

DANH SÁCH NHÓM

STT	Họ và tên	MSSV	Lớp	Ghi chú
1	Kiều Cẩm An	2258420069	22ĐHKL02	Nhóm trưởng
2	Nguyễn Thị Mỹ Hạnh	2258420025	22ĐHKL01	Thành viên
3	Nguyễn Hoàng Huy	2258420096	22ĐHKL02	Thành viên

Cán bộ chấm thi 1 <i>(ký và ghi rõ họ tên)</i>	Cán bộ chấm thi 2 <i>(ký và ghi rõ họ tên)</i>
Cán bộ chấm thi phúc khảo 1 <i>(ký và ghi rõ họ tên)</i>	Cán bộ chấm thi phúc khảo 2 <i>(ký và ghi rõ họ tên)</i>

LỜI CẢM ƠN

Đề tài “*Xây dựng ứng dụng hỗ trợ đề xuất đường bay thay thế tạm thời bằng thuật toán A* trong vùng thông báo bay Hồ Chí Minh*” là kết quả từ sự cố gắng và quá trình triển khai nghiêm túc đến từ các thành viên trong nhóm.

Đầu tiên, nhóm đề tài xin gửi lời cảm ơn đến TS. Phan Thanh Minh và các giảng viên tại Học viện Hàng không Việt Nam đã tận tình hướng dẫn, đóng góp ý kiến giúp nâng cao nội dung đề tài. Nhóm đề tài trân trọng cảm ơn các tổ chức, tác giả đã cung cấp những tài liệu quý giá hỗ trợ quá trình nghiên cứu.

Mặc dù đã cố gắng cập nhật những thông tin mới nhất, đề tài vẫn có thể còn hạn chế. Nhóm đề tài mong nhận được ý kiến đóng góp từ thầy/cô để hoàn thiện hơn và mở rộng hiểu biết trong lĩnh vực này.

Thành phố Hồ Chí Minh, ngày 05 tháng 07 năm 2025

Nhóm thực hiện đề tài

LỜI CAM ĐOAN

Nhóm đề tài cam đoan “*Xây dựng ứng dụng hỗ trợ để xuất đường bay thay thế tạm thời bằng thuật toán A* trong vùng thông báo bay Hồ Chí Minh*” là bài nghiên cứu độc lập của tập thể nhóm cùng sự chỉ dẫn của TS. Phan Thanh Minh.

Các số liệu, kết quả nghiên cứu của các tác giả trên thế giới được tham khảo từ sách, báo, Internet và các nghiên cứu trong và ngoài nước đều được trích dẫn đầy đủ trong phần tài liệu tham khảo. Nhóm đảm bảo kết quả nghiên cứu khách quan, trung thực và không sao chép từ bất kỳ bài nghiên cứu nào khác.

Thành phố Hồ Chí Minh, ngày 05 tháng 07 năm 2025

Nhóm thực hiện đề tài

PHÂN CÔNG NHIỆM VỤ THÀNH VIÊN NHÓM

STT	HỌ VÀ TÊN	NHIỆM VỤ
1	Kiều Cẩm An <i>(Nhóm trưởng)</i>	<ul style="list-style-type: none">- Đóng góp ý tưởng xây dựng bối cảnh.- Tìm kiếm, xây dựng phát triển và hoàn thành các nội dung trong Chương .- Phân chia nhiệm vụ các thành viên.- Cùng đóng góp, chỉnh sửa hoàn chỉnh đề tài nghiên cứu.
2	Nguyễn Thị Mỹ Hạnh	<ul style="list-style-type: none">- Đóng góp ý tưởng xây dựng bối cảnh.- Tìm kiếm, xây dựng phát triển và hoàn thành các nội dung trong Chương .- Cùng đóng góp, chỉnh sửa hoàn chỉnh đề tài nghiên cứu.
3	Nguyễn Hoàng Huy	<ul style="list-style-type: none">- Đóng góp ý tưởng xây dựng bối cảnh.- Tìm kiếm, xây dựng phát triển và hoàn thành các nội dung trong Chương .

		<ul style="list-style-type: none">- Phân chia nhiệm vụ các thành viên.- Cùng đóng góp, chỉnh sửa hoàn chỉnh đề tài nghiên cứu.
--	--	---

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

Ngày tháng năm ...

Giảng viên hướng dẫn

TS. Phan Thanh Minh

DANH MỤC BẢNG BIỂU

DANH MỤC HÌNH ẢNH

DANH MỤC VIẾT TẮT

STT	TƯ VIẾT TẮT	Ý NGHĨA
1	FIR (Flight information region)	Vùng thông báo bay
2	KSVKL	Kiểm soát viên không lưu
3	ICAO (International Civil Aviation Organization)	Tổ chức hàng không dân dụng quốc tế
4	ATM (Air traffic management)	Quản lý không lưu
5	ATFM (Air Traffic Flow Management)	Quản lý luồng không lưu
6	IATA (International Air Transport Association)	Hiệp hội Vận tải Hàng không Quốc Tế
7	CANSO (Civil Air Navigation Services Organisation)	Tổ chức Các nhà cung cấp Dịch vụ Bảo đảm Hoạt động bay Toàn cầu

8	FAA (Federal Aviation Administration)	Cục Hàng không Liên bang Mỹ
9	ATS (Air Traffic Services)	Dịch vụ không lưu
10	ASM (Airspace Management)	Quản lý vùng trời
11	ANSP (Air Navigation Services Providers)	Nhà cung cấp dịch vụ đảm bảo hoạt động bay
12	CAT (Clear Air Turbulance)	Nhiều động trời trong
13	FRA (Free Route Airspace)	Vùng trời tự do
14	NOTAM (Notice to Airmen)	Điện văn thông báo hàng không
15	STAR (Standard Arrival Procedure)	Phương thức đến tiêu chuẩn
16	DSS (Decision Support System)	Hệ thống hỗ trợ quyết định

CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI NGHIÊN CỨU

1.1 Lý do chọn đề tài nghiên cứu

Trong bối cảnh hoạt động bay ngày càng phức tạp, việc đảm bảo an toàn và duy trì luồng không lưu liên tục trong vùng thông báo bay Hồ Chí Minh (FIR HCM) đặt ra nhiều thách thức, đặc biệt khi xuất hiện các tình huống bất thường như đóng cửa vùng trời, hoạt động quân sự, hoặc thời tiết nguy hiểm. Hiện nay, việc lựa chọn đường bay thay thế chủ yếu vẫn dựa vào kinh nghiệm kiểm soát viên không lưu (KSVKL) và quy trình xử lý thủ công, tiềm ẩn nguy cơ thiếu sót quan trọng, chậm trễ và tăng khói lượng công việc trong giai đoạn chiến thuật.

Đề tài "*Xây dựng ứng dụng hỗ trợ để xuất đường bay thay thế tạm thời bằng thuật toán A* trong vùng thông báo bay Hồ Chí Minh*" được lựa chọn nhằm cung cấp một công cụ hỗ trợ KSVKL và nhà quản lý không lưu trong việc nhanh chóng xác định các tuyến bay thay thế khả thi, đảm bảo tuân thủ vùng cấm, vùng hạn chế và tối ưu về mặt thời gian, khoảng cách. Ứng dụng này không chỉ tăng hiệu quả xử lý tình huống khẩn cấp mà còn đóng vai trò như một nền tảng để xây dựng *Playbook* – tập hợp các phương án phản ứng nhanh, giúp chuẩn hóa quy trình điều hành bay trong các điều kiện biến động. Việc nghiên cứu và triển khai công cụ để xuất đường bay bằng thuật toán A* còn mở ra tiềm năng ứng dụng các thuật toán tìm đường hiện đại trong lĩnh vực quản lý không lưu, từ đó góp phần số hóa và tự động hóa một phần quy trình ra quyết định – xu hướng đang được ICAO (International Civil Aviation Organization) và nhiều quốc gia trên thế giới thúc đẩy.

1.2 Mục đích, mục tiêu và nhiệm vụ nghiên cứu

1.2.1 Mục đích nghiên cứu

Mục đích nghiên cứu là tìm hiểu quy trình thiết kế ứng dụng để xuất đường bay thay thế tạm thời trong các tình huống bất thường ảnh hưởng đến luồng không lưu. Đề tài làm rõ vai trò và tính ứng dụng thực tiễn của sản phẩm *Playbook*. Đồng thời, nghiên cứu phân tích thuật toán A* nhằm đánh giá khả năng áp dụng trong xây dựng

phương án bay linh hoạt, tối ưu và an toàn, góp phần hiện đại hóa công tác điều hành bay tại Việt Nam.

1.2.2 Mục tiêu nghiên cứu

Mục tiêu của nghiên cứu là cung cấp cái nhìn tổng quan về công tác lựa chọn đường bay thay thế trong Quản lý không lưu (ATM - Air traffic management), phân tích các vấn đề thực tiễn và yếu tố ảnh hưởng đến quyết định điều chỉnh đường bay. Đề tài hệ thống hóa cơ sở lý thuyết về các công cụ hỗ trợ quyết định trong Quản lý luồng không lưu (ATFM - Air Traffic Flow Management), đồng thời làm rõ cấu trúc tổ chức vùng trời tại Việt Nam – yếu tố then chốt trong hiệp đồng dân sự – quân sự. Ngoài ra, nghiên cứu phân tích thuật toán A* và đề xuất quy trình xây dựng sản phẩm *Playbook* nhằm nâng cao hiệu quả điều hành bay và tối ưu hóa quản lý không lưu.

1.3 Đối tượng và phạm vi nghiên cứu

1.3.1 Đối tượng nghiên cứu

Đối tượng nghiên cứu: Ứng dụng hỗ trợ để xuất đường bay thay thế tạm thời sử dụng thuật toán A*.

Khách thể nghiên cứu: Hệ thống đường hàng không trong vùng trời Việt Nam.

Đối tượng khảo sát: Hệ thống đường hàng không nằm trong FIR HCM.

1.3.2 Phạm vi nghiên cứu

Phạm vi không gian được giới hạn trong các trực đường hàng không chính trên đất liền thuộc FIR HCM.

Phạm vi thời gian của nghiên cứu được thực hiện từ ngày 25/05/2025 đến ngày 03/06/2025.

1.4 Giả thuyết khoa học

Giả thuyết 1: Việc ứng dụng thuật toán A* trong điều hành bay có khả năng nâng cao hiệu quả của hệ thống ATM thông qua việc tìm kiếm lộ trình bay tối ưu trong thời gian ngắn. Nếu được chứng minh, điều này mở ra hướng ứng dụng các thuật toán trí tuệ nhân tạo trong tự động hóa ra quyết định chiến thuật trong ATFM.

Giả thuyết 2: Sản phẩm *Playbook* cung cấp các phương án đường bay thay thế linh hoạt, giúp KSVKL, tổ lái và cơ quan quân sự phối hợp hiệu quả khi xảy ra tình huống bay lệch khỏi đường hàng không. Điều này không chỉ nâng cao tính chủ động trong điều hành mà còn góp phần đảm bảo an toàn và giảm thiểu xung đột trong không phận giới hạn.

Giả thuyết 3: Ứng dụng để xuất đường bay dựa trên thuật toán có tiềm năng tích hợp vào hệ thống ATM hiện tại, giúp đơn giản hóa quá trình ra quyết định trong điều kiện khai thác thực tế. Sự tích hợp này sẽ góp phần hiện đại hóa quy trình điều hành bay, hướng tới hệ thống ATM thông minh và linh hoạt hơn trong tương lai.

1.5 Hướng tiếp cận và phương pháp nghiên cứu

1.5.1 Hướng tiếp cận

Để tài tiếp cận theo hướng hệ thống, bắt đầu từ việc xây dựng cơ sở lý thuyết thông qua khảo cứu tài liệu chuyên ngành và văn bản quy phạm, làm rõ các vấn đề về lập kế hoạch bay, đường bay thay thế, thuật toán A* và cấu trúc FIR HCM. Một trực bay có mật độ cao trong FIR được lựa chọn để khảo sát, thu thập dữ liệu và xây dựng tình huống giả định. Phần mềm thiết kế được phát triển bằng Python, tích hợp thuật toán A* để tự động để xuất phương án bay thay thế trong các tình huống bất thường. Kết quả được đánh giá dựa trên tính khả thi, an toàn, thời gian phản hồi và khả năng phối hợp quân sự, từ đó phân tích ưu nhược điểm và đề xuất hướng tích hợp vào hệ thống ATM hiện đại.

1.5.2 Phương pháp nghiên cứu

Nghiên cứu được thực hiện dựa trên hệ thống tài liệu chuyên ngành, bao gồm công văn, quyết định, giáo trình, tài liệu học thuật và báo cáo từ các tổ chức hàng

không uy tín như ICAO, Hiệp hội Vận tải Hàng không Quốc Tế (International Air Transport Association - IATA), Tổ chức Các nhà cung cấp Dịch vụ Bảo đảm Hoạt động bay Toàn cầu (Civil Air Navigation Services Organisation - CANSO, Cục Hàng không Liên bang Mỹ (Federal Aviation Administration - FAA), Cục Hàng không Việt Nam (CAAV) và Tổng công ty Quản lý bay Việt Nam (TCT QLBVN). Ngoài ra, nhóm còn tham khảo các nghiên cứu quốc tế và tài liệu liên quan đến thuật toán A* và thiết kế phần mềm ứng dụng.

Quá trình nghiên cứu bao gồm thu thập dữ liệu thời tiết dọc theo đường bay bằng phần mềm Windy, cùng với thông tin kỹ thuật về hệ thống đường hàng không trong FIR HCM để nhận diện các mối nguy có thể ảnh hưởng đến an toàn khai thác. Dựa trên cơ sở dữ liệu này, nhóm tiến hành xây dựng phần mềm ứng dụng tích hợp thuật toán A*, phát triển mã nguồn và giao diện xử lý để đề xuất đường bay thay thế. Kết quả đầu ra được đánh giá về tính khả thi và ứng dụng thực tế, làm cơ sở để xây dựng *Playbook* và tích hợp công cụ vào hệ thống ATM.

1.6 Kết cấu bài nghiên cứu

Các nội dung chính của nghiên cứu được trình bày trong 05 chương, không bao gồm phần mục lục, lời cảm ơn, lời cam kết, danh mục chữ viết tắt, danh mục bảng biểu, danh mục hình ảnh, chú thích, phụ lục và tài liệu tham khảo:

Chương 1: Tổng quan đề tài nghiên cứu

Chương 2: Lựa chọn đường bay thay thế trong quản lý không lưu

Chương 3: Thuật toán A* để xuất đường bay thay thế

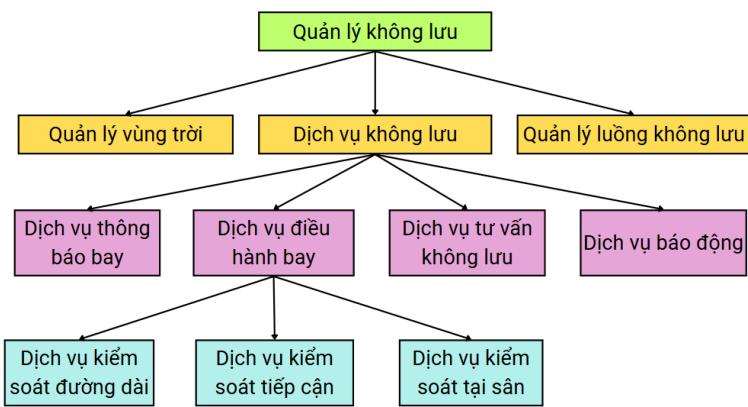
Chương 4: Xây dựng ứng dụng hỗ trợ để xuất đường bay thay thế bằng thuật toán A*

Chương 5: Kết luận và kiến nghị

CHƯƠNG 2: LỰA CHỌN ĐƯỜNG BAY THAY THẾ TRONG QUẢN LÝ KHÔNG LUU

2.1 Khái quát chung về quản lý không lưu

Quản lý không lưu (ATM) là tổng hợp các chức năng trên không và mặt đất nhằm đảm bảo tàu bay di chuyển an toàn, hiệu quả trong mọi giai đoạn bay. Theo ICAO, ATM là quá trình quản lý tích hợp và linh hoạt các hoạt động bay và vùng trời, bao gồm: dịch vụ không lưu (ATS - Air Traffic Services), quản lý vùng trời và luồng không lưu. Việc này được thực hiện thông qua cung cấp hạ tầng, dịch vụ liên mạch và phối hợp chặt chẽ giữa các bên liên quan cả dưới mặt đất và trên không [1]. ATM thường được chia thành ba thành phần cốt lõi được minh họa trong Hình 2.1.



Hình 2.1 Các thành phần cốt lõi của ATM

Dịch vụ không lưu (ATS): Đảm bảo hoạt động bay an toàn, trật tự và hiệu quả bằng cách cung cấp thông tin, cảnh báo và hướng dẫn kịp thời cho phi hành đoàn. ATS giúp ngăn ngừa va chạm thông qua phân cách chuẩn hoặc điều hành chiến thuật của KSVKL, với sự giao tiếp trực tiếp trong suốt chuyến bay [2].

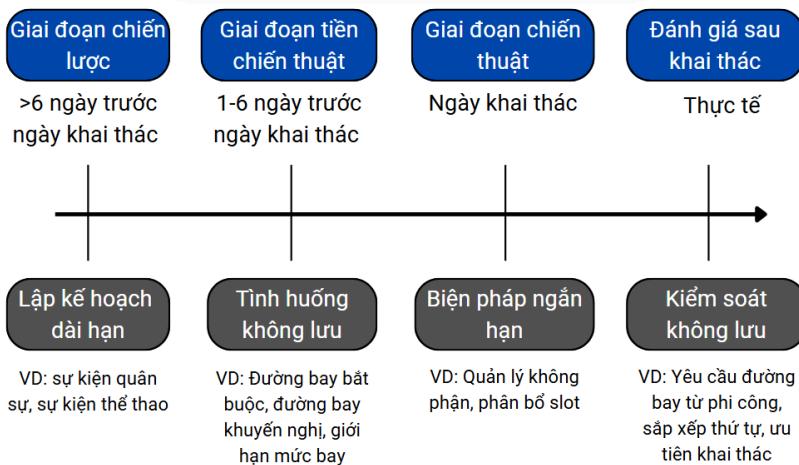
Quản lý luồng không lưu (ATFM): Điều tiết luồng để tránh quá tải tại các khu vực kiểm soát. ATRM cân bằng cung – cầu bằng cách phân bổ nhu cầu theo thời gian và không gian, đồng thời hỗ trợ lập kế hoạch năng lực kiểm soát phù hợp [2].

Quản lý vùng trời (ASM): Tối ưu việc sử dụng vùng trời nhằm đáp ứng nhu cầu đa dạng của cả dân sự và quân sự. ASM bao gồm việc phân bổ và tổ chức vùng trời để hỗ trợ cung cấp các dịch vụ không lưu hiệu quả [2].

Từ ba yếu tố cốt lõi của quản lý không lưu là ATS, ATRM và ASM, có thể thấy việc lựa chọn đường bay thay thế ngoài các tuyến hiện hữu đòi hỏi sự phối hợp chặt chẽ giữa các bên liên quan. Sự liên kết này giúp hài hòa mục tiêu dân sự – quân sự, cân bằng lợi ích hàng không với trách nhiệm kiểm soát viên, đồng thời đảm bảo an toàn tuyệt đối cho tàu bay. Nói cách khác, hệ thống ATM đóng vai trò then chốt trong thiết lập đường bay thay thế, góp phần nâng cao hiệu quả điều hành, tối ưu vùng trời và tăng khả năng ứng phó với tình huống bất thường.

2.2 Các giai đoạn chính trong lựa chọn đường bay thay thế

Việc lựa chọn đường bay thay thế không thể chỉ thực hiện tức thời trong giai đoạn khai thác chuyến bay, vì thời gian quá gấp để KSVKL kịp đánh giá toàn diện và đưa ra quyết định tối ưu. Bên cạnh đó, thay đổi đột xuất có thể ảnh hưởng đến các phép bay đã cấp và làm tăng khối lượng hiệp đồng với cơ quan quân sự. Do đó, các phương án đường bay thay thế cần được thiết kế từ trước, thông qua quá trình chuẩn bị kỹ lưỡng và trao đổi giữa các bên liên quan, với các giai đoạn chính sau:



Hình 2.2 Các giai đoạn chính trong ATM

Giai đoạn chiến lược: Diễn ra từ 6 tháng đến 1 năm trước khi áp dụng, có định hướng dài hạn. Việc lựa chọn đường bay thay thế cần kẽ hoạch chi tiết và phối hợp giữa KSVKL, quân sự và hàng hàng không. Mục tiêu là xác định khu vực ảnh hưởng, tiêu chí áp dụng và đề xuất các tuyến bay mới bằng công cụ hỗ trợ thuật toán.

Giai đoạn tiền chiến thuật: Dựa trên dữ liệu thực tế như mật độ bay, hoạt động quân sự, thời tiết, v.v, các phương án đường bay được mô phỏng và đánh giá khả thi. Đây là lúc chi tiết hóa Playbook nhằm hỗ trợ ra quyết định khi triển khai.

Giai đoạn chiến thuật: Xảy ra gần thời điểm khai thác, khi có tình huống bất thường. KSVKL áp dụng các phương án đã chuẩn bị, phối hợp nhanh với quân sự và hàng bay để đảm bảo hành trình an toàn, hiệu quả và tuân thủ quy định.

2.3 Công tác lập kế hoạch bay

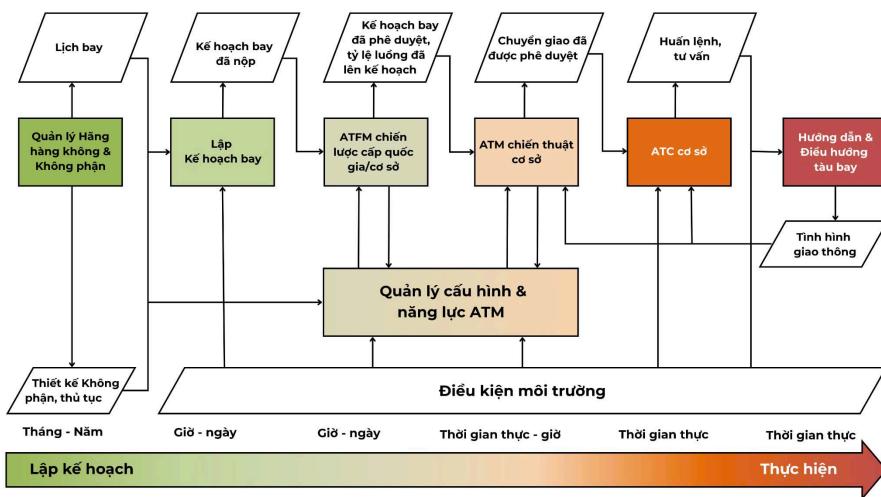
Lập kế hoạch bay là quá trình mô tả quỹ đạo dự kiến của chuyến bay, bao gồm ước tính nhiên liệu và xác định lộ trình phù hợp để đảm bảo an toàn, hiệu quả và tuân thủ kiểm soát không lưu [3]. Quy trình này được minh họa trong Hình 2.3, thể hiện các bước và luồng thông tin diễn ra trong từng giai đoạn của chuyến bay.

Trách nhiệm lập kế hoạch bay thuộc về hàng hàng không, với mục tiêu lựa chọn tuyến bay tối ưu trên cơ sở cân nhắc các yếu tố như độ dài đường bay, thời tiết,

mật độ không lưu và vùng hạn chế. Ở giai đoạn chiến lược, các hãng xây dựng lịch trình theo mục tiêu kinh doanh, thường ưu tiên các đường bay quen thuộc để đảm bảo an toàn và tuân thủ quy định.

Giai đoạn tiền chiến thuật là lúc lập kế hoạch cụ thể cho từng chuyến và gửi đến ATFM và ANSP để xét duyệt. Tuy nhiên, trong thực tế, tình trạng ùn tắc, thời tiết xấu hoặc hoạt động quân sự có thể khiến kế hoạch không còn phù hợp. Khi đó, việc điều chỉnh lộ trình là cần thiết, đòi hỏi sự phối hợp chặt chẽ giữa tổ bay, KSVKL và các công cụ hỗ trợ ra quyết định linh hoạt, chính xác trong thời gian thực [3].

Tại mốc thời gian chiến thuật, KSVKL sẽ cung cấp các thông báo và khuyến cáo cuối cùng để đảm bảo sự tách biệt an toàn, trong khi hệ thống dẫn đường máy bay cho phép thực hiện kế hoạch quỹ đạo đã thỏa thuận. Lúc này, các sản phẩm chiến thuật sẽ giúp ích rất nhiều cho công tác thay đổi kế hoạch bay trong trường hợp tàu bay buộc phải lựa chọn một đường bay khác thay thế [3].



Hình 2.3 Quy trình lập kế hoạch bay

2.4 Các yếu tố ảnh hưởng đến công tác lựa chọn đường bay thay thế

2.4.1 Tính năng tàu bay

Tính năng kỹ thuật của tàu bay là yếu tố then chốt khi lựa chọn đường bay thay thế, vì chúng quyết định mức độ phù hợp và an toàn của phương án. Các thông số như tầm bay, trần bay, tốc độ hành trình, khả năng leo cao, hoạt động trong thời tiết xấu và năng lực dẫn đường ảnh hưởng trực tiếp đến tính khả thi của tuyến bay. Ví dụ, tàu bay phản lực tầm xa có thể bay cao để tránh vùng hạn chế, trong khi tàu cánh quạt tầm ngắn bị giới hạn nhiều hơn [4].

Bên cạnh đó, tiêu thụ nhiên liệu cũng là yếu tố quan trọng. Việc lựa chọn đường bay thay thế phải đảm bảo tàu bay có đủ nhiên liệu để hạ cánh an toàn, đồng thời xem xét lượng nhiên liệu còn lại và mức tiêu thụ theo từng giai đoạn bay. Theo ICAO [4], việc lấy độ cao giữa hai mức khác biệt tiêu tốn nhiên liệu nhiều hơn so với bay ở cùng một khoảng cao độ. Hình 2.4 thể hiện tỉ lệ tiêu thụ nhiên liệu của từng loại tàu bay khác nhau trong từng giai đoạn khác nhau.

Flight phase	Taxi	En route	Arrival Management
Scheduled aviation	12.7	51.6	38.6
Regional aircraft	8.2	24.6	19.9
Narrow body aircraft	11.7	40.1	35.2
Wide body aircraft	25.8	113.9	85.2
Business aviation	NA	9.3	7.7
Rotorcraft	NA	8.8	8.8

Hình 2.4 Tỷ lệ tiêu thụ nhiên liệu trung bình (kg/phút)

2.4.2 Thời tiết

Thời tiết ở tầng khí quyển trên cao có ảnh hưởng rất lớn đến việc lựa chọn đường bay, và tính biến động theo thời gian của nó là một trong những nguyên nhân chính khiến đường bay tối ưu thay đổi theo từng ngày. Nhiệt độ không khí ảnh hưởng đến hiệu suất động cơ vì máy bay thường tiết kiệm nhiên liệu hơn khi bay ở nhiệt độ thấp, tức là bay ở độ cao lớn hơn [4].

Ngoài ra, gió là một trong yếu tố thời tiết có ảnh hưởng lớn nhất. Gió ngược hoặc gió xuôi mạnh có thể làm thay đổi đáng kể thời gian bay và mức tiêu thụ nhiên liệu.

Ví dụ, các chuyến bay xuyên Đại Tây Dương thường bị ảnh hưởng bởi dòng jetstream cực mạnh di chuyển từ Tây sang Đông. Nhờ đó, các chuyến bay hướng đông có thể rút ngắn thời gian đáng kể, trong khi các chuyến bay hướng Tây thường phải tránh dòng tia này bằng cách bay vòng lên phía Bắc hoặc xuống phía Nam. Vì vậy, khi lựa chọn đường bay thay thế, cần xem xét yếu tố gió có ảnh hưởng đến quá trình di chuyển của tàu bay hay không [4].

Thời tiết bất thường như bão, tro bụi núi lửa và nhiễu động trời trong (CAT) có thể gây gián đoạn nghiêm trọng đến hoạt động bay, buộc tàu bay thay đổi độ cao hoặc hướng đi để tránh các hiện tượng nguy hiểm như mây đối lưu, sét hay vùng áp thấp. Tro bụi núi lửa ảnh hưởng đến động cơ, tầm nhìn và cảm biến, khiến khu vực bị ảnh hưởng phải tạm đóng cửa. CAT đặc biệt nguy hiểm vì không thể phát hiện bằng radar và xuất hiện bất ngờ ở độ cao hành trình, nên việc tránh phụ thuộc vào báo cáo phi công và cảnh báo sớm. Những yếu tố này làm tăng độ phức tạp trong điều hành bay, đòi hỏi hệ thống quản lý luồng không lưu phản ứng linh hoạt và phối hợp chặt chẽ để lựa chọn đường bay thay thế [4]. Dữ liệu thời tiết được cung cấp bởi các hệ thống dự báo khí tượng hàng không của từng quốc gia, dù khác nhau nhưng đều đảm bảo độ chính xác và cập nhật. Việc tích hợp dữ liệu này vào hệ thống để xuất đường bay giúp tối ưu hóa thời gian, nhiên liệu và tăng độ an toàn, trở thành yếu tố thiết yếu trong quản lý không lưu hiện đại.

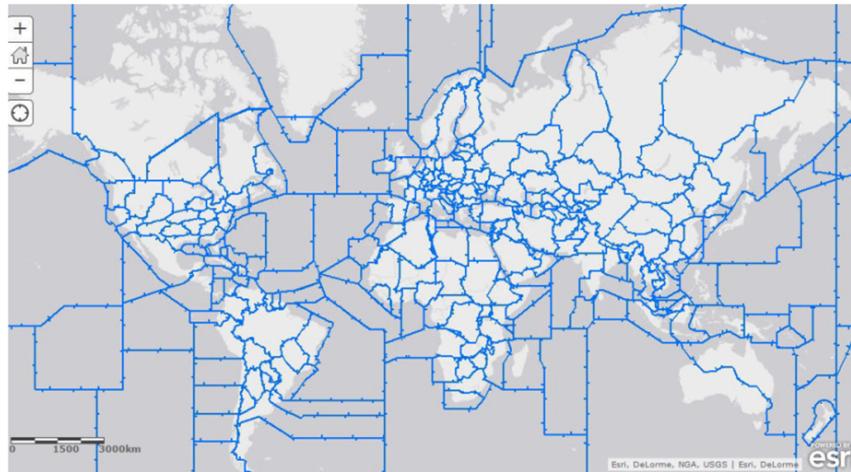


Hình 2.5 Cấu trúc đường bay thay đổi do yếu tố thời tiết

2.4.3 Cấu trúc vùng trời

ICAO đã chia không phận toàn cầu thành các FIR, qua đó xác định quốc gia chịu trách nhiệm kiểm soát một khu vực không phận cụ thể, áp dụng các thủ tục điều hành phù hợp và cung cấp dịch vụ không lưu cơ bản [5]. FIR hiện là đơn vị phân chia không phận lớn nhất được sử dụng phổ biến trên thế giới [2].

Cấu trúc vùng trời cần được thiết kế linh hoạt và kịp thời để đáp ứng đa dạng hoạt động bay và tương thích với các cấu trúc không phận liền kề, đồng thời giảm thiểu chậm trễ và duy trì mức độ an toàn cao. Không phận của mỗi quốc gia được thiết lập dựa trên một cấu hình vùng trời cụ thể, bao gồm sự phối hợp giữa các tuyến đường bay ATS, không phận tuyến tự do, tuyến đầu cuối và các khu vực liên quan, cùng với phân chia vùng kiểm soát không lưu. Cấu hình này giúp hình thành mạng lưới đường bay rõ ràng, linh hoạt và thuận tiện cho việc lựa chọn lộ trình thay thế. Nhờ đó, KSVKL có thể điều phối tàu bay kịp thời, tối ưu hóa luồng không lưu và hạn chế xung đột, đảm bảo hiệu quả vận hành trong mọi tình huống [5].



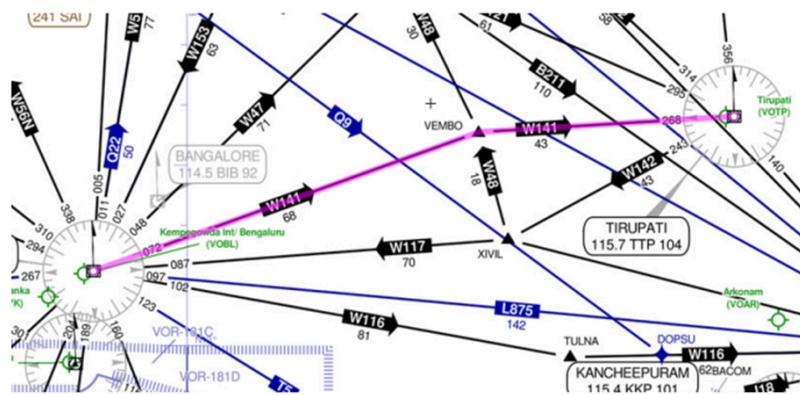
Hình 2.7 Phân chia vùng thông báo bay trên toàn thế giới

Để giúp KSVKL và phi công từng khu vực hiểu rõ cấu hình vùng trời đang áp dụng, vùng trời hiện nay được phân chia thành hai loại chính bao gồm: vùng trời có kiểm soát và vùng trời tự do [6].

