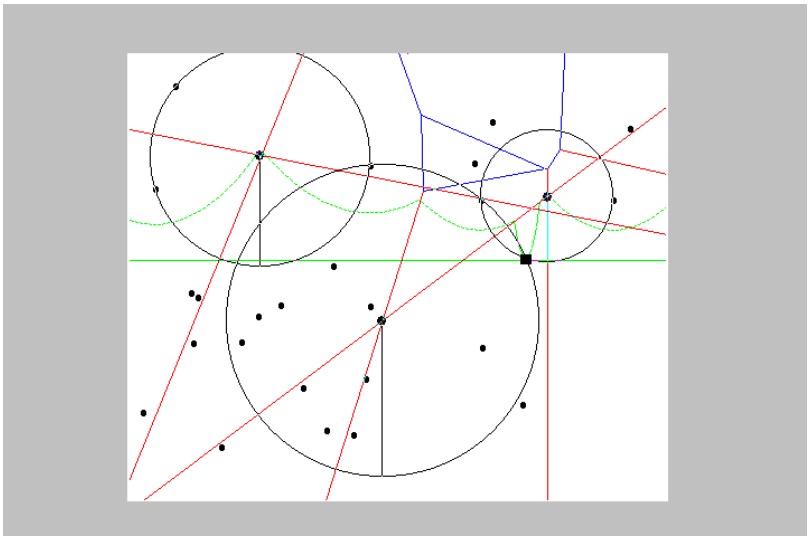

Figura 8: Círculos de Voronoi correspondientes a los bisectores.

Para la segunda localización alcanzada en esta secuencia que se estudia (cuadrado), se observa como forma parte de dos círculos de Voronoi, de los que aparecen otros dos relacionados con el bisector que se ha añadido en este paso:

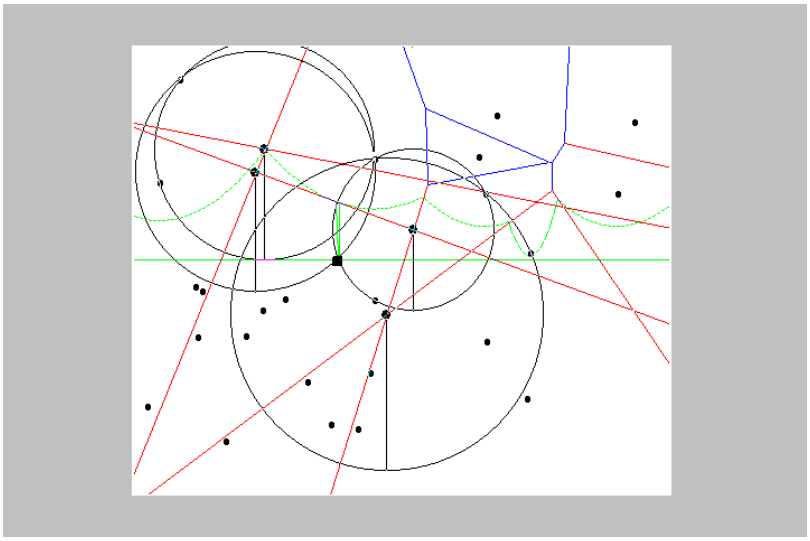

Figura 9: Círculos de Voronoi para los siguientes bisectores.

Si se analiza la primera localización (*Figura 7*), se señala la forma en que se traza uno de los bisectores generados en la fase de barrido que afecta a la localización marcada (aparece ya en *Figura 8*). Se selecciona el círculo de Voronoi más cercano a la línea. Se marca en celeste la distancia desde el centro del círculo a la línea de barrido. El nuevo bisector partirá del vértice candidato correspondiente al centro del círculo, definido además por dos bisectores ya trazados. Pasará por el punto medio de la recta que une la localización y el círculo de la localización de referencia.

Se ilustran una fase muy avanzada (seleccionados muchos subconjuntos de bisectores del diagrama final, en azul) y el diagrama de Voronoi a la salida:

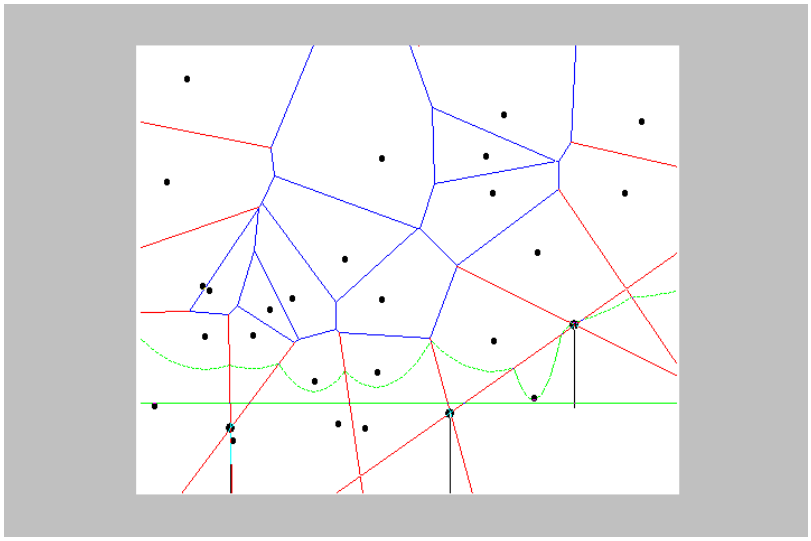

Figura 10: Estado avanzado de elaboración del diagrama.

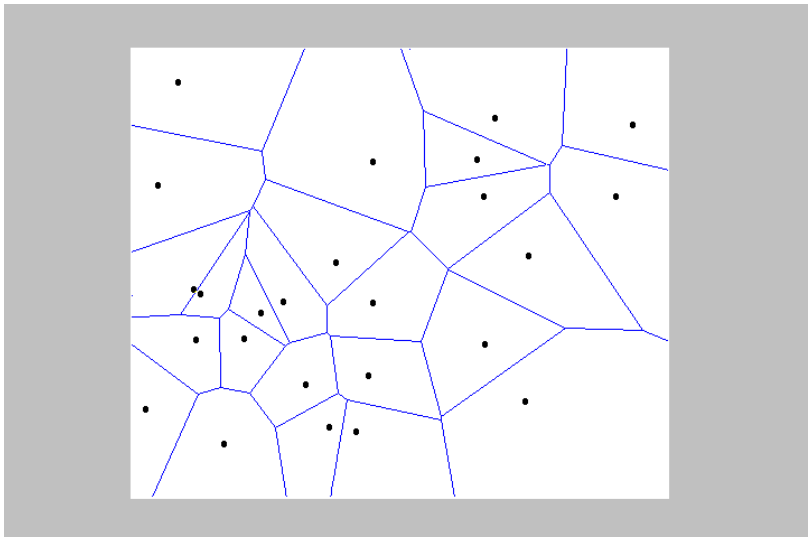

Figura 11: Diagrama de Voronoi del conjunto de localizaciones.

2.3. La búsqueda de los K caminos más cortos.

2.3.1. La complejidad computacional de la búsqueda.

Uno de los mayores problemas del diseño del Planificador de Trayectorias corresponde a la determinación de los caminos más cortos entre los diferentes vehículos y objetivos si se pretende obtener más de una ruta entre ellos. En ese caso, el aplicar un algoritmo como el de Dijkstra entre cada origen y destino introduce un alto coste computacional en el sistema que relentiza la ejecución del módulo. Se recurre al algoritmo de búsqueda de los k caminos desarrollado por Eppstein, muy conveniente para reducir la carga computacional necesaria para determinar las *K* trayectorias.

La búsqueda de las K trayectorias más cortas entre un origen y un destino en un grafo no dirigido se refiere a la determinación de los caminos (conjunto de aristas o arcos del grafo) de longitud mínima. La longitud de un camino es la suma de los costes individuales de los arcos que lo constituyen, referidos a la longitud del arco y al grado de exposición al peligro al que se encuentra sometido. En el grafo se dan bucles. Los arcos del grafo pueden almacenarse en un tiempo proporcional a su número (~*m*).

El uso de métodos diferentes al algoritmo de Dijkstra para la búsqueda de las *K* trayectorias más cortas entre un origen y destino determinados, permite efectuarla en un tiempo ~ (*m+nlogn+k*). Además es posible encontrar las *K* rutas más cortas entre un origen y cualquiera de los destinos en un tiempo ~ (*m+nlogn+kn*). Para la búsqueda en un grafo no dirigido la mejor solución conocida basada en el algoritmo de Dijkstra consume un tiempo ~(*k(m+nlogn)*), manteniéndose constante el tiempo que lleva encontrar cada uno de los caminos. Como puede comprobarse, esta diferencia es apreciable y aconseja la adopción de los métodos de búsqueda rápida.

El algoritmo de Eppstein se basa en la búsqueda de los árboles de peso mínimos incluidos en un grafo, o determinar los *K* nodos de mínimo peso en un árbol elaborado para resolver la cuestión en el que se buscan los mejores árboles dentro de una secuencia de grafos. Esta técnica se ha desarrollado para resolver otros problemas, pero puede adoptarse para la Búsqueda de las *K* trayectorias más cortas.

2.3.2. La Búsqueda Rápida. Método de Eppstein.

Dada la dificultad de la búsqueda entre un origen y destino prefijados desde el punto de vista de la carga computacional, la búsqueda se establecerá entre un origen y cualquiera de los nodos del grafo como destino. Esta búsqueda constituye el algoritmo básico del método. Se puede emplear el Algoritmo de Dijkstra.

PASO 1	➢	Crear lista de caminos de salida (inicialmente sólo el camino de coste cero hacia el origen).
	➢	Crear lista de arcos candidatos a formar parte de la lista de salida (sucesores del origen).
PASO 2	➢	Iterativamente se selecciona el segmento de menor coste de la lista de arcos candidatos.
PASO 3	➢	Se añaden a la lista de arcos candidatos a formar parte de los caminos de salida los sucesores del arco seleccionado en cada iteración y VOLVER AL PASO 2.

Tabla 3: Algoritmo básico de Búsqueda de las K trayectorias.

Una vez efectuada la búsqueda entre el nodo origen y cualquier nodo del grafo, hay que trasladar esos resultados a la búsqueda de las rutas entre un origen y destino determinados. Este aspecto constituye por si mismo un problema por estas causas:

- ✓ El camino más corto entre dos nodos origen y destino *s* y *t*, no tienen por qué estar formado por la unión del camino más cortos entre *s* y un nodo intermedio, y entre ese nodo intermedio y *t*. Considerarlo puede ser fuente de error.

- ✓ Un camino entre un nodo origen s y un nodo destino t se puede representar de diferentes formas a partir de un camino desde el origen a un nodo intermedio seguido del camino más corto desde ese nodo intermedio hasta el destino.

- ✓ El grafo puede variar, por lo que el grado del grafo no está limitado.

- ✓ La búsqueda de caminos ya registrados en el árbol de rutas más cortas como paso intermedio, tiene que efectuarse en el menor tiempo posible.

Para solucionar los dos primeros problemas se establece una función potencial capaz de modificar las longitudes de los lados del grafo de manera que la longitud total de cualquier camino más corto que termine en *t* sea cero. Se pretende que al concatenar caminos hacia cualquier nodo intermedio, y de este hacia el destino *t* no sufra alteración el orden de los caminos ya registrados.

En el segundo problema, al considerar los nuevos caminos el último lado de estos no debe estar incluido en el árbol de los caminos más cortos. Este aspecto es el que genera el último problema enumerado, ya que obliga a tener en cuenta en todo momento los caminos ya registrados a la hora de seleccionar cualquier otro camino.

Se propone una técnica que minimiza los problemas enumerados basada en un tratamiento de la información registrada que minimiza el tiempo que lleva la selección de los *K* caminos más cortos entre un origen y destino prefijados.

Más tarde habrá que tratar la información proporcionada por este algoritmo básico para seleccionar las *K* trayectorias más cortas entre el origen y destino determinados. Se emplearán las técnicas de Frederickson que permiten seleccionar los árboles de menor peso en un grafo.

Se asocia a cada vértice del grafo un conjunto en el que se detallen los lados que no formen parte del árbol de los caminos más cortos y que puedan ser alcanzados desde dicho vértice a través de segmentos incluidos en el árbol de caminos más cortos. Para ahorrar tiempo de computación estos conjuntos compartirán con los destinados a otros nodos los elementos comunes a ambos conjuntos.

La estructura utilizada para llevar a cabo la búsqueda de los caminos más cortos se detalla más adelante y la forma de ser gestionada se basa en las técnicas desarrolladas por Frederickson, que permiten recorrer los conjuntos binarios usando estructuras comunes que disminuyen el tiempo requerido para resolver la cuestión.

A continuación se describen las estructuras de datos que soportan el problema expuesto en este apartado, comenzando por la manera de evaluar los caminos, siguiendo por la estructuración que se haga con ellos, y por último el tratamiento que reciben con el fin de seleccionar las *K* trayectorias más cortas entre origen y destino.

2.3.2.1. El peso de los arcos del grafo.

Se considera un grafo con *N* vértices y *M* arcos en el que $M>C_{N,2.}$ Si la longitud de cada arco es *l(a)*, la longitud del camino *l(c)* será la suma de las longitudes de los arcos que lo constituyen. Se define la distancia entre dos nodos cualquiera *s* y *t* asignándole la longitud del camino más corto entre dichos nodos.

$$d(s \to t) = mn_c \{l(c_{s \to t})\}$$

Los conjuntos asignados a cada vértice son árboles binarios cuyos nodos están tasados por un peso sujeto a la restricción de que los pesos que describen los nodos hijos son mayores o iguales al peso del nodo padre. No preocupa el tamaño de estos árboles ya que se permite el crecimiento del grafo original, por tanto estos árboles pueden crecer lo necesario.

Para optimizar la búsqueda en el caso de tener árboles de grado indeterminado es conveniente encontrar el correspondiente árbol de grado D que tenga las mismas propiedades y pesos. Esta transformación se lleva a cabo en un tiempo de orden lineal. La inserción de cualquier elemento en esta nueva estructura se lleva a cabo en un tiempo de orden logarítmico, como la búsqueda del elemento requerido en esta estructura que es lo que lleva listar los elementos según el peso.

No se utilizan los lados del grafo de forma explícita, sino que previamente son integrados en la estructura de búsqueda. Se registran del orden de ($\sim K.N$) arcos en la búsqueda de *K* rutas, por lo que se requiere un tiempo del orden de ($\sim k^{2.}N$) a no ser que se use las técnica de Fredericksson ideada para dar con los K árboles de mínimo peso en un estructura en la que se forman sucesivos árboles que se diferencian de los antecesores por la eliminación de uno de sus lados y la inclusión de uno nuevo.

Esta estructura se forma con punteros hacia los árboles predecesores y una descripción de cada árbol integrado. Para usar la técnica de búsqueda rápida, cada camino diferirá de su camino previo por la inclusión de un nuevo segmento que no forme parte de la estructura establecida de caminos más cortos y de los ajustes necesarios para poder introducir el nuevo camino en la estructura. Por tanto habrán de considerarse el puntero del nuevo lado al nodo del árbol y la descripción de dicho arco.

Dados el grafo *G*, y los nodos origen *s* y destino *t*, se forma el árbol *T* formado por los caminos más cortos con destino *t* desde cualquier nodo origen. Los demás nodos son etiquetados por la longitud del camino más corto que les lleve a *t*. Se calcula el árbol de caminos más cortos que parta de *s* y llegue a cualquier nodo (la misma operación que la determinación de *T* a partir de *G*, actuando en este caso sobre el grafo revertido). La computación de *T* lleva un tiempo del orden ~(*m+nlogn*).

Cada arco del grafo G se etiqueta no con su longitud, sino que se le aplica una función a ésta. Para describir un arco *a* del camino se tiene en cuenta el sentido en el que se recorre. El extremo de partida se nombra *prt(a)* mientras que el extremo destino se denota *dst(a)*. Cualquier arco a queda caracterizado por la función $\delta(a)$.

$$\delta(a) = l(a) + d(dst(a) \to t) + d(prt(a) \to t)$$

Es una medida del coste que supone incluir al arco *a* en el camino de *s* a *t* con respecto al camino más corto descrito por *T* para esos dos nodos. En el apartado anterior se mencionaban los problemas que suscita la búsqueda entre un origen y cualquier destino como paso intermedio para determinar la solución entre otro par de nodos. $\delta(a)$ trata de solucionar el primero de ellos y es la función potencial propuesta. Por las características de δ(a) resulta una forma eficaz de etiquetar los arcos de G:

- ✓ $\delta(a) \geq 0$. Si el arco *a* forma parte del camino registrado en el árbol *T*, $\delta(a)=0$.

- ✓ Facilita la representación del camino la enumeración de los segmentos que pertenecen a *G* y que difieren de los ya registrados por *T*, conjunto que se conoce por arcos no registrados *anr(c)*. El camino descrito por *T* no necesita volver a ser definido, mientras que los arcos que no se puedan alcanzar desde un vértice de *T* no corresponderán a caminos de *s* a *t*:

$$l(c) = d(s \to t) + \sum_{a \in anr(c)} \delta(a) = d(s \to t) + \sum_{a \in c} \delta(a)$$

- ✓ Un camino *c* se describe a través de su posición en la lista de caminos previos y el lado que lo diferencia del camino previo. El camino padre a partir del cual se origina *c* queda descrito por el conjunto llamado previo: *prv(c)*.

$$l(c) = l(prv(c))$$

2.3.2.2. Las estructuras de datos empleadas en el método.

La estructura que ordena a los caminos más cortos es un árbol en el que cada nodo corresponde a un camino cuyo nodo padre es el camino previo que lo define. A medida que pasamos de padres a hijos se incrementa la longitud del camino.

Cada padre puede tener hasta un máximo de *M* hijos, al menos uno por cada uno de los segmentos que difieren del camino previo, conjunto de segmentos que se conoce por *dif(c)*, siendo c el camino padre. Por lo tanto los segmentos contenidos en *dif(c)* de un camino *c* hijo de otro camino *c'* son los arcos no incluidos en *T* cuyo origen se sitúa dentro de un camino incluido en el árbol de caminos más cortos *T* que una el destino del arco diferencial del camino padre *dst(dif(c'))* con el nodo destino *t*.

Si el grafo *G* contiene bucles, el árbol que registra los caminos crece indefinidamente, y como se ha apuntado no se puede desarrollar la búsqueda que propone Frederickson. En este caso habría que sustituir cada nodo del árbol *c* por el equivalente de grado *D* compuesto por los lados con origen en el camino que parte del destino del lado *dif(c)* hacia *t*. Esos arcos se describen gracias al valor $\delta(a)$.

Para cada vértice *v* del grafo se quiere formar un conjunto $H_G(v)$ con todos los lados en los que figura *v* como origen de un segmento (*prt(a)*) incluido en el camino de *s* a *t*. En $H_G(v)$ se registrarán los valores $\delta(a)$ correspondientes a esos segmentos. Cuando se modifique el árbol de contiene los caminos para poder desarrollar la búsqueda, se reemplazará cada nodo *c* por la copia del conjunto $H_G(dst(dif(c))$.

El conjunto *sal(v)* contiene todos los lados del grafo *G* no contenidos en el árbol *T* que tienen origen en *v* de manera que se puede formar el conjunto $H_{sal}(v)$ en el que se indiquen los pesos $\delta(a)$ de dichos arcos. A diferencia de H_G, el árbol que describe H_{sal} tiene grado dos, y por tanto fuerza que cada hijo sea alcanzado a partir de un sólo padre. Se calcula este conjunto para todos los nodos, operación que lleva aparejada un tiempo del orden del número de elementos del conjunto *sal(v)* para cada vértice. La ruta de salida de cada vértice lo forma el lado que minimiza el valor de las funciones $\delta(a)$ de los arcos que forman parte de *sal(v*):

$$rts(v) = mn_{sal(v)}\{H_{sal}(v)\}$$

La determinación de estos conjuntos se realiza para todos los nodos en un tiempo proporcional a la suma del número de elementos de salida de todos los vértices. Se elabora para cada vértice *v* el conjunto $H_G(v)$. Se incorporan todos los conjuntos $H_{sal(v')}$ correspondientes a vértices *v'* incluidos en el árbol *T* en el camino que une el vértice *v* con *t*. Es equivalente a decir que se incluye $H_{sal(v)}$ dentro del conjunto $H_G(sig_T(v))$ para formar $H_G(v)$.

El grado indeterminado de estos árboles dificulta la búsqueda. Se consigue una estructura balanceada sustituyendo los conjuntos $H_G(v)$ por conjuntos $H_T(v)$ en los que sólo se consideren los vértices incluidos en la ruta de salida de cada vértice *v'* dentro del camino de *v* a *t*. Equivale a insertar el conjunto ruta de salida *rts(v)* dentro del conjunto $H_T(sig_T(v))$. Se tiene que indicar claramente los punteros a los conjuntos $H_T(sig_T(v))$, de esta manera se almacenan los conjuntos $H_T(v)$ sin modificar $H_T(sig_T(v))$ usando tan solo del orden de *log(n)* palabras adicionales en memoria (punteros indicados por *), y permitiendo almacenar el número de nodo del camino.

El conjunto $H_G(v)$ se forma conectando cada vértice incluido en la ruta de salida de los diferente vértices que se encuentran en el camino más corto de *v* hacia *t* con los árboles que incluyan punteros hacia elementos incluidos en el conjunto $H_T(v)$. Es posible construir el conjunto $H_G(v)$ en el mismo tiempo que se tarda en elaborar el conjunto $H_T(v)$. $H_G(v)$ es una estructura de grado *3*, ya que a cada padre se le permite como mucho tres hijos: dos hijos pueden provenir de $H_T(v)$ y uno más de $H_{sal}(v)$ o por el contrario ninguno del primer conjunto y dos del segundo.

Para hacer más intuitivo este procedimiento y comprobar como las estructuras descritas comparten las partes comunes, se puede elaborar un grafo dirigido sin bucles a partir de *G*, que se denota por *D(G)* en el que aparecen los vértices v de la estructura en árbol de los caminos más cortos.

- ✓ El grafo *D(G)* tiene del orden de ~(*m*+*nlogn*) vértices.

- ✓ Cada vértice de *D(G)* corresponde a un lado del grafo *G* no contenido en *T.* El grado de salida de cada vértice de *D(G)* es como mucho *3*.

- ✓ Cada vértice de *D(G)* posee un conjunto de grado *3*, $H_G(v)$, compuesto por vértices que corresponden a lados de *G* etiquetados por $\delta(a)$ no incluidos en *T* cuyo origen forma parte del camino más corto registrado en *T* desde *v* hacia *t*.

Los vértices del grafo *D(G)* proceden bien de $H_T(v)$ o del conjunto $H_{sal}(v)$. Cada nodo de éste último conjunto corresponde a un único lado de *G* no incluido en *T* de manera que los conjuntos $H_{sal}(v)$ proporcionan a *D(G)* como mucho *N-M+1* nodos.

Cada vértice de *G* ofrece un número de nodos de los conjuntos $H_T(v)$ igual a $ceil(log_2(\lambda))$ donde λ es la longitud del camino del vértice a *t*. El número de árboles que han necesitado ser balanceados insertando la ruta de salida del vértice *rts(v)* dentro de $H_T(sig_T(v))$ es $1+log_2\lambda$, del que hay que extraer uno debido a que *rts(v)* está implícito en $H_{sal}(v)$. El número total de nodos es al menos *M+N.log(N)-(c+1).N*. La construcción de *D(G)* ocupa para los diferentes nodos el mismo tiempo y en total para el peor de los casos, si *T* registra un solo camino, computar *D(G)* lleva:

$$\sum_{\lambda} ceil\left(\log_2 \lambda\right) \le N \cdot \log_2\left(N\right) - c \cdot N \;\; ; \quad c \le 2 \, .$$

Se forma el conjunto de vértices denominado *h(v)* a partir de la ruta descrita en $H_G(v)$. Los nodos alcanzados por *h(v)* son los que aparecen en $H_T(v)$ junto con aquellos que provienen de $H_{sal}(v)$. Se recuerda que $H_T(v)$ representa los vértices del camino de v a t mientras que $H_{sal}(v)$ registra los arcos cuyos orígenes se sitúan en los vértices de dicho camino. Se consideran los arcos con origen en el camino de *v* a *t*. Se forma el conjunto $H_G(s)$ evaluando los nodos alcanzados en *D(G)* desde el vértice *s*. Esta estructura registra los caminos que difieren del más corto sólo por la inserción de un lado del grafo *G* que no se encuentre en el árbol *T*. El grafo *D(G)* junto con otros arcos adicionales es capaz de generar cualquier camino que una *s* con *t*.

Se define el grafo de caminos *P(G)* tomando los vértices en *D(G)* que cumplan al incluir un nuevo vértice que la ruta sea la marcada por éste: *r=r(s)*. Los vértices de *P(G)* no tienen peso alguno, no así los arcos que son valorados teniendo en cuenta la diferencia entre los vértices del grafo *D(G)* que corresponden a los de *P(G)*. El arco *(u,v)* de *P(G)* posee una longitud $\delta(v) - \delta(u)$. Los lados son arcos de este conjunto.

A cada vértice *v* de *P(G)* correspondiente a un lado del grafo *G* no considerado en *T* se le asocian algunos pares de vértices *u* y *w* para generar un nuevo arco de unión que partiendo de *v* llegue al conjunto descrito por *h(w)* en el grafo *P(G)* con longitudes $\delta(h(w))$. También se crea un arco inicial entre *r* y *h(s)* de longitud $\delta(h(s))$. Existe una correspondencia unívoca que preserva las longitudes entre los caminos de *s* a *t* en el grafo *G* y los caminos que comienzan desde *r* en el grafo *P(G)*.

No resulta evidente la descripción de un camino entre *s* y *t* sólo por el conjunto de arcos no registrados del camino *c* (lados de *G* no incluidos en *T* cuyo origen se sitúa en el árbol *T*). Si a toda secuencia le corresponde un único camino con origen en el vértice *r* del grafo de caminos *P(G)* y con destino en el nodo del último lado no registrado del camino, *dif(c)*, de la misma manera que cualquier camino que comience en *r* representa a un camino que incluya alguno de los arcos no registrados *anr(c)*.

A partir de un camino *c* se puede construir el correspondiente *c'* en el grafo de caminos *P(G)*. Si el conjunto *anr(c)* está vacío por tratarse del camino más corto, *c'* contendrá el vértice *r* de partida. Además el camino previo *prv(c)* tiene su correspondiente camino *d* que concluirá en el nodo de *P(G)* equivalente al arco *dif(prv(c))*. Para formar *P(G)* a partir de *D(G)* se añade el lado entre el vértice y *h(v)*.

Como *dif(c)* tiene su origen en un vértice del camino de *v* a *t* en el árbol *T*, le corresponde un único nodo en $H_G(v)$, de manera que se puede formar el camino *c'* en *P(G)* concatenando el camino d con el que parte de *h(v)* hacia el nodo. Se demuestra la igualdad entre las longitudes del camino original *c* y c' en el grafo de caminos *P(G)*.

$$l(c) = l(prv(c)) + l(dif(c)) = l(d) + l(dif(c)) = l(c')$$

Para formar un camino *c* entre *s* y *t* en el grafo *G* a partir de un camino *c'* en el grafo *P(G)*. Se forma *sec(c')* tomando de la secuencia de lados de G los de origen, y añadiendo el arco de *G* responsable del vértice final del camino *c'*. Los nodos de *P(G)* alcanzados desde el origen de cada arco de unión *(u,v)* corresponden en $H_G(v)$ a dichos vértices, y cada arco añadido a la secuencia *sec(c')* se encuentra en el árbol *T* entre *v* y *t*. El tiempo para formar el grafo de caminos *P(G)* es del orden *~(M+N.logN)*. *P(G)* junto al correspondiente vértice añadido *r* tiene las siguientes características:

- ✓ Tiene del orden de *M+N.log(N)* vértices. El grado de salida de cada vértice de *P(G)* no excede de cuatro.

- ✓ El peso de los arcos de *P(G)* nunca es negativo.

- ✓ Existe una correspondencia unívoca entre los caminos en *G* y los de *P(G)* que comiencen en el vértice auxiliar *r*. Esta correspondencia preserva la longitud de los caminos. Si la longitud en *P(G)* es *l(c')*, en *G* el camino mide $l(c) + d(s \to t)$

Para finalizar, se construye la última estructura empleada en la selección de los *K* elementos, herramienta utilizada en el problema general de la búsqueda de las *K* trayectorias más cortas.

A partir del grafo de caminos *P(G)* se forma *H(G)* que contiene conjuntos de grado cuatro. Cada nodo de *H(G)* representa un camino en el grafo original *G*. Se construye asignando un nodo a cada camino del grafo *P(G)* con origen en el vértice *r*. En *H(G)* el padre de cada nodo corresponde a un camino que se diferencia del hijo por un arco. Como el grado de salida de los nodos de *P(G)* es cuatro, en *H(G)* un nodo puede tener hasta cuatro hijos. Los pesos son conjuntos ordenados y el peso de cada nodo es la longitud total del camino que representa.

De todo lo anterior se descubre en *H(G)* que esta estructura de grado cuarto tiene una correspondencia unívoca entre sus nodos y los caminos entre *s* y *t* en *G* y en virtud de la cual la longitud de un camino en *G* es *d(s,t)* más el peso del nodo equivalente a dicho camino en la estructura final *H(G)*.

El algoritmo de selección explorará en general una región en *H(G)* formada por un número de nodos de orden *~K*. Se pueden representar los nodos en un espacio constante asignándoles números e indicando para cada uno el padre del que proviene y el lado auxiliar del que parte el camino que lo representa en el grafo de caminos *P(G)*. Resulta sencillo encontrar el hijo de un nodo, basta con encontrar el que le corresponde entre los lados de salida de *P(G)*. Por último el peso de un nodo se calcula gracias al peso del padre.

Todos los aspectos positivos que reúnen las diferentes estructuras descritas a lo largo del apartado y que son de gran ayuda en la selección de los k elementos más pequeños propuesta por Frederickson, no implican en ningún momento la pérdida de equivalencia respecto al grafo original G. Toda la información de un camino entre s y t en G se encuentra en ellas. El siguiente punto aborda la selección rápida de las rutas más cortas.

2.3.2.3. La Selección Rápida de Elementos.

La Selección Rápida diseñada por Frederickson permite determinar las *K* trayectorias más cortas equivalente a seleccionar los *K* vértices con menor peso de cualquier conjunto en el grafo de caminos, en un tiempo del orden *(~K)*. Por lo tanto queda reducido el tiempo que había requerido la búsqueda simple, estimado en ~(*K.logK*).

El diseño del método lo hizo Frederickson para ser aplicado a conjuntos de grado *dos*. En la descripción de las estructuras ha quedado establecido que el tamaño de los conjuntos $H_V(G)$ puede llegar a *cuatro*. En general para permitir que en un conjunto de grado *g* pueda emprenderse la selección rápida habrá que modificar el método original. La forma más sencilla es convertir los conjuntos de grado *g* en binarios haciendo *g-1* copias de cada vértice conectándolas en un árbol binario con *g* hijos, asegurando que los *g.k* pesos más pequeños de los vértices del conjunto binario correspondan exactamente con los *k* pesos más pequeños en el conjunto de grado *g*.

La tarea más ardua es la inclusión del conjunto $H_T(v)$ en cada vértice *v*, cuyos elementos son los vértices que recorre el camino más corto (registrado en el árbol *T*) entre *v* y el destino *t*. La información que almacena cada vértice corresponde al peso de la ruta de salida. Dado el árbol de caminos más cortos *T* se introducen nodos ficticios con un peso excesivamente grande que permita la vigencia de la información contenida en *T*, pero que lo convierta en un árbol binario en el que cada ruta tenga un grado de entrada igual a uno. Para los vértices de *T* con pesos diferentes se usa una partición de orden *z* compuesta por vértices *V* del árbol de caminos más cortos *T* tiene las siguientes características:

- Cada conjunto de la partición contendrá como mucho *z* vértices.
- Cada parte de la partición constituye un subárbol de T.
- Cada parte S de la partición, si está constituida por más de un vértice habrá al menos dos lados de árbol.
- No se pueden combinar dos partes garantizando las propiedades anteriores.

Una partición puede ser detectada fácilmente en un tiempo lineal. Se puede encontrar una partición de orden *dos* a partir de un árbol binario en un tiempo lineal en la cual al menos aparecen *5n/6* partes.

La unión de las diferentes partes integrantes de una partición constituye por si misma un árbol binario. Se define una secuencia de árboles T_i : T_0=T, y para valores de i mayores que cero, T_i se forma a partir de T_{i-1} encontrando una partición válida y conectando las partes. El tamaño de cada árbol es del orden *~((5/6)^{i}n).*

Para cada conjunto *S* de vértices de T_{i-1} forzados a formar un vértice *v* en T_i se define la parte de la partición de T_i que contiene *S* a la que se llama siguiente nivel de *S, snv(S)*. Se dice que *S* es un conjunto interior si el número de vértices es *dos*. Si el nodo destino *t* tiene grado de entrada *uno*, lo mismo sucede para cualquier camino en T_i de manera que *t* no formaría parte de ningún conjunto interior, que tiene grado de entrada y de salida uno. Como T_i forma parte de *T*, cada eje de *Ti* corresponde a un eje de *T*.

Si *e* es el arco de salida del nodo *v* en el árbol T_i, se define la ruta del camino *rdc(S)* que describe el camino de *T* que parte del destino del lado *e*, *dst(e)* hacia el destino final *t*. Si *S* constituye un conjunto interior con un lado de entrada *e'*, se define en la ruta de entrada *rde(S)* el camino de *T* que parte del destino del arco *e' dst(e')* y termina en el origen del lado de salida *e prt(e)*.

Se define un conjunto parcial de grado *m* para formar el par *(M,H)* donde *H* es un conjunto y *M* es una colección de m elementos más pequeños que todos los nodos de *H.* Si *H* es el conjunto vacío, *M* puede tener al menos *m* elementos y se llamará *(M,H)* un conjunto parcial de grado *m*. Para empezar se forma un conjunto de partida $(M_1(S),H_1(S))$ para todos los vértices de *T* en cada camino descrito en la ruta de entrada de *S rde(S)*. La computación de *M1* correspondiente a todos los conjuntos *S* de la partición se puede efectuar en un tiempo ~(N). Estos conjuntos *(Mi,Hi(S))* se van formando de padres a hijos.

A partir de un árbol binario *T* con nodos tasados se puede construir para cada vértice un conjunto binario $H_T(v)$ para todos los nodos v del grafo G en un tiempo *~(m+n+k)*. Con la técnica de Frederickson se ha eliminado por completo la dependencia logarítmica.

2.4. El Planificador de Trayectorias.

El Planificador de Trayectorias es el módulo del sistema encargado de resolver la tarea de la Planificación de Trayectorias. El módulo entabla un diálogo muy intenso con otros, que en el caso de situarse fuera del vehículo aéreo no tripulado, requiere de la actuación del Director de Comunicaciones. Esos son los cometidos que el Planificador de Trayectorias desarrolla en el ámbito de cada UAV:

- ✓ Elaboración del diagrama de Voronoi. A partir de los datos entregados por el Director de Comunicaciones sobre la posición real en ese instante de tiempo de los diferentes elementos que forman parte del escenario de trabajo, el Planificador traza el diagrama de Voronoi, y determina los costes de los segmentos que lo forman. Además añade los segmentos desde la posición del UAV a los vértices de la región en la que se encuentra, calculando el coste de los nuevos segmentos. Las coordenadas de los arcos que forman el diagrama y los costes se almacenan en memoria para ser utilizados por otros módulos.

- ✓ Modificación del diagrama de Voronoi, añadiendo a los segmentos y rayos del diagrama de Voronoi la colección de segmentos que unen la localización del objetivo asignado al UAV con los vértices de la correspondiente región de Voronoi. Se determina el coste de los nuevos segmentos añadidos. Esta operación se realiza sobre el diagrama de Voronoi con los objetivos que correspondan en cada caso, según la configuración del sistema.

- ✓ Selección de las *K* rutas más cortas: Dependiendo de la configuración del Director de Asignación(próximo capítulo), se realiza la búsqueda de las *K* trayectorias menos costosas (método de Eppstein) de una de estas formas:

  - ➢ Entre el UAV y todos los objetivos si se determina la decisión individual para la Asignación de objetivos haciendo uso de los lados de Voronoi. En este caso se guarda toda la información y sólo habrá que rescatar la ruta.

  - ➢ Entre el UAV y un solo objetivo, si el Director de Asignación hace uso del método de distancias. El objetivo será el asignado finalmente al UAV (tantas veces como sea requerido desde el sistema central).

El resultado del Planificador de Trayectorias es utilizado por el Interceptor Simultáneo. Las K trayectorias seleccionadas forma parte de la coordinación temporal del conjunto de múltiples UAVs.

$$Coordinación_i \leftarrow [V_{mn} \quad V_{mx}] \times \{T_{i1},...,T_{ik}\}$$

No es necesaria la comunicación de los puntos de las trayectorias entre el UAV y el objetivo para calcular la coordinación temporal, basta con transmitir las longitudes y los costes de las K trayectorias seleccionadas por el Planificador de Trayectorias de un UAV (ambas variables de coordinación). En la memoria interna de cada UAV quedan registradas a la espera de que el Director de Comunicaciones de el resultado de la Interceptación Simultánea, momento en que se rescata la trayectoria final.

PASO 1	➢	Elaboración del Diagrama de Voronoi básico y uniones del UAV.
	➢	Determinación de los Costes de cada uno de los segmentos.
	➢	Registro en memoria del diagrama básico de Voronoi.
PASO 2	➢	Uniones entre un objetivo y el diagrama. Determinación del coste.
iterativo	➢	Búsqueda de las K rutas más cortas entre UAV y objetivo.
	➢	Registro en memoria de los resultados obtenidos.
	➢	Descartar las uniones para realizar la Búsqueda con otro objetivo.
PASO 3	➢	Selección de resultados del Planificador (archivos de datos).

Tabla 4: Operación de Planificador de Trayectorias.

Dependiendo de la configuración que se haga del planificador y otros módulos la secuencialidad de los pasos puede sufrir alteraciones. Lo habitual es utilizar los costes (de longitud y de exposición al peligro) de los segmentos de las rutas entre los UAVs y todos los objetivos para llevar a cabo la decisión individual en la asignación de objetivos. En este caso, de manera iterativa se determinan las trayectorias con todos y cada uno de los objetivos. La ventaja de esta forma de proceder es que si se almacena en un fichero los resultados de la búsqueda lo único que habrá que hacer más tarde es seleccionar la información que proceda en cada caso.

Si no se dispone de memoria suficiente se realiza la búsqueda con el objetivo una vez asignado. Si el Interceptor lo modifica, se vuelve a realizar la búsqueda.

2.4.1. Codificación del Planificador de Trayectorias.

El Planificador de Trayectorias en un módulo situado en cada uno de los subsistemas UAV. Realiza su función atendiendo a la realidad del UAV que lo controla. No existe interacción con otros planificadores y sólo en el caso de no calcular a la vez todas las rutas (por memoria insuficiente), se espera la ejecución de otros módulos. Es el caso del Director de Asignación que comunica el objetivo del UAV y del Interceptor Simultáneo que puede hacer variar ese objetivo y que en función de las variables de coordinación entregadas por el Planificador determina la velocidad y la ruta final.

Sin pérdida de generalidad se va a describir el procedimiento descrito en el apartado anterior en el que se determinan las K rutas más cortas entre el UAV y todos los objetivos registrados. En el UAV se sitúan las siguientes funciones:

ptuav.m	Control y Configuración del Planificador de Trayectorias.
• ptvuav.m	'DIAGRAMA DE VORONOI'
○ voronoi.m	Diagrama de Voronoi (comando).
○ delaunay.m	Triangulación de Delaunay (comando).
○ ptvu.m	Uniones entre un punto y el Diagrama.
○ ptvc.m	Costes de los segmentos.
○ TESS	Programa C.
• ptkuav.m	'BÚSQUEDA DE LAS K TRAYECTORIAS'
○ ptkf.m	Grabación de los datos en el fichero grafo.gr.
○ ptkg.m	Preparación del grafo para el método de Eppstein.
○ ptkd.m	Algoritmo de búsqueda (computación costosa).
○ EPPSTEIN:	eppstein.c, loadgraph.c, dijkstraReversed.c.

Tabla 5: Software del Planificador de Trayectorias.

Se han agrupado las funciones según el cometido a que están destinadas en dos grandes grupos. Por un lado las funciones encargadas de elaborar el diagrama de Voronoi, por el otro las destinadas a desarrollar la búsqueda de las K trayectorias entre el UAV y el objetivo. Por encima de estos grupos se sitúa la función de control y configuración del módulo. Otro aspecto importante es la descripción de los ficheros utilizados para almacenar la información porque se prima el carácter abierto.

2.4.2. El control y la configuración: la función ptuav.m.

Es el fichero al que se accede para configurar el funcionamiento del Planificador de Trayectorias dentro de las opciones previstas. Cuando sean incorporadas nuevas funciones, se debe actualizar el código para aceptar esas aportaciones (tareas de mantenimiento del sistema). Las opciones de funcionamiento afectan por una parte a la forma de elaborar el diagrama de Voronoi y por otra a la búsqueda de las K trayectorias más cortas.

- ✓ Diagrama de Voronoi. Se puede utilizar los comandos que aporta MATLAB (*voronoi.m* y *delaunay.m*) o emplear otro código como los algoritmos FORTUNE o TESS. La variable configurable es *dv*, una cadena de dos caracteres que toma los valores *'mb'*, *'fo'* o *'ts'*. El comando de MATLAB no limita el espacio (no construye rayos), por lo que habrá que limitar aquellos lados que escapen del escenario de trabajo.

- ✓ Búsqueda de las *K* trayectorias más cortas. Se escoge entre el algoritmo de Dijkstra o el recomendado Método de Eppstein. Según la opción seleccionada la variable *bk* puede tomar los valores *'dj'* o *'ep'*.

El otro aspecto de la función es el control del funcionamiento del planificador de trayectorias. Desde este fichero se preparan los datos y se llama a otras funciones del módulo, en este caso a *ptvuav.m* y ptkuav.m. La secuencia de las llamadas es la descrita en el procedimiento del *apartado 2.4*.

1. PRIMER PASO: la función *ptvuav.m* genera el diagrama básico de Voronoi con las uniones del UAV al diagrama. Se determinan los costes de los segmentos. La función *ptkg.m* dirige los arcos del grafo para el método de Eppstein.

2. SEGUNDO PASO: de forma iterativa se selecciona un objetivo diferente, se llama a la función *ptvuav.m* con la opción *1* para encontrar los arcos entre el objetivo y el diagrama y determinar los costes. Se toman los resultados del paso anterior y se registra el grafo en el fichero *grafo.gr*. Se realiza la búsqueda con la función *ptkuav.m* que devuelve las K rutas más cortas entre el UAV y el objetivo en curso.

3. TERCER PASO: los resultados se guardan en el fichero *rpk.dat*. Por orden, el número de objetivos contabilizados, las trayectorias calculadas para cada uno, las longitudes de las rutas, los costes totales de las mismas, los segmentos que forman las trayectorias, las longitudes de los segmentos de todas las rutas y los costes de exposición al peligro de los mismos. El diagrama de Voronoi básico con los costes más las uniones hasta cada objetivo con los costes añadidos se registra en el fichero *rdv.dat*.

La siguiente tabla resumen contiene las entradas, salidas y principio de funcionamiento de la función *ptuav.m*:

OBJETIVOS	✓ Configuración del módulo. ➢ Diagrama de Voronoi: según dv, 'mb'(Matlab),'fr'(Fortune). ➢ Búsqueda de K rutas: según bk 'dj' (Dijkstra), 'ep'(Eppstein). ✓ Procedimiento de control de la Planificación de Trayectorias: ➢ PASO 1: 'Diagrama de de Voronoi' ➢ PASO 2: 'Búsqueda de las K rutas a cada objetivo'. ➢ PASO 3: 'Registro en los ficheros rdv.dat y rpt.dat.
ENTRADAS	✓ v: coordenadas x e y del UAV. ✓ O: coordenadas x e y de los N_o objetivos. [N_o*2]. ✓ P: coordenadas x e y de las N_p amenazas. [N_p*2].
SALIDAS	✓ err: 0 (ejecución correcta), 1 (ejecución no válida). ✓ FICHERO rdv.dat. ➢ Número de objetivos. ➢ Coordenadas x e y de los extremos de los lados del diagrama. ➢ Costes totales, de peligro y de longitud del diagrama. ➢ Número de uniones totales y por cada objetivo. ➢ Coordenadas x e y de las uniones y costes de las mismas. ✓ FICHERO rpk.dat. ➢ Número de objetivos. ➢ Número de trayectorias por objetivo. ➢ Longitudes y Costes totales de todas las rutas. ➢ Índice con el máximo numero de lados para cada objetivo. ➢ Número de los segmentos que componen cada ruta. ➢ Costes de longitud y de peligro de todos los segmentos.

Tabla 6: Resumen de la función *ptuav.m*.

2.4.3. El diagrama de Voronoi: la función ptvuav.m.

La función ptvuav.m se encarga de la elaboración del diagrama de Voronoi, partida para la planificación de las rutas. Existen dos momentos diferenciados en los que se realizan acciones destinadas a construir el diagrama, por tanto desde la función *ptuav.m* se llama a *ptvuav*.m definiendo el modo de operación y entregando las entradas correspondientes a uno de los dos modos. Dos tareas se diferencian:

- ✓ Elaboración del diagrama básico de Voronoi (*op=0*). Se comunica a la función las coordenadas *x* e *y* del UAV, la de los objetivos y amenazas (entradas *e1, e2* y *e3* respectivamente). Se traza el diagrama básico según la opción *dv* ('*mb*' Matlab o '*fr*' Fortune) entregada. Se calculan las uniones del UAV (función *ptvu.m*) y el coste (función *ptvc.m*) total en función del parámetro *ci*.

- ✓ Uniones hacia el objetivo (*op=1*). Se realiza tantas veces como objetivos se estén computando. Se prepara el diagrama para poder desarrollar la búsqueda entre el UAV y un objetivo. Las entradas contienen las coordenadas x e y del objetivo, las coordenadas *x* e *y* de los vértices de los distintos segmentos y las coordenadas de las amenazas (entradas *e1*, *e2* y *e3* respectivamente). La función *ptvu.m* determina los accesos al objetivo y la función *ptvc.m* los costes.

En ambos casos se devuelven las coordenadas *x* e *y* de los vértices de los segmentos de Voronoi determinados y los costes (salidas *s1*,*s2* y *s3* respectivamente).

OBJETIVOS	✓	Diagrama básico y uniones con el UAV. Costes. (op=0).
	✓	Uniones al objetivo y costes de las uniones (op=1).
ENTRADAS	✓	dv y ci: modo de elaboración del diagrama y parámetro.
	✓	v: coordenada x e y del UAV o del objetivo, según modo op.
	✓	O: coordenada x e y de los N_o objetivos o de los segmento Ns
	✓	P: coordenadas x e y de las N_p amenazas. [N_p*2].
	✓	op: modo de operación de la función.
SALIDAS	✓	sgx, sgy: coordenada x e y de los vértices de los segmentos.
	✓	C: costes totales, de longitud y de exposición al peligro.

Tabla 7: Resumen de *ptuav.m*.

2.4.3.1. Los costes de los segmentos de Voronoi: la función ptvc.m.

Recibe las coordenadas *x* e *y* de las amenazas (*P*), las coordenadas *x* e *y* de los vértices extremos de cada segmento (*sgx* y *sgy*) y el parámetro de ponderación *ci*. A la salida entrega la matriz C de tres columnas correspondientes a los costes totales, de longitud y de exposición al peligro de cada uno de los segmentos.

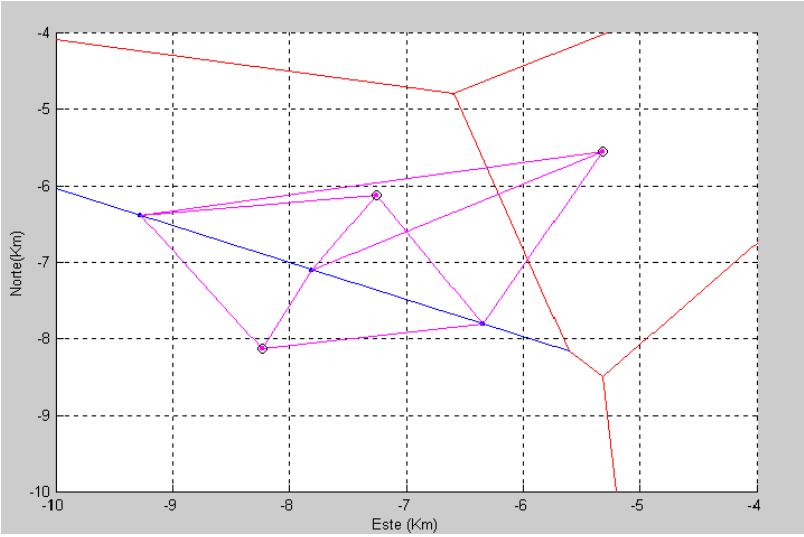

Figura 12: Determinación de los costes de un segmento.

Para un determinado segmento, el módulo determina el coste de longitud (azul). El coste de exposición al peligro se calcula a través de las distancias de todas las amenazas a los puntos *1/2*, *1/6* y *5/6* del segmento (fucsia). La suma de estas distancias multiplicada por el módulo del segmento en cuestión determina el coste de exposición al peligro. El coste total depende de la ponderación ci. En la función estas operaciones se realizan de forma conjunta para todos los segmentos del diagrama.

OBJETIVOS	✓	Costes totales, de exposición al peligro y de longitud.
ENTRADAS	✓	P: coordenadas x e y de las N_p amenazas. [N_p*2].
	✓	sgx: componente x de los extremos de los segmentos. [N_s*2]
	✓	sgy: componente y de los extremos de los segmentos. [N_s*2].
	✓	ci: constante de ponderación de los costes individuales.
SALIDAS	✓	C: costes totales, de longitud y de exposición al peligro.

Tabla 8: Resumen de *ptvc.m*.

2.4.3.2. Las uniones entre un punto y los vértices: la función ptvu.m.

Se recibe la posición del punto que se quiere unir al diagrama básico y las coordenadas x e y de los extremos del segmento de Voronoi. A la salida se proporcionan las coordenadas x e y de las uniones desde el punto al diagrama.

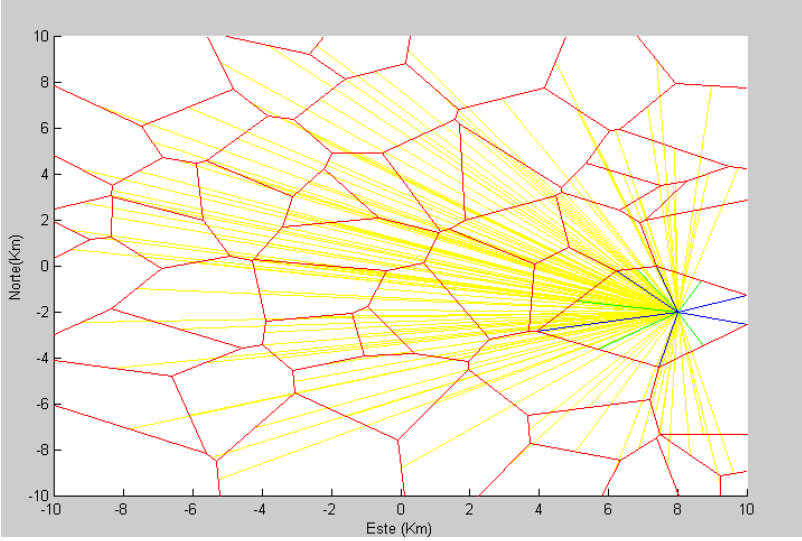

Figura 13: Uniones desde un punto a los vértices del polígono de Voronoi.

En amarillo aparecen las líneas entre el punto y los puntos intermedios de los segmentos. Se evalúan las intersecciones entre líneas y segmentos y se detectan aquellas rectas que no contienen intersecciones entre el punto a unir y el punto intermedio del segmento que lo describe. Esas líneas (verde) determina el polígono en el que se encuentra el punto. Queda seleccionar los vértices del polígono seleccionado y unirlos con el punto (líneas en azul). Las componentes x e y de estas uniones son las salida de la función.

OBJETIVOS	✓	Uniones entre el punto y los vértices de la región de Voronoi.
ENTRADAS	✓	v: coordenadas x e y del punto que se quiere unir.
	✓	sx: componente x de los extemos de los segmentos básicos [N_s*2]
	✓	sy: componente y de los extemos de los segmentos básicos [N_s*2]
SALIDAS	✓	ux: componente x de los extremos de las uniones [N_s*2]
	✓	uy: componente y de los extremos de las uniones [N_s*2].

Tabla 9: Resumen de *ptvu.m*.

2.4.4. La búsqueda de las K trayectorias más cortas: ptkuav.m.

La función *ptkuav.m* coordina la búsqueda de las K trayectorias entre el origen (UAV) y destino (objetivo en curso). Desde este fichero se realizan las llamadas a las funciones que intervienen en el proceso de búsqueda de las K trayectorias entre el origen y el destino de la búsqueda. Existen dos posibilidades.

- ✓ Algoritmo Básico: se aplica un algoritmo de búsqueda codificado en MATLAB muy ineficiente desde el punto de vista temporal cuando se determinan las K rutas más cortas entre el origen y el destino. La codificación en lenguaje MATLAB facilita el intercambio de información sin ficheros de por medio.

- ✓ Método de Eppstein: en este caso el programa que desarrolla la búsqueda está escrito en lenguaje C y tiene que ejecutarse en un entorno que permita su ejecución (sistema operativo QNX, etc...). La entrada de este programa es un fichero que describe el grafo dirigido que se ha elaborado en la función *ptkg.m* y que ha sido grabado en la función *ptkf.m.* La salida es también un fichero de datos que contiene la sucesión de nodos del grafo para cada una de las K rutas. No se ha definido la interfaz de funcionamiento entre los dos programas.

La función ptkuav.m es la encargada del control de las acciones que se efectúan durante la búsqueda, encargándose de llamar a las funciones e interpretar los resultados obtenidos. Las entradas difieren según la configuración del módulo.

- ➢ *bk*: método de Búsqueda seleccionado ‘dj’ (Dijsktra) o ‘ep’ (Eppstein).

- ➢ *k*: número de trayectorias que se van a buscar en el diagrama.

