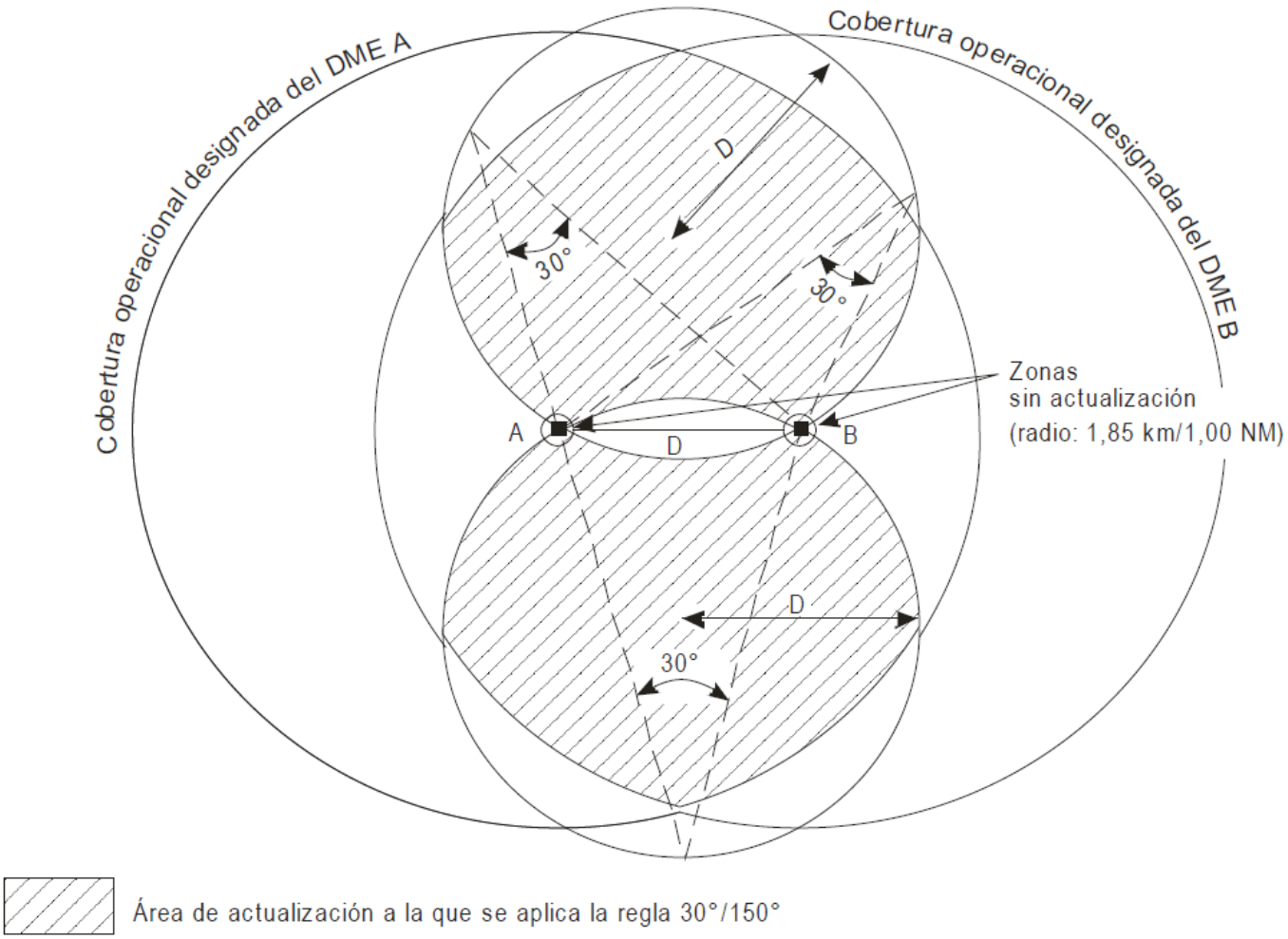
Escuela Técnica Superior de Ingeniería del Diseño

AUTOR: *Vicente Calabuig Simón* **TUTOR:** *Joan Vila Carbó*

GRADO: *Ingeniería Aeroespacial* **CURSO**: 4º

Sistema de posicionamiento RNAV basado en DME's



ÁREA DE ACTUALIZACIÓN DME/DME PARA DOS ESTACIONES DME

Índice de contenidos:

1.- Motivación

Como alumno del *grado en Ingeniería Aeronáutica*, perteneciente a la *rama de Aeronavegación*, tuve claro desde el principio que la **temática** a tratar en el trabajo de fin de grado debía de estar enfocada precisamente a la **Navegación Aérea.**

'Navegación Aérea', como tal, es un tema muy amplio, que abarca muchos otros subtemas, como pueden ser:

- *Procedimientos* de navegación
- *Infraestructuras* aeroportuarias
- Diseño de *hardware* orientado
- Diseño de *software* orientado

Cada uno de estos subtemas puede ser subdividiéndose, dependiendo la forma en que se haga de los criterios que interesen. Por ejemplo:

- *Procedimientos* de navegación
  - Para *vuelo visual*
  - Para *vuelo instrumental*

- *Infraestructuras* aeroportuarias
  - Para *aproximaciones sin precisión*
  - Para *aproximaciones de precisión, I, II, o III*

- Diseño de *hardware/software* orientado
  - Al uso de *controladores aéreos*
  - Al uso de *pilotos (software de abordo)*

De forma que se va pasando subtemas cada vez más específicos. La **cantidad de subtemas** que pueden aparecer siguiendo ésta idea no es para nada despreciable.

En el escenario de un alumno que quiere hacer su trabajo sobre Navegación Aérea, esto significa que:

1. Hay **muchas y variadas opciones** con las que abordar el tema.
   Aspecto que considero positivo en la medida en que aporta más grados de **libertad al alumno** que otros TFG mucho más definidos.

2. Consecuentemente, es necesario **escoger cuidadosamente** una de ellas.

En mi caso, en lugar de pensar directamente en 'que quería hacer', me pareció más idóneo pensar en 'qué no quería hacer' e ir eliminando posibilidades:

De los cuatro ejemplos mencionados al principio descarté enseguida las opciones de *infraestructuras* y *hardware orientado*, principalmente por la **dificultad** que hubiera supuesto la **materialización física** de cualquier elemento en ambos casos. En la mayoría de los casos, esto suele implicar la **dependencia** del alumno para con varios **departamentos** (bien para búsqueda de componentes adecuados o para su montaje), algo que quería evitar en mi caso.

Por otro lado, sabía de antemano que **varios compañeros** de carrera iban a hacer sus respectivos TFG sobre *procedimientos de navegación.* Lo de que alguna manera me hizo descartar la idea de hacer el mío sobre **lo mismo**.

Asique quedaba ***diseño de software orientado***.

Examinando con calma la oferta definitiva de TFG's, me llamo la atención el **proyecto Nº52**, propuesto por Joan Vila, con *descripción inicial*:

'Realización de un sistema que determina la ***posición de una aeronave*** *a partir de una fusión entre las lecturas de todas las radioayudas disponibles en una determinada ubicación. Se trata de determinar en una base de datos las radioayudas disponibles en la posición actual de la aeronave, realizar las lecturas y resolver un sistema sobredeterminado de ecuaciones no lineales para determinar la posición y el error máximo cometido. Se implementará utilizando el* ***simulador de vuelo X-Plane*** *y* ***Matlab*** *o cualquier otro lenguaje de programación.'*

Todos los elementos fundamentales del proyecto me parecían atractivos:

1) **Tutor**, *Joan Vila*: Responsable de la especialidad y antiguo profesor. A diferencia de lo ocurrido en otras asignaturas, sus clases me parecieron siempre interesantes, y sus tutorías realmente solucionaban problemas.

2) **Tema central a tratar**, *estimar la posición de la aeronave, acotando el error máximo cometido*. (**problema fundamental de la navegación área**).

3) **Entorno básico de trabajo**, *X-Plane y Matlab***:** el atractivo de un simulador de vuelo en mi caso no necesita justificación. Por otro lado, con Matlab llevaba trabajando desde primero de carrera, siempre con resultados positivos.

Estas razones, con especial fuerza las tres últimas, son las que me convencieron a dedicarle el tiempo y esfuerzo de un TFG al presente trabajo.

2.- Objetivos del trabajo

A pesar de lo concisa que es *descripción inicial*, las sucesivas reuniones con el tutor a lo largo de la evolución del trabajo han marcado unos objetivos finales algo distintos a lo propuesto inicialmente.

Dichos **objetivos finales** son los *tres siguientes*:

En **primer lugar:** *Determinar las estaciones DME que hacen 'viable' la navegación de una determinada ruta de interés.*

Fase crítica durante el desarrollo de cualquier **plan de vuelo**. Qué se considera 'viable' y los criterios que determinan si una ruta lo es o no serán expuestos con detalle más adelante, junto con el resto de conceptos asociados.

Sí conviene subrayar ya que éste primer objetivo es un proceso de tipo *off-air*, es decir, que va a realizarse con la aeronave todavía en tierra.

En **segundo lugar:** *Estimar la posición (lat, lon) de la aeronave en vuelo en ruta*

Nos interesa **únicamente** estimar la ***longitud***, y la ***latitud*** de la aeronave.

La estimación de la ***altitud*** **no** es un objetivo de éste trabajo.

Éste segundo proceso tiene lugar mientras se vuela la ruta de interés (cuya viabilidad se ha comprobado previamente), y por tanto es de tipo *on-air.* Esto supone una gran diferencia con respecto al caso anterior, sobretodo en términos de criticidad de tiempos de ejecución.

La descripción inicial especificaba utilizar todas las radioayudas disponibles (DME, VOR, GNSS...), sin embargo se utilizaran **únicamente radioayudas de tipo DME.**

En **tercer lugar:** *Acotar y monitonizar el error máximo cometido en la estimación de la posición de la aeronave.*

Si bien los otros dos objetivos siguen siendo bastante interesantes por separado, con éste último adquieren un grado superior de profundidad y utilidad. La **precisión del sistema** es el pilar fundamental sobre el que descansa la navegación por requisitos.

Las ideas de los tres objetivos del trabajo están contempladas por la **OACI**, *cuyo marco regulador* (detallado a continuación) se utilizará como referencia básica durante el desarrollo de las soluciones que permitan la consecución de los objetivos.

3.- Normativa OACI asociada

Viabilidad de la ruta de interés

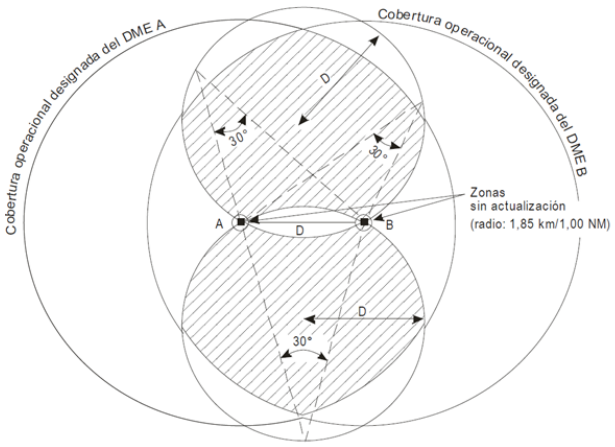
Del documento: *'OACI 8168, Operación de Aeronaves, volumen II (5ª edición / 2006)'*

Capítulo 3: Procedimientos RNAV con DME/DME / Apartado 3.7

Ya que **no es posible saber que instalaciones DME utilizará el sistema de abordo para una actualización de posición**, debería llevarse a cabo una ***verificación de la viabilidad teórica de la ruta*** para garantizar que exista una cobertura DME apropiada disponible en cualquier punto a lo largo de la ruta propuesta, basándose en **por lo menos dos instalaciones seleccionadas** (la cobertura de las estaciones DME se da en la siguiente figura).

La verificación inicial debería llevarse a cabo teniendo en cuenta lo siguiente:

a) El **alcance máximo** de promulgado de la instalación DME, permitiendo un horizonte teórico de radio máximo de la estación de 300 Km (160 NM).
b) El **ángulo de intersección** máximo/mínimo de las estaciones DME (debe estar comprendido entre 30 º y 150º).
c) Que las instalaciones DME situadas **dentro de una distancia de 5.6 Km** (3 NM) respecto de la derrota de diseño **no** puedan utilizarse para la navegación.
d) Las restricciones promulgadas para la cobertura operacional designada, en caso de haberlas.



Área de actualización a la que se aplica la regla 30°/150°

ÁREA DE ACTUALIZACIÓN DME/DME PARA DOS ESTACIONES DME EMPLAZADAS A UNA DISTANCIA "D" ENTRE ELLAS

Etapa 1 — Debe trazarse un círculo con centro en cada estación y radio igual a la cobertura operacional designada (DOC) cuyo valor máximo es de 370,4 km (200 NM).

Etapa 2 — Deben trazarse los círculos de interceptación DME de 30 - 150° con un radio igual a la distancia "D a ambos lados de las dos estaciones DME.

Etapa 3 — Seguidamente se trazan los círculos de las zonas sin actualización de 1,85 km (1,0 NM) con centro e ambas estaciones DME.

El área con actualización doble DME está comprendida dentro del área en que se cumplen las condiciones siguientes:

1. el área dentro del DOC/370,4 km (200 NM), y
2. el área del ángulo de la intersección de 30° - 150°.

Se excluye de la cobertura DME/DME el área comprendida dentro de:

1. los círculos de zonas sin actualización, y
2. el área entre las dos estaciones DME.

Precisión de utilización del sistema

Del documento: *'OACI 8168, Operación de Aeronaves, volumen II (5ª edición / 2006)'*

Capítulo 3: Procedimientos RNAV con DME/DME / Apartado 3.3

La **precisión de utilización del sistema** (DTT, *Dynamic tumor-tracking*) para los sistemas de recepción de abordo se define del modo siguiente:

$$2\sigma = 2\frac{\sqrt{\left(\sigma_{1,air}^2 + \sigma_{1,sis}^2\right) + \left(\sigma_{2,air}^2 + \sigma_{2,sis}^2\right)}}{Sen\alpha}$$

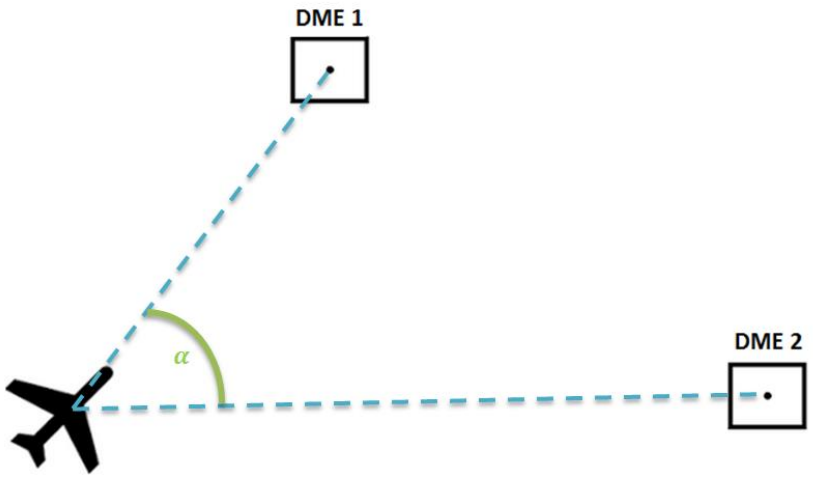
Siendo:

$\sigma_{sis} = 0.05\ NM$

$\sigma_{air} = El\ mayor\ entre$:

- *0.085 NM*
- *El 0.125% de la distancia entre la aeronave y el sistema DME*

$El\ ángulo\ de\ intersección\ (figura\ inferior):\ \ \alpha \in [30º, 150º]$



Monitorización y alerta a bordo

Del documento: '*OACI 9613, Operación de Aeronaves, volumen II (5ª edición / 2006)*'

Capítulo 2: On – Board performance monitoring and alerting

La idea de ésta sección es definir los **requisitos de funcionamiento** del *sistema de monitorización alerta* para navegación que constituye el objetivo número tres (*objetivos)* de éste trabajo.

Para ello, expondremos en primer lugar las características de las distintas **fuentes de error** que degradan la precisión final de navegación.

Navegación lateral

Existen fundamentalmente tres tipos de errores que afectan la precisión de navegación lateral final de la aeronave:

I. PDE: *Path Definition Error*

Tiene lugar cuando la ruta definida por los sistemas de navegación RNAV no se corresponde con la ruta realmente deseada. Éste suceso cobra especial relevancia a la hora de definir secuencias de **giros**; ya que según se empleen waypoints, fly-by points o fly-over points , la ruta definida resultante no es la misma, sino que presenta una cierta variabilidad.

Puesto que en el contexto de éste trabajo no van a definirse como tal rutas de navegación, ésta fuente de error no se tendrá en cuenta.

II. FTE : *Fligth Technical Error*

Éste error está relacionado con la **habilidad de la tripulación** o del **autopiloto** para seguir con precisión la ruta definida previamente.

Por tanto, tampoco tiene relevancia en el ámbito de éste trabajo.

III. NSE : *Navigation System Error*

Éste error es directamente la **diferencia** entre la **posición estimada** de la aeronave y la **posición verdadera** de la aeronave.

Puesto que uno de los objetivos del trabajo es precisamente la **estimación de la posición**, éste es el error que más **directamente nos afecta**, y, por tanto, con el que se va a trabajar.

El NSE se considera un error de tipo **radial**, lo que quiere decir que afecta por igual a la navegación **lateral y** a la **longitudinal.**

El efecto simultáneo de los tres errores anteriores puede agruparse de forma global en el denominado TSE: *Total System Error*

Una figura muy aclarativa acerca de las fuentes de error mencionadas es la siguiente:

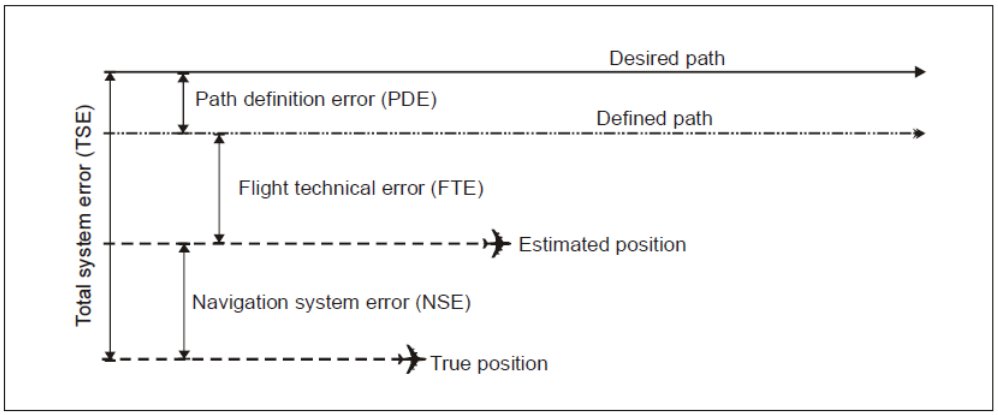


Figure II-A-2-1. Lateral navigation errors (95 per cent)

Distribuciones de los errores

Las **distribuciones** de los errores se asumen que son:

- Independientes
- Gaussianas
  - De media cero.
  - Desviación típica estadística.

Por tanto, la distribución del TSE es también gaussiana, teniéndose además que:

$$TSE = \sqrt{NSE^2 + FTE^2 + PDE^2}$$

Puesto que nosotros sólo vamos a tener en cuenta la contribución del NSE, se tiene entonces directamente que:

$$\boldsymbol{TSE} = \sqrt{NSE^2 + 0^2 + 0^2} = \boldsymbol{NSE}$$

La **distribución** del NSE depende además de:

a. La precisión de los sensores de navegación utilizados.
b. La geometría relativa entre la aeronave y los sensores de navegación utilizados.