2.4.3.1 Vùng trời có kiểm soát

Để đơn giản hóa công tác kiểm soát và do yếu tố lịch sử, các chuyến bay trong vùng trời có kiểm soát buộc phải tuân theo mạng lưới đường hàng không định sẵn. Mỗi đường bay ATS được xác định theo tên định danh, hướng tuyế̄n, khoảng cách giữa các điểm, yêu cầu báo cáo vị trí và độ cao an toàn tối thiểu. Khi thiết lập hay lựa chọn đường bay thay thế, các thông số này cần được xem xét đầy đủ để đảm bảo an toàn và tuân thủ quy định [6].

Trên thực tế, tàu bay thường được ưu tiên tuân thủ đúng tuyến đường đã nêu trong kế hoạch bay. Khi cần thay đổi lộ trình, KSVKL có thể điều phối tàu bay theo các đường hàng không sẵn có trong mạng lưới. Tuy nhiên, do giới hạn cấu trúc mạng lưới, các tuyến thay thế này đôi khi khá xa, gây tiêu tốn nhiên liệu và thời gian. Để khắc phục, KSVKL có thể cho tàu bay bay trực tiếp đến khu vực gần tuyến ban đầu, thay vì theo các đường hàng không cố định. Giải pháp này giúp rút ngắn hành trình, tiết kiệm nhiên liệu và đưa tàu bay quay lại lộ trình một cách hiệu quả [6].

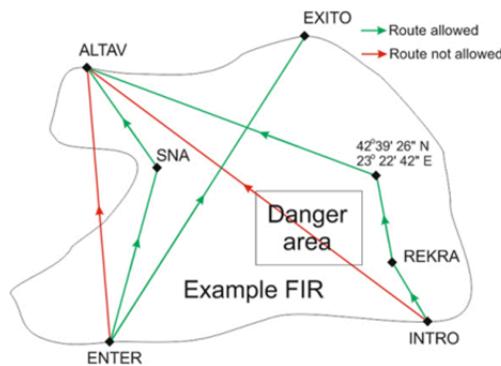


Hình 2.8 Thông số đường bay ATS

2.4.3.2 Vùng trời tự do

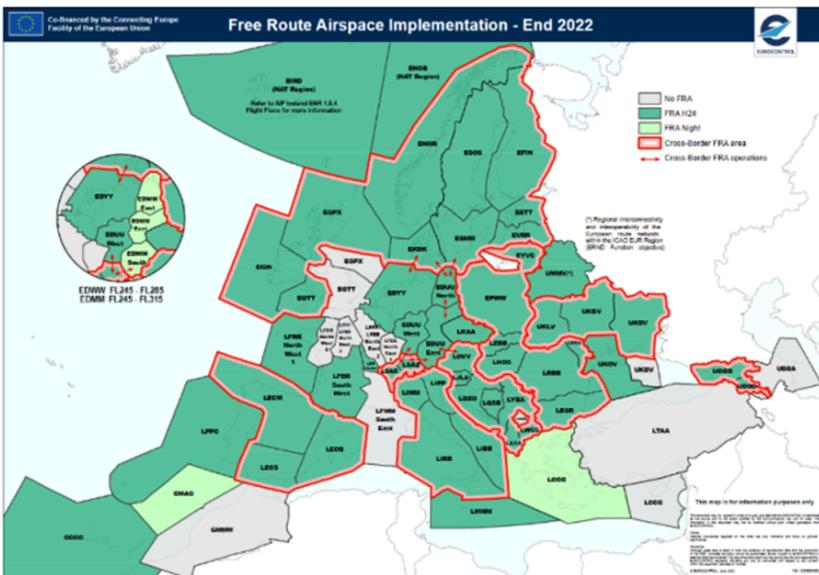
Vùng trời tự do (FRA – Free Route Airspace) là khu vực không phận cho phép các nhà khai thác tàu bay tự do lập kế hoạch tuyến bay từ điểm vào đến điểm ra đã xác định, có thể đi qua các điểm trung gian, mà không cần tuân theo mạng lưới tuyế̄n ATS. Tuy nhiên, việc lập kế hoạch vẫn phải dựa trên tính khả dụng của không phận.

Lưu ý rằng sau khi kế hoạch bay được thiết lập, máy bay phải tuân thủ đúng tuyến đã chỉ định. Do đó, sự linh hoạt chỉ tồn tại trước khi chuyến bay bắt đầu. Trong mọi trường hợp, các chuyến bay trong FRA vẫn phải chịu sự kiểm soát không lưu và tuân theo các hạn chế không phận hiện hành [7].



Hình 2.8 Ví dụ về vùng trời tự do

FRA đã được triển khai tại nhiều quốc gia và chứng minh hiệu quả rõ rệt so với việc bay cố định trên các đường hàng không truyền thống. Tuy vậy, việc triển khai FRA đòi hỏi sự phối hợp chặt chẽ giữa các cơ quan quản lý vùng trời, đặc biệt ở những quốc gia mà hàng không dân dụng và quản lý không phận do hai đơn vị riêng biệt điều hành. Yếu tố then chốt để FRA hoạt động hiệu quả là sự hài hòa và hợp tác giữa các bên liên quan, nhằm đảm bảo tính thống nhất, lợi ích chung và an toàn bay. Tại châu Âu, FRA được triển khai theo lộ trình từng bước, từ cấu trúc tuyến bay cố định sang tích hợp vùng trời tự do. Phần lớn các quốc gia bắt đầu bằng triển khai hạn chế (như vào ban đêm), sau đó mở rộng phạm vi [7]. Đến cuối năm 2022, hầu hết các quốc gia châu Âu đã áp dụng FRA, như thể hiện trong Hình 2.9.



Hình 2.9 Vùng trời tự do ở châu Âu

Nhìn chung, hai loại vùng trời đều có ưu và nhược điểm riêng. Cấu trúc không phận theo mạng đường hàng không giúp chuẩn hóa kế hoạch bay, giảm xung đột và nâng cao hiệu quả khai thác. Tuy nhiên, tính cứng nhắc của nó gây khó khăn trong việc tối ưu quỹ đạo, đặc biệt khi cần linh hoạt trước biến động về thời tiết hoặc mật độ không lưu.

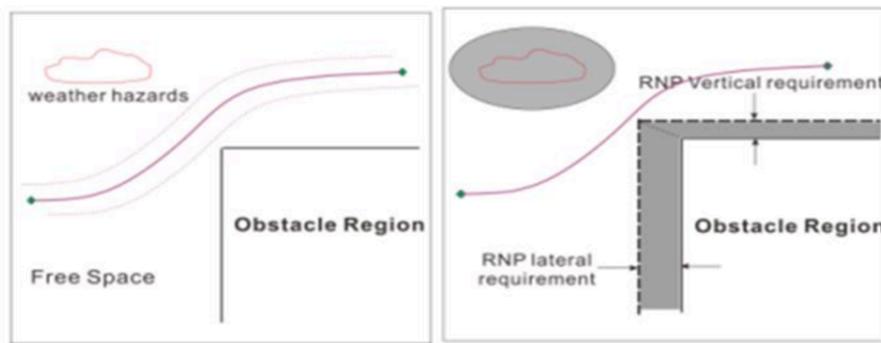
2.4.4 Hạn chế của chuyến bay trong vùng trời

Các vùng cấm, vùng nguy hiểm, vùng hạn chế và các thông báo tạm thời (NOTAM) ảnh hưởng đáng kể đến việc lựa chọn đường bay thay thế. Khi một khu vực bị hạn chế, KSVKL không thể điều phối tàu bay bay qua, dù đó là lộ trình tối ưu. Điều này buộc tàu bay phải vòng tránh, làm tăng quãng đường, tiêu hao nhiên liệu và kéo dài thời gian bay [4].

Trong các tình huống khẩn cấp hoặc cần điều chỉnh nhanh, sự hiện diện của vùng hạn chế làm giảm số phương án khả thi, gây khó khăn cho điều hành bay. Nếu các vùng này nằm gần nhau hoặc trùng với tuyến bay chính, luồng không lưu có thể bị tắc nghẽn cục bộ, làm tăng nguy cơ xung đột giữa các tàu bay [4].

2.4.5 Khoảng cách của đường bay so với chướng ngại vật

Trong thực tế, chướng ngại vật có thể là công trình nhân tạo (tòa nhà, cột tháp) hoặc yếu tố tự nhiên (núi, địa hình hiểm trở, thời tiết nguy hiểm). Bất kỳ vật thể cố định hoặc di động nào nằm trong vùng hoạt động của tàu bay hoặc vượt quá giới hạn an toàn đều được xem là chướng ngại vật. Để đơn giản hóa tính toán mà vẫn đảm bảo độ chính xác, chúng thường được mô hình hóa dưới dạng hình học cơ bản (chữ nhật, vuông, elip, đa giác) với tọa độ xác định, giúp nâng cao hiệu quả trong phân tích và kiểm tra an toàn bay.



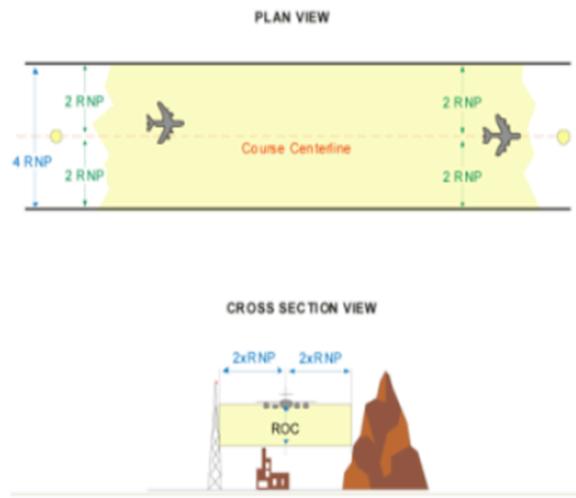
Hình 2.10 Sư mở rộng của chướng ngại vật

Để đảm bảo an toàn bay, chướng ngại vật cần được xử lý bằng cách mở rộng phạm vi ảnh hưởng theo chiều ngang và chiều cao, tùy theo phương thức dẫn đường. Việc này được thực hiện thông qua các lớp biên mở rộng, thiết lập dựa trên địa hình khu vực. Nếu đường bay chồng lấn với lớp biên, tuyến đó bị coi là không đạt chuẩn và cần thay thế. Các lớp biên này đóng vai trò như vùng đệm an toàn, dự phòng cho hoạt động của tàu bay có kích thước lớn, giúp giảm xác suất va chạm gần như bằng không khi tàu bay tiêu chuẩn sử dụng tuyến. Cách tiếp cận này không chỉ tăng độ tin cậy mà còn đơn giản hóa kiểm tra an toàn, khi chỉ cần so sánh với lớp biên thay vì toàn bộ hình khối chướng ngại vật. Nhờ đó, quá trình đánh giá và lựa chọn đường bay thay thế trở nên hiệu quả và nhẹ tính toán hơn.

2.4.6 Sự mở rộng của đường bay

Để tăng cường an toàn chướng ngại vật khi lựa chọn đường bay thay thế, cần xét đến khả năng dẫn đường tối thiểu mà tàu bay yêu cầu cho hoạt động trong vùng trời

cụ thể. Ngoài việc mở rộng vùng an toàn quanh chướng ngại vật, tàu bay cũng phải sử dụng các phương thức dẫn đường hiện đại như RNP hoặc RNAV nhằm xác định chính xác hành lang bay, từ đó đảm bảo không có chướng ngại vật xâm nhập vào giới hạn an toàn theo phương ngang.



Hình 2.11 Sự mở rộng của đường bay

Theo ICAO, vùng bảo vệ ngang cho đường bay RNP được xác định là gấp đôi giá trị RNP về mỗi phía so với trực đường bay. Đồng thời, do tàu bay hoạt động trong không gian ba chiều, vùng bảo vệ cần được mở rộng theo phương thẳng đứng, hình thành thể tích hình hộp bao quanh tuyến bay. Giới hạn dọc này được xác lập dựa trên khoảng cách tối thiểu không có chướng ngại vật, tùy thuộc vào từng loại đường bay. Đặc biệt với các đường bay RNP trong giai đoạn tiếp cận, yêu cầu về giới hạn dọc càng nghiêm ngặt hơn do tàu bay hoạt động ở độ cao thấp, dễ gặp chướng ngại vật hơn so với bay đường dài.

Tóm lại, khi đưa ra một quyết định mang tính chiến lược, các yếu tố quan trọng cần được xem xét khi lựa chọn một đường bay thay thế nhằm đảm bảo an toàn, hiệu quả và phù hợp với điều kiện khai thác thực tế, được thể hiện trong Bảng 2.1

Các yếu tố cần cân nhắc khi lựa chọn đường bay thay thế

- Sân bay khởi hành và sân bay đến
- Thời gian cất cánh dự kiến
- Dự báo thời tiết (gió, nhiệt độ, v.v)
- Loại tàu bay và đặc tính khai thác
- Mạng lưới đường hàng không toàn cầu
- Các ràng buộc vùng trời (vùng trời đóng, hạn chế tạm thời,...)
- Chi phí bay qua các vùng trời có thu phí
- Thông tin khai thác cụ thể của chuyến bay (trọng tải dự kiến, giá nhiên liệu, chi phí theo giờ, tốc độ mong muốn,...)
- Lượng nhiên liệu nạp ban đầu
- Sự mở rộng của chướng ngại vật và đường bay

Bảng 2.1 Các yếu tố cần cân nhắc khi lựa chọn đường bay thay thế

2.5 Playbook - Sản phẩm chiến thuật trong công tác lựa chọn đường bay thay thế

2.5.1 Khái quát chung về Playbook

Trong lĩnh vực quản lý không lưu nói chung và quản lý luồng không lưu nói riêng, *Playbook* đóng vai trò là một công cụ điều hành chiến lược, tiền chiến thuật và chiến thuật nhằm ứng phó linh hoạt với các tình huống ảnh hưởng đến hoạt động bay trong hệ thống quản lý vùng trời quốc gia [8].

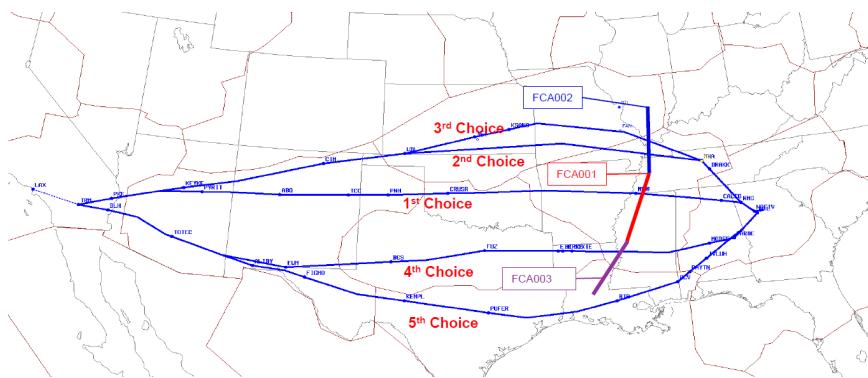
Playbook cần được xây dựng từ giai đoạn chiến lược hoặc tiền chiến thuật, vì đây là tập hợp các tuyến bay thay thế đã được thiết kế, kiểm chứng về an toàn, phối hợp với các trung tâm kiểm soát liên quan và sẵn sàng áp dụng. Việc tạo Playbook trong giai đoạn khai thác là không khả thi do quá trình đề xuất và phê duyệt tuyến bay thay thế cần thời gian, không phù hợp với các tình huống khẩn cấp. Playbook đóng vai trò quan trọng trong quản lý không lưu, đặc biệt trong các tình huống bất thường như thời tiết xấu (mưa dông, giông sét, dải hội tụ), hoạt động quân sự, vùng trời tạm thời hạn chế (NOTAM), tắc nghẽn không lưu tại khu vực cụ thể, v.v [8].

ATFM Measures			
Vertical	Strategic	Pre-Tactical	Tactical
			Rerouting (Level Capping Scenarios)
Horizontal			Fix Balancing
			Rerouting (Rerouting Scenarios)

Hình 2.12 Vị trí của Playbook trong ATFM

Không chỉ là danh sách tuyến bay thay thế, Playbook là công cụ điều phối hệ thống, giúp các cơ quan điều hành và đơn vị khai thác thông nhất phương án ứng phó. Mỗi phương án đều mô tả rõ bối cảnh ảnh hưởng, phạm vi không phận bị tác động, các bên phối hợp và tuyến bay áp dụng tương ứng [8].

Playbook hỗ trợ tối ưu hóa điều phối luồng bay trong điều kiện bị hạn chế, đồng thời duy trì an toàn, trật tự và hiệu quả hoạt động bay. Việc sử dụng các tuyến bay đã được kiểm tra trước giúp rút ngắn thời gian ra quyết định cho kiểm soát viên trong các tình huống khẩn cấp hoặc bất thường. Tuyến bay dự phòng theo Playbook còn tăng tính linh hoạt trong điều hành và đảm bảo an toàn trong các tình huống như thời tiết xấu, hoạt động quân sự hay sự kiện đột xuất khác [8].



Hình 2.13 Các phương án lựa chọn do Playbook đề xuất

2.5.2 Cơ chế hoạt động của Playbook

Xây dựng trước

Các tuyến bay trong Playbook không được tạo ra tức thời mà được thiết kế trước bởi chuyên gia, với sự hỗ trợ của công cụ tích hợp thuật toán đánh giá hiệu quả bay, dựa trên các yếu tố như thời gian, nhiên liệu, tải trọng và điều kiện thời tiết. Trong bối cảnh số hóa và tự động hóa quản lý luồng không lưu, có thể phát triển phần mềm thông minh xử lý dữ liệu thời tiết, mật độ không lưu, độ cao tối ưu và vùng cấm

bay để tự động tạo ra nhiều phương án bay thay thế. Những phương án này được phân tích, lưu trữ trong Playbook nhằm hỗ trợ ra quyết định nhanh chóng và linh hoạt trong thực tế vận hành [8].



Hình 2.14 Giao diện Playbook của FAA

Kích hoạt khi cần thiết

Khi một tình huống phát sinh làm ảnh hưởng đến hoạt động bay bình thường, Trung tâm điều phối luồng không lưu quốc gia sẽ nhanh chóng phát hành bản tin khuyến cáo (Advisory), trong đó chỉ rõ các tuyến bay thuộc Playbook được lựa chọn và áp dụng cho khu vực bị ảnh hưởng. Bản tin này đóng vai trò như một công cụ chỉ đạo chiến thuật, giúp các đơn vị kiểm soát không lưu và hàng hàng không điều chỉnh kế hoạch khai thác một cách đồng bộ, giảm thiểu chậm trễ và tránh quá tải trong vùng trời chịu tác động [8].

Current Reroutes

This page refreshes every minute. Last updated Sat, 13 Jul 2013 14:40:50 UTC

ATCSCC ADVZY 028 DCC 07/13/2013 ROUTE RQD /FL NAME: MIDWEST_TO_FLORIDA_PARTIAL CONSTRAINED AREA: ZJX VALID: ETD 131430 TO 132000	Show Advisory
ATCSCC ADVZY 027 DCC 07/13/2013 ROUTE RQD /FL NAME: OHIO_VALLEY_TO_FLORIDA_PARTIAL CONSTRAINED AREA: ZJX VALID: ETD 131430 TO 132000	Show Advisory
ATCSCC ADVZY 026 DCC 07/13/2013 ROUTE FYI NAME: PHLAYER_SOUTH CONSTRAINED AREA: ZDC VALID: ETD 131400 TO 140200	Show Advisory
ATCSCC ADVZY 023 DCC 07/13/2013 FCA RQD NAME: FCA004:NASCAR_NEW_HAMPSHIRE CONSTRAINED AREA: ZBW VALID: N/A	Show Advisory

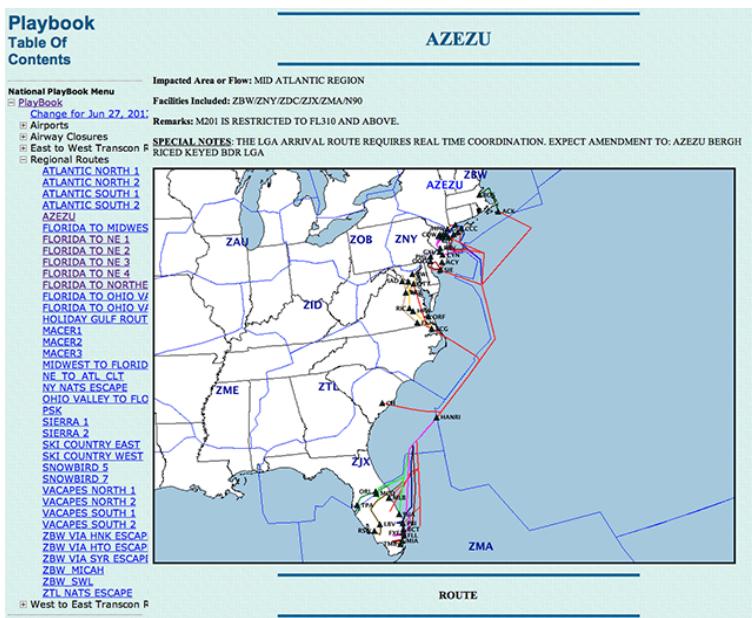
Hình 2.15 Bản tin khuyến cáo đổi đường bay trên Playbook

Quyền sử dụng và điều chỉnh

Khi một tuyến bay trong Playbook được kích hoạt, các tổ lái hoặc đơn vị lập kế hoạch bay có thể chủ động sử dụng nếu phù hợp với hành trình dự kiến. Việc áp dụng các tuyến này giúp tối ưu hóa luồng không lưu và giảm thiểu xung đột trong vùng trời bị ảnh hưởng. Nếu một chuyến bay không nộp kế hoạch theo đúng tuyến trong Playbook đang còn hiệu lực, cơ quan điều hành bay sẽ chủ động điều chỉnh lại đường bay cho phù hợp. Thông tin điều chỉnh sẽ được thông báo kịp thời đến phi công, trước khi khởi hành hoặc trong quá trình bay, đảm bảo tính nhất quán và an toàn điều phối. Bên cạnh đó, công tác phối hợp với cơ quan quản lý vùng trời đã được thiết lập trước, góp phần tạo điều kiện thuận lợi cho việc áp dụng nhanh các thay đổi tuyến bay [8].

2.5.3 Cấu trúc cơ bản của Playbook

Trong cấu trúc một tuyến *Playbook*, các thành phần cơ bản được trình bày rõ ràng nhằm phục vụ hiệu quả cho công tác điều phối luồng không lưu trong các tình huống bất thường như thời tiết xấu hoặc tắc nghẽn vùng trời [9]:



Hình 2.16 Cấu trúc cơ bản của Playbook

- **Khu vực hoặc luồng không lưu bị ảnh hưởng:** xác định phạm vi địa lý nơi áp dụng thay đổi, là khu vực chịu tác động cần thiết lập tuyến bay thay thế [9].
- **Các cơ sở điều hành bay liên quan:** liệt kê các Trung tâm Kiểm soát Vùng thông báo bay chịu trách nhiệm kiểm soát và cung cấp dịch vụ không lưu cho tuyến bay này [9].
- **Giới hạn và lưu ý:** bao gồm các điều kiện ràng buộc hoạt động bay như giới hạn độ cao tối thiểu/tối đa, yêu cầu phối hợp thời gian thực, hoặc điều chỉnh tuyến đến theo tình hình thực tế [9].
- **Bản đồ tuyến bay:** minh họa trực quan các tuyến thay thế được đề xuất, thể hiện rõ điểm đầu, điểm cuối, các điểm trọng yếu trung gian; đồng thời sử dụng màu sắc phân biệt để nhận diện từng tuyến riêng biệt [9].
- **Mô tả tuyến đường:** cung cấp thông tin chi tiết tương ứng với bản đồ, liệt kê đầy đủ lộ trình các điểm mà tàu bay đi qua theo từng phương án định sẵn [9].

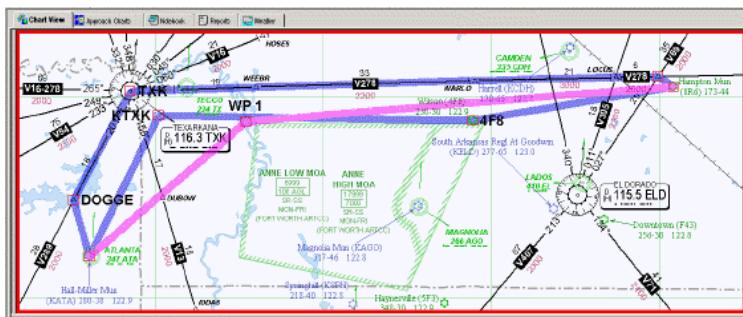
Một tuyến bay trong Playbook không đơn thuần là một đường bay liền mạch, mà được cấu thành từ nhiều đoạn đường bay riêng biệt, được xác định bằng cách kết hợp linh hoạt giữa các thành phần quan trọng của dữ liệu hành trình. Cụ thể, mỗi tuyến đường có thể bao gồm điểm kết thúc của phương thức đến tiêu chuẩn (STAR)

kèm theo số hiệu, các đài dẫn đường vô tuyến, điểm giao nhau, điểm cố định, hoặc các điểm định vị khác. Ngoài ra, việc sử dụng các đường hàng không cũng là thành phần không thể thiếu, với tên gọi thường bao gồm ký hiệu loại và số hiệu cụ thể [8].

ORIGIN	FILTERS	ROUTE	DEST	REMARKS
ZLC		HVE J28 GCH ICT J182 RZC MEM HUTCC KNSAW RUSSA GLAVN1	KATL	SOUTH OPTION
ZLA	-SAN -LAS	TRM BLH J169 TFD J50 SSO EWM J66 ABI J4 FUZ Q184 MERDN ORRKH HOBTT2	KATL	
KSAN		IPL J2 GBN J50 SSO EWM J66 ABI J4 FUZ Q184 MERDN ORRKH HOBTT2	KATL	
KLAS		INV J86 ELP J50 ABI J4 FUZ Q184 MERDN ORRKH HOBTT2	KATL	
ZAB		EWM J66 ABI J4 FUZ Q184 MERDN ORRKH HOBTT2	KATL	
ZFW	-OKC	MEI ORRKH HOBTT2	KATL	
KOKC		FSM MEM HUTCC KNSAW RUSSA GLAVN1	KATL	

Hình 2.17 Mô tả tuyến đường của từng phương án trong Playbook

Việc mô tả một tuyến bay trong Playbook phải tuân thủ một số nguyên tắc cấu trúc nhất định để đảm bảo tính khả thi về mặt điều hành và an toàn bay. Ví dụ, nếu thiết kế một đường bay thay thế trong quá trình tiếp cận, khi sử dụng các phương thức khởi hành hoặc đến, tuyến bay phải bao gồm điểm chuyển tiếp đã được công bố hoặc điểm cố định chung của phương thức đó. Tương tự, trong quá trình bay đường dài, nếu tuyến có chứa đường hàng không, phải xác định rõ điểm vào và điểm ra của đường hàng không tương ứng (ví dụ: AC W1 PLK) [8].



Hình 2.18 Thông tin cơ bản của từng đường bay trong Playbook

Trong trường hợp tuyến bay liên kết hai đường hàng không khác nhau, cần đảm bảo rằng tồn tại một điểm cố định đã được công bố, chung cho cả hai đường và có thể tìm thấy trên bản đồ bay đường dài (ví dụ: AC W1 BMT W12). Nếu không có điểm chung, thì bắt buộc phải liệt kê điểm ra đã công bố của đường đầu tiên và điểm vào đã

công bố của đường thứ hai. Điều này giúp tránh nhầm lẫn và đảm bảo tính liên tục của đường bay trong hệ thống kiểm soát không lưu [8].

Một số quy định về giới hạn cũng được áp dụng để chuẩn hóa hình thức của các tuyến Playbook. Cụ thể, điểm đầu tiên của tuyến bay sau sân khởi hành không được phép là một đường hàng không, và tương tự, điểm cuối cùng trước sân đến cũng không được là một đường hàng không. Ngoài ra, không được sử dụng các đài vô hướng (ví dụ: NDB) trong bất kỳ đoạn tuyến nào của Playbook [8].

2.6 *Tổng quan tình hình nghiên cứu*

2.6.1 **Tình hình nghiên cứu ở nước ngoài**

Đã có nhiều nghiên cứu nhằm tìm ra phương pháp tối ưu hóa đường bay trong hoạt động hàng không, với các hướng tiếp cận đa dạng như mô hình toán học, mô phỏng quỹ đạo, và thuật toán tối ưu đa mục tiêu. Sau khi xây dựng mô hình, các nghiên cứu thường đánh giá hiệu quả thông qua dữ liệu thực tế hoặc giả lập để phân tích các yếu tố ảnh hưởng đến hiệu suất bay. Một số kết quả nghiên cứu tiêu biểu trong lĩnh vực tối ưu hóa đường bay đã được công bố như sau:

- “Lựa chọn đường bay tối ưu và lập kế hoạch bay 3 chiều” (1974) của H.M. de Jong [10]

Nghiên cứu này tập trung vào ứng dụng thuật toán trong quản lý đường bay. Thay vì sử dụng các thuật toán điều khiển tối ưu khó triển khai do hạn chế từ mạng lưới đường hàng không và các ràng buộc thực tế, tác giả đề xuất sử dụng thuật toán Dijkstra ba chiều. Phương pháp này khai thác dữ liệu hiệu suất máy bay và dự báo thời tiết ở định dạng gần với thực tế hiện nay, đồng thời cung cấp các kết quả minh họa để chứng minh tính khả thi của hướng tiếp cận.

- “Phương pháp lập trình động mềm để tối ưu hóa quỹ đạo 4D của máy bay trực tuyến” (1998) của Patrick Hagelauer và cộng sự [11]

Báo cáo này kể thura các nghiên cứu trước và áp dụng kiến thức nền tảng để xây dựng mô hình tối ưu hóa mức tiêu thụ nhiên liệu trong giai đoạn bay hành trình trên tuyến đường ngang cố định. Phương pháp lập trình động được sử dụng để tìm giải pháp tối ưu, với các ràng buộc thời gian được xử lý bằng cách loại bỏ sớm các trạng thái không phù hợp. Nhằm tăng tốc độ tính toán, nhóm tác giả tích hợp mạng nơ-ron, giúp cải thiện hiệu suất tính toán nhiên liệu lên đến 8,5 lần.

- “Kết hợp NLP và MILP trong lập kế hoạch bay theo phương thẳng đứng” (2015) của Liana Amaya Moreno và cộng sự [12]

Các nghiên cứu này tập trung vào tối ưu cấu hình theo chiều dọc bằng cách đơn giản hóa bài toán, chủ yếu sử dụng lập trình hình nón bậc hai và phi tuyến, đôi khi được rót rắc hóa thành mô hình MIP. Tuy nhiên, do phạm vi hạn chế và thời gian chạy dài, các phương pháp này chưa phù hợp để áp dụng thực tế.

- “Phương pháp kiểm soát tối ưu hỗn hợp nhiều giai đoạn để tối ưu hóa quỹ đạo máy bay” (2013) của Pierre Bonami và cộng sự [13]

Bài nghiên cứu mô hình hóa trạng thái máy bay bằng phương trình vi phân và biểu diễn bài toán dưới dạng MINLP, giải bằng thuật toán nhánh và giới hạn. Đây là một trong số ít công trình xét đến phí bay quá mức trong quá trình tối ưu. Tuy nhiên, với thời gian chạy gần 10 phút cho một chuyến bay trên đồ thị nhỏ 40 điểm dừng, thuật toán chưa đủ hiệu quả để áp dụng thực tế nếu không có cải tiến đáng kể.

- “Tối ưu hóa theo chiều thẳng đứng của đường bay phụ thuộc vào tài nguyên” (2016) của Anders Nicolai Knudsen và cộng sự [14]

Bài viết tập trung tối ưu hóa cấu hình thẳng đứng cho một tuyến đường ngang. Một thuật toán A* với hiệu suất thời gian cao được trình bày. Dù bài toán không đáp ứng tính chất FIFO do ảnh hưởng của gió theo thời gian, kết quả cho thấy các thuật toán dựa trên FIFO chỉ mất đi rất ít, nếu có, tính tối ưu.

- “Các biến thể heuristic của thuật toán A* tìm kiếm đường bay cho kế hoạch bay 3D” (2018) của Kim S. Larsen và cộng sự [15]

Bài viết giới thiệu một số biến thể của thuật toán A* trên đồ thị ba chiều. Thuật toán này xem xét gần như toàn bộ độ phức tạp của bài toán và đạt kết quả tính toán hiệu quả. Các hạn chế được giải quyết thông qua kỹ thuật đa nhãn.

