Sistema de Correção de Rotas para Mobilidade Aérea Urbana

Autor[1]: Kevin Matthias Franca Fialho Cerqueira
Autor[2]: Rui Leme Padilha Junior
Orientador[3]: Emerson Santos de Oliveira

Resumo. *Esse trabalho foi desenvolvido com a finalidade de contribuir para a resolução de problemas relacionados à mobilidade aérea urbana, tendo como objetivo principal o desenvolvimento de um sistema capaz de calcular rotas e corrigir uma das rotas quando houver um possível ponto de colisão entre essas. O sistema foi desenvolvido em Python, utilizando bibliotecas como a Geopandas, a qual foi essencial para a construção do mesmo.*
Palavras-Chave: Mobilidade Aérea Urbana; Correção de Rotas; Python.

Abstract. *This work was developed with the purpose of contributing to the resolution of problems related to urban air mobility, with the main objective of developing a system capable of calculating routes and correcting one of the routes when there is a possible point of collision between them. The system was developed in Python, using libraries such as Geopandas, which was essential for its construction.*
Key-words: Urban Air Mobility; Route Correction; Python.

1. INTRODUÇÃO

O transporte é a chave para o desenvolvimento das cidades, mas apesar disso, é responsável por externalidades negativas que impactam na economia e na qualidade de vida das pessoas, que somam mais de 80% nas cidades, devendo, em 2030, alcançar o patamar de 91%, conforme dados da (UN,2014), de acordo com as afirmações realizadas por [Machado e Piccinini 2018]. Esses números fazem com que a mobilidade urbana como conhecemos se torne saturada e prejudicada afetando diretamente a qualidade de vida das pessoas, causando prejuízos na qualidade e agilidade de deslocamento, resultando em danos de diversos tipos, inclusive psicológicos.

Dessa forma, e, segundo [Machado e Piccinini 2018], através do transporte as cidades conseguem alcançar o desenvolvimento, afinal essa mobilidade é necessária para transportar pessoas, mercadorias e serviços. Contudo, quando analisado o contexto da expansão urbana, o transporte pode se tornar um grande problema logístico, pois as vias possuem espaço limitado, assim, com o aumento do número de veículos, o congestionamento se torna inevitável. Esse congestionamento faz surgir a necessidade de desenvolvimento de alternativas para esses problemas.

Aproveitando o conceito de [Bandyopadhyay, Raj e Varghese 2018], as avenidas das cidades são regularmente vítimas de congestionamento em que automóveis, caminhões, teleféricos, bicicletas e motonetas lutam por espaço para avançar entre os pontos A e B o mais rápido possível. Contudo, neste contexto, o trânsito fica paralisado, seja pelo excesso de veículos, seja por acidentes ou obras. Como a condução fica congestionada os danos se desenvolvem, a decepção aumenta, o tempo é desperdiçado e o dinheiro perdido.

Superar essa questão pode rapidamente se transformar em um grande desafio. O congestionamento afeta diariamente as vias das grandes cidades, causando problemas

tanto para a população, como para os órgãos responsáveis por cuidar dessa logística. Contudo, as vias aéreas ainda são pouco utilizadas dentro do perímetro urbano, mas, quando mais usadas abrirão espaço para redução dos problemas de mobilidade, frequentemente, observados em solo.

A Mobilidade Aérea Urbana (UAM – do inglês) é um sistema seguro e eficaz para viagens aéreas e transporte de carga, dentro de um território urbano como conceitua [Bandyopadhyay, Raj e Varghese 2018]. Algumas organizações, tanto pequenas quanto as grandes, estão iniciando o desenvolvimento para tornar a Mobilidade Aérea Urbana possível e em curto espaço de tempo.

Como objetivo, este trabalho possui a finalidade de contribuir para a solução de problemas de mobilidade aérea urbana, buscando como alternativa o desenvolvimento de um sistema com capacidade de fazer a correção de rotas para o deslocamento aéreo de pessoas e cargas.

Especificamente, pretende-se aliviar a problemática de possíveis colisões durante o trajeto aéreo urbano, através de correção de rotas, utilizando de um sistema de visualização de mapas, a partir de coordenadas geográficas, com um exibidor de mapas, por exemplo, para gerar rotas corrigidas, através de um sistema de correção de rotas, e então exibir essas rotas corrigidas por meio da visualização do mapa, possuindo como resultados: I) a rota corrigida; II) um GeoDataFrame - isto é, um quadro de dados geográficos - com os dados da mesma rota corrigida e III) os dados do ponto de colisão. Em consequência, esse sistema pode tornar mais dinâmico o sistema aéreo dentro de cidades, gerando maior segurança à mobilidade aérea reduzindo os acidentes, os quais possam ocorrer.

2. REVISÃO BIBLIOGRÁFICA

Oportunamente, [Bauranov e Rakas 2021] definiram que a Mobilidade Aérea Urbana (UAM) aliada ao desenvolvimento tecnológico em automação impulsionou o desenvolvimento da indústria da aviação urbana. O conceito UAM compreende um conjunto de regras, procedimentos e tecnologias que viabilizam e regulamentam as operações de tráfego aéreo de cargas e passageiros no ambiente urbano. A partir desses conceitos, entende-se que a Mobilidade Aérea Urbana está ligada ao desenvolvimento tecnológico que envolve a automação e, compreende-se que, se faz necessário desenvolver tecnologias que auxiliem e controlem essa movimentação aérea para que não ocorram acidentes e torne possível uma logística e organização funcional e eficiente. Para isso, quando se trata de veículos autônomos ou que possuem piloto automático, torna-se essencial um sistema que organize as rotas aéreas urbanas, com a finalidade de evitar congestionamento aéreo ou até mesmo possíveis acidentes ou incidentes entre aeronaves.