En nuestro caso, los sensores de navegación van a ser siempre estaciones DME, cuya información asociada a los apartados anteriores está previamente expuesta: *precisión_DME_DME*

Requisitos de funcionamiento necesarios

Finalmente, para una especificación RNAV 5 empleando únicamente estaciones DME, se definen los siguientes requisitos para el equipo de abordo:

1. *Precisión:*

   El error lateral total, TSE, debe ser como máximo de 5 NM durante el 95% del tiempo de vuelo

2. *Integridad*:

   La probabilidad de que el equipo de abordo funcione incorrectamente (ofreciendo una precisión de mayor que 5 NM) debe ser de al menos $\frac{10^{-5}\ veces}{hora}$.

El sistema debe avisar a la tripulación cuando no ofrezca los requisitos exigidos.

Así pues, la función que lleve a cabo el proceso de estimación de la posición deberá **mostrar siempre el error cometido, avisando cuando éste sea superior a 5 NM**.

4.- Estimación de la posición

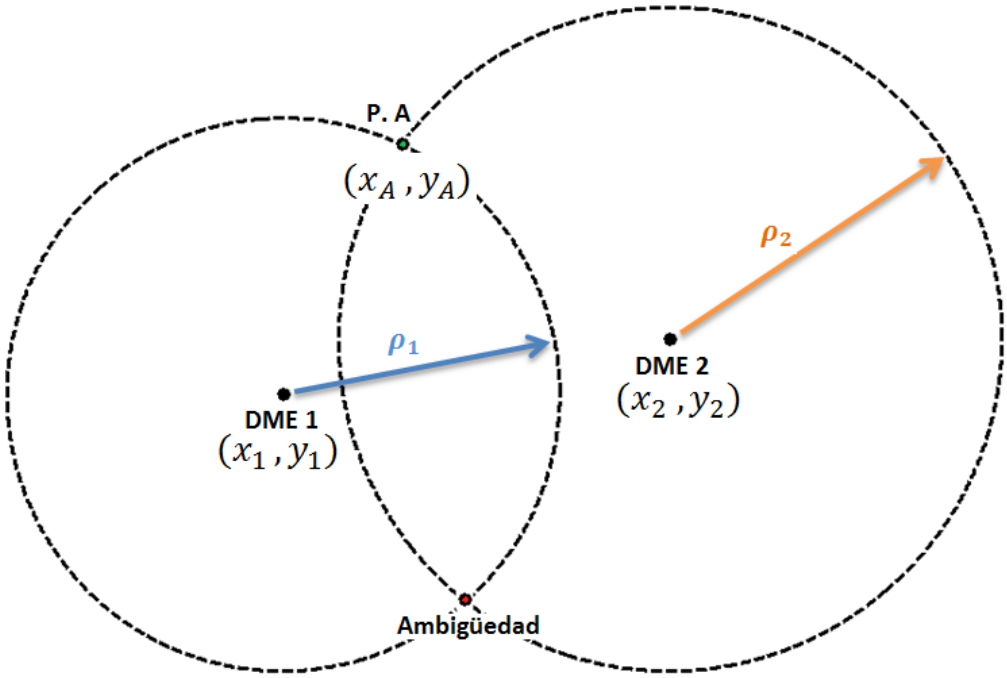
A continuación se presenta un *análisis teórico* de los métodos empleados para estimar la posición de la aeronave.

Recordamos que para ello **emplearemos únicamente radioayudas de tipo DME.**

A éste tipo de métodos de navegación, en los que la **información** de todas las radioayudas empleadas refiere a la **distancia relativa** con la aeronave, se los suele denominar '*navegación rho – rho*' (nombre que quedará justificado enseguida).

El propósito es estimar la posición de la aeronave durante su **fase de vuelo en ruta**, para lo cual podemos

El esquema geométrico inicial del problema es el siguiente:

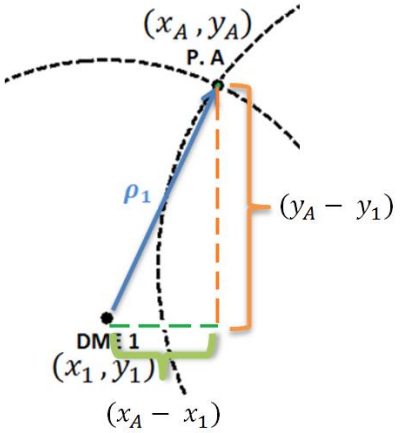


La aeronave (P.A) se encuentra situada a una distancia ρ_1 de la estación DME 1, y, simultáneamente a una distancia ρ_2 de la estación DME 2.

Así, una circunferencia de radio ρ_1, centrada en la posición DME 1, contiene forzosamente a P.A. Análogamente, una circunferencia de radio ρ_2 , centrada en la posición DME 2 también pasa por P.A.

Por tanto, la posición de la aeronave puede hallarse como la **intersección entre circunferencias.**

Para resolver la intersección entre ambas circunferencias, consideremos el siguiente triángulo rectángulo entre la estación DME 1 y P.A:



Por el teorema de Pitágoras se tendrá que:

$$\rho_1{}^2 = (x_A - x_1)^2 + (y_A - y_1)^2$$

De forma análoga se puede proceder con la estación DME 2 y P.A, resultando:

$$\rho_2{}^2 = (x_A - x_2)^2 + (y_A - y_2)^2$$

Con lo que tendríamos un sistema **no lineal,** de 2 ecuaciones y **2 incógnitas**. El problema es que la **solución** del mismo **no es única.**

Utilizar únicamente dos estaciones DME (como se ha hecho por *simplicidad gráfica*) no es suficiente, puesto que aparece una ambigüedad.

Dos posibilidades inmediatas para descartarla son:

- Utilizar estaciones **DME adicionales**
- Contrastar **posiciones anteriores** de la aeronave

En nuestro caso, optaremos por **utilizar siempre tres o más estaciones DME** de manera simultánea.

Así pues, imaginemos que disponemos de las lecturas de un **total de N estaciones.**

Entonces, el sistema a resolver sería:

$$\rho_1{}^2 = (x_A - x_1)^2 + (y_A - y_1)^2$$
$$\rho_2{}^2 = (x_A - x_2)^2 + (y_A - y_2)^2$$
$$\rho_3{}^2 = (x_A - x_3)^2 + (y_A - y_3)^2$$
$$\vdots$$
$$\rho_N{}^2 = (x_A - x_N)^2 + (y_A - y_N)^2$$

- Sistema **no lineal**
- Un total de **N ecuaciones**, es decir, una ecuación por cada estación DME que se utilice.
- Únicamente **2 incógnitas**, independientemente del número de ecuaciones empleadas.
- Se trata por tanto de un sistema **sobredeterminado:** hay más ecuaciones disponibles que incógnitas a resolver.

El sistema **no** puede ser resuelto de forma **analítica/exacta**.

La única posibilidad es por tanto buscar una **aproximación** que podamos considerar **suficientemente buena**: Estableceremos una **tolerancia** al respecto.

Para ello emplearemos el denominado *'Método de Mínimos Cuadrados',* de la siguiente manera:

a) Partimos de una *estimación inicial* de la posición de la aeronave.

b) En un **radio de interés**, entorno a la estimación de posición inicial, calculamos las estaciones DME que se encuentren disponibles.

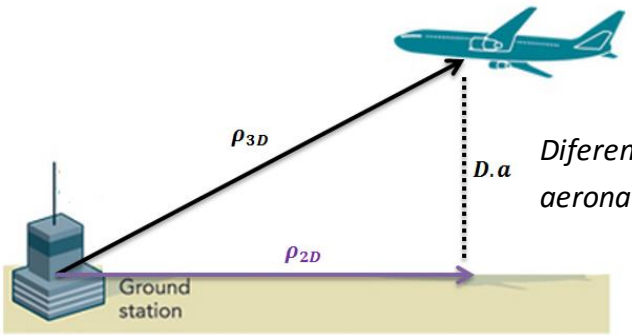
c) Tomando la primera estación DME calculada como referencia, hacemos el siguiente **cambio de coordenadas:**

   1. Desde el *sistema de referencia geodésico LLA* (longitud, latitud, altura), al *sistema de referencia ECEF* (Earth-Centered, Earth-Fixed);

   2. De éste último, al *sistema de navegación NEU* (North, East, Up) centrado en la baliza de referencia.

De esta manera, las coordenadas de todos los elementos empleados quedan dadas relativas a la estación de referencia.

d) Realizamos las **lecturas** de cada estación DME. Puesto que éstas devuelven la distancia 3D o absoluta, y nosotros vamos a trabajar bajo la simplificación 2D de 'tierra plana', será necesario calcular la **proyección horizontal** de la distancia absoluta:



Diferencia de altitud entre la aeronave y la estación DME

Utilizando el teorema de Pitágoras de nuevo, tenemos sencillamente que:

$$\rho_{2D} = \sqrt{\rho_{3D}{}^2 - D.a^2}$$

Mínimos cuadrados ***ponderados***:

1. **Inicialización:** En la primera iteración con la estimación de posición inicial, y en las siguientes con la última aproximación obtenida: $\vec{\boldsymbol{x}}_{\boldsymbol{0}} = (\boldsymbol{x}, \boldsymbol{y})$

2. Cálculo de las **distancia teóricas** $\boldsymbol{\rho_i}$ a cada estación DME $(x_i\,, y_i)$ si la última aproximación obtenida fuera la posición real. Simplemente:

$$\boldsymbol{\rho_i} = \sqrt{(x - x_i)^2 + (y - y_i)^2}$$

3. Cálculo del **error en la última aproximación**, teniendo en cuenta que conocemos la proyección 2D real de cada estación DME a la aeronave:

$$\boldsymbol{\delta\rho_i} = \rho_{2D\,i} - \rho_i$$

4. Formulación del problema de mínimos cuadrados y **cálculo del error** en la estimación de la posición. Sean:

   - $\delta x,\ \delta y$: Los errores cometidos en la estimación de la posición
   - $G = \begin{pmatrix} \frac{x - x_1}{\rho_1} & \frac{y - y_1}{\rho_1} \\ \vdots & \vdots \\ \frac{x - x_N}{\rho_N} & \frac{y - y_N}{\rho_N} \end{pmatrix} \qquad \delta\vec{\rho} = \begin{pmatrix} \delta\rho_1 \\ \delta\rho_2 \\ \vdots \\ \delta\rho_N \end{pmatrix} \qquad W = \begin{pmatrix} \frac{1}{{\sigma_1}^2} & 0 & \cdots & \vdots \\ 0 & \frac{1}{{\sigma_2}^2} & \cdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ \vdots & \vdots & \cdots & \frac{1}{{\sigma_N}^2} \end{pmatrix}$

   Donde σ_i es la precisión que ofrece la estación DME i

   Entonces se tiene la relación matricial: $(G^T \cdot W \cdot G) \cdot \boldsymbol{\delta\vec{x}} = (G^T \cdot W \cdot \delta\vec{p}\,)$

   De donde puede obtenerse:

$$\boldsymbol{\delta\vec{x}} = (G^T \cdot W \cdot G)^{-1} \cdot (G^T \cdot W \cdot \delta\vec{p}\,)$$

5. **Actualización de la estimación:** La nueva estimación de posición se obtiene simplemente añadiendo a la estimación anterior el error de estimación hallado el paso previo:

$$\vec{\boldsymbol{x}} = \overrightarrow{x_0} + \delta\vec{x}$$

6. Comprobar la **convergencia de la solución:** Si el último error de estimación es mayor que la **tolerancia** establecida, seguimos iterando. En caso contrario ya hemos alcanzado la solución deseada.

   La condición a comprobar matemáticamente es: $\sqrt{\delta x^2 + \delta y^2} < Tolerancia$

Remarcar que la solución obtenida está expresada en el sistema de referencia NEU, por lo que hay que **deshacer el cambio de coordenadas** si queremos la posición de la aeronave en el sistema de referencia LLA.

5.- Código Matlab

A continuación se **documentan todas las funciones** desarrolladas en Matlab con el propósito de alcanzar los *tres objetivos* finales del trabajo.

El formato de documentación, para el grueso de las funciones empleadas (*funciones personales*) será el siguiente:

a. *Contexto* del problema que se pretende resolver.
b. *Utilidad* de la función en dicho contexto.
c. *Parámetros de entrada y salida* de la función.
d. *Implementación* de la función.
e. *Ejemplo* de uso.

Si bien la documentación que aquí se ofrece es bastante completa, los propios archivos .m de las funciones contienen siempre al inicio una **descripción completa** de las mismas, que contiene al menos toda la información aquí expuesta, y en algunos casos **notas técnicas adicionales**.

Por supuesto, todas las funciones están pensadas para cumplir con los objetivos del proyecto, cumpliendo con la normativa OACI expuesta anteriormente.

Sin embargo, tengo que destacar que, la **mayor parte del esfuerzo** que me ha supuesto el trabajo han sido las sesiones de prueba y depuración de cada una de las funciones, aunque éste sea un aspecto más relacionado con la programación en sí y no con la aeronáutica.

5.1 Funciones biblioteca Matlab

El método expuesto para la estimación de la posición requiere de una serie de transformaciones de coordenadas entre los sistemas de referencia LLA, ECEF y NEU.

Afortunadamente, la extensa biblioteca de Matlab (a partir de *versión R2014a*) ofrece de forma directa las funciones necesarias para cada uno de los posibles casos de transformación.

El formato de documentación será el que ofrece la propia web: *http://es.mathworks.com*

a. Sintaxis
b. Descripción
c. Ejemplo de uso

5.1.1.- lla2ecef.m

Sintaxis *[lla2ecef.m]*

```
x_ecef = lla2ecef (lla)
```

Descripción *[lla2ecef.m]*

Convierte un conjunto coordenadas geodésicas: latitud [º], longitud [º], altitud [m]; en un conjunto de coordenadas en el sistema de referencia ECEF [m, m, m]

El elipsoide de referencia usado por defecto es el WSG84.

Latitud y longitud pueden tomar cualquier valor; aunque es posible que en +90º y -90º la función devuelva valores inesperados, debido a singularidades en los polos.

Ejemplo de uso *[lla2ecef.m]*

```
x_lla = [15, 15, 1000];              x_ecef = lla2ecef (x_lla)

x_ecef =
     5953150.59         1595141.89          1640358.95
```

5.1.2.- ecef2neu.m

Sintaxis *[ecef2neu.m]*

```
neu = ecef2neu (ecef, orgecef, orgllh)
```

Descripción *[ecef2neu.m]*

Convierte un conjunto de coordenadas ECEF al sistema de referencia East – North – Up con respecto a la localización dada por orgece y orgllh.

'ecef' son las coordenadas que se desean convertir

'orgecef' son las coordenadas de la localización de referencia en formato ECEF

'orgllh' son las coordenadas de la localización de referencia en formato LLA

Ejemplo de uso *[ecef2neu.m]*

```
x_lla = [15, 15, 1000];        r_lla   = [25, 40, 500]
d2r = pi/180;                  r_lla_r = [25*d2r, 40*d2r, 500];

x_ecef = lla2ecef (x_lla)

x_NEU = ecef2neu  (x_ecef', r_lla', r_lla_r')

x_NEU =

         -2604676.43088543

         -874389.421063438

          5755371.03019374
```

5.1.3.- neu2ecef.m

Sintaxis *[ecef2neu.m]*

```
ecef = NEU2ECEF(neu, orgece, orgllh)
```

Descripción *[ecef2neu.m]*

Convierte a ECEF un conjunto de coordenadas NEU referidas a la localización dada por orgece/ orgllh.

'neu' son las coordenadas que se desean convertir

'orgecef' son las coordenadas de la localización de referencia en formato ECEF

'orgllh' son las coordenadas de la localización de referencia en formato LLA

Ejemplo de uso *[ecef2neu.m]*

```
x_lla = [15, 15, 1000];       r_lla   = [25, 40, 500]
d2r = pi/180;                 r_lla_r = [25*d2r, 40*d2r, 500];

x_ecef = lla2ecef (x_lla)

x_NEU = ecef2neu  (x_ecef', r_lla', r_lla_r')

x_ecef = neu2ecef (x_NEU, r_lla', r_lla_r')

x_ecef =

           5953150.5993148

          1595141.89550726

          1640358.95924099
```

5.1.4.- ecef2lla.m

Sintaxis *[ecef2neu.m]*

```
wgs = ECEF2LLA(xyz)
```

Descripción *[ecef2neu.m]*

Convierte a LLA [rad, rad, m] un conjunto de coordenadas ECEF

Ejemplo de uso *[ecef2neu.m]*

```
x_lla = [15, 15, 1000];       r_lla   = [25, 40, 500]
d2r = pi/180;                 r_lla_r = [25*d2r, 40*d2r, 500];

x_ecef = lla2ecef (x_lla)

x_NEU = ecef2neu  (x_ecef', r_lla', r_lla_r')

x_ecef = neu2ecef (x_NEU, r_lla', r_lla_r');

x_lla_r = ECEF2LLA(x_ecef_r);

x_lla_r = [x_lla_r(1:2)*r2d; x_lla_r(3) ]


x_lla_r =

                        15

                        15

                      1000
```

5.2 Funciones proporcionadas por el Tutor

En el contexto de éste trabajo es obvio que se necesita una **comunicación entre Matlab y X-plane**. Dicho de otro modo, necesitamos *acceso* a las variables internas de X-plane, de forma que podamos *leer* sus contenidos, y almacenarlos en variables de Matlab, y el proceso inverso, *escribir* en ellas los valores previamente generados en variables de Matlab.

Las funciones que van a posibilitar dicha comunicación (*comunicación X-Plane)* están basadas en funciones previamente desarrolladas por el tutor del proyecto:

```
function UDPsendDREFfloat (ssock,ip,port,dataref,value)
send_msg=uint8 (zeros (1,509));
send_msg (1:4)=uint8 ('DREF');
send_msg (5)=uint8 (0);

value=single(value);
value_bytes=typecast(value,'uint8');
len=6;
for j=1:4
    send_msg(len)=value_bytes(j);
    len=len+1;
end
dataref_bytes=uint8(dataref);
for j=1:length(dataref)
    send_msg(len)=dataref_bytes(j);
    len=len+1;
end
send_pkt = java.net.DatagramPacket(send_msg,
size(send_msg,2),java.net.InetAddress.getByName(ip),port);
ssock.send(send_pkt);
end

function [index,data]=UDPreceiveDATA(port)
persistent recv_pkt;
if (isempty(recv_pkt))
    recv_pkt = java.net.DatagramPacket(uint8(zeros(1,1085)), 1085);
end
rsock = java.net.DatagramSocket(port);
rsock.receive(recv_pkt);
rsock.close();

recv_data=recv_pkt.getData();

n=recv_pkt.getLength();
i=0;
for j = 6:36:n
    i=i+1;
    index_bytes=recv_data(j:j+3);
    index(i)=typecast(index_bytes,'uint32');
    for k=1:8
        data_bytes=recv_data(j+4*k:j+4*k+3);
        data((i-1)*8+k)=typecast(data_bytes,'single');
    end
end
```

5.3 Funciones desarrolladas personalmente

Escritura en fichero de texto

5.3.1- CellArray2TextFile.m

Contexto *[CellArray2TextFile.m]*

Durante las primeras semanas de trabajo, enseguida surgió la idea de que habría que manejar la base de datos de radioayudas de X-plane (*Manejo BDD*):

a. Leer y almacenar en memoria su información.
b. Operar en memoria de forma conveniente.
c. Generar un nuevo fichero con la nueva información.

En el tercer punto es cuando nos dimos cuenta de que también haría falta una función que **escribiera en ficheros de textos información almacenada en memoria**. Para ello se desarrolló esta función.

