Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

# Desarrollo de una herramienta de planificación de trayectorias de evitación de regiones de riesgo meteorológico

Autor: Andrés Íñiguez Sigüenza

Tutor: Antonio Franco Espín

**Dpto. Ingeniería Aeroespacial**
**Escuela Técnica Superior de Ingeniería**
**Universidad de Sevilla**

Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

# Desarrollo de una herramienta de planificación de trayectorias de evitación de regiones de riesgo meteorológico

Autor:
Andrés Íñiguez Sigüenza

Tutor:
Antonio Franco Espín
Profesor Titular

Dpto. Ingeniería Aeroespacial
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: Desarrollo de una herramienta de planificación de trayectorias de evitación de regiones de riesgo meteorológico

Autor: Andrés Íñiguez Sigüenza
Tutor: Antonio Franco Espín

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

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

# Agradecimientos

*A toda mi familia, la cual me ha apoyado y me ha provisto de los medios necesarios para que esta tarea saliese adelante.*

*A todos mis amigos, los cuales, me han dado las fuerzas y el apoyo necesarios para continuar con el trabajo, sobre todo en los tiempos de cuarentena por los que hemos pasado.*

*A toda la escuela, en general, añorada durante la cuarentena pero, en especial, por supuesto, a mi tutor, Antonio Franco Espín, sin el cual este trabajo no hubiera sido posible, y con el que he trabajado mucho y muy agusto, incluso cuando la programación nunca ha sido mi fuerte.*

*A todos estos y a todos los demás que me han apoyado y han permitido que este trabajo saliese adelante, muchas gracias.*

# Resumen

Una causa frecuente de accidentes en la aviación es la inmersión en una región de riesgo meteorológico, como pueden ser las afectadas por una tormenta. Se propone, en este trabajo, una herramienta que permite obtener la trayectoria de mínimo tiempo que asegure que el seguimiento de la misma no va a producir la entrada de una aeronave en una de estas regiones peligrosas. Se creará, para cumplir tal objetivo, un algoritmo en el lenguaje de programación Matlab que devuelve dicha trayectoria segura y de mínimo coste temporal.

Este algoritmo se divide en diferentes bloques, en los cuales se desarrollarán las diferentes tareas necesarias para cumplir el objetivo anterior:

- **Predicción de la posición de las tormentas: Nowcasts basados en el radar meteorológico.** En esta sección se introducirán cualitativamente los datos que deben ser recibidos del radar meteorológico, datos que van a ser la base de todo el trabajo.
- **Determinación de los obstáculos estáticos equivalentes.** A lo largo de esta sección, se mostrará el algoritmo realizado para, a partir de unos datos meteorológicos aportados por el radar, que dependerán del tiempo y del espacio, crear unos polígonos equivalentes únicamente dependientes del espacio, de forma que estos obstáculos sean aquéllos a evitar por la aeronave. El objetivo será, por tanto, colapsar el tiempo y en el espacio, de forma que se obtengan unas regiones de riesgo sólo función de la posición de las mismas.
- **Construcción del grafo de visibilidad y determinación del camino de evitación de tormentas.** En este apartado, una vez conocidos los elementos a evitar, se trazarán todos los segmentos posibles que la aeronave va a poder seguir sin adentrarse en estas regiones peligrosa, eligiendo, de entre todos los posibles, aquellos que minimicen el tiempo entre el origen y el destino del vuelo.
- **Algoritmo de horizonte deslizante y maniobra de evasión.** Las tormentas no son constantes, estas varían con el tiempo, información que va a ser captada en tiempo real por el radar. Debido a esto y a que se trabajará con predicciones meteorológicas para tiempos futuros, se ejecutarán diferentes iteraciones del programa, actualizando la trayectoria segura que debería seguir la aeronave, con una nueva información meteorológica que el radar capta. El conjunto de estas iteraciones recibe el nombre de algoritmo de horizonte deslizante

  Es posible que, debido a la nueva información meteorológica adquirida, la aeronave se haya visto inmersa en una región de riesgo. Se trazará la trayectoria permita salir de ellas de la forma más segura posible, maniobra que se conoce con el nombre de maniobra de evasión.

La aplicación de estos aspectos y la programación de los mismos va a poder generar la trayectoria segura de mínimo tiempo que la aeronave debería seguir para evitar las regiones tormentosas que se detecten durante su vuelo:

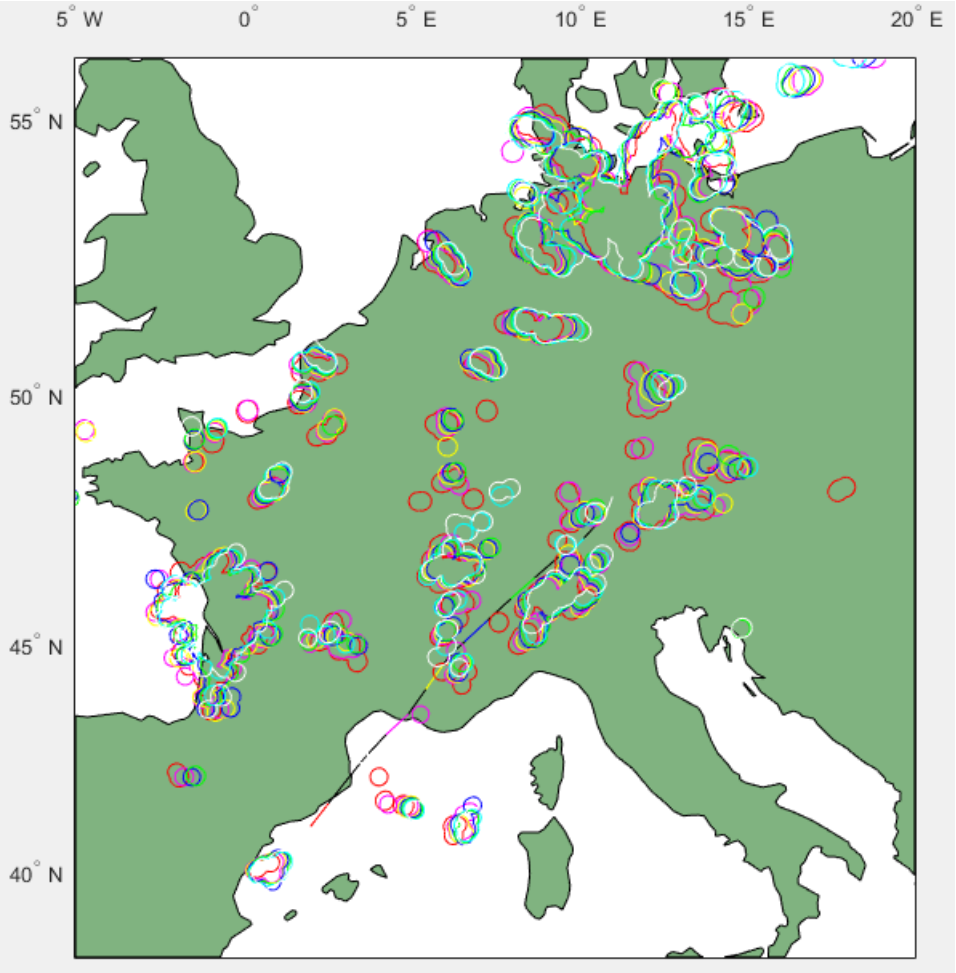



**Figura 1** Representación de los nowcasts y las trayectorias de mínimo tiempo obtenidas.

# Abstract

Accidents in aviation are frequently produced by the entrance into weather risk regions, like ones affected by storms. To solve this, it is proposed a tool that can search the path that minimizes the flight time and that assures that the fact to follow it doesn't cause the entrance into these dangerous regions. It will be created, to achieve this goal, an algorithm in Matlab that returns that safe path with minimum time cost.

This algorithm is divided into different blocks, in which necessary tasks are made to achieve the purpose of this work:

- **Prediction of storms'position: Nowcasts based on the meteorological radar.** In this section, the meteorological information received from the radar will be developed, information that will be used in the rest of the program.
- **Determination of equivalent static obstacles**. A new algorithm will be developed, which can create new equivalent polygons that only depends on space, given some meteorological information that depends on time and space. These polygons will be the obstacles to avoid by the aircraft.

  In other words, the target of this section is to collapse time and space, to get meteorological risk regions only function of their position.
- **Generation of the visibility graph and determination of the storms evading path.** In this section, already known the regions to avoid, every path that the aircraft can follow without entering into these dangerous regions will be calculated, choosing between them, the path that minimizes the flight time between the origin and the destiny.
- **Sliding horizon algorithm and escape manoeuvre.** Storms aren't constant in the time, they suffer some variation through it, information captured by the meteorological radar. Due to this variation and to the use of meteorological predictions for the future, new information about dangerous regions will be introduced each some minutes, running different iterations with the purpose of updating the path that the aircraft has to follow. These tasks receive the name of sliding horizon algorithm.

  It is possible, due to new regions that aircrafts have to avoid with the new meteorological information, that it is detected that the aircraft is immersed in one of these risk regions. To solve this, the program will trace the path that allows the aircraft to exit the regions in the most secure way (Escape manoeuvre).

The application of these aspects allows to generate the fastest safe path that the aircraft should follow to avoid the storming regions detected during the flight:

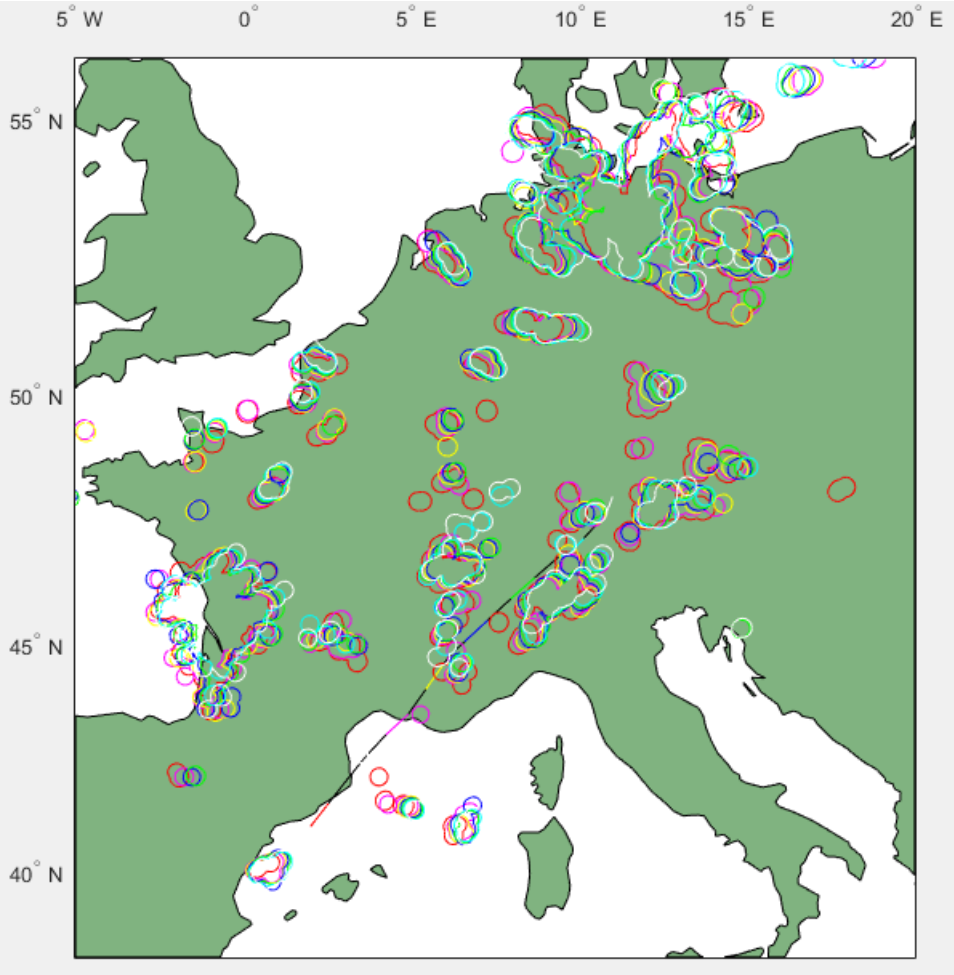



**Figura 2** Representation of nowcasts and minimum time cost paths .

# Índice de Figuras

# Índice

# Notación

| | |
|---|---|
| *dBZ* | Decibelios; unidad que expresa una relación entre dos variables, normalmente, potencias. |
| *FCC* | Flight Control Computers; computadoras que implementan las leyes de vuelo actuando sobre los diferentes sistemas del avión para controlar la aeronave. |
| *.mat* | Formato que tiene Matlab para el almacenamiento de matrices. |
| *cell* | Array de datos de Matlab consistente en una matriz, donde sus celdas pueden almacenar todo tipo de información. |
| $\chi$ | Ángulo de curso de la aeronave. |
| $\lambda$ | Longitud de un punto cualquiera. |
| $\lambda_0$ | Longitud del origen del vuelo en cada iteración. El subíndice 0 hace siempre referencia al punto inicial. |
| $\lambda_D$ | Longitud del destino del vuelo. El subíndice D hace siempre referencia al punto final. |
| $\phi$ | Latitud de un punto cualquiera. |
| $\phi_0$ | Latitud del origen del vuelo en cada iteración. |
| $\phi_D$ | Latitud del destino del vuelo. |
| $P_0$ | Punto de origen. |
| $P_D$ | Punto de destino. |
| $atan()$ | Arcotangente de un ángulo. |
| $tan()$ | Tangente de un ángulo. |
| $sen()$ | Seno de un ángulo. |
| $\|x\|$ | Valor absoluto de un número. |
| $R_t$ | Radio terrestre. |
| $r$ | Distancia entre dos puntos. |
| $W$ | Vector viento. |
| $W_u$ | Componente zonal del viento. |
| $W_v$ | Componente meridional del viento. |
| $W_{at}$ | Componente along-track del viento; paralela a la trayectoria seguida |
| $W_{xt}$ | Componente cross-track del viento; perpendicular a la trayectoria seguida. |
| $V$ | Velocidad aerodinámica. |
| $V_g$ | Velocidad respecto a tierra. |
| $M$ | Mach de vuelo. |

| | |
|---|---|
| $\gamma$ | Coeficiente de dilatación adiabática del aire. |
| $\theta$ | Relación de temperaturas entre la actual de vuelo y la de la superficie terrestre. |
| $\lambda_v$ | Longitud asociada al punto de la malla de vientos de menor longitud. |
| $\phi_v$ | Latitud asociada al punto de la malla de vientos de mayor latitud. |
| $Q$ | Nodos a partir de los cuales se interpolan los vientos. |
| $floor()$ | Comando que permite truncar los decimales de un número. |
| $ode45$ | Comando de Matlab que permite la integración numérica de una función. |
| *inpolygon* | Comando de Matlab para analizar si un punto está dentro de una curva cerrada. |
| *in* | Output del comando inpolygon que será 1 si el punto está dentro de la curva introducida. |
| $find$ | Comando de matlab que devuelve los índices de los elementos que cumplan la condición que se le introduzca al comando. |
| $frametormenta$ | Matriz que almacena 1 y 0, según si la la aeronave, al llegar a un punto es afectada por una región de riesgo. |
| *row* | Vector que almacena el índice de las filas donde hay un uno en la matriz frametormenta. |
| *col* | Vector que almacena el índice de las columnas donde hay un uno en la matriz frametormenta. |
| $x, y$ | Coordenadas horizontal y vertical de los puntos en la proyección de Mercator. |
| *convhull* | Comando de Matlab para trazar la envolvente convexa de un conjunto de puntos dados. |
| *chindice* | Matriz que almacena la longitud del vector que indica los vértices del polígono original que pertenecen a su envolvente convexa. |
| *mlength* | Matriz que almacena el número de nodos que conforman los polígonos estáticos previos a ser sustituidos por su envolvente convexa. |
| *n* | Número de obstáculos tormentosos. |
| *xv*2 | Matriz con las coordenadas x de los vértices de la envolvente convexa de los diferentes polígonos. |
| *yv*2 | Matriz con las coordenadas y de los vértices de la envolvente convexa de los diferentes polígonos. |
| *z* | Variable contador asociada a los polígonos. |
| $fusionpoligonos$ | Función para la fusión de polígonos cuyas envolventes convexas interseccionan. |
| *p* | Variable contador asociada a los polígonos. |
| *polyxpoly* | Comando de Matlab para detectar si existe corte entre dos polilíneas. |
| *match* | Matriz que almacena los números nodales de los vértices de los polígonos. |
| *contch* | Total de nodos considerados. |

| | |
|---|---|
| *xtan* | Matriz que almacena las coordenadas x de los puntos del obstáculo por los que pasa una tangente entre un punto y los diferentes obstáculo. |
| *ytan* | Matriz que almacena las coordenadas y de los puntos del obstáculo por los que pasa una tangente entre un punto y un obstáculo. |
| *zvert*0*poly* | Matriz que almacena los números nodales de los vértices por los que pasa una tangente entre el origen y los polígonos. |
| *zvertDpoly* | Matriz que almacena los números nodales de los vértices por los que pasa una tangente entre los polígonos y el destino. |
| *Origen* − *destino* | Variable que indica si existe visibilidad directa entre el nodo origen y destino. |
| *vectanpoint*0 | Vector que almacena los números nodales de los puntos de tangencia por los que pasa una tangente visible entre el origen y los polígonos. |
| *vectanpointD* | Vector que almacena los números nodales de los puntos de tangencia por los que pasa una tangente visible entre los polígonos y el destino. |
| *tangpolygontopolygon* | Función que calcula los índices de los puntos que conforman las tangentes entre los diferentes polígonos. |
| *indz*; *indp* | Indican los índices de los nodos de dos polígonos diferentes por los que pasa una tangente. |
| *xtanz*, *ytanz* | Matices que indican las coordenadas de los puntos de tangencia entre dos polígonos. |
| *mat* | Matriz que almacena los índices de los puntos de los polígonos por los que pasa una tangente entre polígonos. |
| *mattan* | Matriz que almacena los índices de los puntos de los polígonos por los que pasa una tangente visible entre polígonos. |
| *vectanorden* | Vector que almacena los números nodales de los nodos del obstáculo por los que pasa una tangente entre ellos, pero sólo de uno de cada pareja de polígonos. |
| *vectandorden*2 | Vector que almacena los números nodales de los puntos de tangencia no almacenados en el vector anterior. |
| *graph* | Comando de Matlab para crear grafos a partir de una serie de nodos y conexiones sin dirección. |
| *digraph* | Comando de Matlab para crear grafos a partir de una serie de nodos y conexiones dirigidas. |
| *segment* | Matriz que almacena los nodos que une cada conexión. |
| *matcost* | Matriz que almacena los costes temporales de cada conexión. |
| *contchconf*0 | Variable que indica el número nodal del origen real, o en caso de estar encerrado, el ficticio. |
| *contchconfD* | Variable que indica el número nodal del destino real, o en caso de estar encerrado, el ficticio. |
| *matcostd* | Matriz que almacena los costes en distancia de cada conexión |

| | |
|---|---|
| *shortespath* | Comando de Matlab que detecta el camino más corto entre dos puntos de un grafo. |
| *Costecam* | Vector usado para almacenar el tiempo que tardaría en llegar, desde el origen, a cada uno de los nodos del camino seguro de menor tiempo. |
| *nodoant* | Nodo anterior al punto a 5 minutos del origen. |
| *nodopos* | Nodo posterior al punto a 5 minutos del origen. |
| *tstop* | Tiempo entre el nodo anterior y el punto situado a 5 minutos del origen. |
| *rord* | Vector con las distancias, ordenadas de menor a mayor, entre el origen o destino encerrados con cada uno de los nodos del polígono que lo encierra. |
| *course*1 | Curso que la aeronave seguía cuando se detecta que está encerrada en un obstáculo. Para la primera iteración, este curso coincide con que seguiría al unir de forma directa el origen con el destino real. |
| *track*2 | Comando que permite trazar diferentes tipos de trayectorias dados los puntos de origen y destino. |
| *puntodentro* | Variable que indica si uno de los puntos de origen (puntoordentro) o destino (puntodesdentro) se encuentran dentro de un obstáculo. |
| *zconf* | Variable que indica en el polígono en el que los puntos están encerrados. |
| *plot* | Herramienta de Matlab para graficar funciones. |
| *plotm*, *geoshow* | Herramienta de Matlab para graficar funciones en una proyección diferente de la plana. |

# 1 Introducción

## 1.1 Peligros de la entrada en una región de riesgo meteorológico

Se define una región de riesgo meteorológico como aquella zona del espacio afectada por una tormenta o por otros agentes meteorológicos adversos que pueden causar daños a la aeronave si ésta la atraviesa.

Al mismo tiempo, se define una tormenta como el agente meteorológico producido en la atmósfera terrestre debido a la colisión entre dos masas de aire de diferente temperatura. Las tormentas se producen tras la formación de un tipo especial de nube, los cumulonimbos, que se caracterizan por ser de gran tamaño y de gran desarrollo vertical.

La entrada de la aeronave en una de estas regiones le puede acarrear serios problemas, tanto estructurales, como de estabilidad y control de la misma, así como otros que afectan a su correcta actuación. Estos se van a desarrollar a continuación:

- **Turbulencia** [1]. Efecto muy temido por los pasajeros de las aeronaves, generado por los efectos convectivos que se dan durante la propia formación de la tormenta. Se define como la componente errática del movimiento de un fluido, es decir, como el efecto del cambio de dirección y de intensidad que sufren las corrientes de aire.

  Las tormentas de origen convectivo se producen por un calentamiento de las partículas de aire en la superficie, las cuales, al aumentar su temperatura, suben. Al ir entrando en contacto con capas de diferente temperatura y cambiando la estructura de las masas de aire en el entorno de dichas partículas calientes, se va a producir un posterior movimiento caótico y errático, que es el anteriormente comentado.

  Esta turbulencia también puede darse al colisionar el viento con un obstáculo, como una montaña. Para este caso el viento ascenderá para tratar de evitarla de forma natural, sucediendo, así, lo mismo que en el caso anterior.

  Los efectos de la turbulencia, pueden ir desde un leve y prácticamente inapreciable agitamiento de la aeronave, hasta serios balanceos que van a impedir el control de la aeronave. Estos balanceos son producidos por una distribución de sustentación cambiante, la cual tendrá asociados una distribución de momentos de cabeceo, guiñada y balance, erráticos.

- **Efectos del viento** [2]. A lo largo de la vida de las tormentas, diversos fenónemos ventosos son observados: desde leves microbursts hasta algunos de mucho mayor intensidad, como los macroburst. Ambos se definen como las corrientes frías y descendentes, que se generan en las nubes de tormenta.

  Los efectos de todos los fenómenos ventosos son similares, aunque difieren en la intensidad con los que lo producen. Pueden ir desde cambios en la trayectoria y en la velocidad de la aeronave, hasta la generación de fuertes momentos de guiñada (viento lateral), cabeceo y balanceo (vientos verticales). Estos últimos van a ser de especial interés en las maniobras más críticas del vuelo, como son el despegue o el aterrizaje, donde la aparición estos puede provocar un descontrol de la aeronave y un inminente choque contra el suelo, pudiendo ser un accidente catastrófico.

  Los vientos también pueden provocar un efecto muy temido en la aviación: la **cizalladura**. Se define ésta como la diferencia en la velocidad del viento o su dirección entre dos puntos de la atmósfera, la cual hace que la aeronave pueda estar sometida a cambios de viento en un pequeño espacio de tiempo, siendo incontrololable la aeronave, llegando, en las fases de vuelo cercanas a la superficie terrestre como pueden ser el aterrizaje o el despegue, a causar serios accidentes.

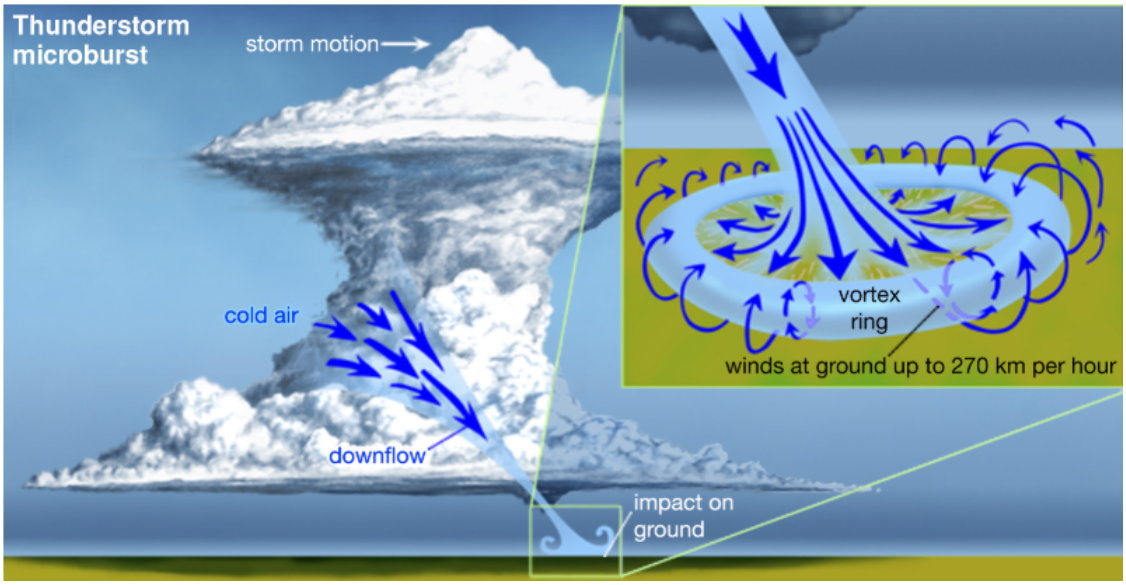



**Figura 1.1** Representación de las microrráfagas.

- **Engelamiento** [3] [4]. Se define como el depósito de hielo que se produce cuando el agua fría en estado de subfusión (agua líquida por debajo de 0º) se congela al impactar con la aeronave. Esta formación de hielo se da normalmente en zonas expuestas al viento relativo, como pueden ser sondas o bordes de ataque de alas y estabilizadores, entradas de motores...

  Una gran humedad del medio, y bajas temperaturas del mismo y de la aeronave, son situaciones que favorecen el engelamiento. Este tendrá los siguientes efectos negativos en las aeronaves:

  Desde el punto de vista de la estabilidad, dicho engelamiento puede suponer un problema. El hielo en el borde de ataque -puntos de mayor contribución a la sustentación de la aeronave-,

engelamiento que se conoce con el nombre de **engelamiento estructural**, va a provocar una reducción apreciable de la capacidad sustentadora de hasta en torno a un 30 %, lo que, sumado al aumento de peso que puede suponer la presencia de dicho hielo, va a provocar un efecto desestabilizador, tanto longitudinal, como, incluso, lateral-direccional.

El momento de guiñada (momento direccional) puede verse intensificado por una distribución de empuje por motor mermada por la formación de hielo en la entrada del motor (**engelamiento interno**) y en las hélices, en caso de que las llevase y por una reducción del gasto debido a la disminución del área de entrada.

El engelamiento en los sensores puede provocar un falseamiento de las señales que van ser recibidas en cabina y en los FCC de una aeronave con sistema fly-by-wire. El problema de esto es que, al ser errónea la información, las superficies estabilizadoras, así como los otros sistemas del avión, van a actuar de forma incorrecta, produciendo situaciones de riesgo. A su vez, se pueden dar señales de altura falseadas, efecto muy peligroso para la aviación.

Otros efectos, como la reducción de visibilidad por la formación de hielo en los mamparos o la deflexión tardía de las superficie por engelamiento en las uniones son también asociados a situaciones tormentosas con nubes frías (con temperaturas en todo momento por debajo de 0º)

**Figura 1.2** Engelamiento en el borde de ataque de un avión.

- **Electricidad en las tormentas** [5]. Los aviones, por normativa, están preparados para soportar el impacto de un rayo en su fuselaje. Normalmente los efectos que producen son leves, esto es debido a que el fuselaje actúa como una jaula de Faraday, aislando el interior de la aeronave. En la actualidad, los aviones no son totalmente metálicos, es por ellos que existen caminos prediseñados de mínima resistencia por las que la radiación del rayo pasará minimizando los daños del mismo en el avión.

  El problema de estas descargas radica en que si las conexiones entre elementos no son demasiado buenas, es decir, no hay continuidad estructural, el rayo se desviará del camino prediseñado, provocando grandes fallos en el sistema eléctrico. Aún si no existe desviación, éstas pueden generar serias interferencias en los sensores y sus señales.

Un impacto de un rayo puede provocar alteraciones en la geometría de la aeronave que pueden suponer la entrada en pérdida de la misma por el desprendimiento de la capa límite, así como daños estructurales severos, que imposibilitan el control normal de la aeronave.

**Figura 1.3** Impacto de un rayo sobre una aeronave.

Van a ser todos estos peligros que pueden tener lugar en una aeronave por la inmersión en una región tormentosa, lo que sustenta el trabajo y su desarollo.

## 1.2 Detección de las regiones de riesgo

En cualquier ruta o trayectoria que un avión realice entre un punto inicial y final, éste se puede encontrar con situaciones de riesgo metereológico, cuyos efectos pueden llegar a ser muy perjudiciales para la aeronave, como se ha podido observar en la sección anterior.

Es por ello que surge la necesidad de llevar a bordo sistemas que permitan detectar estas zonas de riesgo, y evitar así poner en peligro a la aeronave y sus tripulantes. Algunos de los más usados, y de mayor interés para este trabajo, son los siguientes: [6]

- **Radar meteorológico** (weather radar). Situado en el morro del avión, permite mandar unas ondas de radio, similar a como lo harían los demás radares montados en el avión. Cuanto más densa sea la lluvia, mayor será la reflexión de las ondas, y por tanto el radar, gracias a esta reflexión, detectará la mayor intensidad de la lluvia y su mayor peligrosidad.

  Este radar, con el mismo funcionamiento que un radar Doppler, no es más que un conjunto de antenas que permite enviar ondas de radio de forma direccional y recibir su reflexión -llamado también eco o retorno- cuando estas choquen contra algún obstáculo, que para el caso considerado, son las precipitaciones en forma de agua, granizo o nieve. El radar realiza un barrido horizontal de 360º y mediante la inclinación de las antenas mencionadas, permitir obtener, además, información meteorológica para diferentes alturas.

El radar podrá obtener el tipo de objeto, según la potencia del eco recibido, dónde y a cuánta distancia se encuentra el del avión, de forma que se puede obtener, así, una idea de las situaciones de riesgo. Además, dado que los barridos se hacen de forma continua, se puede observar e, incluso, predecir el movimiento de las tormentas.

Se diferenciarán distintos colores en el display del radar, según la intensidad y, por tanto, la peligrosidad de las precipitaciones en el entorno, que se comentarán a continuación. Dichos colores harán referencia a la potencia de reflexión de dichos ecos, medida en decibelios normalizados.

- Magenta ($>$ 60 dBZ). Lluvia de enorme intensidad, riesgo extremo si esa zona es atravesada por la aeronave.
- Rojo ($>$ 45 dBZ). La señal es retornada fuertemente. Lluvia de intensidad alta.
- Amarrillo/Naranja ($>$ 30-35 dBZ). La señal recibida tras ser reflejada por la lluvia tiene una intensidad media, al igual que las precipitaciones detectadas.
- Verde ($>$ 20 dBZ). Precipitaciones débiles, de riesgo leve.
- Negro ($<$ 20 dBZ). Sin riesgo aparente de precipitaciones.

En la figura 1.4 se puede ver el display en cabina del radar meteorológico con la gama de colores indicada anteriormente.

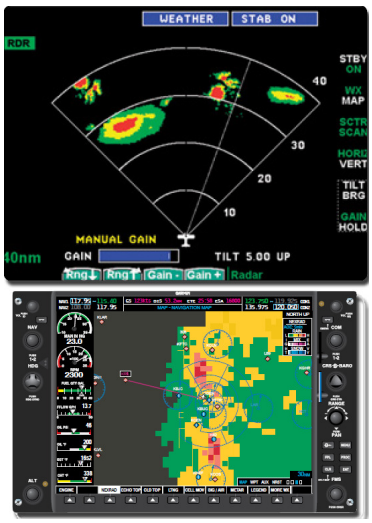



**Figura 1.4** Display del radar meteorológico.

La unión de las diferentes zonas va a crear un mapa que indicará el lugar geométrico de los puntos con precipitaciones, y por tanto de riesgo para la aeronave.

Cabe remarcar que la única finalidad del radar meteorológico es detectar la presencia de lluvias o similares. En el caso de que existiera viento seco u otros agentes meteorológicos las

ondas de radio no retornarían, no siendo, por tanto, percibidos.

Este radar va a permitir al piloto tener una idea de las diferentes zonas de riesgo, con el objetivo de cambiar la ruta que el avión va a seguir y evitar, así, los efectos negativos que estas zonas provocarían en la aeronave.

- **Detector de Rayos** (stormscope). Mediante una antena en espira, se permite conocer el azimut de los rayos por las interferencias electomagnéticas que estos generan en la antena.

  En la figura 1.5, mostrada a continuación, se puede ver el display en cabina del radar detector de rayos, donde los puntos amarillos indican la posición donde dichos rayos han caído.

**Figura 1.5** Display del detector de rayos.

- **Radares vía satélite**. Permite recibir información vía satélite de la situación meteorológica que se va a encontrar la aeronave, así como de otros datos que le pueden servir de interés.

  Supone la ventaja de que la información que se manda a la aeronave está refinada, pues es la conjunción de las diferentes visiones de varios satélites, y la recibirá casi en tiempo real, de forma que va a tener, para cada instante, un mapa real de la situación de riesgo meteorológico.

  Este sistema esta siendo cada vez más utilizado en todos los tipos de aeronaves.

# 2 Objetivo

Como se ha visto en la sección anterior, la inmersión de una aeronave en una región de riesgo meteorológico, puede provocar serios problemas, tanto para la integridad estructural de la misma, como pueden ser el efecto de la caída de un rayo en una de las superficies de control o el engelamiento en las mismas, como para el control y estabilidad de la misma, debido a los efectos de las rachas de viento o del engelamiento en los bordes de ataque de las alas. Estos fenómenos, consecuencias frecuentes de las tormentas, conllevan grandes riesgos para la aeronave. Son estos peligros, y los accidentes que tienen lugar debido a ellos, los que evidencian la necesidad de una herramienta de evitación de regiones tormentosas.

El objetivo de este trabajo es, por tanto, a partir de la información meteorológica provista por el radar, crear una herramienta que calcule todos los posibles segmentos que la aeronave pueda seguir, entre un punto de origen y otro de destino, y que aseguren que no va a estar afectada, en ningún momento, por alguna de las regiones tormentosas procedentes de esa información meteorológica, seleccionando, de entre ellos, aquellos que conforman la trayectoria que minimiza el tiempo de vuelo entre dichos puntos, evitando, así, los riesgos asociados a la inmersión en una región tormentosa y reduciendo los tiempos asociados a realizar estas maniobras.

La importancia de este trabajo radica en que, al ser un programa informático, permite ser implementado en los computadores de vuelo de la aeronave, en caso de llevar un sistema fly-by-wire, lo que va a permitir que este se ejecute de forma automática cuando la información es recibida, efectuando en la aeronave un cambio de rumbo que le permita seguir la trayectoria segura de mínimo coste obtenida, sin necesidad de que el piloto accione los mandos para cambiar dicha trayectoria. Al ser un proceso automatizado aumenta la eficiencia, al reducirse el número de fallos, como los humanos, asociados al aumento de trabajo y carga a la que se tiene que enfrentar la tripulación, como el que sufriría al adentrarse la aeronave en una de estas regiones tormentosas peligrosas. Son estas características y sus implicaciones las que van convertir esta herramienta en un sistema de apoyo en la toma de decisiones de los pilotos.

Los computadores de control de vuelo (*FCC*) no son más que unas computadoras que permiten implementar las diferentes leyes de control de vuelo y que, mediante la interconexión con otra serie de computadores y actuadores, permitirán manejar las superficies de control y, por tanto, la aeronave en su conjunto. Es decir, son estos computadores los que actuarán sobre la aeronave, para adecuarse a la trayectoria segura de mínimo tiempo, calculada en el interior de los mismos.

Además, la inmersión en una región de riesgo meteorológico, como puede ser una tormenta, suele traer consigo situaciones de baja visibilidad, las cuales, con un sistema automático y digitalizado,

como el que se está presentando, no va a suponer un problema, pues no se requiere la visión directa del piloto.

El hecho de que estas tormentas se eviten, en lugar de tener que lidiar con ellas, además de reducir la carga a la que tienen que estar sometidos los pilotos (aumentando la presión y reduciendo la efectividad), podría reducir la redundancia de aquellos sistemas abordo usados para aliviar los peligros a los que un avión se somete cuando se da la inmersión del mismo en una región de riesgo meteorológico, disminuyendo, así, el peso de la aeronave, variable de gran interés en el mundo aeronáutico.

# 3 Desarrollo del algoritmo

En este capítulo de la memoria se van a analizar las diferentes decisiones y pasos que se han tomado y seguido para la realización del programa que determinará la trayectoria óptima a seguir para evitar el riesgo que supone la entrada en un región tormentosa.

Previamente, se describirán, de forma sucinta, los diferentes grandes bloques en los que se divide el algoritmo que se plantea en el trabajo:

- **Predicción de la posición de las tormentas: Nowcasts basados en radar meteorológico.** En esta sección se introducirán cualitativamente los datos que deben ser recibidos del radar meteorológico, base de todo el trabajo. Se desarrollará, además, el tratamiento al que estos se deben someter para poder cumplir la función de la herramienta de forma eficiente.
- **Determinación de los obstáculos estáticos equivalentes.** A lo largo de esta sección, se mostrará el algoritmo realizado para, a partir de unos datos meteorológicos aportados por el radar, que dependerán del tiempo y del espacio, crear unos polígonos equivalentes únicamente dependientes del espacio, de forma que estos obstáculos sean aquellos a evitar por la aeronave. El objetivo será, por tanto, colapsar el tiempo y el espacio, de forma que se obtengan unas regiones de riesgo sólo función de la posición de las mismas.
- **Construcción del grafo de visibilidad y determinación del camino de evitación de tormentas.** En este apartado, una vez conocidos los elementos a evitar, se trazarán todos los segmentos posibles que la aeronave va a poder seguir sin adentrarse en estas regiones peligrosa, eligiendo, de entre todos los posibles, aquellos que minimicen el tiempo entre el origen y el destino del vuelo.
- **Algoritmo de horizonte deslizante y maniobra de evasión.** Las tormentas no son constantes, estas varían con el tiempo, información que va a ser captada en tiempo real por el radar. Debido a esto y a que se trabajará con predicciones meteorológicas para tiempos futuros, se ejecutarán diferentes iteraciones del programa, actualizando la trayectoria segura que debería seguir la aeronave, con una nueva información meteorológica que el radar capta. El conjunto de estas iteraciones recibe el nombre de algoritmo de horizonte deslizante.

  Es posible que, debido a la nueva información meteorológica adquirida, la aeronave se haya visto inmersa en una región de riesgo. Se trazará la trayectoria que permita salir de ellas de la forma más segura posible. Esto también se resolverá a lo largo de este capítulo.

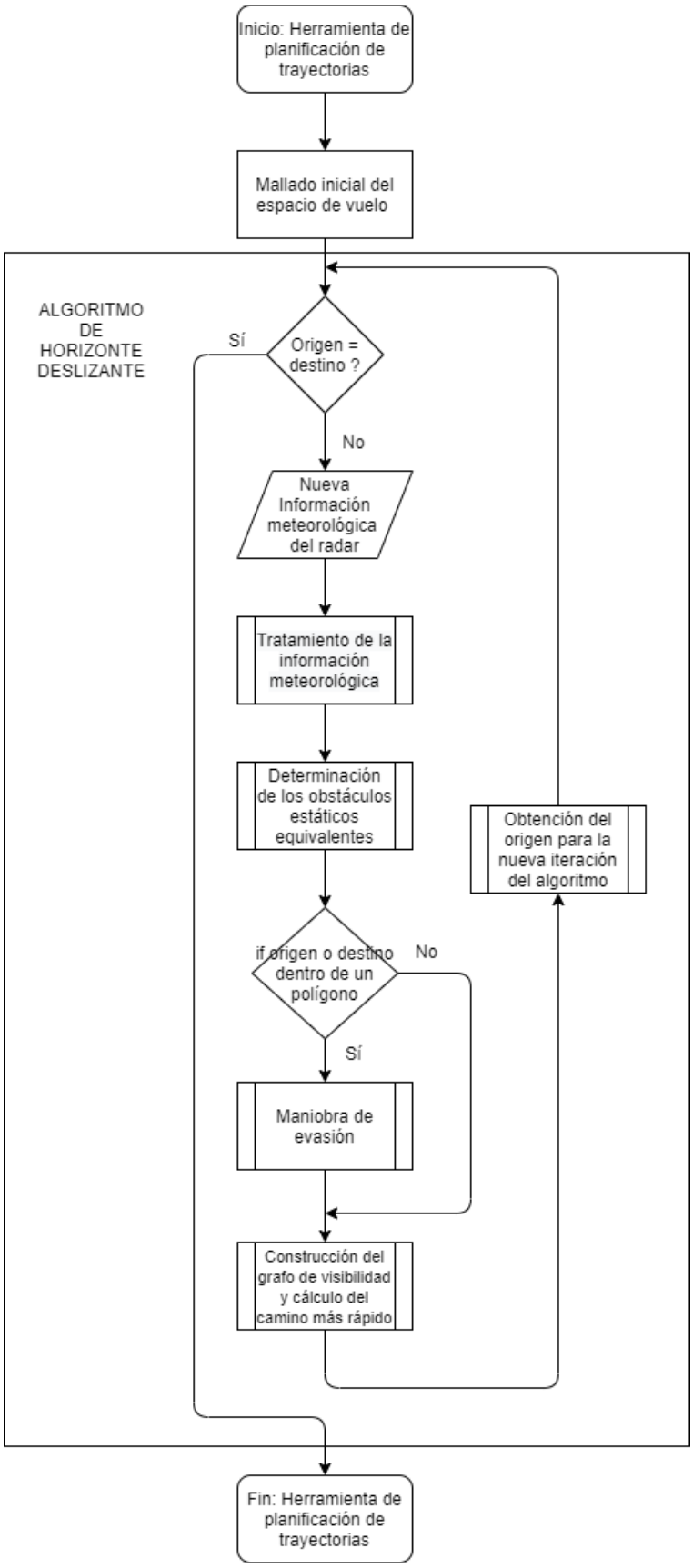



**Figura 3.1** Diagrama de flujo de las operaciones a realizar por programa.

## 3.1 Predicción de la posición de las tormentas: Nowcasts basados en el radar meteorológico.

Mediante la información proporcionada por el radar meteorológico u otros, se podrán obtener las regiones de riesgo meteorológico que la aeronave tendrá que evitar. Como no se dispone de este sistema, será el tutor el que proporcione dicha información meteorológica. Esta consistirá en una serie de nowcasts, con las regiones de riesgo en el momento actual de vuelo, acompañados de unas predicciones, conocidas como forecast, en las que se podrá observar, de forma aproximada, la variación en el tiempo de estas regiones tormentosas.

Como nowcast, en este TFG se ha considerado NowCastMIX-Aviation (NCM-A). NCM-A es un producto del Servicio Alemán de Meteorología (Deutscher Wetterdienst, DWD) que proporciona las reflectividades radar en dBZ como predicciones de corto alcance y cubre un área extensa de Europa Central. Los datos de NCM-A vienen como imagen rasterizada en un arhivo de formato GRIB con una resolución espacial de 1km x 1km. Cada archivo contiene 13 mensajes GRIB: uno para la observación y 12 mensajes de predicción hasta una hora con una granularidad de 5 minutos. El ciclo de actualización es también de 5 minutos, es decir, se proporciona un nuevo conjunto de 13 mensajes cada 5 minutos. Los polígonos de las células convectivas se han obtenido a partir de la imagen rasterizada considerando un umbral de reflectividad de 37dBZ.

De cara a la presentación de resultados, el tutor ha proporcionado las zonas de riesgo meteorológico. Estas se han obtenido partiendo del conjunto de polígonos (tanto en el instante de predicción, como en las extrapolaciones siguientes) previstos por NCM-A para el 29 de junio de 2017 desde las 20:30 horas hasta las 10:00, ampliados con un margen de seguridad de 10NM.

Este conjunto de nowcasts y forecasts, tras una previa manipulación de la misma, será el input meteorológico del trabajo, el cual será provisto por el tutor en formato .mat, donde se encontrarán almacenadas las diferentes regiones tormentosas en vectores con sus coordenadas $\lambda$ y $\phi$ (longitud y latitud, respectivamente) para un vuelo que parte a las 8:30 y con una duración máxima de una hora y media.

Estos datos, serán tratados previamente: las regiones, en la realidad, son más pequeñas que las aportadas, es decir, se realiza un proceso de aumento del tamaño de las regiones de riesgo, con el objetivo, así, de asegurar que la aeronave, en caso de que la región se moviese, colocándose sobre la aeronave, esta no se vea inmersa en la tormenta real. Es decir, existe un margen de seguridad que evita que, aunque el avión se adentre en la región, las tormentas reales afecten al mismo.

Además, en caso de que estas se corten entre ellas, se someterán a un proceso de fusión de las mismas. La información será recibida en estas condiciones (ver diagrama de flujo de la imagen 3.2).

Se llamará a cada fila de la matriz aportada, la cual va a incluir las regiones cada 5 minutos. Estas filas, tienen el formato cell, cuyos elementos pueden incluir información de diferente naturaleza. Para el caso considerado, van a ser los vectores con las coordenadas de los nodos de las diferentes zonas tormentosas, para ese instante y las que se esperan cada 5 minutos después, hasta 60 -pues se tienen 13 filas en cada uno de los elementos de las células anteriores-, de forma que se tenga información de las regiones durante todo el vuelo. A partir de la primera foto, aportada por el radar meteorológico, las imágenes se corresponderán con predicciones del escenario meteorológico, es decir, variaciones del nowcast real obtenido. Como se ha comentado, esto recibe el nombre de forecast, que será dicha extrapolación en el tiempo de las regiones tormentosas detectadas por el

radar.


**Figura 3.2** Diagrama de flujo para la preparación de los datos meteorológicos a usar.

Como se ha comentado, a los 5 min se tendrá nueva información del escenario meteorológico (punto de inicio del algoritmo de horizonte deslizante), actualizando las regiones anteriores, mediante el avance hacia la siguiente fila de la matriz inicial (*stormTFG*). Se llamará, por tanto, a la fila correspondiente según la hora de vuelo actual. En estas ideas se profundizará más adelante (ver apartado 3.6.1).

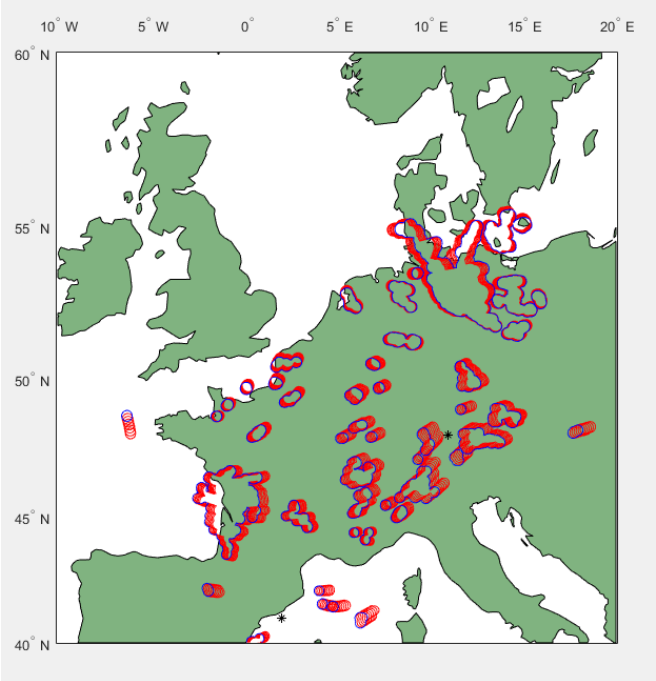



**Figura 3.3** Polígonos detectados en el instante inicial (azul) y con previsión para 60 min posteriores (rojo) en la proyección de Mercator, representados, por simplicidad, cada 10 minutos.

Para la representación de esta imagen, y de las siguientes que se mostrarán en este trabajo, se ha hecho uso de la mapping toolbox de Matlab, la cual va a permitir mostrar en pantalla un mapa obtenido aplicando la proyección deseada (dentro de un conjunto muy amplio de proyecciones disponibles), así como los comandos *plotm* y *geoshow* que son aquellos que permiten la representación de estas imágenes, de forma análoga al comando *plot*.

## 3.2 Determinación de los obstáculos estáticos equivalentes

Como se comentó anteriormente, nuestro objetivo en este bloque va a ser, principalmente, colapsar el tiempo y el espacio, en una misma foto, es decir, conseguir la relación directa entre los estados anteriores. Esto va a permitir conseguir el tratar el problema como uno únicamente dependiente del espacio y poder definir, así, los polígonos que representarán las regiones tormentosas a evitar, que ahora serán solo función de la posición de las mismas.

Para cumplir el objetivo de obtener los obstáculos estáticos definitivos que la aeronave va a tener que sortear durante el vuelo, se llevarán a cabo las diferentes tareas que se observarán en el diagrama de flujo de la imagen 3.4, las cuales se desarrollarán a lo largo de esta sección:

### 3.2.1 Obtención de la matriz de tiempos

Primeramente se realizará un mallado del espacio, cuyos puntos se almacenarán en dos matrices: lambda, que incluye las longitudes del espacio considerado, y phi, las latitudes. Ver **A.1**

Esta discretización dividirá el espacio en pequeños rectángulos, cuyos centros serán los puntos almacenados en las matrices anteriores. Cada uno de estos puntos estará separado 0.1 grados de sus vecinos.

Una vez obtenido el mallado correspondiente, y haciendo uso de unas ecuaciones, conocidas con el nombre de ecuaciones loxodrómicas, que se mostrarán a continuación, se podrá obtener la distancia entre un punto de origen y otro punto del espacio mallado, así como el curso constante que el avión debe seguir para alcanzar dicho punto.