Para os autores [Bharadwaj et al. 2021], A UAM exigirá gerenciamento para voos sob demanda, de alto volume e curto alcance nas proximidades do espaço aéreo urbano (por exemplo, abaixo de 10.000 pés), com aeronaves cada vez mais autônomas, visto que estes veículos estarão operando em espaços aéreos urbanos com altas densidades de tráfego, eles precisam ser capazes de operar com padrões de separação menores do que os atuais. Com isso, pode-se afirmar que, sistemas de gerenciamento e controle de rotas de espaço aéreo existentes são ineficazes quando aplicados a veículos autônomos que operem em rotas curtas, sob demanda e com altas densidades de tráfego. Para isso,

necessita-se de algum sistema que consiga analisar as rotas aéreas e recalcular qual seria a melhor rota alternativa viável para a realização do trajeto, caso a primeira rota esteja congestionada.

Esse sistema necessita de uma ferramenta capaz de descrever cada ponto da rota, para, assim, ter a capacidade de modificar esses pontos individualmente e, dessa forma, realizar a correção funcional das rotas. Para isso, foi utilizada a biblioteca GeoPandas nesse trabalho, pois esta, assim com afirmou [Chawanji 2020], consegue converter um arquivo GeoJSON (do inglês Geographic JavaScript Object Notation, ou seja, um arquivo que contém dados geográficos no formato JSON - sendo importante destacar que o sistema utiliza esse arquivo com a linguagem Python - em um GeoDataFrame, o qual contém campos com os tipos de características, de cada ponto, e as coordenadas geográficas dessas características.

A biblioteca GeoPandas, a qual é responsável por gerar o GeoDataFrame, que possui a capacidade de armazenar colunas geométricas e fazer cálculos espaciais, corroborando, desse modo, com o que foi afirmado por [Lopes, Delbem e Sousa 2021], é uma biblioteca da linguagem de programação Python. Linguagem esta que, especificamente, foi utilizada nesse sistema por possuir essa e outras bibliotecas capazes de executar os processos necessários para garantir o funcionamento desse algoritmo e exibir os dados, visualmente, de modo a facilitar o entendimento dos dados resultantes.

Com um sistema que consiga corrigir rotas somado a um mapeamento da área, esse se tornaria imprescindível para calcular os eventuais riscos de colisão - recalculando as rotas e tornando os trajetos mais seguros e reduzindo a possibilidade de acidentes. Oferecendo, assim, uma bem planejada e desenvolvida mobilidade aérea urbana, corroborando com a ideia de [Machado e Piccinini 2018], retomando a qualidade de vida das pessoas, gerando rapidez nas entregas de mercadorias e serviços que dependam da agilidade no transporte, possibilitando, desse modo, à existência de uma taxonomia sob demanda, como conceituou [Cohen, Shaheen e Farrar 2021], através desse sistema, afinal, este, foi construído para evitar possíveis colisões na mobilidade aérea urbana. Porém, torna-se importante destacar que esse sistema, apenas, não pode ser utilizado como um sistema de controle de navegação, pois o mesmo apresenta somente a correção das rotas consideradas, todavia, o mesmo - possivelmente - pode ser integrado à um sistema de controle.

3. METODOLOGIA

O presente sistema foi desenvolvido na linguagem de programação Python e foram utilizadas bibliotecas presentes na mesma linguagem de programação, as quais podem ser observadas na subseção 3.1.1. Dessa forma, esse algoritmo utiliza funções, que foram desenvolvidas pelos autores para gerar o mapa, gerar as rotas, comparar e corrigir as rotas e exibir as rotas com e sem correção. A funcionalidade desse algoritmo se apresenta quando as rotas são calculadas, corrigidas e exibidas, através de um mapa (o qual este também fornece - com o auxílio de bibliotecas). Uma das funções desse sistema gera esse mapa, como pode ser observado na subseção 3.1.2. A exibição do mapa - com as rotas - pode ser encontrada na subseção 3.2.3.

O intuito dessa seção é mostrar como esse sistema foi desenvolvido e como ele pode ser replicado. Foi utilizado o ambiente JupyterLab para construí-lo, sendo este mol-

dado em células - ou seja, conjuntos de blocos de códigos - as quais foram executadas individualmente, porém em sequência. Essas células estão representadas entre as subseções 3.1.1 e 3.1.9, que podem ser vistas na subseção 3.1 a seguir.

3.1. PROGRAMAÇÃO DO SISTEMA:

As células de código estão divididas entre as subseções 3.1.1 e 3.1.9, essas sequências de código são executadas individualmente, porém, algumas necessitam de outras células para serem executadas. Cada conjunto de código tem sua própria saída, podendo um conjunto ter saída antes da execução de outra célula iniciar.

Parte desse trabalho, como o código presente na subseção 3.1.2, se inspirou na publicação de [Abdishakur 2020], onde o mesmo mostrou como realizar a criação de rotas, as quais ligam um ponto a outro, usando as bibliotecas OSMnx, Geopandas e Plotly Express, da linguagem Python.

3.1.1. IMPORTAÇÃO DAS BIBLIOTECAS:

```
1 # Importacao das Bibliotecas
2 import pandas as pd
3 import geopandas as gpd
4 from shapely.geometry import Point, LineString
5 import dataframe_image as dfi
6 import networkx as nx
7 import osmnx as ox
8 import numpy as np
```

Código 1. Importando as bibliotecas a serem utilizadas.

