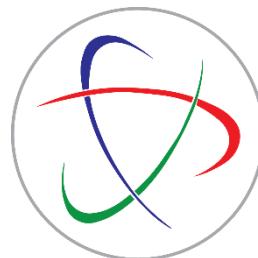


ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN TỬ - VIỄN THÔNG



BÁO CÁO
LẬP TRÌNH MẠNG

ĐỀ TÀI:

MÔ PHỎNG HỆ THỐNG GIAO HÀNG BẰNG DRONE

Sinh viên thực hiện: **Nguyễn Trọng Nhân – 22KTMT1 – 106220229**

Lê Duy Hoàng – 22KTMT2 – 106220252

Nguyễn Phan Hiếu Minh – 22KTMT2 – 106220259

Đà Nẵng, 11/2025

MỤC LỤC

1. Giới thiệu	5
1.1. Lý do chọn đề tài	5
1.2. PX4 Autopilot	5
1.3. Thông số môi trường mô phỏng.....	6
1.4. Thông số mô hình Drone	6
1.5. QGroundControl.....	7
1.6. Thông số vận tốc và gia tốc của Drone	8
1.7. MAVSDK.....	9
1.8. Tổng kết.....	10
2. Quy trình hoạt động và một số hàm cơ bản.....	10
2.1. Sơ đồ khối và quy trình hoạt động	10
2.2. Hàm khởi tạo	12
2.3. Hàm kết nối Drone	12
2.4. Hàm đọc nhiệm vụ từ QGroundControl.....	14
2.5. Hàm tính khoảng cách	15
2.6. Hàm Arm và cất cánh	15
2.7. Hàm bay đến vị trí waypoint.....	16
2.8. Hàm giao hàng.....	17
3. Thuật toán tối ưu tuyến đường giao hàng Drone	17
3.1. Thuật toán Sequential (Tuần tự).....	18
3.2. Thuật toán Nearest Neighbor	18
3.3. Thuật toán 2-Opt.....	20
3.4. So sánh các thuật toán	22
4. Demo Hệ thống mô phỏng Drone giao hàng	23
4.1. Các bước tiến hành mô phỏng	23
4.2. Demo Hệ thống mô phỏng Drone giao hàng với thuật toán Sequential.....	29
4.3. Demo Hệ thống mô phỏng Drone giao hàng với thuật toán Nearest Neighbor	32
4.4. Demo Hệ thống mô phỏng Drone giao hàng với thuật toán 2-Opt.....	34
4.5. Nhận xét chung.....	36

5. T&ograve;ng k&egrave;t	37
T&grave; tài liệu tham khảo	38

BẢNG PHÂN CÔNG CÔNG VIỆC TRONG NHÓM

STT	HỌ VÀ TÊN	NHIỆM VỤ	KHOI LUỢNG
01	Nguyễn Trọng Nhân	Tìm hiểu đề tài, các phần mềm sử dụng, thuật toán, viết chương trình, làm báo cáo, bản tóm tắt, slide thuyết trình.	34%
02	Lê Duy Hoàng	Tìm hiểu đề tài, các phần mềm sử dụng, thuật toán, viết chương trình, làm báo cáo, bản tóm tắt, slide thuyết trình.	33%
03	Nguyễn Phan Hiếu Minh	Tìm hiểu đề tài, các phần mềm sử dụng, thuật toán, viết chương trình, làm báo cáo, bản tóm tắt, slide thuyết trình.	33%

1. Giới thiệu

1.1. Lý do chọn đề tài

Trong những năm gần đây, công nghệ drone (UAV - Unmanned Aerial Vehicle) đã có sự phát triển vượt bậc và được ứng dụng rộng rãi trong nhiều lĩnh vực như nông nghiệp, giám sát, cứu hộ, và đặc biệt là trong lĩnh vực vận chuyển và giao hàng. Việc sử dụng drone để giao hàng mang lại nhiều lợi ích như giảm thời gian vận chuyển, tiết kiệm chi phí nhân công, và có khả năng tiếp cận các khu vực khó khăn.

Tuy nhiên, việc triển khai hệ thống drone giao hàng trên thực tế đòi hỏi chi phí lớn và tiềm ẩn nhiều rủi ro. Do đó, việc xây dựng một hệ thống mô phỏng hoàn chỉnh là bước đầu tiên quan trọng để:

- Nghiên cứu và phát triển: Cho phép thử nghiệm các thuật toán điều khiển, tối ưu hóa tuyến đường mà không cần phần cứng thực tế
- Giảm chi phí: Tránh được các rủi ro hư hỏng thiết bị trong quá trình thử nghiệm
- Đào tạo và học tập: Cung cấp môi trường an toàn để học về điều khiển drone và lập trình tự động
- Kiểm thử nhanh chóng: Dễ dàng thay đổi điều kiện môi trường và kịch bản để đánh giá hiệu năng hệ thống

Dự án này được thực hiện nhằm xây dựng một hệ thống mô phỏng đầy đủ cho drone giao hàng, sử dụng các công cụ mã nguồn mở như PX4, Gazebo, và MAVSDK, tạo nền tảng cho việc nghiên cứu và phát triển các giải pháp điều khiển drone trong tương lai.

1.2. PX4 Autopilot

PX4 Autopilot là một hệ điều hành mã nguồn mở dành cho các phương tiện tự hành (autonomous vehicles), đặc biệt phổ biến trong lĩnh vực drone. PX4 được phát triển bởi cộng đồng Dronecode và được sử dụng rộng rãi trong cả nghiên cứu học thuật lẫn ứng dụng thương mại.

Đặc điểm chính của PX4:

- Mã nguồn mở: Hoàn toàn miễn phí và có cộng đồng phát triển đông đảo.
- Tính linh hoạt cao: Hỗ trợ nhiều loại phương tiện như multirotor, fixed-wing, VTOL, rover.
- Chế độ bay đa dạng: Manual, Stabilized, Altitude, Position, Mission, Offboard...
- Tích hợp tốt: Tương thích với nhiều phần cứng và công cụ mô phỏng (Gazebo, jMAVSIM).

- Hệ thống cảm biến phong phú: Hỗ trợ IMU, GPS, Magnetometer, Barometer, và nhiều cảm biến khác.

Trong dự án này, PX4 được sử dụng trong chế độ SITL (Software-In-The-Loop) để mô phỏng bộ điều khiển bay (flight controller) của drone. Điều này cho phép chạy toàn bộ stack điều khiển của PX4 trên máy tính mà không cần phần cứng thực tế.

1.3. Thông số môi trường mô phỏng

Hệ thống mô phỏng được triển khai trên Ubuntu chạy trên máy ảo VirtualBox, sử dụng Gazebo làm công cụ mô phỏng vật lý và môi trường 3D.

Môi trường mô phỏng sử dụng Empty World - một thế giới ảo cơ bản với các thành phần môi trường tối thiểu:

- Sun (Mặt trời): Nguồn sáng chính cho môi trường.
- Ground Plane: Mặt phẳng đất cơ bản.

Một vài vật lý của môi trường mô phỏng biểu diễn như ở Bảng 1.

Bảng 1: Một vài thông số vật lý của môi trường mô phỏng

Thông số	Giá trị	Ý nghĩa
Trọng lực	0, 0, -9.8066 m/s ²	Gia tốc trọng trường theo trục Z (hướng xuống)
Max Step Size	0.004 s (4 ms)	Bước thời gian tối đa cho mỗi vòng lặp vật lý
Real Time Factor	1.0	Tỷ lệ thời gian thực (1 = thời gian thực)
Real Time Update Rate	250 Hz	Tần số cập nhật mô phỏng (250 lần/giây)

1.4. Thông số mô hình Drone

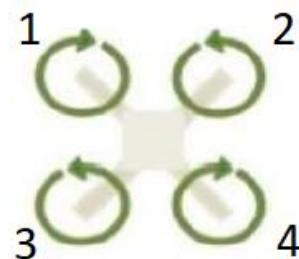
Drone X500 là một mô hình quadcopter (4 cánh quạt) phô biến, được thiết kế theo cấu hình "X" với các đặc điểm kỹ thuật phù hợp cho các ứng dụng giao hàng và khảo sát.

Nguyên lý: Các cánh quạt quay sẽ tạo ra một lực đẩy và momen xoắn. Để drone có thể bay lên, tổng hợp lực đẩy phải lớn hơn trọng lực của vật. Cặp cánh quạt 1-4 quay theo chiều kim đồng hồ, cặp cánh quạt 2-3 quay ngược chiều kim đồng hồ nhằm cân bằng

momen xoắn tạo ra bởi các cánh quạt trên khung. Với việc điều khiển 4 cánh quạt phối hợp tạo ra lực phù hợp thì ta có thể điều khiển được chuyển động của drone theo ý muốn.

Cấu hình động cơ của Drone (biểu diễn như ở Hình 1) gồm:

- Rotor 1 : Quay cùng chiều kim đồng hồ.
- Rotor 2 : Quay ngược chiều kim đồng hồ.
- Rotor 3 : Quay ngược chiều kim đồng hồ.
- Rotor 4 : Quay cùng chiều kim đồng hồ.



Hình 1: Cấu hình động cơ Drone

Thông số cơ bản của drone:

- Khối lượng body chính: 1.5 kg.
- Khối lượng mỗi rotor: 0.005 kg (5 gram).
- Kích thước thân: 0.47 x 0.47 x 0.11 m.
- Bán kính rotor: 0.128 m.

1.5. QGroundControl

QGroundControl (QGC) là một phần mềm trạm điều khiển mặt đất (Ground Control Station - GCS) mã nguồn mở, được thiết kế để cấu hình, giám sát và điều khiển các phương tiện tự hành sử dụng giao thức MAVLink.

Chức năng chính của QGroundControl:

1. Giám sát trạng thái drone:

- Theo dõi vị trí, độ cao, tốc độ của drone theo thời gian thực.
- Hiển thị thông tin cảm biến (GPS, IMU, battery).
- Kiểm tra tình trạng kết nối và sức khỏe hệ thống.

2. Lập kế hoạch bay (Mission Planning):

- Tạo và chỉnh sửa các waypoint (điểm đến) trên bản đồ.
- Thiết lập độ cao, tốc độ bay cho từng waypoint.
- Xuất file .plan chứa toàn bộ thông tin mission.
- Hỗ trợ các lệnh như: Takeoff, Waypoint, Landing, RTL (Return to Launch).

3. Cấu hình tham số: Điều chỉnh các thông số PX4 (vận tốc, gia tốc, hành vi bay...).

4. Điều khiển thủ công:

- Arm/Disarm drone.
- Thay đổi chế độ bay (Manual, Stabilized, Position, Mission, Offboard).
- Emergency operations (Kill switch, Return to Home).