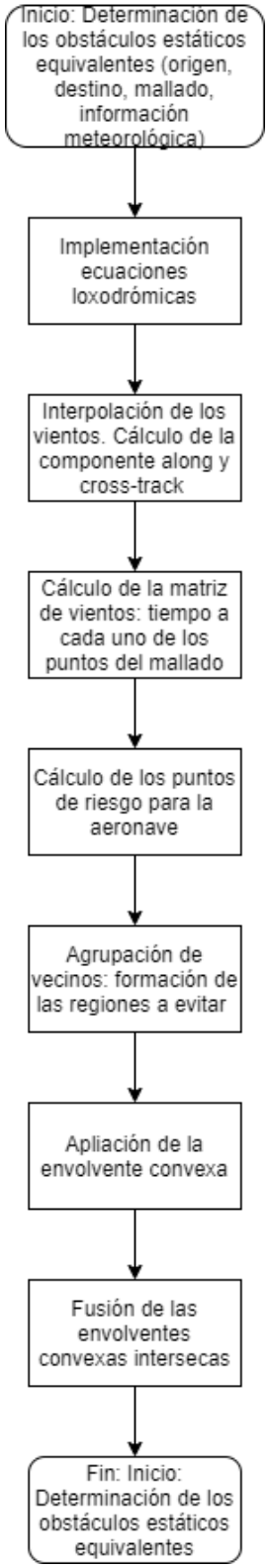



**Figura 3.4** Diagrama de flujo para la obtención de los obstáculos estáticos equivalentes.

Dichas ecuaciones [7] permitirán el cálculo de la distancia y el curso, para un vuelo con curso constante entre dos puntos dados, suponiendo que la tierra es esférica, y son las que se presentan a continuación. En ellas, $\lambda_0$ y $\phi_0$ van a hacer referencia a la longitud y la latitud, respectivamente, del origen de vuelo:

$$\chi = atan(\frac{\lambda + 2\pi - \lambda_0}{ln(\frac{tan(\pi/4-\phi_0/2)}{tan(\pi/4-\phi/2)})}) \; si \; |\lambda - \lambda_0| > \pi \; y \; \lambda < 0 \tag{3.1}$$

$$\chi = atan(\frac{\lambda - 2\pi - \lambda_0}{ln(\frac{tan(\pi/4-\phi_0/2)}{tan(\pi/4-\phi/2)})}) \; si \; |\lambda - \lambda_0| > \pi \; y \; \lambda > 0 \tag{3.2}$$

$$\chi = atan(\frac{\lambda - \lambda_0}{ln(\frac{tan(\pi/4-\phi_0/2)}{tan(\pi/4-\phi/2)})}) \; si \; |\lambda - \lambda_0| < \pi \tag{3.3}$$

$$\chi = \chi + \pi \; si \; \phi < \phi_0 \tag{3.4}$$

Una vez calculado el curso de vuelo se procederá con la distancia recorrida:

$$\alpha_{lox} = \frac{\phi - \phi_0}{cos(\chi)} \tag{3.5}$$

$$r = \alpha_{lox} \cdot R_t \tag{3.6}$$

$$r = R_t \cdot |\lambda - \lambda_0)| \cdot cos(\phi_0) \; si \; (\phi - \phi_0) = 0 \tag{3.7}$$

$$r = R_t \cdot (\phi - \phi_0) \; si \; (\lambda - \lambda_0) = 0 \tag{3.8}$$

Con estas ecuaciones, implementadas en **A.2**, se permite resolver lo que se conoce como el problema inverso, que es aquel en el que se tienen las premisas comentadas en el párrafo anterior (obtener distancia y curso dados el origen y el destino).

El planteamiento de estas ecuaciones y la posterior obtención de la distancia y el curso van a tener como objetivo tratar el espacio como un mallado en distancias entre el origen y los diferentes puntos de la discretización inicial. Esto facilitará el cálculo de la matriz de tiempos, así como el de la velocidad respecto a tierra, variable indispensable para la obtención de dicha matriz. Con las operaciones anteriores, se va a pasar de considerar las variables dependientes de las coordenadas $\lambda$ y $\phi$ (longitud y latitud de los puntos del mallado) a, únicamente, de la distancia recorrida entre dos puntos.

La velocidad respecto a tierra, antes mencionada, no es más que la velocidad medida por una persona situada en la superficie terrestre. Esta es una combinación de la velocidad aerodinámica que la aeronave lleva y la velocidad del viento, que cambiará la dirección de vuelo de la misma. Esta última, descompuesta en una componente meridional y otra zonal, se proyectará en unos ejes ligados a la trayectoria de curso constante que la aeronave sigue (ver figura 3.5), de forma que se obtengan dos nuevas componentes, una *along-track*, tangente a la trayectoria seguida y, por tanto, paralela a la velocidad respecto a tierra, y una componente cruzada, *cross-track*, perpendicular a las anteriores:

$$W_{at} = W_u \cdot cos(\pi/2 - \chi) + W_v \cdot sin(\pi/2 - \chi) \tag{3.9}$$

$$W_{xt} = -W_u \cdot sin(\pi/2 - \chi) + W_v \cdot cos(\pi/2 - \chi); \tag{3.10}$$

$$V_g = \sqrt{V^2 - W_{xt}^2} + W_{at} \tag{3.11}$$

Donde la velocidad aerodinámica tiene la siguiente expresión $V = M^2 \gamma \theta$ con, M, el Mach de vuelo, $\gamma$, la constante del aire, y $\theta$, la relación entre la temperatura a la altitud actual y la que se tiene en el suelo.

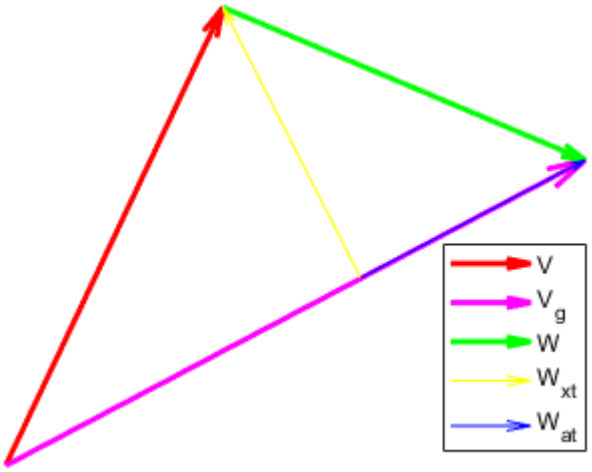



**Figura 3.5** Proyección del vector viento.

Sea un matriz de vientos cualquiera. Puede darse el caso en que no se tenga información del viento real en los puntos del mallado inicial. Entonces no queda más remedio que realizar una interpolación para calcular su valor en los puntos donde se está interesado en obtener la velocidad respecto a tierra (puntos del mallado inicial), para la que, como se observa en las ecuaciones anteriores, el conocimiento de los vientos es indispensable.

Se va crear una función cuyo objetivo principal es obtener los vientos en los puntos donde se demande. Esta función va a tener como datos de entrada, la matriz de vientos original, los puntos donde se conocen estos vientos y las coordenadas del punto donde estamos interesados en calcular su valor. (Ver **A.3**)

La interpolación que se aplicará recibe el nombre de **interpolación bilineal** [8]. Esta permite interpolar funciones de dos variables, siendo en este caso: las coordenadas $\lambda$ y $\phi$ del punto en cuestión y cuyos nodos de interpolación van a ser los vértices de una malla, en los cuales el valor de la función es conocido. Para el caso que concierne, la función a interpolar es el viento, y los extremos donde la función es conocida, son los puntos de la matriz de vientos, cuyas coordenadas están almacenadas en las matrices *lambdavientos* y *phivientos*.

Una vez formada la malla, se procederá con la interpolación, propiamente dicha. Se aplicará la siguiente fórmula. Habrá que tener en cuenta que la numeración usada en ella no tiene por qué

corresponder con la que se va a seguir en el código:

$$W(\lambda,\phi) \approx \frac{W(Q11)}{(x_2-x_1)(y_2-y_1)} \cdot (x_2-\lambda)(y_2-\phi) + \frac{W(Q21)}{(x_2-x_1)(y_2-y_1)} \cdot (\lambda-x_1)(y_2-\phi) + \\ + \frac{W(Q12)}{(x_2-x_1)(y_2-y_1)} \cdot (x_2-\lambda)(\phi-y_1) + \frac{W(Q22)}{(x_2-x_1)(y_2-y_1)} \cdot (\lambda-x_1)(\phi-y_1) \tag{3.12}$$

Dicha fórmula hace referencia a los puntos de la imagen 3.6. Los términos que multiplican a las diferentes fracciones, equivalen a la diferencia entre las coordenadas del punto a conocer los vientos y las del diagonalmente opuesto en la malla al punto donde se evalúa la función en el numerador de las fracciones anteriores.

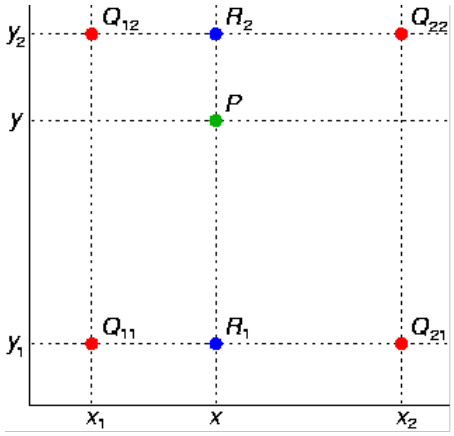



**Figura 3.6** Esquema para la interpolación bilineal.

El objetivo primero de la función de interpolación de vientos elaborada es calcular la esquina superior izquierda (Q12, en la imagen 3.6) de la malla en cuyo interior se encuentra el punto donde se quieren calcular los vientos. La razón por la que se buscará la esquina superior se debe a la forma en la que la matriz de coordenadas ha sido rellenada, en la cual el primer elemento es el correspondiente a aquel de mayor latitud y menor longitud.

Se comenzará calculando la longitud de dicha esquina. El procedimiento es muy sencillo: primeramente se calculará lo que se va a denominar $\Delta\lambda$, que no es más que el incremento de longitud entre dos puntos adyacentes de la malla.

Una vez obtenido este, la tarea ahora es analizar cuántos incrementos de lambda existen entre el punto en cuestión y el punto de menor lambda del mallado, de forma que truncando el valor obtenido y sumándole una unidad, se obtenga el índice de los puntos con una longitud menor al punto donde se quieren calcular los vientos, pero de mayor proximidad. El incremento en una unidad se debe a que los índices en Matlab parten del valor unidad:

$$i = floor(\frac{\lambda-\lambda_v}{\Delta\lambda}) + 1 \tag{3.13}$$

En la ecuación anterior, $\lambda$ es la longitud del punto donde se quieren calcular los vientos, $\lambda_v$; la longitud los puntos de la primera columna de la malla (puntos de menor longitud), y *floor*; un comando que permite truncar el resultado de lo que se encuentre tras él.

El valor del índice i calculado permitirá obtener la longitud de la esquina superior izquierda, coordenada que se corresponderá con la variable $x_1$ en la imagen anterior.

Haciendo un procedimiento análogo para obtener el índice de la coordenada $\phi$, quedará totalmente definida la esquina buscada:

$$j = floor(\frac{\phi_v - \phi}{\Delta\phi}) + 1 \tag{3.14}$$

El valor del índice j calculado permitirá obtener la latitud de la esquina superior izquierda, coordenada que se corresponderá con la variable $y_2$ en la imagen anterior.

Para obtener el resto de las esquinas de la malla donde se encuentra inmerso el punto con vientos a interpolar, se obtendrán incrementando los índices según corresponda. Es importante recordar que, debido a la forma en la que las matrices han sido rellenadas, al aumentar el índice j, se encuentran puntos de menores latitudes, contrariamente a lo que ocurre en la imagen anterior. Son lo indicado en este recordatorio y la simplicidad de la obtención del resto de los nodos de interpolación, las razones, como se comentó en párrafos anteriores, por las que el punto inicialmente buscado es la esquina superior izquierda de la malla, punto que se corresponde con los de menores índices dentro de dicha malla.

Existe un caso límite que ocurre cuando el punto en cuestión está situado en la última fila o en la última columna del mallado de puntos donde se conocen los vientos. En este caso, la esquina superior izquierda se encontrará en dichas localizaciones, lo que implica que al incrementar estos índices, estos pueden sobrepasar las dimensiones del mallado. Por ello, si esto es detectado, se reducirá en uno el índice afectado por la limitación anterior. Al hacer esto, el punto ya se encuentra dentro de una malla formada por cuatro esquinas existentes.

Como comentamos anteriormente y volviendo al tema inicial, el cálculo de la velocidad respecto de tierra es fundamental para la obtención de la matriz de tiempos. Como sabemos la velocidad no es más que la derivada con respecto al tiempo de la posición. Debido a esto, integrando el inverso de velocidad con respecto a la posición se puede obtener, para un punto de llegada dado, el valor del tiempo que la aeronave tarda en llegar a ese destino. Esto es posible gracias a que ahora el espacio se puede mallar en distancias con respecto a los diferentes puntos, pues la velocidad del viento y, por tanto, la velocidad respecto a tierra únicamente dependen de la posición, siendo directa la obtención de los tiempos.

$$V_g = \frac{dx}{dt} \rightarrow t = \int \frac{dx}{V_g} \tag{3.15}$$

Con el objetivo de automatizar la operación de integración, se ha hecho uso del comando *ode45*, el cual dada la función que debe integrar (ver **A.4**), unos valores de los límites de integración -la integración se hará entre 0 y la distancia que separa el origen con el punto a calcular el tiempo-, y un valor inicial de la función -nulo, pues el tiempo en llegar al punto de partida, desde el mismo, es nulo-, permite proporcionar el valor, obtenido de forma numérica, de la función integrada, en

este caso el tiempo, para los límites dados. La función se llamará para cada elemento de la matriz de mallado, de forma que, para cada punto de destino, se indique el tiempo que tardaría en llegar. A este comando se le pueden añadir una serie de especificaciones adicionales para, por ejemplo, cambiar la tolerancia en los cálculos. Para nuestro caso, aquellas que el comando tiene por defecto se han considerado aceptables ($10^{-6}$ y $10^{-3}$, tolerancias absoluta y relativa, respectivamente), no añadiendo dichas especificaciones comentadas.

Con el objetivo de reducir el coste temporal que supondría el bucle anterior, se cambiarán las dimensiones de las matrices para convertirlas en vectores. Además, en versiones anteriores de Matlab a la usada para la realización de este trabajo, se podía utilizar el tipo de bucle *parfor* -las restricciones para su uso se han endurecido, no pudiéndose emplear con la versión utilizada para realizar esta tarea-, el cual permite realizar en paralelo las diferentes iteraciones del bucle, si estas no estan relacionadas entre sí. La velocidad de ejecución dependerá del número de procesadores de la computadora, pues cada uno de los procesadores realiza al mismo tiempo las operaciones de las diferentes iteraciones, disminuyendo el tiempo empleado.

Para la obtención de dicha matriz (*tmat*), únicamente con un *reshape* cambiamos las dimensiones a las iniciales, pues se colocó en forma de vector para reducir los tiempos de ejecución. Una vez realizada esta operación se tiene una matriz que, a cada punto del mallado espacial en distancia, le asigna un tiempo de llegada, teniendo una correspondencia directa entre espacio y tiempo.

### 3.2.2 Obtención de la matriz que recoge los puntos afectados por las regiones de riesgo tormentoso

La finalidad de obtener esta matriz de tiempos es, además de para conocer el tiempo empleado en ir de un punto a otro, permitir conocer si cuando el avión llegue a cada uno de los puntos a los que este podría viajar, es decir, a los puntos del mallado considerado, se va a encontrar en una región tormentosa que pueda suponer un riesgo para el control y la estabilidad de la aeronave.

Para facilitar la obtención de estos puntos, se supondrá que las diferentes regiones de riesgo meteorológico avanzan de forma discreta y cada cierto tiempo prefijado (en este caso 5 min, pues solo se conoce el panorama meteorológica cada dicho tiempo), es decir, la interpolación es tal que en tiempos intermedios, la región ocupa la misma posición que al inicio del intervalo, obtenidas mediante la información provista por el radar meteorológico, como se vio en la sección primera.

Se creará una matriz, de nombre *frametormenta*, que recogerá si en algún momento del tiempo considerado (hasta 60 minutos posteriores) el avión va a estar inmerso en una de dichas regiones (polígonos) cuando la aeronave llegue a cada uno de los posibles puntos de vuelo. (Ver **A.5**)

Para el cálculo de dicha matriz, que sólo contiene unos y ceros, se procederá de la siguiente manera. Para cada intervalo de tiempo considerado (definido el intervalo por la variable k), de 5 minutos de duración cada uno y para cada una de las regiones (incluidas, como se comentó, en los datos aportados por el tutor) y filas de las matrices del mallado, se comprobará, primeramente, si los puntos del espacio recogidos en esa fila están dentro del polígono tormentoso, mediante el comando *inpolygon*.

Este comando, dadas las coordenadas de un conjunto de puntos, así como las de una curva cerrada, tal como un polígono, permite calcular si existen puntos que están dentro de esa curva. Devolverá un vector, cuyo nombre será *in*, de unos y ceros, donde el uno indica que el punto que ocupa la

misma posición en el vector de coordenadas introducido, está incluido dentro de la curva cerrada.

También devolverá un vector que recoge si los puntos están justamente en la frontera de la curva. Este no va a ser de interés, y no se utilizará en los cálculos siguientes.

A continuación, en caso de que el punto de llegada esté dentro de una región peligrosa, es decir, si el elemento correspondiente del vector *in* sea la unidad y, si el tiempo que tarda el avión en llegar a dicho punto está incluido en el intervalo entre k y k+1, en el que el punto se ha detectado que está inmerso en una de las regiones, entonces la matriz *frametormenta* recogerá el valor 1 en dicho punto. Esto implicará que el avión cuando llegue a ese punto (tiempo indicado en la matriz de tiempos), se verá inmerso en una región de riesgo.

Esto viene a decir que habrá que considerar dicho punto como uno a evitar cuando se estén calculando las trayectorias de evasión de tormentas, pues va a formar parte de uno de los polígonos tormentosos que van a afectar a la aeronave.

Aunque un punto esté afectado por una región, si en el instante que la aeronave llega, la tormenta se ha movido y el punto ya no está en su interior, este punto no va a tener que ser evitado, pues no es peligroso para la aeronave. Por ello, se requieren cumplir las dos condiciones anteriores (in(i)=1 y tiempo que tarda la aeronave incluido en el intervalo en el que se da in(i)=1)), para que un punto sea considerado como perteneciente a una región de riesgo, no siendo todos los puntos de las tormentas originales de necesaria evitación.

Como se ha comentado anteriormente, a través de estas operaciones, se ha obtenido una matriz de unos y ceros. Estos unos van a indicar que, cuando el avión llegue a esos puntos, estos van a estar afectados por una región de riesgo meteorológico. Puesto que se considera la evolución de los polígonos tormentosos con el tiempo (cada 5 minutos), esta matriz no es más que una fusión de todos los puntos en los que el avión, en su llegada a los mismos, estará inmerso en una zona tormentosa.

Esto es muy interesante, pues nos va a permitir pasar de diferentes imágenes en distintos instantes de tiempo, a una única imagen espacial, colapso de las anteriores, independiente del tiempo, lo que va a dar lugar a unos obstáculos estáticos que son los que la aeronave va a evitar con esta herramienta.

Como se ha explicado, esto dará lugar a nuevos polígonos, que van a estar formados por puntos del mallado con un uno en la matriz *frametormenta*. La tarea ahora será reunir dichos puntos y establecer estos nuevos polígonos tormentosos.

### 3.2.3 Obtención de los polígonos estáticos equivalentes

El objetivo ahora es formar los polígonos que la aeronave tendrá que evitar durante su vuelo. Esto se consiguirá mediante la agrupación de los puntos que tengan asignado un valor unidad en la matriz *frametormenta*.

Primeramente, para mayor comodidad, se usará el comando *find*, el cual, si no se le da ninguna condición adicional, identifica los elementos distintos de cero y almacena sus índices lineales. En caso de que se le demanden dos outputs, devolverá, en su lugar, las componentes i y j de los elementos. Será de mayor utilidad la segunda opción, por tanto, se obtendrán dichas componentes y se almacenarán, respectivamente, en dos vectores de nombres *row* y *col* . Se ha realizado esta tarea para tener que recorrer menos elementos y reducir el tiempo de ejecución.

Se utilizará una variable de nombre j que -aunque tenga el mismo nombre que la componente segunda de los elementos anteriores, no va a tiene relación alguna- se usará como indicador del punto que se quiere analizar si tiene vecinos.

Se recorrerán dichos vectores (*row* y *col*) para cada una de sus componentes, guardando en una matriz los elementos según si estos tienen vecinos o no a su alrededor. Se analizará, a partir del segundo elemento de los vectores, si, para cada uno de los elementos anteriores al de análisis (definido por el índice j, que se corresponde con la fila los vectores anteriores), tiene como vecino a dicho elemento. Los elementos diagonales se incluirán como vecinos de otro.

Si se encuentra que el punto es vecino de otro ya guardado, es decir, de otro ya analizado, se almacenarán sus índices en su misma fila, lo que va a implicar que van a pertenecer al mismo polígono tormentoso. En caso contrario, se almacenarán como primer elemento de una nueva fila, y por tanto, de un nuevo polígono. Ver **A.6**

El problema de esto es que, debido a la forma en los elementos se han guardado en los vectores *row* y *col*, durante el barrido anterior para saber si un elemento es vecino a otro previamente guardado, puede darse el caso de que se considere que algunos elementos pertenecen a un polígono aparte, cuando en la realidad no es así. Esto ocurre, por ejemplo, en algunos nodos de los laterales de la envolvente de los polígonos. Por ello, es necesario hacer un nuevo barrido para corregir esto y realizar la fusión.

Se harán cuatro barridos, de gran coste computacional, para cada elemento i,j y k,l pertenecientes a las filas siguientes -o polígonos siguientes- analizándose, para cada punto k,l si son vecinos (diagonales incluidas) del punto i,j.

Si lo es, entonces los elementos distintos de cero de esa fila (k) son almacenados al final de la fila del elemento que se está analizando (punto i,j). Se eliminarán, a continuación, todas las variables asociadas a dicha fila, y se inicializará la variable l a 0 manteniéndose k, para que se comience a analizar la que sería la siguiente fila a la eliminada.

Con esto se tienen guardados todos los índices de los diferentes puntos del mallado original que adquiriesen el valor unidad de la matriz *frametormenta* separados por polígonos. Pero, estos puntos del mallado, son en realidad, como ya se comentó, el centro de los diferentes frames donde las tormentas se van a encontrar. Es por ello que lo que se hará a continuación es, por cada elemento de la matriz anterior, añadir 4 puntos más, los cuales se corresponderán con los cuatro vértices del rectángulo que conforma el frame con centro en el elemento de la discretización. Estos puntos formarán una malla cuadrada en la que sus vértices adyacentes están separados 0.1 grados entre sí.

Posteriormente, se eliminarán aquellos puntos que sean iguales, para evitar futuros problemas por la duplicidad de los mismos, obteniéndose la malla representada en la imagen 3.7

A continuación, se realizará un cambio de coordenadas (Ecuaciones 3.16 y 3.17) [9]), pasando de la latitud y la longitud a las coordenadas horizontal y vertical (x e y, respectivamente) de la proyección de Mercator. El uso de la proyección de Mercator está justificado porque, de esta manera, los segmentos rectilíneos que definen el grafo de visibilidad constituyen segmentos loxodrómicos (a curso constante).

$$x = \lambda \tag{3.16}$$

$$y = 0.5 \cdot ln[\frac{1+sen(\phi)}{1-sen(\phi)}] \tag{3.17}$$

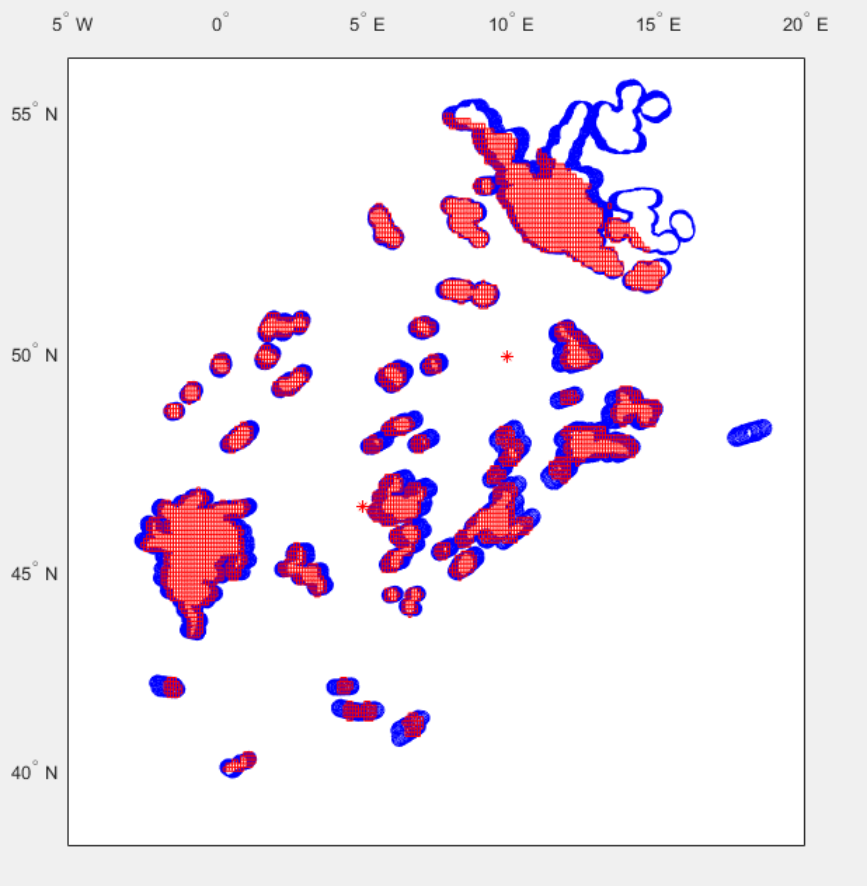



**Figura 3.7** Mallado de las regiones de riesgo meteorológico. Frames de polígonos en la proyección de Mercator.

### 3.2.4 Polígonos tormentosos: envolvente convexa

Se define la **envolvente convexa** como la cobertura que encierra a todos los elementos de un conjunto con el menor perímetro posible, es decir, sin vértices cóncavos [10]. Como es lógico pensar, un polígono convexo coincide con su envolvente convexa. De forma simple, se define un polígono convexo como aquel que, montado sobre él y recorrido en un sentido, no existe ninguna desviación en el sentido de giro, o como aquel polígono cuyos ángulos interiores no superan 180.

El concepto de envolvente convexa va a ser clave en el trabajo, pues se supondrá que todos los polígonos van a poder ser sustituidos por sus envolventes convexas. Su principal desventaja es que, al utilizar dichas envolventes, existe parte adicional del espacio que se está considerando de riesgo, al aumentar el tamaño de las regiones a evitar, cuando en realidad no lo es, consecuencia que puede ser asumida, argumentando que, con ello, se consigue aumentar el margen de seguridad de los obstáculos originales, alejando a la aeronave aún más de las regiones reales de riesgo.

Debido a dicho aumento de las dimensiones de los obstáculos, es posible que los polígonos originales que no interseccionaban entre ellos, al reemplazarlos por su envolvente convexa, sí lo

hagan. Entonces habrá que proceder a la fusión de los mismos y a comprobar que, tras su realización, no existen nuevos solapes con las demás envolventes convexas. En estos casos, se intensifica la desventaja anterior, reduciendo el número de corredores por los que la aeronave podría pasar en su camino de mínimo tiempo y, por tanto, de posibles segmentos que podrían formar parte de dicho camino, aumentando, así, el tiempo en llegar al destino. En la realidad, estos segmentos pueden ser recorridos por la aeronave, es el hecho de tomar las envolventes convexas de los obstáculos, el que imposibita su seguimiento.

Previamente a resolver esto, mediante el comando *convhull* e introduciéndole como entrada los vectores con las componentes x e y, respectivamente, de los puntos de los polígonos tormentosos, independientes del tiempo, como se comentó en los apartados pasados, se permite trazar la envolvente convexa de los polígonos originales (ver figura 3.8). Cabe recordar que estos puntos, introducidos como input, son los vecinos separados por polígonos obtenidos en el apartado anterior.

Este comando devolverá los índices correspondientes de los vértices que conforman dicha envolvente convexa, duplicando el vértice por el que se comienza y se cierra la envolvente. Esta envolvente se calculará para cada polígono. Además se guardará en una matriz, de nombre *chindice*, de dimensiones nx1, el número de elementos que conforman el vector de índices resultante del comando para cada uno de los obstáculos. Esto será de gran utilidad, pues, gracias a ella, se guardarán todos los elementos de las diferentes envolventes convexas en diferentes filas de una sola matriz, rellenando con ceros las posiciones no ocupadas por dichos elementos, de forma que la matriz chindice permitirá recorrer solo las columnas anteriores a los ceros, evitando, así, problemas por la existencia de índices nulos. Esto también supondrá un ahorro temporal en las operaciones con estas envolventes convexas, pues solo se tendrá que recorrer cada una de las filas hasta el número indicado en la matriz anterior. Las coordenadas de los nodos que conformarán la envolvente convexa del polígono tormentoso se almacenerán en las matrices xv2 y yv2, que serán las verdaderamente usadas en el resto del trabajo, con número de elementos distintos de cero los indicados en cada una de las filas de la matriz *chindice*.

Una vez dicho esto y volviendo a lo anterior, es posible que las envolventes convexas de los diferentes polígonos intersecccionen entre sí, aún cuando los polígonos originales no lo hacían, lo que va a dar diversos problemas durante la compilación del programa, caso que se puede observar en algunas de las envolventes de la imagen 3.8. Es por ello que se procederá a la fusión de las mismas.

Se creará una función, de nombre *fusionpoligonos*, para detectar y fusionar dichas envolventes convexas. Dados todos los puntos pertenecientes al polígono y, por otro lado, los pertenecientes a su envolvente, así como el número de elementos distintos de cero de las matrices anteriores -matriz *mlength* y *chindice*, respectivamente-, el número de polígonos y el analizado en ese momento -z-, permitirá devolver los nuevos polígonos fusionados. (Ver **A.7**)

Se hará un bucle en z, variable que se va a usar en el código para indicar el polígono que se está considerando, para el que si esta variable es mayor que el número de polígonos totales, n, saldrá de ese bucle y continuará con el resto del código. La función se ejecutará cuando z sea mayor o igual que dos, para poder comparar, así, con el polígono anterior.

La función anterior incluye un barrido en p. Esta variable indicará los polígonos anteriores a z, con los que se comprobará si existe intersección con este.

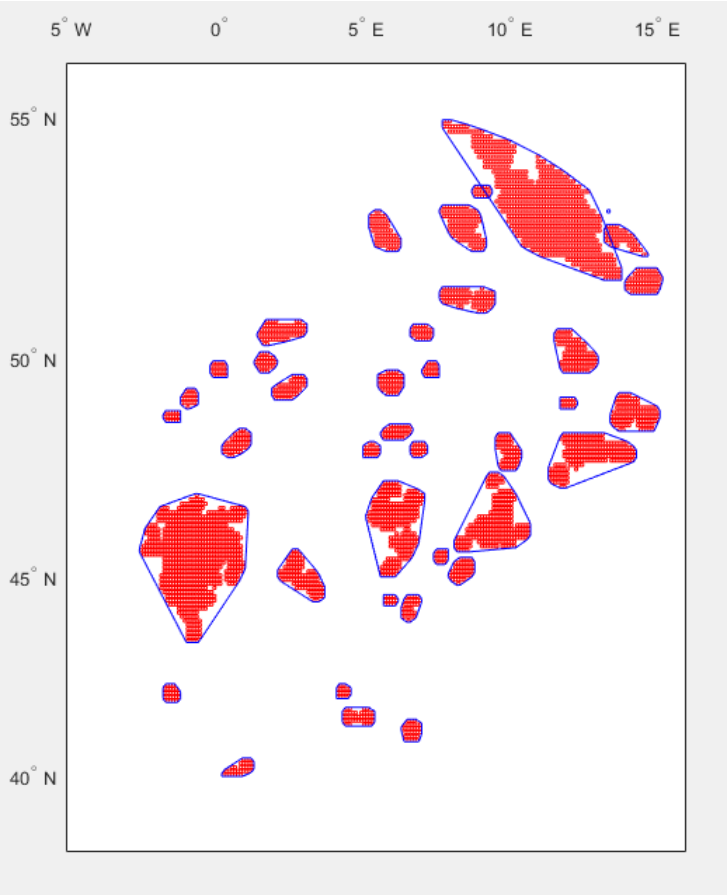



**Figura 3.8** Envolvente convexa de las regiones tormentosas en la proyección de Mercator.

Para la detección se utilizará el comando *polyxpoly*, el cual nos permite calcular, si existiesen, los puntos de intersección entre dos polilíneas, como pueden ser las aristas de un polígono, de forma que, así, se puede conocer si las envolventes convexas de dos polígonos se cortan entre sí.

Una vez detectado el corte, es decir, que la longitud del vector con los puntos de intersección que devuelve el comando sea mayor que 1, se colocarán todos los elementos distintos de cero de la fila asociado a los vertices del polígono z, en la fila de p, de forma que ahora sea un único polígono, fusión de los anteriores.

Es al nuevo polígono, fusión de los anteriores, el que se someterá a todas las operaciones anteriores de obtención de la envolvente convexa. Además, se eliminarán todas las variables asociadas al polígono z, pues no tiene cabida hablar de este polígono tras su fusión.

Se reducirá, por tanto, en uno el número de polígonos y se inicializará la variable de recorrido, z, a uno, con el objetivo de detectar si la nueva envolvente convexa corta a aquellas de otros polígonos que ya habían sido inspeccionados.

El caso más extremo, no considerado con el uso del comando anterior, se da cuando la envolvente convexa de un polígono engloba completamente a las de otros obstáculos. Para detectar esto, se usará el comando inpolygon, ya comentado anteriormente. Cuando el vector que devuelve este comando, de las mismas dimensiones que el vector de vértices del polígono que se quiere comprobar si está contenido en otro, es un vector que sólo contiene unos, entonces se procederá como anteriormente,

eliminando el rastro de aquel polígono que esté incluido en el otro.

Esto último se comprobará dos veces, con el objetivo de saber si p está incluido en z o viceversa.

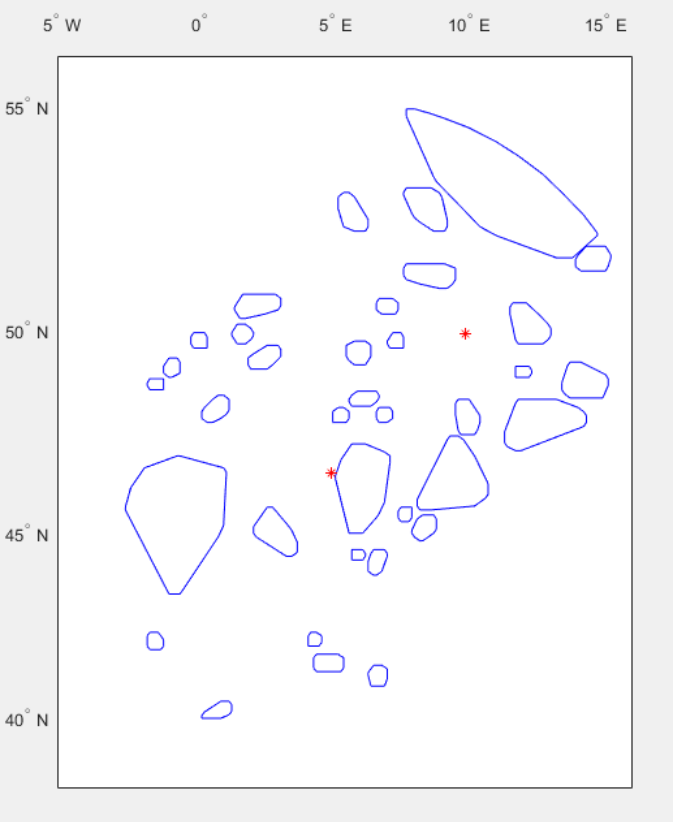



**Figura 3.9** Fusión de las envolventes de la imagen 3.8 en la proyección de Mercator.

Una vez obtenidos los polígonos definitivos a evitar por la aeronave (ver figura 3.9), se creará la matriz *match* que, simplemente, asigna un número, al que llamaremos número nodal, a cada uno de los nodos de los diferentes polígonos, es decir, a los elementos distintos de cero de la matriz de vértices de la envolvente convexa de los polígonos originales -*xv*2 y *yv*2-, con el objetivo posterior de facilitar la identificación de cada uno de esos vértices.

Conociendo los obstáculos tormentosos definivos que la aeronave va a tener que hacer frente durante el vuelo, ya se está en posición de avanzar al siguiente bloque.

## 3.3 Definición y obtención de los diferentes segmentos que componen el grafo de visibilidad

El objetivo de la presente herramienta es la determinación de forma dinámica de un camino que lleve a la aeronave desde su posición actual hasta el destino, con la restricción de evitar las regiones de riesgo meteorológico (actuales y previstas en el futuro cercano) y el objetivo de minimizar alguna actuación de vuelo (distancia recorrida, tiempo de vuelo, consumo de combustible, etc.). Asumiendo un problema plano, el camino que minimiza la distancia entre dos puntos evitando una serie de obstáculos poligonales pertenece a lo que se conoce con el nombre de grafo de visibilidad. Con la atmósfera en calma y volando a velocidad constante, el camino de mínimo tiempo verifica la

misma propiedad, ya que coincide con el que minimiza la distancia. Por el contrario, en presencia de viento, no se tiene ninguna garantía de que la trayectoria de mínimo tiempo pertenezca al grafo de visibilidad (nótese que el camino de menor tiempo entre dos puntos deja de ser la línea recta); sin embargo, restringir la búsqueda del camino óptimo al conjunto de posibles segmentos que pertenecen al grafo de visibilidad constituye una buena solución de compromiso entre optimalidad y simplicidad en la formulación. Adicionalmente, a la hora de construir el grafo de visibilidad, los segmentos rectilíneos no serán geodésicas sino loxodrómicas, lo cual se espera que tenga un impacto despresciable en la optimalidad, pero hace la solución compatible con la operativa real.

En esta sección, se prepararán los datos necesarios para construir el grafo de visibilidad, el cual permite conocer las posibles conexiones entre los diferentes nodos, que la aeronave va a poder seguir entre un punto de partida/origen (de nombre $P_0$) y un punto de llegada/destino ($P_D$), siempre y cuando, seguir una de estas conexiones no implique la inmersión en una de las regiones tormentosas, es decir, que exista visibilidad directa entre los nodos entre los que se vuela. La mayor parte de esta sección va a estar destinada a obtener dichos segmentos seguros, es decir, sin que la aeronave afectada por ninguna de las regiones de riesgo introducidas como input meteorológico.

Una vez obtenida esta información, se procederá con la obtención del camino que minimiza el tiempo de vuelo entre el origen y el destino.

Estas tareas comentadas van a poder verse reflejadas en el siguiente diagrama de flujo (imagen 3.10), tareas en las que se profundizará a lo largo de esta sección y las siguientes.

### 3.3.1 Definición de nodos y conexiones.

En esta sección, como se ha dicho anteriormente, se van a calcular las diferentes conexiones posibles que unen los nodos del sistema. Este apartado, únicamente, viene a mostrar las diferentes líneas de trabajo en la que se va a centrar esta sección.

Estos nodos no son más que los puntos sobre los que la aeronave va a poder pasar durante su vuelo de evitación de tormentas y sobre los que se tiene información de su posición. Algunos de estos van a ser aquellos que conforman los obstáculos a evitar, es decir, aquellos que pertenecen a la envolvente convexa de los polígonos estáticos obtenidos.

Además de estos, se consideran como nodos los puntos de origen y destino, los cuales se unirán mediante las diferentes conexiones posibles con los nodos anteriores.

Estos nodos se van poder conectar entre sí mediante los siguientes segmentos, segmentos que van permitir la formación del grafo de visibilidad:

- Segmentos que unen de forma directa el origen y el destino.
- Segmentos que unen los nodos de un mismo polígono, es decir, las aristas de los propios polígonos.
- Segmentos que conectan los nodos inicial y final con los polígonos tormentosos.
- Segmentos que conectan los nodos de los diferentes polígonos.
- Segmentos que conectan el origen o el destino con su nodo de escape, en el caso de verse estos encerrados en una región de riesgo meteorológico. (ver sección 3.6.2)

A continuación, se desarollará cada una de las conexiones enunciadas en este apartado, para, así, conocer cuáles serán los diferentes segmentos que la aeronave podrá seguir en su vuelo de eviatación de tormentas entre el origen y el destino.


**Figura 3.10** Diagrama de flujo para la obtención del grafo de visibilidad y camino de menor coste temporal.

### 3.3.2 Segmentos que unen de forma directa el origen y el destino.

Puede darse el caso en que el origen y el destino tengan visibilidad directa, es decir, que se pueda volar directamente desde el origen al destino sin atravesar ninguna región de riesgo, cumpliéndose, así, el objetivo para el que esta herramienta se ha diseñado. El hecho anterior va a hacer que al conjunto de segmentos posibles se deba añadir la unión directa entre el origen y el destino. Suele ser, además, este tipo de conexión aquel que permite un vuelo seguro con el menor tiempo posible. Por ello, su consideración es muy importante.

Como se comentará más adelante, puede darse el caso en que el origen, el destino o ambos, siempre y cuando no sea en el mismo obstáculo, estén encerrados por una de las regiones a evitar. Si esto ocurre, se calculará lo que se llamará nodo de escape, punto de la frontera de la región por el que la aeronave va a salir o entrar en el polígono. Estos puntos pasarán ahora a ser los nuevos origen o destino, ficticios, a considerar. En este caso, que exista visibilidad directa implica que es la unión entre estos nodos ficticios, la conexión que no va a atravesar ninguna región durante su seguimiento. Cabe destacar que si alguno de estos nodos no está encerrado, el nodo ficticio coincide con el real.

En caso de que tanto el origen, como el destino, estén encerrados en el mismo obstáculo no va a existir nodo de escape pues existe visibilidad directa entre los mismos.

Se puede pensar que el hecho de que el origen o el destino estén inmersos en una región de riesgo no puede o no debería darse, lo que haría carecer de validez a la herramienta. Se explicará más adelante, en el apartado 3.6.2, que esto no es así.

### 3.3.3 Segmentos que unen los nodos de un mismo polígono.

Otro de los tipos de conexión que la aeronave puede seguir para alcanzar el destino de forma segura, es aquel coincidente con las aristas de los polígonos obstáculo. Puesto que, en realidad, las regiones de riesgo están contenidas en el interior de los obstáculos que la aeronave evitaría con esta herramienta, que el avión avance por su frontera no supone ningún riesgo para la misma, pudiendo, por tanto, ser estos segmentos parte de la trayectoria de evitación de tormentas buscada.

Estas conexiones, como se ha comentado en párrafos anteriores, coinciden con las aristas de los polígonos (envolventes convexas de los obstáculos estáticos originales). Los nodos que conforman estas conexiones han sido almacenados en las matrices $xv2$ y $yv2$, cuyos números nodales se almacenan en la matriz match.

### 3.3.4 Segmentos entre punto y polígono. Cálculo y eliminación de aquellos no visibles.

En este apartado, se tratará la obtención de las diferentes conexiones entre los nodos origen o destino con los diferentes obstáculos estáticos, producto de las operaciones de la sección anterior. Los únicos segmentos que permiten realizar dichas uniones de una forma rápida y segura, sin cambios bruscos en el curso que sigue la aeronave, son las tangentes entre ellos. Se va a crear una función que devuelva uno (mediante la variable tangente, único output de dicha función) si el siguiente nodo de la envolvente convexa al que entra (se explicará el porqué de esto en párrafos posteriores) como input, es tangente al punto que se quiere analizar, input también de la función en cuestión. Cabe destacar que entre un punto y un polígono existen únicamente dos tangentes, número que indicará si las cosas se están haciendo bien. (Ver **A.8**)

Para calcular si por un nodo pasa la tangente entre una envolvente convexa y un punto ajeno a ésta, hay que proceder según se indica en el siguiente diagrama de flujo de la imagen 3.11, el cual se explicará detalladamente a continuación [11]:

- Nos montamos sobre las aristas del polígono, recorriéndolo en el sentido que se considere.
- Si para una arista, el punto externo -para este caso: origen o destino-, está a la derecha de la misma, pero para la siguiente, el punto se encuentra a la izquierda, o viceversa, entonces por el nodo de confluencia de ambas aristas pasa una de las tangentes buscadas (nodo siguiente al que entra como input).

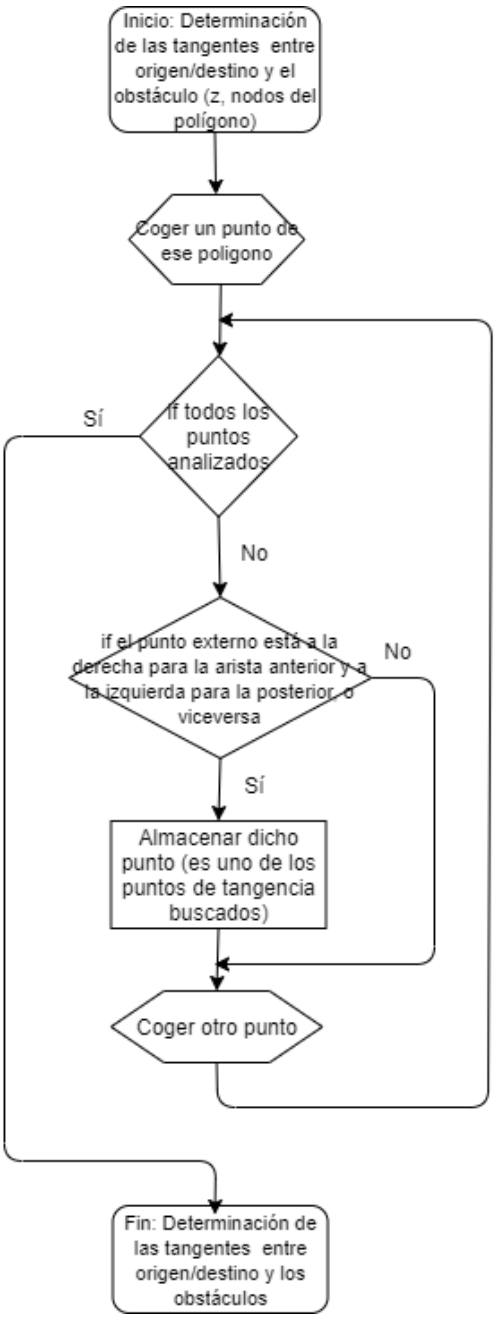



**Figura 3.11** Diagrama de flujo para el cálculo de las tangentes entre un punto y un polígono.

Para la comprobación anterior de si un punto está a la derecha o a la izquierda de las aristas, se usarán los vectores directores de las mismas, obtenidos mediante sus puntos de inicio y final, puntos que se corresponden con los nodos del polígono. Mediante una serie de condiciones, que serán aquellas que nos permiten identificar si el punto se encuentra a la derecha o a izquierda de un vector, se consiguirá calcular las tangentes entre un punto y un polígono. (Ver **A.8**). Las condiciones para dicha identificación se han establecido de forma que se tengan en cuenta todas las implicaciones de que un punto esté a la derecha o la izquierda de un vector, para cualquier orientación (horizontal, vertical, oblicuo...), de forma que se obtengan para cualquier caso, las tangentes buscadas.

Se llamará a dicha función para calcular las tangentes, tanto entre el origen y cada una de las envolventes convexas, como entre el destino y dichas envolventes. Se almacenarán las coordenadas de los puntos de tangencia obtenidos en las matrices xtan0 e ytan0, para las tangentes al origen, y en las matrices xtanD e ytanD, para las tangentes al destino. Se recuerda, que estos puntos se corresponden con los nodos siguientes a los que la variable tangente almacenaba el valor unidad.

Las imágenes siguientes muestran estas tangentes, tanto entre el origen y los obstáculos (imagen 3.12), como entre los polígonos tormentosos y el destino (imagen 3.13).

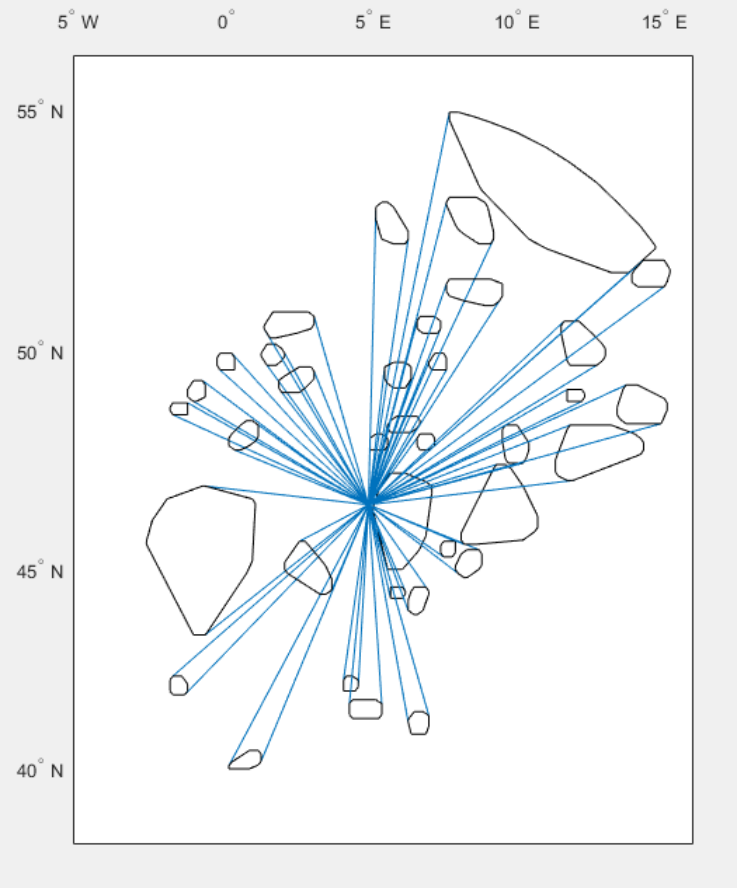



**Figura 3.12** Tangentes entre el punto de origen y los polígonos en la proyección de Mercator.

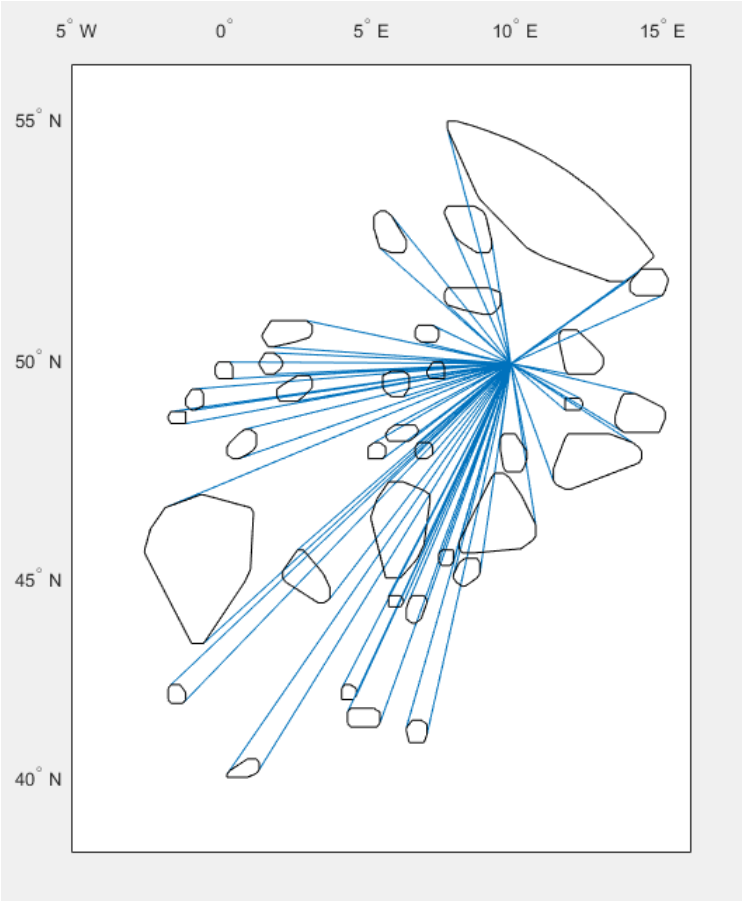



**Figura 3.13** Tangentes entre el punto de destino y los polígonos en la proyección de Mercator.

Como se dijo anteriormente, la aeronave seguirá las tangentes con el objetivo de llegar de forma rápida y sin riesgo de inmersión en una región tormentosa. Pero, para que esto última suceda, entre los puntos unidos por dicha tangente debe existir visibilidad directa, lo que implica que la tangente no puede cortar a las envolventes convexas de otros polígonos tormentosos. Por tanto, habrá que eliminar aquellas tangentes que no cumplan la anterior.