Fonte: Código desenvolvido pelos autores, 2022.

Esse sistema utilizou bibliotecas da linguagem Python para algumas de suas funções. Essas bibliotecas foram as seguintes: Pandas, Geopandas, Dataframe_image, Networkx, OSMnx, Numpy e as funções Point e LineString da função Shapely da biblioteca Geometry. A biblioteca OSMnx foi utilizada especificamente para a geração das rotas mais próximas, dos mapas, etc. Corroborando, assim, com a afirmação de [Boeing 2017], o qual apresentou esta biblioteca como um novo tipo de ferramenta capaz de analisar e baixar redes de vias em qualquer localização do mundo. Também adicionou que a OSMnx tem funções integradas, as quais podem fazer a análise de redes de vias, os cálculos de rotas, a projeção e visualização dessas redes e de maneira precisa e veloz executar cálculos de várias medidas métricas e topológicas, como configurações urbanas e de transporte. Outras bibliotecas essenciais para o desenvolvimento do sistema foram a pandas e a Geopandas, as quais foram utilizadas para a criação dos DataFrames e dos GeoDataFrames, atributos o quais foram imprescindíveis para o sistema de correção de rotas.

3.1.2. FUNÇÃO DE GERAÇÃO DO MAPA:

```
1 # Importacao de dados de um determinado local - para
      construir o mapa
```

```
2 def map_creation(place, distance, transport):
3         G = ox.graph_from_address(place, dist=distance,
    network_type=transport)
4         return G
```

Código 2. Gerando o mapa do local indicado.

Fonte: Código desenvolvido pelos autores, 2022.

A Função de Geração de Mapa foi essencial para a geração do mapa, o qual foi imprescindível para o funcionamento desse sistema - utilizando a biblioteca OSMnx, conforme apresentou [Abdishakur 2020] - por precisar apenas de informações como o nome da cidade a ser utilizada e o tipo de transporte a ser usado, por exemplo. O mapa criado por essa função foi utilizado durante todo o código, tanto para criar rotas, como para, através dela, corrigir e exibir a rota corrigida.

3.1.3. FUNÇÃO DE GERAÇÃO DE ROTAS:

```
1 # Gerar rotas com base na origem e no destino
2 def shortest_paths(G, orig, dest, nodes_proj, file_name):
3     route = nx.shortest_path(G, orig, dest, weight='length'
    )
4
5     # Obter os nos ao longo da rota mais curta
6     route_nodes = nodes_proj.loc[route]
7
8     route_nodes
9
10    route_nodes.to_file(file_name, driver='GeoJSON')
11
12    return route
```

Código 3. Gerando as rotas através dos pontos de origem e destino.

Fonte: Código desenvolvido pelos autores, 2022.

A Função de Geração de Rotas utiliza de pontos de origem e destino previamente atribuídos para a geração da rota mais próxima - com a biblioteca networkx. Em seguida, cria a rota e então a salva em um documento do tipo GeoJSON - tipo o qual armazena, separadamente, dados dessa rota.

3.1.4. FUNÇÃO DE CORREÇÃO DE ROTAS:

```
1 # Funcao de correcao baseada no ponto a ser corrigido
2 def route_correction_function(G, nodes_proj, route, dfJSON,
    nodeID_to_Fix, nodeID_to_Fix_next = 0,
    nodeID_to_Fix_previous = 0):
3
4     # Lista com todos os nos
```

```
5     nodes_Lon_Lat = nodes_proj[['lon', 'lat']]
6
7     nodes_Lon_Lat_List = nodes_Lon_Lat.values.tolist()
8
9     # Lista com todos os nos da rota
10    nodesRoute = dfJSON[['lon', 'lat']]
11
12    nodesRoute_List = nodesRoute.values.tolist()
13
14    # Lista do no
15
16    nodeCoordinates = []
17
18    nodeCoordinates.append([nodes_proj.loc[nodeID_to_Fix].
      lon, nodes_proj.loc[nodeID_to_Fix].lat])
19
20    # Proximo osmid dos nos
21    if(nodeID_to_Fix_next != 0):
22        nodeCoordinates.append([nodes_proj.loc[
      nodeID_to_Fix_next].lon, nodes_proj.loc[
      nodeID_to_Fix_next].lat])
23    else:
24        nodeCoordinates.append([nodes_proj.loc[
      nodeID_to_Fix].lon, nodes_proj.loc[nodeID_to_Fix].lat])
25
26    # Osmid anterior dos nos
27    if(nodeID_to_Fix_previous != 0):
28        nodeCoordinates.append([nodes_proj.loc[
      nodeID_to_Fix_previous].lon, nodes_proj.loc[
      nodeID_to_Fix_previous].lat])
29    else:
30        nodeCoordinates.append([nodes_proj.loc[
      nodeID_to_Fix].lon, nodes_proj.loc[nodeID_to_Fix].lat])
31
32    # Encontrando o no mais proximo atraves do outro no
33
34    def find_nearest_node(nodes, newNode, routeNodes):
35        nearest_node = None
36        nearest_distance = None
37        for point in nodes:
38            distance = ((point[0] - newNode[0][0])**2 + (
      point[1] - newNode[0][1])**2)**0.5
39            if [round(num, 6) for num in point] in
      routeNodes:
40                break
41            else:
```

```
42                  if nearest_distance is None or distance <
    nearest_distance and point != newNode[0] and point !=
    newNode[1] and point != newNode[2]:
43                      nearest_node = point
44                      nearest_distance = distance
45
46          return nearest_node
47
48      # Executando a funcao closest_node
49      closest_node = find_nearest_node(nodes_Lon_Lat_List,
    nodeCoordinates, nodesRoute_List)
50
51      # Encontre o ID do no mais proximo
52      node_id = ox.nearest_nodes(G, closest_node[0],
    closest_node[1])
53
54      aux = nodes_proj
55
56      return aux.loc[node_id], node_id
```

Código 4. Corrigindo rotas através do nó de colisão.

