Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Галузь знань 12 – Інформаційні технології

Спеціальність 123 –Комп'ютерна інженерія

на тему «Метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму»

КвРКІ 160113. 20.15 ПЗ

Виконав: студент 2 курсу, група КІ2м-20-1 _____ Ковальчук П.С.
Підпис Ініціали, прізвище

Керівник доктор техн. наук, професор _____ Лисенко С.М.
Науковий ступінь, вчене звання Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КІІС, д.т.н., проф.
Т.О. Говорущенко
__ _____ 2022 р.

Хмельницький, 2022

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень магістр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма освітньо-наукова програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

" 01 " 09 2021 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Ковальчуку Павлу Сергійовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму

Керівник проекту (роботи) Лисенко С.М., д.т.н., професор
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 06.01.2022 р. № 1

2. Строк подання студентом проекту (роботи) на кафедру 03.05.2022 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Характеристика предметної області та постановка задачі

Математична модель скоординованого планування оптимальної траєкторії руху БПЛА

Метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму

Програмно-технічний засіб для керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., професор кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання «_06_»_____09_____2021р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики ДРМ з керівником	05.09.2021	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	05.10.2021	виконано
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	05.11.2021	виконано
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	05.12.2021	виконано
5	Робота над науковою статтею	05.01.2022	виконано
6	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	15.02.2022	виконано
7	Робота над розділом 4 – проектування та розробка ПЗ для вирішення поставленої задачі, експериментальна частина	05.04.2022	виконано
8	Оформлення пояснювальної записки згідно вимог	15.04.2022	виконано
9	Попередній захист ДРМ	18.04.2022	виконано
10	Захист ДРМ на засіданні ЕК	До 10.05.2022	

Студент _____ П.С. Ковальчук
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ С.М. Лисенко
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема дипломної роботи: «Метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму».

Автор роботи: Ковальчук Павло Сергійович

Керівник роботи: д.т.н., професор Лисенко С.М.

Пояснювальна записка: 89 с., 38 рис., 6 табл., 3 дод., 62 джерела.

АСО, БПЛА, ФЕРОМОНИ, КВАДРОТОР, МУРАШИНІ КОЛОНІЇ, MMAS, АСИНХРОННИЙ АЛГОРИТМ

Об’єктом дослідження є процес керування групою безпілотних літальних апаратів в тому числі процес оптимізації методу керування групи БПЛА з метою досягнення кінцевої точки призначення за найкоротший час.

Предметом дослідження є математичні моделі, удосконалений метод та програмно-технічні засоби для реалізації системи та методу керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму.

Метою дипломної роботи є розробка програмно-технічного засобу, методу та системи керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму для підвищення ефективності виконання групових завдань.

Для розв’язання поставлених задач у роботі використовуються методи аналізу даних, системний аналіз, теорії комп’ютерних систем, та аналіз алгоритмів пошуку найкоротшого шляху.

Наукова новизна отриманих результатів:

− набув подальшого розвитку метод суть роботи якого полягає в тому, щоб представити ієрархічну модель прийняття рішень для спільного планування місій безпілотників для місії розвідки шляхом інтеграції розподілу місії та планування маршруту;

− набула подальшого розвитку інформаційна технологія керування групою безпілотних літальних апаратів, модель реалізації якої буде відбуватись без критично-довгих навчань та з відносно очікуваним результатом.

Практична цінність проекту полягає в розробленому програмно-технічному продукті, який дозволяє організувати оптимальну систему керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму та виконання місій цими БПЛА у найкоротший термін та з високою ефективністю.

Під час виконання досліджень було опубліковано статтю на тему «Метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму» в збірнику наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук АПКН-2021». Хмельницький – 2021. С.117-119.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БПЛА – безпілотний літальний апарат

PID – пропорційна інтегральна похідна

LQR – лінійний квадратичний регулятор

ACO – оптимізація діяльності колонії мурах

TSP – задача комівояжера

MAX-MIN – максимального та мінімального значення

MMAS – MAX-MIN Ant System (мурашина система пошуку максимального та мінімального)

ACS – елітарна мурашина система

ETA (Estimated time of arrival) – запланований час прибуття

SDK (Software development kit) – набір із засобів розробки

ВСТУП

Стрімкий науково-технічний розвиток, швидко змінює сучасне життя та модернізує уставлені системи згідно нових технічних розробок, так і БПЛА починають замінювати багато основних застарілих технологій, і починають масово використовуватись.

Сфери застосування БПЛА ростуть з кожним роком та використовуються як у цивільних (доставка, відеозйомка та ін.), так і військових сферах (розвідка, маневри, доставка).

Актуальність роботи полягає в розробці методу та системи керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму, який являє собою високоефективну модель керування БПЛА, із можливістю досягнення високих оцінок у поставлених задачах.

Метою кваліфікаційної роботи магістра є розробка програмно-технічного засобу, методу та системи керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму для підвищення ефективності виконання групових завдань.

Поставлена мета досягається розв’язанням наступних задач:

1. Проведення аналізу та дослідження предметної області та доступних методів рішення задачі.
2. Розроблення математичної моделі скоординованого планування оптимальної траєкторії руху БПЛА.
3. Проектування удосконаленого ACO Max-Min алгоритму для розрахунку оптимального маршруту БПЛА з урахуванням уникнення спливаючих перешкод.
4. Розроблення та дослідження програмно-апаратної реалізації керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму та проведення ряду імітаційних тестів.

Об’єктом дослідження є процес керування групою безпілотних літальних апаратів в тому числі процес оптимізації методу керування групи БПЛА з метою досягнення кінцевої точки призначення за найкоротший час.

Предметом дослідження є математичні моделі, удосконалений метод та програмно-технічні засоби для реалізації системи та методу керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму.

Для розв'язання поставлених задач у роботі використовуються методи аналізу даних, системний аналіз, теорії комп'ютерних систем, та аналіз алгоритмів пошуку найкоротшого шляху.

Наукова новизна отриманих результатів:

− набув подальшого розвитку метод суть роботи якого полягає в тому, щоб представити ієрархічну модель прийняття рішень для спільного планування місій безпілотників для місії розвідки шляхом інтеграції розподілу місії та планування маршруту;

− набула подальшого розвитку інформаційна технологія керування групою безпілотних літальних апаратів, модель реалізації якої буде відбуватись без критично-довгих навчань та з відносно очікуваним результатом.

Практична цінність проекту полягає в розробленому програмно-технічному продукті, який дозволяє організувати оптимальну систему керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму та виконання місій цими БПЛА у найкоротший термін та з високою ефективністю.

Під час виконання досліджень було опубліковано статтю на тему «Метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму» в збірнику наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021» [1].

1 ХАРАКТЕРИСТИКА ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

На сьогодні БПЛА використовуються в багатьох сферах діяльності та має неймовірні перспективи [2]. Серед сфер використання можна виділити

1. Військові – ймовірно, найдавніше, найвідоміше і суперечливе використання безпілотників – у військових. Британські та американські військові почали використовувати найпростіші форми безпілотників на початку 1940-х років, щоб шпигувати за державами Осі. Сучасні БПЛА набагато досконаліші, ніж БПЛА минулих років, оснащені тепловізором, лазерними далекомірами і навіть інструментами для нанесення авіаударів. Найвідомішим військовим безпілотником, який використовується сьогодні, є MQ-9 Reaper. Літак має довжину 36 футів, може літати на висоті 50 000 футів у повітрі непоміченим і оснащений комбінацією ракет і засобів збору розвідувальних даних [3].

2. Доставка – безпілотники для доставки, як правило, є автономними БПЛА, які використовуються для транспортування їжі, пакетів або товарів до вашого під'їзду. Ці літальні апарати відомі як дрони доставки «останньої милі», оскільки вони використовуються для доставки з магазинів або складів поблизу. Роздрібні торговці та продуктові мережі звертаються до дронів як до більш ефективної альтернативи доставки, замість того, щоб покладатися на водіїв доставки з неефективними вантажівками [4]. Ці БПЛА можуть доставити вражаючі 25 кілограм товарів до ваших дверей, не виходячи з дому. Amazon, Walmart, Google, FedEx, UPS та багато інших великих брендів зараз тестують різні версії дронів для доставки.

3. Аварійно рятувальна допомога – іноді просто недостатньо безпечно відправити людей у рятувальну місію через масштаби або серйозність катастрофи. Ось тут на допомогу приходять дрони. У разі перекидання човна або людини, яка тоне, рятівники можуть запустити у воду автономний підводний апарат (АНП), щоб допомогти в порятунку. При лавині БПЛА розгортаються для пошуку тих, хто потрапив у сніг. Авіа виробник Kaman навіть розробив

безпілотний гелікоптер під назвою K-MAX, призначений для перевезення понад 2700 кілограм вантажу. K-MAX вже використовувався в Китаї та Австралії для допомоги в гасінні пожеж.

4. Сільське господарство – безпілотники також виявилися корисними для сільського господарства, надавши фермерам кілька способів оптимізації своїх ферм, щоб максимізувати ефективність і зменшити фізичне навантаження. Проведення польових обстежень, посів на полях, відстеження худоби та оцінка врожайності – все це полегшується завдяки використанню БПЛА, заощаджуючи при цьому дорогоцінний час спеціалістів сільського господарства.

5. Космос – NASA та ВПС США таємно випробовують безпілотні літаки, призначені для космічних подорожей. БПЛА X-37B – це надсекретний безпілотник ВПС, який виглядає як мініатюрний космічний човник. Останні два роки він тихо кружляє навколо Землі, встановивши рекорд найдовшого польоту з безпілотного літака (більше 719 днів) [5]. Незважаючи на невизначеність, ВПС заявили, що «основні цілі X-37B мають подвійне значення: технології багаторазового використання космічних апаратів для майбутнього Америки в космосі та експлуатаційні експерименти, які можна повернути й дослідити на Землі». Схоже, що дрони стали пріоритетом, коли мова заходить про майбутнє дослідження космосу та інновацій.

6. Дика природа та охорона історії – БПЛА є дешевшою та ефективнішою альтернативою збереженню дикої природи. Відстеження популяцій дикої природи майже неможливо, якщо людина знаходиться на землі. Наявність «око в небі» дозволяє фахівцям з охорони дикої природи відстежувати бродячі групи тварин, починаючи від орангутанів на Борнео і закінчуючи бізонами на Великих рівнинах, щоб отримати краще уявлення про здоров’я їхніх видів та екосистем. Дрони, що охороняють, також є ідеальними інструментами для боротьби з браконьєрством в Азії та Африці. У всьому світі дрони також використовуються для лісовідновлення. Ці безпілотні літальні апарати нишпорять лісові підстилки лісів, знищених пожежею, і скидають насіннєві судини, наповнені насінням, добривами та поживними речовинами, які допоможуть дереву вирости з попелу. З початку 1990-х років було близько 300

мільйонів акрів вирубаних лісів [6]. Те, що людині знадобилося б приблизно 300 років для відновлення лісу, можна ефективніше виконати за допомогою технології БПЛА, що висаджують насіння. Нарешті, безпілотники стають важливими в зусиллях зі збереження історії. БПЛА використовуються для відображення 3D-відображення історичних місць, таких як Чорнобиль, стародавні грецькі місця Ефесу, Туреччини та єврейські кладовища по всій Європі.

7. Ліки – як доставити медичні засоби людям у важкодоступних районах? Який інструмент можна використовувати для доставки органів пацієнтам із трансплантацією? БПЛА – це відповідь на обидва ці питання. Наразі безпілотні літальні апарати використовуються для доставки невідкладних медичних матеріалів та вантажів до населених пунктів у сільській місцевості Аляски. Замість того, щоб покладатися на собачі упряжки, снігоходи або машини швидкої допомоги, які не можуть впоратися зі снігом, жителі Аляски покладаються на дрони, щоб швидко отримати рятівні медичні засоби. Також використовують дрони для доставки донорських органів пацієнтам із трансплантацією. Зовсім нещодавно був випадок, коли спеціально виготовлений безпілотник доставив нирку з однієї лікарні в Меріленд за п’ять хвилин [7]. Це може зменшити тривожно повільну швидкість, з якою зазвичай надходять пожертвування (якщо вони надходять взагалі). Зазвичай органи доставляють чартерними або комерційними рейсами. Затримки та пропуски в судженнях викликають небезпечні затримки на дві години або більше для 4% усіх доставок органів. БПЛА можуть значно скоротити час, пропонуючи більш безпечний і безпечний спосіб транспортування органів.

8. Фотографія – безпілотники є благом для фотографів, які використовують безпілотні апарати, щоб робити великі аерофотознімки. Ви коли-небудь замислювалися про те, як це дивитися на улюблене місто, пляж чи будівлю з висоти пташиного польоту? Існують БПЛА, створені спеціально для фотозйомки, які пропонують новий спосіб фотографувати деякі з ваших улюблених місць зверху.

Ця різноманітність пояснюється тим, що БПЛА мають досить великий функціонал та можуть легко модернізуватись в залежності від сфери використання. Ця технологічність характеризується явними ознаками, які визначають види БПЛА. Можна охарактеризувати БПЛА за різними їх характеристиками та сферою використання, загальний поділ можна переглянути у таблиці 1.1.

Таблиця 1.1 – Типи безпілотних літальних апаратів

Ознака	Види
За масштабом завдань, що вирішуються	Тактичні Оперативно-тактичні Оперативно-стратегічні
За масою	Малорозмірнім Середньорозмірні Великорозмірні Важкі
За радіусом дії	Ближнього радіусу Малого радіусу Середнього радіусу Дальнього радіусу Великої дальності
За кількістю використань	Одноразові Багаторазові
За тривалістю польоту	Малої тривалості Середньої тривалості Великої тривалості

Продовження таблиці 1.1 – Типи безпілотних літальних апаратів

За типом паливного бака	Базові Базово-резервні
За паливною системою	Монозаправні Полізаправні (наземна, платформна (морська, бортова))
За типом літального апарату	За літаковою аеродинамічною схемою За гелікоптерною аеродинамічною схемою Легші за повітря
За практичною стелею польоту	Маловисокі Середньовисокі Висотні Стратосферні
За типом	За підйомом (аеродромні, запускні, палубні, водні, ручні, нетипово підйомні, мультипідйомні) За посадкою (аеродромні, точкові, палубні, водні, безпосадкові, нетипово посадкові, мультипосадкові)
За напрямком	За напрямком підйому (горизонтальні, вертикальні, мультипідйомні) За напрямком посадки (горизонтальні, вертикальні, парашутні, мачтові, безпосадкові, мультиспускові)

Кінець таблиці 1.1 – Типи безпілотних літальних апаратів

За базуванням	Наземні Морські Космічні
За використанням	Військові Цивільні (державні, приватні, комерційні) Антитерористичні
За типом крила	Фіксовані Плаваючі
За правилами польоту	Візуальні Приладові Візуально-приладові
За типом системи керування	Дистанційно пілотовані Дистанційно керовані Автоматичні Дистанційно керовані авіаційною системою

Детальніше можна розглянути типи БПЛА згідно їх конструкції, загалом можна виділити наступні типи дронів:

1. Багатороторний (рисунок 1.1) – найпростішим і найдешевшим варіантом для отримання «око в небі», а оскільки вони дають вам чудовий контроль над положенням і кадруванням, вони ідеально підходять для роботи з аерофотозйомкою [8]. Незважаючи на те, що технологія постійно вдосконалюється, багатороторі принципово дуже неефективні і вимагають багато енергії, щоб боротися з гравітацією та утримувати їх у повітрі.

Рисунок 1.1 – Багатомоторний БПЛА

2. З фіксованим крилом (рисунок 1.2) – безпілотники з нерухомим крилом (на відміну від «поворотного крила», тобто вертольотів) використовують крило, як звичайний літак, щоб забезпечити підйомну силу, а не вертикальні підйомні ротори [9]. Через це їм потрібно лише використовувати енергію, щоб рухатися вперед, а не триматися в повітрі, тому вони набагато ефективніші.

Рисунок 1.2 – БПЛА з фіксованим крилом

3. Одноротoрний (рисунок 1.3) – одноротoрні мають перевагу значно більшої ефективності порівняно з багатороторними, а також те, що вони можуть живитися від газового двигуна для ще більшої витривалості. Загальним правилом аеродинаміки є те, що чим більша лопатка ротора і чим повільніше вона обертається, тим вона ефективніша. Ось чому квадрокоптер ефективніший

за октокоптер, а спеціальні довговічні квадроцикли мають великий діаметр опори [10].

Рисунок 1.3 – Одноротorний БПЛА

4. Гібридний БПЛА з фіксованим крилом (рисунок 1.4). Об’єднання переваг БПЛА з фіксованим крилом і можливістю зависання – це нова категорія гібридів, які також можуть злітати і приземлятися вертикально [11].

Рисунок 1.4 – Гібридний БПЛС

Для демонстрації можна переглянути основні переваги та недоліки кожного типу БПЛА у таблиці 1.2.

Таблиця 2 – Переваги та недоліки конструкцій БПЛА

	Плюси	Мінуси	Користувачі
Багатороторний	Доступність Простота використання Хороше управління камерою Може працювати в обмеженій зоні	Аерофотозйомка та відео	Побутові користувачі
З фіксованим крилом	Довга витривалість Велика площа охоплення Швидка швидкість польоту	Запуск і відновлення Літати важче, потрібно більше тренування Дорого	Аерокартографія, перевірка трубопроводів та ліній електропередач
Одноороторний	Довга витривалість (з газом) Можливість більш важкого навантаження	Небезпечніше Літати важче, потрібно більше тренування Дорого	Повітряне лазерне сканування LIDAR
Гібридний БПЛА з фіксованим крилом	Тривалий політ	Не ідеальний ні для зависання, ні для прямого польоту Ще в розробці	Доставка дроном

1.2 Аналіз існуючих алгоритмів керування БПЛА

1.2.1 Основні методи вирішення проблеми керування БПЛА

Безпілотний літальний апарат є чудовою платформою для дослідження систем управління, його нелінійна природа різна конфігурація роблять його ідеальним варіантом для програмування та аналізу алгоритмів керування .

Після короткого опису систем та БПЛА, було проаналізовано декілька алгоритмів, у тому числі їх переваги та недоліки: PID, Лінійний квадратичний регулятор (LQR), ковзний режим, бекстепінг, лінеаризація зворотного зв’язку, адаптивний, надійний, оптимальний та нейронні мережі.