Para ello, se realizará una nueva función de nombre *cortetangentespuntopoligono* (ver **A.9**) que, dados las coordenadas de los puntos de origen, destino, las matrices que almacenan los puntos de tangencia y sus números nodales (*zvert0poly*, para las tangentes al origen, y *zvertDpoly*, para las tangentes al destino), así como los nodos de todas las envolventes convexas, permite calcular, para cada tangente, si esta intersecciona con alguno de los otros polígonos al que no es tangente. Si es así, el elemento asociado a esa tangente en la matriz de los números nodales de los puntos de tangencia (*zvert0poly* y *zvertDpoly*) se sustituye por el valor cero. Esto lo hará de la forma que se indica en la imagen 3.14, procedimiento que será explicado a continuación:

Se hará uso del comando *polyxpoly*, el cual, permite calcular la existencia de puntos de intersección entre dos polilíneas, como la que encierra un polígono. Esta almacenará en un vector dichos puntos, el cual tendrá dimensión nula en caso de no existir intersección entre ambas líneas. Por ello, se calculará, para cada una de de las tangentes , si estas cortan a algunos de los polígonos, distintos a los que son tangentes.

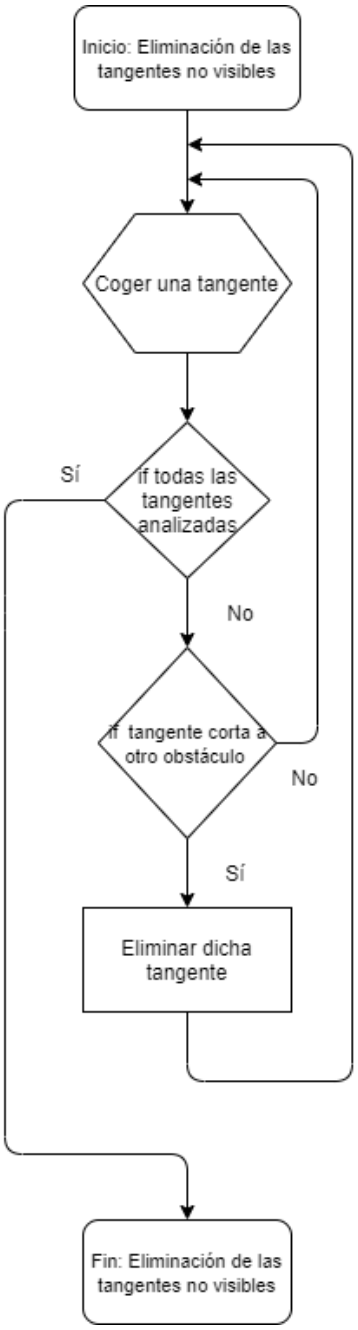



**Figura 3.14** Diagrama de flujo para la eliminación de las tangentes no visibles.

Como se dijo anteriormente, en caso de que se detecte que la tangente corta a alguno de los polígonos, es decir, que la dimensión del vector que devuelve el comando *polyxpoly* sea mayor que 1 -la longitud unidad implica que esa tangente es también tangente al polígono analizado- produciéndose, por tanto, durante el proceso de seguimiento de dicha tangente, la inmersión en una región de riesgo meteorológico, esta tangente se eliminará de las posibles sustituyendo por cero el número nodal del punto del polígono por el que pasa la tangente (recogidos en las matrices *zvert0poly* y *zvertDpoly*). El alterar únicamente los valores de estas matrices va a permitir facilitar los cálculos y evitar errores, pues no se están alterando los puntos originales de tangencia.

Esta función, además, será aprovechada para calcular si existiese visibilidad directa entre el punto de origen y destino (mediante la variable *Origen-destino*), incluso en el caso de que ambos puntos

estuvieran dentro del mismo polígono tormentoso (primeras líneas de la función), mediante el mismo comando (*polyxpoly*), comprobando si la línea directa de unión entre ambos corta a alguno de los polígonos considerados. Si es así, la variable *Origen-destino* se actualizaría a un valor nulo.

Como ya se comentó, en caso de que estén encerrados origen, el destino o ambos, pero en obstáculos diferentes, que exista línea directa entre ellos implica ahora que entre los puntos por los que sale o entra a la región de riesgo existe visibilidad directa. Esto también se tiene en cuenta en esta función.

Una vez que ya se conocen las tangentes visibles y si exisitiese o no visibilidad directa entre el origen y el destino (conexiones representadas en las figuras 3.15 y 3.16) va a ser interesante almacenar los puntos de tangencia anteriores en unas nuevas variables. Se crearán unos nuevos vectores que guardarán los números nodales no nulos que recogían los vectores *zvert0poly* y *zvertDpoly*. El objetivo de esto es preparar los datos necesarios para formar el grafo de visibilidad, pues se almacenan en estos vectores solo los nodos de los segmentos posibles entre los puntos de origen y destino, y los polígonos obstáculo. Estas nuevas variables van a recibir el nombre de *vectanpoint0*, para el origen, y *vectanpointD*, para el destino. Para esto último se hará un barrido en z, variable asociada al polígono tormentoso que se esté analizando, y en l, índice que irá entre 1 y el número de columnas de las matrices zvertpoly, es decir entre el valor unidad y dos, número correspondiente a las dos tangentes existentes entre un punto y un polígono. Se almacenarán en ellas, como se ha dicho en el párrafo anterior, únicamente los valores distinto de 0 recogidos en las matrices anteriores.

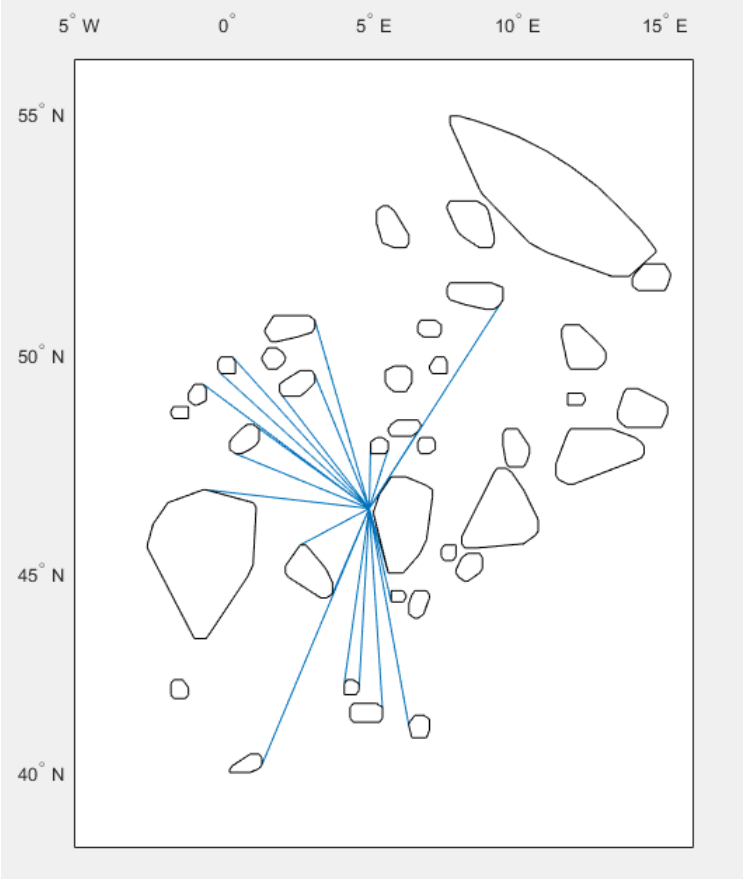



**Figura 3.15** Tangentes visibles entre el punto de origen y los polígonos en la proyección de Mercator.

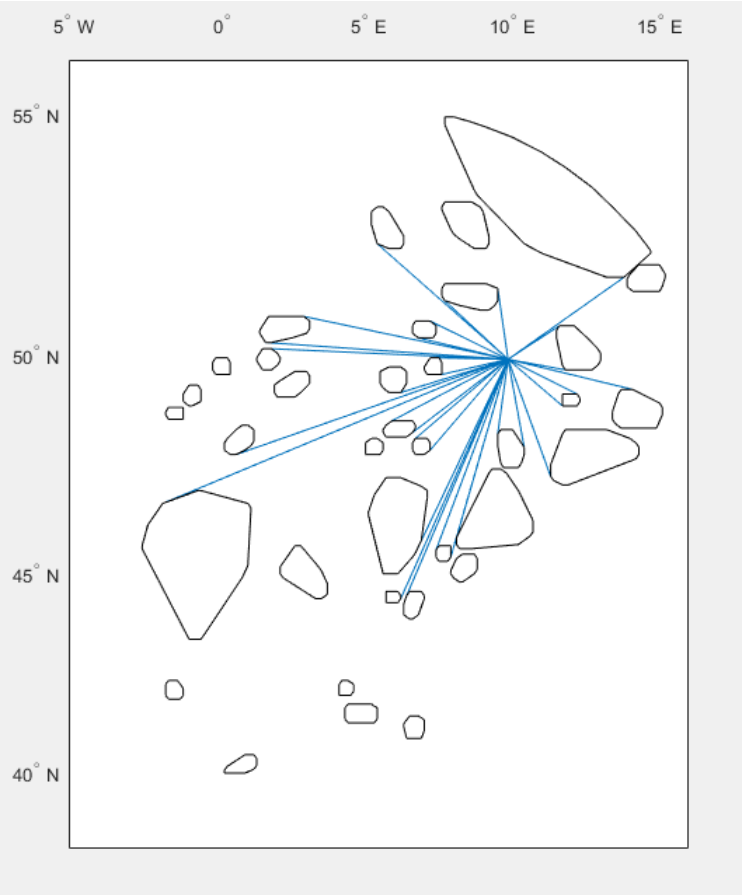



**Figura 3.16** Tangentes visibles entre el punto de destino y los polígonos en la proyección de Mercator.

### 3.3.5 Segmentos entre polígonos. Cálculo y eliminación de aquellos no visibles.

En este apartado se va a desarrollar la obtención de otro de los tipos de segmentos posibles, el que se corresponde con aquellos que permiten unir los nodos de los polígonos entre sí. Como se dijo en el apartado anterior, una forma rápida y segura de evitar estas regiones de riesgo consiste en el seguimiento de las tangentes, en este caso, entre los diferentes polígonos.

Una vez que la función anterior de cálculo de tangentes entre punto y polígono se ha desarrollado, la obtención de las tangentes buscadas se simplificará, pues se hará uso de ella para cumplir la finalidad de este apartado.

Existirán cuatro tangentes entre dos polígonos, dos exteriores y dos interiores. El cálculo de las mismas (ver figuras 3.17 y 3.18) se basará en las ideas del apartado anterior, en lo que a la búsqueda de los puntos de tangencia se refiere. Se creará una nueva función llamada *tangpolygontopolygon*. En ella se calcularán, para cada polígono z y p, distinto de z, por cuáles de sus puntos pasan las tangente buscadas. El objetivo de dicha función es, dados el punto de z que se está analizando, así como ambos polígonos de interés, y las matrices con los puntos de la envolvente convexa, devolver los índices de los puntos de ambos polígonos por los que pasará una de estas tangentes. (Ver **A.10**)

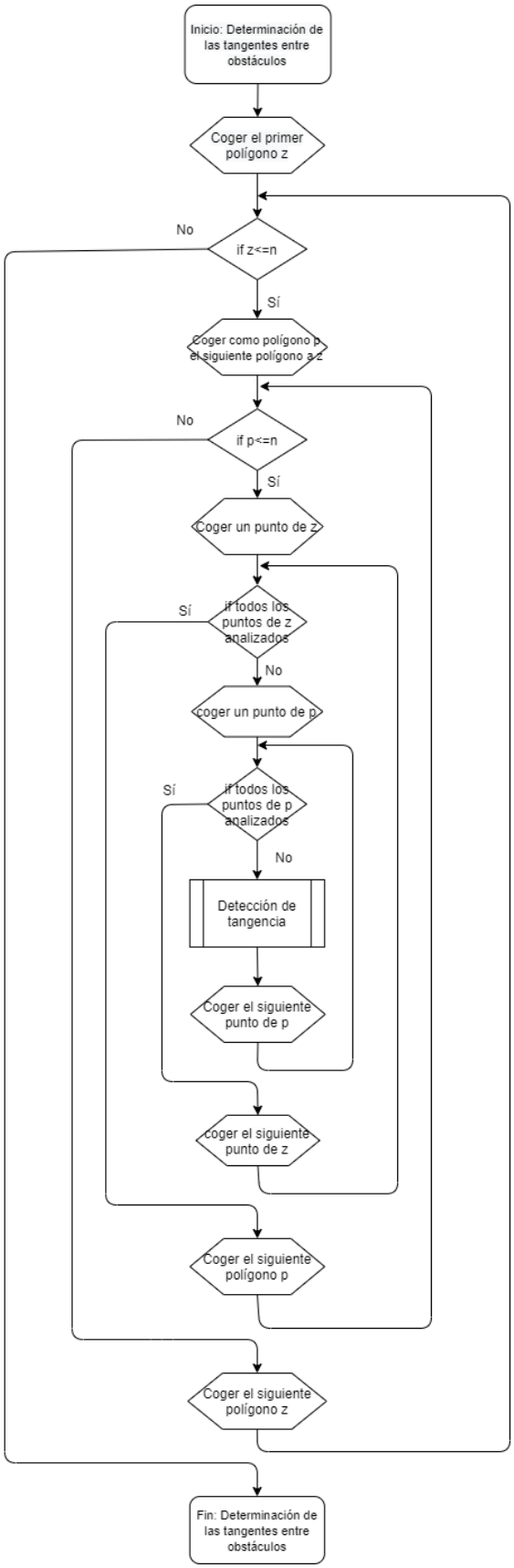



**Figura 3.17** Diagrama de flujo para el cálculo de las tangentes entre polígonos.

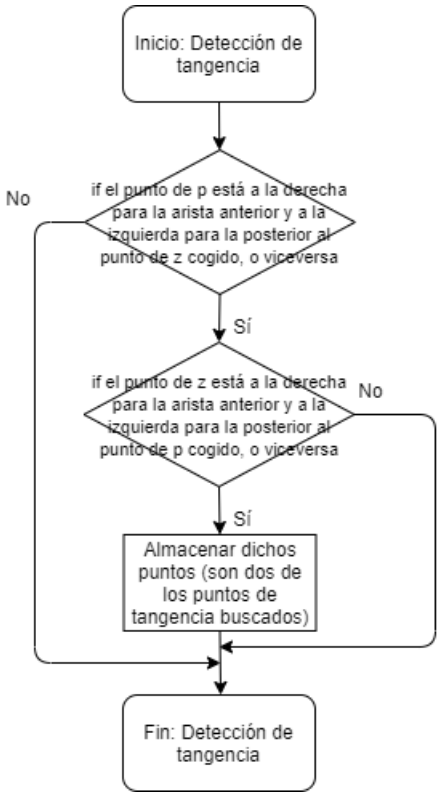



**Figura 3.18** Diagrama de flujo para el cálculo de las tangentes entre polígonos: detección de tangencia.

Como se dijo en el apartado anterior, los nodos por los que pasará una tangente entre un punto y un polígono, serán aquellos del polígono para el cual los vectores directores que entran y salen de ese nodo, ven al punto de análisis -punto exterior- a la izquierda y a la derecha, respectivamente, o al contrario.

Se hará un barrido, para cada uno de los puntos de los polígonos, para, así, calcular todas estas tangentes buscadas. Partiendo de la idea del párrafo anterior, habrá que comprobar que las aristas que entran y salen del nodo t+1 del polígono p, vean a la derecha y a la izquierda, respectivamente, o viceversa, al punto L del polígono z, pero, a su vez, que las aristas del polígono z que confluyen en L, cumplan lo mismo con el punto t+1. Es decir, existirá una tangente entre dos puntos de dos polígonos diferentes, si ambos puntos cumplen la condición del párrafo anterior para el otro polígono. Para ello, la función *tangpolygontopolygon* llamará dos veces a la función *tangpointtopolygon* del apartado anterior. Se almacenarán los índices de los respectivos puntos de tangencia en los vectores *indz* y *indp*.

Es muy importante que la condición de tangencia se dé, tanto para el punto del polígono p, como para el del polígono z, pues en caso contrario no existirá tangencia entre esos puntos.

Las componentes de los vectores anteriores se guardarán en unas nuevas matrices, de nombre *indtanz*, que, una vez que se hayan analizado todos los puntos L del mismo polígono z, va a tener por número de columnas 4, valor que coincide con el número de tangentes entre dos polígonos. También se crearán unas matrices de nombre *xtanz* y *ytanz*, que almacenarán las coordenadas de los nodos que conforman las tangentes, para el polígono z.

Como se ha dicho anteriormente, los bucles exteriores se harán para z y p, variables que designan

los polígonos que se están analizando. Para reducir el coste temporal y para evitar una duplicidad de las tangentes, p siempre debe valer, como mínimo, z+1, de forma que al avanzar z, el valor mínimo de p, también aumentará. Así, se evitará analizar dos veces los mismos puntos y acelerar el proceso de obtención de las tangentes.

Los índices de los vértices que conforman las tangentes se recogerán en las matrices de nombre *mat*, cuya característica principal es que son matrices con diagonal principal nula. En ellas, los elementos (z,p) y (p,z) son los índices de los vértices del polígono z y p, respectivamente, que forman cada una de las tangentes. Se crearán 4 matrices *mat*, pues existen 4 tangentes entre diferentes polígonos.

La representación de todas las tangentes entre polígonos se muestra en la imagen 3.19. Como se ha comentado anteriormente, estas tangentes son algunos de los posibles segmentos que la aeronave va a seguir con el objetivo de evitar atravesar una región de riesgo. Pero, no todos estos segmentos calculados son realmente posibles. Habrá ciertas tangentes que corten al resto de polígonos, lo que quiere decir que, si la aeronave sigue alguna de estas tangentes, acabaría inmersa en una tormenta, sometiéndose a todos los peligros que esto conlleva.

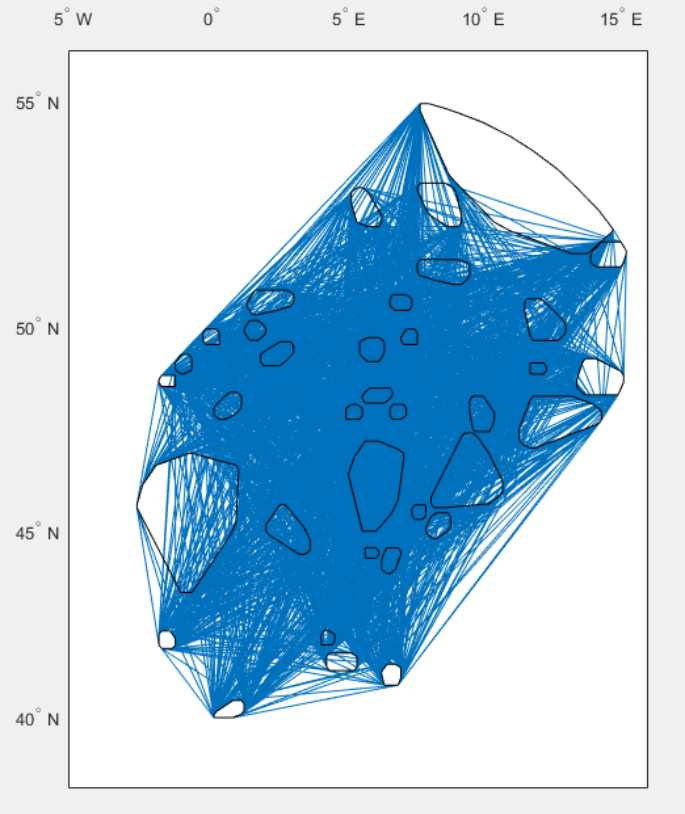



**Figura 3.19** Todas las tangentes entre polígonos en la proyección de Mercator.

Para solventar esto se desarrollará una nueva función que, dadas las matrices *mat*, que almacenan los índices de los puntos de tangencia entre los diversos polígonos, los vértices de los polígonos, a través de las matrices *xv*2 y *yv*2 , y la longitud de cada fila hasta el primer elemento nulo -matriz *chindice*-, permita eliminar aquellas tangentes que conlleven la inmersión en una de estas regiones de riesgo a evitar.

Esta función implementa un barrido en los polígonos p y z, con p distinto de z, de forma que, así, se tengan en cuenta todas las tangentes incluidas en las matrices mat, y un posterior barrido en t, variable que va a estar asociada al resto de polígonos, diferentes de z y p, y que servirá para, mediante la función *polyxpoly*, obtener si va a existir corte entre el contorno de los polígonos t y cada una de las tangentes asociadas a las variables z y p. (Ver **A.11**)

Si existiera corte entre las tangentes y el polígono, es decir, que la longitud del vector de salida de *polyxpoly* fuera mayor que el valor nulo, entonces los elementos de la matriz que almacena dicha tangente, se volverían nulos. En el programa desarrollado, en realidad, solo tornaría estos elementos nulos si la dimensión es mayor que uno, esto es debido a que un valor unidad de la longitud del vector de salida implica que la tangente analizada es, a su vez, tangente al polígono de estudio -t-, cumpliendo, así, las consideraciones anteriores.

Cabe destacar que t tiene que ser obligatoriamente distinto de p y z, pues, en caso contrario, el comando interpretaría el punto donde parte la tangente, como punto de intersección. Además, esto va a permitir reducir el tiempo de ejecución.

De esta función se obtendrán las matrices *mattan* que almacenarán los índices de los vértices de los polígonos que conforman las tangentes, en este caso, **visibles**.

Se representan en la imagen 3.20 las tangentes visibles entre polígonos, es decir, aquellas que no cortan al resto de polígonos a las que no son tangentes. Estas serán las conexiones que formen parte del grafo de visibilidad.

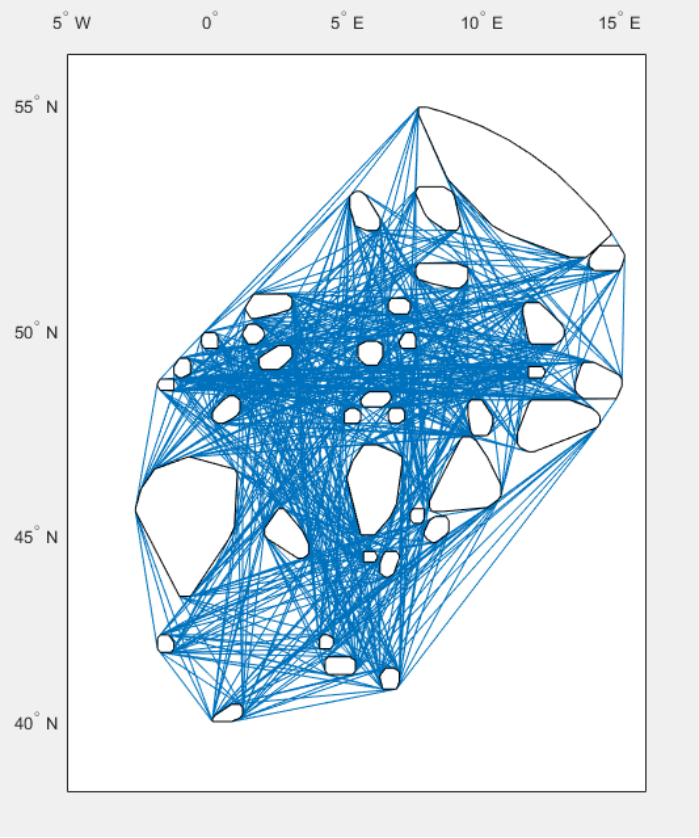



**Figura 3.20** Todas las tangentes visibles entre polígonos en la proyección de Mercator.

A continuación, al igual que se hizo para las tangentes entre los puntos de origen y destino con el resto de polígonos, se almacenarán en dos nuevos vectores los números nodales de los vértices anteriores. Se hará un bucle en z y p y que, en caso de que los elementos sean distintos de cero, almacenará en esos vectores el número nodal correspondiente al elemento (z,p) y (p,z), respectivamente, para cada una de las 4 matrices *mattan*.

### 3.3.6 Segmentos que conectan el origen o el destino con su nodo de escape.

Como se ha comentado en apartados anteriores, cuando el origen, el destino o ambos, en obstáculos diferentes, se ven encerrados o afectados por alguna de las regiones de riesgo a evitar, se definirá un nodo de escape por el que la aeronave saldrá o entrará en ella, según el caso.

El nodo de escape es aquel nodo conocido situado en la frontera de la región de riesgo que minimiza la distancia recorrida para salir del obstáculo, siempre y cuando se cumplan ciertas restricciones de cambio de curso.

Cuando esto ocurre, aparecerán unas nuevas conexiones de obligado recorrido consistentes en la unión del nodo encerrado con el nodo de escape, que se convirtirá en el nuevo origen o destino, ficticio. Es con este punto, con el que se calcularán todos los segmentos anteriores en los que intervenía el origen o destino real (conexión directa entre el origen o el destino, tangentes entre dichos puntos y los obstáculos...). Estos segmentos deberán ser tenidos en cuenta, pues formarán parte del camino de evitación de tormentas buscado.

Las variables *contchconf0* y *contchconfD*, almacenarán el número nodal de estos nuevos puntos descritos para el caso en que el origen y/o el destino estén encerrados, respectivamente. En caso de no estarlo, estas variables obtienen el valor de los números nodales del origen y destino reales (1 y *contch*, respectivamente). La obtención de los nodos escape y del segmento descrito en este apartado, se desarrollará más adelante, en el apartado 3.6.2.

Tras la descripción de este último tipo de conexión, ya se tienen, recogidos en diferentes matrices y vectores, los diferentes segmentos posibles que van a permitir la unión entre el punto de origen y el de destino. La imagen 3.21 recoge estos segmentos explicados, donde los segmentos rojos corresponden a las tangentes visibles entre el nodo de origen y los polígonos obstáculos, las azules; entre el destino y los polígonos, y las verdes; entre polígonos (para el ejemplo arbitrario de la imagen, no se observan algunas de las conexiones descritas). El conjunto de todos estos segmentos va a permitir formar el grafo de visibilidad.

Es importante para la obtención y representación de todos estos segmentos tener claro la proyección en la que se quiere trabajar, pues esta provocará una deformación de los polígonos y de las conexiones calculadas, pudiendo, tangentes que antes eran visibles en el mundo plano, no serlo tras la deformación de las mismas.

Por ello, durante la realización del algoritmo, como se explicó anteriormente, se realizó un cambio de coordenadas, el cual, pasaba de la proyección plana de Plate-Carrée a la de Mercator. Esto obligará a la representación de todas las variables en esta proyección.

Para representar todos estos segmentos se ha usado el comando *track2*, el cual permite representar diversos tipos de curvas, como las loxodrómicas o de curso constante, mediante la adición de una especificación, a partir de los puntos de origen y destino de la conexión. Esta especificación consiste

en una serie de abreviaturas que le permiten al comando entender el tipo de curva en el que se está interesado. Para el caso que atañe, se deberá añadir 'rh' para poder representar curvas loxodrómicas.

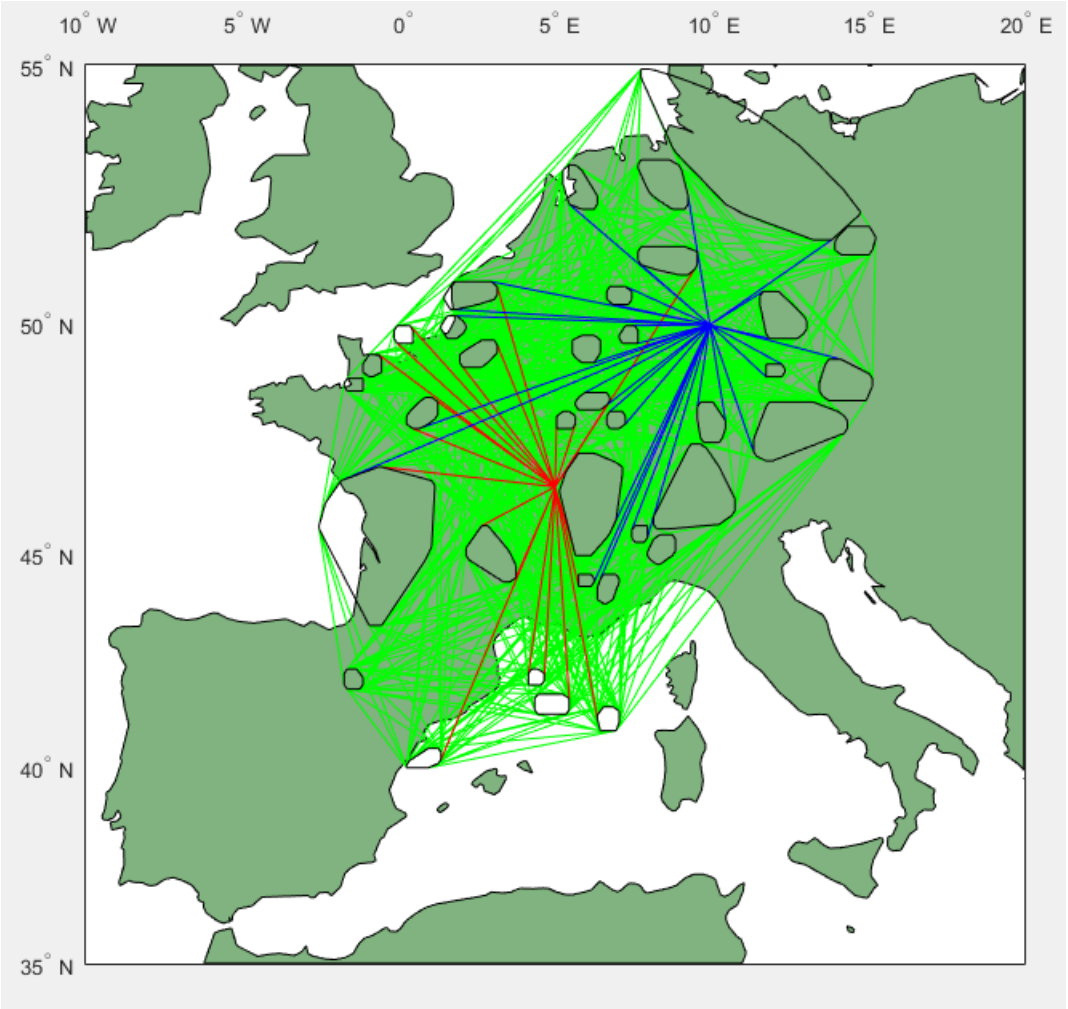



**Figura 3.21** Representación de todos los segmentos posibles en la proyección de Mercator.

Cabe mencionar que, previamente a la representación anterior, se ha deshecho el cambio de coordenadas [9], de forma que las coordenadas de los diferentes nodos son ahora directamente su latitud y longitud, al igual que los obstáculos originales.

Tras esto, ya se está en posición de formar el grafo de visibilidad necesario para calcular, posteriormente, el camino óptimo que une el nodo de origen y de destino.

## 3.4 Montaje del grafo de visibilidad

En este apartado se tratará el montaje del grafo de visibilidad, es decir, de aquel que contiene todos los posibles segmentos que la aeronave podrá seguir en su camino de evitación de tormentas. (Ver **A.12**)

Se define grafo como la representación de un conjunto de nodos conectados entre sí mediante una serie de segmentos. Para el caso analizado, este grafo recibe el nombre de grafo de visibilidad (imagen 3.21), en el que se unirán los nodos del sistema (origen, destino y vértices pertenecientes a la envolvente convexa de los obstáculos estáticos equivalentes) mediante los diferentes segmentos que se han calculado en las subsecciones anteriores (tangentes entre polígonos, entre los nodo origen y destino y los polígonos, las aristas de los polígonos...) visibles entre sí. Cada uno de estos segmentos puede tener asignado un cierto peso, asociado a diferentes aspectos, como pueden ser el tiempo que tarda en recorrer el camino, la distancia entre esos puntos, etc. Estos costes -denominados así, pues para el caso estudiado se tratarán estos pesos como costes invertidos en ir de un punto a otro- dependerán de en qué se está interesado en ponderar los caminos.

Asociada a una red, se puede definir una matriz, conocida como matriz de incidencia nodo-arco, con tantas filas como nodos y tantas columnas como arcos contiene la red. Dado un arco que conecta el nodo j con el nodo k, la columna asociada a ese arco tiene todas sus entradas nulas salvo dos: el elemento j-ésimo, que tiene valor 1, y el elemento k-ésimo, que tiene valor $-1$ [12]. Esta matriz permite almacenar una información similar a la que aporta grafo de visibilidad, pero sin necesidad del montaje del mismo. Puesto que, como se comentará más adelante, el comando usado para obtener el camino de mínimo coste temporal requiere como input el grafo de visibilidad, esta matriz no va a ser de gran utilidad para este trabajo.

Para montar dicho grafo se usará el comando *graph*, el cual, únicamente, necesitará datos de cuáles son los nodos extremos de cada una de las conexiones y el peso de las mismas.

A continuación, se mostrarán y desarrollarán las variables de interés creadas y los pasos seguidos para, una vez calculado todos los caminos visibles (fruto de la sección anterior y de la sección 3.6.2), formar el grafo de visibilidad y su matriz de incidencia nodo-arco.

Se creará una matriz de costes que, como se comentó, recoge el coste empleado en realizar cada una de las diferentes conexiones, almacenada dicha información en cada una de sus filas. El tipo de peso en el que se ponderarán los arcos a recorrer dependerá del objetivo último de optimización del camino a seguir. Puesto que el interés final de esta herramienta es generar la trayectoria segura de menor tiempo, lo más interesante parece ponderar dichos segmentos según el coste temporal empleado en recorrer cada uno de ellos (matriz *matcost*). Si el objetivo cambiase y se quisiera reducir, por ejemplo, la distancia recorrida entre el origen y el destino, se ponderarían los diferentes arcos según un coste en distancia (matriz *matcostd*), también implementada en el programa, en caso de que existiera ese cambio en los requerimientos de la herramienta. Estas matrices de costes tendrán dos columnas, la primera que hará referencia al número de la conexión y el segundo es el propio coste.

Se recuerda que todos los segmentos calculados se corresponden con curvas loxodrómicas, es decir, segmentos con curso constante. Por tanto, se procederá, para el cálculo de dichos costes, de la misma forma que para el cálculo inicial de la matriz de tiempos: se interpolarán los vientos en el punto intermedio de la conexión, se calculará la distancia recorrida y el curso seguido, que serán los datos de entrada para la función que permite integrar el inverso de la velocidad respecto a tierra y, así, calcular el tiempo empleado en realizar cada uno de los segmentos (función *loxodromicsequationint)*.

Se generarán dichas matrices (incidencia -en caso de necesitarla-, costes,...) por bloques, es decir, a medida que se va avanzando en el código, se irán rellenando de forma dinámica con la información referente a los diferentes tipos de conexiones, información adquirida de la sección anterior.

Se creará, también, una matriz de nombre *segment*, cuyas dimensiones son el número de conexiones por tres. El primer elemento de cada fila, asignará un número a la conexión, que coincidirá con el número de conexiones analizadas hasta el momento, pues al igual que las matrices anteriores, esta matriz se rellena de forma dinámica, de forma que no se puede conocer sus dimensiones reales hasta el final. El segundo y el tercer elemento se corresponden con el nodo de origen y destino de esa conexión, respectivamente. Es esta matriz, junto con la de costes, la que se introducirá como input para formar el grafo.

A continuación, se explica cómo la herramienta genera la información necesaria para formar el grafo de visibilidad, la cual se ha mencionado en las líneas finales del párrafo anterior.

Lo primero que el programa analiza es si existe visibilidad directa entre los nodos origen y destino, indicada esta información en la variable *Origen-destino*. En caso afirmativo, es decir, que la variable anterior almacene el valor unidad en su interior, se rellenará la primera fila de la matriz *segment* con el vector de valores [1, 1, *contch*]. Se recuerda que la variable *contch* es el número nodal del destino, es decir, del último nodo, obtenido durante la creación de la matriz de asignación de números nodales, *match*.

Puede darse el caso en que el origen y/o el destino estén inmersos en una región tormentosa. La primera fila de la variable *segment* recibirá, entonces, los valores [1, *contchconf0*, *contchconfD*]. Estas variables hacen referencia al número nodal del origen y el destino, y van a depender de la situación en la que se encuentren los mismos, situaciones que se van a analizar a continuación -para el caso del párrafo anterior, como se mostró, estas variables recibirían los valores 1 y *contch*, respectivamente-.

Cabe destacar que en caso de que se detecte que el nodo de partida y de destino están inmersos en una misma tormenta, la variable Origen-destino adquiere directamente el valor unidad. Aún estando afectado por una región tormentosa, este caso se tratará de forma análoga al anterior. Las variables *contchconf0* y *contchconfD*, mencionadas en el párrafo anterior, reciben el valor 1 y *contch*, respectivamente, pues es el origen real el que se une directamente con el destino real.

En el caso en el que el origen y/o el destino estén inmersos en una de las regiones tormentosas, siempre y cuando no estén afectados por el mismo obstáculo, se añadirán unas nuevas conexiones al grafo, consistentes en un segmento de unión entre el punto de origen o destino inmerso, con su nodo de escape, tal y como se dijo en la sección anterior.

Para esta situación y un valor unidad de la variable Origen-destino, va a existir visibilidad directa entre el origen, que puede ser el ficticio (la variable *contchconf0* obtendrá ahora el valor del número nodal del punto por el que se escapa del obstáculo origen), en caso de estar el origen encerrado por un polígono tormentoso, y el destino, que también puede ser ficticio (la variable *contchconfD* obtendrá ahora el valor del número nodal del punto por el que se escapa del obstáculo destino). En esta situación, la nueva fila de la matriz *segment* será análoga a la del caso anterior, pero con los valores de *contchconf0* y *contchconfD*, actualizados según el caso que les atañe. Cabe recordar que lo recién comentado es únicamente válido en el caso en que los obstáculos en los que podrían estar inmersos el origen y el destino son diferentes.

Como se ha comentado, si el origen o el destino están encerrados en una de estas regiones, una nueva conexión de obligado recorrido ha de ser añadida. Esta consiste en un segmento que une el punto encerrado con el de escape, información que se verá recogida en la matriz *segment* de la forma [contador, 1, *contchconf0*], para el origen, y [contador, *contchconf0*, *contch*], para el destino, donde

la variable contador únicamente viene a decir el número de arcos analizados hasta el momento, para así no almacenar la información de dos conexiones diferentes en la misma fila.

A continuación, se procederá con el resto de tipos de conexión. Se analizarán ahora las tangentes entre punto y polígonos. Los números nodales de los puntos de tangencia de los diferentes obstáculos vendrán recogidos en los vectores *vectanpoint0* y *vectanpointD*. En caso de que dicho punto exterior sea el origen u origen ficticio (el nodo de escape), el segundo elemento de la nueva fila de la matriz *segment* adquirirá el valor *contchconf0* (1 o el número nodal del nodo de escape, respectivamente, según el caso que se plantee), y el tercero; el número nodal de los diferentos puntos de tangencia almacenados en el vector *vectanpoint0*. Para el destino, contrariamente, el segundo elemento almacenará el número nodal de los nodos de los polígonos por los que pasa una de esas tangentes visibles (*vectanpointD)* y el tercero, el del destino (real o ficticio), valor que venía recogido en *contchconfD* .

Hasta el momento las conexiones eran unidireccionales, es decir, para una misma pareja de nodos, solo va a existir una conexión posible entre ellos, en un único sentido, pues el nodo de origen o destino está fijado. Pero las posteriores se van a poder recorrer en ambos sentidos, lo cual implicará que, para cada par de nodos, se definirán dos segmentos, que se verán reflejados en la matriz segments, con dos filas con los mismos elementos, pero intercambiados. Para la matriz de costes, en cambio, los valores no tienen por qué ser iguales, pues el curso cambia, lo que va a tener una repercusión en el tiempo, debido al efecto del viento sobre la aeronave, calculado el mismo en el punto medio del segmento.

Las siguientes líneas del programa van a quedar reservadas al rellenado de las matrices anteriores (*matcost* y *segment*) con los valores pertinentes asociados a los caminos restantes, bidireccionales, como se comentó en párrafos anteriores, que no son más que las aristas de los obstáculos, recogidas por polígonos en las matrices *xv*2 y *yv*2, y las tangentes entre polígonos, recogidos los números nodales de los puntos de tangencia en los vectores *vectanorden* y *vectanorden2*.

A continuación, se presentará una imagen (ver imagen 3.22) que resume cómo varían los diferentes elementos de la matriz segment, según el caso que se presente. También se muestra la estructura de la matriz de incidencia nodo-arco, aunque esta no va a ser de utilidad en las operaciones posteriores, como ya se ha comentado. En la imagen, el índice cont, simplemente hace referencia al nuevo arco que se está analizando, el índice z; al polígono de análisis, y el índice L, a cada uno de los elementos que conforman los vectores, como pueden ser los diferentes nodos de las filas de las matrices *xv*2 y *yv*2 o los diferentes puntos de tangencia almacenados en *vectanpoint0*.

## 3.5 Determinación del camino de evitación de tormentas

El apartado anterior permitía preparar la información necesaria para, mediante ciertos comandos de Matlab que se explicarán a continuación, obtener el camino seguro de menor tiempo.

Como se dijo anteriormente, usando el comando *graph* y añadiendo como entrada la segunda y la tercera columna de la matriz *segment*, con los números nodales de los nodos que conforman cada una de las conexiones, y la segunda columna de la matriz *matcost*, con el tiempo invertido en realizar cada uno de los caminos, se generará el grafo con todos los nodos y uniones ponderadas considerados.

| Tipo de conexión | Matriz de incidencia | Matriz Segment |
|---|---|---|
| Origen-destino=1, nodos no encerrados | Matinc(1,cont)=1<br>Matinc(contch,cont)=-1 | Segment(cont,2:3)=[cont,1,contch] |
| Origen-destino=1, nodos encerrados en el mismo polígono | Matinc(1,cont)=1<br>Matinc(contch,cont)=-1 | Segment(cont,2:3)=[cont,1,contch] |
| Origen-destino=1, origen, destino o ambos encerrados en diferentes polígonos | Matinc(contchconf0,cont)=1<br>Matinc(contchconfD,cont)=1 | Segment(cont,2:3)=[cont, contchconf0,contchconfD] |
| Tangentes origen (real o ficticio)-polígono | Matinc(contchconf0,cont)=1<br>Matinc(vectanpoint0(L),cont)=-1 | Segment(cont,2:3)=[cont, contchconf0, vectanpoint0(L)] |
| Tangentes destino (real o ficticio)-polígono | Matinc(vectanpointD(L),cont)=1<br>Matinc(contchconfD,cont)=-1 | Segment(cont,2:3)=[cont, vectanpointD(l), contchconfD] |
| Tangentes entre polígonos | Matinc(vectanorden(L),cont)=1<br>Matinc(vectanorden2(L),cont)=-1<br>-<br>Matinc(vectanorden(L),cont)=-1<br>Matinc(vectandorden2(L),cont)=1 | Segment(cont,2:3)=[cont, vectanorden(l), vectanorden2(L)]<br>-<br>Segment(cont,2:3)=[cont,vectanorden2(l), vectanorden(L)] |
| Aristas de los polígonos | Matinc(match(z,L),cont)=1<br>Matinc(match(z,L-1),cont)=-1<br>-<br>Matinc(match(z,L),cont)=-1<br>Matinc(match(z,L-1),cont)=1 | Segment(cont,2:3)=[cont, match(z,L), match(z,L-1)]<br>-<br>Segment(cont,2:3)=[cont, match(z,L-1), match(z,L)] |
| Origen-nodo de escape | Matinc(1,cont)=1<br>Matinc(contchconf0,cont)=-1 | Segment(cont,2:3)=[cont,1,contchconf0] |
| Nodo de escape-destino | Matinc(contchconfD,cont)=1<br>Matinc(contch,cont)=-1 | Segment(cont,2:3)=[cont,contchconfD,contch] |

**Figura 3.22** Tabla con los valores de la matriz de incidencia nodo-arco y segment según la conexión analizada.

Para el caso considerado, puesto que se conoce la dirección en la que los caminos son recorridos, es necesario usar el comando *digraph*, donde el primer y el segundo nodo que demanda como input, se corresponden con el origen y el destino de dicha conexión, respectivamente.

A continuación, mediante el comando *shortestpath* se calculará el camino de mínimo coste entre dos puntos. Recibirá como entrada el grafo cuyos segmentos han sido ponderados previamente, y el número nodal de los puntos de origen y destino.

Este comando lo que hace es ejecutar el algoritmo de Dijkstra [13]. De forma simplificada, lo que hace es calcular todos los costes que supondrían de ir de cada uno de los nodos al resto, a partir de un nodo origen, saltando de un nodo a otro, en cada iteración, tomándose dicho nodo como el origen en el que calcular los costes. Puesto que se introduce como input el nodo destino, el algoritmo devolverá el camino de menor coste, entre el origen y el destino.

El comando anterior nos proporcionará los números nodales de los nodos que pertenecen al camino de mínimo coste temporal. Para conocer la correspondencia entre el número nodal y las coordenadas reales del nodo, se formó la matriz de coordenadas. Para el rellenado de esta matriz se usará el comando find, que irá buscando el índice de cada número nodal, el cual estará colocado en la

misma posición que el nodo original en las matrices xv2 e yv2, obteniendo de ellas, las coordenadas de los nodos.

En la imagen 3.23 se puede observar, para el ejemplo arbitrario escogido, la representación de dicha trayectoria segura. En dicha imagen, en color rojo, se observa el tramo de trayectoria que debería seguir la aeronave para evitar, de manera eficiente, los obstáculos estáticos calculados. El segmento magenta hace referencia al tramo del camino que recorrería durante 5 minutos.

Pero, en realidad, el tramo en rojo no va a formar parte de la trayectoria segura final. Esta trayectoria se actualizará cada 5 minutos, mediante la realización de nuevas iteraciones con todas las operaciones descritas anteriormente.

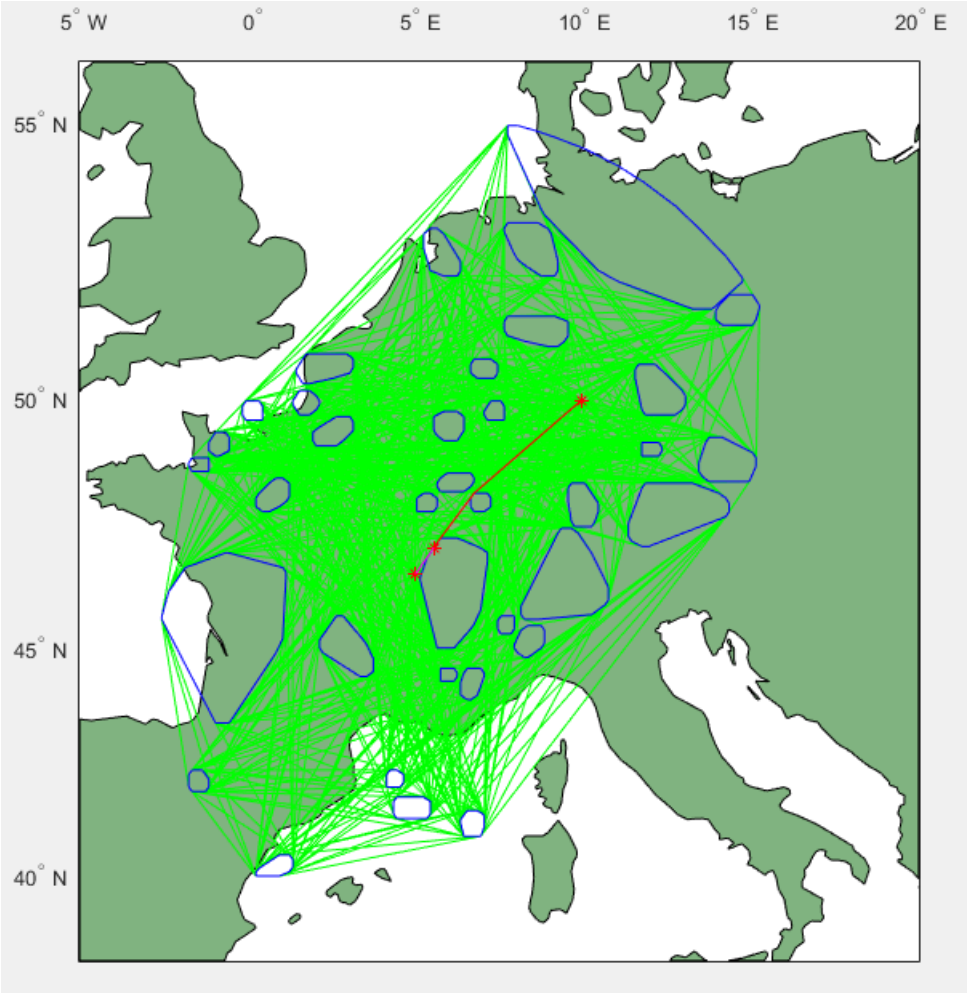



**Figura 3.23** Representación de todos los caminos posibles y la trayectoria de mínimo tiempo para la primera iteración del algoritmo en la proyección de Mercator.