Trong dự án này, QGC được sử dụng để:

- Kết nối với PX4 SITL.
- Tạo file .plan chứa các điểm giao hàng (waypoints).
- Giám sát quá trình bay của drone trong môi trường mô phỏng.
- Xác minh tính đúng đắn của các thuật toán điều khiển.

1.6. Thông số vận tốc và gia tốc của Drone

Các thông số vận tốc và gia tốc của drone được định nghĩa trong PX4 thông qua các tham số MPC (Multirotor Position Controller). Những thông số này ảnh hưởng trực tiếp đến hành vi bay và hiệu năng của drone. Thông số vận tốc (Velocity Parameters) của Drone được biểu diễn ở Bảng 2.

Bảng 2: Thông số vận tốc của Drone

Tham số	Giá trị	Ý nghĩa
MPC_XY_CRUISE	11 mph	Tốc độ bay ngang mặc định
MPC_XY_VEL_MAX	26.8 mph	Tốc độ ngang tối đa
MPC_Z_VEL_MAX_UP	6.7 mph	Tốc độ leo lên tối đa
MPC_Z_VEL_MAX_DN	3.4 mph	Tốc độ hạ xuống tối đa
MPC_LAND_SPEED	1.6 mph	Tốc độ hạ cánh
MPC_TKO_SPEED	3.36 mph	Tốc độ cất cánh

Lưu ý: Để drone bay nhanh hơn trong mission, cần tăng giá trị của MPC_XY_CRUISE và MPC_XY_VEL_MAX. Trong dự án này, các giá trị mặc định đã được tối ưu cho cân bằng giữa tốc độ và sự ổn định.

Thông số gia tốc (Acceleration Parameters) của Drone được biểu diễn ở Bảng 3.

Bảng 3: Thông số gia tốc của Drone

Tham số	Giá trị	Ý nghĩa
MPC ACC HOR MAX	5.0 m/s ²	Gia tốc ngang tối đa
MPC ACC HOR	3.0 m/s ²	Gia tốc ngang mặc định
MPC ACC UP MAX	4.0 m/s ²	Gia tốc leo lên tối đa
MPC ACC DOWN MAX	3.0 m/s ²	Gia tốc hạ xuống tối đa
MPC JERK AUTO	4.0 m/s ³	Jerk (độ giật) tự động
MPC JERK MAX	8 m/s ³	Jerk tối đa

1.7. MAVSDK

MAVSDK là một thư viện client cho MAVLink, cung cấp API để sử dụng để lập trình điều khiển drone. MAVSDK hỗ trợ nhiều ngôn ngữ, trong dự án này sử dụng MAVSDK-Python. Ví dụ kết nối và một số câu lệnh điều khiển cơ bản của Drone được thể hiện như ở Bảng 4 với thư viện điều khiển Drone qua MAVLink: from mavsdk import System.

Bảng 4: Một số câu lệnh điều khiển cơ bản của Drone

Cách lệnh cơ bản:	Code
Kết nối với PX4	drone = System() await drone.connect(system_address="udp://:14540")
Chờ drone kết nối	async for state in drone.core.connection_state(): if state.is_connected: break
Arm	await drone.action.arm()
Cất cánh	await drone.action.takeoff()
Bay đến vị trí chỉ định	await drone.action.goto_location(latitude_deg=21.028511, longitude_deg=105.804817, absolute_altitude_m=50, yaw_deg=0)
Hạ cánh	await drone.action.land()

1.8. Tổng kết

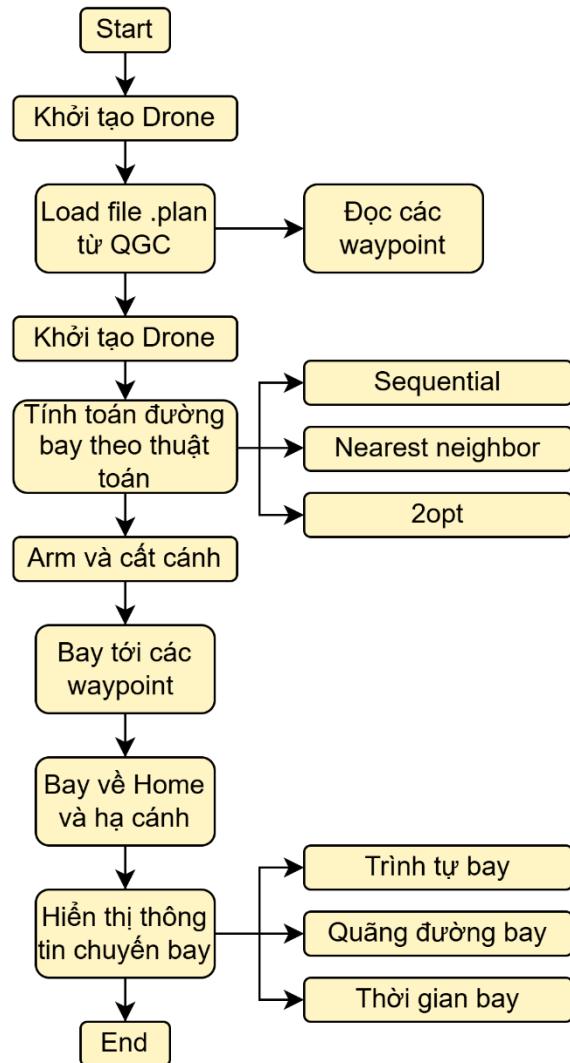
Hệ thống mô phỏng drone giao hàng được xây dựng trên nền tảng các công cụ mã nguồn mở mạnh mẽ và được cộng đồng hỗ trợ rộng rãi. Sự kết hợp giữa PX4 (autopilot), Gazebo (mô trường mô phỏng), QGroundControl (trạm điều khiển), và MAVSDK (thư viện lập trình) tạo nên một hệ sinh thái hoàn chỉnh để nghiên cứu và phát triển ứng dụng drone.

Với các thông số vận tốc và gia tốc được tối ưu hóa, drone X500 trong mô trường mô phỏng có khả năng thực hiện các nhiệm vụ được giao.

2. Quy trình hoạt động và một số hàm cơ bản

2.1. Sơ đồ khái niệm và quy trình hoạt động

Quy trình hoạt động cơ bản của Drone có thể mô tả bằng sơ đồ khái niệm ở Hình 2.



Hình 2: Sơ đồ khái quát quy trình hoạt động của Drone

Hệ thống giao hàng drone tự động hoạt động theo một quy trình có trình tự rõ ràng, bắt đầu từ việc khởi tạo kết nối với drone thông qua giao thức MAVLink tại cổng UDP 14540. Ngay sau khi thiết lập kết nối thành công, chương trình lập tức thu thập thông tin về vị trí xuất phát (home position) bao gồm tọa độ GPS và độ cao tuyệt đối, đồng thời đọc file kế hoạch bay định dạng .plan từ QGroundControl để trích xuất danh sách các điểm giao hàng với đầy đủ thông tin vĩ độ, kinh độ và độ cao của từng waypoint.

Trước khi thực hiện nhiệm vụ, hệ thống tính toán tuyến đường bay theo thuật toán đã đề ra (Sequential, Nearest neighbor hoặc 2opt). Chương trình sử dụng công thức Haversine để tính toán chính xác khoảng cách thực tế giữa các điểm trên bề mặt cầu Trái Đất, từ đó ước lượng tổng quãng đường bay bao gồm cả đoạn về điểm xuất phát. Sau khi hoàn tất việc lập kế hoạch, drone được kích hoạt động cơ (arm) và thực hiện cất cánh tự

động lên độ cao được chỉ định bởi waypoint đầu tiên, trong khi hệ thống bắt đầu ghi nhận thời điểm khởi đầu để đo lường hiệu suất bay.

Quá trình giao hàng diễn ra tuần tự khi drone bay đến từng điểm đích bằng lệnh, trong đó hệ thống liên tục giám sát vị trí GPS thời gian thực thông qua và tính toán khoảng cách còn lại đến đích. Khi drone đã đến nơi, chuyển sang giai đoạn giao hàng - được mô phỏng bằng việc hover tại chỗ trong 5 giây. Sau khi hoàn thành việc giao hàng tại tất cả các điểm, drone tự động bay về vị trí home, thực hiện hạ cánh an toàn, và cuối cùng chương trình xuất ra báo cáo thống kê chi tiết bao gồm tổng thời gian bay, tổng quãng đường, tốc độ trung bình, cùng với dữ liệu phân tích riêng cho từng đoạn bay, cung cấp cái nhìn toàn diện về hiệu suất và độ chính xác của mỗi thuật toán trong việc thực hiện nhiệm vụ giao hàng tự động.

2.2. Hàm khởi tạo

Hàm khởi tạo (biểu diễn ở Hình 3) thiết lập các thuộc tính ban đầu cho đối tượng drone, chuẩn bị môi trường để thực hiện các thao tác điều khiển và giám sát.

```
def __init__(self):
    self.drone = System()          #Tạo đối tượng Drone
    self.delivery_points = []      #Danh sách các điểm giao hàng
    self.home_position = None      #Vị trí xuất phát
    self.total_distance = 0         #Tổng quãng đường bay
    self.total_time = 0             #Tổng thời gian bay
    self.start_time = None         #Thời điểm bắt đầu
    self.segment_distances = []    #Quãng đường từng đoạn
    self.segment_times = []        #Thời gian từng đoạn
```

Hình 3: Hàm khởi tạo

2.3. Hàm kết nối Drone

Hàm kết nối Drone (biểu diễn ở Hình 4) thiết lập kết nối giữa chương trình Python và Drone mô phỏng, đồng thời thu thập thông tin vị trí ban đầu để làm điểm tham chiếu cho toàn bộ nhiệm vụ.

```

async def connect(self): 3 usages (2 dynamic)
    print("🔧 Đang kết nối với drone...")
    await self.drone.connect(system_address="udp://:14540")

    print("⏳ Đợi drone sẵn sàng...")
    async for state in self.drone.core.connection_state():
        if state.is_connected:
            print("✅ Đã kết nối với drone!")
            break

    print("📍 Lấy vị trí home...")
    async for position in self.drone.telemetry.position():
        self.home_position = {
            'lat': position.latitude_deg,
            'lon': position.longitude_deg,
            'alt': position.absolute_altitude_m
        }
        print(
            f"🏠 Home: Lat={self.home_position['lat']:.6f}, "
            f"Lon={self.home_position['lon']:.6f}, "
            f"Alt={self.home_position['alt']:.1f}m"
        )
        break

```

Hình 4: Hàm kết nối Drone

Quy trình hoạt động gồm các bước sau:

- Bước 1: Thiết lập kết nối: Hàm `connect()` sử dụng giao thức UDP tại cổng 14540, đây là cổng mặc định cho PX4 SITL (Software In The Loop) simulation, cho phép chương trình lắng nghe kết nối đến từ drone simulator.
- Bước 2: Xác nhận kết nối: Chương trình sử dụng vòng lặp bất đồng bộ `async for` để liên tục kiểm tra trạng thái kết nối thông qua `drone.core.connection_state()`. Khi thuộc tính `is_connected` trả về True, vòng lặp dừng lại và xác nhận kết nối thành công.
- Bước 3: Thu thập vị trí home: Sau khi kết nối, hàm đọc dữ liệu vị trí từ drone thông qua `drone.telemetry.position()`. Dữ liệu GPS đầu tiên nhận được sẽ được lưu làm `home position`, bao gồm:
 - `latitude_deg`: Vĩ độ tính bằng độ
 - `longitude_deg`: Kinh độ tính bằng độ
 - `absolute_altitude_m`: Độ cao tuyệt đối so với mực nước biển (mét)

Hàm này là bước đầu tiên bắt buộc trong mọi chương trình điều khiển drone. Nếu không thiết lập kết nối thành công, mọi lệnh điều khiển sau đó sẽ thất bại. Việc lưu home position cũng đảm bảo drone luôn có điểm tham chiếu để quay về khi cần thiết.