Utilidad *[CellArray2TextFile.m]*

Escribe en un fichero de texto (.dat, .txt,…) la información de un CellArray almacenado en memoria.

Parámetros de entrada y salida *[CellArray2TextFile.m]*

Entradas:

- *TextFileName*: Nombre que se le quiere dar al fichero de texto. Debe de contener la extensión de archivo (.dat, .txt, …)

- *CellArray*: CellArray almacenado en memoria cuya información se quiere escribir en el fichero de texto.

Salidas:

- ✓ *Fichero de texto con nombre 'TextFileName' que contiene la misma información que CellArray 'CellArray'.*

Implementación *[CellArray2TextFile.m]*

```
% Apertura del fichero de salida:
output_file = fopen(TextFileName,'w');
output = cell(size(CellArray,1),size(CellArray,2));

% Evaluación y escritura del CellArray.
[filas, columnas] = size(CellArray);
for i = 1 : filas
    for j = 1 :columnas
        % Comprobacion del contenido de la celda {i, j}:
        if numel(CellArray{i,j}) == 0
            output{i,j} = '';
       % Si es un número, lo convertimos a cadena de carácteres:
        elseif isnumeric(CellArray{i,j}) || islogical(CellArray{i,j})
            output{i,j} = num2str(CellArray{i,j}(1,1));
       % Si es un caracter o una cadena de caracteres, no es necesaria
ninguna conversión:
        elseif ischar(CellArray{i,j})
            output{i,j} = CellArray{i,j};
        end;
        % Escribimos la celda {i, j} al fichero de salida.
        if j < columnas
            % Todavía no hemos llegado al final de la línea, añadimos
siguiente celda a la derecha:
            fprintf(output_file,['%s','\t'],output{i,j});
        else
            % Cambio de línea:
            fprintf(output_file,'%s\r\n',output{i,j});
        end
    end;
end;
% Cierre del fichero de salida:
fclose(output_file);
```

Ejemplos de uso *[CellArray2TextFile.m]*

```
CellArray = {'Numeros', 'Letras', 'Palabras'; 1, 'A', 'Manzana';  2,
'B', 'Platano'; 3, 'C', 'Fresa'};

CellArray2TextFile('Numeros_Letras_Palabras.txt',CellArray);
```

Manejo de la base de datos

5.3.2- Reading_Database.m

Contexto *[Reading_Database.m]*

Para poder *analizar la viabilidad de una ruta* necesitamos conocer las **estaciones DME** que se encuentran disponibles en su **recorrido**. De igual manera, para *estimar la posición de la aeronave*, también necesitamos conocer las **estaciones DME** que se encuentren a su **alrededor**.

Por lo tanto, el **problema fundamental** a resolver en ambos casos es ser capaces de **extraer**, de la base de datos (BDD) de radioayudas de X-Plane (*earth_nav.dat*) **un conjunto de radioayudas entorno a una determinada posición.**

Más concretamente, la idea fue crear una función con parámetros de entrada una posición y un radio, y cuya salida fuera precisamente el conjunto de radioayudas entorno al punto especificado dentro del radio especificado.

Pero enseguida nos dimos cuenta de que no podíamos empezar directamente por ahí.

Teníamos que idear antes un **proceso de búsqueda** adecuado a la idea.

Analizando el archivo *earth_nav.dat* a partir de sus *especificaciones*, descubrimos que la BDD de radioayudas original de X-Plane venía ordenada **por tipos de radioayudas.** Lo cual resulto positivo y negativo a partes iguales:

- Por un lado, pudimos extraer rápidamente todas las radioayudas que fueran de tipo DME. (*3_12_13_Database_Non_Sorted.dat*).

- Pero por otro lado, no podíamos realizar aún una búsqueda por posición, puesto que no teníamos una BDD ordenada siguiendo ese criterio.

En este punto, el siguiente paso era **generar** una BDD **ordenada por posiciones** a partir de la BDD *3_12_13_Database_Non_Sorted.dat*

Dicho proceso implicaba la lectura y carga en la memoria de Matlab de la BDD anterior. Para lo cual creí conveniente crear una función que realizara específicamente dicha tarea.

Utilidad *[Reading_Database.m]*

Dada una BDD, en formato .txt o .dat, extraer información parcial o total de la misma y almacenarla en memoria para su posterior manejo.

Parámetros de entrada y salida *[Reading_Database.m]*

Entradas:

- DataBase_file: Archivo .txt o .dat que contiene la BDD de interés
- Linea_Inicial : Primera línea que nos interesa de la BDD
- Linea_Final : Última línea que nos interesa de la BDD

Salidas:

- ✓ Database: Cell Array con la información extraída de la BDD

Implementación *[Reading_Database.m]*

```
fileID   = fopen(DataBase_file);

if fileID == -1
    fprintf('ERROR: Sin acceso al fichero\n')
else
    Database ={};
    Incremento_De_Lineas = Linea_Final - Linea_Inicial;

    while (Linea_Inicial - 1)
        fgetl(fileID);
        Linea_Inicial  = Linea_Inicial - 1;
    end

    for (i = 1 : Incremento_De_Lineas  + 1)
        LineaCompleta    = fgetl(fileID);
        Database(i)      = {LineaCompleta};
    end
    Database = Database';
end
```

Ejemplos de uso *[Reading_Database.m]*

```
[Database] = Reading_Database ('earth_nav.dat', 1, 26590)
```

5.3.3- Sort_DataBase_Lat.m

Contexto *[Sort_DataBase_Lat.m]*

Ahora ya estamos en disposición de operar en memoria con la información de la BDD *3_12_13_Database_Non_Sorted.dat,* de forma que a partir de ella podamos generar otra BDD **cuyo criterio de ordenación** esté relacionado con la **posición** de las estaciones DME.

En cualquiera de las BDD mencionadas con anterioridad, disponemos, para cada radioayuda, de la siguiente información asociada:

i. **Tipo** de radioayuda: Viene referido con la siguiente codificación

2: NDB
3: VOR / VOR-DME / VORTAC
4: Localizador de un sistema ILS
5: Localizador
6: Senda de planeo de un sistema ILS
7: Marcas externas (OM) de un sistema ILS
8: Marcas intermedias (MM) de un sistema ILS
9: Marcas internas (IM) de un sistema ILS
12: DME componente de un sistema ILS, VORTAC o VOR - DME
13: DME único / DME componente de un NDB – DME

ii. **Latitud:** Expresada en **grados decimales**
iii. **Longitud:** Expresada en **grados decimales**
iv. **Altura:** Referente al **MSL** y expresada en **ft**
v. **Frecuencia:** Expresada en **MHZ x 100**
vi. **Rango máximo de recepción:** Expresado en **NM**

Por un lado, a la vista de la codificación referente al tipo de radioayudas, el nombre '3_12_13_Database_Non_Sorted.dat' queda ahora justificado.

Por otro lado, disponemos de **tres fuentes de información** sobre las que trabajar un **criterio** de ordenación basado en la **posición**: *latitud, longitud y altura.*

En este trabajo hemos asumido que vamos a trabajar bajo la simplificación 2D de que una posición viene determinada por su *latitud y su longitud* únicamente. Por lo que debemos utilizar una u otra como criterio, quedando *altura* descartada.

Arbitrariamente se decide que el criterio de ordenación será **latitud creciente.**

Utilidad *[Sort_DataBase_Lat.m]*

Ordenar una base de datos de radioayudas de forma parcial o completa, usando como criterio de ordenación latitud creciente.

Parámetros de entrada y salida *[Sort_DataBase_Lat.m]*

Entradas:

- DataBase_file: Archivo .txt o .dat que contiene la BDD de interés
- Linea_Inicial : Primera línea que nos interesa de la BDD
- Linea_Final : Última línea que nos interesa de la BDD
- NewDatName: Nombre que queremos asignarle a la nueva BDD ordenada

Salidas:

- ✓ Database_Sort_Lat : Cell Array con la nueva BDD ordenada
- ✓ NewDatName.dat/txt: Archivo .dat o .txt con la nueva BDD ordenada

Implementación *[Sort_DataBase_Lat.m]*

```
[Database] = Reading_Database(DataBase_file, Linea_Inicial,
Linea_Final);
[filas, columnas] = size(Database);
Lat = [];
%Bucle de lectura
for (i = 1 : filas)
    % Lectura de la LATITUD del elemento i-ésimo:
    CompleteString  = Database{i};
    Lat_i           = str2num(CompleteString(3:14));
    Lat             = [Lat, Lat_i];
end
% Ordenación de la base de datos
for i = 1 : filas
    for j = i+1 : filas
        if Lat(i) > Lat(j)
            Latj         = Lat(j);
            Lat(j)       = Lat(i);
            Lat(i)       = Latj;
            Databasej    = Database(j);
            Database (j) = Database(i);
            Database (i) = Databasej;   %
        else
            continue;
        end
    end
end

DataBase_Sort_Lat = Database;
dlmcell(NewDatName,DataBase_Sort_Lat);
```

Ejemplos de uso *[Sort_DataBase_Lat.m]*

```
[DataBase_Sort_Lat] = Sort_DataBase_Lat('earth_nav.dat', 1, 26590,
'Complete_DataBase_Lat_Sorted');

[DataBase_Sort_Lat] =
Sort_DataBase_Lat('3_12_13_Database_Non_Sorted.dat',1, 9449,
'3_12_13_Database_Lat_Sorted' );
```

5.3.4- Asignar_indice_lat.m

Contexto *[Asignar_indice_lat.m]*

Buscar en la nueva BDD, *3_12_13_Database_Lat_Sorted.dat*, ya es mucho más eficiente que hacerlo en la original *earth_nav.dat*, por dos razones:

1. Su **tamaño** es mucho menor.
2. Está ordenada según un **criterio** de posición.

Sin embargo, todavía puede añadírsele una tercera propiedad que incrementaría sustancialmente la **eficiencia de búsqueda**: *un archivo asociado de índices de posición.*

Para ello, lo que se plantea ahora es:

En la BDD *3_12_13_Database_Lat_Sorted.dat*, ¿Qué **línea de posición** ocupa la **primera** estación DME cuya latitud (sólo **parte entera**) sea -90, -89, -88… -2, -1, 0, 1, 2, … 88, 89, 90?

Es decir, nuestro archivo de índices de posición tendría la siguiente forma:

Grado de **latitud a buscar** : Lat_B	**Línea/Índice de posición** en *3_12_13_Database_Lat_Sorted.dat* de la 1ª Radioayuda cuya latitud entera es Lat_B
-90	Línea de posición correspondiente
-89	Línea de posición correspondiente
…	…

Utilidad *[Asignar_indice_lat.m]*

Sobre una BDD ordenada según latitudes (3_12_13_Database_Lat_Sorted.dat) asignar un índice de posición en dicha base a cada grado de latitud entero.

Parámetros de entrada y salida *[Asignar_indice_lat.m]*

Entradas:

- DataBase_Lat_Sorted: Archivo .txt o .dat que contiene la BDD ordenada

Salidas:

- ✓ NewDatName.dat/txt: Archivo .dat o .txt con el índice de posiciones asociado

Implementación *[Asignar_indice_lat.m]*

```
[Database_Lat_Sorted] = Reading_Database(DataBase_Lat_Sorted,
Linea_inicial, Linea_final);
%Bucle de lectura de la LATITUD (solo parte entera) del elemento i-
ésimo:
Lat = [];
for i = Linea_inicial : Linea_final
    CompleteString  = Database_Lat_Sorted{i};
    Lat_i           = str2num(CompleteString(3:6));
    Lat             = [Lat, Lat_i];
end
% Bucle de busqueda de lat_buscada y asignacion del indice
correspondiente:
lat_buscada         = -90;
indice_asignado     = [];
posicion            = Linea_inicial;
n_iteracion         = 0;
lat_posicion_array  = [];
LatIndex            = [0,0];
while(posicion <  Linea_final)
    n_iteracion    = n_iteracion + 1;
    lat_posicion   = Lat(posicion);
    check = 1;
    if (lat_buscada > lat_posicion)
        check = 0;
        posicion = posicion + 1;
        lat_posicion       = Lat(posicion);
    end
    if (lat_buscada == lat_posicion )
        indice_asignado = [posicion];
        LatIndex = [LatIndex ;lat_buscada, indice_asignado];
        lat_buscada = lat_buscada + 1;
        posicion = posicion + 1;

    elseif (lat_buscada ~= lat_posicion && lat_buscada~=LatIndex(end,
1) && check)
        lat_buscada = lat_buscada + 1;
    end
end
%Elminacion de la fila 0 0 inicial
LatIndex = LatIndex(2:end, 1:2);
% Conversion de array a cell:
for i = 1 : length(LatIndex)
    LatIndexCell(i, 1) = {LatIndex(i,1)};
    LatIndexCell(i, 2)  = {LatIndex(i, 2)};
end
%Escritura en un fichero .dat
CellArray2TextFile(NewDatName,LatIndexCell);
```

Ejemplos de uso *[Asignar_indice_lat.m]*

```
Asignar_indice_lat('Complete_DataBase_Lat_Sorted.dat', 1, 26590,
'LatIndex_Complete.dat')

Asignar_indice_lat('3_12_13_Database_Lat_Sorted.dat', 1, 9449,
'LatIndex_3_12_13.dat')
```

5.3.5- Find_No_Index_LatDeg.m

Contexto *[Find_No_Index_LatDeg.m]*

Utilizando la función anterior somos capaces de generar el siguiente fichero: *LatIndex_3_12_13.dat*, a la vista del cual reconocemos enseguida que:

- **No todos los grados de latitud** tienen al menos una estación DME.

Lo cual nos va afectar directamente cuando vayamos a buscar mediante índices de posición. Por tanto, tenemos que **conocer de antemano** todos aquellos grados de latitud en los que no hay estaciones DME situadas.

Utilidad *[Find_No_Index_LatDeg.m]*

Genera un fichero .dat con los grados de latitud que no tienen índice de posición asignado en el fichero LatIndex_3_12_13.dat.

Parámetros de entrada y salida *[Find_No_Index_LatDeg.m]*

Entradas:

- *LatIndexDatfile*: Archivo .txt/.dat con la relación de grados de latitud y su posición en una BDD.
- *NewDatName*: Nombre que se le asignará al nuevo fichero .dat con los grados de latitud sin índice.
- *Linea_Final:* Última línea en el fichero con la relación grados de latitud – índices de posición.

Salidas:

- ✓ NewDatName.dat/txt: Archivo .dat o .txt con los grados de latitud que no tienen índice de posición asignado.

Implementación *[Find_No_Index_LatDeg.m]*

```
LatIndexSel= Reading_Database(LatIndexDatfile, 1 ,Linea_Final);
i=0;
faltan =[];
for lat_teo = -90 : 90
    if i < 133
    i=i+1;
    end
     %Leemos las filas por completo:             Ej:  -89 1
    CompleteString         =   LatIndexSel{i};
    %Buscamos donde está el espacio en la fila leída:
    BlankOrNot = isspace(CompleteString);
    BlankPos   = 1;
    while (1)
        if BlankOrNot(BlankPos)
            BlankPos = BlankPos;
            break
        else
            BlankPos = BlankPos + 1;
        end
    end
    %Ahora ya podemos diferenciar los dos elementos de cada fila
    lat_i                  =   CompleteString(1:BlankPos);
    lat_i                  =   lat_i(~isspace(lat_i));
    lat_real               =   str2num(lat_i);
    % Vamos a comparar la latitud teorica con la que existe en la base
de datos
    if (lat_teo ~= lat_real)
        faltan =[faltan; lat_teo];
        i = i-1;
    end
end
% Conversion de array a cell:
for i = 1 : length(faltan)
    LatIndexCell(i, 1)  = {faltan(i,1)};
end
%Escritura en un fichero .dat
CellArray2TextFile(NewDatName,LatIndexCell);
```

Ejemplos de uso *[Find_No_Index_LatDeg.m]*

```
Find_No_Index_LatDeg ('LatIndex_3_12_13.dat',
'LatNOindex_3_12_13.dat', 133);
```

5.3.6- Buscar_Indice.m

Contexto *[Buscar_Indice.m]*

Ahora que conocemos los grados de latitud que carecen de estaciones DME (fichero *LatNOindex_3_12_13.dat*), ya estamos casi en disposición de realizar búsquedas de estaciones DME de manera eficiente.

El único paso que nos falta es **buscar en el fichero anterior el índice de posición** asociado a una determinada latitud (de una posición conocida que nos resulte de interés)

Utilidad *[Buscar_Indice.m]*

Busca en el fichero LatNOindex_3_12_13.dat, el índice de posición en la BDD 3_12_13_Database_Lat_Sorted.dat de una determinada latitud de interés.

Parámetros de entrada y salida *[Buscar_Indice.m]*

Entradas:

- *lat_a_buscar*: latitud de interés cuyo índice de posición se desea conocer. **Debe ser** un **número entero** entre - 90 y + 90.

- *redondeo*: Sirve para resolver los casos en los que el grado de latitud de interés no tenga índice de posición asignado
  - redondeo = 0 -> Se sustituye el grado sin índice por el **menor** más próximo que sí tenga índice.
  - redondeo = 1 -> Se sustituye el grado sin índice por el **mayor** más próximo que sí tenga índice.

Salidas:

- ✓ *Indice:* Índice de posición en *3_12_13_Database_Lat_Sorted.dat* del grado de latitud de interés o del sustituto por redondeo.

```
fileID   = fopen('LatIndex_3_12_13.dat');
Indice   = 0;
Lat_a_buscar_string = num2str(lat_a_buscar);

% Comprobamos si el grado de latitud buscado tiene o no asignado un
índice:
% Almacenamos en memoria las relaciones de asignación con tipo cell:
[LatIndex_3_12_13] = Reading_Database('LatIndex_3_12_13.dat', 1, 133);
[LatNOindex_3_12_13] = Reading_Database('LatNOindex_3_12_13.dat', 1,
48);

% Convertimos las relaciones de asignación a arrays:
LatIndex_3_12_13_Array =[];
LatNOindex_3_12_13_Array =[];
for i = 1: 133
    LatIndex_3_12_13_Array =[LatIndex_3_12_13_Array ;
str2num(LatIndex_3_12_13{i})];
end
for i = 1: 48
    LatNOindex_3_12_13_Array =[LatNOindex_3_12_13_Array ;
str2num(LatNOindex_3_12_13{i})];
end
% Comparamos  el grado a buscar con todos los que no tienen índice
asignado:
Out = 0;
for i = 1 : 48
    if (lat_a_buscar == LatNOindex_3_12_13_Array(i))
        fprintf('El grado de latitud %g no tiene índice de posición
asignado\n', lat_a_buscar);
        % Redondeo hacia abajo
        if (redondeo == 0)
            for j = 1 : 133
                if (LatIndex_3_12_13_Array(end - j, 1) <
LatNOindex_3_12_13_Array(i) )
                    lat_a_buscar = LatIndex_3_12_13_Array(end - j, 1);
                    Lat_a_buscar_string = num2str(lat_a_buscar);
                    out = 1;
                    break;
                end
            end
            fprintf('El grado de latitud MENOR, con asignación,  más
próximo es %g', lat_a_buscar)
            if (out == 1)
                break;
            end
        end
        % Redondeo hacia arriba
        if (redondeo == 1)
            for j = 1 : 133
                if (LatIndex_3_12_13_Array(j, 1) >
LatNOindex_3_12_13_Array(i) )
                    lat_a_buscar = LatIndex_3_12_13_Array(j, 1);
                    Lat_a_buscar_string = num2str(lat_a_buscar);
                    out = 1;
                    break;
                end
            end
            fprintf('El grado de latitud MAYOR, con asignación, más
próximo es %g', lat_a_buscar)
```

```
            if (out == 1)
                break;
            end
        end
    end
end

while (1)
    %Leemos las filas por completo:             Ej:  -89 1
    CompleteString          =   fgetl(fileID);
    %Buscamos donde está el espacio en la fila leída:
    BlankOrNot = isspace(CompleteString);
    BlankPos   = 1;
    while (1)
        if BlankOrNot(BlankPos)
            BlankPos = BlankPos;
            break
        else
            BlankPos = BlankPos + 1;
        end
    end
%Ahora ya podemos diferenciar los dos elementos (grado, índice de
posición) de cada fila
    lat_i                   =   CompleteString(1:BlankPos);
    lat_i                   =   lat_i(~isspace(lat_i));
    Indice_i                =   CompleteString(BlankPos:end);
    Indice_i                =   Indice_i(~isspace(Indice_i));
    if  strcmp(Lat_a_buscar_string, lat_i)
        [Indice] = Indice_i;
        Indice   = str2num(Indice);
        break;
    end
end
```

Ejemplos de uso *[Buscar_Indice.m]*

```
[Indice] = Buscar_Indice(45, 0)      45 SI tiene índice de posición

[Indice] = Buscar_Indice(-47, 0)     -47 NO tiene índice de posición
[Indice] = Buscar_Indice(-47, 1)
```

Manejo de conjuntos de radioayudas

5.3.7- Search_NavAid.m

Contexto *[Search_NavAid.m]*

Ya disponemos de todos los elementos necesarios para realizar una **búsqueda** estaciones DME entorno a una determinada posición **de manera eficiente**:

- Una BDD con todas las estaciones DME, **ordenada** por latitudes.
- Un fichero asociado a la BDD anterior, con los **índices de posición** de cada uno de los grados de latitud.
- Una función que nos devuelve el índice de posición del grado de latitud que nos interese.