## 3.6 Algoritmo de horizonte deslizante y maniobras de evasión

### 3.6.1 Algoritmo de horizonte deslizante

Como se comentó al comienzo de este capítulo, en la realidad lo que se evita son obstáculos meteorológicos provenientes de predicciones del nowcast inicial provisto por el radar meteorológico:

las regiones de riesgo reales pueden desplazarse en el espacio, a causa del viento o de otros factores, unirse, dividirse o disiparse, cambiando la estructura de las mismas con respecto a la que se tenía inicialmente y pudiendo diferir esta de las predicciones que se tenían de ella. Si esto pasa, por tanto, se estarían evitando regiones que poco tienen que ver con la realidad, y pudiendo, incluso, ser las tormentas reales, atravesadas por el camino de evitación de los obstáculos equivalentes.

Por ello, surge la necesidad de este algoritmo. De forma sucinta, este algoritmo consiste en ir actualizando la trayectoria de mínimo coste según una nueva información meteorológica que es recibida. Cuando transcurra el tiempo necesario para recibir una nueva información meteorológica (en la aplicación de este TFG, 5 minutos) la aeronave se encontrará en un punto distinto (punto que se obtiene tras aplicar el algoritmo presentado en el diagrama de la imagen 3.24), el cual se toma como el origen para la búsqueda del camino más adecuado hasta el destino. La porción de trayectoria hasta el punto actual no se ve modificada (ya que el pasado no puede cambiarse), pero el trozo que conecta el punto en que se encuentra la aeronave con el destino (tramo rojo en la imagen 3.23) podrá cambiar con respecto de la iteración anterior (al producirse cambios en la predicción meteorológica). Este algoritmo finalizará, generalmente, cuando el origen de la siguiente iteración coincida con el destino del vuelo o cuando se prevea alcanzar el destino en un tiempo inferior al tiempo de actualización de la información meteorológica.

Como se dijo en el párrafo anterior, se recibirá una nueva información meteorológica sobre las regiones de riesgo en el espacio a los 5 minutos. Durante este tiempo la aeronave estará volando siguiendo la trayectoria de mínimo coste temporal con la información meteorológica anterior. El objetivo será ahora calcular el punto que sobrevuela al cabo de esos 5 minutos, el cual se convertirá en el nodo origen para la iteración posterior.

Para calcular el nuevo origen, lo primero que se va a hacer es calcular el tiempo necesario para llegar a cada uno de los nodos que pertenecen a camino óptimo (vector P). Para ello, vamos a crear la variable *Costecam*, de las mismas dimensiones que el vector con los nodos pertenecientes al camino de mínimo coste (output del comando *shortestpath*), que acumulará en cada uno de sus elementos el tiempo que empleado en ir desde el origen a cada uno de los nodos, siguiendo el camino calculado, de forma que el último elemento contenga el tiempo total en completar dicho camino para esa iteración. **A.13**

Para esto se hará un bucle que permita buscar el número de la conexión con nodo final, el analizado (nodo P(i)) y el origen, el nodo anterior (P(i-1)) en el vector del camino óptimo, información que se buscará, con el comando *find*, en la matriz *segment*. Una vez que se haya encontrado dicho número, que se corresponderá con la fila donde esa información ha sido almacenada, se cogerá el coste de realizar dicho arco en la matriz *matcost*, que se encuentra en la fila obtenida y en la segunda columna de esa matriz. Se cogen los elementos de esa matriz, pues, se recuerda que cada uno de los segmentos fueron ponderados según el coste temporal empleado en completar dichos segmentos, tiempos almacenados en dicha matriz *matcost*.

En cada iteración del bucle del párrafo anterior, se sumará el coste temporal recién obtenido al del elemento anterior del vector *Costecam*, pues lo que se está calculando es el tiempo en llegar desde el origen a cada uno de los nodos del camino óptimo. Como cabría esperar, el valor del primer elemento del vector *Costecam* es nulo, pues el tiempo en llegar a un punto desde donde el avión parte es 0.

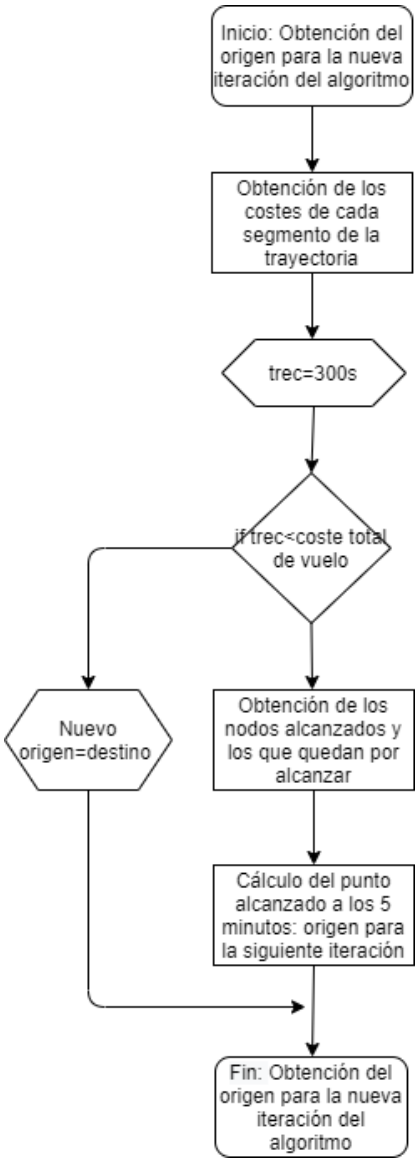



**Figura 3.24** Diagrama de flujo para la obtención del punto donde se actualiza la trayectoria.

El tiempo que tardará el avión en llegar al nuevo origen de la iteración siguiente va a ser de 300 segundos (equivalente a 5 minutos), tiempo que se va a almacenar en una variable de nombre *tstop*, punto donde la información meteorológica va a ser recibida. Con este valor se va a calcular entre qué nodo y qué nodo está situado dicho punto, puntos que recibirán el nombre de *nodoant* y *nodopos*, respectivamente. El objetivo principal de conocer estos nodos, es conocer el curso que el avión va a tener cuando se dirija al punto buscado, que será el mismo que el avión siga entre *nodoant* y *nosopos*. En caso de que *nodoant* sea distinto del origen inicial, se actualizará la variable *tstop*, restando a esos 300 segundos, el tiempo empleado en llegar a ese nodo, pues ya va a llevar un cierto tiempo recorrido (tiempo almacenado en la variable *Costecam*).

Antes de continuar con la explicación, es necesario destacar que si esos 300 segundos superan el tiempo que se tarda en llegar desde el nodo inicial al final, entonces para la siguiente iteración se tomará como nodo origen, el nodo de destino -se comentará más adelante que, en este caso, el algoritmo se habrá completado y se saldrá del bucle que implementa este algoritmo-.

Una vez que se haya calculado el curso a seguir entre los nodos con la función *loxodromicequations*, comentada al inicio de esta memoria, se requerirá calcular la distancia recorrida hasta llegar al nuevo origen de la siguiente iteración. Para ello, se usará el comando *ode45* y la función *loxodromicequationsint*, la cual permitía el cálculo de la velocidad respeto a tierra. Integrando su inverso con el comando anterior, como ya se explicó, se obtendría el tiempo empleado en el recorrido.

El objetivo ahora no es integrar la velocidad entre unos límites de integración fijados (la distancia entre los puntos *nodoant* y *nodopos*), si no calcular el límite que haría que la integración de la velocidad dé el tiempo buscado (*tstop*). Esto se hará mediante la adición al comando *ode45* de una especificación adicional consistente en una función evento. (Ver **A.14**)

El objetivo de esta función evento va a ser parar la integración cuando se alcance el valor del tiempo integrado requerido, es decir, para este caso, cuando, para una distancia recorrida, se alcance *tstop*. En este caso *ode45* dejará de integrar, dando dicha distancia como salida. Esta va a ser la que separa *nodoant* del punto que se está buscando.

Mediante una nueva función llamada *loxodromicequationsinv* (Ver **A.14**) que, dados los puntos de origen del segmento, el curso y la distancia recorrida, permite, a partir de las ecuaciones loxodrómicas, obtener el punto buscado, nodo que va a ser, como ya se explicó anteriormente, el origen para la siguiente iteración donde nueva información es recibida y analizada.

Las coordenadas de los nodos por los que pasa en su trayectoria entre el punto inicial original y el destino, incluyendo los orígenes de las diferentes iteraciones, van a ser almacenados en las matrices *nodospaso*, con el objetivo, además de conocer los puntos clave por los que la aeronave se mueve, permitir una posterior representación gráfica de las trayectorias seguidas.

A modo de resumen, este algoritmo permitirá la actualización de la trayectoria de evitación de regiones de riesgo cuando se reciba una nueva información meteorológica. Este se ejecutará cada 5 minutos y finalizará cuando el punto de origen de la iteración coincida con el destino, habiendo cumplido la misión de la aeronave, o cuando el vuelo sea más largo de lo previsto y no se tiene suficiente información del entorno para dichos instantes de vuelo, situación que se daría cuando todos los nowcasts introducidos como input hayan sido analizados. Esto último ocurre, pues, para este caso, sólo se han provisto datos meteorológicos para un vuelo un máximo de una hora y media de duración. En caso de que se alcance un tiempo mayor al anterior (contabilizado en el programa con la variable bucle, que lleva un conteo de las iteraciones, realizadas cada 5 minutos), se saldrá del bucle que implente este algoritmo, uniendo de forma directa el nodo origen actual con el destino, al desconocer la información meteorológica en dichos instantes.

La aplicación del algoritmo de horizonte deslizante, tiene como ventaja principal el actualizar la trayectoria a seguir para sortear los nuevos obstáculos que se presenten, obtenidos a partir de la información meteorológica provista, los cuales serán, en su mayoría, una variación de los evitados en la iteración anterior. Esto va a permitir reducir la posibilidad de inmersión en uno de estos polígonos, haciendo más seguro el vuelo, así como más rápido, pues se irán calculando nuevos caminos óptimos en cada una de las iteraciones.

### 3.6.2 Maniobras de evasión

Sea una iteración de este algoritmo. Puede darse el caso de que, debido al movimiento de las regiones tormentosas que se intentan evitar, en el nuevo nowcast meteorológico el origen y/o el destino de la aeronave estén inmersos en una de estas regiones cuando previamente no lo estaban (esto es

culpa de la interpolación temporal realizada y de no tener información meteorológica en tiempo real).

Lo primero que habrá que hacer será detectar si en la nueva iteración estos nodos (origen y destino) están dentro de una región tormentosa, así como almacenar, si se diera lo anterior, el polígono en el cual están encerrados. Esto se conseguirá mediante el uso del comando *inpolygon* y un barrido en z, variable que va a permitir analizar lo comentado para cada uno de los obstáculos -en caso afirmativo, se tornarán a uno unas variables: *puntoordentro*=1, si es el origen el que está encerrado por alguno de esos obstáculos meteorológicos y/o *puntodesdentro*=1, si es el destino el que está inmerso), y se almacenará el obstáculo que los encierra, en las variables *zorconf*, para el origen, y *zdesconf*, para el destino-.

Como cabría esperar, en caso de que ninguno de estos puntos esté afectado por alguna de las regiones tormentosas, todas las variables descritas almacenarán el valor nulo, no realizándose, así, la maniobra de evasión.

Antes de explicar la forma de proceder para salir de esos polígonos, lo que se va a conocer con el nombre de **maniobra de evasión** (ver imagen 3.25) se hará una distinción. En caso de que ambos puntos estén dentro del mismo polígono (mismo valor de las variables *zconf*), se tornarán las variables *puntodentro* a 0, de forma que, así, la maniobra de evasión del polígono no se ejecute, pues existe visibilidad directa entre los puntos. Aunque el programa tenga en cuenta este caso, no se dará en la realidad. Se realizarán vuelos de espera u otras maniobras previas, que evitarán que el avión entre en dicha región.

Si es el punto de origen el que se ve encerrado en una de estas regiones, se tratará de salir de él por uno de los nodos del polígono conocidos. Este se tomará como nuevo nodo origen (ficticio), calculando todas las conexiones descritas anteriormente con este punto, como se haría con el original. Este punto es el que se comentaba en apartados anteriores (subsección 3.3.1) y será el denominado punto de escape de ese obstáculo tormentoso. (Ver **A.15**)

Para calcular dicho punto se procederá de la siguiente forma:

**1.** Se calculará la distancia y el curso entre cada uno de los vértices de la envolvente convexa y el origen encerrado.

**2.** Se ordenarán distancias obtenidas en el paso anterior de forma ascendente (nuevo vector *rord*). Es decir, este vector almacenará las distancias, ordenadas de menor a mayor, entre el origen encerrado con cada uno de los nodos del polígono que lo encierra.

**3.** Si el origen encerrado es diferente del inicial, es decir, que no es la primera iteración del algoritmo y que, por tanto, la aeronave ya lleva recorrido un tramo del camino seguro de menor tiempo, van a existir restricciones en el cambio de curso que debe sufrir la aeronave para salir de la región. Es decir, el curso que llevará para salir del obstáculo no puede distar en gran medida del que llevaba anteriormente (*course1*).

Por ello, se analizarán los cursos almacenados, previamente ordenados según sus distancias correspondientes en *rord*. El primer punto del vector *rord* que cumpla que la variación del curso con respecto al que llevaba sea menor que un máximo, se tomará como nuevo origen, saliéndose del bucle que realiza esta comprobación. Para los casos que se muestran, la variación máxima admisible se ha considerado de 45º

Se calcularán los tiempos empleados en ir del origen a ese punto, así como otras variables que serán de utilidad más adelante.

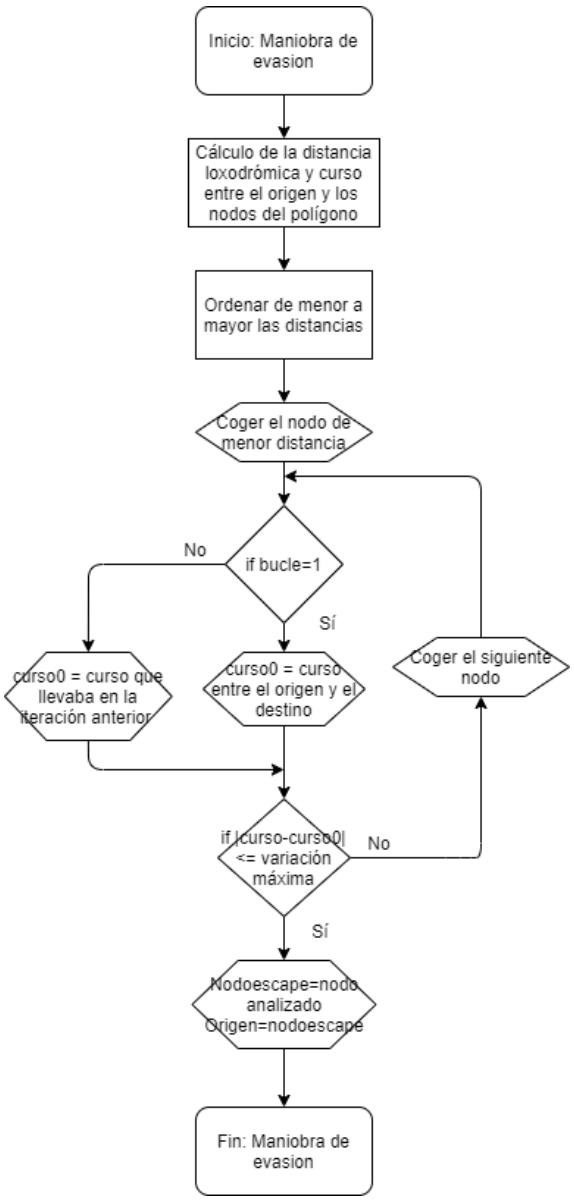



**Figura 3.25** Diagrama de flujo para trazar la maniobra de evasión.

Para la realización correcta de este algoritmo de escape, habrá que tener en cuenta lo siguiente: la detección de si las variaciones de curso son bruscas dependerá del signo del curso, cambiando las condiciones para dicha detección. Por ello, se ha utilizado el comando *if* para comprobar si el signo entre el curso del elemento de *rord* y el que llevaba (*course1*) son el mismo o no, con el objetivo de plantear, así, las condiciones pertinentes para cada uno de los casos. Estas condiciones se muestran en el apéndice **A.15**

También se ha analizado el caso en que, inicialmente, en el despegue, la aeronave ya esté inmersa en una región tormentosa. En este caso no existiría curso anterior. Para evitar un gran desvío de la trayectoria entre el punto de origen y de destino reales, se tomará como curso inicial (*course1*) el que llevaría siguiendo el segmento que une directamente estos nodos. Es decir, se analizarán los cursos almacenados según los elementos de *rord* hasta encontrar uno que no se desvíe más de un

cierto ángulo máximo del curso que llevaría siguiendo un segmento que una el origen con el destino de forma directa. Análogamente, se ha considerado que esta desviación no puede superar los 45º.

Aunque el programa tenga en cuenta este caso, en la realidad se esperaría a que el tiempo mejorase para iniciar el vuelo y evitar, así, problemas innecesarios.

Para el caso en el que sea el destino el que esté inmerso en una de estas regiones peligrosas, se procederá de la siguiente forma. Al contrario que en el caso anterior, primará recorrer la menor distancia posible en el interior del obstáculo. Por ello, en lugar de mirar limitaciones por cambio de curso, se tomará como nodo de escape, el correspondiente al primer elemento del vector *rord*, siendo el punto más cercano al destino. Al igual que para el origen, este punto, se convertirá en el nuevo destino.

Una vez determinado los nuevos origen y destino, se buscará el número nodal asociado a los mismos (*contchconf*), valores que van a ser de gran utilidad para el cálculo del camino óptimo.

La unión de los estos nuevos puntos calculados, los nodos de escape, con los respectivos nodos encerrados, conforman el otro tipo de conexión restante, explicada de forma sucinta en la sección 3.3. Al conocerse perfectamente los números nodales del origen y del destino reales (1 y *contch*, respectivamente) y de los nodos de escape, estas conexiones quedarán totalmente definidas. Estos segmentos permitirán a la aeronave salir, en caso de ser el origen el encerrado, o adentrarse en las regiones para alcanzar el destino. Estas conexiones van a ser de obligado recorrido e indispensables para poder calcular el camino seguro de mínimo coste que pueda enlazar el origen y destino reales.

Como se comentó a anteriormente, es a estos puntos a los que se les van a calcular las tangentes, variando un poco los programas cuando se dé este caso para evitar errores, olvidando los nodos originales (Ver figuras 3.26 y 3.27).

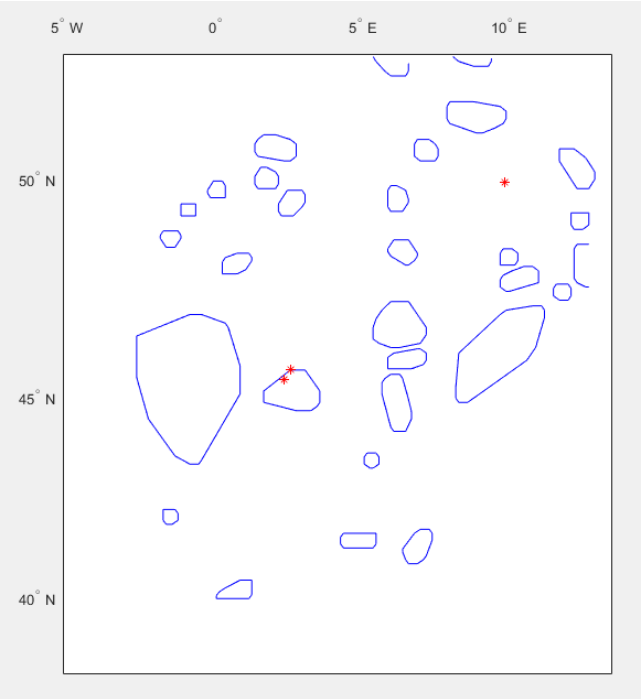



**Figura 3.26** Detección de la necesidad de realizar una maniobra de evasión.

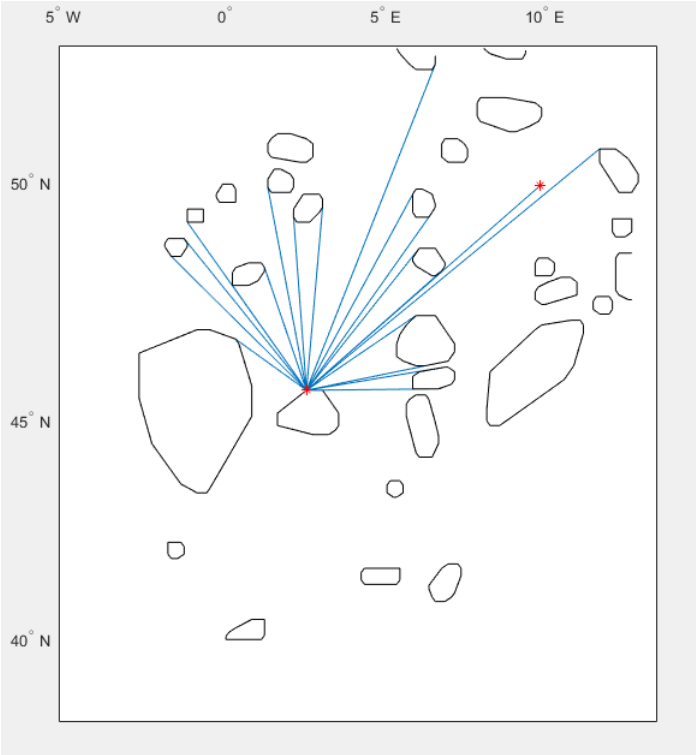
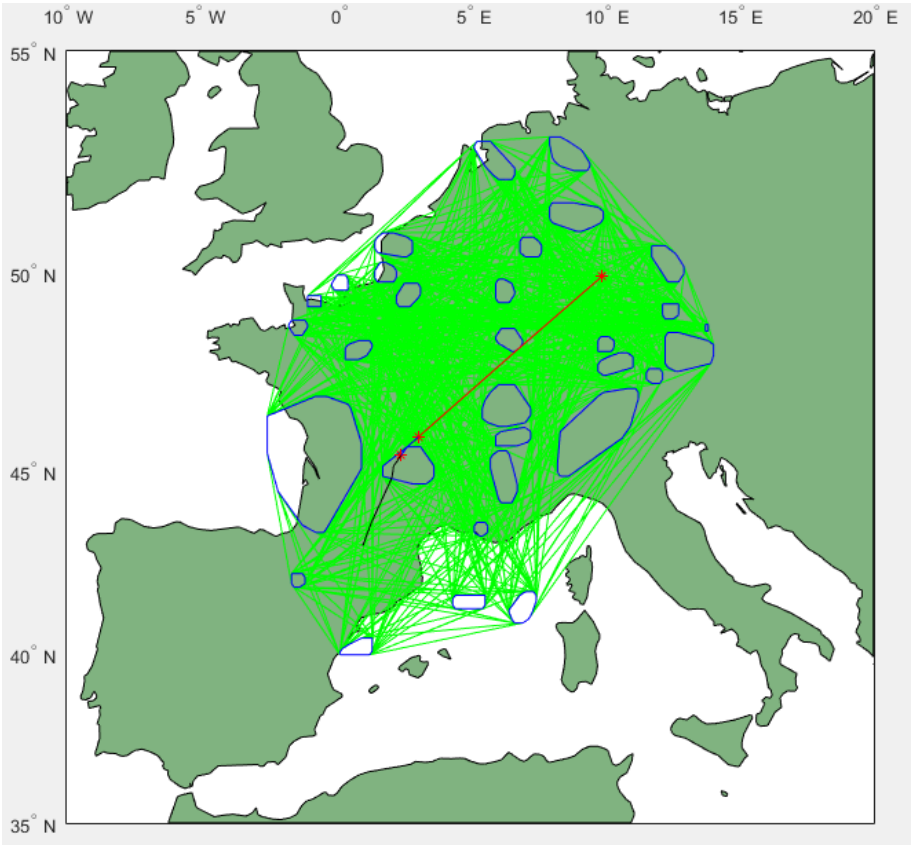


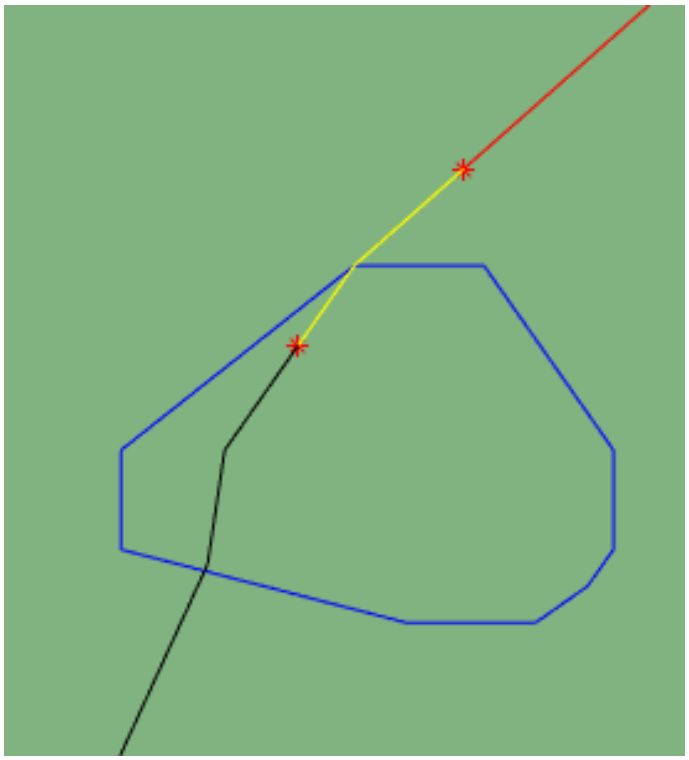

**Figura 3.27** Tangentes entre el nodo de escape y los polígonos en la proyección de Mercator.

En las imágenes siguientes se presentan diferentes trayectorias de evitación de tormentas con los casos comentados anteriormente. La imagen 3.28 a) muestra esta trayectoria cuando en una de las iteraciones, distinta de la inicial, la aeronave se ha visto inmersa en una región de riesgo , y en la b), la maniobra de escape que ejecutaría la aeronave para salir de ese obstáculo. La imagen 3.29 muestra la trayectoria que debería seguir la aeronave inicialmente cuando tanto el origen como el destino están inmersos en diferentes obstáculos tormentosos. Estas imágenes han sido obtenidas para otros ejemplos arbitrarios diferentes del inicial.

La pregunta que puede surgir a continuación es: ¿Cuál es la validez del algoritmo si, por la forma de interpolación temporal de la información, la aeronave puede verse inmersa en uno de estos obstáculos en la iteración siguiente? Como se comentó en la sección inicial de tratamiento de los nowcasts proporcionados, estos son previamente adaptados, aumentando el área frente a la real de riesgo. Existe un cierto margen de seguridad, pero, en realidad, este no debería perderse. Sería aconsejable intentar caracterizar cuánto es la penetración máxima de la aeronave en estas regiones de riesgo debido a este motivo, para posteriormente incrementar el margen de seguridad frente al inicial, manteniendo, así, intacto dicho margen. Esta tarea queda fuera del objetivo esencial de este TFG y se plantea para futuras modificaciones y mejoras del algoritmo propuesto.




**(a)** Trayectoria de evasión empleada en la proyección de Mercator.



**(b)** Escape del obstáculo.

**Figura 3.28** Trayectoria de evasión empleada en la proyección de Mercator cuando el origen de la iteración está inmerso en una región tormentosa.

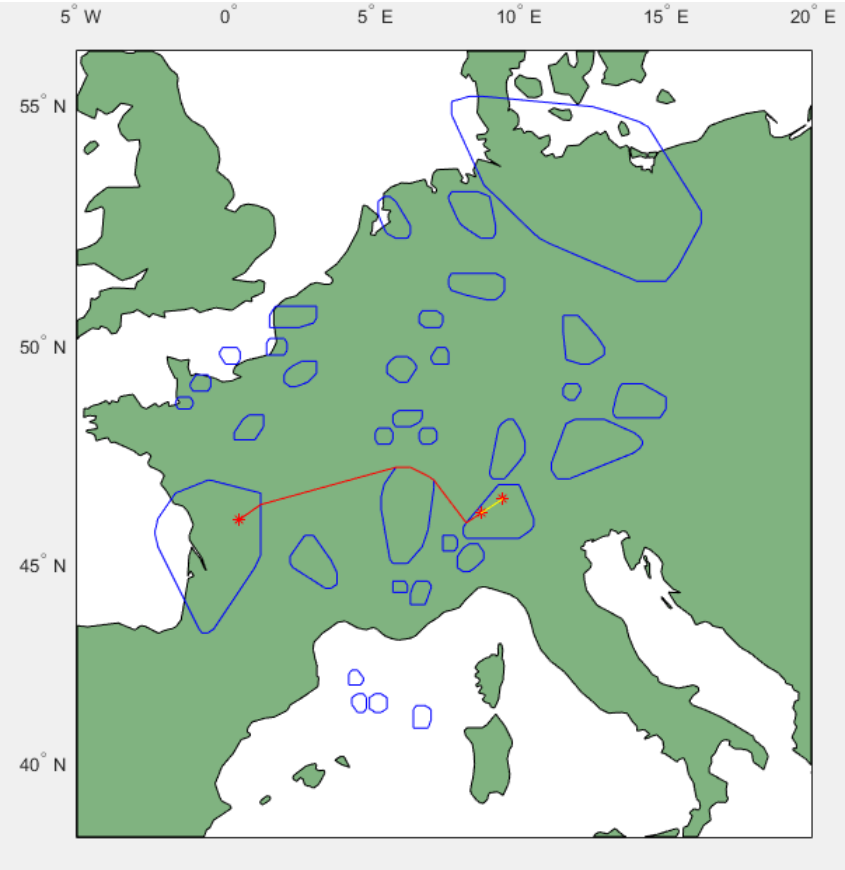



**Figura 3.29** Ejecución de la maniobra de evasión en la proyección de Mercator cuando el origen y el destino iniciales están inmersos en diferentes regiones tormentosas..

# 4 Resultados

## 4.1 Ejemplo 1. Barcelona-Munich

En esta sección se va analizar el vuelo real de un avión que parte del aeropuerto de Barcelona, con longitud de 2.18 y una latitud de 41.38, hasta el aeropuerto de Munich, con una longitud de 11.58 y una latitud de 48.14, cuyo crucero se realizará a presión constante, con un M=0.8 y con una matriz de vientos arbitraria, en este caso, es una matriz del mismo número de puntos que el mallado del espacio (200·160), pero con puntos no coincidentes. Se tomará un viento con solo componente meridional, de valor 10 m/s en cualquier punto (matriz de vientos constante), puesto que los vientos se interpolan, cualquier matriz sería válida.

Se supondrá que volar a presión constante, implica hacerlo a una altura constante que, aunque en la realidad no sea exactamente así, no es una aproximación grosera.

Con este ejemplo se pondrá de manifiesto la bondad de los resultados de este programa, así como la potencialidad de la herramienta que se muestra, la cual hará los cálculos de forma automatizada, trazando los segmentos seguros de evitación de tormentas y seleccionando de entre ellos, aquellos que conformen la trayectoria que minimiza el tiempo de vuelo emtre el origen y el destino.

Se comentarán, a continuación, paso a paso, los resultados que el algoritmo muestra por pantalla:

Como se ha ido explicando en el capítulo anterior, la primera tarea a realizar por el programa es, dada una malla de puntos de vuelo, calcular el tiempo que tardaría la aeronave en llegar a cada uno de los puntos, obtenido el mismo de la integración del inverso de la velocidad respecto a tierra respecto a la distancia entre el origen, en este caso Barcelona, y dichos puntos.

Tras esto, se procedía a calcular los obstáculos estáticos equivalentes, mediante la formación de la matriz *frametormenta*, de igual dimensión que las matrices del mallado, cuyos elementos eran 1, si los puntos correspondientes estaban dentro de alguna de las regiones detectadas por el radar y, además, para esos instantes de tiempo, la aeronave alcanza esos puntos. (Ver apartado 3.2.2)

Mediante la agrupación de los elementos con valor 1 de dicha matriz, se obtenían unos obstáculos estáticos equivalentes, no definitivos. Para la iteración primera del algoritmo del ejemplo estudiado, se obtienen los obstáculos de la imagen 4.1. En dicha imagen, se puede observar la correspondencia entre los obstáculos reales, función conjunta del espacio y el tiempo, y los obstáculos estáticos equivalentes.

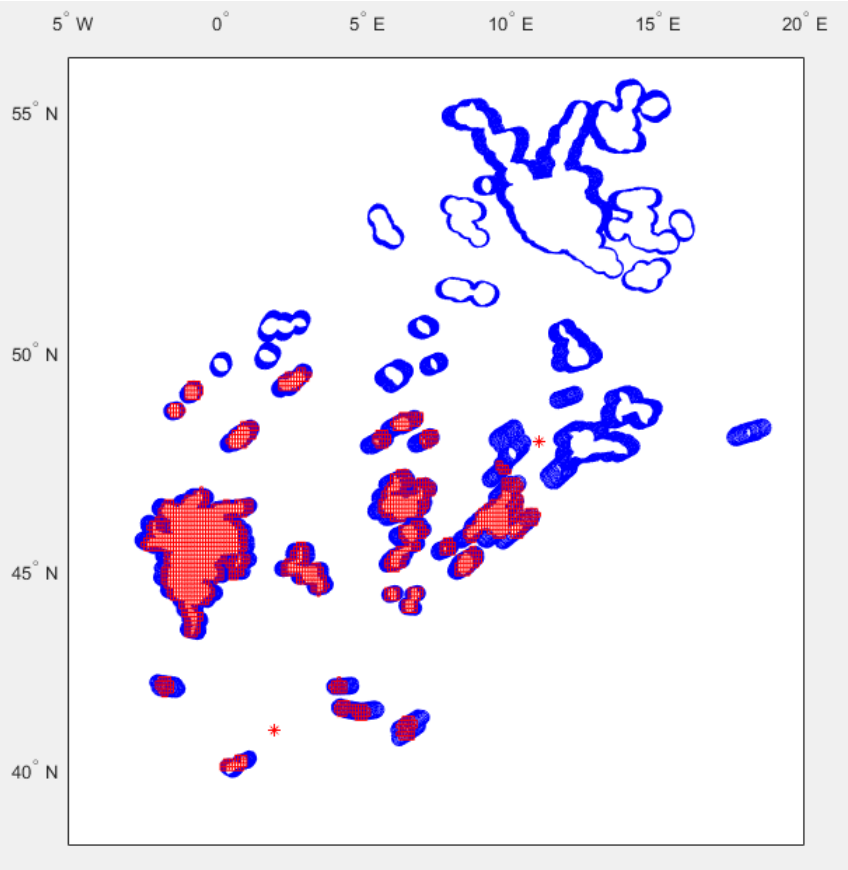



**Figura 4.1** Obstáculos estáticos equivalentes mallados para la primera iteración en la proyección de Mercator para el ejemplo Barcelona-Munich.

Los obstáculos estáticos equivalentes son los mallados en rojo en la imagen anterior. Como se puede observar, los centros de los diferentes cuadrados que conforman la malla (que se corresponden con los puntos que obtuvieron el valor 1 en la matriz *frametormenta*) están contenidos en los obstáculos reales, lo que da un aviso de que las tareas hasta el momento han sido implementadas correctamente, pues, que estos puntos estuviesen fuera del polígono real, implicaría que se estaría considerando como puntos a evitar, unos que no pertenecerían a las regiones de riesgo reales, puntos que no supondrían ningún peligro para la aeronave.

Se tomará, a continuación, la envolvente convexa de los mismos. En la imagen 4.2, se observa perfectamente el aumento de área del obstáculo por haber tomado su envolvente convexa y, por tanto, también del margen de seguridad que rodearía a las regiones de riesgo meteorológico reales. Esto permitirá reducir la cercanía a las tormentas reales, disminuyendo el riesgo de que, al moverse las regiones en la iteración siguiente, la aeronave se viese inmersa en una de estas tormentas reales, pero, por contra, el espacio aéreo y el número de conexiones posibles que la aeronave podría seguir, tal y como se comentó en el capítulo anterior. Serán las curvas azules que rodean al mallado en rojo de la imagen 4.2, las envolventes convexas de los obstáculos reales, las cuales darán forma a los obstáculos a los que se va a tener que enfrentar la aeronave durante su vuelo hasta Munich, previo a analizar si existen cortes entre ellas.

Una vez que estos polígonos están totalmente identificados y fusionados (Ver figura 4.3), se avanzará al cálculo de los diversos segmentos que la aeronave va a poder seguir para evitar la inmersión de la misma estas regiones de riesgo.

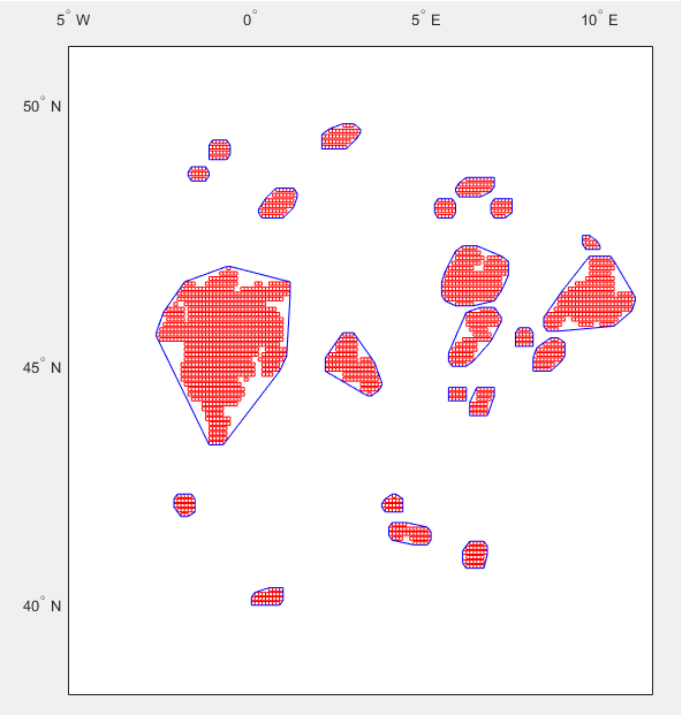



**Figura 4.2** Envolventes de las regiones a evitar para la primera iteración en la proyección de Mercator para el ejemplo Barcelona-Munich .

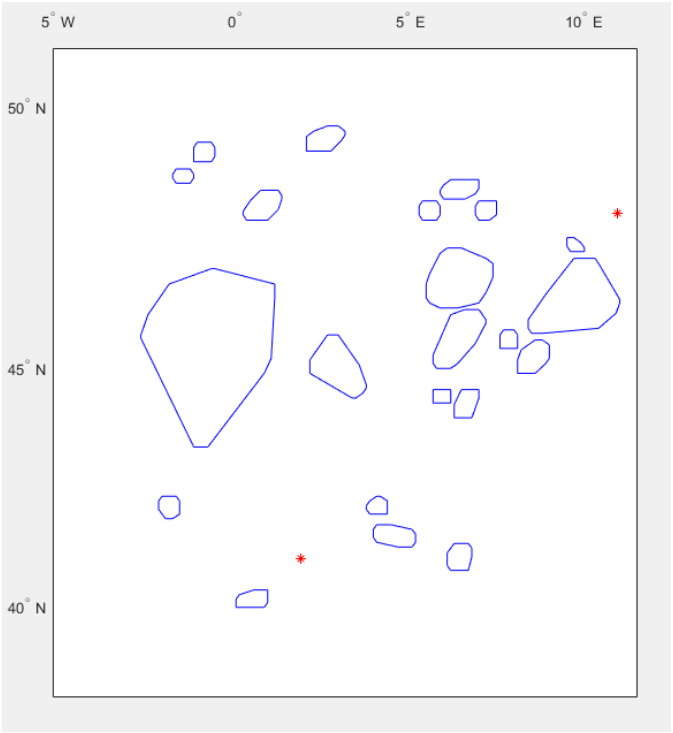
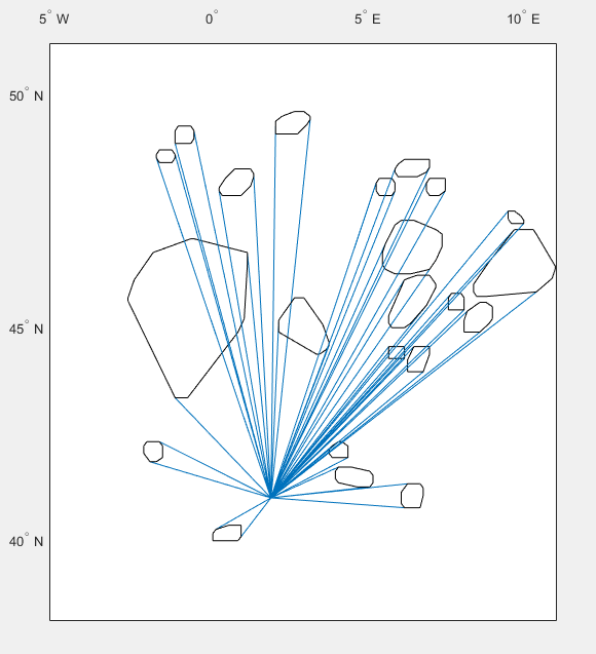


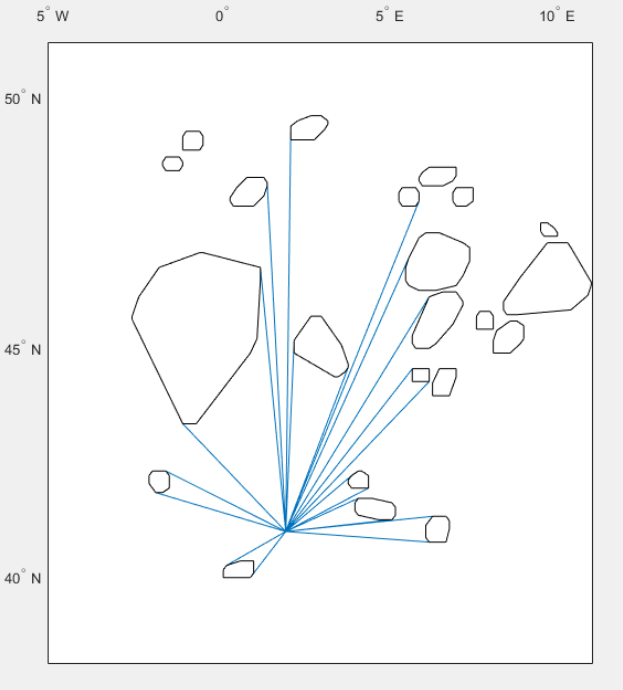

**Figura 4.3** Obstáculos estáticos definitivos a evitar en la proyección de Mercator para el ejemplo Barcelona-Munich.

Los nodos origen y destino (Barcelona y Munich), representados con un asterisco en la imagen 4.3, se van a unir al resto de polígonos mediante las conexiones de las imágenes 4.4 y 4.5, previamente

eliminadas aquellas que conllevaban la inmersión en otra región tormentosa ajena a la de unión. La posibilidad de tangentes no visibles es la culpable de que, aunque existan dos tangentes entre un punto y un polígono, las dos no tengan por que ser válidas, eliminandose aquéllas que intersecciones a otro obstáculo.

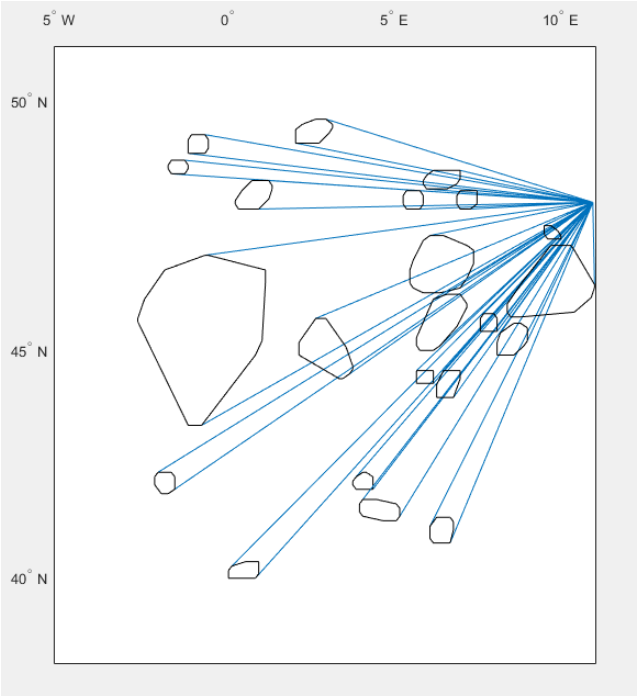



**(a)** Todas las tangentes origen-polígonos .

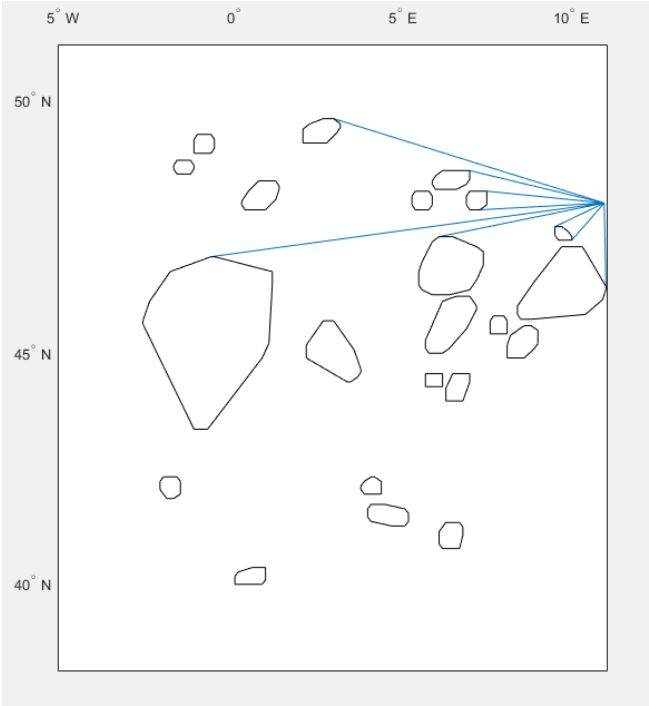



**(b)** Tangentes visibles origen-polígonos.

**Figura 4.4** Tangentes origen-polígonos para la primera iteración en la proyección de Mercator para el ejemplo Barcelona-Munich.




**(a)** Todas las tangentes entre los polígonos y el destino.

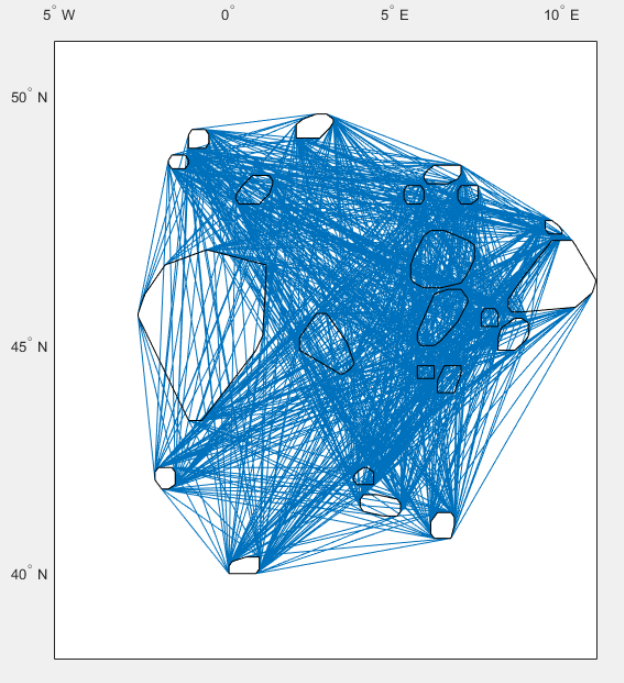



**(b)** Todas las tangentes visibles entre los polígonos y el destino.

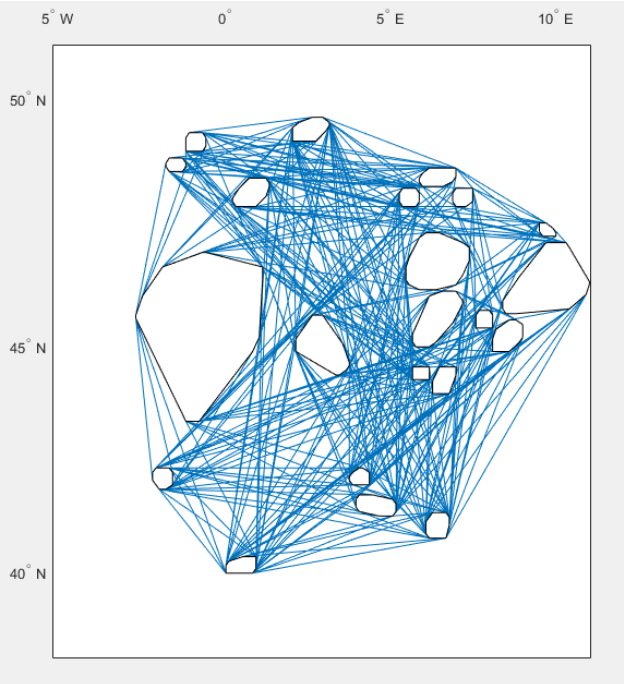
**Figura 4.5** Tangentes destino-polígonos para la primera iteración en la proyección de Mercator para el ejemplo Barcelona-Munich.

Las imágenes anteriores muestran que el cálculo y la eliminación de las tangentes no visibles se ha llevado a cabo correctamente. Rectas que no son tangentes a los polígonos o cortes de las mismas con alguno de los otros polígonos, avisarían de un error en el programa.

Una vez que la aeronave ha llegado a uno de los polígonos de riesgo, tiene varias opciones: podría seguir una tangente entre ese punto y el destino, en caso de su existencia, avanzar a lo largo de las aristas del polígono o tirar por una tangente entre ese polígono y otro, que le permita acercarse hasta el nodo de destino. Estas últimas van a estar representadas en las imágenes 4.6 a y b posteriores.




**(a)** Todas las tangentes entre polígonos.




**(b)** Tangentes visibles entre polígonos.

**Figura 4.6** Tangentes entre polígonos para la primera iteración en la proyección de Mercator para el ejemplo Barcelona-Munich.