Fonte: Código desenvolvido pelos autores, 2022.

Observa-se que, a presente função é uma das principais desse sistema, pelo fato de que esta faz a correção das rotas, possibilitando, assim, a funcionalidade para a qual esse algoritmo foi projetado.

Essa função recebe dados os quais são essenciais para a sua execução, como o mapa do local, o nó de colisão e a rota a ser corrigida, por exemplo. O último item, particularmente, serve como referência para a correção da rota, pois esse sistema identifica, dentro dessa rota, qual dos nós é o nó de colisão e o substitui por um nó mais próximo, que não pode ser nem ele mesmo, nem o nó anterior, nem o nó seguinte dessa rota, impulsionando, dessa forma, o algoritmo a encontrar um outro nó, dentro dos pontos disponíveis no mapa, que esteja próximo ao nó de colisão. Então esse nó substituinte - que tem como objetivo substituir o nó de colisão - retorna no final da função, disponibilizando-se para ser utilizado, posteriormente, no sistema.

3.1.5. FUNÇÃO DE COMPARAÇÃO E CORREÇÃO DE ROTAS:

```
1 # Funcao para comparar rotas e corrigi-las de acordo com
    nos de colisao
2 def comparing_and_correcting_routes(G, nodes_proj, route_2,
    dfJSON, dfJSON_2, file_name):
3
4     dfJSON_same_point = dfJSON.reset_index().merge(dfJSON_2
    .reset_index(), how = 'inner',indicator=False)
5
```

```
6       listJSON_same_point = dfJSON_same_point.iloc[:, 0].
    to_numpy()
7
8       # Correcao da rota
9       dfJSON_corrected = dfJSON_2
10
11      previous_osmid = 0
12
13      for n in range(0, len(listJSON_same_point)):
14
15          dfJSON_aux = dfJSON_2.reset_index()
16
17          same_point_index = dfJSON_aux.index[dfJSON_aux.
    osmid == listJSON_same_point[n]]
18
19          next_osmid = int(dfJSON_aux.loc[dfJSON_aux.index ==
     same_point_index[0] + 1].osmid)
20
21          if(n > 0):
22              previous_osmid = int(dfJSON_aux.loc[dfJSON_aux.
    index == same_point_index[0] - 1].osmid)
23
24          dfJSON_corrected.loc[listJSON_same_point[n]],
    node_id = route_correction_function(G, nodes_proj,
    route_2, dfJSON_corrected, listJSON_same_point[n],
    next_osmid, previous_osmid)
25
26          dfJSON_corrected.rename(index={listJSON_same_point[
    n]: node_id}, inplace=True)
27
28      route_line_final_2 = LineString(list(dfJSON_corrected.
    geometry.values))
29
30      dfJSON_corrected.to_file(file_name, driver='GeoJSON')
31
32      dfi.export(dfJSON_corrected, 'dfJSON_corrected.png',
    max_rows=15)
33
34      return dfJSON_corrected, route_line_final_2
```

Código 5. Comparando as rotas e corrigindo-as de acordo com o nó de colisão.

Fonte: Código desenvolvido pelos autores, 2022.

Essa função utiliza de duas rotas atribuídas para verificar se existe algum ponto em comum entre elas, possuindo algum ponto em comum, ou seja, havendo um ou mais nós de colisão - nó o qual há possibilidade de colisão, caso ambas as rotas sejam executadas - então a segunda rota é corrigida, utilizando a Função de Correção de Rotas (Código

4), usando com parâmetro os nós de colisão encontrados. Após a substituição dos nós de colisão pelos nós substituintes - nós que substituem os nós de colisão, corrigindo a rota - a função retorna um GeoDataFrame com a segunda rota corrigida e também retorna a rota corrigida em formato de linha.

3.1.6. FUNÇÃO PARA CRIAR AS ROTAS E RODAR O SISTEMA:

```
# Criar as rotas e rodar o sistema
def create_route_and_run(place, transport, orig1, dest1, orig2, dest2, JSON1_name, JSON2_name):
    # Primeira rota
    G = map_creation(place, 5000, transport)

    G = ox.add_edge_speeds(G)

    G = ox.add_edge_travel_times(G) # Tempo da viagem

    # Graph Proj - projecao do grafico
    graph_proj = ox.project_graph(G)

    # Obter arestas e nos
    nodes_proj, edges_proj = ox.graph_to_gdfs(graph_proj, nodes=True, edges=True)

    orig = list(G.nodes())[orig1]

    dest = list(G.nodes())[dest1]

    route = shortest_paths(G, orig, dest, nodes_proj, JSON1_name)

    # Obter os nos de um arquivo JSON - primeira rota
    dfJSON = gpd.read_file(JSON1_name).set_index('osmid')

    dfi.export(dfJSON, 'dfJSON.png', max_rows=15)

    # Criar uma geometria para a primeira rota
    route_line = LineString(list(dfJSON.geometry.values))

    # Segunda rota
    orig_2 = list(G.nodes())[orig2]

    dest_2 = list(G.nodes())[dest2]

    route_2 = shortest_paths(G, orig_2, dest_2, nodes_proj, JSON2_name)
```

```
36
37      # Obter os nos de um arquivo JSON - segunda rota
38      dfJSON_2 = gpd.read_file(JSON2_name).set_index('osmid')
39
40      dfi.export(dfJSON_2, 'dfJSON_2.png', max_rows=15)
41
42      # Criar uma geometria para a segunda rota
43      route_line_2 = LineString(list(dfJSON_2.geometry.values
        ))
44
45      return G, dfJSON, dfJSON_2, route_line, route_line_2,
        route, route_2, nodes_proj, edges_proj, orig, dest,
        orig_2, dest_2
```

Código 6. Criando as rotas através dos pontos de origem e destino definidos e rodando o sistema.

