ANALIZA RUCHU LOTNICZEGO ORAZ BADANIE POTENCJALNYCH KOLIZJI

Adam Nowak

Uniwersytet Warszawski

Warszawa

25.06.2024

Spis treści

1. Cel projektu

Celem projektu była analiza ruchu lotniczego oraz badanie potencjalnych kolizji. Do zadania wykorzystano bazę danych „fsig-raw” dostępną na serwerze fsig-students.ocean.icm.edu.pl.

Program został napisany w języku Python ze względu na dużą ilość danych. Na serwerze użyto wersji 2.7, natomiast po uzyskaniu danych lokalnie wykorzystano wersję 3.11 oraz środowisko Jupyter Notebook.

2. Przegląd danych oraz obróbka danych

Początkowo skupiono się na przeanalizowaniu dostępnych danych. Każdy zestaw zawiera kolejno pola: pitr, type, ident, alt, clock, lon, lat oraz inne istotne zmienne (Rysunek 1).


```
> db.rawFaSignals.find().limit(2);
{ "_id" : ObjectId("5a497a04870c008695942e4d"), "key" : "1514764800#SJY590#72592
    fc3bcb6e36761fb4982e3bda1dc#v2", "value" : { "pitr" : "1514764800", "type" : "position", "ident" : "
    SJY590", "alt" : "5300", "clock" : "1514764792", "facility_hash" : "910
    f66749f1ce9655b9f41ce7850e239974cd798", "facility_name" : "FlightAware ADS-B", "id" : "SJY590
    -1514584800-schedule-0001", "gs" : "213", "heading" : "30", "hexid" : "8A0670", "lat" : "-5.33489", "
    lon" : "119.40396", "updateType" : "A", "altChange" : "D", "air_ground" : "A" } }
{ "_id" : ObjectId("5a497a04870c008695942e48"), "key" : "1514764800#CSH9141#13502
    bb3289128da9b3ce044cfca818c#v2", "value" : { "pitr" : "1514764800", "type" : "position", "ident" : "
    CSH9141", "alt" : "26125", "clock" : "1514764791", "facility_hash" : "376
    de0ee67533952227347fa52bebdcaf1f7dece", "facility_name" : "FlightAware ADS-B", "id" : "CSH9141
    -1514588100-schedule-0000", "gs" : "371", "heading" : "0", "hexid" : "78057B", "lat" : "36.89665", "
    lon" : "120.76399", "reg" : "B5260", "squawk" : "6347", "updateType" : "A", "altChange" : "D", "
    air_ground" : "A" } }
```


Rysunek 1 Przegląd danych

Z zestawu o parametrze type: flight plan początkowo wyciągnięto identyfikatory lotów (ident) oraz rejestracje samolotu (reg) (Rysunek 2). Otrzymano w ten sposób plik typu JSON, w którym zapisano kolejne rekordy z danymi (Rysunek 3).

```python
if ident and pitr:
    # Kryteria wyszukiwania na podstawie ident i pitr
    criteria = {
        "$and": [
            {"value.pitr": pitr},
            {"value.type": "flightplan"},
            {"value.ident": ident},
        ]
    }
```

Rysunek 2 Wycinek kodu, kryteria wyszukiwania


```
{"ident": "VOZ214", "reg": "VHVUK", "pitr": "2018-01-01 01:00:00"}
{"ident": "THY1", "reg": "TCLJF", "pitr": "2018-01-01 01:00:00"}
{"ident": "QFA79", "reg": "VHQPJ", "pitr": "2018-01-01 01:00:00"}
{"ident": "AAL2254", "reg": "N173AN", "pitr": "2018-01-01 01:00:00"}
{"ident": "KAL124", "reg": "HL8003", "pitr": "2018-01-01 01:00:00"}
{"ident": "SWA622", "reg": "N8711Q", "pitr": "2018-01-01 01:00:00"}
{"ident": "AAL1203", "reg": "N324RA", "pitr": "2018-01-01 01:00:00"}
{"ident": "UAL595", "reg": "N815UA", "pitr": "2018-01-01 01:00:00"}
{"ident": "SWA4193", "reg": "N441WN", "pitr": "2018-01-01 01:00:00"}
```