2.6.2. Tình hình nghiên cứu tại Việt Nam

Tại Việt Nam, việc tối ưu hóa đường bay bằng các thuật toán đã được thử nghiệm trong một số báo cáo nghiên cứu như sau:

- “Phát triển công cụ hỗ trợ quyết định cho hệ thống lập kế hoạch tuyến bay” (2022) của Nhựt Ngô và cộng sự [16]

Bài báo cáo tập trung phân tích hiệu quả của thuật toán A* trong việc tối ưu hóa đường bay tại Việt Nam dựa trên nền tảng vùng trời tự do. Các mô hình và trang web cho người dùng được xây dựng nhằm triển khai quy trình ứng dụng cho một đường bay cụ thể khi bay qua các vùng có chướng ngại vật

- “Lập kế hoạch đường bay cho UAV đa cánh quạt sử dụng thuật toán tối ưu hóa dựa trên dạy-học” (2022) của Hoàng Văn Trường và cộng sự [17]

Bài báo đề xuất một thuật toán lập kế hoạch đường bay cho UAV đa cánh quạt dựa trên phương pháp tối ưu hóa dạy-học (Teaching-Learning-Based Optimization - TLBO). Thuật toán này nhằm tạo ra lộ trình bay khả thi, tránh chướng ngại vật và đáp ứng các yêu cầu nhiệm vụ cụ thể.

2.7.3 Đánh giá về tính hình nghiên cứu

Đối với các nghiên cứu nước ngoài

Các nghiên cứu cho thấy sự đa dạng trong cách tiếp cận bài toán tối ưu hóa đường bay, từ các thuật toán cổ điển như Dijkstra 3D đến các kỹ thuật hiện đại như A*, MINLP, NLP, lập trình động và tối ưu hóa hỗn hợp. Nhiều mô hình đã khai thác dữ liệu thực tế như thời tiết, hiệu suất máy bay và ràng buộc mạng lưới để tăng tính ứng dụng, trong đó một số đạt cải tiến đáng kể như tăng tốc bằng mạng nơ-ron hay kỹ

thuật thống trị nhãn. Một số nghiên cứu mở rộng phạm vi xét đến phí bay, vùng trời tự do hoặc ràng buộc luồng không lưu.

Đáng chú ý, một vài công trình đã xây dựng hệ thống có giao diện web để hỗ trợ đề xuất tuyến bay thay thế – đặt nền móng cho việc phát triển Playbook trong thực tế. Tuy nhiên, phần lớn nghiên cứu vẫn còn hạn chế về tính ứng dụng do giả định đơn giản (thời tiết tĩnh, độ cao cố định), thời gian chạy dài, hoặc thiếu kết quả thực nghiệm. Các phương pháp hiện đại như NLP–MILP hay tối ưu rời rạc–liên tục vẫn cần được cải thiện về hiệu năng để tích hợp hiệu quả vào hệ thống điều hành bay.

Đối với các nghiên cứu tại Việt Nam

Các nghiên cứu tại Việt Nam về tối ưu hóa đường bay cho thấy nỗ lực đáng kể trong việc ứng dụng các thuật toán hiện đại, đặc biệt trong lĩnh vực UAV và mô phỏng bay 3D. Nhiều công trình sử dụng các thuật toán như A*, PSO, TLBO hoặc AI, đồng thời kết hợp dữ liệu thực tế như địa hình 3D, thời tiết và giới hạn không phận, góp phần xây dựng nền tảng cho các công cụ lập kế hoạch bay bán tự động. Một số nghiên cứu đã đạt cài tiến rõ rệt về tốc độ và độ chính xác so với phương pháp truyền thống.

Tuy nhiên, các công trình trong nước vẫn chủ yếu dừng ở mức mô phỏng UAV, chưa áp dụng được vào điều hành bay dân dụng thực tế. Hầu hết nghiên cứu mang tính học thuật, thiếu dữ liệu thực nghiệm, chưa xét đến các yếu tố như phí bay, phân bổ luồng không lưu hay khả năng tích hợp hệ thống. Hiện chưa có phần mềm cụ thể nào được phát triển thành sản phẩm có thể tích hợp vào hệ thống kiểm soát bay quốc gia, cho thấy khoảng cách giữa nghiên cứu và ứng dụng thực tiễn vẫn còn khá lớn.

CHƯƠNG 3: THUẬT TOÁN A* ĐỂ XUẤT ĐƯỜNG BAY THAY THẾ

3.1 Các ràng buộc của công tác lập kế hoạch bay

Bài toán lập kế hoạch bay để đề xuất ra được một đường bay thay thế tối ưu là xác định các biến quỹ đạo bay từ trạng thái ban đầu đã cho đến trạng thái cuối cùng được xác định trước, đồng thời thỏa mãn tất cả các ràng buộc và tối thiểu hóa chỉ số hiệu suất. Các ràng buộc có thể thuộc nhiều loại khác nhau [18]. Bao gồm:

- + Ràng buộc trên đường bay (interior path constraints): Các điểm cố định mà tàu bay phải bay qua, tuy nhiên các biến trạng thái tại đó như độ cao, vận tốc, hướng bay,... có thể được chỉ định cụ thể hoặc không có tùy vào yêu cầu của nhiệm vụ [18].
- + Các vùng không thể bay qua (unusable airspace): Bao gồm vùng cấm bay (no-fly zones) hoặc khu vực có nhiễu động mạnh hoặc thời tiết xấu, cần phải tránh hoàn toàn trong quỹ đạo bay [18].
- + Thủ tục cất/hạ cánh do kiểm soát không lưu (ATC) quy định: Các thủ tục này được chia thành các đoạn bay (segments), trong đó một hoặc nhiều biến trạng thái được giữ cố định. Mỗi đoạn có điều kiện chuyển tiếp (capture conditions) xác định khi nào chuyển sang đoạn tiếp theo [18].
- + Các hạn chế liên quan đến tính năng tàu bay: Phản ánh mức độ thoải mái của hành khách, đặc biệt quan trọng với các chuyến bay thương mại như giới hạn gia tốc, góc nghiêng, độ dốc lên/xuống, v.v [18].

Vì vậy, để giải bài toán này, lời giải cần hiệu quả về mặt tính toán, linh hoạt, và đáp ứng đầy đủ các ràng buộc đã nêu. Do đó, cần một thuật toán có khả năng xử lý nhiều bài toán khác nhau để có thể đề xuất ra được một đường bay thay thế đồng thời đáp ứng được các yêu cầu về ràng buộc.

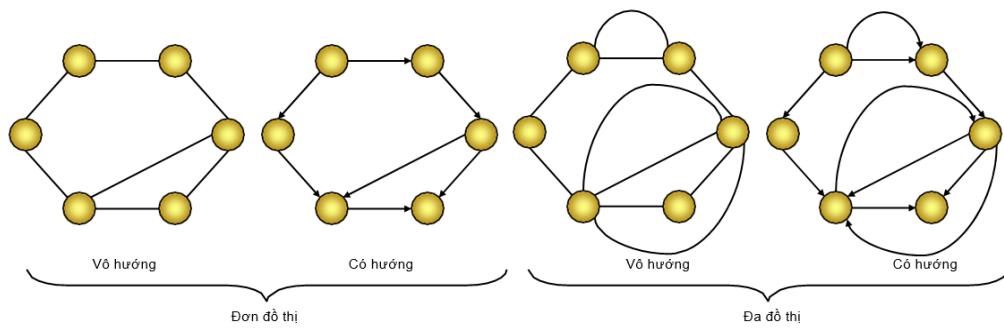
3.2 Khái quát về thuật toán tìm đường bay ngắn nhất

4.2.1 Khái quát chung về đồ thị

Đồ thị (graph) là một cấu trúc dữ liệu cơ bản và quan trọng trong khoa học máy tính, toán học và nhiều lĩnh vực kỹ thuật. Về mặt định nghĩa, một đồ thị là tập hợp các đối tượng được gọi là đỉnh hoặc nút (*vertices* hoặc *nodes*), được kết nối với nhau

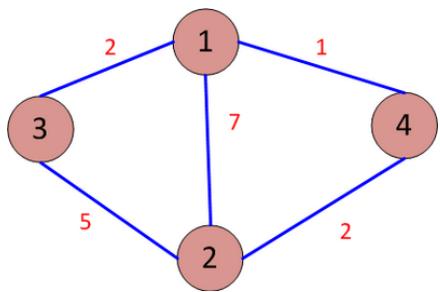
thông qua các cạnh (*edges*). Các cạnh này đại diện cho mối quan hệ hoặc sự liên kết giữa các cặp đỉnh trong đồ thị [18].

Tùy theo mối liên kết giữa các đỉnh, cạnh trong đồ thị có thể là có hướng hoặc vô hướng. Cạnh vô hướng biểu thị quan hệ hai chiều giữa hai đỉnh (A – B và B – A là như nhau), trong khi cạnh có hướng thể hiện quan hệ một chiều (chỉ đi từ A đến B). Đồ thị với các cạnh có hướng được gọi là đồ thị có hướng (*directed graph*). Đồ thị thường được minh họa bằng các đỉnh là điểm và các cạnh là đoạn thẳng hoặc mũi tên nối các đỉnh, tùy thuộc vào tính hướng của đồ thị [18].

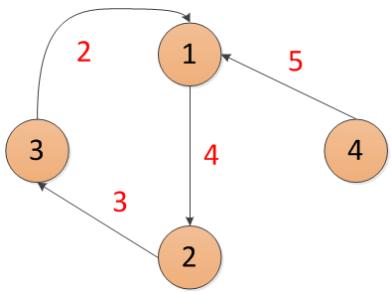


Hình 3.1 Ví dụ về đồ thị vô hướng và có hướng

Một trong những cách mở rộng phổ biến và hữu ích của đồ thị là gán trọng số (*weights*) cho các cạnh. Trọng số là một giá trị số thể hiện mức độ, chi phí, khoảng cách hoặc thời gian liên quan đến mối liên kết giữa hai đỉnh. Khi mỗi cạnh mang một trọng số, ta thu được đồ thị có trọng số (*weighted graph*). Cấu trúc này cho phép mô hình hóa các hệ thống phức tạp và thực tế hơn. Ví dụ, trong mạng lưới giao thông, các trọng số có thể biểu thị độ dài tuyến đường, thời gian di chuyển, hoặc mức độ tắc nghẽn giao thông giữa các vị trí. Trong thực tế sẽ giúp chúng ta thường giải quyết được những tình huống như từ địa điểm A đến địa điểm B trong thành phố, chúng ta chọn đường đi ngắn nhất (xét đến độ dài), có lúc lại cần chọn đường đi nhanh nhất (xét đến thời gian) và có lúc phải cân nhắc để chọn đường đi có chi phí thấp nhất, v.v. Vì vậy sự xuất hiện của giá trị trọng số sẽ đóng vai trò then chốt để thực hiện những bài toán này [18].

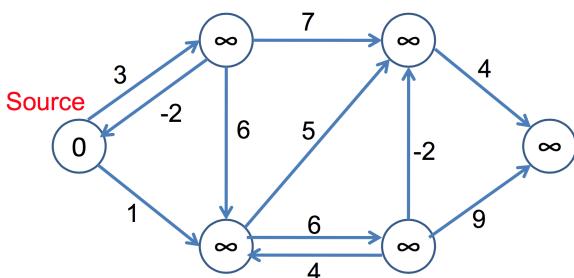


Hình 3.2 Đồ thị có trọng số vô hướng



Hình 3.3 Đồ thị có trọng số có hướng

Khi kết hợp cả hai đặc tính có hướng và có trọng số, ta thu được một loại đồ thị đặc biệt gọi là lưới (grid), thường dùng để mô phỏng mạng giao thông một chiều có khoảng cách cụ thể, cũng như các hệ thống viễn thông, máy tính hoặc phân phối năng lượng. Trong hàng không, bài toán tìm đường bay thay thế tối ưu là một bài toán đồ thị điển hình, yêu cầu thuật toán xác định lộ trình ngắn nhất, an toàn và tiết kiệm chi phí, nhất là khi cần thay đổi kế hoạch bay do thời tiết, sự cố kỹ thuật hoặc giới hạn không phận [19].



Hình 3.4 Một lưới (grid) gồm các thông số về đỉnh, cạnh, hướng và trọng số

4.2.2 Bài toán tìm đường ngắn nhất

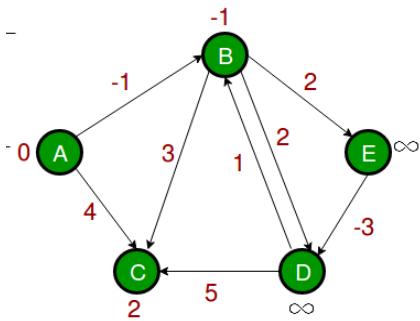
Bài toán định tuyến đường đi ngắn nhất nhằm tìm lộ trình tối ưu giữa một cặp điểm nguồn–đích trong mạng lưới được mô hình hóa bằng đồ thị, sao cho tổng trọng số các cạnh là nhỏ nhất. Trong nghiên cứu này, mục tiêu là tìm tuyến bay thay thế ngắn nhất khi tuyến gốc không khả thi, đồng thời đảm bảo an toàn và phù hợp với khai thác. Trọng số của đồ thị được xác định theo khoảng cách địa lý giữa các điểm, chưa xét đến chi phí nhiên liệu.

Lý thuyết và các lời giải cho bài toán đường đi ngắn nhất (shortest path problem) đã xuất hiện từ lâu đời và hiện nay đã phát triển khá hoàn thiện. Các thuật toán nổi tiếng được sử dụng để giải bài toán này bao gồm Dijkstra, Bellman-Ford, A*, Floyd-Warshall, Johnson hay Lý thuyết nhiễu (Perturbation theory). Tuy nhiên mỗi thuật toán sẽ đều có các đặc điểm nhất định được thể hiện qua bảng sau [18]:

Thuật toán	Loại bài toán giải quyết	Ưu điểm chính	Nhược điểm chính
Bellman-Ford	Đơn nguồn, có thể có trọng số âm	Xử lý được trọng số âm, phát hiện chu trình âm	Chậm hơn các thuật toán khác, không hiệu quả với đồ thị lớn
Dijkstra	Đơn nguồn, không trọng số âm	Nhanh, hiệu quả với trọng số dương	Không xử lý được trọng số âm, không phù hợp với mọi tình huống, cần tìm <i>mỗi cặp → châm</i>
Floyd-Warshall	Mọi cặp đỉnh, có trọng số âm	Giải toàn bộ bảng đường đi ngắn nhất, triển khai dễ	Độ phức tạp rất cao, không phù hợp với đồ thị lớn
Johnson	Mọi cặp đỉnh, có trọng số âm	Hiệu quả với đồ thịitura, xử lý trọng số âm, kết hợp ưu điểm Dijkstra	Phức tạp khi triển khai, cần bước tiền xử lý bằng Bellman-Ford
A*	Đơn nguồn → đích cụ thể, có heuristic, dựa trên cơ sở Dijkstra	Tìm đích nhanh nhờ heuristic, ứng dụng rộng trong robot, GIS, trò chơi	Cần xây dựng hàm heuristic tốt

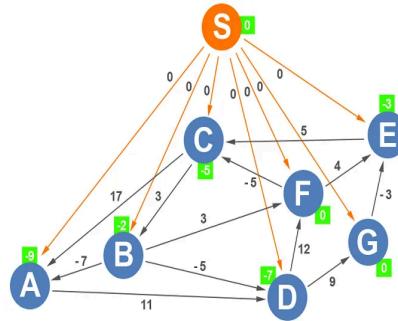
Hình 3.5 Bảng so sánh các thuật toán tìm đường phô biến

Trong các thuật toán tìm đường đi ngắn nhất, A* được xem là phù hợp nhất cho bài toán xác định đường bay thay thế giữa hai sân bay cụ thể, nhờ khả năng kết hợp giữa thuật toán Dijkstra và hàm heuristic. Nhờ đó, A* có thể ưu tiên mở rộng các nút tiềm cận đích, giúp rút ngắn thời gian tìm kiếm thay vì phải duyệt toàn bộ đồ thị. Điều này đặc biệt hiệu quả trong hàng không, nơi mỗi nút là một điểm báo cáo hoặc sân bay, và các cạnh có trọng số dương như thời gian, chi phí hoặc khoảng cách. Ngoài ra, do trọng số luôn không âm, A* tránh được các hạn chế của Bellman-Ford và không tính toán dư thừa như Floyd-Warshall hay Johnson [18].



Hình 3.6

Thuật toán Bellman-Ford với trọng số âm

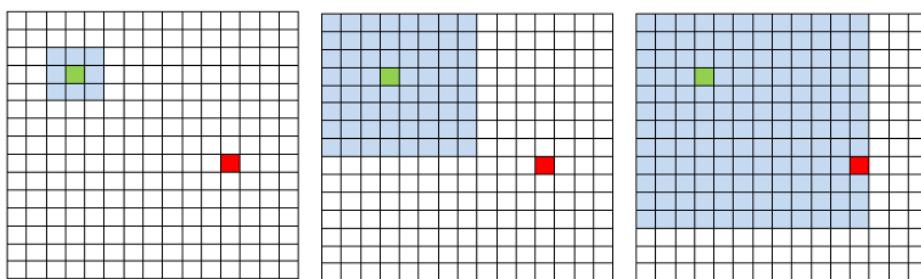


Hình 3.7

Thuật toán Johnson

4.3 Thuật toán tìm đường ngắn nhất của Dijkstra

Thuật toán Dijkstra là một phương pháp tham lam dùng để tìm đường đi ngắn nhất từ một đỉnh xuất phát đến các đỉnh còn lại trong đồ thị có trọng số không âm. Trong hàng không, nếu xem các đỉnh là sân bay và trọng số là khoảng cách hoặc thời gian bay, thuật toán này giúp xác định đường bay hiệu quả giữa các sân bay. Thuật toán khởi đầu bằng cách gán chi phí vô hạn cho tất cả các đỉnh, trừ đỉnh xuất phát có giá trị 0. Ở mỗi bước, nó chọn đỉnh chưa xét có chi phí nhỏ nhất, cập nhật chi phí các đỉnh lân cận nếu tìm được đường đi tốt hơn, đồng thời ghi nhận “nút cha” để truy vết lộ trình. Quá trình lặp lại cho đến khi tìm được đường đi đến đích. Nhờ cách tiếp cận này, Dijkstra luôn đảm bảo tìm được đường đi ngắn nhất nếu tồn tại [18].



Hình 3.8 Ba giai đoạn triển khai của thuật toán Dijkstra sử dụng đồ thị dạng lưới

Tuy nhiên, hiệu năng của Dijkstra bị giới hạn trong các bài toán lớn. Thuật toán gốc có độ phức tạp tính toán là $O(n^2)$ nếu sử dụng mảng đơn giản, hoặc $O((n + m) \log n)$ nếu sử dụng hàng đợi ưu tiên (heap), với n là số đỉnh và m là số cạnh. Trong thực

tế, khi số lượng đỉnh và cạnh tăng lên đáng kể, Dijkstra có xu hướng trở nên kém hiệu quả vì không có cơ chế ưu tiên hướng về đích. Nó mở rộng tìm kiếm một cách đồng đều, bao gồm cả các vùng không có khả năng góp phần vào lời giải, làm tăng đáng kể thời gian xử lý trong không gian tìm kiếm lớn [18].

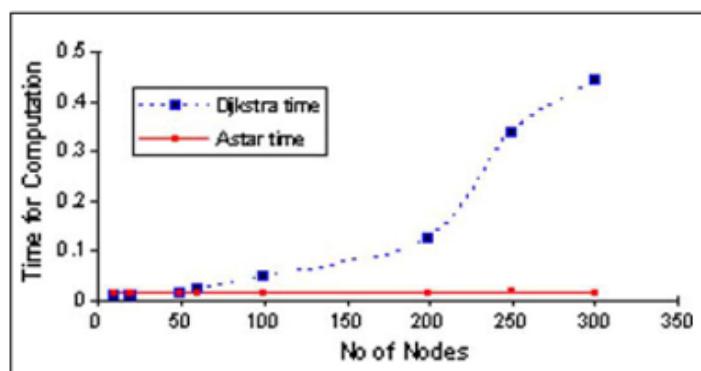
Iteration	Unvisited	Visited	Current	Neighbors	Node (distance, parent node)						
					1	2	3	4	5	6	7
1	{1,2,3,4,5,6,7}	{}			(0, -)	(∞ , -)	(∞ , -)	(∞ , -)	(∞ , -)	(∞ , -)	(∞ , -)
2	{2,3,4,5,6,7}	{1}	1	{2,4}		(33, 1)	(∞ , -)	(95, 1)	(∞ , -)	(∞ , -)	(∞ , -)
3	{3,4,5,6,7}	{1,2}	2	{3,4}			(188, 2)	(95, 1)	(∞ , -)	(∞ , -)	(∞ , -)
4	{3,5,6,7}	{1,2,4}	4	{3,6}				(188, 2)	(∞ , -)	(195, 4)	(∞ , -)
5	{5,6,7}	{1,2,3,4}	3	{5,6}					(293, 3)	(195, 4)	(∞ , -)
6	{5,7}	{1,2,3,4,6}	6	{5,7}					(293, 3)		(272, 6)
7	{5}	1,2,3,4,6,7	7								

Hình 3.9 Thuật toán Dijkstra phiên bản cơ bản, có độ phức tạp $O(n^2)$

Hình 3.9 cho thấy mạng lưới gồm 7 nút cho thấy thuật toán cần qua 7 vòng lặp để tìm đường từ Dallas đến San Antonio. Mặc dù kết quả chính xác, nhưng so với các thuật toán có định hướng như A*, Dijkstra không tận dụng được thông tin vị trí để rút ngắn quá trình tìm kiếm, dẫn đến hiệu suất kém hơn trong các tình huống phức tạp.

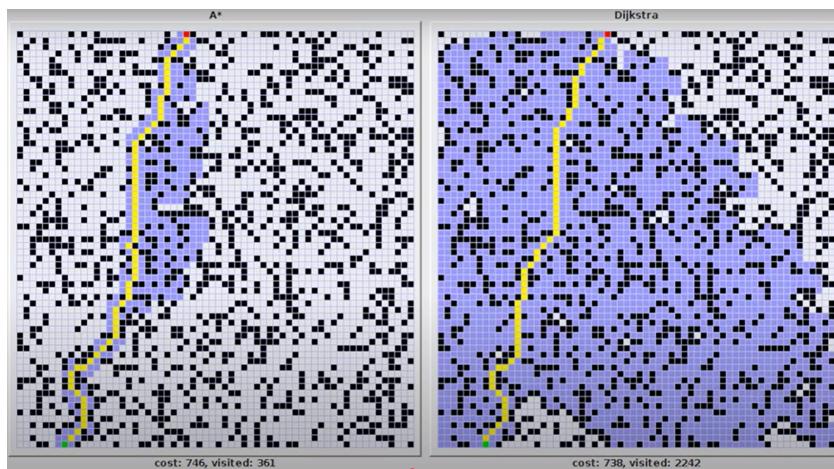
Tuy nhiên, bản chất thuật toán A* tương tự Dijkstra, nhưng A* tối ưu hơn vì khắc phục được nhược điểm về tính toán dư thừa của Dijkstra, trong đó:

Thứ nhất, thuật toán A* có thời gian xử lý nhanh hơn do sử dụng hàm đặc trưng (heuristic), yếu tố mà Dijkstra không có (Soltani et al., 2003). Nếu hàm này được thiết kế tốt, A* có thể tìm ra kết quả “tốt” rất nhanh; ngược lại, nếu đánh giá chưa chính xác, thuật toán sẽ mất nhiều thời gian hơn hoặc không đảm bảo tối ưu (Patel, 2011) [19].



Hình 3.12. Thời gian xử lý giữa thuật toán A-Star và thuật toán Dijkstra

Thứ hai, phạm vi xét của thuật toán Dijkstra lớn hơn thuật toán A* bởi vì thuật toán Dijkstra tìm kiếm theo chiều rộng có trọng số (mở rộng đều) nên thuật toán vẫn sẽ xét đến nhiều nút dù nó không cần thiết điều này cũng dẫn đến thuật toán sẽ phức tạp hơn và mất nhiều thời gian hơn để thực hiện [20].



Hình 3 So sánh phạm vi xử lí của thuật toán A* và Djstar

Kết luận: Qua phần so sánh trên, vì tính tối ưu của thuật toán A* nên nhóm đề tài đã lựa chọn thuật toán A* để áp dụng vào việc “Tìm đường bay thay thế tối ưu”.

4.4 Thuật toán A-Star (A*)

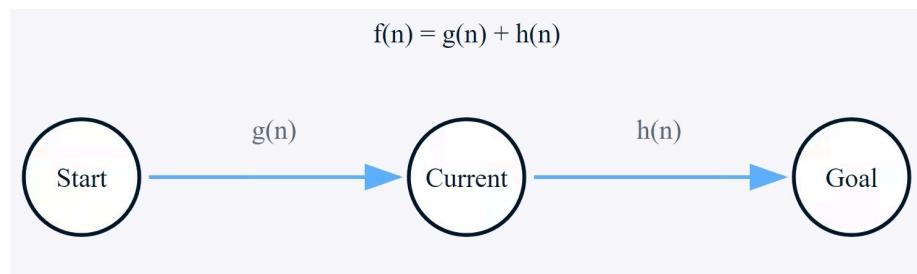
4.4.1 Khái quát chung về thuật toán A*

Thuật toán A* được giới thiệu lần đầu vào năm 1968 bởi Hart, Nilsson và Raphael là một sự kết hợp giữa phương pháp toán học và heuristic để tìm đường đi có chi phí thấp nhất từ điểm khởi hành đến điểm đích. Phát triển từ thuật toán Dijkstra, A* bổ sung yếu tố heuristic để định hướng quá trình tìm kiếm hiệu quả hơn [20].

Sự kết hợp này giúp thuật toán A* vừa đảm bảo tìm được tuyến bay có chi phí tối thiểu, vừa giảm thiểu số lượng điểm trung gian cần xét. Thứ tự ưu tiên cho mỗi đường đi được xác định thông qua hàm heuristic, được đánh giá theo công thức [20]:

$$f(x) = g(x) + h(x)$$

- + $g(x)$ là hàm chi phí thực tế của đường đi từ điểm xuất phát cho đến thời điểm hiện tại.
- + $h(x)$ là hàm ước lượng chi phí từ điểm hiện tại đến điểm đích $f(x)$ thường có giá trị càng thấp thì độ ưu tiên càng cao.
- + $f(x)$ là chi phí ước tính nhỏ nhất.

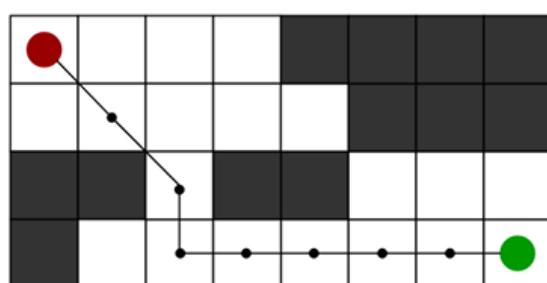


Hình 3. Hàm chi phí thuật toán A*

Thuật giải A* sử dụng 2 tập hợp sau đây:

OPEN: tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến \Rightarrow OPEN là 1 hàng đợi ưu tiên (priority queue) mà trong đó, phần tử có độ ưu tiên cao nhất là phần tử tốt nhất.

CLOSE là tập chứa các trạng thái đã được xét, được lưu trong bộ nhớ để tránh lặp lại khi gặp lại các trạng thái cũ. Nếu không gian tìm kiếm là cây (không có chu trình), có thể không cần sử dụng tập này. Ngoài ra, mỗi trạng thái còn liên kết với một trạng thái cha (father) – tức điểm có chi phí ước tính nhỏ nhất trong tập OPEN tại thời điểm mở rộng.

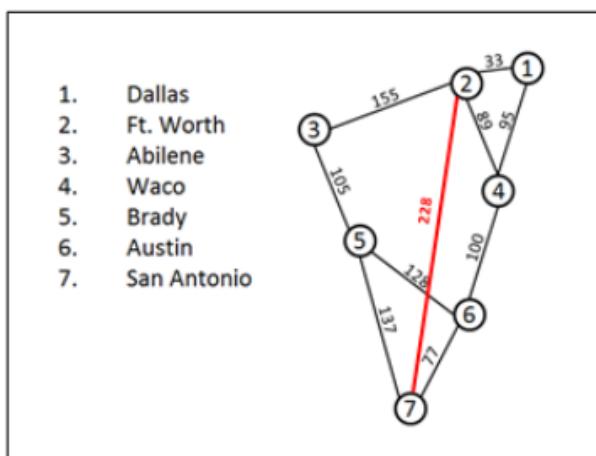


Hình 3. Ví dụ về tìm đường thuật toán A*

4.4.2 Hàm chi phí thực tế $g(x)$ trong A*

Hàm $g(x)$ phản ánh chính xác chi phí đã bỏ ra để đi từ **điểm xuất phát đến vị trí hiện tại**. Trong các bài toán như tìm đường trên bản đồ, $g(x)$ có thể là tổng độ dài các đoạn đường, tổng thời gian di chuyển, hoặc tổng chi phí (như tiền bạc, nhiên liệu, công sức...).Thêm vào đó, vì phản ánh phần chi phí đã đi qua, nên $g(x)$ luôn là giá trị chính xác, không thay đổi theo dự đoán như $h(x)$. Tuy nhiên đặt trong bối cảnh ý tưởng của bài toán thì giá trị $g(x)$ được tính bằng công thức [21]:

$$g(x) = g(\text{nút cha}) + \text{cost}(\text{nút cha} - \text{điểm kế tiếp}) \quad (1)$$



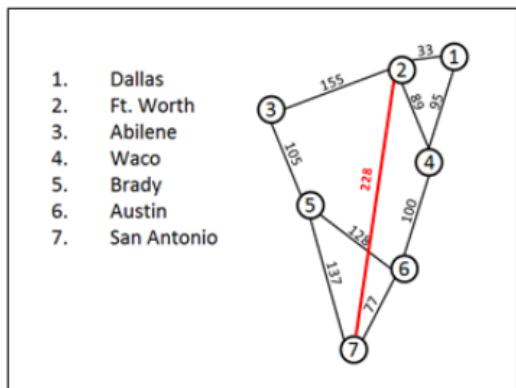
Hình 3. Giá trị $g(x)$ giữa các điểm

4.4.3 Hàm chi phí ước lượng (heuristic) $h(x)$ trong A*

Thuật toán tìm kiếm đánh giá các nút bằng hàm chi phí tổng $f(n)$, với $f(n) = g(n) + h(n)$. Nếu chỉ dùng $g(n)$ – chi phí từ điểm xuất phát đến nút hiện tại – thì về bản chất, thuật toán tương đương với Dijkstra: mở rộng các nút có chi phí thấp nhất nhưng không đảm bảo tiến gần đích. Thành phần heuristic $h(n)$ đóng vai trò định hướng tìm kiếm về phía mục tiêu.

Heuristic là hàm ước lượng chi phí từ một nút đến đích (Beeker, 2004), giúp cải thiện hiệu quả tìm kiếm bằng cách thu hẹp không gian trạng thái cần xét (Zanakis & Evans, 1981). Nó thường được ví như “đường chim bay” – ước lượng khoảng cách

ngắn nhất từ vị trí hiện tại đến đích. Tuy nhiên, do chỉ dựa trên thông tin hạn chế, heuristic không thể dự đoán chính xác diễn tiến của không gian trạng thái, nên thuật toán có thể chỉ tìm được lời giải gần đúng hoặc không tìm được lời giải nào. Đây là giới hạn cố hữu của phương pháp heuristic.



Hình 3. Đường chim bay ước lượng

Thuật toán Heuristic gồm hai thành phần: hàm đánh giá heuristic và thuật toán tìm kiếm khai thác hàm này trong không gian trạng thái. Trong tối ưu hóa mạng đường bay, một heuristic phổ biến là khoảng cách Euclid – nhờ tính đơn giản và hiệu quả. Hàm này ước lượng chi phí từ nút hiện tại đến đích, giúp hệ thống xác định các điểm có tiềm năng dẫn đến lời giải tối ưu hơn.

Khoảng cách Euclid giữa hai điểm trong không gian hai chiều (hoặc ba chiều) được tính bằng công thức:

Trong không gian 2 chiều:

$$h(n) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (2)$$

Trong không gian 3 chiều:

$$h(n) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad (3)$$

Trong đó:

- + (x_1, y_1) và (x_2, y_2) là tọa độ của hai điểm trong mặt phẳng.
- + (x_1, y_1, z_1) và (x_2, y_2, z_2) là tọa độ của hai điểm trong không gian ba chiều.

Một thuộc tính quan trọng của heuristic là tính chấp nhận được (admissibility).

Một heuristic được coi là "chấp nhận được" nếu chi phí ước tính để đến điểm đích luôn nhỏ hơn hoặc bằng chi phí thực tế đối với tất cả các nút đến điểm đích. Tức là:

$$h'(n) \leq h(n) \quad \forall n \in N \text{ (Beeker, 2004)}$$

Thuật toán A-Star với một heuristic chấp nhận được sẽ đảm bảo tìm được đường đi tối ưu (nếu tồn tại), trong khi một heuristic không chấp nhận được thì không đảm bảo điều này.

4.4.4 Nguyên lý hoạt động của thuật toán A*

Thuật toán A* kết hợp giữa chi phí thực tế đã đi ($g(n)$) và chi phí ước lượng còn lại đến đích ($h(n)$) để xác định nút nào nên được mở rộng tiếp theo. Cụ thể, thuật toán hoạt động như sau [21]:

1. Khởi đầu tại nút xuất phát, A* sẽ tính giá trị $f(n) = g(n) + h(n)$ cho nút này.
2. Thuật toán sẽ đặt nút bắt đầu vào danh sách mở (open list) — là danh sách các nút đang chờ được khám phá.
3. Lặp lại các bước sau cho đến khi tìm được đường đi đến đích hoặc không còn nút nào để khám phá:
 - + Chọn nút trong danh sách mở có giá trị $f(n)$ thấp nhất (tức là nút có tiềm năng dẫn đến đích với chi phí thấp nhất).
 - + Di chuyển nút đó sang danh sách đóng (closed list) — danh sách các nút đã được khám phá.
4. Xem xét tất cả nút kề của nút hiện tại. Với mỗi nút kề:

+ Tính $g(n)$, $h(n)$, và $f(n)$.