Fonte: Código desenvolvido pelos autores, 2022.

Essa função cria o mapa utilizando o local e tipo de transporte atribuídos através da Função de Geração do Mapa (Código 2); gera os nós utilizando os pontos de origem e destino indicados; cria as rotas através da Função de Geração de Rotas (Código 3), utilizando os nós que foram gerados; abre os arquivos com o formato GeoJSON, atribuídos; gera as rotas em formato de linha, utilizando os arquivos GeoJSON abertos e retorna dados como o mapa, os arquivos GeoJSON abertos, as rotas em formato de lista e de linha e os pontos de origem e destino.

3.1.7. INICIAR O SISTEMA E GERAR ROTAS:

```
1 # Iniciar a criacao de rotas e obter os dados gerados por
      elas
2 G, dfJSON, dfJSON_2, route_line, route_line_2, route,
      route_2, nodes_proj, edges_proj, orig, dest, orig_2,
      dest_2 = create_route_and_run('San Francisco', 'drive',
      -11, 8, 30, 1, "route.geojson", "route2.geojson")
```

Código 7. Iniciando o sistema e obtendo os dados gerados através das rotas criadas.

Fonte: Código desenvolvido pelos autores, 2022.

Esse código, em específico, quando executado, indica os dados da cidade - a qual o mapa será gerado - o tipo de transporte, os pontos de origem e destino e os nomes a serem atribuídos aos arquivos com formato GeoJSON, utilizando a Função para Criar as Rotas e Rodar o Sistema (Código 6). Através dessa função, variáveis são utilizadas para armazenar os dados gerados pelo retorno da mesma.

3.1.8. OBTER A ROTA CORRIGIDA ATRAVÉS DA CORREÇÃO DE ROTAS:

```
# Corrigir a rota e obter o dataframe da rota e a linha da rota
dfJSON_corrected, route_line_final_2 = comparing_and_correcting_routes(G, nodes_proj, route_2, dfJSON, dfJSON_2, "route_corrected.geojson")

dfJSON_corrected
```

Código 8. Rodando a correção de rotas com os dados obtidos anteriormente e obtendo a rota corrigida.

Fonte: Código desenvolvido pelos autores, 2022.

Esse código utiliza a Função de Comparação e Correção de Rotas (Código 5) para comparar a primeira rota com a segunda e, caso haja algum nó de colisão, corrigir a segunda rota, gerando um arquivo GeoJSON da rota corrigida e retornando um GeoDataFrame da rota corrigida. Contudo, caso não haja nenhum nó de colisão - não havendo motivo então para a correção - o retorno se dá por um arquivo GeoJSON e um GeoDataFrame igual aos gerados pela segunda rota.

3.1.9. FUNÇÃO DE EXIBIÇÃO DE ROTAS NO MAPA:

```
# Exibir a rota em um grafico
def show_route(G, nodes_proj, edges_proj, orig, dest, orig_2, dest_2, route_line, route_line_2):

    # Criar um GeoDataFrame
    route_geom = gpd.GeoDataFrame([[route_line]], geometry='geometry', crs=edges_proj.crs, columns=['geometry'])

    # Criar um segundo GeoDataFrame
    route_geom_2 = gpd.GeoDataFrame([[route_line_2]], geometry='geometry', crs=edges_proj.crs, columns=['geometry'])

    # Calcular o comprimento da rota
    route_geom['length_m'] = route_geom.length

    # Recuperar as linhas dos nos do GeoDataFrame
    o_closest = nodes_proj.loc[orig]
    t_closest = nodes_proj.loc[dest]

    # Segundo
    o_closest_2 = nodes_proj.loc[orig_2]
    t_closest_2 = nodes_proj.loc[dest_2]

    # Criar um GeoDataFrame a partir dos pontos de origem e destino
```

```
22      od_nodes = gpd.GeoDataFrame([o_closest, t_closest],
    geometry='geometry', crs=nodes_proj.crs)
23
24      # Criar um segundo GeoDataFrame a partir dos pontos de
    origem e destino
25      od_2_nodes = gpd.GeoDataFrame([o_closest_2, t_closest_2
    ], geometry='geometry', crs=nodes_proj.crs)
26
27      # Plotar arestas e nos
28      ax = edges_proj.plot(linewidth=0.75, color='grey')
29      ax = nodes_proj.plot(ax=ax, markersize=1, color='grey')
30
31      # Adicionar a primeira rota
32      ax = route_geom.plot(ax=ax, linewidth=0.5, linestyle='-
    ', color='blue')
33
34      # Adicionar a segunda rota
35      ax = route_geom_2.plot(ax=ax, linewidth=0.5, linestyle=
    '-', color='red')
36
37      # Adicionar os nos de origem e de destino da primeira
    rota
38      ax = od_nodes.plot(ax=ax, markersize=2, color='blue')
39
40      # Adicionar os nos de origem e de destino da segunda
    rota
41      ax = od_2_nodes.plot(ax=ax, markersize=2, color='red')
42
43      return ax
```

Código 9. Função para exibir no mapa as rotas indicadas.