Rysunek 3 Dane zawierające identyfikator lotu

Po otrzymaniu identyfikatorów lotu zdecydowano się wyszukać 15 lotów o otrzymanych nazwach, lecz z datą 02-01-2018. Loty 1 stycznia często nie posiadały punktów startowych, ponieważ od początku znajdowały się w powietrzu. Dla znalezionych identyfikatorów lotu wyciągnięto dane z typu „Position", takie jak: identyfikator, numer rejestracyjny samolotu, długość geograficzną, szerokość geograficzną oraz wysokość. Czas PITR w tym przypadku nie był potrzebny ze względu na czas clock, który pozwalał określić czas danego lotu. Parametr air_ground, określający, czy samolot jest w powietrzu lub na ziemi, również nie był potrzebny ze względu na posiadaną wysokość (Rysunek 4).

```
output = {
    "ident": ident,
    "pitr": pitr,
    "air_ground": air_ground,
    "lat": lat,
    "lon": lon,
    "alt": alt,
    "clock": clock,
    "reg": reg,
}
```

```
"ident": "AAL639",
"reg": "N573UW",
"waypoints": [
  {
    "lat": 33.43014,
    "lon": -112.04216,
    "alt": 2075,
    "clock": "2018-01-01 23:20:50"
  },
```

Rysunek 4 Pobranie niezbędnych danych oraz ich obróbka

W ten sposób otrzymano 15 plików JSON z lotami zawierających niezbędne informacje.

3. Przygotowanie danych do programu

Przed rozpoczęciem analizy należało zmodyfikować format danych w plikach i zmergować je w jeden plik. W tym celu napisano krótki program, który najpierw ujednolicał zapis w plikach, a następnie scalał je w jeden plik (Rysunek 5).

```
# Utwórz docelowy obiekt JSON
output_data = {
    "ident": ident,
    "reg": reg,
    "waypoints": waypoints
}
```

```
# Przetwarzanie każdego pliku
for filename in files_to_process:
    file_path = os.path.join(folder_path, filename)

    # Wczytaj dane z pliku JSON
    with open(file_path, 'r') as file:
        try:
            # Parsuj dane z pliku JSON
            data = json.load(file)

            # Dodaj dane do listy merged_data
            merged_data.append(data)
```

Rysunek 5 Dalsza obróbka oraz mergowanie danych

4. Główny program

Program składa się z części wczytania bibliotek, załadowania danych, trzech funkcji, wywołania funkcji oraz przedstawienia wyników zarówno graficznie, jak i tekstowo.

4.1 Biblioteki

- JSON - praca z danymi w formacie JSON (wczytywanie i zapisywanie)
- OS - operacje na plikach i ścieżkach (sprawdzanie istnienia plików)
- GEOPY - obliczenia odległości na powierzchni Ziemi między dwoma punktami
- SHAPEL - reprezentowanie i manipulacja punktami geograficznymi
- DATETIME - operacje na datach i czasach (parsowanie, formatowanie)
- FOLIUM - tworzenie interaktywnych map z zaznaczonymi punktami i liniami
- ITERTOOLS - tworzenie cyklicznych iteratorów
- MATH – obliczenia matematyczne

4.2 Wczytanie danych

Dane zostały wczytane z poprzednio utworzonego pliku zawierającego zestawy lotów.

```python
import json
import os
from geopy.distance import great_circle
from shapely.geometry import Point
from shapely.geometry import LineString
from datetime import datetime
import folium
from itertools import cycle
from datetime import datetime
from math import sqrt, pow, inf
```

```python
# Ścieżka do pliku JSON
file_path = r'A:\repo_git\aircraft_signals\dane\json\merged_output.json'

# Lista do przechowywania wszystkich obiektów JSON
data = []

# Wczytanie danych z pliku JSON
try:
    with open(file_path, 'r') as file:
        data = json.load(file)
except FileNotFoundError:
    print(f"Plik {file_path} nie został znaleziony.")
    exit(1)
except json.JSONDecodeError as e:
    print(f"Błąd podczas parsowania JSON: {str(e)}")
    exit(1)
```

Rysunek 6 Użyte biblioteki oraz wczytanie danych

4.3 Funkcje

1. **check_collisions(paths, idents)**: Funkcja ta sprawdza kolizje między trasami lotów na podstawie ich ścieżek reprezentowanych przez obiekty LineString. Iteruje po parach ścieżek, sprawdza ich przecięcia i znajduje najbliższe punkty przecięcia. Następnie używa funkcji **find_altitude_and_clock** do pobrania danych wysokościowych, czasowych oraz współrzędnych dla tych najbliższych (Rysunek 9).