2.4. Hàm đọc nhiệm vụ từ QGroundControl

Hàm đọc nhiệm vụ từ QGroundControl (biểu diễn ở Hình 5) thực hiện đọc và phân tích file kế hoạch bay định dạng .plan từ QGroundControl, trích xuất thông tin các waypoint để tạo danh sách điểm giao hàng.

```
def load_mission_from_qgc(self, plan_file): 1 usage
    print(f"📁 Đang đọc file: {plan_file}")
    try:
        with open(plan_file, 'r') as f:
            data = json.load(f)

        items = data.get('mission', {}).get('items', [])
        for i, item in enumerate(items):
            if item.get('type') == 'SimpleItem' and item.get('command') == 16:
                params = item.get('params', [])
                if len(params) >= 7:
                    self.delivery_points.append({
                        'name': f'Diểm {i+1}',
                        'lat': params[4],
                        'lon': params[5],
                        'alt': params[6]
                    })

        print(f"✅ Đã load {len(self.delivery_points)} điểm giao hàng:")
        for point in self.delivery_points:
            print(
                f" • {point['name']}: Lat={point['lat']:.6f}, "
                f"Lon={point['lon']:.6f}, Alt={point['alt']:.1f}m"
            )
    
```

Hình 5: Hàm đọc nhiệm vụ từ QGroundControl

File .plan là định dạng JSON được QGroundControl sử dụng để lưu trữ kế hoạch bay. Chương trình thực hiện đọc và parse file .plan thành cấu trúc dữ liệu Python (dictionary và list). Sau đó truy xuất danh sách items bằng hàm get() để truy cập vào các key lồng nhau tiền hành trích xuất thông tin mỗi waypoint bao gồm: tên, vĩ độ, kinh độ và độ cao tuyệt đối. Mỗi waypoint được chuyển đổi thành dictionary với các key chuẩn hóa (name, lat, lon, alt) và thêm vào self.delivery_points.

2.5. Hàm tính khoảng cách

Hàm tính khoảng cách (biểu diễn ở Hình 6) thực hiện tính toán khoảng cách thực tế giữa hai điểm GPS trên bề mặt cầu Trái Đất sử dụng công thức Haversine.

```
def haversine_distance(self, lat1, lon1, lat2, lon2): 4 usages
    """Tính khoảng cách giữa 2 điểm GPS (Haversine)"""
    R = 6371000
    phi1 = math.radians(lat1)
    phi2 = math.radians(lat2)
    delta_phi = math.radians(lat2 - lat1)
    delta_lambda = math.radians(lon2 - lon1)

    a = math.sin(delta_phi/2)**2 + math.cos(phi1) * math.cos(phi2) * math.sin(delta_lambda/2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))

    return R * c
```

Hình 6: Hàm tính khoảng cách

2.6. Hàm Arm và cất cánh

Hàm Arm và cất cánh (biểu diễn ở Hình 7) kích hoạt động cơ drone và thực hiện cất cánh tự động lên độ cao chỉ định.

```
async def arm_and_takeoff(self, altitude=10): 1 usage
    print("🔧 Đang arm drone...")
    await self.drone.action.arm()

    print(f"✈️ Cất cánh lên độ cao {altitude}m...")
    await self.drone.action.set_takeoff_altitude(altitude)
    await self.drone.action.takeoff()
    await asyncio.sleep(10)

    print("✅ Đã cất cánh")
```

Hình 7: Hàm Arm và cất cánh

Quy trình thực hiện như sau:

- Arm động cơ: Lệnh `arm()` kích hoạt các động cơ, cho phép chúng quay. Đây là tính năng an toàn bắt buộc trong PX4 - động cơ chỉ hoạt động khi được arm chủ động.
- Đặt độ cao cất cánh: `set_takeoff_altitude()` cấu hình độ cao mục tiêu cho thao tác cất cánh. PX4 sẽ tự động điều khiển drone leo lên độ cao này.

- Thực hiện cất cánh: Lệnh takeoff() kích hoạt chế độ cất cánh tự động.
- Chờ ổn định: asyncio.sleep(10) cho drone 10 giây để hoàn tất cất cánh và ổn định hoàn toàn trước khi thực hiện lệnh tiếp theo. Thời gian này đảm bảo tất cả các cảm biến đã cân bằng và vị trí GPS đã chính xác.

2.7. Hàm bay đến vị trí waypoint

Hàm bay đến vị trí waypoint (biểu diễn ở Hình 8) điều khiển drone bay đến tọa độ GPS cụ thể và giám sát quá trình bay để xác định khi nào đã đến đích.

```
async def fly_to_gps_location(self, lat, lon, alt, yaw=float('nan')): 2 usages
    segment_start = time.time()
    async for position in self.drone.telemetry.position():
        start_lat = position.latitude_deg
        start_lon = position.longitude_deg
        break
    distance = self.haversine_distance(start_lat, start_lon, lat, lon)
    print(
        f"📍 Bay đến: Lat={lat:.6f}, Lon={lon:.6f}, Alt={alt:.1f}m "
        f"(khoảng cách: {distance:.1f}m)"
    )
    await self.drone.action.goto_location(lat, lon, alt, yaw)
    while True:
        async for position in self.drone.telemetry.position():
            current_distance = self.haversine_distance(
                position.latitude_deg,
                position.longitude_deg,
                lat,
                lon
            )
            if current_distance < 2:
                segment_time = time.time() - segment_start
                print(
                    f"✅ Đã đến vị trí (còn cách {current_distance:.1f}m) - "
                    f"Thời gian: {segment_time:.1f}s, Quãng đường: {distance:.1f}m"
                )
                self.segment_distances.append(distance)
                self.segment_times.append(segment_time)
                return
        await asyncio.sleep(1)
```

Hình 8: Hàm bay đến vị trí waypoint

Chức năng chính bao gồm:

- Đo thời gian: Ghi lại thời điểm bắt đầu đoạn bay bằng time.time() để tính toán thời gian bay thực tế.
- Lấy vị trí hiện tại: Đọc GPS của drone trước khi bay để tính khoảng cách cần bay.

- Tính khoảng cách đoạn: Sử dụng công thức Haversine để tính khoảng cách từ vị trí hiện tại đến đích.
- Gửi lệnh bay: goto_location() là lệnh điều khiển của MAVSDK. PX4 autopilot sẽ: Tính toán quỹ đạo bay, điều chỉnh hướng bay, kiểm soát tốc độ, duy trì độ cao.
- Vòng lặp giám sát: Chương trình liên tục đọc vị trí GPS và tính khoảng cách còn lại đến đích. Vòng lặp này chạy cho đến khi điều kiện đến nơi được thỏa mãn. Xác định đã đến đích khi khoảng cách < 2 mét. Sai số 2m là khoảng cách đủ nhỏ cho mục đích giao hàng.
- Ghi nhận dữ liệu: Khi đã đến nơi, hàm tính toán và lưu thời gian bay thực tế của đoạn, quãng đường thực tế đã bay để phân tích sau.

2.8. Hàm giao hàng

Hàm giao hàng (biểu diễn ở Hình 9) mô phỏng quá trình giao hàng tại điểm đích bằng cách duy trì trạng thái hover trong khoảng thời gian nhất định.

```
async def deliver_package(self, point_name, duration=5):  1 usage
    print(f"📦 Đang giao hàng tại {point_name}...")
    await asyncio.sleep(duration)
    print(f"✓ Đã giao hàng tại {point_name}")
```

Hình 9: Hàm giao hàng

Trong môi trường mô phỏng, không có thiết bị thả hàng vật lý. Hàm này sử dụng asyncio.sleep() để tạm dừng chương trình một khoảng thời gian (cụ thể là 5s), đại diện cho thời gian thực hiện hành động giao hàng.

3. Thuật toán tối ưu tuyến đường giao hàng Drone

Bài toán tối ưu hóa tuyến đường giao hàng là một yếu tố quan trọng trong hệ thống mô phỏng drone giao hàng. Với drone giao hàng, việc lựa chọn tuyến đường bay hiệu quả không chỉ giúp tiết kiệm thời gian mà còn tối ưu hóa mức tiêu thụ pin - yếu tố then chốt quyết định khả năng hoạt động của drone. Trong hệ thống này, ba thuật toán được triển khai và so sánh: Sequential (tuần tự), Nearest Neighbor và 2-Opt. Mỗi thuật toán có cách tiếp cận khác nhau về độ phức tạp, thời gian tính toán và chất lượng kết quả, phù hợp với các tình huống ứng dụng cụ thể.

3.1. Thuật toán Sequential (Tuần tự)

Thuật toán Sequential là thuật toán đơn giản nhất, drone bay theo đúng thứ tự các điểm giao hàng đã được định nghĩa trong file kế hoạch .plan từ QGroundControl. Thuật toán không thực hiện bất kỳ phép tối ưu hóa nào, chỉ đơn thuần sao chép danh sách điểm giao hàng ban đầu (biểu diễn như ở Hình 9).

```
def sequential_route(self): 1 usage
    return self.delivery_points.copy()
```

Hình 9: Thuật toán Sequential

Giả sử có 4 điểm giao hàng A, B, C, D được định nghĩa theo thứ tự trong file .plan thì thứ tự giao hàng của Drone sẽ lần lượt là: Home → A → B → C → D → Home. Drone sẽ bay chính xác theo trình tự này mà không xem xét bất kì điều kiện nào khác.

Ưu điểm:

- Cực kì đơn giản và dễ triển khai.
- Dễ debug và bảo trì.
- Phù hợp khi thứ tự giao hàng có ý nghĩa đặc biệt (ưu tiên, deadline).
- Kết quả có thể dự đoán trước, dễ kiểm soát.
- Thời gian tính toán dường như bằng 0.