Análogamente a las imágenes anteriores, las imágenes 4.6 a y b permiten detectar si existiesen fallos en el programa, ligados a rectas no tangentes a ambos polígonos, cortes de las tangentes a los polígonos... asegurando, en caso de su inexistencia, la bondad de los resultados. En la imagen 4.6 b, se puede ver como las tangentes conectan los diferentes nodos sin atravesar en ningún momento los obstáculos, contorneados con una línea negra, asegurando que seguir estas conexiones no va a implicar adentrarse en una de estas regiones tormentosas.

A continuación, se muestra el conjunto de todos los segmentos que la aeronave podrá seguir con el objetivo de evitar las regiones de riesgo tormentoso en su camino entre Barcelona y Munich, para los primeros 5 minutos de vuelo, es decir, se muestra el grafo de visibilidad para la primera iteración del algoritmo de horizonte deslizante.

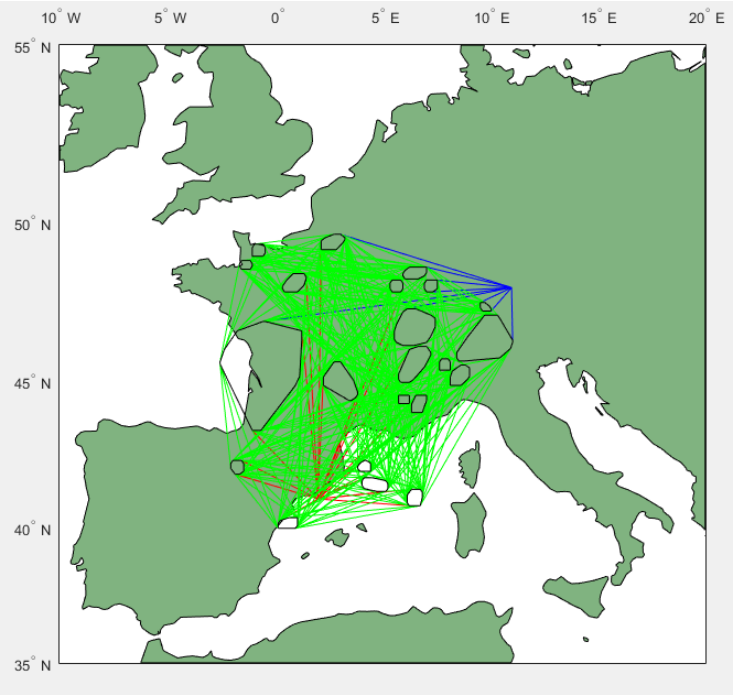



**Figura 4.7** Conjunto de posibles caminos a seguir para la primera iteración en la proyección de Mercator para el ejemplo Barcelona-Munich.

En la imagen 4.7, se muestran en rojo las diferentes tangentes visibles entre el origen y los obstáculos estáticos, en azul; las tangentes visibles entre el destino y las regiones de riesgo, y en verde; las diferentes tangentes visibles entre los polígonos tormentosos. Para la primera iteración únicamente están presentes las conexiones descritas. El resto de tipos, explicados en el apartado 3.3.1, se podrá observar en iteraciones futuras.

Una vez formado el grafo de visibilidad, representado gráficamente en la imagen 4.7, y obtenidas las matrices que almacenaban las conexiones, costes y otras magnitudes de interés, se tienen todas las variables listas para el cálculo del camino de mínimo coste temporal.

Para la información meteorológica actual, se obtiene el trayectoria de mínimo tiempo de la imagen 4.8. En ella, se puede observar, en rojo, los nodos origen y destino actuales, así como el origen para la siguiente iteración, que se recuerda que es el punto que se alcanza tras 5 minutos de vuelo. En este mismo color se representa el tramo de la trayectoria segura de mínimo tiempo que queda

por realizar tras esos primeros 5 minutos, tramo que va desde el origen de la siguiente iteración al destino real. En azul, se representan, por un lado los obstáculos estáticos a evitar y, por otro lado, el tramo recorrido en esos 5 minutos previos a la siguiente iteración. Posteriormente, este tramo se representará en amarillo, para mejorar la visibilidad y evitar confusión.

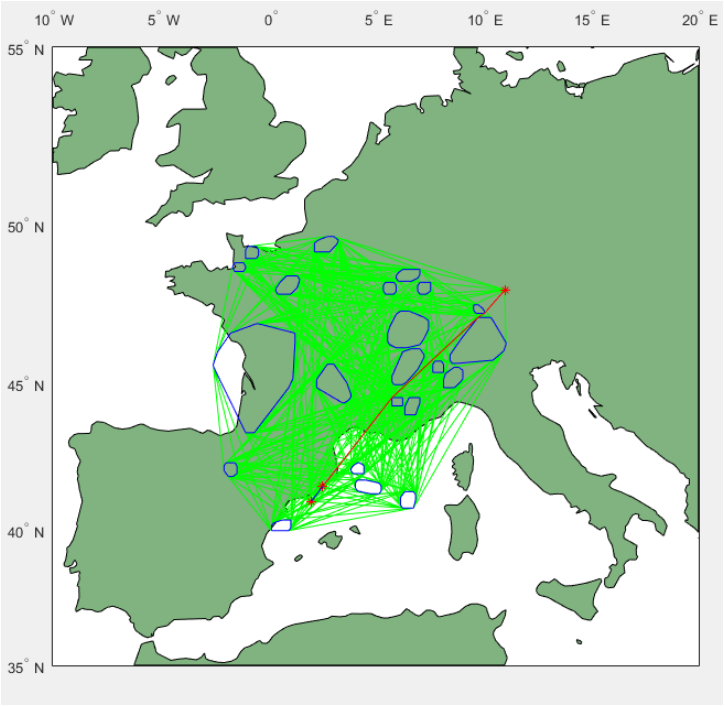
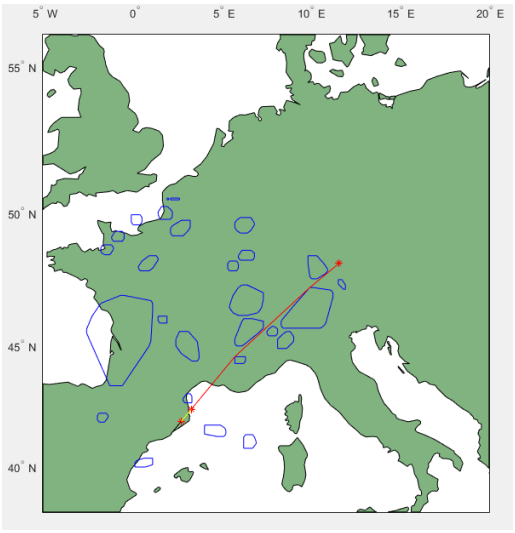



**Figura 4.8** Trayectoria de mínimo coste a seguir para la primera iteración en la proyección de Mercator para el ejemplo Barcelona-Munich.

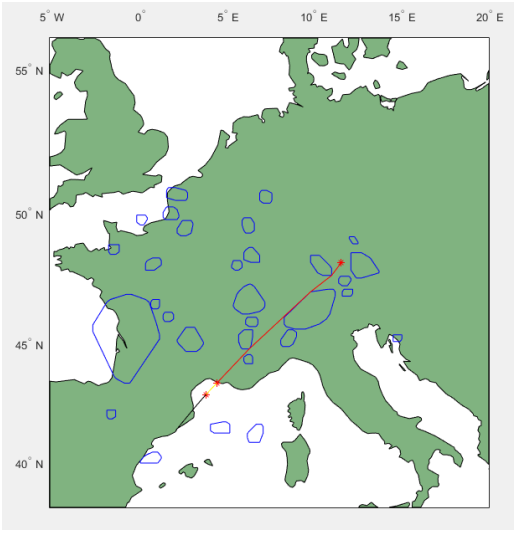
El asterisco siguiente al utilizado para marcar el punto de origen, se corresponde, como ya se ha comentado, con el punto que sobrevolará a los 5 minutos. Es, al pasar dicho tiempo y alcanzar ese punto, cuando una nueva información meteorológica es recibida, la cual será el input para volver a aplicar el algoritmo, calculando un nuevo camino óptimo con los nuevos obstáculos captados por el radar, tomando como origen ese punto marcado.

La maraña de líneas verdes de la imagen anterior son todos los segmentos que la aeronave podría tomar entre los diferentes nodos considerados. De entre todos ellos, solo se elegirán aquellos que permitan minimizar el coste temporal entre el nodo origen y el destino, resultando el camino inicial que se observa en la imagen 4.8. Pero este camino no será el definitivo, pues al recibir una nueva vista del panorama meteorológico, se actualizará la trayectoria de acuerdo al mismo.

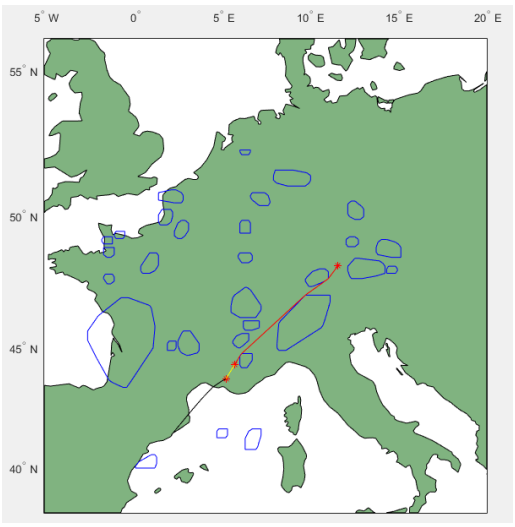
Se muestran, en las imágenes 4.9 y 4.10, las diferentes iteraciones del algoritmo, en las que se trazará la trayectoria, para cada una de ellas, de mínimo tiempo entre el nuevo origen de la misma y el destino. En cada una de ellas, se recorrerá un pequeño tramo seguro, que finalizará con la iteración siguiente al recibir nuevos datos meteorológicos del entorno de vuelo. Como se ha comentado, el punto marcado está distanciado del origen en 5 minutos. La curva marcada en negro, no es más que el tramo de vuelo que lleva ya recorrido de las iteraciones anteriores y el amarillo; el que seguirá en la iteración actual, de duración 5 minutos.



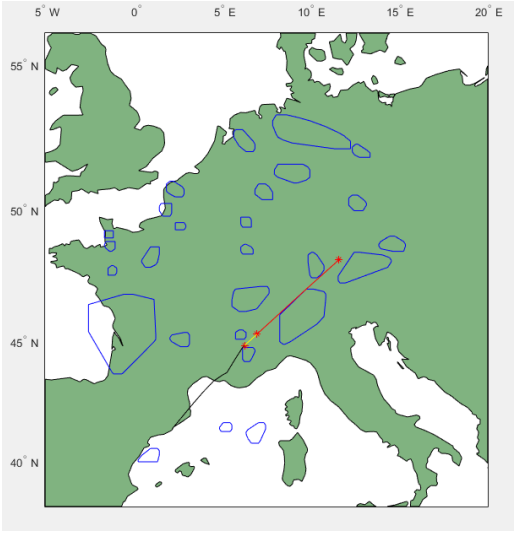

**(a)** Trayectoria a seguir en la proyección de Mercator en la iteración 2.




**(b)** Trayectoria a seguir en la proyección de Mercator en la iteración 4.

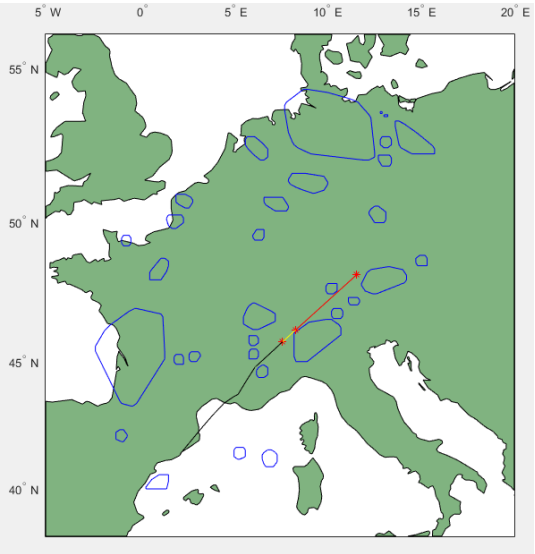



**(c)** Trayectoria a seguir en la proyección de Mercator en la iteración 6.

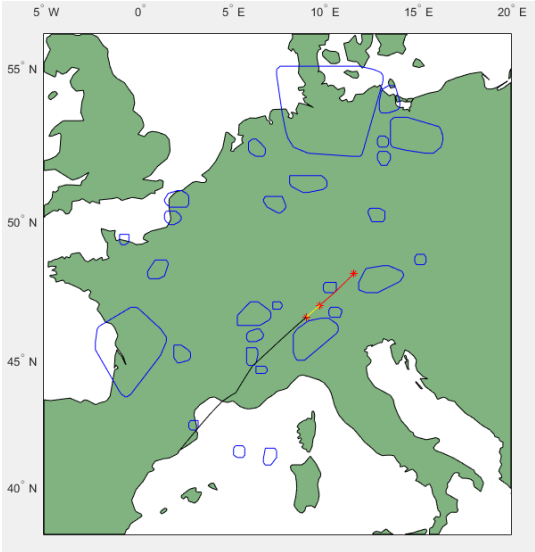



**(d)** Trayectoria a seguir en la proyección de Mercator en la iteración 8.

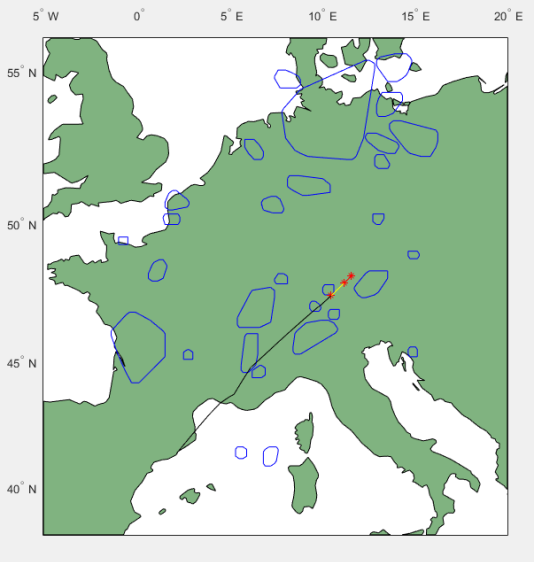
**Figura 4.9** Trayectoria de evitación de tormentas de mínimo coste temporal para las iteraciones 2, 4, 6 y 8 del algoritmo para el ejemplo Barcelona-Munich.



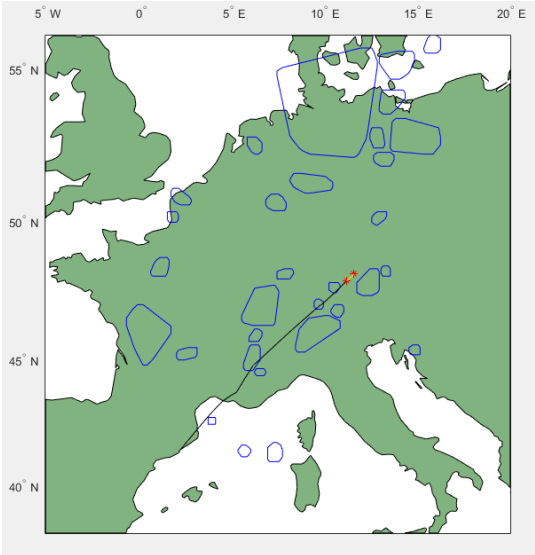

**(a)** Trayectoria a seguir en la proyección de Mercator en la iteración 10.




**(b)** Trayectoria a seguir en la proyección de Mercator en la iteración 12.




**(c)** Trayectoria a seguir en la proyección de Mercator en la iteración 14.




**(d)** Trayectoria a seguir en la proyección de Mercator en la iteración 15.

**Figura 4.10** Trayectoria de evitación de tormentas de mínimo coste temporal para las iteraciones 10, 12, 14 y 15 del algoritmo para el ejemplo Barcelona-Munich.

La curva negra, como se puede observar, por ejemplo, en la imagen 4.10 (d), puede cortar a los polígonos que aparezcan en el mapa. Esto, aunque de la sensación de que implica la existencia de algún error, no es así. La curva negra indica los puntos por los que la aeronave ya ha pasado, análogo a la estela que deja el avión al volar, curva que desembocará en el punto actual, indicado por el primer asterisco rojo. Por ello, una intersección entre la curva negra y el resto de obstáculos no supone un problema. Sería peligroso si, en cambio, la curva roja o la amarilla cortasen a una de estas regiones, ya que indican el trozo de trayectoria que se propone que recorra la aeronave a partir de su posición actual.

Para el ejemplo mostrado, el algoritmo ejecuta 15 iteraciones. Para la última de estas iteraciones

(ver imagen 4.10 (d)), el origen está situado a menos de 5 minutos del destino, lo que implicará que el origen de la que sería la siguiente iteración coincidirá con el propio destino, poniendo fin a la aplicación del algoritmo de horizonte deslizante.

En la imagen 4.11 se puede observar la representación de algunos nowcasts para diferentes instantes de tiempo, así como diferentes tramos de la trayectoria segura de mínimo tiempo, con diferentes colores. Por un lado, los diferentes colores de las regiones hacen referencia al nowcast del que han sido extraídas, representadas estas cada diez minutos, por simplicidad. Por otro lado, los colores de los diferentes tramos de la trayectoria segura de mínimo tiempo relacionan estos con la información meteorológica que ha sido utilizada para su obtención. Como se puede ver, los tramos de diferentes colores, evitan perfectamente las regiones del mismo color que el tramo, cumpliendo, así, el objetivo de este trabajo. Los tramos en negro son aquellos pertenecientes a la trayectoria de mínimo tiempo para los que, por simplicidad, su información meteorológica origen no está representada.

En la imagen 4.12, se puede observar la trayectoria que debe seguir la aeronave entre Barcelona y Munich para surcar el cielo de forma segura y con el menor tiempo posible.

Otro aspecto muy importante a considerar es el tiempo de ejecución de cada una de las iteraciones. Estas deben tardar mucho menos de 5 minutos, pues es el tiempo que se tarda en recibir una nueva información meteorológica. Si el tiempo aumenta hasta ese límite, habría un choque de información, pudiendo obtener resultados ilógicos que reduzcan la eficiencia del algoritmo, o en el mejor de los casos, un retraso en la obtención de los resultados, lo que supondría evitar polígonos que han podido sufrir grandes cambios a lo largo de ese periodo de tiempo y disten bastante de los que se están evitando en esa iteración, además de no encontrarse el avión donde se predijo que iba a estar.

Actualmente, dicho tiempo de ejecución, para una zona mallada de 200·160 puntos, que distan una décima de grado entre ellos, se encuentra en torno a 2 minutos, tiempo durante el cual el avión seguirá volando sin conocer la trayectoria segura de menor tiempo. Por tanto, el tiempo va a suponer un problema, existiendo un cierto retraso en la obtención de la información sobre la trayectoria a seguir. En vista a estos resultados temporales, será necesario modificar el algoritmo para que tenga en cuenta que la información va a ser recibida con un cierto retraso.

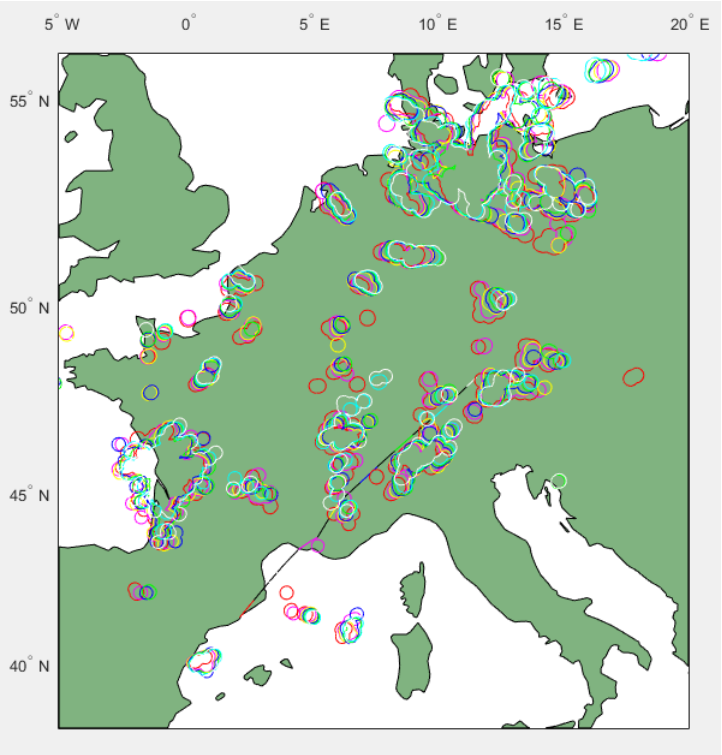



**Figura 4.11** Representación de los nowcasts y las trayectorias de mínimo tiempo obtenidas en la proyección de Mercator para el ejemplo Barcelona-Munich.

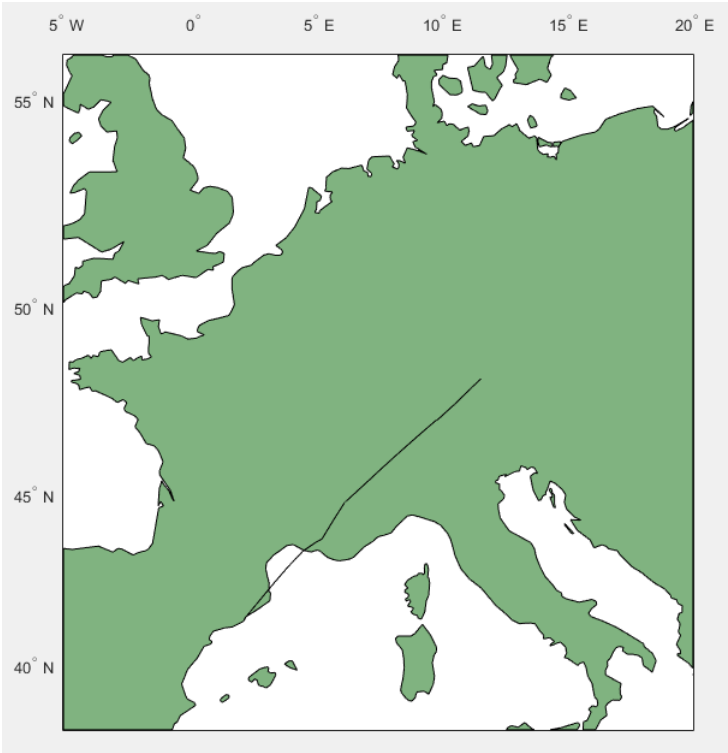
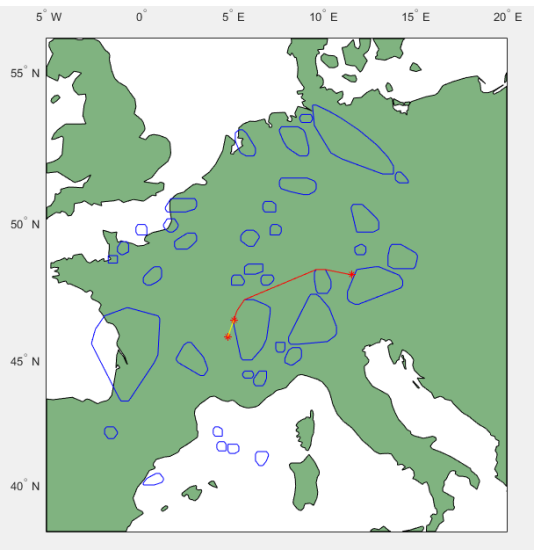



**Figura 4.12** Trayectoria seguida en la proyección de Mercator para el ejemplo Barcelona-Munich.

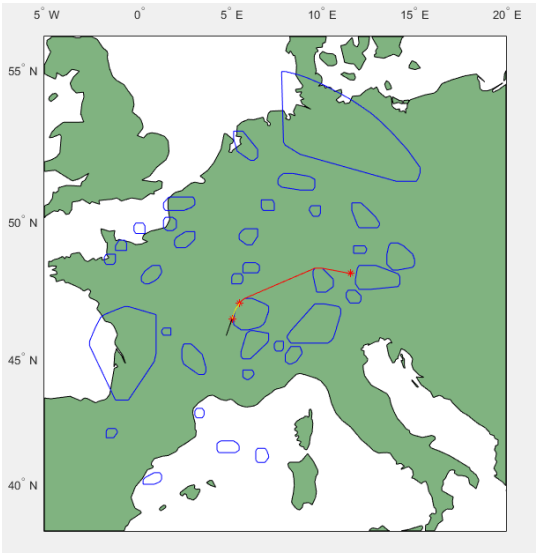
## 4.2 Ejemplo 2. Lyon-Munich

Con el objetivo de dar una idea de transparencia y de demostrar que la herramienta es válida sea cual sea el par origen-destino, se presenta a continuación otro ejemplo, consistente en un vuelo entre la ciudad francesa de Lyon con longitud, 4.85, y latitud, 45.76, y Munich, cuyas coordenadas se mostraron en el ejemplo anterior.

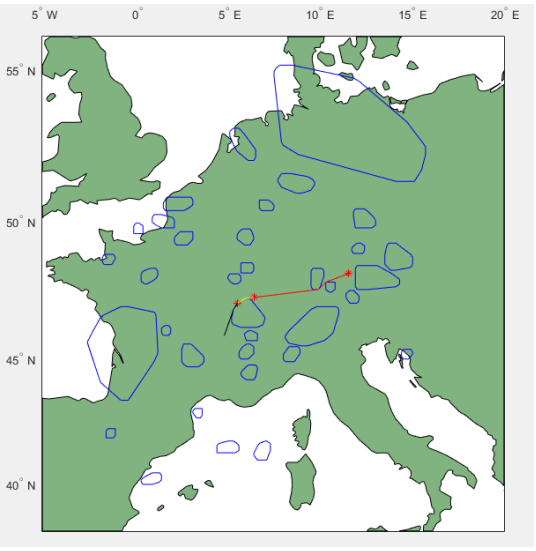
Los comentarios utilizados para la explicación del ejemplo anterior son igualmente válidos para el que se presenta en este apartado. Por ello, por simplicidad y para evitar repeticiones, únicamente se mostrarán las imágenes obtenidas de las diferentes iteraciones, suficientes para dar una idea de la bondad de los resultados.



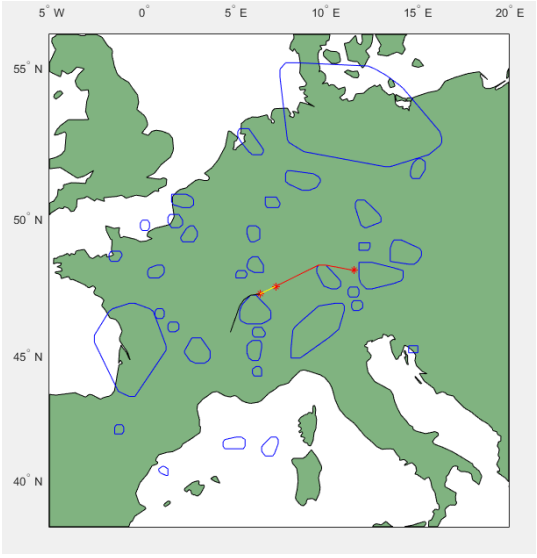

**(a)** Trayectoria a seguir en la proyección de Mercator.




**(b)** Trayectoria a seguir en la proyección de Mercator.

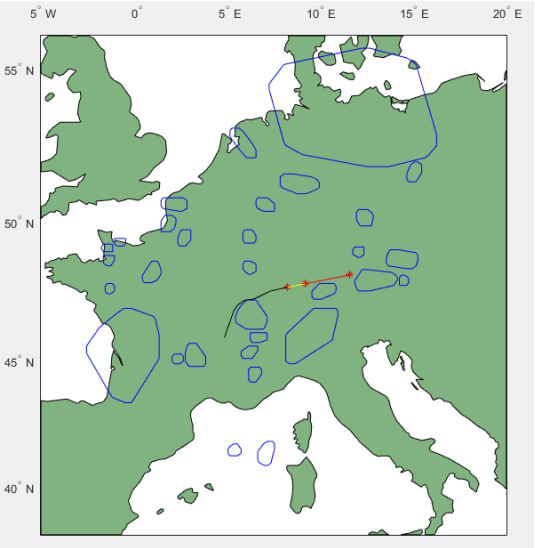



**(c)** Trayectoria a seguir en la proyección de Mercator.

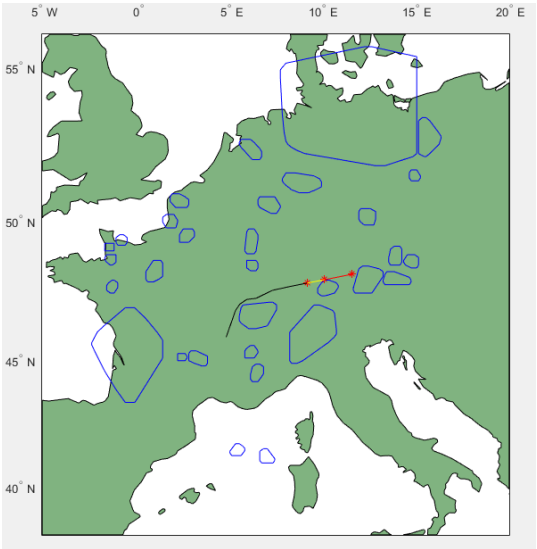



**(d)** Trayectoria a seguir en la proyección de Mercator.

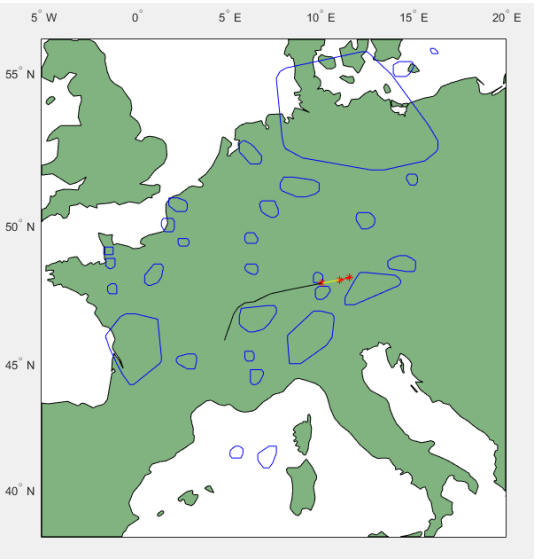
**Figura 4.13** Trayectoria de evitación de tormentas de mínimo coste temporal para las iteraciones 1, 2, 3 y 4 del algoritmo para el ejemplo Lyon-Munich.



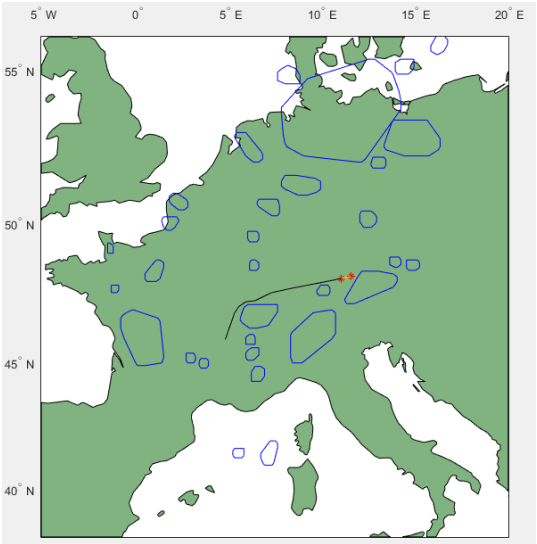

**(a)** Trayectoria a seguir en la proyección de Mercator.




**(b)** Trayectoria a seguir en la proyección de Mercator.




**(c)** Trayectoria a seguir en la proyección de Mercator.




**(d)** Trayectoria a seguir en la proyección de Mercator.

**Figura 4.14** Trayectoria de evitación de tormentas de mínimo coste temporal para las iteraciones 6, 7, 8 y 9 del algoritmo para el ejemplo Lyon-Munich.

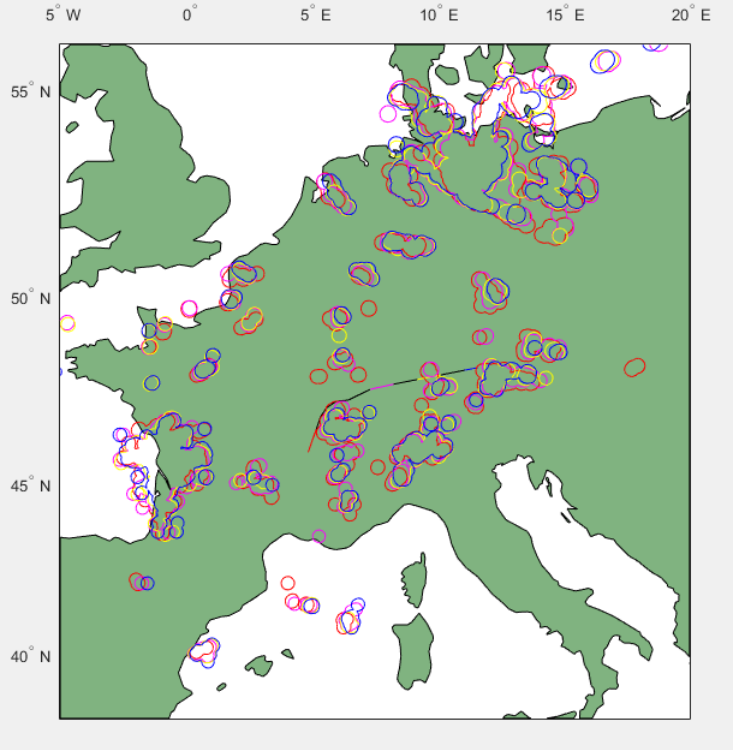



**Figura 4.15** Representación de los nowcasts y las trayectorias de mínimo tiempo obtenidas en la proyección de Mercator para el ejemplo Lyon-Munich.

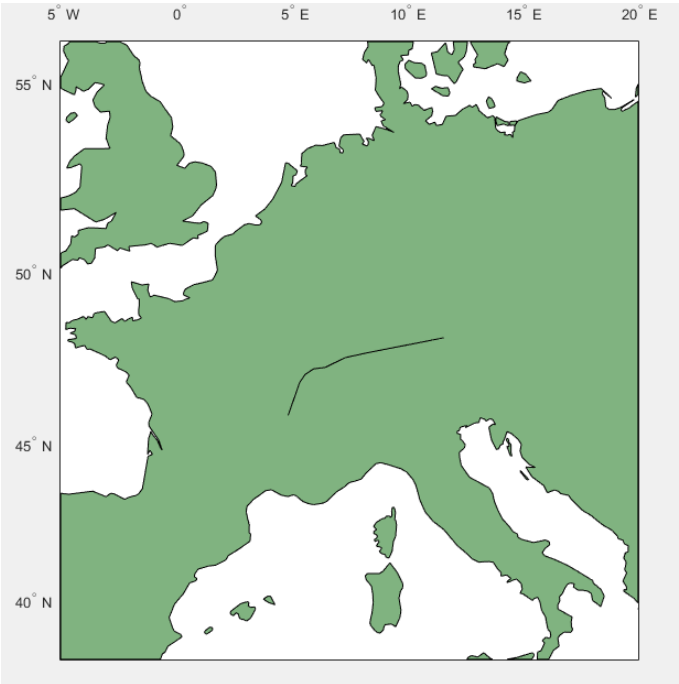



**Figura 4.16** Trayectoria seguida en la proyección de Mercator para el ejemplo Lyon-Munich.

# 5 Conclusiones

Como se ha podido ver durante los capítulos anteriores, la herramienta va a permitir a la aeronave y a sus pasajeros gozar de un viaje seguro y rápido, mediante la obtención de la trayectoria de mínimo coste temporal que evite la inmersión de la aeronave en regiones de riesgo meteorológico, zonas que pueden poner en peligro a la aeronave (efectos desestabilidades y de reducción del control, posibles riesgos a la integridad estructural de la aeronave...) , y aumentar la carga y el estrés de los pilotos, pues son situaciones en las que los pilotos pueden controlan manualmente la aeronave.

Este algoritmo, implementado en los computadores de control de vuelo de la aeronave, se ejecutará automáticamente, una vez que la información proporcionada por el radar meteorológico es recibida. Esto pone de manifiesto sobre todo un aspecto muy importante:

Es una herramienta automática, sin necesidad de control directo del piloto. Es esta característica la que permitirá a la herramienta servir de apoyo en la toma de decisiones del piloto, además de facilitar tareas al mismo y reducir de forma considerable los posibles errores humanos que este pueda cometer durante el vuelo, causados, por ejemplo, por el estrés debido a los efectos de la inmersión en una de las regiones tormentosas, que son las que se quieren evitar con esta herramienta.

Como se puede observar, es una herramienta muy potente, capaz de, dado una simple información referente a los obstáculos meteorológicos que el radar detecta durante el vuelo, trazar todos los segmentos que la aeronave puede seguir y que aseguran, para el nowcast actual y las predicciones recibidas -al recibir los siguientes nowcasts, obtenidos cada 5 minutos, no hay seguridad de que la aeronave no se haya visto inmersa en una de estas regiones por la interpolación temporal utilizada-, que en ningún momento se atravesará ninguna de estas regiones -salvo que el origen o el destino estén ya encerrados por alguna de estos obstáculos tormentosos-. De entre todos estos posibles segmentos a elegir, se seleccionará aquellos que permitan conseguir el mínimo tiempo de vuelo, reduciendo, así, el combustible de la aeronave y el DOC, variable que permite cuantificar el coste de vuelo, sin, como se ha dicho antes, adentrarse en ninguna de estas regiones tormentosas -es importante remarcar esto, pues las variables anteriores pueden aumentar con respecto a las que se obtendrían al volar directamente del origen al destino, maniobra de gran riesgo para la aeronave y los pasajeros-.

Para paliar el hecho de que está trabajando, además de con el nowcast actual, con predicciones que pueden distar de la realidad, se introducirá nueva información sobre el panorama meteorológico actual cada 5 minutos, actualizando la trayectoria segura a seguir y, dando, así, realidad a la herramienta.

El aumento de seguridad (safety del avión) que el seguimiento de esta trayectoria otorga a los vuelos, sumada a los aspectos comentados en los párrafos anteriores, ponen sobre la mesa la

necesidad de la implementación de esta herramienta, herramienta que va a suponer innumerables ventajas y mejoras sobre la aviación actual.

# Bibliografía


[1] W. L. Golding, *Turbulence and Its Impact on Commercial Aviation*. Journal of Aviation/Aerospace Education & Research, 2000.

[2] T. N. S. S. Laboratory. Types of damaging winds. [Online]. Available: https://www.nssl.noaa.gov/education/svrwx101/wind/types/

[3] Skybrary. (2019, March) Aircraft and in flight icing risks. [Online]. Available: https://www.aircraftsystemstech.com/2017/05/weather-radar.html

[4] B. González, *Meteorología Aeronáutica*. Biblioteca Homo Legends, 2013.

[5] J. M. Picón and M. Poole, *Integración de Sistemas y Pruebas funcionales*. ETSI, Universidad de Sevilla, 2015.

[6] A. Guide. (2017, May) Aircraft weather radar. [Online]. Available: https://www.aircraftsystemstech.com/2017/05/weather-radar.html

[7] R. V. Valenzuela and F. G. Jiménez, *Rutas*. ETSI, Universidad de Sevilla.

[8] Wikipedia.org. (2019) Bilinear interpolation. [Online]. Available: https://en.wikipedia.org/wiki/Bilinear_interpolation

[9] ——. (2020) Mercator projection. [Online]. Available: https://en.wikipedia.org/wiki/Mercator_projection

[10] EcuRed. Envoltura convexa. [Online]. Available: https://www.ecured.cu/Envoltura_convexa

[11] D. Sunday. (2012) Tangens to & between 2d polygons. [Online]. Available: http://geomalgorithms.com/a15-_tangents.html

[12] P. Jáuregui and I. Escalante. (2013) Modelos de redes en producción. [Online]. Available: https://www.gestiopolis.com/modelos-de-redes-en-produccion/

[13] E.W.Dijkstra, *A note on two problems in connexion with graphs*. Numerische Mathematik, 1959.

# Apéndice A
# Herramienta de planificación de trayectorias de evitación de regiones de riesgo meteorológico

## A.1 Datos introductorios y mallado inicial

```
%Datos iniciales
Pinf=200e2;
Minf=0.8;
gamma=1.4;
Rg=286.9;
vrefISA=sqrt(Minf^2*gamma*Rg*217.434);
Theta=1;
Rt=6378.14e3;
bucle=0;
puntoordentro=0;
puntodesdentro=0;
puntospaso=1;
nodospasox(puntospaso)=0;
zorconf=0;
 %1. Realizamos la discretización

M=(17.5+2.5)/0.1
N=(56-40)/0.1
for i=1:N+1
    for j=1:M+1
        phi(i,j)=(40+(56-40)*(N-i+1)/N)*pi/180;
        lambda(i,j)=(-2.5+(17.5+2.5)*(j-1)/M)*pi/180;
        phiviento(i,j)=(38+(58-38)*(N-i+1)/N)*pi/180;
        lambdaviento(i,j)=(-4.5+(19.5+4.5)*(j-1)/M)*pi/180;
        Wu(i,j)=0;
        Wv(i,j)=10;
    end
end
```

```
pasomallalambda=(lambdaviento(1,2)-lambdaviento(1,1))*180/pi
pasomallaphi=(phiviento(1,1)-phiviento(2,1))*180/pi

 lambda0=2*pi/180 %1*pi/180%lambda(3*N/5,3*N/5) %(1,1) %origen ejemplo
phi0=41*pi/180% phi(1,1) %phi(1,1) %origen ejemplo
lambdaD=11*pi/180 %Destino ejemplo
phiD=48*pi/180 %Destino ejemplo
P0=[lambda0,phi0];
PD=[lambdaD,phiD];
storms_TFG = load('storms_TFG.mat')
storms_TFG = struct2cell(storms_TFG)
```

## A.2 Implementación de las ecuaciones loxodrómicas

```
function [r,course]=loxodromicequations(phi0,lambda0,phi,lambda,Rt,Theta
    ,vrefISA,N,M)
for i=1:N+1
    for j=1:M+1
        if abs(lambda(i,j)-lambda0)>pi
            if lambda(i,j)<0
                course(i,j)=atan((lambda(i,j)+2*pi-lambda0)/log(tan(pi/4-
                    phi0/2)/tan(pi/4-phi(i,j)/2)));
            elseif lambda(i,j)>0
                course(i,j)=atan((lambda(i,j)-2*pi-lambda0)/log(tan(pi/4-
                    phi0/2)/tan(pi/4-phi(i,j)/2)));
%           else
%               course(i,j)=atan((lambda(i,j)-lambda0)/log(tan(pi/4-phi
    0/2)/tan(pi/4-phi(i,j)/2)));
            end
        elseif abs(lambda(i,j)-lambda0)<pi
            course(i,j)=atan((lambda(i,j)-lambda0)/log(tan(pi/4-phi0/2)/
                tan(pi/4-phi(i,j)/2)));
        else
            course(i,j)=0;
        end
        if phi(i,j)<phi0
            course(i,j)=course(i,j)+pi;
        end
if lambda0>lambda(i,j)&&phi0>phi(i,j)
    course(i,j)=course(i,j)-2*pi;
    end

%Calculo del curso, constante para cada par de puntos

if (phi(i,j)-phi0)==0
    %r(i,j)=acos(cos(pi/2-phi0)*cos(pi/2-phi0) + sin(pi/2-phi0)*sin(pi
        /2-phi0)*cos(abs(lambda(i,j)-lambda0)))*Rt*sqrt((cos(lambda0)*cos
        (phi0))^2+(sin(lambda0)*cos(phi0))^2)
```

```
    r(i,j)=abs(lambda(i,j)-lambda0)*cos(phi0)*Rt;
elseif lambda(i,j)==lambda0
    r(i,j)=Rt*abs(phi(i,j)-phi0);
elseif (phi(i,j)-phi0)==0&&(lambda(i,j)-lambda0)==0
    course(i,j)=0;
    r(i,j)=0;
else
alphalox(i,j)=(phi(i,j)-phi0)/cos(course(i,j));
r(i,j)=alphalox(i,j)*Rt;
end
end
end
%Problema inverso, a partir de r calculamos los phi correspondientes
end
```

## A.3 Interpolación de los vientos

```
function [viento]=funcioninterpolavientos(lambdavientos,phivientos,
    lambda,phi,W,pasomallalambda,pasomallaphi,M,N)
%Calculamos los indices del punto superior izquierdo.
i=floor((lambda-lambdavientos(1,1))*180/pi/pasomallalambda)+1;
    j=floor((phivientos(1,1)-phi)*180/pi/pasomallaphi)+1;
    [lambda*180/pi;lambdavientos(1,1)*180/pi];
    [j,i];
if i==M+1
    i=i-1;
end
if j==N+1
    j=j-1;
end
    viento=(W(j,i)*(lambdavientos(j+1,i+1)-lambda)*(phi-phivientos(j+1,i
        +1))...
          +W(j,i+1)*(lambda-lambdavientos(j+1,i))*(phi-phivientos(j+1,i))
              ...
          +W(j+1,i)*(lambdavientos(j,i+1)-lambda)*(phivientos(j,i+1)-phi)
              ...
          +W(j+1,i+1)*(lambda-lambdavientos(j,i))*(phivientos(j,i)-phi))
              ...
          /((lambdavientos(j+1,i+1)-lambdavientos(j,i))*(phivientos(j,i)-
              phivientos(j+1,i+1)));
end
```

## A.4 Función a integrar para obtener la matriz de tiempos

```
    function [dtdr]=loxodromicequationsint(phi0,lambda0,Rt,Wu,Wv,Theta,
        vrefISA,curso,y,t)
```

```
%Problema inverso, a partir de r calculamos los phi correspondientes
       phi2=phi0+y/Rt*cos(curso);
       lambda2=lambda0+tan(curso)*log(tan(pi/4-phi0/2)/tan(pi/4-phi2/2))
           ;
%Calculo de las valocidades:
    V=vrefISA*sqrt(Theta);
    Wat=Wu*cos(pi/2-curso)+Wv*sin(pi/2-curso);
    Wxt=-Wu*sin(pi/2-curso)+Wv*cos(pi/2-curso);
        %Velocidad respecto a tierra en función de r
    Vg=sqrt(V^2-Wxt^2)+Wat;
    dtdr=1./Vg;
    end
```

## A.5 Determinación de los puntos en los que la aeronave se verá inmersa en una región tormentosa

```
tiemposinterpol=[0:5:65];
poltormenta=zeros(N+1,M+1);
frametormenta=zeros(N+1,M+1);
xv=zeros(1);
yv=zeros(1);

for k=65/5:-1:1
%     for z=1:n
%         for t=1:length(randi(z,:))
% xv(z,t) =(lambda(randi(z,t),randj(z,t))+Wu(randi(z,t),randj(z,t))/Rt
    *5*60*k)'; %Hay que arreglar esto
% yv(z,t) =(phi(randi(z,t),randj(z,t))+Wv(randi(z,t),randj(z,t))/Rt
    *5*60*k)';
%
%         end
storm_TFG(k);
[n(k),A]=size(storm_TFG{k});
n(k);
for z=1:n(k)

lengthstorm(z)=length(storm_TFG{k}(z).Lat);
[z,lengthstorm(z)];
xv(z,1:lengthstorm(z))=storm_TFG{k}(z).Lon'*pi/180;
yv(z,1:lengthstorm(z))=storm_TFG{k}(z).Lat'*pi/180;
% if k==1
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'b')

% hold on
% elseif k==3
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')

% hold on
```

```
% elseif k==5
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')

% hold on
% elseif k==7
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')

% hold on
% elseif k==9
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')

% hold on
% elseif k==11
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')

% hold on
% elseif k==13
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')

% hold on
% end
for i=1:N+1
    xq=lambda(i,:);
    yq=phi(i,:);
    [in,on] = inpolygon(xq,yq,xv(z,1:lengthstorm(z)),yv(z,1:lengthstorm(
        z)));
    inmat(i,:)=in;
%    plot(xv,yv) % polygon
% % axis equal
% % hold on
% % plot(xq(in),yq(in),'r+') % points inside
% % plot(xq(~in),yq(~in),'bo') % points outside
% % hold on

for j=1:M+1
     if inmat(i,j)==1&&tmat(i,j)>=tiemposinterpol(k)&&tmat(i,j)<
         tiemposinterpol(k+1)
           tormenta(i,j)=1; %Hay una tormenta en ese punto cuando el avi
               ón llegue
       else
           tormenta(i,j)=0; %Dicha tormenta no está
       end
end
end
size(tormenta);
poltormenta=poltormenta+tormenta; %total de puntos afectados en un
    frame / intervalo de tiempo afectados por la tormenta
end
%     pause(0.05)
%     hold off
```

```
frametormenta=frametormenta|poltormenta; %total de puntos afectados
    para todos los frames temporales.
end
```