Fonte: Código desenvolvido pelos autores, 2022.

Essa função utiliza a biblioteca Geopandas para exibir, no mapa, as rotas indicadas, mostrando a primeira rota em azul e a segunda rota em vermelho - para possibilitar a distinção das rotas quando o mapa for observado. As vias e os pontos do mapa são apresentados em cinza, para manter o destacamento das rotas em evidência.

3.2. RESULTADOS OBTIDOS:

Conforme esperado, os resultados os quais foram obtidos através desse sistema, podem ser observados na seguinte sequência: a rota corrigida pode ser encontrada na subseção 3.2.3, mais especificamente no item B da Figura 3, a qual apresenta a rota corrigida, a primeira rota e o ponto de colisão circulado em amarelo, no mapa do local determinado; o GeoDataFrame com os dados da rota corrigida, o qual se encontra na subseção 3.2.7, na Figura 7, a qual mostra os dados do GeoDataFrame entre a posição 60 e 69, com a posição 68 em destaque e, por fim, os dados do ponto de colisão, que podem ser localizados na subseção 3.2.8, precisamente no item B da Figura 8, que mostra dados como as coordenadas ”x”e ”y”, a latitude, a longitude e a geometria do ponto.

Os demais resultados obtidos são apenas como acréscimo ao trabalho, os quais possuem o objetivo de demonstrar a funcionalidade do sistema e apresentar, de forma visual, como esse algoritmo procede, quando aplicado.

3.2.1. EXIBIÇÃO DO MAPA GERADO:

Figura 1 - Exibindo o mapa gerado.

Fonte: Imagem gerada pelos autores, 2022.

A Figura 1 apresenta o mapa gerado a partir do Código 2, presente na subseção 3.1.2. Esse mapa representa uma parte da cidade de São Francisco, localizada no estado da Califórnia, nos Estados Unidos da América. Cada ponto desse mapa é um nó, o qual é ligado por vias - sejam elas ruas ou avenidas, por exemplo - esses nós são a representação visual dos nós utilizados nas rotas desse sistema, o qual utilizou dos mesmos para realizar a correção de rota.

3.2.2. EXIBIÇÃO DAS LINHAS DAS ROTAS:

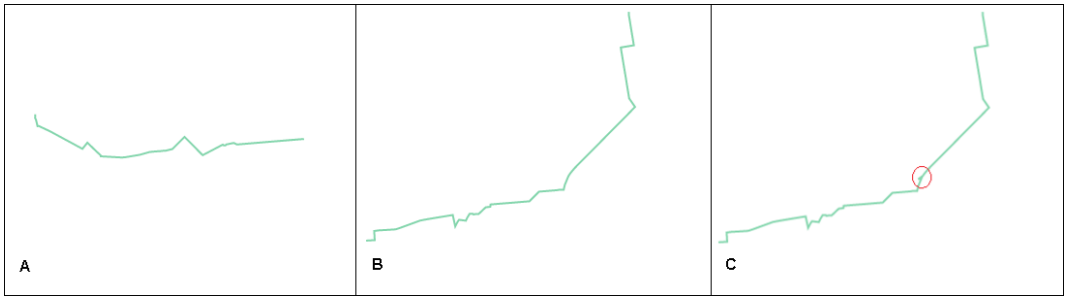

Figura 2 - O item A é a primeira rota; o item B é a segunda rota antes da correção; o item C é a segunda rota corrigida.

Fonte: Imagem gerada pelos autores, 2022.

O item A da Figura 2 exibe a primeira rota, a qual foi gerada através do Código 3, presente na subseção 3.1.3, mas o formato de linha que é observado só é possível a partir do Código 6, presente na subseção 3.1.6.

Na Figura 2, o item B mostra a segunda rota antes de ser corrigida, em formato de linha - esse formato para essa rota, também, apenas é possível por causa do Código 6, presente na subseção 3.1.6.

A segunda rota após a correção é representada pelo item C da Figura 2, em formato de linha, utilizando do Código 5, da subseção 3.1.5, entretanto foi adicionado um circulo ao item C, com o objetivo de tornar nítido o local da correção da rota.

3.2.3. EXIBIÇÃO DAS ROTAS NO MAPA ANTES E APÓS A CORREÇÃO:

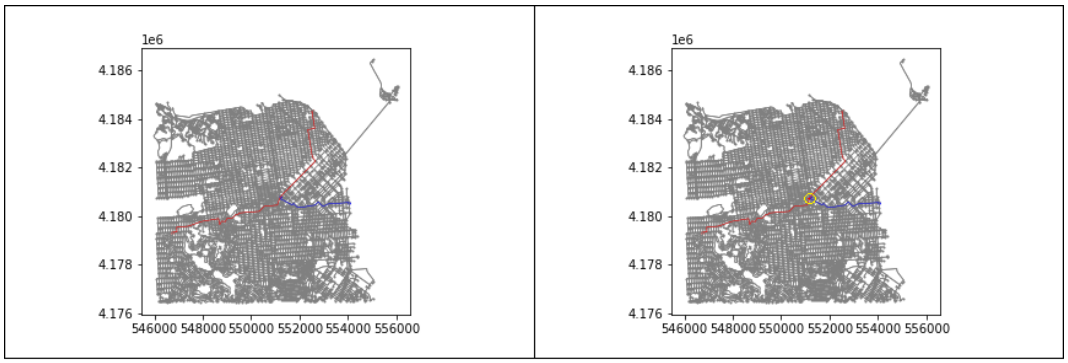

Figura 3 - Exibindo as rotas no mapa antes e após a correção ser feita.