1.2.2 Пропорційна інтегральна похідна (PID)

PID-регулятор застосовувався в широкому діапазоні застосувань регуляторів. Це дійсно найбільш застосовуваний контролер у промисловості.

У класичний лінійного PID-регулятора коефіцієнт посилення параметрів легко регулюється, він простий у проектуванні та має хорошу надійність. Однак деякі з основних проблем із БПЛА включають нелінійність, пов'язану з математичною моделлю, і неточну природу моделі через немодельне або неточне математичне моделювання частини динаміки.

Тому застосування PID-регулятора до БПЛА обмежує його продуктивність. Часто для регулювання положення БПЛА використовувався PID-регулятор, а для контролю висоти – динамічний поверхневий контроль (DSC). Застосовуючи критерії стабільності Ляпунова, було доведено, що всі сигнали БПЛА були рівномірно обмеженими. Це означало, що БПЛА був стійким до умов зависання. Однак із симуляційних та експериментальних графіків видно, що PID-регулятор краще працював у відстеженні кута нахилу, тоді як при відстеженні кута нахилу можна було спостерігати великі похибки усталеного стану.

В іншому варіанті для регулювання положення та орієнтації квадротора був застосований PID-регулятор [12]. Підсилення параметра PID було обрано інтуїтивно. Продуктивність PID-регулятора вказує на відносно хорошу стабілізацію положення. Час відгуку був хорошим, майже з нульовою помилкою усталеного режиму та з невеликим перевищенням. У літературі загалом встановлено, що PID-регулятор успішно застосовувався до БПЛА, хоча з деякими обмеженнями. Налаштування PID-регулятора може створити певні проблеми, оскільки все має бути сформовано навколо точки рівноваги, яка є точкою висіння, щоб забезпечити хорошу продуктивність [13]. На рисунку 1.5 показана загальна блок-схема ПІД-регулятора для квадротора.

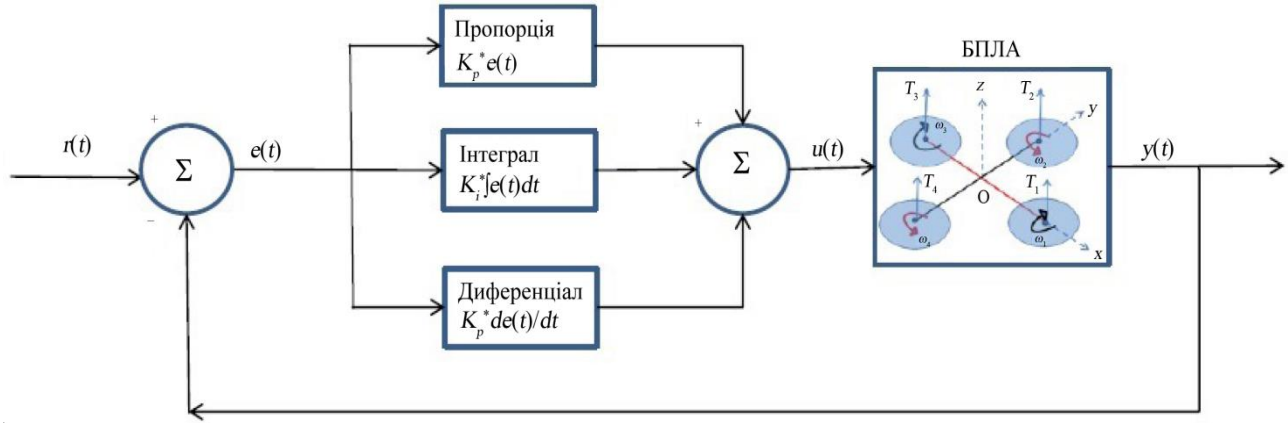


Рисунок 1.5 – Структурна схема PID-регулятора, нанесеного на БПЛА

1.2.3 Лінійний квадратичний регулятор (LQR)

Алгоритм оптимального керування LQR керує динамічною системою шляхом мінімізації відповідної функції витрат. Вчені застосували алгоритм LQR до БПЛА і порівняли його продуктивність з продуктивністю PID-регулятора в. Однак PID застосовується до спрощеної динаміки БПЛА , а LQR – до повної моделі. Обидва підходи дали середні результати, але неявно було зрозуміло, що підхід LQR мав кращу продуктивність, враховуючи той факт, що він застосовувався до більш повної динамічної моделі.

Показано, що при моделюванні з використанням оптимальних траєкторій у реальному часі, незважаючи на наявність вітру та інших збурень, було

досягнуто точного проходження шляху. Здавалося, контролер втратив відстежуваність після того, як уникнув перешкоду. Його працездатність за наявності багатьох перешкод ще потрібно проаналізувати. У поєднанні з лінійним квадратичним оцінювачем (LQE) і фільтром Калмана алгоритм LQR перетворюється на лінійний квадратичний алгоритм Гауса (LQG) [14-16].

Цей алгоритм призначений для систем з гаусовим шумом та неповною інформацією про стан. На рисунку 1.6 показана загальна блок-схема контролера LQG для БПЛА.

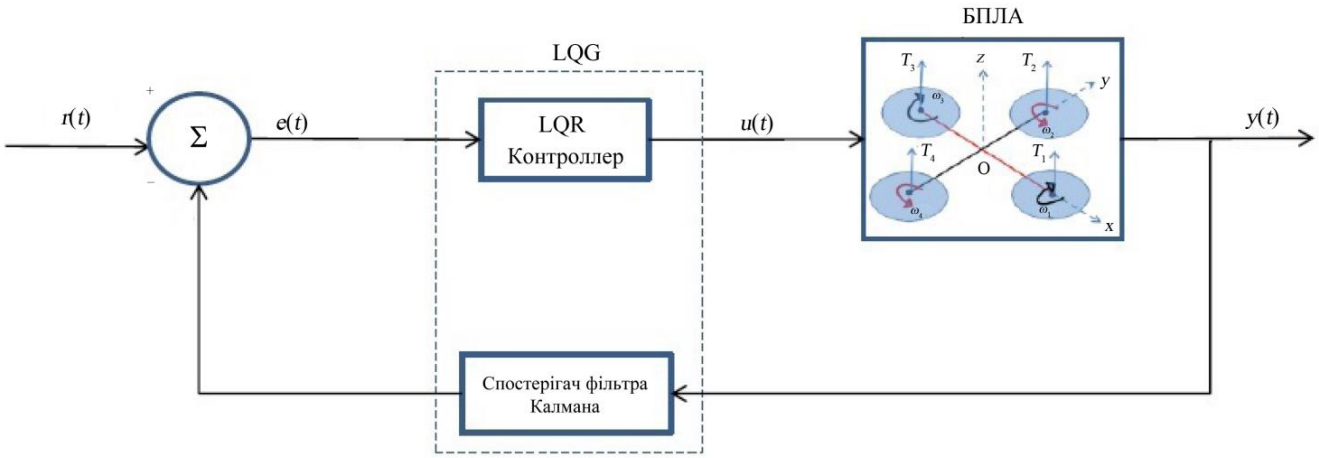


Рисунок 1.6 – Структурна схема контролера LQG, нанесеного на БПЛА

1.2.4 Адаптивний алгоритм

Адаптивні алгоритми керування спрямовані на адаптацію до змін параметрів системи. Параметри або невизначені, або змінюються з часом. Безперервний змінний у часі адаптивний контролер, який демонструє хороші характеристики, був реалізований науковцями з відомими невизначеністю маси, моментів інерції та аеродинамічних коефіцієнтів демпфування.

Також науковці реалізували адаптивну схему керування з використанням лінеаризації зворотного зв’язку (FBL) для квадроторів з динамічними змінами центру ваги [17]. Було помічено, що коли центр ваги змінюється, PD та регулярні методи лінеаризації зворотного зв’язку не змогли стабілізувати систему, але адаптивний контролер зміг її стабілізувати.

Метод адаптивного керування, заснований на нормі прямолінійної відстані з компромісом між продуктивністю керування та надійністю. Модифікована (лінеаризована) модель змогла компенсувати постійні та помірні пориви вітру.

На рисунку 1.7 показана загальна блок-схема адаптивного контролера для очищення БПЛА, що показує оцінювач параметрів і модель БПЛА.

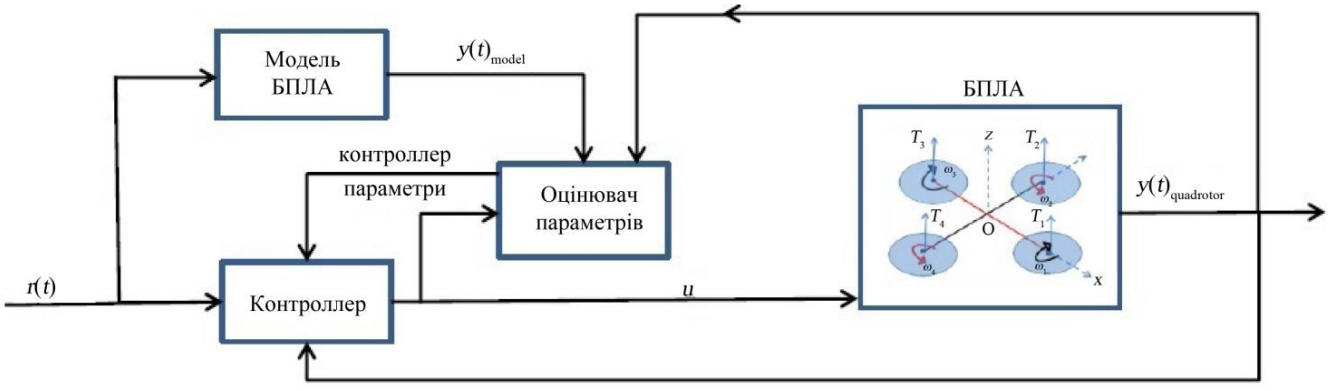


Рисунок 1.7 – Структурна схема контролера LQG, нанесеного на БПЛА.

1.2.5 Надійний алгоритм

Надійні алгоритми керування розроблені для боротьби з невизначеністю параметрів системи або порушеннями. Це гарантує роботу контролера в межах прийнятних діапазонів перешкод або немодельованих параметрів системи [18].

Основним обмеженням, яке зазвичай спостерігається з надійними контролерами, є погана здатність відстеження. Надійний контролер був реалізований науковцями для підсистеми регулятора орієнтації, заснованої на лінійному управлінні та надійній компенсації.

Результати експерименту підтвердили ефективність контролера для контролю позиції. Також представлений надійний алгоритм відстеження , який досягає асимптотичної стабільності за наявності параметричних невизначеностей та невідомих нелінійних збурень. Це було досягнуто за допомогою незатратного закону керування, відсутності спостерігачів, функціональних апроксиматорів чи законів адаптивного онлайн оновлення.

1.2.6 Оптимальний алгоритм

Алгоритми оптимізації розроблені для мінімізації змінних та отримання найкращої функціональних затрат з набору альтернатив. Спеціалізований випадок оптимізації відомий як опукла оптимізація. Це метод математичної оптимізації, який стосується мінімізації опуклої змінної в опуклому наборі. Поширені оптимальні алгоритми включають LQR, L1, H∞ і фільтр Калмана.

Основним обмеженням алгоритмів оптимізації, як правило, є їх низька надійність. Контролер, який є одночасно відносно надійним і оптимальним на L1 алгоритмі, був застосований для відстеження як положення, так і курсу. Продуктивність контролера була експериментально перевірена і була ефективною в мінімізації помилок і відхиленні постійних збурень. Алгоритм оптимізації на основі циклу H∞ був застосований до БПЛА зі спрощеною динамікою для ітераційної ідентифікації параметрів та контролю положення.

Алгоритм мав кращу продуктивність щодо відхилення збурення навіть в умовах насичення збурення. Однак компенсатор “мураха”, заснований на рівняннях Ріккаті [19], не показав обчислювальної ефективності. Інтегральний оптимальний прогнозний H∞ регулятор був реалізований для стабілізації обертального руху БПЛА та для слідування за траєкторією (з використанням прогнозованого керуванням моделлю). Контролер досяг стійкого зменшення перешкод і хорошої надійності. Комплексна дія була ключовою для досягнення хорошого відстеження в цьому алгоритмі.

1.2.7 Нейронні мережі

Інтелектуальні алгоритми керування застосовують кілька підходів штучного інтелекту, деякі з яких є біологічними, для керування системою. Приклади включають нечітку логіку, нейронні мережі, машинне навчання та генетичний алгоритм. Вони, як правило, містять значну невизначеність і математичну складність. Ця складність і необхідні обчислювальні ресурси обмежують використання інтелектуальних систем. Інтелектуальне керування не

обмежується нечіткою логікою та нейронними мережами, але вони найбільш широко використовуються. Багато інших алгоритмів існують і продовжують формулюватися.

Алгоритми нечіткої логіки мають справу з багатозначною логікою, а не з дискретними рівнями істини. Інтелектуальний нечіткий контролер був застосований для контролю положення та орієнтації БПЛА з хорошою реакцією при моделюванні. Однак основним обмеженням цієї роботи був підхід методом проб і помилок для налаштування вхідних змінних. Штучні нейронні мережі біологічно натхненні центральною нервовою системою та мозком. Цей алгоритм був оптимізований для стабілізації проти помилок моделювання та значного вітрового збурення. Спосіб продемонстрував покращення щодо досягнення бажаного позиціонуванні та зменшення дрейфу ваги.

Управління вихідним зворотним зв'язком було реалізовано на БПЛА з використанням нейронних мереж для формування БПЛА-лідерів та БПЛА-послідовників, щоб дізнатися повну динаміку БПЛА, включаючи немодельовану динаміку. Для керування всіма шістьма ступенями свободи з чотирьох керуючих входів було реалізовано управління віртуальною нейронною мережею. Використовуючи теорію стійкості Ляпунова [20], було показано, що положення, орієнтація, помилки стеження за швидкістю, помилки оцінки спостерігача та віртуальне керування були напів глобально рівномірно обмежені остаточно. Для стабілізації БПЛА за наявності синусоїдального збурення в роботі була застосована схема адаптивної нейронної мережі. Запропоноване рішення двох паралельних одиничних прихованих шарів виявилося плідним, оскільки було досягнуто зменшення похибки відстеження та відсутності дрейфу ваги [21-22].

На рисунку 1.8 показана загальна блок-схема контролера нечіткої логіки, реалізованого на БПЛА.

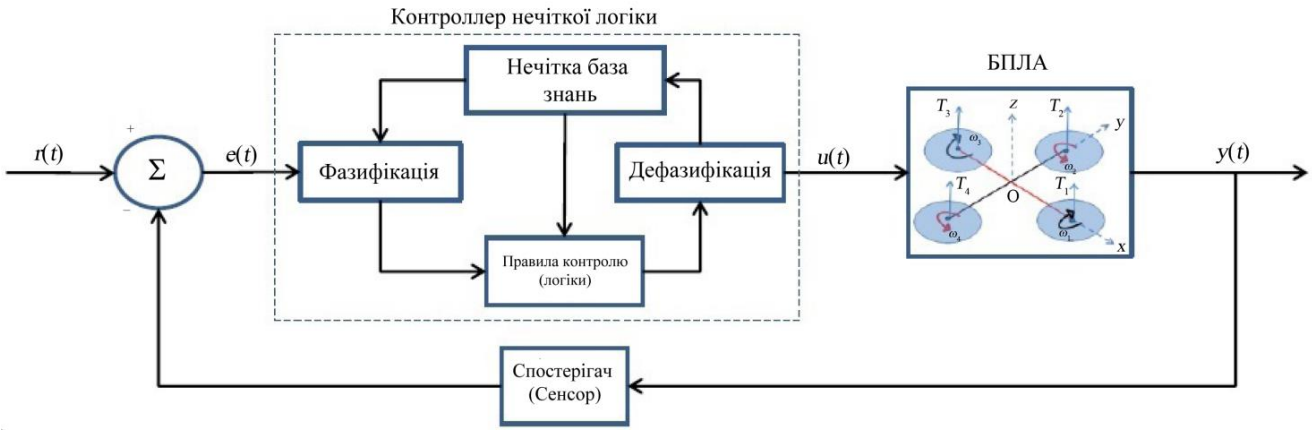


Рисунок 1.8 – Блок-схема контролера нечіткої логіки, застосованого до БПЛА.

1.3 Постановка задачі

Внаслідок вивчення отриманого матеріалу та аналізу предметної області, можемо зробити висновок, про існуючі проблеми в заданих алгоритмах та їх застосування до окремих видів БПЛА. Визначити методи покращення, та виправлення основних помилок, у новому програмно-апаратному засобі на основі мурашиного алгоритму.

Об’єктом дослідження є група безпілотних літальних апаратів.

Метою дослідження є модернізація та оптимізація алгоритму керування групою БПЛА з метою швидкого та безпечного виконання ними місій. Заради досягнення мети потрібно виконати поставлені завдання:

1. Провести аналіз існуючих алгоритмів керування БПЛА, виявити їх недоліки.
2. Провести аналіз нечіткого мурашиного алгоритму, який буде застосовуватись для реалізації.
3. Розробити проектування програмно-технічної системи на основі дослідження.
4. Створити симуляцію системи групи керування БПЛА.
5. Провести тести реалізації системи.
6. Провести аналіз отриманих результатів та оцінити їх практичне застосування.

Актуальність даного дослідження пояснюється необхідністю програмно-технічного засобу який зміг би пришвидшити виконання місій БПЛА, із найменшою часткою похибок на колізій під час процесу, та малими затратами у навчанні алгоритму.

Застосовуючи теоретичну складову нечіткого мурашиного алгоритму заснованого на дії феромонів мурах, взаємодія елементів системи буде відбуватись без критично-довгих навчань та відносно очікуваним результатом. Так як дані схожі системи використовують при координації літальних апаратів для посадки космічних модулів чи літаків, при посадці, проводячи аналогію із ними можна визначити що даний алгоритм буде актуальним та матиме місце використанням а у багатьох сферах де використовуються групи БПЛА.

1.4 Висновки

У даному розділі досліджено методи та алгоритми, які використовуються для керування групою безпілотних літальних апаратів, з метою виконання завдань за найкоротший час із найвищою ефективністю.

В результаті проведеного аналізу, було виведено висновок, що дані методи мають значні недоліки, а саме:

1. Низька швидкодія.
2. Конфліктна робота БПЛА.
3. Низька ефективність в реагуванні на спливаючі перешкоди.
4. Довге навчання алгоритму.