- ➢ *arcs*: según bk, si ‘dj’ se trata de una matriz de cuatro columnas con las componentes x (dos primeras) e y (resto) de los extremos de los arcos. Si se usa Eppstein es una matriz de dos columnas (números de nodos extremos).

- ➢ *orig*: según bk, si se selecciona ‘dj’ se pasan las coordenadas x e y del UAV, en caso contrario el número de nodo correspondiente al UAV en el grafo.

- ➢ *dest*: es análogo al anterior, pero correspondiendo al objetivo destino.

Una vez interpretados los datos se llama a la función que corresponda a cada caso con los parámetros que le han sido entregados por *ptuav.m*. En el caso del programa *Eppstein* se llama a la función *ptkf.m* encargada de la formación del fichero *grafo.gr*. No se ha integrado el código en el sistema de ficheros ya que es un procedimiento interno del módulo y no de interacción con otros.

Se realiza la búsqueda de las K trayectorias más cortas, bien en el fichero *ptkd.m* o por el programa *EPPSTEIN*. El resultado de éste último se lee desde la función ptkf.m, introduciendo el modo *uno*. En todo caso se tiene una matriz *Tv* de *K* filas y número de columnas variable en la que cada fila muestra el número de segmento de Voronoi que integra esa ruta. Cada fila es una trayectoria.

Tras esta operación se pueden seleccionar los costes de longitud y de peligro de los segmentos involucrados en cada ruta. La suma de éstos constituye los costes de exposición al peligro y de longitud de la ruta completa.

OBJETIVOS
- ✓ Búsqueda de las K trayectorias más cortas entre origen y destino
  - ➢ Algoritmo Básico (Dijsktra).
  - ➢ Método de Eppstein.

ENTRADAS
- ✓ *bk*: algoritmo de búsqueda: '*dj*' o '*ep*'.
- ✓ K: número de rutas que se desea computar.
- ✓ *arcs*: arcos del grafo de la búsqueda, según bk:
  - ➢ *'dj'*: sg=[sgx,sgy]
  - ➢ '*ep*' nodos.
- ✓ *orig*: Origen de la búsqueda, según bk:
  - ➢ '*dj*', coordenadas x e y del UAV.
  - ➢ *'ep'* nodo s (corresponde al UAV en el grafo).
- ✓ *dest*: destino de la búsqueda, según bk:
  - ➢ *'dj'*, coordenadas x e y del objetivo.
  - ➢ '*ep*' nodo t (corresponde al objetivo en el grafo).

SALIDAS
- ✓ T_v: matriz con los segmentos de la trayectoria.
- ✓ *L*t : costes de longitud de los segmentos de cada ruta.
- ✓ *Ct*: coste de exposición al peligro de los segmentos de las rutas.
- ✓ C_v: coste total de cada ruta.
- ✓ C_l: coste de longitud de cada ruta.

Tabla 10: Resumen de *ptkuav.m.*

2.4.4.1. La construcción del grafo dirigido: la función ptkg.m.

A partir de los segmentos que proporciona la función *ptvuav.m* (sólo el diagrama básico compuesto por los segmentos propiamente dichos y las uniones del UAV) se construye un grafo dirigido para que pueda ser utilizado por el método de Eppstein. Para llevar a cabo esta operación se realizan estas acciones:

- ✓ Se recorren los vértices de los segmentos asignándoles un número. Hay que tener cuidado de no asignar a un mismo vértice números diferentes, ya que pueden aparecer en diferentes segmentos. Se identifica el vértice del UAV.

- ✓ Se forman dos matrices auxiliares para ejecutar el algoritmo de *Dijkstra*. Se construye la matriz *A* con tantas filas y columnas como nodos existan en el grafo. Un elemento puede valer *inf* o el *coste* del arco si existe el que una los nodos correspondientes a esa fila y columna. La matriz *B* es análoga a *A* pero sustituyendo los valores *inf* por *ceros*.

- ✓ Se aplica el algoritmo de Dijkstra entre el nodo correspondiente al UAV y el resto de los nodos del grafo. Se efectúa simultáneamente y tiene una complejidad ~(*M+N.logN*). La matriz *A* sirve para escoger los arcos en función de los costes. La matriz *B* sirve para dirigir los arcos. Para un nodo recién escogido (fila), los nodos que se alcancen se representan por elementos distintos de cero en las columnas correspondientes y se hacen cero los elementos de la columna del nodo recién escogido que no se hayan evaluado.

- ✓ Para cada fila, las columnas distintas de cero en B corresponden a los destinos de los segmentos que parten del nodo correspondiente a la fila.

OBJETIVOS	✓	Construcción del grafo dirigido para el método de Eppstein.
ENTRADAS	✓	sx,sy: componente x e y de los extremos Dos matrices *[N_s*2].*
	✓	*V:* coordenadas x e y del UAV.
	✓	C: matriz de costes de los segmentos del diagrama básico. *[N_s*3].*
SALIDAS	✓	*nn* (número nodos),*ns*(nodos estremos),*s*(nodo origen), *sgx, sgy*.

Tabla 11: Resumen de *ptkg.m*.

2.4.4.2. La gestión de los ficheros en Eppstein: la función ptkf.m.

Se encarga de la gestión de los ficheros involucrados en el Método de Eppstein. Según el valor de la entrada *op* se realiza una de estas acciones:

Si *op=0*, se recibe la lista *nn* con los nodos extremos de cada uno de los arcos del grafo dirigido, el nodo origen *s*, el destino *p* y el coste de los arcos del grafo *C(:,1)*. Se encarga de formar el fichero *grafo.gr* que actúa como entrada del método de Eppstein. Además del fichero, la salida es el error, *cero* (correcto) o *uno* (error). Se muestra el siguiente ejemplo con un pequeño grafo con tres arcos y nodos.

```
n 3
m 3
s 1
t 2
a 1 2 1
a 2 3 1
a 3 1 1
```

Si *op=1*, se leen los resultados del método desde el fichero rep.dat. El fichero contiene el parámetro *K*, las rutas realmente computadas, y tantas filas como indique el dato anterior. En cada fila el primer elemento es el número de arcos de la ruta y el resto de elementos de la fila son los nodos que se alcanzan sucesivamente en la ruta. Se construye la matriz *Tv* en la que se encapsulan todas las rutas, rellenando con ceros las posiciones necesarias para tener una matriz cuadrada (la última columna es un vector de *ceros*). Si se computan menos de *K* rutas se repite la última hasta *K* filas. Se muestra un ejemplo en el que se han computado diez rutas en un archivo *rep.dat*:

```
10
10
9  68 50 42 19 66 67 51 43 70
8  68 42 19 66 67 51 43 70
10 68 41 50 42 19 66 67 51 43 70
9  68 50 42 19 66 65 51 43 70
8  68 42 19 66 65 51 43 70
10 68 41 50 42 19 66 65 51 43 70
11 68 50 42 19 66 67 51 43 52 44 70
10 68 42 19 66 67 51 43 52 44 70
12 68 41 50 42 19 66 67 51 43 52 44 70
9  68 50 42 19 66 65 3 39 70
```

ENTRADAS	✓	*op* (0 escribir,1 leer*), ng*(nodos), *lg*(longitudes),*s*(origen),*t*(destino)
SALIDAS	✓	*err, grafo gz* (op=0) *ó Tv* (op=1).

Tabla 12: Resumen de *ptkf.m*.

2.4.4.3. El algoritmo básico para la búsqueda: la función ptkd.m.

Se trata de una función MATLAB que efectúa la búsqueda de las K rutas más cortas entre un vértice origen y un nodo destino. Computacionalmente es mucho peor que el método de Eppstein, pero al estar codificado en MATLAB permite la ejecución del sistema en el entorno de ese programa.

Es un algoritmo básico que recibe las coordenadas x e y de los extremos de los segmentos y los costes de cada uno, además de las coordenadas del origen y destino de la búsqueda. A la salida se proporciona la matriz *Pk* con las rutas seleccionadas (en cada fila una ruta con los números de segmento en cada columna, ordenados secuencialmente. La operación se resume en los siguientes pasos, comenzando siempre con el nodo correspondiente al UAV:

- ✓ Paso 1: Desde cada nodo evaluado (comenzando por el UAV) se seleccionan los arcos que partan de ese nodo y que no hayan sido computados. Se añaden a la lista de arcos candidatos *ia* ordenados en orden creciente del coste que se almacena en *ja*.

- ✓ Paso 2: De la lista de arcos candidatos se selecciona en cada iteración el primer elemento (menor coste). Y se añade en la matriz de salida *P10* en las filas cuyos últimos elementos sean los arcos precedentes o añadiendo nuevas filas (rutas) en el caso en que existan arcos intermedios que sean precedentes del arco. El arco seleccionado se elimina de la lista de arcos candidatos.

- ✓ Paso 3: Se evalúa el número de rutas que tienen su final en el nodo destino (objetivo). En el caso en que se hayan computado *K* rutas se sale del procedimiento. También se sale de él cuando no existan arcos en la lista de arcos candidatos. Puede ocurrir que no se hayan computado las *K* rutas. En caso contrario, volver al Paso 1.

- ✓ Paso 4: Se forma la matriz con los segmentos recorridos en cada ruta. Si todo va bien se obtienen K filas para la matriz *Pk* de salida. En otro caso se rellena hasta tenerlas repitiendo la última ruta.

2.4.5. Los programas en C del Planificador de Trayectorias.

2.4.5.1. El programa EPPSTEIN.

El programa EPPSTEIN cuenta con las siguientes funciones programadas en C y que desarrollan la búsqueda rápida de las K trayectorias según las especificaciones que se proporcionen en el archivo *grafo.gr* y guardando el resultado en *rep.dat*:

- ✓ loadgraph.c: Se encarga de leer el fichero de entrada *grafo.gr* para poder formar las estructuras definidas en el método.

- ✓ dijkstraRev.c: algoritmo básico que permite la búsqueda de las ruta más corta desde un nodo hasta el resto de los nodos del grafo, tanto en el grafo original desde el origen como en el grafo revertido desde los demás nodos.

- ✓ EPPSTEIN.c: contiene la rutina principal de funcionamiento. Se encarga del control del procedimiento, realizando las llamadas a las funciones que se precisen e interpretando los datos que les entregue. Se forma el fichero de salida rep.c.

- ✓ chronometer.c: Si se desea computar el tiempo que lleva ejecutar el algoritmo básico en cada ejemplo. Con la técnica empleada es un tiempo despreciable si se dispone de pocos nodos.

- ✓ Asociadas a las anteriores, aparecen las siguientes librerías: Loadgraph.h, DijkstraRev.h,, chronometer.h y EPPSTEIN.h.

- ✓ Makefile: se encarga de compilar el programa, generando los archivos objeto loadgraph.o, chronometer.o y dijkstra.o. Genera el programa ejecutable EPPSTEIN.

Estas funciones se adjuntan en el Anexo 1. Para compilar el programa se utiliza el comando que genera los objetos y el programa ejecutable:

>>make EPPSTEIN

Para evaluar el grafo existen diversas opciones que ofrece el programa, como la computación del tiempo o la posibilidad de aparecer en pantalla los números de los caminos. La orden empleada es la siguiente:

>>gcc EPPSTEIN grafo.gr K [-paths] [-tdijkstra]

El valor K corresponde al número de rutas que se desean computar. Entre corchetes aparecen las opciones enumeradas, eliminándolos si no se consideran. Si se desean utilizar aparecería el texto sin corchetes.

La forma de los ficheros se ha descrito en el *apartado 2.4.4.2*. correspondiente a la función *ptkf.m*.

2.4.5.2. La elaboración del diagrama de Voronoi: el programa TESS.

Existen programas que en lenguaje C elaboran el diagrama de Voronoi de una forma más eficiente que el comando MATLAB y sobre todo más completa (rayos). Son métodos que están a disposición fácilmente por lo que no se adjuntan en este proyecto. Las funciones que componen este programa son las siguientes_

- ✓ *main.c*: función principal del programa. Desde este fichero se realizan las acciones a través de llamadas a las demás funciones del programa.
- ✓ *voronoi.c*: implementación del algoritmo básico de Fortune.

- ✓ *heap.c:* gestión de los conjuntos que se almacenan en las estructuras.
- ✓ *geometry.c*: inclusión de los conjuntos anteriores en las estructuras de la búsqueda.
- ✓ *edgelist.c*: control de la tabla de barrido.
- ✓ *output.c*: resultados de salida en el espacio primal (grafo original), ya que la formación del diagrama se ha realizado en el transformado.
- ✓ *memory.c*: gestión de la memoria requerida para guardar las estructuras.

- ✓ *defs.h*: librería asociada al programa.
- ✓ *makefile*: fichero para compilar el programa.

Capítulo 3: Asignación de objetivos.

La asignación de objetivos a los diferentes UAVs es la segunda tarea que tiene que resolver el sistema, cometido encargado al Director de Asignación. La asignación de N_o objetivos a N_v vehículos aéreos no tripulados constituye un problema de complejidad N_o^{Nv} . Esta complejidad se puede reducir asumiendo ciertas simplificaciones que van a dar lugar a un algoritmo de asignación de objetivos a vehículos no tripulados evitando los efectos de las amenazas.

El problema de la Asignación es un problema de optimización muy estudiado, se trata de asignar a los UAVs ciertas localizaciones (objetivos a alcanzar) evitando la presencia de elementos que representan un riesgo (en aplicaciones civiles pueden tratarse de obstáculos físicos y en aplicaciones militares ser radares que detecten la presencia del vehículo aéreo no tripulado). La evaluación de estos aspectos constituye la decisión individual de cada UAV.

Desafortunadamente, una asignación óptima requiere una complejidad computacional que se vuelve intratable a medida que aumenta el número de participantes (UAVs y objetivos). Debido a esta circunstancia se debe optar por una aproximación al problema de asignación basada en un principio básico, el conjunto de soluciones óptimas se pueden restringir a un conjunto de asignaciones individuales. Como a cada vehículo se le ha de asignar una localización, ninguno de ellos debería ser asignado a un objetivo que diese lugar a un alto riesgo.

En el problema que ocupa en el presente proyecto se ha optado por desarrollar una solución inscrita en la Teoría de Decisión que ajuste las necesidades de los vehículos no tripulados individuales con las del conjunto completo que aparezca en el escenario de trabajo en cada momento. Estas técnicas ya se han empleado con éxito en problemas de asignación de localizaciones a robots o satélites, o en situaciones de interacción entre diversos agentes.

3.1. Asignación coordinada de objetivos.

Se dispone de un escenario de trabajo a lo largo del cual se encuentran situados los N_V vehículos aéreos no tripulados. En un instante dado se realiza la asignación de los N_o objetivos a estos vehículos. Antes el sistema tiene que tener registradas la posición de las N_P localizaciones que representan un riesgo grave para la integridad de los UAVs, razón por la cual deberán ser evitadas.

La tarea del módulo es asignar a cada UAV una localización objetivo de manera que el posterior camino hacia el objetivo no implique un riesgo inasumible para el sistema porque pase muy cerca de las localizaciones que representan amenazas. Además se tienen que lograr rutas lo más cortas posible, cubrir el mayor número posible de objetivos y conseguir que los éstos sean atacados por el mayor número de vehículos posible para garantizar la efectividad de dicho ataque.

La asignación de objetivos a los diferentes vehículos se desarrolla de forma centralizada, gracias a un protocolo de asignación que proporciona una solución deseable para todos los agentes que intervengan, para lo cual debe existir un diálogo o negociación entre el módulo centralizado encargado de la tarea y los diferentes UAVs para los que la asignación no puede ocasionarles problemas graves para alcanzar con éxito el objetivo.

La aproximación a la solución óptima se basa en encontrar una función de coordinación de asignación que atienda a todos los requerimientos del problema de manera que al hacer máxima dicha función objetivo se logre la decisión adecuada para el problema. Se recuerda que la complejidad de una solución óptima al problema de Asignación en el que intervengan N_v vehículos y N_o objetivos es N_o^{Nv}, lo cual hace extraordinariamente compleja la obtención de la asignación óptima a medida que crece en el escenario de trabajo la presencia de vehículos y objetivos.

La función de coordinación de asignación evalúa cada asignación posible, valorando el cumplimiento de los diferentes objetivos que se persiguen en el problema de la asignación de objetivos. El Director de Asignación evalúa esta función objetivo para detectar la solución adecuada. La función objetivo de asignación se diseña con la intención de obtener una solución adecuada que prime los siguientes criterios:

Se fijan cuatro criterios de asignación, los dos primeros (mínima distancia y mínima exposición al riesgo) afectan particularmente a cada uno de los UAVs que forman parte del problema, mientras que los dos últimos (máximo número de vehículos en cada objetivo y máxima cobertura de objetivos) afectan al conjunto de múltiples UAVs que intervienen en la asignación.

- ✓ Mínima Distancia (mínimo gasto de combustible): entre la localización en el escenario del UAV y el objetivo asignado. Este criterio se relaciona muy estrechamente con la necesidad de ahorrar combustible, que redunda en una disminución de los costes de operación del sistema. Es obvio que cuanto menor sea el espacio recorrido por los vehículos aéreos menor será el consumo de fuel. Además cuando se sincronicen las llegadas a un objetivo de sus vehículos asignados, serán los que recorran menos distancia los que lo hagan a menos velocidad, lo cual también supone un menor gasto energético.

- ✓ Mínima Exposición al Peligro (o mínimo riesgo): de los diferentes vehículos en su camino hacia el objetivo asignado. Para ello se hace máxima la distancia entre la posición del vehículo en cada momento a lo largo de su camino hacia el objetivo y la situación de las diferentes localizaciones que representan un riesgo potencial para los vehículos aéreos no tripulados.

- ✓ Máxima Eficacia: que se consigue haciendo máximo el número de UAVs asignados a un mismo objetivo, para garantizar la eficacia de la acción represora contra el objetivo. Cuanto mayor sea el número de vehículos asignados mayor será la efectividad del ataque.

- ✓ Máxima Cobertura: se busca que el mayor número de objetivos posible sean asignados a alguno de los vehículos aéreos autónomos operativos en el sistema, de manera que se cubran la mayoría de localizaciones que tras el estudio del escenario de trabajo se han considerado objetivos.

La metodología que se sigue para la obtención de una solución adecuada escoge entre las posibles asignaciones aquellas que satisfagan a cada uno de los vehículos del sistema por separado, es decir, se permite a cada uno de los UAVs descartar los objetivos que le supongan una exposición al riesgo inaceptable o un recorrido hasta alcanzar el objetivo no asumible.

Posteriormente se garantiza el cumplimiento de los criterios que afectan al conjunto de entre las soluciones anteriormente seleccionadas escogiendo aquella que mejor los satisfaga. Será el momento de aplicar la función objetivo para seleccionar las asignaciones que hagan máximo el valor de dicha función.

3.1.1. La decisión individual.

Se ha desacoplado el problema de la asignación de manera que en un principio son los vehículos aéreos no tripulados los que valoran los criterios de Mínima Distancia y Mínima Exposición al Peligro. Para la decisión individual se proponen dos métodos muy diferentes desde el punto de vista de la operación del sistema:

- ✓ Método de las distancias: analiza una posible asignación UAV-objetivo recurriendo a las distancias entre localizaciones. Se fijará la distancia euclidiana entre el vehículo y el objetivo y la distancia al peligro en función de las distancias de la amenaza a la recta que une a los agentes asignados.

- ✓ Método de evaluación de las rutas: hace uso de los resultados obtenidos por el Planificador de Trayectorias. A partir del diagrama de Voronoi se obtienen las *K* rutas más cortas entre el UAV y los objetivos. El efecto que tienen las amenazas sobre cada segmento evalúa el criterio de mínimo peligro.

3.1.1.1. El método de las distancias.

El método de las distancias puede operar de forma autónoma con respecto al planificador de trayectorias, ya que en ningún momento se apoya en resultados arrojados por éste. Se explica como paso previo al método de evaluación de rutas que resulta más apropiado al establecer el cálculo sobre las rutas potencialmente seleccionadas para ser la referencia final.

La exposición de este método como paso previo se adjunta por resultar muy intuitiva para la comprensión del mecanismo local de asignación. No es tan riguroso como el siguiente porque trabaja sobre la línea recta que une el UAV con el objetivo, resultando inconveniente para escenarios en el que el número de amenazas sea elevado. Ante un número reducido de amenazas pueden adoptarse sus resultados.

Desde la perspectiva de cada uno de los vehículos aéreos no tripulados, y teniendo en cuenta que cada uno se dirige a uno y sólo uno de los posibles objetivos, se prefieren las localizaciones más cercanas a los vehículos, ya que les supone un menor gasto energético. Pero hay que tener en cuenta que la cercanía del objetivo no puede hacer olvidar la necesidad de escapar de las amenazas en el escenario de trabajo, pues no se puede asumir una misión que implique un riesgo alto para el UAV.

Se mide el cumplimiento del primer criterio selectivo. Si se asigna un objetivo O_j al vehículo V_i, se traza una línea recta entre ambas localizaciones cuyo módulo representa la distancia euclidiana entre ellas, denotada por $D(V_i,O_j)$.

La bondad de una asignación entre los agentes V_i y O_j, desde el punto de vista del criterio de Mínima Distancia, se mide con la siguiente fracción denotada por $\mu_{Ai}(O_j)$. El grado de 'aceptación' de una asignación UAV – objetivo si sólo se considera este criterio queda evaluado en esta expresión.

$$\mu_{Ai}(O_J) = \frac{mx_{Oj}\{D(V_i,O_j)\} - D(V_i,O_j)}{mx_{Oj}\{D(V_i,O_j)\} - mn_{Oj}\{D(V_i,O_j)\}}$$

Se trata de una relación inversa de manera que cuanto más próximos estén los objetivos al vehículo (menor distancia euclidiana) más próximo a *1* se encontrará la correspondiente función μ de dicha asignación, cuyos posibles valores se encuentran dentro del rango [*0 – 1*]. El orden de tiempo que lleva computar dicha función es $N_v.N_o$.

El rechazo de las localizaciones peligrosas constituye el segundo de los criterios que afectan a la decisión individual de cada UAV, desde el punto de vista del rechazo de aquellas localizaciones que supongan un grado alto de exposición a las amenazas. Cada UAV analiza el riesgo que le supone cada uno de los objetivos ante el escenario amenazante. para valorar el escenario, reconocer las amenazas que se le presentan y decidir qué objetivos es capaz de alcanzar sin suponerle un riesgo inasumible.

A diferencia del criterio de selección anterior, ahora se va a proponer una función que cuantifique la inconveniencia de una asignación, que se ocupa de cuantificar cuanto se expone un vehículo en su camino hacia un posible objetivo debido a la presencia de localizaciones que le representan un riesgo.

La asignación entre un vehículo V_i y un objetivo O_j se considera que tiene un grado de exposición al riesgo medido por la función $\mu_{Ri}(O_j)$. Ahora deberá tenerse en cuenta la ubicación de las diferentes localizaciones que supongan un riesgo potencial. Se define una distancia al peligro $D_P(V_i,G_j,T_k)$ que evalúa el riesgo que supone la amenaza P_k al vehículo V_i cuando a éste se le asigna el objetivo O_j

- Si la distancia entre el vehículo y la amenaza (d_1) es superior a la distancia entre la amenaza y el objetivo (d_2), esta última será la distancia al peligro.

$$D_P(V_i, O_j, P_k) = d_2$$

- En el caso de estar más próxima la amenaza al vehículo que al objetivo ($d_1<d_2$), la distancia al peligro se calcula proyectando ortogonalmente desde la amenaza hacia la recta que une el vehículo y el objetivo y tomando el módulo de este nuevo segmento (si se proyecta dentro del segmento):

$$D_P(V_i, O_j, P_k) = d_3$$

- Si $d_1<d_2$ pero el punto al proyectar ortogonalmente queda fuera de la recta vehículo – objetivo, se considera la distancia entre la amenaza y el objetivo:

$$D_P(V_i, O_j, P_k) = d_1$$

En la siguiente figura se muestra una situación en la que un UAV y un objetivo se encuentran cerca de dos localizaciones peligrosas. La localización T1 está más próxima del UAV que del objetivo, razón por la que se toma la distancia *d3*, mientras que T2 está mas cerca del objetivo, seleccionando la distancia *d2*:

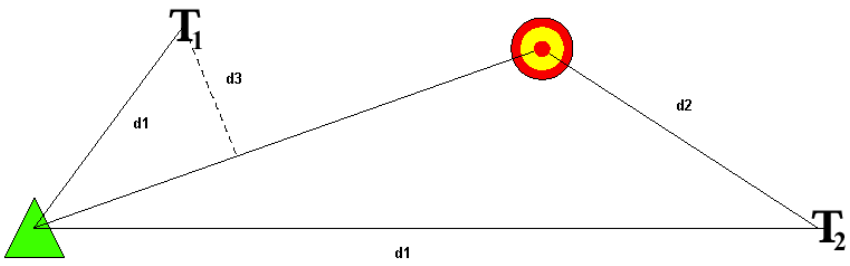

Figura 14: Determinación de la distancia al peligro.

La distancia al peligro entre el vehículo V_i y el objetivo O_j, una vez determinadas las distancias al peligro que ocasionan cada uno de las amenazas, será la mínima de todas ellas (la amenaza más peligrosa será la más cercana al camino entre el vehículo y el objetivo, y por tanto la más significativa).

$$D_P\left(V_i, O_j\right) = mn_{Pk}\left\{D_P\left(V_i, O_j, P_k\right)\right\}$$

Se propone la función $\mu_R(V_i,O_j)$ para evaluar la inconveniencia de la posible asignación entre el vehículo V_i y el objetivo O_j. La función $\mu_R(V_i,O_j)$ mide el rechazo que provoca la asignación del objetivo O_j al vehículo aéreo no tripulado V_i: desde el punto de vista de la Mínima Exposición al Peligro:

$$\mu_{\Re i}\left(O_j\right) = \frac{D_P\left(V_i, O_j\right) - mn_{Oj}\left\{D_P\left(V_i, O_j\right)\right\}}{mx_{Oj}\left\{D_P\left(V_i, O_j\right)\right\} - mn_{Oj}\left\{D_P\left(V_i, O_j\right)\right\}}$$

Cuanto menor sea la distancia al peligro, menor será el valor de la función μ_R. Toma valores comprendidos entre *0 – 1*, los más cercanos a *cero* indican una menor exposición al peligro para el vehículo ante el objetivo, mientras que valores próximos a *uno* representan asignaciones con un alto grado de riesgo. La computación de esta función se resuelve en un tiempo del orden $N_v.N_o$.

Una vez evaluados ambos criterios se identifican el conjunto de objetivos que pueden ser cubiertos por un determinado vehículo. Constituye la Decisión Individual del UAV correspondiente al subíndice *i*, que junto al del resto de vehículos será tomada en consideración para coordinar la asignación de los objetivos.

$$S_{Vi} = \left\{O_j : \mu_{Ai}\left(O_j\right) \geq a_i \cdot \mu_{Ri}\left(O_j\right)\right\}$$

Se trata de un balance para cada uno de los vehículos en el que se somete a todos los objetivos a la evaluación de los criterios de Mínima Distancia y Mínima Exposición al Riesgo. Se aceptarán las asignaciones que sean lo suficientemente cortas sin afectar al vehículo a un grado de exposición al peligro inaceptable.

Se puede ajustar la importancia de cada uno de los criterios anteriores a través del parámetro a_i de la función objetivo y cuyo valor se encuentra dentro del rango *[0-1]*. Cuanto más cerca de 0 más se despreciarán los efectos de las amenazas.

Un vehículo aceptará aquellos objetivos que cumplan la desigualdad anterior, evaluada respecto de los casos más y menos favorable. Una vez determinadas las asignaciones válidas desde el punto de vista de una decisión individual de cada uno de los vehículos aéreos no tripulados habrán de tenerse en cuenta otros factores que influyen en la decisión del conjunto (asignación coordinada de objetivos).

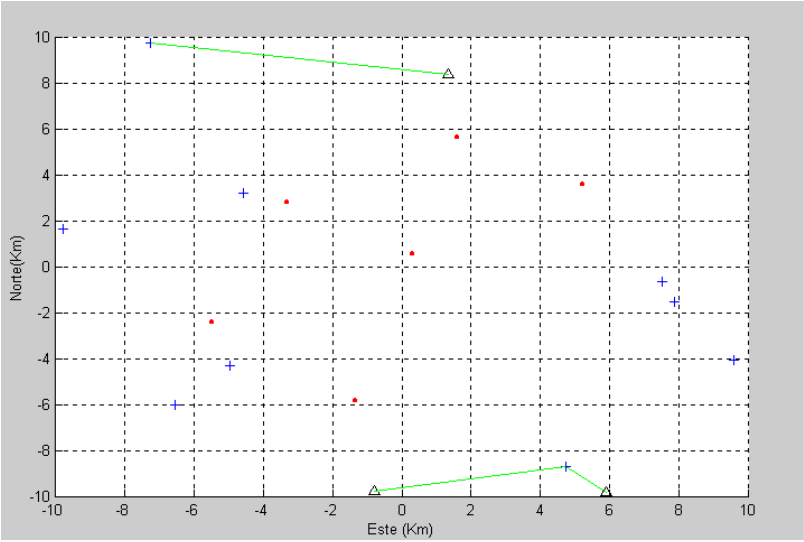

Figura 15: Decisiones individuales de los UAVs.

3.1.1.2. Método de evaluación de las rutas.

Se recomienda esta versión para efectuar la decisión individual ya que genera resultados más adecuados sobre todo al aumentar el número de localizaciones amenazantes en el escenario. El método anterior funciona correctamente cuando el número de vehículos, objetivos y amenazas en menor de diez. La proliferación de localizaciones peligrosas resulta muy sensible ya que deja de ser significativa una amenaza para representar el grado de exposición al peligro que sufre un vehículo asignado a un determinado objetivo.

El método de evaluación de rutas tiene en cuenta las trayectorias planificadas para cada pareja UAV-objetivo. Por tanto la ejecución del Planificador de Trayectorias debe ser previa y necesaria para la buena decisión individual del UAV. Al disponer de las K rutas más cortas entre el UAV y cada objetivo se evalúan los criterios sobre las trayectorias que van a ser analizadas en futuros procesamientos, a diferencia del método anterior que parte de una aproximación (recta entre los dos agentes).

En lugar de recurrir a las distancias descritas en el método anterior, se hace referencia en el presente a los costes de longitud y de exposición al peligro de los segmentos que forman parte de las diferentes rutas. La definición del coste de un segmento *s* se ha detallado en el capítulo anterior (apartado 2.1.2.).

$$C_L^S = L_S$$
$$C_P^S = L_S \cdot \sum_{p=1}^{P} \left(\frac{1}{D_{1/6,S,p}^4} + \frac{1}{D_{1/2,S,p}^4} + \frac{1}{D_{5/6,S,p}^4} \right)$$

Si se valora la asignación del objetivo *j* al UAV *i*, se tendrán en cuenta los costes de los segmentos de cada una de las *K* rutas planificadas entre ambos. Para cada ruta se realiza la media de los costes de longitud y exposición al peligro de los arcos incluidos en dichas trayectorias.

El grado de aceptabilidad que presenta la asignación V_i - O_j mide el criterio de Mínima Distancia en función del coste medio por segmento de las *K* rutas planificadas considerando las rutas de mayor y menor longitud así como el coste medio por ruta.

$$\mu_{Ai}(O_j) = \frac{mx_{Tk}\left\{\overline{C_L^S}(V_i, O_j)\right\} - \overline{C_L^S}(V_i, O_j)}{mx_{Tk}\left\{\overline{C_L^S}(V_i, O_j)\right\} - mn_{Tk}\left\{\overline{C_L^S}(V_i, O_j)\right\}}$$

El grado de rechazo de la asignación V_i - O_j desde el punto de vista de la Exposición al Peligro se mide a través del coste de peligro de los segmentos de las K trayectorias planificadas. Se consideran las trayectorias afectadas en mayor y menor grado por las amenazas y se calcula el grado medio de exposición al peligro por ruta:

$$\mu_{Ri}(O_j) = \frac{\overline{C_P^S}(V_i, O_j) - mn_{Tk}\left\{\overline{C_P^S}(V_i, O_j)\right\}}{mx_{Tk}\left\{\overline{C_P^S}(V_i, O_j)\right\} - mn_{Tk}\left\{\overline{C_P^S}(V_i, O_j)\right\}}$$

La decisión individual se toma en función de los criterios de Mínima Longitud y Exposición al Peligro. Se pueden evaluar en un tiempo ~($K.N_V.N_O$). El conjunto de decisiones individuales de todos los UAVs tiene una complejidad del orden del producto de la computación de cada vehículo y es en todo caso muy inferior a N_o^{Nv}.

$$S_{Vi} = \left\{O_j : \mu_{Ai}(O_j) \geq a_i \cdot \mu_{Ri}(O_j)\right\}$$

3.1.2. La decisión colectiva.

Los criterios que han servido para evaluar las asignaciones para cada vehículo de manera individualizada deben someterse ahora a una puesta para seleccionar de forma coordinada para el conjunto de múltiples UAVS la asignación más adecuada de acuerdo con los criterios de Máxima Efectividad para cada objetivo y Máxima cobertura. Se trata de una tarea dificultosa pues se dispone de un número elevado de asignaciones, sobre todo al aumentar el número de agentes involucrados.

La complejidad de llevar a cabo la decisión individual, si $|S_{Vi}|$ es el tamaño del conjunto S_{Vi} (decisión individual del UAV *i*), será del orden $\Pi_{i=1...N}|S_{Vi}|$, muy inferior a N_o^{Nv} debido a que $|S_{Vi}| < N_o$.

Ahora se tendrán en cuenta los dos criterios colectivos que compiten en la decisión coordinada de la asignación, por un lado el de Máxima Eficacia que exige que sea el mayor número posible de UAVs el que ataque un objetivo, por el otro el de Máxima Cobertura que demanda que el mayor número posible de objetivos sean cubiertos por vehículos aéreos no tripulados.

3.1.2.1. Criterio de máxima eficacia.

El criterio de Máxima Eficacia se puede evaluar a través de un valor que indique el tamaño del conjunto de vehículos asignados a cada objetivo, de manera que muestre cuanto mejor es la elección de un conjunto amplio de vehículos que uno pequeño.

Se necesita una función creciente que varíe significativamente entre *uno* y *tres* miembros del conjunto de vehículos asignados a un objetivo, ya que se considera *dos* vehículos asignados a un objetivo como un mínimo aceptable desde el punto de vista de la eficacia. Se consideran los objetivos asignados a cada vehículo $O=\{O_{j1}\ldots O_{jN}\}$ donde $O_{j\,i}$ es el objetivo asignado al vehículo i. Para medir el cumplimiento del criterio de Máxima Eficacia se propone la siguiente fracción:

$$U_{ME} = \frac{1}{1 + e^{-d1\cdot(m(G_J)-d2)}}$$

Con respecto a la función $U_{ME,,}$ al consideran oportunos los conjuntos de al menos *dos* vehículos aéreos para los objetivos que se decidan cubrir, el exponente reduce en *dos* unidades el número de vehículos asignados a un objetivo. Por tanto el exponente debe hacer que la función refleje un salto importante entre el hecho de asignar *uno* o *dos* UAVs al objetivo. Además se prefieren los conjuntos similares que cuenten entre *dos* y *seis* vehículos, tendiendo a conjuntos de *cuatro* al introducir el criterio de Máxima Cobertura.

	1	2	> 3 <	4	5
1 UAV	0,2689	0,1192	0,0474	0,0180	0,0067
2 UAVs	0,5000	0,5000	0,5000	0,5000	0,5000
3 UAVs	0,7511	0,8808	0,9526	0,9820	0,9933
4 UAVs	0,8808	0,9820	0,9975	0,9997	1,0000
5 UAVs	0,9526	0,9975	0,9999	1,0000	1,0000
6 UAVs	0,9820	0,9997	1,0000	1,0000	1,0000
7 UAVs	0,9933	1,0000	1,0000	1,0000	1,0000

Tabla 13: U_{ME} para distintos factores multiplicativos en el exponente.

Sólo los factores multiplicativos mayores o iguales que *tres* diferencian claramente el salto entre *uno* y *dos* vehículos. Se comprueba como U_{ME}, escogiendo un factor multiplicativo *tres* en el exponente, crece entre *dos* y *seis* miembros, diferenciando claramente el número de UAVs asignados. Se demuestra la idoneidad de la elección del factor multiplicativo *tres* en el exponente cuando se pretenden conjuntos de UAVs entre dos y seis ya que a partir de ese número el valor de la función se mantiene constante e igual a *uno*.

Dependiendo de las especificaciones se pueden variar los parámetros d_1 y d_2 de la función U_{ME}. La tabla de los factores en el exponente d_1 se ha elaborado para un mínimo de UAVs admisible en cada objetivo igual a *dos*. Si se varía esta especificación se evaluaría la función en la nueva situación. Sin pérdida de generalidad se va a trabajar con las siguientes especificaciones:

- $d_2 = 2$: Tamaño mínimo del conjunto →*2* UAVs por objetivo.

- $d_1 = 3$: Número máximo de UAVs recomendado →*6* UAVs por objetivo.

El número de UAVs asignados al objetivo *O,* queda determinado en el conjunto $m(O_j) = | \{ V_i : O_j = O_{ji} \} |$, de manera que $U_{ME}(O_j)$ valora la asignación desde el punto de vista de la Máxima Eficacia para el objetivo *j*.

Para una posible asignación global $\Gamma(A)=\{O_j\}$, $|\Gamma(A)|$ evalúa el número de objetivos que tengan al menos un vehículo asignado. Midiendo la efectividad del ataque a cada objetivo se cuantifica el criterio de máxima eficacia. Es el producto de los valores U_{ME} de los objetivos a los que se ha asignado al menos un vehículo. Esta medida del criterio de Máxima Eficacia será mayor cuanto mayor número de vehículos sean asignados a los objetivos, es decir se priman los conjuntos con un gran número de UAVs asignados a un objetivo.

$$V_{ME} = \prod_{O_j \in \Gamma(A)} U_{ME}(O_j)$$

3.1.2.2. Máxima cobertura posible.

Se recomienda sean cubiertos el mayor número posible de objetivos por al menos un UAV. Es el último de los criterios que afectan a la decisión colectiva. Se ha formado el conjunto de objetivos cubiertos por vehículos, denominado $\Gamma(A)$, por tanto una manera de medir el cumplimiento de este criterio de selección es el tamaño de este conjunto ya que muestra el número de objetivos alcanzados por al menos un vehículo. La cobertura de objetivos queda expresada de la siguiente manera:

$$V_{MC} = |\Gamma(A)|$$

3.1.2.3. Máxima Eficacia y Cobertura: dos objetivos contrapuestos.

Los dos criterios que intervienen en la decisión coordinada de los objetivos a cada vehículo dificultan la elección de una asignación óptima debido a que los dos intereses se contrarrestan. Desde el punto de vista de Máxima Eficacia para cada objetivo se prefiere un conjunto amplio de UAVs asignados, opción que dificulta que el mayor número de objetivos posible sea alcanzado al menos por UAV, que se corresponde con el criterio de Máxima Cobertura.

Al valorar conjuntamente los criterios, la decisión colectiva se puede expresar de esta manera: $V_{col} = V_{MC} \times V_{ME}$. Este producto rechaza las asignaciones que no convengan al sistema. La experiencia avala que se prefieren múltiples conjuntos de tamaño parecido en lugar de múltiples conjuntos de tamaño muy diferente.

3.1.3. Valoración de la asignación: la función objetivo.

La complejidad del Problema de Asignación ha quedado manifiesta, ya que el el número de agentes, UAVs u objetivos, puede dificultarlo extraordinariamente. Se han identificado los cuatro criterios que afectan al problema, y como éstos se agrupan según afecten a las decisiones individuales o colectivas. La mínima distancia y exposición al peligro se valoran desde cada vehículo, analizando los objetivos presentes y determinando los que está en disposición de alcanzar. Se mide para cada objetivo el cumplimiento de estos dos criterios. La operación se resume en:

$$S_{Vi} = \left\{ O_j : \mu_{Ai}\left(O_j\right) \geq a_i \cdot \mu_{Ri}\left(O_j\right) \right\}$$

La máxima eficacia y cobertura se valoran de forma coordinada para todos los UAVs. No se trata de aspectos que puedan decidir los vehículos de forma individualizada, ya que han de tenerse en cuenta todos los agentes involucrados. De forma conjunta y centralizada se decide el número y tamaño de los conjuntos de UAVs asignados a cada objetivo. Se recuerda de forma compacta la decisión colectiva:

$$V_{col} = \left|\Gamma\left(A\right)\right| \times \prod_{O_j \in \Gamma\left(A\right)} U_{ME}\left(O_j\right)$$

En general primarán los criterios que valoran el escenario de forma conjunta (máxima eficacia y máxima cobertura) frente a las necesidades e intereses individuales de los diferentes vehículos. Se debe a que la decisión colectiva es una operación más compleja y crítica. Por ello se le otorga a la decisión colectiva mayor importancia, a través de un factor que multiplique a la decisión individual de valor comprendido entre 0 y 1 (*b [0…1]*). La función de coordinación de las asignaciones es:

$$FC_A = V_{col} + b \cdot mn_i \left\{ \mu_{Ai}\left(O_j\right) - a_i \cdot \mu_{Ri}\left(O_j\right) \right\}$$

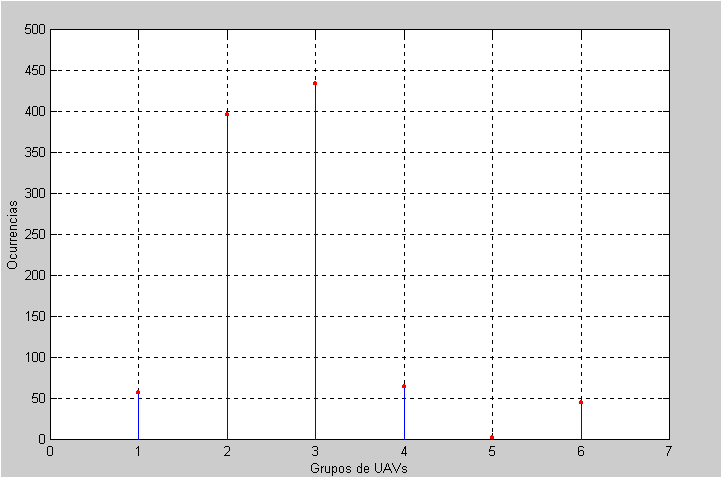

Figura 16: Decisiones de 1000 escenarios diferentes.

Para ver como actúa la función objetivo, para 1000 escenarios aleatorios compuestos por 6 UAVs, 9 objetivos y 20 amenazas se obtienen las siguientes ocurrencias para grupos de objetivos en este orden: más de 3 objetivos, tres objetivos cubiertos por 2 UAVs, 2 objetivos cubiertos por 3 UAVs, 2 objetivos cubiertos por 4 y 2 UAVs respectivamente, 2 objetivos cubiertos por 5 y 1 UAV respectivamente y 1 objetivo cubierto por 6 UAVs. Se tiende a conjuntos de 2 objetivos con 3 UAVs y conjuntos de 2 UAVs con 2 UAVs cada uno.

Se expone ahora un ejemplo. El resultado final del Director de Asignación (*Figura 17*) demuestra como se han valorado los criterios colectivos sin despreciar los individuales. Existen dos objetivos atacados por tres UAVs cada uno. El valor de la función objetivo vale *1.8586*, mucho mayor que la primitiva *0.4705* (*Figura 16*).

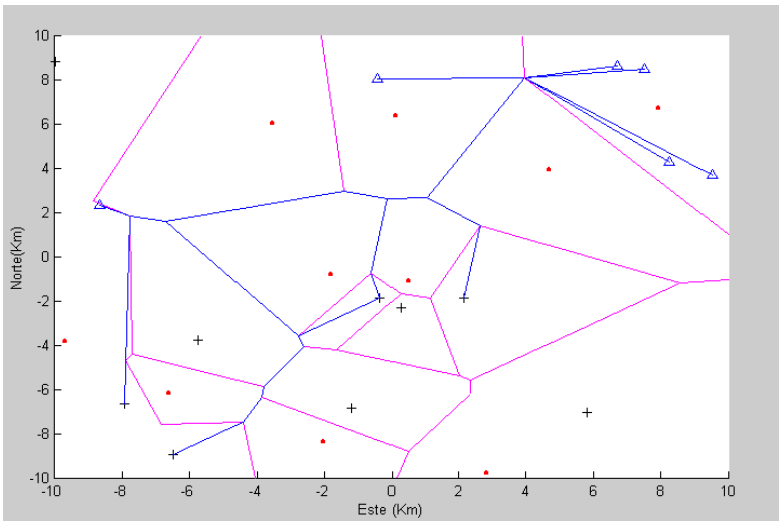

Figura 17: Decisiones individuales de un conjunto de UAVs.

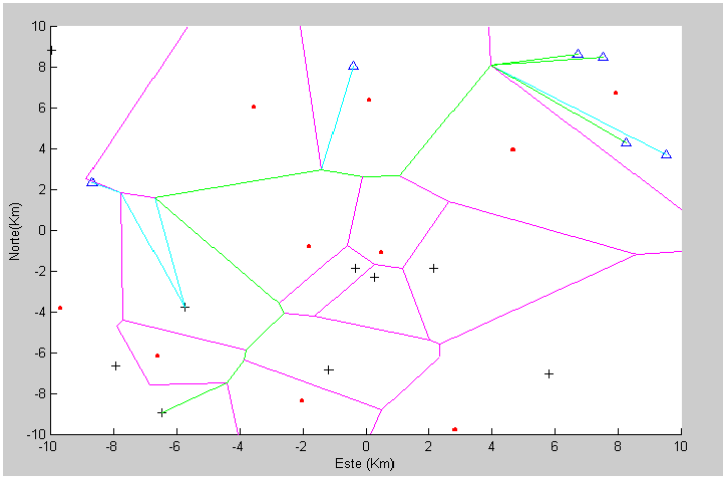

Figura 18: Asignación realizada con el método de evaluación de rutas.

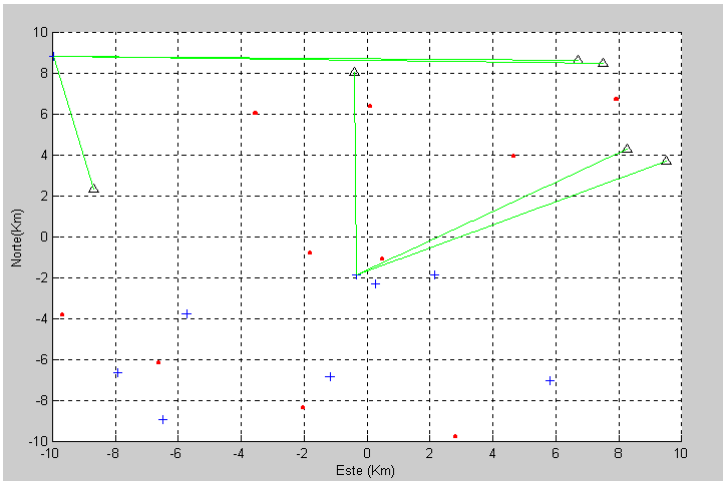

Figura 19: Asignación realizada con el método de las distancias.

Si se selecciona el método de las distancias se consigue una asignación compuesta por dos objetivos atacados por tres UAVs cada uno. En este caso no se atiende a la densidad de segmentos de Voronoi, ya que se evalúa la línea recta entre objetivo y UAV. Cuando hay pocas localizaciones el método de las distancias obtiene una solución muy adecuada y además en un tiempo inferior al método de evaluación de rutas, que es mucho mejor cuando la densidad de amenazas es considerable.

En el método de evaluación de las rutas se ha evitado la asignación realizada del objetivo situado en la parte central ya que es considerable la presencia de obstáculos en esa zona. Por el contrario se ha optado por un objetivo más alejado que el que resulta en el método de las distancias, pero mucho más accesible. En cualquier caso la solución es adecuada, aunque seguramente no la mejor de todas. La determinación de la solución óptima requiere un procesamiento muy exigente.

3.2. El Director de Asignación.

En el *apartado 2.2*. se ha establecido el procedimiento que da lugar a la asignación encargada a este módulo. En el Director de Asignación se consideran todas las uniones posibles UAV - objetivo, para seleccionar aquellas que cumplen la restricción de mínima distancia y mínima exposición al peligro (*2.2.1*.). Esta operación se conoce como decisión individual al evaluar cada vehículo aéreo no tripulado los objetivos que potencialmente está en disposición de llegar a ellos.

Una vez seleccionados los objetivos, en cada UAV se ha de tomar la decisión que mejore al máximo la cobertura de objetivos y la efectividad de la operación (*2.2.2*). Es una decisión conjunta tomada de forma centralizada. En el epígrafe *2.2.3*. aparece formulada la función objetivo que realiza la asignación de los objetivos de forma coordinada para los diferentes vehículos de acuerdo con los criterios señalados.

3.2.1. Codificación del módulo.

El Director de Asignación sitúa a sus funciones en los dos ámbitos de la arquitectura del sistema, en el sistema central y en los diferentes subsistemas UAV.

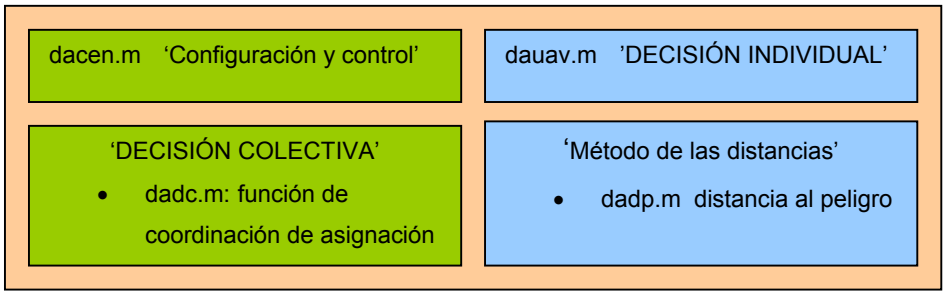

Tabla 14: Software del Director de Asignación.

En el UAV se sitúan las funciones encargadas del cálculo de la decisión individual, ya sea por el método de evaluación de las rutas o por el de las distancias, caso en el que se adjunta la función que determina la distancia al peligro. En el Sistema Central se encuentra la función encargada de la configuración y el control del módulo, así como la función que evalúa la función de coordinación de cada asignación.

Este es el resumen de las funciones que intervienen en la Asignación de Objetivos a los distintos UAVs:

- ✓ dacen.m: incluye las tareas relacionadas con el control y configuración del módulo. En esta función se inician los valores de los parámetros de la función objetivo, que determinan la importancia que se da a los diferentes criterios de selección: *a* (Mínima Longitud frente a Mínima Exposición al Peligro) y *b* (Decisión Colectiva frente a Selección Individual). Se codifica también el número máximo de UAVs preferente gracias al parámetro *e*. Se encarga de invocar la ejecución de las funciones del módulo en el sistema central.

- ✓ dadc.m: evaluación de la función de coordinación de asignación que mide el cumplimiento de los diferentes criterios de la decisión colectiva para una posible asignación. El resultado es un valor que depende de los cuatro criterios y de los parámetros que regulan su importancia. En el sistema central.

- ✓ dauav.m. en cada UAV. Se encarga de evaluar los criterios de selección de la decisión individual de los vehículos (mínima longitud y mínima exposición al peligro). Mide el cumplimiento que de los anteriores supone la asignación de un objetivo a un vehículo aéreo no tripulado. Se recuerda que son dos los métodos analizados aunque se recomiende el de evaluación de las rutas.

- ✓ dadp.m: si la decisión individual se realiza con el método de las distancias se utiliza esta función para evaluar la distancia al peligro UAV – objetivo.

Desde el comienzo ha quedado clara la intención de independizar al máximo cada módulo. Aunque las funciones se han codificado en MATLAB, para incrementar la portabilidad del sistema y la independencia de los componentes que lo forman se ha optado por guardar el resultado del módulo en un archivo.

El fichero *rda.dat* queda registrado en la memoria del sistema central y contiene las asignaciones seleccionadas en el orden de mayor a menor valor de la función de coordinación de asignación. Este archivo será consultado tantas veces como lo estime necesaria la cooperación entre el Interceptor Simultáneo y el Director de Asignación, de acuerdo con la configuración que se haga del Interceptor. De este diálogo surge la coordinación temporal del conjunto de múltiples UAVS.

3.2.2. Configuración y control del módulo: la función dacen.m

Incluye las tareas de configuración y control del módulo. Para tener presente la manera de llevar a cabo la asignación, se recuerda la función de coordinación:

$$FC_A = |\Gamma(A)| \times \prod_{O_j \in \Gamma(A)} U_{ME}(O_j) + b \cdot mn_i \{\mu_{Ai}(O_j) - a_i \cdot \mu_{Ri}(O_j)\}$$

El primer sumando representa la decisión colectiva, compuesta por dos factores que representan a cada uno de los criterios que la forman (máxima eficacia y cobertura del conjunto). El segundo sumando corresponde a la Decisión Individual que afecta a cada vehículo, considerando el caso más desfavorable que afecte a un vehículo como la medida del cumplimiento de los criterios de selección de Mínima Longitud y Mínimo Peligro.

La configuración del módulo desarrolla en las primeras líneas de código. Hay que introducir los parámetros que determinan las relaciones entre los diferentes criterios de selección que aparecen en la función de coordinación de asignación:

- ✓ *a*: relación entre los criterios de selección individuales de Mínima Longitud frente a Mínima Exposición al Peligro. [*0...1*]

- ✓ *b*: relación entre los criterios de selección colectiva (Máxima Cobertura y Eficacia) y los individuales (Mínima Longitud y Exposición al Peligro). [*0...1*].

- ✓ *e*: factor multiplicativo en el exponente de la función que evalúa la eficacia y que codifica la preferencia en el tamaño de los conjuntos. Normalmente *tres*.

Existe la posibilidad de considerar el mismo valor para todos los vehículos o por el contrario introducir una relación diferente para cada uno, dependiendo de las especificaciones de los vehículos. Lo habitual, si todos los vehículos tienen las mismas especificaciones técnicas, es asignar los mismos parámetro a todos los UAVs.