2. **find_flight_data(ident)**: Funkcja ta służy do wyszukiwania danych lotu na podstawie identyfikatora. Przeszukuje ona dane lotów, aby znaleźć odpowiedni rekord na podstawie identyfikatora. Zwraca rejestrację lotu, wysokość pierwszego punktu docelowego oraz czas pierwszego punktu docelowego, jeśli dostępne są w danych(Rysunek 8Rysunek 8).

3. **find_altitude_and_clock(closest_point, ident, data)**: Funkcja ta znajduje punkt docelowy lotu o najbliższej odległości do podanego punktu. Iteruje po wszystkich punktach docelowych dla danego lotu, oblicza odległość od danego punktu za pomocą geometrii Shapely i zwraca wysokość, czas, oraz współrzędne geograficzne tego najbliższego punktu (Równanie: 1), (Rysunek 7).

```python
def find_altitude_and_clock(closest_point, ident, data):
    closest_waypoint = None
    min_distance = inf
    alt = 'Brak danych wysokości'
    clock = 'Brak danych zegarowych'

    for flight_data in data:
        if flight_data['ident'] == ident:
            waypoints = flight_data.get('waypoints', [])

            # Zmienne do przechowywania informacji o punkcie o najmniejszej odległości
            closest_wp_index = -1
            closest_wp_point = None

            # Iteracja po wszystkich punktach
            for index, wp in enumerate(waypoints):
                wp_lon = wp.get('lon')
                wp_lat = wp.get('lat')

                #Na podstawie najmniejszej różnicy odszukanie najbliższego punktu
                if wp_lon is not None and wp_lat is not None:
                    waypoint_point = Point(wp_lon, wp_lat)
                    distance = sqrt(pow(wp_lon - closest_point.x, 2) + pow(wp_lat - closest_point.y, 2))
                    #print(f"Punkt: {waypoint_point}, Odległość: {distance}")

                    # Aktualizacja najmniejszej odległości i punktu
                    if distance < min_distance:
                        min_distance = distance
                        closest_wp_index = index
                        closest_wp_point = waypoint_point

            # Pobranie danych w przypadku znalezienia punktu
            if closest_wp_index != -1:
                closest_waypoint = waypoints[closest_wp_index]
                alt = closest_waypoint.get('alt', 'Brak danych wysokości')
                clock = closest_waypoint.get('clock', 'Brak danych zegarowych')
                lon = closest_waypoint.get('lon', 'Brak danych lon')
                lat = closest_waypoint.get('lat', 'Brak danych lat')

                #print(f"Najbliższy punkt: {closest_wp_point}")
                break

    return alt, clock, lon, lat
```

Rysunek 7 Funkcja: find_altitude_and_clock

$$distance = \sqrt{\left(wp_{lon} - closest_{point_{lon}}\right)^2 + \left(wp_{lat} - closest_{point_{lat}}\right)^2} \quad (1)$$

```python
def find_flight_data(ident):
    for flight_data in data:
        if flight_data['ident'] == ident:
            reg = flight_data.get('reg', 'Unknown registration')
            waypoints = flight_data.get('waypoints', [])
            if waypoints:
                alt = waypoints[0].get('alt', 'No altitude data')
                clock = waypoints[0].get('clock', 'No clock data')
            else:
                alt = 'No altitude data'
                clock = 'No clock data'
            return reg, alt, clock

    return 'Unknown registration', 'No altitude data', 'No clock data'
```