Nhược điểm:

- Phụ thuộc hoàn toàn vào cách người dùng sắp xếp waypoint.
- Không tối ưu lộ trình bay dẫn đến tốn năng lượng và thời gian.
- Không phù hợp cho hệ thống tự động hóa hoàn toàn.

Với những ưu, nhược điểm trên, Drone có thể sử dụng thuật toán này trong một vài trường hợp cố định như: Thứ tự giao hàng bắt buộc (khách hàng VIP, hàng khẩn cấp), số điểm giao hàng ít, demo, kiểm thử ban đầu hay người dùng đã tự tối ưu thứ tự trước khi tạo file .plan.

3.2. Thuật toán Nearest Neighbor

Thuật toán Nearest Neighbor (biểu diễn ở Hình 10) hoạt động theo quy tắc là tại mỗi bước luôn chọn điểm giao hàng gần nhất chưa được ghé thăm. Thuật toán bắt đầu từ vị trí Home, tìm điểm gần nhất trong danh sách còn lại, bay đến điểm đó, sau đó lại tìm điểm gần nhất tiếp theo từ vị trí hiện tại. Quy trình này được lặp lại cho đến khi không còn bắt cứ điểm giao hàng nào còn lại trong danh sách. Sau đó Drone sẽ bay về vị trí Home ban đầu.

```

def nearest_neighbor_route(self): 1 usage
    """Thuật toán nearest neighbor để tối ưu route"""
    if not self.delivery_points or not self.home_position:
        return []

    route = []
    remaining = self.delivery_points.copy()

    current_lat = self.home_position['lat']
    current_lon = self.home_position['lon']

    while remaining:
        nearest = min(
            remaining,
            key=lambda p: self.haversine_distance(
                current_lat, current_lon, p['lat'], p['lon'])
        )
        route.append(nearest)
        remaining.remove(nearest)
        current_lat = nearest['lat']
        current_lon = nearest['lon']

    return route

```

Hình 10: Thuật toán Nearest Neighbor

Đầu tiên, chương trình kiểm tra xem danh sách điểm giao và vị trí home có hợp lệ hay không. Sau đó, nó sao chép danh sách điểm giao còn lại và bắt đầu từ tọa độ home. Trong vòng lặp, hàm min() được dùng để tìm waypoint gần nhất so với vị trí hiện tại dựa trên khoảng cách Haversine. Điểm gần nhất được thêm vào route, đồng thời bị xóa khỏi danh sách "remaining". Vị trí hiện tại được cập nhật sang điểm vừa chọn và quá trình lặp tiếp tục cho tới khi không còn điểm nào. Cuối cùng, hàm trả về route đã được tối ưu theo chiến lược chọn điểm gần nhất từng bước.

Ưu điểm:

- Đơn giản, dễ hiểu, dễ triển khai.
- Cải thiện hơn so với Sequential.
- Cho ra kết quả đủ tốt trong hầu hết các trường hợp.
- Thời gian tính toán tương đối nhanh.

Nhược điểm:

- Vẫn không đảm bảo tuyến đường tối ưu toàn cục.
- Kết quả phụ thuộc vào điểm xuất phát.

- Với các điểm phân bố phức tạp, có thể cho ra kết quả kém.

Với những ưu, nhược điểm trên, Drone có thể sử dụng thuật toán này trong một số trường hợp như cần tối ưu đủ tốt nhanh chóng, hệ thống realtime, cần phản hồi ngay, cần cân bằng giữa hiệu quả và tốc độ.

3.3. Thuật toán 2-Opt

2-Opt (Two-Edge Exchange Optimization) (biểu diễn ở Hình 11) là một phương pháp tối ưu cục bộ được sử dụng rộng rãi trong việc giải gần đúng bài toán tìm đường đi ngắn nhất cho Drone giao hàng. Đây là thuật toán tối ưu hóa cải tiến, hoạt động dựa trên ý tưởng "cắt và đảo ngược" các đoạn trong tuyến đường. Thuật toán lấy một tuyến đường ban đầu bằng cách sử dụng thuật toán Nearest Neighbor, sau đó liên tục thử đảo ngược các đoạn con để tìm cách cải thiện. Các tuyến đường mới được tạo nếu kém hiệu quả thì sẽ bị loại bỏ, nếu hiệu quả hơn sẽ được giữ lại và tiếp tục lặp lại quá trình cho đến khi tìm được tuyến đường hiệu quả nhất.

Giả sử route có các điểm: A, B, C, D, E. Thuật toán 2-Opt có thể được dùng để xử lý như sau:

- Bước 1: Lấy route ban đầu theo thuật toán Nearest Neighbor, giả sử ta có được route theo Nearest Neighbor là: A → B → C → D → E.
- Bước 2: Duyệt tất cả cặp cạnh có thể. Chọn hai cạnh không kề nhau, ví dụ như cạnh (B–C) và cạnh (D–E).
- Bước 3: Kiểm tra thay thế. Ta xét tổng hai cạnh cũ và tổng hai cạnh mới nếu đảo đoạn. Nếu $(\text{distance}(B, C) + \text{distance}(D, E)) > (\text{distance}(B, D) + \text{distance}(C, E))$ thì đảo đoạn C–D sẽ rút ngắn hành trình.
- Bước 4: Thực hiện đảo route: A → B → C → D → E thành A → B → D → C → E.
- Bước 5: Ta sẽ lặp lại các Bước 2, 3, 4 cho đến khi không còn cặp cạnh nào làm route ngắn hơn nữa.

```

def two_opt_optimize(self, route): 1 usage
    if len(route) < 4:
        return route

    improved = True
    best_route = route.copy()
    best_distance = self.calculate_route_distance(best_route)
    iterations = 0

    print("⌚ Đang tối ưu route bằng 2-Opt...")

    while improved:
        improved = False
        iterations += 1

        for i in range(len(best_route) - 1):
            for j in range(i + 2, len(best_route)):

                new_route = best_route[:i+1] + best_route[i+1:j+1][::-1] + best_route[j+1:]
                new_distance = self.calculate_route_distance(new_route)

                if new_distance < best_distance:
                    best_route = new_route
                    best_distance = new_distance
                    improved = True
                    print(f"✓ Iteration {iterations}: Cải thiện {best_distance:.1f}m "
                          f"(đảo đoạn {i+1}-{j+1})")

    return best_route

```

Hình 11: Thuật toán 2-Opt

Đoạn code trên mô tả cách thuật toán 2-Opt tối ưu lại một lộ trình đã có bằng cách thử đảo ngược các đoạn con để rút ngắn tổng quãng đường. Đầu tiên, chương trình kiểm tra nếu route quá ngắn (dưới 4 điểm) thì không cần tối ưu và trả về luôn. Sau đó, nó tạo bản sao của route ban đầu và tính khoảng cách hiện tại. Thuật toán chạy trong vòng lặp while improved, nghĩa là tiếp tục tối ưu cho đến khi không còn cải thiện nào nữa. Bên trong, hai vòng lặp for i, j lần lượt chọn mọi cặp vị trí trong route để thử đảo ngược đoạn từ i+1 đến j. Mỗi lần đảo, chương trình tạo ra một route mới, tính lại tổng quãng đường, rồi so sánh với best_distance hiện tại. Nếu route mới tốt hơn, nó cập nhật best_route, best_distance và đánh dấu improved = True để tiếp tục vòng lặp. Quá trình này lặp lại nhiều lần cho đến khi không còn bất kỳ phép đảo nào giúp giảm quãng đường, và cuối cùng trả về tuyến đường tối ưu thu được.

Ưu điểm:

- Cải thiện đáng kể so với các thuật toán trước (5 - 15% quãng đường).
- Đảm bảo tìm được đường tối ưu cục bộ.
- Kết quả ổn định, ít phụ thuộc route ban đầu.
- Loại bỏ được các đường giao nhau trong tuyến.

Nhược điểm:

- Thời gian tính toán lâu hơn so với các thuật toán trước.
- Càng xử lý nhiều điểm, tốn càng nhiều thời gian.
- Cần route khởi tạo đủ tốt để thuật toán dễ tối ưu.

Với những ưu, nhược điểm như vật thì Drone có thể sử dụng thuật toán này trong một số trường hợp như yêu cầu tuyến đường tối ưu nhất có thể, có khoảng thời gian tính toán trước, nhiệm vụ quan trọng, cần tiết kiệm năng lượng.

3.4. So sánh các thuật toán

Với những thông tin trên, chúng ta có thể tạo một bảng so sánh để thấy được sự khác biệt giữa 3 thuật toán như trong Bảng 5.

Bảng 5: So sánh các thuật toán

Tiêu chí	Sequential (Theo thứ tự)	Nearest Neighbor (Láng giềng gần nhất)	2-Opt (Tối ưu hóa)
Nguyên lý	Giữ nguyên thứ tự waypoint từ file	Luôn chọn điểm gần nhất chưa đi	Đảo ngược các đoạn để chọn ra tuyến tối ưu nhất
Độ phức tạp	Đơn giản	Đơn giản	Trung bình
Chất lượng route	Kém nhất	Khá tốt	Tốt nhất
Tốc độ tính toán	Tức thì	Nhanh	Trung bình
Tiết kiệm pin	Kém	Khá	Tốt
Loại bỏ đường giao nhau	Không	Không	Có
Tối ưu	Cục bộ từng bước	Cục bộ từng bước	Cục bộ toàn route
Khi nào dùng	Thứ tự bắt buộc, demo/test, không quan tâm khoảng cách	Cần nhanh, số điểm vừa phải, cần kết quả khá tốt	Cần tối ưu cao, có thời gian tính toán, tiết kiệm năng lượng
Trường hợp tốt nhất	Waypoint đã được sắp xếp tốt	Các điểm phân bố đều	Route có nhiều đường giao nhau

Trường hợp tệ nhất	Waypoint ngẫu nhiên	"Trap case" - điểm gần tạo đường xa	Route đã gần tối ưu (lãng phí thời gian)
---------------------------	---------------------	-------------------------------------	--

4. Demo Hệ thống mô phỏng Drone giao hàng

4.1. Các bước tiến hành mô phỏng

Hệ thống mô phỏng Drone giao hàng được chạy trên hệ điều hành Ubuntu được cài đặt trên máy ảo VirtualBox gồm các bước tiến hành được mô tả dưới đây.

Bước 1: Khởi động Hệ thống PX4 và Gazebo: Bước đầu tiên là khởi động bộ mô phỏng PX4 cùng với engine vật lý Gazebo. Từ thư mục PX4-Autopilot, ta thực hiện lệnh: make px4_sitl gz_x500. Lệnh này sẽ:

- Biên dịch mã nguồn PX4 cho chế độ SITL (Software In The Loop) (biểu diễn ở Hình 12).
- Khởi động Gazebo với mô hình drone x500 (model iris) trong môi trường empty.world (biểu diễn ở Hình 13).
- Tạo các kết nối MAVLink trên các cổng mặc định.