Si *a* vale *cero* sólo se tiene en cuenta el criterio de mínima distancia mientras que se valoran de la misma manera los criterios de decisión individual (mínima distancia y exposición al peligro) cuando se asigna *uno*.

b relaciona los criterios de decisión colectiva (Máxima Cobertura y Eficacia) y los individuales (Mínima Longitud y Exposición al Peligro). Se recomiendan valores muy próximos a cero, ya que la decisión sobre el conjunto se realiza partiendo de las decisiones individuales, influyendo desde el comienzo y a lo largo del procedimiento. Los cambios de vehículo se realizan en función del valor que toma la matriz que representa a estos criterios individuales.

Por tanto lo que se ha de considerar en la selección colectiva son los criterios que afectan a la totalidad de los vehículos, y la valoración de los criterios colectivos frente a los individuales es mayor cuanto más cercano a cero se encuentre el valor del parámetro *b*.

e corresponde al factor multiplicativo en el exponente de la función que evalúa la eficacia de una asignación. Dependiendo de la preferencia en el tamaño de los conjuntos de vehículos asignados a cada objetivo, el exponente de la función debe ajustarse (tabla del *apartado 3.1.2.1*.). Se opta por un factor *tres* ya que se prefieren conjuntos de *dos* a *seis* UAVs en cada objetivo atacado.

Se controla la ejecución del módulo con el fin para coordinar la asignación de objetivos en el conjunto de múltiples UAVs. La función *dacen.m* recibe el conjunto de decisiones individuales de los vehículos a través del Director de Comunicaciones. La matriz M_{Ip} [N_v*N_o] contiene en cada fila la evaluación de los criterios individuales con respecto a cada objetivo (columnas). El cumplimiento es mejor cuanto mayor sea el valor del elemento (*i,j*) que corresponde a la expresión:

$$\mu_{Ai}(O_j) - a_i \cdot \mu_{Ri}(O_j)$$

dacen.m devuelve el conjunto de n_a posibles asignaciones entre vehículos y objetivos, incluidas en la matriz *A* [n_a*N_V]. Cada columna corresponde a una asignación siendo el elemento de cada fila el objetivo adjudicado al UAV de esa fila. Las asignaciones se ordenan de mayor a menor (de derecha a izquierda) según el valor de la función de coordinación de asignación. En función de los criterios individuales se forma la asignación que sirve de partida para determinar la mejor asignación considerando además de la decisión individual la colectiva.

La primera asignación corresponde al máximo, para cada UAV, de la evaluación de la decisión individual de entre todos los objetivos. Esta primera asignación se registra en *S0*, a partir de la cual se evalúan los otros dos criterios. S0 es una matriz binaria de tamaño N_V*N_o. Un elementos vale *uno* cuando se decide unir los correspondientes vehículo y objetivo, y *cero* en caso contrario.

Después se calcula el valor de la función de coordinación para esta primera asignación. Se comunica a la función *dadc.m* los parámetros *b* y *e* configurados, la asignación primitiva *S0* y la matriz de decisión individual *Mlp*. La evaluación de la función objetivo de asignación se registra en la variable de salida *s*.

Se establece el orden en el que se realizan los cambios en la búsqueda de nuevas asignaciones a las que evaluar el valor *s*. Se computarán aquellas que tengan un valor positivo en la matriz M_{lp}, en orden descendente desde la asignación primitiva según el valor que presente dicha decisión. El orden de cambio queda en C_{lp} [N_V*N_o].

Iterativamente se prueban nuevas asignaciones en el orden determinado por la función C_{lp}. En cada caso se calcula el valor de la función objetivo de la asignación candidata gracias a *dadc.m* . Si el parámetro *s* mejora el de la asignación anterior se registra en la primera columna por la derecha de la matriz *A*. Este proceso se repite cuantas veces lo permita la matriz de cambios C_{lp}, proporcionando un conjunto de asignaciones entre los N_V vehículos y los N_o objetivos ordenado en orden descendente según el valor *s* en la matriz *A*.

Es importante disponer de más de una asignación, porque podría suceder que el Interceptor Simultáneo fuera incapaz de sincronizar las llegadas de los vehículos a sus objetivos. Si sucediese esta circunstancia, y dependiendo de la configuración del Interceptor, podría recurrirse a otra asignación registrada en *A* de menor valor de la función de coordinación de asignación si ésta garantizase la sincronización global. He aquí otra colisión de intereses que hay que ajustar a través gracias a la configuración de los modos de funcionamiento de los módulos implicados.

El Director de Asignación registra las asignaciones en *A* en el archivo rda.dat que contiene el número de UAVs de los que se ha recibido la decisión colectiva y el número de objetivos registrados. Contiene la matriz *A* con las asignaciones registradas. El Interceptor Simultáneo accede desde el sistema de ficheros a *rda.dat* para coordinar temporalmente la asignación en curso.

El resumen de la función de configuración y control del Director de Asignación en el sistema central se adjunta bajo estas líneas:

OBJETIVOS	✓	Configuración del módulo (*a, b, e*).
	✓	Control del funcionamiento del Director de Asignación. Llamada a la función que evalúa la decisión colectiva *dadc.m*.
ENTRADAS	✓	Procedentes de los UAVs, (Director de Comunicaciones): ➢ *Mlp*: matriz [N_v*N_o]. Decisiones individuales.
SALIDAS	✓	*err*: *0* (sin error), *1* (no hay asignaciones disponibles).
	✓	Fichero *rda.dat* contiene: ➢ n_a: número de asignaciones registradas. ➢ N_o: número de objetivos registrados. ➢ *A*: asignaciones (objetivo asignado a cada UAV).

Tabla 15: Resumen de la función *dacen.m*.

3.2.2.1. La función de coordinación de asignación: dadc.m.

Se encarga de evaluar la función de coordinación de asignación para una posible combinación de vehículos y objetivos. Recibe la asignación en cuestión a través de la matriz *S0*, la matriz de decisiones individuales M_{lp} y el parámetro *b* que relaciona las decisiones colectiva frente a individual. Devuelve a la función *dacen.c* el valor de la función de coordinación de la asignación a través de la variable *s*.

Se considera representativa de la decisión individual la pareja UAV – objetivo que hace mínima la diferencia entre las funciones que evalúan los objetivos de mínima distancia y mínima exposición al peligro. Esta será, de entre todas las posibilidades, la que representa a la decisión individual tomada por todos los vehículos:

$$s_i = mn_i \left\{ \mu_{Ai}\left(O_j\right) - a_i \cdot \mu_{Ri}\left(O_j\right) \right\}$$

La decisión colectiva tiene en cuenta el cumplimiento de los criterios de máxima efectividad y cobertura: La máxima efectividad se mide evaluando para los objetivos a los que la asignación concede al menos un vehículo el valor de la función U_{ME}. La máxima cobertura se mide evaluando el número de objetivos atacados.

La variable *sme* contiene el valor de la máxima eficacia de la asignació V_{ME}, mientras que *smc* mide la máxima cobertura. El producto de ambas, como se ha visto, corresponde al valor de la decisión colectiva de la asignación y se denota por *sc* (se recuerda el carácter contrapuesto de los criterios colectivos):

$$smc = \left|\Gamma(A)\right|$$
$$sme = \prod_{O_j \in \Gamma(A)} U_{ME}(O_j)$$
$$sc = sme \times smc$$

Por último, y dependiendo del valor del parámetro *b*, se calcula el valor de la función de coordinación de la asignación descrita por *S0* y que codifica los cuatro criterios selectivos, ya sean individuales o colectivos.

$$s = sc + b \cdot si$$

Se adjunta bajo estas líneas el resumen de la función encargada del cálculo de la función de coordinación de asignación. Se recuerda que la función es llamada desde el interior de la función *dacen.m* y que está registrada en el sistema central.

OBJETIVOS	✓	Evaluación de la función de coordinación de asignación, que codifica los cuatro criterios selectivos: ➢ Mínima distancia. ➢ Mínima exposición al peligro. ➢ Máxima eficacia. ➢ Máxima cobertura.
ENTRADAS	✓	Procedente de *dacen.m*: ➢ *Mlp*: matriz [N_v*N_o]. Decisiones individuales. ➢ *S0*: asignación evaluable. ➢ *b*: configuración de la función de coordinación.
SALIDAS	✓	Hacia el fichero *dacen.m*. ➢ *s*: valor de la función para la asignación evaluada.

Tabla 16: Resumen de *dadc.m*.

3.2.3. La decisión individual de cada UAV: la función dauav.m.

La decisión individual la toma cada UAV en el seno del subsistema correspondiente. La función dauav.m configura y determina para un vehículo aéreo no tripulado el valor de la decisión individual tomada para los objetivos registrados en el sistema. Se recuerda que existen dos alternativas, el método de las distancias y el de evaluación de las rutas que lleva a que los parámetros de entrada difieran en cada caso. Genéricamente son *e1, e2* y *e3*. La salida del fichero es un archivo rdai.dat que contiene la decisión individual del UAV.

La configuración del módulo se refiere a la elección del método aplicable y de la relación entre los criterios de mínima distancia y exposición al peligro:

- ✓ *ai*: relación entre los criterios de selección individuales de Mínima Longitud frente a Mínima Exposición al Peligro para el UAV *i*. [*0...1*]

- ✓ *mdi*: método elegido para la decisión individual del vehículo aéreo no tripulado, y que provocará la ejecución de un código independiente para cada caso. Según *mdi*: '*md*' (método de las distancias) ó '*mt*' (evaluación de las rutas).

Las entradas difieren según el método configurado. Las entradas *e1*, *e2*, *e3*, dependiendo de la cadena de caracteres *mdi* corresponden a:

- ➢ Si '*md*': *e1* son las coordenadas *x* e *y* del UAV, *e2* las correspondientes a los objetivos mientras que *e3* se refieren a la posición de las amenazas.

- ➢ Si '*mt*': *e1* corresponde al índice que muestra las uniones correspondientes a cada objetivo (ver sistema de ficheros), *e2* es la matriz con los costes de longitud de los segmentos mientras que *e3* corresponde al coste de exposición al peligro.

En cualquiera de los casos, la salida registrada en el fichero *rdai.dat* que se almacena en el sistema de ficheros interno del UAV contiene el número de objetivos y un vector m_{lp} de N_o elementos en el que la posición *j* representa para el UAV *i*:

$$\mu_{Ai}(O_j) - a_i \cdot \mu_{Ri}(O_j)$$

Los valores negativos indican la imposibilidad que supone al vehículo en cuestión el emparejamiento con el objetivo. Por tanto, *dauav.m* implementa la decisión de los objetivos factibles para cada UAV. En función de los valores de m_{lp} se coordina la asignación colectiva, en la se tendrán en cuenta otros criterios.

Lo primero que se calcula esta función es el grado de aceptación de cada asignación desde el punto de vista de la mínima longitud de las trayectorias que unen cada pareja UAV - objetivo. Los elementos $\mu_{Ai}(O_j)$ se almacenan en el vector *L* [N_0]. En segundo lugar se evalúa el criterio de mínimo riesgo, que almacena en el vector *P* [N_0] el valor de los elementos $\mu_{Ri}(O_j)$. Dependiendo del método seleccionado se aplican las expresiones del fundamento teórico (subapartados del 3.1.1.).

- ✓ Método de las distancias. Evaluación de la distancia euclidiana UAV-Objetivo (mínima distancia) y de la distancia al peligro (función *dadp.m*) para cada pareja posible. Se calcula de acuerdo a la expresión del apartado el vector *L* (mínima distancia) y *P* (mínima exposición al peligro).

- ✓ Método de Evaluación de las rutas. Se seleccionan los segmentos según el índice de las rutas (*e1*), accediendo a los valores del coste de longitud (*e2*) y exposición al peligro (*e3*). De acuerdo con la expresión acordada en el fundamento teórico de la solución se determina la media por segmento para cada una de las K trayectorias, seleccionando los valores mínimo, máximo y medio de los costes de longitud y exposición al peligro del conjunto de rutas. Se calcula de acuerdo a la expresión del apartado el vector *L* (mínima distancia) y *P* (mínima exposición al peligro).

Por último se formula el vector m_{lp}, en función de las matrices *L* y *P* y del parámetro *a*i configurado en *dauav.m* (el subíndice *i* denota el número del UAV).

$$m_{LP} = \left[L_i - a_i \cdot P_i \right]$$

El Fichero *rdai.dat* almacena el número de objetivos y el vector m_{lp} en la memoria correspondiente al sistema de ficheros de cada UAV. Además se transmite vía Director de Comunicaciones hacia el Sistema Central para que *dacen.m* decida de manera coordinada la asignación global.

OBJETIVOS	✓	Configuración de la decisión individual (*a y mdi*).
	✓	Decisión individual del UAV según el método seleccionado.
	✓	Si método de las distancias, llamada a la función *dadp.m*.
ENTRADAS	✓	Entradas genéricas *e1, e2* y *e3* dependientes del método:
		➢ Si 'md': v, O, P respectivamente
		➢ Si 'mt': I (índice de segmentos), Cl y Cp (costes individuales).
SALIDAS	✓	*err*: *0* (sin error), *1* (no hay asignaciones disponibles).
	✓	Fichero *rdai.dat* que contiene:
		➢ N_o: número de objetivos registrados.
		➢ *mlp*: decisión individual del UAV hacia cada objetivo.

Tabla 17: Resumen de *dauav.m*.

3.2.3.1. La distancia al peligro en el método de distancias: dadp.m.

Sólo se invoca en el caso en que se configure el método de distancias para la decisión individual del UAV. A partir de la posición del UAV y de los objetivos y amenazas determina el vector d_2 [N_o] con la distancia al peligro entre el UAV y cada uno de los objetivos. En el apartado 3.1.1.1 se detalla el procedimiento que da lugar a la distancia al peligro. Se ha compactado el cálculo para todos los objetivos.

OBJETIVOS	✓	Configuración de la decisión individual (*a y mdi*).
	✓	Decisión individual del UAV según el método seleccionado.
	✓	Si método de las distancias, llamada a la función *dadp.m*.
ENTRADAS	✓	Entradas genéricas *e1, e2* y *e3* dependientes del método:
		➢ Si 'md': v, O, P respectivamente
		➢ Si 'mt': I (índice de segmentos), Cl y Cp (costes individuales).
SALIDAS	✓	*err*: *0* (sin error), *1* (no hay asignaciones disponibles).
	✓	Fichero *rdai.dat* que contiene:
		➢ N_o: número de objetivos registrados.
		➢ *mlp*: decisión individual del UAV hacia cada objetivo.

Tabla 18: Resumen de *dadp.m*.

Capítulo 4: Interceptación Simultánea.

En determinadas aplicaciones resulta esencial la llegada de los UAVs a los objetivos en el mismo instante de tiempo para potenciar el 'factor sorpresa' en la represión de los mismos. La máxima exigencia de simultaneidad suele producirse en aplicaciones de tipo militar en las que se ordena a los UAVs atacar determinados blancos en un tiempo global mínimo de manera que minimice la respuesta del sistema de defensa atacado (detección a través de radar o represión de los UAVs). No se descarta en ningún caso la traslación de esta exigencia a aplicaciones de tipo civil, pues refuerza el criterio de Máxima Eficacia en la represión de un objetivo en virtud del cual se tiende a asignar varios vehículos aéreos no tripulados a un mismo objetivo.

Antes de afrontar las cuestiones relacionadas con la sincronización habrá que valorar las exigencias que presenta el sistema. La sincronización global de todos los UAVs a sus objetivos favorece que las llegadas a los objetivos cubiertos por el Director de Asignación se produzcan en el mismo instante de tiempo. En caso de imposibilidad de simultanear las llegadas con la asignación actual se puede optar por dos vías, sustituir la globalidad por sincronizaciones locales en cada objetivo o descartar la asignación determinada en el Director por otra de menor valor de función objetivo de asignación, pero que permita la sincronización. Se detallarán todos los aspectos.

El Interceptor Simultáneo es el módulo encargado de sincronizar las llegadas a los objetivos cubiertos en el ataque. A partir de las K rutas seleccionadas en el Planificador de Trayectorias de cada UAV, este módulo determina el conjunto de N_v rutas relativas a cada UAV que garantice el cumplimiento de esta exigencia temporal. Se escogerá para cada vehículo aéreo no tripulado la trayectoria que junto a las de los demás UAVs minimicen la función objetivo de la interceptación, que aparecerá más adelante. Varios son los enfoques desde los que se puede desarrollar el Interceptor Simultáneo, todos ellos valorados en el capítulo.

4.1. El algoritmo de cooperación.

El Interceptor Simultáneo es el módulo que evalúa la función de coordinación que permite el objetivo planteado de simultaneidad temporal de las llegadas. Las prescripciones temporales pueden ser variadas, además de esta sincronización en el ataque a los objetivos. A veces se opta por seguir una determinada secuencialidad en la represión de los objetivos. Por tanto, y dependiendo de las necesidades del sistema se adopta la coordinación temporal más adecuada. Sin pérdida de generalidad se estudia la simultaneidad de llegadas porque es la más coherente con los requerimientos del sistema (Criterio de Máxima Eficacia).

El criterio de asignación de Máxima Eficacia obliga a la llegada simultánea de los vehículos a un mismo objetivo, de manera que el Interceptor Simultáneo debe determinar el Tiempo de Interceptación Global de todos los UAVs. Éste es el resultado de la función de coordinación que adquiere como variables de coordinación los tiempos de interceptación al objetivo que los diferentes UAVs son capaces de describir para las rutas almacenadas por el Planificador de trayectorias a una velocidad admitida dentro del rango [V_{mn} , V_{mx}].

El Interceptor Simultáneo genera a la salida un conjunto de puntos entre cada UAV y su objetivo y la velocidad a la que han de recorrerse. Estas trayectorias son las obtenidas por el Planificador de Trayectorias. Por tanto, el conjunto de puntos cumple con el rechazo de las localizaciones peligrosas y la mínima longitud de las rutas. Para la coordinación temporal el Interceptor Simultáneo necesita variables de coordinación. Del Planificador de Trayectorias de cada UAV se recogen los costes de longitud (longitud total de las rutas) y los costes totales de las trayectorias trazadas por el UAV.

El Director de Asignación es esencial en la coordinación temporal del conjunto de los UAVs. De acuerdo a los criterios de selección de los objetivos para los vehículos se genera una misión para el conjunto. Es éste el marco para el que se calcula la interceptación simultánea, a partir de los resultados que el Planificador de Trayectorias ha determinado para el marco de actuación (objetivos asignados a los UAVs). Con las longitudes y costes totales de las rutas el Interceptor Simultáneo coordina temporalmente las llegadas a los objetivos. Si con la asignación el Interceptor no es capaz de lograr el objetivo de la sincronización, se recurrirá a otra asignación

El resultado del Interceptor Simultáneo es la solución del problema de optimización de la función de coordinación. Al hacer máximo el valor de dicha función, el UAV conoce cuál de las *K* rutas que lo unen con el objetivo será la trayectoria de referencia y a qué velocidad lineal debe recorrerla para estar coordinado temporalmente con los demás vehículos aéreos no tripulados.

$$Coordinación_i \leftarrow [V_{mn} \quad V_{mx}] \times \{T_{i1},...,T_{ik}\}$$

El Interceptor Simultáneo velará por el conjunto, de manera que quede minimizado el coste global de exposición al peligro. Recibe del Planificador de Trayectorias todas las rutas candidatas entre cada UAV y el objetivo en cuestión. Cada trayectoria viene descrita por un conjunto de puntos ξ_i, mientras que a cada vehículo se le asigna una velocidad directa V_i. El Tiempo de Interceptación del Objetivo depende de estos dos parámetros, la trayectoria seleccionada y la velocidad del vehículo. Por otra parte este módulo garantiza que el coste global del conjunto sea mínimo, función que depende a su vez de la trayectoria y la velocidad de recorrido.

Los datos que se transmiten son las posibles trayectorias para cada vehículo ξ_i, y las velocidades a las que son recorridas V_i, pero en lugar de ser intercambiadas entre los diferentes vehículos resulta más eficiente la transmisión hacia un elemento centralizado que evalúe la función de coordinación, que se trata del coste total del UAV al alcanzar el objetivo en un tiempo de interceptación al objetivo TIO_i.

Hay que arbitrar el procedimiento que de lugar a la evaluación de la función de coordinación temporal. Hay que repartir la carga computacional entre los procesadores del sistema central y los a bordo de los distintos UAVs. Como se ha dicho la evaluación de la función de coordinación se lleva a cabo en el sistema central, y en función de las variables de coordinación de salida del Interceptor Simultáneo, cada UAV puede determinar la ruta y velocidad de recorrido.

Este es un aspecto susceptible de discusión. Aunque resulte más intuitiva la decisión en función de las velocidades de recorrido a través de las rutas, es más eficiente y facilita la distribución de tareas el uso de los tiempos de interceptación individuales.

Los dos algoritmos de coordinación temporal propuestos utilizan las mismas variables de coordinación de entrada (Costes de longitud y de exposición al peligro de las K rutas entre cada UAV y el objetivo). La manera de evaluar la función de coordinación es diferente, así como la manera de conocer cada UAV la velocidad y la trayectoria que facilitan la coordinación temporal:

- ✓ Método de velocidades lineales: hace uso de las velocidades para evaluar la función de coordinación. El UAV sólo permite recorrer la ruta ξ_i a una velocidad V_i dentro del rango de velocidades lineales permitidas [V_{min} , V_{max}]. Se pretende que el conjunto de las decisiones produzca el coste global mínimo, aunque individualmente no sean las rutas más ventajosas. Es un esquema algo más centralizado que el método de los tiempos de interceptación y para la evaluación de la función se necesitan las longitudes y velocidades. Una vez determinada la mejor solución se comunica a los UAVs el número de trayectoria y la velocidad lineal (el UAV no realiza ningún cálculo).

- ✓ Método de los tiempos de interceptación: en este método se escoge el Tiempo de Interceptación al Objetivo más ventajoso para todos los UAV, conocido por Tiempo de Interceptación Global, y que sean los UAVs los que determinen la velocidad y ruta más ventajosa que cumplan con ese requisito temporal. Se trata de un esquema semi – distribuido, a diferencia del anterior de carácter marcadamente centralizado. En todo caso, la evaluación de la función de coordinación se desarrolla en el procesador del sistema central, pero a partir de los tiempos de interceptación al objetivo que cada UAV admite para cada ruta en el rango de velocidades admisibles. El resultado de la función de coordinación temporal es el tiempo de interceptación global (variable de coordinación) que en cada UAV se utiliza para seleccionar la trayectoria y la velocidad lineal de recorrido.

En los siguientes epígrafes se muestran los aspectos teóricos que inspiran ambas soluciones al problema de la Interceptación Simultánea. Se parte de las variables de coordinación de entrada, que son comunes aunque el tratamiento de ellas distinga a los métodos:

- ➢ Coste de longitud de las trayectorias (longitud de las rutas de cada UAV).
- ➢ Coste de exposición al peligro (de todas las rutas para los diferentes UAVs).

4.1.1. Método de velocidades lineales.

Esta solución al problema de la Interceptación Simultánea de los objetivos presentes en el sistema dado su carácter más intuitivo, a costa de presentar mayor centralidad en la decisión. Se parte de las dos restricciones que afectan a este tipo de decisión sobre las trayectorias del sistema:

- ✓ Limitación del rango de velocidades: debido a las restricciones de funcionamiento de los UAVs, se presenta un rango de velocidades lineales que los vehículos aéreos no tripulados no pueden infringir. De los documentos de especificaciones técnicas de los UAVs se obtiene el rango [V_{mn} , V_{mx}].

- ✓ Minimización del coste global de exposición al peligro de la decisión: a partir de los costes calculados para cada una de las trayectorias (segmentos de Voronoi que determinan esas trayectorias). Es uno de los factores que generan la función de coordinación temporal.

Desde cada UAV se transmite al módulo centralizado las longitudes totales de las trayectorias seleccionadas por el Planificador de Trayectorias, y los costes de exposición al peligro de estas. En el módulo central se buscan las combinaciones de trayectorias de entre las *K* rutas de cada UAV, de manera que se encuentre la trayectoria de cada vehículo aéreo no tripulado, que puedan ser recorridas a una velocidad lineal dentro del rango de validez, y que el conjunto de costes de exposición al peligro de esta combinación de rutas sea el mínimo posible. Es ésta la acción principal del Algoritmo de Cooperación.

Una vez concluidas las operaciones del Algoritmo de Cooperación, se comunica a los UAVs el número de trayectoria entre las registradas en cada uno de ellos y la velocidad lineal media que tiene que desarrollar. Individualmente los vehículos aéreos autónomos inician las tareas que le permiten interceptar el objetivo de manera simultánea con otros UAVs (Generador de Trayectoria y Controlador).

En este caso el Tiempo de Interceptación al Objetivo no se calcula directamente, sino que viene implícito a partir de la decisión de las velocidades lineales y las trayectorias que describirán los vehículos aéreos no tripulados.

De las dos restricciones, la primera en evaluarse es la correspondiente a la velocidad del vehículo aéreo. A partir de las *K* rutas por UAV se forman las posibles combinaciones. Para cada una de estas combinaciones se escoge la que presenta mayor longitud, asignándole la máxima velocidad permitida, V_{mx}, al resto de rutas de los demás UAVs, se les asigna una velocidad en función de sus longitudes, escaladas con respecto a la ruta de longitud máxima. De esta manera, para un conjunto de trayectorias se asegura que el tiempo de interceptación global sea el menor posible.

Hay que seleccionar la combinación de trayectorias de mínimo coste global. Se va a actuar sobre el coste de las rutas que recibe el módulo central, para determinar la función objetivo de interceptación. A partir de los costes de las trayectorias se introduce el factor velocidad ya determinado para cualquiera de las combinaciones posibles. La función objetivo de interceptación dependerá del coste de interceptación que presente la trayectoria en el contexto de una posible combinación, $C_{i,k,c}$:

- Si la velocidad lineal se encuentra dentro del rango de velocidades [V_{mn} , V_{mx}], se asigna a la trayectoria en el contexto de la combinación en que se encuentre el coste de la trayectoria (suma de costes de sus segmentos de Voronoi), C_{ik} .

$$C_{i,k,c} = C_{i.k}$$

- Si la velocidad lineal es inferior a la velocidad mínima, V_{mn}, el coste de interceptación se calcula sumando al coste de la trayectoria C_{ik} dos sumandos más, el máximo coste entre las N_v*K trayectorias registradas (así se asegura la elección de las trayectorias que no violen el rango de velocidades permitido si existen) y un término que indica el margen de velocidad no superado.

$$C_{i,k,c} = C_{i,k} + mx_{i=1\ldots Nv,k=1\ldots K}\left\{C_{i,k}\right\} + \left(V_{i,k,c} - V_{mn}\right)$$

- Si la velocidad lineal es superior a la máxima, se trata de la misma manera que en el caso anterior pero castigando en mayor grado el segundo sumando.

$$C_{i,k,c} = C_{i,k} + K \cdot mx_{i=1\ldots Nv,k=1\ldots K}\left\{C_{i,k}\right\} + \left(V_{mx} - V_{i,k,c}\right)$$

Una vez determinados los costes de interceptación de cada una de las trayectorias en el contexto de cada combinación $C_{i,k,c}$, se escogerá aquel conjunto de rutas pertenecientes a los diferentes UAVs que minimice la función de coordinación, que consiste en la suma de los costes de interceptación de las trayectorias correspondientes a los distintos UAVs en el contexto de cada combinación de rutas.

La función de Coordinación es la suma de los costes de interceptación que sufren los distintos UAVs. El objetivo de la Interceptación Simultánea, evaluado por el Algoritmo de Cooperación registrado se resume en la siguiente expresión:

$$mn\{FOi\} = mn_c \left\{ \sum_{i=1...Nv} C_{i,k,c} \right\}$$

De la definición de los costes de interceptación para cada trayectoria en el contexto de una posible elección de rutas, se garantiza la elección de las de mínimo coste de interceptación. Por lo tanto, siempre que el escenario de trabajo lo permita se escogerán las rutas de coste mínimo. Se garantiza la no elección de trayectorias en el contexto que les asigne una velocidad que supere la máxima permitida en el sistema.

También se logra en el caso de existir posibilidad de obtener rutas con velocidades en el rango permitido, la elección de éstas frente a las que poseen una velocidad superior a la máxima. Como se ha indicado, en determinadas situaciones será imposible encontrar una combinación óptima, y se recurrirán a rutas con velocidades superiores a la máxima, pero en el orden creciente de costes de interceptación.

El Algoritmo de Cooperación implementado con el método de velocidades lineales garantiza la determinación de la mejor opción. Por lo tanto la solución al problema de la interceptación simultánea es óptima. Como contrapartidas figuran dos aspectos, el primero tiene que ver con el coste computacional que introduce debido a que valora todas las posibilidades, el segundo tiene que ver con su carácter centralizado, que deja al UAV sin la participación que puede prestar.

Sería posible un esquema más descentralizado, cediendo a los UAV el cálculo de la función de coordinación. Pero habría que comunicar a cada UAV las longitudes de las rutas de los demás vehículos aéreos si se pretenden considerar las posibilidades que garanticen la determinación de la solución óptima.

Son N_v*K longitudes de ruta, y el UAV devolvería otros tantos valores de costes de interceptación. Pero si es posible un esquema semidistribuido en el que el UAV determina la trayectoria y la velocidad lineal a partir del tiempo global de interceptación del objetivo. Este método se conoce por tiempos de interceptación.

4.1.2. Método de los tiempos de interceptación.

Este enfoque consiste en la determinación del tiempo de interceptación global (TIG), que obliga a todos los vehículos aéreos no tripulados a llegar a sus objetivos en el tiempo marcado por ese valor temporal. El objetivo de ambos métodos es el mismo, ya que tiempos de recorrido y velocidades están relacionados, pero en este caso la decisión no se toma en función de las velocidades, sino que se hace en función de los rangos de tiempos de interceptación calculados por los diferentes UAVs para las rutas registradas por el Planificador de Trayectorias de manera que se recorran dentro del rango de velocidades admitidas.

Se está describiendo un esquema descentralizado a medias. Parte de las operaciones del Algoritmo de Cooperación se realizan en los UAVs, pero la evaluación de la función de coordinación temporal se produce en el módulo central:

1. Comunicación de Información: en los distintos UAVs se calculan los rangos de tiempo de interceptación de las rutas para ser recorridas en el rango de velocidades [V_{mn} , V_{mx}]. Se comunica al sistema central las costes de longitud y los de exposición al peligro para que pueda determinar los rangos temporales y los costes de interceptación.

2. Evaluación de la función de Coordinación: el módulo central escoge el Tiempo de Interceptación Global, que permite a cada UAV desarrollar en un tiempo de interceptación al objetivo previsto por cada vehículo y minimizando la suma de los costes de interceptación de las distintas rutas. Una vez tomada la decisión se comunica a los diferentes UAVs el Tiempo de Interceptación Global.

3. Operación en los UAVs: los vehículos aéreos no tripulados reciben el TIG. A partir de ese momento determinan cuál de las rutas registradas permite recorrer el camino hacia el objetivo en el tiempo señalado desde el sistema central minimizando el coste de interceptación de la ruta. A partir del TIG se determina la velocidad a la que tiene que recorrer la trayectoria seleccionada.

La determinación de los tiempos de interceptación es una tarea encargada a cada uno de los UAVs. Se determina el rango de tiempos de interceptación al objetivo, *TIO*, que presentan las diferentes trayectorias, {S_{TIOi}}

A partir de las longitudes que cada UAV guarda de las trayectorias determinadas en instantes anteriores y del rango de velocidades admitidas *[V_{min} , V_{max}]*, se calcula para cada una de las *K* trayectorias los instantes de tiempo mínimo y máximo. Si L_k es la longitud total de la trayectoria *k*, los límites temporales admitidos para dicha ruta serán *[L_k/V_{mx} , L_k/V_{mn}]*. Se realiza el cálculo para las *K* trayectorias.

Se define una función de coordinación, a partir de la cual se evalúan las distintas combinaciones de rutas de diferentes UAVs para seleccionar la que minimice la función de coordinación. Los costes de interceptación de las diferentes rutas tendrán en cuenta sus costes asignados por el Planificador de Trayectorias, y valorarán las circunstancias adversas por un tiempo que no pertenezca al rango de tiempos de interceptación al objetivo permitidos. El Coste de Interceptación de la ruta *k* del vehículo aéreo no tripulado *i* se detalla en función de alguna de las circunstancias:

- Para valores de tiempo *t* dentro del rango permitido *[L_{ik}/V_{mx} , L_{ik}/V_{mn}]*, el coste de interceptación al objetivo se define por el valor correspondiente del coste total de la trayectoria C_{ik}, más el tiempo transcurrido desde el extremo inferior del rango. La función de coordinación en el instante t de la ruta es:

$$CI_{i,k,t} = C_{i,k} + \left(t - L_{i.k} \middle/ V_{mx} \right)$$

- Para una determinada ruta, si *t* está fuera del rango de validez de tiempos de interceptación porque supere el tiempo máximo admisible para la ruta L_{ik}/V_{mn} el coste de interceptación deberá permitir la elección de un TIG que si se encuentre dentro del rango. Se suma el máximo de los costes de las rutas y además la diferencia de tiempo. Se propone el coste de interceptación:

$$CI_{i,k,t} = C_{i,k} + mx_{i=1\ldots Nv,k=1\ldots K}\left\{C_{i,k}\right\} + \left(L_{i.k} \middle/ V_{mn} - t \right)$$

- Si *t* está fuera del rango de validez de tiempos de interceptación porque no alcance el tiempo mínimo para la ruta L_{ik}/V_{mx}, el coste de interceptación deberá permitir la elección de un TIG que si se encuentre dentro del rango. Se suma el máximo de los costes de las rutas y además la diferencia de tiempo.

$$CI_{i,k,t} = C_{i,k} + K \cdot mx_{i=1\ldots Nv,k=1\ldots K}\left\{C_{i,k}\right\} + \left(t - L_{i.k} \middle/ V_{mx} \right)$$

Al minimizar después la función de coordinación se tenderán, dentro de los rangos válidos para todos los UAVs, a valores de tiempo más pequeños de manera que la misión dure el menor tiempo posible y aumente la efectividad del ataque disminuyendo el riesgo de ser detectado. A partir de los costes de interceptación se determina la función de coordinación.

La función de coordinación la constituyen estos términos del coste de interceptación para las diferentes rutas, que a su vez depende de los distintos tiempos de interceptación y de los costes de las trayectorias. El sistema central tendrá en cuenta la función de coordinación, que tratará de minimizar.

El resultado permitirá determinar el Tiempo de Interceptación Global (*TIG*) o tiempos de interceptación al objetivo (*TIO*) que emplean los diferentes UAVs para alcanzar el objetivo asignado de manera que se asegura la sincronización del ataque y minimizando el coste global del conjunto (función objetivo de coordinación).

$$TIG \leftarrow mn\{FC\} = mn_t \left\{ \sum_{i=1...Nv} CI_{i,k,t} \right\}$$

Como se ha indicado la determinación del *TIG* permite asegurar el coste de interceptación mínimo del conjunto, aunque en algunos de los UAVs no resulte el mínimo Coste de Interceptación. Por otro lado, cada UAV escoge para el *TIG* determinado la ruta que minimiza su coste de interceptación al objetivo. Si el cálculo lo efectúa el UAV *i*:

$$k \leftarrow mn_{k=1...K} \{C_{i,k,TIG}\}$$

Una vez determinado el tiempo de interceptación al objetivo *TIG*, cada vehículo aéreo no tripulado habrá fijado la trayectoria que finalmente seguirá en su camino hacia el objetivo. El UAV selecciona la trayectoria *k* de acuerdo con la expresión anterior (minimizando el coste de interceptación) y gracias a la longitud que guarda de ella determina la velocidad lineal:

$$V_i = {}^{L_{i,k}}\!/_{TIG}$$

He aquí la diferencia del algoritmo de Cooperación basado en el Método de tiempos de interceptación frente al de velocidades lineales. Partiendo de una misma limitación en el rango de velocidades, este método reduce el problema a la determinación del tiempo de interceptación al objetivo más adecuado para todos los UAVs. Es inmediata la determinación de la velocidad lineal que junto a la ruta seleccionada se pasan al Generador de Trayectorias.

A continuación se resume el Algoritmo de cooperación en sus distintos pasos. Se ha pretendido compartir la computación del método entre los distintos UAVs para descargar al módulo centralizado de la cantidad de tarea que tenía que desarrollar en el método anterior. El intercambio de los tiempos de interceptación al objetivo es más eficiente. El resumen se muestra en el cuadro:

PASO 1 'DETERMINACIÓN DE LOS INTERVALOS DE TIEMPO EN LOS UAVs'

- ✓ *TIO1*: Instantes de tiempo inferiores en el intervalo de tiempos de interceptación al objetivo para cada UAV.
- ✓ *TIO2*: Instantes de tiempo superiores de los intervalos de tiempo de interceptación al objetivo para los UAVs.

PASO 2 'DETERMINACIÓN DEL TIG (tiempo de interceptación global)'

- ✓ Se recibe en el módulo central procedentes de los distintos UAVs:
  - ➢ *Cl*. Costes de longitud de las rutas de todos los UAVs
  - ➢ *Cr*: Costes de peligro de las rutas de cada UAV.
- ✓ Se desarrolla el algoritmo de interceptación en el módulo central:
  - ➢ *CI*: Costes de intercepción de las rutas en los contextos.
  - ➢ *FC*: Función de coordinación que hay que minimizar.
  - ➢ *TIG*: Tiempo de interceptación global.
- ✓ Se comunica a los diferentes UAVs el *TIG*.

PASO 3 'DETERMINACIÓN DE LA VELOCIDAD Y NÚMERO DE RUTA'

- ✓ Lo lleva a cabo cada UAV.
- ✓ En función del TIG y de los costes de intercepción se determina:
  - ➢ V_l: velocidad lineal de recorrido.
  - ➢ *nr*: de las *K* trayectorias registradas, la ruta que ha de seguir.

Tabla 19: Algoritmo de Cooperación.

Se ilustra un ejemplo de operación del Interceptor Simultáneo con *6* UAVs y *9* objetivos. La velocidad V_{mn}*=0.121Km/s* y V_{mx}*=0.141Km/s*. Sobre el diagrama de Voronoi aparecen las rutas, en el mismo color para los UAVs que lleguen al mismo objetivo. De las asignaciones planificadas por el Director de Asignación se ha descartado la primera por la imposibilidad de alcanzar la sincronía global. La segunda asignación si la permite, coordinando el Interceptor Simultáneo un *TIG=130.32 seg*.

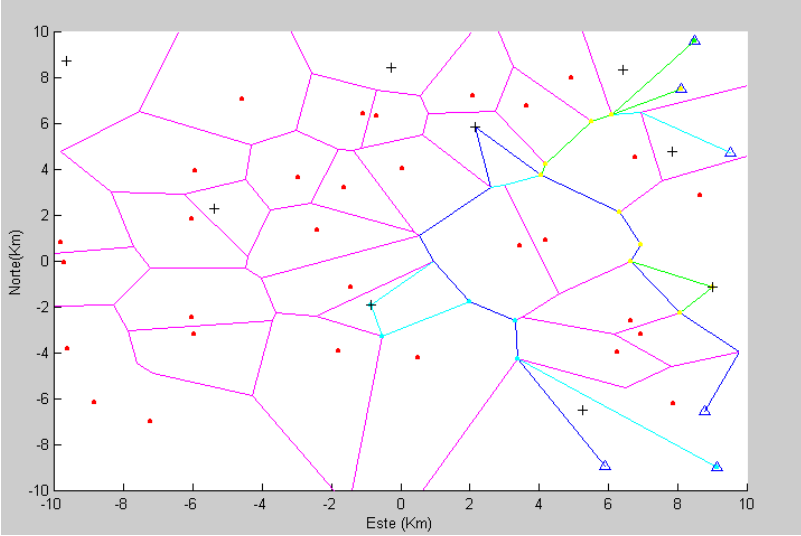

Figura 20: Interceptación Simultánea Global.

No sólo se ha descartado la opción más favorable según el Director de Asignación, también se ha pasado por alto en alguno de los casos la ruta menos costosa que el Planificador de Trayectorias del UAV ha trazado hacia el objetivo concreto, debido a que se tienen en cuenta las distancias de las trayectorias que permitan la sincronización global (homogeneización de las longitudes de las rutas que llevan a un mismo objetivo). Los puntos señalan variaciones en este sentido.

UAV	Objetivo (D.A)	Objetivo (I.S)	Ruta	Velocidad
1	1	3	1	0.12786449
2	9	1	5	0.122767561
3	9	3	1	0.141000000
4	8	7	2	0.121908331
5	1	1	7	0.121202291
6	8	7	1	0.127356781

Tabla 20: Operación del Interceptor Simultáneo para un escenario concreto.

4.2. El Interceptor Simultáneo.

4.2.1. Codificación del Interceptor Simultáneo.

Se han detallado a lo largo del apartado anterior dos visiones diferenciadas del mismo problema. La forma de abordarlo ha dado como resultado un esquema con bastantes diferencias. El Interceptor Simultáneo cuenta con funciones situadas en el sistema central y en los subsistemas UAVs.

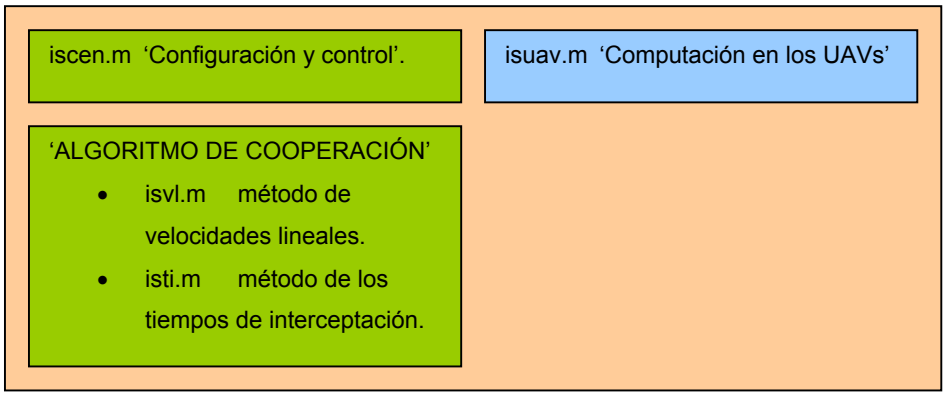

Tabla 21: Software del Interceptor Simultáneo.

Es en el sistema central donde se configura el funcionamiento del módulo. Se recuerda que puede seleccionarse el tipo de algoritmo de cooperación y la opción de simultaneidad. A partir de la recepción en el módulo central de los costes de longitud y peligro totales para cada una de las rutas desde el fichero *iscen.m* se manda simultanear las llegadas gracias a la evaluación de la función de coordinación, que se codifica en los ficheros *isvl.m* ó *isti.m*, según la opción seleccionada. Si no se consigue una solución adecuada, en función de la configuración de la opción se actúa en una u otra dirección.

Una vez evaluada la función de coordinación se comunica a los UAVs las variables de coordinación que permita a cada uno determinar la trayectoria de las registradas por el Planificador de Trayectorias y la velocidad lineal. El fichero *isuav.m* realiza la operación tras haberle sido entregados los resultados del sistema central por parte del Director de Comunicaciones. A continuación se detalla cada uno de los ficheros que forman el Interceptor Simultáneo.

4.2.2. La configuración y el control: la función iscen.m.

El Interceptor Simultáneo inicia el funcionamiento tras terminar el Director de Asignación la adjudicación de los objetivos a los diferentes UAVs. La variable *a* indica el objetivo que le ha correspondido a cada UAV *i*). Se encarga de hacer simultánea la llegada de todos los vehículos a sus objetivos para garantizar el cumplimiento del objetivo de Máxima Eficacia ('factor sorpresa'), gracias al algoritmo de cooperación (Método de velocidades lineales o tiempos de interceptación). En este fichero se configura el módulo dentro de las opciones prefijadas y se controla la ejecución del mismo en función de esa configuración.

En primer lugar se fijan las velocidades máxima y mínima, de acuerdo con las especificaciones técnicas de los vehículos aéreos no tripulados del sistema. Esta limitación en el rango de velocidades determina el margen temporal que cada ruta admite. En función de los resultados entregados, si no son válidos porque para alguno de los UAVs se exceda la velocidad lineal permitida o no se alcance la mínima, y dependiendo de la configuración del módulo que se haga en este fichero (variable *op*), el comportamiento del Interceptor Simultáneo será distinto:

- ✓ Truncar al valor de la velocidad mínima *(op=0)*: Se trunca el valor de las velocidades lineales que alcanzan la velocidad mínima permitida al valor $V_{mn.}$ Por tanto se pierde la simultaneidad de algunas llegadas, quedando seriamente dañado el criterio de Máxima Eficacia. Todos los UAVs llegarán a la vez, salvo aquellos que en virtud de la primera asignación le hayan sido asignadas velocidades inferior a la velocidad mínima.

- ✓ Sincronización local de los objetivos *(op=1)*: Se sustituye el concepto de simultaneidad global por el de local, es decir, en lugar de producirse una llegada simultánea global de vehículos a los objetivos, se optará por la simultaneidad de llegadas para cada objetivo de forma individualizada. Se ejecutará el algoritmo de interceptación tantas veces como objetivos estén afectados por la pérdida de sincronización, de manera que no se pasarán todas las trayectorias, sino aquellas que atañan a los vehículos involucrados en cada llamada. Es posible aunque no seguro, que al relajar los requerimientos se de con una solución válida. Si el resultado no es adecuado se trunca al valor V_{mn}.

- ✓ Sincronización global (*op=2*): resulta la más exigente. Si alguna velocidad de alguno de los UAVs no alcanza el valor mínimo, no se admite la asignación en curso y se recurre al Director de Asignación para probar con la siguiente asignación de máximo valor de la función de coordinación de asignación. El Director de Asignación entregará al sistema otra solución a partir de la cual se planifique nuevas trayectorias y simultaneen las llegadas en este módulo.

Existe otro aspecto en la configuración del módulo, la elección del algoritmo de cooperación. Se recuerdan las dos alternativas, el Método de las velocidades lineales y el de los tiempos de interceptación. Según el valor de la cadena de caracteres *ac*:

- Si *ac='vl'*: se aplicará al módulo el Método de velocidades lineales. Es un esquema más centralizado. No se carga en los diferentes UAVs ningún tipo de función relacionada con la interceptación simultánea. Los UAVs comunican al sistema central los costes de longitud y de exposición al peligro de las trayectorias almacenadas. El sistema central determina la mejor opción o combinación de rutas de diferentes UAVs gracias a la función *isvl.m*. El módulo central comunica al vehículo aéreo no tripulado el número de trayectoria y la velocidad lineal que ordena el sistema cumplir para llegar al objetivo en el TIG.

- Si *ac='ti'*: se aplica el Método de los tiempos de interceptación que presenta un esquema más distribuido y por lo tanto repartirá las funciones a realizar entre los distintos UAVs reservando al sistema central la decisión del conjunto. En los UAVs comunican al sistema central los costes de longitud y de exposición al peligro de las rutas, de manera que a partir de los costes de interceptación se determinará el mínimo valor de la función de coordinación. El sistema central comunica a los UAVs el tiempo de interceptación global (*TIG*) para que los UAVs rescaten la trayectoria con menor coste de interceptación para ese tiempo, y en función de ésta ruta la velocidad lineal de recorrido.

El resultado del sistema central se comunica a los diferentes UAVs. Para que puedan interpretar la información se tiene que comunicar el tipo de algoritmo de cooperación utilizado. Los parámetros que se intercambien con los UAVs son distintos según la opción, pero se conocen de antemano.gracias a la configuración. El código ejecutable para los dos algoritmos es diferente aunque compartan la función *iscen.m*, por eso se han introducido interruptores.

El resultado del algoritmo de cooperación puede no ser válido (*err=1*), desarrollando las tareas correctoras descritas en cada una de las opciones o asignando si la opción no es válida la máxima velocidad a todos los UAVs (mínimo tiempo de interceptación al objetivo), perdiendo cualquier sincronía en el sistema. En el caso *op=2* se priman los aspectos relacionados con la simultaneidad (factor sorpresa) y la máxima eficacia en contra de otros implícitos en los demás criterios de selección en la asignación de objetivos.

Cuando se dispone de la solución final se transfiere el control a cada UAV. Se comunica a través del fichero *uavrx3.dat* las salidas del Interceptor Simultáneo central:

- Para el método de escalado de velocidad se graba en el fichero las velocidades lineales y el número de trayectoria (*1...K*) para cada UAV.

- Para el método de tiempos de interceptación se comunica el tiempo de interceptación al objetivo de cada UAV (*TIG* o *TIL*) y un indicador de sincronía.

OBJETIVOS	✓ Configuración del módulo (*Vmx,Vmn,op,ac*). ✓ Control del funcionamiento del Interceptor Simultáneo. Llamada al algoritmo de cooperación en función de las opciones (*opc,ac*): ➢ Si *ac='vl'*, llamada al método de velocidades lineales (isvl.c). ➢ Si *ac='ti'*, llamada al método tiempos interceptación (isti.c). ➢ En función de los resultados, y dependiendo de *op* se trunca al valor mínimo de la velocidad, se prueba con la simultaneidad local o se deshecha la asignación *a* .
ENTRADAS	✓ Procedentes del módulo centralizado: *a* (asignación [N_v*1]). ✓ Procedentes de los UAVs, (Director de Comunicaciones): ➢ *Cl*: matriz [*K*N*$_v$].Coste de longitud de los caminos. ➢ *Cp*: matriz [*K*N*$_v$].Coste de peligro de los caminos.
SALIDAS	✓ Si *ac='vl'*: el fichero *uavrx3.dat* contiene: ➢ *vl*: vector [N_v].Velocidades lineales de los N_v UAVs. ➢ *nr*: vector [N_v].Número de trayectoria para los UAVs. ✓ Si *ac='ti'*, el fichero *uavrx3.dat* contiene: ➢ *TIG*: Tiempo de interceptación global. ➢ Indicador de sincronización (local).

Tabla 22: Resumen de la función *iscen.m*.

4.2.3. Método de velocidades lineales: la función isvl.m.

Es la versión más centralizada del algoritmo de cooperación. Se escoge la opción *ac='ev'*, que no genera computación en los UAVs. Desde el fichero de control y configuración del módulo se comunica a la función *isvl.m* las siguientes variables de coordinación que se evalúan en la función de coordinación.

- ✓ Costes de longitud de las rutas de las los diferentes UAVs (L_c*: [K*N_v]*).
- ✓ Costes de exposición al peligro de las trayectorias (C_c*: [K*N_v]*).

Una vez efectuada la adjudicación de vehículos a objetivos, *iscen.c* recibe además del vector de asignación (*a* vector de tamaño [N_v] que indica el objetivo de cada UAV), las longitudes y costes de las *K* trayectorias que unen cada pareja UAV-objetivo. Se llama a *isvl.c* cuyo cometido es seleccionar la combinación de rutas para todos los UAVs de entre las K posibles trayectorias que unen cada pareja vehículo – objetivo. Se tiene que garantizar la sincronización y el menor coste conjunto.

Las salidas que se transmiten a los UAVs, si la configuración del módulo así lo estima, son las proporcionadas por esta función *isvl.m*. Se trata de los vectores de velocidades lineales y número de ruta de cada UAV (v_l *,* n_r*:* vectores de tamaño *[*N_v*]*). La transmisión de los mismos se realiza vía Director de Comunicaciones (gracias al fichero *uavrx3.dat*).

Se ha detallado el Método de velocidades lineales en el apartado 4.1.1, cuyo objetivo es la determinación del Tiempo de Interceptación al Objetivo que respete el rango de validez de las velocidades lineales de los vehículos aéreos no tripulados procurando que la combinación de trayectorias tenga mínimo coste de intercepción global de los costes propios.

El procedimiento comienza elaborando una matriz cuadrada V_e de tamaño N_v**K*, en la que en cada fila domina una de las trayectorias. Los N_v**K* elementos de esa fila corresponden a su vez con cada una de las trayectorias de los N_v vehículos, de manera que se escalan estos elementos respecto al elemento diagonal de dicha fila y se multiplica el resultado por la velocidad V_{mx}. Es obvio que la diagonal de dicha matriz está compuesta por elementos de valor V_{mx}.

V_e, muestra las velocidades de los UAVs cuando se escoge la trayectoria que indique la posición en la matriz (de entre las *K* posibles para cada vehículo) y tomando como referencia la trayectoria correspondiente al elemento diagonal a la hora de escalar el resto de la fila. Se descartarán aquellas velocidades que sean superiores a la velocidad máxima. Las que sean inferiores al límite marcado por la velocidad mínima se escogerán sólo si no hay posibilidad de dar con una combinación de trayectorias que garantice que todas las velocidades cumplan el rango de velocidad especificado para los UAVs [V_{mn} , V_{mx}].

$$V_e = V_{mx} \cdot \begin{bmatrix} L_{11}/L_{11} & L_{12}/L_{11} & \cdots & L_{1K}/L_{11} & \cdots & L_{K1}/L_{11} & L_{K2}/L_{11} & \cdots & L_{KK}/L_{11} \\ L_{11}/L_{12} & L_{12}/L_{12} & \cdots & L_{1K}/L_{12} & \cdots & L_{K1}/L_{12} & L_{K2}/L_{12} & \cdots & L_{KK}/L_{12} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ L_{11}/L_{1K} & L_{12}/L_{1K} & \cdots & L_{1K}/L_{1K} & \cdots & L_{K1}/L_{1K} & L_{K2}/L_{1K} & \cdots & L_{KK}/L_{1K} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ L_{11}/L_{K1} & L_{12}/L_{K1} & \cdots & L_{1K}/L_{K1} & \cdots & L_{K1}/L_{K1} & L_{K2}/L_{K1} & \cdots & L_{KK}/L_{K1} \\ L_{11}/L_{K2} & L_{12}/L_{K2} & \cdots & L_{1K}/L_{K2} & \cdots & L_{K1}/L_{K2} & L_{K2}/L_{K2} & \cdots & L_{KK}/L_{K2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ L_{11}/L_{KK} & L_{12}/L_{KK} & \cdots & L_{1K}/L_{KK} & \cdots & L_{K1}/L_{KK} & L_{K2}/L_{KK} & \cdots & L_{KK}/L_{KK} \end{bmatrix}$$

La coordinación temporal tiene por objeto conseguir que el conjunto de trayectorias seleccionadas sea el de menor coste de intercepción global, aunque para algún UAV el coste de la ruta no sea el menor de los posibles. Se escogerá para un vehículo una ruta de coste mayor si garantiza la sincronización temporal del conjunto de rutas. Se definirá una función de coordinación que seleccione las trayectorias, teniendo en consideración el coste de interceptación de las mismas. La función objetivo será la suma de los costes de interceptación de las rutas de diferentes UAVs.