Rysunek 8 Funkcja: find_flight_data

```python
def check_collisions(paths, idents):
    collisions = []

    for i in range(len(paths)):
        path1 = paths[i]
        ident1 = idents[i]
        reg1, _, _ = find_flight_data(ident1)

        for j in range(i + 1, len(paths)):
            path2 = paths[j]
            ident2 = idents[j]
            reg2, _, _ = find_flight_data(ident2)

            #Wyszukanie przecięcia tras
            if path1.intersects(path2):
                intersection = path1.intersection(path2)

                if intersection.is_empty or intersection.geom_type != 'Point':
                    continue

                intersection_point = (intersection.x, intersection.y)

                #Interpolacja najbliższych punktów na trasach
                closest_point1 = path1.interpolate(path1.project(intersection))
                closest_point2 = path2.interpolate(path2.project(intersection))


                # Odszukanie najbliższego punktu oraz jego parametru
                alt1, clock1, lon1, lat1 = find_altitude_and_clock(closest_point1, ident1,data)
                alt2, clock2, lon2, lat2 = find_altitude_and_clock(closest_point2, ident2,data)

                if clock1 != 'No clock data' and clock2 != 'No clock data':
                    # Obliczenie różnicy w czasie
                    time_format = "%Y-%m-%d %H:%M:%S"
                    time1 = datetime.strptime(clock1, time_format)
                    time2 = datetime.strptime(clock2, time_format)
                    time_diff = abs((time1 - time2).total_seconds())
                    time_diff_hours = int(time_diff // 3600)
                    time_diff_minutes = int((time_diff % 3600) // 60)
                else:
                    time_diff_hours = -1  # Placeholder
                    time_diff_minutes = -1  # Placeholder

                # Obliczenie różnicy wysokości
                if alt1 != 'No altitude data' and alt2 != 'No altitude data':
                    alt_diff = abs(int(alt1) - int(alt2))
                else:
                    alt_diff = -1  # Placeholder

                collisions.append((ident1, reg1, alt1, clock1,
                                   ident2, reg2, alt2, clock2,
                                   intersection_point[1], intersection_point[0],
                                   lon1, lat1,
                                   lon2, lat2,
                                   time_diff_hours, time_diff_minutes,
                                   alt_diff))

    return collisions
```

Rysunek 9 Główna funkcja: check_collisions

4.4 Wywołanie funkcji

Następnie, dla każdego lotu z danych, utworzono listy paths i idents, które zostały wypełnione trasami lotów oraz identyfikatorami tych lotów na podstawie punktów przejścia. Kolejno wywołano funkcję check_collisions(paths, idents), której wyniki, potencjalne kolizje tras lotów, zostały zapisane w liście collisions. Na zakończenie wyniki zostały wyświetlone na ekranie, przedstawiając informacje o potencjalnych kolizjach tras lotów oraz różnice w wysokości i czasie między nimi, jeśli takie zostały znalezione (Rysunek 10).

```
paths = []
idents = []

for flight in data:
    waypoints = flight.get('waypoints', [])
    points = [(wp['lon'], wp['lat']) for wp in waypoints if 'lat' in wp and 'lon' in wp]
    if points:
        line = LineString(points)
        paths.append(line)
        idents.append(flight['ident'])

# Displaying the results
collisions = check_collisions(paths, idents)

if collisions:
    print("Found potential collisions:")
    for collision in collisions:
        print(f"Flight {collision[0]} (Reg: {collision[1]}) and Flight {collision[4]} (Reg: {collision[5]})")
        print(f"Intersect at point ({collision[9]}, {collision[8]})")
        print(f"Closest point on Flight {collision[0]} trajectory: ({collision[10]}, {collision[11]}), Alt: {collision[2]}, Clock: {collision[3]}")
        print(f"Closest point on Flight {collision[4]} trajectory: ({collision[12]}, {collision[13]}), Alt: {collision[6]}, Clock: {collision[7]}")
        print(f"Altitude difference: {collision[16]} and time difference: {collision[14]} hours and {collision[15]} minutes")
        print()
else:
    print("No potential collisions found.")
```

Rysunek 10 Wywołanie funkcji

4.5 Wyniki oraz wnioski

Na podstawie wyników zidentyfikowanych kolizji tras lotów (Rysunek 11) można wyciągnąć następujące wnioski:

- W większości przypadków znaleziono loty, których trajektorie przecinały się, jednak różnice w czasie przelotu oraz wysokości były znaczne, co sugeruje dobrze zaplanowane i bezpieczne operacje lotnicze.