+ Nếu nút này chưa có trong open list hoặc closed list, thêm nó vào open list. Nếu nút đã có nhưng có giá trị $f(n)$ nhỏ hơn trước, cập nhật đường đi mới tốt hơn. Ngược lại nếu như có giá trị $f(n)$ cao hơn nút trước, trực tiếp loại bỏ.

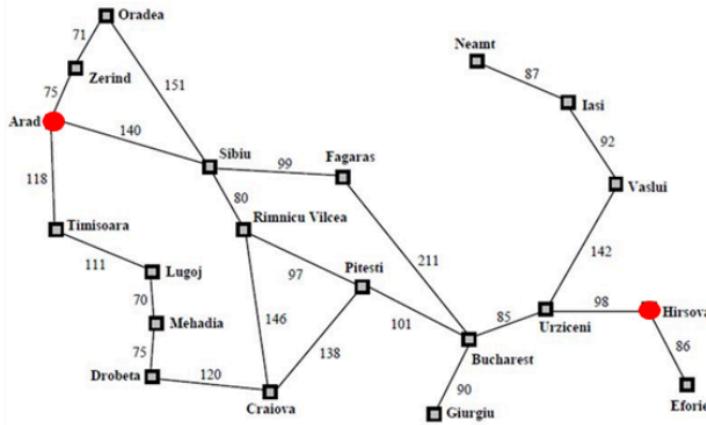
Quá trình tiếp tục cho đến khi đến được nút đích. Khi đó, thuật toán sẽ truy vết lại đường đi ngắn nhất từ đích về xuất phát bằng cách lần theo các nút cha đã lưu trong quá trình tìm kiếm.



Hình 3. Nguyên lý hoạt động của thuật toán A*

4.4.5 Bài toán mô phỏng của thuật toán A*

Để minh họa rõ cách thức bản chất và cách thức hoạt động của thuật toán A*, nhóm đề tài mô phỏng một bài toán tìm đường đi ngắn nhất như sau:



Hình 3. Bài toán mô phỏng thuật toán A*

Giả sử chúng ta có bài toán “Sử dụng thuật toán A* để tìm quãng đường ngắn nhất đi từ Arad đến Hirsova. Dưới đây là phần mô phỏng cách thuật toán A* chạy với giá trị heuristic (h_x) được tính toán và thể hiện qua bảng sau:

$h(Arad) = 336$	$h(Hirsova) = 0$	$h(Rimnicu Vilcea) = 193$
$h(Bucharest) = 20$	$h(Iasi) = 226$	$h(Sibiu) = 253$
$h(Craiova) = 160$	$h(Lugoj) = 244$	$h(Timisoara) = 329$
$h(Drobeta) = 242$	$h(Mehadia) = 241$	$h(Urziceni) = 10$
$h(Eforie) = 161$	$h(Neamt) = 234$	$h(Vaslui) = 199$
$h(Fagaras) = 176$	$h(Oradea) = 380$	$h(Zerind) = 374$
$h(Giurgiu) = 77$	$h(Pitesti) = 100$	

- Cách thuật toán tính ra đường đi gần nhất từ điểm Arad đến Hirsova:

$$\text{OPEN} = \{(Arad, g = 0, h = 0, f = 0)\}.$$

$$\text{CLOSE} = \{\}.$$

Do OPEN chỉ chứa có 1 thành phố nên thành phố này sẽ là thành phố tốt nhất.
Nghĩa là ta chọn $T_{max} = Arad$. Lấy Arad ra khỏi OPEN và đưa vào CLOSE.

$$OPEN = CLOSE = (Arad, g = 0, h = 0, f = 0).$$

Từ Arad có thể đi được đến 3 thành phố Sibiu, Timisoara và Zerind. Ta lần lượt tính f, g và h của 3 thành phố này. Do cả 3 nút mới tạo ra này chưa có nút cha nên ban đầu nút cha của chúng đều là Arad.

- $h(Sibiu) = 253$

$$g(Sibiu) = g(Arad) + cost(Arad, Sibiu) = 0 + 140 = 140$$

$$f(Sibiu) = g(Sibiu) + h(Sibiu) = 140 + 253 = 393$$

$$Cha(Sibiu) = Arad$$

- $h(Timisoara) = 329$

$$g(Timisoara) = g(Arad) + cost(Arad, Timisoara) = 0 + 118 = 140$$

$$f(Timisoara) = g(Timisoara) + h(Timisoara) = 118 + 329 = 447$$

$$Cha(Timisoara) = Arad$$

- $h(Zerind) = 374$

$$g(Zerind) = g(Arad) + cost(Arad, Zerind) = 0 + 75 = 75$$

$$f(Zerind) = g(Zerind) + h(Zerind) = 75 + 374 = 449$$

$$Cha(Zerind) = Arad$$

Do Sibiu, Timisoara và Zerind đều không có trong cả OPEN và CLOSE nên ta thêm 3 nút này vào OPEN.

$$OPEN = \{(Sibiu, g = 140, h = 253, f = 393, Cha = Arad),$$

$$(Timisoara, g = 118, h = 329, f = 447, Cha = Arad),$$

$(Zerind, g = 75, h = 374, f = 449, Cha = Arad)\}.$

CLOSE = $\{(Arad, g = 0, h = 0, f = 0)\}.$

Trong tập OPEN, Sibiu là nút có giá trị f nhỏ nhất nên ta sẽ chọn $Tmax = Sibiu$.
Ta lấy Sibiu ra khỏi OPEN và đưa vào CLOSE.

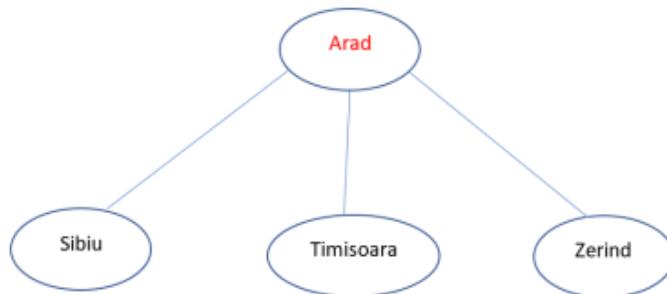
OPEN = $\{(Timisoara, g = 118, h = 329, f = 447, Cha = Arad),$

$(Zerind, g = 75, h = 374, f = 449, Cha = Arad)\}.$

CLOSE = $\{(Arad, g = 0, h = 0, f = 0),$

$(Sibiu, g = 140, h = 253, f = 393, Cha = Arad)\}.$

Từ Sibiu có thể đi được đến Arad, Fagaras, Oradea, R. Vilcea. Ta lần lượt tính h, g và f của các nút này.



- $h(Arad) = 366$

$$g(Arad) = g(Sibiu) + cost(Sibiu, Arad) = 140 + 140 = 280$$

$$f(Arad) = g(Arad) + h(Arad) = 280 + 366 = 646$$

- $h(Fagaras) = 176$

$$g(Fagaras) = g(Sibiu) + cost(Sibiu, Fagaras) = 140 + 99 = 239$$

$$f(Timisoara) = g(Fagaras) + h(Fagaras) = 176 + 239 = 415$$

- $h(Oradea) = 380$

$$g(Oradea) = g(Sibiu) + cost(Sibiu, Oradea) = 140 + 151 = 291$$

$$f(Oradea) = g(Oradea) + h(Oradea) = 380 + 291 = 671$$

Nút Arad đã có trong CLOSE và $g(Arad)$ mới được tạo ra có giá trị là 280 lớn hơn $g(Arad)$ lưu trong CLOSE có giá trị là 0 nên ta sẽ không cập nhật giá trị g và f của Arad lưu trong CLOSE. 3 nút Fagaras, Oradea và R. Vilcea đều không có trong OPEN và CLOSE nên ta sẽ thêm 3 nút này vào OPEN, đặt cha của chúng là Sibiu.

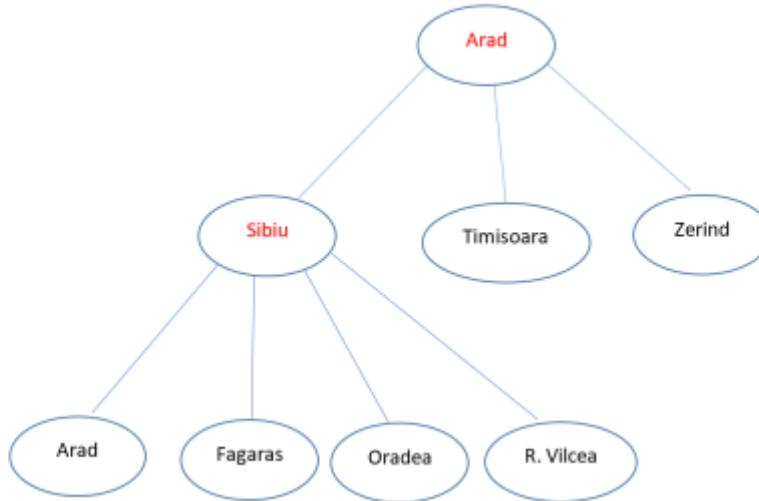
$$\text{OPEN} = \{(Timisoara, g = 118, h = 329, f = 447, Cha = Arad),$$

$$(Zerind, g = 75, h = 374, f = 449, Cha = Arad),$$

$$(Fagaras, g = 239, h = 176, f = 415, Cha = Sibiu),$$

$$(Oradea, g = 291, h = 380, f = 617, Cha = Sibiu),$$

$$(R. Vilcea, g = 220, h = 193, f = 413, Cha = Arad)\}.$$



$$\text{CLOSE} = \{(Arad, g = 0, h = 0, f = 0),$$

$$(Sibiu, g = 140, h = 253, f = 393, Cha = Arad)\}.$$

Trong tập OPEN, R. Vilcea là nút có giá trị f nhỏ nhất nên ta chọn $T_{max} = R. Vilcea$. Chuyển R. Vilcea từ tập OPEN sang tập CLOSE. Từ R. Vilcea có

thì đi được tới 3 thành phố là Craiova, Pitesti và Sibiu. Ta lần lượt tính các giá trị h , g và f của 3 thành phố này.

- $h(Sibiu) = 253$

$$g(Sibiu) = g(R. Vilcea) + \text{cost}(R. Vilcea, Sibiu) = 220 + 80 = 300$$

$$f(Sibiu) = g(Sibiu) + h(Sibiu) = 300 + 253 = 553$$

- $h(Craiova) = 160$

$$g(Craiova) = g(R. Vilcea) + \text{cost}(R. Vilcea, Craiova) = 220 + 146 = 366$$

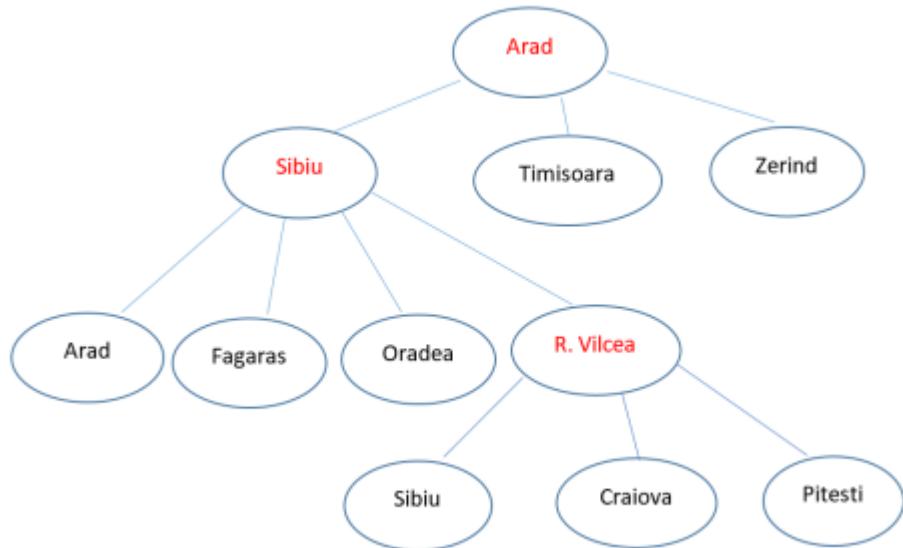
$$f(Craiova) = g(Fagaras) + h(Fagaras) = 366 + 160 = 526$$

- $h(Pitesti) = 100$

$$g(Pitesti) = g(R. Vilcea) + \text{cost}(R. Vilcea, Pitesti) = 220 + 97 = 317$$

$$f(Pitesti) = g(Pitesti) + h(Oradea) = 100 + 317 = 417$$

Do Sibiu đã có trong CLOSE và $g(Sibiu)$ mới có giá trị là 553 lớn hơn $g(Sibiu)$ trong CLOSE có giá trị là 393 nên ta không cập nhật lại các giá trị Sibiu được lưu trong CLOSE. Craiova và Pitesti đều không có trong OPEN lần CLOSE nên ta sẽ đưa chúng vào OPEN và đặt cha của chúng là R.Vilcea.



Tương tự như vậy, thuật toán sẽ đưa ra được các tập CLOSE như sau:

CLOSE = $\{(Arad, g = 0, h = 0, f = 0),$

$(Sibiu, g = 140, h = 253, f = 393, Cha = Arad),$

$(Timisoara, g = 118, h = 329, f = 447, Cha = Arad),$

$(R. Vilcea, g = 220, h = 193, f = 413, Cha = Sibiu),$

$(Fagaras, g = 239, h = 176, f = 415, Cha = Sibiu),$

$(Pitesti, g = 317, h = 100, f = 417, Cha = R. Vilcea),$

$(Bucharest, g = 418, h = 20, f = 438, Cha = Pitesti),$

$(Lugoj, g = 228, h = 176, f = 404, Cha = Timisoara),$

$(Zerind, g = 75, h = 374, f = 449, Cha = Arad),$

$(Urziceni, g = 503, h = 10, f = 513, Cha = Bucharest),$

$(Oradea, g = 146, h = 380, f = 526, Cha = Sibiu),$

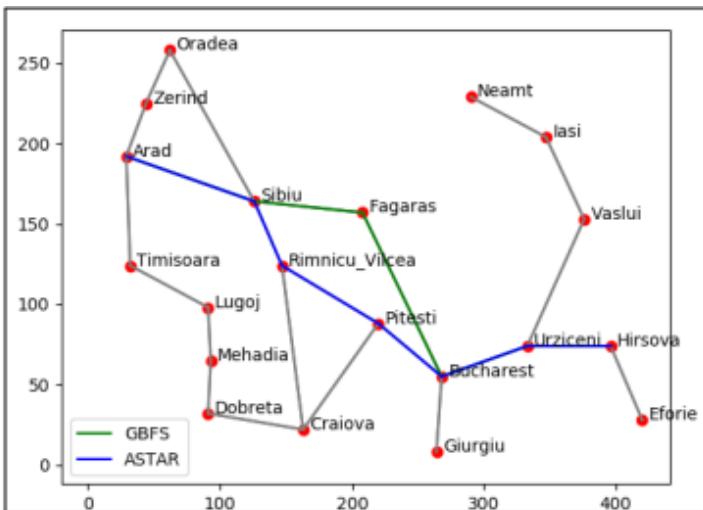
$(Craiova, g = 366, h = 160, f = 526, Cha = R. Vilcea),$

$(Mehadia, g = 298, h = 241, f = 539, Cha = Lugoj),$

$(Giurgiu, g = 508, h = 77, f = 585, Cha = Bucharest)\}$

Trong tập OPEN, nút Hirsova có f nhỏ nhất nên $Tmax = Hirsova$. Mà ta có $TG = Hirsova$, do đó $Tmax = TG$. Ta đã gặp trạng thái đích nên sẽ dừng thuật toán. Tại đây thuật toán vẫn chưa dừng lại, mà tại điểm đích, thuật toán A* sẽ dùng phương pháp truy ngược dựa trên các nút cha (CLOSE) để tìm ra đường đi có giá trị thấp nhất (dựa trên gn). Từ đó ta thấy, kết quả khi chạy thuật toán A* là:

Arad → Sibiu → R. Vilcea → Pitesti → Bucharest → Urziceni → Hirsova



Hình 3. Kết quả của thuật toán A* (đường màu xanh dương)

CHƯƠNG 4: XÂY DỰNG ỨNG DỤNG HỖ TRỢ ĐỀ XUẤT ĐƯỜNG BAY THAY THẾ BẰNG THUẬT TOÁN A*

4.1 Khái quát quy trình thực hiện xây dựng ứng dụng hỗ trợ:

Để có thể xây dựng ứng dụng hỗ trợ tìm đường bay thay thế tạm thời hỗ trợ cho việc xây dựng Playbook, nhóm đề tài tiến hành xây dựng sơ đồ quy trình thực hiện như sau:

Bước 1: Thu thập dữ liệu đầu vào

- + Thu thập dữ liệu các chuyến bay hiện có từ hệ thống nội bộ.
- + Lấy dữ liệu sân bay, mạng lưới đường bay bao gồm: Tên đường bay, các điểm trọng yếu, khoảng cách, chiều của đường bay,...
- + Theo dõi và thu thập dữ liệu bất thường: Nhóm sẽ thu thập dữ liệu thời tiết và các vùng hạn chế, vùng cấm bay do hoạt động bay quân sự.

Bước 2: Thiết kế thuật toán tìm kiếm đường bay thay thế

- + Xây dựng mã code A* trong việc tìm đường bay thay thế tạm thời.
- + Xây dựng thêm những hạn chế, tiêu chí trong việc lựa chọn đường bay thay thế.

Bước 3: Xây dựng ứng dụng hỗ trợ đề xuất đường bay thay thế

- + Xây dựng giao diện của ứng dụng.

+ Tích hợp giao diện với thuật toán A*.

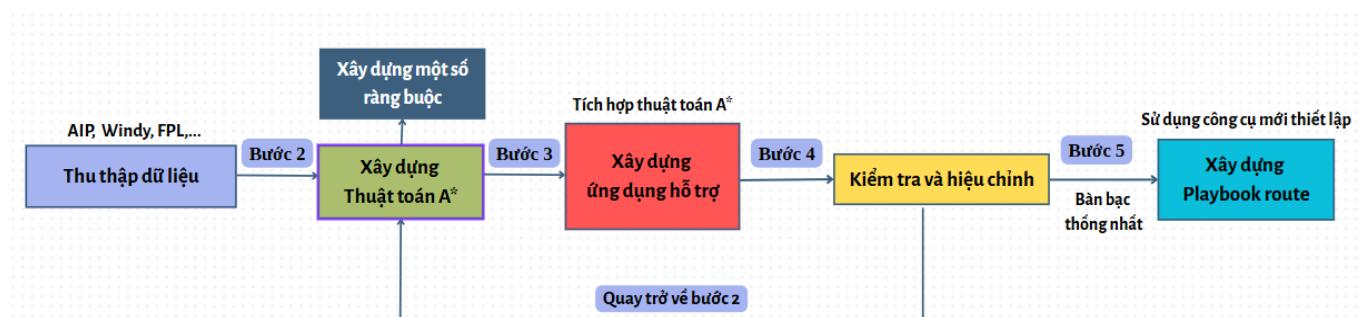
Bước 4: Kiểm tra và hiệu chỉnh

+ Chạy thử công cụ với các tình huống thực tế.

+ Điều chỉnh thuật toán.

+ Bàn bạc và thống nhất với các bên liên quan.

Bước 5: Ứng dụng công cụ trong công tác xây dựng Playbook route.



Hình 4.1. Quá trình xây dựng ứng dụng hỗ trợ cho thiết lập Playbook

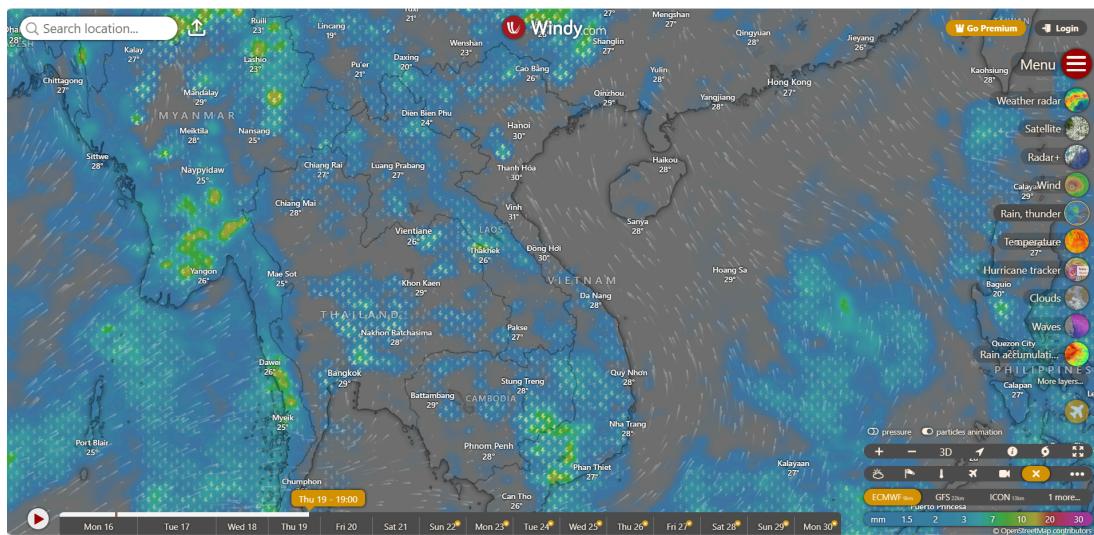
4.2 Phương pháp thu thập dữ liệu

Để hỗ trợ quá trình nghiên cứu, nhóm đồ án thực hiện việc thu thập dữ liệu bằng cách kết hợp quan sát trên ứng dụng với phân tích dữ liệu thứ cấp.

4.2.1 Dữ liệu thời tiết

Dữ liệu thời tiết được thu thập thông qua ứng dụng **Windy** truy cập tại:

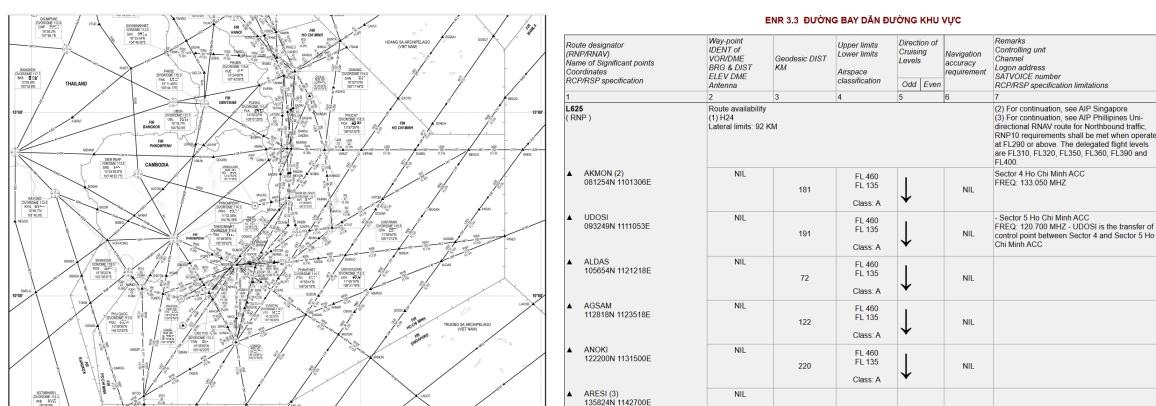
Windy: Rain, thunder. Thời gian quan sát từ ngày **15/05/2025 đến ngày 15/06/2025**. Mục đích của việc thu thập này là để xác định các khu vực thường xuyên bị ảnh hưởng bởi điều kiện thời tiết bất lợi (mưa, giông, vùng CAT...), từ đó đánh giá mức độ tác động cụ thể của thời tiết đến **các tuyến đường bay** trong khu vực nghiên cứu.



Hình 4.2 Giao diện của trang web Windy

4.2.2 Dữ liệu đường bay

Dữ liệu đường bay được thu thập từ trang web chính thức của **Trung tâm Thông báo tin tức Hàng không Việt Nam (VNAIC)** truy cập tại [eAIS Package for Viet Nam](#).



Hình 4.3 Giao diện của trang web VNAIC

Các thông tin được thu thập bao gồm: tên các đường bay, tọa độ các điểm, khoảng cách giữa các điểm, đường bay một chiều, hai chiều như Hình 4.4 và Hình 4.5. Nguồn dữ liệu này phục vụ cho việc thiết kế và phát triển ứng dụng mô phỏng tất cả các tuyến đường bay có trong FIR HCM.

		VỊ ĐỘ	KINH ĐỘ	Ghi chú: ↑: hai chiều ↓: một chiều
W2	KONCO	170000	1071113	↑↑
	BIGBO	162919	1073738	↑↑
	HUE	162408	1074206	↑↑
	DAN	160310	1081154	↑↑
	CQ	152444	1084216	↑↑
	KUMUN	145817	1084827	↑↑
	PCA	135726	1090234	↑↑
	KAMGO	131715	1090605	↑↑
	KARAN	123856	1090924	↑↑
	NHATA	121335	1091204	↑↑
	CRA	115940	1091312	↑↑
	IBUNU	114008	1085205	↑↑
	PTH	105541	1080419	↑↑
	VEPMA	105600	1074018	↑↑
	AC	105621	1071118	↑↑
	TSH	104906	1063902	↑↑

Hình 4.4 Bảng thu thập dữ liệu tọa độ các điểm và chiều của đường bay

Từ	Đến	Khoảng cách (NM)	Tên đường
KONCO	BIGBO	74	W2
BIGBO	PHU BAI DVOR/DME (HUE)	12	
PHU BAI DVOR/DME (HUE)	DA NANG DVOR/DME (DAN)	66	
DA NANG DVOR/DME (DAN)	CHU LAI NDB (CQ)	89	
CHU LAI NDB (CQ)	KUMUN	50	
KUMUN	PHU CAT DVOR/DME (PCA)	115	
PHU CAT DVOR/DME (PCA)	KAMGO	76	
KAMGO	KARAN	69	
KARAN	NHATA	47	
NHATA	CAM RANH DVOR/DME (CRA)	26	
CAM RANH DVOR/DME (CRA)	IBUNU	53	
IBUNU	PHAN THIET VOR/DME (PTH)	120	
PHAN THIET VOR/DME (PTH)	VEPMA	44	
VEPMA	AN LOC	53	
AN LOC	IAN SON NHAT DVOR/DME (TSH)	60	

Hình 4.5 Bảng thu thập dữ liệu khoảng cách giữa các điểm

4.2.3 Dữ liệu kế hoạch bay

Nhóm đề tài đã thu thập dữ liệu kế hoạch bay từ các hãng hàng không thông qua các bản tin kế hoạch bay chính thức. Tập dữ liệu này bao gồm các chuyến bay hoạt động trong FIR HCM. Hình 4.6 thể hiện các thông tin thu thập được bao gồm số hiệu chuyến bay, tuyến đường bay, xác định điểm đầu, điểm cuối và điểm trung gian của đường bay đó để phục vụ cho việc nhập dữ liệu đầu vào của thuật toán.

```
■LYA0065 170001  
FF VVTSZPZX  
170001 VVTSVJCX  
■(FPL-VJC318-IS  
-A321/M-SDGHIRWYZ/LB1  
-VVTS0215  
-N0461F310 KADUM2K PATMA Q2 LATOM PHADE2A  
-VVPB0102 VVCR VVDN  
-PBN/A1B2C1D1L102S1S2 NAV/RNP2 DOF/250617 REG/VNA630 EET/VVHN0040  
SEL/FQAK CODE/888133 OPR/VIETJET AIR PER/C TALT/VVCR  
RMK/TCAS EQUIPPED  
-E/0230 P/TBN R/UVE J/L D/4 176 C YLW  
A/RED AND WHITE  
C/13681 OLIVEIRA OSMAR)
```

Hình 4.6 Ví dụ thông tin của một chuyến bay trong kế hoạch bay

4.3 Thiết kế giao diện ứng dụng người dùng

Giao diện người dùng được xây dựng nhằm trực quan hóa các yếu tố quan trọng trong vùng FIR HCM và hỗ trợ người sử dụng thao tác thuận tiện. Phần này mô tả cấu trúc hiển thị và các chức năng chính được thiết kế cho ứng dụng.

Phần mã HTML trong Hình 4.7 đóng vai trò tạo khung chứa bản đồ và nhúng thư viện Leaflet – một thư viện JavaScript mã nguồn mở phổ biến dùng để hiển thị bản đồ tương tác.

```
<div id="map"></div>  
<script src="https://unpkg.com/leaflet/dist/leaflet.js"></script>
```

Hình

Hiểu một cách đơn giản là:

<div id="map"></div> là phần tạo ra “chỗ trống” để hiện bản đồ trên giao diện.
<script src="https://unpkg.com/leaflet/dist/leaflet.js"></script> là đoạn mã để lấy thư viện Leaflet từ internet, giúp bản đồ hoạt động được.

Nhờ vậy, người dùng có thể xem trực quan các đường bay và các thông tin quan trọng khác trên bản đồ trong vùng FIR HCM.

Tiếp theo, ứng dụng được thiết kế thêm một thanh menu để người dùng dễ dàng thao tác với các chức năng quan trọng. Thanh menu này hiển thị trên màn hình

dưới dạng các nút bấm, giúp người dùng nhanh chóng chọn các tính năng như chuyển bay, xem thông tin thời tiết, lọc dữ liệu hoặc điều chỉnh chế độ hiển thị bản đồ.

Đoạn mã Hình... dưới đây minh họa cấu trúc HTML của thanh menu đó. Mỗi mục trong menu được định nghĩa bằng một thẻ `<div>` với tên và chức năng riêng, giúp giao diện trở nên trực quan và tiện lợi khi sử dụng.

```
<div class="menu z-50 absolute top-4 left-4 bg-white p-2 rounded shadow">
  <div class="menu-item cursor-pointer font-semibold" id="add-flight"><b>FLIGHT</b></div>
  <div class="menu-item" id="add-no-flight">WEATHER</div>
  <div class="menu-item" id="option-manament">FILTERS</div>
  <div class="menu-item" id="MAPOPTIONS">MAPOPTIONS</div>
</div>
```

Nếu người dùng ấn chọn thanh menu “Flight” thì một biểu mẫu (form) sẽ hiện ra để người dùng nhập thông tin về chuyến bay mới. Biểu mẫu này có tên là “Thêm chuyến bay” bao gồm các trường lần lượt là Tên chuyến bay, điểm bắt đầu, điểm kết thúc và gửi chuyến bay. Sau khi hoàn tất nhập và chọn lựa đầy đủ thông tin người dùng ấn vào “gửi chuyến bay” thì những thông tin này sẽ được lưu trữ lại để hiển thị trên bản đồ.

Đoạn mã Hình... dưới đây minh họa cấu trúc của biểu mẫu đó. Phần này được thiết kế hiển thị dưới dạng cửa sổ nhỏ trên màn hình (có thể đóng/mở), với các trường thông tin được trình bày rõ ràng để người dùng nhập liệu một cách thuận tiện và trực quan.

```

<div id="flightBox" class="hidden fixed top-1/2 left-1/2 transform -translate-x-1/2 -translate-y-1/2 bg-white p-6 rounded-xl shadow-xl max-w-md w-full z-50">
  <button id="closeFlightFormBtn" class="absolute top-4 right-4 text-gray-500 hover:text-red-600 text-2xl font-bold focus:outline-none" title="Đóng">&times;

```

Khi người dùng ánh chọn thanh Menu “Weather” thì biểu mẫu tiếp theo sẽ được hiển thị là “Thêm vùng cấm bay”. Biểu mẫu này được hiển thị dưới dạng một cửa sổ nhỏ có thể đóng hoặc mở tùy ý. Trong biểu mẫu, người dùng có thể nhập **tên của vùng cấm bay** để dễ nhận biết sau đó **chọn kiểu vùng cấm bay**: gồm hai loại cơ bản là vùng cấm dạng đa giác (Polygon) với nhiều điểm bao quanh hoặc vùng cấm dạng hình tròn (Circle) xác định bằng tâm và bán kính. Tùy theo loại vùng cấm đã chọn, giao diện sẽ hiển thị thêm các ô nhập phù hợp.

Nếu chọn **đa giác**, người dùng sẽ nhập danh sách tọa độ của các điểm tạo thành đa giác. Nếu chọn **hình tròn**, người dùng sẽ nhập tọa độ tâm và bán kính của vùng cấm.

Cuối cùng, người dùng có thể bấm nút thêm vào bản đồ để **thêm vùng cấm bay này vào bản đồ**, để hệ thống sẽ trực quan hóa khu vực cấm bay một cách trực tiếp và giúp người quản lý dễ dàng theo dõi, điều chỉnh khi cần thiết.