Los costes de intercepción se registran en una matriz cuadrada de tamaño N_v*K de nombre *CI*. Cada elemento de la matriz corresponde al coste de interceptación de la trayectoria correspondiente en el contexto de una determinada combinación de rutas. Los elementos de esta matriz no sólo se ordenan de la misma forma que V_e, sino que además dependen de la velocidad escalada. Por filas se ha asegurado la simultaneidad de las llegadas, de manera que en función del resultado obtenido en la matriz de velocidades se construirá la de costes de interceptación:

El coste de interceptación de la trayectoria *k* del UAV *i* en el contexto *c* (fila) que determina una posible combinación de rutas viene dado por:

- Si la velocidad (elemento en la misma posición de la matriz V_e) está dentro del rango [V_{mn} , V_{mx}], el coste de intercepción $CI_{i,k}^{c}$ que coincide con el coste de exposición al peligro de la trayectoria:

$$CI_{i,k}^{c} = C_{i,k}$$

- Si la velocidad no alcanza el valor V_{mn}, el coste de intercepción se ajusta para penalizar esta circunstancia frente a la anterior. Se suma al coste de peligro de la ruta el máximo de los costes registrados para todas ellas. Así se asegura la no elección de rutas en el contexto actual frente a otras que cumplan el rango de validez de velocidades. Además se suma la diferencia de velocidad con V_{mn}.. El coste de intercepción de la ruta $CI_{i,k}^{c}$ resulta:

$$CI_{i,k}^{c} = C_{i,k} + mx_{i=1\ldots Nv,k=1\ldots K}\left\{C_{i,k}\right\} + \left(V_{i.k}^{c} - V_{mn}\right)$$

- Si la velocidad es superior a V_{mx}, hay que corregir el coste para evitar que sea escogida una combinación de rutas con velocidades que infrinjan V_{mx}. La penalización será la mayor para asegurar la no elección:

$$CI_{i,k}^{c} = C_{i,k} + K \cdot mx_{i=1\ldots Nv,k=1\ldots K}\left\{C_{i,k}\right\} + \left(V_{mx} - V_{i.k}^{c}\right)$$

La matriz que almacena los costes de intercepción de las rutas en los diferentes contextos tiene la siguiente estructura. Almacena el valor de la función de coordinación de todas las trayectorias en los diferentes contextos. Se escogerá aquel que minimice los costes de interceptación de las rutas seleccionadas para cada UAV:

$$CI = \begin{bmatrix}
CI_{11}^{11} & CI_{11}^{11} & \cdots & CI_{11}^{11} & \cdots & CI_{K1}^{11} & CI_{K2}^{11} & \cdots & CI_{KK}^{11} \\
CI_{11}^{12} & CI_{12}^{12} & \cdots & CI_{1K}^{12} & \cdots & CI_{K1}^{12} & CI_{K2}^{12} & \cdots & CI_{KK}^{12} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
CI_{11}^{1K} & CI_{12}^{1K} & \cdots & CI_{1K}^{1K} & \cdots & CI_{K1}^{1K} & CI_{K2}^{1K} & \cdots & CI_{KK}^{1K} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
CI_{11}^{K1} & CI_{12}^{K1} & \cdots & CI_{1K}^{K1} & \cdots & CI_{K1}^{K1} & CI_{K2}^{K1} & \cdots & CI_{KK}^{K1} \\
CI_{11}^{K2} & CI_{12}^{K2} & \cdots & CI_{1K}^{K2} & \cdots & CI_{K1}^{K2} & CI_{K2}^{K2} & \cdots & CI_{KK}^{K2} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
CI_{11}^{KK} & CI_{12}^{KK} & \cdots & CI_{1K}^{KK} & \cdots & CI_{K1}^{KK} & CI_{K2}^{KK} & \cdots & CI_{KK}^{KK}
\end{bmatrix}$$

La simultaneidad se asegura en cada fila, pero queda por garantizar el mínimo coste global del conjunto. La selección de trayectorias se lleva a cabo en función de los distintos valores que toma la función de coordinación en la matriz *CI*.

Por filas, de las *K* rutas de cada UAV se selecciona aquella que haga mínimo el coste de interceptación en el contexto de la fila. Al aplicar esta norma a los N_v vehículos aéreos no tripulados se obtiene una matriz con los costes de interceptación mínimos para cada UAV en los diferentes contextos. La matriz *mCI* almacena estos costes y la matriz *iCI* que registra la posición de estos mínimos en la matriz *CI*. Para una fila (contexto *c*), el mínimo coste de intercepción del UAV *i* se calcula:

$$mCI_i^c = mn_{k=1\ldots K}\left\{CI_{i,k}^c\right\}$$

A partir de la matriz *mCI* de tamaño [N_v.K*N_v] se tiene para cada contexto *c* (fila) los costes de intercepción mínimos para cada UAV (corresponderán a situaciones de velocidad en el rango, siempre que exista esta posibilidad en el contexto). Si se suman las columnas de cada fila de la matriz *mCI* se obtiene los costes globales de intercepción de los N_v UAVs para cada uno de los N_v.K contextos.

Se trata de la función objetivo que decide la ruta de cada UAV. El vector *FO* de tamaño N_v.K contiene el coste global de interceptación de cada contexto (suma de los costes de interceptación de la ruta seleccionada para cada UAV). La solución al problema de coordinación temporal se consigue minimizando dicha función objetivo, es decir, determinando el mínimo elemento de *FO* que corresponderá a la fila del contexto más favorable.

$$FO_c = \sum_{i=1\ldots Nv}\left\{mCI_i^c\right\}$$
$$c \leftarrow mn_{c=1\ldots NvK}\left\{FO_c\right\}$$

Una vez determinada la fila *c* se conoce el contexto de interceptación de mínimo coste global. La fila *c* de la matriz *iCI* contiene la posición en las tablas *CI* y V_e de las rutas más favorables para los UAVs. Al acudir a las matrices, se rescata la velocidad lineal calculada para el UAV y el numero de trayectoria para cada vehículo que minimiza la función de coordinación temporal.

Esta información se comunicará a los UAVs (*iscen.m* escribe en el fichero *uavrx3.dat*), para que éstos pongan en marcha el Generador de Trayectorias para parametrizar en el tiempo la ruta seleccionada a a velocidad lineal media calculada.

- ✓ vector *vl* con N_v elementos.

- ✓ vector *nr* de N_v elementos de valor comprendido entre 1 y *K*.

Se muestra la tabla resumen de la función *isvl.c* que implementa el algoritmo de interceptación según el Método de velocidades lineales. Se recuerda que está implementada en el módulo centralizado y que es invocada desde la función *iscen.c*, encargada de la configuración y el control del Interceptor Simultáneo. La tabla contiene las entradas y salidas, tras haber repasado los objetivos del fichero:

OBJETIVOS	✓	Sincronización de las llegadas de los UAVs a sus objetivos siguiendo el método de velocidades lineales. ➢ Comprobación de la velocidad para asegurar que se encuentre en el rango [V_{mn} , V_{mx}]. ➢ Procurar el mínimo coste de interceptación global del conjunto.
	✓	Si no se consigue la sincronización global se puede optar por: ➢ Pérdida de sincronía para el UAV que la impide (*op=0*) ➢ Simultaneidad local para cada objetivo (*op=1*). ➢ Probar otra asignación (*op=2*). ➢ Asignar V_{mx} , factor sorpresa sin coordinación temporal (*op=3*)
ENTRADAS	✓	Procedentes del sistema central (configuración *iscen.c*). ➢ V_{mx}: velocidad máxima permitida. ➢ V_{mn}: velocidad mínima permitida.
	✓	Procedentes de los sistemas UAVs (recibidas por *dccen.c*): ➢ *Lr*: longitud de las rutas de los UAVs. Matriz [$K*N_v$]. ➢ *Cr*: costes de las rutas (longitud y exposición al peligro de sus segmentos de Voronoi) de los UAVs. Matriz [$K*N_v$].
SALIDAS	✓	Hacia la función i*scen.c*, más tarde enviadas a los UAVs: ➢ *vl*: vector [N_v].Velocidades lineales de los N_v UAVs. ➢ *nr*: vector [N_v].Número de trayectoria para los UAVs.

Tabla 23: Resumen de la función *isvl.c*

4.2.4. Método de tiempos de interceptación: la función isti.m.

Es la segunda versión del algoritmo de cooperación. Todos los aspectos teóricos han sido detallados a lo largo del apartado 4.1.2. y se recuerda que la filosofía que lo ha inspirado es el reparto de la carga computacional entre los UAVs y la parte centralizada del módulo. Es obvio que la decisión se toma en dicha parte central, pues habrán de considerarse las circunstancias de todos los UAVs.

El control general del módulo corresponde a la función *iscen.m*, fichero en el que se trata de forma diferenciada esta opción frente al escalado de velocidades lineales. En el caso de escoger este método, además del control del módulo, el Interceptor Simultáneo contará de los dos ficheros siguientes La función *isti.m* que se ejecuta en el sistema central y codifica el *paso 2* del algoritmo de cooperación y la función isuav.c que codifica los *pasos 1 y 3*.

En los UAVs se ejecuta el *paso 1*. La función isuav.m determina los costes de interceptación de las *K* rutas evaluados en los instantes de interceptación inferiores. A través del Director de Comunicaciones se transmite dichos instantes (*TIO1*), los extremos temporales superiores de cada ruta (*TIO2*) y los costes de interceptación de las trayectorias. Todos estos datos son variables de coordinación que en el sistema central, en el *paso 2*, sirven para evaluar la función de coordinación que determina el tiempo de Interceptación Global que asegure el mínimo coste de interceptación global.

En resumen, éstas son las etapas que determina las trayectorias de los UAVs y sus velocidades que garantizan la coordinación temporal de los UAVs:

1. Paso 1: en los UAVs se determina los tiempos de interceptación al objetivo (en función de V_{mn} y V_{mx} de cada UAV y las longitudes). Se comunica el Coste de Interceptación evaluados en los instantes de interceptación al objetivo inferior.

2. Paso 2: evaluación de la función de coordinación. La función *isti.m* genera el tiempo de interceptación global que permita el mínimo coste conjunto.

3. Paso3: Determinación de la velocidad y el número de ruta de cada UAV. Se calcula la función *isuav.m*.

A partir de los parámetros de los diferentes UAVs recibidos a través del Director de Comunicaciones se calcula los costes de interceptación que experimentan cada una de las rutas en un tiempo de interceptación al objetivo. Se recopila la información procedente de los N_V UAVs, que es la entrada de la función *isti.m*.

- ✓ *TIO1*: extremos inferiores de los intervalos de tiempos de interceptación al objetivo para las diferentes rutas (determinados por la velocidad máxima de recorrido). Tras haber sido recibida la información desde los UAVs se ha almacenado la información en un vector fila de N_v. K elementos (N_v bloques de K elementos).

- ✓ *TIO2*: extremos temporales superior para los tiempos de interceptación al objetivo. Todas las rutas de los N_v vehículos aéreos no tripulados se han considerado recorridas a la velocidad mínima. Resulta este vector de tiempos con $N_v.K$ elementos.

- ✓ C_r: Vector de $N_v.K$ elementos que contiene el coste de interceptación de las rutas en los instantes TIO1 (suma de los costes de exposición al peligro de los lados de Voronoi). A partir de estos, y teniendo en cuenta el factor temporal se calcularán los costes de interceptación que decidirán el tiempo global de interceptación.

Como se vio en el apartado *4.1.2*, los costes de interceptación dependerán del coste de las rutas más dos factores para los tiempos que estén fuera de los rangos temporales propios. El primer sumando adicional era el máximo de los costes registrados, el segundo la diferencia respecto al mínimo temporal. Esto asegura que se seleccione uno de los mínimos temporales, es por ello por lo que se va a medir el coste de intercepción sólo en esos instantes de tiempo almacenados *TIO1*.

Se evalúa para cada una de las K rutas de los N_v UAVs el Coste de Intercepción al objetivo evaluado en un tiempo de interceptación al objetivo *tio*. La determinación del coste sigue la siguiente norma en función del tiempo de interceptación al objetivo *tio*. Se determina el Coste de Interceptación de las trayectorias en todos los instantes *TIO1*. Estos costes sirven para evaluar la función de coordinación temporal en las diferentes rutas de los UAVs. Se tratará de hacer mínima la suma de estos costes asignados a rutas para los distintos vehículos.

Se construye la matriz cuadrada *CI* de tamaño N_v*K en la que cada columna representa un tiempo de interceptación determinado (tiempos mínimos de los intervalos de las K rutas de los N_v UAVs) mientras que en las diferentes filas de esa columna se representa el coste de interceptación de cada ruta evaluado en el tiempo de interceptación al objetivo *tio*. El coste de interceptación de la ruta k del UAV i para un tiempo de interceptación al objetivo *tio* se calcula así:

- Si *tio* esté dentro del rango temporal de la ruta [L_{ik}/V_{mx} , L_{ik}/V_{mn}], el coste de interceptación es el coste de exposición al peligro de la ruta calculado por el UAV correspondiente:

$$CI_{i,k}^{tio} = C_{i,k}$$

- Si *tio* supera el máximo instante de interceptación permitido (L_{ik}/V_{mn}) al UAV i para alcanzar su objetivo a través de la ruta k, entonces el coste de interceptación se incrementará la cantidad correspondiente al máximo de los costes de las rutas, más la diferencia entre *tio* y el tiempo máximo:

$$CI_{i,k}^{tio} = C_{i,k} + mx_{i=1\ldots Nv,k=1\ldots K}\left\{C_{i,k}\right\} + \left(\frac{L_{ik}}{V_{mn}} - tio\right)$$

- Si *tio* es inferior al tiempo mínimo permitido al UAV i para recorrer la trayectoria k, se forzaría al vehículo a recorrer la ruta a una velocidad superior a la máxima permitida. Este caso constituye el más desfavorable de todos, y el coste de interceptación se elevará para evitar que sea seleccionado para esa ruta:

$$CI_{i,k}^{tio} = C_{i,k} + K \cdot mx_{i=1\ldots Nv,k=1\ldots K}\left\{C_{i,k}\right\} + \left(\frac{L_{ik}}{V_{mx}} - tio\right)$$

Para cada fila (un determinado tiempo de intercepción al objetivo) hay que seleccionar la ruta entre las K trayectorias de cada UAV que hace mínimo el coste de intercepción al objetivo. Resulta una matriz que se denomina *mCI*, a partir de la cual se elabora la función objetivo con la que se decidirá el tiempo de interceptación global. La matriz *mCI* dispone de N_v*K columnas (coste de intercepción de cada ruta para *tio*) y N_v filas correspondientes a los diferentes tiempo de intercepción, extremos inferiores:

$$mCI_{i}^{tio} = mn_{k=1\ldots K}\left\{CI_{i,k}^{tio}\right\}$$

La función de coordinación temporal es la suma para cada tiempo de interceptación al objetivo de los costes de intercepción de los distintos UAVs. Se obtiene una vector de N_v*K filas.

$$FO_{tio} = \sum_{i=1...Nv} \left\{ mCI_i^{tio} \right\}$$

La minimización de la función de coordinación permite escoger la fila correspondiente al tiempo de interceptación al objetivo que ha hecho mínima la suma de los costes de interceptación mínimos de cada UAV. A partir de aquí se selecciona en el vector *TIO1* el instante de tiempo que hace mínima la función objetivo. El proceso se puede expresar matemáticamente así:

$$tio \leftarrow mn_{tio=1...NvK} \left\{ FO_{tio} \right\}$$

Se rescata el Tiempo de Interceptación global de entre los posibles valores de tiempos de interceptación al objetivo del vector *TIO1* (instantes más pequeños para agilizar la misión y fomentar el 'factor sorpresa'). Se comunicará a los UAVs este Tiempo de interceptación Global '*TIG*'.

$$TIG = TIO_1(tio)$$

En el siguiente apartado se detalla el uso que hace cada UAV del *TIG* para determinar la velocidad lineal y el número de ruta, que se recuerda es el objetivo fundamental del Interceptor Simultáneo (en el fichero *isuav.m*).

El uso que se haga de la función *isti.m* depende del resultado obtenido y de la configuración realizada en el fichero *iscen.m*:

- ✓ Sincronización global: al comienzo de la operación del módulo. En caso de imposibilidad de coordinación temporal, si se selecciona *op=2* se produce de nuevo la llamada a esta función después de seleccionar otra asignación.

- ✓ Sincronización local: ante la imposibilidad de sincronización global con la asignación actual, si *op=1* se opta por sincronizar localmente las llegadas de los UAVs involucrados en cada objetivo.

Este es el cuadro resumen de la función *isti.c*, en el que se muestran los objetivos de la función, así como las entradas que requiere el fichero para proporcionar las salidas esperadas:

OBJETIVOS	✓	Sincronización global de los UAVs a sus objetivos siguiendo el 'método de tiempos de interceptación'. ➢ Comprobación del tiempo de interceptación al objetivo para asegurar que se encuentre en el rango [L/V_{mx} , L/V_{mn}]. ➢ Procurar el mínimo coste de interceptación global. Es la función de coordinación temporal.
	✓	Sincronización local para cada objetivo (*op=2*).si falla la anterior.
ENTRADAS	✓	Procedentes de los diferentes UAVs: ➢ *TIO1*: instante temporal inferior del rango de tiempo permitido a las rutas de los UAVs (L_{ik} / V_{mx}). Vector [*K*Nv*]. ➢ *TIO2*: instante temporal superior del rango de tiempo para las rutas de los UAVs (L_{ik} /V_{mn}). Vector de [*K*Nv*] elementos. ➢ *Cr*: costes de exposición al peligro de los segmentos de Voronoi) de los UAVs. Vector de [*K*Nv*] elementos.
	✓	Procedente del sistema central: ➢ V_{mn}: velocidad mínima permitida a los UAVs. ➢ V_{mx}: velocidad máxima permitida a los UAVs.
SALIDAS	✓	*TIG:* Tiempo de interceptación Global. Es el tiempo de interceptación al objetivo que minimiza la función de coordinación temporal (suma de los costes de intercepción al objetivo).
	✓	*Indicador local*: en caso de sincronización local.

Tabla 24: Resumen de *isti.m.*

4.2.5. La operación en el UAV: la función isuav.m.

La función *isuav.c* es la función encargada de realizar la tarea de interceptación simultánea encomendada al UAV. Dependiendo del algoritmo de cooperación utilizado en el sistema central (funciones *isvl.m* ó *isti.m*) se realizarán operaciones distintas.

En el caso del escalado de velocidades lineales, lo único que se realiza es la lectura de las salidas entregadas (fichero *uavrx3.dat*), que corresponden con la velocidad lineal y el número de ruta respectivamente.

En el caso del Método de tiempos de interceptación se tienen dos fases bien diferenciadas correspondientes a los pasos 1 y 3 del algoritmo de cooperación:

- Paso 1: Intervalos de tiempo para cada una de las *K* rutas que el UAV puede emplear para poder interceptar el objetivo a una velocidad permitida [V_{mn} , V_{mx}]. (sólo en el método de los tiempos de interceptación).

- Paso 3: a partir del *TIG* entregado desde el sistema central se determina el número de ruta y la velocidad lineal de recorrido.

El fichero *ris.dat* almacena el resultado del Interceptor Simultáneo que será utilizado más tarde por el Generador de Trayectoria. Éste accederá al sistema de ficheros para recopilar la trayectoria que tiene que parametrizar en el tiempo.

- ✓ Velocidad lineal de recorrido.

- ✓ Número de ruta (*1...K*).

Se resumen las características de la función que realiza la actividad en el UAV.

OBJETIVOS	✓ Determinación de la velocidad lineal y número de ruta (paso3) . ✓ Cálculo de los intervalos temporales del UAV (paso1).
ENTRADAS	✓ Procedentes de los diferentes UAVs: ➢ *Lr*: costes de longitud de las rutas del UAVs. Vector [*K*Nv*]. ➢ *Cr*: costes de exposición al peligro de los segmentos de Voronoi) de los UAVs. Vector de [*K*Nv*] elementos. ✓ Procedente del sistema central: ➢ Algoritmo de cooperación (‘vl’ ó ‘ti’). ✓ *TIG:* Tiempo de interceptación Global. Es el tiempo de interceptación al objetivo que minimiza la función de coordinación temporal (suma de los costes de intercepción al objetivo).
SALIDAS	✓ Fichero *ris.dat:* ➢ *vl*: velocidad lineal para el UAV. ➢ *nr*: número de trayectoria (*1...K*).

Tabla 25: Resumen de la función *isuav.m*.

Capítulo 5: Generación de la trayectoria.

Una vez determinada para cada UAV una trayectoria que le permita llegar al objetivo de forma coordinada con el resto de vehículos en un tiempo de interceptación global, desde cada vehículo aéreo no tripulado se emprenderán las tareas que le permitan cumplir con la misión asignada. A partir de la sucesión de segmentos que determina la ruta se adecuará a las restricciones de movimiento del vehículo para que pueda ser recorrida. La trayectoria de referencia compatible con la cinemática del UAV será recorrida accionando las señales de control que permitan al vehículo aéreo el seguimiento de la trayectoria de forma real.

El Generador de Trayectoria es el módulo del sistema encargado de la generación de trayectorias. Cada UAV cuenta con la implementación de diferentes funciones que le permite determinar la trayectoria de referencia que más tarde será seguida accionando las señales de control. La acción se descompone en dos fases:

1. Adecuación de la trayectoria: a partir de la sucesión de lados del diagrama de Voronoi se entrega una trayectoria compuesta por porciones de los segmentos unidos por sectores de circunferencia. De esta manera se resuelve la incapacidad que presenta el vehículo aéreo a variaciones abruptas del ángulo de recorrido. En las intersecciones de dos segmentos se producen variaciones de este tipo, que se suavizan gracias a la inserción en el ángulo de sectores de circunferencia que permitan transitar de manera continua y suave.

2. Parametrización en el tiempo de la trayectoria adecuada a las características de movimiento del UAV. Se ha determinado para el UAV un tiempo de interceptación al objetivo resultado de la coordinación de los múltiples UAVs. La trayectoria de referencia se debe recorrer en dicho tiempo a la velocidad media que ha resuelto el Interceptor simultáneo. Se entrega al Controlador la trayectoria de referencia parametrizada en el tiempo.

5.1. La generación de una trayectoria real.

En una sucesión de aristas la imposibilidad de recorrerla la encuentra el UAV en las uniones de los segmentos. Estos vértices son puntos críticos ya que la continuidad de la trayectoria encuentra en ellos variaciones muy bruscas en el ángulo de dirección. Se tiene que suavizar el paso por el vértice de manera que la unión de todos los segmentos constituya una trayectoria factible para el vehículo aéreo.

El Generador de Trayectoria se encarga de adecuar la transición entre dos segmentos suavizando los picos que presenta la trayectoria en la unión de dos segmentos por un arco de circunferencia que partiendo del primero desemboque en el segundo asegurando que la que la variación del ángulo no sea abrupta, sino gradual. La trayectoria tendrá carácter continuo y será diferenciable. El Controlador del sistema, además de la trayectoria de referencia, emplea la derivada en el tiempo de las componentes (X_R, Y_R) determinadas por el módulo.

El Generador de Trayectoria determina la trayectoria de referencia a partir de las coordenadas $(X_R\ ,\ Y_R)$ correspondientes a los puntos que la describen en el tiempo. El procedimiento tiene que decidir en cada momento qué tipo de movimiento describe el UAV en cada momento. Estas son las opciones:

- ✓ Movimiento lineal: a través del segmento hasta el momento de emprender el movimiento circular hacia el siguiente segmento.

- ✓ Movimiento circular: durante el tiempo que permita describir el sector de circunferencia que permita al UAV seguir la dirección del siguiente segmento.

La parametrización en el tiempo de la trayectoria tiene que tener en cuenta los dos movimientos reseñados. Una vez que la sucesión de segmentos de Voronoi se ha adecuado a las características dinámicas del UAV se busca la forma de indicar al vehículo la posición que tiene que tener en cada instante de tiempo entre su partida y el tiempo de interceptación al objetivo.

Este aspecto se resuelve gracias a la implementación de un filtro no lineal que emule su comportamiento dinámico del vehículo aéreo no tripulado. Se analizan las ecuaciones que del funcionamiento del UAV.

5.1.1. Un filtro de primer orden compatible con el UAV.

Como se ha indicado, la forma de generar una trayectoria parametrizada en el tiempo y compatible con el UAV se obtiene recurriendo a las ecuaciones que describen su movimiento. De esta manera se asegura la compatibilidad de la trayectoria con la dinámica del vehículo aéreo no tripulado, descrita por las siguientes expresiones:

$$\begin{aligned}
\overset{\circ}{X}_i &= V_i \cdot \cos(\Psi_i) \\
\overset{\circ}{Y}_i &= V_i \cdot sen(\Psi_i) \\
\overset{\circ}{\Psi}_i &= \alpha_{\Psi i} \cdot \left(\Psi_{Ci} - \Psi_i\right) \\
\overset{\circ}{V}_i &= \alpha_{Vi} \cdot \left(V_{Ci} - V_i\right) \\
\overset{\circ}{H}_i &= -\alpha_{\overset{\circ}{Hi}} \cdot \overset{\circ}{H}_i + \alpha_{Hi} \cdot \left(H_{Ci} - H_i\right)
\end{aligned}$$

A partir de estas ecuaciones se diseña el filtro que sirva a los propósitos del módulo. La última ecuación hace referencia a la altura del UAV, variable contenida en el plano vertical y que no influye en absoluto en la generación de la trayectoria. Por eso se obvia el comportamiento del perfil de altura, y se simplifican las ecuaciones que modelan el comportamiento de las variables del movimiento del UAV en el tiempo.

La determinación de la trayectoria de referencia parte de las ecuaciones que se refiere a las componentes *(X,Y)* de los puntos que la formen. Por último se escoge la variación de ángulo que permita recorrer una porción de segmento (movimiento lineal) o un sector de circunferencia para transitar entre segmentos de Voronoi (movimiento circular). A la salida del filtro se obtienen las componentes $(X_R\ ,\ Y_R)$ de la trayectoria de referencia parametrizada en el tiempo y la fase que posee el UAV en cada instante.

$$\begin{aligned}
\overset{\circ}{X}_R &= V_R \cdot \cos(\Psi_R) \\
\overset{\circ}{Y}_R &= V_R \cdot sen(\Psi_R) \\
\overset{\circ}{\Psi}_R &= u_1 \\
\overset{\circ}{V}_R &= u_2 \\
\overset{\circ}{H}_R &= 0
\end{aligned}$$

De todas ellas es la correspondiente a la variación del ángulo de la dirección respecto al eje *x* la que permite el control del movimiento. Se limita la variación del ángulo de dirección respecto al eje *X* a un valor constante, que implica que la primera derivada del ángulo respecto del tiempo sea constante. Dependiendo del valor que tome u_1, se dará alguno de estos comportamientos:

- ✓ Recorrido lineal: $u_1 = 0$. No se producirá variación alguna en el ángulo de la dirección con el eje *X*. Éste será el valor que tome u_1 cuando se pretenda que el UAV describa la parte de la trayectoria correspondiente a una porción de segmento de Voronoi. Que el ángulo sea constante implica que las variaciones que se produzcan en las componentes X_R e Y_R también lo sean, garantizando estas expresiones que se describa un movimiento lineal.

$$\begin{aligned}
&\overset{\circ}{X}_R = V_R \cdot \cos(\Psi_R) \rightarrow X_R \approx a \cdot t + b \\
&\overset{\circ}{Y}_R = V_R \cdot sen(\Psi_R) \rightarrow Y_R \approx c \cdot t + d \\
&\overset{\circ}{\Psi}_R = 0 \rightarrow \Psi_R = CTE
\end{aligned}$$

- ✓ Recorrido circular: $u_1 = \pm c$. Se permite al UAV variar de forma constante con el tiempo el ángulo de la dirección con respecto a *X*. El comportamiento del ángulo es lineal respecto al tiempo, hecho que al ser incorporado a las ecuaciones de las componentes de la trayectoria de referencia, produce que el movimiento del UAV sea circular en el sentido que indique el signo que afecta a u_1. El movimiento que describe el UAV es circular, ya que la suma de los cuadrados de las componentes X_R, e Y_R normalizadas es constante. Un aspecto significativo es el sentido de recorrido del vehículo cuando describa el movimiento circular que propicia este caso. La información de este aspecto se guarda en el signo de u_1.

$$\begin{aligned}
&\overset{\circ}{X}_R = V_R \cdot \cos(\Psi_R) \rightarrow X_R \approx \frac{V_R}{c} \cdot sen(c \cdot t) \\
&\overset{\circ}{Y}_R = V_R \cdot sen(\Psi_R) \rightarrow Y_R \approx -\frac{V_R}{c} \cdot \cos(c \cdot t) \\
&\overset{\circ}{\Psi}_R = \pm c \rightarrow \Psi_R = \pm c \cdot t + cte \approx \pm c \cdot t
\end{aligned}$$

En el caso de recorrido circular se pueden dar dos posibilidades de giro. El parámetro *u1* permite controlar el sentido de giro del UAV cuando tenga que transitar desde un segmento al siguiente. El giro lo discrimina el signo de *u1*.

- $u_1 > 0$. El sentido de giro es el contrario a las agujas del reloj o a izquierdas.

- $u_1 < 0$. El sentido de giro se produce a derechas o en el sentido favorable de las agujas del reloj.

En el Generador de Trayectoria no influye el perfil de altura que en cada momento presente el UAV, ya que se ha desacoplado el comportamiento en los planos vertical y horizontal. Por ello no se analizan en este módulo las variaciones de la altura. Tampoco se introduce la determinación de la velocidad, ya que el objetivo de este módulo es proporcionar trayectorias válidas para el vehículo, y será el controlador el que monitorice la velocidad que ha de llevar cada uno a lo largo de la trayectoria.

Por tanto en este filtro paso bajo que suaviza la trayectoria se considera la velocidad lineal media calculada por el Interceptor simultáneo y no se da lugar a variaciones de la velocidad respecto a la media.

Al igual que un filtro paso de baja, además de eliminar las componentes de frecuencia elevadas, suaviza las variaciones bruscas, este filtro diseñado para el Generador de Trayectorias hace lo mismo con las variaciones bruscas en la posición del vehículo, de manera que los picos que describían las transiciones entre segmentos (cambio en la dirección del vehículo) pasan a ser arcos de circunferencia (cambio de dirección gradual).

5.1.1.1. Breve análisis del movimiento circular.

El diseño del Generador de Trayectoria ha permitido que los vértices por los que no puede transitar el UAV se suavicen, convirtiendo la unión de dos segmentos en una transición gradual del ángulo de dirección, compatible con el movimiento del vehículo aéreo no tripulado.

Las especificaciones pueden ser variadas y centrarse en el hecho de pasar por el punto vértice, garantizar la longitud de las rutas o minimizar el tiempo de recorrido. De estas tres especificaciones se opta por la relativa al mínimo tiempo, ya que el 'factor sorpresa' que ha inspirado el diseño así lo recomienda. Para esta especificación en el Generador de Trayectorias se construyen circunferencias tangentes a los dos segmentos en el interior del ángulo (centro en la bisectriz).

Se determinará la distancia al vértice para encontrar el punto en el segmento a partir del cual se ordena al UAV describir la porción de circunferencia que le lleve a un punto del siguiente segmento situado a la misma distancia del vértice, y que le permita a partir de éste recorrer el nuevo lado de Voronoi gracias al ángulo de dirección alcanzado. El siguiente ejemplo ilustra esta situación:

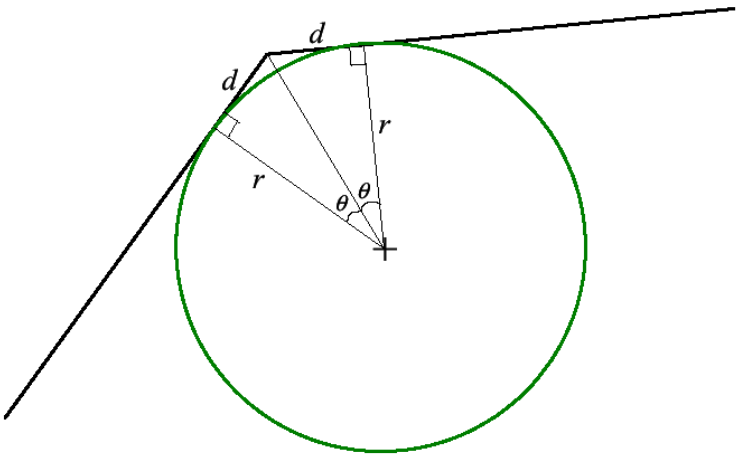

Figura 21: Transición suavizada entre segmentos (mínimo tiempo de recorrido).

El radio del arco de circunferencia depende de la máxima variación del ángulo ψ que se le permita soportar al UAV o que no pueda superar debido a las prestaciones del vehículo. El radio de la circunferencia tangente a los dos segmentos:

$$R = V_R / c\,.$$

Al sistema se le proporciona la máxima variación del ángulo permitida por el fabricante, de esta manera y gracias a la relación inversa del radio respecto a la variación máxima del ángulo se respeta en un mayor grado la trayectoria original compuesta por segmentos de Voronoi que aseguran la equidistancia de las amenazas más próximas. Los arcos de circunferencia se encontrarán lo más cerca posible de los vértices de los segmentos sin violar los principios de funcionamiento del UAV.

Se permite relajar esta restricción alejando la variación del ángulo del máximo permitido. De esta manera el radio de las circunferencias tangentes a cada pareja de lados consecutivos será mayor, y por tanto menor la porción de segmento finalmente recorrida. Por lo tanto existe una relación directa entre el radio de la circunferencia y la distancia del punto de intersección de la circunferencia con el segmento al vértice.

La distancia dependerá también de la diferencia entre los ángulos de dirección del UAV en el primer y segundo segmento. Como se muestra en la figura anterior, θ corresponde a la mitad de esa diferencia de ángulos. Del análisis trigonométrico del ángulo recto formado por los puntos vértice – centro de la circunferencia – intersección se obtiene la distancia al vértice del punto de intersección del segmento con la circunferencia *d*:

$$d = \frac{R}{tg(\theta)} = \frac{V_R \cdot tg(\theta)}{c} = \frac{V_R}{c \cdot tg\left(\left(\Psi_2 - \Psi_1\right)/2\right)} = \frac{V_R}{c \cdot tg(\alpha)}$$

El centro de la circunferencia depende también del sentido de giro, codificado en el signo del parámetro u_1. Si se tienen las coordenadas (X_R, Y_R) del punto de intersección del lado de Voronoi con la circunferencia, las coordenadas (X_C, Y_C) se calculan de la siguiente manera:

$$\begin{bmatrix} X_C \\ Y_C \end{bmatrix} = \begin{bmatrix} X_R \\ Y_R \end{bmatrix} \pm \begin{bmatrix} -V_R \cdot sen(\Psi_R)/c \\ V_R \cdot \cos(\Psi_R)/c \end{bmatrix}$$

De la misma manera que estos cálculos se han aplicado a la circunferencia inscrita y tangente a los dos segmentos, proporcionado las coordenadas de cualquier otro punto del primero o segundo segmento se obtiene una circunferencia a la derecha o izquierda, tangente al segmento que permite describir cualquier otro movimiento.

Este aspecto será muy importante en la resolución de comportamientos anómalos, como el caso en que no sea posible inscribir una circunferencia entre los dos segmentos o más concretamente en la porción de segmento libre tras haber sido trazada alguna.

5.1.2. El control del proceso de generación de trayectoria.

Se ha diseñado un filtro no lineal capaz de describir una trayectoria adecuada para las especificaciones técnicas de los vehículos. Cuando se necesite transitar a lo largo de un segmento, fijando a cero la variación del ángulo, el vehículo recorrerá la dirección marcada por el segmento.

Si por el contrario se tiene que transitar hacia otro segmento, permitiendo la variación gradual en el ángulo se logra describir la circunferencia que transfiere hacia la dirección indicada por el segundo segmento. Dependiendo del sentido del giro, bien sea a derecha o a izquierda, se fijará el signo que afecta a la variación del ángulo.

El Algoritmo de Control del Generador de Trayectoria se puede sintetizar en tres pasos que se repetirán cada vez que se produzca un cambio en la dirección del vehículo, debido a la intersección de dos segmentos:

1. Para seguir la dirección marcada por un segmento, hacer *c=0*. Se tiene que monitorizar la distancia que quede hasta el punto en el que se inicie la transición suave hacia el nuevo segmento. Se traza la bisectriz al ángulo que forman las dos rectas, y se considera el ángulo α entre la bisectriz y el segmento. Al inscribir entre los dos segmentos una circunferencia de radio prefijado R, usando relaciones trigonométricas elementales se determina la distancia que existe entre el vértice y el punto de intersección de la circunferencia y el segmento: $d = R / tan(\alpha)$. Conociendo el punto inicio del arco de circunferencia el sistema puede monitorizar el instante en el que se llega a dicho punto.

2. Una vez llegado al punto en el que debe describirse la sección de arco de circunferencia necesario para alcanzar el segmento posterior, se debe variar c de cero al valor de la máxima variación de ángulo permitido. El signo dependerá del sentido de giro. Cuando lo haga a derechas será positivo y por el contrario a izquierdas negativo.

3. Una vez recorrido el sector equivalente al ángulo 2α a través del arco de circunferencia se habrá desembocado en el segmento siguiente, a una distancia d del vértice, por lo que se dispondrá la variación cero del ángulo (*c=0*) y volverá al paso número 1.

Se resume el Algoritmo de Control de la Generación de Trayectorias en el siguiente cuadro resumen:

PASO 1 'RECORRIDO LINEAL A TRAVÉS DE UN SEGMENTO'($u_1 = 0$)

- ✓ Monitorizar la distancia hasta la intersección del segmento con la circunferencia.
- ✓ Mientras no se llegue a ese punto se mantiene a cero la variación del ángulo de dirección del UAV respecto al eje *X*.
- ✓ Al llegar al punto de intersección con la circunferencia, a una distancia *d* del vértice, se ejecuta el PASO 2.

PASO 2 'RECORRIDO CIRCULAR DE TRANSICIÓN' ($u_1 = \pm c$)

- ✓ Se determina el tiempo necesario para pasar del ángulo de dirección correspondiente al primer segmento al del segundo lado de Voronoi.
- ✓ Se comunica al filtro no lineal de primer orden la variación máxima del ángulo, de manera que se describe el movimiento circular. Dependiendo del signo, se ejecutará en un sentido u otro:
  - ➢ $u_1 > 0$: sentido antihorario (a izquierda).
  - ➢ $u_1 < 0$: sentido horario (a derecha).
- ✓ Al alcanzar la fase correspondiente al segundo segmento de Voronoi se ejecuta el PASO 3.

PASO 3 'RECORRIDO LINEAL DEL NUEVO SEGMENTO' ($u_1 = 0$)

- ✓ Se comunica al filtro no lineal este valor del parámetro u_1 que imposibilita el movimiento circular convirtiéndolo en lineal.
- ✓ Se vuelve al PASO 1.

Tabla 26: Algoritmo de Control de la Generación de Trayectorias.

El algoritmo anterior proporciona una transición entre segmentos en el menor tiempo posible. Si existiese la obligación de hacer pasar el vehículo aéreo por el vértice que une ambos segmentos el algoritmo sería similar pero pasando por tres arcos de circunferencia descritos en un sentido diferente (el central respecto a los extremos). Es una opción más compleja que además exige mayor recorrido al vehículo, por esta razón se escoge la opción del arco inscrito entre los segmentos que evita pasar por el vértice. También se descarta la que garantiza la igualdad en la longitud recorrida, pues el aspecto temporal es el crítico en el sistema automático de coordinación de múltiples UAVs.

A partir de este punto se aprecia la divergencia en algunos tramos de la trayectoria parametrizada en tiempo para el vehículo respecto a la trazada por el Planificador de Trayectorias. Es el precio que hay que pagar al pasar de una situación idealizada a estar sujeto a las restricciones de movimiento del vehículo que opera en el sistema.

Todos los cálculos desarrollados por los módulos encargados de planificar trayectorias, asignar objetivos y simultanear las llegadas se hicieron obviando las restricciones planteadas en este módulo. Dada la complejidad de esas acciones que se relacionan entre ellas y exigen el diálogo profundo entre los módulos que las llevan a cabo se ha hecho necesario acudir a estas aproximaciones. Por tanto, parámetros calculados como la longitud recorrida en cada trayectoria o la velocidad necesaria para lograr la simultaneidad no son del todo reales.

Es el Generador de Trayectorias el que genera en tiempo real la trayectoria que puede seguir el UAV sin violar las restricciones de movimiento impuestas a cada uno. A cambio de ello la longitud recorrida será inferior a la establecida por el Planificador de Trayectorias debido a que se ha evitado el paso por el vértice.

Este aspecto, que afecta a todos los vehículos del sistema puede empeorar la simultaneidad de las llegadas, aunque en un grado muy pequeño. Dependiendo de la aplicación a la que se destine el sistema y de la exigencia que presente puede ser conveniente la corrección de las velocidades lineales. Para ello se tendrá en cuenta la longitud de la ruta corregida.

5.1.3. Situaciones anómalas en la generación de trayectorias.

El mayor problema que puede surgir en la Generación de Trayectoria provoca la imposibilidad de suavizar la forma picuda que determina la intersección de dos lados de Voronoi, por la que el UAV no está en disposición de transitar. Este comportamiento anómalo se produce cuando las longitudes de los segmentos que intersectan en el vértice no son suficientemente largas para permitir una circunferencia inscrita. La experiencia demuestra que pueden darse situaciones en las que sea imposible suavizar la transición entre dos segmentos, análisis que se realiza a continuación. La experiencia demuestra que pueden darse este tipo de situaciones.

Las situaciones en que se presenta este comportamiento anómalo son variadas. Se enumeran algunas de ellas:

- ✓ Uno de los segmentos que intersectan en el vértice tiene una distancia inferior a la distancia necesaria entre el vértice y el comienzo de la circunferencia: $L_i<D$.

- ✓ Las dos longitudes de los segmentos que intersectan en el vértice son menores que la distancia necesaria: $L_1 , L_2 < D$.

- ✓ En el lado de partida no hay suficiente porción de segmento para inscribir la circunferencia hacia el siguiente, debido a que la transición anterior ha recortado al segmento de partida una distancia D', que provoca que la porción de segmento disponible sea inferior a D: $L - D' < D$.

Ante la presencia de alguna de estas situaciones existe la posibilidad de actuar de manera distinta, teniendo en cuenta que el sistema no sólo se compone del Generador de Trayectoria, sino que presenta diferentes módulos interactuando entre sí y con intereses distintos y contrapuestos en algún caso.

1. Tomar nota de la situación pero no emprender ninguna tarea correctora, primando los objetivos de otros módulos.

2. Emprender los cambios necesarios para conseguir una trayectoria adecuada, sin picos no transitables por el UAV, a costa de otros criterios.

5.1.3.1. Proposición de una trayectoria no adecuada para el UAV.

Se entiende llegado este punto que lo primordial es el mantenimiento de los criterios de decisión planteados en los diferentes módulos del sistema. Se devalúa frente a éstos el presente en el Generador de Trayectorias y que le obliga a entregar una trayectoria de referencia que pueda seguir el UAV sin violar las reglas de movimiento que le vienen impuestas. Por lo tanto se avisará de que existen puntos problemáticos que pueden dar lugar a la imposibilidad de recorrerla o a pequeños desvíos.

En este caso no se está dispuesto a asumir un desplazamiento respecto a los lados del diagrama de Voronoi superior al determinado en el Algoritmo de Control de la generación de trayectorias. Cualquier tarea correctora puede suponer una variación sustancial del diagrama de Voronoi y la trayectoria entregada por el Planificador de Trayectorias a partir de él. Se priman por tanto los criterios planteados en el Director de Asignación y Planificador de Trayectorias, que anteponen a cualquier otro el paso del UAV lo más lejos posible de los obstáculos que lo amenazan.

A consecuencia de esta manera de proceder, el Algoritmo de Control del Generador de Trayectoria queda reducido a los pasos 1 y 3 en las transiciones anómalas entre segmentos. De esta manera se obliga al UAV en la trayectoria de referencia continuar el recorrido lineal hasta el vértice problemático, y sin ningún tipo de variación gradual, cambiar el ángulo de dirección respecto al eje *X*, al valor que requiera el segundo segmento.

Si la variación de ángulo entre los segmentos es pequeña, es decir, existe un ángulo cercano a π entre los dos lados de Voronoi, el Controlador del Sistema corregirá el comportamiento. Si por el contrario, el ángulo entre ellos es muy pequeño, el carácter agudo del ángulo hará imposible la adecuación a la fase indicada por el segundo segmento. Si se dispone de esta variación máxima permisible, se puede avisar a priori al Controlador de este problema, asumiendo variaciones sustanciales a la llegada del UAV al objetivo.

5.1.3.2. Corrección de las situaciones anómalas planteadas.

Este comportamiento se produce en los segmentos con una longitud muy pequeña. Se propone modificar el recorrido del UAV evitando su paso por este segmento de Voronoi y transitando directamente hacia el siguiente. En este caso se está dispuesto a alejarse de la trayectoria decidida por los módulos que coordinan al sistema automático de múltiples UAVs.

Los otros esquemas que se suelen proponer en la generación de trayectorias, el que exige el paso por el vértice y el que garantiza el mantenimiento de la longitud de las rutas incrementan sustancialmente el riesgo de situaciones anómalas. Desde este punto de vista el esquema adoptado minimiza la aparición de estas circunstancias, que aún así pueden producirse.

5.2. El Generador de Trayectoria.

5.2.1. Descripción general.

Una vez que el Interceptor Simultáneo ha seleccionado entre las trayectorias planificadas el conjunto de ellas que serán recorridas por los vehículos aéreos no tripulados asegurando la simultaneidad de las llegadas, se debe comunicar a cada vehículo su trayectoria y la orden de recorrerlo.

El Generador de Trayectoria recibe desde el sistema central, gracias a la intermediación del Director de Comunicaciones, dos informaciones fundamentales que le permite conocer:

- ✓ La trayectoria seleccionada en la lista de *K* rutas que lo unen con el objetivo asignado y que guarda en su memoria.

- ✓ La velocidad lineal media que el Interceptor Simultáneo ha coordinado para conseguir la simultaneidad de todas las llegadas a los diferentes objetivos.

El UAV rescata de su memoria interna la sucesión de segmentos. El Generador de Trayectoria rescata la colección de segmentos de Voronoi correspondiente a la trayectoria seleccionada y tiene la obligación de hacer de ella una ruta que el vehículo aéreo no tripulado pueda recorrer desde su situación original hasta del destino sin violar las restricciones que impone la tecnología con que fue fabricado.

Se ofrecerá al vehículo una trayectoria lo suficientemente suavizada para ser recorrida sin violar las restricciones de movimiento que tienen presentan los vehículos y que determinan su dinámica interna. Son los vértices de los segmentos los que implican la mayor dificultad, pues representan variaciones muy abruptas en la trayectoria a las que el vehículo no es capaz de adecuarse. La tarea del Generador de Trayectoria consiste fundamentalmente en suavizar las transiciones entre diferentes segmentos. No olvidar que tiene que entregar al controlador del sistema la ruta parametrizada en el tiempo de manera que su seguimiento implique resolver la misión asignada en el tiempo de interceptación global.

5.2.2. Codificación del Generador de Trayectoria.

El módulo se ha codificado usando el lenguaje MATLAB, gracias a la sencillez con la que se tratan las estructuras matemáticas y la elevada capacidad de cálculo que presenta. Dispone de accesorios muy adecuados como Simulink, que ha facilitado la implementación del filtro no lineal, basado en el comportamiento dinámico del UAV, gracias al cual se genera la trayectoria de referencia parametrizada en el tiempo.

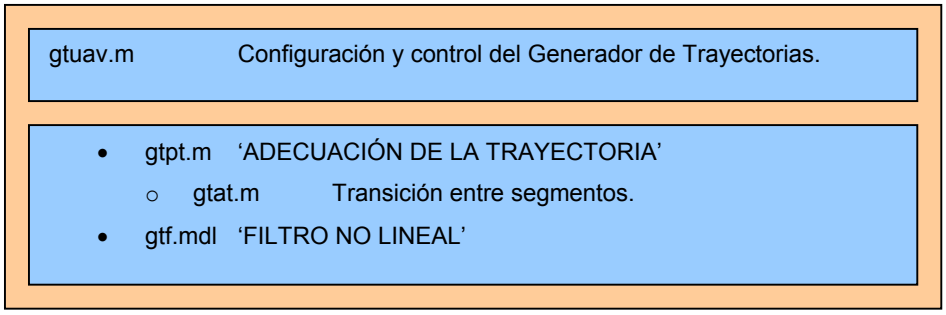

Tabla 27: Software del Generador de Trayectorias.

gtuav.m codifica el algoritmo de control y configuración del módulo, que incluye las ordenes y llamadas a otras funciones necesarias para generar la trayectoria seleccionada por el Interceptor Simultáneo conjuntamente con el Planificador y el Director de Asignación. En función de los parámetros que le cede la función *gtpt.m* encargada de la adecuación de la trayectoria, se comunica al filtro *gtf.mdl* el parámetro de giro *u1* y el tiempo de simulación del modelo.

gtat determina los ángulos entre segmentos, se trazan las bisectrices y se inscriben las circunferencias entre segmentos. De esta manera se conoce la distancia desde el vértice al punto de intersección entre circunferencia y segmento, el sentido de giro en cada transición, y el sector de circunferencia que tiene que describirse. Es el fichero que ofrece la información en función de la cual los otros ficheros generan la trayectoria de salida.

gtpt.m realiza todos los cálculos relacionados con la monitorización de la distancia que tiene que recorrer cada vehículo aéreo a lo largo de una recta antes de comenzar el recorrido a lo largo del sector de circunferencia. Determina los tramos que tienen que seguirse (lineales o circulares) y el tiempo asignado a cada uno de ellos. Lleva el control del giro (parámetro *u1*) que garantice la generación correcta.

5.2.2.1. El control del módulo: la función gtuav.m.

Al comienzo del fichero se introduce el máximo valor permitido para la variación del ángulo respecto al eje *X*. Este parámetro se consultará en las especificaciones técnicas del UAV y determinará la capacidad del vehículo para transitar de un segmento al siguiente a través del arco de circunferencia (variaciones grandes permiten arcos de circunferencia de radio menor, más ajustados al vértice).

Tras la configuración llega el control, se recibe desde la función *gtpt.m* tres variables a partir de las cuales se efectúa la simulación del filtro no lineal: el vector *g* (control del giro que toma el valor cero para recorrido lineal y *+1* o *–1* para recorridos circulares a izquierda y derecha respectivamente). El vector *tsm* contiene los tiempos de simulación para cada tramo y el vector *ps* la fase inicial de cada segmento (para corregir situaciones anómalas).

El número de elementos de los vectores *tsm* y *g* determina el número de tramos, que se simulan uno a uno, seleccionando el control de giro que multiplicado por *umx* determina el parámetro *u1* del filtro y el tiempo de simulación. Cuando se concluye se registran las componentes *x* e *y* de referencia junto a los instantes de muestreo efectuados por el filtro en el fichero *rgt.dat*.

OBJETIVOS	✓ Configuración del módulo: *umx* (máxima variación del ángulo). ✓ Algoritmo de Control de la Generación de Trayectorias.
ENTRADAS	✓ Procedentes del Planificador de Trayectoria:. *(rpt.dat*): ➢ *tvx*: vector [ns*2]. Coordenadas X de los extremos. ➢ *tvy*: vector [ns*2]. Coordenadas Y de los extremos. ➢ *lv* [ns*1]. Longitudes de los segmentos de la trayectoria. ✓ Procedentes del Interceptor Simultáneo (*ris.dat*): ➢ *vv*: Velocidad lineal de referencia calculada para el UAV.
SALIDAS	FICHERO *rgt.dat* ✓ *nm*: número de muestras de las que se dispone. ✓ X_R: Componente X de los puntos de la trayectoria de referencia. ✓ Y_R Componente Y de los puntos de la trayectoria de referencia. ✓ T_R Instantes de muestreo de la trayectoria entre 0 y *TIO*

Tabla 28: Esquema de la función *gtuav.m*

5.2.2.2. La monitorización del proceso: la función. gtpt m.

Se comunica desde *gtuav.m* la velocidad lineal (*vv*), las coordenadas x e y de los extremos de los segmentos ordenados (*tvx* y *tvy*) y la longitud de los segmentos (*lv*). Se pretende en la medida de lo posible que la función no tenga que realizar cálculos que evaluados en diferentes UAVs puedan exigir tiempos de cómputo elevados o muy diferentes. Estos parámetros se comunican a su vez a la función *gtat.m* que determina los datos relacionados con transiciones entre segmentos:

- *dg:* distancia a recorrer en cada uno de los segmentos.
- *sg*: sentido de giro: *0* (lineal), *+1,-1* (circular a izqda y drcha).
- *ag*: variación de ángulo en una transición (respecto a la dirección de partida).
- *ps*: ángulo de dirección de cada segmento respecto al eje *x*.

Los tres primeros son vectores, de manera que cada elemento corresponde a un tramo que puede ser circular o lineal. *gtpt.m* se encarga de evaluar las distancias *dg* para comprobar que no se producen situaciones anómalas (toda esta información se guarda en el vector *cg*). Además determina para cada caso el tiempo de recorrido del tramo en función de la longitud y la velocidad que le ha sido comunicada. La salida: corresponde al vector *g* con los giros permitidos *(+1,-1*) o el movimiento lineal (*0*) de cada tramo y *tsm* con los tiempos de simulación de los tramos (si es *cero* corresponde a una situación anómala y que el Generador no puede adecuar correctamente)

OBJETIVOS	✓	Monitorización del proceso de parametrización en el tiempo.
ENTRADAS	✓	*tvx*: vector [ns*2]. Coordenadas X de los extremos.
	✓	*tvy*: vector [ns*2]. Coordenadas Y de los extremos.
	✓	*lv* [ns*1]. Longitudes de los segmentos de la trayectoria.
	✓	*vv*: Velocidad lineal de referencia calculada para el UAV.
SALIDAS	✓	*g*: Control del giro para cada movimiento a generar:
		➢ *g=0*: movimiento lineal.
		➢ *g=1*: movimiento circular a izquierda.
		➢ *g=-1*: movimiento circular a derecha.
	✓	*tsm*: tiempos de simulación para cada tramo.
	✓	*ps*: ángulo de dirección respecto a x de cada segmento.