Sólo queda concretar el **algoritmo de búsqueda**, que es el siguiente:

1. Partimos de conocer la posición de interés {latI, lonI} y vamos a definir un radio máximo de búsqueda RMax
2. Asumiendo la simplificación 2D de 'Tierra Plana' :
   - 1 grado latitud = 110 km | 1 grado longitud = 111 km

   Transformamos el radio de búsqueda en un incremento de latitud:

$$\Delta_{\text{lat}} = \text{R}_{\text{Max}}\ (\text{km}) \cdot \frac{1\ º}{110\ \text{Km}}$$

3. Un primer filtro (filtro 'Vertical') lo constituyen estaciones DME cuya latitud se encuentre en el intervalo centrado en latI:

$$(\ \text{lat}_\text{I} - \Delta_{\text{lat}};\ \ \text{lat}_\text{I} + \Delta_{\text{lat}})$$

   Se puede implementar sencillamente como sigue:

   a) Definir los grados de búsqueda máximo y mínimo.

   b) Hallar sus correspondientes índices de posición.

   c) Extraer de la BDD la subsección definida por los índices de posición anteriores.

4. Ahora sólo quedaría implementar un segundo filtro ('Filtro Circular') en el que se compruebe directamente si la distancia hasta cada una de las estaciones DME es menor o no al radio máximo de búsqueda. Subproceso que no hubiera tenido sentido hacerse antes, puesto que había muchas más estaciones DME, lo que habría resultado en un tiempo de ejecución exponencialmente mayor.

Utilidad *[Search_NavAid.m]*

Devuelve el conjunto de estaciones DME situadas a una distancia inferior a un radio máximo (a especificar) de la posición de interés.

Parámetros de entrada y salida *[Search_NavAid.m]*

Entradas:

- *PosicionLLA*: Vector con la posición alrededor de la cual se pretende buscar las estaciones DME.

- *Rmax*: Distancia máxima en la que se pretende encontrar radioayudas alrededor de la posición anterior.

Salidas:

- ✓ *NavAid_Valid:* Conjunto de estaciones DME cuya distancia a *PosicionLLA* es inferior a *Rmax.*

Implementación *[Search_NavAid.m]*

```
disp(' ');
fprintf('Posición de la aeronave: Latitud: %g °     |       Longitud:
%g °     |       Altura: %g ft\n', PosicionLLA(1), PosicionLLA(2),
PosicionLLA(3));
disp(' ');
fprintf('Radioayudas situadas a menos de %g Km de la posición de la
aeronave : %g\n', Rmax);
disp(' ');
% Separamos lat_0, lon_0 y alt_0 del array de posición
lat_x0 = PosicionLLA(1);    lon_x0 = PosicionLLA(2);     alt_x0 =
PosicionLLA(3);
% Asumiendo la simplificacion de 'Tierra Plana' : 1 grado latitud  =
110 km | 1 grado longitud = 111 km
latDeg2Km = 110;             Km2LatDeg = 1/latDeg2Km;
lonDeg2Km = 111;             Km2LonDeg = 1/lonDeg2Km;

% Transformamos los km de radio de búsqueda en incrementos de latitud:
'Filtro Vertical'
Delta_lat = Km2LatDeg*Rmax;
```

```
lat_busqueda_min  =(lat_x0-Delta_lat);              lat_busqueda_max
=(lat_x0+Delta_lat);

%Redondeos necesarios para buscar por índice:
lat_busqueda_min = floor(lat_busqueda_min);         lat_busqueda_max =
ceil(lat_busqueda_max);


% Indices que determinan que rango de la base de datos ordenada hay
que leer:
[Indice_min] = Buscar_Indice(lat_busqueda_min, 0);        %Si fuera
neceseario, se redonderia hacia ABAJO
[Indice_max] = Buscar_Indice(lat_busqueda_max, 1);        %Si fuera
neceseario, se redonderia hacia ARRIBA
 Indices = [Indice_min, Indice_max];

% Lectura del rango de la base de datos correspondiente:
[Partial_Database] = Reading_Database(Database, Indice_min,
Indice_max);

% Lectura de los datos correspondientes a las radioayudas halladas:
[NAtype, NAlat, NAlon, NAalt,  NAfreq, NArange] =
Get_NavAid_Data(Partial_Database);

NavAid_total = length(NAtype);

distancia_proyectada = [];      distancia_verdadera   = [];
ft2m = 0.3048;

for i = 1 : NavAid_total
    %Coordenadas de la radioayuda i-esima
    lat_i          = NAlat(i);
    lon_i          = NAlon(i);
    alt_i          = NAalt(i);
    % Diferencia con respecto a la posición de la aeronave
    Delta_lon_i    = lon_i - lon_x0;
    Delta_lat_i    = lat_i - lat_x0;
    Delta_alt_i    = alt_i - alt_x0;
    % Calculo de la distancia proyectada a cada radioayuda
(Simplificacion Tierra Plana)
    dp_i = sqrt(( Delta_lon_i * lonDeg2Km )^2 + (( Delta_lat_i ) *
latDeg2Km )^2); %Km
    distancia_proyectada = [distancia_proyectada; dp_i]; %Km

    % Calculo de la distancia verdadera a cada radioayuda
(Simplificacion Tierra Plana)
    Delta_alt_i_Km = Delta_alt_i * ft2m * (1/1000);

    dv_i = sqrt( dp_i^2 + Delta_alt_i_Km^2);

    distancia_verdadera   = [distancia_verdadera; dv_i]; %Km
end

% Seleccion de las radioayudas que esten a una distancia inferior al
limite establecido: 'Filtro Circular'

Indices_en_Partial_Database_de_NavAid_Valid = [];   NavAid_Valid = {};
j = 0;
for i = 1 : length(distancia_verdadera)
    if (distancia_verdadera(i) < Rmax)
```

```
        j = j+1;
        Indices_en_Partial_Database_de_NavAid_Valid =
[Indices_en_Partial_Database_de_NavAid_Valid ; i];
        StringOut_i =
Partial_Database{Indices_en_Partial_Database_de_NavAid_Valid(end)};
        NavAid_Valid{j, 1} = StringOut_i;
    end
end
```

Ejemplos de uso *[Search_NavAid.m]*

```
lat_0 = 39.56;     lon_0 = 2.743;         alt_0 = 6.466;
x0_LLA = [lat_0, lon_0, alt_0];           (LEPA | PMI)

[NavAid_Valid] = Search_NavAid(x0_LLA, 25)

lat_0 = 39.50;     lon_0 = -0.479;         alt_0 = 225;
x0_LLA = [lat_0, lon_0, alt_0];            (LEVC | VLC)

[NavAid_Valid] = Search_NavAid(x0_LLA, 25)
```

5.3.8- Get_NavAid_Data.m

Contexto *[Get_NavAid_Data.m]*

Siendo ya capaces de encontrar conjunto de radioayudas de interés, sería conveniente ahora una función que **extrajera todos los datos del conjunto**, y los almacenara en distintos arrays por tipo de información.

Utilidad *[Get_NavAid_Data.m]*

Almacena todos los datos referentes a un conjunto de estaciones DME en distintos arrays.

Parámetros de entrada y salida *[Get_NavAid_Data.m]*

Entradas:

- *NavAidSet*: Conjunto de estaciones DME de interés en formato CellArray.

Salidas:

- ✓ *NAtype: Array con el tipo de radioayuda (3, 12, 13).*
- ✓ *NAlat: Array con las latitudes de cada radioayuda.*
- ✓ *NAlon: Array con las longitudes de cada radioayuda.*
- ✓ *NAalt: Array con las altitudes de cada radioayuda.*
- ✓ *NAfreq: Array con las frecuencias de sintonización de cada radioayuda.*
- ✓ *NArange: Array con los rangos máximos de funcionamiento de cada radioayuda.*
- ✓ *NAid: Array con el identificar de la radioayuda.*

Implementación *[Get_NavAid_Data.m]*

```
format long;

% Asignaciones iniciales:
NAtype = [];      NAlat = [];     NAlon = [];     NAalt = [];
NAfreq = [];    NArange = [];     NAid ={};
[filas, X] = size(NavAidSet);

for i = 1 : filas
    S          = NavAidSet{i}
    type       = S( 1  : 2  );      NAtype (i) = str2num(type);
    lat        = S( 3  : 15 );      NAlat  (i) = str2num(lat);
    lon        = S( 16 : 29 );      NAlon  (i) = str2num(lon);
    alt        = S( 30 : 36 );      NAalt  (i) = str2num(alt);
    freq       = S( 37 : 42 );      NAfreq (i) = str2num(freq);
    range      = S( 43 : 46 );      NArange(i) = str2num(range);

    id         = S( 58 : end );     id= id(~isspace(id));
```

```
                                        NAid {i} = id;
End

% Preferimos los arrays en columna:
NAtype = NAtype';   NAlat = NAlat';     NAlon = NAlon' NAalt = NAalt';
NAfreq = NAfreq';   NArange = NArange';     NAid = NAid';
```

Ejemplos de uso *[Get_NavAid_Data.m]*

```
[Database] = Reading_Database('earth_nav.dat', 1, 100) [NAtype, NAlat,
NAlon, NAalt,  NAfreq, NArange] = Get_NavAid_Data(Database);
```

5.3.9- Filter_NavAid.m

Contexto *[Filter_NavAid.m]*

Disponiendo ya del conjunto de radioayudas entorno a una determinada posición, es conveniente ahora filtrarla y quedarnos únicamente con aquellas radioayudas que estén en **rango de funcionamiento válido**.

Utilidad *[Filter_NavAid.m]*

Aplica nuevos filtros un subconjunto de estaciones DME previamente hallado.

- Filtro 1: Que la distancia entre la aeronave y las estaciones DME sea menor que el rango máximo de actuación de éstas.

Parámetros de entrada y salida *[Filter_NavAid.m]*

Entradas:

- *NavAid_Valid*: Conjunto de estaciones DME al que se le quieren aplicar los filtros.

Salidas:

- *NavAid_Valid_Filtered: Subconjunto de estaciones DME que han pasado los filtros aplicados.*

Implementación *[Filter_NavAid.m]*

```
% Posicion de la aeronave con respecto a la cual se va a filtrar el
conjunto de radioayudas:
lat_0 = PosicionLLA(1);  lon_0 = PosicionLLA(2);    alt_0 =
PosicionLLA(3);

% Extraemos todos los datos de las radioayudas en diferentes arrays:
[NAtype, NAlat, NAlon, NAalt,  NAfreq, NArange] =
Get_NavAid_Data(NavAid_Valid);

% Filtro 1) : Distancia (Aeronave, Radioayuda) < Rango actuación
radioayuda
disp(' '); fprintf('Filtro #1 Activo : Subconjunto de radioayudas en
rango de actuación válido\n');  disp(' ');
Nm2Km = 1.852;
NA_total = length(NAtype);

indices = [];  NavAid_Filtered =[]; j=0;
% Calculamos la distancia entre la radioayuda i-ésima y la aeronave
% Asumiendo la simplificacion de 'Tierra Plana' : 1 grado latitud  =
110 km | 1 grado longitud = 111 km
```

```
latDeg2Km = 110;              Km2LatDeg = 1/latDeg2Km;      Nm2Km =
1.852;
lonDeg2Km = 111;              Km2LonDeg = 1/lonDeg2Km;      ft2m  =
0.3048;

distancia_proyectada = [];  distancia_verdadera   = [];
for i = 1 : NA_total
    lat_i           = NAlat(i);
    lon_i           = NAlon(i);
    alt_i           = NAalt(i);
    % Diferencia con respecto a la posición de la aeronave
    Delta_lon_i     = lon_i - lon_0;
    Delta_lat_i     = lat_i - lat_0;
    Delta_alt_i     = alt_i - alt_0;
    % Calculo de la distancia proyectada a cada radioayuda
(Simplificacion Tierra Plana)
    dp_i = sqrt(( Delta_lon_i * lonDeg2Km )^2 + (( Delta_lat_i ) *
latDeg2Km )^2); %Km
    distancia_proyectada = [distancia_proyectada; dp_i]; %Km

    % Calculo de la distancia verdadera a cada radioayuda
(Simplificacion Tierra Plana)
    Delta_alt_i_Km = Delta_alt_i * ft2m * (1/1000);
    dv_i = sqrt( dp_i^2 + Delta_alt_i_Km^2);
    distancia_verdadera   = [distancia_verdadera; dv_i]; %Km
end

% Ahora comprobamos si el rango de la radioayuda i-ésima es mayor que
su distancia verdadera a la aeronave:
indices = [];  NavAid_Filtered =[];
j=0;
for i = 1 : NA_total
    if (NArange(i)*Nm2Km > distancia_verdadera(i))
        j = j+1;
        indice               = [indices, i];
        StringOut_i          = NavAid_Valid{i};
        NavAid_Filtered{j,1} = StringOut_i;
    end
end
```

Ejemplos de uso *[Filter_NavAid.m]*

```
lat_0 = 39.56;     lon_0 = 2.743;      alt_0 = 6.466;     PosicionLLA
= [lat_0, lon_0, alt_0];     Rmax = 200;  (LEPA | PMI)

[NavAid_Valid] = Search_NavAid(PosicionLLA, Rmax,
'3_12_13_Database_Lat_Sorted.dat')

[NavAid_Valid_Filtered] = Filter_NavAid(NavAid_Valid, PosicionLLA)
```

Comunicación con X-Plane

5.3.10- WriteFrequencyNav1Nav2.m

Contexto *[WriteFrequencyNav1Nav2.m]*

La lectura de la distancia DME – aeronave requiere previamente que se **sintonice** la frecuencia de la estación DME correspondiente.

Puesto que X-Plane ofrece dos radios para ello, es más eficiente una función que pueda sintonizar ambas simultáneamente.

Utilidad *[WriteFrequencyNav1Nav2.m]*

Sintonizar en las radios de abordo las frecuencias de las radioayudas que se tenga pensado utilizar.

Parámetros de entrada y salida *[WriteFrequencyNav1Nav2.m]*

Entradas:

- *freq1*: Frecuencia que desea sintonizarse en la primera radio
- freq2 : Frecuencia que desea sintonizarse en la segunda radio
- t_w : Intervalo de tiempo que quiere asignársele al proceso de escritura en X-plane

Salidas:

- ✓ *[/]*

Implementación *[WriteFrequencyNav1Nav2.m]*

```
ip      = '127.0.0.1';
    port    = 6500;
    try
    % Open the receiving socket
    ss = java.net.DatagramSocket();
```

```
    % Writing frequency on Nav_1
    value=single(freq_1);
    dataref='sim/cockpit/radios/nav1_freq_hz';
    UDPsendDREFfloat(ss,ip,port,dataref,value);
    pause(t_w);

    % Writing frequency on Nav_2
    value=single(freq_2);
    dataref='sim/cockpit/radios/nav2_freq_hz';
    UDPsendDREFfloat(ss,ip,port,dataref,value);
    pause(t_w);

    ss.close();
    catch err
    ss.close();
    rethrow(err);
    end

    function UDPsendDREFfloat(ssock,ip,port,dataref,value)
    send_msg=uint8(zeros(1,509));
    send_msg(1:4)=uint8('DREF');
    send_msg(5)=uint8(0);

    value=single(value);
    value_bytes=typecast(value,'uint8');
    len=6;
    for j=1:4
        send_msg(len)=value_bytes(j);
        len=len+1;
    end
    dataref_bytes=uint8(dataref);
    for j=1:length(dataref)
        send_msg(len)=dataref_bytes(j);
        len=len+1;
    end
    send_pkt = java.net.DatagramPacket(send_msg,
size(send_msg,2),java.net.InetAddress.getByName(ip),port);
    ssock.send(send_pkt);
    end
end
```

Ejemplos de uso *[WriteFrequencyNav1Nav2.m]*

```
WriteFrequencyNav1Nav2(10840, 11330, 1)
WriteFrequencyNav1Nav2(00000, 99999, 1)
```

5.3.11- Read_DME_distance.m

Contexto *[Read_DME_distance.m]*

Toda vez que ya tenemos sintonizadas las estaciones DME a utilizar, lo que necesitamos ahora es una función que nos devuelva la **lectura** de la distancia entre las estaciones DME y la aeronave que las sintoniza.

Utilidad *[Read_DME_distance.m]*

Previamente se debe haber sintonizado la frecuencia de la estación DME en Nav_1 o Nav_2.

Una vez sintonizada, ésta función extrae de x-plane la distancia absoluta (slant-range) entre la aeronave y la estación DME sintonizada.

Parámetros de entrada y salida *[Read_DME_distance.m]*

Entradas:

- *Nav_12*: Utilizar '1' si se quiere la lectura de distancia de la estación DME sintonizada en Nav_2. Utilizar '2' si la estación DME está sintonizada en Nav_2.

Salidas:

- ✓ DME_distance: Lectura de distancia obtenida de la estación DME sintonizada.

Implementación *[Read_DME_distance.m]*

```
% nav1 : numeracion 99      Nav 2 : numeracion 100
xp_port_out=49000;
[index,data]=UDPreceiveDATA(xp_port_out);

n=length(index);

for j = 1:n
    switch index(j)
        case 99
            DME_Nav1 = data((j-1)*8+5);
        case 100
            DME_Nav2 = data((j-1)*8+5);
    end
end

% Comprobación de la radio utilizada (Nav_1 o Nav_2)
if (Nav_12 == 1)
```

```
    DME_distance = DME_Nav1;
elseif (Nav_12 == 2)
    DME_distance = DME_Nav2;
else
    fprintf('Error. Seleccione Nav_1 o Nav_2\n');
end

end



function [index,data]=UDPreceiveDATA(port)
persistent recv_pkt;
if (isempty(recv_pkt))
    recv_pkt = java.net.DatagramPacket(uint8(zeros(1,1085)), 1085);
end
rsock = java.net.DatagramSocket(port);
rsock.receive(recv_pkt);
rsock.close();

recv_data=recv_pkt.getData();

n=recv_pkt.getLength();
i=0;
for j = 6:36:n
    i=i+1;
    index_bytes=recv_data(j:j+3);
    index(i)=typecast(index_bytes,'uint32');
    for k=1:8
        data_bytes=recv_data(j+4*k:j+4*k+3);
        data((i-1)*8+k)=typecast(data_bytes,'single');
    end
end
end
```

Ejemplos de uso *[Read_DME_distance.m]*

```
r1 = Read_DME_distance(1);
r2 = Read_DME_distance(2);
```

Objetivo 1: Viabilidad ruta de interés

5.3.12- Database_Segment_filter.m

Contexto *[Database_Segment_filter.m]*

Para analizar la viabilidad de una posible ruta de interés, en términos de las estaciones DME que haya disponibles, lo primero que necesitamos es **filtrar de la base de datos globa**l todas aquellas radioayudas que, en primera instancia, consideremos que puedan estar disponibles a lo largo de la ruta.