Sau khi lệnh chạy thành công, Gazebo sẽ hiển thị cửa sổ với drone được đặt trên mặt đất, sẵn sàng cho các lệnh điều khiển từ các ứng dụng khác.

```

miyamura@Ubuntu:~/PX4-Autopilot
miyamura@Ubuntu:~/PX4-Autopilot$ make px4_sitl gz_x500
[0/1] cd /home/miyamura/PX4-Autopilot/...topilot/build/px4_sitl_default/bin/px4

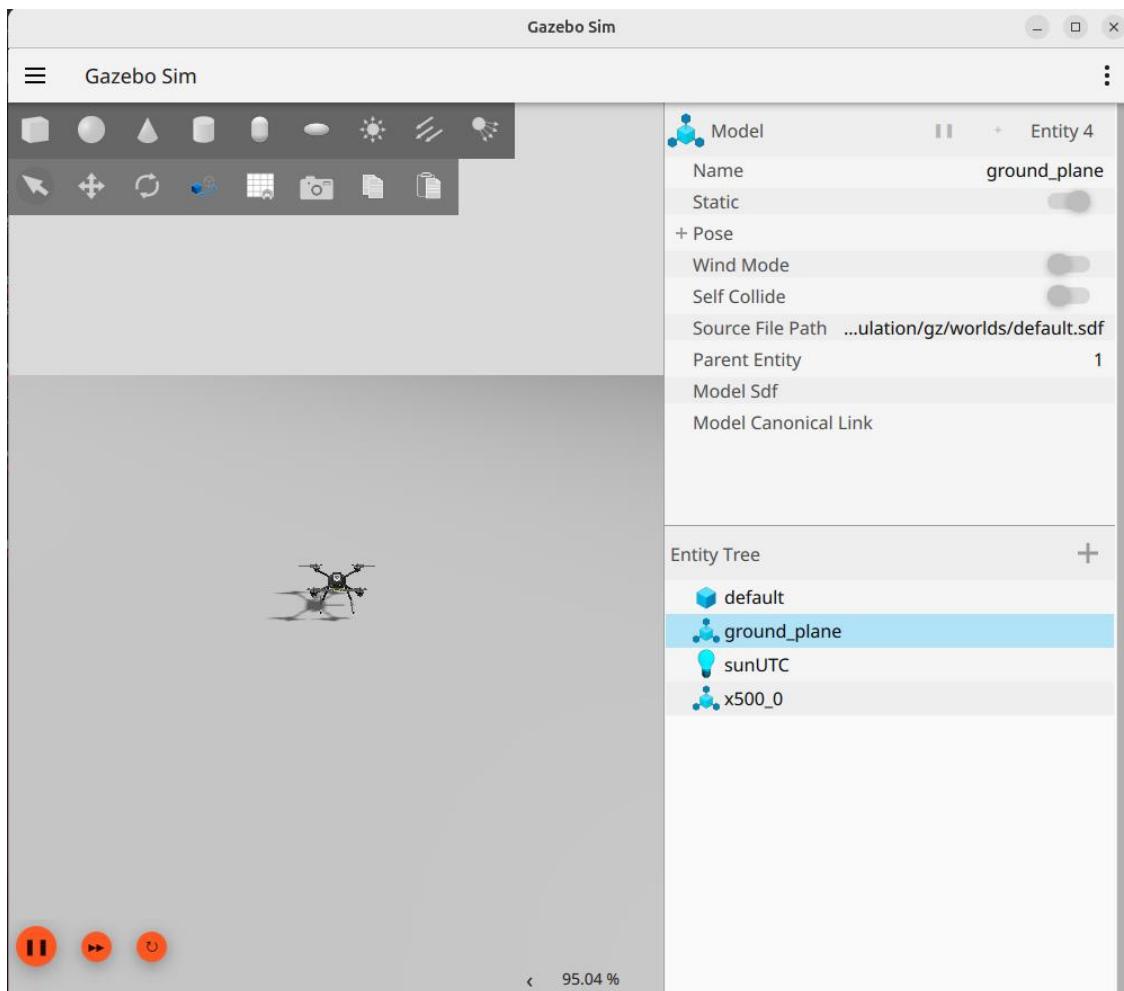
|__\ \ \ \ / / | / | |
| \ / \ \ / / | / |
| __/ / \ \ / / | / |
| | / ^ \ \ \ \ | / |
\_| \ \ \ \ \ \ \ \_| / |

px4 starting.

INFO [px4] startup script: /bin/sh etc/init.d-posix/rcS 0
INFO [init] found model autostart file as SYS_AUTOSTART=4001
INFO [param] selected parameter default file parameters.bson
INFO [param] importing from 'parameters.bson'
INFO [parameters] BSON document size 463 bytes, decoded 463 bytes (INT32:17, FLOAT:6)
INFO [param] selected parameter backup file parameters_backup.bson
INFO [dataman] data manager file './dataman' size is 1208528 bytes
INFO [init] Gazebo simulator 8.10.0
INFO [init] Starting gazebo with world: /home/miyamura/PX4-Autopilot/Tools/simulation/gz/worlds/default.sdf

```

Hình 12: Giao diện PX4



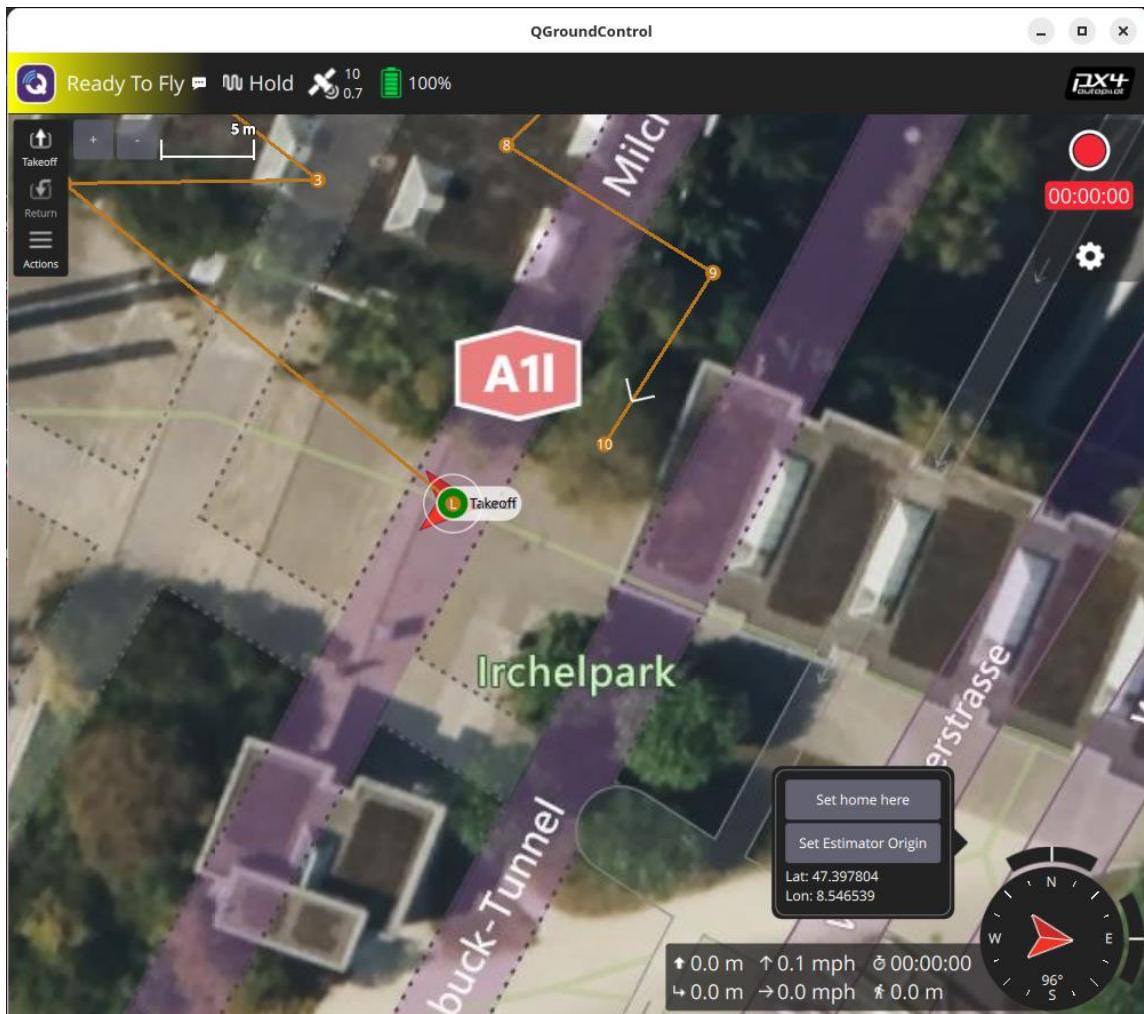
Hình 13: Giao diện Gazebo

Bước 2: Khởi động QGroundControl: Trong một terminal riêng, khởi động ứng dụng QGroundControl bằng lệnh: ./QGroundControl-x86_64.AppImage (biểu diễn ở Hình 14). QGroundControl sẽ tự động kết nối với PX4. Sau khi kết nối thành công, giao diện sẽ hiển thị như ở Hình 15 gồm:

- Trạng thái drone (Ready To Fly, pin, GPS).
- Vị trí drone trên bản đồ.
- Các thông số bay như độ cao, vận tốc, hướng.

```
miyamura@Ubuntu:~/PX4-Autopilot$ cd  
miyamura@Ubuntu:~/PX4-Autopilot$ ./QGroundControl-x86_64.AppImage  
0.135 - debug: Settings location "/home/miyamura/.config/QGroundControl/QGroundControl.ini" Is writable?: true (qgc.qgcapplication:unknown:0)  
0.161 - debug: Filter rules "*Log.debug=false\nqgc.*.debug=false\nqgc.video manager.videoreceiver.gstremmer.api.debug=true\nqtc.qml.connections=false" (qgc.utilities.qgcloggingcategory:unknown:0)  
14.903 - critical: Error loading text-to-speech plug-in "speechd" (default:unknown:0)
```

