Documentación de la Simulación de Tráfico Aéreo

Introducción

Este documento explica de manera sencilla el funcionamiento del código de simulación de tráfico aéreo, detallando los cálculos principales, el propósito de cada función y el uso de las librerías empleadas.

Librerías Utilizadas

- **numpy**: Para cálculos matemáticos y generación de números aleatorios.
- **matplotlib.pyplot** y **matplotlib.animation**: Para graficar el mapa y animar el movimiento de los aviones.
- **cartopy.crs** y **cartopy.feature**: Para crear mapas geográficos y añadir características como costas y fronteras.
- **datetime**: Para registrar la fecha y hora de las colisiones.
- **matplotlib.offsetbox.AnnotationBbox, OffsetImage**: Para mostrar iconos de aviones en el mapa.
- **matplotlib.image**: Para cargar imágenes de los aviones.
- **matplotlib.widgets.Button**: Para crear botones interactivos en la interfaz gráfica.
- **tkinter, tkinter.messagebox**: Para mostrar mensajes emergentes de información y advertencia.
- **pymysql**: Para registrar y consultar colisiones en una base de datos MySQL.

Estructura del Código

1. Configuración Inicial

- El usuario ingresa el número de aviones y la velocidad de simulación.
- Se validan los valores para asegurar que sean positivos y estén dentro de un rango razonable.

2. Clase Aircraft

- Representa cada avión con atributos como identificador, posición, destino, velocidad y altitud.
- Método update_position: Calcula la nueva posición del avión usando trigonometría y conversiones de grados a kilómetros.
    - **Cálculo de distancia**: Usa la fórmula de distancia euclidiana en grados y la convierte a kilómetros considerando la latitud.
    - **Actualización de posición**: Avanza el avión hacia su destino proporcionalmente a la velocidad y el tiempo transcurrido.

3. Generación de Aviones

- Se crean objetos Aircraft con posiciones y destinos aleatorios dentro de un área geográfica definida (bounding box).

4. Detección de Colisiones

- Calcula la distancia entre dos aviones usando la fórmula:

distancia=(Δlat×111)2+(Δlon×111×cos(lat promedio))2distancia=(Δ*lat*×111)2+(Δ*lon*×111×cos(lat promedio))2

- Si la distancia es menor a 50 km, se considera una colisión.

5. Registro de Colisiones en MySQL

- Cuando ocurre una colisión, se guarda la información en una base de datos MySQL.
- Se muestra un mensaje de advertencia y se marca el punto de colisión en el mapa.

6. Configuración del Mapa

- Se utiliza Cartopy para crear un mapa con proyección PlateCarree.
- Se añaden características geográficas: tierra, océano, costas y fronteras.

7. Aviones e Iconos

- Se cargan y muestran iconos de aviones en el mapa en sus posiciones iniciales.
- Se añade texto con información de vuelo y se dibujan las trayectorias de cada avión.

8. Animación

- Se actualizan las posiciones de los aviones en cada frame de la animación.
- Se verifica si algún avión ha llegado a su destino o si ocurre una colisión.
- Se actualizan los iconos, textos y trayectorias en el mapa.

9. Mostrar Colisiones

- Se consulta la base de datos y se muestran las colisiones registradas en un mensaje emergente.

10. Inicio de la Simulación

- Se inicia la animación con Matplotlib.
- Se agrega un botón para mostrar las colisiones registradas.
- Se usa Tkinter para mostrar mensajes emergentes.

Explicación de los Cálculos

- **Conversión de velocidad**: De nudos a kilómetros por segundo: velocidad (km/s)=nudos×1.852/3600velocidad (km/s)=nudos×1.852/3600
- **Conversión de grados a kilómetros**:
    - 1 grado de latitud ≈ 111 km.
    - 1 grado de longitud ≈ 111 km × cos(latitud).
- **Distancia entre dos puntos**: Se usa el teorema de Pitágoras adaptado a la superficie terrestre.
- **Colisión**: Si la distancia calculada es menor a un umbral (50 km), se considera que los aviones han colisionado.