Fonte: Imagem gerada pelos autores, 2022.

Na Figura 3, o item A exibe a primeira rota e a segunda rota antes de ser corrigida, no mapa, o qual a sua exibição é possível a partir do Código 9, presente na subseção 3.1.9. A primeira rota aparece em azul e a segunda rota aparece em vermelho, ambas em formato de linha. Esse mapa também apresenta parte da cidade em questão - que é a cidade de São Francisco - e os nós e vias representados em cinza.

O item B, da Figura 3, mostra o mapa do local, contudo, dessa vez contém a primeira rota e a segunda rota após correção, sendo que a primeira rota é representada através da linha azul e a segunda rota, corrigida, é representada pela linha vermelha. Também foi adicionado um círculo amarelo no local do possível impacto, para mostrar nitidamente a correção da rota.

3.2.4. VERIFICAR O NÓ DE COLISÃO:

	osmid	y	x	street_count	lon	lat	highway	ref	geometry
0	260118044	4.180771e+06	551199.901849	4	-122.418631	37.772842	traffic_signals	None	POINT (551199.90185 4180770.95747)

Figura 4 - Encontrando e exibindo o nó de colisão.

Fonte: Imagem gerada pelos autores, 2022.

A Figura 4 contém o nó de colisão, esse dado é possível obter comparando a primeira com a segunda rota, retornando quais são seus pontos em comum. Nesse caso, o ponto em comum contém o nó com número ”osmid”(do inglês OpenStreetMap ID - ou seja, é um número de identificação de um ponto/nó presente no OpenStreetMap) igual a 260118044. Essa informação é importante para análises posteriores.

3.2.5. VERIFICAR A ROTA CORRIGIDA:

osmid	y	x	street_count	lon	lat	highway	ref	geometry

Figura 5 - Verificando se ainda existe o nó de colisão na rota corrigida.

Fonte: Imagem gerada pelos autores, 2022.

Também foi analisada a segunda rota corrigida, com a finalidade de constatar se haveria algum ponto em comum entre ela e a primeira rota, comparando as duas. Porém, conforme é possível observar na Figura 5, não foi exibido nenhum ponto em comum, corroborando, assim, com o êxito da funcionalidade do presente sistema, evidenciando que, o mesmo, realizou com sucesso a correção da segunda rota.

3.2.6. ENCONTRAR O NÓ DE COLISÃO NA SEGUNDA ROTA:

	osmid	y	x	street_count	lon	lat	highway	ref	geometry
Posição 60	276543369	4.180456e+06	551064.815665	4	-122.420187	37.770009	traffic_signals	None	POINT (551064.81567 4180455.75453)
	65362975	4.180456e+06	551084.237320	4	-122.419967	37.770006	traffic_signals	None	POINT (551084.23732 4180455.58646)
	276543853	4.180467e+06	551087.519244	4	-122.419928	37.770113	traffic_signals	None	POINT (551087.51924 4180467.47883)
	300515878	4.180477e+06	551088.914634	4	-122.419912	37.770197	traffic_signals	None	POINT (551088.91463 4180476.80757)
	65281253	4.180498e+06	551093.441099	3	-122.419859	37.770390	None	None	POINT (551093.44110 4180498.20528)
	65337860	4.180526e+06	551100.967873	3	-122.419772	37.770638	None	None	POINT (551100.96787 4180525.76841)
	65325654	4.180567e+06	551111.755036	3	-122.419646	37.771011	None	None	POINT (551111.75504 4180567.25426)
Posição 68	3386704350	4.180732e+06	551179.561342	3	-122.418865	37.772492	None	None	POINT (551179.56134 4180732.01954)
	260118044	4.180771e+06	551199.901849	4	-122.418631	37.772842	traffic_signals	None	POINT (551199.90185 4180770.95747)
Posição 69	65363176	4.180784e+06	551208.467902	4	-122.418533	37.772961	traffic_signals	None	POINT (551208.46790 4180784.24747)

Figura 6 - Buscando a posição do nó de colisão na segunda rota antes da correção.

Fonte: Imagem gerada pelos autores, 2022.

Após identificar, manualmente, qual é o nó de colisão - o qual possui número ”osmid”igual a 260118044, esse nó foi buscado e encontrado na segunda rota, antes da correção, entre o nó 60 e o nó 69, mais especificamente na posição 68 dessa rota, como pode ser visto na Figura 6. Também é visível que esse nó, assim como os outros nós, possui dados como coordenadas ”x”e ”y”, latitude, longitude e a geometria do nó, por exemplo.

3.2.7. VERIFICAR SE HÁ O NÓ DE COLISÃO NA ROTA CORRIGIDA:

	osmid	y	x	street_count	lon	lat	highway	ref	geometry
Posição 60	276543369	4.180456e+06	551064.815665	4	-122.420187	37.770009	traffic_signals	None	POINT (551064.81567 4180455.75453)
	65362975	4.180456e+06	551084.237320	4	-122.419967	37.770006	traffic_signals	None	POINT (551084.23732 4180455.58646)
	276543853	4.180467e+06	551087.519244	4	-122.419928	37.770113	traffic_signals	None	POINT (551087.51924 4180467.47883)
	300515878	4.180477e+06	551088.914634	4	-122.419912	37.770197	traffic_signals	None	POINT (551088.91463 4180476.80757)
	65281253	4.180498e+06	551093.441099	3	-122.419859	37.770390	None	None	POINT (551093.44110 4180498.20528)
	65337860	4.180526e+06	551100.967873	3	-122.419772	37.770638	None	None	POINT (551100.96787 4180525.76841)
	65325654	4.180567e+06	551111.755036	3	-122.419646	37.771011	None	None	POINT (551111.75504 4180567.25426)
Posição 68	3386704350	4.180732e+06	551179.561342	3	-122.418865	37.772492	None	None	POINT (551179.56134 4180732.01954)
	65315835	4.180711e+06	551108.602204	3	-122.419672	37.772303	NaN	NaN	POINT (551108.60220 4180710.59767)
Posição 69	65363176	4.180784e+06	551208.467902	4	-122.418533	37.772961	traffic_signals	None	POINT (551208.46790 4180784.24747)

Figura 7 - Verificando se o nó de colisão está presente na rota corrigida.