Кожен із приведених у розділі алгоритмів, має ті чи інші недоліки, і може бути вдосконаленим.

Тому необхідно розробити відповідний математичну модель та сконструювати алгоритм із покращеними властивостями. Основними покращеннями якого будуть: покращена швидкодія, ефективна групова робота та швидке реагування на перешкоди.

2 МАТЕМАТИЧНА МОДЕЛЬ СКООРДИНОВАНОГО ПЛАНУВАННЯ ОПТИМАЛЬНОЇ ТРАЄКТОРІЇ РУХУ БПЛА

2.1 Модель роботи чотирьох роторного БПЛА із PID контроллером

Для реалізації методу контролю групи БПЛА, виходячи із матеріальної складової, та оптимального обладнання БПЛА, враховуючи схеми контролю було обрано БПЛА типу квадотор.

Квадротор є нелінійною системою з шістьма ступенями свободи, має багато вхідну та багато вихідну систему під приводу. Він керується зміною сил тяги кожного ротора та балансуванням крутного моменту опору.

Квадротор має два комплекти пропелерів, які обертаються протилежно, що нейтралізує ефективний аеродинамічний опір. Він має три основних режими роботи (рисунок 2.1).

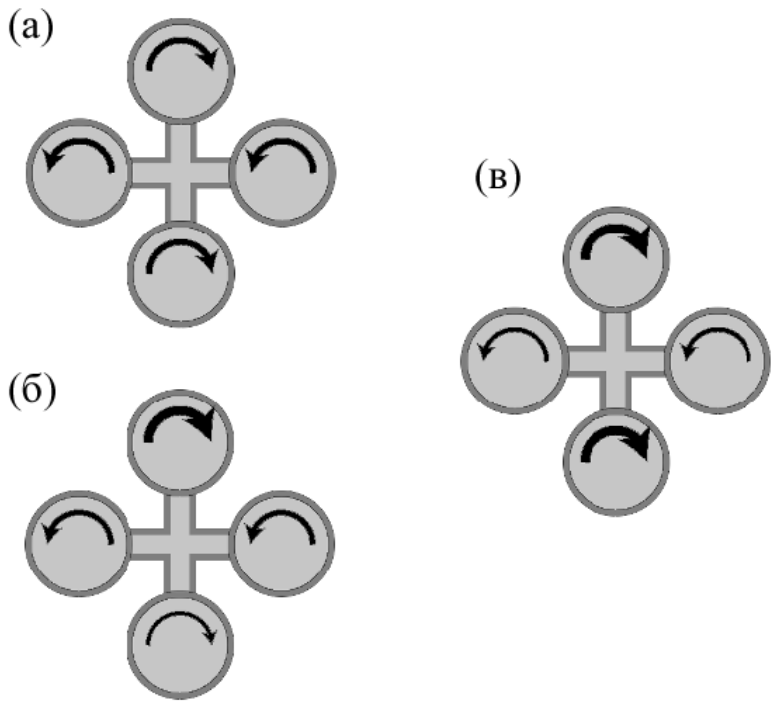


Рисунок 2.1 – Основні моделі польоту квадротора. (а) зависаючий рух і вертикальний рух завдяки збалансованим крутним моментам; (б) різниця в тягі для створення моменту крену / кроку; (в) різниця в крутному моменті для створення моменту повороту

Ці основні три принципи, мають наступні характеристики:

1. Вертикальне переміщення контролюється одночасним збільшенням або зменшенням тяги всіх роторів.

2. Поворотний момент створюється шляхом пропорційного зміни швидкостей деталей, що обертаються протилежно.

3. Кути крену та нахилу можна керувати шляхом застосування диференціальних сил тяги до протилежних роторів квадротора.

Для і-го ротора з кутовою швидкістю W_i підйомна сила задається формулою (2.1) і момент опору зображено на формулі (2.2).

$$F_i = bW_i^2, \tag{2.1}$$

$$\tau_i = dW_i^2 \tag{2.2}$$

де b і d – моменти тяги і опору відповідно.

Загальна сила тяги може бути представлена як:

$$f = \sum_{i=0}^{4} F_i. \tag{2.3}$$

За основу досліджуваного об’єкта для розбору основ кінематики та схемо техніки візьмемо БПЛА на мікрокомп’ютері Odroid-XU4 [22], який побудований на на процесорі Samsung Exynos 5422. Цей виріб виконаний на архітектурі ARM big.LITTLE, він містить чотири ядра Cortex-A15 з тактовою частотою 2,0 ГГц і таку кількість ядер Cortex-A7 з частотою 1.4 ГГц. Обробкою графіки працює інтегрований контролер Mali-T628 MP6. Для охолодження чіпа застосовується маленький вентилятор.

Опис лицевої частини плати зображено на рисунку 2.2, а опис тилової частини плати комп’ютера зображена на рисунку 2.3.

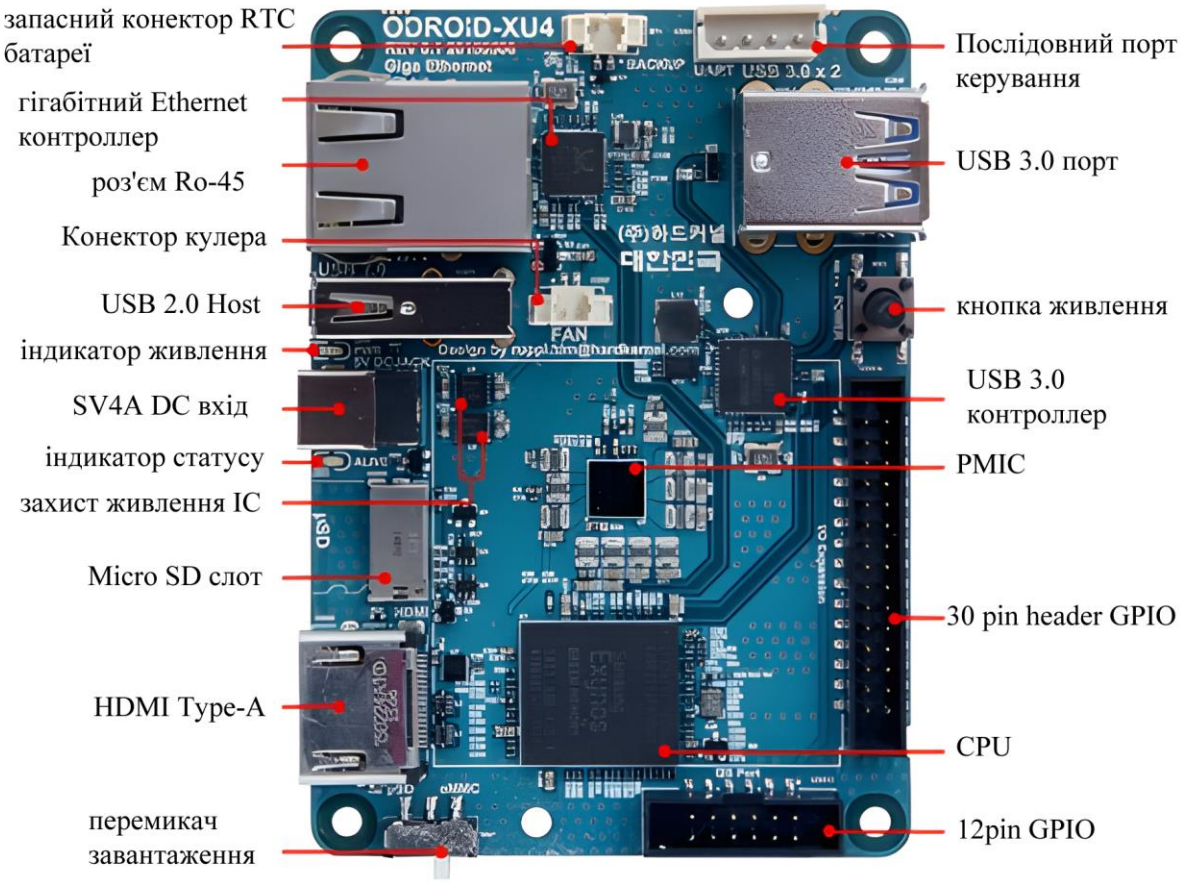


Рисунок 2.2 – Системна плата Odroid-XU4 лицева частина

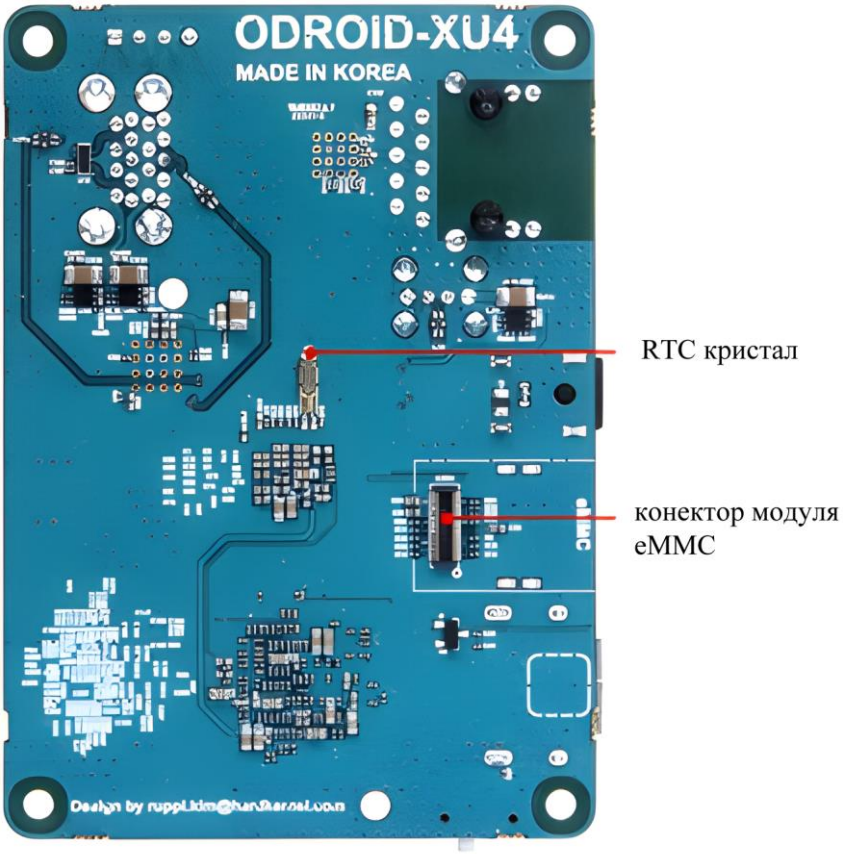


Рисунок 2.3 – Системна плата Odroid-XU4 тилова частина

За основу «мозку» БПЛА буде вважатись мікрокомп’ютер Odroid-XU4, самий ж каркас БПЛА із чотирьох роторною основою та під’єднаним комп’ютером зображений на рисунку 2.4.

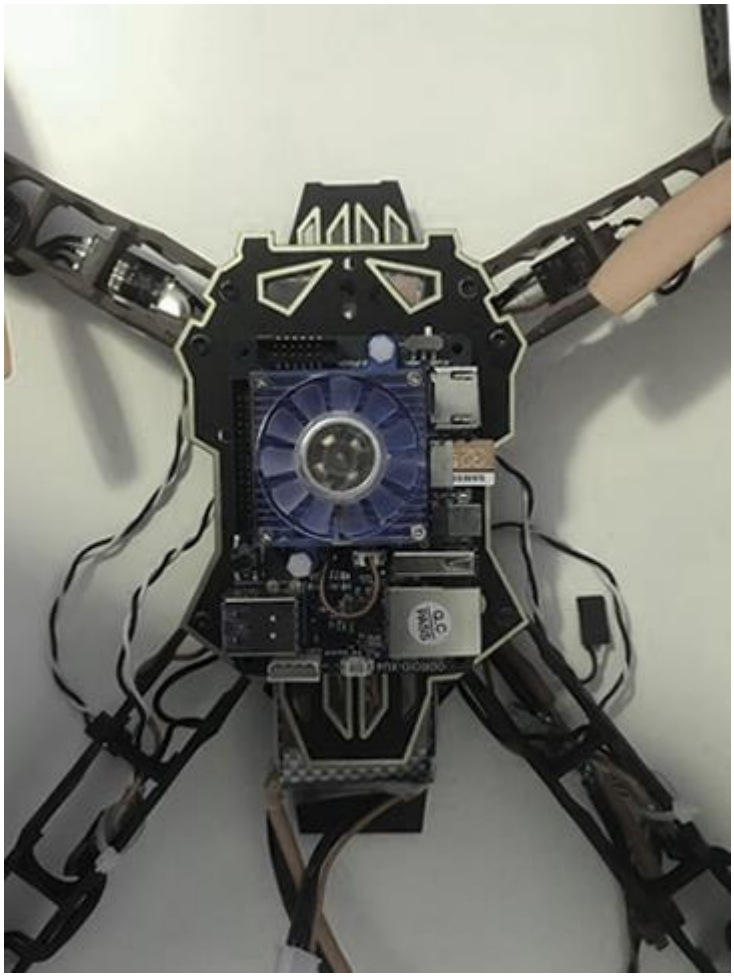


Рисунок 2.4 – Odroid-XU4 під’єднаний до каркасу квадротора

Враховуючи особливості будови БПЛА, та роботу його роторів, можна описати роботу кінематики апарату.

Припустимо що (u; c; p) – це лінійна швидкість уздовж напрямку осі крену, кут повороту та кутова швидкість навколо осі крену. (w; h; r) – це лінійна швидкість уздовж напрямку осі рискання, кут повороту та кутова швидкість навколо осі. (v; o; q) – лінійна швидкість уздовж напрямку осі кроку, кут повороту та кутова швидкість навколо осі кроку. (x; y; z) – положення квадрокоптера вздовж осей i, j та k інертної системи відліку.

Всього існує п’ять систем координат [23]. А саме: інерційна рама, рама транспортного засобу, рама з регулюванням по рисканню, рама з регулюванням нахилу та рама кузова.

Інерційна рама закріплена на землі в попередньо визначеному місці. Одиничний вектор і спрямований на північ, j на схід, а k – на Землю. Рама транспортного засобу має осі, паралельні інерціальній рамі, але початок координат зміщений до центру ваги квадрокоптера.

Поворот рами транспортного засобу налаштовується відповідно до повороту квадрокоптера, щоб отримати раму з коригуванням нахилу, яка потім регулюється, щоб отримати раму з нахилом.

Нарешті, каркас кузова отримують шляхом регулювання крену рами з регулюваним кроком. Перетворення з інерційної на раму транспортного засобу є лише простим переходом. Перетворення від транспортного засобу до кузова задається наступною матрицею обертання:

$$R_v^b(c, o, h) = R_p^b(c)\, R_y^p(o)\, R_v^y(h). \qquad (2.4)$$

Де R_p^b – це перетворення від рами з коригуванням нахилу в раму кузова, R_y^p – перетворення від рами з коригуванням нахилу до рами з коригуванням нахилу, R_v^y – перетворення від рами транспортного засобу до рами з коригуванням нахилу і R_v^b є перетворення з кузова БПЛА на раму кузова. Оскільки кожен з параметрів квадротора: крен, крок і рискання вимірюються відносно різних кадрів, ці перетворення допомагають у взаємодії змінних.

Що ж до самої кінематики, то її можна розділити на два типи, поступальна кінематика та кінематика обертання.

Для поступальної кінематики можна виділити наступні змінні (x, y, z), які є параметрами інерціальної системи, тоді як значення швидкості (u,v,w) є параметрами рамки тіла [24]. Вони можуть бути пов'язані за допомогою матриці перетворення наступним чином:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R_b^v \begin{pmatrix} u \\ v \\ w \end{pmatrix} = (R_v^b)^T \begin{pmatrix} u \\ v \\ w \end{pmatrix}. \quad (2.5)$$

В кінематиці обертання, оскільки рискання (відносно рами транспортного засобу), нахил (відносно рами з коригуванням нахилу) і крен (відносно рами з нахилом) вимірюються відносно різних систем координат, перетворення для кожної з них є різними. Отже, кутові швидкості (p; q; r) отримують таким чином:

$$\begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} c \\ 0 \\ 0 \end{pmatrix} + R_p^b(c) \begin{pmatrix} 0 \\ o \\ 0 \end{pmatrix} + R_p^b(c) R_y^p(o) \begin{pmatrix} 0 \\ 0 \\ h \end{pmatrix}. \quad (2.6)$$

Аналогічним чином потрібно описати динаміку, яку також можна розділити на поступальну та обертальну.

В поступальній кінематиці використовуючи другий закон Нютона, можна отримати:

$$F = m\frac{dv}{dt} = m\left(\frac{dv}{dt} + w_b \times v\right) \quad (2.7)$$

де $v = (u, v, w)^T$ і $w_b = (p, q, r)^T$.

При застосуванні законю Нютону для обертальної динаміки можна отримати

$$\tau = \frac{dL}{dt} = \left(\frac{dL}{dt} + w_b \times L\right) \quad (2.8)$$

де $L = Jw_b$ – кутовий моменту, а τ – прикладений момент, а J основна матриця.

2.2 Застосування моделі оптимізації діяльності колонії мурах для вирішення задачі скоординованого планування оптимальної траєкторії руху БПЛА

Для вирішення поставленої задачі було використано модель, яка основана на оптимізації діяльності колонії мурах (ACO), та називається мурашиною системою, Колонія мурах є високоорганізованою системою в якій один взаємодіє з іншими за допомогою феромонів у повній гармонії. Проблеми оптимізації можна вирішити за допомогою моделювання поведінки мурах. З тих пір, як був запропонований перший алгоритм мурашиної системи, в ACO було багато моделей.

В алгоритмі системи мурашиних колоній локальний феромон використовується для мурах для пошуку оптимального результату. Однак велика кількість обчислень є її недоліком, а іноді призводить до неефективності системи. Томас Штуцле та ін. представив MAX-MIN Ant System (MMAS) у 2000 році. Це один з найкращих алгоритмів ACO [25].

Він обмежує загальну кількість феромонів у кожній спробі руху або підз’єднанні, щоб уникнути локальної конвергенції. Однак обмеження феромонів уповільнює швидкість зближення в MMAS.

Добре відомо, що в алгоритмі оптимізації при пошуку локального оптимального рішення або коли мурахи приходять у стан стагнації, алгоритм може більше не шукати глобальне найкраще оптимальне значення. Тому існує модернізований алгоритм.