Uso de Cada Librería

Librería	**Propósito principal**
numpy	Cálculos matemáticos, generación de números aleatorios
matplotlib	Graficar y animar los aviones y sus trayectorias
cartopy	Crear mapas geográficos y añadir características como costas y fronteras
pymysql	Registrar y consultar colisiones en una base de datos MySQL
tkinter	Mostrar mensajes emergentes al usuario

1. Cálculo de la nueva posición de cada avión

Cada avión tiene una posición actual (latitud, longitud) y un destino (latitud, longitud). Para moverlo, el simulador hace:

a. Calcular la diferencia de posición:

- dlat=latitud destino−latitud actual*dlat*=latitud destino−latitud actual
- dlon=longitud destino−longitud actual*dlon*=longitud destino−longitud actual

Esto da el "vector" hacia el destino.

b. Calcular la distancia en grados:

- dist_deg=dlat2+dlon2*dist_deg*=*dlat*2+*dlon*2

c. Convertir la velocidad de nudos a kilómetros por segundo:

- Los aviones tienen una velocidad base en nudos (knots).
- Se multiplica por el **factor de velocidad** elegido por el usuario.
- Se convierte a km/h: velocidad (km/h)=nudos×1.852velocidad (km/h)=nudos×1.852
- Se convierte a km/s: velocidad (km/s)=km/h÷3600velocidad (km/s)=km/h÷3600

d. Calcular cuántos kilómetros avanzará el avión en el intervalo de tiempo:

- dist_km=velocidad (km/s)×delta_time (s)*dist_km*=velocidad (km/s)×delta_time (s)

e. Convertir grados a kilómetros:

- 1 grado de latitud ≈ 111 km.
- 1 grado de longitud
  ≈ 111×cos(latitud actual en radianes)111×cos(latitud actual en radianes) km.

f. Calcular el paso en grados para avanzar:

- step_lat=dlatdist_deg×dist_kmkm_deg_lat*step_lat*=*dist_degdlat*×*km_deg_latdist_km*
- step_lon=dlondist_deg×dist_kmkm_deg_lon*step_lon*=*dist_degdlon*×*km_deg_londist_km*

Esto da el avance en grados de latitud y longitud para el siguiente paso.

g. Actualizar la posición:

- Nueva latitud: latitud actual+step_latlatitud actual+*step_lat*
- Nueva longitud: longitud actual+step_lonlongitud actual+*step_lon*

Si el avión está muy cerca del destino (dist_deg<0.001*dist_deg*<0.001), se considera que llegó.

2. Cálculo de distancia entre dos aviones (para detectar colisión)

El simulador revisa si dos aviones están peligrosamente cerca:

a. Diferencia de posiciones:

- dlat=latitud avio´n 2−latitud avio´n 1*dlat*=latitud avio´n 2−latitud avio´n 1
- dlon=longitud avio´n 2−longitud avio´n 1*dlon*=longitud avio´n 2−longitud avio´n 1
- avg_lat=latitud avio´n 1+latitud avio´n 22*avg_lat*=2latitud avio´n 1+latitud avio´n 2

b. Convertir diferencias a kilómetros:

- d_km_lat=dlat×111*d_km_lat*=*dlat*×111
- d_km_lon=dlon×111×cos(avg_lat en radianes)*d_km_lon*=*dlon*×111×cos(*avg_lat* en radia nes)

c. Calcular la distancia real:

- distancia=d_km_lat2+d_km_lon2distancia=*d_km_lat*2+*d_km_lon*2

d. Comparar con el umbral de colisión:

- Si la distancia es menor a 50 km, se considera colisión.

3. Resumen del flujo de cálculos

1. **Cada avión** calcula su nuevo paso hacia el destino usando su velocidad y la geometría de la Tierra.
2. **Cada ciclo**, el simulador revisa todas las parejas de aviones para ver si están a menos de 50 km.
3. **Si hay colisión**, se registra en la base de datos y se notifica visualmente.

4. ¿Por qué estos cálculos?

- Se usan conversiones de grados a kilómetros porque las posiciones están en coordenadas geográficas (lat/lon), pero el "peligro" y el movimiento real se miden en kilómetros.
- Se usa el coseno de la latitud para ajustar la longitud, porque los meridianos se acercan en los polos.
- El umbral de 50 km es un valor de seguridad típico en simulaciones sencillas, aunque en la realidad puede variar según el espacio aéreo y la altitud.

5. Ejemplo numérico sencillo

Supón dos aviones:

- Avión 1: lat 10°, lon -80°
- Avión 2: lat 10.3°, lon -80.2°

a. Diferencias:

- dlat = 0.3
- dlon = -0.2
- avg_lat = 10.15°

b. En km:

- d_km_lat = 0.3 × 111 = 33.3 km
- d_km_lon = -0.2 × 111 × cos(10.15°) ≈ -0.2 × 111 × 0.984 = -21.85 km

c. Distancia:

- distancia = sqrt($33.3^2 + 21.85^2$) ≈ sqrt(1108 + 478) ≈ sqrt(1586) ≈ 39.85 km

d. Resultado:

- Como 39.85 km < 50 km, **hay colisión**.

Estos cálculos, aunque aproximados, son estándar en simulaciones de tráfico aéreo para modelar el movimiento y la interacción entre aeronaves de manera eficiente y comprensible.