Fonte: Imagem gerada pelos autores, 2022.

Assim como na Figura 6, o nó de colisão também foi buscado entre a posição 60 e 69 da segunda rota corrigida, porém, o número ”osmid”de valor 260118044 não foi encontrado, em vez disso, na posição 68 foi encontrado o número ”osmid”igual a 65315835, conforme é possível observar na Figura 7, ou seja, o nó de colisão foi substituído com sucesso, assim como todas os dados presentes nele, desse modo, observa-se que as coordenadas ”x”e ”y”, a latitude, a longitude e a geometria do nó são diferentes das encontradas no nó de colisão presente na Figura 6, por exemplo.

3.2.8. DETALHAMENTO DO NÓ DE COLISÃO E DO NÓ SUBSTITUINTE:

```
y                                                  4180770.957472
x                                                  551199.901849
street_count                                                   4
lon                                                  -122.418631
lat                                                    37.772842
highway                                          traffic_signals
ref                                                          None
geometry        POINT (551199.9018491833 4180770.9574723584)
Name: 260118044, dtype: object
```

A

```
y                                                  4180710.597675
x                                                  551108.602204
street_count                                                   3
lon                                                  -122.419672
lat                                                    37.772303
highway                                                      NaN
ref                                                          NaN
geometry        POINT (551108.6022037211 4180710.5976749836)
Name: 65315835, dtype: object
```

B

Figura 8 - Exibindo os dados presentes no nó de colisão e no nó substituinte.

Fonte: Imagem gerada pelos autores, 2022.

Todos os dados presentes no nó de colisão podem ser observados no item A da Figura 8. Esses dados foram obtidos através da posição 68 da segunda rota antes da

correção, a qual pode ser encontrada também, como destacado, anteriormente, na Figura 6.

De maneira semelhante ao nó de colisão, todos os dados presentes no nó substituinte podem ser encontrados na posição 68 da segunda rota corrigida, como é possível localizar no item B da Figura 8. Diferentemente do ocorrido no item A, foi exibido um outro nó, divergente ao nó de colisão, contendo informações, como latitude (lat) e longitude (lon) distintas ao dados presentes no nó de colisão.

4. CONSIDERAÇÕES FINAIS

Em suma, os resultados esperados foram obtidos com êxito, evidenciando, assim, a funcionalidade desse sistema de correção de rotas, proporcionando, desse modo, uma possível melhora no desenvolvimento das cidades através do transporte ao aplicar a Mobilidade Aérea Urbana, caso esse algoritmo venha a ser implementado em sistemas de controle de tráfego aéreo. Sendo muito importante frisar que esse sistema por si só não é capaz de realizar o controle de tráfego, contudo, existe a possibilidade de implementá-lo a um sistema de controle existente.

5. REFERÊNCIAS BIBLIOGRÁFICAS

ABDISHAKUR. *Routing street networks: Find your way with Python*. 2020. Disponível em: https://shakasom.medium.com/routing-street-networks-find-your-way-with-python-9ba498147342. Acesso em: 08 de Outubro de 2022.

BANDYOPADHYAY, A.; RAJ, N. S. S.; VARGHESE, J. T. Coexisting in a world with urban air mobility: A revolutionary transportation system. IEEE Transactions on Intelligent Transportation Systems, Dubai, United Arab Emirates, p. 1–6, 2018.

BAURANOV, A.; RAKAS, J. Designing airspace for urban air mobility: A review of concepts and approaches. *Progress in Aerospace Sciences*, v. 125, p. 100726, ago. 2021.

BHARADWAJ, S. et al. Decentralized control synthesis for air traffic management in urban air mobility. *IEEE Transactions on Control of Network Systems*, v. 8, n. 2, p. 598–608, 2021.

BOEING, G. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, v. 65, p. 126–139, 2017.

CHAWANJI, S. *Vector tile cache connecting effective spatial communication and geospatial AI*. Tese (Doutorado) — Technischen Universität Wien, Department für Geodäsie und Geoinformation, 2020.

COHEN, A. P.; SHAHEEN, S. A.; FARRAR, E. M. Urban air mobility: History, ecosystem, market potential, and challenges. *IEEE Transactions on Intelligent Transportation Systems*, v. 22, n. 9, p. 6074–6087, 2021.

LOPES, G.; DELBEM, A.; SOUSA, J. Introdução à análise de dados geoespaciais com python. In: *Minicursos do XIV Encontro Unificado de Computação do Piauí (ENUCOMPI) e XI Simpósio de Sistemas de Informação (SINFO)*. Picos, Piauí: SBC, 2021. p. 82–106.

MACHADO, L.; PICCININI, L. S. Os desafios para a efetividade da implementação dos planos de mobilidade urbana: uma revisão sistemática. *urbe. Revista Brasileira de Gestão Urbana*, SciELO Brasil, v. 10, p. 72–94, 2018.