Wyjątki stanowią konkretne przypadki:

- Flight SWA622 (Reg: N8633A) and Flight AAL2254 (Reg: N177AN)
- Flight CBJ480 (Reg: B8550) and Flight QFA79 (Reg: VHQPI)
- Flight KAL124 (Reg: HL7586) and Flight QFA79 (Reg: VHQPI)
- Flight QFA79 (Reg: VHQPI) and Flight VOZ214 (Reg: VHYVC)
- Flight AAL2254 (Reg: N177AN) and Flight SWA622 (Reg: N8633A)

W pierwszych trzech przypadkach odnotowaną mała różnicę w pułapie, a samoloty minęły się w odstępie około dwóch godzin. W pozostałych dwóch przypadkach samoloty również minęły się w okolicy dwóch godzin, lecz znajdowały się w tych miejscach na różniących się wysokościach.

Jednakże, takie różnice w czasie przelotu są na tyle duże, że nie można tego traktować jako bezpośrednie zagrożenie.

```
Found potential collisions:
Flight SWA622 (Reg: N8633A) and Flight AAL2254 (Reg: N177AN)
Intersect at point (-79.98487707535958, 25.99567699112497)
Closest point on Flight SWA622 trajectory: (-79.96436, 25.98681), Alt: 30000, Clock: 2018-01-02 02:10:08
Closest point on Flight AAL2254 trajectory: (-79.98559, 25.99556), Alt: 15900, Clock: 2018-01-02 00:26:39
Altitude difference: 14100 and time difference: 1 hours and 43 minutes

Flight SWA622 (Reg: N8633A) and Flight DAL446 (Reg: N355NB)
Intersect at point (-94.11626131348235, 31.81216786066269)
Closest point on Flight SWA622 trajectory: (-94.14311, 31.82162), Alt: 36000, Clock: 2018-01-02 05:40:11
Closest point on Flight DAL446 trajectory: (-94.15028, 31.77556), Alt: 35000, Clock: 2018-01-02 22:01:01
Altitude difference: 1000 and time difference: 16 hours and 20 minutes

Flight AAL1632 (Reg: N170US) and Flight DAL446 (Reg: N355NB)
Intersect at point (-90.02745229745015, 34.999802856572956)
Closest point on Flight AAL1632 trajectory: (-89.96194, 35.0225), Alt: 30900, Clock: 2018-01-02 01:16:05
Closest point on Flight DAL446 trajectory: (-89.99667, 35.0225), Alt: 34900, Clock: 2018-01-02 22:35:44
Altitude difference: 4000 and time difference: 21 hours and 19 minutes

Flight AAL2254 (Reg: N177AN) and Flight DAL446 (Reg: N355NB)
Intersect at point (-73.471327869252, 40.484711358672875)
Closest point on Flight AAL2254 trajectory: (-73.48156, 40.49044), Alt: 1575, Clock: 2018-01-02 02:34:32
Closest point on Flight DAL446 trajectory: (-73.45528, 40.50167), Alt: 4000, Clock: 2018-01-03 00:42:31
Altitude difference: 2425 and time difference: 22 hours and 7 minutes

Flight AAL2254 (Reg: N177AN) and Flight JBU1197 (Reg: N796JB)
Intersect at point (-73.34799333743534, 39.609096176139836)
Closest point on Flight AAL2254 trajectory: (-73.34509, 39.60692), Alt: 13550, Clock: 2018-01-02 02:18:06
Closest point on Flight JBU1197 trajectory: (-73.35482, 39.60228), Alt: 35975, Clock: 2018-01-01 23:23:05
Altitude difference: 22425 and time difference: 2 hours and 55 minutes

Flight AAL2254 (Reg: N177AN) and Flight SWA622 (Reg: N8633A)
Intersect at point (-79.98487707535958, 25.99567699112497)
Closest point on Flight AAL2254 trajectory: (-79.98559, 25.99556), Alt: 15900, Clock: 2018-01-02 00:26:39
Closest point on Flight SWA622 trajectory: (-79.96436, 25.98681), Alt: 30000, Clock: 2018-01-02 02:10:08
Altitude difference: 14100 and time difference: 1 hours and 43 minutes

Flight CBJ480 (Reg: B8550) and Flight QFA79 (Reg: VHQPI)
Intersect at point (145.45008631493425, -26.824897875714065)
Closest point on Flight CBJ480 trajectory: (145.463, -26.84095), Alt: 36000, Clock: 2018-01-02 02:38:43
Closest point on Flight QFA79 trajectory: (145.45085, -26.79337), Alt: 36000, Clock: 2018-01-02 01:10:07
Altitude difference: 0 and time difference: 1 hours and 28 minutes

Flight DAL446 (Reg: N355NB) and Flight SWA622 (Reg: N8633A)
Intersect at point (-94.11626131348235, 31.81216786066269)
Closest point on Flight DAL446 trajectory: (-94.15028, 31.77556), Alt: 35000, Clock: 2018-01-02 22:01:01
Closest point on Flight SWA622 trajectory: (-94.14311, 31.82162), Alt: 36000, Clock: 2018-01-02 05:40:11
Altitude difference: 1000 and time difference: 16 hours and 20 minutes

Flight KAL124 (Reg: HL7586) and Flight QFA79 (Reg: VHQPI)
Intersect at point (143.13333496561637, 5.9372524446306185)
Closest point on Flight KAL124 trajectory: (142.9, 6.38333), Alt: 38000, Clock: 2018-01-02 04:16:27
Closest point on Flight QFA79 trajectory: (143.25, 5.45), Alt: 38000, Clock: 2018-01-02 05:18:53
Altitude difference: 0 and time difference: 1 hours and 2 minutes

Flight QFA79 (Reg: VHQPI) and Flight VOZ214 (Reg: VHYVC)
Intersect at point (144.80957271105592, -37.516800457486404)
Closest point on Flight QFA79 trajectory: (144.81897, -37.50155), Alt: 12925, Clock: 2018-01-01 23:43:53
Closest point on Flight VOZ214 trajectory: (144.8119, -37.52948), Alt: 2975, Clock: 2018-01-02 01:22:24
Altitude difference: 9950 and time difference: 1 hours and 38 minutes
```