Коли мурахи приходять до локального оптимального рішення або до стану стагнації, вони не наближаються до глобального найкращого оптимального рішення. У роботі Джун Оуянга пропонується модифікований алгоритм, система мультиколоній мурах на основі феромонного арифметичного кросовера та оператора відштовхування, щоб уникнути такого стану стагнації [26]. У цьому алгоритмі спочатку створюється кілька колоній мурашиної системи, а потім вони виконують ітерацію та оновлення своїх феромонних масивів відповідно до тих

пір, поки одна система колоній мурах не досягне свого локального оптимального рішення.

Кожна система колонії мурах володіє своїм масивом феромонів і параметрами і записує своє оптимальне локальне рішення. Більше того, як тільки система колоній мурах приходить до свого локального оптимального рішення, вона оновлює його і надсилає це рішення до глобального центру найкращих значень.

По-третє, коли стару систему колонії мурашок вибирають відповідно до правил елімінації, вона буде знищена та повторно ініціалізована за допомогою застосування феромонного арифметичного кросовера та оператора відштовхування, заснованого на кількох глобальних найкращих на сьогоднішній день оптимальних рішеннях. Весь алгоритм реалізує ітерації до тих пір, поки не буде знайдено оптимальне глобальне рішення. У наступних розділах будуть представлені деякі концепції та правила цієї системи з багатьма колоніями мурах.

2.3 Опис основ нечіткого мурашиного алгоритму моделі ACO для БПЛА

Принцип нечіткого мурашиного алгоритму полягає в тому, що під час подорожі мурах на землі залишається спеціальний хімічний слід (феромон), який спрямовує інших мурах до цілі. Більше феромонів залишається, коли більше мурах проходить заданий маршрут, що підвищує ймовірність того, що інші мурахи виберуть цей шлях. Крім того, цей хімічний слід (феромон) з часом зменшується через випаровування слідів. Крім того, кількість феромонів, яку залишають мурахи, залежить від кількості мурах, які використовують цей маршрут.

На рисунку 2.5 представлено процес прийняття рішення мурахами щодо вибору своїх маршрутів. Коли мурахи зустрічаються в точці прийняття рішень «A», деякі вибирають одну сторону, а інші обирають випадковим чином із того що залишилось. Припустимо, що ці мурахи повзають з однаковою швидкістю, ті,

хто вибирає коротку сторону, приходять до точки прийняття рішення «B» швидше, ніж ті, хто вибирає довгу сторону.

Мурахи, випадково вибравши коротку сторону, першими дістаються до місця зустрічі. Таким чином, коротка сторона отримує феромон раніше, ніж довга, і цей факт підвищує ймовірність того, що подальші мурахи вибирають її, а не довгу. В результаті кількість феромонів формується швидше в короткій стороні, ніж на довгій, тому що більше мурах вибирають коротку сторону, ніж довгу.

Кількість ламаних на рисунку 2.5 має приблизно пряме відношення до кількості мурах. Система штучних мурашок створена за принципом системи мурашиних колоній для вирішення різного роду задач оптимізації. Феромон є ключем до прийняття рішень мурахами.

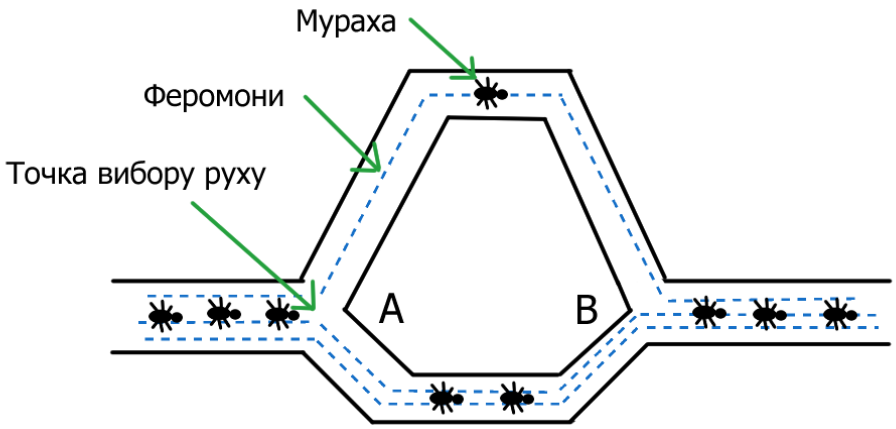


Рисунок 2.5 - Процес вибору маршруту мурахами, за кількістю феромонів

ACO спочатку застосовувався до задачі комівояжера (TSP). TSP є класичною задачею оптимізації і є однією з класу NP-проблем [27-29]. Враховуючи набір з N міст, TSP можна сформулювати як проблему пошуку мінімальної тривалості закритого руху, який відвідує кожне місто один раз. Кожне місто – це точка прийняття рішень мурах. Визначення (i,j) край міста - i, та міста - j. Кожному краю (i,j) присвоюється значення (довжина) d_{ij}, що є відстанню між містами i та j.

2.3.1 Оновлення феромонів

Мурахи залишають свій феромон на краях під час кожної подорожі, при завершенні однієї ітерації. Сума феромона одного ребра визначається так:

$$\tau_{ij}\,(t\ +1)\ =\ \Delta\tau_{ij}\ +\ (1-\ \rho)\tau_{ij}\,(t) \tag{2.9}$$

де $\rho \in (0,1)$, 1-ρ – швидкість збереження попереднього феромона. ρ визначається як швидкість випаровування феромона.

У MMAS лише найкраща мураха оновлює сліди феромонів і значення феромону привязане до неї.

Отже, правило оновлення феромонів задається:

$$\tau_{ij}\,(t\ +1)\ =\ \left[\Delta\tau_{ij}^{best}\ +\ (1-\ \rho)\tau_{ij}\,(t)_{\tau_{min}}^{\tau_{max}}\right] \tag{2.10}$$

де τ_{max} і τ_{min} – відповідно верхня та нижня межі, накладені на феромон; і Δ найкращий $\Delta\tau_{ij}^{best}$є:

$$\Delta\tau_{ij}^{best}\ =\ \begin{cases}\frac{1}{L_{best}},\ \text{якщо } (i,j) \text{ в найкращій вибірці}\\ 0,\ \text{у інших варіантах}\end{cases} \tag{2.11}$$

де L_{best} - це вартість рішення або найкращої ітерації, або найкращої на даний момент, або комбінації обох.

2.3.2 Правило руху мурах

Мурахи переходять з одного міста (колонії) в інше відповідно до ймовірності. По-перше, міста, до яких здійснюється доступ, повинні бути поміщені в таблицю табу.

Потрібно визначити набір міст, до яких ніколи не звертався k-й мураха як дозволений k. По-друге, потрібно визначити видимий градус η_{ij}, $\eta_{ij} = \frac{1}{d_{ij}}$. Імовірність того, що k-й мураха вибере місто, визначається як:

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{k\in\text{доступні}_k}[\tau_{ik}(t)]^{\alpha}[\eta_{ik}]^{\beta}}, j \in \text{доступні}_k \\ 0, \text{ у інших варіантах} \end{cases} \quad (2.12)$$

де α та β – важливі параметри, які визначають відносний вплив феромона сліду та евристичної інформації, а j:

$$j = \begin{cases} arg_{k\in\text{доступні}_k}\ max\{[\tau_{ik}(t)]^{\alpha}[\eta_{ik}]^{\beta}\}, \text{ якщо } \rho \le \rho_0 \\ J, \text{ у інших варіантах} \end{cases}. \quad (2.13)$$

2.3.3 Ініціалізація феромонного сліду та його зупинка

На початку прогону ми встановлюємо $\tau_{max} = 1 / ((1 - \rho)C_{nn})$, $\tau_{min} = \tau_{max}/2Ni$ початкові значення феромонів $\tau_{ij}(0) = \tau_{max}$, де C_{nn} – тривалість маршруту, створеного евристичний найближчий сусід, а N – загальна кількість міст.

Існує багато умов, щоб мурахи зупинили свою подорож, наприклад, обмеження кількості ітерацій, обмеження часу центрального процесора або знаходження найкращого рішення.

Описами вище, ми можемо отримати детальну процедуру MMAS. MMAS є одним з найбільш вивчених алгоритмів ACO і одним із найуспішніших варіантів.

2.4 Мурашина система з багатьма колоніями на основі арифметичного кросовера феромонів і оператора відштовхування

2.4.1 Концепт механізму

Першим кроком є ініціація системи мультиколонійних мурах. Кожна система колонії мурашок володіє своїм масивом феромонів і параметрами α, β та ρ. Зокрема, кожна колонія може мати власну арифметичну політику. Наприклад, усі колонії використовують різні алгоритми ACO відповідно. Деякі використовують базову мурашину систему.

Деякі використовують елітарну мурашину систему, ACS, MMAS або версію мурашиної системи на основі рангу тощо. Інші, можливо, використовують структуру гіперкубів для ACO [30]. Кожна система колоній мурах починає ітерувати та оновлювати свій масив феромонів відповідно, поки не досягне свого локального оптимального рішення. Використовується власна політика пошуку. Потім він надсилає це локальне оптимальне рішення до найкращого глобального центру.

У глобальному найкращому центрі зберігаються топові глобальні рішення M, які досі шукають усі колонії мурашиної системи. Глобальний центр найкращих результатів також містить параметри α, β та ρ для кожного рішення. Ці параметри дорівнюють параметрам колонії, поки колонія знаходить це рішення. Зазвичай M більше, ніж кількість колоній.

Другим кроком є правило ліквідації старої колонії мурах. Знищення однієї зі старих колоній відбувається за такими правилами:

1. Колонія, яка володіє найменшим локальним оптимальним рішенням серед усіх колоній;
2. колонія, яка володіє найбільшими поколіннями з моменту її останнього локального оптимального рішення;
3. колонія, яка втратила різноманітність. Загалом, передбачається, що в ACO існує принаймні два типи різноманітності: різноманітність у пошуку маршрутів та різноманітність у відкладенні феромонів.

Третім кроком є створення нової колонії мурах за допомогою феромонного кросовера. По-перше, ми вибираємо випадковим чином m (m<<M) рішень із M глобальних найкращих на сьогоднішній день оптимумів у глобальному центрі найкращих знайдених рішень.

По-друге, ми навмисно ініціалізуємо сліди феромонів цієї нової колонії до ρ(t), яке починається з $\rho(t) = \tau_{max}(t) = \frac{1}{((1-\rho)\cdot L_{best}(t))}$, досягаючи таким чином більш високого рівня дослідження рішень на початку алгоритму і більш високоточне використання поблизу m глобальних оптимальних рішень наприкінці алгоритму. Де $L_{best}(t))$ – це найкраща вартість рішення для всіх колоній за поточний час t. Потім ці маршрути модифікуються за допомогою арифметичного кросоверу:

$$\tau_{ij} = \rho(t) + \sum_{k=1}^{m} c_k rand_k() \Delta \tau_{ij}^k . \quad (2.14)$$

Де $\Delta\tau_{ij}^k = \frac{1}{L_{best}^k}$, в якому ребро (i,j) знаходиться на k-му найкращому глобальному рішенні, а L_{best}^k позначає k-му найкраще глобальне рішення в m вибраних рішеннях; $rand_k()$– це випадкова функція, рівномірно розподілена в діапазоні [0,1]; c_k – вага $\Delta\tau_{ij}^k$ і $\sum_{k=1}^{m} c_k = 2$ оскільки математичне очікування $rand_k()$ дорівнює 1/2 . Нарешті, параметри α , β та ρ встановлюються за допомогою арифметичного кросовера:

$$\begin{aligned} \alpha &= \sum_{k=1}^{m} c_k rand_k() \alpha_k , \\ \beta &= \sum_{k=1}^{m} c_k rand_k() \beta_k, \\ \rho &= \sum_{k=1}^{m} c_k rand_k() \rho_k \end{aligned} \quad (2.15)$$

де α_k, β_k і ρ_k належать k-му глобально найкращому розв’язку з m вибраних рішень. Після цих операцій колонія починає повторювати та оновлювати свій локальний феромон заново.

Четвертим кроком є створення оператору відштовхування.

Як у алгоритмі «Shepherd and Sheepdog» [31], тут вводиться привабливий і відразливий ACO, намагаючись ще більше зменшити ймовірність передчасної конвергенції. Ми визначаємо фазу залучення лише як основний алгоритм ACO. На цьому етапі хороші рішення функціонують як «приваблювачі».

У колонії цієї фази мурахи будуть притягуватися до місця знаходження хорошого рішення. Однак нова колонія, яка щойно була повторно ініціалізована за допомогою феромонного арифметичного кросовера, можливо, знову буде перемальована до того самого найкращого локального оптимального рішення, яке було знайдено лише хвилину тому.

В результаті це витрачає багато обчислювальних ресурсів. Тому потрібно визначити другу фазу відштовхування, віднімаючи термін $\Delta\tau_{ij}^{best}$ у зв'язку з найкращим на даний момент рішенням у формулі оновлення феромонів, коли повторно ініціалізується нова колонія. Тоді рівняння (2.6) буде мати вигляд:

$$\tau_{ij} = \rho(t) + \sum_{k=1}^{m} c_k rand_k() \Delta\tau_{ij}^{k} - C_{best}\Delta\tau_{ij}^{best} . \qquad (2.16)$$

Де найкращий $\Delta\tau_{ij}^{best} = 1/L_{best}$, у якому ребро (i,j) знаходиться на найкращому на даний момент рішенні, L_{best} позначає найкращу вартість рішення, C_{best} – це вага найкращого $\Delta\tau_{ij}^{best}$, а інші коефіцієнти – це те саме, що й у рівнянні (2.14). На цій фазі найкраще на сьогоднішній день рішення функціонує як «відлякувач», щоб мурахи могли віддалятися від найкращого рішення. Ідентифікується ця реалізація моделі на основі феромонного кросовера та оператора відштовхування з акронімом MCA [32].

2.4.2 Опис паралельного асинхронного алгоритму для алгоритмів з кількома колоніями

Існує паралельний асинхронний алгоритм для алгоритму з кількома колоніями мурах, щоб ефективно використовувати всі доступні процесори в гетерогенному кластері або гетерогенних обчислювальних середовищах. Цей процес проектування слідує парадигмі ведучий/підпорядкований.

Головний процесор утримує глобальний найкращий центр, надсилає параметри ініціалізації колонії до підпорядкованих процесорів і виконує всі процеси прийняття рішень, такі як оновлення та сортування глобального найкращого центру, перевірка збіжності. Він не виконує жодної ітерації алгоритму колонії мурах. Однак підпорядковані процесори багаторазово виконують ітерацію алгоритму мурашиної колонії, використовуючи параметри, призначені їм.

Завдання, що виконуються головним і підпорядкованим процесорами, такі:

1. Головний процесор:

a) Ініціалізація всіх параметрів колоній та надсилання їх підпорядкованим процесорам.

b) Отримання глобально найкращого центра значень, який зберігає найкращі глобальні рішення M та їх параметри.

c) Отримування локального оптимального рішення та параметрів від підпорядкованих процесорів і оновлення свого глобального найкращий центра значень.

d) Оцінка ефективності колоній мурах у підпорядкованих процесорах.

e) Ініціалізація набіру параметрів нової колонії, використовуючи їх як феромонний кросовер, так і як оператор для відштовхування на основі мультиоптимуму для найгіршої колонії мурах.

f) Вибір однієї з найгірших колоній мурах для знищення та надсилання нових параметрів колонії та команди знищення підпорядкованому процесору, якому належить убита колонія.

g) Перевірка на збіжність.

2. Підпорядкований процесор:

1. Отримування набору параметрів колонії від головного процесора;

2. Ініціалізація колонії мурах і початок ітерації;

3. Відправка свого локального оптимального рішення та параметрів головному процесору;

4. Отримує команду знищення та параметри від головного процесора, а потім використовує ці параметри для повторної ініціалізації та початку ітерації відповідно до рівняння.

Після того, як головний процесор виконав крок ініціалізації, параметри ініціалізації надсилаються до підлеглих процесорів для виконання ітерації алгоритму мурашиної колонії.

Оскільки вміст зв’язку між головним процесором і підлеглими процесорами є лише деякими параметрами і неоптимальними рішеннями, відношення часу зв’язку між головним і підлеглими процесорами до часу обчислень процесорів цієї системи є відносно малим.

Зв'язок може бути здійснений за допомогою схеми зв'язку "точка-точка", реалізованої за допомогою інтерфейсу передачі повідомлень (MPI) [33].

Тільки після отримання свого локального оптимального рішення підпорядкований процесор задає повідомлення головному процесору (рисунок 2.6).

Протягом цього періоду підпорядкований процесор продовжує свою ітерацію, поки не отримає команду знищення від головного процесора.

Потім підпорядкований процесор ініціалізує нову колонію мурах і повторює алгоритм знову.