Tất cả nội dung trên được thể hiện thông qua đoạn code Hình... bên dưới

```
div id="no-flight-form" class="hidden fixed top-10 right-10 bg-white p-6 rounded-xl shadow-xl max-w-md w-full z-[2000]">
  <button id="closeNoFlightFormBtn" class="absolute top-4 right-4 text-gray-500 hover:text-red-600 text-2xl font-bold focus:outline-none" title="Đóng">&times;
  <h3>Thêm vùng cấm bay</h3>
  <label>Tên vùng:</label>
  <input type="text" id="zoneName" placeholder="Ví dụ: Zone A" style="width: 100%; margin-bottom: 10px;"><br>

  <label>Kiểu vùng cấm bay:</label>
  <select id="zoneType" style="width:100%; margin-bottom:10px;">
    <option value="polygon">Đa giác (nhiều điểm)</option>
    <option value="circle">Hình tròn (tâm + bán kính)</option>
  </select>

  <!-- Nhóm nhập đa giác -->
  <div id="polygon-inputs">
    <label>Tọa độ (độ,phút,giây) mỗi dòng:</label>
    <textarea id="coordinatesInput" rows="5" style="width: 100%;" placeholder="13°59'45";N,108°05'03";E&#10;15°60'22";N,111°04'09";E"></textarea>
  </div>

  <!-- Nhóm nhập hình tròn -->
  <div id="circle-inputs" style="display:none;">
    <label>Tọa độ tâm (Độ,phút,giây)</label>
    <input type="text" id="circleCenter" placeholder="13°59'45";N,108°05'03" style="width:100%; margin-bottom:10px;">
    <label>Bán kính (nm):</label>
    <input type="number" id="circleRadius" placeholder="VD: 1000" style="width:100%; margin-bottom:10px;">
  </div>
  <button id="submitNoFlightZoneButton" onclick="submitNoFlightZone()" style="margin-top: 10px;">Thêm vào bản đồ
</div>
```

Khi người dùng ấn chọn thanh Menu “Filter” một bảng **“Quản lý các chuyến bay”** sẽ được hiển thị nhằm giúp người dùng dễ dàng theo dõi, chỉnh sửa hoặc xóa các chuyến bay đã thêm vào hệ thống (đây là những chuyến bay mà người dùng đã nhập và gửi thành công ở biểu mẫu “Thêm chuyến bay”).

Phần giao diện này được thiết kế dưới dạng một cửa sổ nổi có thể đóng hoặc mở khi cần, hiển thị danh sách tất cả các chuyến bay hiện có. Bên trong cửa sổ, dữ liệu được sắp xếp thành bảng gồm ba cột chính đó là: **Tên chuyến bay** hiển thị tên mà người dùng đã đặt. **Danh sách điểm** thể hiện các điểm tạo nên lộ trình của chuyến bay, bao gồm điểm đầu, điểm cuối và các điểm trung gian. Trường **Hành động** chứa các nút để người dùng có thể chỉnh sửa hoặc xóa chuyến bay đó.

Ngoài ra, bảng này có cho phép cuộn dọc để nhìn giao diện một cách gọn gàng và dễ quan sát, ngay cả khi có nhiều chuyến bay được thêm vào. Những chức năng trên được thể hiện rõ qua đoạn mã minh họa ở dưới Hình...

```
v id="flightManagementBox" class="hidden fixed top-8 right-8 bg-white p-8 rounded-2xl shadow-2xl max-w-3xl w-[480px] z-[1000]">
  <button id="closerLightBoxBtn" class="absolute top-4 right-4 text-gray-500 hover:text-red-600 text-2xl font-bold focus:outline-none" title="Đóng">&times;
  <h2>Quản lý các chuyến bay</h2>
  <div class="table-container" style="max-height:420px; overflow-y:auto;">
    <table class="w-full table-auto border-collapse border border-gray-300 min-w-[600px]">
      <thead class="bg-gray-200">
        <tr>
          <th class="border px-4 py-2">Tên chuyến bay</th>
          <th class="border px-4 py-2">Danh sách điểm</th>
          <th class="border px-4 py-2">Hành động</th>
        </tr>
      </thead>
      <tbody id="flightTableBody"></tbody>
    </table>
  </div>
<div>
```

Hình...

Bên cạnh đó khi người dùng ấn nút xem chuyến bay được hiển thị ở biểu mẫu quản lý chuyến bay thì một biểu mẫu mới là “Chi tiết chuyến bay” sẽ được hiển thị. Biểu mẫu chứa toàn bộ chi tiết của chuyến bay được chọn và những vùng cấm mà nó vi phạm giúp người dùng nhanh chóng nắm bắt thông tin mà không cần rời khỏi màn hình chính. Cuối cùng khi người dùng ấn nút “*Chạy thuật toán A**”, lúc này cho phép người dùng thực hiện thuật toán tìm đường đi tối ưu giữa các điểm đã định sẵn. Tất cả những tính năng trên được xây dựng dựa trên đoạn code minh họa dưới Hình...,

```
<div id="flightDetailModal" class="hidden fixed bottom-8 right-8 bg-white p-6 rounded-2xl shadow-2xl max-w-lg w-[400px] z-[1100]"> <h2 class="text-xl font-bold mb-4">Ch</div>
  <div id="flightDetailContent" class="space-y-2 text-sm"></div>
  <button onclick=OnCloseModalFlightDetail() class="mt-4 bg-gray-300 hover:bg-gray-400 text-gray-800 font-semibold py-1 px-4 rounded">
    Đóng
  </button>
</div>

<div id="astarContainer" class="mt-4 hidden">
  <button id="runAstarBtn" class="bg-purple-600 hover:bg-purple-800 text-white font-semibold py-1 px-4 rounded">
    Chạy A*
  </button>
  <div id="astarResults" class="mt-3 space-y-2 text-sm"></div>
</div>
```

Hình....

Sau khi người dùng ấn “Chạy thuật toán A*” một bảng tên “Kết quả A*” sẽ được hiển thị. Bảng này gồm các cột lần lượt là: “**Phương án**” thể hiện số thứ tự và tên phương án do thuật toán đề xuất, “**Đường bay thay thế**” liệt kê các điểm trên đường bay thay thế, “**Quãng đường (NM)**” hiển thị tổng chiều dài của phương án tính theo hải lý, “**Hành động**” cho phép người dùng thực hiện các thao tác tiếp theo như chọn, khi người dùng ấn chọn, đường bay này sẽ được hiển thị trực quan lên trên bản đồ. Tất cả những gì người dùng nhìn thấy và thao tác được trong giao diện thực tế đã nêu trên, thực chất đều do đoạn code ở dưới Hình dụng nêu.

```
<div id="astarResultBox" class="hidden fixed bottom-8 right-8 bg-white p-8 rounded-2xl shadow-2xl max-w-3xl w-[480px] z-[1200]">
  <button id="CloseAstarBoxBtn" class="absolute top-4 right-4 text-gray-500 hover:text-red-600 text-2xl font-bold focus:outline-none" title="Đóng">&times;;</button>
  <h2 class="text-2xl font-bold mb-6 text-center">Kết quả A*</h2>
  <div class="table-container" style="max-height:420px;overflow-y:auto;">
    <table class="w-full table-auto border-collapse border-gray-300 min-w-[600px]">
      <thead class="bg-gray-200">
        <tr>
          <th class="border px-4 py-2">Phương án</th>
          <th class="border px-4 py-2">Đường bay thay thế</th>
          <th class="border px-4 py-2">Quãng đường (NM)</th>
          <th class="border px-4 py-2">Hành động</th>
        </tr>
      </thead>
      <tbody id="astarTableBody"></tbody>
    </table>
  </div>
</div>
```

Hình....

Cuối cùng, khi người dùng ấn chọn thanh Menu “**Mapoptions**” trang web sẽ hiển thị ra bảng “**Quản lý vùng cấm bay**” ở phần này sẽ chứa tất cả những thông tin về vùng cấm mà người dùng đã nhập ở biểu mẫu “Thêm vùng cấm bay” cụ thể là tên vùng cấm và kích thước của vùng cấm đó, nhưng một điểm khác ở bảng này đã có thêm trường “Hành động”, trường này cho phép người dùng thao tác xóa những vùng cấm bay không cần thiết. Tất cả những chức năng được nêu ở trên được xây dựng thông qua đoạn code Hình....

```
<div id="noFlightZoneBox" class="hidden fixed top-8 right-8 bg-white p-8 rounded-2xl shadow-2xl max-w-3xl w-[480px] z-[1000]">
  <button id="closeNoFlightZoneBoxBtn" class="absolute top-4 right-4 text-gray-500 hover:text-red-600 text-2xl font-bold focus:outline-none" title="Đóng">&times;;</button>
  <h2 class="text-2xl font-bold mb-6 text-center">Quản lý vùng cấm bay</h2>
  <div class="text-container" style="max-height:420px;overflow-y:auto;">
    <table class="w-full table-auto border-collapse border-gray-300 min-w-[600px]">
      <thead class="bg-gray-200">
        <tr>
          <th class="border px-4 py-2">Tên vùng cấm</th>
          <th class="border px-4 py-2">Kích thước</th>
          <th class="border px-4 py-2">Hành động</th>
        </tr>
      </thead>
      <tbody id="noFlightZoneTableBody"></tbody>
    </table>
  </div>
</div>
```

Hình...

4.4 Kiến trúc hoạt động và nguyên lý xử lý dữ liệu của ứng dụng

Bên cạnh giao diện, hệ thống còn bao gồm phần xử lý dữ liệu ở tầng backend, đảm nhiệm việc phân tích, tính toán và đề xuất đường bay thay thế. Nội dung sau sẽ trình bày cấu trúc hoạt động và quy trình xử lý dữ liệu của ứng dụng.

4.4.1 Thiết lập kiến trúc ứng dụng

Quá trình xây dựng backend được thực hiện theo các bước sau để đảm bảo khả năng xử lý dữ liệu và đề xuất đường bay hiệu quả:

Bước 1: Khởi tạo môi trường và thư viện cần thiết

Đoạn mã trong Hình..... là một phần quan trọng trong việc thiết lập môi trường cho một ứng dụng web được xây dựng bằng Python, sử dụng Flask làm framework backend (hiểu đơn giản là một khung để xây dựng web).

```
from flask import Flask, request, jsonify
from pymongo import MongoClient
from flask_cors import CORS
from bson.json_util import dumps
from shapely.geometry import LineString, Polygon, Point
import requests
import heapq
import os
```

Hình 4

Các thành phần cần thiết từ thư viện *Flask* được nhập vào, bao gồm lớp *Flask* để tạo ứng dụng, *request* để truy cập dữ liệu từ các yêu cầu *HTTP*, và *jsonify* để chuyển đổi dữ liệu *Python* thành định dạng *JSON* phục vụ cho việc trả về cho *client*.

MongoClient từ thư viện *pymongo* được sử dụng để kết nối và tương tác với cơ sở dữ liệu *MongoDB*, cho phép lưu trữ và truy xuất dữ liệu từ *MongoDB* một cách hiệu quả.

Để hỗ trợ việc xử lý các yêu cầu từ các nguồn khác nhau, thư viện *flask_cors* được nhập vào, cho phép cấu hình *Cross-Origin Resource Sharing (CORS)* cho ứng dụng.

Tóm lại, việc thiết lập môi trường cho một ứng dụng web sử dụng *Flask* không chỉ bao gồm việc nhập các thư viện cần thiết mà còn đảm bảo rằng ứng dụng có khả năng tương tác hiệu quả với cơ sở dữ liệu *MongoDB* và hỗ trợ việc xử lý các yêu cầu từ nhiều nguồn khác nhau thông qua CORS.

Bước 2: Cài đặt Backend Flask API

```
# Khởi tạo Flask app
app = Flask(__name__)
CORS(app)
connectAPI = 'https://algorithma-84og.onrender.com'
```

Hình 4

Gọi API Flask đã được deploy trên nền tảng [Render.com](#) nhằm những mục đích sau:

Đầu tiên, khi deploy backend lên Render thông qua đường link <https://algorithma-84og.onrender.com>, từ đó có thể lấy tất cả các dữ liệu có trong API này, ví dụ như Hình... là tất cả các dữ liệu nằm trong API được nhóm để tài sử dụng.

```
[[{"center": [10.06222222222223, 104.58944444444444], "name": "Zone H", "radius": 30000, "type": "circle"}, {"center": [11.91694444444443, 108.10805555555555], "name": "hanh", "radius": 40000, "type": "circle"}]]
```

Thứ hai, tạo hệ thống hoàn chỉnh từ Frontend sang Backend. Có nghĩa là Phần frontend chỉ là giao diện – nó như: hiển thị bản đồ, cho người dùng chọn điểm đầu/cuối, hiển thị đường bay/vùng cấm, v.v. Tuy nhiên nó không thể tự tính toán được đường bay, không truy cập được cơ sở dữ liệu. Thay vào đó Backend mới là nơi xử lý dữ liệu như: truy xuất dữ liệu waypoint, vùng cấm từ MongoDB, chạy thuật toán A*, Yen để tính toán đường bay thay thế và trả kết quả về cho Frontend. Vậy kết nối Frontend - Backend có nghĩa là Frontend call API đến Flask Backend sau đó Backend xử lý dữ liệu để trả lại dữ liệu, sau đó Frontend lấy dữ liệu đó để vẽ đường bay, vùng cấm,..

Thứ ba, cho phép giao diện người dùng hoạt động từ xa có nghĩa là khi tạo giao diện người dùng thì gọi API để cho phép tất cả mọi người có thể truy cập vào được trang web.

Bước 3: Cài đặt hệ quản trị cơ sở dữ liệu

MongoDB được sử dụng làm phần mềm mã nguồn mở để quản trị cơ sở dữ liệu. Khi người dùng thao tác trên ứng dụng web, họ sẽ cần nhập hoặc truy xuất dữ liệu, do đó cần một hệ thống lưu trữ dữ liệu ổn định và linh hoạt. Những bộ dữ liệu này sẽ hỗ trợ quá trình phân tích, gợi ý hoặc trực quan hóa các phương án đường bay thay thế trong tình huống khẩn cấp.

Hình..... là đoạn mã code thiết lập bộ dữ liệu MongoDB.

```
# Kết nối MongoDB
MONGO_URI = os.getenv(
    "MONGO_URI",
    "mongodb+srv://admin2:123123a@cluster0.nyw26.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0"
)
client = MongoClient(MONGO_URI)
db = client["Flight_A"]
road_collection = db["Road"]
point_collection = db["data_point"]
No_Flight_collection = db["No_Flight"]
distance_collection = db["distance"]
flight_path_new_collection = db["Flight_path_new"]
flight_path_collection = db["flight_path"]
```

Hình

Hệ thống sẽ tạo một đối tượng, cụ thể là một chuyến bay, kết nối đến cơ sở dữ liệu **MongoDB** nhằm thiết lập liên lạc giữa ứng dụng và kho dữ liệu. Sau khi kết nối thành công, hệ thống sẽ truy cập vào cơ sở dữ liệu có tên là *Flight_A* chứa tất cả các dữ liệu cần thiết để xử lý cho một chuyến bay. Bộ dữ liệu của một chuyến cụ thể được thiết lập gồm: các tuyến đường bay con (*road_collection*), các lô điểm (*point_collection*), vùng cấm bay (*no_flight_collection*), khoảng cách giữa các điểm (*distance_collection*), đường bay hoàn chỉnh của chuyến bay (*flight_path_collection*) và đường bay mới được đề xuất (*flight_path_new_collection*). Khi ứng dụng thực hiện nhiệm vụ tìm đường bay thay thế khi xuất hiện một chướng ngại vật nào đó, hệ thống sẽ thực hiện truy vấn các dữ liệu đó để từ đó tính toán và đề xuất đường bay.

4.4.2 Nguyên lý xử lý dữ liệu

4.4.2.1 Xử lý dữ liệu đường bay và lô điểm

- Thêm dữ liệu đường bay vào cơ sở dữ liệu (POST /data-road)

MongoDB lưu trữ thông tin các tuyến đường bay nhằm hỗ trợ backend trong việc truy xuất và hiển thị trên giao diện. Để đáp ứng nhu cầu người dùng tự khai báo tuyến bay dự kiến, hệ thống cung cấp endpoint **POST /data-road**, tiếp nhận tên tuyến đường chính và danh sách các đoạn đường con. Hệ thống sẽ kiểm tra tính đầy đủ của dữ liệu đầu vào: nếu thiếu sẽ từ chối và trả về thông báo lỗi; nếu hợp lệ, thông tin được lưu vào MongoDB và phản hồi kèm mã ID của bản ghi mới.

```
# Route POST để thêm dữ liệu road
@app.route("/data-road", methods=["POST"])
def add_data_road():
    data = request.json
    if not data or "ten_duong_chinh" not in data or "cac_ten_duong_con" not in data:
        return jsonify({"error": "Missing required fields"}), 400
    result = road_collection.insert_one(data)
    return jsonify({
        "message": "Road added successfully",
        "id": str(result.inserted_id)
    }), 201
```

Hình 4

- Truy xuất lại dữ liệu đường kèm các lô điểm cùng tọa độ (GET /data-road).

Sau khi đường bay dự kiến vừa báo nhập thành công, đoạn mã trong Hình..... định nghĩa một route GET tại */data-road*, được sử dụng để truy xuất danh sách các tuyến đường chính cùng với thông tin chi tiết của từng điểm nằm trên đó. Khi có yêu cầu từ phía người dùng, hàm *get_data_road_with_coordinates()* sẽ được gọi.

Hàm lấy toàn bộ dữ liệu các tuyến đường chính từ *road_collection*. Với mỗi tuyến đường, hệ thống đọc danh sách các tên các lô điểm và lần lượt tra cứu thông tin chi tiết của từng điểm này trong *point_collection*. Các thông tin hiển thị bao gồm: tên đường, vĩ độ, kinh độ và chiều bay. Nếu tìm thấy điểm tương ứng, dữ liệu sẽ được gắn vào kết quả trả về; nếu không, các giá trị sẽ để là *None*.

Cuối cùng, danh sách đã xử lý sẽ được trả về dưới dạng JSON thông qua lệnh *jsonify(result)* và báo hiệu thành công. Mục đích chính của đoạn mã là cung cấp dữ liệu đầy đủ về tuyến đường và các điểm trọng yếu để phục vụ việc hiển thị chính xác trên giao diện người dùng.

```

# Route GET kết hợp tọa độ và chiều
@app.route("/data-road", methods=["GET"])
def get_data_road_with_coordinates():
    roads = list(road_collection.find({}, {"_id": 0}))
    result = []

    for road in roads:
        duong_chinh = road["ten_duong_chinh"]
        ten_duong_con = road["cac_ten_duong_con"]

        enriched_con = []
        for name in ten_duong_con:
            point = point_collection.find_one(
                {"ten_duong": name},
                {"_id": 0, "vi_do": 1, "kinh_do": 1, "chieu": 1}
            )
            if point:
                enriched_con.append({
                    "ten_duong": name,
                    "vi_do": point.get("vi_do"),
                    "kinh_do": point.get("kinh_do"),
                    "chieu": point.get("chieu")
                })
            else:
                enriched_con.append({
                    "ten_duong": name,
                    "vi_do": None,
                    "kinh_do": None,
                    "chieu": None
                })

        result.append({
            "ten_duong_chinh": duong_chinh,
            "cac_ten_duong_con": enriched_con
        })
    return jsonify(result), 200

```

Hình

4.4.2.2 Xử lý dữ liệu vùng cấm bay

- **Thêm dữ liệu vùng cấm bay**

Đoạn mã trong Hình định nghĩa một API POST tại địa chỉ /no_flight, cho phép người dùng thêm dữ liệu về vùng cấm bay vào hệ thống. API này tiếp nhận dữ liệu ở định dạng JSON và phân loại vùng cấm bay theo hai dạng hình học: vùng đa giác (polygon) và vùng hình tròn (circle). Mỗi loại vùng đều yêu cầu các thông tin đặc trưng để có thể lưu trữ và xử lý chính xác.

```

# Route POST để thêm dữ liệu vùng cấm bay
@app.route("/no_flight", methods=["POST"])
def add_no_flight():
    data = request.json
    if not data or "name" not in data:
        return jsonify({"error": "Thiếu tên vùng cấm bay"}), 400

    name = data["name"]

```

Hình

Hệ thống yêu cầu người dùng nhập tên của vùng cấm bay và kiểm tra xem dữ liệu gửi lên có trường "name" hay không. Nếu thiếu tên vùng cấm bay, hệ thống sẽ trả về thông báo lỗi. Sau đó, API tiếp tục xác định hình ảnh vùng cấm bay dựa trên các trường trong dữ liệu.

```

# Nếu là vùng đa giác (polygon)
if "coordinates" in data:
    coordinates = data["coordinates"]
    if not isinstance(coordinates, list) or not all(
        isinstance(coord, list) and len(coord) == 2 and all(isinstance(x, (int, float)) for x in coord)
        for coord in coordinates
    ):
        return jsonify({"error": "Tọa độ phải là danh sách các cặp [lat, lng]"}), 400

    no_flight_data = {
        "name": name,
        "type": "polygon",
        "coordinates": coordinates
    }
    result = No_Flight_collection.insert_one(no_flight_data)
    return jsonify({
        "message": "Đã thêm vùng cấm bay đa giác",
        "id": str(result.inserted_id)
    }), 201

```

Hình

Nếu dữ liệu chứa trường "coordinates", hệ thống hiểu đây là vùng cấm bay dạng đa giác. Tọa độ phải là một danh sách các cặp [vĩ độ, kinh độ] hợp lệ, với mỗi giá trị là kiểu số. Nếu dữ liệu hợp lệ, hệ thống sẽ gán kiểu "polygon" cho vùng này và lưu thông tin vào cơ sở dữ liệu MongoDB. Kết quả trả về bao gồm thông báo thành công và ID của bản ghi vừa được thêm.

```

# Nếu là vùng đa giác (polygon)
if "coordinates" in data:
    coordinates = data["coordinates"]
    if not isinstance(coordinates, list) or not all(
        isinstance(coord, list) and len(coord) == 2 and all(isinstance(x, (int, float)) for x in coord)
        for coord in coordinates
    ):
        return jsonify({"error": "Tọa độ phải là danh sách các cặp [lat, lng]"}), 400

    no_flight_data = {
        "name": name,
        "type": "polygon",
        "coordinates": coordinates
    }
    result = No_Flight_collection.insert_one(no_flight_data)
    return jsonify({
        "message": "Đã thêm vùng cấm bay đa giác",
        "id": str(result.inserted_id)
    }), 201

```

Hình

Trong trường hợp dữ liệu chưa trùm "center" và "radius", hệ thống sẽ xử lý như một vùng cấm bay hình tròn. Tọa độ tâm phải là một cặp $[lat, lng]$ và bán kính là một số dương. Sau khi kiểm tra hợp lệ, dữ liệu sẽ được gán kiểu "circle" và lưu vào cơ sở dữ liệu. Tương tự, kết quả trả về gồm thông báo và ID.

```
else:  
    return jsonify({"error": "Thiếu dữ liệu vùng cấm bay"}), 400
```

Nếu không đáp ứng được bất kỳ định dạng nào ở trên, hệ thống sẽ trả lỗi báo thiếu dữ liệu cần thiết để xác định vùng cấm bay. Đoạn mã này đảm bảo rằng mọi dữ liệu được nhập vào đều có cấu trúc rõ ràng, đầy đủ và hợp lệ, nhằm hỗ trợ tốt cho việc quản lý không phận trong ứng dụng.

- **Truy xuất lại dữ liệu vùng cấm bay**

Đoạn mã trên định nghĩa một API endpoint với phương thức GET tại địa chỉ `/data-no-flight`, cho phép truy xuất dữ liệu các vùng cấm bay từ cơ sở dữ liệu MongoDB. Khi có yêu cầu được gửi đến, hệ thống sẽ truy cập vào collection `No_Flight` và trích xuất các trường thông tin cần thiết như tên vùng (name), loại vùng (type), tọa độ ranh giới (coordinates), tọa độ (center) và bán kính (radius). Kết quả vùng cấm được trả về dưới dạng JSON giúp frontend dễ dàng sử dụng để hiển thị các vùng cấm bay trên giao diện bản đồ.

```
# API GET DATA CẤM BAY  
@app.route("/data-no-flight", methods=["GET"])  
def get_data_no_flight():  
    no_flight_collection = db["No_Flight"]  
    no_flight_zones = list(no_flight_collection.find({}, {  
        "_id": 0,  
        "name": 1,  
        "type": 1,  
        "coordinates": 1,  
        "center": 1,  
        "radius": 1  
    }))  
    return jsonify(no_flight_zones), 200
```

Hình

4.4.2.3 Xử lý logic và kiểm tra tích hợp dữ liệu đường bay và vùng cấm bay

Đoạn code này chính là phần xử lý chức năng thêm đường bay cho giao diện "Quản lý các chuyến bay". Khi người dùng trên giao diện nhập tên chuyến bay và danh sách các điểm waypoint, rồi nhấn nút để thêm mới, hệ thống sẽ gửi yêu cầu POST đến địa chỉ /flight_path, kèm theo dữ liệu JSON chứa thông tin đó. Đoạn code sẽ kiểm tra dữ liệu hợp lệ và nếu hợp lệ, nó sẽ lưu thông tin đường bay mới vào cơ sở dữ liệu. Nhờ vậy, sau khi thêm thành công, danh sách các đường bay trên giao diện sẽ được cập nhật, hiển thị chuyến bay vừa tạo cùng các điểm trong hành trình. Đây là cầu nối giữa giao diện người dùng và phần backend xử lý lưu trữ dữ liệu.

```
@app.route("/flight_path", methods=["POST"])
def add_flight_path():
    data = request.json
    if not data or "name" not in data or "waypoints" not in data:
        return jsonify({"error": "Thiếu 'name' hoặc 'waypoints'"}, 400

    name = data["name"]
    waypoints = data["waypoints"]

    if not isinstance(waypoints, list) or not all(isinstance(p, str) for p in waypoints):
        return jsonify({"error": "'waypoints' phải là danh sách các chuỗi tên waypoint"}), 400

    flight_path_data = {
        "name": name,
        "waypoints": waypoints
    }

    result = flight_path_collection.insert_one(flight_path_data)
    return jsonify({
        "message": "Đã thêm đường bay",
        "id": str(result.inserted_id)
    }), 201
```

Hình

- **Lấy tất cả đường bay**

Hình... mô tả phương thức **GET** tại /flight_path/all nhằm **tất cả** danh sách các **đường bay** đang được lưu trữ trong cơ sở dữ liệu flight_path_collection. Khi người dùng gửi yêu cầu muốn xem chi tiết chuyến này, hệ thống sẽ truy vấn toàn bộ tài liệu trong bộ sưu tập, sau đó trả về danh sách các đường bay (tên đường bay và các lô điểm) dưới dạng JSON. Mục đích của chức năng này là cầu nối giữa giao diện người dùng và phần backend xử lý lưu trữ dữ liệu.

```

@app.route("/flight_path/all", methods=["GET"])
def get_all_flight_paths():
    flights = list(flight_path_collection.find({}, {"_id": 0}))
    return jsonify(flights), 200

```

Hình 4

- **Lấy khoảng cách các đoạn bay**

Hình 4. được thiết lập nhằm mục đích tạo một API để lấy dữ liệu khoảng cách giữa các đoạn bay từ cơ sở dữ liệu. Khi người dùng gửi yêu cầu GET đến /data-distance, hệ thống sẽ trả về danh sách các đoạn bay cùng với khoảng cách tương ứng mục đích của việc thiết lập này là để cung cấp thông tin khoảng cách phục vụ cho chức năng tính toán độ dài của đường bay nhằm đưa ra những đường bay có lựa chọn tối ưu.

```

# API GET DATA KHOẢNG CÁCH
@app.route("/data-distance", methods=["GET"])
def get_data_distance():
    distances = list(distance_collection.find({}, {"_id": 0, "from": 1, "to": 1, "distance_nm": 1}))
    return jsonify(distances), 200

```

- **Nhận và kiểm tra dữ liệu đầu vào**

Hình dùng để **nhận dữ liệu JSON** từ yêu cầu POST gửi tới endpoint /flight_path/check. Dữ liệu kỳ vọng có trường "waypoints" (danh sách các tên điểm đường bay). Nếu danh sách này có ít hơn 2 điểm thì trả về lỗi kèm thông báo "Cần ít nhất 2 điểm" vì một đường bay hợp lệ phải có ít nhất một đoạn (2 điểm).

```

@app.route("/flight_path/check", methods=["POST"])
def check_flight_pathViolation():
    data = request.json
    waypoints = data.get("waypoints", [])
    if len(waypoints) < 2:
        return jsonify({"error": "Cần ít nhất 2 điểm"}), 400

```

Hình

- **Tiền xử lý vùng cấm bay:**

```

data_no_flight = requests.get(f"{connectAPI2}/data-no-flight").json()
no_flight_zones = [
    {
        "name": zone["name"],
        "polygon": Polygon([(coord[1], coord[0]) for coord in zone["coordinates"]])
    }
    for zone in data_no_flight
]

```

Hình 4. thực hiện bước tiền xử lý vùng cấm bay bằng cách truy vấn API /data-no-flight để lấy danh sách các vùng cấm dưới dạng tập hợp tọa độ. Các tọa độ này được chuyển thành đối tượng Polygon bằng thư viện Shapely nhằm phục vụ cho các phép toán hình học, như kiểm tra đoạn đường bay có cắt hoặc nằm trong vùng cấm hay không.

- **Kiểm tra đường bay nhập thủ công có vi phạm không (POST/flight_path/check)**

```

# Tạo các đoạn và kiểm tra
violations = []
for i in range(len(waypoints) - 1):
    p1, p2 = waypoints[i], waypoints[i+1]
    if p1 not in point_map or p2 not in point_map:
        continue
    segment = LineString([point_map[p1], point_map[p2]])
    if is_blocked_by_no_flight(segment, zones):
        violations.append({"from": p1, "to": p2})

return jsonify({"violations": violations}), 200

```

Hình..... Kiểm tra đường bay nhập thủ công có vi phạm no-flight hay không

Mục đích: Việc kiểm tra xem đường bay do người dùng nhập có đi qua vùng cấm bay hay không giúp đánh giá tính khả thi và mức độ an toàn của lộ trình. Nếu phát hiện vi phạm, hệ thống sẽ cảnh báo và gợi ý sử dụng thuật toán A* để tìm đường thay thế.

Trước tiên, hệ thống truy xuất các waypoint từ cơ sở dữ liệu point_collection, lưu vào point_map dưới dạng từ điển với key là tên đường (ten_duong) và value là tọa độ dạng tuple (kinh_do, vi_do). Đồng thời, danh sách vùng cấm bay được tải thông qua hàm load_no_flight_zones, trả về các đối tượng hình học (ví dụ: Polygon, Circle) đã được dựng sẵn.

Sau khi có đủ dữ liệu, chương trình khởi tạo danh sách rỗng violations để ghi nhận các đoạn bay vi phạm. Hệ thống duyệt qua các waypoint do người dùng cung cấp, tạo các cặp điểm liên tiếp (p1, p2). Với mỗi cặp, nếu cả hai điểm tồn tại trong point_map, đoạn thẳng LineString sẽ được tạo bằng thư viện **Shapely**. Đoạn này được truyền vào hàm is_blocked_by_no_flight để kiểm tra xem có cắt hoặc nằm trong vùng cấm bay nào không. Nếu có, hệ thống ghi nhận đoạn vi phạm vào violations bằng cách lưu tên điểm đầu và cuối.

Sau khi xử lý toàn bộ lộ trình, API sẽ trả về danh sách các đoạn vi phạm dưới dạng JSON, giúp người dùng biết chính xác phần nào của đường bay đã đi vào vùng cấm. Đây là bước xử lý quan trọng trong API POST /flight_path/check, hỗ trợ người dùng kiểm tra thủ công trước khi lưu đường bay hoặc thực hiện bước tiếp theo.

- **Xem toàn bộ đoạn đường nào đi qua vùng cấm (GET /get-point-no-flight)**

Mục đích: Việc phân tích các **đường bay chính đã được lưu trữ** nhằm kiểm tra xem chúng có bị chèn lấn hoặc cắt qua các vùng cấm hiện có hay không là điều cần thiết. Điều này giúp đảm bảo rằng **thuật toán A*** sẽ chỉ đề xuất những đường bay thay thế dựa trên các tuyến có sẵn được xác nhận là hợp lệ và không vi phạm vùng cấm bay.

Hình Truy xuất các đường bay đi qua vùng cấm

Đoạn mã trên định nghĩa một API GET tại đường dẫn /get-point-no-flight, có chức năng kiểm tra xem các đoạn đường bay có cắt qua vùng cấm bay hay không. Khi được gọi, hàm sẽ gửi yêu cầu tới hai API khác để lấy dữ liệu: một API cung cấp các điểm tọa độ của các đường bay (/data-road), và API còn lại cung cấp danh sách các

vùng cấm bay (/data-no-flight) - đã được.....

```
violations = []

for road in data_points:
    ten_duong_chinh = road["ten_duong_chinh"]
    points = road["cac_ten_duong_con"]

    valid_points = [
        (p["kinh_do"], p["vi_do"]) for p in points
        if p["kinh_do"] is not None and p["vi_do"] is not None
    ]

    for i in range(len(valid_points) - 1):
        p1 = valid_points[i]
        p2 = valid_points[i + 1]
        segment = LineString([p1, p2])

        for zone in no_flight_zones:
            if segment.crosses(zone["polygon"]) or segment.within(zone["polygon"]):
                violations.append({
                    "ten_duong_chinh": ten_duong_chinh,
                    "zone": zone["name"],
                    "from": {"kinh_do": p1[0], "vi_do": p1[1]},
                    "to": {"kinh_do": p2[0], "vi_do": p2[1]}
                })

return jsonify(violations)
```

Hình.....Kiểm tra vi phạm vùng cấm hay không

Sau khi thu thập dữ liệu về các tuyến đường bay và vùng cấm, hệ thống sẽ đánh giá xem liệu các đường chính có đi qua vùng cấm hay không. Đầu tiên, một danh sách rỗng violations được khởi tạo để lưu các đoạn vi phạm. Chương trình duyệt qua từng tuyến trong data_points, lấy danh sách các điểm con và kiểm tra tính hợp lệ của tọa độ (kinh độ, vĩ độ), sau đó tạo danh sách valid_points chứa các cặp tọa độ hợp lệ.