Rysunek 11 Wyniki - potencjalne kolizje

4.6 Wizualizacja wyników

Wynik graficzny (Rysunek 12) został załączony jako interaktywny plik o nazwie:

„flight_route_map_with_collisions.html”.

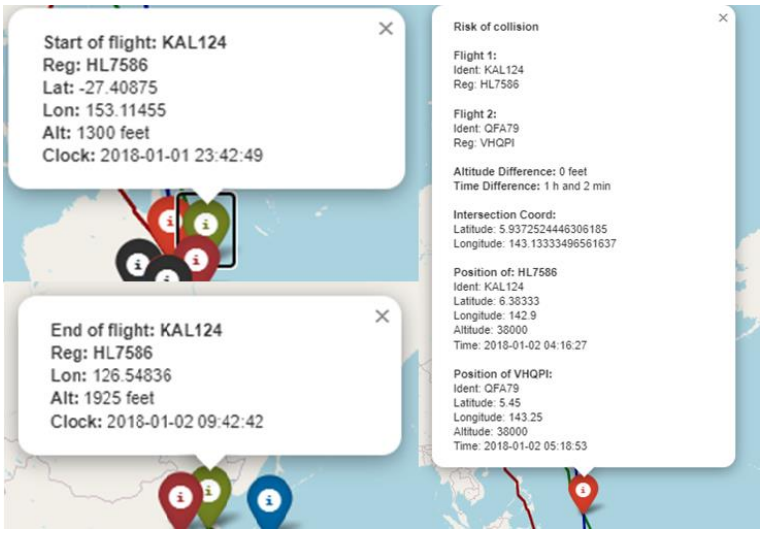
Rysunek 12 Wizualizacja wyników

Objaśnienia (Rysunek 13):

Każda trasa posiada znacznik początkowy oraz końcowy a także ścieżkę oznaczoną tym samym kolorem co markery.

Punkty kolizji są oznaczone kolorem CZERWONYM.

Po najechaniu na danych znacznik zostaną wyświetlone informacje:



Rysunek 13 Objaśnienia

5. Spis ilustracji