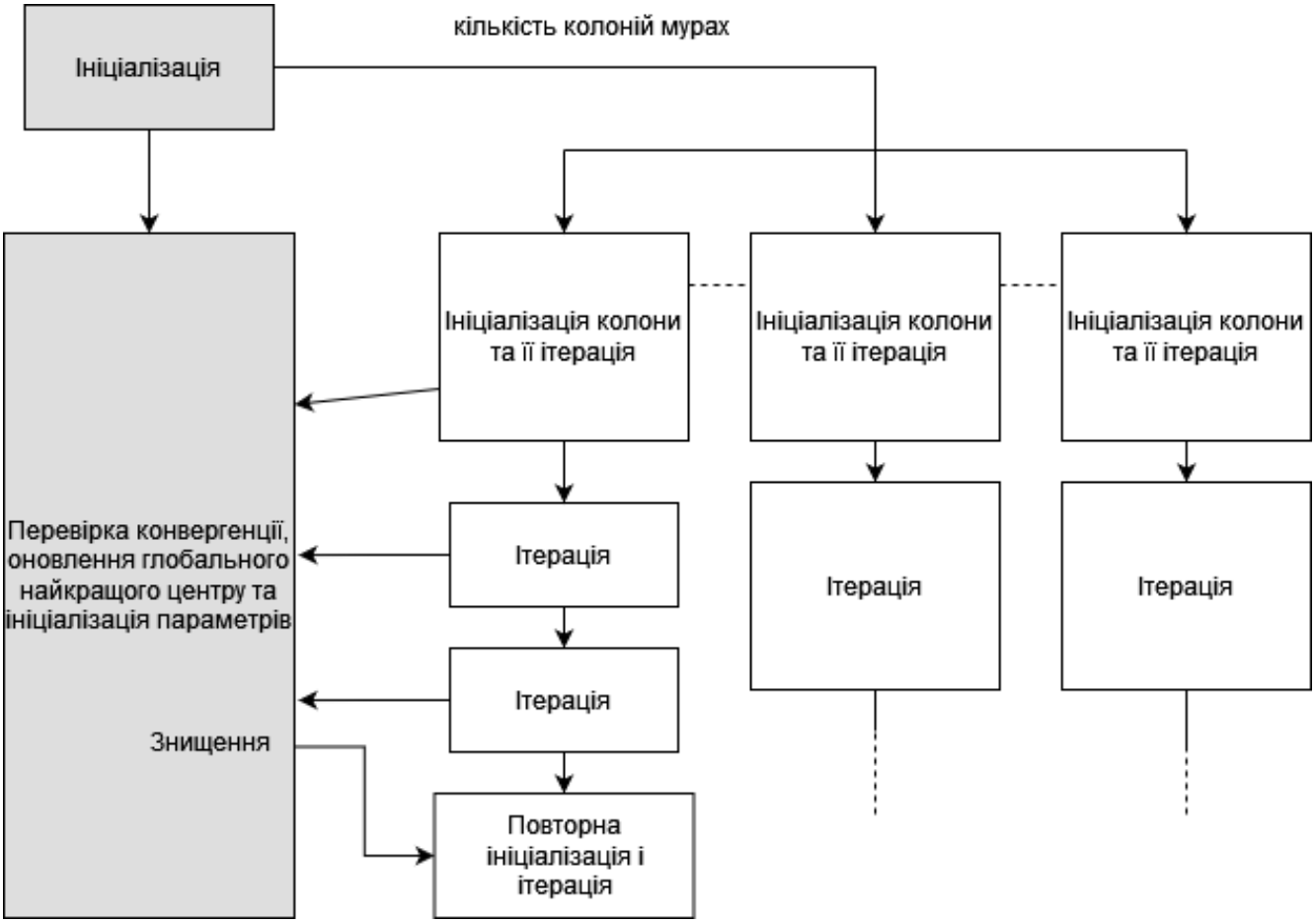


Рисунок 2.6 - Блок-схема для паралельного асинхронного алгоритму. Сірі поля вказують на дії на головному процесорі.

Псевдокод паралельного асинхронного алгоритму МСА представлений алгоритмом 2.1.

Алгоритм 2.1 – Псевдокод паралельного асинхронного алгоритму МСА

Головний процесор

Ініціалізація оптимізації

Ініціалізація параметрів всіх колоній

Відправити їх до підпорядкованих процесорів;

Виконання основного циклу

Отримання локального оптимального рішення і параметри від підпорядкованих процесорів

Оновлення глобального найкращого центру

Перевірка збіжності

If (правило виключення виконано), than

Знаходження найгіршої колонії

Відправка їй команди знищення і встановлення набір нових параметрів

Звіт про результати

Підпорядкований процесор

Отримання параметра ініціалізації від головного процесора

Ініціалізовування нової місцевої колонії мурах

Виконання оптимізації

For k = 1 кількість ітерацій

For i = 1 кількість мурах

Побудова нового рішення

Якщо (отримано команду знищення і набір нових параметрів), тоді

Перехід до ініціалізації нової локальної колонії мурах;

Endfor

Зміна локального масиву феромонів

Відправка локального оптимального рішення та параметрів на головний процесор

Endfor

Кінець

2.5 Експериментальні результати тестування алгоритму

2.5.1 Паралельні незалежні запуски та послідовний алгоритм

У цій паралельній моделі k копій одного і того ж послідовного алгоритму MMAS одночасно і незалежно виконуються з використанням різних випадкових початкових даних. Кінцевим результатом є найкраще рішення серед усіх отриманих. Використання паралельних незалежних запусків є пріоритетним, оскільки в більшості випадків не потрібні комунікаційні витрати і майже не потрібні додаткові зусилля щодо впровадження алгоритму. Ідентифікація реалізації цієї моделі за допомогою абревіатури PIR [34]. Було виявлено, що PIR володіє кращою продуктивністю, ніж будь-яка інша паралельна модель [35].

Щоб мати еталонний алгоритм для порівняння, було також протестовано еквівалентний послідовний алгоритм MMAS [36]. Він працює для тих самих поколінь, що й паралельний алгоритм (k-тих поколінь паралельного алгоритму). Іде ідентифікація реалізації цієї моделі за допомогою абревіатури SEQ [37].

Використавши тестові дані за запустивши симуляцію даних варіацій алгоритму по еволюційній моделі життя та накопичення колоній мурах отримуємо результат зображений на рисунку 2.7.

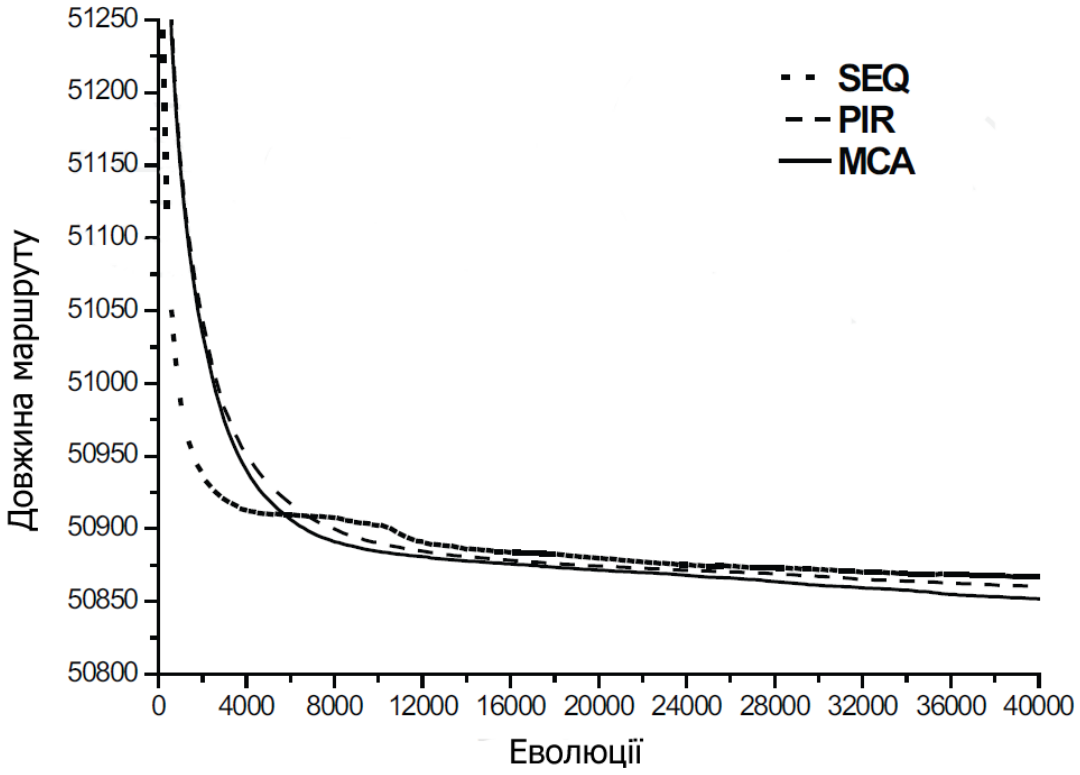


Рисунок 2.7 – Сукупний розподіл часу виконання під час тестування

2.5.2 Результати та висновки тестування алгоритму

Основна мета методу мультиколоній мурашиного алгоритму є – підвищити здатність колоній мурашок виходити з застійного стану.

Цей алгоритм створює нову колонію мурах для ітерації, що досягається шляхом застосування арифметичного кросовера феромонів і оператора відштовхування, заснованого на мультиоптимумі при зустрічі в стані стагнації ітерації або локального оптимуму.

У той же час основні параметри α, β та ρ алгоритму є само адаптивними. Також існує паралельний асинхронний алгоритм.

Аналіз відомих результатів експериментів показав, що алгоритм мультиколонійних мурах є точним методом для транспортних задач.

Отже є різні версії нечіткого мурашиного алгоритму основаної на кількості елементів взаємодії системи, як кількість мурах, що дає можливість вибирати та удосконалювати метод в залежності від потреб і наявних вимог систем.

2.6 Висновки

В розділі було представлено та описано конкретну модель, БПЛА квадротора на основі мікрокомп’ютера Odroid-XU4, із описом його характеристик, схемотехніки та основної функціональності. Представлено математичну модель робити апарату на основі роботи роторів із кінематикою та динамікою.

Також було описано математичну модель асинхронного мурашиного алгоритму ACO для контролю за групою БПЛА. А саме описано роботу алгоритму, представлені кроки можливої реалізації, та взаємодії із технічною частиною апарату. В результаті чого було сформовано, блок схему роботи паралельного асинхронного алгоритму, де продемонстровано роботу головного і побічних процесів групи БПЛА. На основі даної блок схеми було сформовано відповідний алгоритм, та прогнозовано можливі результати роботи як алгоритму, так і цілком можливого програмно -технічного засобу.

3 МЕТОД КЕРУВАННЯ ГРУПОЮ БЕЗПІЛОТНИХ ЛІТАЛЬНИХ АПАРАТІВ НА ОСНОВІ НЕЧІТКОГО МУРАШИНОГО АЛГОРИТМУ

3.1 Модель скоординованого планування оптимальної траєкторії руху БПЛА

3.1.1 Опис скоординованого планування оптимальної траєкторії з кількома БПЛА

Скоординоване планування траєкторії для кількох БПЛА полягає у створенні безпечної та короткої траєкторії для кожного БПЛА. Крім того, траєкторія повинна задовольняти вимогам щодо координації кількох БПЛА. Тому в питанні координації кількох БПЛА заплановані траєкторії можуть бути не оптимальними для окремого транспортного засобу, але вони повинні бути оптимальними або майже оптимальними для всієї команди.

Припустимо, що для прольоту ворожої території та для нападу на ті самі або різні відомі цілі необхідне формування з кількох БПЛА. У польотному середовищі існує ряд загроз, деякі з них відомі апріорі, а інші з’являються або стають відомими лише тоді, коли БПЛА маневрує в безпосередній близькості до загрози. Ми припускаємо, що кожен БПЛА оснащений датчиками, щоб вони могли виявляти спливаючі загрози в своєму оточенні.

Також припускається, що багаторазові БПЛА обладнані мережею зв’язку, щоб вони могли інформувати інші БПЛА про інформацію про спливаючі загрози, які вони щойно виявили. Як показано на рисунку 3.1, формування БПЛА отримує команду атакувати об'єкт противника.

БПЛА летять за заздалегідь спланованими траєкторіями з відповідною швидкістю польоту, і вони повинні прибути в один і той же час, визначений планом. Коли спливаюча загроза з’являється лише на маршруті польоту одного БПЛА і становить для нього загрозу, поточна траєкторія неможлива, або навіть небезпечна.

В цей момент БПЛА має знайти іншу нову траєкторію. Водночас, потрібно забезпечити, щоб формування з кількох БПЛА розпочало атаку одночасно та уникнути зіткнення, також необхідно, щоб інші БПЛА в з'єднанні відрегулювали

свої відповідні траєкторії, і, таким чином, скоординоване перепланування траєкторії для всього формування є неминучим. Крім того, поряд зі зміною траєкторії кількох БПЛА, відповідно до цього знову підтверджується ЕТА, а також швидкість польоту кожного БПЛА.

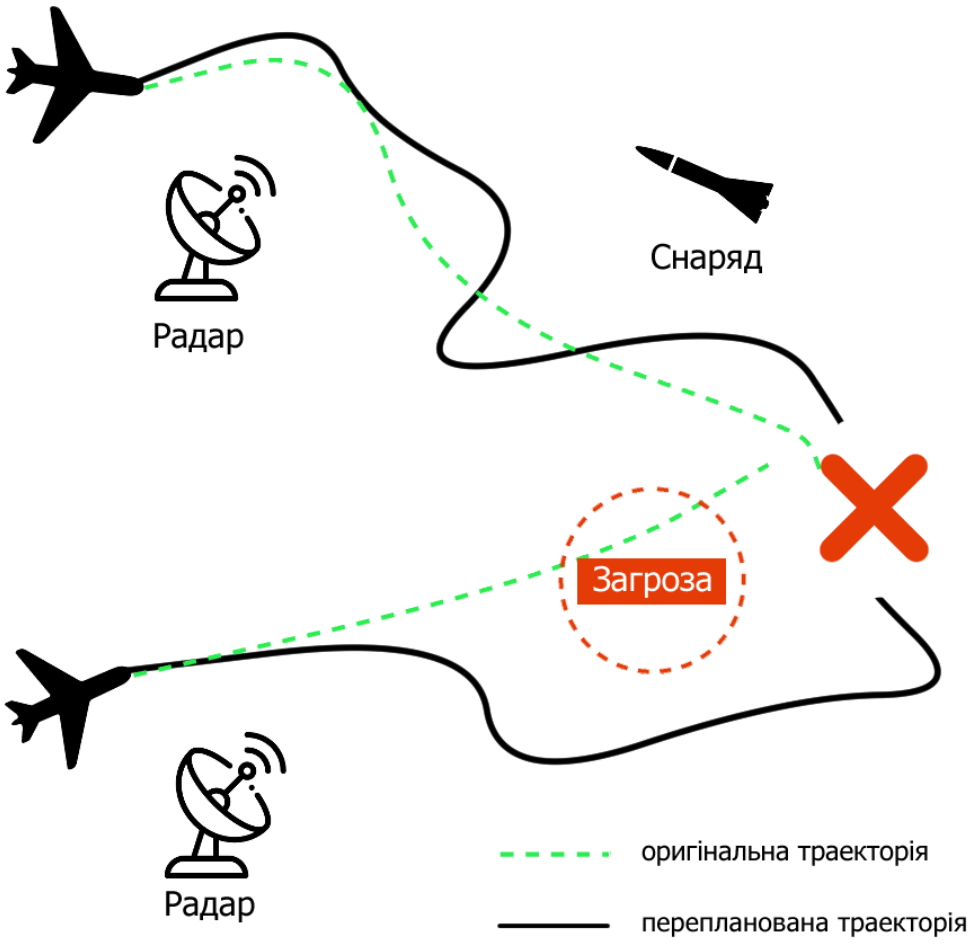


Рисунок 3.1 – Опис скоординованого перепланування траєкторії з кількома БПЛА

3.1.2 Моделювання джерел загроз

Моделювання джерел загроз є ключовим завданням у перепланування оптимальної траєкторії з кількома БПЛА [38]. Щоб спростити проблему перепланування траєкторії багаторазових БПЛА, у загрозливих діапазонах радіолокаційної та зенітної зброї противника потрібно визначити вартість загрози, яку необхідно мінімізувати при плануванні траєкторії, – це отримана радіолокаційна енергія від радіолокаційних місць, а саме: повітряний простір поля бою можна розділити на двовимірну мережеву діаграму, що з'єднує початкову точку та точку мети (показано на рис. 3.2). Таким чином, проблему оптимального планування траєкторії БПЛА можна розглядати як загальну задачу оптимізації траєкторії по суті.

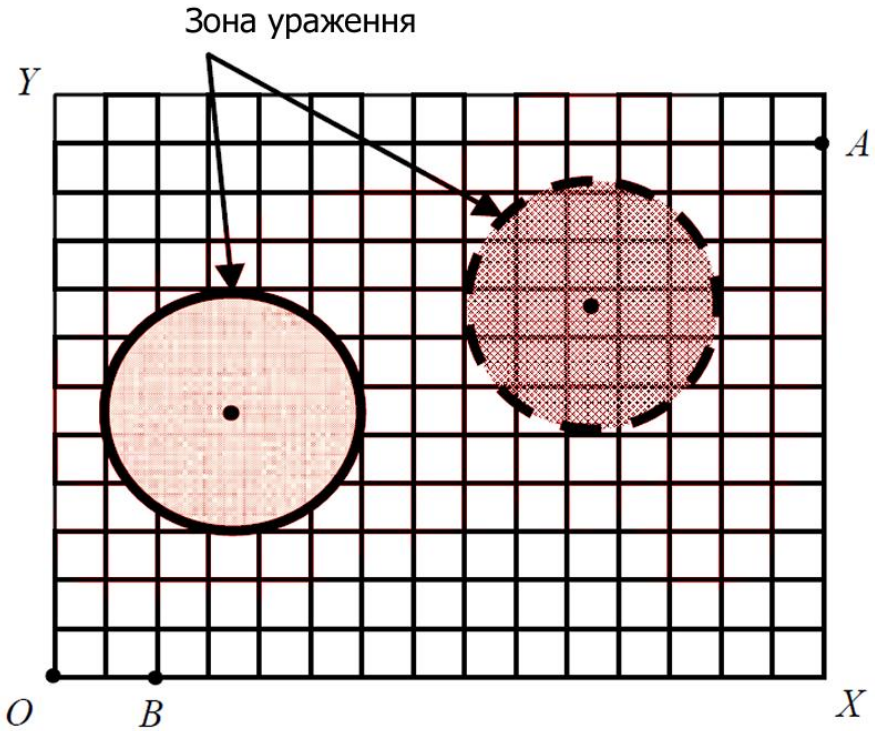


Рисунок 3.2 – Типова модель поля битви БПЛА

Припустимо, що на рисунку 3.2, політ БПЛА виконується від вузла B до вузла A. На полі бою є деякі загрозливі зони. Можна встановити систему координат з OX як вісь x і OY як вісь y, яка охоплює весь регіон завдання,

включаючи вихідні точки, цілі та всі загрозливі регіони, які можуть становити загрозу для БПЛА. Потім ми ділимо OX на m підрозділів, а OY ділимо на n підрозділів порівну. Таким чином, існує m+1 вертикальних ліній і n+1 горизонтальних ліній [39-42]. Де m+1 вертикальних ліній можна позначити L1, L2, …, Lm+1. m+1 вертикальних ліній і n+1 горизонтальних перехресних ліній складають $(\mathrm{m}+1)*(\mathrm{n}+1)$ вузли. Назвемо ці вузли $\mathrm{L}_1(\mathrm{x}_1,\mathrm{y}_1), \mathrm{L}_1(\mathrm{x}_1,\mathrm{y}_1), \ldots\ \mathrm{L}_{\mathrm{m}+1}(\mathrm{x}_{\mathrm{m}+1},\mathrm{y}_1), \ldots, \mathrm{L}_1(\mathrm{x}_1,\mathrm{y}_{\mathrm{n}+1}), \ldots\ , \mathrm{L}_{\mathrm{m}+1}(\mathrm{x}_{\mathrm{m}+1},\mathrm{y}_{\mathrm{n}+1})$ де $\mathrm{L}_\mathrm{i}(\mathrm{x}_\mathrm{i},\mathrm{y}_\mathrm{i})$ позначає точку перетину i-ї вертикальної лінії та j-ї горизонтальної лінії. Таким чином, шлях від початкової точки B до цільової точки A можна описати таким чином:

$$Path = \left\{B, L_{i_1}\left(Lx_{i_1}, y_{j_1}\right), L_{i_2}\left(Lx_{i_2}, y_{j_2}\right), \ldots, L_{i_k}\left(Lx_{i_k}, y_{j_k}\right), A\right\} \qquad (3.1)$$

де $\mathrm{i}_\mathrm{k} = 1, 2, \ldots, \mathrm{m}+1, \mathrm{j}_\mathrm{k} = 1, 2, \ldots, \mathrm{n}+1$.