Figura 22: Resumen de la función *gtpt.m*.

5.2.2.3. La transición entre segmentos: la función gtat.m.

Se encuentra en el cuerpo de la función *gtpt.m*, de la que obtiene las coordenadas de los extremos, la longitud de los segmentos, la velocidad lineal y la máxima variación del ángulo para el UAV. Aplica los resultados del apartado *5.1.1.1*.

ae es el ángulo entre segmentos. (producto escalar de segmentos concurrentes). La diferencia de este respecto a 180º es la variación que se exige *ag*. Se determina además la distancia de cada segmento.

$$a_e = \arccos\left(\vec{t_1} \cdot \vec{t_2}\right)$$
$$a_g = \pi - a_e$$
$$a_m = a_e / 2$$
$$d_g = \frac{V_V}{umx \cdot tg(a_m)}$$

El signo de la componente z del producto vectorial de los segmentos concurrentes determina el sentido de giro para cada transición entre segmentos.

OBJETIVO	✓	Parámetros para la monitorización de la generación.
ENTRADAS	✓	*tvx*: vector [ns*2]. Coordenadas X de los extremos.
	✓	*tvy*: vector [ns*2]. Coordenadas Y de los extremos.
	✓	*lv* [ns*1]. Longitudes de los segmentos de la trayectoria.
	✓	*vv*: Velocidad lineal de referencia calculada para el UAV.
SALIDAS	✓	*dg*: distancia real de recorrido a lo largo del segmento.
	✓	*ag*: variación de ángulo respecto a la línea que se sigue.
	✓	*sg*: Control del giro para cada movimiento a generar:
		➢ *sg=0*: movimiento lineal.
		➢ *sg=1*: movimiento circular a izquierda.
		➢ *sg=-1*: movimiento circular a derecha.
	✓	*ps*: ángulo de dirección respecto a x de cada segmento.

Tabla 29: Esquema de la función *gtat.m*.

5.2.2.4. La simulación de la dinámica interna del UAV: gtf.mdl.

Se ha codificado en Simulink las expresiones correspondientes a la dinámica interna del UAV dejando a un lado el control de la altura del vehículo y asumiendo variaciones constantes del ángulo de dirección respecto a x.

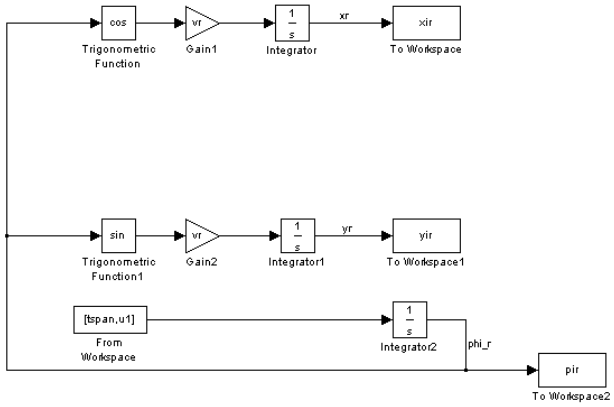

Figura 23: Modelo en Simulink del filtro no lineal

Se comprueba como el modelo recibe el valor del parámetro u_1 correspondiente a la variación del ángulo en cada fase de la simulación y el tiempo de simulación *tspan*. A la salida genera los puntos *Xir,Yir,*y *Pir*, referencias para el tramo simulado.

OBJETIVO	✓ Modelado y Simulación del filtro no lineal.
ENTRADAS	FROM WORKSPACE ✓ *tspan:* Tiempo de simulación del modelo gtf.mdl. ✓ *u1*: Variación del ángulo permitida en esta fase de la simulación.
SALIDAS	TO WORKSPACE ✓ Xi_R : Registro de la coordenada *X* de los puntos de la trayectoria de referencia generados en esta fase de simulación. ✓ Yi_R : Registro de la coordenada *Y* de los puntos de la trayectoria de referencia generados en esta fase de simulación. ✓ Pi_R : Registro del ángulo respecto al eje *X* de los puntos de la trayectoria de referencia en esta fase de la generación.

Tabla 30: Esquema del modelo *gtf.mdl*.

5.2.3. Ejemplo de generación de trayectorias.

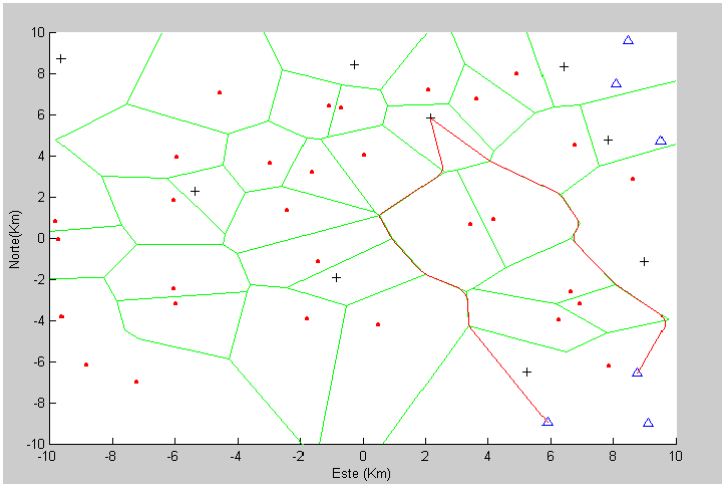

Figura 24: Generación de trayectorias de los UAVs del primer objetivo.

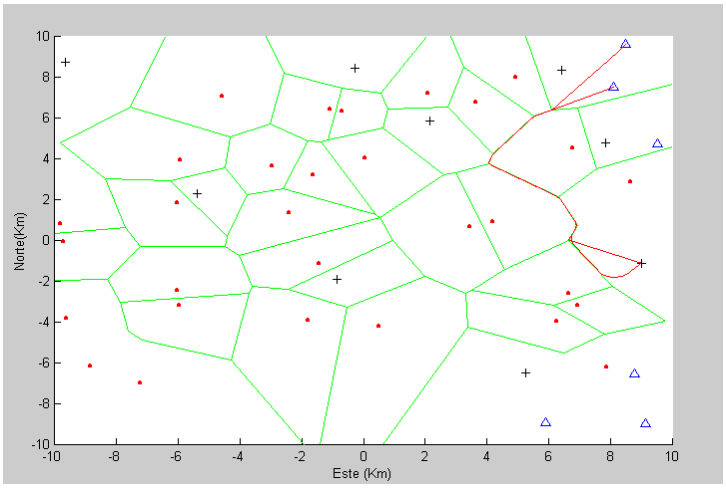

Figura 25: Generación de trayectorias de los UAVs el segundo objetivo.

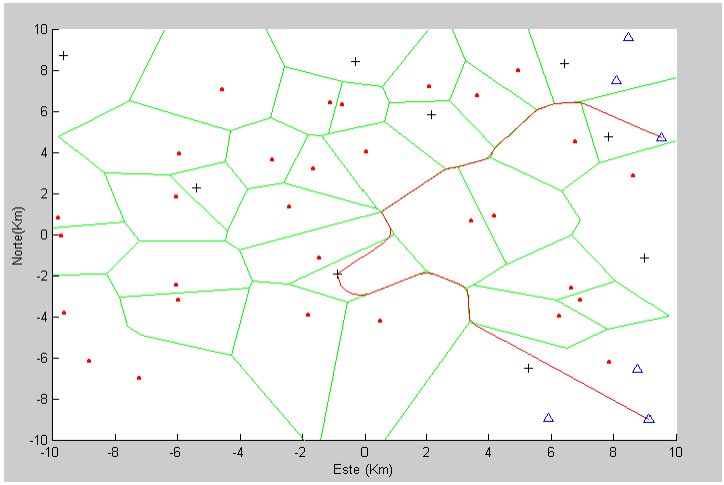

Figura 26: Generación de trayectorias de los UAVs del tercer objetivo.

Capítulo 6: Seguimiento de la trayectoria.

El UAV debe ser capaz de describir la ruta que el Generador de Trayectoria ha adecuado al movimiento de los vehículos aéreos no tripulados. El módulo anterior también ha parametrizado en tiempo la trayectoria, para asegurar que se alcanza el objetivo en el tiempo determinado por el sistema. A bordo de cada UAV se encuentra el sistema encargado de hacerle desarrollar la trayectoria. Cada vehículo contará con tres pilotos automáticos que corresponden a la posición, al ángulo de dirección y a la altitud. Estos mandos serán accionados por el Controlador a bordo del UAV. En el diseño del sistema de control se ha optado por un modelo desacoplado en el que las variables contenidas en los planos horizontal y vertical serán tratadas por separado. Se enumeran los dos subsistemas del controlador:

- ✓ Subsistema de control horizontal: dinámica del vehículo aéreo en el plano horizontal. Incluye el control de la posición del vehículo en función de las coordenadas x e y. El objeto del control alcanza también al ángulo con respecto al eje x de la dirección del movimiento del UAV y a la velocidad que experimenta el vehículo en su viaje hacia el objetivo. La implementación de este subsistema constituye el bucle externo de control, y la ejecución de las órdenes que proporcione el subsistema sobre los mandos a bordo del UAV, permitirán a éste cumplir con los resultados.

- ✓ Subsistema de control vertical: se refiere al control del vehículo aéreo en el plano vertical. La variable de control protagonista en este subsistema es la altura del UAV. El esquema de control para la altitud se conoce como bucle interno del sistema y permite el control de la magnitud de la altura, que será ejecutada mediante la acción sobre el mando en cuestión.

El diseño del Controlador del sistema incluye un modelo matemático basado en ecuaciones de primer y segundo orden, capaz de describir el movimiento del UAV. La técnica de control es descendente y el diseño no debe olvidar la interfaz necesaria para el intercambio de información entre los diferentes componentes del sistema.

6.1. Nomenclatura.

Estas son las variables que afectan a los diferentes subsistemas del controlador con que cuentan a bordo los diferentes UAVs. Todas las variables deberían contar con un supraíndice indicativo del UAV. Se prescinde de él, pero nunca se perderá de vista que las siguientes variables pertenecen al ámbito del controlador a bordo de cada UAV.

POSICIÓN REAL DEL UAV:

H: Perfil de altura de seguimiento de la trayectoria.

X: coordenada x del vehículo aéreo no tripulado.

Y: coordenada y del vehículo aéreo no tripulado.

Z: coordenada z del vehículo aéreo no tripulado.

V: velocidad transversal.

ψ*:* ángulo de dirección (respecto al eje x positivo).

DATOS ENTREGADOS POR EL GENERADOR DE TRAYECTORIA:

H_R*:* perfil de altura de referencia para el UAV.

X_R*:* coordenada x de referencia para vehículo aéreo no tripulado.

Y_R*:* coordenada y de referencia del vehículo aéreo no tripulado.

V_R*:* velocidad transversal de referencia.

ψ_R*:* ángulo de dirección (respecto al eje x positivo) de referencia.

SEÑALES DE CONTROL:

ψ_c*:* señal de control del error en el ángulo de cabeza.

V_c*:* señal de control del error en la velocidad.

K_x*:* comando del controlador de la componente x.

K_y*:* comando del controlador de la componente y.

K_z*:* comando del controlador de la componente z.

CONSTANTES DEL CONTROLADOR:

$\alpha_\psi\alpha_\upsilon$: constantes características del controlador de velocidad.

$\alpha_h\alpha_{h'}$*:* constantes características del controlador de altitud.

6.2. Subsistema de control horizontal.

El Controlador de Velocidad recibe a la entrada una serie de puntos que reproduce la trayectoria parametrizada en el tiempo que se desea recorrer (componentes X_r e Y_r), el ángulo de la dirección que se va a alcanzar (ψ_r) y la velocidad requerida (V_r). El subsistema de control horizontal entrega a la salida las variables que describen el estado del vehículo (X, Y, V, ψ) , estado que se alcanza gracias a la acción de las señales de control (V_C, ψ_C).

Los UAVs equipados con mandos de control de altura, de velocidad y de ángulo de dirección pueden ser descritos mediante ecuaciones muy similares a las del control de robots móviles. Se diferencian fundamentalmente en la velocidad lineal, pero la velocidad angular y el ángulo de dirección son aspectos muy parecidos en ambas situaciones. Para los UAVs existe una velocidad lineal mínima mayor que cero.

El seguimiento atiende al modelo dinámico del vehículo aéreo autónomo que está sujeto a restricciones en el ángulo de dirección y en la velocidad. Se resuelve el problema mediante el uso de funciones de control de Lyapunov variables en el tiempo para el control de las coordenadas del plano horizontal (X e Y) y del ángulo de dirección del vehículo aéreo. Las ecuaciones del controlador del plano horizontal para cada uno de los vehículos aéreos no tripulados se indican a continuación:

$$
\begin{aligned}
X_i &= V_i \cdot \cos(\Psi_i) \\
Y_i &= V_i \cdot sen(\Psi_i) \\
\overset{\circ}{\Psi} &= \alpha_\Psi \cdot (\Psi_i^c - \Psi_i) \\
\overset{\circ}{V}_i &= \alpha_V \cdot (V_i^c - V_i)
\end{aligned}
$$

El objetivo de control es encontrar los comandos de control V_c y ψ_c de manera que se tienda asintóticamente a las componentes de referencia:

$$
\begin{aligned}
X &\to X_R \\
Y &\to Y_R \\
Ph &\to Ph_R \\
V &\to V_R
\end{aligned}
$$

6.2.1. La estructura del controlador.

El integrador es el elemento principal de un sistema de control. Permite determinar la solución vía determinación de la primera derivada de la variable controlada. La primera derivada describe las variaciones que se producen en la variable con el paso del tiempo, variaciones que serán inducidas por el sistema de control.

El integrador empleado en el control del sistema tiene la siguiente estructura matemática:

$$\overset{\circ}{x} = f(x) + g(x) \cdot u, \quad f(0) = 0, x \in \Re^n, u \in \Re$$

En un principio se consideran las señales *V* y Ψ como señales de control que facilitan la determinación de las variables *X* e *Y* que indican la posición del UAV en el plano horizontal. Para el determinación de las coordenadas en el plano horizontal *X* e *Y* (más adelante se calcula el término de la función que afecta a la velocidad) se ha propuesto la siguiente función de Lyapunov, de la que se determina la primera derivada:

$$W_1 = 0{,}5 \cdot (X - X_R)^2 + 0{,}5 \cdot (Y - Y_R)^2$$
$$\overset{\circ}{W}_1 = (X - X_R) \cdot \left(V \cdot \cos(\Psi) - \overset{\circ}{X}_R \right) + (Y - Y_R) \cdot \left(V \cdot sen(\Psi) - \overset{\circ}{Y}_R \right)$$

Para conseguir que la primera derivada de la función de Lyapunov tenga signo negativo, debe de verificarse la siguiente condición:

$$V_R \cdot \cos(\Psi_R) = \overset{\circ}{X}_R - K_X \cdot (X - X_R)$$
$$V_R \cdot sen(\Psi_R) = \overset{\circ}{Y}_R - K_Y \cdot (Y - Y_R)$$

La elección realizada en el Generador de Trayectorias resulta fundamental para el cumplimiento de las igualdades anteriores. Se ajustarán las constantes positivas K_x y K_u. Las señales de control se despejan:

$$\Psi_R = arctg\left(\frac{\overset{\circ}{Y}_R - K_Y \cdot (Y - Y_R)}{\overset{\circ}{X}_R - K_X \cdot (X - X_R)}\right)$$

$$V_d = \sqrt{\left(\overset{\circ}{X}_R - K_X \cdot (X - X_R)\right)^2 + \left(\overset{\circ}{Y}_R - K_Y \cdot (Y - Y_R)\right)^2}$$

Si se sustituyen las expresiones anteriores en el cuerpo de la primera derivada de la función de Lyapunov se obtiene:

$$\overset{\circ}{W}_1 = -K_x \cdot (X - X_R)^2 - K_Y \cdot (Y - Y_R)^2 + (X - X_R) \cdot (V \cdot \cos(\Psi) - V_R \cdot \cos(\Psi_R)) + (Y - Y_R) \cdot (V \cdot sen(\Psi) - V_R \cdot sen(\Psi_R))$$

Se comprueba como el control depende del error en cada momento de la velocidad del UAV. Si se consideran las componentes x e y de la velocidad *(V.cos(ψ),V.sen(ψ))*, el error en esta variable y la su primera derivada:

$$e = (V \cdot \cos(\Psi) - V_R \cdot \cos(\Psi_R), V.sen(\Psi) - V_R \cdot sen(\Psi_R))$$

$$\overset{\circ}{e} = \begin{bmatrix} -V \cdot sen(\Psi) & \cos(\Psi) \\ V \cdot \cos(\Psi) & sen(\Psi) \end{bmatrix} \cdot \begin{bmatrix} \overset{\circ}{\Psi} \\ \overset{\circ}{V} \end{bmatrix} - \begin{bmatrix} -V_R \cdot sen(\Psi_R) & \cos(\Psi_R) \\ V_R \cdot \cos(\Psi_R) & sen(\Psi_R) \end{bmatrix} \cdot \begin{bmatrix} \overset{\circ}{\Psi}_R \\ \overset{\circ}{V}_R \end{bmatrix}$$

La función de Lyapunov completa debe asegurar la determinación tanto de las componentes X e Y, a partir de las cuales se infiere el ángulo Ψ, como de la velocidad del UAV. Se propone la siguiente función de Lyapunov para el sistema completo, con una primera derivada que indique las variaciones en la posición y velocidad del UAV:

$$W = W_1 + W_2 = W_1 + 0{,}5 \cdot e^T \cdot e$$

$$\overset{\circ}{W} = -K_X \cdot (X - X_R)^2 - K_Y \cdot (Y - Y_R)^2 + \left(\begin{bmatrix} X - X_R \\ Y - Y_R \end{bmatrix}^T + \overset{\circ}{e}^T\right) \cdot e$$

$$\overset{\circ}{e} = -K_Z \cdot e - \begin{bmatrix} X - X_R \\ Y - Y_R \end{bmatrix}$$

$$\overset{\circ}{W} = -K_X \cdot (X - X_R)^2 - K_Y \cdot (Y - Y_R)^2 - K_V \cdot e^T \cdot e$$

Para garantizar que la derivada de la función de Lyapunov completa sea negativa, y recordando que ya se ha impuesto la condición a la derivada de W_1, se tiene que dar la siguiente relación:

$$\begin{bmatrix}\Psi\\ V\end{bmatrix}=\begin{bmatrix}V^{-1}\cdot sen(\Psi) & V^{-1}\cdot\cos(\Psi)\\ \cos(\Psi) & sen(\Psi)\end{bmatrix}-\begin{bmatrix}-V_R\cdot sen(\Psi_R) & \cos(\Psi_R)\\ V_R\cdot\cos(\Psi_R) & sen(\Psi_R)\end{bmatrix}\cdot\begin{bmatrix}\overset{\circ}{\Psi}_R\\ \overset{\circ}{V}_R\end{bmatrix}+$$
$$+K_Z\cdot e+\begin{bmatrix}X-X_R\\ Y-Y_R\end{bmatrix}$$

La *señal de control* ψ_C estabiliza el valor del ángulo de dirección respecto a x. También influye en el seguimiento de las componentes *X* e *Y*. La dependencia de X e Y respecto a la velocidad se elimina con el cociente de las variables. Después de sustituir en las ecuaciones las expresiones correspondientes a la velocidad y ángulo de referencia y a las primeras derivadas de la velocidad y el ángulo se obtiene.

$$\Psi_c=\Psi+$$
$$\left(\frac{\overset{\circ}{\Psi}_R\cdot sen(\Psi)}{\alpha_\Psi\cdot V}+\frac{K_Z\cdot\cos(\Psi)}{\alpha_\Psi\cdot V}\right)\cdot\left(V_R\cdot sen(\Psi_R)\right)-\frac{\overset{\circ}{V}_R\cdot\cos(\Psi_R)\cdot sen(\Psi)}{\alpha_\Psi\cdot V}+\frac{\overset{\circ}{V}_R\cdot sen(\Psi_R)\cdot\cos(\Psi)}{\alpha_\Psi\cdot V}$$
$$+\left(\frac{\overset{\circ}{\Psi}_R\cdot\cos(\Psi)}{\alpha_\Psi\cdot V}-\frac{K_Z\cdot sen(\Psi)}{\alpha_\Psi\cdot V}\right)\cdot\left(V_R\cdot\cos(\Psi_R)\right)+\frac{(X-X_R)\cdot sen(\Psi)-(Y-Y_R)\cdot\cos(\Psi)}{\alpha_\Psi\cdot V}$$

Se extrae la *señal de control* V_C de la expresión correspondiente a la velocidad en las ecuaciones de la dinámica del UAV. El control de la velocidad, al igual que el del ángulo, afecta al seguimiento de las componentes *X* e *Y*. Sustituyendo las expresiones ya determinadas en la correspondiente al control de la velocidad se obtiene la señal de control:

$$V_c=V+$$
$$\left(\frac{\overset{\circ}{\Psi}_R\cdot sen(\Psi)}{\alpha_V}+\frac{K_Z\cdot\cos(\Psi)}{\alpha_V}\right)\cdot\left(V_R\cdot\cos(\Psi_R)\right)+\frac{\overset{\circ}{V}_R\cdot sen(\Psi_R)\cdot sen(\Psi)}{\alpha_V}+\frac{\overset{\circ}{V}_R\cdot\cos(\Psi_R)\cdot\cos(\Psi)}{\alpha_V}$$
$$+\left(\frac{K_Z\cdot\cos(\Psi)}{\alpha_V}-\frac{\overset{\circ}{\Psi}_R\cdot\cos(\Psi)}{\alpha_V}\right)\cdot\left(V_R\cdot sen(\Psi_R)\right)-\frac{(X-X_R)\cdot\cos(\Psi)+(Y-Y_R)\cdot sen(\Psi)}{\alpha_V}$$

6.3. Subsistema de control vertical.

El control de la altura del UAV en cada momento es responsabilidad del subsistema de control vertical, segundo elemento del Controlador del Sistema y desacoplado del anterior. Recibe la altura que se aspira alcanzar en cada momento (altitud de referencia) y el valor de las dos primeras derivadas (H_R, H_R', H_R''). El control vertical permite alcanzar la altura de referencia en el menor tiempo posible.

Del análisis de los cambios producidos en el perfil de altura de referencia a lo largo del tiempo, el dispositivo tendrá que responder a dichas variaciones y adecuarse a ellas. Al subsistema de control vertical se le proporciona la variación de referencia (H_R') y genera el seguimiento de la variación del perfil de altura real (H').

El subsistema de control vertical se describe a través de dos estados (z, H) y tiene como misión encontrar la entrada de control (H_c), que acciona los mandos a bordo del UAV que desarrollan el seguimiento del perfil de altura de referencia. El control de la variable altura debe basarse en un modelo matemático compatible con la dinámica permitida a cada UAV. Se toma por tanto como punto de partida la ecuación del movimiento vertical:

$$\ddot{H} = -\alpha_{\dot{H}} \cdot \dot{H} + \alpha_H \cdot \left(H_C - H\right)$$

El control de la altura requiere la evaluación del error cometido La señal de control debe permitir que el error en la altura cumpla la siguiente ecuación diferencial de segundo grado genérica, de la que se determinan las constantes del controlador K_H y $K_{H'}$, constantes positivas capaces de hacer frente a la mayoría de las situaciones:

$$e_H = H - H_R$$
$$\ddot{e}_H + K_{\dot{H}} \cdot \dot{e}_H + K_H \cdot e_H = 0$$

El error del perfil de altura cumplirá también la ecuación del movimiento vertical. La combinación de ambas ecuaciones es la vía que permite la determinación de la señal de control. El primer paso es la inserción de la ley del error en la ecuación del movimiento vertical:

$$-\alpha_{\dot{H}} \cdot \dot{H} + \alpha_H \cdot (H_C - H) - \ddot{H}_d + K_H \cdot \dot{e}_H + K_H \cdot e_H = 0$$

Se despeja la señal de control de la altura, que accionará el mando a bordo del UAV, operación que permitirá al vehículo aéreo no tripulado desarrollar la altura indicada en la variable H:

$$H_C = H + \frac{\left(\ddot{H}_R + \alpha_{\dot{H}} \cdot \dot{H} - K_{\dot{H}} \cdot \left(\dot{H} - \dot{H}_R \right) - K_H \cdot (H - H_R) \right)}{\alpha_H}$$

6.4. El Controlador del Sistema.

6.4.1. Codificación del Controlador.

Aunque se ha desacoplado el control de las variables contenidas en los planos horizontal y vertical, las funciones que implementan el control comparten las acciones correspondientes a ambos subsistemas de control. Dos son los ficheros que codifican el control de las variables de ambos planos desde cada subsistema UAV.

csuav.m	'CONFIGURACIÓN Y CONTROL DEL CONTROLADOR'
• cshv.mdl	CONTROLADOR DE LOS SUBSISTEMAS: ○ Control del plano horizontal. ○ Control del plano vertical.

Tabla 31: Software del Controlador del Sistema.

csuav.m corresponde a la configuración y el control del módulo. La configuración consiste en el ajuste de las constantes de ambos controladores mientras que el control se encarga de establecer las condiciones de partida que permitan al UAV el seguimiento de la trayectoria de referencia que se entrega a este módulo, desde la función *csuav.m* se ejecuta el modelo Simulink *cshv.mdl*.

6.4.2. La configuración y el control del módulo: csuav.m.

Es la función encargada de la configuración del Controlador del Sistema. Los parámetros del controlador se han escogido tomando como referencia una aeronave similar al *Boeing X-45* cuya dinámica interna aparece en el *apartado 1.4*. Los parámetros que afectan al control de los planos horizontal y vertical son los de la tabla, bajo esta consideración que no implica pérdida de generalidad:

α_v	0.2
α_Ψ	1.33
α_H	0.3364
α_H'	1.4680

Tabla 32: Parámetros del controlador del *Boeing X-45*.

Las parámetros anteriores se han configurado en el fichero *csuav.m* bajo la denominación *av*, *ap*, *ah* y *ah1* respectivamente. Las constantes del controlador se han ajustado con el fin de evitar sobreoscilaciones que afecten a la estabilidad del sistema. Se ha optado por un comportamiento levemente subamortiguado que tienda a la referencia en el menor tiempo posible. Se comprueba en las simulaciones que se adjuntan la respuesta extraordinariamente rápida en el tiempo y fiel en el seguimiento.

K_X	0.42
K_Y	0.42
K_Z	2.75

Tabla 33: Constantes del controlador del plano horizontal.

K_H	100
K_{H1}	15

Tabla 34: Constantes del controlador del plano vertical.

Si *Kx* o *Ky* alcanzan valores superiores la salida comienza a sobreoscilar. El aumento de *Kz* es muy sensible también para la estabilidad del sistema. Si *Kh* se incrementa se produce la sobreoscilación en el perfil de altura del UAV. Lo mismo ocurre con *Kh1*.

La otra tarea de la que se encarga es el control del seguimiento de la trayectoria de referencia gracias a la simulación por intervalos de simulación del modelo *cshv.mdl*. Se dispone de las referencias en los instantes *Tr* de muestreo de las siguientes variables que van a ser introducidas en el modelo Simulink *cshv.mdl*: X_R : componente *x* de la trayectoria de referencia. Y_R: componente *y* de la trayectoria de referencia. H_R: perfil de altura para el UAV. dH_R: primera derivada del perfil de altura.

La función *csuav.m* realiza la primera derivada de las dos primeras referencias y la segunda derivada del perfil de altura del UAV (derivando la última expresión). Se fija la condición inicial (de partida del UAV) ya que los integradores del modelo del controlador necesitan dicha condición para fijar el valor de la variable integrada en el instante inicial de simulación (seguimientos X_S, Y_S, V_S, Ph_S, H_S, dH_S).

Una vez cumplidos los requisitos se puede ejecutar el modelo *cshv.mdl* para las muestras de tiempo comprendidas entre el instante inicial y el tiempo de interceptación al objetivo o *TIG*. Las variables de salida del modelo se incorporan a los seguimientos anteriormente iniciados y se guarda en el fichero las señales de control de los pilotos automáticos de la velocidad, ángulo de dirección y altura.

OBJETIVOS	✓ Configuración de los controladores horizontal y vertical. ✓ Seguimiento de la trayectoria de referencia gracias a la simulación en el modelo *cshv.mdl* de las variables muestreadas en T_r
ENTRADAS	✓ Controlador del plano horizontal: ➢ *Xr* : componente x de referencia. ➢ *Yr*: componente y de referencia. ✓ Controlador del plano vertical: ➢ H_r. perfil de altura del UAV. ✓ T_R Instantes de muestreo de las referencias. Los extremos temporales son el instante inicial *cero* y el *TIG*.
SALIDAS	✓ Fichero *rcs.dat:* ➢ *cv*: señal de control de la velocidad. ➢ *cp*: señal de control del ángulo de dirección. ➢ *Ch*: señal de control de la altura. ✓ Seguimiento de las referencias*:* X_S, Y_S, V_S, Ph_S, H_S, dH_S.

Tabla 35: Resumen de la función *csuav.m*.

6.4.3. El controlador del sistema: el modelo Simulink cshv.mdl.

Se ha implementado en Simulink los controladores de los planos horizontal y vertical respectivamente. Aparecen en el mismo modelo ya que se simulan en los mismos instantes de muestreo. Pese a ello el control se efectúa de forma independiente, no existiendo ningún tipo de interacción entre ellos.

6.4.3.1. El controlador del plano horizontal.

Se compone de cuatro lazos realimentados correspondientes al control de las variables *X*, *Y*, *Ph* y *V*. Se modela en Simulink la dinámica del UAV:

$$\dot{X} = V \cdot \cos(\Psi)$$
$$\dot{Y} = V \cdot sen(\Psi)$$
$$\dot{\Psi} = \alpha_{\Psi i} \cdot \left(\Psi_C - \Psi\right)$$
$$\dot{V} = \alpha_{Vi} \cdot \left(V_{Ci} - V\right)$$

Para obtener las señales de control de velocidad y ángulo de dirección, que afectan al seguimiento, se determina la velocidad y ángulo de referencia a partir de las coordenadas X_R e Y_R de la trayectoria de referencia y sus derivadas:

$$\Psi_R = arctg\left(\frac{\dot{Y}_R - K_Y \cdot (Y - Y_R)}{\dot{X}_R - K_X \cdot (X - X_R)}\right)$$
$$V_R = \sqrt{\left(\dot{X}_R - K_X \cdot (X - X_R)\right)^2 + \left(\dot{Y}_R - K_Y \cdot (Y - Y_R)\right)^2}$$

Las señales de control de la velocidad y ángulo de dirección dependen de las componentes de referencia (X_R,Y_R) y de la velocidad y ángulo de referencia determinada por las ecuaciones anteriores. Influyen también las componentes X e Y de seguimiento, así como la velocidad y ángulo de dirección respecto a x. Se recuerdan las expresiones de estas señales de control que son las que se introducen en las ecuaciones de la dinámica del UAV, permitiendo el seguimiento de la trayectoria gracias al control del error que se produce en cada momento.

$$\Psi_c = \Psi +$$
$$\left(\frac{\overset{\circ}{\Psi}_R \cdot sen(\Psi)}{\alpha_\Psi \cdot V} + \frac{K_Z \cdot \cos(\Psi)}{\alpha_\Psi \cdot V} \right) \cdot \left(V_R \cdot sen(\Psi_R) \right) - \frac{\overset{\circ}{V}_R \cdot \cos(\Psi_R) \cdot sen(\Psi)}{\alpha_\Psi \cdot V} + \frac{\overset{\circ}{V}_R \cdot sen(\Psi_R) \cdot \cos(\Psi)}{\alpha_\Psi \cdot V}$$
$$+ \left(\frac{\overset{\circ}{\Psi}_R \cdot \cos(\Psi)}{\alpha_\Psi \cdot V} - \frac{K_Z \cdot sen(\Psi)}{\alpha_\Psi \cdot V} \right) \cdot \left(V_R \cdot \cos(\Psi_R) \right) + \frac{(X - X_R) \cdot sen(\Psi) - (Y - Y_R) \cdot \cos(\Psi)}{\alpha_\Psi \cdot V}$$

$$V_c = V +$$
$$\left(\frac{\overset{\circ}{\Psi}_R \cdot sen(\Psi)}{\alpha_V} + \frac{K_Z \cdot \cos(\Psi)}{\alpha_V} \right) \cdot \left(V_R \cdot \cos(\Psi_R) \right) + \frac{\overset{\circ}{V}_R \cdot sen(\Psi_R) \cdot sen(\Psi)}{\alpha_V} + \frac{\overset{\circ}{V}_R \cdot \cos(\Psi_R) \cdot \cos(\Psi)}{\alpha_V}$$
$$+ \left(\frac{K_Z \cdot \cos(\Psi)}{\alpha_V} - \frac{\overset{\circ}{\Psi}_R \cdot \cos(\Psi)}{\alpha_V} \right) \cdot \left(V_R \cdot sen(\Psi_R) \right) - \frac{(X - X_R) \cdot \cos(\Psi) + (Y - Y_R) \cdot sen(\Psi)}{\alpha_V}$$

6.4.3.2. El controlador del plano vertical.

Existe una sóla señal de control correspondiente a la altura, pero el modelo en Simulink implementa una ecuación diferencial de segundo grado (derivadas primera y segunda del perfil de altura). La dinámica del UAV respecto a la altura y la señal de control H_C son las que componen el modelo del subsistema de control vertical.

$$\overset{\circ\circ}{H} = -\alpha_{\overset{\circ}{H}} \cdot \overset{\circ}{H} + \alpha_H \cdot (H_C - H)$$

$$H_C = H + \frac{\left(\overset{\circ\circ}{H}_R + \alpha_{\overset{\circ}{H}} \cdot \overset{\circ}{H} - K_{\overset{\circ}{H}} \cdot \left(\overset{\circ}{H} - \overset{\circ}{H}_R \right) - K_H \cdot (H - H_R) \right)}{\alpha_H}$$

OBJETIVOS	✓	Modelado de los subsistemas de control horizontal y vertical.
ENTRADAS	FROM WORKSPACE (T_R Instantes de muestreo de las referencias)	
	✓	X_R , dX_R : componente x de referencia y primera derivada.
	✓	Y_R , dY_R componente y de referencia y primera derivada.
	✓	H_R $dH1_R$, $dH2_R$: perfil de altura, primera y segunda derivada.
SALIDAS	✓	Fichero *rcs.dat:*
	➢	*cp cv, ch*: señales de control de la velocidad, ángulo y altura.
	✓	Seguimiento de las referencias*:* X_{si}, Y_{si}, V_{si}, P_{si}, H_{si}, dH_{si}.

Tabla 36: Resumen del modelo *cshv.mdl*.

6.4.4. Ejemplo de seguimiento de trayectorias.

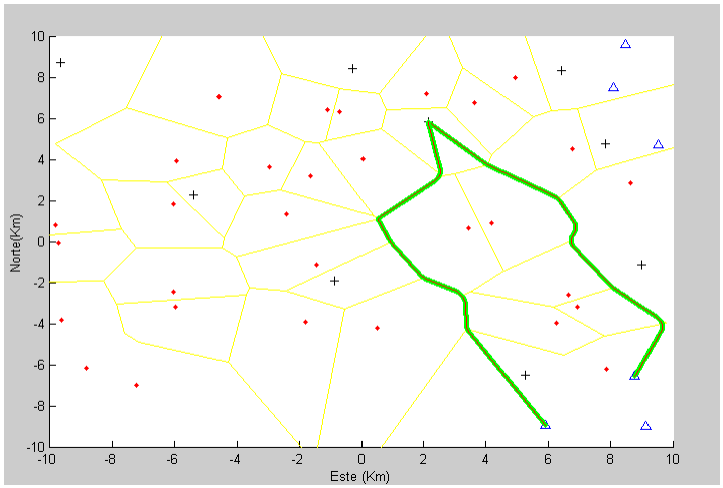

Figura 27: Seguimiento de la trayectoria para los UAVs del primer objetivo.

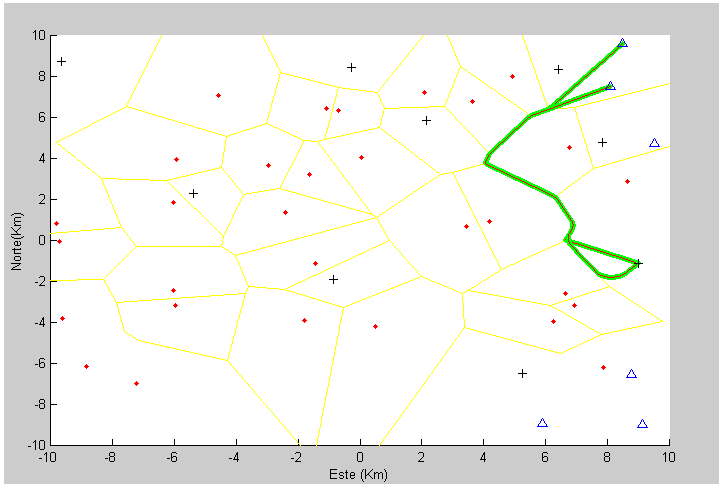

Figura 28: Seguimiento de la trayectoria para los UAVs del segundo objetivo.

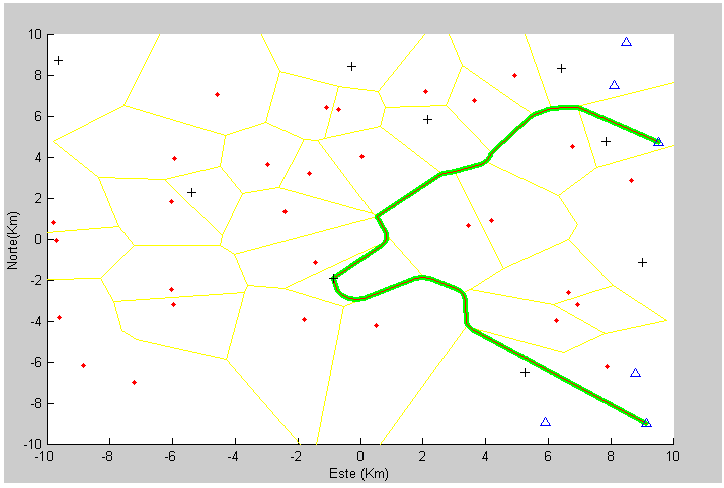

Figura 29: Seguimiento de la trayectoria para los UAVs del tercer objetivo.

Capítulo 7: Estrategia de comunicación.

La comunicación entre módulos tiene que resolverse de forma clara y primando el carácter abierto del sistema, que lleva a independizar al máximo el funcionamiento de cada módulo para favorecer la portabilidad de programas y lenguajes de programación. Con tal motivo el resultado de los módulos se registra en memoria gracias a ficheros de texto con extensión .DAT a los que se accede desde el resto de módulos si se requiere información desde alguno.

Se recuerda que los módulos operan a veces desde distintos subsistemas. Tanto en el Sistema Central como en los diferentes UAVs se dispone de memoria suficiente para almacenar estos datos. Habrá que diferenciar la comunicación que se produce entre módulos dependiendo del punto de partida y destino de la información:

- ✓ Sistema de Ficheros (dentro del mismo ámbito): puede ser entre componentes de módulos instalados en el sistema central o por el contrario en alguno de los subsistemas UAVs. En este caso la comunicación entre módulos es sencilla y consiste en acceder al sistema de ficheros interno (sistema de ficheros del Sistema Central o del subsistema UAV).

- ✓ Director de Comunicaciones (entre subsistemas diferentes): en este caso los intervinientes son componentes codificados en diferentes ámbitos. El componente que envía información escribe la información en un fichero de transmisión que el Director de Comunicaciones transmite al otro subsistema, momento a partir del cual el componente que recibe la información puede leerla. La comunicación en un sistema real será inalámbrica y habrá que establecer la frecuencia de transmisión y el ancho de banda que permita transmitir el volumen de información. Estos aspectos, por quedar fuera del objeto del proyecto no van a ser abordados y sólo se arbitra el formato de la información intercambiada.

7.1. Sistema de Ficheros.

Como se ha apuntado arbitra la comunicación interna de un determinado subsistema. El Sistema de Ficheros lo forma el conjunto de archivos reservados en la memoria para que cada módulo escriba la información de salida y pueda ser así consultada por el resto de módulos si lo necesitan cuando los componentes de estos se sitúen en la misma máquina.

Se diferencian dos clases de sistema de ficheros, dependiendo del ámbito en que actúe el intercambio de la información:

- ✓ Sistema de Ficheros del sistema central: instalado en el procesador encargado de desarrollar las tareas centralizadas del sistema. Se recuerda que puede situarse en un UAV que posea el carácter de líder o en una estación terrena. En la memoria del procesador se guardan los archivos de información, accediendo a ella para leer o escribir los resultados.

- ✓ Sistema de Ficheros en el UAV: habrá tantos como vehículos aéreos no tripulados operen en el sistema. La independencia de cada uno es total, almacenado los ficheros con las variables que el subsistema en cuestión requiere en los diferentes instantes de funcionamiento del UAV. La memoria es mayor que en el caso anterior (más cantidad de información).

En los apartados siguientes se abordan por separado cada uno de ellos, describiendo la estructura de cada fichero que registre los resultados de un módulo y analizando el código que apunta las operaciones de lectura y escritura en el sistema de ficheros. Las operaciones de lectura y escritura se realizan en función de la necesidad de información que presenten los módulos.

Todas las operaciones de lectura y escritura de los sistemas de ficheros se realizan desde los programas de gestión de los distintos subsistemas (*scen.m* ó *suav.m*). La información de entrada a los componentes de los módulos instalados viene fijada de antemano por lo que antes de invocar la ejecución de ese componente, desde el código que gestiona al subsistema se accede al sistema de ficheros y se recogen los datos conocidos y necesarios.

7.1.1. Sistema de ficheros en el sistema central.

El fichero *sfcen.*m contiene las órdenes de lectura de la información contenida en los archivos del sistema de ficheros central. La escritura de resultados se realiza al final del código de los componentes de los módulos que operan en el Sistema Central.

Han sido instalados componentes de dos módulos, que además cuentan con complementos en los subsistemas UAV. Por lo tanto se distinguen dos operaciones de escritura en el sistema de ficheros:

- ✓ Director de Asignación Central: coordina la asignación de objetivos a los UAVs. Evalúa la función de coordinación para la asignación resultado de la cual se obtienen una serie de asignaciones que servirán más tarde al Interceptor Simultáneo Central para coordinar temporalmente el ataque. El fichero *rda.dat* contiene el conjunto de asignaciones grabadas desde la función *dacen.m*.

- ✓ Interceptor Simultáneo Central: coordina temporalmente la llegada de UAVs a objetivos. Se recuerda que existen dos métodos para codificar el algoritmo de cooperación entre UAVs, de manera que la información grabada desde *iscen.m* diferirá según el caso. Esta información no se vuelve a usar en el sistema central y es útil en los subsistemas UAVs. El archivo es empleado por el Director de Comunicaciones y se ha llamado *uavrx3.dat*.

7.1.1.1. El archivo rda.dat.

Contiene la información del Director de Asignación. Los datos almacenados por filas son los que siguen:

- n_a: número de asignaciones realizadas.
- n: número de UAVs.
- Asignaciones (n_a filas) de n elementos que indiquen el número de objetivo.

```
4
6
8  4  4  3  3  8
8  4  3  3  3  8
8  4  3  1  3  8
8  4  3  1  8  8
```

7.1.1.2. El archivo uavrx3.dat.

La denominación del fichero se va a comprender en el apartado siguiente, debido a que se utiliza por el Director de Comunicaciones para ser enviado hacia los subsistemas UAVs. Se ha señalado cómo el Interceptor Simultáneo Central dispone de dos alternativas cuando desea aplicar el algoritmo de cooperación. El resultado del componente es diferente en cada caso y también lo será la interpretación que de los resultados se haga en el UAV. Por tanto habrá que indicar al UAV el método empleado y la configuración realizada en el módulo que corresponde al sistema central.

En el caso del método de velocidades lineales la estructura del fichero cuenta con los siguientes campos de información (por filas):

- *n*: número de UAVs coordinados.
- V_{mn}: velocidad mínima configurada.
- V_{mx}: velocidad máxima configurada.
- Algoritmo de cooperación (cadena de dos caracteres): 'vl'
- Velocidades de los *n* UAVs.
- Número de ruta (1...K).

```
6
0.12100000000000
0.14800000000000
ti
181.22775573 181.22775573 181.22775573 181.22775573 181.22775573 181.22775573
0 0 0 0 0 0
```

En el caso del método de tiempos de interceptación la estructura del fichero cuenta con los siguientes campos de información (por filas):

- *n*: número de UAVs coordinados.
- V_{mn}: velocidad mínima configurada.
- V_{mx}: velocidad máxima configurada.
- Algoritmo de cooperación (cadena de dos caracteres): 'ti'
- Tiempos de interceptación para cada UAV.
- Indicador de simultaneidad (*0* si es global y número de objetivo si local).

```
6
0.12100000000000
0.24800000000000
ti
125.6324499 125.6324499 64.9931351 64.9931351 64.9931351 125.6324499
4 4 8 8 8 4
```

7.1.2. Sistema de ficheros en el subsistema UAV.

El subsistema UAV cuenta con componentes de todos los módulos instalados. La necesidad de información es mayor que en el caso anterior por lo que se reserva una mayor cantidad de memoria. Desde el código del gestor del subsistema UAV, y como paso previo a la ejecución de un módulo se accede al sistema de ficheros para recabar la información requerida a través de la función *sfuav.m*. A continuación se enumeran los módulos para describir la información que registran en el sistema de ficheros del UAV:

- ✓ Planificador de Trayectorias: el módulo sólo opera desde el subsistema UAV y se recuerda que tiene encomendadas dos grandes operaciones, la elaboración del diagrama de Voronoi por un lado y la determinación de las trayectorias entre el vehículo y los objetivos. El resultado del módulo se guarda en los ficheros *rdv.dat* y *rpt.dat* respectivamente.

- ✓ Director de Asignación UAV: se encarga de la decisión individual sobre los objetivos. Esta información se transfiere hacia el sistema central por el Director de comunicaciones pero queda registrada en el sistema de ficheros en el archivo *rdai.dat* en el que se escribe al final de *dauav.m*.

- ✓ Interceptor Simultáneo UAV: de la lectura del fichero *uavrx3.dat* se sabe el algoritmo de cooperación empleado, las velocidades configuradas y los datos que permiten al vehículo determinar la velocidad lineal y el número de ruta. La escritura de los datos se realiza al final de la función *isuav.m*.

- ✓ Generador de Trayectoria: el funcionamiento corresponde íntegramente al UAV de manera que tras obtener del fichero *rpt.dat* la trayectoria al objetivo determinada por el Interceptor y la velocidad lineal en el fichero *ris.dat* se adecua la trayectoria de referencia que se almacena en *rgt.dat*.

- ✓ Controlador del Sistema: A través de la función *sfuav.m* se recaban los datos del fichero *rgt.dat* correspondientes a las componentes x, y de referencia y al vector de instantes de muestreo. También se recoge la altura de vuelo. El resultado del controlador es el seguimiento de la trayectoria gracias a la acción de los pilotos correspondientes que se almacenan en el fichero *rcs.dat*.

7.1.2.1. El archivo del diagrama de Voronoi: rdv.dat.

Registra la información relativa al diagrama de Voronoi. Se recuerda que el diagrama de Voronoi lo forma para cada UAV una parte básica compuesta por los lados del diagrama y las uniones del UAV a los vértices del polígono más una serie de aristas que unen los diferentes objetivos al diagrama y que se tendrán en cuenta sólo en el contexto del objetivo seleccionado. La información se estructura por filas:

- Segmentos del diagrama básico de Voronoi:
  - ns: número de segmentos del diagrama básico.
  - coordenada x del primer vértice del segmento (ns columnas).
  - coordenada x del segundo vértice del segmento (ns columnas).
  - coordenada y del primer vértice del segmento (ns columnas).
  - coordenada y del segundo vértice del segmento (ns columnas).
  - Coste total del segmento (ns columnas).
  - Coste de longitud del segmento (ns columnas).
  - Coste de exposición al peligro del segmento (ns columnas).
- Uniones hacia el diagrama básico de Voronoi:
  - nu: número de segmentos del diagrama básico.
  - iu: índice de uniones (N_o columnas con el número de uniones del objetivo).
  - coordenada x del primer vértice de la unión (nu columnas).
  - coordenada x del segundo vértice de la unión (nu columnas).
  - coordenada y del primer vértice de la unión (nu columnas).
  - coordenada y del segundo vértice de la unión (nu columnas).
  - Coste total de la unión (nu columnas).
  - Coste de longitud de la unión (nu columnas).
  - Coste de exposición al peligro de la unión (nu columnas).

7.1.2.2. El archivo de la planificación de trayectorias: rpt.dat.

Registra las trayectorias entre el UAV y los objetivos a los que se ordena llegar. De forma compacta se guardan todos los resultados que serán consultados por otros módulos en instantes de funcionamiento posteriores. La información se estructura por filas en los siguientes campos de datos:

- Información general de la ruta:
  - no: número de objetivos calculados conjuntamente.
  - k: número de rutas seleccionadas para cada objetivo.
  - ns: número de columnas para describir los segmentos de las rutas.
  - coste de longitud total de cada ruta (no*k columnas).
  - coste de exposición al peligro total de cada ruta (no*k columnas).
- Información de los segmentos que componen cada ruta:
  - ir: número de columnas que describe las rutas hacia cada objetivo.
  - Número de segmento que compone la trayectoria (bloques de columnas según el valor que ir asigne a ese objetivo). Existen k filas de este tipo.
  - Coste de longitud de los segmentos anteriores (se rellena con ceros cada bloque correspondiente a un objetivo según ir). Existen k filas de este tipo.
  - Coste de exposición al peligro de los segmentos (se rellena con ceros cada bloque correspondiente a un objetivo según ir). Existen k filas de este tipo.

7.1.2.3. La velocidad lineal y el número de trayectoria: el archivo ris.dat.

Registra la velocidad lineal que permite la interceptación simultánea global y el número de trayectoria correspondiente al objetivo asignado:

```
1
0.13433352000000
```

7.1.2.4. La trayectoria de referencia: el archivo rgt.dat.

En cada fila contiene la siguiente información correspondiente a la trayectoria adecuada para el UAV por el Generador de trayectorias:

- número de UAV.
- velocidad de referencia.
- ángulo de dirección inicial
- *nm*: número de muestras de la trayectoria de referencia.
- *tr*: *nm* instantes de muestreo entre *0* y el *TIG*.
- *xr*: *nm* muestras de la componente *x* de referencia.
- *yr*: *nm* muestras de la componente *y* de referencia.

7.1.2.5. Las señales de control: el archivo rcs.dat.

Contiene la acción que se tiene que desarrollar sobre los pilotos automáticos de velocidad, ángulo de dirección y perfil de altura en los diferentes instantes de tiempo. La información se estructura de esta forma.

- *nm*: número de muestras de la trayectoria de referencia.
- *tr*: *nm* instantes de muestreo entre *0* y el *TIG*.
- *vc*: *nm* muestras de la señal de control de velocidad.
- *phc*: *nm* muestras de la señal de control del ángulo de dirección.
- *hc*: *nm* muestras de la señal de control del perfil de altura.

7.1.2.6. Los ficheros del método de Eppstein.

Se hace notar que existen dos ficheros asociados al método de Eppstein y que se almacenan en memoria como los precedentes. La operación de lectura y escritura no se realiza desde la función *sfuav.m*, sino dentro del cuerpo de la función que coordina el manejo de la información intercambiada con el método de Eppstein. Esta función es *ptkf.m.* Estos ficheros de texto contienen la siguiente información:

El fichero de entrada al método, grafo.gr contiene los siguientes campos de información relativos al grafo en el que se realiza la búsqueda:

- *n:*.Número de nodos del grafo antecedido por la letra n.
- *m:* Número de arcos del grafo precedido por la letra m.
- *s:* nodo origen de la búsqueda (UAV) precedido por la letra s.
- *t:* nodo destino de la búsqueda (objetivo) precedido por la letra t.
- *arcos:* nodos extremos de cada arco y su coste precedidos por la letra a.

El fichero de salida rep.dat contiene los siguientes campos de información acerca de las rutas seleccionadas:

- *k:* rutas que se quiere computar.
- *K1:* rutas realmente computadas.
- Por filas, número de nodos de cada ruta y el indicador de cada uno.

7.2. El Director de Comunicaciones.

El Director de Comunicaciones es el módulo encargado de trasladar información de un subsistema hacia el otro. Esta comunicación inalámbrica requerirá de una frecuencia y ancho de banda que no se aborda en el proyecto. Se emula la comunicación de datos a través de operaciones de lectura y escritura de los ficheros.

Se establecen dos grandes grupos en función del punto de partida y llegada de la información transmitida, según corresponda al Sistema Central o alguno de los subsistemas UAV. El Director de Comunicaciones cuenta con una función implementadas en cada subsistema que regula la escritura de los datos que se quieren transferir o la lectura de los archivos que contienen la información recibida desde el otro extremo de la función. Las funciones que regulan la comunicación desde cada extremo reciben las siguientes denominaciones:

- ✓ dccen.m: se sitúa en el sistema central y está compuesta por un interruptor al que se accede a través de un código (*cenrxi* ó *centxi*) que indica el tipo de operación y en función de la cual los parámetros de entrada (e1..e6) y los de salida (s1...s3) adquieren significado. El nombre de los archivos en los que se escribe toman la denominación *uavrxi.dat*, ya que serán transmitidos hacia lo subsistemas UAV, donde *i* expresa el orden de las transmisiones efectuadas. Los archivos leídos fueron escritos por los UAVs y toman el nombre *cenrxi.dat* donde *i* expresa en este caso el número de recepción en el módulo central.

- ✓ dcuav.m: Es análogo al anterior, compuesto por un interruptor que lleva a un código según los caracteres indicados (*uavrxi* ó *uavtxi*) y a la interpretación de las entradas y salidas del fichero. Cuando se transmitan datos hacia el sistema central se escribirá en los ficheros con la denominación *cenrxi.dat* mientras que la recepción desde aquel subsistema se emula leyendo los ficheros con la denominación *uavrxi.dat*. En todos los casos *i* determina el orden en que se producen las transmisiones y recepciones.

En el Anexo A aparecen los códigos fuente de estos ficheros. Lo que se muestra a continuación es la estructura de cada uno de los ficheros utilizados por el Director de Comunicaciones.