Utilidad *[Database_Segment_filter.m]*

Un punto inicial (lat0, lon0) y un punto final (latf, lonf) forman un segmento de recta.

Un área rectangular queda definida por la longitud de dicho segmento (L) y una anchura (A).

La utilidad de ésta función es realizar el siguiente filtrado:

1) Seleccionar de la base de datos '3_12_13_Database_Lat_Sorted.dat' todas aquellas radioayudas que estén dentro del área rectangular anterior.

2) Excluir del subconjunto anterior todas aquellas radioayudas cuya distancia a la derrota de diseño sea inferior a 3 NM (5.6 Km) (OACI 8168 - V2, pág. 539)

Se escribirá en un fichero .txt la base de datos filtrada (de forma que pueda cargarse rápidamente para usos posteriores)

Parámetros de entrada y salida *[Database_Segment_filter.m]*

Entradas:

- *Database_file*: Base de datos que va a utilizarse en el proceso de filtrado.

- lat0: latitud del punto inicial
- latf : latitud del punto final
- lon0 : longitud del punto inicial
- lonf : longitud del punto final

- A: Anchura transversal a la longitud del segmento.
- From_Name: Nombre identificativo del punto inicial
- To_Name : Nombre identificativo del punto final
- Save_Folder: Nombre de la carpeta en la que se desea guardar el fichero de texto generado.

Salidas:

- ✓ NavAid_inside: Cell que contiene la base de datos filtrada para el área de interés.

Implementación *[Database_Segment_filter.m]*

```
% FILTRO 1 : RADIOAYUDAS EN LA REGIÓN DE INTERÉS
% FASE 1 ) PLOT DE LA REGION DE INTERES
% Asumiendo la simplificacion de 'Tierra Plana' : 1 grado latitud  ~
110 Km | 1 grado longitud ~ 111 Km
latDeg2Km = 110;   lonDeg2Km = 111;          Km2LatDeg = 1/latDeg2Km;
Km2LonDeg = 1/lonDeg2Km;


% Trabaremos ahora en unidades de distancia (Km)
x0 = lat0 * latDeg2Km ;     y0 = lon0 * lonDeg2Km ;             P0 =
[x0, y0];               %Km
xf = latf * latDeg2Km ;     yf = lonf * lonDeg2Km;              Pf =
[xf, yf];               %Km

% Conversiones de unidades
r2d = 180/pi;       d2r = pi/180;
alpha = atan (  ( xf - x0 ) / ( yf - y0 )  ) *r2d;

% Segmentos paralelos a la linea central:
    % Segmento paralelo superior:
x1s =  ( x0 - (A/2)*sind (alpha) );        y1s = ( y0 +
(A/2)*cosd(alpha) ) ;        X1S = [ x1s,  y1s ];
x2s =  ( xf - (A/2)*sind (alpha) );        y2s = ( yf +
(A/2)*cosd(alpha) ) ;        X2S = [ x1s,  y1s ];
    % Segmento paralelo inferior:
x1i =  ( x0 + (A/2)*sind (alpha) );        y1i = ( y0 -
(A/2)*cosd(alpha) ) ;        X1i = [ x1i,  y1i ];
x2i =  ( xf + (A/2)*sind (alpha) );        y2i = ( yf -
(A/2)*cosd(alpha) ) ;        X2i = [ x1i,  y1i ];

% Vectores genereales X e Y con las coordenadas de TODOS los puntos de
interes
% El órden será : P0, Pf, PSi, PSd, Pii, Pid

X = [x0, xf, x1s, x2s, x1i, x2i];
Y = [y0, yf, y1s, y2s, y1i, y2i];

% Plot del escenario:
```

```
for i = 1:6
    if     (0 < i && i <= 2)
        plot( X(i), Y(i),     '^blue','LineWidth',1.5);      hold on;
    elseif (2 < i && i <= 6)
        plot( X(i), Y(i),     'oblack','LineWidth',1.3);     hold on;
    end
end
% Linea central
plot ( [ x0, xf ], [ y0, yf ],       '--blue', 'LineWidth',1.5);
% Segmentos paralelos
plot ( [ x1s, x2s ], [ y1s, y2s ],       '-black');     plot ( [ x1i,
x2i ], [ y1i, y2i ],       '-black');
% Segmentos transversales
plot ( [ x1s, x1i ], [ y1s, y1i ],       '-black');     plot ( [ x2s,
x2i ], [ y2s, y2i ],       '-black');
axis square

hold on;

% FASE 2 ) CHECK DE RADIOAYUDAS EN LA REGION

% Rectángulo definido por los 4 vértices: (x1i, y1i)    / (x2i, y2i)
/   (x2s, y2s) /   (x1s, y1s)

xv = [x1i;  x2i;  x2s;  x1s];     xv = [xv; xv(1)];     % Ojo al
'orden' necesario para cerrar el polígono
yv = [y1i;  y2i;  y2s;  y1s];     yv = [yv; yv(1)];

% Cargamos en memoria las radioayudas 1 a 1; y realizamos el proceso
de comprobación
Index_inside = [];      x_inside = [];      y_inside =[];
NavAid_inside = [];

% Optimizamos la busqueda acotando los grados de latitud de interés:
m_lat = min(lat0, latf);    M_lat = max(lat0, latf);    Delta_lat =
(A/2)*Km2LatDeg;

lat_busqueda_min = floor(m_lat - Delta_lat);
lat_busqueda_max = ceil(M_lat  + Delta_lat);
[Indice_min] = Buscar_Indice(lat_busqueda_min, 0);     % Si fuera
neceseario, se redonderia hacia ABAJO
[Indice_max] = Buscar_Indice(lat_busqueda_max, 1);     %Si fuera
neceseario, se redonderia hacia ARRIBA

for i = Indice_min : Indice_max        % Longitud de
3_12_13_Database_Lat_Sorted.dat
    [NavAid_i] = Reading_Database(DataBase_file, i, i);

    [NAtype, NAlat, NAlon, NAalt,  NAfreq, NArange] =
Get_NavAid_Data(NavAid_i);

    xKm = NAlat * latDeg2Km;    yKm = NAlon * lonDeg2Km;

    % FILTRO 1: PERTENENCIA A LA REGIÓN POLIGONAL DE INTERÉS

    plot (xKm, yKm,'ogreen','LineWidth',2);                % Check
visual
    hold on;

    in = inpolygon(xKm,yKm,xv,yv);
```

```
     % FILTRO 2 : EXCLUSIÓN DE RADIAYUDAS SITUADAS A MENOS DE 3 NM
(5.6 Km) DE LA DERROTA DE DISEÑO
     d_2D_valid = 1;

    % Simplificación al caso 2D, por desconocimiento de la altura con
la que se pretende recorrer la ruta de diseño
    % Se trata simplemente de calcular la distancia de un punto
(radioayuda i-ésima) a un segmento de recta

    % Punto inicial del segmento:        x0        y0      z0 = 0;
%Km
    z0 = 0;          X0 = [x0, y0, z0];

    % Punto final del segmento:          xf        yf      zf = 0;
%Km
    zf = 0;         XF = [xf, yf, zf];

    % Coordenadas radioayudas i-ésima    xKm        yKm      zKm = 0;
%Km
    zKm = 0;        P = [xKm, yKm, zKm];

    % Algoritmo de cálculo de la distancia 2D:
    a = X0 - XF;      b = P - XF;               d_2D =
norm(cross(a,b)) / norm(a);

    % Exclusión en caso de que d_2D < 5.6 Km
    if d_2D > 5.6
        d_2D_valid = 1;
    else
        d_2D_valid = 0;
    end

    % Aplicación simultánea de los filtros 1 y 2
    if (in && d_2D_valid)
        Index_inside = [Index_inside;  i];
        x_inside     = [x_inside; xKm];
        y_inside     = [y_inside; yKm];

        NavAid_inside = [NavAid_inside; NavAid_i];

    end
    fclose('all');      % Evita el error  'too many files open'.
end

% ELMINACIÓN de radioayudas con las MISMAS COORDENADAS:
[NAtype, NAlat, NAlon, NAalt,  NAfreq, NArange, NAid] =
Get_NavAid_Data(NavAid_inside);
for i = 1: size(NavAid_inside, 1)
    eliminado = 0;
    j = 1;
    while (1 - eliminado)
        if j > size(NavAid_inside, 1) || i > size(NavAid_inside, 1)
            break;
        end
        if i ~= j && NAlat(i) == NAlat(j) && NAlon(i) == NAlon(j)
             NavAid_inside(j) = [];
            [NAtype, NAlat, NAlon, NAalt,  NAfreq, NArange, NAid] =
Get_NavAid_Data(NavAid_inside);
             eliminado = 1;
        end
        j = j+1;
```

```
    end
end

% ELIMINACIÓN de radioayudas con el MISMO IDENTIFICADOR:
[NAtype, NAlat, NAlon, NAalt,  NAfreq, NArange, NAid] =
Get_NavAid_Data(NavAid_inside);
for i = 1: size(NAid, 1)
    eliminado = 0;
    j = 1;
    while (1 - eliminado)
        if j>size(NAid, 1) || i>size(NAid, 1)
            break;
        end
        if i ~= j && strcmp( NAid(j), NAid(i))
            NavAid_inside(j) = [];
           [NAtype, NAlat, NAlon, NAalt,  NAfreq, NArange, NAid] =
Get_NavAid_Data(NavAid_inside);
            eliminado = 1;
        end
        j = j+1;
    end
end

% ESCRITURA EN UN FICHERO .txt DE LAS RADIOAYUDAS FILTRADAS
tiempo = clock;        hora = num2str(tiempo(4));     mins =
num2str(tiempo(5));

if      (~ischar(From_Name))*(~ischar(To_Name))
    txt_Name = ['FROM ','[',num2str(lat0), ' ; ',num2str(lon0),']
','TO ','[',num2str(latf),' ; ',num2str(lonf),'] '...
    'A ',num2str(A),'___',Date, '_', hora, '_', mins, '.txt'];
elseif  (ischar(From_Name))*(~ischar(To_Name))
    txt_Name = ['FROM ',From_Name,' ','[',num2str(lat0), ' ;
',num2str(lon0),'] ','TO ','[',num2str(latf),' ; ',num2str(lonf),']
'...
    'A ',num2str(A),'___',Date, '_', hora, '_', mins, '.txt'];
elseif (~ischar(From_Name))*(ischar(To_Name))
    txt_Name = ['FROM ','[',num2str(lat0), ' ; ',num2str(lon0),']
','TO ',To_Name,' ','[',num2str(latf),' ; ',num2str(lonf),'] '...
    'A ',num2str(A),'___',Date, '_', hora, '_', mins, '.txt'];
elseif (ischar(From_Name))*(ischar(To_Name))
    txt_Name = ['FROM ',From_Name,' ','[',num2str(lat0), ' ;
',num2str(lon0),'] ','TO ',To_Name,' ','[',num2str(latf),' ;
',num2str(lonf),'] '...
    'A ',num2str(A),'___',Date, '_', hora, '_', mins, '.txt'];
end

CellArray2TextFile(txt_Name,NavAid_inside);
% Movemos el .txt generado a la carpeta escogida:

F = exist(Save_Folder);
if F == 0
   mkdir(pwd, Save_Folder)  % Creamos la carpeta si no existe
previamente
   addpath(Save_Folder);
end

if ispc
    Save_Folder_Path    = [pwd, '/',Save_Folder];
else
    Save_Folder_Path    = [pwd, '\',Save_Folder];
```

```
end
movefile(txt_Name, Save_Folder_Path);
```

Ejemplos de uso *[Database_Segment_filter.m]*

```
Ruta: Valencia --> Madrid

lat0 =  39.4894;     lon0 = -0.4816;
latf = 40.4722;      lonf =  -3.5608;            A =  300;

[NavAid_inside] =
Database_Segment_filter('3_12_13_Database_Lat_Sorted.dat', lat0, lon0,
latf, lonf, A, 'LEVC', 'LEMD', 'Filtros_Offline')
```

5.3.13- DME_DME_Upgrade_area.m

Contexto *[DME_DME_Upgrade_area]*

Disponiendo ya del conjunto filtrado de radioayudas para el segmento de ruta de interés, el siguiente paso es **comprobar la validez de cobertura de los puntos de ruta**, en los términos que se ha especificado previamente (*requisitos viabilidad*)

Utilidad *[DME_DME_Upgrade_area]*

Análisis de la figura III - 1-3-1 del documento OACI 8168 - V2:

> *Área maxima de actualización de dos estaciones DME*

Parámetros de entrada y salida *[DME_DME_Upgrade_area]*

Entradas:

- *lat_a : latitud de la aeronave*
- lon_a : longitud de la aeronave

- lat_D1 : latitud de la estación DME #1
- lon_D1: longitud de la estación DME #1

- lat_D2 : latitud de la estación DME #2
- lon_D2 : longitud de la estación DME #2

- R1 : Rango de funcionamiento del DME #1
- R2 : Rango de funcionamiento del DME #2

- PLOT: Puede vale '1' o '0' permitiendo elegir entre hacer un plot del escenario.
- fh: Puede valer '0' o el valor de figure handler en el que se quiera el plot. Si vale '0' la función crea su propia figura.

Salidas:

- ✓ alpha: Ángulo de intersección entre la aeronave y la pareja de DME's
- ✓ sigma: Precisión de la pareja de DME's respecto a la posición de la aeronave.
- ✓ in_out : Indica si la aeronave está dentro o no de la zona de cobertura válida
- ✓ validez : Indica si alpha, sigma, e in_out son adecuados

```
% Creación propia figura // Dejando posibilidad a que sea ploteada en
otra.
if strcmp(PLOT, 'P')
    if  (fh == 0)
       update_area = figure('Name', 'Check Cobertura');
figure(update_area);
    else
        figure(fh);
    end;
end

% Tiempo de espera y numero de puntos [DEPURACION]
t=0;    np = 200;        na = 200;

% Manejo de varias figuras:

% Conversion de unidades( NM -> Km )a los rangos:
NM2Km = 1.852;    Km2NM = 1/NM2Km;      R1 = R1 * NM2Km;       R2 = R2
* NM2Km;


% Asumiendo la simplificacion de 'Tierra Plana' : 1 grado latitud  ~
110 Km | 1 grado longitud ~ 111 Km
latDeg2Km = 110;             Km2LatDeg = 1/latDeg2Km;       lonDeg2Km
= 111;      Km2LonDeg = 1/lonDeg2Km;

% Trabaremos ahora en unidades de distancia (Km)
x_a  = lat_a  * latDeg2Km ;     y_a  = lon_a  * lonDeg2Km ;
% Aeronave
x_D1 = lat_D1 * latDeg2Km ;     y_D1 = lon_D1 * lonDeg2Km ;
% DME 1
x_D2 = lat_D2 * latDeg2Km ;     y_D2 = lon_D2 * lonDeg2Km ;
% DME 2


%% PLOT RANGOS ESTACIONES DME
% Vectores genereales X e Y con las coordenadas de los puntos de
interes
X0 = [x_a, x_D1, x_D2];        Y0 = [y_a, y_D1, y_D2];

% Circunferencias con centro en cada estación DME y radio igual a su
rango de funcionamiento:
a = linspace(0, 360, np);  NM2Km = 1.852;
% Circunferencia sobre el DME 1
x1 = R1*cosd(a) + x_D1;   y1 = R1*sind(a) + y_D1;  if strcmp('P',
PLOT)  plot(x1, y1, '-black', 'LineWidth',2); hold on; end
% Circunferencia sobre el DME 2
x2 = R2*cosd(a) + x_D2;   y2 = R2*sind(a) + y_D2;  if strcmp('P',
PLOT)  plot(x2, y2, '-black', 'LineWidth',2); hold on; end

% Circunferencias de intercepción con radio 'D': D es la distancia
entre los DME en Km
D = sqrt ( (x_D1 - x_D2 )^2 + (y_D1 - y_D2 )^2 );
%% PLOT AERONAVE Y DME's antes de regiones de intersección.
% Posiciones puntuales:
if strcmp('P', PLOT)
    for i = 1:3
        if      i == 1             % Posición Aeronave
```

```
            plot( X0(i), Y0(i),      '^red','LineWidth',4);         hold
on;
        else                          % Posiciones DME 1 y DME 2
            plot( X0(i), Y0(i),      '^blue', 'LineWidth',4);       hold
on;
        end
    end
end

% Segmento Aeronave - DME 1 - Prolongar un 10% de D en direccion
Aeronave -> DME
ms1  = (y_D1 - y_a)/(x_D1 - x_a);
if      x_a > x_D1   % Hay que incrementar la x hacia la izquierda
(negativamente)
    s1 = -1;
elseif  x_a == x_D1  % Solo hay que incrementar la y
    s1 = 0;
elseif x_a < x_D1    % Hay que incrementar la x hacia la derecha,
(positivamente)
    s1 = +1;
end
% Comprobacion pendiente infinita:
if ms1 ~= Inf && ms1 ~= -Inf
    b1 = y_a - ms1*x_a;    x_f1 = x_D1 + s1*(0.1)*D;        y_f1 =
ms1*(x_f1) + b1;
elseif ms1 == -Inf
    b1 = 0;                x_f1 = x_D1;                     y_f1 =
y_D1 - (0.1)*D;
elseif ms1 == Inf
    b1 = 0;                x_f1 = x_D1;                     y_f1 =
y_D1 + (0.1)*D;
end
if strcmp('P', PLOT) plot([x_a, x_f1],[y_a, y_f1], '--black',
'LineWidth',2);      hold on; end
% Segmento Aeronave - DME 2
ms2  = (y_D2 - y_a)/(x_D2 - x_a);
if      x_a > x_D2   % Hay que incrementar la x hacia la izquierda
(negativamente)
    s2 = -1;
elseif  x_a == x_D2  % Solo hay que incrementar la y
    s2= 0;
elseif x_a < x_D2    % Hay que incrementar la x hacia la derecha,
(positivamente)
    s2 = +1;
end
% Comprobacion pendiente infinita:
if ms2 ~= Inf && ms2 ~= -Inf
    b2 = y_a - ms2*x_a;    x_f2 = x_D2 + s2*(0.1)*D;        y_f2 =
ms2*(x_f2) + b2;
elseif ms2 == -Inf
    b2 = 0;                x_f2 = x_D2;                     y_f2 =
y_D2 - (0.1)*D;
elseif ms2 == Inf
    b2 = 0;                x_f2 = x_D2;                     y_f2 =
y_D2 + (0.1)*D;
end
if strcmp('P', PLOT) plot([x_a, x_f2],[y_a, y_f2], '--black',
'LineWidth',2);      hold on; end
if D>= R1+R2
    fprintf('Pareja de DMEs no válida, no hay zona de cobertura
común\n');
```

```
    fprintf('Distancia entre las estaciones: D = %g Km\n', D);
    fprintf('Rango de la estación 1: R1 = %g Km\n', R1);
fprintf('Rango de la estación 2: R2 = %g Km\n', R2);
    fprintf('La suma de ambos rangos cubre una distancia R1+R2 = %g
Km\n', R1+R2);
    fprintf('Quedando un espacio sin cobertura de dimensión: D-(R1+R2)
= %g Km\n', D -(R1+R2));
    alpha = 0; sigma = 0; in_out = 0; validez = 0; if strcmp(PLOT,
'P');    figure(gcf);  hold off;    end
    return;
elseif (x_D1 == x_D2 && y_D1== y_D2)
    fprintf('Error: Ambas estaciones DME no pueden tener las mismas
coordenadas\n');
    return;

elseif  (x_a == x_D1 && y_a == y_D1) ||  (x_a == x_D2 && y_a == y_D2)
    fprintf('Error: Aeronave situada en las mismas coordenadas que una
de las estaciones DME\n');
    if update_area;  close gcf; end
    return;
end

%% Circunferencias de intersección con radio D
D = sqrt ( (x_D1 - x_D2 )^2 + (y_D1 - y_D2 )^2 );
syms x0s y0s
ec1 = (x_D1 - x0s)^2 + (y_D1 - y0s)^2 - D^2;             ec2 = (x_D2 -
x0s)^2 + (y_D2 - y0s)^2 - D^2;
[x0s, y0s] = solve(ec1, ec2);                             x0n =
double(x0s);          y0n = double(y0s);
xi1 = D*cosd(a) + x0n(1);  yi1 = D*sind(a) + y0n(1);    if strcmp('P',
PLOT) plot(xi1, yi1, '-blue', 'LineWidth',2); hold on;  end
xi2 = D*cosd(a) + x0n(2);  yi2 = D*sind(a) + y0n(2);    if strcmp('P',
PLOT)plot(xi2, yi2, '-blue', 'LineWidth',2); hold on;   end
%% Circunferencias de actualizacion de radio mínimo:
Rmin = 1*NM2Km;
xm1 = Rmin*cosd(a) + x_D1;  ym1 = Rmin*sind(a) + y_D1;     if
strcmp('P', PLOT)  plot(xm1, ym1, '-red', 'LineWidth',2); hold on; end
xm2 = Rmin*cosd(a) + x_D2;  ym2 = Rmin*sind(a) + y_D2;     if
strcmp('P', PLOT)plot(xm2, ym2, '-red', 'LineWidth',2); hold on;   end

%% CALCULO del Ángulo de Interseccion alpha y de la precision sigma de
la pareja:

% Vector director unitario del segmento que va desde la aeronave hasta
el DME_1
vd1(1) = x_D1 - x_a;        vd1(2) = y_D1 - y_a;       u1 =
vd1/norm(vd1);
% Vector director unitario del segmento que va desde la aeronave hasta
el DME_2
vd2(1) = x_D2 - x_a;        vd2(2) = y_D2 - y_a;       u2 =
vd2/norm(vd2);

% Alpha como el ángulo formado entre ambos vectores directores
CosenoAlpha = ( abs( u1(1)*u2(1) +  u1(2)*u2(2)) ) / ( sqrt(u1(1)^2 +
u1(2)^2)* sqrt(u2(1)^2 + u2(2)^2) );
alpha = acosd(CosenoAlpha);   alpha = real(alpha);   % [°]   % La
formula siempre devuelve alpha

% Queremos el ángulo INTERIOR a los segmentos
% Necesitamos la distancia de la aeronave a cada estacion DME: en Km
```

```
d1 = sqrt ( (x_D1 - x_a)^2 + (y_D1 - y_a)^2 );  d2 = sqrt ( (x_D2 -
x_a)^2 + (y_D2 - y_a)^2 );   %Km
if D> max(d1, d2)
    alpha = 180 - alpha;
end

% Calculo de la precisión sigma de la pareja de DMEs:  OACI 8168 V2
pág 537
s_sis = 0.05 * NM2Km;
s1_air = max (0.085*NM2Km, 0.00125*d1);          s2_air = max
(0.085*NM2Km, 0.00125*d2);             %Km

sigma = sqrt( (s1_air^2 + s_sis^2) + (s2_air^2 + s_sis^2) )/( sind
(alpha) );
sigma = sigma*Km2NM;

%% REGIONES DE INTERSECCION:
% Intersección entre las circunferencias de rango máximo de cada
estación DME
from_2_into_1 = inpolygon(x2,y2,x1,y1);         from_1_into_2 =
inpolygon(x1,y1,x2,y2);
xri = [];   yri =[];
for i = 1 : length(x2)
    if      from_2_into_1(i)
                xri = [xri; x2(i)];     yri = [yri; y2(i)];
                %plot(x2(i), y2(i), 'oblack', 'LineWidth',2);
pause(t);
    elseif  from_1_into_2(i)
                xri = [xri; x1(i)];     yri = [yri; y1(i)];
                %plot(x1(i), y1(i), 'oblack', 'LineWidth',2);
pause(t);
    end
end
[xri, yri] = Arrange(xri, yri);
% Region anterior que además intersecta con la Circunferencia de
interseccion 1   : Región de actualizacion 1
xri1 = [];   yri1 =[];
check = inpolygon(xri,yri,xi1,yi1) ;
for i = 1 : length(xri)
    if      check(i)
                xri1 = [xri1; xri(i)];     yri1 = [yri1; yri(i)];
                %plot(xri(i), yri(i), 'ogreen', 'LineWidth',2);
pause(t);
    end
end
check = inpolygon(xi1,yi1,xri,yri) ;
for i = 1:length(xi1);
    if  check(i)
                xri1 = [xri1; xi1(i)];     yri1 = [yri1; yi1(i)];
                %plot(xi1(i), yi1(i), 'ogreen', 'LineWidth',2);
pause(t);
    end
end
[xri1, yri1] = Arrange(xri1, yri1);
if strcmp('P', PLOT)
    fill(xri1, yri1, 'green')
end
% Region anterior que además intersecta con la Circunferencia de
interseccion 2    : Región de actualización 2
xri2 = [];   yri2 =[];
check = inpolygon(xri,yri,xi2,yi2) ;
```

```
for i = 1 : length(xri)
    if      check(i)
                xri2 = [xri2; xri(i)];     yri2 = [yri2; yri(i)];
                %plot(xri(i), yri(i), 'ogreen', 'LineWidth',2);
pause(t);
    end
end
check = inpolygon(xi2,yi2,xri,yri) ;
for i = 1:length(xi2);
    if  check(i)
                xri2 = [xri2; xi2(i)];     yri2 = [yri2; yi2(i)];
                %plot(xi2(i), yi2(i), 'ogreen', 'LineWidth',2);
pause(t);
    end
end
[xri2, yri2] = Arrange(xri2, yri2);
if strcmp('P', PLOT)
    fill(xri2, yri2, 'green')
end

% Region de interseccion entre ambas circunferencias de interseccion:
NO actualizable
xri3 = [];   yri3 =[];
check = inpolygon(xi1,yi1,xi2,yi2) ;
for i = 1 : length(xi1)
    if      check(i)
                xri3 = [xri3; xi1(i)];     yri3 = [yri3; yi1(i)];
    end
end
check = inpolygon(xi2,yi2,xi1,yi1) ;
for i = 1:length(xi2);
    if  check(i)
                xri3 = [xri3; xi2(i)];     yri3 = [yri3; yi2(i)];
    end
end
[xri3, yri3] = Arrange(xri3, yri3);
if strcmp('P', PLOT)
    fill(xri3, yri3, 'white')
end
% Regiones definidas por las circunferencias de radio mínimo:
if strcmp('P', PLOT)
    fill(xm1, ym1, 'white');            fill(xm2, ym2, 'white');
plot( x_a, y_a, '^red','LineWidth',4);
end
[xri3, yri3] = Arrange(xri3, yri3);
% Comprobamos si la aeronave está dentro o fuera de la región de
actualización:
check1 = inpolygon(x_a, y_a, xri1, yri1);
check2 = inpolygon(x_a, y_a, xri2, yri2);        check3 =
inpolygon(x_a, y_a, xri3, yri3);
check4 = inpolygon(x_a, y_a, xm1,  ym1);         check5 =
inpolygon(x_a, y_a, xm2,  ym2);

if check3 || check4 || check5
    in_out = 0;
elseif  check1 || check2
    in_out = 1;
else
    in_out = 0;
end
```

```
%% Validez final de la cobertura, una vez se conocen alpha, sigma e
in_out:
if ((30<alpha && alpha<150)  && (in_out) && (sigma<5))   validez = 1;
else        validez = 0;       end
%% PLOT AERONAVE Y DME's
% Posiciones puntuales:
if strcmp('P', PLOT)
    for i = 1:3
        if      i == 1              % Posición Aeronave
            plot( X0(i), Y0(i),      '^red','LineWidth',4);    hold on;
        else                        % Posiciones DME 1 y DME 2
            plot( X0(i), Y0(i),      '^blue', 'LineWidth',4);  hold on;
        end
    end
end

% Segmento Aeronave - DME 1 - Prolongar un 40% de D en direccion
Aeronave -> DME
ms1  = (y_D1 - y_a)/(x_D1 - x_a);
if      x_a > x_D1   % Hay que incrementar la x hacia la izquierda
(negativamente)
    s1 = -1;
elseif  x_a == x_D1  % Solo hay que incrementar la y
    s1 = 0;
elseif x_a < x_D1    % Hay que incrementar la x hacia la derecha,
(positivamente)
    s1 = +1;
end
% Comprobacion pendiente infinita:
if ms1 ~= Inf && ms1 ~= -Inf
    b1 = y_a - ms1*x_a;    x_f1 = x_D1 + s1*(0.4)*D;        y_f1 =
ms1*(x_f1) + b1;
elseif ms1 == -Inf
    b1 = 0;                x_f1 = x_D1;                     y_f1 =
y_D1 - (0.4)*D;
elseif ms1 == Inf
    b1 = 0;                x_f1 = x_D1;                     y_f1 =
y_D1 + (0.4)*D;
end
if strcmp('P', PLOT) plot([x_a, x_f1],[y_a, y_f1], '--black',
'LineWidth',2);      hold on; end
% Segmento Aeronave - DME 2
ms2  = (y_D2 - y_a)/(x_D2 - x_a);
if      x_a > x_D2   % Hay que incrementar la x hacia la izquierda
(negativamente)
    s2 = -1;
elseif  x_a == x_D2  % Solo hay que incrementar la y
    s2= 0;
elseif x_a < x_D2    % Hay que incrementar la x hacia la derecha,
(positivamente)
    s2 = +1;
end
% Comprobacion pendiente infinita:
if ms2 ~= Inf && ms2 ~= -Inf
    b2 = y_a - ms2*x_a;    x_f2 = x_D2 + s2*(0.4)*D;        y_f2 =
ms2*(x_f2) + b2;
elseif ms2 == -Inf
    b2 = 0;                x_f2 = x_D2;                     y_f2 =
y_D2 - (0.4)*D;
elseif ms2 == Inf
```

```
    b2 = 0;                 x_f2 = x_D2;                        y_f2 =
y_D2 + (0.4)*D;
end
if strcmp('P', PLOT) plot([x_a, x_f2],[y_a, y_f2], '--black',
'LineWidth',2);      hold on; end

%% PLOT DEL ANGULO
if strcmp('P', PLOT)
       Beta1 = atand(ms1);          Beta2 = atand(ms2);
    % Ángulo inicial a utilizar:   9 casos posible
    X = [min(x_D1,x_D2), max(x_D1, x_D2)];          Y =
[min(y_D1,y_D2), max(y_D1, y_D2)];

   % Ecuación recta que pasa por ambas estaciones DME
   mDD  = (Y(2) - Y(1) )/(X(2) - X(1));   bDD = Y(1) - mDD*X(1);

   % Centroide del triángulo formado por ambas estaciones y la
aeronave:
    x_ct = (x_a + x_D1 + x_D2) /3;       y_ct = (y_a + y_D1 + y_D2)
/3;
    D_ct_a = sqrt(   (x_ct - x_a)^2   +   (y_ct - y_a)^2   );
   % Distancias a las estaciones DME
    d = [d1, d2];
    if      X(1) < x_a && x_a < X(2)  % RECTA CENTRAL
        RA = D/2;
        if X(1) == x_a && x_a == X(2)
                    B  = atand(mDD);                        a0 = B;
sa = 1;
        end
        if      (y_a == mDD*x_a + bDD) % sobre la recta:   alpha =
180°
                    B  = atand(mDD);                        a0 = B;
sa = 1;
        elseif y_a > mDD*x_a + bDD  % Por encima
                    B  = max(atand(ms1), atand(ms2));       a0 = B
+180;       sa = 1;

        elseif y_a < mDD*x_a + bDD    %Por debajo
                    B  = max(atand(ms1), atand(ms2));       a0 = B;
sa = 1;
        end
    elseif      X(2)<= x_a    % A la DERECHA de la recta
EN ELLO ESTAMOS
        if      min(d) == d(1)   RA = d1/2;  elseif  min(d) == d(2)
RA = d2/2;  end
        if      (y_a == mDD*x_a + bDD) % Sobre la recta:   alpha = 0°
                    B  = atand(mDD);                        a0 = B;
sa = 1;
        elseif y_a > mDD*x_a + bDD  % Por encima
                    B  = min(atand(ms1), atand(ms2));
                if x_a ~= X(2)          a0 = B +180;       sa = 1;
else    a0 = B;             sa = -1;        end

        elseif y_a < mDD*x_a + bDD    %Por debajo
               B  = min(atand(ms1), atand(ms2));
               if x_a ~= X(2)          a0 = B + 180;      sa = 1;  else
a0 = B + 180;             sa = -1;        end
        end
    elseif      x_a <= X(1)   % A la IZQUIERDA
        if      min(d) == d(1)   RA = d1/2;  elseif  min(d) == d(2)
RA = d2/2;  end
```

```
        if      (y_a == mDD*x_a + bDD) % Sobre la recta:   alpha = 0°
                    B  = atand(mDD);                          a0 = B;
sa = 1;
        elseif y_a > mDD*x_a + bDD  % Por encima
                    B  = min(atand(ms1), atand(ms2));
                    if x_a~= X(1)     a0 = B;            sa = 1; else
a0 = B;            sa = 1;        end

        elseif y_a < mDD*x_a + bDD    %Por debajo
                    B  = min(atand(ms1), atand(ms2));
                    if x_a~= X(1)       a0 = B;             sa = 1;
else      a0 = B;             sa = 1;    end
        end
    end
    af = a0 + (sa)*alpha;
    % Incremento angular en coordenadas polares
     a0r = a0 * pi/180;       afr = af * pi/180;              theta =
linspace(a0r, afr, na);       rho  = ones(1,na) * RA;

    % Transformación a coordenadas cartesianas
    [x,y] = pol2cart(theta,rho);        x = x + x_a;      y = y + y_a;
plot(x,y, '-red', 'LineWidth', 2);        hold on

    % Ángulo numerico como texto
    L  = length(x);     xt = x(L/2);                yt = y(L/2);
    s1 = '\alpha = ';   s2 = num2str(alpha);    string = [s1, s2,
'°'];
    text(xt,yt, string);     axis equal;

    %% ENFOQUE de los elementos : Tiene preferencia siempre el
trianguo formado por los DME y la aeronave
    X0 = [x_a, x_D1, x_D2];        Y0 = [y_a, y_D1, y_D2];
dprefs = [d1, d2, D];    dpref = max(dprefs);
    X = X0;                        Y = Y0;
    if R1 < 3.5*dpref         % Orden de magnitud de R1 es tal que
pueda graficarse sin afectar a la visualizacion del triangulo? SI, lo
añadimos
        X = [X, x1];    Y = [Y, y1];
    end
    if R2 < 3.5*dpref
        X = [X, x2];    Y = [Y, y2];
    end
    if D < 3.5*dpref
        X = [X, xi1, xi2];    Y = [Y, yi1, yi2];
    end

    x_min = min(X);     x_max = max(X);          y_min = min(Y);
y_max = max(Y);        axis([x_min x_max y_min y_max])

    grid off;    axis equal;     xlabel('Eje Y: Latitud en Km (1°~ 110
Km)','FontSize',20);    ylabel('Eje X: Longitud en Km (1°~ 111
Km)','FontSize',20);  hold off;

    % TEXTOS en gráfica: Cuadro resumen en la esquina superior derecha
    % alpha entre 30° y 150°:
    sa1 = '30° < \alpha < 150° : ';    if (30<alpha && alpha<150)  sa2
= ['\color{black}SI ', '\color{green} \surd'];  else       sa2 =
['\color{black}NO ', '\color{red} X'] ;   end
    sa = [sa1, sa2];
    % in_out
```

```
    si_o1 = '\color{black}Dentro-Fuera :  ';       if (in_out)  si_o2
= ['\color{black}DENTRO ', '\color{green} \surd'];  else si_o2 =
['\color{black}FUERA ','\color{red} X'];       end
    si_o = [si_o1, si_o2];
    % Precision sigma
    sig1 = '\color{black}Precisión \sigma :  ';     sig2 =
num2str(sigma); sig3 = ' NM';   if sigma<5 sig4 =['\color{green}
\surd']; else sig4 =['\color{red} X']; end
    sig = [sig1, sig2, sig3, sig4];
    % Evaluacion final de la cobertura:
    fin1 = '\color{black}\bfCobertura Válida: ';
    if validez   fin2= '\color{black}SI';      else        fin2 =
'\color{black}NO';       end
    fin = [fin1, fin2];
    % Escritura de las lineas en el cuadro de texto
    annotation(gcf,'textbox','String',{sa,'',si_o,'', sig,'' ,fin},
...
   'FontSize',14,'units','pix', 'Position', [1500 700 400
200],'Interpreter','Tex', 'BackgroundColor', 'white');

    % LONGITUD AL EJE X  // LATITUD AL EJE Y
    view(-90,90)
    set(gca,'ydir','reverse')
2+2;
end
end

function [x_ord, y_ord] = Arrange(x, y)
for i = 1 : (length(x)-1)        % Pasamos por todos los puntos menos
el ultimo
        d = [];
    for j = (i+1): length(x)     % Anteriores ya ordenados
        d(j) = sqrt( (x(i) - x(j) )^2 + ( y(i) - y(j) )^2 );
    end
    % El punto siguiente más cercano tiene como índice imin:
    ipos = find(d>0);
    if isempty(ipos)
       break;
       fprintf('ipos está vacio\n');
       return;
    end
    [value,imin] = min(d(ipos));
    imin = imin + i;
    % Colócalo justo siguiendo al punto actual: intercambia posiciones
    save  = x(i+1);
    x(i+1)= x(imin);
    x(imin) = save;

    save  = y(i+1);
    y(i+1)= y(imin);
    y(imin) = save;
end
% Por si el punto final fuera problemático
x (end) = x(1);                  y(end) = y(1);
x_ord = x;      y_ord = y;
end
```

Ejemplos de uso *[DME_DME_Upgrade_area]*

```
[alpha,sigma,in_out, validez] = DME_DME_Upgrade_Area (10, 11,  12, 13,
120,  14, 13, 130, 'P', 0)
```

Objetivo 2: Estimación de la posición

5.3.14 - Solve_problem_MC_Weighted.m

Contexto *[Solve_problem_MC_Weighted]*

En este punto se dispone ya de todas las funciones necesarias para llevar a cabo la realización de los objetivos 2 y 3 del trabajo: **Estimar la posición de la aeronave y monitorizar el error cometido**.

Utilidad *[Solve_problem_MC_Weighted]*

Resuelve la posición de la aeronave: latitud y longitud (ALTURA NO) por el método de mínimos cuadrados ponderados, permitiendo monitorizar el error cometido.

Parámetros de entrada y salida *[Solve_problem_MC_Weighted]*

Entradas:

- *Rmax* : Radio máximo de búsqueda
- *t_w* : Intervalo de tiempo para el proceso de escritura en X-Plane
- *tolerancia*: criterio de convergencia.