Після побудови двовимірної сітки, що охоплює всю область завдання, слід розрахувати вартість загрози для кожного краю. Що стосується цих меж у загрозливих діапазонах радіолокаційної та зенітної зброї противника, то вартість загрози, яку необхідно мінімізувати при плануванні траєкторії, – це отримана радіолокаційна енергія від радіолокаційних місць, яка становить:

$$J_{ij,threat} = \int_0^{L_{ij}} \sum_{k=2}^{N_t} \frac{1}{((x-x_k)^2+((y-y_k)^2)^2}\, dl \qquad (3.2)$$

де L_{ij} позначає довжину краю, що зв'язує i-й і j-й вузли у створеній сітці;

$(\mathrm{x}_\mathrm{k}, \mathrm{y}_\mathrm{k})$ – координата k-го джерела загрози;

N_t – кількість загроз, включаючи радари, ракети та зенітні гармати, які можуть становити загрозу краю (i, j).

Однак це рівняння важко обчислити, тому обчислювальною більш ефективним і прийнятною точним наближенням до точного рішення є обчислення вартості загрози в кількох місцях уздовж краю та врахування

довжини краю. Вартість загрози для прикладу буде розраховуватись в п’яти точках уздовж кожного краю, як показано на рис. 3.3.

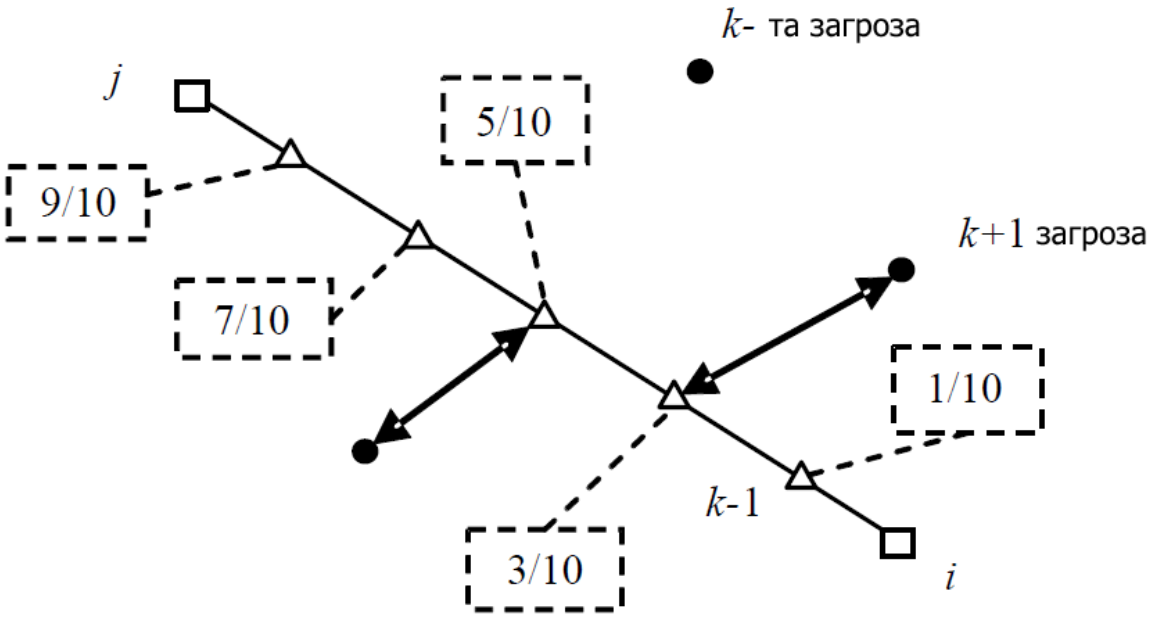


Рисунок 3.3 – Вартість загрози в п’яти точках уздовж краю

Вартість загрози, пов’язана з ребром (i, j), визначається наступним рівнянням:

$$J_{ij,threat} = L_{ij} \sum_{k=1}^{N_t} W_k \left(\frac{1}{d_{0.1,k}^4} + \frac{1}{d_{0.3,k}^4}\frac{1}{d_{0.5,k}^4}\frac{1}{d_{0.7,k}^4}\frac{1}{d_{0.9,k}^4}\right) \quad (3.3)$$

де $d_{0.1,k}$ відстань від точки 1/10 на краю (i, j) до k-ї загрози, w_k позначає ваговий коефіцієнт загрози, який використовується для оцінки рівня загрози для кожного джерела загрози [43].

Більше того, може просто вважатись, що вартість палива J_{fuel} дорівнює довжині шляху L, і, таким чином, вартість палива, пов’язана з кожним ребром $J_{ij,fuel}$, дорівнює L_{ij}. Загальна вартість проїзду уздовж краю (i, j) складається із зваженої суми загрози та витрат на паливо:

$$J_{ij} = f \cdot J_{ij,threat} + (1-f) \cdot J_{ij,fuel} (0 \le f \le 1). \quad (3.4)$$

Вибір f між 0 і 1 дає розробнику певну гнучкість у розпорядженні співвідношення між ступенем загрози та витратою палива. Коли f ближче до 0, потрібен коротший шлях, щоб зменшити витрату палива, приділяючи менше уваги загрозі з радара. Навпаки, коли f ближче до 1, це вимагає уникати загрози, наскільки це можливо, за рахунок втрати довжини траєкторії. Для цієї роботи було знайдено значення 0,75 для створення шляхів, які були збалансовані з точки зору уникнення загроз і довжини шляху.

3.1.3 Обмеження умов зміни траєкторії кількох скоординованих БПЛА

У порівнянні з плануванням траєкторії одного БПЛА, відмінність скоординованого планування траєкторії з кількома БПЛА також полягає в обмеженнях, які слід враховувати.

Окрім фізичних властивостей та вимог місії окремого транспортного засобу, координація та співпраця між різними БПЛА накладають кілька додаткових супутніх обмежень, включаючи обмеження часу, коли БПЛА повинні досягати цілей одночасно, уникнення зіткнень тощо. В основному включаються три обмеження [44].

Перше обмеження це – обмеження мінімального радіуса розвороту польоту: враховуючи маневреність БПЛА, радіус повороту в створеній траєкторії має бути більшим за мінімальний радіус повороту кожного БПЛА, що означає, що запланована траєкторія повинна уникати більшого повороту.

Встановлюється обмеження на вибір маршрутної точки (вузла), як показано на рисунку 3.4. На рисунку 3.4 центральний вузол у квадраті є поточним місцем розташування БПЛА:

1. (b) – остання точка шляху безпосередньо перед поточною.
2. (a) – враховуючи, що БПЛА не може робити занадто великий поворот, наступну точку можна вибрати лише з таких точок.

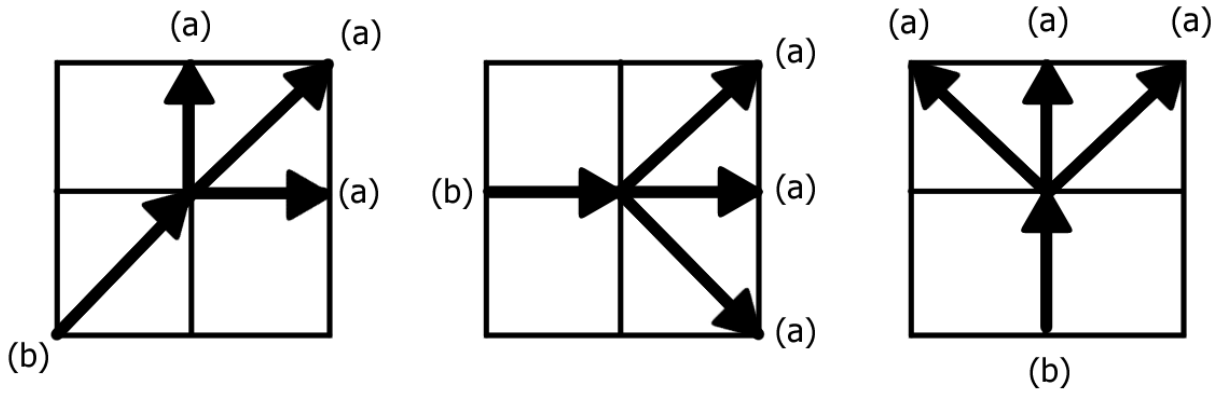


Рисунок 3.4 – Обмеження на вибір вузла

Друге обмеження це – обмеження координації часу: за такого типу обмеження координації мета полягає в тому, щоб визначити скоординований час прибуття для кількох БПЛА [45]. Траєкторія, створена для кількох БПЛА, повинна забезпечувати одночасне прибуття кількох БПЛА до своїх відповідних цілей (як показано на рисунку 3.5). Багато БПЛА повинні мінімізувати свій вплив загроз через обмеження одночасного прибуття. Тому ми повинні всебічно розглянути як довжину траєкторії, так і швидкість польоту БПЛА, щоб призначити оптимальний для команди ETA для формування кількох БПЛА.

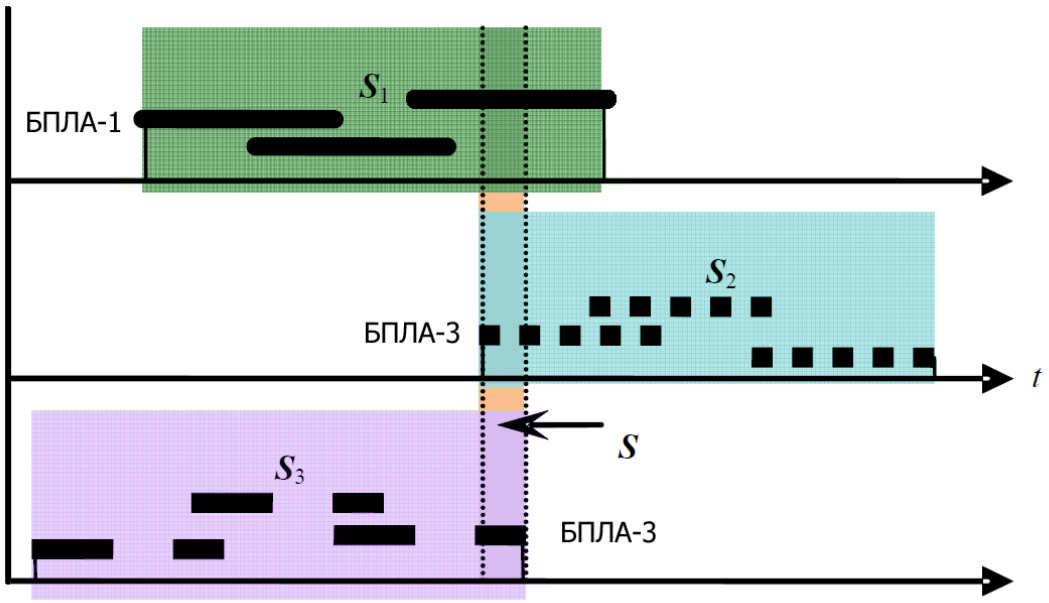


Рисунок 3.5 – Можлива зона прибуття

Припустимо, що в льотній місії беруть участь N БПЛА. Кожен транспортний засіб летить по своєму маршруту з обмеженнями швидкості v ∈ $[V_{min}, V_{max}]$. Для j-ї траєкторії, запланованої для і-го БПЛА, довжина якого позначена як $L_{i,j}$, визначаємо дальність його ETA як:

$$T_{ij} \in \left[\frac{L_{i,j}}{V_{max}}, \frac{L_{i,j}}{V_{min}}\right]. \quad (3.5)$$

Існує припущення, що і-й БПЛА сформував кількість траєкторій-кандидатів [46], і, таким чином, його очікуваний час прибуття є об'єднанням S_i, визначеним як:

$$S_i = \left[\frac{L_{i,1}}{V_{max}}, \frac{L_{i,1}}{V_{min}}\right] \cup \left[\frac{L_{i,2}}{V_{max}}, \frac{L_{i,2}}{V_{min}}\right] \cup \ldots \cup \left[\frac{L_{i,num}}{V_{max}}, \frac{L_{i,num}}{V_{min}}\right]. \quad (3.6)$$

Що стосується формування БПЛА з N транспортних засобів, то час прибуття має міститися на перетині часу таким чином:

$$S = S_1 \cap S_2 \cap \ldots \cap S_n . \quad (3.7)$$

Якщо S не є порожнечею, припустимо такий час $T_a \in S$, і кожен БПЛА повинен мати принаймні одну траєкторію, що задовольняє час прибуття T_a, і, таким чином, T_a можна просто розглядати як ETA. За допомогою цього методу він доступний для кількох БПЛА, щоб задовольнити вимогу одночасного прибуття.

Третє обмеження це уникнення повітряних зіткнень. Інша вимога щодо координації, яка стосується перепланування траєкторії координації кількох БПЛА, полягає в зменшенні ризику зіткнення, що є так званою координацією повітряного простору [47].

Оскільки дана робота заснована на двовимірному просторі з припущенням, що всі окремі БПЛА літають на однаковій висоті, правильний підхід для того, щоб гарантувати, що зіткнення не відбудеться, полягає в тому, щоб усунути

будь-яке перекриття між траєкторіями двох БПЛА. Зрозуміло, що якщо частка перекриттів у всій траєкторії більша, то ймовірність зіткнення більша. Таким чином, координована траєкторія багатьох БПЛА повинна забезпечувати, наскільки це можливо, не перекривання для реалізації координації повітряного простору.

3.1.4 Функція координації

У скоординованому перепланування траєкторії кількома БПЛА основна ідея полягає в тому, що якщо кожен транспортний засіб знає змінну координації і реагує належним чином і скоординована поведінка може бути досягнута. Для обмеження синхронізації часу одночасного прибуття, ключовою змінною координації є час прибуття.

Тобто, ключовою роботою координації кількох БПЛА є вибір належного коефіцієнта з часового перетину S, визначеного в рівнянні (3.7), як значення координаційної змінної. Для цього необхідно побудувати координаційну функцію J_{co} для визначення координаційної змінної.

$$J_{co,i,j} = f_1 \cdot J_{i,j} + f_2 \cdot T_{i,j} . \tag{3.8}$$

Де f_1 та f_2 дві константи, які можуть бути однаковими або різними для різних БПЛА. Змінна $J_{i,j}$ позначає вартість j-ї траєкторії, запланованої і-м БПЛА, яка визначається рівнянням $f \cdot J_{i,j,threat} + (1 - f) \cdot J_{i,j,threat}\ (0 \le f \le 1)$, що є визначеним для певної траєкторії [48]. Таким чином, час прибуття $T_{i,j}$ є єдиною незалежною змінною, що визначає $J_{co,i,j}$ Повні затрати багаторазових БПЛА становить:

$$J_{co} = \sum_{i=1}^{N} \sum_{j=1}^{Num} J_{co,i,j}\ (T_{i,j}) . \tag{3.9}$$

На рисунку 3.6 показано співвідношення між $J_{co,i,j}$ та $T_{i,j}$ для кожної траєкторії. Як показано на рисунку 3.6, щоб мінімізувати всю функцію координації, координований час прибуття T_a часто вибирає мінімум можливої області, тобто $T_a = \min S$.

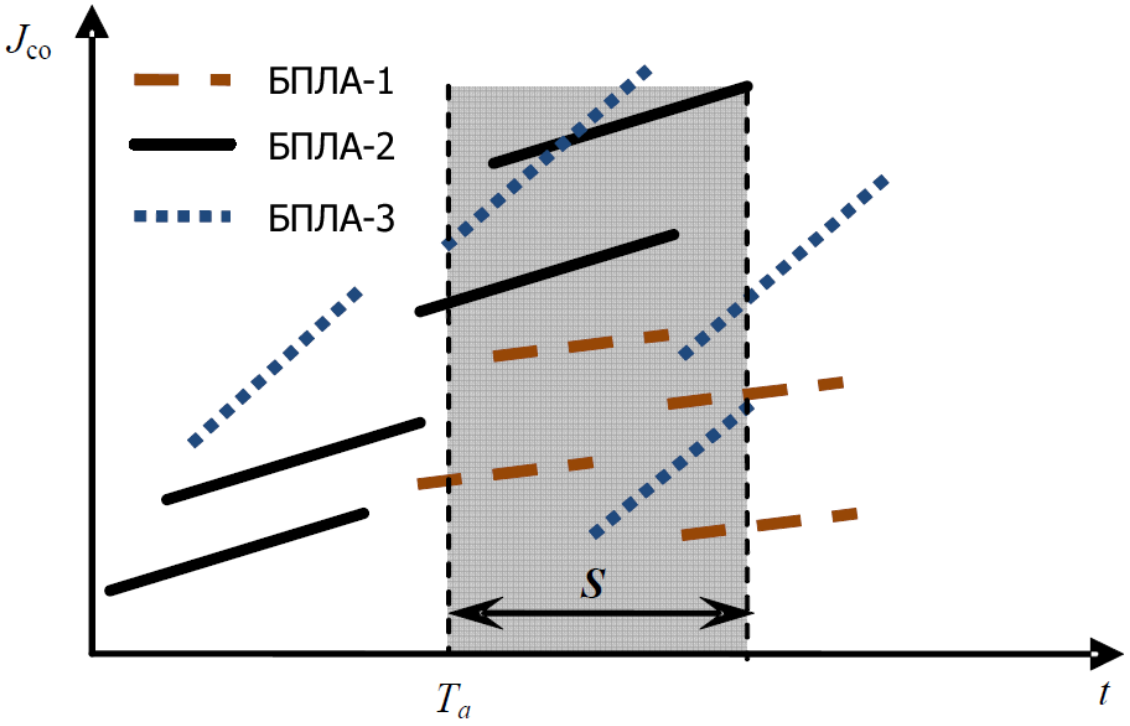


Рисунок 3.6 – Визначення координаційної змінної

3.1.5 Координаційний механізм перепланування траєкторії багатопланових БПЛА

Структура планування траєкторії для окремого БПЛА складається з трьох основних шарів: координаційного вирішувача, планувальника траєкторії та зглажувача траєкторії [49-53]. Планувальник траєкторій швидко обчислює серію досить безпечних прямолінійних траєкторій.