7.2.1. La comunicación desde el Sistema Central.

En el fichero *dccen.m* se diferencian dos transmisiones hacia los UAVs y dos recepciones de información. Se dispone de cuatro entradas, más el código y de dos salidas. Estas son las comunicaciones realizadas:

7.2.1.1. Transmisión del escenario de trabajo: 'centx1'.

Se reciben las posiciones en el espacio de vehículos, obstáculos y amenazas respectivamente y se escriben en el fichero *uavrx1.dat* los siguientes campos de información en cada fila:

- *nv*: número de vehículos registrados.
  - componente x de cada UAV (*nv* columnas).
  - componente y de cada UAV (*nv* columnas).
- *no*: número de objetivos registrados.
  - componente x de cada objetivo (*no* columnas).
  - componente y de cada objetivo (*no* columnas).
- *np*: número de amenazas registradas.
  - componente x de cada amenaza (*np* columnas).
  - componente y de cada amenaza (*np* columnas).

7.2.1.2. Transmisión de la asignación efectuada: 'centx2'

El Sistema central comunica a los subsistemas UAVs la asignación coordinada por el Director de Comunicaciones. Se recibe el vector de asignación que se registra en el fichero *uavrx2.dat* cuya estructura se describe a continuación:

- *nv*: número de UAV coordinados.
- *a*: asignación (nv columnas), cada elemento representa el objetivo asignado.

7.2.1.3. Recepción de las decisiones individuales: 'cenrx1'

Al fichero *cenrx1.dat* acceden los diferentes UAVs para escribir en el las decisiones individuales con respecto a los distintos objetivos registrados en el sistema.

En este caso se lee dicho archivo seleccionado compactamente el conjunto de las decisiones individuales tomadas. La salida en esta circunstancia es la matriz M_{lp} que contiene *nv* filas y *no* columnas. Cada fila corresponde a la decisión individual acordada por el vehículo aéreo de dicha fila. La entrada del módulo sólo sirve para comprobación de errores.

7.2.1.4. Recepción de los costes de las rutas: 'cenrx2.dat'

Se accede al fichero *cenrx2.dat* con la intención de leer los datos correspondientes a los costes de longitud y exposición al peligro totales que se usan en la coordinación temporal. La entrada corresponde al número de UAVs y sirve sólo para la comprobación de posibles errores. Las salidas son dos vectores con $K*N_V$ elementos cada uno con los costes totales de las K rutas seleccionadas por cada UAV entre su posición y la del objetivo asignado por el Director de Comunicaciones en el Sistema Central.

7.2.2. La comunicación desde el subsistema UAV.

La función *dcuav.m* arbitra la comunicación para los distintos UAVs. Cada uno cuenta en su procesador con el código de esta función que le permite leer los datos enviados desde el sistema central o escribir en los archivos que se transmiten hacia el Sistema Central. Se enumeran tres recepciones y dos transmisiones de datos. En cada caso se describe el uso que se haga del fichero utilizado.

7.2.2.1. La transmisión de la decisión individual: 'uavtx1'

Se ha modelado el proceso de la siguiente manera. Se escribe en la variable global *M1*, en la fila correspondiente (entrada *e1* es el número de UAV) la decisión individual que cuentra con no columnas (entrada *e2*). La escritura en el fichero *centx1.dat* se efectuará en el momento en que todos los UAVs hayan comunicado el resultado de la decisión individual (comprobación de la suma de las columnas por fila).

- *nv*: número de UAVs.
- *no*: número de objetivos.
- *Mlp*: decisión individual del UAV (no columnas). Tantas filas como UAVs.

7.2.2.2. La transmisión de los costes de las rutas: 'uavtx2'

De manera análoga al caso anterior cada UAV escribe en las variables globales *L1* y *C1* el contenido de los costes de longitud y riesgo de las *K* rutas planificadas hacia el objetivo. Cuando todos hayan informado se accede al fichero *cenrx2.dat* en el que se registran los siguientes campos de información:

- *nv*: número de UAVs registrados.
- *k*: número de trayectorias seleccionadas.
- *cl*: coste de longitud de todas las rutas. (K*nv columnas).
- *cp*: coste de longitud de todas las rutas. (K*nv columnas).

7.2.2.3. La recepción del escenario de trabajo: 'uavrx1'

Se produce la lectura del archivo *uavrx1.dat* generado en el Sistema Central. La entrada *e1* corresponde al número de UAV y sólo tiene utilidad para comprobar errores. Las salidas *s1*, *s2* y *s3* corresponden a las localizaciones de los UAVs, objetivos y amenazas.

7.2.2.4. La recepción de la asignación: 'uavrx2'

Se lee la información contenida en el fichero *uavrx2.dat*. La entrada sirve para la detección de errores y la salida es el objetivo asignado que se encuentra en la posición correspondiente al número de UAV.

7.2.2.5. La recepción de la coordinación temporal: 'uavrx3'

Se lee la información contenida en el fichero *uavrx3.dat*. La entrada sirve para la detección de errores y las salidas corresponden por un lado a la velocidad lineal asignada al UAV (*s1*) y al número de ruta seleccionada entre las *K* posibles (*s2*).

NOTA: No se ha incluido en el Sistema Central la comunicación de la interceptación, ya que el fichero *uavrx3.dat* se escribe dentro del fichero *iscen.m*.

Capítulo 8: Conclusiones.

Una vez desarrollados los principios operativos del sistema de coordinación de múltiples UAVs y a la luz de los resultados obtenidos se señalan los aspectos superados y que se expusieron al comienzo de la memoria.

Se ha logrado establecer una acción coordinada entre los diferentes módulos integrantes del sistema de control de múltiples UAVs. Se ha conseguido establecer un diálogo entre los módulos, pero respetando la autonomía y simplificando al máximo la operación de estos módulos. Los componentes coordinados en el sistema son:

- ✓ Planificador de Trayectorias: ha realizado un trabajo utilizado por el resto de los módulos. La autonomía consiste en que opera previamente al resto y la interacción con el resto de módulos se realiza a través de los ficheros *rdv.dat* y *rpt.dat* que almacenan el diagrama de Voronoi y las rutas seleccionadas. Se ha reducido la búsqueda de trayectorias a un problema de dimensión finita tomando como base de la búsqueda los lados del diagrama de Voronoi y los costes definidos para éstos. Se ha aplicado un algoritmo de selección de rutas muy eficiente para poder determinar el conjunto de trayectorias hacia todos los objetivos. La técnica empleada ha sido el algoritmo de *Eppstein*.

- ✓ Director de Asignación: ha aplicado criterios de selección para determinar la solución adecuada que maximice dichos objetivos. Se ha resuelto el problema reduciendo considerablemente la complejidad inicial del mismo, haciendo partícipe de la operación a todos los UAVs junto al Sistema Central. Es autónomo en cuanto que opera una vez y la interacción con el Interceptor se produce a través del fichero *rda.dat*.

- ✓ Interceptor Simultáneo: se ha detallado el procedimiento que permite la *coordinación temporal* de las llegadas. Se han previsto diferentes configuraciones para el módulo que determinan el diálogo entre éste y el Director de Asignación por un lado y el Planificador por otro. De la configuración depende también la participación de los UAVs en el algoritmo de cooperación, siendo más importante en el método de los tiempos de interceptación que en el de velocidades lineales.

De la interacción de estos módulos se ha obtenido una solución coordinada temporalmente que lleva a los UAVs hacia los objetivos cubiertos en el ataque (que cuentan con un número de vehículos entre dos y seis resultado de la asignación) en el tiempo de interceptación global TIG determinado por el Interceptor Simultáneo. Esta cooperación se produce sólo entre estos tres módulos con componentes en el UAV y el Sistema Central (sólo los dos últimos).

A partir de este momento se ejecuta la solución desde cada UAV sin interactuar con el resto de vehículos. Dos son los módulos que intervienen en esta etapa:

- ✓ Generador de Trayectoria: adecua la trayectoria compuesta por la colección de segmentos entregada por el Planificador (rpt.dat) y parametriza en el tiempo dicha solución. Se forma la trayectoria de referencia.

- ✓ Controlador del Sistema: se encarga del seguimiento de la trayectoria de referencia en los instantes de muestreo de la misma, cuyos extremos espaciales corresponden a la posición del UAV y la del objetivo mientras que los instantes de muestreo extremos corresponden a *cero* y al *TIG* para asegurar el cumplimiento de la solución coordinada para el conjunto. El resultado es la acción sobre los pilotos automáticos correspondientes a la velocidad, ángulo de dirección y altura.

Se señalan por último las exigencias planteadas a los sistemas automáticos de coordinación de múltiples UAVs en la actualidad y que han quedado en el diseño:

- ✓ Carácter abierto: el sistema permite la inclusión de nuevos procedimientos para las tareas encomendadas al sistema.

- ✓ Portabilidad: permite la convivencia de diferentes lenguajes (MATLAB y C) y la inclusión de otros. Se respeta la estructura de datos del sistema de ficheros.

- ✓ Soporta diferentes escenarios de trabajo desde el punto de vista del número de agentes intervinientes. Es una solución muy adecuada.

- ✓ Dinamismo y tiempo real: son los puntos más débiles y los que se proponen para futuros desarrollos.

Capítulo 9: Desarrollos futuros.

Como se ha introducido al comienzo del documento, la filosofía que tiene que inspirar el diseño del sistema de múltiples UAVs debe enmarcarse en la búsqueda del carácter abierto del sistema. En este sistema de múltiples UAVs se han propuesto soluciones para todas las tareas. El diseño del sistema ha partido de ciertas especificaciones para las cuales la solución propuesta es adecuada, sin embargo la diversidad de escenarios de trabajo puede llevar a requerir otras soluciones.

Por tanto, el sistema debe permitir la inclusión rápida de los métodos más adecuados en cada caso. Existen formas muy distintas de afrontar tareas como la Planificación de Trayectorias (grafos, fuerzas virtuales, RRPs) o la asignación de objetivos a los UAVs. Por lo tanto se propone incorporar al sistema de múltiples UAVs aquellos componentes que le permitan adecuarse a la situación del escenario de trabajo. El sistema cuenta con las siguientes propiedades:

- ✓ Adaptabilidad o Reconfiguración dinámica: estos sistemas de control deben ser capaces de permitir una rápida y satisfactoria respuesta a los cambios motivados por nuevas expectativas o necesidades. Deben permitir las modificaciones sin violar la integridad del sistema.

- ✓ Modificación de los algoritmos de control: el sistema debe ser capaz de soportar los cambios en el algoritmo de control del sistema. Se tiene que insertar la nueva tecnología en la arquitectura del sistema sin la necesidad de rediseñar los componentes que ya forman parte de dicha arquitectura.

- ✓ Interoperatividad: Los sistemas de control actuales operan en un escenario heterogéneo, con componentes distribuidos. Este control debe llegar y ejecutarse en diferentes procesadores, los cuales pueden llegar a usar diferentes lenguajes de programación o situarse sobre diversas plataformas.

- ✓ Carácter Abierto: cualquier reconfiguración o intercambio de nuevos componentes requiere de una arquitectura de sistema flexible capaz de permitir la incorporación de algoritmos con distintos lenguajes de programación (C/C++, MATLAB, AMPL., etc...).

Se han independizado los módulos del sistema, especificando un sistema de ficheros en el que se guarda la información. De manera que la inclusión de nuevos métodos debe hacerse de acuerdo a estos ficheros de datos, manteniendo la estructura de resultados de cada módulo aunque el método empleado varíe. Partiendo del carácter abierto del sistema (independencia de los módulos) se proponen las siguientes tareas para aumentar la operatividad del sistema de múltiples UAVs:

1. Acciones de reconocimiento del escenario de trabajo, analizando las características de los agentes en el mismo (número, posición, tamaño, dinamismo,...). En función de las características de los obstáculos y del escenario general se tomarán las decisiones de operatividad en el sistema.

2. Arbitrar un procedimiento de elección de los métodos más oportunos para las características en el sistema. Se escogerá el algoritmo más adecuado para planificar las trayectorias (grafos, fuerzas virtuales, grafos de visibilidad, RRTs,...), la asignación de objetivos a los UAVs (métodos combinados o aproximados). En función de la configuración del sistema y de los datos arrojados por el reconocimiento el sitema escogerá los procedimientos.

3. Tareas de reconfiguración del sistema. Deben cargarse en los procesadores de los UAVs los algoritmos de los métodos seleccionados. Se recuerda que al añadir un método al sistema, los resultados de éste se han debido poner de acuerdo desde un principio para ser aceptados con normalidad por los demás módulos cuando accedan a los ficheros de datos.

4. Debe de asegurarse una comunicación en tiempo real entre estos componentes susceptibles de ser controlados, satisfaciendo las necesidades de ancho de banda, tiempo de respuesta o seguridad. Se recomienda el diseño de una interfaz de comunicación entre los distintos agentes en el sistema. En tiempo real los UAVs pueden recibir información sobre la presencia de obstáculos (dinamismo puede dar lugar a la presencia de nuevas amenazas), la altura del vehículos o emplear este sistema para evitar la colisión con otros UAVs.

Todo lo anterior incide en la profundización en el carácter abierto del sistema de múltiples UAVs. También pueden abordarse otras mejoras.

Para homogeneizar el lenguaje de programación de los diferentes módulos una vez introducidos otros métodos en el sistema, se recomienda la traducción de los códigos MATLAB a lenguaje C.

Algunas de las soluciones aportadas para el sistema de múltiples UAVs se pueden mejorar desde el punto de vista de la carga computacional recurriendo a versiones desarrolladas en C++ con la presencia de librerías como LEDA. Es el caso de la elaboración del diagrama de Voronoi o la búsqueda de los K caminos más cortos según el método de Eppstein. Facilitará el intercambio de información entre los diferentes componentes del sistema empleando comunicación entre procesos (programas y librerías muy adecuadas en C++).

Se propone por tanto seguir avanzando en cuestiones relativas al dinamismo del sistema automático de coordinación. Cuestiones como la prevención de colisiones puede incorporarse en el sistema ampliando la cooperación entre los UAVs. La solución propuesta tiene en cuenta una Planificación de Trayectorias de bajo nivel independiente en cada subsistema UAV que traza su diagrama y las trayectorias sin intermediación de otro UAV.

Se propone establecer una cooperación entre los planificadores de trayectorias de los UAVs que incluya comunicación entre vehículos y otorgar al vehículo la capacidad de influir en otro en el caso en que se produzcan variaciones sustanciales en el escenario de trabajo. De esta manera, además de la detección de colisiones en tiempo real es posible hacer frente a la presencia de amenazas dinámicas.

Habrá que arbitrar la relación entre los vehículos y asignar a uno de ellos la capacidad de liderazgo en función de la cual controle la operación de los UAVs. Se requieren en este caso una estrategia de comunicación de mayor volumen que incrementa las necesidades de ancho de banda en las transmisiones.

Con todos estos apuntes el sistema automático de coordinación de múltiples UAVs adquirirá cualidades cada vez más acordes con la necesidad de operación en tiempo real y evaluación dinámica de las situaciones que se presenten en el escenario de trabajo. Resulta conveniente el estudio de diferentes escenarios en función de los cuales se escojan los métodos de planificación (RRT: Rapidly Exploring Random Trees, RPM: Probabilistic Road Maps, etc).

ANEXO.1. Código Fuente del sistema de múltiples UAVs.

Existen dos ámbitos en los que se produce el funcionamiento del sistema de múltiples UAVs. Dado el carácter distribuido con que se ha dotado a la arquitectura del sistema, se ha procurado repartir las tareas entre los distintos procesadores con que cuenta el sistema a bordo de los diferentes vehículos aéreos no tripulados.

Los módulos que conforman el sistema realizan operaciones que a veces es posible encargar a los UAVs. De esta manera se pueden desarrollar de forma paralela ahorrando tiempo de computación en el funcionamiento. Sin embargo determinadas decisiones habrán de tomarse en conjunto en función de los resultados entregados por los UAVs. Los algoritmos encargados de tomar estas decisiones irán equipados en el sistema central, y serán los imprescindibles.

Se ha procurado independizar los módulos en el mayor grado posible para facilitar la portabilidad de los programas de manera que sean posibles las tareas de mantenimiento de los programas sin afectar la ejecución del resto. Con tal motivo todos los resultados de los módulos se graban en ficheros cuya estructura de datos en conocida. Si se pretende cambiar el código de algún módulo o el lenguaje de programación, habrá que poner de acuerdo los resultados de este con el código antiguo.

Todos los resultados de los módulos se guardan en el sistema de ficheros que se describirá más adelante. Se leerán los parámetros de entrada y se escribirá el resultado de salida. Debido a que los módulos cuentan con algoritmos tanto en el sistema central como en los diferentes UAVs, el intercambio de datos entre los dos ámbitos se modela mediante la lectura y escritura de los ficheros dedicados a tal fin. Estos aspectos aparecen en el Director de Comunicaciones.

En los siguientes apartados se detalla el código fuente de los módulos de la arquitectura del sistema, así como código destinado a la gestión global del mismo. Se diferencia para cada caso los componentes centralizados y los a bordo de los UAVs.

A.1.1. La gestión de los subsistemas.

Las partes del código fuente encargadas de la gestión global del sistema se caracterizan por llevar el control de la ejecución del mismo. Se requiere el funcionamiento de los componentes de los módulos equipados en dichos subsistemas, y se efectúa la sincronización del proceso a través de las llamadas de lectura y escritura de los ficheros que guardan los resultados sin los que no se puede arrancar el funcionamiento de un determinado módulo.

El código que se proporciona corresponde a los algoritmos a bordo de los sistemas central y UAV. Existe una versión aplicada a la obtención de resultados gráficos que se omite en el presente Anexo. Tampoco aparecen otros programas que se han empleado para pruebas varias.

El sistema de múltiples UAVs cuenta con el sistema central y los diferentes subsistemas UAVs. El código correspondiente a la gestión de los mismos se expone a continuación.

A.1.1.1. Gestión del sistema central: la función scen.m.

El fichero *scen.m* cuenta con las llamadas a los módulos con componentes en el sistema central. Las entradas necesarias para ejecutar este algoritmo los recoge de los diferentes ficheros. La sincronización de la ejecución de los procesos se lleva a cabo permitiendo la lectura de los ficheros por parte de los diferentes subsistemas UAVs. Las llamadas al Director de Comunicaciones aparecen en este fichero, así como la ejecución de los componentes de los módulos en el sistema central.

```
%*************************************************************************
% 'GESTIÓN DEL SISTEMA CENTRAL'.
%*************************************************************************
mad='ga'; % Modo de adquisición: 'ga'(generación aleatoria) 'fd'(fichero).
nv=3;     % Número de UAVs en el sistema.
np=15;    % Número de amenazas registradas en el sistema.
no=3;     % Número de objetivos presentes en el sistema.
le=20;    % Longitud del lado del cuadrado (km) del escenario de trabajo.
me=0.2*le;% Margen de confinamiento de los UAVs cerca del borde exterior (km).
switch mad   % Se recibe la posición del escenario de trabajo del sistema.
   case 'ga' % Generación aleatoria de los agentes en el escenario.
      [V,O,P]=escmp(nv,no,np,le,me);       % Escenario virtual aleatorio.
   case 'fd' % Adquisición de los datos a través del fichero.
      [V,O,P]=sf('fd',nv,no,np);% Se recogen los datos posicion.dat.
   otherwise                    % Si la opción no es válida.
```

```
      return              % No se pueden adquirir los datos de la posición.
end
[ok]=dccen('ctx1',V,O,P);     % Planificador de Trayectorias de los UAVs.
Mlp=dccen('crx1',nv);   % decisiones individuales de todos los UAVs
err=dacen(Mlp);         % Director de Asignación genera la matriz de asignación
if err==1               % Si se produce un error en el Director de Asignación.
   return               % Acabar con el funcionamiento.
end
[err,A]=sfcen('da',nv); % Se accede al sistema de fichros para leer rda.dat.
if err==1               % Si se produce un error en la lectura del fichero.
   return               % Acabar con el funcionamiento.
end
ia=1;                   % Índice de recorrido de asignaciones.
na=size(A,2);    % tt asignaciones en orden descendente del valor de la FO.
while (ia<=na)   % Mientras se cuente con una nueva asignación en la matriz A.
   a=A(:,ia);    % Se selecciona la asignación (columna determinada por na).
   [ok]=dccen('ctx2',a);
   [Lr,Cr]=dccen('crx2',n);% Se reciben las longitudes y costes de las rutas.
   [err]=iscen(a,Lr,Cr,1); % Las llegadas a objetivos se simultanean.
   if err==1     % Si no sincronización de los UAVs, probar nueva asignación.
      ia=ia+1;   % Incrementar el índice de recorrido de asignación A.
   else          % Si el Interceptor Simultáneo genera resultados adecuados.
       na=0;     % Se fuerza la salida del bucle.
   end
end
if err==1        % Si ninguna de las asignaciones consigue la sincronización.
   a=A(:,1);     % Se selecciona la primera de ellas (función objetivo máxima).
   [err]=iscen(a,Lr,Cr,0); % Sincronización de las llegadas a objetivos.
end
```

A.1.1.2. La gestión del subsistema UAV: el fichero suav.m.

El subsistema UAV va equipado en cada uno de los vehículos aéreos no tripulados. El fichero MATLAB *suav.m* realiza las llamadas a los módulos equipados a bordo de cada vehículo (funciones *ptuav*, *dauav.m*, *isuav.m*, *gtuav.m*, *csuav.m* y *dauav.m*). La recogida de datos desde el sistema central se realiza gracias al Director de Comunicaciones del lado de cada UAV, la función *dcuav.m*.

```
%************************************************************************
% 'GESTIÓN DEL VEHÍCULO AÉREO NO TRIPULADO i'.
%************************************************************************
i=1;
k=10;
xm=10;
global M1;
global L1;
global C1;
[V,O,P]=dcuav('urx1',i);   % localizaciones del diagrama de Voronoi.
n=size(V,1);               % Número de UAVs del sistema de múltiples vehículos.
m=size(O,1);
M1=zeros(n,m);
L1=zeros(k,n);
C1=zeros(k,n);
v=V(i,:);                  % Posición UAV_i.
[err]=ptuav(v,O,P,xm);     % PLANIFICADOR DE TRAYECTORIAS.
[I,Cl,Cp]=sfuav('da',m);   % Costes de longitud y exposición al peligro.
[err]=dauav(I,Cl,Cp);   % DIRECTOR DE ASIGNACIÓN. Decisión individual del UAV.
mlp=sfuav('dai',m);
```

```
[ok]=dcuav('utx1',i,n,mlp);% Se transmite la decisión individual.
err=1;                     % El control se va a transferir al sistema central.
while err~=0
    [o]=dcuav('urx2',i,n);  % El Director de Asignación adjudica objetivo.
    [Lv,Cv]=sfuav('is',o);  % rpt.dat recoge longitudes y rutas.
    [ok]=dcuav('utx2',i,n,Lv,Cv); % Al sistema central las longitudes y costes.
    [err,vmn,vmx,ac,c1,c2]=dcuav('urx3',i,n);   % Interceptor Simultáneo.
    if err==0     % Si resultado válido, velocidad del UAV y el número de ruta.
       [err]=isuav(ac,vmn,vmx,c1,c2,Lv,Cv);        % El Interceptor: vl y nr.
   end            % Si el resultado no es válido se espera una nueva asignación.
end
[vv,tvx,tvy,lv]=sfuav('gt',o,m);  % Información Generador de Trayectorias.
gtuav             % GENERADOR DE TRAYECTORIA:
[ok]=dcuav('utx3',i);      % Trayectoria  ya parametrizada en el tiempo.
[vr,phr,nm,tr,xr,yr]=sfuav('cs',i);     % Se ordena atacar el objetivo.
csuav         % SEGUIMIENTO DE LA TRAYECTORIA: Se pone en marcha controlador.
```

A.1.2. El Planificador de Trayectorias.

Todos los algoritmos encargados de planificar las posibles trayectorias entre el UAV y el objetivo asignado se encuentran programados a bordo de cada UAV. No existen tareas que hayan de tomarse de forma centralizada, luego cada vehículo aéreo no tripulado traza sus trayectorias de manera independiente, lo que permite la mayor adecuación de la tarea a las características de cada UAV.

A.1.2.1. El control y la configuración: el fichero ptuav.m.

El Planificador de Trayectorias desarrolla dos tareas diferenciadas. Desde el fichero de configuración y control *ptuav.m* se invoca la ejecución de las funciones que vienen a continuación:

- ✓ Composición del diagrama de Voronoi: *ptvuav.m*. Desde este se realizan las llamadas a las demás funciones, dependiendo de la configuración prevista (*delaunay.m*, *voronoi.m*, programa *TESS*, *ptvu.m* y *ptvc.m*). Existen dos posibilidades de ejecución dependiendo del instante en que se llame. Se discrimina esa diferencia de funcionamiento a través de un parámetro opción.

- ✓ Búsqueda de las K trayectorias más cortas entre origen y destino: en el fichero *ptkuav.m*. La configuración lleva a la ejecución de unas u otras funciones, dependiendo de los métodos escogidos. Los algoritmos que forman parte de la tarea de búsqueda son *ptkd.m*, *dijkstra.m*, *ptkf.m* y el programa *EPPSTEIN*.

```
function [err]=ptuav(v,O,P,xm)
%*************************************************************************
% MÓDULO: 'PLANIFICADOR DE TRAYECTORIAS'.
%   'Control y configuración'
%*************************************************************************
% CONFIGURACIÓN:
ci=0.5;    % Relación coste total del segmento y los CL y CP.
dv='mb';   % Método del Diagrama de Voronoi: 'mb' (matlab) 'fr' (fortune,C).
k=10;      % Número de trayectorias entre el UAV v y el objetivo.
bk='dj';   % Método de Búsqueda de las rutas: 'dj'(dijkstra), 'ep'(eppstein).
% PLANIFICADOR DE TRAYECTORIAS.
m=size(O,1);  % Número de objetivos que se tienen que cubrir.
T=[];         % Matriz de sucesión de segmentos.
Cl=[];        % Matriz con las longitudes de los segmentos.
Cp=[];        % Matriz con los costes de peligro.
I=[];         % Índice.
Ck=[];        % Matriz con los costes totales de las trayectorias.
Lk=[];        % Matriz con las longitudes totales de las rutas.
Cu=[];        % Matriz de costes de las uniones.
ug=[];        % Matriz de coordenadas de las uniones.
iu=[];        % Matriz de índices de las uniones.
[sx,sy,Ca]=ptvuav(dv,ci,v,O,P,0,xm);% DIAGRAMA DE VORONOI.
[ns,nn,s,sx,sy]=ptkg(v,sx,sy,Ca); % Se construye el grafo dirigido.
Ca=ptvc(P,sx,sy,ci);              % Costes total, de peligro y de longitud.
sg1=[sx,sy];               % Componente x,y de los vértices del lado [Ns*4].
t=nn+1;                    % Nodo destino (objetivo en curso).
for j=1:m                  % Se recorren todos los objetivos del sistema.
   o=O(j,:);               % Objetivo que tiene que alcanzarse.
   [ux,uy,Cb]=ptvuav(dv,ci,o,sg1,P,1,xm);   % UNIÓN DEL OBJETIVO CON VÉRTICES.
   nu=size(ux,1);                           % Número de uniones.
   ng=ns;
   for p=1:nu
      [i1,j1]=find(sx==ux(p,2) & sy==uy(p,2));
      ng=[ng;[ns(i1(1)),t]];
   end
   ug1=[[ux(:,2),ux(:,1)],[uy(:,2),uy(:,1)]];  % Accesos objetivo al diagrama.
   sg2=[sg1;ug1];                              % Accesos objetivo al diagrama.
   C=[Ca;Cb];                 % Costes de los segmentos anteriores.
   [err]=ptkf(ng,C(:,1),s,t); % Se registra el grafo en el archivo grafo.gr.
   if bk=='dj'                % Si se aplica el algoritmo de Dijkstra.
      [Tv,Lt,Ct,Cv,Lv]=ptkuav(bk,k,v,o,sg2,C); % BÚSQUEDA DE LAS K RUTAS.
   elseif bk=='ep'            % Si se aplica el método de Eppstein.
      [Tv,Lt,Ct,Cv,Lv]=ptkuav(bk,k,s,t,ng,C);  % BÚSQUEDA DE LAS K RUTAS.
   else                       % Si no se dispone de método de búsqueda.
      err=1;                  % Error de funcionamiento.
      return                  % Salir del procedimiento.
   end
   T=[T,Tv];                  % Número de segmento de las No*K trayectorias.
   Cl=[Cl,Lt];                % Longitudes de los segmentos que las forman.
   Cp=[Cp,Ct];                % Costes de amenaza de los segmentos.
   Ck=[Ck;Cv];                % Costes totales de las No*K trayectorias.
   Lk=[Lk;Lv];                % Longitudes totales de las No*K trayectorias.
   I=[I;size(Lt,2)];          % Índice de las matrices de segmentos.
   iu=[iu,size(Cb,1)];                  % Número de uniones.
   ug=[ug,[ug1(:,1:2)';ug1(:,3:4)']];   % Uniones a objetivos.
   Cu=[Cu,Cb'];                         % Costes de uniones a objetivos.
end
tam=size(Cl);              % FICHERO DE LAS K RUTAS: rtp.dat.
fid=fopen('rpt.dat','w');  % Se abre el fichero rpt.dat.
fprintf(fid,'%d \n',m);    % Número de objetivos.
fprintf(fid,'%d ',tam(1));
fprintf(fid,'\n');
fprintf(fid,'%d ',tam(2));
fprintf(fid,'\n');
fprintf(fid,'%4.14f ',Lk); % Se graban las longitudes totales de las rutas.
fprintf(fid,'\n');
fprintf(fid,'%4.14f ',Ck); % Se graban los costes totales de las rutas.
fprintf(fid,'\n');
```

```
fprintf(fid,'%d ',I);      % Se graban los segmentos de las trayectorias.
fprintf(fid,'\n');
for i=1:tam(1)
    fprintf(fid,'%d ',T(i,:)');% Se graban los segmentos de las trayectorias.
    fprintf(fid,'\n');
end
for i=1:tam(1)
    fprintf(fid,'%4.14f ',Cl(i,:)'); % Se graban las longitudes de las rutas.
    fprintf(fid,'\n');
end
for i=1:tam(1)
    fprintf(fid,'%4.14f ',Cp(i,:)'); % Se graban los costes de las rutas.
    fprintf(fid,'\n');
end
fclose(fid);         % Se cierra el fichero rgt.dat.
ns=size(sg1,1);      % Número de segmentos originales del diagrama.
nu=sum(iu);          % Número de uniones realizadas.
fid=fopen('rdv.dat','w');         % INFORMACIÓN DEL DIAGRAMA: fichero rdv.dat.
fprintf(fid,'%d \n',ns);          % Número de objetivos.
for i=1:4
    fprintf(fid,'%4.14f ',sg1(:,i)'); % Coor x,y de los vértices de segmentos.
    fprintf(fid,'\n');
end
for i=1:3
    fprintf(fid,'%4.14f ',C(:,i)');   % Costes totales, CP y CL del segmento.
    fprintf(fid,'\n');
end
fprintf(fid,'%d \n',nu);              % Número de uniones a objetivos.
fprintf(fid,'%d ',iu);                % Número de uniones por cada objetivo.
fprintf(fid,'\n');
for i=1:4
    fprintf(fid,'%4.14f ',ug(i,:));   % Coor x,y de los vértices de las uniones.
    fprintf(fid,'\n');
end
for i=1:3
    fprintf(fid,'%4.14f ',Cu(i,:));   % Costes totales, CL y CP la unión.
    fprintf(fid,'\n');
end
fclose(fid);
err=0;
```

A.1.2.2. El Diagrama de Voronoi: la función ptvuav.m.

Existen dos formas de composición del diagrama. Por un lado el empleo de los comandos de MÁTLAB *delaunay.m* y *voronoi.m*. Por el otro la ejecución de los programas en lenguaje C que implementan el método de Fortune introduciendo las variaciones que consiguen una mayor velocidad de ejecución. Una vez trazado el diagrama habrá que determinar los accesos desde al UAV a los vértices del polígono correspondiente (función *ptvu.m*) y calcular los costes totales, de longitud y de exposición al peligro de cada lado de Voronoi (función *ptvc.m*).

```
function [sgx,sgy,C]=ptvuav(dv,ci,e1,e2,e3,op,xm)
%**************************************************************************
% MÓDULO: 'PLANIFICADOR DE TRAYECTORIAS'
%   'Diagrama de Voronoi'
%**************************************************************************
if op==0     % La primera entrada en la función genera el diagrama de Voronoi.
```

```
    v=e1;            % Posición del UAV.
    O=e2;            % Posición de los objetivos.
    P=e3;            % Posición de las amenazas.
    L=[P;O];  % Estas son las localizaciones del diagrama de Voronoi. [Nl*2].
    switch dv
        case 'mb' % Si se emplean las órdenes de Matlab 'delaunay' y 'voronoi'.
            tri=delaunay(L(:,1),L(:,2));  % triangulación de Delaunay.
            [sx,sy]=voronoi(L(:,1),L(:,2),tri);  % diagrama de Voronoi.
            sg=[sx',sy'];                        % Matriz de segmentos compacta.
            [mxg,jmx]=max(abs(sg),[],2);% Máximo valor de componente x segmento.
            imx=find(mxg>xm);          % Segmentos que sobresalen del escenario.
            nmx=length(imx);        % Número de segmentos exteriores.
            if nmx>0
                jmx=jmx(imx);       % Componente exterior al escenario.
                smx=sg(imx,:);      % Componentes del segmento.
                xb=smx(:,1:2);      % Componentes x.
                yb=smx(:,3:4);      % Componentes y.
                a=(yb(:,1)-yb(:,2))./(xb(:,1)-xb(:,2)); % y=ax+b del segmento.
                b=yb(:,1)-a.*xb(:,1);                   % y=ax+b del segmento.
                for i=1:nmx
                    i1=imx(i);              % Índice x.
                    j1=jmx(i);              % Índice y.
                    xmx=sign(smx(i,j1))*xm;% Truncar el valor máximo al permitido.
                    if j1<3          % Si el máximo corresponde a componente x.
                        sg(i1,j1+2)=a(i)*xmx+b(i);    % y=ax+b del segmento.
                        sg(i1,j1)=xmx;
                    else
                        sg(i1,j1-2)=(xmx-b(i))/a(i);
                        sg(i1,j1)=xmx;
                    end
                end
                [mxg,jmx]=max(abs(sg),[],2);
                imx=find(mxg<=xm);
                sg=sg(imx,:);
            end
            sgx=sg(:,1:2);
            sgy=sg(:,3:4);
        case 'fr' % Si se emplea el 'método de Fortune' en lenguaje C.
            disp ('Ejecutar FORTUNE O TESS')
            disp ('Pulse cuando lo haya ejecutado')
            pause
        otherwise
            return % Si la opción introducida no es adecuada, se concluye.
    end
    [ux,uy]=ptvu(v,sx',sy');   % Segmentos que unen el UAV con los vértices.
    sgx=[sx';ux];% Coordenada x, se añaden uniones del UAV con el diagrama.
    sgy=[sy';uy];% Coordenada y, se añaden uniones del UAV con el diagrama.
    C=ptvc(P,sgx,sgy,ci);%Coste de Voronoi.3 columnas [total,longitud,peligro].
elseif op==1  % uniones del objetivo con los vértices del polígono.
    v=e1;                   % Objetivo a unir con el diagrama.
    sx=e2(:,1:2);           % Componente x de los segmentos del diagrama.
    sy=e2(:,3:4);           % Componente y de los segmentos del diagrama.
    P=e3;                   % Componente y de los segmentos ya determinados.
    [sgx,sgy]=ptvu(v,sx,sy);% Segmentos que unen el UAV con los vértices.
    C=ptvc(P,sgx,sgy,ci);   % Se calcula el coste de los accesos al objetivo.
else
    return % Error al introducir op.
end
```

A.1.2.2.1. Las uniones de un punto a los vértices del polígono: ptvu.m.

```
function [ux,uy]=ptvu(v,sx,sy)
%*************************************************************************
% MÓDULO: 'PLANIFICADOR DE TRAYECTORIAS'.
%   'Uniones'
%*************************************************************************
```

```
ux=[];              % Coordenadas x de las uniones.
uy=[];              % Coordenadas y de las uniones.
ns=size(sx,1);         % Número de segmentos del diagrama.
ons=ones(ns,1);     % Vector de expansión.
int=zeros(ns,ns);      % Matriz auxiliar de comparación.
a11=(sy(:,1)-sy(:,2))./(sx(:,1)-sx(:,2));   % y=a11*x+b11 (segmentos).
b11=sy(:,1)-a11.*sx(:,1);                   % y=a11*x+b11 (segmentos).
dx=[v(1)*ons,(sx(:,1)+sx(:,2))/2];   % comp x de línea desde v al punto medio.
dy=[v(2)*ons,(sy(:,1)+sy(:,2))/2];   % comp y de línea desde v al punto medio.
a12=(dy(:,1)-dy(:,2))./(dx(:,1)-dx(:,2));   % y=a12*x+b12 (líneas).
b12=dy(:,2)-a12.*dx(:,2);                   % y=a12*x+b12 (líneas).
a1=a11*ons';        % Expansión de a11 (por columnas).
b1=b11*ons';        % Expansión de b11 (por columnas).
a2=ons*a12';        % Expansión de a12 (por filas).
b2=ons*b12';        % Expansión de b12 (por filas).
ix=(b2-b1)./(a1-a2);% Comp x de intersección segmentos y líneas v-punto medio.
iy=a1.*ix+b1;       % Comp y de intersección segmentos y líneas v-punto medio.
for i=1:ns
   ix(i,i)=dx(i,2); % Se ajusta el valor exacto de la diagonal.
   iy(i,i)=dy(i,2); % Se ajusta el valor exacto de la diagonal.
end
sx1=sx(:,1)*ons';   % Componente x del primer vértice de los segmentos.
sx2=sx(:,2)*ons';   % Componente x del segundo vértice de los segmentos.
sy1=sy(:,1)*ons';   % Componente y del primer vértice de los segmentos.
sy2=sy(:,2)*ons';   % Componente y del segundo vértice de los segmentos.
dx1=ons*dx(:,1)';   % Comp x del primer vértice de las líneas v-punto medio.
dx2=ons*dx(:,2)';   % Comp x del segundo vértice de las líneas v-punto medio.
dy1=ons*dy(:,1)';   % Comp y del primer vértice de las líneas v-punto medio.
dy2=ons*dy(:,2)';   % Comp y del segundo vértice de las líneas v-punto medio
int1=int+1.*[(ix>sx1 & ix<=sx2) | (ix>sx2 & ix<=sx1)];% punto en segmento.
int2=int+1.*[int1==1 & ( (iy>sy1 & iy<=sy2) | (iy>sy2 & iy<=sy1) )];  % segm.
int3=int+1.*[int2==1 & ( (ix>dx1 & ix<dx2) | (ix>dx2 & ix<dx1) )];    % línea.
int4=int+1.*[int3==1 & ( (iy>dy1 & iy<dy2) | (iy>dy2 & iy<dy1) )]; % válidos.
[i,j]=find(sum(int4)'==0);    % Líneas sin puntos entre v y el punto medio.
nu=2*length(i);               % Número de uniones.
j=1;                          % SELECCIÓN DE LAS UNIONES DE SALIDA.
if nu>0
   ux1=[dx(i,1),sx(i,1)];                 % Componente x uniones v-vértices.
   uy1=[dy(i,1),sy(i,1)];                 % Componente y uniones v-vértices.
   ux1=[ux1;[dx(i,1),sx(i,2)]];           % Componente x uniones v-vértices.
   uy1=[uy1;[dy(i,1),sy(i,2)]];           % Componente y uniones v-vértices.
   while nu>0
      ucx=ux1(:,2)-ux1(j,2).*ones(nu,1);  % Comprobación uniones redundantes.
      ucy=uy1(:,2)-uy1(j,2).*ones(nu,1);  % Comprobación uniones redundantes.
      ux=[ux;ux1(1,:)];                   % Comp x de uniones de salida.
      uy=[uy;uy1(1,:)];                   % Comp y de uniones de salida.
      [i1,j1]=find(ucx~=0 | ucy~=0);% Uniones diferentes a la registrada.
      nu=length(i1);                % Uniones que quedan por evaluar.
      if nu>0
         ux1=ux1(i1,:);             % Comprobación uniones redundantes.
         uy1=uy1(i1,:);             % Comprobación uniones redundantes.
      end
   end
end
```

A.1.2.2.2. El coste de los segmentos de Voronoi: la función ptvc.m.

La función *ptvc.m* recibe a la entrada la posición de las amenazas registradas (obstáculos, demás UAVs), las coordenadas *x* e *y* de los vértices extremos de los segmentos de Voronoi, y el peso de cada uno de los costes individuales (de longitud y de exposición al peligro).

El coste de longitud es la distancia del segmento, el de exposición al peligro depende de las distancias de las amenazas a los puntos intermedio, un sexto y cinco sextos del segmento de Voronoi. La salida es la matriz C, con tantas filas como segmentos y tres columnas correspondientes a los costes totales, de exposición al peligro y de longitud de los mismos.

```
function C=ptvc(P,sgx,sgy,ci)
%*************************************************************************
% MÓDULO: 'PLANIFICADOR DE TRAYECTORIAS'.
%  'Costes totales, de exposición al peligro y de longitud'
%*************************************************************************
% DECLARACIÓN DE VARIABLES:
p=size(P,1);     % Número de amenazas registradas en el sistema.
s=size(sgx,1);   % Número de segmentos del diagrama de Voronoi.
D=zeros(s,2);    % Matriz auxiliar para el cálculo de la matriz de costes.
C=zeros(s,3);     % Matriz de Costes de los segmentos [Ns*3].
ep=ones(1,p);    % Matriz generadora de la expansión para p columnas.
es=ones(s,1);     % Matriz generadora de la expansión para s filas.
% EVALUACIÓN DE LA MATRIZ DE COSTES.
S1x=sgx(:,1)*ep; % Coor x de los primeros vértices de los segmentos de Voronoi.
S2x=sgx(:,2)*ep; % Coor x de los segundos vértices de los segmentos de Voronoi.
S1y=sgy(:,1)*ep; % Coor y de los primeros vértices de los segmentos de Voronoi.
S2y=sgy(:,2)*ep; % Coor y de los segundos vértices de los segmentos de Voronoi.
Px=es*P(:,1)';   % Coor x de las amenazas (columnas) expandidas por filas.
Py=es*P(:,2)';   % Coor y de las amenazas (columnas) expandidas por filas.
Sx=S1x+S2x;      % Suma de comp x de los vértices extremos de los segmentos.
Sy=S1y+S2y;      % Suma de comp y de los vértices extremos de los segmentos.
S6x=(S2x-S1x)/6; % 1/6 de la diferencia de x de los vértices extremos.
S6y=(S2y-S1y)/6; % 1/6 de la diferencia de y de los vértices extremos.
S16x=S1x+S6x;       % Coordenada x del punto en el segmento a 1/6 del UAV.
S16y=S1y+S6y;       % Coordenada y del punto en el segmento a 1/6 del UAV.
S12x=Sx/2;          % Coord x del punto medio del segmento de Voronoi al UAV.
S12y=Sy/2;          % Coord y del punto medio del segmento de Voronoi al UAV.
S56x=S1x+5*S6x;     % Coordenada x del punto en el segmento a 5/6 del UAV.
S56y=S1y+5*S6y;     % Coordenada y del punto en el segmento a 5/6 del UAV.
C(:,3)=sqrt((S1x(:,1)-S2x(:,1)).^2+(S1y(:,1)-S2y(:,1)).^2);     % Longitud.
E=((S16x-Px).^2+(S16y-Py).^2).^(-2); % E=Distancias amenazas al punto 1/6.
E=E+((S12x-Px).^2+(S12y-Py).^2).^(-2);% E=E+Distancias amenazas al punto 1/2.
E=E+((S56x-Px).^2+(S56y-Py).^2).^(-2);% E=E+Distancias amenazas al punto 5/6.
C(:,2)=C(:,3).*sum(E')';          % Coste de Exposición al Peligro.
C(:,1)=ci*C(:,3)+(1-ci)*C(:,2);   % CosteTotal=CosteLongitud+CostePeligro.
```

A.1.2.3. La búsqueda de las K rutas: la función ptkuav.m.

Se trata del segundo fichero del Planificador de Trayectorias al que acceder para escoger las opciones que se permiten en la búsqueda de las rutas entre el origen y el destino. Hay que optar por el algoritmo de Dijkstra (muy costoso computacionalmente) o el método de Eppstein que es más eficiente (lenguaje C).

```
function [Tv,Lt,Ct,Cv,Lv]=ptkuav(bk,k,orig,dest,arcs,C)
%*************************************************************************
% MÓDULO: 'PLANIFICADOR DE TRAYECTORIAS'.
%   'K Rutas'
%*************************************************************************
```

```
switch bk
    case 'dj'                       % Si se selecciona el algoritmo de Dijkstra.
        sgx=arcs(:,1:2);                        % Coordenada x de los segmentos.
        sgy=arcs(:,3:4);                        % Coordenada y de los segmentos.
        v=orig;                                 % Coordenadas x, y del UAV.
        o=dest;                                 % Coordenadas x, y del objetivo.
        [Tv,Cv]=ptkd(C(:,1),sgx,sgy,v,o,k);  % Algoritmo de Dijkstra.
    case 'ep'                       % Si se selecciona el método de Eppstein.
        ng=arcs;                                % Nodos extremos de los segmentos.
        s=orig;                                 % Nodo correspondiente al UAV.
        t=dest;                                 % Nodo correspondiente al objetivo.
        [err]=ptkf(0,ng,C(:,1),s,t);% Se registra el grafo en grafo.gr.
        disp ('Ejecutar EPPSTEIN')
        disp ('Pulse cuando lo haya ejecutado')
        pause
        Tv=ptkf(1,1,1,1,1);         % Se lee el fichero rep.dat.
        ntv=size(Tv);               % Tamaño de la matriz de trayectorias.
        for i=1:ntv(1)              % Se recorren todas las filas.
            for j=1:ntv(2)-1            % Se recorren todas las columnas.
                [i1,j1]=find(ng(:,1)==Tv(i,j) & ng(:,2)==Tv(i,j+1));
                if length(i1)>0
                    Tv(i,j)=i1(1);   % cambia en primer nodo por número de arco.
                else
                    Tv(i,j)=t+1;     % Si no hay nodo (cero), no se asigna arco.
                end
            end
            Tv(i,j+1)=t+1;          % El último elemento nunca es un segmento.
        end
    otherwise                       % Error. Opción de búsqueda incorrecta.
        return
end
C=[C;zeros(1,3)];   % Elementos de Tv que no correspondan a segmentos reales.
if Tv(1,1)~=0
    Lt=zeros(size(Tv)); % Coste de longitud de los segmentos de la salida.
    Ct=zeros(size(Tv)); % Costes de exposición al peligro de salida.
    for j=1:k
        Lt(j,:)=C(Tv(j,:),3)';      % CL de los segmentos de las K rutas.
        Ct(j,:)=C(Tv(j,:),2)';      % CP de los segmentos de las K rutas.
    end
    Lv=sum(Lt')';% Coste de longitud de las K rutas calculadas para el UAV_i.
    Cv=sum((Lt+Ct)')';  % Coste total de las k rutas calculadas para el UAVi.
else
    Lt=0;               % En el caso de error total.
    Ct=0;
    Cv=0;
    Lv=0;
end
```

A.1.2.3.1. La gestión de los ficheros Eppstein: la función ptkf.m.

```
function [s1]=ptkf(op,ng,lg,s,t)
%*************************************************************************
% MÓDULO: 'PLANIFICADOR DE TRAYECTORIAS'.
%   'Escritura del fichero grafo.gr o lectura del resultado rep.dat'
%*************************************************************************

if op==0
    n=t;                    % Número de nodos.
    m=size(ng,1);           % Número de arcos.
    lg2=ceil(lg*10^4);      % Longitudes de los arcos.
    arcs=[ng';lg2'];        % Los nodos y la longitud de cada arco del grafo.
    err=1;                  % Si no se genera el fichero se devuelve 1.
    if n<0
        return                  % Error (salir de la función).
```

```
    elseif m<0
        return                  % Error (salir de la función).
    elseif ( s<0 | s>n )
        return                  % Error (salir de la función).
    else
        err=0;                          % No se han producido errores.
        fid=fopen('grafo.gr','w');      % Se abre el fichero grafo.gr.
        fprintf(fid,'n %d \n',n);       % Se escribe el número de nodos del grafo.
        fprintf(fid,'m %d \n',m);       % Se escribe el número de arcos del grafo.
        fprintf(fid,'s %d \n',s);       % Se escribe el origen de la búsqueda.
        fprintf(fid,'t %d \n',t);       % Se escribe el destino de la búsqueda.
        fprintf(fid,'a %d %d %d \n',arcs);   % Información de los arcos.
        fclose(fid);                    % Se cierra el fichero grafo.gr.
    end
    s1=err;                         % Se devuelve el error.
elseif op==1
    fid=fopen('rep.dat','r');       % Lectura en el fichero rpt.dat.
    k=fscanf(fid,'%d',[1,1]);       % Lectura de K.
    k1=fscanf(fid,'%d',[1,1]);      % Lectura del número realmente computado.
    if k1<=0
        s1=0;                       % Error fatal.
        fclose(fid);
        return
    end
    Tv=[];                          % Se inician los datos.
    nc=0;
    nf=1;
    for i=1:k1
        ne=fscanf(fid,'%d',[1,1]); % número de elementos de la ruta.
        ti=fscanf(fid,'%d',[1,ne]);% Vector con los números de nodos.
        if (ne>nc & nc>0)
            Tv=[[Tv,zeros(nf,ne-nc)];ti]; % Se rellena la matriz ya existente.
        elseif ne<nc
            Tv=[Tv;[ti,zeros(1,nc-ne)]];  % Se rellena la nueva fila.
        else
            Tv=[Tv;ti];                   % Tienen igual longitud.
        end
        tam=size(Tv);                     % Nueva matriz de trayectorias.
        nc=tam(2);
        nf=tam(1);
    end
    fclose(fid);                          % Se cierra el fichero rep.dat.
    if k1<k                               % Si se comutan menos rutas.
        Tv=[Tv;[ones(k-k1,1)*Tv(nf,:)]];  % Se rellena con la última fila.
    end
    s1=[Tv,zeros(size(Tv,1),1)];          % Se añade una última fila.
else
    s1=1;                                 % Error fatal.
end
```

A.1.2.3.2. El grafo dirigido para el método de Eppstein: ptkg.m-

```
function [ns,nn,s,sgx,sgy]=ptkg(v,sx,sy,C)
%*************************************************************************
% MÓDULO: 'PLANIFICADOR DE TRAYECTORIAS'.
%   'Grafo dirigido'
%*************************************************************************
n=size(sx,1);    % Número de segmentos del diagrama (arcos).
ns=zeros(n,2);   % Se reserva el espacio para los nodos estremos de los arcos.
nx=zeros(n,1);   % Coordenada x de los nodos.
ny=zeros(n,1);   % Coordenada y de los nodos.
sgx=zeros(n,2);  % Se reserva el espacio para las coordenadas dirigidas.
sgy=zeros(n,2);  % Se reserva el espacio para las coordenadas dirigidas.
nn=0;            % Índice de nodos.
```

```
mm=n;            % Número de arcos del grafo.
for is=1:mm      % Se recorren los arcos.
   xl=sx(is,1);
   yl=sy(is,1);
   i1=find(sx==xl & sy==yl & ns(is,1)==0 );       % Arcos confluyentes.
   if ( length(i1)>0 ) % Si hay arcos confluyentes.
      nn=nn+1;      % Índice de los nodos se incrementa.
      ns(i1)=nn;    % Se graba en las posiciones de la tabla de nodos.
      nx(nn)=xl;    % Coordenada x del primer vértice correspondiente al nodo.
      ny(nn)=yl;    % Coordenada y del primer vértice correspondiente al nodo.
      lxy(nn)=C(is,1); % Costes de los arcos afectados por el nodo nn.
   end
end
i2=find(ns==0 );        % Se detectan los nodos que no han sido numerados.
n1=length(i2);          % Número de nodos no detectados.
if ( n1>0 )
   for is=1:n1          % Se recorren los nodos no numerados.
      xl=sx(i2(is));       % Coordenada x del nodo.
      yl=sy(i2(is));       % Coordenada y del nodo.
      i3=find(  sx==xl & sy==yl & ns(i2(is))==0 );   % arcos confluyentes.
      if ( length(i3)>0 ) % El nodo no puede estar fuera.
         nn=nn+1;         % Se incrementa el valor del índice de nodos.
         ns(i3)=nn;       % Se le asigna el valor nn en la tabla de nodos.
         nx(nn)=xl;       % Componente x del primer nodo del arco.
         ny(nn)=yl;       % Componente y del segundo nodo del arco.
      end
   end
end
n=nn;                              % Número de nodos del grafo.
m=mm;                              % Número de arcos del grafo.
i1=find(sx==v(1) & sy==v(2));      % Nodo origen de la búsqueda.
if (length(i1)>0)                  % Si se encuentra un origen.
   s=ns(i1(1));                    % Número de nodo origen.
else
   s=0;                            % Si se produce un error.
   return
end
long=C(:,1);
A=Inf*ones(nn,nn);                 % Matriz de costes de los arcos entre nodos.
B=zeros(nn,nn);                    % Matriz del sentido de los arcos.
for i=1:nn                         % Se recorren todos los nodos.
   [i1,j1]=find(ns==i);               % arcos que confluyen en el arco i.
   n1=ns(i1,1);                    % Se rescata el número (primer nodo).
   n2=ns(i1,2);                    % Se rescata el número (segundo nodo).
   ln=long(i1);                    % Longitudes de los arcos confluyentes.
   na=length(i1);                  % Número de arcos confluyentes.
   ac=zeros(na,1);                 % Vector para formar la matriz A y B.
   ac=ac+n1.*[j1==2]+n2.*[j1==1];% orden de los nodos del arco.
   A(i,ac)=ln';                    % Longitud del arco.
   B(i,ac)=ln';                    % Longitud del arco.
end
N=size(A,1);                       % Número de nodos.
S=zeros(1,N);    % Conjunto de nodos registrados, al comienzo sólo s.
a3=zeros(1,N);   % Vector auxiliar para el algoritmo.
% ALGORITMO DE DIJKSTRA (entre el origen y todos los nodos del grafo).
S(s) = 1;                          % Se registra el origen.
B(:,s)=zeros(N,1);                 % Se impide el regreso a través de otro arco.
Dist=A(s,:);            % Se registra la Distancia de s a todos los nodos.
for i=1:N-1;                       % Se recorre el resto de nodos del grafo.
   cw=Inf;                            % Máximo coste posible.
   for k=1:N;                         % Para el resto de nodos
      if (S(k)==0 & Dist(k)<cw)  % Si k no incluido en S y Dist(k)<cw.
         cw=Dist(k);   % Se sustituye el coste por la longitud del nodo.
         p=k;             % Se incluye en el conjunto de nodos registrados.
      end
   end
   S(p)=1;                         % Se inserta el nodo s en el conjunto S.
   for k=1:N;                         % Se recorren todos los nodos.
```

```
        if (S(k)==0)                %Para todo nodo k que no se encuentre en S.
            if (Dist(p)+A(p,k)<Dist(k))    % Si mejora el coste registrado.
                Dist(k)=Dist(p)+A(p,k); % Se registra el nuevo coste.
            end
            B(k,p)=0;     % Se impide la vuelta a s a través de estos nodos.
        end
    end
end
[i,j]=find(B~=0);       % Posiciones de la matriz de sentido del arco.
sgx(:,1)=nx(i);      % Componente x de los nodos ordenados.
sgy(:,1)=ny(i);      % Componente y de los nodos ordenados.
sgx(:,2)=nx(j);      % Componente x de los nodos ordenados.
sgy(:,2)=ny(j);      % Componente y de los nodos ordenados.
ns=[i,j];         % Matriz de nodos de los segmentos del diagrama.
```

A.1.2.3.3. Un algoritmo básico de búsqueda de K rutas: ptkd.m.