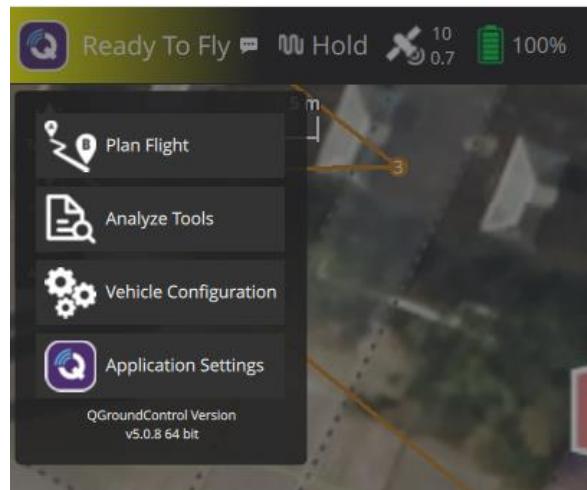
Hình 14: Khởi động QGroundControl



Hình 15: Giao diện QGroundControl

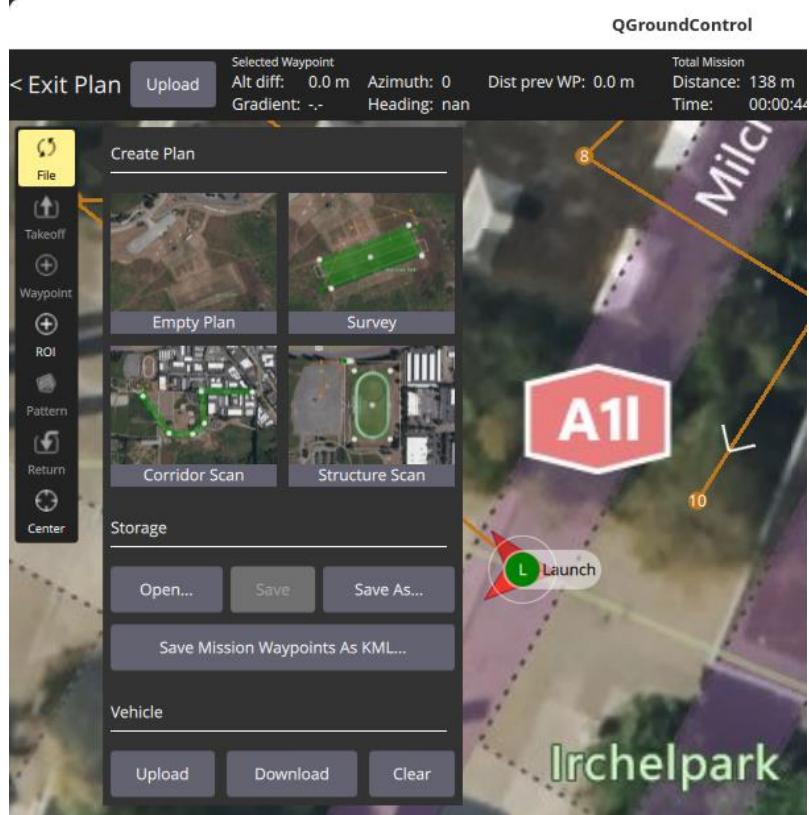
Bước 3: Tạo kế hoạch bay:

Để tạo kế hoạch bay cho Drone thì tại màn hình chính của QGroundControl, nhấp vào menu "Plan Flight" (ký hiệu như ở hình 16).

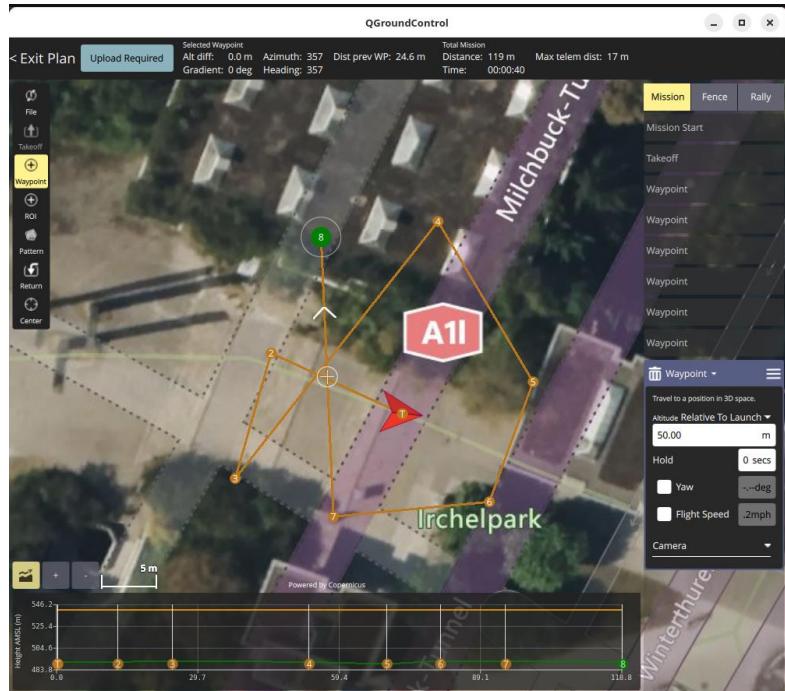


Hình 16: Plan Flight

Tiếp đó, tại thanh menu của “Plan Flight” chọn “File” rồi chọn “Empty Plan” để bắt đầu một kế hoạch mới (biểu diễn ở Hình 17). Điều này sẽ mở giao diện lập kế hoạch với bản đồ trống. Sau đó, ta thêm các waypoint bằng cách nhấp vào nút “Waypoint” ở thanh menu bên trái rồi nhấn vào vị trí trên bản đồ để đặt các “waypoint” tạo ra một chuỗi các điểm giao hàng được đánh số theo thứ tự đặt (biểu diễn ở Hình 18).



Hình 17: Bảng menu “File” của “Plan Flight”



Hình 18: Các waypoint được tạo

Bước 4: Lưu kế hoạch bay:

Sau khi tạo xong các waypoint cần thiết, nhấn vào nút “Upload Required” ở góc trên bên trái màn hình. Lưu kế hoạch bay dưới dạng file .plan bằng cách nhấp vào “File” chọn “Save As” và chọn thư mục lưu trữ, đặt tên file rồi nhấn “Save”. Kết quả sẽ là một file .plan dạng JSON có nội dung là lộ trình bay vừa tạo.

Bước 5: Cấu hình đường dẫn file trong chương trình:

Để Drone có thể bay đúng lộ trình mong muốn, chúng ta cần cập nhật đường dẫn tới file .plan trong chương trình như ở Hình 19.

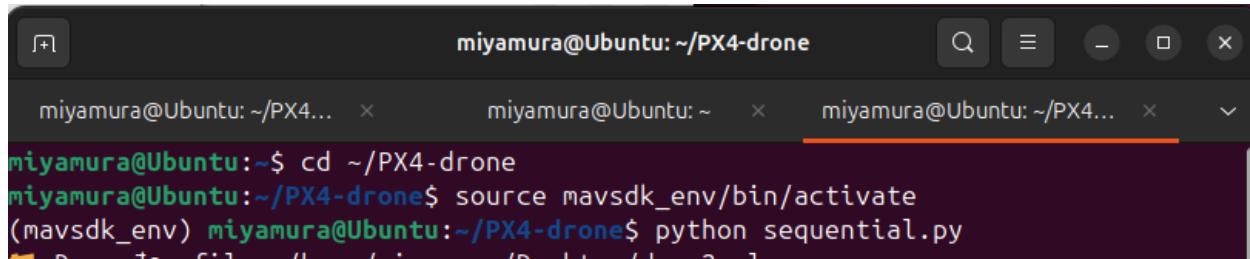
```
async def main(): 1 usage
    delivery = DroneDeliverySequential()
    plan_file = "/home/miyamura/Desktop/demo2.plan"
    delivery.load_mission_from_qgc(plan_file)

    if delivery.delivery_points:
        print("\n" + "="*70)
        await delivery.execute_delivery_mission()
    else:
        print("\n✖ Không có điểm giao hàng. Vui lòng kiểm tra file .plan")
```

Hình 19: Cập nhật đường dẫn file

Bước 6: Chạy chương trình điều khiển:

Trong một terminal mới, kích hoạt môi trường MAVSDK và chạy chương trình Python như trong Hình 20.



```
miyamura@Ubuntu:~/PX4-drone$ cd ~/PX4-drone
miyamura@Ubuntu:~/PX4-drone$ source mavsdk_env/bin/activate
(mavsdk_env) miyamura@Ubuntu:~/PX4-drone$ python sequential.py
```

Hình 20: Chạy chương trình Python bằng MavSDK

Chương trình sẽ kết nối với PX4 thông qua MAVLink, tải các waypoint từ file .plan, thực thi kế hoạch bay, điều khiển drone tới từng điểm giao hàng theo quãng đường tạo ra từ từng thuật toán. Sau đó, in ra các thông tin trạng thái như: vị trí hiện tại, tốc độ, độ cao, trạng thái nhiệm vụ. Đồng thời, trong QGroundControl, ta có thể theo dõi chuyển động của drone trên bản đồ và xem các thống kê bay theo thời gian thực. Ở báo cáo, chúng ta sẽ xây dựng các waypoint có cấu trúc như Hình 21 để có thể mô phỏng dùng với các thuật toán.



Hình 21: Cấu trúc Waypoint mô phỏng

4.2. Demo Hệ thống mô phỏng Drone giao hàng với thuật toán Sequential

Khi chạy thuật toán Sequential trên plan demo thì Drone có thể hoàn thành đi được hết các waypoint và quay về vị trí ban đầu. Kết quả thực thi cụ thể được thể hiện trong Hình 22 và Bảng 6.