Tiếp theo, hệ thống xét từng đoạn thẳng nối hai điểm liên tiếp và chuyển thành đối tượng LineString. Với mỗi đoạn, chương trình kiểm tra xem nó có cắt ngang (crosses) hoặc nằm hoàn toàn bên trong (within) bất kỳ vùng cấm nào không. Nếu phát hiện vi phạm, hệ thống ghi nhận vào violations gồm tên tuyến, tên vùng cấm bị ảnh hưởng và tọa độ hai đầu đoạn vi phạm.

Cuối cùng, danh sách vi phạm được trả về dưới dạng JSON để phục vụ hiển thị trực quan hoặc xử lý tiếp theo trên giao diện người dùng.

4.5 Tìm đường đi thay thế (bằng thuật toán A*)

- Haversine(p1, p2)

```
def haversine(p1, p2):  
    from math import radians, cos, sin, asin, sqrt  
    lat1, lon1 = p1  
    lat2, lon2 = p2  
    R = 6371 # km  
    dlat = radians(lat2 - lat1)  
    dlon = radians(lon2 - lon1)  
    a = sin(dlat / 2)**2 + cos(radians(lat1)) * cos(radians(lat2)) * sin(dlon / 2)**2  
    c = 2 * asin(sqrt(a))  
    return R * c * 0.539957
```

Hình..... Tính khoảng cách bằng hàm haversine

Đoạn mã trên định nghĩa hàm haversine(p1, p2) — một hàm dùng để tính khoảng cách thực tế (còn gọi là đường chim bay) giữa hai điểm địa lý được xác định bởi cặp tọa độ vĩ độ và kinh độ. Hàm này sử dụng công thức Haversine, một công thức phổ biến trong địa lý học để đo khoảng cách giữa hai điểm trên bề mặt trái đất, có tính đến độ cong của địa cầu. Kết quả trả về là đơn vị hải lý (nautical miles).

- **Load_no_flight_zones()**

```
def load_no_flight_zones():
    zones = []
    for z in No_Flight_collection.find({}, {"_id": 0}):
        z_type = z.get("type")

        if z_type == "polygon" and z.get("coordinates"):
            try:
                coords = [(pt[1], pt[0]) for pt in z["coordinates"]]
                polygon = Polygon(coords)
                zones.append(polygon)
            except Exception as e:
                print(f"[!] Lỗi polygon {z.get('name')}: {e}")

        elif z_type == "circle" and z.get("center") and z.get("radius"):
            try:
                lat, lng = z["center"]
                lat = float(lat)
                lng = float(lng)
                center_point = Point(lng, lat)

                radius_m = float(z["radius"]) # radius theo mét
                buffer_deg = radius_m / 111320 # mét → độ
                circle_polygon = center_point.buffer(buffer_deg)

                zones.append(circle_polygon)
            except Exception as e:
                print(f"[!] Lỗi circle {z.get('name')}: {e}")


```

Hình tải dữ liệu vùng cấm bay

Đoạn mã trên định nghĩa hàm `load_no_flight_zones()` với mục tiêu tải và chuẩn hóa dữ liệu các vùng cấm bay từ cơ sở dữ liệu MongoDB, để phục vụ việc kiểm tra va chạm (intersection) giữa đường bay và vùng cấm.

Truy vấn dữ liệu từ MongoDB: Hàm lấy toàn bộ tài liệu trong collection `No_Flight`, loại bỏ `_id` vì không cần thiết trong xử lý hình học.

Phân loại và xử lý vùng cấm: Nếu vùng cấm có kiểu `polygon` (đa giác) và có đủ `coordinates`, các tọa độ này (dưới dạng vĩ độ, kinh độ) sẽ được chuyển đổi thành các cặp (kinh độ, vĩ độ) để tạo một đối tượng `Polygon` từ thư viện Shapely. Đây là định dạng chuẩn (x, y) cho xử lý hình học. Nếu vùng cấm có kiểu `circle` (hình tròn), với `center` (tâm) và `radius` (bán kính) thì `Polygon` (tuy nhiên với hình tròn sẽ gọi là buffered point) cũng sẽ tạo ra một đa giác hình tròn đại diện cho vùng cấm.

```

    else:
        print(f"[!] Bỏ qua vùng không hợp lệ: {z.get('name')}")
    print(f"[+] Đã load {len(zones)} vùng cấm bay")
    return zones

```

Hình.... Xử lý phần kiểm tra vùng không hợp lệ và in ra số vùng cấm bay đã load được

Xử lý ngoại lệ: Nếu dữ liệu không hợp lệ hoặc thiếu thông tin cần thiết, hệ thống sẽ ghi log lỗi nhưng không dừng toàn bộ quá trình.

Kết quả: Trả về danh sách các vùng cấm đã được biểu diễn dưới dạng hình học (Polygon) để phục vụ việc kiểm tra va chạm với đường bay trong các thuật toán như A* hoặc Yen's. Tất cả vùng hợp lệ sẽ được đưa vào danh sách zones, và sau đó được dùng để kiểm tra cắt qua vùng cấm.

- **is_blocked_by_no_flight(segment, zones)**

Việc nhóm đề tài lựa chọn không sử dụng các đường bay có sẵn mà thay vào đó cho phép người dùng nhập tùy ý điểm bắt đầu và điểm kết thúc khiến cho quá trình tìm đường bay trở nên linh hoạt hơn vậy nên điều này đòi hỏi sự kiểm tra vùng cấm phải được thực hiện trực tiếp trong thuật toán tìm đường. Đó chính là lí do vì sao trong khi API GET /get-point-no-flight đã thực hiện kiểm tra các tuyến bay cố định xem có vi phạm vùng cấm hay không, thì thuật toán A* vẫn cần tự đánh giá lại các đoạn đường trong quá trình tìm kiếm.

Mục đích là đảm bảo rằng với bất kỳ tổ hợp điểm bay nào do người dùng chỉ định, hệ thống đều sẽ phát hiện và loại bỏ các phương án có khả năng đi qua vùng cấm. Nhờ đó, những đường bay được đề xuất cuối cùng luôn đảm bảo an toàn và không xâm phạm không phận bị giới hạn. Đây là bước xử lý phù hợp và cần thiết trong trường hợp ứng dụng cho phép bay theo mô hình "điểm nối điểm" thay vì tuân theo các tuyến định trước.

```

def is_blocked_by_no_flight(segment, zones):
    for zone in zones:
        if segment.intersects(zone):
            print(f"[X] Đoạn bị chặn bởi zone")
            return True
    return False

```

Hình..... Kiểm tra phạm vi vùng cấm bay

Trong quá trình kiểm tra, hàm sẽ duyệt qua từng vùng trong danh sách zones - mỗi vùng là một đối tượng hình học (Polygon hoặc buffered Point) mô phỏng vùng cấm bay. Nếu đoạn đường bay segment giao nhau với bất kỳ vùng cấm nào (thông qua hàm intersects của Shapely), hàm sẽ in ra cảnh báo và trả về True, tức là đoạn đường đó bị chặn bởi vùng cấm. Nếu không có vùng cấm nào bị vi phạm, hàm sẽ trả về False. Đây là một phần quan trọng trong logic đảm bảo an toàn bay, giúp loại bỏ các phương án đường bay không hợp lệ trong các thuật toán như A*.

- **a_star(start, goal, points_dict, graph_edges, no_flight_polygons)**

```

def a_star(start, goal, points_dict, graph_edges, no_flight_polygons):
    open_set = [(0, [start])]
    paths = []

    while open_set:
        total_cost, path = heapq.heappop(open_set)
        current = path[-1]

        if current == goal:
            return path

        for neighbor, dist in graph_edges.get(current, []):
            if neighbor in path:
                continue
            if neighbor not in points_dict or current not in points_dict:
                continue

            segment = LineString([
                (points_dict[current][1], points_dict[current][0]), # (lng, lat)
                (points_dict[neighbor][1], points_dict[neighbor][0])
            ])

            if is_blocked_by_no_flight(segment, no_flight_polygons):
                continue
            # violate = False
            # for zone in no_flight_polygons:
            #     if isinstance(zone, Polygon):
            #         if segment.intersects(zone):
            #             violate = True

```

```

#             break
#     elif isinstance(zone, tuple):
#         center, radius_km = zone
#         circle = center.buffer(radius_km / 111.32)
#         if segment.intersects(circle):
#             violate = True
#             break
# if violate:
#     continue

new_path = path + [neighbor]
cost = sum(
    graph_edges[new_path[i]][j][1]
    for i in range(len(new_path) - 1)
    for j in range(len(graph_edges[new_path[i]])))
    if graph_edges[new_path[i]][j][0] == new_path[i + 1]
)
est = haversine(points_dict[neighbor], points_dict[goal])
heappush(open_set, (cost + est, new_path))

return []

```

Hình..... Đoạn code của thuật toán A*

Đoạn mã trên triển khai thuật toán A* nhằm tìm đường bay tối ưu từ điểm xuất phát start đến điểm đích goal, trong khi vẫn đảm bảo rằng các đoạn đường đi không cắt qua vùng cấm bay.

Thuật toán bắt đầu bằng cách khởi tạo một hàng đợi ưu tiên open_set, nơi mỗi phần tử gồm một tổng chi phí và đường đi tương ứng. Trong mỗi vòng lặp, thuật toán lấy ra đường có chi phí nhỏ nhất để tiếp tục mở rộng. Với mỗi điểm lân cận (neighbor) của điểm hiện tại (current), thuật toán tạo một đoạn đường (segment) nối hai điểm và sử dụng hàm is_blocked_by_no_flight() để kiểm tra xem đoạn này có cắt qua vùng cấm bay hay không. Nếu có, đoạn đó bị loại bỏ khỏi đường đi.

Nếu đoạn hợp lệ, thuật toán tính lại chi phí thực tế từ đầu đến điểm đó và ước lượng chi phí còn lại bằng công thức Haversine. Sau đó, nó đưa đường đi mới vào open_set để tiếp tục xét trong các vòng lặp tiếp theo. Thuật toán kết thúc khi tìm được một đường hợp lệ đến đích hoặc khi không còn đường nào khả thi.

Nhờ cơ chế kiểm tra vùng cấm tích hợp, thuật toán này đảm bảo rằng tất cả các đường bay được đề xuất đều hợp lệ và không vi phạm an toàn không phận.

4.6. Lấy các đường bay có tọa độ đầy đủ

Hình ... mô tả quá trình truy xuất danh sách các đường bay có đầy đủ tọa độ các lô điểm. Hệ thống kiểm tra từng đường bay và chỉ giữ lại các lô điểm có đủ thông tin vĩ độ và kinh độ. Chỉ những đường bay có từ hai điểm tọa độ trở lên mới được đưa vào kết quả trả về, nhằm đảm bảo dữ liệu đủ điều kiện cho các xử lý tiếp theo.

Chức năng này phục vụ cho các tác vụ như hiển thị bản đồ, vẽ chính xác tuyến bay, kiểm tra vi phạm vùng cấm và phân tích tuyến đường, bao gồm cả việc tính toán khoảng cách bay.

```
@app.route("/get-data-flight_path", methods=["GET"])
def get_data_flight_path():
    flight_paths = list(flight_path_collection.find({}, {"_id": 0, "name": 1, "waypoints": 1}))

    result = []

    for flight in flight_paths:
        name = flight["name"]
        waypoint_names = flight["waypoints"]

        coordinates = []
        for wp in waypoint_names:
            point = point_collection.find_one(
                {"ten_duong": wp},
                {"_id": 0, "vi_do": 1, "kinh_do": 1}
            )
            if point and point.get("vi_do") is not None and point.get("kinh_do") is not None:
                coordinates.append([point["vi_do"], point["kinh_do"]])

        if len(coordinates) >= 2:
            result.append({
                "name": name,
                "coordinates": coordinates
            })

    return jsonify(result), 200
```

Hình..... Lấy đường bay có tọa độ đầy đủ

4.7 Xóa dữ liệu

- Xóa vùng cấm bay

Hình 4.2 xây dựng một API với phương thức DELETE tại /no_flight, cho phép người dùng xóa một vùng cấm bay khỏi cơ sở dữ liệu dựa trên tên được cung cấp. Khi nhận yêu cầu, hệ thống kiểm tra xem trường "name" có tồn tại trong dữ liệu đầu vào

hay không; nếu không có, sẽ trả về thông báo lỗi kèm mã trạng thái. Nếu tên hợp lệ, hệ thống tìm và xóa bản ghi tương ứng trong MongoDB.

```
@app.route("/no_flight", methods=["DELETE"])
def delete_no_flight_zone():
    data = request.json
    name = data.get("name")
    if not name:
        return jsonify({"error": "Thiếu tên vùng cấm"}), 400

    result = No_Flight_collection.delete_one({"name": name})
    if result.deleted_count > 0:
        return jsonify({"message": f"Đã xóa vùng cấm {name}"}), 200
    else:
        return jsonify({"error": "Không tìm thấy vùng cấm để xóa"}), 404
```

Hình

- Xóa đường bay

Hình 4. xây dựng một API với phương thức POST tại /flight_path/delete, cho phép xóa một đường bay khỏi cơ sở dữ liệu dựa trên tên được cung cấp từ phía người dùng. Khi nhận yêu cầu, hệ thống sẽ kiểm tra xem dữ liệu có chứa trường "name" hay không; nếu thiếu, phản hồi lỗi kèm mã trạng thái. Trường hợp tên hợp lệ, hệ thống thực hiện xóa bản ghi tương ứng trong MongoDB.

```
@app.route("/flight_path/delete", methods=["POST"])
def delete_flight_path():
    data = request.json
    name = data.get("name")
    if not name:
        return jsonify({"error": "Thiếu tên chuyến bay để xóa"}), 400

    result = flight_path_collection.delete_one({"name": name})
    if result.deleted_count > 0:
        return jsonify({"message": f"Đã xóa chuyến bay {name}"}), 200
    else:
        return jsonify({"error": "Không tìm thấy chuyến bay để xóa"}), 404
```

Hình

4.8 Thiết lập tình huống mô phỏng trên ứng dụng

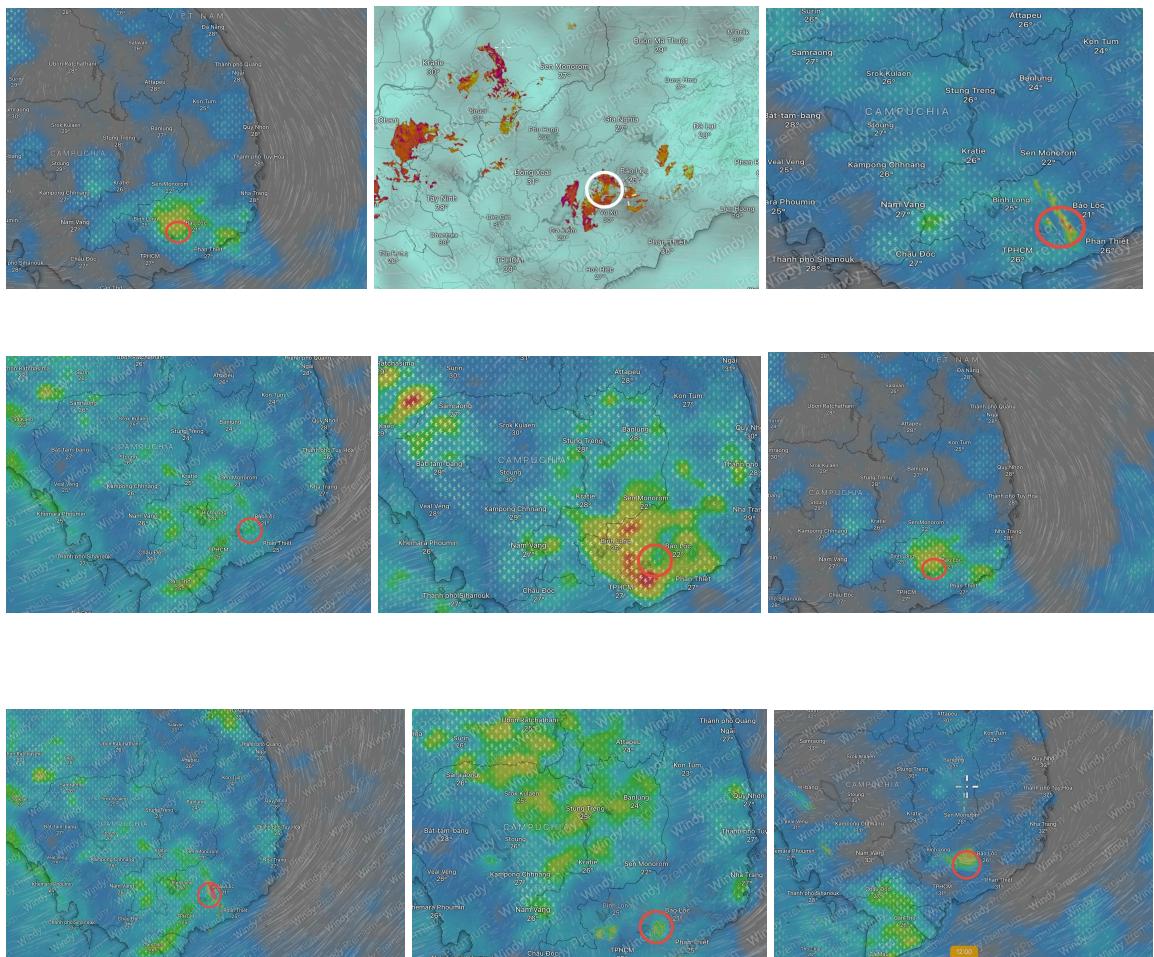
Nhằm đánh giá hiệu quả của ứng dụng trong việc đề xuất đường bay thay thế, nhóm đề tài thiết lập một tình huống mô phỏng để kiểm thử thuật toán A* trong các kịch bản giả định khác nhau, qua đó đánh giá tính khả thi và hiệu suất của hệ thống.

5.5.1 Khảo sát thực tế

Để xây dựng hệ thống tuyến bay thay thế với thực tế khai thác, nhóm đề tài lựa chọn đường bay W1 làm đối tượng khảo sát. Trong vòng hai tuần từ ngày 25/05/2025 đến ngày 05/06/2025, nhóm tiến hành thu thập và ghi nhận số liệu về hiện tượng thời tiết ảnh hưởng đến khu vực này, được tổng hợp tại Bảng

Ngày	Giờ	Loại thời tiết	Vùng ảnh hưởng
25/05/2025	09:00	CAT	(11°37'45"N, 107°36'37"E), (11°33'49"N, 107°36'15"E), (11°31'41"N, 107°32'52"E), (11°33'54"N, 107°35'16"E)
	17:00	Mưa cường độ trung bình	11°35'45"N, 107°36'23"E, bán kính 8NM
26/05/2025	09:00		Không có
	17:00	Mưa cường độ trung bình	(11°35'12"N, 107°33'31"E), bán kính 5NM
27/05/2025	09:00	Mưa cường độ mạnh	(11°35'58"N, 107°37'11"E), bán kính 12NM
	17:00		Không có
28/05/2025	09:00		Không có
	17:00	Mưa cường độ trung bình	(11°31'42"N, 107°33'34"E), (11°35'41"N, 107°36'8"E), (11°37'48"N, 107°38'50"E)
29/05/2025	09:00	Giông mạnh	(11°39'35"N, 107°32'34"E), (12°31'49"N, 107°31'19"E), (11°31'44"N, 107°35'57"E), (12°38'58"N, 107°14'22"E)
	17:00	Mưa cường độ mạnh	11°36'46"N, 107°36'32"E, bán kính 5NM
30/05/2025	09:00	CAT	11°35'41"N, 107°37'8"E, bán kính 7NM
	17:00	Mưa cường độ mạnh	11°35'59"N, 107°37'44"E, bán kính 10NM
31/05/2025	09:00		Không có
	17:00	Mưa cường độ trung bình	(11°32'43"N, 107°34'35"E), (11°36'47"N, 107°38'18"E), (11°39'51"N, 107°32'37"E)
1/6/2025	09:00	Giông mạnh	11°17'51"N, 107°16'27"E, bán kính 9NM
	17:00	Mưa cường độ mạnh	(11°37'45"N, 107°36'37"E), (11°35'41"N, 107°37'8"E), (11°37'48"N, 107°36'50"E), (11°36'58"N, 107°36'27"E)
2/6/2025	09:00		Không có
	17:00		Không có
3/6/2025	09:00	Mưa cường độ mạnh	(11°11'25"N, 107°16'57"E), (12°15'11"N, 107°17'58"E), (11°37'43"N, 107°26'55"E), (11°56'52"N, 107°35'26"E)
	17:00	CAT	11°37'48"N, 107°36'50"E, bán kính 12NM
4/6/2025	09:00		Không có
	17:00		Không có
5/6/2025	09:00	CAT	(11°14'45"N, 107°35'47"E), (12°16'29"N, 107°17'19"E), (11°32'15"N, 107°42'51"E)
	17:00	Giông mạnh	11°37'33"N, 107°37'21"E, bán kính 6NM
4/6/2025	09:00		Không có
	17:00		Không có
5/6/2025	09:00	Giông mạnh	11°36'58"N, 107°36'27"E, bán kính 10NM
	17:00		Không có

Dựa trên dữ liệu từ phần mềm Windy, mặc dù vùng ảnh hưởng thời tiết thay đổi theo từng ngày, nhưng một đặc điểm đáng chú ý là đoạn từ điểm AC đến BMT trên đường bay W1 thường xuyên xuất hiện thời tiết xấu như mưa cường độ mạnh, mưa cường độ trung bình, giông mạnh và CAT – được minh họa tại Hình Trong phần lớn các ngày khảo sát, tàu bay có xu hướng đổi độ cao hoặc chuyển sang đường bay khác để tránh khu vực này, cho thấy đây là một điểm nghẽn trong mạng lưới đường bay hiện tại.

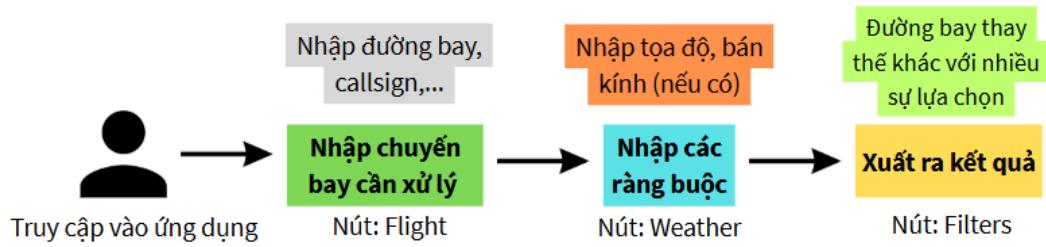


Hình.... Kết quả khảo sát khu vực tiêu biểu thường xuyên xảy ra các hiện tượng thời tiết cực đoan

Nhóm sẽ lấy vùng ảnh hưởng của ngày 29/05/2025 lúc 17h để làm ví dụ cho phần ứng dụng có tâm tọa độ là [11°36'46"N 107°36'32"E](#) và bán kính ảnh hưởng 5NM.

5.5.2 Chạy ứng dụng và xuất kết quả

Sau khi hoàn thành việc xây dựng và phát triển phần mã nguồn cho ứng dụng, nhóm dự án sẽ tiến hành bước tiếp theo là kiểm thử vận hành ứng dụng trong môi trường mô phỏng hoặc thực tế. Quá trình sử dụng ứng dụng sẽ được thực hiện theo một chuỗi các bước tuần tự nhằm đảm bảo việc khai thác chức năng một cách hiệu quả, đúng với mục tiêu thiết kế ban đầu. Các bước cụ thể trong quy trình sử dụng ứng dụng được minh họa như Hình.



Hình: Quy trình sử dụng ứng dụng

Dựa trên quy trình này, nhóm sẽ tiến hành thực nghiệm giả định với một chuyến bay cụ thể và bị ràng buộc bởi điều kiện thời tiết khiến cho tàu bay cần phải có các phương án đề xuất đường bay khác và thực hiện đề xuất chúng thông qua ứng dụng này.

Bước 1: Người dùng truy cập vào ứng dụng



Hình..... Giao diện mô phỏng đường hàng không Việt Nam của ứng dụng

Bước 2: Người dùng tiến hành nhập thông tin của chuyến bay

(Chèn ảnh KHBay)

Trong lần chạy thử nghiệm này, nhóm tiến hành mô phỏng một tình huống giả định đối với chuyến bay mang số hiệu HVN921, khởi hành từ Cảng hàng không quốc tế Tân Sơn Nhất (SGN) và dự kiến hạ cánh tại Cảng hàng không quốc tế Đà Nẵng (DAD). Dựa trên kế hoạch bay đã được thiết lập, có thể xác định rằng chuyến bay sẽ có đường đi là:.... đi qua các điểm cố định (fix) theo trình tự sau: TSH → AC → BMT → MUMGA → BANSU → DAN. Sau khi thu thập đầy đủ thông tin kế hoạch bay, nhóm tiến hành thao tác trên giao diện mô phỏng như sau:

- Chọn thanh công cụ “**Flight**”;
- Nhập các thông tin tương ứng vào các trường hiển thị, bao gồm: số hiệu chuyến bay, sân bay khởi hành, sân bay đến, các điểm cố định, v.v.;
- Nhấn nút “**Gửi chuyến bay**” để hệ thống tiếp nhận và xử lý kế hoạch bay.



Hình.... Giao diện của người dùng khi thực hiện thao tác thêm chuyến bay

Sau khi hoàn tất các bước trên, hệ thống sẽ hiển thị giao diện mô phỏng hành trình bay của HVN921, như được minh họa ở hình dưới đây. Giao diện thể hiện tuyến bay dưới dạng đường nối giữa các điểm cố định, đồng thời cung cấp các thông tin chi tiết về hành trình để phục vụ cho các bước mô phỏng tiếp theo.

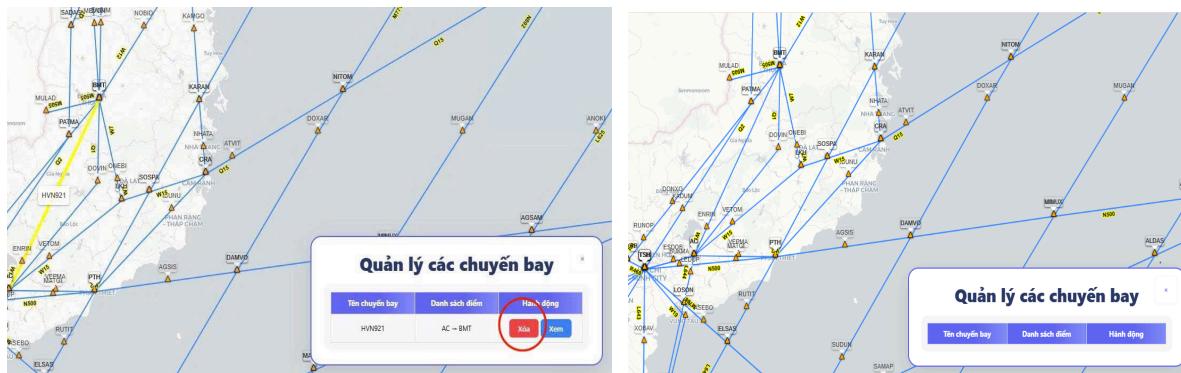


Hình.... Giao diện của ứng dụng sau khi thêm chuyến bay (đường màu vàng là đường bay bị ảnh hưởng của chuyến bay HVN921)

Trong trường hợp người dùng muốn xóa một chuyến bay đã nhập, bao gồm cả thông tin và đường bay hiển thị trên bản đồ, thực hiện theo các bước sau:

1. Truy cập vào mục “**Filter**” trên thanh công cụ;
 2. Chọn chuyến bay cần xóa trong danh sách;
 3. Nhấn nút “**Xóa**” để loại bỏ chuyến bay khỏi hệ thống.

Sau khi xác nhận thao tác, toàn bộ dữ liệu liên quan đến chuyến bay đó, bao gồm đường bay hiển thị trên bản đồ, sẽ được xóa hoàn toàn khỏi giao diện, giúp người dùng dễ dàng quản lý và cập nhật lại kế hoạch bay khi cần thiết.



Hình....Giao diện quản lý thao tác các chuyến bay và giao diện của ứng dụng sau khi nhấn lệnh “Xóa” chuyến bay đã nhập

Bước 3: Người dùng tiến hành nhập thông tin các ràng buộc

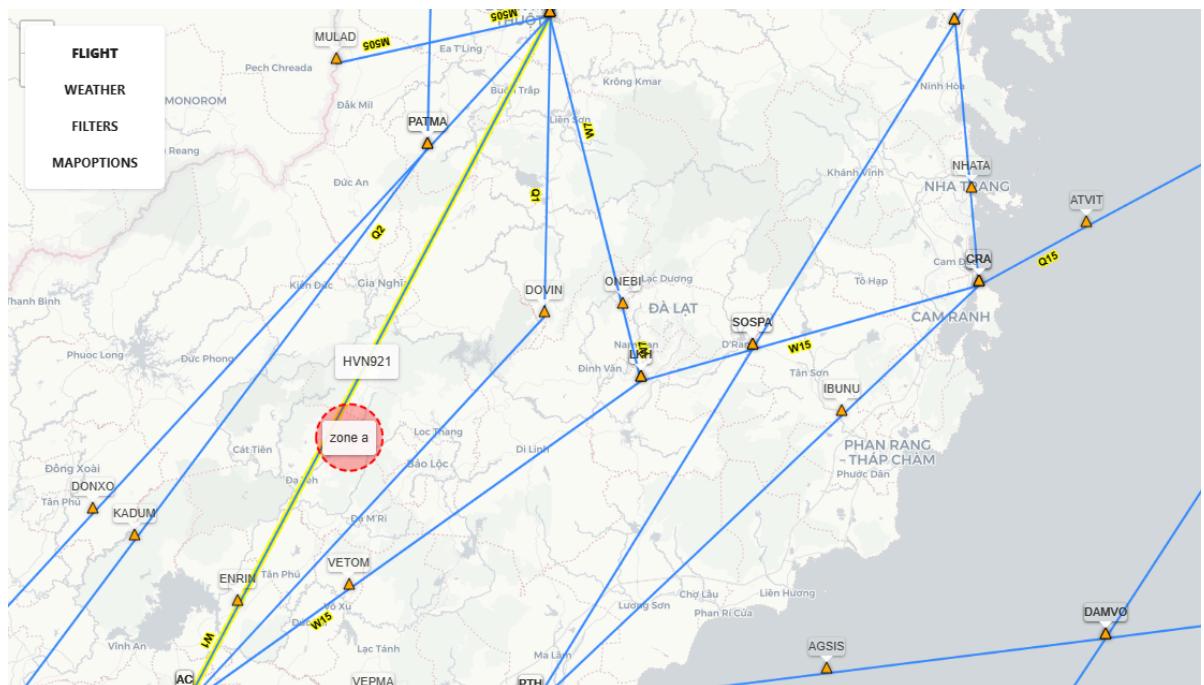
Tại bước này, dựa trên kết quả phân tích hiện tượng thời tiết đã được trình bày tại mục 5.5.1, nhóm tiến hành xác định một khu vực có tần suất xảy ra hiện tượng thời tiết cực đoan cao, được ghi nhận liên tục trong suốt thời gian quan sát. Kết quả cho thấy khu vực bị ảnh hưởng nằm dọc theo đường hàng không W1, với tọa độ trung tâm là: $11^{\circ}36'46''N$, $107^{\circ}36'32''E$. Phạm vi ảnh hưởng của khu vực này được xác định với bán kính 5 hải lý (5 NM). Sau khi đã thu thập được đầy đủ thông tin, người dùng sẽ tiến hành thao tác như sau:

- Chọn thanh công cụ “**Weather**” trên giao diện ứng dụng;
- Nhập các thông tin liên quan, bao gồm tọa độ, bán kính ảnh hưởng, loại hiện tượng thời tiết, v.v.;
- Nhấn nút “**Thêm vào bản đồ**” để hiển thị khu vực nguy hiểm trên giao diện bản đồ.



Hình.....: Giao diện của ứng dụng khi thực hiện thao tác nhập khu vực ràng buộc

Sau khi hoàn tất, hệ thống sẽ hiển thị một vùng chướng ngại thời tiết tại vị trí đã nhập, minh họa rõ ràng bằng ký hiệu hoặc màu sắc nổi bật. Hình dưới đây minh họa kết quả sau khi thêm khu vực thời tiết nguy hiểm vào bản đồ.