Es un código muy tedioso y extenso. No se adjunta en el presente documento, pero se pone a disposición del Departamento de Ingeniería de Sistemas y Automática.

A.1.3. El Director de Asignación.

Se trata de un módulo cuya ejecución se realiza en los dos subsistemas con que cuenta el sistema de múltiples UAVs. Por una parte en el sistema central se lleva a cabo la decisión de conjunto, tras evaluar la función objetivo de asignación para cada asignación. En los diferentes UAVs se cuenta con parte del Director de Asignación encargado de evaluar la decisión individual acerca de los objetivos presentes en el escenario.

A.1.3.1. El Director de Asignación central: la función dacen.m.

Se encarga de realizar las llamadas a las funciones que evalúan la función objetivo de una asignación y reciben a través del director de comunicaciones la matriz de las decisiones individuales de los diferentes UAVs. A continuación se muestra el código fuente de *dacen.m*. Es el fichero de control y configuración del Director de Asignación en el sistema central.

```
function [err]=dacen(Mlp)
%*************************************************************************
% MÓDULO: 'DIRECTOR DE ASIGNACIÓN'.
%   'Configuración y Control del Director de Asignación Central'
%*************************************************************************
```

```
ai=1;     % Criterio de mínima longitud frente a mínima exposición al peligro.
bi=1*10^0;% Relación entre criterios globales (3 y 4) y los locales (1 y 2).
ei=3;     % Máximo número de UAVs por objetivo: ei=3 (6 UAVs) ei=4 (7 UAVs).
% DECLARACIÓN DE VARIABLES:
n=size(Mlp,1);   % Número de UAVs en el escenario de trabajo.
m=size(Mlp,2);   % Número de objetivos que se quiere cubrir.
S0=zeros(n,m);   % Matriz de asignación de UAVs a objetivos [Nv*No].
err=0;           % Error fatal (en principio ninguno).
mx=max(Mlp,[],2)*ones(1,m);% Máximos de decisiones individuales para cada UAV.
if max(mx)<0
   err=1;
   return
end
mn=min(Mlp,[],2)*ones(1,m); % Mínimos de decisiones individuales para UAV.
S0=(1.*[Mlp>=mx & mx>=0])+0;% A cada objetivo se asigna el UAV de máximo Mlp.
while max(diag(S0*S0'))~=1    % Si un UAV está asignado a más de un objetivo.
   i=find(diag(S0*S0')==max(diag(S0*S0')));    % Detección de los casos.
   [ik,k]=find(S0(i,:)==1);   % Se buscan las posiciones de los UAVs.
   Saux=S0'*ones(n,1);        % Se guarda la asignación inicial.
   [j,jk]=find([Saux(k)]==min([Saux(k)])); % Objetivos con menos UAVs.
   S0(i(1),:)=0;              % Se elimina las posiciones redundantes.
   S0(i(1),k(j(1)))=1;        % Se asignan a los objetivos menos cubiertos.
end
A0=S0*[1:m]';% Asignación primitiva (número de objetivo para cada UAV [Nv]).
s=dadc(bi,ei,S0,Mlp);   % Función Objetivo de la asignación primitiva.
s0=s;                   % Se inicia el registro de funciones objetivo.
c0=s;                   % Se inicia el registro del valor del cambio.
i0=1;                   % Se inicia el registro de la posición UAV del cambio.
j0=1;                   % Se inicia el registro del objetivo del cambio.
Clp=(mx-Mlp).*[Mlp>=0 & S0==0]+2.*[Mlp<0 | S0==1];    % Matriz de cambios.
[iv,jm]=min(Clp,[],2);  % Primer cambio: posiciones objetivo.
[im,jx]=find(iv~=2);    % Segundo cambio: posiciones UAV.
jm=jm(im);              % Posición UAV de los cambios siguientes válidos.
iv=iv(im);              % Valor de cambio válido.
nc=length(im);          % Número de cambios válidos.
while (nc>0)            % Mientras existan nuevas asignaciones para evaluar.
   na=length(c0);       % Número de asignaciones registradas.
   iaux=[];
   caux=[];
   saux=[];             % Se borran los registros auxiliares.
   for i=1:nc              % Se evalúan los cambios.
      S00=S0;              % Nueva matriz de asignación S00.
      S00(im(i),:)=zeros(1,m);% Se borra la asignación previa al cambio.
      S00(im(i),jm(i))=1;     % Se asigna el UAV al objetivo según el cambio.
      s00=dadc(bi,ei,S00,Mlp);% Se evalúa la función objetivo de asignación.
      if s00>=s            % Si mejora el valor de la función objetivo previo.
         c0=[s00,c0];      % Valor del cambio.
         i0=[im(i),i0];    % Posición UAV del cambio.
         j0=[jm(i),j0];    % Posición objetivo del cambio.
         s0=[s00,s0];              % Valor objetivo de asignación del cambio.
         A0=[S00*[1:m]',A0];     % Asignación del cambio.
      else
         caux=[caux,iv(i)];      % Cambios de las asignaciones que no mejoran.
         saux=[saux,s00];        % Valores objetivos inferiores al primitivo.
         iaux=[iaux,i];             % Número de cambio que no mejora.
      end
   end
   n1=length(c0)-na;    % Número de cambios efectuados.
   if n1>0
      [vn,jn]=max(c0); % Asignación primitiva para la siguiente iteración.
      c0(jn)=0;        % Se impide evaluar la asignación primitiva.
      inc=i0(jn);      % Posición UAV del nuevo cambio (asignación primitiva).
      jnc=j0(jn);      % Posición objetivo cambio nuevo(asingación primitiva).
   elseif (length(saux)>0)    % Si no se han registrado nuevas asignaciones.
      [vn,jn]=max(saux);      % Mínimo cambio determina el siguiente.
      inc=im(iaux(jn));% Posición UAV del nuevo cambio (asignación primitiva).
      jnc=jm(iaux(jn));% objetivo nuevo cambio (asignación primitiva).
   end
```

```
    Clp(inc,jnc)=2;  % Se evita volver a evaluar la nueva asignación primitiva.
    S0(inc,:)=zeros(1,m);   % Se borra  UAV-objetivo previa al cambio.
    S0(inc,jnc)=1;          % Se asigna el UAV al objetivo según el cambio.
    [iv,jm]=min(Clp,[],2); % Posición objetivo de los cambios siguientes.
    [im,jx]=find(iv~=2);    % Posición UAV de los cambios siguientes válidos.
    jm=jm(im);              % Posición UAV de los cambios siguientes válidos.
    iv=iv(im);              % Valor de cambio válido.
    nc=length(im);       % Número de cambios siguientes.
end
[s00,j]=sort(-s0);          % Se ordena de mayor a menor valor función objetivo.
A=A0(:,j);                  % Se comunican las asignaciones en ese orden.
na=size(A,2);               % Número de asignaciones calculadas.
if na>0                     % Si se dispone de al menos una asignación.
    fid=fopen('rda.dat','w');  % Se abre el fichero rda.dat (sistema ficheros).
    fprintf(fid,'%d \n%d \n',na,n);% Número de asignaciones realizadas.
    for j=1:na
        fprintf(fid,'%d ',A(:,j)); % En cada fila una asignación.
        fprintf(fid,'\n');
    end
    fclose(fid);            % Se cierra el fichero rda.dat.
else
    err=1;                  % Si no hay asignaciones realizadas, error.
end
```

A.1.3.1.1. La evaluación de la función objetivo: la función dadc.m.

```
function s=dadc(bi,ei,S0,Mlp)
%*************************************************************************
% MÓDULO: 'DIRECTOR DE ASIGNACIÓN'.
%   'Decisión colectiva (evaluación función objetivo)'
%*************************************************************************
m=size(S0,2);% Número de objetivos presentes en el sistema.
mc=zeros(1,m);% Objetivos cubiertos por algún UAV. Máxima Cobertura.
Slp=(Mlp.*[S0==1])+(2.*[S0==0]);  % mlp si el UAV se asigna al objetivo, 2 eoc.
si=min(min(Slp));    % Se considera el máximo de la fila en Slp (cada UAV).
me=sum(S0);          % Número de UAVs asignados a cada objetivo. [1*No].
Sme=1./(1+exp(-ei*(me-2)));% Función de Máxima Eficacia evaluada en objetivos.
sm0=1./(1+exp(-ei*(0-2))); % Valor del objetivo cuando no tiene ningún UAV.
Sme=Sme.*[Sme~=sm0]+1.*[Sme==sm0];   %objetivos cubiertos por algún UAV.
sme=prod(Sme);   % Máxima Eficacia: producto de funciones de objetivos con UAVs
mc=mc+1.*[me~=0];    % Objetivos que se han cubierto por al menos un UAV.
smc=sum(mc);     % Máxima Cobertura: objetivos alcanzados por algún UAV.
sc=sme*smc;      % Decisión Colectiva: Máxima Eficacia y Máxima Cobertura.
s=sc+bi*si;      % FUNCIÓN OBJETIVO DE ASIGNACIÓN.
```

A.1.3.2. La decisión individual del UAV: la función dauav.m.

En el UAV se cuenta con la función *dauav.m* de configuración y control del Director de Asignación en el subsistema UAV y con la función *dadp.*m que calcula las distancias al peligro sólo en el caso en que se seleccione el método de las distancias.

```
function [err]=dauav(e1,e2,e3)
%*************************************************************************
% MÓDULO: 'DIRECTOR DE ASIGNACIÓN'.
%*************************************************************************
ai=1;% valoración de los criterios de Mínima longitud y exposición al peligro.
```

```
mdi='mt';
err=0;
switch mdi
    case 'md'        %MÉTODO DE LAS DISTANCIAS'
        v=e1;
        O=e2;
        P=e3;
        m=size(O,1); % Número de objetivos en el sistema.
        p=size(P,1); % Número de amenazas registradas.
        E=ones(1,m); % Matriz de evaluación de la Exposición al peligro.
        L=zeros(1,m);% CRITERIO DE SELECCIÓN DE MÍNIMA LONGITUD.
        Dux=v(1)*ones(1,m); % Comp x del UAV_i (en todas las columnas).
        Duy=v(2)*ones(1,m); % Comp y del UAV_i (en todas las columnas).
        Dox=O(:,1)';                          % Componentes x de los objetivos.
        Doy=O(:,2)';                          % Componentes y de los objetivos.
        D1=sqrt((Dux-Dox).^2+(Duy-Doy).^2); % Longitudes UAV y los objetivos.
        D1mn=min(D1,[],2);  % Mínimos de las distancias UAV-objetivo.
        D1mx=max(D1,[],2);  % Máximos de las distancias UAV-objetivo.
        if (D1mx-D1mn~=0)
            L=(D1mx-D1)./(D1mx-D1mn);  % Mínima Long [0(inaceptable)1(óptimo)].
        end             % CRITERIO DE MÍNIMA EXPOSICIÓN AL PELIGRO.
        D2=dadp(v,O,P);  % Distancias al peligro UAV a todos los objetivos [No].
        D2mn=min(D2,[],2);  % Mínima distancia al peligro UAV_i a los objtivos.
        D2mx=max(D2,[],2);  % Máxima distancia al peligro UAV_i a los objtivos.
        if (D2mx-D2mn~=0)   % Se asegura que no se produzca división por cero.
            E=(D2-D2mn)./(D2mx-D2mn);  % 0(buena)...1(inaceptable).
        end
        mlp=L-ai*E;      % DECISIÓN INDIVIDUAL PARA CADA UAV
    case 'mt'  % MÉTODO DE EVALUACIÓN DE LAS RUTAS.
        I=[0;e1];           % Índice de los segmentos.
        Cl=e2;              % Costes de longitud de los segmentos de las rutas.
        Cp=e3;              % Costes de peligro de los segmentos de las rutas.
        m=size(e1,1);       % Número de objetivos en el escenario.
        k=size(Cl,1);       % Número de trayectorias por objetivo.
        mL=zeros(k,m);   % Coste de longitud medio de los segmentos por ruta.
        mP=zeros(k,m);   % Coste de peligro medio de los segmentos de cada ruta.
        L=zeros(1,m);    % Criterio de mínima longitud.
        E=ones(1,m);     % Criterio de mínima exposición al peligro.
        for j=1:m        % Para todos los objetivos del escenario.
            j1=sum(I(1:j))+1;   % Offset inicial de la matriz de costes.
            [iv,jv]=min(Cl(:,j1:j1+I(j+1)-1),[],2);        % CL
            mL(:,j)=sum(Cl(:,j1:j1+I(j+1)-1)')'./(jv-1);   % Media CL.
            mP(:,j)=sum(Cp(:,j1:j1+I(j+1)-1)')'./(jv-1); % Media CP.
        end
        mxL=max(mL,[],1);       % Máximo CL de las rutas de cada objetivo.
        mxP=max(mP,[],1);       % Máximo CP de las rutas de cada objetivo.
        mnL=min(mL,[],1);       % Mínimo CL de las rutas de cada objetivo.
        mnP=min(mP,[],1);       % Mínimo CP de las rutas de cada objetivo.
        mdL=mean(mL,1);      % CL medio de las rutas de cada objetivo.
        mdP=mean(mP,1);      % CP medio de las rutas de cada objetivo.
        if (mxL-mnL~=0)
            L=(mxL-mdL)./(mxL-mnL);% CRITERIO DE MÍNIMA LONGITUD.
        end
        if (mxP-mnP~=0)
            E=(mdP-mnP)./(mxP-mnP);% CRITERIO DE MÍNIMA EXPOSICIÓN AL PELIGRO.
        end
        mlp=L-ai*E;          % DECISIÓN INDIVIDUAL DEL UAV (para cada objetivo).
    otherwise
        m=0;
        mlp=0;
        err=1
        return
end
fid=fopen('rdai.dat','w'); % Se abre el fichero rdai.dat.
fprintf(fid,'%d \n',m);    % Número de objetivos.
fprintf(fid,'%4.14f ',mlp);% Decisión individual del UAV.
fprintf(fid,'\n');
fclose(fid);               % Se cierra el fichero rdai.dat.
```

A.1.3.2.1. Distancia al peligro para el Método de distancias: dadp.m

```
function d2=dadp(v,O,P)
%*************************************************************************
% MÓDULO: 'DIRECTOR DE ASIGNACIÓN'.
%
%*************************************************************************
m=size(O,1);     % Número de objetivos en el escenario de trabajo.
p=size(P,1);     % Número de amenazas en el escenario de trabajo.
v1=zeros(p,m);   % Pendiente de la proyección ortogonal sobre el segmento.
v2=zeros(p,m);   % Pendiente de la proyección ortogonal sobre el segmento.
nx=zeros(p,m);   % numerador para determinar la componente x de la proyección.
x=zeros(p,m);% x, proyección ortogonal sobre recta UAV-objetivo, desde amenaza
y=zeros(p,m);% y, proyección ortogonal sobre recta UAV-objetivo, desde amenaza
D=-ones(p,m);% Matriz para el tratamiento de la distancia del peligro.
Vx=v(1)*ones(p,m);  % Coordenada x de UAV (todas las columnas y filas).
Vy=v(2)*ones(p,m);  % Coordenada y del UAV (todas las columnas y filas).
Px=P(:,1)*ones(1,m);% Coordenada x de las amenazas (todas las columnas).
Py=P(:,2)*ones(1,m);% Coordenadas y de las amenazas (todas las columnas).
Ox=ones(p,1)*O(:,1)';% Coordenadas x de los objetivos (todas las columnas).
Oy=ones(p,1)*O(:,2)';% Coordenadas y de los objetivos (todas las columnas).
[ip,io]=find((Oy-Vy)~=0 & (Ox-Vx)~=0);  % Se evitan colineales con los ejes.
v1(ip,io)=(Oy(ip,io)-Vy(ip,io))./(Ox(ip,io)-Vx(ip,io));  % pendiente UAV-obj.
v2(ip,io)=(-Ox(ip,io)+Vx(ip,io))./(Oy(ip,io)-Vy(ip,io)); % pendiente UAV-obj.
nx(ip,io)=(Py(ip,io)-Vy(ip,io)-v2(ip,io).*Px(ip,io)+v1(ip,io).*Vx(ip,io));
x(ip,io)=nx(ip,io)./(v1(ip,io)-v2(ip,io));  % x: intersección proy-UAV-obj.
y(ip,io)=Py(ip,io) + v2(ip,io).*(x(ip,io)-Px(ip,io));% coordenada y.
x=x.*[(Oy-Vy)~=0]+Px.*[(Oy-Vy)==0];  % Si UAV-objetivo es vertical, x es Px.
y=y.*[(Oy-Vy)~=0]+Vy.*[(Oy-Vy)==0];  % Si UAV-objetivo es vertical, y es Vy.
x=x.*[(Ox-Vx)~=0]+Vx.*[(Ox-Vx)==0];  % Si UAV-objetivo es horizontal, x es Vx.
y=y.*[(Ox-Vx)~=0]+Py.*[(Ox-Vx)==0];  % Si UAV-objetivo es horizontal, y es Py.
Da=sqrt((Vx-Px).^2+(Vy-Py).^2);      % Distancias entre el UAV y las amenazas.
Db=sqrt((Ox-Px).^2+(Oy-Py).^2);      % Distancias objetivos y amenazas.
D=min(Da,Db).*[(Vx>x & Ox>x) | (Vx<x & Ox<x)]; % Si no intersecta uav-obj.
D=D+sqrt((x-Px).^2+(y-Py).^2).*[D==0];  % Distancias al peligro-amenazas.
d2=min(D,[],1);  % Distancias al peligro UAV-objetivos (amenaza de mínimo D).
```

A.1.4. El Interceptor Simultáneo.

Es un módulo con componentes en los dos subsistemas, tanto en el sistema central como en los diferentes UAVs. Se recuerda que hay propuestos dos métodos de interceptación simultánea de los objetivos, el método de escalado de velocidades lineales y el algoritmo de cooperación, éste último el más descentralizado. La decisión sobre la simultaneidad se toma en el sistema central en función de los cálculos desarrollados en los UAVs. Se estructura la descripción de los códigos fuentes según el lugar en el que vayan programados.

Tres son los algoritmos con que cuenta el Interceptor Simultáneo en el sistema central, el encargado de la configuración y el control de la ejecución del mismo por un lado, y otros dos que corresponden al algoritmo de interceptación simultánea utilizado (método de escalado de velocidades lineales o algoritmo de cooperación).

A.1.4.1. La sincronización global: la función iscen.m.

Desde este fichero se programa el modo de operación del Interceptor y se indica la velocidad mínima y máxima de los UAVs. Se especifica en el fichero el funcionamiento del Interceptor Simultáneo desde esta parte del sistema, invocando los algoritmos necesarios, y por último grabando en el archivo *uavtx3.dat* los resultados.

```
function [err]=iscen(a,Lr,Cr,op)
%*************************************************************************
% MÓDULO: 'INTERCEPTOR SIMULTÁNEO'.
%   'Control y Configuració del IS central'
%*************************************************************************
% CONFIGURACIÓN:
vmn=0.121;    % Velocidad mínima del vehículo aéreo autónomo en km/s.
vmx=0.141;    % Velocidad máxima del vehículo aéreo autónomo en km/s.
opc=2*op;
ac='ti';  %'ev'(escalado velocidad) 'ac'(algoritmo cooperación).
n=size(a,1); % Número de UAVs: Nv
tt=size(Lr,1);   % tt: número de trayectorias totales.
k=tt/n;          % K: Número de trayectorias por UAV.
err=0; % Error en el módulo: 1 (cuando intercepción no válida y opción 3).
c1=ones(n,1);% Según ac: 'ev'(velocidad lineal) o 'ac'(tig).
c2=ones(n,1);% Según ac: 'ev'(número de ruta) o 'ac'(sincronización).
switch ac
   case 'vl'   % MÉTODO DE ESCALADO DE VELOCIDADES LINEALES.
   [vl,nr,evl]=isvl(vmn,vmx,Lr,Cr,n);   % Sincronización global.
   sevl=sum(evl);   % Número de errores cometidos (UAVs con vl<vmn).
   if (sevl>0)      % Si hay UAVs con una velocidad no válida, corregir:
      if opc==0        % Si sólo se permite truncar la solución:
         vl=vl+vmn.*[evl==1];    % velocidad mínima a los UAVs seleccionados.
      elseif opc==1           % Si sólo se permite simultanear localmente:
         objevl=zeros(n,1)+a.*[evl~=0];    % objetivos con UAVs vl<vmn.
         [iv,in]=min(objevl,[],1);     % Se selecciona el primero de ellos.
         while iv~=0
            [i1,j1]=find(a==iv);% UAVs que cubren el objetivo.
            ni1=length(i1);     % Número de UAVs en la sincronización local.
            if (ni1>0)
               lr=[];
               cr=[];
               for i=1:ni1         % Se prepara la información.
                  lr=[lr;Lr((i1(i)-1)*k+1:i1(i)*k)];   % Longitudes locales.
                  cr=[cr;Cr((i1(i)-1)*k+1:i1(i)*k)];   % Costes locales.
               end
               [vg,ng,evg]=isvl(vmn,vmx,lr,cr,ni1); % Simultanear local.
               for j=1:ni1
                  vl(i1(j))=vg(j);     % Velocidad de los UAVs.
                  nr(i1(j))=ng(j);     % Números de ruta.
               end
            end
            objevl(i1)=0;       % Una vez conseguida la sincronización local.
            [iv,in]=min(objevl,[],1);  % Se selecciona el siguiente objetivo.
         end
         c1=vl; % La salida c1 para el caso 'ev', velocidades de cada UAV.
         c2=nr; % La salida c2 para el caso 'ev', número de ruta de cada UAV.
      elseif opc==2
         err=1; % Si opc==2, hay que probar otra asignación (err=1).
      else          % En otro caso, la opción introducida es incorrecta:
         c1=vmx*c1;% Velocidad máxima a costa de ausencia de sincronización.
      end
   else      % Si no errores, se envía a los diferentes UAVs el resultado.
      c1=vl;  % La salida c1 para el caso 'ev', velocidades de cada UAV.
```

```
      c2=nr;  % La salida c2 para el caso 'ev', número de ruta de cada UAV.
   end
case 'ti'   % METODO DE LOS TIEMPOS DE INTERCEPTACIÓN.
   TIO1=Lr/vmx;  % Tiempo mínimo del intervalo de intercepción por ruta.
   TIO2=Lr/vmn;  % Tiempo máximo del intervalo de intercepción por ruta.
   [TIG,eTIG]=isti(TIO1,TIO2,Cr,n);  % Tiempo de Interceptación Global (TIG).
   neTIG=sum(eTIG); % Número de errores (TIG exterior a los intervalos TIO).
   if (neTIG~=0)    % Si hay intervalos temporales que no incluyen el TIG.
      if opc==0     % Si  truncar al valor de la velocidad mínima:
         c1=TIG*c1;    % Se envía el TIG calculado.
         c2=0*c2.*[eTIG==0]-1*c2.*[eTIG==1];  %0(simultáneo) -1(pérdida).
      elseif opc==1  % Si sólo se permite simultanear localmente (opc==1):
         c1=TIG*c1;    % objetivos no afectados por UAVs quedan igual.
         c2=0*c2;  % Se consideran sincronizados.
         objTIG=zeros(n,1)+a.*[eTIG~=0];   % Objetivos (sincronización local).
         [iv,in]=max(objTIG,[],1);  % Se selecciona el primero de ellos.
         while iv~=0
            [i1,j1]=find(a==iv);% UAVs que atacan al objetivo en curso.
            ni1=length(i1);     % Número de UAVs en la interceptación local.
            if (ni1>0)          % Información.
               tio1=[];
               tio2=[];
               cr=[];
               for i=1:ni1
                  tio1=[tio1;TIO1((i1(i)-1)*k+1:i1(i)*k)];% Tiempos mínimos
                  tio2=[tio2;TIO2((i1(i)-1)*k+1:i1(i)*k)];% Tiempos máximos
                  cr=[cr;Cr((i1(i)-1)*k+1:i1(i)*k)];      % Costes
               end
               [TIL,eTIL]=isti(tio1,tio2,cr,ni1);   % Sincronización local
               neTIL=sum(eTIL); % Número de errores cometidos (UAVs v<vmn).
               c1(i1)=TIL;      % La salida c1 contiene en  i1 el TIL.
               c2(i1)=iv;       % máximo coste para los UAVs del objetivo.
            end
            objTIG(i1)=0;% El objetivo sincronizado localmente por los UAVs.
            [iv,in]=max(objTIG,[],1);% Siguiente objetivo local.
         end
      elseif opc==2
         err=1; % Cuando opc==2, se requiere una nueva asignación: err=1.
      else   % Si la opción introducida es errónea. Pérdida de sincronización.
         c1=TIO1(1:k:tt-k+1).*c1;   % Tiempos mínimos para cada UAV.
         c2=-c2;                    % c2(i)=-1. Ausencia de sincronización.
      end
   else   % Si sincronización global de todos los UAVs en los objetivos.
      c1=TIG*c1;    % Se envía a todos los UAVs el TIG.
      c2=0*c2;  % c2(i)=0: Sincronización global de todos los UAVs.
   end
otherwise     % Si no se ha introducido un algoritmo de interceptación válido.
   c1=0;
   c2=0;
   err=1;
end
if err==0     % Si se ha logrado el objetivo de la interceptación simultánea.
   fid=fopen('uavrx3.dat','w');      % Se abre el fichero rdi.dat.
   fprintf(fid,'%d \n',n);           % Número de UAVs que se han simultaneado.
   fprintf(fid,'%4.14f \n%4.14f \n',vmn,vmx);% Velocidad mínima y máxima.
   fprintf(fid,'%s \n',ac);          % Algoritmo utilizado.
   fprintf(fid,'%4.14f ',c1');       % Salida c1 (depende del tipo de ac.
   fprintf(fid,'\n');
   fprintf(fid,'%d ',c2');           % Salida c2 (depende del tipo de ac.
   fclose(fid);
end
```

A.1.4.1.1. El método de velocidades lineales: la función isvl.m.

```
function [vl,nr,evl]=isvl(vmn,vmx,Lr,Cr,n)
%*************************************************************************
% MÓDULO: 'INTERCEPTOR SIMULTÁNEO'.
%   ' Método de Velocidade Lineales'
%*************************************************************************
in=[1:n];        % Índice con los números de UAVs.
tt=size(Lr,1);   % Números de rutas totales de los Nv UAVs.
k=tt/n;          % Número de trayectorias por UAV.
mcr=max(Cr);     % Máximo valor de coste de ruta entre todas las trayectorias.
CI=zeros(tt,tt); % Coste de Intercepción al objetivo de cada ruta por contexto.
mCI=zeros(tt,n); % Mínimos costes de intercepción en los diferentes contextos.
iCI=zeros(tt,n); % Número de rutas que hacen mínimo el coste de intercepción.
Ve=vmx*(Lr.^-1)*Lr';    % Velocidad de la ruta en su contexto. Matriz [Nv*K].
Ce=ones(tt,1)*Cr';      % Se inicia el coste de las rutas al valor cr [Nv*K].
CI=Ce+(k*mcr+Ve-vmx).*[Ve>vmx]+(mcr+vmn-Ve).*[Ve<vmn];   % Se construye CI.
for j=1:n % Por filas, mínimo coste del bloque de k rutas de todos los UAVs.
   [mCI(:,j),iCI(:,j)]=min(CI(:,(j-1)*k+1:j*k),[],2);    % Mínimos CI de UAVs.
   mCI((j-1)*k+1:j*k,j)=diag(CI((j-1)*k+1:j*k,(j-1)*k+1:j*k)); % diagonales.
   iCI((j-1)*k+1:j*k,j)=[1:k]';   % Índice de los elementos diagonales.
end              % Función objetivo: sumar los costes de interceptación.
FO=sum(mCI')';   % Por filas (contextos), sumar los mínimos CI (columnas).
[CIG,c]=min(FO,[],1);   % CIG es el coste global y c el contexto más favorable.
vl=Ve(c(1),(in-1)*k+iCI(c(1),in))'; % vl [Nv]: velocidad lineal de los UAVs.
nr=iCI(c(1),:)';                    % nr [Nv]: número de ruta de los UAVs.
evl=zeros(size(vl));    % vector de errores cometidos (vl<vmn).
evl=evl+1.*[vl<vmn];    % Se comprueba que velocidades no excedan el mínimo.
vl=vl.*[evl==0]+vmn.*[evl==1];       % Si hay un error se trunca al valor Vmn.
```

A.1.4.1.2. El Método de los tiempos de interceptación: la función isti.m.

```
function [TIG,eTIG]=isti(TIO1,TIO2,Cr,n)
%*************************************************************************
% MÓDULO: 'INTERCEPTOR SIMULTÁNEO'.
%*************************************************************************
tt=size(Cr,1);   % Número de rutas totales.
k=tt/n;          % Número de rutas por UAV.
mxc=max(Cr);     % Máximo valor de coste (de entre todas las rutas).
mCI=zeros(tt,n); % Mínimos costes de intercepción en los diferentes contextos.
iCI=zeros(tt,n); % Número de rutas que hacen mínimo el coste de intercepción.
eTIG=zeros(n,1); % Vector de errores: si el TIG incumple alguno de los rangos
Ti1=TIO1*ones(1,tt);    % Tiempo mínimo de las rutas (para todas las columnas).
Ti2=TIO2*ones(1,tt);    % Tiempo máximo de las rutas (para todas las columnas).
T1=ones(tt,1)*TIO1';% Tiempo mínimo del intervalo de referencia (columnas).
T2=ones(tt,1)*TIO2';% Tiempo máximo del intervalo de referencia (columnas).
Ce=ones(tt,1)*Cr';  % Se inicia el coste de las rutas al valor recibido.
CI=Ce+(k*mxc+T1-Ti1).*[Ti1<T1]+(mxc+Ti1-T2).*[Ti1>T2]; % Costes de Intercep.
for j=1:n
   [mCI(:,j),iCI(:,j)]=min(CI(:,(j-1)*k+1:j*k),[],2);% mínimos CI de los UAVs.
   mCI((j-1)*k+1:j*k,j)=diag(CI((j-1)*k+1:j*k,(j-1)*k+1:j*k)); % diagonales.
   iCI((j-1)*k+1:j*k,j)=[1:k]';                                % diagonales.
end
FO=sum(mCI')';   % Por filas (contextos), se suman los mínimos CI en columnas.
[CIG,c]=min(FO,[],1);  % CIG es el coste mínimo y c el contexto más favorable.
TIG=TIO1(c);     % TIG corresponde a la posición c de TIO1.
mTIO1=TIO1([0:k:(n-1)*k]'+iCI(c,:)');   % Si TIG inferior al tiempo máximo.
mTIO2=TIO2([0:k:(n-1)*k]'+iCI(c,:)');   % Si TIG superior al tiempo mínimo.
eTIG=eTIG+1.*[mTIO1>TIG | mTIO2<TIG]; % Indicador de ERRORES.
```

A.1.4.2. El Interceptor Simultáneo local: la función isuav.m.

En el UAV se determina la velocidad lineal de recorrido y el número de trayectoria por la que transitar. Dependiendo del método seleccionado, las entradas contienen la información necesaria para el cálculo de estos parámetros.

```
function [err]=isuav(ac,vmn,vmx,c1,c2,lv,cv);
%*************************************************************************
% MÓDULO: 'INTERCEPTOR SIMULTÁNEO'.
%   'Selección de la velocidad y número de ruta en cada UAV'
%*************************************************************************
err=0;
switch ac
    case 'vl' % Método de Velocidades lineales:
        vl=c1; % c1(i) contiene la velocidad lineal del UAV_i.
        nr=c2; % c2(i) corresponde al número de ruta (1..K) del UAV_i.
    case 'ti'  % Método de los tiempos de interceptación:
        TIG=c1;    % c1(i) es el Tiempo de Interceptación Global del UAV_i.
        oi=c2; % Si sincronización local, objetivo que afecta al UAV.
        mxc=max(cv);         % máximo coste local.
        k=size(lv,1);        % El número total de rutas calculadas por UAV.
        TIO1=lv/vmx;  % Tiempo mínimo de intercepción (cada fila, una ruta) [K].
        TIO2=lv/vmn;  % Tiempo máximo de intercepción (cada fila, una ruta) [K].
        % CI: Función de coordinación (para cada ruta se evalúa el CI)
        CI=cv+(mxc+TIG-TIO2).*[TIO2<TIG]+(k*mxc+TIO1-TIG).*[TIO1>TIG];
        [mCI,iCI]=min(CI);  % mínimo CI para TIG determina la ruta seleccionada.
        nr=iCI(1);           % Índice indica el número de ruta del UAV_i: 1...K.
        vl=lv(nr)/TIG;       % Velocidad lineal: longitud de la ruta entre TIG.
    otherwise % Si no se introduce una opción válida, sin sincronización:
        vl=vmx;    % Se selecciona la velocidad máxima (factor sorpresa).
        nr=1;  % Se selecciona para el UAV_i la ruta más corta (menor coste).
end
```

A.1.5. El Generador de Trayectoria.

El funcionamiento del Generador de Trayectoria se produce íntegramente en el UAV. A partir de los resultados del Interceptor Simultáneo, y rescatando las rutas almacenadas por el Planificador de Trayectorias, se parametriza en el tiempo la ruta. La configuración y el control del módulo se encuentra en el fichero gtuav.m, desde donde se llaman a las funciones *gtat.m*, *gtpt.m* y al modelo de Simulink *gtf.mdl*.

A.1.5.1. El control y la configuración: la función gtuav.m.

Es el fichero al que se accede para indicar la máxima variación del ángulo del UAV. Desde el fichero se ordena calcular los datos relativos a transiciones entre segmentos en el fichero *gtat.m* y se planifica la ruta, parametrizándola en el tiempo, en la función *gtpt.m*. Por último se simula la trayectoria en el modelo en Simulink *gtf.mdl*.

```
%*************************************************************************
% MÓDULO: 'GENERADOR DE TRAYECTORIA'.
%*************************************************************************
```

```
umx=0.1941;   % Máxima variación de ángulo que se permite ejecutar al UAV.
tv=Ts(rv,:);  % Segmentos que componen la trayectoria de referencia.
lv=Ls(rv,:);  % Longitudes de los segmentos de la trayectoria de referencia.
[jj,j]=min(lv,[],2);% Segmentos que no pertenecen a trayectoria de referencia.
j=1:j-1;            % Índice de recorrido de los segmentos de la trayectoria.
tv=tv(j);           % Segmentos seleccionados.
lv=lv(j)';          % Longitudes seleccionadas.
n=size(lv,1);       % Número de segmentos.
tvx=sgx(tv,:);      % Componentes x de los segmentos de Voronoi.
tvy=sgy(tv,:);      % Componentes y de los segmentos de Voronoi.
for j=2:n           % Se recorren uno a uno los segmentos de la trayectoria.
   if tvx(j,1)~=tvx(j-1,2)    % Si no existe secuencialidad directa.
      b=[tvx(j,1) tvy(j,1)]; % Se redefine el sentido del arco.
      tvx(j,1)=tvx(j,2); tvy(j,1)=tvy(j,2);
      tvx(j,2)=b(1); tvy(j,2)=b(2);
   end
end
[g,tsm]=gtpt(vv,lv,tvx,tvy,umx); % Indicador de giro y Tiempos de simulación.
tsm=tsm.*[tsm>=0]+0.*[tsm<0];    % Corrección ante posibles fallos.
nsm=length(tsm);                 % Número de tramos de simulación de la ruta.
tio=sum(tsm);                    % Tiempo de interceptación al objetivo.
j=1;          % Índice de recorrido de los tramos de trayectoria de referencia.
xr=tvx(1,1);  % Condición de partida. Componente x de referencia.
yr=tvy(1,1);  % Condición de partida. Componente y de referencia.
vr=vv;        % Condición de partida. Velocidad lineal de referencia.
phr=ps(1);    % Condición de partida. Ángulo respecto al eje x de referencia.
tr=0;         % Instante inicial de la simulación.
while j<=nsm % Se recorren los segmentos de la trayectoria de referencia.
   u2=0;                         % Variación de la velocidad de referencia.
   tspan=[0:tsm(j)/500:tsm(j)]'; % Tiempo de simulación del filtro (fase).
   u1=g(j)*umx*ones(size(tspan));% u1, según g: 0(lineal) +-1(circular).
   options = simset('refine',10);% Opciones para la simulación del modelo.
   sim ('gtf',tspan);            % Simulación del filtro durante tspan.
   tam=size(xir);                % Número de muestras en los registros.
   fil=tam(1);                   % Número de filas.
   col=tam(2);                   % Número de columnas (1 si todo correcto).
   xr=[xr,xir(2:fil,col)'];      % Componente x de referencia del UAV.
   yr=[yr,yir(2:fil,col)'];      % Componente y de referencia del UAV.
   phr=[phr,pir(2:fil,col)'];    % Ángulo respecto a x de referencia.
   ntr=length(tr);               % Número de instantes ya registrados.
   tr1=tr(ntr)*ones(1,fil-1);    % Último instante de tiempo previo.
   tr=[tr,tr1+tout(2:fil)'];     % Se añaden los instantes de muestreo tr.
   j=j+1;       % Siguiente tramo de trayectoria (linear o circular).
end
nms=length(xr);  % Número de muestras disponibles de referencia.
if nms>0                         % Si se dispone de referencia generada.
   fid=fopen('rgt.dat','w');     % Se abre el fichero rgt.dat.
   fprintf(fid,'%d \n',i);       % Número de UAV.
   fprintf(fid,'%f \n',vr);      % Velocidad de referencia del UAV.
   fprintf(fid,'%f \n',phr(1));  % Fase de referencia del UAV.
   fprintf(fid,'%d \n',nms);     % Número de muestras registradas.
   fprintf(fid,'%4.14f ',tr);    % Trayectoria de referencia.
   fprintf(fid,'\n');
   fprintf(fid,'%4.14f ',xr);    % Trayectoria de referencia.
   fprintf(fid,'\n');
   fprintf(fid,'%4.14f ',yr);    % Trayectoria de referencia.
   fprintf(fid,'\n');
   fclose(fid);                  % Se cierra el fichero rgt.dat.
end
```

A.1.5.2. La adecuación de la trayectoria: la función gtat.m.

Esta función se encarga de determinar los ángulos de giro entre cada pareja de segmentos. Se determina la distancia del vértice a la que el UAV debe comenzar a transitar hacia el siguiente segmento y el sentido de giro.

```
function [dg,sg,ag,ps]=gtat(vv,lv,tvx,tvy,umx)
%*************************************************************************
% MÓDULO: GENERADOR DE TRAYECTORIA.
% 'Adecuación de la trayectoria'
%*************************************************************************
n=size(lv,1);    % Número de segmentos que componen la referencia.
j=1:n;           % vector índice del número de segmentos de la trayectoria.
sg=zeros(n-1,1); % Sentido de giro en las transiciones entre segmentos.
if n>1       % Si la trayectoria de referencia la forman más de un segmento.
   ps=atan((tvy(j,2)-tvy(j,1))./(tvx(j,2)-tvx(j,1)));% Ángulos respecto a x.
   ps(j)=ps(j)-pi.*[tvx(j,2)-tvx(j,1)<0];   % Corrección para ser -pi o pi.
   dp=ps(2:n)-ps(1:n-1);                    % Diferencia de fase.
   tx=tvx(:,2)-tvx(:,1);          % Diferencia en las componentes x.
   ty=tvy(:,2)-tvy(:,1);          % Diferencia en las componentes y.
   pex=-tx(1:n-1,:).*tx(2:n,:);   % Productos de componentes x consecutivos.
   pey=-ty(1:n-1,:).*ty(2:n,:);   % Productos de componentes y consecutivos.
   pe=sum([pex,pey],2);           % Producto escalar consecutivos.
   le=lv(1:n-1).*lv(2:n);         % Producto de los módulos consecutivos.
   ce=pe./le;    % Coseno de los ángulos que forman los segmentos consecutivos.
   ae=acos(ce);  % Ángulos que forman los segmentos anteriores entre [-pi,pi].
   am=ae/2;      % Ángulo que forma cada segmento con la bisectriz del ángulo.
   dg=(vv/umx)*(1./tan(am));      % distancia vértice-intersección.
   dg=[0;dg;0];  % Distancias al comienzo del giro, monitorización del proceso.
   pv=tx(1:n-1,1).*ty(2:n,1)-tx(2:n,1).*ty(1:n-1,1); % z, producto vectorial.
   sg=[0;sign(pv);0];         % Sentido de giro: >0 a izquierda, <0 a derecha.
   ag=[0;pi-ae]; % Variación de ángulo del UAV con respecto a la recta.
else
   ag=0;  % Sólo hay un segmento, no hay ángulos entre segmentos consecutivos.
   sg=0; % Por tanto no hay giros debido a transiciones entre segmentos.
   dg=0; % Tampoco hay distancias al comienzo del giro.
   ps=atan((tvy(1,2)-tvy(1,1))./(tvx(1,2)-tvx(1,1)));       % Fase de éste.
   ps=ps-pi.*[tvx(1,2)-tvx(1,1)<0];            % Corrección para ser -pi o pi.
end
```

A.1.5.3. La parametrización en el tiempo: el fichero gtpt.m.

Esta función determina el tiempo de simulación de cada uno de los tramos de la trayectoria de referencia, así como el control de los giros (detección de situaciones anómalas y sentido del movimiento en el giro).

```
function [g,tsm]=gtpt(vv,lv,tvx,tvy,umx)
%*************************************************************************
% MÓDULO: GENERADOR DE TRAYECTORIA.
%   'Monitorización del proceso'
%*************************************************************************
% DECLARACIÓN DE VARIABLES:
nsr=size(tvx,1);     % Número de segmentos que componen la referencia.
cg=ones(1,nsr+1);    % Control del giro en las distintas transiciones.
lg=[lv;0];           % Longitudes de los segmentos (añadir un cero).
tsm=[];              % Tiempo de interceptación al objetivo por parte del UAV.
j=1;                 % Índice de recorrido de los segmentos (primero).
% ADAPTACIÓN DE LA TRAYECTORIA A LAS CARACTERÍSTICAS DEL UAV:
[dg,sg,ag,ps]=gtat(vv,lv,tvx,tvy,umx);  % ángulo, distancia y sentido de giro.
```

```
g=0;   % La primera simulación corresponde a una porción del primer segmento.
while j<=nsr     % Del primero al último de los segmentos de la referencia.
   ng=length(g); % Número de elementos del vector g (tramos ya registrados).
   if g(ng)==0   % Si se tiene que recorrer una porción de segmento (lineal).
      if ( cg(j)*dg(j)+dg(j+1)<lg(j) & dg(j+1)<lg(j+1)) % Se autoriza el giro.
         t=(lg(j)-cg(j)*dg(j)-dg(j+1))/vv; % Tiempo de simulación lineal.
         tsm=[tsm,t];% Se añade el tiempo de simulación correspondiente.
         cg(j+1)=1; % Se autoriza recorrido circular (control del giro cg=1).
      else          % Si no se dispone de longitud para la distancia del giro.
         t=(lg(j)-cg(j)*dg(j))/vv;   % Tiempo de simulación lineal.
         tsm=[tsm,t];   % Se añade el tiempo de simulación correspondiente.
         cg(j+1)=0;     % No se puede autorizar movimiento circular (anomalía).
      end
      if ( cg(j+1)==1 & dg(j+1)~=0 ) % Se permite el movimiento cirular.
         g=[g,sg(j+1)];% Se toma el sentido de giro corespondiente.
      else
         g=[g,0,0];     % El siguiente será lineal a lo largo del segmento.
         tsm=[tsm,0];   % El siguiente será lineal a lo largo del segmento.
      end
      j=j+1;        % Se selecciona el siguiente segmento de la trayectoria.
    else            % Si el movimiento es circular para alcanzar el siguiente.
      t=ag(j)/umx;  % tsim para el recorrido circular: phi=w*tsm.
      tsm=[tsm,t];  % Se incorpora el tsim a la interceptación.
      g=[g,0];      % Tras recorrido circular se continúa con el segmento.
   end
end
g=g(1:ng);% Movimiento: 0 (lineal) 1 (circular a izqda) -1 (circular a drcha).
```

A.1.5.4. El modelo en Simulink del filtro no lineal: gtf.mdl.

A partir de la entrada *u1* que indica la variación del ángulo, el modelo genera las componentes X_{ir} , Y_{ir} y P_{ir}.

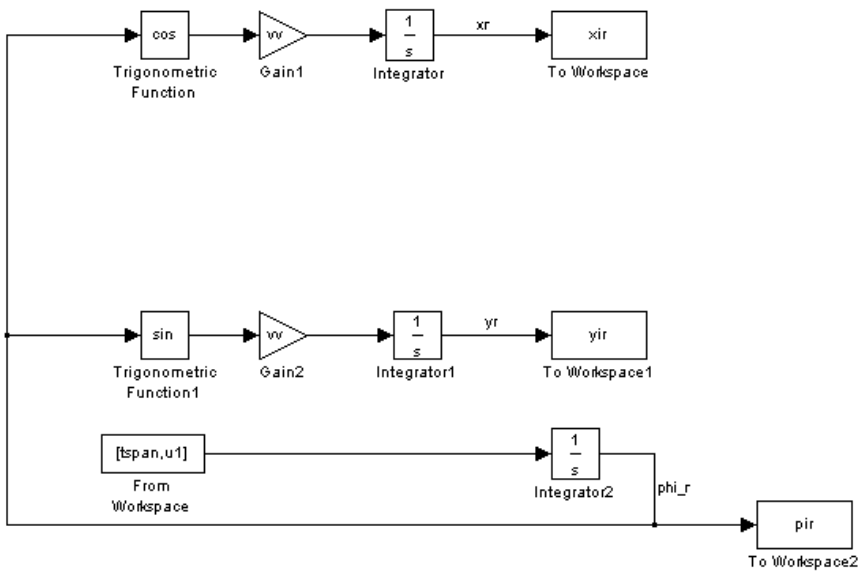

Figura 30: Modelo en Simulink del filtro no lineal *gtf.mdl*.

A.1.6. El Controlador del Sistema.

Se han desacoplado el control de las variables de los plano horizontal y vertical. El fichero *csuav.m* contiene las órdenes necesarias para simular el controlador completo. El modelo del controlador se guarda en el fichero cshv.mdl, un fichero Simulink que permite el seguimiento de la trayectoria de referencia en el tiempo de interceptación al objetivo calculado por el Interceptor Simultáneo.

A.1.6.1. La configuración del controlador: el fichero csuav.m.

El control del sistema tiene como misión generar las señales de mando que permitan el seguimiento de la trayectoria de referencia Se ajustan las constantes del controlador. A partir de las condiciones iniciales de la trayectoria de referencia se simula el modelo de manera que gracias a las señales de mando se obtiene una trayectoria de seguimiento muy fiel a la referencia.

```
%*************************************************************************
% 'CONTROLADOR DE LOS PLANOS HORIZONTAL Y VERTICAL'.
%*************************************************************************
ap=1.33;      % Parámetros del Controlador del Plano Horizontal.
av=0.2;
kz=2.75;      % Constantes del Controlador del Plano Horizontal.
ky=0.42;
kx=0.42;
ah=0.3364;    % Parámetros del Controlador del Plano Vertical.
ah1=1.4680;
kh=100;       % Constantes del Controlador del Plano Vertical.
kh1=15;
dxr=[]; dyr=[];     % Derivadas de x e y de la trayectoria de referencia.
d1hr=[]; d2hr=[];   % Derivada primera y segunda del perfil de altura del UAV.
xsi=[]; ysi=[]; psi=[]; vsi=[];   % Señales de seguimiento de la trayectoria.
hsi=[]; dhsi=[];    % Señales de seguimiento del perfil de altura.
vci=[]; pci=[]; vdi=[]; pdi=[];   % Señales de control V_c y Phi_c.
chi=[];                           % Señales de control H_c y H'_c.
hr=500+sin(xr);
dxr=diff(xr)./diff(tr);    % Derivada primera de la componente x de referencia.
dyr=diff(yr)./diff(tr);    % Derivada primera de la componente y de referencia.
dh1r=diff(hr)./diff(tr);   % Primera derivada del perfil de altura referencia.
dhr=[dh1r(1);dh1r];        % Primera derivada del perfil de altura referencia.
dh2r=diff(dhr)./diff(tr);  % Derivada segunda del perfil de altura referencia.
xs=xr(1);                  % Condición de partida de la componente xs.
ys=yr(1);                  % Condición de partida de la componente ys.
phs=phr(1);                % Condición de partida de la fase phs.
hs=hr(1);                  % Condición de partida de la altura hs.
dhs=dh1r(1);               % Condición de partida de la variación de altura.
vs=vr;                     % Condición de partida de la velocidad.
xr=xr(2:nm);     % Se elimina la primera componente x de referencia.
yr=yr(2:nm);     % Se elimina la primera componente y de referencia.
hr=hr(2:nm);     % Se elimina la primera componente h de referencia.
tr=tr(2:nm);     % Se elimina el primer instante de muestreo.
options = simset('refine',10);    % Opciones para la simulación del modelo.
```

```
sim ('cshv',timespan);  % SEGUIMIENTO DE LA TRAYECTORIA DE REFERENCIA.
tam=size(xsi);          % Recogida de los resultados.
xs=[xs,xsi(2:tam(1),tam(2))'];% Coordenada x de seguimiento de la referencia.
ys=[ys,ysi(2:tam(1),tam(2))'];% Coordenada y de seguimiento de la referencia.
phs=[phs,psi(2:tam(1),tam(2))'];  % Ángulo respecto al eje x de seguimiento.
vs=[vs,vsi(2:tam(1),tam(2))'];    % Velocidad de seguimiento de la referencia.
hs=[hs,hsi(2:tam(1),tam(2))'];    % Seguimiento del perfil de altura.
vc=vci(1:tam(1),tam(2))';         % Señal de control de la velocidad.
pc=pci(1:tam(1),tam(2))';  % Señal de control de la fase.
hc=hci(2:tam(1),tam(2))';  % Señal de control de la altura.
vd=vdi(1:tam(1),tam(2))';  % Control interno de la velocidad de referencia.
phd=pdi(1:tam(1),tam(2))'; % Control interno de la fase de referencia.
```

A.1.6.2. El modelo simulink del controlador: cshv.mdl.

Se ha integrado en un mismo modelo los controladores de los planos horizontal y vertical, para ser simulados en los mismos instantes. Aún así se encuentran totalmente desacoplados: El controlador del plano horizontal cuenta con cuatro bucles realimentados para el control de las variables *X*, *Y, V, Ph.* El controlador del plano vertical controla el perfil de altura a través de la altura de referencia y la derivada. Este diagrama describe el contenido del fichero *cshv.mdl* que permite la determinación de las señales de control *Vc, Phc* y *Hc*.

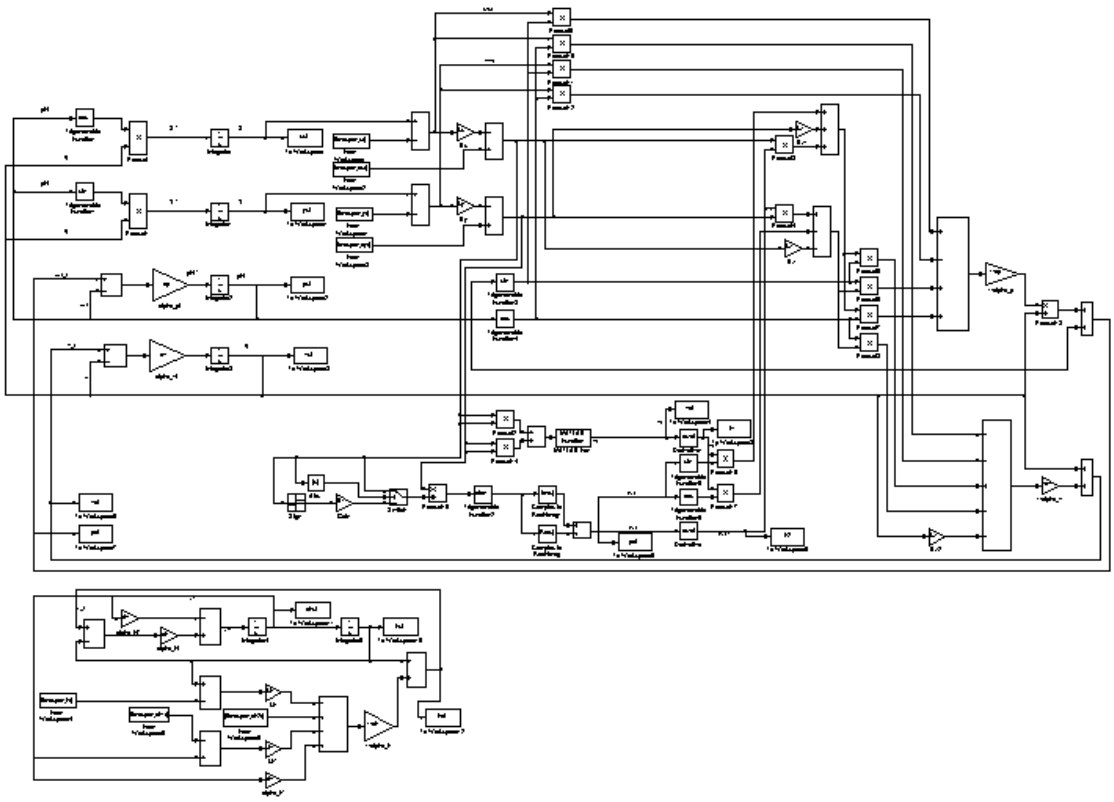

Figura 31: Modelo del controlador del sistema. *cshv.mdl*.