## A.6 Agrupación de los puntos vecinos para formar los obstáculos a evitar

```
[row,col] = find(frametormenta);
length(row);
length(col);
vecinosi=zeros(1,1);
vecinosj=zeros(1,1);
vecinosi(1,1)=row(1);
vecinosj(1,1)=col(1);
contador=zeros(1,1);
fila=1;
cont=1;
k=1;
contador(fila,:)=2;
for j=1:length(row)-1
 r=row(j);
 c=col(j);
   if j>=2
      k=j-1;
      while k>0
           if row(k)==r&&col(k)==c-1;
              a=find(vecinosi==r);
              b=find(vecinosj==c-1);
              h=1;
              while h<=length(a)
                  d=find(a(h)==b(:));
                  if length(d)==1
                      h=length(a)+1;
                  else
                      h=h+1;
                  end
              end
               [fila,columna]=ind2sub(size(vecinosi),b(d));
               vecinosi(fila,contador(fila,:))=row(j);
               vecinosj(fila,contador(fila,:))=col(j);
               contador(fila,:)=contador(fila,:)+1;
               k=-5;
           elseif row(k)==r&&col(k)==c+1
               a=find(vecinosi==r);
              b=find(vecinosj==c+1);
              h=1;
              while h<=length(a)
                  d=find(a(h)==b(:));
                  if length(d)==1
```

```
                h=length(a)+1;
            else
                h=h+1;
            end
        end
         [fila,columna]=ind2sub(size(vecinosi),b(d));
         vecinosi(fila,contador(fila,:))=row(j);
         vecinosj(fila,contador(fila,:))=col(j);
         contador(fila,:)=contador(fila,:)+1;
         k=-5;
    elseif row(k)==r-1&&col(k)==c
         a=find(vecinosi==r-1);
        b=find(vecinosj==c);
        h=1;
        while h<=length(a)
            d=find(a(h)==b(:));
            if length(d)==1
                h=length(a)+1;
            else
                h=h+1;
            end
        end
         [fila,columna]=ind2sub(size(vecinosi),b(d));
         vecinosi(fila,contador(fila,:))=row(j);
         vecinosj(fila,contador(fila,:))=col(j);
         contador(fila,:)=contador(fila,:)+1;
         k=-5;
    elseif row(k)==r+1&&col(k)==c
         a=find(vecinosi==r+1);
        b=find(vecinosj==c);
        h=1;
        while h<=length(a)
            d=find(a(h)==b(:));
            if length(d)==1
                h=length(a)+1;
            else
                h=h+1;
            end
        end
         [fila,columna]=ind2sub(size(vecinosi),b(d));
         vecinosi(fila,contador(fila,:))=row(j);
         vecinosj(fila,contador(fila,:))=col(j);
         contador(fila,:)=contador(fila,:)+1;
         k=-5;
         elseif row(k)==r+1&&col(k)==c-1
         a=find(vecinosi==r+1);
        b=find(vecinosj==c-1);
        h=1;
        while h<=length(a)
            d=find(a(h)==b(:));
```

```
    if length(d)==1
        h=length(a)+1;
    else
        h=h+1;
    end
end
 [fila,columna]=ind2sub(size(vecinosi),b(d));
 vecinosi(fila,contador(fila,:))=row(j);
 vecinosj(fila,contador(fila,:))=col(j);
 contador(fila,:)=contador(fila,:)+1;
 k=-5;
 elseif row(k)==r+1&&col(k)==c+1
 a=find(vecinosi==r+1);
b=find(vecinosj==c+1);
h=1;
while h<=length(a)
    d=find(a(h)==b(:));
    if length(d)==1
        h=length(a)+1;
    else
        h=h+1;
    end
end
 [fila,columna]=ind2sub(size(vecinosi),b(d));
 vecinosi(fila,contador(fila,:))=row(j);
 vecinosj(fila,contador(fila,:))=col(j);
 contador(fila,:)=contador(fila,:)+1;
 k=-5;
 elseif row(k)==r-1&&col(k)==c-1
 a=find(vecinosi==r-1);
b=find(vecinosj==c-1);
h=1;
while h<=length(a)
    d=find(a(h)==b(:));
    if length(d)==1
        h=length(a)+1;
    else
        h=h+1;
    end
end
 [fila,columna]=ind2sub(size(vecinosi),b(d));
 vecinosi(fila,contador(fila,:))=row(j);
 vecinosj(fila,contador(fila,:))=col(j);
 contador(fila,:)=contador(fila,:)+1;
 k=-5;
 elseif row(k)==r-1&&col(k)==c+1
 a=find(vecinosi==r-1);
b=find(vecinosj==c+1);
h=1;
while h<=length(a)
```

```
                    d=find(a(h)==b(:));
                    if length(d)==1
                        h=length(a)+1;
                    else
                        h=h+1;
                    end
                end
                 [fila,columna]=ind2sub(size(vecinosi),b(d));
                 vecinosi(fila,contador(fila,:))=row(j);
                 vecinosj(fila,contador(fila,:))=col(j);
                 contador(fila,:)=contador(fila,:)+1;
                 k=-5;
              end
            k=k-1;

        end
         if k==0
         cont=cont+1;
         fila=cont;
         vecinosi(fila,1)=row(j);
         vecinosj(fila,1)=col(j);
         contador(fila,:)=2;
          end

    end
end
vecinosi;
vecinosj;
contador(:,1)=contador(:,1)-1;
contador;

[F,C]=size(vecinosi);
cuent=1;
i=0;
while i<=F-1
    i=i+1;
    j=1;
    while j<=contador(i,:)
        k=i+1;
        while k<=F
            [i,k,F];
            l=1;
            while l<=contador(k,:)

                [l,contador(k,:)];
                if vecinosi(i,j)==vecinosi(k,l)&&vecinosj(i,j)==vecinosj(
                    k,l)+1||vecinosi(i,j)==vecinosi(k,l)&&vecinosj(i,j)==
                    vecinosj(k,l)-1||vecinosi(i,j)==vecinosi(k,l)+1&&
                    vecinosj(i,j)==vecinosj(k,l)||vecinosi(i,j)==vecinosi(
                    k,l)-1&&vecinosj(i,j)==vecinosj(k,l)...
```

```
                ||vecinosi(i,j)==vecinosi(k,l)-1&&vecinosj(i,j)==
                    vecinosj(k,l)+1||vecinosi(i,j)==vecinosi(k,l)-1&&
                    vecinosj(i,j)==vecinosj(k,l)-1||vecinosi(i,j)==
                    vecinosi(k,l)+1&&vecinosj(i,j)==vecinosj(k,l)-1||
                    vecinosi(i,j)==vecinosi(k,l)+1&&vecinosj(i,j)==
                    vecinosj(k,l)+1
                matcambios(cuent,:)=[vecinosi(i,j),vecinosj(i,j),
                    vecinosi(k,l),vecinosj(k,l)];
                cuent=cuent+1;
                vecinosi(i,contador(i,:)+1:contador(i,:)+contador(k,:))
                    =vecinosi(k,1:contador(k,:));
                vecinosj(i,contador(i,:)+1:contador(i,:)+contador(k,:))
                    =vecinosj(k,1:contador(k,:));
                contador(i,:)=contador(i,:)+contador(k,:);
                vecinosi(k,:)=[];
                vecinosj(k,:)=[];
                contador(k,:)=[];
                F=F-1; l=0; %%%%%%%%%%%%%%%%%%%%%%
            end
            if k>F
                break
            end
            l=l+1;
        end
        k=k+1;
    end
    j=j+1;
  end
end
```

## A.7 Función que permite la fusión de las envolventes convexas interseccionadas

```
 z=1;
 while z<=n
     if z>=2
[xv,yv,ch,chindice,xv2,yv2,z,n,mlength]=fusionpoligonos(xv2,yv2,z,xv,yv,
   ch,chindice,n,mlength);
     end
 z=z+1;
end

function [xv,yv,ch,chindice,xv2,yv2,z,n,mlength]=...
   fusionpoligonos(xv2,yv2,z,xv,yv,ch,chindice,n,mlength)
   p=1;
   while p<z
       if z~=p
[xi,yi]=polyxpoly(xv2(p,1:chindice(p)),yv2(p,1:chindice(p)),...
```

```
    xv2(z,1:chindice(z)),yv2(z,1:chindice(z)));
[in1,on1]=inpolygon(xv2(p,1:chindice(p)),yv2(p,1:chindice(p)),...
    xv2(z,1:chindice(z)),yv2(z,1:chindice(z)));
[in2,on2]=inpolygon(xv2(z,1:chindice(z)),yv2(z,1:chindice(z)),...
    xv2(p,1:chindice(p)),yv2(p,1:chindice(p)));
xi3=find(in2);
xi2=find(in1);
if length(xi)>1||length(xi3)==length(in2)
    plength=mlength(z,:)+mlength(p,:);
    xv(p,1:(mlength(z,:)+mlength(p,:)))=...
        [xv(p,1:mlength(p,:)),xv(z,1:mlength(z,:))];
    yv(p,1:(mlength(z,:)+mlength(p,:)))=...
        [yv(p,1:mlength(p,:)),yv(z,1:mlength(z,:))];
    mlength(p,:)=plength;
    chv=convhull(xv(p,1:mlength(p,:)),yv(p,1:mlength(p,:)))';
    ch(p,1:length(chv))=chv(:);
    chindice(p)=length(chv);
    xv1=xv(p,chv(:));yv1=yv(p,chv(:));
    xv2(p,1:length(chv))=xv1; yv2(p,1:length(chv))=yv1;
    xv(z,:)=[];yv(z,:)=[];ch(z,:)=[];chindice(z)=[];
    xv2(z,:)=[];yv2(z,:)=[];mlength(z,:)=[];
    n=n-1; z=0; p=0;
elseif length(xi2)==length(in1)
     plength=mlength(z,:)+mlength(p,:);
    xv(z,1:(mlength(z,:)+mlength(p,:)))=...
        [xv(z,1:mlength(z,:)),xv(p,1:mlength(p,:))];
    yv(z,1:(mlength(z,:)+mlength(p,:)))=...
        [yv(z,1:mlength(z,:)),yv(p,1:mlength(p,:))];
    mlength(z,:)=plength;
    chv=convhull(xv(z,1:mlength(z,:)),yv(z,1:mlength(z,:)))';
    ch(z,1:length(chv))=chv(:);
    chindice(z)=length(chv);
    xv1=xv(z,chv(:));yv1=yv(z,chv(:));
    xv2(z,1:length(chv))=xv1; yv2(z,1:length(chv))=yv1;
    xv(p,:)=[];yv(p,:)=[];ch(p,:)=[];chindice(p)=[];
    xv2(p,:)=[];yv2(p,:)=[];mlength(p,:)=[];
    n=n-1; z=0; p=0;
end
        end
        p=p+1;
    end
end
```

## A.8 Obtención de las tangentes entre puntos y polígonos

```
function [tangente]=tangpointtopolygon(P,xv,yv,l,chindice)
tangente=0;
e1=[xv(l+1)-xv(l),yv(l+1)-yv(l)];
```

```
if l<chindice-1
e2=[xv(l+2)-xv(l+1),yv(l+2)-yv(l+1)];
else
e2=[xv(2)-xv(l+1),yv(2)-yv(l+1)];
end
if e1(1)<0&&(P(1)-xv(l+1))*(e1(2)/e1(1))<=(P(2)-yv(l+1))||...
   e1(1)>0&&(P(1)-xv(l+1))*(e1(2)/e1(1))>=(P(2)-yv(l+1))||...
   e1(1)==0&&e1(2)>0&&(P(1)-xv(l+1))>=0||...
   e1(1)==0&&e1(2)<0&&(P(1)-xv(l+1))<=0||...
   e1(2)==0&&e1(1)>0&&(P(2)-yv(l+1))<=0||...
   e1(2)==0&&e1(1)<0&&(P(2)-yv(l+1))>=0 %a la derecha del vertice
   if e2(1)>0&&(P(1)-xv(l+1))*(e2(2)/e2(1))<=(P(2)-yv(l+1))||...
      e2(1)<0&&(P(1)-xv(l+1))*(e2(2)/e2(1))>=(P(2)-yv(l+1))||...
      e2(1)==0&&e2(2)>0&&(P(1)-xv(l+1))<=0||...
      e2(1)==0&&e2(2)<0&&(P(1)-xv(l+1))>=0||...
      e2(2)==0&&e2(1)>0&&(P(2)-yv(l+1))>=0||...
      e2(2)==0&&e2(1)<0&&(P(2)-yv(l+1))<=0
         tangente=1;
end
elseif e1(1)<0&&(P(1)-xv(l+1))*(e1(2)/e1(1))>=(P(2)-yv(l+1))||e1(1)>0&&(
    P(1)-xv(l+1))*(e1(2)/e1(1))<=(P(2)-yv(l+1))||e1(1)==0&&e1(2)>0&&(P
    (1)-xv(l+1))<=0||e1(1)==0&&e1(2)<0&&(P(1)-xv(l+1))>=0||e1(2)==0&&e
    1(1)>0&&(P(2)-yv(l+1))>=0||e1(2)==0&&e1(1)<0&&(P(2)-yv(l+1))<=0
if e2(1)>0&&(P(1)-xv(l+1))*(e2(2)/e2(1))>=(P(2)-yv(l+1))||e2(1)<0&&(P(1)
    -xv(l+1))*(e2(2)/e2(1))<=(P(2)-yv(l+1))||e2(1)==0&&e2(2)>0&&(P(1)-xv
    (l+1))>=0||e2(1)==0&&e2(2)<0&&(P(1)-xv(l+1))<=0||e2(2)==0&&e2(1)
    >0&&(P(2)-yv(l+1))<=0||e2(2)==0&&e2(1)<0&&(P(2)-yv(l+1))>=0
tangente=1;
end
end
end
```

## A.9 Eliminación de las tangentes entre punto y polígonos no visibles

```
function [xtan,ytan,zvert0poly,Origen_destino]=
    cortetangentespuntopoligono(P,xtan,ytan,xv,yv,n,zvert0poly,PD,
    chindice,zconf,puntodentro)
         Origen_destino=1;
         if puntodentro==0&&zconf~=0
             [X,Y]=size(xtan);
             zvert0poly=zeros(X,Y);
             Origen_destino=1;
         else
         for z=1:n
             if z~=zconf
                 for l=1:length(xtan(z,:))
                     for p=n:-1:1
             if z~=p
```

```
                x1=[P(1),xtan(z,l)];
                y1=[P(2),ytan(z,l)] ;
            [xi,yi]=polyxpoly(x1,y1,xv(p,1:chindice(p)),yv(p,1:chindice(
                p))) ;
               L=length(xi);
              %if L>2
            for i=1:length(xi);
                if xi(i)==P(1)&&yi(i)==P(2)
                    L=L-1;
                end
            end
              %end
            if p==zconf&&L>0
            zvert0poly(z,l)=0;
            break;
            elseif p~=zconf&&L>1
                zvert0poly(z,l)=0;
                break
            end
%             end
            end
            end
            %end
            end
            if Origen_destino==1
            x2=[P(1),PD(1)];y2=[P(2),PD(2)];

            [xi2,yi2]=polyxpoly(x2,y2,xv(z,1:chindice(z)),yv(z,1:
                chindice(z)));
            [z,Origen_destino];
            xi2;
            if length(xi2)>1
                Origen_destino=0;
            end
            end
            end
        end
        end
    end
```

## A.10 Obtención de las tangentes entre polígonos

```
for z=1:n
    for p=z+1:n
    cont=1;
    for l=2:chindice(z)
        [indz,indp]=tangpolygontopolygon(xv2,yv2,l,z,p,chindice) ;
```

```
        for t=1:length(indz)
            if indz(t)~=0
            indtanz(cuentador,cont)=indz(t); %matriz que recoge los
                indices de los puntos de tangencia del polígono original
                z
            indtanp(cuentador,cont)=indp(t);
            %matriz que recoge los indices de los puntos de tangencia del
                 polígono original z+1
            xtanz_1(cont)=xv2(z,indz(t));
            xtanz(cont)=xv2(p,indp(t));
            ytanz_1(cont)=yv2(z,indz(t));
            ytanz(cont)=yv2(p,indp(t));
            cont=cont+1;
            end
        end
    end
    mat1(z,p)=indtanz(cuentador,1);
    mat1(p,z)=indtanp(cuentador,1);
    mat2(z,p)=indtanz(cuentador,2);
    mat2(p,z)=indtanp(cuentador,2);
    mat3(z,p)=indtanz(cuentador,3);
    mat3(p,z)=indtanp(cuentador,3);
    mat4(z,p)=indtanz(cuentador,4);
    mat4(p,z)=indtanp(cuentador,4);
    cuentador=cuentador+1;
    end
end

function [indz,indp]=tangpolygontopolygon(xv2,yv2,l,z,p,chindice)
  P=[xv2(z,l),yv2(z,l)];
  contador=1;
  for t=1:(chindice(p)-1)
    tangente=0;
    [tangente]=tangpointtopolygon(P,xv2(p,:),yv2(p,:),t,chindice(p));
    if tangente==1
       tangente=0;
       %[xv(z,t),yv(z,t)]
       [tangente]=tangpointtopolygon([xv2(p,t+1),yv2(p,t+1)],xv2(z,:),...

       yv2(z,:),l-1,chindice(z));
         if tangente==1
             indz(contador)=l;
             indp(contador)=t+1;
             contador=contador+1;
         end
             indz(contador)=0;
             indp(contador)=0;
             contador=contador+1;
         end
         end
```

```
  end
```

## A.11 Eliminación de las tangentes entre polígonos no visibles

```
function [mat1,mat2,mat3,mat4]=cortetangenteentrepoligono(xv,yv,mat1,mat
    2,mat3,mat4,n,chindice)
        %se ha cambiado y las curvas son al rumbo constante de matlab

         for z=1:n
             for p=z+1:n
                 xline1=[xv(z,mat1(z,p)),xv(p,mat1(p,z))];
                 xline2=[xv(z,mat2(z,p)),xv(p,mat2(p,z))];
                 xline3=[xv(z,mat3(z,p)),xv(p,mat3(p,z))];
                 xline4=[xv(z,mat4(z,p)),xv(p,mat4(p,z))];
                 yline1=[yv(z,mat1(z,p)),yv(p,mat1(p,z))];
                 yline2=[yv(z,mat2(z,p)),yv(p,mat2(p,z))];
                 yline3=[yv(z,mat3(z,p)),yv(p,mat3(p,z))];
                 yline4=[yv(z,mat4(z,p)),yv(p,mat4(p,z))];
%                  [a,b]=track2('rh',yv(z,mat1(z,p)),xv(z,mat1(z,p)),yv(p,
    mat1(p,z)),xv(p,mat1(p,z)));
%                  [c,d]=track2('rh',yv(z,mat2(z,p)),xv(z,mat2(z,p)),yv(p,
    mat2(p,z)),xv(p,mat2(p,z)));
%                  [e,f]=track2('rh',yv(z,mat3(z,p)),xv(z,mat3(z,p)),yv(p,
    mat3(p,z)),xv(p,mat3(p,z)));
%                  [g,h]=track2('rh',yv(z,mat4(z,p)),xv(z,mat4(z,p)),yv(p,
    mat4(p,z)),xv(p,mat4(p,z)));
                 for t=1:n
                     if t~=z&&t~=p
%                           s=geoshape(yv(t,1:chindice(t)),xv(t,1:chindice(
    t)));
%                           [x1,y1] = polyxpoly(s.Longitude,s.Latitude,b,a);

                        [x1,y1] = polyxpoly(xline1,yline1,xv(t,1:chindice
                            (t)),yv(t,1:chindice(t)));
                        if length(x1)>1
                            mat1(z,p)=0;
                            mat1(p,z)=0;
                        end
                        %[x2,y2] = polyxpoly(s.Longitude,s.Latitude,d,c);
                        [x2,y2] = polyxpoly(xline2,yline2,xv(t,1:chindice
                            (t)),yv(t,1:chindice(t)));
                        if length(x2)>1
                            mat2(z,p)=0;
                            mat2(p,z)=0;
                        end
                        %[x3,y3] = polyxpoly(s.Longitude,s.Latitude,f,e);
```

```
                        [x3,y3] = polyxpoly(xline3,yline3,xv(t,1:chindice
                            (t)),yv(t,1:chindice(t)));
                        if length(x3)>1
                            mat3(z,p)=0;
                            mat3(p,z)=0;
                        end
                        %[x4,y4] = polyxpoly(s.Longitude,s.Latitude,h,g);
                        [x4,y4] = polyxpoly(xline4,yline4,xv(t,1:chindice
                            (t)),yv(t,1:chindice(t)));
                        if length(x4)>1
                            mat4(z,p)=0;
                            mat4(p,z)=0;
                        end

                    end
                end
            end
        end
    end


   end
```

## A.12 Obtención de la matriz de incidencia, costes y coordenadas

```
contador=1;
if puntoordentro==1
    matinc(1,contador)=1
    matinc(contchconf0,contador)=-1
    segment(contador,:)=[contador,1,contchconf0];
    matcost(contador,:)=[contador,t0];
    matcostd(contador,:)=[contador,rec0];
    matcourse(contador,:)=[contador,course0]
    contador=contador+1
end
if Origen_destino==1
    matinc(contchconf0,contador)=1;
    matinc(contchconfD,contador)=-1;
    segment(contador,:)=[contador,contchconf0,contchconfD];
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(PD(1)+P0(1)
        )/2,(PD(2)+P0(2))/2,Wu,pasomallalambda,pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(PD(1)+P0(1)
        )/2,(PD(2)+P0(2))/2,Wv,pasomallalambda,pasomallaphi,M,N);
    [r,course]=loxodromicequations(P0(2),P0(1),PD(2),PD(1),Rt,Theta,
        vrefISA,0,0);
    [y,t]=ode45(@(y,t)loxodromicequationsint(P0(2),P0(1),Rt,WuintD,
        WvintD,Theta,vrefISA,course,y,t),[0,r],0);
```

```
    %matcost(contador,:)=[contador,Rt*sqrt((PD(2)-P0(2))^2+(cos((PD(2)+P
        0(2))/2)*(PD(1)-P0(1)))^2)];
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
    contador=contador+1;

end

puntoordentro
zorconf
%Seguramente se podrá cambiar los for por :
for l=1:length(vectanpoint0)
    if puntoordentro==1
    matinc(contchconf0,contador)=1;
    matinc(vectanpoint0(l),contador)=-1;
    segment(contador,:)=[contador,contchconf0,vectanpoint0(l)];
    [ind_x,ind_y]=ind2sub(size(match),find(vectanpoint0(l)==match,1));
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+P0(1))/2,(yv2(ind_x,ind_y)+P0(2))/2,Wu,pasomallalambda,
        pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+P0(1))/2,(yv2(ind_x,ind_y)+P0(2))/2,Wv,pasomallalambda,
        pasomallaphi,M,N);
    [r,course]=loxodromicequations(P0(2),P0(1),yv2(ind_x,ind_y),xv2(ind_
        x,ind_y),Rt,Theta,vrefISA,0,0);
    [y,t]=ode45(@(y,t)loxodromicequationsint(P0(2),P0(1),Rt,WuintD,
        WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
    %matcost(contador,:)=[contador,Rt*sqrt((yv2(ind_x,ind_y)-P0(2))^2+(
        cos((yv2(ind_x,ind_y)+P0(2))/2)*(xv2(ind_x,ind_y)-P0(1)))^2)];
    contador=contador+1;
    elseif puntoordentro==0&&zorconf==0
    matinc(1,contador)=1;
    matinc(vectanpoint0(l),contador)=-1;
    segment(contador,:)=[contador,1,vectanpoint0(l)];
    [ind_x,ind_y]=ind2sub(size(match),find(vectanpoint0(l)==match,1));
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+P0(1))/2,(yv2(ind_x,ind_y)+P0(2))/2,Wu,pasomallalambda,
        pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+P0(1))/2,(yv2(ind_x,ind_y)+P0(2))/2,Wv,pasomallalambda,
        pasomallaphi,M,N);
    [r,course]=loxodromicequations(P0(2),P0(1),yv2(ind_x,ind_y),xv2(ind_
        x,ind_y),Rt,Theta,vrefISA,0,0);
    [y,t]=ode45(@(y,t)loxodromicequationsint(P0(2),P0(1),Rt,WuintD,
        WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    matcost(contador,:)=[contador,t(length(t))];
```

```
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
    %matcost(contador,:)=[contador,Rt*sqrt((yv2(ind_x,ind_y)-P0(2))^2+(
        cos((yv2(ind_x,ind_y)+P0(2))/2)*(xv2(ind_x,ind_y)-P0(1)))^2)];
    contador=contador+1;
    end
end
for l=1:length(vectanpointD)
    if puntodesdentro==1
    matinc(contchconfD,contador)=-1;
    matinc(vectanpointD(l),contador)=1;
    segment(contador,:)=[contador,vectanpointD(l),contchconfD];
    [ind_x,ind_y]=ind2sub(size(match),find(vectanpointD(l)==match,1));
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+PD(1))/2,(yv2(ind_x,ind_y)+PD(2))/2,Wu,pasomallalambda,
        pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+PD(1))/2,(yv2(ind_x,ind_y)+PD(2))/2,Wv,pasomallalambda,
        pasomallaphi,M,N);
    [yv2(ind_x,ind_y),xv2(ind_x,ind_y)];
    [r,course]=loxodromicequations(yv2(ind_x,ind_y),xv2(ind_x,ind_y),PD
        (2),PD(1),Rt,Theta,vrefISA,0,0);
    [y,t]=ode45(@(y,t)loxodromicequationsint(yv2(ind_x,ind_y),xv2(ind_x,
        ind_y),Rt,WuintD,WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
    %matcost(contador,:)=[contador,Rt*sqrt((yv2(ind_x,ind_y)-PD(2))^2+(
        cos((yv2(ind_x,ind_y)+PD(2))/2)*(xv2(ind_x,ind_y)-PD(1)))^2)];
    contador=contador+1;
    elseif puntodesdentro==0&&zdesconf==0
    matinc(contch,contador)=1;
    matinc(vectanpointD(l),contador)=-1;
    segment(contador,:)=[contador,vectanpointD(l),contch];
    [ind_x,ind_y]=ind2sub(size(match),find(vectanpointD(l)==match,1));
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+PD(1))/2,(yv2(ind_x,ind_y)+PD(2))/2,Wu,pasomallalambda,
        pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+PD(1))/2,(yv2(ind_x,ind_y)+PD(2))/2,Wv,pasomallalambda,
        pasomallaphi,M,N);
    [yv2(ind_x,ind_y),xv2(ind_x,ind_y)];
    [r,course]=loxodromicequations(yv2(ind_x,ind_y),xv2(ind_x,ind_y),PD
        (2),PD(1),Rt,Theta,vrefISA,0,0);
    [y,t]=ode45(@(y,t)loxodromicequationsint(yv2(ind_x,ind_y),xv2(ind_x,
        ind_y),Rt,WuintD,WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
```

```
    %matcost(contador,:)=[contador,Rt*sqrt((yv2(ind_x,ind_y)-PD(2))^2+(
        cos((yv2(ind_x,ind_y)+PD(2))/2)*(xv2(ind_x,ind_y)-PD(1)))^2)];
    contador=contador+1;
    end
end

for z=1:n
    for l=chindice(z):-1:2
        matinc(match(z,l),contador)=1;
        matinc(match(z,l-1),contador)=-1;
        segment(contador,:)=[contador,match(z,l),match(z,l-1)];
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(z,l-1)+
        xv2(z,l))/2,(yv2(z,l-1)+yv2(z,l))/2,Wu,pasomallalambda,
        pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(z,l-1)+
        xv2(z,l))/2,(yv2(z,l-1)+yv2(z,l))/2,Wv,pasomallalambda,
        pasomallaphi,M,N);
    [r,course]=loxodromicequations(yv2(z,l),xv2(z,l),yv2(z,l-1),xv2(z,l
        -1),Rt,Theta,vrefISA,0,0) ;
    if r~=0
    [y,t]=ode45(@(y,t)loxodromicequationsint(yv2(z,l),xv2(z,l),Rt,WuintD,
        WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    else
    t=0;
    end
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
        %matcost(contador,:)=[contador,Rt*sqrt((yv2(z,t)-yv2(z,t-1))^2+(
            cos((yv2(z,t)+yv2(z,t-1))/2)*(xv2(z,t)-xv2(z,t-1)))^2)];
        contador=contador+1;
        matinc(match(z,l),contador)=-1;
        matinc(match(z,l-1),contador)=1;
        segment(contador,:)=[contador,match(z,l-1),match(z,l)];
        [Wuint0]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(z,l
            -1)+xv2(z,l))/2,(yv2(z,l-1)+yv2(z,l))/2,Wu,pasomallalambda,
            pasomallaphi,M,N);
    [Wvint0]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(z,l-1)+
        xv2(z,l))/2,(yv2(z,l-1)+yv2(z,l))/2,Wv,pasomallalambda,
        pasomallaphi,M,N);
    [r,course]=loxodromicequations(yv2(z,l-1),xv2(z,l-1),yv2(z,l),xv2(z,
        l),Rt,Theta,vrefISA,0,0) ;
    if r~=0
    [y,t]=ode45(@(y,t)loxodromicequationsint(yv2(z,l-1),xv2(z,l-1),Rt,
        Wuint0,Wvint0,Theta,vrefISA,course,y,t),[0,r],0);
    else
    t=0;
    end
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
```

```
    matcourse(contador,:)=[contador,course];
        %matcost(contador,:)=[contador,Rt*sqrt((yv2(z,t)-yv2(z,t-1))^2+(
            cos((yv2(z,t)+yv2(z,t-1))/2)*(xv2(z,t)-xv2(z,t-1)))^2)];
        contador=contador+1;


    end
end
for l=1:length(vectanorden)
    matinc(vectanorden(l),contador)=1;
    matinc(vectanorden2(l),contador)=-1;
    segment(contador,:)=[contador,vectanorden(l),vectanorden2(l)];
    [ind_x1,ind_y1]=ind2sub(size(match),find(vectanorden(l)==match,1));
    [ind_x2,ind_y2]=ind2sub(size(match),find(vectanorden2(l)==match,1));
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x2,
        ind_y2)+xv2(ind_x1,ind_y1))/2,(yv2(ind_x2,ind_y2)+yv2(ind_x1,ind_
        y1))/2,Wu,pasomallalambda,pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x2,
        ind_y2)+xv2(ind_x1,ind_y1))/2,(yv2(ind_x2,ind_y2)+yv2(ind_x1,ind_
        y1))/2,Wv,pasomallalambda,pasomallaphi,M,N);
    [r,course]=loxodromicequations(yv2(ind_x1,ind_y1),xv2(ind_x1,ind_y1),
        yv2(ind_x2,ind_y2),xv2(ind_x2,ind_y2),Rt,Theta,vrefISA,0,0);
    if r~=0
    [y,t]=ode45(@(y,t)loxodromicequationsint(yv2(ind_x1,ind_y1),xv2(ind_
        x1,ind_y1),Rt,WuintD,WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    else
    t=0
    end
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
    %matcost(contador,:)=[contador,Rt*sqrt((yv2(ind_x1,ind_y1)-yv2(ind_x
        2,ind_y2))^2+(cos((yv2(ind_x1,ind_y1)+yv2(ind_x2,ind_y2))/2)*(xv
        2(ind_x1,ind_y1)-xv2(ind_x2,ind_y2)))^2)];
    contador=contador+1;
    matinc(vectanorden(l),contador)=-1;
    matinc(vectanorden2(l),contador)=1;
    segment(contador,:)=[contador,vectanorden2(l),vectanorden(l)];
    [Wuint0]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x2,
        ind_y2)+xv2(ind_x1,ind_y1))/2,(yv2(ind_x2,ind_y2)+yv2(ind_x1,ind_
        y1))/2,Wu,pasomallalambda,pasomallaphi,M,N);
    [Wvint0]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x2,
        ind_y2)+xv2(ind_x1,ind_y1))/2,(yv2(ind_x2,ind_y2)+yv2(ind_x1,ind_
        y1))/2,Wv,pasomallalambda,pasomallaphi,M,N);
    [r,course]=loxodromicequations(yv2(ind_x2,ind_y2),xv2(ind_x2,ind_y2),
        yv2(ind_x1,ind_y1),xv2(ind_x1,ind_y1),Rt,Theta,vrefISA,0,0);
    if r~=0
    [y,t]=ode45(@(y,t)loxodromicequationsint(yv2(ind_x2,ind_y2),xv2(ind_
        x2,ind_y2),Rt,Wuint0,Wvint0,Theta,vrefISA,course,y,t),[0,r],0);
    else
```

```
    t=0
    end
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
     %matcost(contador,:)=[contador,Rt*sqrt((yv2(ind_x1,ind_y1)-yv2(ind_
        x2,ind_y2))^2+(cos((yv2(ind_x1,ind_y1)+yv2(ind_x2,ind_y2))/2)*(
        xv2(ind_x1,ind_y1)-xv2(ind_x2,ind_y2)))^2)];
    contador=contador+1;
end

if puntodesdentro==1
    matinc(contch,contador)=-1
    matinc(contchconfD,contador)=1
    segment(contador,:)=[contador,contchconfD,contch];
    matcost(contador,:)=[contador,tD];
    matcostd(contador,:)=[contador,recD];
    matcourse(contador,:)=[contador,courseD];
    contador=contador+1
end
```

## A.13 Implementación del algoritmo de horizonte deslizante

```
G=digraph(segment(:,2),segment(:,3),matcost(:,2));
P = shortestpath(G,1,contch)
Costecam=zeros(1);
Costecamd=zeros(1);
for i=2:length(P(:))
a=find(segment(:,2)==P(i-1));
b=find(segment(:,3)==P(i));
h=1;
while h<=length(a)
d=find(a(h)==b(:));
if length(d)==1
h=length(a)+1;
else
h=h+1;
end
end
Costecam(i)=matcost(b(d),2)+Costecam(i-1)
Costecamd(i)=matcostd(b(d),2)+Costecamd(i-1);
end

trec=5*60;
if trec>=Costecam(length(P(:)))
    lambda0=lambdaD;
    phi0=phiD;
    P0=[lambda0,phi0];
```

```
    if bucle>0
    [a,b]=track2('rh',nodospasoy(length(nodospasox))*180/pi,nodospasox(
       length(nodospasox))*180/pi,phi0*180/pi,lambda0*180/pi);
       plotm(a,b,'k')
       hold on
end
else
nodopos=P(find(Costecam>trec,1))
nodoant=P(length(find(Costecam<trec)))
tstop=trec-Costecam(length(find(Costecam<trec)));
xvant=matcoord(nodoant,2)
yvant=matcoord(nodoant,3)
xvpos=matcoord(nodopos,2)
yvpos=matcoord(nodopos,3)
%Dudas
[Wuint]=funcioninterpolavientos(lambdaviento,phiviento,(xvant+xvpos)/2,(
   yvant+yvpos)/2,Wu,pasomallalambda,pasomallaphi,M,N);
[Wvint]=funcioninterpolavientos(lambdaviento,phiviento,(xvant+xvpos)/2,(
   yvant+yvpos)/2,Wv,pasomallalambda,pasomallaphi,M,N);
[r1,course1]=loxodromicequations(yvant,xvant,yvpos,xvpos,Rt,Theta,
   vrefISA,0,0)
options = odeset('Events',@(y,t)myEventsFcn(tstop,y,t));
[y,t,ye,te,ie]=ode45(@(y,t)loxodromicequationsint(yvant,xvant,Rt,Wuint,
   Wvint,Theta,vrefISA,course1,y,t),[Costecamd(find(Costecam<trec,1)),
   Costecamd(find(Costecam>trec,1))-Costecamd(length(find(Costecam<trec
   ,1)))],0,options);
[y(length(y)),t(length(y))]
[ye,te]

if bucle>0
    [a,b]=track2('rh',nodospasoy(length(nodospasox))*180/pi,nodospasox(
       length(nodospasox))*180/pi,phi0*180/pi,lambda0*180/pi);
       plotm(a,b,'k')
       hold on
end
course1
[phi0,lambda0]=loxodromicequationsinv(yvant,xvant,Rt,Theta,vrefISA,
   course1,ye,yvpos,xvpos);
P0=[lambda0,phi0]
end
for i=1:length(find(Costecam<trec))
    nodospasox(puntospaso)=matcoord(P(i),2);
    nodospasoy(puntospaso)=matcoord(P(i),3);
    puntospaso=puntospaso+1
end
segmentositer(bucle+1)=length(find(Costecam<trec))+1
bucle=bucle+1
```

## A.14 Función evento para obtener la posición del nuevo origen

```
function [value,isterminal,direction] = myEventsFcn(tstop,y,t)
value=t-tstop;
 isterminal=1;
direction=0;
end
```

## A.15 Maniobra de evasión

```
 for z=1:n
     [in1_2,on1_2]=inpolygon(lambda0,phi0,xv2(z,1:chindice(z)),yv2(z,1:
        chindice(z)));
[in2_2,on2_2]=inpolygon(lambdaD,phiD,xv2(z,1:chindice(z)),yv2(z,1:
   chindice(z)));
if in1_2==1
     puntoordentro=1;
     zorconf=z;
end%Punto origen dentro de un polígono
if in2_2==1
    puntodesdentro=1;
    zdesconf=z
end
 end
if puntoordentro==1&&zdesconf==zorconf
    puntoordentro=0;
    puntodesdentro=0;
else
 if puntoordentro==1
     rint0=zeros(1);
     courseint0=zeros(1);
    for j=1:chindice(zorconf)
       yv2(zorconf,j)=atan(sinh(yv2(zorconf,j)));
    end
phi0=atan(sinh(phi0));
phiD=atan(sinh(phiD));
PD=[lambdaD,phiD];
P0=[lambda0,phi0];
     for l=1:chindice(zorconf)
       P0=[lambda0,phi0];
       [rint0(l),courseint0(l)]=loxodromicequations(phi0,lambda0,yv2(
          zorconf,l),xv2(zorconf,l),Rt,Theta,vrefISA,0,0);

    end
    [rord,ind]=sort(rint0);
    courseint=courseint0(ind(:))
    bucle
```

```
if bucle==0
        [r0,course1]=loxodromicequations(phi0,lambda0,phiD,lambdaD,Rt
            ,Theta,vrefISA,0,0);
         for l=1:chindice(zorconf)
             course0=courseint(l);
             if abs(course1)<=3*pi/4&&course1+pi/4>=course0&&course1-
                 pi/4<=course0||course1>3*pi/4&&course0<0&&course
                 0<=-(3*pi/4+pi-course1)||course1>3*pi/4&&course0>0&&
                 course1-pi/4<=course0&&pi-course0<=pi-course1||course
                 1<-3*pi/4&course0<0&&abs(course1)-pi/4<=abs(course0)
                 &&pi-abs(course0)<=pi-abs(course1)||course1<-3*pi/4&&
                 course0>0&&-course0<=-(3*pi/4+pi-abs(course1))
         nodoescape0=[xv2(zorconf,ind(l)),yv2(zorconf,ind(l))];
        [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(P
            0(1)+nodoescape0(1))/2,(P0(2)+nodoescape0(2))/2,Wu,
            pasomallalambda,pasomallaphi,M,N);
        [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(P
            0(1)+nodoescape0(1))/2,(P0(2)+nodoescape0(2))/2,Wv,
            pasomallalambda,pasomallaphi,M,N);
        [y,t]=ode45(@(y,t)loxodromicequationsint(P0(2),P0(1),Rt,
            WuintD,WvintD,Theta,vrefISA,courseint(l),y,t),[0,rord(l)
            ],0);
        t0=t(length(t));

        for j=1:chindice(zorconf)
         yv2(zorconf,j)=0.5*log((1+sin(yv2(zorconf,j)))/(1-sin(yv2(
             zorconf,j))));
        end
        phi0=0.5*log((1+sin(phi0))/(1-sin(phi0)));
        nodoescape0(2)=0.5*log((1+sin(nodoescape0(2)))/(1-sin(
            nodoescape0(2))));
        phiD=0.5*log((1+sin(phiD))/(1-sin(phiD)));
        P0=[lambda0,phi0];
        PD=[lambdaD,phiD];
        rec0=rord(l);
        lescape0=ind(l);


        nodoaux0=P0(:)';
        P0=nodoescape0(:)';
        lambda0=P0(1);
        phi0=P0(2);
        plot(P0(1)*180/pi,atan(sinh(P0(2)))*180/pi,'r*')
        break
             end
         end
else
for l=1:chindice(zorconf)
        course0=courseint(l);
        if sign(course1)==sign(course0)
```

```
    if abs(course1-courseint(l))<=45*pi/180
nodoescape0=[xv2(zorconf,ind(l)),yv2(zorconf,ind(l))]
[WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(P
    0(1)+nodoescape0(1))/2,(P0(2)+nodoescape0(2))/2,Wu,
    pasomallalambda,pasomallaphi,M,N);
[WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(P
    0(1)+nodoescape0(1))/2,(P0(2)+nodoescape0(2))/2,Wv,
    pasomallalambda,pasomallaphi,M,N);
[y,t]=ode45(@(y,t)loxodromicequationsint(P0(2),P0(1),Rt,
    WuintD,WvintD,Theta,vrefISA,courseint(l),y,t),[0,rord(l)
    ],0);
t0=t(length(t));
course0=courseint(l);
for j=1:chindice(zorconf)
 yv2(zorconf,j)=0.5*log((1+sin(yv2(zorconf,j)))/(1-sin(yv2(
     zorconf,j))));
end
phi0=0.5*log((1+sin(phi0))/(1-sin(phi0)));
nodoescape0(2)=0.5*log((1+sin(nodoescape0(2)))/(1-sin(
    nodoescape0(2))));
phiD=0.5*log((1+sin(phiD))/(1-sin(phiD)));
P0=[lambda0,phi0];
PD=[lambdaD,phiD];
rec0=rord(l);
lescape0=ind(l);

nodoaux0=P0(:)';
P0=nodoescape0(:)';
lambda0=P0(1);
phi0=P0(2);
plot(P0(1)*180/pi,atan(sinh(P0(2)))*180/pi,'r*')
break
    end
else
    if abs(course0)>pi/2&&(pi-abs(course0))+(pi-abs(course1))
        <=pi/4||abs(course0)<pi/2&&abs(course0)+abs(course1)<=
        pi/4
nodoescape0=[xv2(zorconf,ind(l)),yv2(zorconf,ind(l))];
[WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(P
    0(1)+nodoescape0(1))/2,(P0(2)+nodoescape0(2))/2,Wu,
    pasomallalambda,pasomallaphi,M,N);
[WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(P
    0(1)+nodoescape0(1))/2,(P0(2)+nodoescape0(2))/2,Wv,
    pasomallalambda,pasomallaphi,M,N);
[y,t]=ode45(@(y,t)loxodromicequationsint(P0(2),P0(1),Rt,
    WuintD,WvintD,Theta,vrefISA,courseint(l),y,t),[0,rord(l)
    ],0);
t0=t(length(t));
course0=courseint(l);
for j=1:chindice(zorconf);
```

```
            yv2(zorconf,j)=0.5*log((1+sin(yv2(zorconf,j)))/(1-sin(yv2(
                zorconf,j))));
            end
            phi0=0.5*log((1+sin(phi0))/(1-sin(phi0)));
            nodoescape0(2)=0.5*log((1+sin(nodoescape0(2)))/(1-sin(
               nodoescape0(2))));
            phiD=0.5*log((1+sin(phiD))/(1-sin(phiD)));
            P0=[lambda0,phi0];
            PD=[lambdaD,phiD];
            rec0=rord(l);
            lescape0=ind(l);

            nodoaux0=P0(:)';
            P0=nodoescape0(:)';
            lambda0=P0(1);
            phi0=P0(2);
            plot(P0(1)*180/pi,atan(sinh(P0(2)))*180/pi,'r*')
            hold on
            break
        end
    end
    end
    end
 end
 puntoordentro
 nodoaux0
 P0

if puntodesdentro==1
    rintD=zeros(1);
    courseintD=zeros(1);
    courseint=zeros(1);
    rordD=zeros(1);
    indD=zeros(1);
     for j=1:chindice(zdesconf)
        yv2(zdesconf,j)=atan(sinh(yv2(zdesconf,j)));
    end
phi0=atan(sinh(phi0));
phiD=atan(sinh(phiD));
PD=[lambdaD,phiD];
P0=[lambda0,phi0];
     for l=1:chindice(zdesconf)
        PD=[lambdaD,phiD];
        [rintD(l),courseintD(l)]=loxodromicequations(phiD,lambdaD,yv2(
            zdesconf,l),xv2(zdesconf,l),Rt,Theta,vrefISA,0,0);
     end
    [rordD,indD]=sort(rintD);
    courseint=courseintD(indD(:));
%    if bucle==0
```

```
    [r0,courseD]=loxodromicequations(phiD,lambdaD,phi0,lambda0,Rt,Theta,
       vrefISA,0,0);
%     else
%        if courseD>=0
%            courseD=courseD-pi;
%        else
%            courseD=courseD+pi;
%        end
%     end
    l=1;
                 course0=courseint(l);
                 %if abs(courseD)<=2*pi/3&&courseD+pi/3>=course0&&courseD
                     -pi/3<=course0||courseD>2*pi/3&&course0<0&&course
                     0<=-(2*pi/3+pi-courseD)||courseD>2*pi/2&&course0>0&&
                     courseD-pi/3<=course0&&pi-course0<=pi-courseD||
                     courseD<-2*pi/3&course0<0&&abs(courseD)-pi/3<=abs(
                     course0)&&pi-abs(course0)<=pi-abs(courseD)||courseD
                     <-2*pi/3&&course0>0&&-course0<=-(2*pi/3+pi-abs(
                     courseD))

              nodoescapeD=[xv2(zdesconf,indD(l)),yv2(zdesconf,indD(l))];
             [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(PD
                (1)+nodoescapeD(1))/2,(PD(2)+nodoescapeD(2))/2,Wu,
                pasomallalambda,pasomallaphi,M,N);
             [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(PD
                (1)+nodoescapeD(1))/2,(PD(2)+nodoescapeD(2))/2,Wv,
                pasomallalambda,pasomallaphi,M,N);
             [y,t]=ode45(@(y,t)loxodromicequationsint(nodoescapeD(2),
                nodoescapeD(1),Rt,WuintD,WvintD,Theta,vrefISA,courseint(l)
                ,y,t),[0,rordD(l)],0);
             tD=t(length(t));
             for j=1:chindice(zdesconf)
              yv2(zdesconf,j)=0.5*log((1+sin(yv2(zdesconf,j)))/(1-sin(yv2(
                  zdesconf,j))));
             end
             phi0=0.5*log((1+sin(phi0))/(1-sin(phi0)));
             nodoescapeD(2)=0.5*log((1+sin(nodoescapeD(2)))/(1-sin(
                nodoescapeD(2))));
             phiD=0.5*log((1+sin(phiD))/(1-sin(phiD)));
             P0=[lambda0,phi0];
             PD=[lambdaD,phiD];
             recD=rordD(l);
             courseD=course0
             lescapeD=indD(l);
             nodoauxD=PD(:)';
             PD=nodoescapeD(:)';
             lambdaD=PD(1);
             phiD=PD(2);
             plot(PD(1)*180/pi,atan(sinh(PD(2)))*180/pi,'r*')
             hold on
```

```
          % break
              % end
    %end
            end
end
```