Зв'язок між планувальником траєкторії та координаційним вирішувачем може допомогти генерувати кандидатські траєкторії, уникаючи збігу з траєкторіями інших БПЛА.

Ці кандидатські траєкторії використовуються координаційним рішенням для визначення координаційної інформації, такої як координований час.

Оскільки прямолінійні траєкторії, створені планувальником траєкторії, динамічно неможливі для польоту БПЛА, згладжувач траєкторії використовується для генерування льотних траєкторій і надсилання команд автопілоту БПЛА. Функція динамічного згладжувача траєкторії полягає в згладжуванні переходів на траєкторії з послідовністю радіальних дуг, по яких БПЛА може летіти [54]. Для місій, які мають важливе значення для часу, важливо, щоб довжина вихідної прямолінійної траєкторії була збережена в процесі згладжування. На рисунку 3.7 показаний цей механізм.

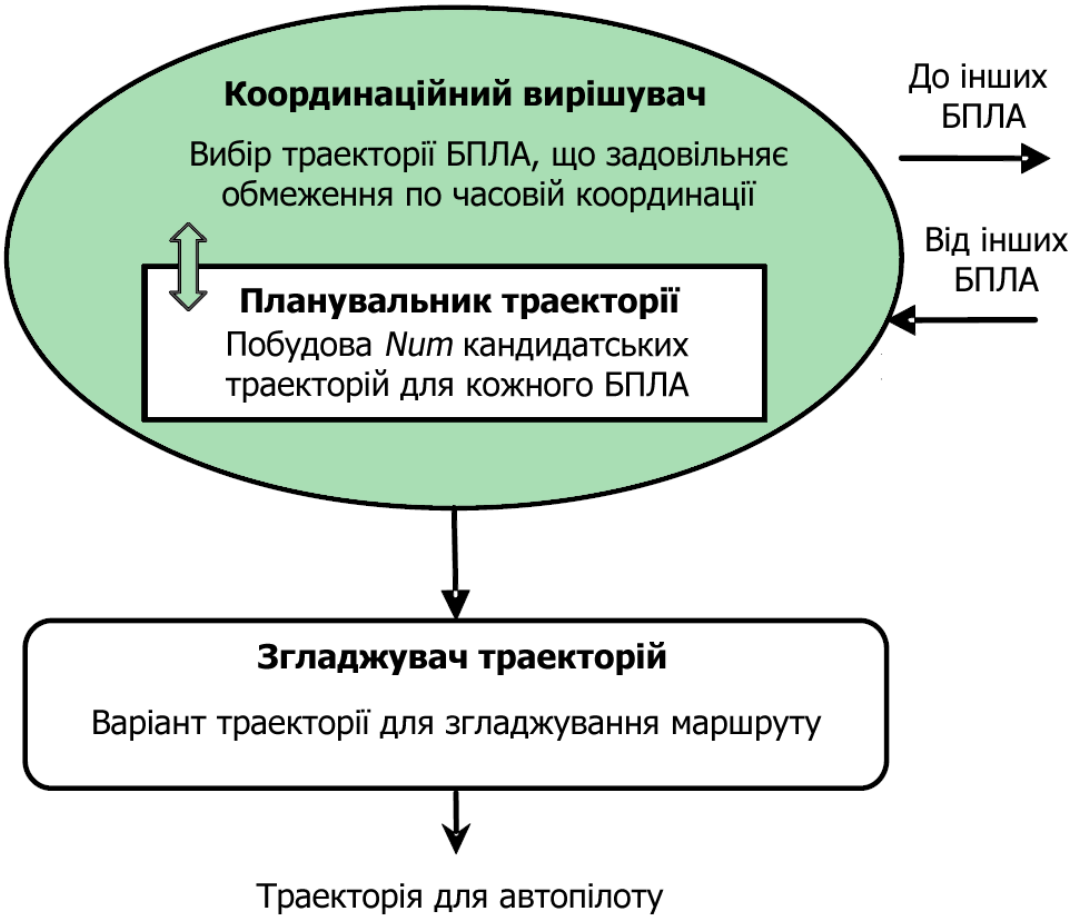


Рисунок 3.7 – Механізм планування траєкторії для окремого БПЛА

На рисунку 3.8 описується механізм координації перепланування траєкторії кількох БПЛА. Структура розподілена, що дозволяє кожному БПЛА виконувати власні підсистеми планування траєкторії. Алгоритм, реалізований на кожному БПЛА в координаційному вирішувачі, ідентичний. Багато БПЛА

передають інформацію про координацію у формі функції координації та змінної координації на рівні вирішувача координації. Використовуючи координаційну інформацію, багато БПЛА обчислюють скоординований час прибуття для формування БПЛА, який є лише ETA.

Після цього координаційний дешифратор обчислює швидкість польоту для кожного БПЛА відповідно до згенерованої траєкторії та ETA. Хоча на рис. 3.8 показано лише три БПЛА, розподілена структура каркаса, очевидно, стосується більшої кількості багаторазових БПЛА.

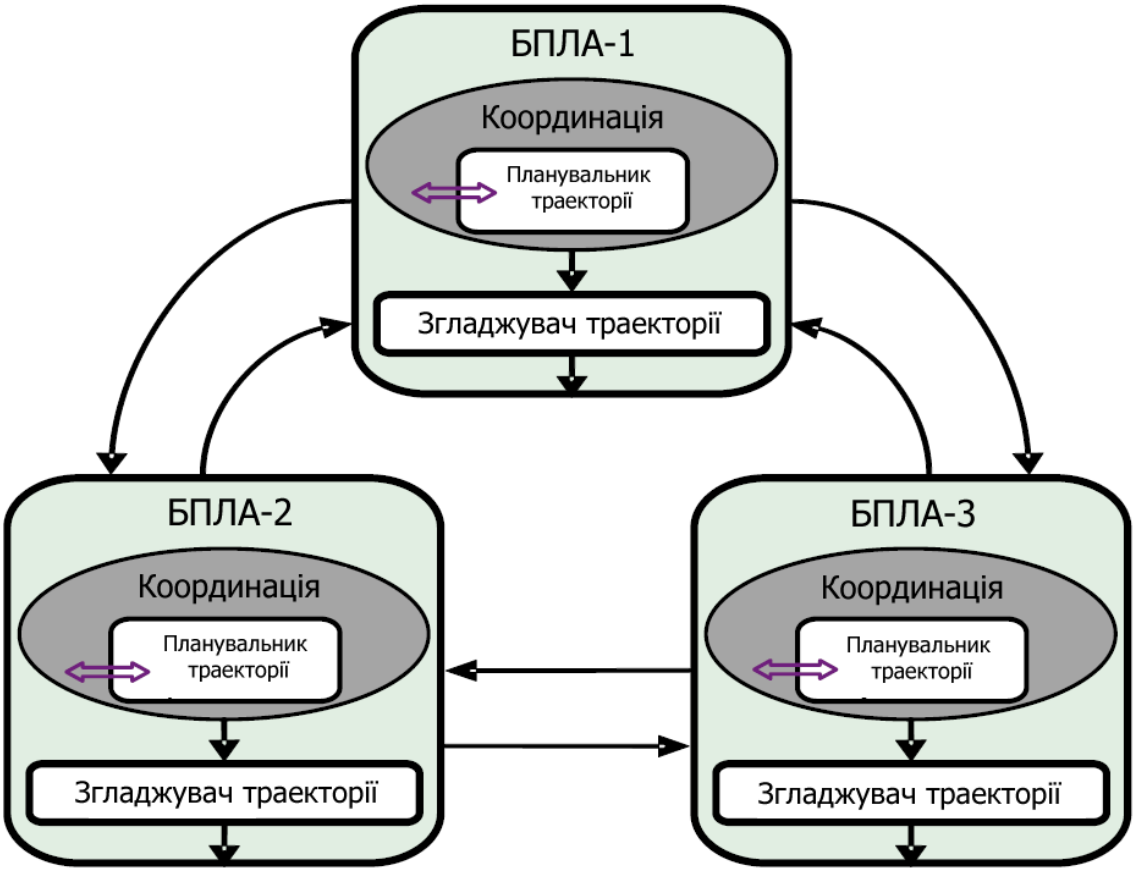


Рисунок 3.8 – Координаційний механізм перепланування траєкторії багаторазових БПЛА

Крім того, сучасні БПЛА оснащені можливістю виявляти своє оточення, щоб вони могли відчувати спливаючі загрози, що виникають у регіоні місії. Причина перепланування траєкторії полягає в наступних аспектах: спливаюча

загроза виявляється тільки на маршруті польоту попереду, і транспортний засіб повинен змінити свій маршрут для безпеки; БПЛА не загрожує спливаючою загрозою, але з огляду на координацію він також отримує команду перепланування; місія змінюється.

Коли перепланування стане неминучим, інформація про спливаючі загрози, включно з розташуванням і класами загроз, незабаром буде надана кількома БПЛА. Потім кожен БПЛА уповільнює або прискорюється і летить до сусіднього безпечного вузла, який служить відправною точкою для наступного перепланування.

Протягом цього проміжку нова перепланована траєкторія генерується для кількох БПЛА, а також новий ETA. У процедурі перепланування, на основі початкової вартості краю двовимірної сітки, першим кроком є оновлення початкової вартості загроз тих країв, яким загрожують спливаючі загрози, та перерахунок їх значення функції вартості [55].

Потім багаторазові БПЛА починають реалізовувати узгоджене перепланування траєкторії від нових вихідних точок до впорядкованих цілей. У цій процедурі, як показано на рисунку 3.9, необхідно відтворити набір нових траєкторій, і БПЛА також повинні перерахувати ETA та повторно відрегулювати відповідну швидкість польоту. Процедура перепланування траєкторії, можливо, буде проводитися не один раз через непевну зміну складного бойового середовища [56].

Враховуючи нове перепланування, БПЛА будуть аналізувати зовнішнє середовище на явні загрози, та вирішувати їх пріоритетність в залежності від типу загрози. В будь якому випадку феромонний слід мурах буде змінюватись і наступні БПЛА будуть рухатись по новій траєкторії враховуючи нові загрози.

Також нові загрози будуть класифікуватись в залежності їхньої частити та шкоди для групи БПЛА і загрозі зриву місії. Кожній загрозі буде кваліфікована оцінка загрози, згідно якої буде змінюватись поведінка та траєкторія руху БПЛА, враховуючи усі можливі зміни маршрутів по заданій траєкторії.

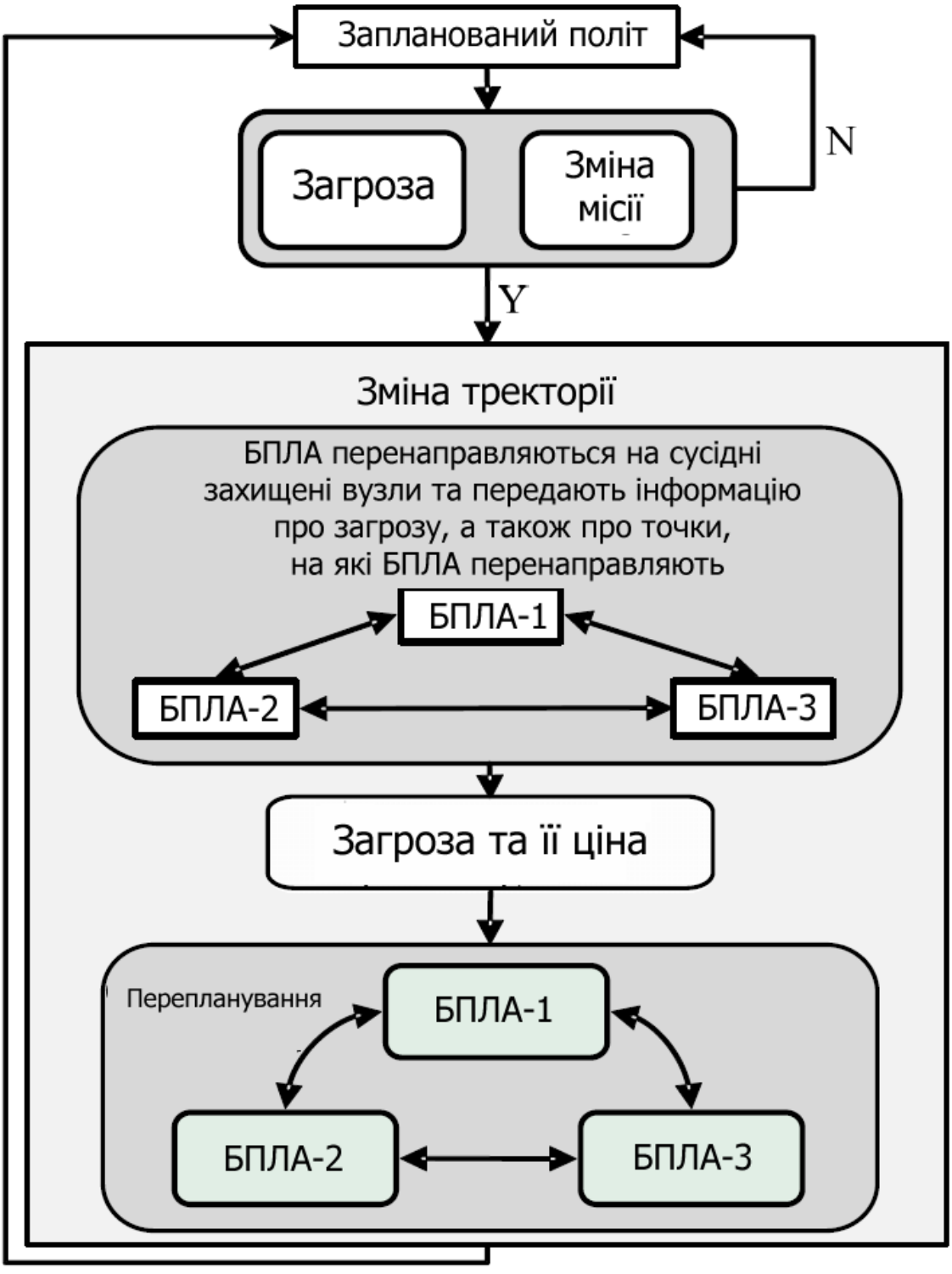


Рисунок 3.9 – Процедура перепланування траєкторії БПЛА

3.2 Вирішення проблеми повітряного трафіку

3.2.1 Моделювання проблеми

Типова задача, використана прикладу, описана на рисунку 3.10. Розглянемо дану проблемо окремо від минулого підрозділу із подальшою імплементацією проблеми у вирішення глобальної задачі. Певна кількість (n) БПЛА розташовані на колі радіуса R і летять до центру кола з однаковою швидкістю. Їхнє призначення – це точка кола, розташована на іншому кінці діаметра кола, на якому вони летять. Відомо, що цю проблему важко розв’язати, оскільки кожен БПЛА чи їх група, конфліктує з кожним іншим. Мета полягає в тому, щоб знайти нові траєкторії для кожного БПЛА, які вирішують кожен конфлікт і мінімізують додаткову відстань.

При моделюванні на БПЛА дається один маневр. Час дискретизовано. Маневр починається в певний час T_1і закінчується в деякий час T_2. Конфлікти вирішуються горизонтально: БПЛА надається зміна курсу на 10, 20 або 30 градусів вправо або вліво (рисунок 3.11).

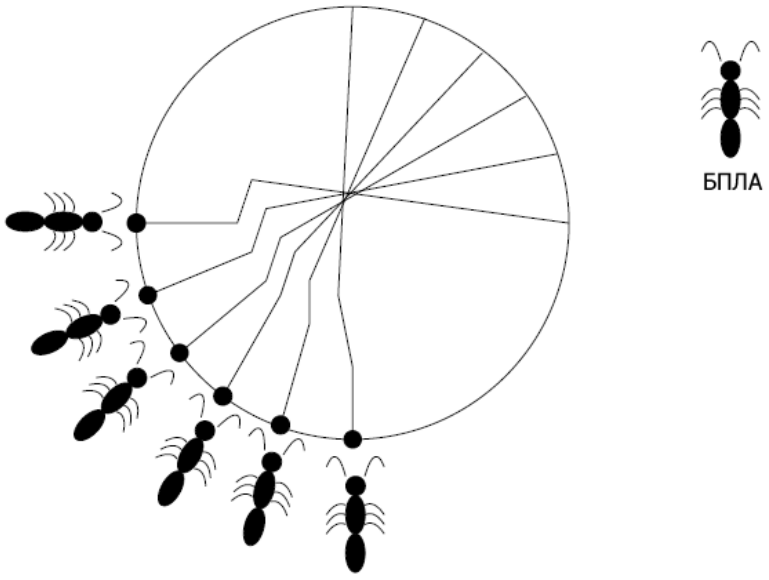


Рисунок 3.10 – Конфлікт трафіку БПЛА на прикладі мурах

Траєкторію ПБЛА можна змоделювати за допомогою графіка. Час дискретизовано в n_t часових кроків. Графік позиції БПЛА можна визначити так:

кожен вузол представляє час і позицію БПЛА. Перехід із положення і в положення і+1 представлено ребром, на якому мурахи відкладають феромони. БПЛА може перебувати послідовно в трьох станах. БПЛА перебуває в стані U перед будь-яким маневром [57-58].