A.1.7. El Director de Comunicaciones.

En diversas situaciones se requiere el intercambio de información entre los subsistemas central y UAVs. Este flujo de información se produce en ambos sentidos en distintas fases de la ejecución del sistema debido a que un mismo módulo tenga componentes instalados en distintos subsistemas o que para la ejecución de uno se requiera información de otro módulo situado en un subsistema diferente. Se enumeran las comunicaciones según el sentido del flujo.

- ✓ Desde el sistema central hacia los UAVs:
  - ➢ Posición de los UAVs, obstáculos y objetivos para que el Planificador de Trayectorias pueda trazar el diagrama de Voronoi y aplicar Dijkstra.
  - ➢ Asignación proporcionada por el director de Asignación.
  - ➢ Información procedente del Interceptor Simultáneo central que al local le permite determinar la velocidad lineal y número de ruta.
- ✓ Desde los subsistemas UAVs hacia el sistema central.
  - ➢ Decisión individual tomada en función de los objetivos de mínima distancia y mínima exposición al peligro.
  - ➢ Longitudes de las rutas y costes de las mismas. De esta manera el Interceptor Simultáneo coordina las trayectorias de todos los UAVs para que las llegadas se produzcan en el mismo momento.

El Director de Comunicaciones cuenta con dos ficheros. Cada uno se ocupa de uno de los ámbitos, sistema central o UAV, encargándose de leer o escribir en los ficheros, dependiendo de que se produzca recepción o transmisión de datos hacia el otro subsistema.

A.1.7.1. La comunicación desde el sistema central: dccen.m.

En función de la variable de entrada *fun*, el interruptor llevará a ese caso que puede corresponder a la escritura en el fichero (transmisión hacia los UAVs) o a la lectura de un archivo (recepción de los datos procedentes de los UAVs). Las entradas y salidas se declaran genéricamente, y en cada una de las situaciones será descrito su contenido.

```
function [s1,s2]=dccen(fun,e1,e2,e3,e4);
% MÓDULO: 'DIRECTOR DE COMUNICACIONES'  SC
switch fun
case 'ctx1'   % Hacia el PLANIFICADOR DE TRAYECTORIAS-diagrama de Voronoi.
   V=e1;             % Entrada 1: posición de los UAVs.
   O=e2;             % Entrada 2: posición de los objetivos.
   P=e3;             % Entrada 3: posición de las amenazas.
   n=size(V,1);  % Número de UAVs.
   m=size(O,1);  % Número de objetivos.
   p=size(P,1);  % Número de amenazas.
   fid=fopen('uavrx1.dat','w');      % Primera recepción de UAVs: uavrx1.dat.
   fprintf(fid,'%d \n',n);           % Número de UAVs.
   if n>0
     fprintf(fid,'%4.14f ',V(:,1)); % 1ª fila: n componentes x de los UAVs.
     fprintf(fid,'\n');
     fprintf(fid,'%4.14f ',V(:,2)); % 2ª fila: n componentes y de los UAVs.
      fprintf(fid,'\n');
   end
   fprintf(fid,'%d \n',m);           % Número de objetivos.
   if m>0
     fprintf(fid,'%4.14f ',O(:,1)); % 1ª fila: m componentes x de objetivos.
     fprintf(fid,'\n');
     fprintf(fid,'%4.14f ',O(:,2)); % 2ª fila: m componentes y de objetivos.
      fprintf(fid,'\n');
   end
   fprintf(fid,'%d \n',p);           % Número de amenazas.
   if p>0
     fprintf(fid,'%4.14f ',P(:,1)); % 1ª fila: p componentes x de amenazas.
     fprintf(fid,'\n');
     fprintf(fid,'%4.14f ',P(:,2)); % 2ª fila: p componentes y de amenazas.
      fprintf(fid,'\n');
   end
   fclose(fid);     % A partir de este momento podrá ser leido por los UAVs.
   s1=0;            % Datos transmitidos.
case 'ctx2'   % TRANSIMISIÓN A LOS UAVs DE LOS OBJETIVOS ASIGNADOS.
   a=e1;            % El Director de Asignación, objetivos de cada UAV.
   n=size(a,1);     % Número de UAVs a los que se le han asignado objetivos.
   fid=fopen('uavrx2.dat','w');   % Segunda recepción de los UAVs: uavrx2.dat.
   fprintf(fid,'%d \n',n);        % Número de UAVs.
   fprintf(fid,'%d ',a);          % 1ª fila: por UAV, objetivo asignado.
   fprintf(fid,'\n');
   fclose(fid);  % A partir de ahora los UAVs pueden leer el objetivo asignado.
   s1=0;         % Información transimitida hacia los UAVs.
case 'crx1'   % RECEPCIÓN DE LA SITUACIÓN DE LOS UAVs, OBJETIVOS Y AMENAZAS.
   n=e1;                          % Número de UAVs.
   fid=fopen('cenrx1.dat','r');   % Primera recepción: fichero cenrx1.dat.
   n1=fscanf(fid,'%d',[1,1]);     % Posición de los UAVs [Nv*2].
   if n1~=n
       return
   end
   m=fscanf(fid,'%d',[1,1]);      % Posición de los objetivos [No*2].
   s1=fscanf(fid,'%f',[n,m]);     % Posición de las amenazas [Np*2].
   fclose(fid);  % A partir de este momento se conoce el escenario de trabajo.
case 'crx2'   % RECEPCIÓN DE LONGITUDES Y COSTES (Interceptor Simultáneo).
   n=e1;                          % Número de UAVs.
    fid=fopen('cenrx2.dat','r');  % Segunda recepción: fichero cenrx2.dat.
   n1=fscanf(fid,'%d',[1,1]);     % UAVs que han comunicado los resultados.
   if n1~=n      % Si la información no es completa, salir.
       s1=0; s2=0;
      return
   end
   k=fscanf(fid,'%d',[1,1]);      % Número de rutas registradas.
   s1=[]; s2=[];
   for i=1:n
       s11=fscanf(fid,'%f',[1,k]);% Se recibe la longitud de cada ruta.
      s1=[s1;s11'];               % Primera Recepción: vector longitud.
      s22=fscanf(fid,'%f',[1,k]); % Se recibe el coste de la ruta.
      s2=[s2;s22'];               % Segunda Recepción: vector coste.
```

```
    end
    fclose(fid);  % El Interceptor Simultáneo central puede comenzar.
otherwise
    s1=1;
end
```

A.1.7.2. La comunicación desde el UAV: el fichero dcuav.m.

```
function [s1,s2,s3,s4,s5,s6]=dcuav(fun,e1,e2,e3,e4,e5);
% MÓDULO: 'DIRECTOR DE COMUNICACIONES'  UAV
global M1;
global C1;
global L1;
switch fun
    case 'utx1'   % Hacia el Director de Asignación del Sistema Central.
       i=e1;      % Número de UAV que comunica la decisión individual.
       mlp=e2;           % Decisión individual tomada.
       m=length(mlp);    % Número de objetivos en el sistema.
        M1(i,:)=mlp;     % Se inserta en la variable global para todos los UAVs.
       f=min(sum(abs(M1)')',[],1);       % Comprobación del número de UAVs.
       if f~=0           % Si se dispone de todas las decisiones individuales.
          fid=fopen('cenrx1.dat','w');   % Se abre el fichero cenrx1.dat.
          fprintf(fid,'%d \n%d \n',n,m); % Número de UAVs y de objetivos.
          for j=1:n
           fprintf(fid,'%4.14f ',M1(j,:)');% En cada fila, decisión individual.
             fprintf(fid,'\n');
          end
           fclose(fid); % Se cierra el fichero cenrx1.dat.
       end        % cenrx1.dat contiene en la fila i el vector de m elementos.
       s1=0;
   case 'utx2'   % Longitud y costes hacia Interceptor Simultáneo central.
       i=e1;             % Número de UAV que desea escribir en el fichero.
       Lv=e2;            % Longitud de las rutas planificadas en el UAV_i.
       Cv=e3;            % Coste de las rutas planificadas en el UAV_i.
       k=size(L1,1);     % Número de rutas por cada UAV.
       L1(:,i)=Lv;       % Se inserta en la variable global Lr la Lv del UAV_i.
       C1(:,i)=Cv;       % Se inserta en la variable global Cr la Cv del UAV_i.
       f=min(sum(abs(C1)),[],2);   % Comprobación del número UAVs registrados.
       if f~=0           % Si se dispone de todas las longitudes y costes.
          fid=fopen('cenrx2.dat','w');   % Se abre el fichero cenrx2.dat.
          fprintf(fid,'%d \n%d \n',n,k); % Número de UAVs y de rutas.
          for i=1:n      % Se recorren uno a uno los UAVs.
             fprintf(fid,'%4.14f ',L1(:,i)');   % Primera fila, longitudes.
             fprintf(fid,' \n');                % Siguiente fila.
             fprintf(fid,'%4.14f ',C1(:,i)');   % Costes del UAV.
             fprintf(fid,' \n');                % Siguiente decisión individual.
          end
          fclose(fid);   % Se cierra el fichero cenrx1.dat.
       end    % cenrx2.dat, en las filas 2*i-1, 2*i las k longitudes y costes.
       s1=0;
   case 'utx3'   % Se indica al Sistema central que hay trayectoria referencia.
       i=e1;
       fid=fopen('cenrx3.dat','w');
       fprintf(fid,'uav %d \n',i);
       fclose(fid);
   case 'urx1'
       i=e1;
       fid=fopen('uavrx1.dat','r');   % Lectura en el fichero uavrx1.dat.
       n1=fscanf(fid,'%d',[1,1]);     % Lectura del número de UAVs.
       if n1~=n      % Si los datos recibidos no son los esperados, salir.
          fclose(fid);
          return
       end
       if n1>0
```

```
        v1=fscanf(fid,'%f',2*n);   % Posición de los UAVs [Nv*2].
        s1=[v1(1:n),v1(n+1:2*n)];
    else
        s1=[];
    end
    m=fscanf(fid,'%d',[1,1]);      % Número de objetivos.
    if n1>0
        o1=fscanf(fid,'%f',2*m);       % Posición de los objetivos [No*2].
        s2=[o1(1:m),o1(m+1:2*m)];
    else
        s2=[];
    end
    p=fscanf(fid,'%d',[1,1]);      % Número de amenazas.
    if p>0
        p1=fscanf(fid,'%f',2*p);       % Posición de las amenazas [No*2].
        s3=[p1(1:p),p1(p+1:2*p)];
    else
        s3=[];
    end
    fclose(fid);                   % Se cierra el fichero uavrx1.dat.
case 'urx2'
    i=e1;
    fid=fopen('uavrx2.dat','r');  % Lectura en el fichero uavrx2.dat.
    n1=fscanf(fid,'%d',[1,1]);    % Lectura del número de UAVs.
    if n1~=n      % Si los datos recibidos no son los esperados, salir.
        return
    end
    a=fscanf(fid,'%f',[1,n]);     % Objetivos asignados a cada UAV [Nv*1].
    fclose(fid);                  % Se cierra el fichero uavrx2.dat.
    s1=a(i);                      % Objetivo asignado al UAV_i.
case 'urx3'
    i=e1;
    fid=fopen('uavrx3.dat','r');  % Se abre el fichero.
    n1=fscanf(fid,'%d',[1,1]);    % Número de UAVs.
    if n1~=n      % Si los datos recibidos no son los esperados, salir.
        s1=1; s2=0; s3=0;
        s4=0; s5=0; s6=0;
        fclose(fid);
        return
    end
    s1=0;         % Sin error de operación del Interceptor Simultáneo.
    s2=fscanf(fid,'%f',[1,1]);    % Velocidad mínima para los UAVs.
    s3=fscanf(fid,'%f',[1,1]);    % Velocidad máxima de funcionamiento.
    s4=fscanf(fid,'%s',[1,1]);    % Algoritmo de Interceptación utilizado.
    c1=fscanf(fid,'%f',[1,n]);    % Salida c1 (depende de ais).
    c2=fscanf(fid,'%d',[1,n]);    % Salida c2 (depende de ais).
    s5=c1(i);
    s6=c2(i);
    fclose(fid);                  % Se cierra el fichero.
case 'urx4'
   i=e1;
   fid=fopen('rgt.dat','r');      % Se permite la lectura en rgt.dat.
   i1=fscanf(fid,'%d',[1,1]);     % Lectura del número de UAVs.
   if i~=i1      % Si los datos recibidos no son los esperados, salir.
      fclose(fid);
      return
   end
   s1=fscanf(fid,'%f',[1,1]);     % Velocidad de referencia.
   s2=fscanf(fid,'%f',[1,1]);     % Velocidad de referencia.
   s3=fscanf(fid,'%d',[1,1]);     % Número de muestras.
   s4=fscanf(fid,'%f',s3);        % Instantes de muestreo de la trayectoria.
   s5=fscanf(fid,'%f',s3);        % Trayectoria de referencia, componente x.
   s6=fscanf(fid,'%f',s3);        % Trayectoria de referencia, componente y.
   fclose(fid);
otherwise        % Si la opción no es válida, salir de la función:
   return
end
```

A.1.7.3. El Sistema de ficheros.

A.1.7.3.1. El sistema de ficheros del sistema central.

```
function [s1,s2,s3,s4,s5]=sfcen(mod,e1,e2,e3,e4)
% ************************************************************************
% MÓDULO: 'GESTIÓN DEL SUBSISTEMA CENTRAL'
% ACCESO AL SISTEMA DE FICHEROS DEL SISTEMA CENTRAL.
%*************************************************************************
switch mod
   case 'da'
      n=e1;                          % Número de UAVs.
      s1=0;                          % Error iniciado a cero.
      fid=fopen('rda.dat','r');      % Se abre el fichero.
      na=fscanf(fid,'%d',[1,1]);     % asignaciones realizadas (na columnas).
      n1=fscanf(fid,'%d',[1,1]);     % Número de UAVs en el sistema (n1 filas).
      if n1~=n         % Si no hay registrados Nv UAVs, error en el sistema.
         s1=1;         % Se devuelve err=1;
         s2=0;
         fclose(fid);
         return                      % Acaba el funcionamiento.
      end
      s2=fscanf(fid,'%f',[n,na]);    % Si no hay error se rescata la matriz A.
      fclose(fid);                % Se cierra el fichero.
   case 'is'
      n=e1;
      fid=fopen('ris.dat','r');      % Se abre el fichero.
      s2=fscanf(fid,'%s',[1,1]);     % Algoritmo de Interceptación utilizado.
      s3=fscanf(fid,'%f',[1,1]);     % Velocidad mínima para los UAVs.
      s4=fscanf(fid,'%f',[1,1]);     % Velocidad máxima de funcionamiento.
      c1=fscanf(fid,'%f',[1,n]);     % Salida c1 (depende de ac).
      c2=fscanf(fid,'%d',[1,n]);     % Salida c2 (depende de ac).
      s5=c1';
      s6=c2';
      fclose(fid);                   % Se cierra el fichero.
   otherwise
      return
end
```

A.1.7.3.2. El sistema de ficheros UAV: la función sfuav.m.

```
function [s1,s2,s3,s4,s5,s6]=sfuav(mod,e1,e2)
%*************************************************************************
% MÓDULO: 'GESTIÓN DEL SUBSISTEMA UAV'
% ACCESO AL SISTEMA DE FICHEROS DEL UAV.
%*************************************************************************
switch mod
   case 'da'     % MATRICES DE COSTES DE LONGITUD Y PELIGRO.
      m=e1;
      fid=fopen('rpt.dat','r');   % lectura en el fichero rpt.dat.
      m1=fscanf(fid,'%d',[1,1]);  % Lectura del número de objetivos.
      if m1~=m
         fclose(fid);
         return
      end
      k=fscanf(fid,'%d',[1,1]);   % Número de rutas pot vehículo.
      nt=fscanf(fid,'%d',[1,1]);  % Número de segmentos de todas las rutas.
      lr=fscanf(fid,'%f',m*k);    % Número de muestras.
```

```
    cr=fscanf(fid,'%f',m*k);
    s1=fscanf(fid,'%d',m);      % Índice.
    tr=fscanf(fid,'%f',[nt,k]);% Matriz con las longitudes de los caminos.
    cl=fscanf(fid,'%f',[nt,k]);% Matriz con las longitudes de los caminos.
    cp=fscanf(fid,'%f',[nt,k]);% Matriz con las costes de los caminos.
    s2=cl'; s3=cp';
    fclose(fid);
case 'dai'    % DECISIÓN INDIVIDUAL DEL UAV.
    m=e1;                       % Número de objetivos calculados.
    fid=fopen('rdai.dat','r');  % lectura en el fichero rdai.dat.
    m1=fscanf(fid,'%d',[1,1]);  % Lectura del número de objetivos.
    if m1~=m
       s1=0;
       fclose(fid);
       return
    end
    mlp=fscanf(fid,'%f',m);     % Decisión individual del UAV.
     s1=mlp';                   % s1: m columnas.
case 'is'     % LONGITUDES Y COSTES TOTALES DE LAS RUTAS AL OBJETIVO.
    j=e1;
    fid=fopen('rpt.dat','r');   % lectura en el fichero rpt.dat.
    m=fscanf(fid,'%d',[1,1]);   % Lectura del número de objetivos.
    if j>m
       fclose(fid);
       return
    end
    k=fscanf(fid,'%d',[1,1]);   % Número de rutas para cada objetivo.
     io=(j-1)*k+1:j*k;          % Selección de los resultados para obj j.
    nt=fscanf(fid,'%d',[1,1]);  % Número de segmentos de todas las rutas.
    lr=fscanf(fid,'%f',m*k);    % Longitud de las trayectorias de o.
    cr=fscanf(fid,'%f',m*k);    % Costes de las trayectorias de o.
    s1=lr(io);
    s2=cr(io);
    fclose(fid);
case 'gt'     % SUCESIÓN DE SEGMENTOS DE LA RUTA Y VELOCIDAD DE REFERENCIA.
    j=e1;                          % Objetivo seleccionado.
    m=e2;                          % Número de objetivos.
    fid=fopen('ris.dat','r');   % lectura en el fichero rpt.dat.
    nr=fscanf(fid,'%d',[1,1]);  % Lectura del número de la ruta..
    s1=fscanf(fid,'%f',[1,1]);  % Velocidad lineal de referencia.
    fclose(fid);
    fid=fopen('rpt.dat','r');   % lectura en el fichero rpt.dat.
    m1=fscanf(fid,'%d',[1,1]);  % Lectura del número de objetivos.
    if (m1~=m | j>m1)           % Si objetivo no está almacenado, salir.
       fclose(fid);
       return
    end
    k=fscanf(fid,'%d',[1,1]);   % Número de rutas para cada objetivo.
    if (s1>k)                   % Si la ruta no está almacenada, salir.
       fclose(fid);
       return
    end
     nt=fscanf(fid,'%d',[1,1]); % Número de segmentos de todas las rutas.
     lr=fscanf(fid,'%f',m*k);   % longitudes de las rutas.
     cr=fscanf(fid,'%f',m*k);   % costes de las rutas.
     ir=fscanf(fid,'%d',m);     % Índice de los segmentos.
     tr=fscanf(fid,'%f',[nt,k]);% Matriz con las longitudes de los caminos.
     fclose(fid);               % Se cierra el fichero ris.dat.
     I=[0;ir];                  % Índice de segmentos.
     j1=sum(I(1:j))+1;          % Offset inicial de la matriz de segmentos.
     j2=sum(I(1:j+1));             % Offset final de la matriz de segmentos.
     tvr=tr(j1:j2,nr);             % Segmentos de la ruta seleccionada.
     [j1,j2]=max(tvr);             % Segmentos fuera del rango admitido.
     tv=tvr(1:j2-1);            % Sólo se admiten segmentos registrados.
     fid=fopen('rdv.dat','r');  % lectura en el fichero rpt.dat.
     ns=fscanf(fid,'%d',[1,1]); % Lectura del número de segmentos.
     sg=fscanf(fid,'%f',[ns,4]);% Velocidad lineal de referencia.
     cg=fscanf(fid,'%f',[ns,3]);% Velocidad lineal de referencia.
```

```
        nu=fscanf(fid,'%d',[1,1]); % Lectura del número de uniones.
        iu=fscanf(fid,'%d',m);     % Lectura del número de uniones.
        ug=fscanf(fid,'%f',[nu,4]);% Velocidad lineal de referencia.
        cu=fscanf(fid,'%f',[nu,3]);% Velocidad lineal de referencia.
        fclose(fid);               % Se cierra el fichero rdv.dat.
        I=[0;iu];                  % Índice de uniones.
        j1=sum(I(1:j))+1;          % Offset inicial de la matriz de segmentos.
        j2=sum(I(1:j+1));             % Offset final de la matriz de segmentos.
        sgx=[[sg(:,1),sg(:,2)];[ug(j1:j2,1),ug(j1:j2,2)]];    % componentes x.
        sgy=[[sg(:,3),sg(:,4)];[ug(j1:j2,3),ug(j1:j2,4)]];    % componentes y.
        Cl=[cg(:,3);cu(j1:j2,3)];  % Costes de las longitudes.
        s2=sgx(tv,:);              % Comp x de los segmentos de la trayectoria.
        s3=sgy(tv,:);              % Comp y de los segmentos de la trayectoria.
        nr=length(tv);             % Número de tramos de la trayectoria.
        for j=2:nr                    % segmentos de la trayectoria.
            if s2(j,1)~=s2(j-1,2)  % Se redifine el sentido del arco.
               b=[s2(j,1) s3(j,1)];
               s2(j,1)=s2(j,2); s3(j,1)=s3(j,2);
               s2(j,2)=b(1); s3(j,2)=b(2);
            end
        end
        s4=Cl(tv,:);               % Longitudes de los segmentos.
    case 'cs'     % TRAYECTORIA DE REFERENCIA.
        i=e1;
        fid=fopen('rgt.dat','r');  % lectura en el fichero rgt.dat.
        i1=fscanf(fid,'%d',[1,1]); % Lectura del número de UAVs.
        if i~=i1                   % Si los datos recibidos erróneos.
            fclose(fid);
            return
        end
        s1=fscanf(fid,'%f',[1,1]);     % Velocidad de referencia.
        s2=fscanf(fid,'%f',[1,1]);     % Velocidad de referencia.
        s3=fscanf(fid,'%d',[1,1]);     % Número de muestras.
        s4=fscanf(fid,'%f',s3);        % Instantes de muestreo de la trayectoria.
        s5=fscanf(fid,'%f',s3);        % Trayectoria de referencia, componente x.
        s6=fscanf(fid,'%f',s3);        % Trayectoria de referencia, componente y.
        fclose(fid);
    otherwise
        return
end
```

A.1.8. La Búsqueda de las K trayectorias.

A.1.8.1. La función principal del programa EPPSTEIN.

```
#define VERSION "1.1"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#ifdef DEBUG
#include <assert.h>
#include <string.h>
#endif
#include <EPPSTEIN.h>
#include <loadgraph.h>
#include <dijkstraRev.h>
#include <chronometer.h>
```

```
typedef struct DevPath {
  COST_TYPE cost;
  struct DevPath *parent;      /* Elemento anterior en la secuencia */
  struct Heap3Node *heap3;     /* Último elemento de la secuencia */
} DevPath;     /* camino se representa por secuencia de elementos del árbol de caminos más cortos     */
typedef struct DevPath *PtrDevPath;

/* variables globales */
Heap3Node *heap3;
Heap3Node *firstFreeHeap3Node;
int heap3Size;
DevPath    *devPaths;
DevPath    *firstFreeDevPath;
PtrDevPath *searchHeap;
int searchHeapSize = 0;
PtrDevPath *bestDevPaths;
int numberComputedPaths = 0;
PtrNode *stack;
PtrDevPath * traceBackStack;
float *cumulatedSeconds;

Heap3Node * Heap3Malloc() {       /* reserva de espacio en memoria */

  Heap3Node * oldFirst = firstFreeHeap3Node;
#ifdef DEBUG
  if (firstFreeHeap3Node - heap3 > heap3Size - 1) {
    printf("ERROR FATAL: No existen nodos libres en heap3\n");
    exit(1);
  }
#endif
  firstFreeHeap3Node++;
  return oldFirst;
}

/* FUNCIÓN: HeapifiyOutHeap */
void HeapifyOutHeap (Heap3Node *heap, int heapSize) {
  register int i;
  for (i = heapSize/2; i >= 1; i--) {/* Desde heap[1] a heap[heapSize], se modifica sólo campos de dato */
    Heap3Data item = heap[i].data;
    register int j = 2*i;
    while (j <= heapSize) {
      if (j+1 <= heapSize && heap[j+1].data.cost < heap[j].data.cost) j += 1;
      if (item.cost <= heap[j].data.cost) break;
      heap[j/2].data = heap[j].data;
      j = 2*j;
    }
    heap[j/2].data = item;
  }
}

/* FUNCIÓN: BuildOutHeap */
void BuildOutHeap (Node *node) {
/* Construye conjuntos binarios de lados G-T cuyo fin está en el nodo, */
/* restricción de que el sucesor del conjunto tenga sólo un hijo
  PtrArc *arc;                  /* El peso de cada lado e se denota delta(e) */
  int numberArcs;
  Heap3Data auxData;
  Heap3Node *bestPos, *pos, *father, *child;
  COST_TYPE bestCost;
  int heapSize = 0;
#ifdef DEBUG
  assert (node!=NULL);
#endif
  if (node->bestCost >= INFINITY_COST) {
    node->heap3 = NULL;
    node->heap3Done = 1;
    return;
```

```
  }
  bestCost = INFINITY_COST;
  bestPos = NULL;
  for (arc = node->firstArcOut, numberArcs = node->numberArcsOut;
       numberArcs != 0; arc++, numberArcs--) {
    if ( (*arc)->dest != node->bestPath && (*arc)->dest->bestCost < INFINITY_COST) {
      int delta = (*arc)->cost + (*arc)->dest->bestCost - node->bestCost;
      if (bestPos == NULL) node->heap3 = pos = bestPos = Heap3Malloc ();
      else pos = Heap3Malloc ();
      heapSize++;
      pos->data.cost = delta;
      pos->data.arc  = *arc;
      if (delta < bestCost) {
            bestCost = delta;
            bestPos  = pos;
      }
    }
  }
  if (heapSize==0) {
    node->heap3 = NULL;
    node->heap3Done = 1;
  } else {
    if (bestPos != node->heap3) {
      auxData = bestPos->data;
      bestPos->data = node->heap3->data;
      node->heap3->data = auxData;
    }
    node->heap3[0].left = node->heap3[0].right = NULL;
    if (heapSize > 1) node->heap3[0].center = node->heap3 + 1;
    else
      node->heap3[0].center = NULL;
    for (father=node->heap3 + 1, child=node->heap3 + 2; child < node->heap3 + heapSize; father++) {
      father->left = child++;
      father->right = (child < node->heap3 + heapSize) ? child++ : NULL;
      father->center = NULL;
    }
    for ( ; father < node->heap3 + heapSize; father++)
      father->left = father->right = father->center = NULL;
    if (heapSize > 2)
      HeapifyOutHeap (node->heap3, heapSize-1); /* Heapify from node->heap3[1] */
  }
}

/* FUNCIÓN: MergeHeap3 */
Heap3Node * MergeHeap3(Heap3Node * outHeap, Heap3Node * devHeap) {

  Heap3Node * newNode;   /* Insertar el camino de outHeap en devHeap asegurando la continuidad */
  if (devHeap == NULL) return outHeap;
  if (outHeap == NULL) return devHeap;
  if (outHeap->data.cost <= devHeap->data.cost) {
    if (devHeap->desc == 0) {
      outHeap->left = devHeap;
      outHeap->desc = 1;
      return outHeap;
    }
    newNode = Heap3Malloc();
    newNode->data = devHeap->data;
    newNode->left = newNode->right = NULL;
    newNode->center = devHeap->center;
    newNode->desc = 0;
    if (devHeap->right!=NULL && devHeap->right->desc > devHeap->left->desc) {
      outHeap->left = MergeHeap3(newNode, devHeap->left);
      outHeap->right = devHeap->right;
    }
    else {
      outHeap->left = devHeap->left;
      outHeap->right = MergeHeap3(newNode, devHeap->right);
```

```
    }
    outHeap->desc = devHeap->desc + 1;
    return outHeap;
  }
  else {
    newNode = Heap3Malloc();
    *newNode = *devHeap;
    if (newNode->left == NULL ||
        (newNode->right!=NULL && newNode->right->desc > newNode->left->desc)) {
      newNode->left = MergeHeap3(outHeap, newNode->left);
    }
    else {
      newNode->right = MergeHeap3(outHeap, newNode->right);
    }
    newNode->desc++;
    return newNode;
  }
}

/* FUNCIÓN: BuildDeviationHeaps */
void BuildDeviationHeaps (Graph * graph) {

  int  numberNodes;
  Node *node, *node2;
  int top = -1;
  BuildOutHeap(graph->finalNode);
  graph->finalNode->heap3Done = 1;
  for (node = graph->node, numberNodes = graph->numberNodes;
       numberNodes != 0; node++, numberNodes--) {
    if (node->heap3Done == 1) continue;
    for (node2 = node; node2->heap3Done != 1 && node2->bestPath != NULL;
         node2 = node2->bestPath)
      stack[++top] = node2;
    if (top > -1) {
      node2 = stack[top--];
      BuildOutHeap (node2);
      if (node2->bestPath != NULL)
            node2->heap3 = MergeHeap3 (node2->heap3, node2->bestPath->heap3);
      node2->heap3Done = 1;
      for ( ; top >= 0; --top) {
            node2 = stack[top];
#ifdef DEBUG
            assert (node2 != NULL);
#endif
            BuildOutHeap (node2);
            node2->heap3 = MergeHeap3 (node2->heap3, stack[top+1]->heap3);
            node2->heap3Done = 1;
      }
    }
  }
}

/* Función RemoveMinFromSearchHeap */
void RemoveMinFromSearchHeap(void) {

  if (searchHeapSize == 0) return;
  searchHeap[0] = searchHeap[--searchHeapSize];
  {
    register DevPath * item = searchHeap[0];
    register int j = 1;
    while (j<searchHeapSize) {
      if (j<searchHeapSize-1 && searchHeap[j+1]->cost < searchHeap[j]->cost) j++;
      if (item->cost <= searchHeap[j]->cost) break;
      searchHeap[(j-1)/2] = searchHeap[j];
      j = 2*j+1;
    }
    searchHeap[(j-1)/2] = item;
```

```
  }
}

/* FUNCIÓN: InsertInSearchHeap */
void InsertInSearchHeap(DevPath * cand) {
  register int child, father;
  child = searchHeapSize++;
  while (child > 0 && searchHeap[father=(child-1)>>1]->cost > cand->cost) {
    searchHeap[child] = searchHeap[father];
    child = father;
  }
  searchHeap[child] = cand;
}

/* FUNCIÓN: SearchDeviationPaths */
void SearchDeviationPaths (Graph *graph, int K) {

  if (graph->initialNode->heap3 == NULL) return;
  bestDevPaths[0] = NULL;
  numberComputedPaths=1;
  searchHeap[0] = firstFreeDevPath++;
  searchHeap[0]->parent = NULL;
  searchHeap[0]->heap3 = graph->initialNode->heap3;
  searchHeap[0]->cost = graph->initialNode->heap3->data.cost;
  searchHeapSize=1;
  while(numberComputedPaths < K && searchHeapSize > 0) {
    DevPath * best = bestDevPaths[numberComputedPaths] = searchHeap[0];
    Heap3Node * bestHeap3 = best->heap3;
    RemoveMinFromSearchHeap();
    if (bestHeap3->left != NULL) {
      DevPath * cand = firstFreeDevPath++;
      cand->parent = best->parent;
      cand->heap3  = bestHeap3->left;
      cand->cost   = best->cost + (bestHeap3->left->data.cost - bestHeap3->data.cost);
      InsertInSearchHeap(cand);
    }
    if (bestHeap3->right != NULL) {
      DevPath * cand = firstFreeDevPath++;
      cand->parent = best->parent;
      cand->heap3  = bestHeap3->right;
      cand->cost   = best->cost + (bestHeap3->right->data.cost - bestHeap3->data.cost);
      InsertInSearchHeap(cand);
    }
    if (bestHeap3->center != NULL) {
      DevPath * cand = firstFreeDevPath++;
      cand->parent   = best->parent;
      cand->heap3    = bestHeap3->center;
      cand->cost     = best->cost + (bestHeap3->center->data.cost - bestHeap3->data.cost);
      InsertInSearchHeap(cand);
    }
    if (bestHeap3->data.arc->dest->heap3 != NULL) {
      DevPath * cand = firstFreeDevPath++;
      cand->parent   = best;
      cand->heap3    = bestHeap3->data.arc->dest->heap3;
      cand->cost     = best->cost + cand->heap3->data.cost;
      InsertInSearchHeap(cand);
    }
    cumulatedSeconds[numberComputedPaths] = ClockTotal();
    numberComputedPaths++;
  }
}

/* FUNCIÓN: PrintBestPath */
void PrintBestPath (Node *node) {

  Node *nextNode;
#ifdef DEBUG
```

```
  assert (node != NULL);
#endif
  if (node->bestCost < INFINITY_COST) {
    for (nextNode = node; nextNode != NULL; nextNode = nextNode->bestPath)
      printf ("%i-", nextNode->name);
    printf (" \t(Cost: %i)", node->bestCost);
  }
}

/* FUNCIÓN PRINCIPAL */
int main (int argc, char **argv) {

  Graph  graph;
  Node   *node;
  int    i, numberPaths = 1;
  time_t date;
  struct tm *localDate;
  char   hostName[200] = "";
  int    showPaths = 0;
  int    measureDijkstra = 0;
#ifdef DEBUG
  int    numberNodes;
#endif
  FILE *fid;            /* Nombre del fichero de alamacenamiento de datos */
  if (argc < 3) Help (argv[0]);  /* leer la línea de comandos de la función principal */
  numberPaths = atoi (argv[2]);
  for (i=3; i < argc; i++) {
    if (strcmp(argv[i], "-paths") == 0) showPaths = 1;
    else if (strcmp(argv[i], "-tdijkstra") == 0)  measureDijkstra = 1;
    else Help(argv[0]);
  }
  gethostname (hostName, 200);
  date = time (NULL);
  localDate = localtime (&date);
  printf ("CommandLine: ");
  for (i = 0; i < argc; i++)
    printf (" %s", argv[i]);
  printf ("\nHostname: %s", hostName);
  printf ("\nDate: %s", asctime(localDate));
  LoadGraph(&graph, argv[1]);  /* se carga el grafo que previamente se ha grabado en el fichero grafo.gr */

  heap3Size = graph.numberArcs - graph.numberNodes + 1
            + graph.numberNodes*log(graph.numberNodes)/log(2.0) + 1;
  firstFreeHeap3Node = heap3 = malloc(sizeof(heap3[0])*heap3Size);
  if (heap3 == NULL) {  /* se dispone memoria para los conjuntos considerando el peor de los casos */
    perror("Sin memoria suficiente para heap3\n");
    exit(1);
  }
  cumulatedSeconds = malloc (sizeof(cumulatedSeconds[0])*numberPaths);
  if (cumulatedSeconds == NULL) {
    perror("Sin memoria suficiente para los contadores.\n");
    exit(1);
  }
  stack = malloc (sizeof(stack[0]) * graph.numberNodes);
  if (stack == NULL) {
    perror("Sin memoria suficiente para pilas auxiliares.\n");
    exit(1);
  }
  traceBackStack = malloc (sizeof(traceBackStack[0]) * numberPaths);
  if (traceBackStack == NULL) {
    perror("Sin memoria suficiente para pilas auxiliares.\n");
    exit(1);
  }
  firstFreeDevPath = devPaths = malloc(sizeof(devPaths[0]) * 4 * numberPaths);
  /* caso peor: cada camino se incluye en 4 posibilidades */
  if (devPaths == NULL) {
    perror("Sin memoria suficiente para devPaths\n");
```

```
    exit(1);
  }
  searchHeap = malloc(sizeof(searchHeap[0]) * 3 * numberPaths);
  /* caso peor:  para cada camino: 1 borrado y 4 inclusiones */
  if (searchHeap == NULL) {
    perror("Sin memoria suficiente para searchHeap\n");
    exit(1);
  }
  bestDevPaths = malloc(sizeof(bestDevPaths[0]) * numberPaths);
  if (bestDevPaths== NULL) {
    perror("Sin memoria suficiente para bestDevPaths\n");
    exit(1);
  }
  /* se calculan los caminos más cortos en el árbol */
  if (measureDijkstra == 1)
    ClockReset ();
  DijkstraReversed (&graph);
  if (measureDijkstra == 1)
    cumulatedSeconds[0] =  ClockTotal();
  else {
    cumulatedSeconds[0] =  0;
    ClockReset ();
  }
#ifdef DEBUG
  for (node = graph.node, numberNodes = graph.numberNodes; numberNodes != 0;
       node++, numberNodes--) {
    printf ("\nMejor ruta para Nodo=%i: ", node->name);
    PrintBestPath (node);
  }
#endif      /* se calculan los k caminos más cortos */
  BuildDeviationHeaps(&graph);  /* se construyen los conjuntos. LLAMADA a BuildDeviationHeaps */
  SearchDeviationPaths (&graph, numberPaths);         /* Conjuntos, llamada a SearchDeviationPaths */
   fid = fopen("rep.dat", "w+");                              /*Se abre el fichero de salida de las rutas */
   fprintf (fid,"%i\n",numberComputedPaths);
  for(i=0; i<numberComputedPaths && i < numberPaths; i++) {
    int top = -1;
    DevPath * p;
    printf ("\nN=%i:\t", i+1);
    fprintf (fid,"\n");
    for (p = bestDevPaths[i]; p!=NULL; p=p->parent) traceBackStack[++top] = p;
    node = graph.initialNode;
    if (showPaths) printf("%d-", node->name);              /* se imprimen junto con los contadores de tiempo */
    fprintf(fid,"%d ", node->name);
    while (top >= 0 || node != graph.finalNode) {
      if (top>=0 && node == traceBackStack[top]->heap3->data.arc->source)
            node = traceBackStack[top--]->heap3->data.arc->dest;
      else
            node = node->bestPath;
      if (showPaths) printf("%d-", node->name);
      fprintf(fid,"%d ", node->name);
    }
    printf (" \t(Coste: %d)", graph.initialNode->bestCost + ((i>0) ? bestDevPaths[i]->cost : 0));
    printf (" \t(Tiempo: %.2f)", (float) cumulatedSeconds[i]);
  }
  fclose(fid);
  printf ("\nTiempo Total: %.2f\n", (float) cumulatedSeconds[numberComputedPaths-1]);
  return (0);
}
```

A.1.8.2. El algoritmo de Dijkstra.

```
#include <stdlib.h>
#include <stdio.h>
#ifdef DEBUG
#include <assert.h>
```

```
#endif
#include <EPPSTEIN.h>
/* A continuación las estructuras de datos y funciones implementan un conjunto binario que incluye la
posibilidad de establecer la prioridad de un elemento de dicho conjunto. Se usarán en la función
DijkstraReversed. */
typedef struct {
  COST_TYPE cost;                 /* Prioridad                                  */
  int       name;                 /* Identificador en el rango 0...dimension-1  */
  int       position;             /* Posición en la secuencia de elementos.     */
} DijkstraHeapElement;
typedef struct  {
#ifdef DEBUG
  int               dimension;    /* espacio asignado                   */
#endif
  int               size;         /* número de elementos                */
  DijkstraHeapElement *info;      /* campos de información, indexado  */
  DijkstraHeapElement **elt;      /* punteros a los campos de información  */
} DijkstraHeap;
#define NOT_IN_HEAP -1
#define PRIORITY(_i_) (heap->elt[_i_]->cost)

/* FUNCIÓN: CreateDijkstraHeap */
__inline__ void CreateDijkstraHeap (DijkstraHeap * heap, int dimension) {

  int i;
#ifdef DEBUG
  heap->dimension = dimension;
#endif
  heap->size = 0;  /*se asigna espacio para dimensión+1 elementos, insertar a partir de la posición 1 */
  heap->elt = malloc ((dimension+1)*sizeof(heap->elt[0]));
  if (heap->elt == NULL) {
    perror("No hay suficiente memoria para conjunto Dijkstra\n");
    exit(1);
  }
  heap->info = malloc(dimension*sizeof(heap->info[0]));
  if (heap->info == NULL) {
    perror("No hay suficiente memoria para conjunto Dijkstra\n");
    exit(1);
  }
  for (i = 0; i < dimension; i++)
    heap->info[i].position = NOT_IN_HEAP;
}
/* FUNCIÓN: FreeDijkstraHeap */
__inline__ void FreeDijkstraHeap (DijkstraHeap * heap) {
  free(heap->info);
  free(heap->elt);
}
/* FUNCIÓN: InsertInDijkstraHeap */
__inline__ void InsertInDijkstraHeap (DijkstraHeap * heap, COST_TYPE cost, int name) {
int position, parent;
DijkstraHeapElement *elt;
#ifdef DEBUG
  assert (heap->size < heap->dimension);
#endif
heap->size++;
position = heap->size;
parent   = position>>1;                         /* posicion/2 */
while (parent >= 1 && PRIORITY(parent) > cost) {
    heap->elt[position]          = heap->elt[parent];
    heap->elt[position]->position = position;
    position = parent;
    parent  = position>>1;                      /* posicion/2 */
}
  elt = heap->info + name;
  heap->elt[position] = elt;
  elt->position       = position;
  elt->cost           = cost;
```

```
  elt->name          = name;
}
/* FUNCIÓN: DeleteBestInDikstraHeap */
__inline__ COST_TYPE DeleteBestInDijkstraHeap (DijkstraHeap * heap, int * name) {

  COST_TYPE bestCost;
  DijkstraHeapElement *item;
  int position, parent;
#ifdef DEBUG
  assert (heap->size > 0);
#endif
  *name = heap->elt[1]->name;
  heap->elt[1]->position = NOT_IN_HEAP;
  bestCost = PRIORITY(1);
  item = heap->elt[heap->size];
  heap->size--;
  position = 2;
  while (position<=heap->size) {
    if (position<=heap->size-1 && PRIORITY(position+1) < PRIORITY(position)) position++;
    if (item->cost <= PRIORITY(position)) break;
    parent = position>>1;                          /* posicion/2 */
    heap->elt[parent]          = heap->elt[position];
    heap->elt[parent]->position = parent;
    position = position<<1;                        /* 2*posicion */
  }
  parent = position>>1;                            /* posicion/2 */
  heap->elt[parent]          = item;
  heap->elt[parent]->position = parent;

  return (bestCost);
}
/* FUNCIÓN: BelongsToDijkstraHeap */
__inline__ int BelongsToDijkstraHeap(DijkstraHeap *heap, int name) {

  return (heap->info[name].position != NOT_IN_HEAP);
}

/* FUNCIÓN: DecreaseCostInDijkstraHeap */
__inline__ void DecreaseCostInDijkstraHeap(DijkstraHeap *heap, COST_TYPE cost, int name) {

  int parent, position;
  position = heap->info[name].position;
#ifdef DEBUG
  assert (position != NOT_IN_HEAP);
  assert (cost < PRIORITY(position));
#endif
  parent   = position>>1;                          /* posicion/2 */
  while (parent >= 1 && PRIORITY(parent) > cost) {
    heap->elt[position]          = heap->elt[parent];
    heap->elt[position]->position = position;
    position = parent;
    parent = position>>1;                          /* posicion/2 */
  }
  heap->elt[position]          = heap->info + name;
  heap->elt[position]->position = position;
  heap->elt[position]->cost = cost;
}

/* FUNCIÓN PRINCIPAL: DijkstraReversed */
void DijkstraReversed (Graph *graph) {
/* Calcula el árbol T con el mejor camino desde cada nodo del grafo hasta
el nodo final. Trabaja con grafos con pesos positivos.
node->bestPath y node->bestCost se han iniciado en LoadGraph para cada nodo. */
  COST_TYPE    bestCost;
  int          indexNode, numberArcs;
  Node         *node;
  DijkstraHeap heap;
```

```
  Arc         *arc;
  CreateDijkstraHeap (&heap, graph->numberNodes);           /* LLAMADA CreateDijkstraHeap */
#ifdef DEBUG
  assert (graph->finalNode != NULL);
#endif
  graph->finalNode->bestCost = 0;
  graph->finalNode->bestPath = NULL;
  indexNode = graph->finalNode->name - 1;
  InsertInDijkstraHeap (&heap, 0, indexNode); /* LLAMADA InsertInDijkstraHeap*/
  while ( heap.size > 0 ) {
    bestCost = DeleteBestInDijkstraHeap (&heap, &indexNode);
    node = graph->node + indexNode;
    for (arc = node->firstArcIn, numberArcs = node->numberArcsIn;
        numberArcs != 0; arc++, numberArcs--) {
      Node *source = arc->source;
      if (bestCost + arc->cost < source->bestCost) {
            indexNode = source->name - 1;
            if (BelongsToDijkstraHeap (&heap, indexNode))
              DecreaseCostInDijkstraHeap (&heap, bestCost + arc->cost, indexNode);
            else
              InsertInDijkstraHeap (&heap, bestCost + arc->cost, indexNode);
            source->bestCost = bestCost + arc->cost;
            source->bestPath = node;
      }
    }
  }
  FreeDijkstraHeap (&heap);
}
```

A.1.8.3. La lectura del grafo: la función loadgraph.c.

```
#include <stdio.h>
#include <stdlib.h>
#include <EPPSTEIN.h>
#define MAX_LINE_LENGTH 100
/* FUNCIÓN: LoadGraph */
void LoadGraph(Graph *graph, char *fileName) {
/* Leer un multigrafo desde un fichero en tiempo O(número de arcos) */

  FILE *file;
  char line[MAX_LINE_LENGTH];
  int initialNodeName, finalNodeName, nodeName, sourceName, destName,
      numberArcs;
  COST_TYPE cost;
  Node *node, *source, *dest;
  Arc  *arc;
#ifdef DEBUG_LOAD_GRAPH
  PtrArc *ptrArc;
#endif
  if ((file = fopen (fileName, "r")) == NULL) {
    perror ("No se ha encontrado el fichero con el grafo");
    exit (1);
  }           /* se leen el número de nodos, de arcos, el nodo inicial y final */
  graph->numberNodes = graph->numberArcs = initialNodeName = finalNodeName = -1;
  while (fgets (line, MAX_LINE_LENGTH , file) != NULL) {
    if (line[0] == 'n')
      sscanf(line, "n%d\n", &graph->numberNodes);
    else if (line[0] == 'm')
      sscanf(line, "m%d\n", &graph->numberArcs);
    else if (line[0] == 's')
      sscanf(line, "s%d\n", &initialNodeName);
    else if (line[0] == 't')
      sscanf(line, "t%d\n", &finalNodeName);
    /* n, m, s, t se declaran antes que los arcos para consumir menos tiempo */
```

```
    /* la restricción se elimina eliminando elseif, pero consume más tiempo  */
    else if (line[0] == 'a') break;
  }
  if (graph->numberNodes <= 0) {
    fprintf (stderr, "\nNúmero de nodos del fichero incorrecto %s\n", fileName);
    exit (1);
  }
  if (graph->numberArcs <= 0) {
    fprintf (stderr, "\nNúmero de arcos del fichero incorrecto %s\n", fileName);
    exit (1);
  }
  if (initialNodeName <= 0 || initialNodeName > graph->numberNodes) {
    fprintf (stderr, "\nNodo inicial del fichero incorrecto %s\n", fileName);
    exit (1);
  }
  if (finalNodeName <= 0 || finalNodeName > graph->numberNodes) {
    fprintf (stderr, "\nNodo final del fichero incorrecto %s\n", fileName);
    exit (1);
  }
#ifdef DEBUG_LOAD_GRAPH
  printf ("NumberNodes=%d\nNumberArcs=%d\nInitialNode=%d\nFinalNode=%d\n",
          graph->numberNodes, graph->numberArcs,
          initialNodeName, finalNodeName);
#endif
  /* se asigna memoria para almacenar nodos y arcos */
  graph->node = malloc (sizeof (graph->node[0])*graph->numberNodes);
  if (graph->node == NULL) {
    perror("No hay suficiente memoria para loadgraph\n");
    exit(1);
  }
  graph->arc = malloc (sizeof (graph->arc[0])*graph->numberArcs);
  if (graph->arc == NULL) {
    perror("No hay suficiente memoria para loadgraph\n");
    exit(1);
  }
  graph->arcOut = malloc(sizeof ((graph->arcOut)[0])*graph->numberArcs);
  if (graph->arcOut == NULL) {
    perror("No hay suficiente memoria para loadgraph\n");
    exit(1);
  }                                 /* Fijar los punteros hacia los nodos inicial y final */
  graph->finalNode   = graph->node + (finalNodeName - 1);
  graph->initialNode = graph->node + (initialNodeName - 1);
  for (node = graph->node, nodeName = 1; nodeName <= graph->numberNodes;
       node++, nodeName++) { /* Iniciar para cada nodo: los contadores de arcos de entrada y salida */
    node->numberArcsIn       = 0;
    node->numberArcsOut      = 0;
    node->name               = nodeName;
    node->bestPath      = NULL;
    node->bestCost      = INFINITY_COST;
    node->heap3Done     = 0;
  }                     /* el conjunto de caminos candidatos, el mejor camino y el nombre del nodo */
  rewind (file); /* Se leen desde el fichero los arcos y contadores para ver los nodos de entrada y salida */
  numberArcs = 0;
  while (fgets (line, MAX_LINE_LENGTH , file) != NULL) {
    if (line[0] == 'a') {
      if (sscanf(line, "a%d %d %d\n", &sourceName, &destName, &cost) != 3) {
            fprintf (stderr, "\nError %s línea:\n%s\n", fileName, line);
            exit (1);
      }
      if (sourceName <= 0 || sourceName > graph->numberNodes) {
            fprintf (stderr, "\nNodo incorrecto del fichero %s línea:\n%s\n",
              fileName, line);
            exit (1);
      }
      if (destName <= 0 || destName > graph->numberNodes) {
            fprintf (stderr, "\nNodo incorrecto del fichero %s línea:\n%s\n",
              fileName, line);
```

```
            exit (1);
        }
        graph->node[sourceName-1].numberArcsOut++;
        graph->node[destName-1].numberArcsIn++;
        numberArcs++;
      }
    }
    if (numberArcs != graph->numberArcs) {
      fprintf (stderr, "\nEl número de arcos encotrados en %s difiere de m\n",
            fileName);
      exit (1);
    }
    numberArcs = 0;             /* Para cada nodo se inicia el puntero hacia el primer arco de salida */

    for (node = graph->node, nodeName = 1; nodeName <= graph->numberNodes;
        node++, nodeName++) {
      node->firstArcOut = graph->arcOut + numberArcs;
      numberArcs += node->numberArcsOut;
    }
    numberArcs = 0;  /* Para cada nodo se inicia el puntero hacia el primer arco de entrada */
    for (node = graph->node, nodeName = 1; nodeName <= graph->numberNodes;
        node++, nodeName++) {
      node->firstArcIn = graph->arc + numberArcs;
      numberArcs += node->numberArcsIn;
    }
    rewind (file);                                  /* se registran los arcos grabados en la memoria */
    for (node = graph->node, nodeName = 1; nodeName <= graph->numberNodes;
        node++, nodeName++) {
      node->numberArcsIn  = 0;
      node->numberArcsOut = 0;
    }
    while (fgets (line, MAX_LINE_LENGTH , file) != NULL) {
      if (line[0] == 'a') {
        sscanf(line, "a%d %d %d\n", &sourceName, &destName, &cost);
        source = graph->node + (sourceName - 1);
        dest   = graph->node + (destName - 1);
        arc    = dest->firstArcIn + dest->numberArcsIn;
        arc->source = source;
        arc->dest   = dest;
        arc->cost   = cost;
        source->firstArcOut[source->numberArcsOut] = arc;
        source->numberArcsOut++;
        dest->numberArcsIn++;
      }
    }
    fclose (file);
#ifdef DEBUG_LOAD_GRAPH
    for (node = graph->node, nodeName = 1; nodeName <= graph->numberNodes;
        node++, nodeName++) {
      printf ("Node=%d:\tNumberOutputArcs=%d:\n",
            node->name, node->numberArcsOut);
      for (ptrArc = node->firstArcOut, numberArcs = 1;
          numberArcs <= node->numberArcsOut; ptrArc++, numberArcs++)
        printf ("\t\t(%i, %i, %i)\n", (*ptrArc)->source->name,
              (*ptrArc)->dest->name, (*ptrArc)->cost);
      printf ("\tNumberInputArcs=%d:\n", node->numberArcsIn);
      for (arc = node->firstArcIn, numberArcs = 1;
          numberArcs <= node->numberArcsIn; arc++, numberArcs++)
        printf ("\t\t(%i, %i, %i)\n", arc->source->name,
                  arc->dest->name, arc->cost);
    }
#endif
}
```

REFERENCIAS

‘Coordinated Target Assignment and Intercept for Unmanned Air Vehicles’:

Randal W.Beard, Timothy W McLain, Michael A. Goodrich

IEEE Transactions on Robotics and Automation, Vol XX.

‘Coordination Variables, Coordination Functions and Cooperative Timing Missions’

Timothy W. McLain, Randal W. Beard.

IEEE Transactions on Robotics and Automation.

‘A Satisficing Approach to Assigning Vehicles to Targets’

Michael A. Goodrich

Proceedings of the IEEE International Conference on Systems.

‘An Efficient Implementation of Fortune’s Plane Sweep Algorithm for Voronoi diagrams’

Kenny Wong, Hausi A. Müller

‘Finding the k shortest paths’

David. Eppstein

SIAM Journal of Computing 1999, vol 28, no 2, pp 652-673.

‘Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning tress’

G.N. Frederickson.

Proc 32nd –symp. Foundations of Computer Science. pp 632-641. IEEE 1991