## A.16 Programa al completo

```
 function TFGsem2()
%Programa TFG ejemplo 1
clc;
close all;
clear all;
% close all;
%Datos iniciales
Pinf=200e2;
Minf=0.8;
gamma=1.4;
Rg=286.9;
vrefISA=sqrt(Minf^2*gamma*Rg*217.434);
Theta=1;
Rt=6378.14e3;
bucle=0;
puntoordentro=0;
puntodesdentro=0;
puntospaso=1;
nodospasox(puntospaso)=0;
zorconf=0;
 %1. Realizamos la discretización

M=(17.5+2.5)/0.1
N=(56-40)/0.1
for i=1:N+1
    for j=1:M+1
        phi(i,j)=(40+(56-40)*(N-i+1)/N)*pi/180;
        lambda(i,j)=(-2.5+(17.5+2.5)*(j-1)/M)*pi/180;
        phiviento(i,j)=(38+(58-38)*(N-i+1)/N)*pi/180;
        lambdaviento(i,j)=(-4.5+(19.5+4.5)*(j-1)/M)*pi/180;
        Wu(i,j)=0;
        Wv(i,j)=10;
    end
end
pasomallalambda=(lambdaviento(1,2)-lambdaviento(1,1))*180/pi
pasomallaphi=(phiviento(1,1)-phiviento(2,1))*180/pi
% M=100
% N=100
% for i=1:N+1
%     for j=1:M+1
```

```
%        phi(i,j)=(40+10*(N-i+1)/N)*pi/180;
%        lambda(i,j)=(10*(j-1)/M)*pi/180;
%        Wu(i,j)=0;
%        Wv(i,j)=10;
%    end
% end
 lambda0=2*pi/180 %1*pi/180%lambda(3*N/5,3*N/5) %(1,1) %origen ejemplo
phi0=41*pi/180% phi(1,1) %phi(1,1) %origen ejemplo
lambdaD=11*pi/180 %Destino ejemplo
phiD=48*pi/180 %Destino ejemplo
P0=[lambda0,phi0];
PD=[lambdaD,phiD];
storms_TFG = load('storms_TFG.mat')
storms_TFG = struct2cell(storms_TFG)

while P0(1)~=PD(1)||P0(2)~=PD(2)&&5*bucle<=90
    puntoordentro=0;
puntodesdentro=0;
zorconf=0;
zdesconf=0;
bucle
% stormsfields= fieldnames(storms_TFG)
% stormsfields(bucle+1,1)
storm_TFG = storms_TFG{bucle+1}
% bucle+1
% storm_TFP=storms_TFG{bucle+1}
[r,course]=loxodromicequations(phi0,lambda0,phi,lambda,Rt,Theta,vrefISA,
    N,M);
r=reshape(r,[(N+1)*(M+1),1]);
course=reshape(course,[(N+1)*(M+1),1]);
for il=1:(N+1)*(M+1)
    if r(il)~=0
        [Wuint]=funcioninterpolavientos(lambdaviento,phiviento,lambda(il)
            ,phi(il),Wu,pasomallalambda,pasomallaphi,M,N);
        [Wvint]=funcioninterpolavientos(lambdaviento,phiviento,lambda(il)
            ,phi(il),Wv,pasomallalambda,pasomallaphi,M,N);
[y,t]=ode45(@(y,t)loxodromicequationsint(phi0,lambda0,Rt,Wuint,Wvint,
    Theta,vrefISA,course(il),y,t),[0,r(il)],0);
tmat(il)=t(length(t))/60;
    else
        tmat(il)=0;

    end


%    V2=vrefISA*sqrt(Theta);
%    Wat2(i,j)=Wu*cos(pi/2-course(i,j))+Wv*sin(pi/2-course(i,j));
%    Wxt2(i,j)=-Wu*sin(pi/2-course(i,j))+Wv*cos(pi/2-course(i,j));
%       % Velocidad respecto a tierra en función de r
%    Vg2(i,j)=sqrt(V2^2-Wxt2(i,j)^2)+Wat2(i,j);
```

```
%     taprox(i,j)=r(i,j)./(Vg2(i,j).*60);
%     maxerror(i,j)=(tmat(i,j)-taprox(i,j))/taprox(i,j)*100;
%
end
tmat=reshape(tmat,[N+1,M+1]);
Wu=reshape(Wu,[N+1,M+1]);
Wv=reshape(Wv,[N+1,M+1]);

%Semana 3 TFG Sem 19/2/20
%Comprobamos comando inpolygon
%Sea un pentagono equilátero:

tiemposinterpol=[0:5:65];
poltormenta=zeros(N+1,M+1);
frametormenta=zeros(N+1,M+1);

% randi
   =[1,2,8,9,20,27,1;41,42,48,49,60,65,41;32,38,39,50,57,61,32;31,32,38,39,50,57,31;6

% randj
   =[7,17,23,33,31,25,7;22,28,29,40,47,33,22;47,53,63,61,55,52,47;68,69,80,87,91,75,6

% % randi
   =[1,2,8,9,20,27,31,15,1;31,32,38,39,50,57,61,45,31;1,2,8,9,20,27,31,15,1;61,62,68,

% % randj
   =[7,17,23,33,31,25,22,10,7;37,47,53,63,61,55,52,40,37;37,47,53,63,61,55,52,40,37;3

% figure
% latlim = [40 60];
% lonlim = [-10 20];
% axesm('mercator','MapLatLimit',latlim,'MapLonLimit',lonlim,'Origin
   ',[0,0],'MeridianLabel','on','MLabelLocation
   ',[-10,-5,0,5,10,15,20],'ParallelLabel','on','PLabelLocation
   ',[40,45,50,55,60])

% m=gcm;
% latlim=m.maplatlimit;
% lonlim=m.maplonlimit;
% BoundingBox=[lonlim(1) latlim(1); lonlim(2) latlim(2)]
% land = shaperead('landareas', 'UseGeoCoords', true);
% geoshow(land, 'FaceColor', [0.5 0.7 0.5]);
% tightmap
% hold on
xv=zeros(1);
yv=zeros(1);

for k=65/5:-1:1
%     for z=1:n
%         for t=1:length(randi(z,:))
```

```
% xv(z,t) =(lambda(randi(z,t),randj(z,t))+Wu(randi(z,t),randj(z,t))/Rt
    *5*60*k)'; %Hay que arreglar esto
% yv(z,t) =(phi(randi(z,t),randj(z,t))+Wu(randi(z,t),randj(z,t))/Rt
    *5*60*k)';
%
%         end
storm_TFG(k);
[n(k),A]=size(storm_TFG{k});
n(k);
for z=1:n(k)

lengthstorm(z)=length(storm_TFG{k}(z).Lat);
[z,lengthstorm(z)];
xv(z,1:lengthstorm(z))=storm_TFG{k}(z).Lon'*pi/180;
yv(z,1:lengthstorm(z))=storm_TFG{k}(z).Lat'*pi/180;
% if k==1
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'b')

% hold on
% elseif k==3
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')

% hold on
% elseif k==5
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')

% hold on
% elseif k==7
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')

% hold on
% elseif k==9
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')

% hold on
% elseif k==11
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')

% hold on
% elseif k==13
% plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')

% hold on
% end
for i=1:N+1
    xq=lambda(i,:);
    yq=phi(i,:);
    [in,on] = inpolygon(xq,yq,xv(z,1:lengthstorm(z)),yv(z,1:lengthstorm(
        z)));
    inmat(i,:)=in;
```

```
%   plot(xv,yv) % polygon
% % axis equal
% % hold on
% % plot(xq(in),yq(in),'r+') % points inside
% % plot(xq(~in),yq(~in),'bo') % points outside
% % hold on

for j=1:M+1
     if inmat(i,j)==1&&tmat(i,j)>=tiemposinterpol(k)&&tmat(i,j)<
        tiemposinterpol(k+1)
          tormenta(i,j)=1; %Hay una tormenta en ese punto cuando el avi
              ón llegue
       else
          tormenta(i,j)=0; %Dicha tormenta no está
       end
end
end
size(tormenta);
poltormenta=poltormenta+tormenta; %total de puntos afectados en un
   frame / intervalo de tiempo afectados por la tormenta
end
%     pause(0.05)
%     hold off
frametormenta=frametormenta|poltormenta; %total de puntos afectados
   para todos los frames temporales.
end
% hold on
% plotm(phi0*180/pi,lambda0*180/pi,'k*')
% hold on
% plotm(phiD*180/pi,lambdaD*180/pi,'k*')
% L = linspace(0,2*pi,9);
frametormenta;
%xv2=zeros(n,length(randi(1,:)))
%yv2=zeros(n,length(randi(1,:)))
%numero de polígonos
%ch=zeros(n,length(randi(1,:)))
%matvert=zeros(n,length(randi(1,:)))
%match=zeros(n,length(randi(1,:)))
%contch=0
contvert=1;
z=1;
% figure
% for k=1:65/5
%     [n(k),A]=size(storm_TFG{k});
%
% for z=1:n(k)
%     lengthstorm(z)=length(storm_TFG{k}(z).Lat);
%
% xv(z,1:lengthstorm(z))=storm_TFG{k}(z).Lon'*pi/180;
% yv(z,1:lengthstorm(z))=storm_TFG{k}(z).Lat'*pi/180;
```

```
% plot(xv(z,1:lengthstorm(z))*180/pi,yv(z,1:lengthstorm(z))*180/pi,'b')
% xlabel('\lambda (º)')
% ylabel('\phi (º)')
% hold on
% end
% end
% plot(P0(1)*180/pi,P0(2)*180/pi,'r*')
% hold on
% plot(PD(1)*180/pi,PD(2)*180/pi,'r*')
% hold on
%Una vez definida la matriz frametormenta con ceros y unos hay que
   definir
%los nuevos polígonos con dichos unos, a esos polígonos son a los que
   habrá
%que hacerle la envolvente convexa.
[row,col] = find(frametormenta);
length(row);
length(col);
vecinosi=zeros(1,1);
vecinosj=zeros(1,1);
vecinosi(1,1)=row(1);
vecinosj(1,1)=col(1);
contador=zeros(1,1);
fila=1;
cont=1;
k=1;
contador(fila,:)=2;
for j=1:length(row)-1
 r=row(j);
 c=col(j);
   if j>=2
      k=j-1;
      while k>0
           if row(k)==r&&col(k)==c-1;
              a=find(vecinosi==r);
              b=find(vecinosj==c-1);
              h=1;
              while h<=length(a)
                  d=find(a(h)==b(:));
                  if length(d)==1
                      h=length(a)+1;
                  else
                      h=h+1;
                  end
              end
               [fila,columna]=ind2sub(size(vecinosi),b(d));
               vecinosi(fila,contador(fila,:))=row(j);
               vecinosj(fila,contador(fila,:))=col(j);
               contador(fila,:)=contador(fila,:)+1;
               k=-5;
```

```
elseif row(k)==r&&col(k)==c+1
    a=find(vecinosi==r);
   b=find(vecinosj==c+1);
   h=1;
   while h<=length(a)
       d=find(a(h)==b(:));
       if length(d)==1
           h=length(a)+1;
       else
           h=h+1;
       end
   end
    [fila,columna]=ind2sub(size(vecinosi),b(d));
    vecinosi(fila,contador(fila,:))=row(j);
    vecinosj(fila,contador(fila,:))=col(j);
    contador(fila,:)=contador(fila,:)+1;
    k=-5;
elseif row(k)==r-1&&col(k)==c
    a=find(vecinosi==r-1);
   b=find(vecinosj==c);
   h=1;
   while h<=length(a)
       d=find(a(h)==b(:));
       if length(d)==1
           h=length(a)+1;
       else
           h=h+1;
       end
   end
    [fila,columna]=ind2sub(size(vecinosi),b(d));
    vecinosi(fila,contador(fila,:))=row(j);
    vecinosj(fila,contador(fila,:))=col(j);
    contador(fila,:)=contador(fila,:)+1;
    k=-5;
elseif row(k)==r+1&&col(k)==c
    a=find(vecinosi==r+1);
   b=find(vecinosj==c);
   h=1;
   while h<=length(a)
       d=find(a(h)==b(:));
       if length(d)==1
           h=length(a)+1;
       else
           h=h+1;
       end
   end
    [fila,columna]=ind2sub(size(vecinosi),b(d));
    vecinosi(fila,contador(fila,:))=row(j);
    vecinosj(fila,contador(fila,:))=col(j);
    contador(fila,:)=contador(fila,:)+1;
```

```
 k=-5;
 elseif row(k)==r+1&&col(k)==c-1
 a=find(vecinosi==r+1);
b=find(vecinosj==c-1);
h=1;
while h<=length(a)
    d=find(a(h)==b(:));
    if length(d)==1
        h=length(a)+1;
    else
        h=h+1;
    end
end
 [fila,columna]=ind2sub(size(vecinosi),b(d));
 vecinosi(fila,contador(fila,:))=row(j);
 vecinosj(fila,contador(fila,:))=col(j);
 contador(fila,:)=contador(fila,:)+1;
 k=-5;
 elseif row(k)==r+1&&col(k)==c+1
 a=find(vecinosi==r+1);
b=find(vecinosj==c+1);
h=1;
while h<=length(a)
    d=find(a(h)==b(:));
    if length(d)==1
        h=length(a)+1;
    else
        h=h+1;
    end
end
 [fila,columna]=ind2sub(size(vecinosi),b(d));
 vecinosi(fila,contador(fila,:))=row(j);
 vecinosj(fila,contador(fila,:))=col(j);
 contador(fila,:)=contador(fila,:)+1;
 k=-5;
 elseif row(k)==r-1&&col(k)==c-1
 a=find(vecinosi==r-1);
b=find(vecinosj==c-1);
h=1;
while h<=length(a)
    d=find(a(h)==b(:));
    if length(d)==1
        h=length(a)+1;
    else
        h=h+1;
    end
end
 [fila,columna]=ind2sub(size(vecinosi),b(d));
 vecinosi(fila,contador(fila,:))=row(j);
 vecinosj(fila,contador(fila,:))=col(j);
```

```
                contador(fila,:)=contador(fila,:)+1;
                k=-5;
                elseif row(k)==r-1&&col(k)==c+1
                a=find(vecinosi==r-1);
               b=find(vecinosj==c+1);
               h=1;
               while h<=length(a)
                   d=find(a(h)==b(:));
                   if length(d)==1
                       h=length(a)+1;
                   else
                       h=h+1;
                   end
               end
                [fila,columna]=ind2sub(size(vecinosi),b(d));
                vecinosi(fila,contador(fila,:))=row(j);
                vecinosj(fila,contador(fila,:))=col(j);
                contador(fila,:)=contador(fila,:)+1;
                k=-5;
             end
          k=k-1;

       end
        if k==0
        cont=cont+1;
        fila=cont;
        vecinosi(fila,1)=row(j);
        vecinosj(fila,1)=col(j);
        contador(fila,:)=2;
         end

    end
end
vecinosi;
vecinosj;
contador(:,1)=contador(:,1)-1;
contador;

[F,C]=size(vecinosi);
cuent=1;
i=0;
while i<=F-1
    i=i+1;
    j=1;
    while j<=contador(i,:)
        k=i+1;
        while k<=F
            [i,k,F];
            l=1;
            while l<=contador(k,:)
```

```
            [l,contador(k,:)];
            if vecinosi(i,j)==vecinosi(k,l)&&vecinosj(i,j)==vecinosj(
                k,l)+1||vecinosi(i,j)==vecinosi(k,l)&&vecinosj(i,j)==
                vecinosj(k,l)-1||vecinosi(i,j)==vecinosi(k,l)+1&&
                vecinosj(i,j)==vecinosj(k,l)||vecinosi(i,j)==vecinosi(
                k,l)-1&&vecinosj(i,j)==vecinosj(k,l)...
               ||vecinosi(i,j)==vecinosi(k,l)-1&&vecinosj(i,j)==
                   vecinosj(k,l)+1||vecinosi(i,j)==vecinosi(k,l)-1&&
                   vecinosj(i,j)==vecinosj(k,l)-1||vecinosi(i,j)==
                   vecinosi(k,l)+1&&vecinosj(i,j)==vecinosj(k,l)-1||
                   vecinosi(i,j)==vecinosi(k,l)+1&&vecinosj(i,j)==
                   vecinosj(k,l)+1
               matcambios(cuent,:)=[vecinosi(i,j),vecinosj(i,j),
                   vecinosi(k,l),vecinosj(k,l)];
               cuent=cuent+1;
               vecinosi(i,contador(i,:)+1:contador(i,:)+contador(k,:))
                   =vecinosi(k,1:contador(k,:));
               vecinosj(i,contador(i,:)+1:contador(i,:)+contador(k,:))
                   =vecinosj(k,1:contador(k,:));
               contador(i,:)=contador(i,:)+contador(k,:);
               vecinosi(k,:)=[];
               vecinosj(k,:)=[];
               contador(k,:)=[];
               F=F-1; l=0; %%%%%%%%%%%%%%%%%%%%%%
            end
            if k>F
                break
            end
            l=l+1;
        end
        k=k+1;
    end
    j=j+1;
  end
end
 vecinosi;
 vecinosj;
 xv=zeros(1);
yv=zeros(1);
cont=zeros(1);
for i=1:F
  cont(i,:)=1;
  for j=1:(contador(i,:))
    [i,j];
    xv(i,cont(i,:):cont(i,:)+3)=[lambda(vecinosi(i,j),vecinosj(i,j))
        -0.05*pi/180,lambda(vecinosi(i,j),vecinosj(i,j))-0.05*pi/180,
        lambda(vecinosi(i,j),vecinosj(i,j))+0.05*pi/180,lambda(vecinosi(i
        ,j),vecinosj(i,j))+0.05*pi/180];
```

```
    yv(i,cont(i,:):cont(i,:)+3)=[phi(vecinosi(i,j),vecinosj(i,j))-0.05*
        pi/180,phi(vecinosi(i,j),vecinosj(i,j))+0.05*pi/180,phi(vecinosi(
        i,j),vecinosj(i,j))+0.05*pi/180,phi(vecinosi(i,j),vecinosj(i,j))
        -0.05*pi/180];
%    plot([xv(i,cont(i,:):cont(i,:)+3)*180/pi,xv(i,cont(i,:))*180/pi],[
    yv(i,cont(i,:):cont(i,:)+3)*180/pi,yv(i,cont(i,:))*180/pi],'r')
%    hold on
    cont(i,:)=cont(i,:)+4;
    end
end

xv;
n=F;
[F,C]=size(xv);
cont;
for z=1:n
    z
    i=1
   while i<=cont(z,:)-2
       j=i+1;
         while j<=cont(z,:)-1
             if xv(z,i)==xv(z,j)&&yv(z,i)==yv(z,j)
                 xv(z,j:length(xv(z,:))-1)=xv(z,j+1:length(xv(z,:)));
                 yv(z,j:length(xv(z,:))-1)=yv(z,j+1:length(xv(z,:)));
                 cont(z,:)=cont(z,:)-1;
             end
             j=j+1;

         end
         i=i+1;
   end
end
ch=zeros(1,1);
xv2=zeros(1,1);
yv2=zeros(1,1);
chindice=0;
match=zeros(1,1);

%hacemos cambio de variable con las variables en el mundo x,y
x=lambda;
y=phi

for z=1:n
    for j=1:cont(z,:)-1
        yv(z,j)=0.5*log((1+sin(yv(z,j)))/(1-sin(yv(z,j))));
    end
end
phi;

for i=1:N+1
```

```
    for j=1:M+1
        phi(i,j)=0.5*log((1+sin(phi(i,j)))/(1-sin(phi(i,j))));
    end
end
nodoaux0(2)=0.5*log((1+sin(phi0))/(1-sin(phi0)));
phi0=0.5*log((1+sin(phi0))/(1-sin(phi0)));
phiD=0.5*log((1+sin(phiD))/(1-sin(phiD)));
nodoaux0(1)=lambda0;
P0=[lambda0,phi0];
PD=[lambdaD,phiD];
nodoaux0
P0
%figure
% xlabel('\lambda (º)')
% ylabel('\phi (º)')
% for z=1:n
%    plot(xv(z,1:cont(z)-1)*180/pi,yv(z,1:cont(z)-1)*180/pi)
%    hold on
%
% end
% figure
for i=1:F
    cont(i,:)=1;
    for j=1:(contador(i,:))
        [i,j];
        xv(i,cont(i,:):cont(i,:)+3)=[lambda(vecinosi(i,j),vecinosj(i,j))
            -0.05*pi/180,lambda(vecinosi(i,j),vecinosj(i,j))-0.05*pi/180,
            lambda(vecinosi(i,j),vecinosj(i,j))+0.05*pi/180,lambda(
            vecinosi(i,j),vecinosj(i,j))+0.05*pi/180];
        yv(i,cont(i,:):cont(i,:)+3)=[phi(vecinosi(i,j),vecinosj(i,j))
            -0.05*pi/180,phi(vecinosi(i,j),vecinosj(i,j))+0.05*pi/180,phi
            (vecinosi(i,j),vecinosj(i,j))+0.05*pi/180,phi(vecinosi(i,j),
            vecinosj(i,j))-0.05*pi/180];
%        plot([xv(i,cont(i,:):cont(i,:)+3)*180/pi,xv(i,cont(i,:))*180/pi
    ],[atan(sinh(yv(i,cont(i,:))))*180/pi,atan(sinh(yv(i,cont(i,:)+1)))
    *180/pi,atan(sinh(yv(i,cont(i,:)+2)))*180/pi,atan(sinh(yv(i,cont(i
    ,:)+3)))*180/pi,atan(sinh(yv(i,cont(i,:))))*180/pi],'r')
%       xlabel('\lambda (º)')
%        ylabel('\phi (º)')
%    hold on
       cont(i,:)=cont(i,:)+4;
    end
end
      for z=1:n
% for t=1:length(randi(z,:))
% xv(z,t) =(lambda(randi(z,t),randj(z,t))+Wu(randi(z,t),randj(z,t))/Rt
    *5*60*k)';%Hay que arreglar esto
% yv(z,t) =(phi(randi(z,t),randj(z,t))+Wu(randi(z,t),randj(z,t))/Rt
    *5*60*k)';
% % matvert(z,t)=contvert;
```

```
% % %vert(t+(z-1)*length(randi))=contvert; en el caso de que los polí
    gonos
% % %tengan distintas longitudes, habría que poner un si elto~=0 en el
    calculo
% % %de matinc
% % contvert=contvert+1;
%  end
xv;
yv;
chv=convhull(xv(z,1:(cont(z,:)-1)),yv(z,1:(cont(z,:)-1)))';
ch(z,1:length(chv))=chv(:);
chindice(z)=length(chv);
%match(z,1:chindice(z))=[contch+1+1,chindice(z)+contch-1+1:-1:contch
    +2+1,contch+1+1]
%contch=chindice(z)-1+contch
xv1=xv(z,chv(:));
yv1=yv(z,chv(:));
xv2(z,1:length(chv))=xv1;
yv2(z,1:length(chv))=yv1;
%Filtro para ver si los puntos de inicio o destino estan dentro de una
%convex hull:
%[in1,on1]=inpolygon(lambda0,phi0,xv(z,:),yv(z,:));
%[in2,on2]=inpolygon(lambdaD,phiD,xv(z,:),yv(z,:));
% [in1_2,on1_2]=inpolygon(lambda0,phi0,xv2(z,1:chindice(z)),yv2(z,1:
    chindice(z)));
% [in2_2,on2_2]=inpolygon(lambdaD,phiD,xv2(z,1:chindice(z)),yv2(z,1:
    chindice(z)));
% if in1_2==1||in2_2==1
%     puntodentro=1%Punto origen dentro de un polígono
%  end

% plot(xv2(z,1:chindice(z)).*180/pi,atan(sinh(yv2(z,1:chindice(z))))
    .*180/pi,'b')
% hold on
mlength(z,:)=cont(z,:)-1;

%    disp('Punto origen dentro de un polígono')
%    break
% elseif in1_2==1&&in1==0
%    disp('punto origen dentro de una convex hull')
%    break
% end
% if in2==1 %Punto destino
%    disp('Punto destino dentro de un polígono')
%    break
% elseif in2_2==1&&in2==0
%    disp('punto destino dentro de una convex hull')
%    break
% end
      end
```

```
% if puntoordentro==1
%    break
% end
% figure
% for z=1:n
%    plot(xu(z,1:cont(z)-1)*180/pi,yu(z,1:cont(z)-1)*180/pi)
%    hold on
%
% end
 % figure
% latlim = [40 50];
% lonlim = [0 10];
% axesm('Mercator','MapLatLimit',latlim,'MapLonLimit',lonlim)
% land = shaperead('landareas', 'UseGeoCoords', true);
% geoshow(land, 'FaceColor', [0.5 0.7 0.5]);

% for z=1:n
%    plot(xu2(z,1:chindice(z))*180/pi,yu2(z,1:chindice(z))*180/pi,'k')
%    hold on
% end
%  plot(P0(1)*180/pi,atan(sinh(P0(2)))*180/pi,'r*')
%   hold on
%   plot(PD(1)*180/pi,atan(sinh(PD(2)))*180/pi,'r*')
% pause (0.01)
% figure
% for z=1:n
% plot(xu2(z,1:chindice(z)),yu2(z,1:chindice(z)),'r')
% hold on
% end
z=1;
while z<=n
     if z>=2
[xv,yv,ch,chindice,xu2,yu2,z,n,mlength]=fusionpoligonos(xu2,yu2,z,xv,yv,
   ch,chindice,n,mlength);
     end
z=z+1;
end


% latlim = [40 50];
% lonlim = [0 10];
% axesm('Mercator','MapLatLimit',latlim,'MapLonLimit',lonlim)
% land = shaperead('landareas', 'UseGeoCoords', true);
% geoshow(land, 'FaceColor', [0.5 0.7 0.5]);
% hold on

% for z=1:n
%    plot(xu2(z,1:chindice(z))*180/pi,yu2(z,1:chindice(z))*180/pi)
%    hold on
% end
```

```
 for z=1:n
      [in1_2,on1_2]=inpolygon(lambda0,phi0,xv2(z,1:chindice(z)),yv2(z,1:
         chindice(z)));
[in2_2,on2_2]=inpolygon(lambdaD,phiD,xv2(z,1:chindice(z)),yv2(z,1:
    chindice(z)));
if in1_2==1
     puntoordentro=1;
     zorconf=z;
end%Punto origen dentro de un polígono
if in2_2==1
    puntodesdentro=1;
    zdesconf=z
end



 end
 figure

 for z=1:n
    plot(xv2(z,1:chindice(z)).*180/pi,atan(sinh(yv2(z,1:chindice(z))))
        .*180/pi,'b')
     xlabel('\lambda (º)')
     ylabel('\phi (º)')
 hold on
 end
  plot(P0(1)*180/pi,atan(sinh(P0(2)))*180/pi,'r*')
   hold on
   plot(PD(1)*180/pi,atan(sinh(PD(2)))*180/pi,'r*')
   hold on
 puntoordentro
 puntodesdentro
 zdesconf
 zorconf
if puntoordentro==1&&zdesconf==zorconf
    puntoordentro=0;
    puntodesdentro=0;
else
 if puntoordentro==1
     rint0=zeros(1);
     courseint0=zeros(1);
    for j=1:chindice(zorconf)
        yv2(zorconf,j)=atan(sinh(yv2(zorconf,j)));
    end
phi0=atan(sinh(phi0));
phiD=atan(sinh(phiD));
PD=[lambdaD,phiD];
P0=[lambda0,phi0];
     for l=1:chindice(zorconf)
        P0=[lambda0,phi0];
```

```
    [rint0(l),courseint0(l)]=loxodromicequations(phi0,lambda0,yv2(
       zorconf,l),xv2(zorconf,l),Rt,Theta,vrefISA,0,0);

end
[rord,ind]=sort(rint0);
courseint=courseint0(ind(:))
bucle
if bucle==0
       [r0,course1]=loxodromicequations(phi0,lambda0,phiD,lambdaD,Rt
          ,Theta,vrefISA,0,0);
        for l=1:chindice(zorconf)
            course0=courseint(l);
            if abs(course1)<=3*pi/4&&course1+pi/4>=course0&&course1-
               pi/4<=course0||course1>3*pi/4&&course0<0&&course
               0<=-(3*pi/4+pi-course1)||course1>3*pi/4&&course0>0&&
               course1-pi/4<=course0&&pi-course0<=pi-course1||course
               1<-3*pi/4&course0<0&&abs(course1)-pi/4<=abs(course0)
               &&pi-abs(course0)<=pi-abs(course1)||course1<-3*pi/4&&
               course0>0&&-course0<=-(3*pi/4+pi-abs(course1))
        nodoescape0=[xv2(zorconf,ind(l)),yv2(zorconf,ind(l))];
       [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(P
          0(1)+nodoescape0(1))/2,(P0(2)+nodoescape0(2))/2,Wu,
          pasomallalambda,pasomallaphi,M,N);
       [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(P
          0(1)+nodoescape0(1))/2,(P0(2)+nodoescape0(2))/2,Wv,
          pasomallalambda,pasomallaphi,M,N);
       [y,t]=ode45(@(y,t)loxodromicequationsint(P0(2),P0(1),Rt,
          WuintD,WvintD,Theta,vrefISA,courseint(l),y,t),[0,rord(l)
          ],0);
       t0=t(length(t));

       for j=1:chindice(zorconf)
        yv2(zorconf,j)=0.5*log((1+sin(yv2(zorconf,j)))/(1-sin(yv2(
            zorconf,j))));
       end
       phi0=0.5*log((1+sin(phi0))/(1-sin(phi0)));
       nodoescape0(2)=0.5*log((1+sin(nodoescape0(2)))/(1-sin(
          nodoescape0(2))));
       phiD=0.5*log((1+sin(phiD))/(1-sin(phiD)));
       P0=[lambda0,phi0];
       PD=[lambdaD,phiD];
       rec0=rord(l);
       lescape0=ind(l);


       nodoaux0=P0(:)';
       P0=nodoescape0(:)';
       lambda0=P0(1);
       phi0=P0(2);
       plot(P0(1)*180/pi,atan(sinh(P0(2)))*180/pi,'r*')
```

```
            break
                end
             end
    else
    for l=1:chindice(zorconf)
            course0=courseint(l);
            if sign(course1)==sign(course0)
                if abs(course1-courseint(l))<=45*pi/180
            nodoescape0=[xv2(zorconf,ind(l)),yv2(zorconf,ind(l))]
            [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(P
                0(1)+nodoescape0(1))/2,(P0(2)+nodoescape0(2))/2,Wu,
                pasomallalambda,pasomallaphi,M,N);
            [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(P
                0(1)+nodoescape0(1))/2,(P0(2)+nodoescape0(2))/2,Wv,
                pasomallalambda,pasomallaphi,M,N);
            [y,t]=ode45(@(y,t)loxodromicequationsint(P0(2),P0(1),Rt,
                WuintD,WvintD,Theta,vrefISA,courseint(l),y,t),[0,rord(l)
                ],0);
            t0=t(length(t));
            course0=courseint(l);
            for j=1:chindice(zorconf)
             yv2(zorconf,j)=0.5*log((1+sin(yv2(zorconf,j)))/(1-sin(yv2(
                 zorconf,j))));
            end
            phi0=0.5*log((1+sin(phi0))/(1-sin(phi0)));
            nodoescape0(2)=0.5*log((1+sin(nodoescape0(2)))/(1-sin(
                nodoescape0(2))));
            phiD=0.5*log((1+sin(phiD))/(1-sin(phiD)));
            P0=[lambda0,phi0];
            PD=[lambdaD,phiD];
            rec0=rord(l);
            lescape0=ind(l);

            nodoaux0=P0(:)';
            P0=nodoescape0(:)';
            lambda0=P0(1);
            phi0=P0(2);
            plot(P0(1)*180/pi,atan(sinh(P0(2)))*180/pi,'r*')
            break
                end
            else
                if abs(course0)>pi/2&&(pi-abs(course0))+(pi-abs(course1))
                    <=pi/4||abs(course0)<pi/2&&abs(course0)+abs(course1)<=
                    pi/4
            nodoescape0=[xv2(zorconf,ind(l)),yv2(zorconf,ind(l))];
            [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(P
                0(1)+nodoescape0(1))/2,(P0(2)+nodoescape0(2))/2,Wu,
                pasomallalambda,pasomallaphi,M,N);
```

```
            [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(P
               0(1)+nodoescape0(1))/2,(P0(2)+nodoescape0(2))/2,Wv,
               pasomallalambda,pasomallaphi,M,N);
            [y,t]=ode45(@(y,t)loxodromicequationsint(P0(2),P0(1),Rt,
               WuintD,WvintD,Theta,vrefISA,courseint(l),y,t),[0,rord(l)
               ],0);
            t0=t(length(t));
            course0=courseint(l);
            for j=1:chindice(zorconf);
             yv2(zorconf,j)=0.5*log((1+sin(yv2(zorconf,j)))/(1-sin(yv2(
                 zorconf,j))));
            end
            phi0=0.5*log((1+sin(phi0))/(1-sin(phi0)));
            nodoescape0(2)=0.5*log((1+sin(nodoescape0(2)))/(1-sin(
               nodoescape0(2))));
            phiD=0.5*log((1+sin(phiD))/(1-sin(phiD)));
            P0=[lambda0,phi0];
            PD=[lambdaD,phiD];
            rec0=rord(l);
            lescape0=ind(l);

            nodoaux0=P0(:)';
            P0=nodoescape0(:)';
            lambda0=P0(1);
            phi0=P0(2);
            plot(P0(1)*180/pi,atan(sinh(P0(2)))*180/pi,'r*')
            hold on
            break
        end
    end
    end
    end
 end
 puntoordentro
 nodoaux0
 P0

if puntodesdentro==1
    rintD=zeros(1);
    courseintD=zeros(1);
    courseint=zeros(1);
    rordD=zeros(1);
    indD=zeros(1);
     for j=1:chindice(zdesconf)
        yv2(zdesconf,j)=atan(sinh(yv2(zdesconf,j)));
    end
phi0=atan(sinh(phi0));
phiD=atan(sinh(phiD));
PD=[lambdaD,phiD];
P0=[lambda0,phi0];
```

```
     for l=1:chindice(zdesconf)
        PD=[lambdaD,phiD];
        [rintD(l),courseintD(l)]=loxodromicequations(phiD,lambdaD,yv2(
            zdesconf,l),xv2(zdesconf,l),Rt,Theta,vrefISA,0,0);
     end
    [rordD,indD]=sort(rintD);
    courseint=courseintD(indD(:));
%     if bucle==0
    [r0,courseD]=loxodromicequations(phiD,lambdaD,phi0,lambda0,Rt,Theta,
        vrefISA,0,0);
%     else
%         if courseD>=0
%             courseD=courseD-pi;
%         else
%             courseD=courseD+pi;
%         end
%     end
    l=1;
                course0=courseint(l);
                %if abs(courseD)<=2*pi/3&&courseD+pi/3>=course0&&courseD
                    -pi/3<=course0||courseD>2*pi/3&&course0<0&&course
                    0<=-(2*pi/3+pi-courseD)||courseD>2*pi/2&&course0>0&&
                    courseD-pi/3<=course0&&pi-course0<=pi-courseD||
                    courseD<-2*pi/3&course0<0&&abs(courseD)-pi/3<=abs(
                    course0)&&pi-abs(course0)<=pi-abs(courseD)||courseD
                    <-2*pi/3&&course0>0&&-course0<=-(2*pi/3+pi-abs(
                    courseD))

             nodoescapeD=[xv2(zdesconf,indD(l)),yv2(zdesconf,indD(l))];
            [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(PD
                (1)+nodoescapeD(1))/2,(PD(2)+nodoescapeD(2))/2,Wu,
                pasomallalambda,pasomallaphi,M,N);
            [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(PD
                (1)+nodoescapeD(1))/2,(PD(2)+nodoescapeD(2))/2,Wv,
                pasomallalambda,pasomallaphi,M,N);
            [y,t]=ode45(@(y,t)loxodromicequationsint(nodoescapeD(2),
                nodoescapeD(1),Rt,WuintD,WvintD,Theta,vrefISA,courseint(l)
                ,y,t),[0,rordD(l)],0);
            tD=t(length(t));
            for j=1:chindice(zdesconf)
             yv2(zdesconf,j)=0.5*log((1+sin(yv2(zdesconf,j)))/(1-sin(yv2(
                 zdesconf,j))));
            end
            phi0=0.5*log((1+sin(phi0))/(1-sin(phi0)));
            nodoescapeD(2)=0.5*log((1+sin(nodoescapeD(2)))/(1-sin(
                nodoescapeD(2))));
            phiD=0.5*log((1+sin(phiD))/(1-sin(phiD)));
            P0=[lambda0,phi0];
            PD=[lambdaD,phiD];
            recD=rordD(l);
```

```
            courseD=course0
            lescapeD=indD(l);
            nodoauxD=PD(:)';
            PD=nodoescapeD(:)';
            lambdaD=PD(1);
            phiD=PD(2);
            plot(PD(1)*180/pi,atan(sinh(PD(2)))*180/pi,'r*')
            hold on
           % break
                % end
    %end
             end
end


% course0;
%  course1;
 pause (0.01)
    contch=0;
% if zorconf==zdesconf
%     P0=nodoaux0;
% lambda0=P0(1);
% phi0=P0(2);
% PD=nodoauxD;
% lambdaD=PD(1);
% phiD=PD(2)
% end
% [in1,on1]=inpolygon(lambda0,phi0,xv(z,:),yv(z,:));
% [in2,on2]=inpolygon(lambdaD,phiD,xv(z,:),yv(z,:));

 for z=1:n
  match(z,1:chindice(z))=[contch+1+1,chindice(z)+contch-1+1:-1:contch
      +2+1,contch+1+1];
contch=chindice(z)-1+contch;
 end
 contch=contch+2;%contch corregido, da un número más que el maximo de la
      matriz, para que tenga en cuenta el destino
%Calculamos las tangentes del origen y destino a los polígonos
  if puntoordentro==1
      lescape0
 contchconf0=match(zorconf,lescape0);
  else
  contchconf0=1;
  end
if puntodesdentro==1
 contchconfD=match(zdesconf,lescapeD);
else
   contchconfD=contch;
 end
% for z=1:n
```

```
% plot(xv2(z,1:chindice(z)),yv2(z,1:chindice(z)),'k')
% hold on
% end
% plot(xv2(n,1:chindice(n)),yv2(n,1:chindice(n)),'k')
xtan0=zeros(1);
ytan0=zeros(1);
zvert0poly=zeros(1);
xtanD=zeros(1);
ytanD=zeros(1);
zvertDpoly=zeros(1);


    for z=1:n
        contador=1;
        cont=1;

for l=1:(chindice(z)-1)
   % if xv2(z,l)~=P0(1)&&yv2(z,l)~=P0(2)
[tangente]=tangpointtopolygon(P0,xv2(z,:),yv2(z,:),l,chindice(z));
   % end
  % if xv2(z,l)~=PD(1)&&yv2(z,l)~=PD(2)
[tang]=tangpointtopolygon(PD,xv2(z,:),yv2(z,:),l,chindice(z));
   % end
if tangente==1
    xtan0(z,cont)=xv2(z,l+1);
    ytan0(z,cont)=yv2(z,l+1);
    zvert0poly(z,cont)=match(z,l+1); %matriz que almacena los vertices
        de los polígonos que toca.
    cont=cont+1;
end
if tang==1
    xtanD(z,contador)=xv2(z,l+1);
    ytanD(z,contador)=yv2(z,l+1);
    zvertDpoly(z,contador)=match(z,l+1);
    contador=contador+1;
end
end
    end
    zvert0poly;
    zvertDpoly;
    if puntoordentro==1
        [K,J]=size(xtan0);
        if J>2
            xtan0(:,3:J)=[];
            zvert0poly(:,3:J)=[];
            ytan0(:,3:J)=[];
        end
       zvert0poly(zorconf,:)=0;
    end
     if puntodesdentro==1
```

```
        [K,J]=size(xtanD);
        if J>2
            xtanD(:,3:J)=[];
            zvertDpoly(:,3:J)=[];
            ytanD(:,3:J)=[];
        end
       zvertDpoly(zdesconf,:)=0;
    end
  zvert0poly;
  zvertDpoly;
%        PD
%     xtanD
%     ytanD
%     ytan0
%     xtan0
%    figure
%
% for z=1:n
%     for l=1:length(xtan0(z,:))
%         if zvert0poly(z,l)~=0
%             plot([lambda0*180/pi,xtan0(z,l)*180/pi],[atan(sinh(phi0))
    *180/pi,atan(sinh(ytan0(z,l)))*180/pi])
%           xlabel('\lambda (º)')
%           ylabel('\phi (º)')
%         hold on
%         end
%         plot(xv2(z,1:chindice(z)).*180/pi,atan(sinh(yv2(z,1:chindice(z)
    ))).*180/pi,'k')
%         hold on
%     end
% end
% figure
%
% for z=1:n
%     for l=1:length(xtanD(z,:))
%         if zvertDpoly(z,l)~=0
%             plot([lambdaD*180/pi,xtanD(z,l)*180/pi],[atan(sinh(phiD))
    *180/pi,atan(sinh(ytanD(z,l)))*180/pi])
% xlabel('\lambda (º)')
% ylabel('\phi (º)')
%             hold on
%         end
%         plot(xv2(z,1:chindice(z)).*180/pi,atan(sinh(yv2(z,1:chindice(z)
    ))).*180/pi,'k')
%         hold on
%     end
% end
% xtan0
% ytan0
% P0
```

```
[xtan0,ytan0,zvert0poly,Origen_destino]=cortetangentespuntopoligono(P0,
    xtan0,ytan0,xv2,yv2,n,zvert0poly,PD,chindice,zorconf,puntoordentro);
% xtan0
% ytan0
zvert0poly;
% figure
%
% for z=1:n
%     for l=1:length(xtan0(z,:))
%         if zvert0poly(z,l)~=0
%             plot([lambda0*180/pi,xtan0(z,l)*180/pi],[atan(sinh(phi0))
    *180/pi,atan(sinh(ytan0(z,l)))*180/pi])
% xlabel('\lambda (º)')
% ylabel('\phi (º)')
%             hold on
%         end
%         plot(xv2(z,1:chindice(z)).*180/pi,atan(sinh(yv2(z,1:chindice(z)
    ))).*180/pi,'k')
%         hold on
%     end
% end
% hold on
% if Origen_destino==1
%     plot([P0(1)*180/pi,PD(1)*180/pi],[atan(sinh(P0(2)))*180/pi,atan(
    sinh(PD(2)))*180/pi])
%     hold on
%     plot(P0(1)*180/pi,atan(sinh(P0(2)))*180/pi)
%     hold on
%     plot(PD(1)*180/pi,atan(sinh(PD(2)))*180/pi)
% end

[xtanD,ytanD,zvertDpoly,Origen_destino]=cortetangentespuntopoligono(PD,
    xtanD,ytanD,xv2,yv2,n,zvertDpoly,P0,chindice,zdesconf,puntodesdentro
    );
% Origen_destino
% xtanD
% ytanD
cont0=1;
contD=1;
% figure
% for z=1:n
%     for l=1:length(xtanD(z,:))
%         if zvertDpoly(z,l)~=0
%             plot([lambdaD*180/pi,xtanD(z,l)*180/pi],[atan(sinh(phiD))
    *180/pi,atan(sinh(ytanD(z,l)))*180/pi])
% xlabel('\lambda (º)')
% ylabel('\phi (º)')
%             hold on
%         end
```

```
%        plot(xv2(z,1:chindice(z)).*180/pi,atan(sinh(yv2(z,1:chindice(z)
    ))).*180/pi,'k')
%        hold on
%    end
% end

%De cara a la matriz de incidencia definimos con que nodo se
    corresponde
ch;
match;
contch;
zvert0poly;
zvertDpoly;
chindice;
vectanpoint0=zeros(1);
vectanpointD=zeros(1);
for z=1:n
    for l=1:length(zvert0poly(z,:))
        if zvert0poly(z,l)~=0
vectanpoint0(cont0)=zvert0poly(z,l);
cont0=cont0+1;
        end
    end
     for l=1:length(zvertDpoly(z,:))
        if zvertDpoly(z,l)~=0
vectanpointD(contD)=zvertDpoly(z,l);
contD=contD+1;
        end
    end
end

 vectanpoint0;
 vectanpointD;
[vectanpoint0]=sort(vectanpoint0);
[vectanpointD]=sort(vectanpointD);
%figure

% for z=1:n
%    for l=1:length(xtan0(z,:))
%        if zvert0poly(z,l)~=0
%            plot([lambda0,xtan0(z,l)],[phi0,ytan0(z,l)])
%            hold on
%        end
%        plot(xv2(z,1:chindice(z)),yv2(z,1:chindice(z)))
%        hold on
%    end
% end
% hold on
% if Origen_destino==1
%    plot([P0(1),PD(1)],[P0(2),PD(2)])
```

```
% end
% figure
% for z=1:n
%    for l=1:length(xtanD(z,:))
%        if zvertDpoly(z,l)~=0
%            plot([lambdaD,xtanD(z,l)],[phiD,ytanD(z,l)])
%            hold on
%        end
%        plot(xv2(z,1:chindice(z)),yv2(z,1:chindice(z)))
%        hold on
%    end
% end
% hold on
% if Origen_destino==1
%    plot([P0(1),PD(1)],[P0(2),PD(2)])
% end
   xv2;
   yv2;
   xv;
   yv;
   chindice;
cuentador=1;
mat1=zeros(n);
mat2=zeros(n);
mat3=zeros(n);
mat4=zeros(n);
% figure
for z=1:n
    for p=z+1:n
    cont=1;
    for l=2:chindice(z)
        [indz,indp]=tangpolygontopolygon(xv2,yv2,l,z,p,chindice) ;
        for t=1:length(indz)
            if indz(t)~=0
            indtanz(cuentador,cont)=indz(t); %matriz que recoge los
                indices de los puntos de tangencia del polígono original
                z
            indtanp(cuentador,cont)=indp(t);
            %matriz que recoge los indices de los puntos de tangencia del
                 polígono original z+1
            xtanz_1(cont)=xv2(z,indz(t));
            xtanz(cont)=xv2(p,indp(t));
            ytanz_1(cont)=yv2(z,indz(t));
            ytanz(cont)=yv2(p,indp(t));
            cont=cont+1;
            end
        end
    end
    mat1(z,p)=indtanz(cuentador,1);
    mat1(p,z)=indtanp(cuentador,1);
```

```
    mat2(z,p)=indtanz(cuentador,2);
    mat2(p,z)=indtanp(cuentador,2);
    mat3(z,p)=indtanz(cuentador,3);
    mat3(p,z)=indtanp(cuentador,3);
    mat4(z,p)=indtanz(cuentador,4);
    mat4(p,z)=indtanp(cuentador,4);
    cuentador=cuentador+1;
    xtanz_1;
    xtanz;
   ytanz_1;

% plot([xtanz_1(1)*180/pi,xtanz(1)*180/pi],[atan(sinh(ytanz_1(1)))*180/
    pi,atan(sinh(ytanz(1)))*180/pi])
%  xlabel('\lambda (º)')
% ylabel('\phi (º)')
% hold on
% plot([xtanz_1(2)*180/pi,xtanz(2)*180/pi],[atan(sinh(ytanz_1(2)))*180/
    pi,atan(sinh(ytanz(2)))*180/pi])
% hold on
% plot([xtanz_1(3)*180/pi,xtanz(3)*180/pi],[atan(sinh(ytanz_1(3)))*180/
    pi,atan(sinh(ytanz(3)))*180/pi])
% hold on
% plot([xtanz_1(4)*180/pi,xtanz(4)*180/pi],[atan(sinh(ytanz_1(4)))*180/
    pi,atan(sinh(ytanz(4)))*180/pi])
%
% hold on

   end
% plot(xv2(z,1:chindice(z)).*180/pi,atan(sinh(yv2(z,1:chindice(z))))
    .*180/pi,'k')
% hold on
end
mat1;
mat2;
mat3;
mat4;
%Está bien hasta aquí.
%
    ------------------------------------------------------------------%

% [mat1,mat2,mat3,mat4]=cortetangentepoligono(xv,yv,mat1,mat2,mat3,mat4,
    n)
[mattan1,mattan2,mattan3,mattan4]=cortetangenteentrepoligono(xv2,yv2,mat
    1,mat2,mat3,mat4,n,chindice);
%mattan son las matrices que van a recoger las tangentes que no cortan
mat1=abs(mat1-mattan1);
mat2=abs(mat2-mattan2);
mat3=abs(mat3-mattan3);
mat4=abs(mat4-mattan4);
% figure
```

```
% for z=1:n
%     for p=z+1:n
% if mat1(z,p)~=0
% plot([xv2(z,mat1(z,p)),xv2(p,mat1(p,z))],[yv2(z,mat1(z,p)),yv2(p,mat
    1(p,z))])
% hold on
% end
% if mat2(z,p)~=0
% plot([xv2(z,mat2(z,p)),xv2(p,mat2(p,z))],[yv2(z,mat2(z,p)),yv2(p,mat
    2(p,z))])
% hold on
% end
% if mat3(z,p)~=0
% plot([xv2(z,mat3(z,p)),xv2(p,mat3(p,z))],[yv2(z,mat3(z,p)),yv2(p,mat
    3(p,z))])
% hold on
% end
% if mat4(z,p)~=0
% plot([xv2(z,mat4(z,p)),xv2(p,mat4(p,z))],[yv2(z,mat4(z,p)),yv2(p,mat
    4(p,z))])
% hold on
% end
%     end
%    plot(xv2(z,1:chindice(z)),yv2(z,1:chindice(z)))
% % % hold on
% end
vectortan=zeros(1,1);
% vectortan2=zeros(1,1);
%  figure
% xlabel('\lambda (º)')
% ylabel('\phi (º)')
contador=1;
contador2=1;
for z=1:n
    for p=z+1:n
if mattan1(z,p)~=0
% plot([xv2(z,mattan1(z,p))*180/pi,xv2(p,mattan1(p,z))*180/pi],[atan(
    sinh(yv2(z,mattan1(z,p))))*180/pi,atan(sinh(yv2(p,mattan1(p,z))))
    *180/pi])
%  xlabel('\lambda (º)')
% ylabel('\phi (º)')
% hold on
vectortan(contador)=match(z,mattan1(z,p));
contador=contador+1;
vectortan2(contador2)=match(p,mattan1(p,z));
contador2=contador2+1;

%Así lo que hacemos es meter en un vector de longitud el numero de
    aristas
%del grafo de visibilidad, y que en su interior contenga z y el indice
```

```
end
if mattan2(z,p)~=0
% plot([xv2(z,mattan2(z,p))*180/pi,xv2(p,mattan2(p,z))*180/pi],[atan(
    sinh(yv2(z,mattan2(z,p))))*180/pi,atan(sinh(yv2(p,mattan2(p,z))))
    *180/pi])
% xlabel('\lambda (º)')
% ylabel('\phi (º)')
% hold on
vectortan(contador)=match(z,mattan2(z,p));
contador=contador+1;
vectortan2(contador2)=match(p,mattan2(p,z));
contador2=contador2+1;
end
if mattan3(z,p)~=0
% plot([xv2(z,mattan3(z,p))*180/pi,xv2(p,mattan3(p,z))*180/pi],[atan(
    sinh(yv2(z,mattan3(z,p))))*180/pi,atan(sinh(yv2(p,mattan3(p,z))))
    *180/pi])
% xlabel('\lambda (º)')
% ylabel('\phi (º)')
% hold on
vectortan(contador)=match(z,mattan3(z,p));
contador=contador+1;
vectortan2(contador2)=match(p,mattan3(p,z));
contador2=contador2+1;
end
if mattan4(z,p)~=0
% plot([xv2(z,mattan4(z,p))*180/pi,xv2(p,mattan4(p,z))*180/pi],[atan(
    sinh(yv2(z,mattan4(z,p))))*180/pi,atan(sinh(yv2(p,mattan4(p,z))))
    *180/pi])
% xlabel('\lambda (º)')
% ylabel('\phi (º)')
% hold on
vectortan(contador)=match(z,mattan4(z,p));
contador=contador+1;
vectortan2(contador2)=match(p,mattan4(p,z));
contador2=contador2+1;
end
    end
%         plot(xv2(z,1:chindice(z)).*180/pi,atan(sinh(yv2(z,1:chindice(z
    ))))).*180/pi,'k')
% hold on
%         plot(lambda0,phi0,'k*')
%         hold on
%         plot(lambdaD,phiD,'k*')

end
% pause(0.01)
vectanorden=vectortan;
vectanorden2=vectortan2;
length(vectanorden);
```

```
length(vectanorden2);
% [vectanorden,indvec]=sort(vectortan);
% vectanorden2=vectortan2(indvec(:));
% for t=1:length(vectanorden)-1
%    for k=length(vectanorden):-1:t
%        if vectanorden(t)==vectanorden(k)
%            if vectanorden2(k)<vectanorden2(t)
%                aux=vectanorden2(k);
%                vectanorden2(k)=vectanorden2(t);
%                vectanorden2(t)=aux;
%            end
%        end
%    end
% end
% figure
contador=1;
contador2=1;
% for z=1:n
%    for p=z+1:n
% if mattan1(z,p)~=0
% plot([xv2(z,mattan1(z,p)),xv2(p,mattan1(p,z))],[yv2(z,mattan1(z,p)),
   yv2(p,mattan1(p,z))])
% hold on
% end
% if mattan2(z,p)~=0
% plot([xv2(z,mattan2(z,p)),xv2(p,mattan2(p,z))],[yv2(z,mattan2(z,p)),
   yv2(p,mattan2(p,z))])
% hold on
% end
% if mattan3(z,p)~=0
% plot([xv2(z,mattan3(z,p)),xv2(p,mattan3(p,z))],[yv2(z,mattan3(z,p)),
   yv2(p,mattan3(p,z))])
% hold on
% end
% if mattan4(z,p)~=0
% plot([xv2(z,mattan4(z,p)),xv2(p,mattan4(p,z))],[yv2(z,mattan4(z,p)),
   yv2(p,mattan4(p,z))])
% hold on
% end
%    end
% %    for l=1:length(xtan0(z,:))
% %        if xtan0(z,l)~=0
% %            plot([lambda0,xtan0(z,l)],[phi0,ytan0(z,l)])
% %            hold on
% %        end
% %        plot(xv2(z,1:chindice(z)),yv2(z,1:chindice(z)))
% %        hold on
% %    end
% %        for l=1:length(xtanD(z,:))
% %        if xtanD(z,l)~=0
```

```
% %           plot([lambdaD,xtanD(z,l)],[phiD,ytanD(z,l)])
% %           hold on
% %        end
%         plot(xv2(z,1:chindice(z)),yv2(z,1:chindice(z)),'k')
%         hold on
% %    end
% end

%Escribimos ahora las matrices de incidencia donde los vértices sean
    los
%diferentes nodos que son las diferentes filas

for z=1:n
    for j=1:chindice(z)
        yv2(z,j)=atan(sinh(yv2(z,j)));
    end
end
if puntoordentro==1
    nodoaux0(2)=atan(sinh(nodoaux0(2)));
    nodoescape0(2)=atan(sinh(nodoescape0(2)));
end
if puntodesdentro==1
    nodoauxD(2)=atan(sinh(nodoauxD(2)));
    nodoescapeD(2)=atan(sinh(nodoescapeD(2)));
end
phi
for i=1:N+1
    for j=1:M+1
        phi(i,j)=atan(sinh(phi(i,j)));
    end
end
phi0=atan(sinh(phi0));
phiD=atan(sinh(phiD));
P0=[lambda0,phi0];
PD=[lambdaD,phiD];
vectanpoint0;
length(vectanpoint0);
%  figure
% latlim = [35 55];
% lonlim = [-10 20];
% axesm('mercator','MapLatLimit',latlim,'MapLonLimit',lonlim,'Origin
    ',[0,0],'MeridianLabel','on','MLabelLocation
    ',[-10,-5,0,5,10,15,20],'ParallelLabel','on','PLabelLocation
    ',[35,40,45,50,55])
% land = shaperead('landareas', 'UseGeoCoords', true);
% geoshow(land, 'FaceColor', [0.5 0.7 0.5]);
% tightmap
% hold on
%
%    plotm(P0(1),P0(2),'k*')
```

```
%    hold on
%    plotm(PD(1),PD(2),'k*')