Salidas:

- ✓ *xf_LLA_calculada: Array con la latitud y longitud calculadas estimadas.*
- ✓ *error_cometido: diferencia entre la estimación obtenida y la posición verdadera de la aeronave.*

Implementación *[Solve_problem_MC_Weighted]*

```
tic
% Asumiendo la simplificacion de 'Tierra Plana' : 1 grado
latitud  ~ 110 Km | 1 grado longitud ~ 111 Km
latDeg2Km = 110;   lonDeg2Km = 111;          Km2LatDeg =
1/latDeg2Km;    Km2LonDeg = 1/lonDeg2Km;

%Configuraciones de lectura - escritura referentes a X-Plane :
port_lectura    = 49000;
port_escritura  = 6500;

% 1) Lectura de la posición de la aeronave ¿Sentido..?

[position] = readXPLANE(port_lectura);
```

```
ft2m = 0.3048;
lat_0 = position.latitude;          %Grados centesimales
lon_0 = position.longitude;         %Grados centesimales
alt_0 = position.altitude * ft2m;   %m

x0_LLA_original = [lat_0, lon_0, alt_0]; % Posición de la
aeronave en sistema LLA

% Introducción de errores intencionados para provar la
convergencia del método:
% Los errores deben seguir una distribución normal de media 0:
lat_error = random('norm', 0, 1);   lon_error = random('norm',
0, 1);   Errores_intencionados = [lat_error, lon_error, 0];
%Posicion inicial 'desviada' intencionadamente
x0_LLA = x0_LLA_original + Errores_intencionados;

% 2) Obtención del conjunto de radioayudas de interés

% Radioayudas situadas a una distancia inferior a Rmax
[NavAid_Valid] = Search_NavAid(x0_LLA_original, Rmax, database,
lat_index_dat_file)

% Subconjunto filtrado de radioayudas anteriores (Cumplimiento
de otros requisitos)
[NavAid_Valid_Filtered] = Filter_NavAid(NavAid_Valid, x0_LLA)

% 3) Separación en distintos arrays de todos los datos
referentes al subconjunto de radioayudas filtradas:

[NAtype, NAlat, NAlon, NAalt,  NAfreq, NArange] =
Get_NavAid_Data(NavAid_Valid_Filtered);
NavAid_Total_filtered = length(NAtype);


% 4) Obtención de las lecturas de las radioayudas (DME_distance
de cada una de ellas)

% Proceso : Sintonización --> Lectura de 2 en 2 (Nav_1, Nav_2)

% Comprobamos si el numero de radioayudas filtradas es par o
impar
if      (-1)^(NavAid_Total_filtered) == 1
            Par_impar = 'par';
elseif (-1)^(NavAid_Total_filtered)  == -1
            Par_impar = 'impar';
end

DME_distance = [];      NM2m  = 1852;
% Si es impar, prescindimos de la última radioayuda:
NECESITAMOS estrictamente un numero par de radioayudas
if  (strcmp(Par_impar,'impar'))
    NavAid_Valid_Filtered = NavAid_Valid_Filtered(1:(end-1));
    NavAid_Total_filtered = NavAid_Total_filtered - 1;
```

```
    [NAtype, NAlat, NAlon, NAalt,  NAfreq, NArange] =
Get_NavAid_Data(NavAid_Valid_Filtered);
    disp('Se ha eliminado la última radioayuda para que el
número total sea par: ')
   NavAid_Valid_Filtered
end
% Una vez que el numero de radioayudas filtradas es par:

if (NavAid_Total_filtered < 4)
    error('myApp:argChk', 'No se han detectado al menos 4
radioayudas con las que operar');
end

j = 1;
for i = 1 : (NavAid_Total_filtered/2)
    %realizamos las lecturas por parejas, de 2 en 2
    WriteFrequencyNav1Nav2(NAfreq(j), NAfreq(j + 1), t_w);
% Sintonización
    r1 = Read_DME_distance(1);        r2 = Read_DME_distance(2);
% Lectura
    [DME_distance]= [DME_distance; r1*NM2m; r2*NM2m]; %m
    % Siguiente pareja:
    j = j + 2;
end

% Precision de cada radioayuda
alphas = []; sigmas = [];          s1_sis = 0.05*NM2m;
s2_sis = 0.05*NM2m;
for i = 1: (NavAid_Total_filtered/2)
    k = NavAid_Total_filtered -(i-1);
    % Calculamos alpha, sigma, in_out validez de la pareja: (Sin
PLOT) --> Matrid de pesos
    [alpha,sigma,in_out, validez] = DME_DME_Upgrade_Area (lat_0,
lon_0,  NAlat(i), NAlon(i), NArange(i),  NAlat(k), NAlon(k),
NArange(k), 0, 0);
    alphas(i) = alpha;        alphas(k) = alpha;
    sigma = sigma *NM2m;
    sigmas(i) = sigma;        sigmas(k) = sigma;

    % MATRIZ DE PESOS asociada
    W (i, i) = 1/((sigma)^2);
    W (k, k) = 1/((sigma)^2);
end


% DME_distance es la distancia absoluta: necesitamos la
proyección horizontal:
Horizontal_distance = [];
for i = 1 : NavAid_Total_filtered
    Nav_alt_i               = NAalt(i) * ft2m;
    Diferencia_alturas(i)   = Nav_alt_i - alt_0;
    h_distance              = sqrt( (DME_distance(i) )^2 - (
Diferencia_alturas(i) )^2 );
    Horizontal_distance     = [Horizontal_distance; h_distance];
end
```

```
% 5) RESOLUCIÓN POR MÍNIMOS CUADRADOS PONDERADOS :
    fprintf('Resolviendo por mínimos cuadrados
ponderados...\n'); disp(' ');
% Elegimos una radiobaliza de referencia ¿CRITERIO? -> Ninguno
de momento -> La primera de la lista

 % Conversión de LLA a ECEF

 x0_ECEF = lla2ecef(x0_LLA);

for i = 1 : NavAid_Total_filtered
    Nav_Aid_LLA  (i, 1:3) = [NAlat(i), NAlon(i), NAalt(i)*ft2m];
    Nav_Aid_ECEF (i, 1:3) = lla2ecef_2( Nav_Aid_LLA(i, 1:3) );
end

% Conversión de ECEF a NEU:

orgece = Nav_Aid_ECEF(1, 1:3);            % Radiobaliza de
referencia en ECEF (1ª de la lista de radioayudas)
orgllh = Nav_Aid_LLA (1, 1:3) * (pi/180); % Radiobaliza de
referencia en LLA ?

for i = 1 : NavAid_Total_filtered

Nav_Aid_NEU (i, 1:3) = ECEF2NEU (Nav_Aid_ECEF(i,1:3)', orgece',
orgllh');

end
x0_NEU = ECEF2NEU (x0_ECEF', orgece', orgllh');

% Proceso ITERATIVO de resolución en el marco NEU de la
radiobaliza de referencia:

convergencia            = 0;
numero_de_iteraciones   = 0;

while (1 - convergencia)
    numero_de_iteraciones = numero_de_iteraciones + 1 ;
% 1) Inicialización:
    x = x0_NEU(1:2)';      %Posicion actual como la ultima
solucion obtenida (x0_NEU va a actualizarse al final del bucle)

% 2) Previsión de la distancia a las radioayudas:

    for j = 1 : NavAid_Total_filtered

        rho_teo(j) = sqrt( (x(1)-Nav_Aid_NEU(j, 1) )^2 + ( x(2)-
Nav_Aid_NEU (j, 2) )^2);

       % 3 Cálculo del error en la medida:

        delta_rho(j) = Horizontal_distance(j) - rho_teo(j);
```

```
    % 4) Formmulación del problema de Mínimos Cuadrados:

        for k = 1:2
             G (j, k) = ( x(k) - Nav_Aid_NEU (j, k) ) /
rho_teo(j);
        end
    end

%delta_x = (inv((G'*G))*G')*delta_rho';
delta_x  =  inv(G'*W*G)*G'*W*delta_rho';

% 5) Actualización de la estimación de la posición inicial:

x0_NEU(1:2) = (x0_NEU(1:2) + delta_x);

% 6) Comprobacion de la convergencia de la solución:

        if sqrt( delta_x(1)^2 + delta_x(2)^2 ) < tolerancia
           convergencia = 1;
        end

        if numero_de_iteraciones > 100000
            fprintf('No se ha convergido después de %g
iteraciones\n', numero_de_iteraciones);
            convergencia = 1;
        end
end
% 7) Conversión  NEU -> ECEF -> LLA

%NEU -> ECEF

xf_ECEF = NEU2ECEF (x0_NEU, orgece', orgllh');

%ECEF -> LLA

xf_LLA = ECEF2LLA (xf_ECEF);

% Lat y lon a grados centesimales:
xf_LLA(1:2) = xf_LLA(1:2) * (180/pi);
xf_LLA_calculada = xf_LLA(1:2)';
% Error cometido:
x0 = x0_LLA_original(1); y0 = x0_LLA_original(2); xf =
xf_LLA_calculada(1); yf = xf_LLA_calculada(2);
error_cometido = sqrt( (latDeg2Km *(xf-x0))^2 + (lonDeg2Km*(yf-
y0))^2); %Km
error_cometido = error_cometido * 1/1.852;     % NM (<5 ?)

%% AVISO SI EL ERROR ES MAYOR QUE 5 NM

if error_cometido < 5
    fprintf('El sistema de navegación funciona correctamente:
SNE = %g NM\n', error_cometido)
else
```

```
    fprintf('El sistema de navegación NO funciona correctamente:
SNE = %g NM\n ', error_cometido)
end

%% INFO POR PANTALLA..
format short g
x0_LLA_original
x0_LLA

xf_LLA_calculada = xf_LLA(1:2)';
numero_de_iteraciones

toc
```

Ejemplos de uso *[Solve_problem_MC_Weighted]*

```
1) Desde Xplane: Situar la aeronave en el lugar deseado

2) [xf_LLA_calculada, error_cometido] = Solve_problem_MC_Weighted
(200, 0.5, 0.01, '3_12_13_Lat_Sorted_WO_R_ID_COOR.dat',
'LatIndex_3_12_13_Lat_Sorted_WO_R_ID_COOR.dat')
```

6.- Resultados

Consideraciones previas

A continuación se presentan, de manera esencialmente **gráfica**, los resultados que pueden obtenerse haciendo usos de las funciones desarrolladas durante el trabajo.

Debe tenerse en cuenta que lo aquí expuesto serán únicamente un par de **demostraciones**, enfocadas exclusivamente a probar el cumplimiento de los **objetivos** fundamentales del **proyecto**.

Dicho de otro modo, considero que la '**biblioteca' de funciones** desarrollada durante el trabajo tiene un **potencial de uso** que cubre muchas otras posibilidades. **Cualquier otro proyecto** que estuviera relacionado con **radioayudas** podría beneficiarse de utilidades como:

- *Poder leer, de **forma selectiva**, la base de datos de radioayudas de X-plane.*
- *Disponer de dicha base de radioayudas **ordenada** según un criterio de posición.*
- *Poder **buscar** determinadas radioayudas de interés **entorno a una posición.***
- *Poder disponer de un **fichero de radioayudas filtrado**, que contuviera únicamente las radioayudas situadas dentro de un determinado segmento de ruta.*

Lo que, creo, aporta un cierto 'valor añadido' al trabajo que quizás no pueda verse reflejado completamente en éste apartado. De ahí el inciso.

Viabilidad de la ruta de interés

NOTA: *Ver* *DEMO_Viabilidad_segmento_VM.m*
DEMO_Viabilidad_segmento_MB.m

Si, por razones de tipo aeronáuticas, nos interesara:

1. Conocer de qué estaciones DME podemos disponer para navegar una determinada ruta aérea.
2. Analizar la **calidad de la cobertura** que dichas estaciones DME ofrecen a lo largo de la ruta.

Las funciones:

- *Database_Segment_filter.m*
- *DME_DME_Upgrade_Area.m*

Han sido implementadas para cubrir dicho propósito.

Por ejemplo, analicemos la ruta *Valencia → Madrid → Barcelona.*

En primer lugar, dividimos la ruta en tantos **segmentos rectilíneos** sean necesarios, en este caso dos:

I. *Valencia → Madrid*
II. *Madrid → Barcelona*

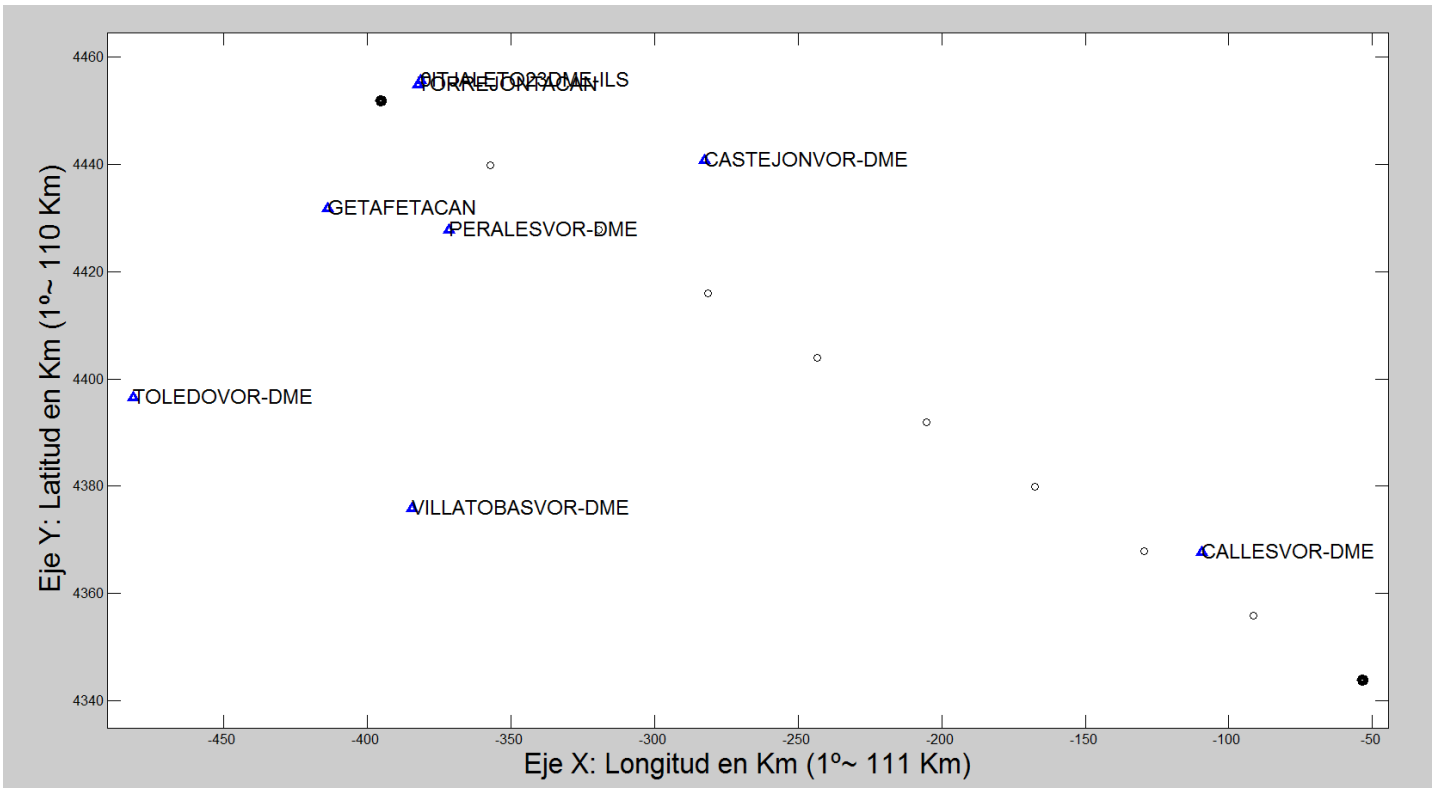
Ahora analicemos cada segmento por separado:

¿Qué estaciones DME hay disponibles en el 1er segmento de ruta?

Haciendo uso adecuado de la función *Database_Segment_filter.m:*

```
39.7071 -0.9863   1839 11755 130    0.0 CLS  CALLES VOR-DME
39.7807 -3.4641   2320 11270  60    0.0 VTB  VILLATOBAS VOR-DME
39.9694 -4.3374   1946 11320  50   -3.0 TLD  TOLEDO VOR-DME
40.2529 -3.3478   2513 11695 130    0.0 PDT  PERALES VOR-DME
40.2891 -3.7271   2029 11205 130    0.0 VGE  GETAFE TACAN
40.3718 -2.5446   3501 11560  27    0.0 CJN  CASTEJON VOR-DME
40.4988 -3.44548  2011 11250  25    0.0 TJZ  TORREJON TACAN
40.5066 -3.4353   2011 10950  18     0.200 ITJA LETO 23  DME-ILS
```

La posición de las mismas respecto al segmento de ruta I es la siguiente:

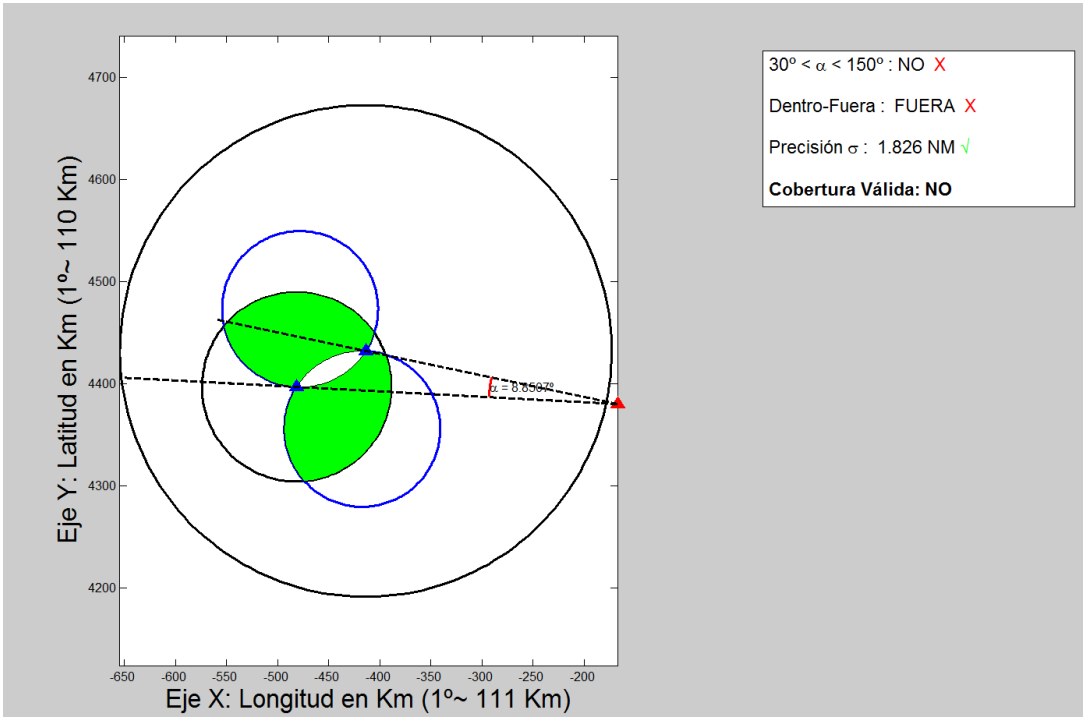


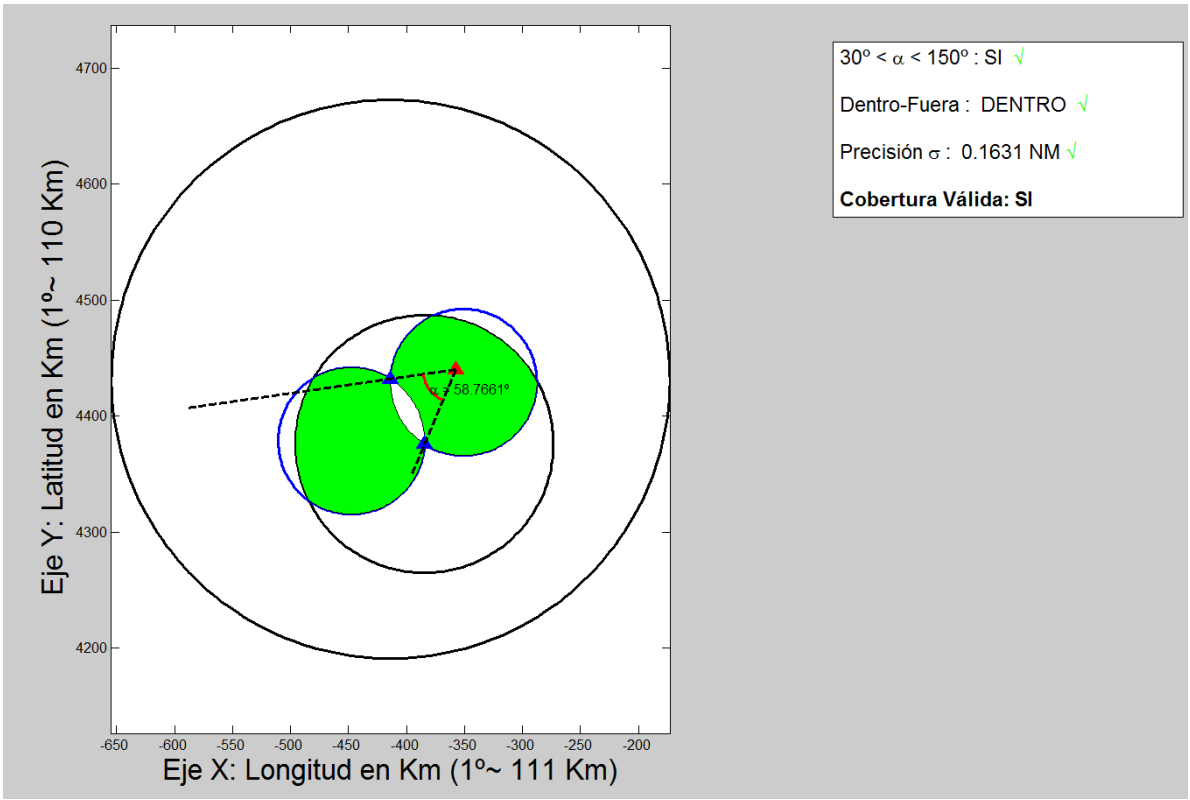
Ahora comenzamos a analizar, para cada uno de los puntos de ruta (en este ejemplo se han escogido 10, por razones de tiempo de visualización en la DEMO) si **existe alguna combinación** de estaciones DME que proporcione una cobertura adecuada.