Bảng 6: Kết quả mô phỏng Drone với thuật toán Sequential

Thông số	Kết quả
Lộ trình Drone	$T \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow T$
Quãng đường	101.9 m
Thời gian	129.7s (2.16p)
Số điểm giao hàng	5
Vận tốc trung bình	0.79 m/s

Nhận xét: Drone hoàn thành đi hết tới tất cả các waypoint và quay trở về với lộ trình đi đúng như nội dung thuật toán Sequential. Tuy nhiên, do thứ tự các waypoint không được tối ưu cho Sequential nên Drone tốn khá nhiều thời gian và phải đi quãng đường dài hơn nhiều so với cần thiết để có thể hoàn thành chuyến bay. Qua đó, cho thấy tầm quan trọng trong việc lựa chọn thứ tự các waypoint khi quyết định sử dụng Sequential trong việc giao hàng với Drone.

```
(navsdk_env) miyamura@Ubuntu:~/PX4-drone$ python sequential.py
👉 Đang đọc file: /home/miyamura/Desktop/mo2.plan
✓ Đã load 5 điểm giao hàng:
  • Điểm 2: Lat=47.397973, Lon=8.546072, Alt=50.0m
  • Điểm 3: Lat=47.397974, Lon=8.546298, Alt=50.0m
  • Điểm 4: Lat=47.398019, Lon=8.546072, Alt=50.0m
  • Điểm 5: Lat=47.397977, Lon=8.546368, Alt=50.0m
  • Điểm 6: Lat=47.398062, Lon=8.546066, Alt=50.0m

=====
👉 Đang kết nối với drone...
Connection using udp:// is deprecated, please use udpin:// or udpout:// (cli_arg.cpp:28)
Vehicle type changed (new type: 2, old type: 0) (system_impl.cpp:323)
⏳ Đợi drone sẵn sàng...
✓ Đã kết nối với drone!
📍 Lấy vị trí home...
🏠 Home: Lat=47.397971, Lon=8.546167, Alt=0.3m

🌐 Bay theo thứ tự waypoint đã chọn (Sequential)...
📋 Route giao hàng (5 điểm):
  1. Điểm 2: 7.1m từ điểm trước
  2. Điểm 3: 17.0m từ điểm trước
  3. Điểm 4: 17.8m từ điểm trước
  4. Điểm 5: 22.7m từ điểm trước
  5. Điểm 6: 24.6m từ điểm trước
Về home: 12.7m
💡 Tổng quãng đường ước tính: 102.0m
🔧 Đang arm drone...
🚀 Cất cánh lên độ cao 50m...
✓ Đã cất cánh
```

Hình 22.1. Kết quả chạy thuật toán Sequential

```

 Bắt đầu giao hàng...
--- Điểm 1/5 ---
● Bay đến: Lat=47.397973, Lon=8.546072, Alt=50.0m (khoảng cách: 7.1m)
Received ack for not-existing command: 176! Ignoring... (mavlink_command_sender.cpp:304)
✓ Đã đến vị trí (còn cách 0.1m) - Thời gian: 24.6s, Quãng đường: 7.1m
📦 Đang giao hàng tại Điểm 2...
✓ Đã giao hàng tại Điểm 2

--- Điểm 2/5 ---
● Bay đến: Lat=47.397974, Lon=8.546298, Alt=50.0m (khoảng cách: 17.0m)
✓ Đã đến vị trí (còn cách 0.1m) - Thời gian: 12.5s, Quãng đường: 17.0m
📦 Đang giao hàng tại Điểm 3...
✓ Đã giao hàng tại Điểm 3

--- Điểm 3/5 ---
● Bay đến: Lat=47.398019, Lon=8.546072, Alt=50.0m (khoảng cách: 17.8m)
✓ Đã đến vị trí (còn cách 0.1m) - Thời gian: 12.3s, Quãng đường: 17.8m
📦 Đang giao hàng tại Điểm 4...
✓ Đã giao hàng tại Điểm 4

--- Điểm 4/5 ---
● Bay đến: Lat=47.397977, Lon=8.546368, Alt=50.0m (khoảng cách: 22.8m)
✓ Đã đến vị trí (còn cách 0.1m) - Thời gian: 13.4s, Quãng đường: 22.8m
📦 Đang giao hàng tại Điểm 5...
✓ Đã giao hàng tại Điểm 5

--- Điểm 5/5 ---
● Bay đến: Lat=47.398062, Lon=8.546066, Alt=50.0m (khoảng cách: 24.6m)
✓ Đã đến vị trí (còn cách 0.1m) - Thời gian: 15.4s, Quãng đường: 24.6m
📦 Đang giao hàng tại Điểm 6...
✓ Đã giao hàng tại Điểm 6

 Quay về điểm xuất phát...
● Bay đến: Lat=47.397971, Lon=8.546167, Alt=50.3m (khoảng cách: 12.7m)
✓ Đã đến vị trí (còn cách 0.1m) - Thời gian: 11.4s, Quãng đường: 12.7m
 Hạ cánh...

```

Hình 22.2. Kết quả chạy thuật toán Sequential

```

=====
 THỐNG KÊ CHUYẾN BAY - SEQUENTIAL (THEO THỨ TỰ)
=====

⌚ Tổng thời gian bay: 129.7s (2.16 phút)
👉 Tổng quãng đường: 101.9m
🕒 Số điểm giao hàng: 5
⚡ Tốc độ trung bình: 0.79 m/s

⌚ Chi tiết từng đoạn:
Đoạn 1 (đến Điểm 2): 7.1m - 24.6s - 0.29m/s
Đoạn 2 (đến Điểm 3): 17.0m - 12.5s - 1.36m/s
Đoạn 3 (đến Điểm 4): 17.8m - 12.3s - 1.45m/s
Đoạn 4 (đến Điểm 5): 22.8m - 13.4s - 1.70m/s
Đoạn 5 (đến Điểm 6): 24.6m - 15.4s - 1.60m/s
Đoạn 6 (về Home): 12.7m - 11.4s - 1.11m/s

=====

✓ Hoàn thành mission giao hàng!
=====
```

Hình 22.3. Kết quả chạy thuật toán Sequential

4.3. Demo Hệ thống mô phỏng Drone giao hàng với thuật toán Nearest Neighbor

Khi chạy thuật toán Nearest Neighbor trên plan demo thì Drone có thể hoàn thành đi được hết các waypoint và quay về vị trí ban đầu. Kết quả thực thi cụ thể được thể hiện trong Hình 23 và Bảng 7.

Bảng 7: Kết quả mô phỏng Drone với thuật toán Nearest Neighbor

Thông số	Kết quả
Lộ trình Drone	T → 2 → 4 → 6 → 3 → 5 → T
Quãng đường	57.5 m
Thời gian	119.0s (1.98p)
Số điểm giao hàng	5
Vận tốc trung bình	0.48 m/s

Nhận xét: Drone hoàn thành đi hết tới tất cả các waypoint và quay trở về với lộ trình đi đúng như nội dung thuật toán Nearest Neighbor ưu tiên đi tới điểm gần nhất chưa đến. Quãng đường và thời gian được cải thiện đáng kể. Thời gian tính toán lộ trình tương đối ngắn cho thấy sự phù hợp khi ứng dụng trong thực tế. Tuy nhiên, thuật toán vẫn chưa thể hiện được lộ trình tối ưu nhất.

```
(mavsdk_env) miyamura@Ubuntu:~/PX4-drone$ python nearest_neighbor.py
👉 Đang đọc file: /home/miyamura/Desktop/demo2.plan
✓ Đã load 5 điểm giao hàng:
  • Điểm 2: Lat=47.397973, Lon=8.546072, Alt=50.0m
  • Điểm 3: Lat=47.397974, Lon=8.546298, Alt=50.0m
  • Điểm 4: Lat=47.398019, Lon=8.546072, Alt=50.0m
  • Điểm 5: Lat=47.397977, Lon=8.546368, Alt=50.0m
  • Điểm 6: Lat=47.398062, Lon=8.546066, Alt=50.0m

=====
👉 Đang kết nối với drone...
Connection using udp:// is deprecated, please use udpin:// or udppout:// (cli_arg.cpp:28)
Vehicle type changed (new type: 2, old type: 0) (system_impl.cpp:323)
⏳ Đợi drone sẵn sàng...
✓ Đã kết nối với drone!
📍 Lấy vị trí home...
🏡 Home: Lat=47.397971, Lon=8.546164, Alt=0.2m

💡 Tính toán route tối ưu (Nearest Neighbor)...
👉 Route giao hàng (5 điểm):
  1. Điểm 2: 6.9m từ điểm trước
  2. Điểm 4: 5.1m từ điểm trước
  3. Điểm 6: 4.9m từ điểm trước
  4. Điểm 3: 20.1m từ điểm trước
  5. Điểm 5: 5.2m từ điểm trước
Về home: 15.4m
👉 Tổng quãng đường ước tính: 57.6m
👉 Đang arm drone...
👉 Cất cánh lên độ cao 50m...
✓ Đã cất cánh
```

Hình 23.1. Kết quả chạy thuật toán Nearest Neighbor

```

⚡ Bắt đầu giao hàng...

--- Điểm 1/5 ---
🔴 Bay đến: Lat=47.397973, Lon=8.546072, Alt=50.0m (khoảng cách: 6.9m)
Received ack for not-existing command: 176! Ignoring... (mavlink_command_sender.cpp:304)
✅ Đã đến vị trí (còn cách 0.1m) - Thời gian: 24.5s, Quãng đường: 6.9m
📦 Đang giao hàng tại Điểm 2...
✅ Đã giao hàng tại Điểm 2

--- Điểm 2/5 ---
🔴 Bay đến: Lat=47.398019, Lon=8.546072, Alt=50.0m (khoảng cách: 5.1m)
✅ Đã đến vị trí (còn cách 0.1m) - Thời gian: 11.4s, Quãng đường: 5.1m
📦 Đang giao hàng tại Điểm 4...
✅ Đã giao hàng tại Điểm 4

--- Điểm 3/5 ---
🔴 Bay đến: Lat=47.398062, Lon=8.546066, Alt=50.0m (khoảng cách: 4.9m)
✅ Đã đến vị trí (còn cách 0.1m) - Thời gian: 10.2s, Quãng đường: 4.9m
📦 Đang giao hàng tại Điểm 6...
✅ Đã giao hàng tại Điểm 6

--- Điểm 4/5 ---
🔴 Bay đến: Lat=47.397974, Lon=8.546298, Alt=50.0m (khoảng cách: 20.1m)
✅ Đã đến vị trí (còn cách 0.1m) - Thời gian: 12.4s, Quãng đường: 20.1m
📦 Đang giao hàng tại Điểm 3...
✅ Đã giao hàng tại Điểm 3

--- Điểm 5/5 ---
🔴 Bay đến: Lat=47.397977, Lon=8.546368, Alt=50.0m (khoảng cách: 5.2m)
✅ Đã đến vị trí (còn cách 0.0m) - Thời gian: 8.2s, Quãng đường: 5.2m
📦 Đang giao hàng tại Điểm 5...
✅ Đã giao hàng tại Điểm 5

🏠 Quay về điểm xuất phát...
🔴 Bay đến: Lat=47.397971, Lon=8.546164, Alt=50.2m (khoảng cách: 15.3m)
✅ Đã đến vị trí (còn cách 0.1m) - Thời gian: 12.3s, Quãng đường: 15.3m
⚡ Hạ cánh...

```

Hình 23.2. Kết quả chạy thuật toán Nearest Neighbor

```

=====
📊 THỐNG KÊ CHUYẾN BAY - NEAREST NEIGHBOR
=====

⌚ Tổng thời gian bay: 119.0s (1.98 phút)
👉 Tổng quãng đường: 57.5m
⌚ Số điểm giao hàng: 5
⚡ Tốc độ trung bình: 0.48 m/s

⌚ Chi tiết từng đoạn:
Đoạn 1 (đến Điểm 2): 6.9m - 24.5s - 0.28m/s
Đoạn 2 (đến Điểm 4): 5.1m - 11.4s - 0.45m/s
Đoạn 3 (đến Điểm 6): 4.9m - 10.2s - 0.48m/s
Đoạn 4 (đến Điểm 3): 20.1m - 12.4s - 1.63m/s
Đoạn 5 (đến Điểm 5): 5.2m - 8.2s - 0.64m/s
Đoạn 6 (về Home): 15.3m - 12.3s - 1.25m/s

=====

✅ Hoàn thành mission giao hàng!