%    hold on
%    for t=1:length(vectanpoint0)
%        [fil,colu]=find(vectanpoint0(t)==match,1);
%
%        [a,b]=track2('rh',P0(2)*180/pi,P0(1)*180/pi,yv2(fil,colu)*180/
   pi,xv2(fil,colu)*180/pi);
%        plotm(a,b,'r','LineWidth',0.5)
%        linem([P0(2)*180/pi,yv2(fil,colu)*180/pi],[P0(1)*180/pi,xv2(fil
   ,colu)*180/pi],'g');
%        hold on
%    end
%    if puntodesdentro==0&&zdesconf==0||puntodesdentro==1&&zdesconf~=0
%    for t=1:length(vectanpointD)
%        [fil,colu]=find(vectanpointD(t)==match,1);
%        [a,b]=track2('rh',yv2(fil,colu)*180/pi,xv2(fil,colu)*180/pi,PD
   (2)*180/pi,PD(1)*180/pi);
%        plotm(a,b,'b','LineWidth',0.5)
%       linem([yv2(fil,colu)*180/pi,PD(2)*180/pi],[xv2(fil,colu)*180/pi,
   PD(1)*180/pi],'g');
%        hold on
%    end
%    end
%    for t=1:length(vectanorden)
%        [fil,colu]=find(vectanorden(t)==match,1);
%        [fil2,colu2]=find(vectanorden2(t)==match,1);
%         [a,b]=track2('rh',yv2(fil,colu)*180/pi,xv2(fil,colu)*180/pi,yv
   2(fil2,colu2)*180/pi,xv2(fil2,colu2)*180/pi);
%         plotm(a,b,'g','LineWidth',0.5)
%        linem([yv2(fil,colu)*180/pi,yv2(fil2,colu2)*180/pi],[xv2(fil,
   colu)*180/pi,xv2(fil2,colu2)*180/pi],'g');
%        hold on
%    end
%
% for z=1:n
%    plotm(yv2(z,1:chindice(z))*180/pi,xv2(z,1:chindice(z))*180/pi,'k');

%    hold on
% end
% if puntoordentro==1
%    plotm([nodoaux0(2),nodoescape0(2)]*180/pi,[nodoaux0(1),nodoescape
   0(1)]*180/pi,'r');
%    hold on
% end
% if puntodesdentro==1
%    plotm([nodoauxD(2),nodoescapeD(2)]*180/pi,[nodoauxD(1),nodoescapeD
   (1)]*180/pi,'r');
%    hold on
% end
```

```
% if Origen_destino==1
%    plotm([P0(2),PD(2)],[P0(1),PD(2)],'k')
% end

% %    for t=1:length(vectanpoint0)
% %        [fil,colu]=find(vectanpoint0(t)==match,1);
% %        plot([P0(1),xv2(fil,colu)],[P0(2),yv2(fil,colu)],'b')
% %        hold on
% %    end
% %    for t=1:length(vectanpointD)
% %        [fil,colu]=find(vectanpointD(t)==match,1);
% %        plot([PD(1),xv2(fil,colu)],[PD(2),yv2(fil,colu)],'r')
% %        hold on
% %    end
% %    for t=1:length(vectanorden)
% %        [fil,colu]=find(vectanorden(t)==match,1);
% %        [fil2,colu2]=find(vectanorden2(t)==match,1);
% %        plot([xv2(fil,colu),xv2(fil2,colu2)],[yv2(fil,colu),yv2(fil2,
    colu2)],'g')
% %        hold on
% %    end
% %    for z=1:n
% %        plot(xv2(z,1:chindice(z)),yv2(z,1:chindice(z)),'k')
% %        hold on
% %    end
% pause(0.01)
r=zeros(1)
matinc=zeros(1,1);
segment=zeros(1,3);
matcost=zeros(1,2);
matcost=zeros(1,2);
matcourse=zeros(1,2)
contador=1;
if puntoordentro==1
    matinc(1,contador)=1
    matinc(contchconf0,contador)=-1
    segment(contador,:)=[contador,1,contchconf0];
    matcost(contador,:)=[contador,t0];
    matcostd(contador,:)=[contador,rec0];
    matcourse(contador,:)=[contador,course0]
    contador=contador+1
end
if Origen_destino==1
    matinc(contchconf0,contador)=1;
    matinc(contchconfD,contador)=-1;
    segment(contador,:)=[contador,contchconf0,contchconfD];
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(PD(1)+P0(1)
        )/2,(PD(2)+P0(2))/2,Wu,pasomallalambda,pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(PD(1)+P0(1)
        )/2,(PD(2)+P0(2))/2,Wv,pasomallalambda,pasomallaphi,M,N);
```

```
    [r,course]=loxodromicequations(P0(2),P0(1),PD(2),PD(1),Rt,Theta,
        vrefISA,0,0);
    [y,t]=ode45(@(y,t)loxodromicequationsint(P0(2),P0(1),Rt,WuintD,
        WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    %matcost(contador,:)=[contador,Rt*sqrt((PD(2)-P0(2))^2+(cos((PD(2)+P
        0(2))/2)*(PD(1)-P0(1)))^2)];
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
    contador=contador+1;

end

puntoordentro
zorconf
%Seguramente se podrá cambiar los for por :
for l=1:length(vectanpoint0)
    if puntoordentro==1
    matinc(contchconf0,contador)=1;
    matinc(vectanpoint0(l),contador)=-1;
    segment(contador,:)=[contador,contchconf0,vectanpoint0(l)];
    [ind_x,ind_y]=ind2sub(size(match),find(vectanpoint0(l)==match,1));
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+P0(1))/2,(yv2(ind_x,ind_y)+P0(2))/2,Wu,pasomallalambda,
        pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+P0(1))/2,(yv2(ind_x,ind_y)+P0(2))/2,Wv,pasomallalambda,
        pasomallaphi,M,N);
    [r,course]=loxodromicequations(P0(2),P0(1),yv2(ind_x,ind_y),xv2(ind_
        x,ind_y),Rt,Theta,vrefISA,0,0);
    [y,t]=ode45(@(y,t)loxodromicequationsint(P0(2),P0(1),Rt,WuintD,
        WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
    %matcost(contador,:)=[contador,Rt*sqrt((yv2(ind_x,ind_y)-P0(2))^2+(
        cos((yv2(ind_x,ind_y)+P0(2))/2)*(xv2(ind_x,ind_y)-P0(1)))^2)];
    contador=contador+1;
    elseif puntoordentro==0&&zorconf==0
    matinc(1,contador)=1;
    matinc(vectanpoint0(l),contador)=-1;
    segment(contador,:)=[contador,1,vectanpoint0(l)];
    [ind_x,ind_y]=ind2sub(size(match),find(vectanpoint0(l)==match,1));
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+P0(1))/2,(yv2(ind_x,ind_y)+P0(2))/2,Wu,pasomallalambda,
        pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+P0(1))/2,(yv2(ind_x,ind_y)+P0(2))/2,Wv,pasomallalambda,
        pasomallaphi,M,N);
```

```
    [r,course]=loxodromicequations(P0(2),P0(1),yv2(ind_x,ind_y),xv2(ind_
        x,ind_y),Rt,Theta,vrefISA,0,0);
    [y,t]=ode45(@(y,t)loxodromicequationsint(P0(2),P0(1),Rt,WuintD,
        WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
    %matcost(contador,:)=[contador,Rt*sqrt((yv2(ind_x,ind_y)-P0(2))^2+(
        cos((yv2(ind_x,ind_y)+P0(2))/2)*(xv2(ind_x,ind_y)-P0(1)))^2)];
    contador=contador+1;
    end
end
for l=1:length(vectanpointD)
    if puntodesdentro==1
    matinc(contchconfD,contador)=-1;
    matinc(vectanpointD(l),contador)=1;
    segment(contador,:)=[contador,vectanpointD(l),contchconfD];
    [ind_x,ind_y]=ind2sub(size(match),find(vectanpointD(l)==match,1));
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+PD(1))/2,(yv2(ind_x,ind_y)+PD(2))/2,Wu,pasomallalambda,
        pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+PD(1))/2,(yv2(ind_x,ind_y)+PD(2))/2,Wv,pasomallalambda,
        pasomallaphi,M,N);
    [yv2(ind_x,ind_y),xv2(ind_x,ind_y)];
    [r,course]=loxodromicequations(yv2(ind_x,ind_y),xv2(ind_x,ind_y),PD
        (2),PD(1),Rt,Theta,vrefISA,0,0);
    [y,t]=ode45(@(y,t)loxodromicequationsint(yv2(ind_x,ind_y),xv2(ind_x,
        ind_y),Rt,WuintD,WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
    %matcost(contador,:)=[contador,Rt*sqrt((yv2(ind_x,ind_y)-PD(2))^2+(
        cos((yv2(ind_x,ind_y)+PD(2))/2)*(xv2(ind_x,ind_y)-PD(1)))^2)];
    contador=contador+1;
    elseif puntodesdentro==0&&zdesconf==0
    matinc(contch,contador)=1;
    matinc(vectanpointD(l),contador)=-1;
    segment(contador,:)=[contador,vectanpointD(l),contch];
    [ind_x,ind_y]=ind2sub(size(match),find(vectanpointD(l)==match,1));
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+PD(1))/2,(yv2(ind_x,ind_y)+PD(2))/2,Wu,pasomallalambda,
        pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x,
        ind_y)+PD(1))/2,(yv2(ind_x,ind_y)+PD(2))/2,Wv,pasomallalambda,
        pasomallaphi,M,N);
    [yv2(ind_x,ind_y),xv2(ind_x,ind_y)];
    [r,course]=loxodromicequations(yv2(ind_x,ind_y),xv2(ind_x,ind_y),PD
        (2),PD(1),Rt,Theta,vrefISA,0,0);
```

```
    [y,t]=ode45(@(y,t)loxodromicequationsint(yv2(ind_x,ind_y),xv2(ind_x,
        ind_y),Rt,WuintD,WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
    %matcost(contador,:)=[contador,Rt*sqrt((yv2(ind_x,ind_y)-PD(2))^2+(
        cos((yv2(ind_x,ind_y)+PD(2))/2)*(xv2(ind_x,ind_y)-PD(1)))^2)];
    contador=contador+1;
    end
end

for z=1:n
    for l=chindice(z):-1:2
        matinc(match(z,l),contador)=1;
        matinc(match(z,l-1),contador)=-1;
        segment(contador,:)=[contador,match(z,l),match(z,l-1)];
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(z,l-1)+
        xv2(z,l))/2,(yv2(z,l-1)+yv2(z,l))/2,Wu,pasomallalambda,
        pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(z,l-1)+
        xv2(z,l))/2,(yv2(z,l-1)+yv2(z,l))/2,Wv,pasomallalambda,
        pasomallaphi,M,N);
    [r,course]=loxodromicequations(yv2(z,l),xv2(z,l),yv2(z,l-1),xv2(z,l
        -1),Rt,Theta,vrefISA,0,0) ;
    if r~=0
    [y,t]=ode45(@(y,t)loxodromicequationsint(yv2(z,l),xv2(z,l),Rt,WuintD,
        WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    else
    t=0;
    end
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
        %matcost(contador,:)=[contador,Rt*sqrt((yv2(z,t)-yv2(z,t-1))^2+(
            cos((yv2(z,t)+yv2(z,t-1))/2)*(xv2(z,t)-xv2(z,t-1)))^2)];
        contador=contador+1;
        matinc(match(z,l),contador)=-1;
        matinc(match(z,l-1),contador)=1;
        segment(contador,:)=[contador,match(z,l-1),match(z,l)];
        [Wuint0]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(z,l
            -1)+xv2(z,l))/2,(yv2(z,l-1)+yv2(z,l))/2,Wu,pasomallalambda,
            pasomallaphi,M,N);
    [Wvint0]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(z,l-1)+
        xv2(z,l))/2,(yv2(z,l-1)+yv2(z,l))/2,Wv,pasomallalambda,
        pasomallaphi,M,N);
    [r,course]=loxodromicequations(yv2(z,l-1),xv2(z,l-1),yv2(z,l),xv2(z,
        l),Rt,Theta,vrefISA,0,0) ;
    if r~=0
    [y,t]=ode45(@(y,t)loxodromicequationsint(yv2(z,l-1),xv2(z,l-1),Rt,
        Wuint0,Wvint0,Theta,vrefISA,course,y,t),[0,r],0);
```

```
    else
    t=0;
    end
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
        %matcost(contador,:)=[contador,Rt*sqrt((yv2(z,t)-yv2(z,t-1))^2+(
            cos((yv2(z,t)+yv2(z,t-1))/2)*(xv2(z,t)-xv2(z,t-1)))^2)];
        contador=contador+1;


    end
end
for l=1:length(vectanorden)
    matinc(vectanorden(l),contador)=1;
    matinc(vectanorden2(l),contador)=-1;
    segment(contador,:)=[contador,vectanorden(l),vectanorden2(l)];
    [ind_x1,ind_y1]=ind2sub(size(match),find(vectanorden(l)==match,1));
    [ind_x2,ind_y2]=ind2sub(size(match),find(vectanorden2(l)==match,1));
    [WuintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x2,
        ind_y2)+xv2(ind_x1,ind_y1))/2,(yv2(ind_x2,ind_y2)+yv2(ind_x1,ind_
        y1))/2,Wu,pasomallalambda,pasomallaphi,M,N);
    [WvintD]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x2,
        ind_y2)+xv2(ind_x1,ind_y1))/2,(yv2(ind_x2,ind_y2)+yv2(ind_x1,ind_
        y1))/2,Wv,pasomallalambda,pasomallaphi,M,N);
    [r,course]=loxodromicequations(yv2(ind_x1,ind_y1),xv2(ind_x1,ind_y1),
        yv2(ind_x2,ind_y2),xv2(ind_x2,ind_y2),Rt,Theta,vrefISA,0,0);
    if r~=0
    [y,t]=ode45(@(y,t)loxodromicequationsint(yv2(ind_x1,ind_y1),xv2(ind_
        x1,ind_y1),Rt,WuintD,WvintD,Theta,vrefISA,course,y,t),[0,r],0);
    else
    t=0
    end
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
    %matcost(contador,:)=[contador,Rt*sqrt((yv2(ind_x1,ind_y1)-yv2(ind_x
        2,ind_y2))^2+(cos((yv2(ind_x1,ind_y1)+yv2(ind_x2,ind_y2))/2)*(xv
        2(ind_x1,ind_y1)-xv2(ind_x2,ind_y2)))^2)];
    contador=contador+1;
    matinc(vectanorden(l),contador)=-1;
    matinc(vectanorden2(l),contador)=1;
    segment(contador,:)=[contador,vectanorden2(l),vectanorden(l)];
    [Wuint0]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x2,
        ind_y2)+xv2(ind_x1,ind_y1))/2,(yv2(ind_x2,ind_y2)+yv2(ind_x1,ind_
        y1))/2,Wu,pasomallalambda,pasomallaphi,M,N);
    [Wvint0]=funcioninterpolavientos(lambdaviento,phiviento,(xv2(ind_x2,
        ind_y2)+xv2(ind_x1,ind_y1))/2,(yv2(ind_x2,ind_y2)+yv2(ind_x1,ind_
        y1))/2,Wv,pasomallalambda,pasomallaphi,M,N);
```

```
    [r,course]=loxodromicequations(yv2(ind_x2,ind_y2),xv2(ind_x2,ind_y2),
       yv2(ind_x1,ind_y1),xv2(ind_x1,ind_y1),Rt,Theta,vrefISA,0,0);
    if r~=0
    [y,t]=ode45(@(y,t)loxodromicequationsint(yv2(ind_x2,ind_y2),xv2(ind_
       x2,ind_y2),Rt,Wuint0,Wvint0,Theta,vrefISA,course,y,t),[0,r],0);
    else
    t=0
    end
    matcost(contador,:)=[contador,t(length(t))];
    matcostd(contador,:)=[contador,r];
    matcourse(contador,:)=[contador,course];
     %matcost(contador,:)=[contador,Rt*sqrt((yv2(ind_x1,ind_y1)-yv2(ind_
        x2,ind_y2))^2+(cos((yv2(ind_x1,ind_y1)+yv2(ind_x2,ind_y2))/2)*(
        xv2(ind_x1,ind_y1)-xv2(ind_x2,ind_y2)))^2)];
    contador=contador+1;
end

if puntodesdentro==1
    matinc(contch,contador)=-1
    matinc(contchconfD,contador)=1
    segment(contador,:)=[contador,contchconfD,contch];
    matcost(contador,:)=[contador,tD];
    matcostd(contador,:)=[contador,recD];
    matcourse(contador,:)=[contador,courseD];
    contador=contador+1
end
if puntoordentro==1
P0=nodoaux0;
lambda0=P0(1);
phi0=P0(2)
end
if puntodesdentro==1
PD=nodoauxD;
lambdaD=PD(1);
phiD=PD(2)
end
matcoord(1:contch,1)=1:contch;
matcoord(1,2:3)=P0;
matcoord(contch,2:3)=PD;
for j=2:contch-1
    [ind_x,ind_y]=ind2sub(size(match),find(j==match,1));
    matcoord(j,2:3)=[xv2(ind_x,ind_y),yv2(ind_x,ind_y)];
end
% for j=1:length(segments(:,2))
%     [ind_x1,ind_y1]=ind2sub(size(match),find(segments(j,2)==match,1)
%     [ind_x2,ind_y2]=ind2sub(size(match),find(segments(j,3)==match,1)
%     s(:,j)=[xv2(ind_x1,ind_y1),yv2(ind_x1,ind_y1)]
%     t
contch
G=digraph(segment(:,2),segment(:,3),matcost(:,2));
```

```
P = shortestpath(G,1,contch)
Costecam=zeros(1);
Costecamd=zeros(1);
for i=2:length(P(:))
a=find(segment(:,2)==P(i-1));
b=find(segment(:,3)==P(i));
h=1;
while h<=length(a)
d=find(a(h)==b(:));
if length(d)==1
h=length(a)+1;
else
h=h+1;
end
end
Costecam(i)=matcost(b(d),2)+Costecam(i-1)
Costecamd(i)=matcostd(b(d),2)+Costecamd(i-1);
end
% if puntodesdentro==1
% i=length(P(:))-1
% a=find(segment(:,2)==P(i-1));
% b=find(segment(:,3)==P(i));
% h=1;
% while h<=length(a)
% d=find(a(h)==b(:));
% if length(d)==1
% h=length(a)+1;
% else
% h=h+1;
% end
% end
% courseD=matcourse(b(d),2);
% end
  figure
latlim = [38 56];
lonlim = [-5 20];
axesm('mercator','MapLatLimit',latlim,'MapLonLimit',lonlim,'Origin
    ',[0,0],'MeridianLabel','on','MLabelLocation',[-5,0,5,10,15,20],'
    ParallelLabel','on','PLabelLocation',[40,45,50,55])
land = shaperead('landareas', 'UseGeoCoords', true);
geoshow(land, 'FaceColor', [0.5 0.7 0.5]);
tightmap
hold on
% if puntoordentro==0&&zorconf~=0
% else
% for t=1:length(vectanpoint0)
%        [fil,colu]=find(vectanpoint0(t)==match,1);
%        if puntoordentro==1
%            [a,b]=track2('rh',nodoescape0(2)*180/pi,nodoescape0(1)*180/
    pi,yv2(fil,colu)*180/pi,xv2(fil,colu)*180/pi);
```

```
%        else
%        [a,b]=track2('rh',P0(2)*180/pi,P0(1)*180/pi,yv2(fil,colu)*180/
   pi,xv2(fil,colu)*180/pi);
%        end
%        plotm(a,b,'g','LineWidth',0.5)
%       % linem([P0(2)*180/pi,yv2(fil,colu)*180/pi],[P0(1)*180/pi,xv2(
   fil,colu)*180/pi],'g');
%        hold on
% end
% end
% zdesconf
% puntodesdentro
% if puntodesdentro==0&&zdesconf~=0
% else
%    for t=1:length(vectanpointD)
%        [fil,colu]=find(vectanpointD(t)==match,1);
%        if puntodesdentro==1
%            [a,b]=track2('rh',yv2(fil,colu)*180/pi,xv2(fil,colu)*180/pi,
   nodoescapeD(2)*180/pi,nodoescapeD(1)*180/pi)
%        else
%        [a,b]=track2('rh',yv2(fil,colu)*180/pi,xv2(fil,colu)*180/pi,PD
   (2)*180/pi,PD(1)*180/pi);
%        end
%        plotm(a,b,'g','LineWidth',0.5)
%       % linem([P0(2)*180/pi,yv2(fil,colu)*180/pi],[P0(1)*180/pi,xv2(
   fil,colu)*180/pi],'g');
%        hold on
%    end
% end
% %       if puntodesdentro==0&&zdesconf~=0
% %    for t=1:length(vectanpointD)
% %        [fil,colu]=find(vectanpointD(t)==match,1);
% %        [a,b]=track2('rh',yv2(fil,colu)*180/pi,xv2(fil,colu)*180/pi,
   PD(2)*180/pi,PD(1)*180/pi);
% %        plotm(a,b,'g','LineWidth',0.5)
% %       linem([yv2(fil,colu)*180/pi,PD(2)*180/pi],[xv2(fil,colu)*180/
   pi,PD(1)*180/pi],'g');
% %        hold on
% %    end
% %       elseif puntodesdentro==1
% %    end
%    for t=1:length(vectanorden)
%        [fil,colu]=find(vectanorden(t)==match,1);
%        [fil2,colu2]=find(vectanorden2(t)==match,1);
%        [a,b]=track2('rh',yv2(fil,colu)*180/pi,xv2(fil,colu)*180/pi,yv
   2(fil2,colu2)*180/pi,xv2(fil2,colu2)*180/pi);
%        plotm(a,b,'g','LineWidth',0.5)
% %      linem([yv2(fil,colu)*180/pi,yv2(fil2,colu2)*180/pi],[xv2(fil,
   colu)*180/pi,xv2(fil2,colu2)*180/pi],'g');
%        hold on
```

```
%     end
%     if Origen_destino==1
%         if puntoordentro==1
%             [a,b]=track2('rh',nodoescape0(2)*180/pi,nodoescape0(1)*180/
    pi,PD(2)*180/pi,PD(1)*180/pi);
%         else
%         [a,b]=track2('rh',P0(2)*180/pi,P0(1)*180/pi,PD(2)*180/pi,PD(1)
    *180/pi);
%         end
%         plotm(a,b,'g','LineWidth',0.5)
%         hold on
%     end
%     if puntoordentro==1
%     plotm([nodoaux0(2),nodoescape0(2)]*180/pi,[nodoaux0(1),nodoescape
    0(1)]*180/pi,'r');
%     hold on
% end
% if puntodesdentro==1
%     plotm([nodoauxD(2),nodoescapeD(2)]*180/pi,[nodoauxD(1),nodoescapeD
    (1)]*180/pi,'r');
%    hold on
% end

for z=1:n
    s=geoshape(yv2(z,1:chindice(z))*180/pi,xv2(z,1:chindice(z))*180/pi);
    geoshow(s.Latitude,s.Longitude)
    hold on

end
% Costecam2(1)=0
% P2=[1,267,573]
% for i=2:length(P2(:))
% a=find(segment(:,2)==P2(i-1));
% b=find(segment(:,3)==P2(i));
% h=1;
% while h<=length(a)
% d=find(a(h)==b(:));
% if length(d)==1
% h=length(a)+1;
% else
% h=h+1;
% end
% end
% matcost(b(d),2)
% Costecam2(i)=matcost(b(d),2)+Costecam2(i-1);
% end
% Costecam2
 Costecam;
 %[dist,P] = dijkstra2(matcoord,segment,1,contch)
```

```
for i=1:length(P(:))-1
[a,b]=track2('rh',matcoord(P(i),3)*180/pi,matcoord(P(i),2)*180/pi,
    matcoord(P(i+1),3)*180/pi,matcoord(P(i+1),2)*180/pi);
plotm(a,b,'r')
hold on
end
% [a,b]=track2('rh',matcoord(26,3)*180/pi,matcoord(26,2)*180/pi,
    matcoord(27,3)*180/pi,matcoord(27,2)*180/pi);
% plotm(a,b,'k')
% hold on
% [a,b]=track2('rh',matcoord(536,3)*180/pi,matcoord(536,2)*180/pi,
    matcoord(537,3)*180/pi,matcoord(537,2)*180/pi);
% plotm(a,b,'k')
% hold on
% [a,b]=track2('rh',matcoord(15,3)*180/pi,matcoord(15,2)*180/pi,
    matcoord(16,3)*180/pi,matcoord(16,2)*180/pi);
% plotm(a,b,'k')
% hold on
plotm(P0(2)*180/pi,P0(1)*180/pi,'r*')
hold on
plotm(PD(2)*180/pi,PD(1)*180/pi,'r*')
hold on

trec=5*60;
if trec>=Costecam(length(P(:)))
    lambda0=lambdaD;
    phi0=phiD;
    P0=[lambda0,phi0];
    if bucle>0
    [a,b]=track2('rh',nodospasoy(length(nodospasox))*180/pi,nodospasox(
        length(nodospasox))*180/pi,phi0*180/pi,lambda0*180/pi);
        plotm(a,b,'k')
        hold on
end
else
nodopos=P(find(Costecam>trec,1))
nodoant=P(length(find(Costecam<trec)))
tstop=trec-Costecam(length(find(Costecam<trec)));
xvant=matcoord(nodoant,2)
yvant=matcoord(nodoant,3)
xvpos=matcoord(nodopos,2)
yvpos=matcoord(nodopos,3)
%Dudas
[Wuint]=funcioninterpolavientos(lambdaviento,phiviento,(xvant+xvpos)/2,(
    yvant+yvpos)/2,Wu,pasomallalambda,pasomallaphi,M,N);
[Wvint]=funcioninterpolavientos(lambdaviento,phiviento,(xvant+xvpos)/2,(
    yvant+yvpos)/2,Wv,pasomallalambda,pasomallaphi,M,N);
```

```
[r1,course1]=loxodromicequations(yvant,xvant,yvpos,xvpos,Rt,Theta,
    vrefISA,0,0)
options = odeset('Events',@(y,t)myEventsFcn(tstop,y,t));
[y,t,ye,te,ie]=ode45(@(y,t)loxodromicequationsint(yvant,xvant,Rt,Wuint,
    Wvint,Theta,vrefISA,course1,y,t),[Costecamd(find(Costecam<trec,1)),
    Costecamd(find(Costecam>trec,1))-Costecamd(length(find(Costecam<trec
    ,1)))],0,options);
[y(length(y)),t(length(y))]
[ye,te]

if bucle>0
    [a,b]=track2('rh',nodospasoy(length(nodospasox))*180/pi,nodospasox(
        length(nodospasox))*180/pi,phi0*180/pi,lambda0*180/pi);
        plotm(a,b,'k')
        hold on
end
course1
[phi0,lambda0]=loxodromicequationsinv(yvant,xvant,Rt,Theta,vrefISA,
    course1,ye,yvpos,xvpos);
P0=[lambda0,phi0]
end
plotm(P0(2)*180/pi,P0(1)*180/pi,'r*');
hold on
if length(find(Costecam<trec))>1
for i=1:length(find(Costecam<trec))-1
[a,b]=track2('rh',matcoord(P(i),3)*180/pi,matcoord(P(i),2)*180/pi,
    matcoord(P(i+1),3)*180/pi,matcoord(P(i+1),2)*180/pi);
plotm(a,b,'b')
hold on
end
[a,b]=track2('rh',matcoord(P(length(find(Costecam<trec))),3)*180/pi,
    matcoord(P(length(find(Costecam<trec))),2)*180/pi,phi0*180/pi,lambda
    0*180/pi);
plotm(a,b,'b')
hold on

else
[a,b]=track2('rh',matcoord(P(1),3)*180/pi,matcoord(P(1),2)*180/pi,phi
    0*180/pi,lambda0*180/pi);
plotm(a,b,'b')
hold on
end

if length(nodospasox)>1
    for i=1:length(nodospasox)-1
        [a,b]=track2('rh',nodospasoy(i)*180/pi,nodospasox(i)*180/pi,
            nodospasoy(i+1)*180/pi,nodospasox(i+1)*180/pi);
        plotm(a,b,'k')
        hold on
    end
```

```
end


for i=1:length(find(Costecam<trec))
    nodospasox(puntospaso)=matcoord(P(i),2);
    nodospasoy(puntospaso)=matcoord(P(i),3);
    puntospaso=puntospaso+1
end
segmentositer(bucle+1)=length(find(Costecam<trec))+1
bucle=bucle+1
pause(0.01)

end
  figure
latlim = [38 56];
lonlim = [-5 20];
axesm('mercator','MapLatLimit',latlim,'MapLonLimit',lonlim,'Origin
    ',[0,0],'MeridianLabel','on','MLabelLocation',[-5,0,5,10,15,20],'
    ParallelLabel','on','PLabelLocation',[40,45,50,55])
land = shaperead('landareas', 'UseGeoCoords', true);
geoshow(land, 'FaceColor', [0.5 0.7 0.5]);
tightmap
hold on
contador=0
for i=1:bucle-1
    if i==1
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
        *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
        ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
        nodospasox(contador+2:contador+segmentositer(i))*180/pi);
        plotm(a,b,'r')
        hold on
elseif i==2
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
        *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
        ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
        nodospasox(contador+2:contador+segmentositer(i))*180/pi);
        plotm(a,b,'b')
        hold on
elseif i==3
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
        *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
        ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
        nodospasox(contador+2:contador+segmentositer(i))*180/pi);
        plotm(a,b,'g')
        hold on
elseif i==6
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
        *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
```

```
       ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
       nodospasox(contador+2:contador+segmentositer(i))*180/pi);
       plotm(a,b,'p')
       hold on
elseif i==5
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
       *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
       ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
       nodospasox(contador+2:contador+segmentositer(i))*180/pi);
       plotm(a,b,'m')
       hold on
elseif i==7
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
       *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
       ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
       nodospasox(contador+2:contador+segmentositer(i))*180/pi);
       plotm(a,b,'y')
       hold on
elseif i==4
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
       *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
       ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
       nodospasox(contador+2:contador+segmentositer(i))*180/pi);
       plotm(a,b,'c')
       hold on
elseif i==8
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
       *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
       ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
       nodospasox(contador+2:contador+segmentositer(i))*180/pi);
       plotm(a,b,'r')
       hold on
elseif i==9
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
       *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
       ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
       nodospasox(contador+2:contador+segmentositer(i))*180/pi);
       plotm(a,b,'b')
       hold on
elseif i==10
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
       *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
       ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
       nodospasox(contador+2:contador+segmentositer(i))*180/pi);
       plotm(a,b,'g')
       hold on
elseif i==11
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
       *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
```

```
        ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
        nodospasox(contador+2:contador+segmentositer(i))*180/pi);
        plotm(a,b,'k')
        hold on
elseif i==12
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
        *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
        ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
        nodospasox(contador+2:contador+segmentositer(i))*180/pi);
        plotm(a,b,'k')
        hold on
elseif i==13
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
        *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
        ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
        nodospasox(contador+2:contador+segmentositer(i))*180/pi);
        plotm(a,b,'y')
        hold on
elseif i==14
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
        *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
        ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
        nodospasox(contador+2:contador+segmentositer(i))*180/pi);
        plotm(a,b,'c')
        hold on
elseif i==15
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
        *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
        ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
        nodospasox(contador+2:contador+segmentositer(i))*180/pi);
        plotm(a,b,'r')
        hold on
elseif i==16
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
        *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
        ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
        nodospasox(contador+2:contador+segmentositer(i))*180/pi);
        plotm(a,b,'b')
        hold on
elseif i==17
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
        *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
        ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
        nodospasox(contador+2:contador+segmentositer(i))*180/pi);
        plotm(a,b,'g')
        hold on
elseif i==18
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
        *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
```

```
       ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
       nodospasox(contador+2:contador+segmentositer(i))*180/pi);
       plotm(a,b,'k')
       hold on
elseif i==19
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
       *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
       ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
       nodospasox(contador+2:contador+segmentositer(i))*180/pi);
       plotm(a,b,'m')
       hold on
elseif i==20
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
       *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
       ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
       nodospasox(contador+2:contador+segmentositer(i))*180/pi);
       plotm(a,b,'y')
       hold on
elseif i==21
    [a,b]=track2('rh',nodospasoy(contador+1:contador+segmentositer(i)-1)
       *180/pi,nodospasox(contador+1:contador+segmentositer(i)-1)*180/pi
       ,nodospasoy(contador+2:contador+segmentositer(i))*180/pi,
       nodospasox(contador+2:contador+segmentositer(i))*180/pi);
       plotm(a,b,'c')
       hold on
    end
       contador=contador+segmentositer(i)-1;

%break
end
for i=1:bucle-1
    storms_TFG = load('storms_TFG.mat')
    storms_TFG = struct2cell(storms_TFG)
    storm_TFG = storms_TFG{bucle};
    k=1;
    storm_TFG(k);
[n(k),A]=size(storm_TFG{k});
n(k);
for z=1:n(k)
lengthstorm(z)=length(storm_TFG{k}(z).Lat);
[z,lengthstorm(z)];
xv(z,1:lengthstorm(z))=storm_TFG{k}(z).Lon'*pi/180;
yv(z,1:lengthstorm(z))=storm_TFG{k}(z).Lat'*pi/180;
if i==1
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')
hold on
elseif i==2
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'b')
hold on
elseif i==3
```

```
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'g')
hold on
elseif i==6
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'k')
hold on
elseif i==5
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'m')
hold on
elseif i==7
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'y')
hold on
elseif i==4
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'c')
hold on
elseif i==8
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')
hold on
elseif i==9
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'b')
hold on
elseif i==10
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'g')
hold on
elseif i==11
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'k')
hold on
elseif i==12
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'m')
hold on
elseif i==13
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'y')
hold on
elseif i==14
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'c')
hold on
elseif i==15
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'r')
hold on
elseif i==16
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'b')
hold on
elseif i==17
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'g')
hold on
elseif i==18
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'k')
hold on
elseif i==19
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'m')
hold on
```

```
elseif i==20
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'y')
hold on
elseif i==21
plotm(yv(z,1:lengthstorm(z))*180/pi,xv(z,1:lengthstorm(z))*180/pi,'c')
hold on
end
end
end
 if puntoordentro==1
     disp('punto inicial o final dentro de un polígono')
 end
%----------FUNCIONES USADAS------------%
%
    ------------------------------------------------------------------------------------------

function [r,course]=loxodromicequations(phi0,lambda0,phi,lambda,Rt,Theta
    ,vrefISA,N,M)
for i=1:N+1
    for j=1:M+1
        if abs(lambda(i,j)-lambda0)>pi
            if lambda(i,j)<0
                course(i,j)=atan((lambda(i,j)+2*pi-lambda0)/log(tan(pi/4-
                    phi0/2)/tan(pi/4-phi(i,j)/2)));
            elseif lambda(i,j)>0
                course(i,j)=atan((lambda(i,j)-2*pi-lambda0)/log(tan(pi/4-
                    phi0/2)/tan(pi/4-phi(i,j)/2)));
%             else
%                 course(i,j)=atan((lambda(i,j)-lambda0)/log(tan(pi/4-phi
    0/2)/tan(pi/4-phi(i,j)/2)));
            end
        elseif abs(lambda(i,j)-lambda0)<pi
            course(i,j)=atan((lambda(i,j)-lambda0)/log(tan(pi/4-phi0/2)/
                tan(pi/4-phi(i,j)/2)));
        else
            course(i,j)=0;
        end
        if phi(i,j)<phi0
            course(i,j)=course(i,j)+pi;
        end
if lambda0>lambda(i,j)&&phi0>phi(i,j)
    course(i,j)=course(i,j)-2*pi;
    end

%Calculo del curso, constante para cada par de puntos

if (phi(i,j)-phi0)==0
    %r(i,j)=acos(cos(pi/2-phi0)*cos(pi/2-phi0) + sin(pi/2-phi0)*sin(pi
        /2-phi0)*cos(abs(lambda(i,j)-lambda0)))*Rt*sqrt((cos(lambda0)*cos
        (phi0))^2+(sin(lambda0)*cos(phi0))^2)
```

```
    r(i,j)=abs(lambda(i,j)-lambda0)*cos(phi0)*Rt;
elseif lambda(i,j)==lambda0
    r(i,j)=Rt*abs(phi(i,j)-phi0);
elseif (phi(i,j)-phi0)==0&&(lambda(i,j)-lambda0)==0
    course(i,j)=0;
    r(i,j)=0;
else
alphalox(i,j)=(phi(i,j)-phi0)/cos(course(i,j));
r(i,j)=alphalox(i,j)*Rt;
end
end
end
%Problema inverso, a partir de r calculamos los phi correspondientes
end
function [viento]=funcioninterpolavientos(lambdavientos,phivientos,
    lambda,phi,W,pasomallalambda,pasomallaphi,M,N)
%Calculamos los indices del punto superior izquierdo.
i=floor((lambda-lambdavientos(1,1))*180/pi/pasomallalambda)+1;
    j=floor((phivientos(1,1)-phi)*180/pi/pasomallaphi)+1;
    [lambda*180/pi;lambdavientos(1,1)*180/pi];
    [j,i];
if i==M+1
    i=i-1;
end
if j==N+1
    j=j-1;
end
    viento=(W(j,i)*(lambdavientos(j+1,i+1)-lambda)*(phi-phivientos(j+1,i
        +1))...
          +W(j,i+1)*(lambda-lambdavientos(j+1,i))*(phi-phivientos(j+1,i))
              ...
          +W(j+1,i)*(lambdavientos(j,i+1)-lambda)*(phivientos(j,i+1)-phi)
              ...
          +W(j+1,i+1)*(lambda-lambdavientos(j,i))*(phivientos(j,i)-phi))
              ...
          /((lambdavientos(j+1,i+1)-lambdavientos(j,i))*(phivientos(j,i)-
              phivientos(j+1,i+1)));
end
    function [dtdr]=loxodromicequationsint(phi0,lambda0,Rt,Wu,Wv,Theta,
        vrefISA,curso,y,t)
%Problema inverso, a partir de r calculamos los phi correspondientes
        phi2=phi0+y/Rt*cos(curso);
        lambda2=lambda0+tan(curso)*log(tan(pi/4-phi0/2)/tan(pi/4-phi2/2))
            ;
%Calculo de las valocidades:
    V=vrefISA*sqrt(Theta);
    Wat=Wu*cos(pi/2-curso)+Wv*sin(pi/2-curso);
    Wxt=-Wu*sin(pi/2-curso)+Wv*cos(pi/2-curso);
        %Velocidad respecto a tierra en función de r
    Vg=sqrt(V^2-Wxt^2)+Wat;
```

```
    dtdr=1./Vg;
end
function [phi2,lambda2]=loxodromicequationsinv(phi0,lambda0,Rt,Theta,
    vrefISA,curso,y,phiD,lambdaD)
%Problema inverso, a partir de r calculamos los phi correspondientes
if lambda0==lambdaD
       phi2=phi0+y*cos(curso)/Rt
       lambda2=lambda0
elseif phi0==phiD
    lambda2=lambda0+y*sin(curso)/(cos(phi0)*Rt)
    phi2=phi0
else
    phi2=phi0+y/Rt*cos(curso);
        lambda2=lambda0+tan(curso)*log(tan(pi/4-phi0/2)/tan(pi/4-phi2/2))
            ;
end
end


function [value,isterminal,direction] = myEventsFcn(tstop,y,t)
value=t-tstop;
 isterminal=1;
direction=0;
end

function [xv,yv,ch,chindice,xv2,yv2,z,n,mlength]=...
    fusionpoligonos(xv2,yv2,z,xv,yv,ch,chindice,n,mlength)
    p=1;
    while p<z
         if z~=p
[xi,yi]=polyxpoly(xv2(p,1:chindice(p)),yv2(p,1:chindice(p)),...
    xv2(z,1:chindice(z)),yv2(z,1:chindice(z)));
[in1,on1]=inpolygon(xv2(p,1:chindice(p)),yv2(p,1:chindice(p)),...
    xv2(z,1:chindice(z)),yv2(z,1:chindice(z)));
[in2,on2]=inpolygon(xv2(z,1:chindice(z)),yv2(z,1:chindice(z)),...
    xv2(p,1:chindice(p)),yv2(p,1:chindice(p)));
xi3=find(in2);
xi2=find(in1);
if length(xi)>1||length(xi3)==length(in2)
    plength=mlength(z,:)+mlength(p,:);
    xv(p,1:(mlength(z,:)+mlength(p,:)))=...
        [xv(p,1:mlength(p,:)),xv(z,1:mlength(z,:))];
    yv(p,1:(mlength(z,:)+mlength(p,:)))=...
        [yv(p,1:mlength(p,:)),yv(z,1:mlength(z,:))];
    mlength(p,:)=plength;
    chv=convhull(xv(p,1:mlength(p,:)),yv(p,1:mlength(p,:)))';
    ch(p,1:length(chv))=chv(:);
    chindice(p)=length(chv);
    xv1=xv(p,chv(:));yv1=yv(p,chv(:));
    xv2(p,1:length(chv))=xv1; yv2(p,1:length(chv))=yv1;
```

```
    xv(z,:)=[];yv(z,:)=[];ch(z,:)=[];chindice(z)=[];
    xv2(z,:)=[];yv2(z,:)=[];mlength(z,:)=[];
    n=n-1; z=0; p=0;
elseif length(xi2)==length(in1)
     plength=mlength(z,:)+mlength(p,:);
    xv(z,1:(mlength(z,:)+mlength(p,:)))=...
        [xv(z,1:mlength(z,:)),xv(p,1:mlength(p,:))];
    yv(z,1:(mlength(z,:)+mlength(p,:)))=...
        [yv(z,1:mlength(z,:)),yv(p,1:mlength(p,:))];
    mlength(z,:)=plength;
    chv=convhull(xv(z,1:mlength(z,:)),yv(z,1:mlength(z,:)))';
    ch(z,1:length(chv))=chv(:);
    chindice(z)=length(chv);
    xv1=xv(z,chv(:));yv1=yv(z,chv(:));
    xv2(z,1:length(chv))=xv1; yv2(z,1:length(chv))=yv1;
    xv(p,:)=[];yv(p,:)=[];ch(p,:)=[];chindice(p)=[];
    xv2(p,:)=[];yv2(p,:)=[];mlength(p,:)=[];
    n=n-1; z=0; p=0;
end
        end
        p=p+1;
    end
end

function [tangente]=tangpointtopolygon(P,xv,yv,l,chindice)
tangente=0;
e1=[xv(l+1)-xv(l),yv(l+1)-yv(l)];
if l<chindice-1
e2=[xv(l+2)-xv(l+1),yv(l+2)-yv(l+1)];
else
e2=[xv(2)-xv(l+1),yv(2)-yv(l+1)];
end
if e1(1)<0&&(P(1)-xv(l+1))*(e1(2)/e1(1))<=(P(2)-yv(l+1))||...
   e1(1)>0&&(P(1)-xv(l+1))*(e1(2)/e1(1))>=(P(2)-yv(l+1))||...
   e1(1)==0&&e1(2)>0&&(P(1)-xv(l+1))>=0||...
   e1(1)==0&&e1(2)<0&&(P(1)-xv(l+1))<=0||...
   e1(2)==0&&e1(1)>0&&(P(2)-yv(l+1))<=0||...
   e1(2)==0&&e1(1)<0&&(P(2)-yv(l+1))>=0 %a la derecha del vertice
   if e2(1)>0&&(P(1)-xv(l+1))*(e2(2)/e2(1))<=(P(2)-yv(l+1))||...
      e2(1)<0&&(P(1)-xv(l+1))*(e2(2)/e2(1))>=(P(2)-yv(l+1))||...
      e2(1)==0&&e2(2)>0&&(P(1)-xv(l+1))<=0||...
      e2(1)==0&&e2(2)<0&&(P(1)-xv(l+1))>=0||...
      e2(2)==0&&e2(1)>0&&(P(2)-yv(l+1))>=0||...
      e2(2)==0&&e2(1)<0&&(P(2)-yv(l+1))<=0
         tangente=1;
end
elseif e1(1)<0&&(P(1)-xv(l+1))*(e1(2)/e1(1))>=(P(2)-yv(l+1))||e1(1)>0&&(
    P(1)-xv(l+1))*(e1(2)/e1(1))<=(P(2)-yv(l+1))||e1(1)==0&&e1(2)>0&&(P
    (1)-xv(l+1))<=0||e1(1)==0&&e1(2)<0&&(P(1)-xv(l+1))>=0||e1(2)==0&&e
    1(1)>0&&(P(2)-yv(l+1))>=0||e1(2)==0&&e1(1)<0&&(P(2)-yv(l+1))<=0
```

```
if e2(1)>0&&(P(1)-xv(l+1))*(e2(2)/e2(1))>=(P(2)-yv(l+1))||e2(1)<0&&(P(1)
    -xv(l+1))*(e2(2)/e2(1))<=(P(2)-yv(l+1))||e2(1)==0&&e2(2)>0&&(P(1)-xv
    (l+1))>=0||e2(1)==0&&e2(2)<0&&(P(1)-xv(l+1))<=0||e2(2)==0&&e2(1)
    >0&&(P(2)-yv(l+1))<=0||e2(2)==0&&e2(1)<0&&(P(2)-yv(l+1))>=0
tangente=1;
end
end
end
     function [xtan,ytan,zvert0poly,Origen_destino]=
         cortetangentespuntopoligono(P,xtan,ytan,xv,yv,n,zvert0poly,PD,
         chindice,zconf,puntodentro)
         Origen_destino=1;
         if puntodentro==0&&zconf~=0
             [X,Y]=size(xtan);
             zvert0poly=zeros(X,Y);
             Origen_destino=1;
         else
         for z=1:n
             if z~=zconf
                 for l=1:length(xtan(z,:))
                     for p=n:-1:1
             if z~=p

                 x1=[P(1),xtan(z,l)];
                 y1=[P(2),ytan(z,l)] ;
             [xi,yi]=polyxpoly(x1,y1,xv(p,1:chindice(p)),yv(p,1:chindice(
                p))) ;
               L=length(xi);
               %if L>2
             for i=1:length(xi);
                 if xi(i)==P(1)&&yi(i)==P(2)
                     L=L-1;
                 end
             end
               %end
             if p==zconf&&L>0
             zvert0poly(z,l)=0;
             break;
             elseif p~=zconf&&L>1
                 zvert0poly(z,l)=0;
                 break
             end
%              end
             end
             end
             %end
             end
             if Origen_destino==1
             x2=[P(1),PD(1)];y2=[P(2),PD(2)];
```

```
            [xi2,yi2]=polyxpoly(x2,y2,xv(z,1:chindice(z)),yv(z,1:
                chindice(z)));
            [z,Origen_destino];
            xi2;
            if length(xi2)>1
                Origen_destino=0;
            end
            end
            end
        end
        end
    end
function [indz,indp]=tangpolygontopolygon(xv2,yv2,l,z,p,chindice)
 P=[xv2(z,l),yv2(z,l)];
 contador=1;
 for t=1:(chindice(p)-1)
   tangente=0;
   [tangente]=tangpointtopolygon(P,xv2(p,:),yv2(p,:),t,chindice(p));
   if tangente==1
      tangente=0;
      %[xv(z,t),yv(z,t)]
      [tangente]=tangpointtopolygon([xv2(p,t+1),yv2(p,t+1)],xv2(z,:),...

      yv2(z,:),l-1,chindice(z));
        if tangente==1
            indz(contador)=l;
            indp(contador)=t+1;
            contador=contador+1;
        end
            indz(contador)=0;
            indp(contador)=0;
            contador=contador+1;
        end
        end

 end
    function [mat1,mat2,mat3,mat4]=cortetangenteentrepoligono(xv,yv,mat
        1,mat2,mat3,mat4,n,chindice)
       %se ha cambiado y las curvas son al rumbo constante de matlab

        for z=1:n
            for p=z+1:n
                xline1=[xv(z,mat1(z,p)),xv(p,mat1(p,z))];
                xline2=[xv(z,mat2(z,p)),xv(p,mat2(p,z))];
                xline3=[xv(z,mat3(z,p)),xv(p,mat3(p,z))];
                xline4=[xv(z,mat4(z,p)),xv(p,mat4(p,z))];
                yline1=[yv(z,mat1(z,p)),yv(p,mat1(p,z))];
                yline2=[yv(z,mat2(z,p)),yv(p,mat2(p,z))];
                yline3=[yv(z,mat3(z,p)),yv(p,mat3(p,z))];
                yline4=[yv(z,mat4(z,p)),yv(p,mat4(p,z))];
```

```
%                [a,b]=track2('rh',yu(z,mat1(z,p)),xu(z,mat1(z,p)),yu(p,
    mat1(p,z)),xu(p,mat1(p,z)));
%                [c,d]=track2('rh',yu(z,mat2(z,p)),xu(z,mat2(z,p)),yu(p,
    mat2(p,z)),xu(p,mat2(p,z)));
%                [e,f]=track2('rh',yu(z,mat3(z,p)),xu(z,mat3(z,p)),yu(p,
    mat3(p,z)),xu(p,mat3(p,z)));
%                [g,h]=track2('rh',yu(z,mat4(z,p)),xu(z,mat4(z,p)),yu(p,
    mat4(p,z)),xu(p,mat4(p,z)));
              for t=1:n
                  if t~=z&&t~=p
%                       s=geoshape(yu(t,1:chindice(t)),xu(t,1:chindice(
    t)));
%                       [x1,y1] = polyxpoly(s.Longitude,s.Latitude,b,a);

                      [x1,y1] = polyxpoly(xline1,yline1,xv(t,1:chindice
                          (t)),yv(t,1:chindice(t)));
                      if length(x1)>1
                          mat1(z,p)=0;
                          mat1(p,z)=0;
                      end
                      %[x2,y2] = polyxpoly(s.Longitude,s.Latitude,d,c);
                      [x2,y2] = polyxpoly(xline2,yline2,xv(t,1:chindice
                          (t)),yv(t,1:chindice(t)));
                      if length(x2)>1
                          mat2(z,p)=0;
                          mat2(p,z)=0;
                      end
                      %[x3,y3] = polyxpoly(s.Longitude,s.Latitude,f,e);
                      [x3,y3] = polyxpoly(xline3,yline3,xv(t,1:chindice
                          (t)),yv(t,1:chindice(t)));
                      if length(x3)>1
                          mat3(z,p)=0;
                          mat3(p,z)=0;
                      end
                      %[x4,y4] = polyxpoly(s.Longitude,s.Latitude,h,g);
                      [x4,y4] = polyxpoly(xline4,yline4,xv(t,1:chindice
                          (t)),yv(t,1:chindice(t)));
                      if length(x4)>1
                          mat4(z,p)=0;
                          mat4(p,z)=0;
                      end

                  end
              end
          end
      end
  end


end
```