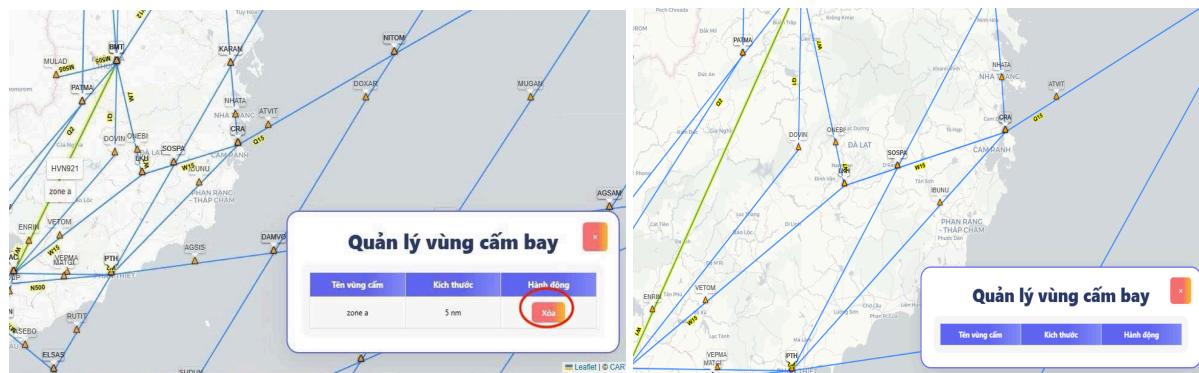


Hình.....Giao diện của ứng dụng sau khi thêm khu vực ràng buộc (màu đỏ), đường bay mong muốn (màu vàng)

Trong trường hợp người sử dụng muốn xóa dữ liệu “Ràng buộc” (trong tình huống này là khu vực có hiện tượng thời tiết cực đoan đã được thêm vào bản đồ), người dùng thực hiện theo các bước sau:

- Truy cập vào mục “**Map Options**” trên thanh công cụ chính của ứng dụng;
- Tại đây, chọn nút “**Xóa**” để tiến hành loại bỏ tất cả các ràng buộc đang hiển thị trên bản đồ.

Sau khi thao tác thành công, các vùng dữ liệu liên quan đến ràng buộc – bao gồm khu vực thời tiết nguy hiểm – sẽ được gỡ bỏ khỏi bản đồ, trả lại giao diện hiển thị mặc định cho người dùng tiếp tục thao tác các bước tiếp theo như tái lập kế hoạch bay hoặc chọn phương án thay thế được hiển thị trên hình ...



Hình....Giao diện quản lý thao tác vùng cấm bay và giao diện của ứng dụng sau khi nhấn lệnh “Xóa” khu vực ràng buộc đã nhập

Bước 4: Thực hiện chạy thuật toán A* tìm đường bay thay thế

Tại bước này, do điều kiện thời tiết xấu, tàu bay có khả năng không thể tiếp tục bay qua đoạn đường hàng không ENRIN – BMT. Trước tình huống đó, nhóm đã quyết định áp dụng thuật toán A* để tìm kiếm các phương án đường bay thay thế khác, dựa trên tiêu chí tối ưu về khoảng cách, được sắp xếp từ ngắn nhất đến dài nhất. Xét đến cấu trúc mạng đường hàng không tại Việt Nam, cũng như tính khả thi trong điều phối, nhóm quyết định thiết lập cho ứng dụng để xuất tối đa 5 phương án thay thế khả thi

sau khi thuật toán A* được thực thi. Để tiến hành tìm kiếm, người dùng thực hiện như sau:

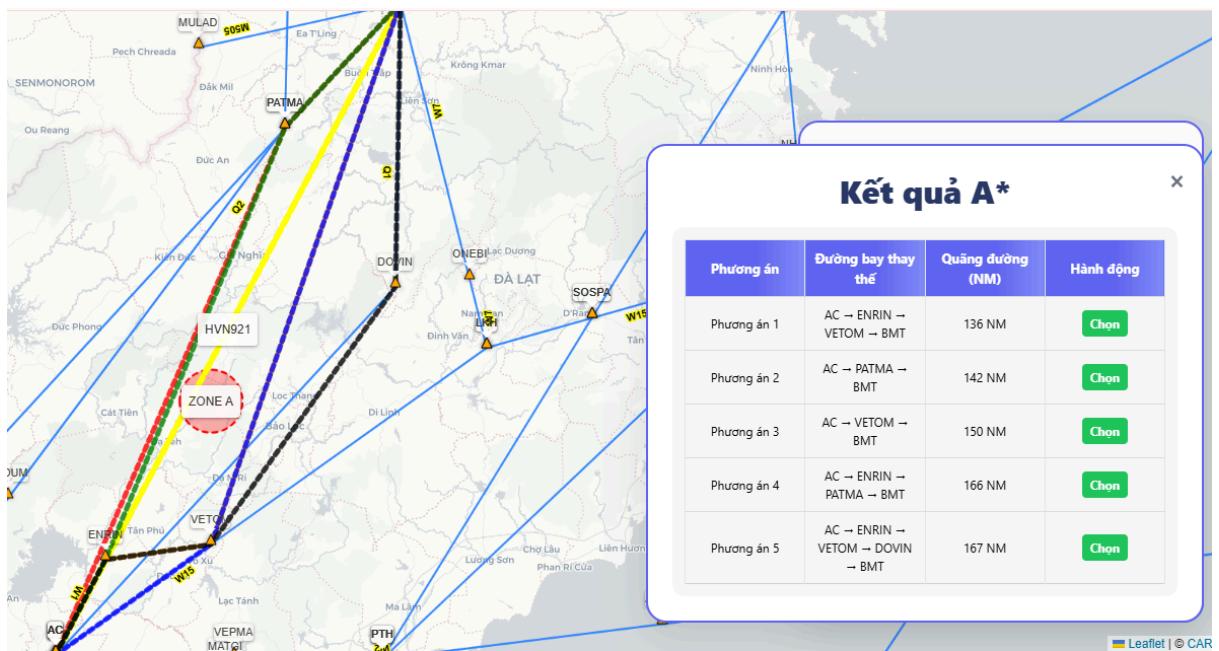
- Chọn mục “**Filters**” trên thanh công cụ giao diện;
- Chọn mục “**Hành động**” nếu như người dùng có nhu cầu tiếp tục chạy thuật toán A*, ngược lại chọn mục “**Hủy**” nếu như người dùng muốn xóa chuyến bay đó trên bản đồ
- Nhấn nút “**Chạy thuật toán A***” để khởi động quá trình tính toán.



*Hình.... Giao diện của ứng dụng cho phần thuật toán A**

Đối với mỗi phương án đường bay thay thế do thuật toán A* đề xuất, người dùng có thể thao tác trực tiếp trên giao diện bản đồ như sau:

- Nhấn nút “**Chọn**” để hiển thị đường bay đề xuất tương ứng trên bản đồ;
- Nhấn nút “**Hủy**” để xóa đường bay đó khỏi bản đồ.



Hình.....Giao diện của ứng dụng khi chạy ra kết quả của thuật toán A*

Nhằm đảm bảo sự phân biệt trực quan rõ ràng giữa các phương án, mỗi đường bay được hiển thị bằng một màu sắc khác nhau. Từ đó ta thấy, khi chuyến bay HVN921 cần phải đổi đường do có sự xuất hiện của thời tiết xấu tại đường W1, thuật toán sẽ đề xuất cho người sử dụng những phương án như sau: (chèn ảnh)

5.6 Đánh giá kết quả ứng dụng

Sau khi áp dụng ứng dụng do nhóm đề tài phát triển vào tình huống giả lập nêu trên, kết quả cho thấy thuật toán A* đã đề xuất được nhiều phương án đường bay thay thế, phù hợp với các điều kiện ràng buộc đã thiết lập. Đáng chú ý, phần lớn các phương án được đề xuất đều có xu hướng tuân theo mô hình bay tự do, tức là kết nối trực tiếp giữa các điểm thay vì đi dọc theo các đường hàng không cố định. Đây cũng chính là tinh thần của việc xây dựng Playbook – thiết kế những đường bay ngắn hơn, linh hoạt hơn, nhằm né tránh các vùng ảnh hưởng mà vẫn đảm bảo hiệu quả vận hành.

Xét trên khía cạnh kiểm soát không lưu, trong thực tế tàu bay hoàn toàn có thể thực hiện các đường bay dưới dạng như vậy, đặc biệt trong các tình huống bất thường hoặc khi không lưu cần thiết lập các phương án linh hoạt. Hơn nữa, trong vùng FIR Hồ Chí Minh, phần lớn không phận đều được phủ bởi hệ thống radar và ADS-B, giúp

duy trì khả năng giám sát và cập nhật vị trí tàu bay một cách liên tục, đảm bảo an toàn và hiệu quả trong điều hành.

Tuy nhiên, xét ở góc độ đồng vùng trời, việc tàu bay bay lêch khỏi các đường hàng không cố định lại đặt ra yêu cầu bắt buộc phải có sự phối hợp giữa các cơ quan quản lý không phận, đặc biệt là giữa dân sự và quân sự. Đây là yếu tố then chốt để đảm bảo hoạt động bay hài hòa và không phát sinh xung đột trong khai thác không phận dùng chung. Trong bối cảnh đó, việc sử dụng các tuyến bay đã được xác lập sẵn trong Playbook thay vì để kiểm soát viên điều phối tàu bay theo các đường không xác định là một giải pháp chủ động và có hệ thống hơn.

Sau khi sử dụng phần mềm để xây dựng và kiểm thử các phương án, các cơ quan dân sự cần tiến hành đánh giá hiệu quả của từng tuyến bay thay thế, đồng thời trao đổi, đàm phán với phía quân sự nhằm xem xét các ràng buộc cần thiết. Đây là bước quan trọng nhằm xác minh tính khả thi của các tuyến bay đề xuất, làm rõ những giới hạn về vùng cấm, vùng hạn chế, cũng như các điều kiện áp dụng cụ thể. Trên cơ sở thống nhất giữa các bên, các tuyến bay thay thế khả thi sẽ được đưa vào tài Playbook bao gồm đầy đủ thông tin cần thiết và được phổ biến giữa các đơn vị liên quan.

Trong quá trình khai thác thực tế, khi một chuyến bay gặp tình huống tương tự như tình huống mô phỏng, KSVKL sẽ có cơ sở pháp lý và nghiệp vụ rõ ràng để chủ động sử dụng tuyến bay thay thế đã được cho phép. Đồng thời, việc sớm đưa tàu bay trở lại đường bay kế hoạch cũng được đặt làm ưu tiên nhằm bảo đảm sự ổn định trong vận hành. Về phía quân sự, khi đã nắm rõ trước lộ trình của tàu bay thông qua Playbook, việc giám sát và quản lý vùng trời cũng trở nên dễ dàng, chính xác và minh bạch hơn.

Ngoài ra, việc điều hướng tàu bay theo các tuyến cố định trong Playbook, thay vì xử lý tình huống một cách ngẫu nhiên, sẽ góp phần làm ổn định luồng không lưu, nâng cao trật tự điều hành và hiệu quả của hoạt động kiểm soát không lưu. Hình ... dưới đây minh họa một ví dụ Playbook route được nhóm thiết kế dựa trên tình huống giả lập đã trình bày.

CHƯƠNG 6: KẾT LUẬN VÀ KIẾN NGHỊ

6.1 Ưu điểm và hạn chế của đề tài nghiên cứu

6.1.1 Ưu điểm của đề tài nghiên cứu

Đề tài đã cho thấy một số điểm tích cực đáng ghi nhận trong quá trình triển khai. Nhóm đề tài lựa chọn hướng tiếp cận thực tiễn khi kết hợp thuật toán A* với bài toán điều hành bay. Thay vì dùng ở mô phỏng lý thuyết, đề tài hướng đến xây dựng công cụ ứng dụng, tạo nền tảng ban đầu cho việc đề xuất tuyến bay thay thế trong các tình huống bất thường.

Bên cạnh đó, việc tập trung nghiên cứu tại FIR HCM, một khu vực có mật độ bay cao và thường xuyên chịu ảnh hưởng bởi thời tiết, điều hành đã tăng tính thực tiễn

và mức độ thách thức cho đề tài. Đây là môi trường đủ phức tạp để kiểm nghiệm khả năng ứng dụng của mô hình trong điều kiện gần với thực tế vận hành.

Ngoài ra, công cụ phát triển trong đề tài bước đầu cho thấy tiềm năng hỗ trợ đề xuất và đánh giá tuyến bay thay thế tạm thời, đặc biệt trong giai đoạn tiền chiến thuật hoặc chiến thuật ngắn hạn. Dù chưa hoàn thiện, hệ thống đã thể hiện khả năng gợi ý lộ trình hợp lý dựa trên thuật toán tìm đường ngắn nhất có xét đến ràng buộc địa lý. Đây là nền tảng quan trọng cho các nghiên cứu mở rộng, nhất là khi được nâng cấp về dữ liệu, giao diện và khả năng tích hợp với các hệ thống điều hành bay hiện hữu.

6.1.2 Hạn chế của đề tài nghiên cứu

Mặc dù đề tài đã thể hiện được những điểm tích cực về mặt ý tưởng và tính ứng dụng, nhưng vẫn còn tồn tại một số hạn chế nhất định cả về phạm vi triển khai lẫn khả năng ứng dụng thực tiễn như sau:

- Phần mềm hiện chỉ áp dụng trong FIR Hồ Chí Minh, khiến mô hình mạng lưới bị đơn giản hóa, chưa phản ánh đúng độ phức tạp của hệ thống hàng không liên vùng. Do đó, các phương án thay thế do thuật toán A đề xuất còn đơn điệu, thiếu tính linh hoạt và khó mở rộng sang quy mô toàn vùng.
- Phần mềm vẫn ở giai đoạn thử nghiệm, tốc độ xử lý còn chậm và chưa tích hợp các dữ liệu động như thời tiết, mật độ không lưu hay vùng cấm bay. Việc thiếu các yếu tố này khiến hệ thống chỉ hoạt động trong môi trường giả lập đơn giản, chưa đủ năng lực hỗ trợ điều phối thời gian thực. Dữ liệu đầu vào chủ yếu được thu thập thủ công, làm giảm tính cập nhật và hiệu quả vận hành.
- Giao diện phần mềm còn sơ khai, chưa thân thiện với môi trường điều hành thực tế. Thiếu các tính năng như hiển thị đa lớp, bộ lọc dữ liệu, cảnh báo tự động hay bản đồ số chính xác khiến phần mềm chưa thể đóng vai trò như một công cụ hỗ trợ ra quyết định chuyên nghiệp.
- Việc chỉ sử dụng thuật toán A* mà chưa tích hợp các cơ chế điều chỉnh thích nghi hay đánh giá đa mục tiêu khiến kết quả tìm đường mang tính tối ưu về hình học, nhưng chưa phù hợp với các ràng buộc vận hành thực tế như không lưu, độ cao hay thời gian sử dụng đường bay.

6.2 Hướng phát triển của đề tài

Đề nâng cao chất lượng và tính ứng dụng thực tiễn của ứng dụng, đề tài trong tương lai có thể được phát triển theo một số hướng cụ thể như sau:

- Cần mở rộng phạm vi áp dụng, từ FIR HCM sang các vùng FIR kế cận, nhằm phản ánh đúng tính chất kết nối phức tạp của mạng lưới hàng không và cho phép đề xuất phương án thay thế linh hoạt hơn khi luồng bay bị ảnh hưởng.
- Hệ thống cần tích hợp dữ liệu thời gian thực từ nhiều nguồn: thời tiết, hoạt động bay và điều phối không lưu. Toàn bộ quy trình thu thập – xử lý – hiển thị dữ liệu sẽ được tự động hóa và đồng bộ, tăng khả năng phản ứng với tình huống thực tế và giảm thao tác thủ công.
- Mở rộng thử nghiệm với các thuật toán khác như Dijkstra, D*, Theta*, ACO, GA,... nhằm so sánh hiệu quả tính toán và hướng đến tối ưu hóa đa tiêu chí như: tránh nhiễu loạn, tiết kiệm nhiên liệu, hạn chế độ lệch tuyến.
- Mô hình định tuyến sẽ được nâng cấp để hỗ trợ tuyến bay dạng cong, giúp mô phỏng chính xác hơn quỹ đạo thực tế và loại bỏ các đoạn bay dư thừa. Đồng thời, hệ thống sẽ tích hợp khả năng đánh giá chi phí tuyến theo các yếu tố như chiều dài, lệch thời gian, và ảnh hưởng đến vùng kiểm soát lân cận.
- Phần mềm cần cải tiến giao diện người dùng với khả năng tương tác trực quan trên bản đồ số, hỗ trợ nhập – xuất dữ liệu linh hoạt, đồng thời hướng tới tích hợp với các hệ thống huấn luyện và phân tích chiến lược điều hành bay.

6.3 Kết luận chung

Đề tài “*Xây dựng công cụ hỗ trợ đề xuất đường bay thay thế tạm thời bằng thuật toán A* trong vùng thông báo bay Hồ Chí Minh*” là một bước tiếp cận ban đầu trong việc ứng dụng công nghệ vào lĩnh vực điều phối luồng không lưu. Mặc dù công cụ còn ở mức đơn giản và mang tính thử nghiệm, nhưng đã thể hiện được tiềm năng trong việc đề xuất lộ trình thay thế khi xảy ra các tình huống bất thường như thời tiết xấu hoặc tắc nghẽn vùng trời. Việc lựa chọn thuật toán A* giúp tối ưu hóa việc tìm đường ngắn nhất trong một mạng lưới cố định, từ đó tạo nền tảng để mở rộng nghiên cứu. Tuy nhiên, đề tài vẫn còn nhiều điểm hạn chế về dữ liệu thực tế, phạm vi ứng

dụng và khả năng tích hợp hệ thống. Trong tương lai, nếu tiếp tục được hoàn thiện, đề tài có thể đóng vai trò như một công cụ hỗ trợ ra quyết định trong điều hành bay, góp phần nâng cao hiệu quả quản lý không lưu trong khu vực.

TÀI LIỆU THAM KHẢO

- [1] A. Cook (2007). *European air traffic management: Principles, practice, and research*
- [2] O. Rodionova (2015). *Aircraft trajectory optimization in North Atlantic oceanic airspace*. Université Paul Sabatier - Toulouse III.
- [3] M.C.R.Murça (2018). *Data-Driven Modeling of Air Traffic Flows for Advanced Air Traffic Management*. MIT International Center for Air Transportation. Department of Aeronautics & Astronautics.
- [4] M. Sandoval (2023). *Optimization Algorithms for the Flight Planning Problem*
- [5] M.S. Nolan (2010). *Fundamentals of Air Traffic Control*. Cengage Learning, 5th edition
- [6] S.N. Rodriguez (2023). *Crossing Waypoint Optimization in Free Route Airspace*

- [7] N. Eurocontrol (2021). *European Route Network Improvement Plan (ERNIP) - Part 1*. Eurocontrol, 2.5 edition
- [8] Federal Aviation Administration. *National Playbook*. Traffic Management National, Center, and Terminal, được lấy về từ https://www.faa.gov/air_traffic/publications/atpubs/foa_html/chap18_section_22.html ngày 01 tháng 07 năm 2025
- [9] Federal Aviation Administration. *Playbook*, được lấy về từ <https://www.fly.faa.gov/PLAYBOOK/pbindex.html> ngày 01 tháng 07 năm 2025
- [10] H.M. de Jong (1974). *Optimal Track Selection and 3-dimensional flight planning*. Tech.rep. 1974.
- [11] P. Hagelauer et al (1998). *A soft dynamic programming approach for on-line aircraft 4D-trajectory optimization*. In: European Journal of Operational Research 107.1, pp. 87–95.
- [12] L.A. Moreno et al (2015). *Combining NLP and MILP in Vertical Flight Planning*. In: Operations Research Proceedings, Selected Papers of the International Conference of the German, Austrian and Swiss Operations Research Societies
- [13] P. Bonami et al (2013). *Multiphase Mixed-Integer Optimal Control Approach to Aircraft Trajectory Optimization*. In: Journal of Guidance, Control, and Dynamics 36.5, pp. 1267–1277
- [14] A.N. Knudsen et al (2016). *Vertical Optimization of Resource Dependent Flight Paths*. In: 22nd European Conference on Artificial Intelligence, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence.
- [15] K.S. Larsen et al (2018). *Heuristic Variants of A* Search for 3D Flight Planning*. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference.

- [16] Nhut Ngo et al (2022). *Developing a Decision Support Tool for Air Route Planning System*. Proceedings of 10th International Conference on Recent Advances in Civil Aviation, pp.113-124
- [17] H. V. Truong et al (2022). *Enhanced Teaching-Learning-based Optimization for 3D Path Planning of Multicopter UAVs*
- [18] S. Russell et (2010). *Artificial Intelligence: A modern approach.*
- [19] R. Devulapalli (2012). *An Efficient Algorithm for Commercial Aircraft Trajectory Optimization in the Air Traffic System*. University of Minnesota.
- [20] F. Garret D (2014). *Aircraft Route Optimization Using the A-Star Algorithm.*
- [21] R. Kumar (2024). *The A* Algorithm: A Complete Guide*, được lấy về từ <https://www.datacamp.com/tutorial/a-star-algorithm> ngày 01 tháng 07 năm 2025

CODE

CHƯƠNG 5 XÂY DỰNG ỨNG DỤNG HỖ TRỢ ĐỀ XUẤT ĐƯỜNG BAY THAY THẾ

5.1 Xây dựng giao diện ứng dụng

Nhóm đề tài sẽ thực hiện thiết kế một giao diện ứng dụng phần mềm nhằm hỗ trợ những người thực hiện công tác nghiên cứu lựa chọn nhanh các phương án đường bay thay thế trong tình huống khẩn cấp để có thể xuất bản ra *Playbook*. Giao diện ứng dụng này trực quan hóa bản đồ vùng trời FIR Hồ Chí Minh với các điểm trọng yếu và đường hàng không, đồng thời tích hợp các tính năng hỗ trợ như nhập các số liệu để hiển thị khu vực hạn chế, điều kiện thời tiết và gợi ý đường bay thay thế. Cách thức thiết lập và xây dựng giao diện được mô tả theo trình tự các bước được đề cập dưới đây.

Bước 1: Lựa chọn nền tảng xây dựng ứng dụng web và hệ quản trị cơ sở dữ liệu

```
from flask import Flask, request, jsonify
from pymongo import MongoClient
from flask_cors import CORS
from bson.json_util import dumps
from shapely.geometry import LineString, Polygon, Point
import requests
import heapq
import os
```

Đoạn mã được trình bày là một phần quan trọng trong việc thiết lập môi trường cho một ứng dụng web được xây dựng bằng Python, sử dụng Flask làm framework back-end (hiểu đơn giản là một khung để xây dựng web).

- + Đầu tiên, các thành phần cần thiết từ thư viện *Flask* được nhập vào, bao gồm lớp *Flask* để tạo ứng dụng, *request* để truy cập dữ liệu từ các yêu cầu *HTTP*, và *jsonify* để chuyển đổi dữ liệu *Python* thành định dạng *JSON* phục vụ cho việc trả về cho *client*.

- + Tiếp theo, *MongoClient* từ thư viện *pymongo* được sử dụng để kết nối và tương tác với cơ sở dữ liệu *MongoDB*, cho phép lưu trữ và truy xuất dữ liệu từ *MongoDB* một cách hiệu quả.
- + Để hỗ trợ việc xử lý các yêu cầu từ các nguồn khác nhau, thư viện *flask_cors* được nhập vào, cho phép cấu hình *Cross-Origin Resource Sharing (CORS)* cho ứng dụng.

Tóm lại, việc thiết lập môi trường cho một ứng dụng web sử dụng Flask không chỉ bao gồm việc nhập các thư viện cần thiết mà còn đảm bảo rằng ứng dụng có khả năng tương tác hiệu quả với cơ sở dữ liệu *MongoDB* và hỗ trợ việc xử lý các yêu cầu từ nhiều nguồn khác nhau thông qua CORS. Những thành phần này kết hợp lại tạo nên một nền tảng vững chắc cho việc phát triển ứng dụng web, giúp tối ưu hóa hiệu suất và khả năng mở rộng trong tương lai.

Bước 2: Cài đặt Backend API

```
# Khởi tạo Flask app
app = Flask(__name__)
CORS(app)
connectAPI = 'https://algorithma-84og.onrender.com'
```

Gọi API Flask đã được deploy trên nền tảng [Render.com](#) nhằm những mục đích sau:

Đầu tiên, khi deploy backend lên Render thông qua đường link <https://algorithma-84og.onrender.com>, từ đó có thể lấy tất cả các dữ liệu có trong API này, ví dụ như Hình... là tất cả các dữ liệu nằm trong API được nhóm để tài sử dụng.

```
["center": [10.06222222222223, 104.58944444444444], "name": "Zone H", "radius": 30000, "type": "circle"}, {"center": [11.916944444444443, 108.10805555555555], "name": "hanh", "radius": 40000, "type": "circle"}]
```

Thứ hai, tạo hệ thống hoàn chỉnh từ Frontend sang Backend. Có nghĩa là Phần frontend chỉ là giao diện – nó như: hiển thị bản đồ, cho người dùng chọn điểm đầu/cuối, hiển thị đường bay/vùng cấm, v.v. Tuy nhiên nó không thể tự tính toán được đường bay, không truy cập được cơ sở dữ liệu. Thay vào đó Backend mới là nơi xử lý dữ liệu như: truy xuất dữ liệu waypoint, vùng cấm từ MongoDB, chạy thuật toán A*, Yen để tính toán đường bay thay thế và trả kết quả về cho Frontend. Vậy kết nối Frontend - Backend có nghĩa là Frontend call API đến Flask Backend sau đó Backend xử lý dữ liệu để trả lại dữ liệu, sau đó Frontend lấy dữ liệu đó để vẽ đường bay, vùng cấm,..

Thứ ba, cho phép giao diện người dùng hoạt động từ xa có nghĩa là khi tạo giao diện người dùng thì gọi API để cho phép tất cả mọi người có thể truy cập vào được trang web.

Bước 3: Thiết lập hệ quản trị dữ liệu

Nhóm đề tài sử dụng **MongoDB** làm phần mềm mã nguồn mở để quản trị cơ sở dữ liệu. Nói một cách dễ hiểu, khi người dùng thao tác trên ứng dụng web, họ sẽ cần

nhập hoặc truy xuất dữ liệu, do đó cần một hệ thống lưu trữ dữ liệu ổn định và linh hoạt. Những bộ dữ liệu này sẽ hỗ trợ quá trình phân tích, gợi ý hoặc trực quan hóa các phương án đường bay thay thế trong tình huống khẩn cấp.

MongoDB được lựa chọn vì sử dụng định dạng JSON linh hoạt, phù hợp với các loại dữ liệu không cố định. Cụ thể, mỗi đường bay có thể đi qua số lượng điểm khác nhau và chứa thông tin khác nhau, nên không cần ràng buộc cấu trúc dữ liệu chặt chẽ. Đây là đặc điểm rất phù hợp cho các ứng dụng bản đồ bay, nơi dữ liệu thường có cấu trúc đa dạng và thay đổi linh hoạt. Hình.... là đoạn mã code thiết lập bộ dữ liệu MongoDB.

```
# Kết nối MongoDB
MONGO_URI = os.getenv(
    "MONGO_URI",
    "mongodb+srv://admin2:123123a@cluster0.nyw26.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0"
)
client = MongoClient(MONGO_URI)
db = client["Flight_A"]
road_collection = db["Road"]
point_collection = db["data_point"]
No_Flight_collection = db["No_Flight"]
distance_collection = db["distance"]
flight_path_new_collection = db["flight_path_new"]
flight_path_collection = db["flight_path"]
```

Đầu tiên, hệ thống sẽ tạo một đối tượng, cụ thể ở đây là một chuyến bay, kết nối đến cơ sở dữ liệu **MongoDB** nhằm thiết lập liên lạc giữa ứng dụng và kho dữ liệu. Sau khi kết nối thành công, hệ thống sẽ truy cập vào cơ sở dữ liệu có tên là *Flight_A* chứa tất cả các dữ liệu cần thiết để xử lý cho một chuyến bay. Bộ dữ liệu mà nhóm đề tài thiết lập bao gồm: tuyến đường bay, các điểm trọng yếu, vùng cấm bay, khoảng cách giữa các điểm, đường bay mới được đề xuất. Khi ứng dụng thực hiện nhiệm vụ tìm đường bay thay thế khi xuất hiện một chướng ngại vật nào đó, hệ thống sẽ thực hiện truy vấn các dữ liệu đó để từ đó tính toán và đề xuất đường bay.

5.2 Nhập dữ liệu trên giao diện ứng dụng

5.2.1. Thêm đường bay và điểm bay

- Thêm dữ liệu đường bay vào cơ sở dữ liệu (POST /data-road)

MongoDB được sử dụng làm hệ quản trị cơ sở dữ liệu, lưu trữ thông tin về các tuyến đường bay nhằm hỗ trợ backend trong việc truy xuất và hiển thị dữ liệu trên giao diện. Để đáp ứng nhu cầu người dùng muốn chủ động nhập và khai báo thêm các tuyến bay dự kiến, hệ thống triển khai một API endpoint tại địa chỉ `/data-road` với phương thức POST. Endpoint này tiếp nhận dữ liệu gồm tên tuyến đường chính và danh sách các đoạn đường con. Hệ thống sẽ kiểm tra tính đầy đủ của thông tin đầu vào; nếu thiếu, yêu cầu sẽ bị từ chối và trả về thông báo lỗi. Ngược lại, nếu hợp lệ, dữ liệu sẽ được lưu vào MongoDB và hệ thống phản hồi với thông báo thành công và trả về mã ID của chuyến bay với bản ghi mới. Đây là bước quan trọng để xây dựng bản đồ đường bay động dựa trên dữ liệu do người dùng cung cấp.

```
# Route POST để thêm dữ liệu road
@app.route("/data-road", methods=["POST"])
def add_data_road():
    data = request.json
    if not data or "ten_duong_chinh" not in data or "cac_ten_duong_con" not in data:
        return jsonify({"error": "Missing required fields"}), 400
    result = road_collection.insert_one(data)
    return jsonify({
        "message": "Road added successfully",
        "id": str(result.inserted_id)
    }), 201
```

- Truy xuất lại dữ liệu đường kèm các lô điểm cùng tọa độ (GET /data-road).

Sau khi đường bay dự kiến vừa báo nhập thành công, đoạn mã trong Hình..... trên định nghĩa một route GET tại địa chỉ `/data-road`, được sử dụng để truy xuất danh sách các tuyến đường chính cùng với thông tin chi tiết của từng điểm nằm trên đó. Khi có yêu cầu từ phía người dùng hoặc frontend, hàm `get_data_road_with_coordinates()` sẽ được gọi.

Trước tiên, hàm lấy toàn bộ dữ liệu các tuyến đường chính từ `road_collection`, loại bỏ trường `_id` để dễ hiển thị. Với mỗi tuyến đường, hệ thống đọc danh sách các tên các lô điểm và lần lượt tra cứu thông tin chi tiết của từng điểm này trong `point_collection`. Các thông tin hiển thị bao gồm: tên đường, vĩ độ, kinh độ và chiều bay. Nếu tìm thấy điểm tương ứng, dữ liệu sẽ được gắn vào kết quả trả về; nếu không, các giá trị sẽ để là `None`.