```

Hình 23.3. Kết quả chạy thuật toán Nearest Neighbor

4.4. Demo Hệ thống mô phỏng Drone giao hàng với thuật toán 2-Opt

Khi chạy thuật toán 2-Opt trên plan demo thì Drone có thể hoàn thành đi được hết các waypoint và quay về vị trí ban đầu. Kết quả thực thi cụ thể được thể hiện trong Hình 24 và Bảng 8.

Bảng 8: Kết quả mô phỏng Drone với thuật toán 2-Opt

Thông số	Kết quả
Lộ trình Drone	T → 2 → 4 → 6 → 5 → 3 → T
Quãng đường	56.8 m
Thời gian	89.6s (1.49p)
Số điểm giao hàng	5
Vận tốc trung bình	0.63 m/s

Nhận xét: Drone hoàn thành đi hết tới tất cả các waypoint và quay trở về với lộ trình đi đúng như nội dung thuật toán 2-Opt với lộ trình bay đã được cải thiện so với Nearest Neighbor. Quãng đường và thời gian được cải thiện nhất. Thời gian tính toán lộ trình lại dài hơn. Có thể dùng khi có khoảng thời gian để tính toán và muốn tiết kiệm năng lượng.

```
(mavsdk_env) miyamura@Ubuntu:~/PX4-drone$ python 2opt.py
📁 Đang đọc file: /home/miyamura/Desktop/demo3.plan
✅ Đã load 5 điểm giao hàng:
  • Điểm 2: Lat=47.397973, Lon=8.546072, Alt=50.0m
  • Điểm 3: Lat=47.397974, Lon=8.546298, Alt=50.0m
  • Điểm 4: Lat=47.398019, Lon=8.546072, Alt=50.0m
  • Điểm 5: Lat=47.397977, Lon=8.546368, Alt=50.0m
  • Điểm 6: Lat=47.398062, Lon=8.546066, Alt=50.0m

=====
📡 Đang kết nối với drone...
Connection using udp:// is deprecated, please use udpin:// or udppout:// (cli_arg.cpp:28)
Vehicle type changed (new type: 2, old type: 0) (system_impl.cpp:323)
⌛ Đợi drone sẵn sàng...
✅ Đã kết nối với drone!
📍 Lấy vị trí home...
🏡 Home: Lat=47.397971, Lon=8.546164, Alt=0.2m
🌐 Đang tối ưu route bằng 2-opt...

📋 Route giao hàng tối ưu (5 điểm):
  1. Điểm 2: 6.9m từ điểm trước
  2. Điểm 4: 5.1m từ điểm trước
  3. Điểm 6: 4.9m từ điểm trước
  4. Điểm 5: 24.6m từ điểm trước
  5. Điểm 3: 5.2m từ điểm trước
Về home: 10.1m
💡 Tổng quãng đường ước tính: 56.9m
🔧 Đang arm drone...
🚀 Cất cánh lên độ cao 50m...
✅ Đã cất cánh
```

Hình 24.1. Kết quả chạy thuật toán 2-Opt

```

 Bắt đầu giao hàng...
--- Điểm 1/5 ---
● Bay đến: Lat=47.397973, Lon=8.546072, Alt=50.0m (khoảng cách: 6.9m)
Received ack for not-existing command: 176! Ignoring... (mavlink_command_sender.cpp:304)
✓ Đã đến vị trí (còn cách 0.7m) - Thời gian: 18.4s, Quãng đường: 6.9m
📦 Đang giao hàng tại Điểm 2...
✓ Đã giao hàng tại Điểm 2

--- Điểm 2/5 ---
↑ Bay đến: Lat=47.398019, Lon=8.546072, Alt=50.0m (khoảng cách: 5.1m)
✓ Đã đến vị trí (còn cách 0.9m) - Thời gian: 4.2s, Quãng đường: 5.1m
📦 Đang giao hàng tại Điểm 4...
✓ Đã giao hàng tại Điểm 4

--- Điểm 3/5 ---
● Bay đến: Lat=47.398062, Lon=8.546066, Alt=50.0m (khoảng cách: 4.9m)
✓ Đã đến vị trí (còn cách 1.0m) - Thời gian: 4.2s, Quãng đường: 4.9m
📦 Đang giao hàng tại Điểm 6...
✓ Đã giao hàng tại Điểm 6

--- Điểm 4/5 ---
↑ Bay đến: Lat=47.397977, Lon=8.546368, Alt=50.0m (khoảng cách: 24.6m)
✓ Đã đến vị trí (còn cách 0.8m) - Thời gian: 11.3s, Quãng đường: 24.6m
📦 Đang giao hàng tại Điểm 5...
✓ Đã giao hàng tại Điểm 5

--- Điểm 5/5 ---
● Bay đến: Lat=47.397974, Lon=8.546298, Alt=50.0m (khoảng cách: 5.2m)
✓ Đã đến vị trí (còn cách 0.8m) - Thời gian: 4.1s, Quãng đường: 5.2m
📦 Đang giao hàng tại Điểm 3...
✓ Đã giao hàng tại Điểm 3

🏠 Quay về điểm xuất phát...
↑ Bay đến: Lat=47.397971, Lon=8.546164, Alt=50.2m (khoảng cách: 10.1m)
✓ Đã đến vị trí (còn cách 0.7m) - Thời gian: 7.2s, Quãng đường: 10.1m
✈ Hạ cánh...

```

Hình 24.2. Kết quả chạy thuật toán 2-Opt

```

=====
 THỐNG KÊ CHUYẾN BAY - 2-OPT OPTIMIZATION
=====
⌚ Tổng thời gian bay: 89.6s (1.49 phút)
⚡ Tổng quãng đường: 56.8m
📍 Số điểm giao hàng: 5
⚡ Tốc độ trung bình: 0.63 m/s
🚀 Cải thiện so với Nearest Neighbor: 0.7m

📍 Chi tiết từng đoạn:
Đoạn 1 (đến Điểm 2): 6.9m - 18.4s - 0.38m/s
Đoạn 2 (đến Điểm 4): 5.1m - 4.2s - 1.22m/s
Đoạn 3 (đến Điểm 6): 4.9m - 4.2s - 1.17m/s
Đoạn 4 (đến Điểm 5): 24.6m - 11.3s - 2.18m/s
Đoạn 5 (đến Điểm 3): 5.2m - 4.1s - 1.27m/s
Đoạn 6 (về Home): 10.1m - 7.2s - 1.39m/s
=====

✓ Hoàn thành mission giao hàng!

```

Hình 24.3. Kết quả chạy thuật toán 2-Opt

4.5. Nhận xét chung

Từ kết quả thống kê chuyến bay của từng thuật toán, ta có thể lập bảng so sánh hiệu suất của 3 thuật toán với nhau như được trình bày trong Bảng 9.

Bảng 9: Bảng so sánh hiệu suất giữa 3 thuật toán

Thông số	Kết quả		
	Sequential	Nearest Neighbor	2-Opt
Lộ trình Drone	$T \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow T$	$T \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 3 \rightarrow 5 \rightarrow T$	$T \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 5 \rightarrow 3 \rightarrow T$
Quãng đường	101.9 m	57.5 m	56.8 m
Thời gian	129.7s (2.16p)	119.0s (1.98p)	89.6s (1.49p)
Số điểm giao hàng	5	5	5
Vận tốc trung bình	0.79 m/s	0.48 m/s	0.63 m/s

Từ bảng so sánh trên, ta có một số nhận xét như sau:

- Sequential cho quãng đường (101.9 mét) và thời gian (129.7) lâu nhất trong ba thuật toán. Điều này là dễ hiểu vì Sequential chỉ tuân theo thứ tự cố định mà không có bất kỳ tối ưu hóa nào. Drone buộc phải bay tới các điểm xa nhau liên tiếp tạo ra quãng đường lãng phí.
- Nearest Neighbor giảm quãng đường xuống 57.5 mét, tương ứng với mức tiết kiệm 43.6% so với Sequential. Sự cải thiện này đáng kể và phản ánh hiệu quả thuật toán, nơi drone lựa chọn điểm gần nhất để tránh những chuyến bay dài liên tiếp không cần thiết. Tuy nhiên, thời gian lại giảm không quá nhiều, có thể trong các đoạn bay ngắn trong Nearest Neighbor có thể yêu cầu thời gian điều chỉnh hướng lâu hơn.
- 2-Opt tiếp tục cải thiện với quãng đường 56.8 mét, chỉ tiết kiệm thêm 0.7 mét so với Nearest Neighbor (1.2% cải thiện) cùng với lượng thời gian thấp hơn hẳn so với 2 thuật toán trước. Mặc dù mức cải thiện này không quá lớn, nhưng nó cho thấy rằng 2-Opt, với khả năng tối ưu hóa cấu trúc đường đi, có thể tìm ra những điều chỉnh tinh tế mà Nearest Neighbor bỏ lỡ.

5. Tổng kết

Dự án đã thành công trong việc triển khai mô phỏng drone với ba thuật toán lập kế hoạch đường đi: Sequential, Nearest Neighbor, và 2-Opt. Kết quả thực nghiệm cho thấy sự khác biệt rõ rệt chứng minh rằng thuật toán lập kế hoạch đóng vai trò quyết định trong tối ưu hóa hiệu suất drone.

Tuy nhiên, dự án vẫn tồn tại hạn chế: chỉ test trên mô phỏng chứ chưa trên drone thực, chỉ với 5 waypoint, và môi trường đơn giản (không gió, vật cản). Để nâng cao giá trị thực tế, các hướng phát triển tiếp theo gồm: triển khai test trên drone thực, thêm các thuật toán nâng cao (Genetic Algorithm, Particle Swarm Optimization), mở rộng số lượng waypoint lên 20-100 điểm, và tích hợp các ràng buộc thực tế như gió, quản lý pin, và bay 3D. Những cải tiến này sẽ giúp dự án tiến tới ứng dụng thực tiễn và trở thành công cụ hữu ích cho nghiên cứu trong lĩnh vực UAV.

Tài liệu tham khảo

[Simulation | PX4 Guide \(main\)](#)

[Gazebo Simulation | PX4 User Guide \(v1.12\)](#)

[mavlink/MAVSDK: API and library for MAVLink compatible systems written in C++17](#)

[QGroundControl Guide \(Daily Builds\) | QGC Guide](#)

[2-opt - Wikipedia](#)