Recordemos que en nuestro contexto de navegación RNAV 5, 'adecuada' significa que proporcione una precisión de **al menos 5 NM**.

Para comprobar si existe alguna combinación válida, emplearemos la función *DME_DME_Upgrade_Area.m* en una serie de bucles anidados.

Un par de imágenes al respecto:

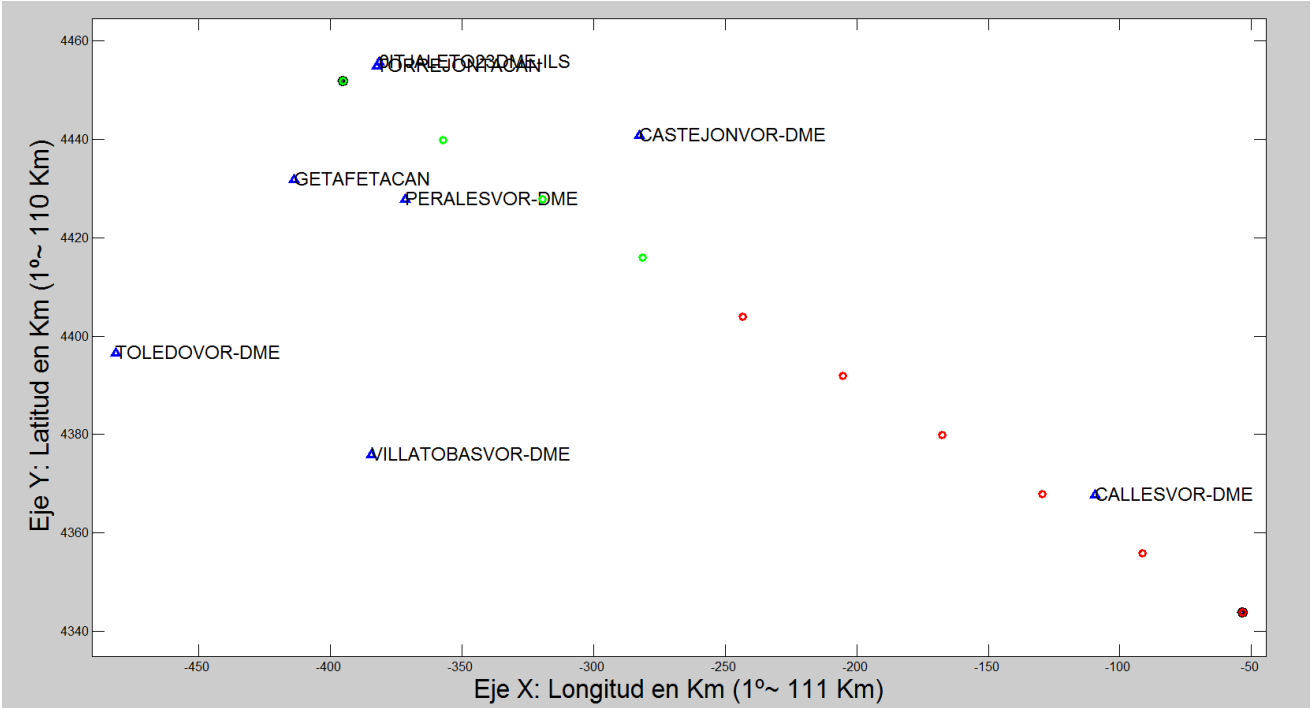




La serie de bucles comprueba todas las posibles combinaciones de radioayudas para cada punto de la ruta.

Si alguna de las combinaciones es válida, ese punto tiene cobertura DME válida y se pasa al siguiente. Sino, se siguen probando combinaciones hasta encontrar una válida, o hasta que se acaben. Si se acaban las combinaciones sin encontrar ninguna válida, ese punto de ruta no tiene cobertura válida.

Imagen del resultado final para el primer segmento:



El resultado es más o menos el que cabría esperar por inspección inicial: la geometría relativa de las estaciones DME con respecto a los primeros puntos de ruta no es buena, y el resultado confirma que dichos puntos carecen de cobertura DME válida.

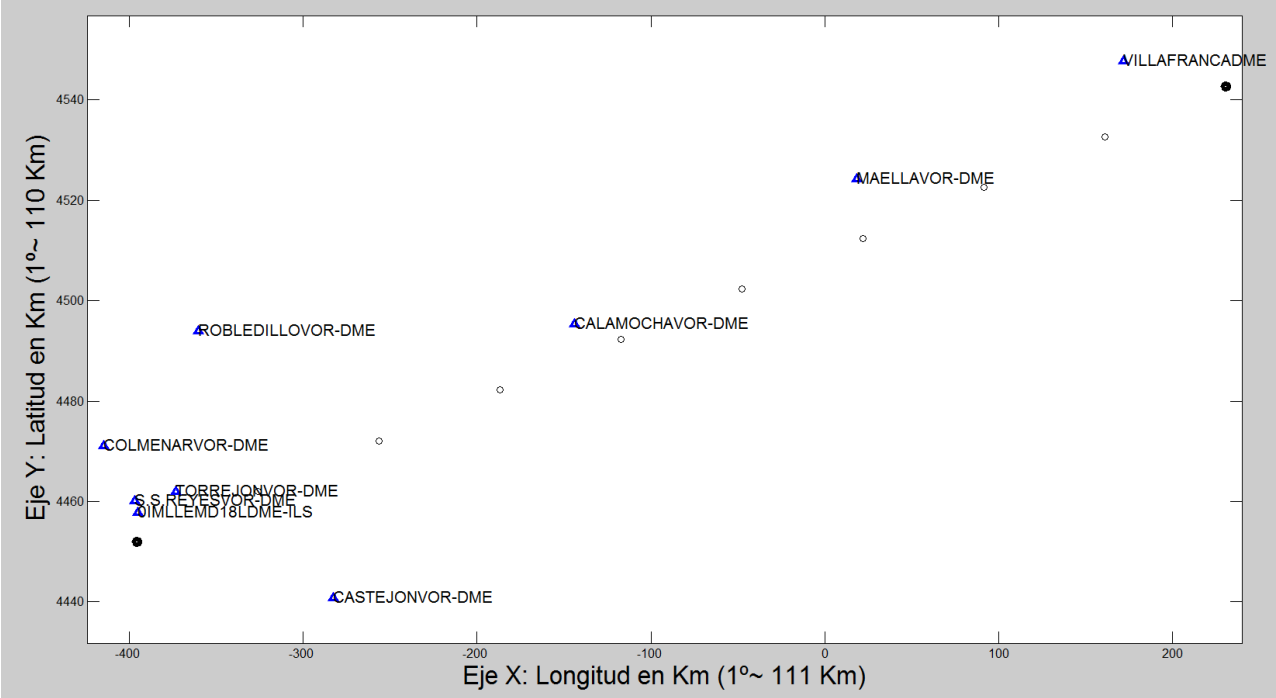
Conforme nos acercamos a Madrid, empiezan a aparecer más combinaciones que resulten en ángulos de intersección favorables (90º es el óptimo), resultando en la confirmación de que los últimos 4 puntos de ruta sí tienen cobertura DME válida.

Siguiendo un proceso análogo para el segmento II:

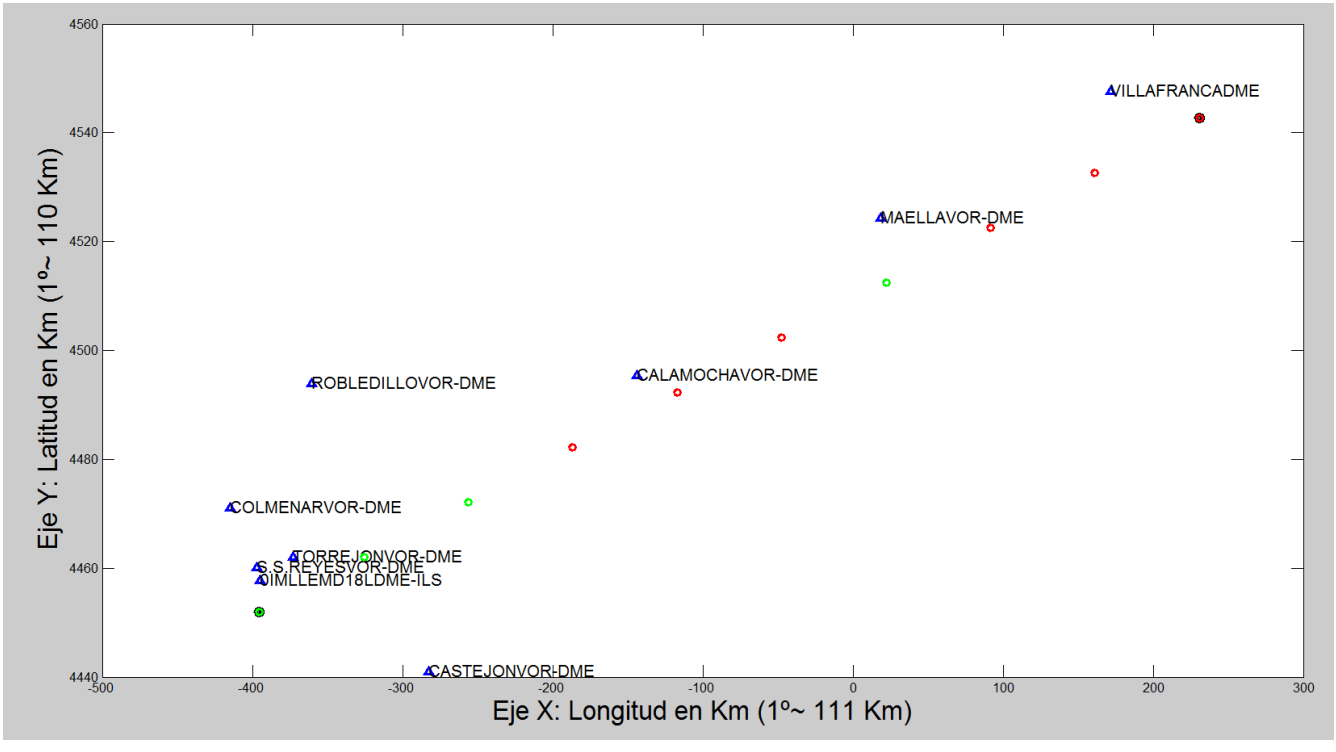
Estaciones DME disponibles:

```
40.37183 -2.5446   3501 11560  27    0.0 CJN  CASTEJON VOR-DME
40.52541 -3.5582   1916 11150  18    0.200 IML  LEMD 18L DME-ILS
40.54642 -3.5753   2012 11785  25    0.0 SSY  S.S. REYES VOR-DME
40.56401 -3.3632   2380 11510  25   -2.2 VJZ  TORREJON VOR-DME
40.64597 -3.7358   2690 11730 130   -3.0 CNR  COLMENAR VOR-DME
40.85387 -3.2466   3119 11395 130   -2.2 RBO  ROBLEDILLO VOR-DME
40.86723 -1.2979   3150 11600  40    0.0 CMA  CALAMOCHA VOR-DME
41.12959  0.1652   1184 11210  35   -2.0 MLA  MAELLA VOR-DME
41.34261  001.54   2165 11315 130    0.0 VLA  VILLAFRANCA DME
```

Disposición de las estaciones DME respecto a los puntos de ruta:



Resultados tras analizar la cobertura en cada punto de ruta:



Estimación de la posición

Una demostración de la utilidad de las funciones desarrolladas para estimar la posición de la aeronave consiste simplemente en ejecutar la función:

- *Solve_problem_MC_Weighted.m*

Ejecutando X-plane, y situando la aeronave en el aeropuerto de Valencia, obtenemos:

```
Radioayudas situadas a menos de 200 Km de la posición de la aeronave :

NavAid_Valid =

    '3  38.85386100 -003.97080600   2165 11430  20   -2.0 ADQ  CIUDAD REAL VOR-DME'
    '12  39.78076100 -003.46413600   2320 11270  60    0.0 VTB  VILLATOBAS VOR-DME'
    '3  39.96945600 -004.33746900   1946 11320  50   -3.0 TLD  TOLEDO VOR-DME'
    '12  40.25290000 -003.34785600   2513 11695 130    0.0 PDT  PERALES VOR-DME'
    '13  40.28918100 -003.72710000   2029 11205 130    0.0 VGE  GETAFE TACAN'
    '12  40.36856100 -004.24933900   2516 11495 130    0.0 NVS  NAVAS VOR-DME'
    '12  40.37183100 -002.54466400   3501 11560  27    0.0 CJN  CASTEJON VOR-DME'
    '12  40.46570300 -003.55514200   1981 10990  18      0.200 MAA  LEMD 33L DME-ILS'
    '13  40.46902800 -003.55766700   1969 11645 130    0.0 BRA  BARAJAS DME'
    '3  40.46913900 -003.55752800   1969 11645 130   -2.0 BRA  BARAJAS VOR-DME'
    '12  40.47617500 -003.53727500   1886 10910  18      0.200 MBB  LEMD 33R DME-ILS'
    '13  40.49889400 -003.44548900   2011 11250  25    0.0 TJZ  TORREJON TACAN'
    '12  40.50661400 -003.43536400   2011 10950  18      0.200 ITJA LETO 23  DME-ILS'
    '12  40.51996900 -003.57383900   1986 11070  18      0.200 IMR  LEMD 18R DME-ILS'
    '12  40.52541900 -003.55822200   1916 11150  18      0.200 IML  LEMD 18L DME-ILS'
    '12  40.54642200 -003.57536900   2012 11785  25    0.0 SSY  S.S. REYES VOR-DME'
    '3  40.56408600 -003.36033300   2380 11510  25   -2.2 VJZ  TORREJON VOR-DME'
    '3  40.64597800 -003.73584400   2690 11730 130   -3.0 CNR  COLMENAR VOR-DME'
    '3  40.85387200 -003.24663100   3119 11395 130   -2.2 RBO  ROBLEDILLO VOR-DME'
    '12  41.15168100 -003.60484700   5659 11540 130    0.0 SIE  SOMOSIERRA VOR-DME'
    '13  41.70411100 -004.84980600   2787 10820 130    0.0 TVD  VALLADOLID TACAN'
    '12  42.02760800 -004.10913100   3002 11675  40    0.0 NEA  TABANERA VOR-DME'
```

```
Filtro #1 Activo : Subconjunto de radioayudas en rango de actuación válido


NavAid_Valid_Filtered =

    '12  40.25290000 -003.34785600   2513 11695 130    0.0 PDT  PERALES VOR-DME'
    '13  40.28918100 -003.72710000   2029 11205 130    0.0 VGE  GETAFE TACAN'
    '13  40.46902800 -003.55766700   1969 11645 130    0.0 BRA  BARAJAS DME'
    '3   40.46913900 -003.55752800   1969 11645 130   -2.0 BRA  BARAJAS VOR-DME'
    '3   40.64597800 -003.73584400   2690 11730 130   -3.0 CNR  COLMENAR VOR-DME'
    '3   40.85387200 -003.24663100   3119 11395 130   -2.2 RBO  ROBLEDILLO VOR-DME'
    '12  41.15168100 -003.60484700   5659 11540 130    0.0 SIE  SOMOSIERRA VOR-DME'
Resolviendo por mínimos cuadrados ponderados...

El sistema de navegación funciona correctamente: SNE = 0.0301828 NM

x0_LLA_original =

        40.485       -3.5757        1.0689


x0_LLA =

        41.022       -1.7419        1.0689


numero_de_iteraciones =

     7

Elapsed time is 8.153280 seconds.

xf_LLA_calculada =

       40.485      -3.5755


error_cometido =

      0.030183
```

En éste caso vemos que el proceso general ha sido bastante bueno:

- El **error** en la estimación es muy inferior a 5 NM.
- Únicamente hicieron falta 7 **iteraciones** para alcanzar tolerancia deseada.
- En consecuencia, el **tiempo de ejecución** es relativamente pequeño.

En otras situaciones, donde la geometría relativa de las radioayudas filtradas fuera más complicada, los tres parámetros anteriores podrían verse incrementados.

De cualquier modo, el error cometido está siendo monitorizado, y la función avisaría si el éste fuera superior a 5 NM.

7.- Bibliografía

A continuación se adjunta una lista con los documentos de referencia empleados durante la elaboración del trabajo. Todos ellos se encuentran disponibles en formato .pdf dentro de la carpeta 'Bibliografía' en el CD del trabajo.

Documentos .pdf

- *X –Plane Navigation Data (nav.dat & Earth_nav.dat) file specification*
- *Métodos de determinación de posición Rho – Rho*
- *DOC – OACI- 8168: Operación de aeronaves – Volumen II – 5º edición*
- *DOC – OACI -9613: Performance-based Navigation Manual – 4º edición*

Por supuesto, también se ha hecho uso intensivo de Internet para búsquedas de diversa índole, mereciendo la pena resaltar al menos los siguientes enlaces web:

Enlaces web

- *https://en.wikipedia.org/wiki/Least_squares#Weighted_least_squares*
- *http://stackoverflow.com*
- *http://es.mathworks.com*