Cuối cùng, danh sách đã xử lý sẽ được trả về dưới dạng JSON thông qua lệnh `jsonify(result)` và báo hiệu thành công. Mục đích chính của đoạn mã là cung cấp dữ liệu đầy đủ về tuyến đường và các điểm trọng yếu để phục vụ việc hiển thị chính xác trên giao diện người dùng.

```
# Route GET kết hợp tọa độ và chiều
@app.route("/data-road", methods=[ "GET"])
def get_data_road_with_coordinates():
    roads = list(road_collection.find({}, {"_id": 0}))
    result = []

    for road in roads:
        duong_chinh = road["ten_duong_chinh"]
        ten_duong_con = road["cac_ten_duong_con"]

        enriched_con = []
        for name in ten_duong_con:
            point = point_collection.find_one(
                {"ten_duong": name},
                {"_id": 0, "vi_do": 1, "kinh_do": 1, "chieu": 1}
            )
            if point:
                enriched_con.append({
                    "ten_duong": name,
                    "vi_do": point.get("vi_do"),
                    "kinh_do": point.get("kinh_do"),
                    "chieu": point.get("chieu")
                })
            else:
                enriched_con.append({
                    "ten_duong": name,
                    "vi_do": None,
                    "kinh_do": None,
                    "chieu": None
                })

        result.append({
            "ten_duong_chinh": duong_chinh,
            "cac_ten_duong_con": enriched_con
        })
    return jsonify(result), 200
```

5.12. Thêm vùng cấm bay

- Thêm dữ liệu vùng cấm bay

Đoạn mã trong Hình định nghĩa một API POST tại địa chỉ `/no_flight`, cho phép người dùng thêm dữ liệu về vùng cấm bay vào hệ thống. API này tiếp nhận dữ liệu ở định dạng JSON và phân loại vùng cấm bay theo hai dạng hình học: vùng đa giác (polygon) và vùng hình tròn (circle). Mỗi loại vùng đều yêu cầu các thông tin đặc trưng để có thể lưu trữ và xử lý chính xác.

```
# Route POST để thêm dữ liệu vùng cấm bay
@app.route("/no_flight", methods=[ "POST"])
def add_no_flight():
    data = request.json
    if not data or "name" not in data:
        return jsonify({"error": "Thiếu tên vùng cấm bay"}), 400

    name = data["name"]
```

Trước tiên, hệ thống yêu cầu người dùng nhập tên của vùng cấm bay và kiểm tra xem dữ liệu gửi lên có trường "name" hay không. Nếu thiếu tên vùng cấm bay, hệ thống sẽ trả về thông báo lỗi. Sau đó, API tiếp tục xác định hình vùng cấm bay dựa trên các trường trong dữ liệu.

```
# Nếu là vùng đa giác (polygon)
if "coordinates" in data:
    coordinates = data["coordinates"]
    if not isinstance(coordinates, list) or not all(
        isinstance(coord, list) and len(coord) == 2 and all(isinstance(x, (int, float)) for x in coord)
        for coord in coordinates
    ):
        return jsonify({"error": "Tọa độ phải là danh sách các cặp [lat, lng]"}), 400

    no_flight_data = {
        "name": name,
        "type": "polygon",
        "coordinates": coordinates
    }
    result = No_Flight_collection.insert_one(no_flight_data)
    return jsonify({
        "message": "Đã thêm vùng cấm bay đa giác",
        "id": str(result.inserted_id)
    }), 201
```

Nếu dữ liệu chứa trường "coordinates", hệ thống hiểu đây là vùng cấm bay dạng đa giác. Tọa độ phải là một danh sách các cặp [*vĩ độ*, *kinh độ*] hợp lệ, với mỗi giá trị là kiểu số. Nếu dữ liệu hợp lệ, hệ thống sẽ gán kiểu "polygon" cho vùng này và lưu thông tin vào cơ sở dữ liệu MongoDB. Kết quả trả về bao gồm thông báo thành công và ID của bản ghi vừa được thêm.

```
# Nếu là vùng đa giác (polygon)
if "coordinates" in data:
    coordinates = data["coordinates"]
    if not isinstance(coordinates, list) or not all(
        isinstance(coord, list) and len(coord) == 2 and all(isinstance(x, (int, float)) for x in coord)
        for coord in coordinates
    ):
        return jsonify({"error": "Tọa độ phải là danh sách các cặp [lat, lng]"}), 400

    no_flight_data = {
        "name": name,
        "type": "polygon",
        "coordinates": coordinates
    }
    result = No_Flight_collection.insert_one(no_flight_data)
    return jsonify({
        "message": "Đã thêm vùng cấm bay đa giác",
        "id": str(result.inserted_id)
    }), 201
```

Trong trường hợp dữ liệu chứa trường "center" và "radius", hệ thống sẽ xử lý như một vùng cấm bay hình tròn. Tọa độ tâm phải là một cặp [*lat*, *lng*] và bán kính là một số dương. Sau khi kiểm tra hợp lệ, dữ liệu sẽ được gán kiểu "circle" và lưu vào cơ sở dữ liệu. Tương tự, kết quả trả về gồm thông báo và ID.

```
else:
    return jsonify({"error": "Thiếu dữ liệu vùng cấm bay"}), 400
```

Nếu không đáp ứng được bất kỳ định dạng nào ở trên, hệ thống sẽ trả lỗi báo thiếu dữ liệu cần thiết để xác định vùng cấm bay. Đoạn mã này đảm bảo rằng mọi dữ liệu được nhập vào đều có cấu trúc rõ ràng, đầy đủ và hợp lệ, nhằm hỗ trợ tốt cho việc quản lý không phận trong ứng dụng.

- Truy xuất lại dữ liệu vùng cấm bay

Đoạn mã trên định nghĩa một API endpoint với phương thức GET tại địa chỉ `/data-no-flight`, cho phép truy xuất dữ liệu các vùng cấm bay từ cơ sở dữ liệu MongoDB. Khi có yêu cầu được gửi đến, hệ thống sẽ truy cập vào collection `No_Flight` và trích xuất các trường thông tin cần thiết như tên vùng (name), loại vùng (type), tọa độ ranh giới (coordinates), tọa độ (center) và bán kính (radius). Kết quả vùng cấm được trả về dưới dạng JSON giúp frontend dễ dàng sử dụng để hiển thị các vùng cấm bay trên giao diện bản đồ.

```
# API GET DATA CẤM BAY
@app.route("/data-no-flight", methods=["GET"])
def get_data_no_flight():
    no_flight_collection = db["No_Flight"]
    no_flight_zones = list(no_flight_collection.find({}, {
        "_id": 0,
        "name": 1,
        "type": 1,
        "coordinates": 1,
        "center": 1,
        "radius": 1
    }))
    return jsonify(no_flight_zones), 200
```

5.1.3. Thêm đường bay

Đoạn code này chính là phần xử lý chức năng thêm đường bay cho giao diện "Quản lý các chuyến bay". Khi người dùng trên giao diện nhập tên chuyến bay và danh sách các điểm waypoint, rồi nhấn nút để thêm mới, hệ thống sẽ gửi yêu cầu POST đến địa chỉ `/flight_path`, kèm theo dữ liệu JSON chứa thông tin đó. Đoạn code sẽ kiểm tra dữ liệu hợp lệ và nếu hợp lệ, nó sẽ lưu thông tin đường bay mới vào cơ sở dữ liệu. Nhờ vậy, sau khi thêm thành công, danh sách các đường bay trên giao diện sẽ được cập nhật, hiển thị chuyến bay vừa tạo cùng các điểm trong hành trình. Đây là cầu nối giữa giao diện người dùng và phần backend xử lý lưu trữ dữ liệu.

```

@app.route("/flight_path", methods=["POST"])
def add_flight_path():
    data = request.json
    if not data or "name" not in data or "waypoints" not in data:
        return jsonify({"error": "Thiếu 'name' hoặc 'waypoints'"}), 400

    name = data["name"]
    waypoints = data["waypoints"]

    if not isinstance(waypoints, list) or not all(isinstance(p, str) for p in waypoints):
        return jsonify({"error": "'waypoints' phải là danh sách các chuỗi tên waypoint"}), 400

    flight_path_data = {
        "name": name,
        "waypoints": waypoints
    }

    result = flight_path_collection.insert_one(flight_path_data)
    return jsonify({
        "message": "Đã thêm đường bay",
        "id": str(result.inserted_id)
    }), 201

```

5.2 Lấy tất cả đường bay

Đoạn code dưới Hình... mô tả phương thức **GET tại địa chỉ /flight_path/all** nhằm **lấy toàn bộ danh sách các đường bay** đang được lưu trữ trong cơ sở dữ liệu `flight_path_collection`. Khi người dùng gửi yêu cầu muốn xem chi tiết chuyến này, hệ thống sẽ truy vấn toàn bộ tài liệu trong bộ sưu tập, sau đó trả về danh sách các đường bay (tên đường bay và danh sách các lô điểm) dưới dạng JSON. Mục đích của chức năng này là cầu nối giữa giao diện người dùng và phần backend xử lý lưu trữ dữ liệu.

```

@app.route("/flight_path/all", methods=["GET"])
def get_all_flight_paths():
    flights = list(flight_path_collection.find({}, {"_id": 0}))
    return jsonify(flights), 200

```

5.6. Lấy khoảng cách các đoạn bay

Đoạn code trên được thiết lập nhằm mục đích tạo một API để lấy dữ liệu khoảng cách giữa các đoạn bay từ cơ sở dữ liệu. Khi người dùng gửi yêu cầu GET đến đường dẫn `/data-distance`, hệ thống sẽ trả về danh sách các đoạn bay cùng với khoảng cách tương ứng mục đích của việc thiết lập này là để cung cấp thông tin khoảng cách phục vụ cho chức năng tính toán độ dài của đường bay nhằm đưa ra những đường bay có lựa chọn tối ưu.

```
# API GET DATA KHOẢNG CÁCH
@app.route("/data-distance", methods=["GET"])
def get_data_distance():
    distances = list(distance_collection.find({}, {"_id": 0, "from": 1, "to": 1, "distance_nm": 1}))
    return jsonify(distances), 200
```

5.2.1. Nhận và kiểm tra dữ liệu đầu vào

Phần này dùng để **nhận dữ liệu JSON** từ yêu cầu POST gửi tới endpoint /flight_path/check. Dữ liệu kỳ vọng có trường "waypoints" (danh sách các tên điểm đường bay). Nếu danh sách này có ít hơn 2 điểm thì trả về lỗi kèm thông báo "Cần ít nhất 2 điểm" vì một đường bay hợp lệ phải có ít nhất một đoạn (2 điểm).

```
@app.route("/flight_path/check", methods=[ "POST"])
def check_flight_pathViolation():
    data = request.json
    waypoints = data.get("waypoints", [])
    if len(waypoints) < 2:
        return jsonify({"error": "Cần ít nhất 2 điểm"}), 400
```

5.2.2. Tải dữ liệu tọa độ các điểm

Ở bước này, đoạn code có nhiệm vụ tải toàn bộ dữ liệu các điểm thuộc đường bay từ cơ sở dữ liệu, cụ thể là từ bảng đang lưu thông tin các điểm định tuyến trong hệ thống quản lý chuyến bay. Mỗi điểm sẽ có các thuộc tính như: tên điểm, kinh độ và vĩ độ.

Tuy nhiên, chỉ những điểm nào có đầy đủ cả kinh độ và vĩ độ mới được sử dụng, nhằm đảm bảo đủ thông tin tọa độ để xử lý tiếp theo. Các điểm hợp lệ sẽ được lưu vào một bảng ánh xạ dưới dạng → Mục đích của bước này là chuẩn bị dữ liệu đầu vào cho các thuật toán A* để tìm đường đi ngắn nhất giữa các điểm, và phục vụ cho việc vẽ bản đồ đường bay sau này. Việc có sẵn tọa độ từng điểm là điều kiện bắt buộc để vẽ các đường bay lên bản đồ.

```
# Load points
point_map = {
    p["ten_duong"]: (p["kinh_do"], p["vi_do"])
    for p in point_collection.find({}, {"_id": 0, "ten_duong": 1, "vi_do": 1, "kinh_do": 1})
}
```

Đoạn mã thực hiện việc tải toàn bộ dữ liệu các điểm đường bay từ cơ sở dữ liệu point_collection. Mỗi điểm được lọc để đảm bảo có đầy đủ cả kinh độ và vĩ độ thông qua điều kiện kinh độ và vĩ độ. Các điểm hợp lệ sau đó được lưu vào một từ điển có cấu trúc point_map. Mục đích chính của bước này là chuẩn bị dữ liệu đầu vào để phục

vụ cho việc dựng các đoạn đường bay trên bản đồ, đồng thời hỗ trợ các thao tác kiểm tra không gian như phát hiện vi phạm vùng cấm bay.

5.2.3. Tải vùng cấm bay

```
# Load vùng cấm bay
zones = load_no_flight_zones()
```

Hàm `load_no_flight_zones()` được sử dụng để truy xuất thông tin về các vùng cấm bay, phục vụ cho quá trình kiểm tra xâm phạm không phận. Các vùng này thường được biểu diễn dưới dạng đa giác (polygon), giúp hệ thống dễ dàng thực hiện các phép toán hình học nhằm phát hiện sự giao cắt hoặc vi phạm trong quá trình xử lý đường bay. Việc gọi hàm này cho phép hệ thống chuẩn bị sẵn các vùng hạn chế cần thiết trước khi thực hiện các kiểm tra liên quan.

5.3. Xử lý logic và kiểm tra:

5.3.1 Tiền xử lí vùng cấm bay:

```
data_no_flight = requests.get(f"{connectAPI2}/data-no-flight").json()
no_flight_zones = [
    {
        "name": zone["name"],
        "polygon": Polygon([(coord[1], coord[0]) for coord in zone["coordinates"]])
    }
    for zone in data_no_flight
]
```

Đoạn mã trên thực hiện bước tiền xử lý dữ liệu vùng cấm bay bằng cách truy vấn API /data-no-flight để lấy danh sách tất cả các vùng cấm hiện có. Mỗi vùng cấm được mô tả bằng tập hợp các tọa độ, thường dưới dạng danh sách các cặp vĩ độ và kinh độ. Để phục vụ cho việc kiểm tra xem một đoạn đường bay có vi phạm vùng cấm hay không, các tọa độ này sẽ được chuyển đổi thành các đối tượng hình học Polygon thông qua thư viện Shapely. Việc chuyển đổi này không nhằm mục đích trực quan hóa hay hiển thị đồ họa lên bản đồ, mà để phục vụ cho các phép toán hình học — chẳng hạn như xác định đoạn đường có **cắt**, **nằm trong**, hay **giao nhau** với vùng cấm hay không. Nói cách khác, đây là bước xây dựng mô hình không gian (spatial model) để thực hiện các kiểm tra logic không gian chứ không phải hiển thị trực tiếp lên giao diện người dùng. Đây là bước quan trọng giúp hệ thống xử lý không gian hiệu quả và chính xác trong các thuật toán như A* hoặc Yen's.

5.3.2 Kiểm tra đường bay nhập thủ công có vi phạm không (POST /flight_path/check)

```

# Load points
point_map = {
    p["ten_duong"]: (p["kinh_do"], p["vi_do"])
    for p in point_collection.find({}, {"_id": 0, "ten_duong": 1, "vi_do": 1, "kinh_do": 1})
    if p.get("vi_do") and p.get("kinh_do")
}

# Load vùng cấm bay
zones = load_no_flight_zones()

# Tạo các đoạn và kiểm tra
violations = []
for i in range(len(waypoints) - 1):
    p1, p2 = waypoints[i], waypoints[i+1]
    if p1 not in point_map or p2 not in point_map:
        continue
    segment = LineString([point_map[p1], point_map[p2]])
    if is_blocked_by_no_flight(segment, zones):
        violations.append({"from": p1, "to": p2})

return jsonify({"violations": violations}), 200

```

Mục đích: Việc kiểm tra xem đường bay do người dùng nhập vào có đi qua vùng cấm bay hay không giúp xác định tính khả thi và mức độ an toàn của lộ trình hiện tại. Nếu phát hiện có vi phạm vùng cấm, hệ thống sẽ thông báo để người dùng biết và có thể lựa chọn tính năng tìm đường bay thay thế sử dụng thuật toán A*

Đầu tiên, tập hợp các điểm tọa độ tương ứng với các waypoint sẽ được lấy từ cơ sở dữ liệu point_collection, được lưu vào point_map dưới dạng từ điển với key là tên đường (ten_duong) và value là một tuple tọa độ (kinh_do, vi_do). Tiếp theo, danh sách vùng cấm bay sẽ được tải thông qua hàm load_no_flight_zones, trả về các đối tượng hình học (chẳng hạn Polygon hoặc Circle) đã được dựng sẵn từ trước đó.

Sau khi đã thu thập đầy đủ dữ liệu, chương trình sẽ khởi tạo một danh sách rỗng có tên violations để lưu lại các đoạn đường bay vi phạm. Sau đó, nó duyệt qua toàn bộ danh sách waypoints do người dùng cung cấp, tạo ra các cặp điểm liên tiếp (ví dụ: từ điểm 1 đến điểm 2, điểm 2 đến điểm 3,...). Với mỗi cặp điểm p1 và p2, chương trình kiểm tra xem cả hai điểm có tồn tại trong bản đồ tọa độ (point_map) hay không. Nếu một trong hai không có, đoạn đó sẽ bị bỏ qua.

Nếu cả hai điểm đều tồn tại, đoạn mã sử dụng thư viện shapely để tạo một đoạn thẳng (LineString) nối từ p1 đến p2. Đoạn bay này sau đó được truyền vào hàm is_blocked_by_no_flight, nơi sẽ xác định xem đoạn bay có giao cắt hoặc nằm trong bất kỳ vùng cấm bay nào không. Nếu có, chương trình sẽ ghi nhận đoạn bay vi phạm đó vào danh sách violations bằng cách lưu lại tên điểm bắt đầu và kết thúc. Sau khi duyệt qua toàn bộ các đoạn, API sẽ trả về danh sách các đoạn vi phạm dưới dạng JSON để người dùng biết chính xác phần nào của đường bay đã đi vào khu vực cấm. Đây là phần xử lý quan trọng trong API POST /flight_path/check, giúp hỗ trợ người dùng kiểm tra thủ công trước khi lưu đường bay hoặc thực hiện các bước tiếp theo.

5.3.3. Xem toàn bộ đoạn đường nào đi qua vùng cấm (GET /get-point-no-flight)

Mục đích: Việc phân tích các **đường bay chính đã được lưu trữ** nhằm kiểm tra xem chúng có bị chèn lấn hoặc cắt qua các vùng cấm hiện có hay không là điều cần thiết. Điều này giúp đảm bảo rằng **thuật toán A*** sẽ chỉ đề xuất những đường bay thay thế dựa trên các tuyến có sẵn được xác nhận là hợp lệ và không vi phạm vùng cấm bay.

```
@app.route("/get-point-no-flight", methods=["GET"])
def get_no_flight_zones():
    data_points = requests.get(f"{connectAPI2}/data-road").json()
    data_no_flight = requests.get(f"{connectAPI2}/data-no-flight").json()
    no_flight_zones = [
        {
            "name": zone["name"],
            "polygon": Polygon([(coord[1], coord[0]) for coord in zone["coordinates"]])
        }
        for zone in data_no_flight
    ]
```

Đoạn mã trên định nghĩa một API GET tại đường dẫn /get-point-no-flight, có chức năng kiểm tra xem các đoạn đường bay có cắt qua vùng cấm bay hay không. Khi được gọi, hàm sẽ gửi yêu cầu tới hai API khác để lấy dữ liệu: một API cung cấp các điểm tọa độ của các đường bay (/data-road), và API còn lại cung cấp danh sách các vùng cấm bay (/data-no-flight) - đã được.....

```
violations = []

for road in data_points:
    ten_duong_chinh = road["ten_duong_chinh"]
    points = road["cac_ten_duong_con"]

    valid_points = [
        (p["kinh_do"], p["vi_do"]) for p in points
        if p["kinh_do"] is not None and p["vi_do"] is not None
    ]

    for i in range(len(valid_points) - 1):
        p1 = valid_points[i]
        p2 = valid_points[i + 1]
        segment = LineString([p1, p2])

        for zone in no_flight_zones:
            if segment.crosses(zone["polygon"]) or segment.within(zone["polygon"]):
                violations.append({
                    "ten_duong_chinh": ten_duong_chinh,
                    "zone": zone["name"],
                    "from": {"kinh_do": p1[0], "vi_do": p1[1]},
                    "to": {"kinh_do": p2[0], "vi_do": p2[1]}
                })

return jsonify(violations)
```

Sau khi đã có được các dữ liệu đường chính và dữ liệu các vùng cấm bay, chúng sẽ xem xét đánh giá xem liệu đường chính này có bay qua vùng cấm hay không. Trước tiên, chương trình khởi tạo một danh sách rỗng violations để lưu trữ các đoạn bị vi phạm. Sau đó, nó duyệt qua từng đường chính trong dữ liệu các tuyến đường bay (data_points) và lấy ra danh sách các điểm con tương ứng. Với mỗi điểm con, chương trình kiểm tra xem các tọa độ kinh độ và vĩ độ có hợp lệ không, sau đó tạo thành một danh sách các điểm hợp lệ (valid_points) dưới dạng cặp tọa độ (kinh độ, vĩ độ). Tiếp theo, chương trình xét từng đoạn thẳng nối hai điểm liên tiếp và chuyển chúng thành đối tượng LineString – tức là một đoạn đường bay. Với mỗi đoạn như vậy, chương trình tiếp tục kiểm tra xem đoạn đó có cắt qua (crosses) hoặc nằm hoàn toàn bên trong (within) bất kỳ vùng cấm bay nào hay không. Nếu có, nó thêm một mục vào danh sách violations, trong đó ghi rõ tên đường chính, tên vùng cấm bị vi phạm, cũng như tọa độ điểm đầu và điểm cuối của đoạn vi phạm. Cuối cùng, toàn bộ danh sách các đoạn đường bay vi phạm vùng cấm được trả về dưới dạng JSON để phục vụ cho việc hiển thị trực quan hoặc xử lý tiếp theo trên giao diện người dùng.

Trả kết quả

```
return jsonify({"violations": violations}), 200
```

Kết thúc hàm, trả về danh sách các đoạn đường bay vi phạm dưới dạng JSON với giao diện trực quan cho người sử dụng

5.4. Tìm đường đi thay thế (bằng thuật toán A*)

5.4.1. haversine(p1, p2)

```
def haversine(p1, p2):
    from math import radians, cos, sin, asin, sqrt
    lat1, lon1 = p1
    lat2, lon2 = p2
    R = 6371 # km
    dlat = radians(lat2 - lat1)
    dlon = radians(lon2 - lon1)
    a = sin(dlat / 2)**2 + cos(radians(lat1)) * cos(radians(lat2)) * sin(dlon / 2)**2
    c = 2 * asin(sqrt(a))
    return R * c * 0.539957
```

Đoạn mã trên định nghĩa hàm haversine(p1, p2) — một hàm dùng để tính khoảng cách thực tế (còn gọi là đường chim bay) giữa hai điểm địa lý được xác định bởi cặp tọa độ vĩ độ và kinh độ. Hàm này sử dụng công thức Haversine, một công thức phổ biến trong địa lý học để đo khoảng cách giữa hai điểm trên bề mặt trái đất, có tính đến độ cong của địa cầu. Kết quả trả về là đơn vị hải lý (nautical miles) — một chuẩn đo thường dùng trong hàng không và hàng hải để thể hiện khoảng cách giữa các

vị trí địa lý. Việc sử dụng đơn vị này giúp đảm bảo tính chính xác và tương thích với các tiêu chuẩn định tuyến hàng không quốc tế.

5.4.2. load_no_flight_zones()

```
def load_no_flight_zones():
    zones = []
    for z in No_Flight_collection.find({}, {"_id": 0}):
        z_type = z.get("type")

        if z_type == "polygon" and z.get("coordinates"):
            try:
                coords = [(pt[1], pt[0]) for pt in z["coordinates"]]
                polygon = Polygon(coords)
                zones.append(polygon)
            except Exception as e:
                print(f"[!] Lỗi polygon {z.get('name')}: {e}")

        elif z_type == "circle" and z.get("center") and z.get("radius"):
            try:
                lat, lng = z["center"]
                lat = float(lat)
                lng = float(lng)
                center_point = Point(lng, lat)

                radius_m = float(z["radius"]) # radius theo mét
                buffer_deg = radius_m / 111320 # mét → độ
                circle_polygon = center_point.buffer(buffer_deg)

                zones.append(circle_polygon)
            except Exception as e:
                print(f"[!] Lỗi circle {z.get('name')}: {e}")


```

Đoạn mã trên định nghĩa hàm load_no_flight_zones() với mục tiêu tải và chuẩn hóa dữ liệu các vùng cấm bay từ cơ sở dữ liệu MongoDB, để phục vụ việc kiểm tra va chạm (intersection) giữa đường bay và vùng cấm.

- Truy vấn dữ liệu từ MongoDB:** Hàm lấy toàn bộ tài liệu trong collection No_Flight, loại bỏ _id vì không cần thiết trong xử lý hình học.
- Phân loại và xử lý vùng cấm:**
Nếu vùng cấm có kiểu polygon (đa giác) và có đủ coordinates, các tọa độ này (dưới dạng vĩ độ, kinh độ) sẽ được chuyển đổi thành các cặp (kinh độ, vĩ độ) để tạo một đối tượng Polygon từ thư viện Shapely. Đây là định dạng chuẩn (x, y) cho xử lý hình học. Nếu vùng cấm có kiểu circle (hình tròn), với center (tâm) và radius (bán kính) thì Polygon (tuy nhiên với hình tròn sẽ gọi là buffered point) cũng sẽ tạo ra một đa giác hình tròn đại diện cho vùng cấm.

```

    else:
        print(f"[!] Bỏ qua vùng không hợp lệ: {z.get('name')}")

print(f"[+] Đã load {len(zones)} vùng cấm bay")
return zones

```

3. **Xử lý ngoại lệ:** Nếu dữ liệu không hợp lệ hoặc thiếu thông tin cần thiết, hệ thống sẽ ghi log lỗi nhưng không dừng toàn bộ quá trình.
4. **Kết quả:** Trả về danh sách các vùng cấm đã được biểu diễn dưới dạng hình học (Polygon) để phục vụ việc kiểm tra va chạm với đường bay trong các thuật toán như A* hoặc Yen's. Tất cả vùng hợp lệ sẽ được đưa vào danh sách zones, và sau đó được dùng để kiểm tra cắt qua vùng cấm.

5.4.3. is_blocked_by_no_flight(segment, zones)

Việc nhóm đề tài lựa chọn không sử dụng các đường bay có sẵn mà thay vào đó cho phép người dùng nhập tùy ý điểm bắt đầu và điểm kết thúc khiền cho quá trình tìm đường bay trở nên linh hoạt hơn vậy nên điều này đòi hỏi sự kiểm tra vùng cấm phải được thực hiện trực tiếp trong thuật toán tìm đường. Đó chính là lí do vì sao trong khi API GET /get-point-no-flight đã thực hiện kiểm tra các tuyến bay cố định xem có vi phạm vùng cấm hay không, thì thuật toán A* vẫn cần tự đánh giá lại các đoạn đường trong quá trình tìm kiếm.

Mục đích là đảm bảo rằng với bất kỳ tổ hợp điểm bay nào do người dùng chỉ định, hệ thống đều sẽ phát hiện và loại bỏ các phương án có khả năng đi qua vùng cấm. Nhờ đó, những đường bay được đề xuất cuối cùng luôn đảm bảo an toàn và không xâm phạm không phận bị giới hạn. Đây là bước xử lý phù hợp và cần thiết trong trường hợp ứng dụng cho phép bay theo mô hình "điểm nối điểm" thay vì tuân theo các tuyến định trước.

```

def is_blocked_by_no_flight(segment, zones):
    for zone in zones:
        if segment.intersects(zone):
            print(f"[X] Đoạn bị chặn bởi zone")
            return True
    return False

```

Trong quá trình kiểm tra, hàm sẽ duyệt qua từng vùng trong danh sách zones - mỗi vùng là một đối tượng hình học (Polygon hoặc buffered Point) mô phỏng vùng cấm bay. Nếu đoạn đường bay segment giao nhau với bất kỳ vùng cấm nào (thông qua hàm intersects của Shapely), hàm sẽ in ra cảnh báo và trả về True, tức là đoạn đường đó bị chặn bởi vùng cấm. Nếu không có vùng cấm nào bị vi phạm, hàm sẽ trả về

False. Đây là một phần quan trọng trong logic đảm bảo an toàn bay, giúp loại bỏ các phương án đường bay không hợp lệ trong các thuật toán như A*.

5.4.4. a_star(start, goal, points_dict, graph_edges, no_flight_polygons)

```
return False
def a_star(start, goal, points_dict, graph_edges, no_flight_polygons):
    open_set = [(0, [start])]
    paths = []

    while open_set:
        total_cost, path = heapq.heappop(open_set)
        current = path[-1]

        if current == goal:
            return path

        for neighbor, dist in graph_edges.get(current, []):
            if neighbor in path:
                continue
            if neighbor not in points_dict or current not in points_dict:
                continue

            segment = LineString([
                (points_dict[current][1], points_dict[current][0]), # (lng, lat)
                (points_dict[neighbor][1], points_dict[neighbor][0])
            ])

            if is_blocked_by_no_flight(segment, no_flight_polygons):
                continue
            # violate = False
            # for zone in no_flight_polygons:
            #     if isinstance(zone, Polygon):
            #         if segment.intersects(zone):
            #             violate = True

            # if violate:
            #     continue

            new_path = path + [neighbor]
            cost = sum(
                graph_edges[new_path[i]][j][1]
                for i in range(len(new_path) - 1)
                for j in range(len(graph_edges[new_path[i]])))
                if graph_edges[new_path[i]][j][0] == new_path[i + 1]
            )
            est = haversine(points_dict[neighbor], points_dict[goal])
            heapq.heappush(open_set, (cost + est, new_path))

    return []
```

Đoạn mã trên triển khai thuật toán A* nhằm tìm đường bay tối ưu từ điểm xuất phát start đến điểm đích goal, trong khi vẫn đảm bảo rằng các đoạn đường đi không cắt qua vùng cấm bay.

Thuật toán bắt đầu bằng cách khởi tạo một hàng đợi ưu tiên open_set, nơi mỗi phần tử gồm một tổng chi phí và đường đi tương ứng. Trong mỗi vòng lặp, thuật toán lấy ra đường có chi phí nhỏ nhất để tiếp tục mở rộng. Với mỗi điểm lân cận (neighbor) của điểm hiện tại (current), thuật toán tạo một đoạn đường (segment) nối hai điểm và sử dụng hàm is_blocked_by_no_flight() để kiểm tra xem đoạn này có cắt qua vùng cấm bay hay không. Nếu có, đoạn đó bị loại bỏ khỏi đường đi.

Nếu đoạn hợp lệ, thuật toán tính lại chi phí thực tế từ đầu đến điểm đó và ước lượng chi phí còn lại bằng công thức Haversine. Sau đó, nó đưa đường đi mới vào open_set để tiếp tục xét trong các vòng lặp tiếp theo. Thuật toán kết thúc khi tìm được một đường hợp lệ đến đích hoặc khi không còn đường nào khả thi.

Nhờ cơ chế kiểm tra vùng cấm tích hợp, thuật toán này đảm bảo rằng tất cả các đường bay được đề xuất đều hợp lệ và không vi phạm an toàn không phận.

5.5. Lấy các đường bay có tọa độ đầy đủ

Đoạn code Hình... được sử dụng để lấy danh sách các đường bay có đầy đủ tọa độ vị trí của các lô điểm. Cụ thể, hệ thống sẽ truy xuất tất cả các đường bay và kiểm tra từng lô điểm của mỗi đường bay đó. Chỉ khi lô điểm có đủ thông tin về vĩ độ và kinh độ thì mới được đưa vào danh sách tọa độ. Cuối cùng, chỉ những đường bay nào có từ hai điểm tọa độ trở lên mới được đưa vào kết quả trả về để đảm bảo rằng các đường bay được xử lý đều có thông tin đầy đủ về vị trí địa lý. Mục đích của chức năng này là để phục vụ công tác xử lý bản đồ và phân tích cụ thể là vẽ tuyến bay chính xác trên bản đồ, kiểm tra vi phạm vùng cấm bay và phân tích tuyến đường, tính toán khoảng cách của tuyến đường.

```

@app.route("/get-data-flight_path", methods=["GET"])
def get_data_flight_path():
    flight_paths = list(flight_path_collection.find({}, {"_id": 0, "name": 1, "waypoints": 1}))

    result = []

    for flight in flight_paths:
        name = flight["name"]
        waypoint_names = flight["waypoints"]

        coordinates = []
        for wp in waypoint_names:
            point = point_collection.find_one(
                {"ten_duong": wp},
                {"_id": 0, "vi_do": 1, "kinh_do": 1}
            )
            if point and point.get("vi_do") is not None and point.get("kinh_do") is not None:
                coordinates.append([point["vi_do"], point["kinh_do"]])

        if len(coordinates) >= 2:
            result.append({
                "name": name,
                "coordinates": coordinates
            })

    return jsonify(result), 200

```

6. Xóa dữ liệu

- Xóa vùng cấm bay

Đoạn mã trong hình định nghĩa một API với phương thức DELETE tại địa chỉ /no_flight, cho phép người dùng xóa một vùng cấm bay khỏi cơ sở dữ liệu dựa trên tên được cung cấp. Khi nhận yêu cầu, hệ thống kiểm tra xem trường "name" có tồn tại trong dữ liệu đầu vào hay không; nếu không có, sẽ trả về thông báo lỗi kèm mã trạng thái. Nếu tên hợp lệ, hệ thống tìm và xóa bản ghi tương ứng trong collection No_Flight. Nếu xóa thành công (tức có vùng cấm được tìm thấy), hệ thống phản hồi với thông báo xác nhận.

```

@app.route("/no_flight", methods=["DELETE"])
def delete_no_flight_zone():
    data = request.json
    name = data.get("name")
    if not name:
        return jsonify({"error": "Thiếu tên vùng cấm"}), 400

    result = No_Flight_collection.delete_one({"name": name})
    if result.deleted_count > 0:
        return jsonify({"message": f"Đã xóa vùng cấm {name}"}), 200
    else:
        return jsonify({"error": "Không tìm thấy vùng cấm để xóa"}), 404

```

- Xóa đường bay

Đoạn mã trong Hình xây dựng một API với phương thức POST tại địa chỉ /flight_path/delete, cho phép xóa một đường bay khỏi cơ sở dữ liệu dựa trên tên được cung cấp từ phía người dùng. Khi nhận yêu cầu, hệ thống sẽ kiểm tra xem dữ liệu có chứa trường "name" hay không; nếu thiếu, phản hồi lỗi kèm mã trạng thái. Trường hợp tên hợp lệ, hệ thống thực hiện xóa bản ghi tương ứng trong MongoDB.

```
@app.route("/flight_path/delete", methods=["POST"])
def delete_flight_path():
    data = request.json
    name = data.get("name")
    if not name:
        return jsonify({"error": "Thiếu tên chuyến bay để xóa"}), 400

    result = flight_path_collection.delete_one({"name": name})
    if result.deleted_count > 0:
        return jsonify({"message": f"Đã xóa chuyến bay {name}"}), 200
    else:
        return jsonify({"error": "Không tìm thấy chuyến bay để xóa"}), 404
```