Коли маневр починається в T_1, він знаходиться в стані V. Нарешті, коли маневр закінчується в T_2, БПЛА перебуває в стані W. Якщо U_i, V_i і W_i є числом можливих позицій БПЛА відповідно в стані U, V і W у момент часу і, тоді:

$$U_{i+1} = U_i, V_{i+1} = V_i + 6.\ W_i = V_i. \tag{3.10}$$

Якщо вважати, що одна мураха являє собою одне рішення конфлікту АСО, то для n_a літака та n_t кроків часу кількість можливих слідів у момент і дорівнює $(12i - 5)^{n_a}$, а загальна кількість можливих слідів є $(12n_t - 5)^{n_a}$. Для $n_a = 5$ і $n_a = 10$ можна отримати більше 1010 слідів. Дане моделювання розглядає групу варіантів для вирішення конфліктної проблеми БПЛА.

Мурахи обробляються незалежно, за винятком обчислення кількості феромонів, які відкладаються, що залежить від кількості конфліктів, які зміг вирішити кожен мураха з групи. Це моделювання зменшує кількість слідів для оновлення до $(12n_t - 5)^{n_a}$). Для $n_a = 5$ і $n_a = 10$ кількість слідів для оновлення становить 575, що набагато менше, ніж 1010. Для $n_a = 30$ і $n_a = 20$ кількість слідів для оновлення становить лише 7050 замість 1071 за попереднім моделюванням [59].

3.2.2 Часткова імплементація алгоритму

Алгоритм АСО – це імовірнісний метод розв’язування обчислювальних задач, який можна звести до пошуку хороших шляхів за допомогою графів. Як вже було загадано раніше перші алгоритми були протестовані на задачі комівояжера.

Алгоритм імітує поведінку мурах, які шукають шлях між своєю колонією та джерелом однієї мурахи для n БПЛА, одна мураха на БПЛА. На рисунку 3.10 – одна мураха на кластер або одна мураха на БПЛА.

Як було сказано у минулому розділі, у описі роботи алгоритму мурахи намагаються дістатись цілі, минуючи перешкоди. З тих пір ця ідея була урізноманітнена для вирішення більш широкого класу чисельних задач.

Мурахи використовують навколишнє середовище як засіб спілкування. Вони обмінюються інформацією, відкладаючи феромони. Обмін інформацією має локальний масштаб. У задачі комівояжера перші мурахи вибирають свій шлях випадковим чином і відкладають все більше феромонів, оскільки обраний шлях короткий.

Потім нові мурахи обирають свій шлях, враховуючи кількість феромонів, які вони знаходять локально. Чим більше феромонів вони знайдуть на заданому шляху, тим імовірніше, що вони його оберуть. Розсіювання шляху випаровуванням феромонів запобігає передчасній конвергенції алгоритму. ACO, який тут використовується, є класичним ACO. Відмінність полягає в тому, що мураха замінюється купою на мурах, що представляють БПЛА. Мураха може перебувати в трьох різних станах, як показано на рисунку 3.11.

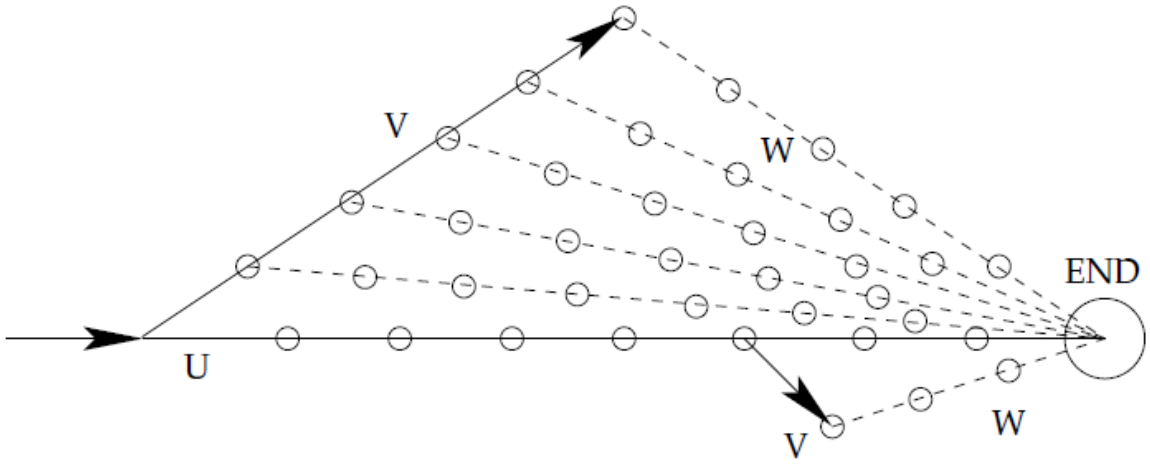


Рисунок 3.11 – Моделювання руху БПЛА

Ці стани можна описати наступним чином:

1. Перед будь-яким маневром мураха знаходиться в стані U.
2. Після T_0 він змінює свій заголовок і переходить у стан V.
3. Після T_1 він змінює свій заголовок і переходить у стан W.

У кожному вузлі графіка, що представляє можливі траєкторії БПЛА, мураха вибирає наступний вузол з імовірністю, що залежить від кількості феромонів, що залишилися на краю, що з'єднує два вузли.

Потім траєкторії перевіряються, щоб перевірити наявні конфлікти. Мурахи, що представляють конфліктні БПЛА, не відкладають жодних феромонів, тоді як мурахи, які представляють БПЛА без конфлікту, відкладають феромони. Кількість феромонів, що відкладаються, зменшується із затримкою через маневр БПЛА.

Графік можливих шляхів будується для того, щоб прийняти максимальну затримку для кожного БПЛА. На початку алгоритму початкові феромони розподіляються по графіку, щоб забезпечити рівну ймовірність для кожного обраного шляху [60].

На рисунку 3.12 наведено приклад розподілу початкових феромонів, що забезпечує рівні шанси для кожного шляху. У цьому простому прикладі БПЛА може повернути вліво або вправо (30 градусів) або йти прямо, і представлено лише кілька кроків.

Таким чином, кількість феромонів на кожному краю пропорційна кількості можливого шляху, що залишився після проходження через це ребро. Починаючи з END, стан W отримує одиницю феромонів, а потім феромони додаються до кожного вузла, щоб заповнити весь графік.

Кожний крок вузла має сталий градус змін, але може варіюватись в залежності від умов, початкових даних та допустимих втратах при виконанні місії.

Підсумовуючи кількість феромонів БПЛА вибирає оптимальний маршрут, або створює новий викидаючи необхідну кількість нових феромонів.

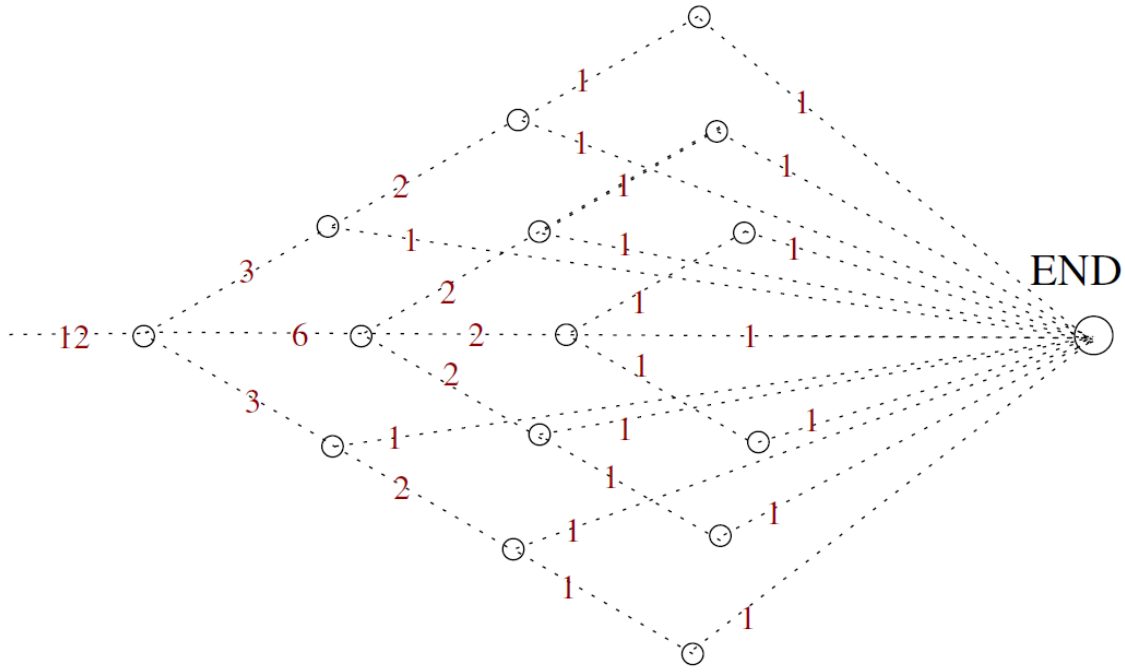


Рисунок 3.12 – Кількість феромонів згідно графу

Якщо n_a – кількість БПЛА, створюється група $p \times n_a$ мурашок, щоб представити кожен БПЛА у кожному поколінні. Таким чином, загальна кількість мурах дорівнює $p \times n_a^2$ а (у прикладі p = 10).

Кожен шлях отримує бал. Чим менший бал, тим кращий шлях. Оскільки пряма лінія є найкоротшим шляхом, проходження через стан U не змінює рахунок. Стан V додає 2 бали, а стан W – 1 бал. Ця оцінка дає перевагу маневрам, які починаються із запізненням: коли БПЛА перебуває в стані W, бал збільшується, тоді як у стані U – ні.

У кожному вузлі мураха вибирає наступне ребро з імовірністю, що залежить від кількості феромонів, що залишилися на наступному ребрі. Якщо мураха конфліктує з іншою мурахою, вона не відкладає ніяких феромонів. Однак, коли конфлікту немає, він залишає за собою кількість феромонів, рівну:

$$\Delta\tau = \frac{n_a - n_{out}}{n_a} \cdot \frac{\tau_0}{s_{path}} \qquad (3.11)$$

де n_{out} – це кількість «втрачених» мурах, τ_0 – початкова кількість феромонів, а s_{path} – оцінка шляху, яким йде мураха. Ця сума враховує кількість мурах, які нарешті знайшли дійсний шлях.

Після кожного покоління і перед запуском нового покоління на існуючих трасах застосовується принцип випаровування. Кількість феромонів зменшується на x% (у прикладах x = 10%) в кінці кожного покоління. Алгоритм закінчується, коли оцінка, отримана кожною групою мурах, що представляють кожен БПЛА, не зменшується деякий час або коли закінчується час, відведений для алгоритму.

Враховуючи цю загальну теорію, уніфікований алгоритм, та детально описаний принцип керуванням групою БПЛА для досягнення цілі найшвидшим шляхом можемо перейти до реалізації алгоритму, але перед цим потрібно описати принцип посадки БПЛА.

3.3 Вирішення задачі посадок БПЛА

Для того, щоб застосувати алгоритм колонії мурах для проблеми посадок БПЛА, використаємо граф зображений на рисунку 3.13. Цей рисунок базовано на дворівневому графіку. На першому рівні фіксуємо наявні злітно-посадкові смуги, а на другому – БПЛА. Додаємо два фіктивні вузли D і F, що відповідають відповідно до початку і кінця графа.

Кожний БПЛА має необмежену можливість вибору маршрутів, на початку використання алгоритму, при створенні нових феромонів маршрут обирається випадково, або згідно заданих початкових даних, які основані на минулих дослідженнях.

При цьому потрібно не забувати при супутні загрози. При достатній комунікації між БПЛА, проблем виникати не буде, але за наявності обмеженої кількості даних, інші БПЛА можуть вважатись як достеменно відомі загрози із низьким пріоритетом.

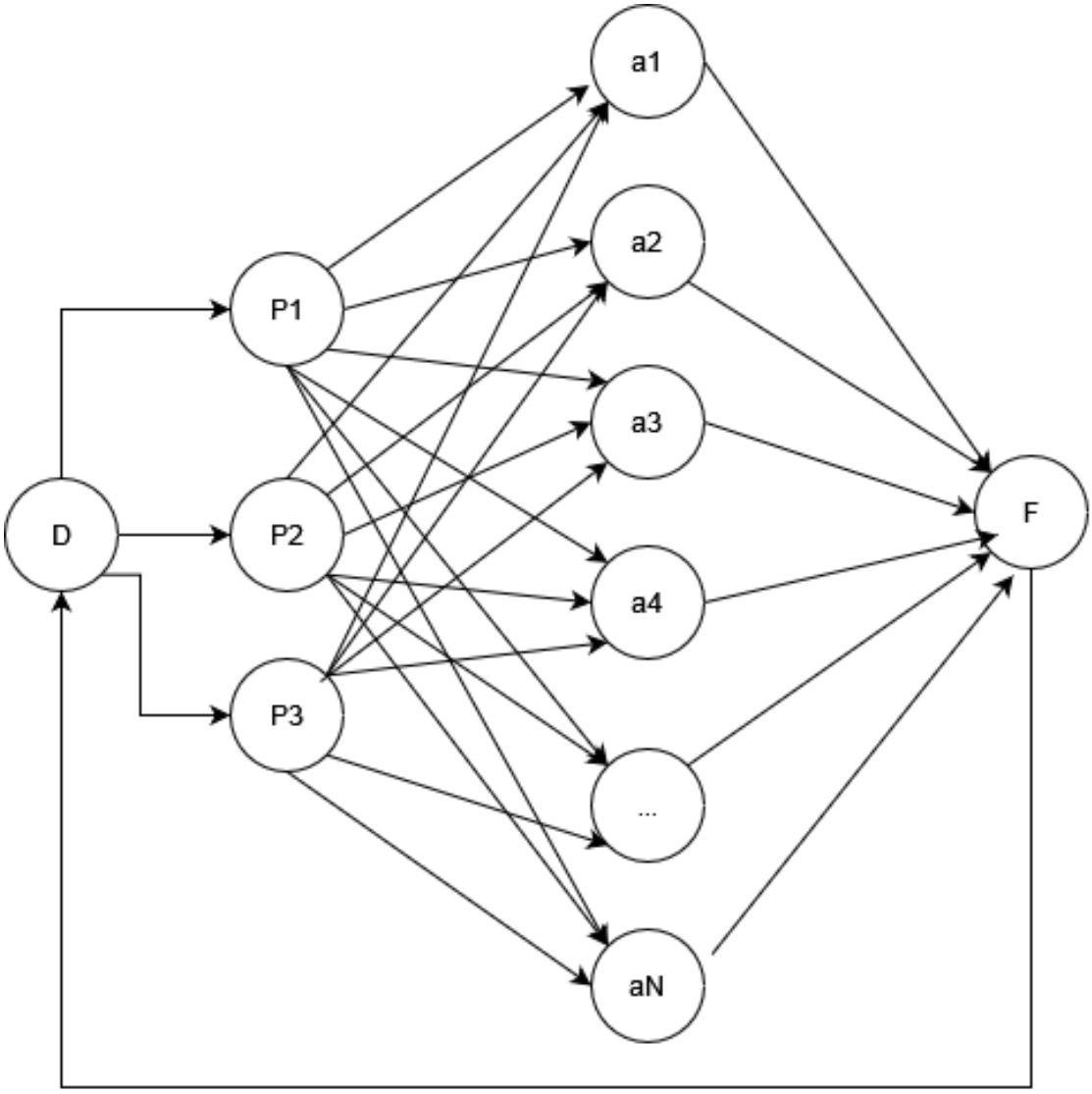


Рисунок 3.13 – Графічна репрезентація проблеми посадок БПЛА

Мураха починає свою траєкторію з початку графіка, що відповідає фіктивному вузлу D. Спочатку він вибирає злітно-посадкову смугу, куди вставить наступний БПЛА; цей вибір може залежати від зайнятості на злітно-посадковій смузі або злітно-посадкової смуги, яка швидше стане вільною.

Після вибору злітно-посадкової смуги мураха має вибрати наступний БПЛА для посадки на цю злітно-посадкову смугу; цей вибір суттєво залежить від пріоритету БПЛА в порівнянні з іншим літальним апаратом та пам'яті колонії мурах. Цей процес повторюється до тих пір, поки не буде доступних для посадки БПЛА.

Спочатку йде вибір посадкової області. Правило ймовірності вибору злітно-посадкової смуги r з початку графіка D можна виразити наступним рівнянням:

$$p_{Dr}^{k} = \begin{cases} argmin_{r=1,\ldots,R}(\text{кількість бпла}), \text{якщо } q < q_0 \\ r_0, \text{ інакше} \end{cases} \quad (3.12)$$

де D являє собою вузол, що відповідає початку графіка;

$0 < q_0 < 1$ є константою алгоритму забезпечення диверсифікації;

q – значення, випадково взяте в [0,1];

r_0 – індекс, вибраний випадковим чином у {1, …, R}.

У цьому випадку ми вибираємо злітно-посадкову смугу з найменшою кількістю БПЛА, не враховуючи інтервалів безпеки. Це правило ймовірності може бути корисним, якщо інтервали безпеки однакові для всіх БПЛА, але насправді вони залежать від природи БПЛА. Тому пропонується інше правило ймовірності, яке вибирає злітно-посадкову смугу відповідно до часу її готовності для прийому нових БПЛА [61]. Це виражається таким рівнянням:

$$p_{Dr}^{k} = \begin{cases} argmin_{r=1,\ldots,R}\left(min_{j \in \text{кандидати}} \left(x_{i0}^{r} + S_{i_0 j}\right)\right), \text{якщо } q < q_0 \\ r_0, \text{ у іншому випадку} \end{cases} \quad (3.13)$$

де кандидати – це список літаків, які чекають на посадку;

i_0 останнє повітряне судно, яке залежить від злітно-посадкової смуги r;

x_{i0}^{r} – час приземлення останнього повітряного судна, який залежить від злітно-посадкової смуги r для мурахи k.

Після вибору посадкової області чи смуги, потрібно обрати який БПЛА буде сідати. Цей вибір залежить від двох параметрів.

Перший – це пріоритет БПЛА, такий як час найранішої посадки, цільовий час, штрафна вартість БПЛА. По-друге, вартість, розрахована після того, як ми призначили БПЛА час на посадку за алгоритмом призначення. Вагою цих двох параметрів є «інформаційна евристика» алгоритму мурашиної колонії.

$$\eta_{rj} = \left(\frac{1}{\text{пріорітет}(j)+1}\right)^{\beta_1} \cdot \left(\frac{1}{\text{ціна\_ризику}(j)+1}\right)^{\beta_2} \quad (3.14)$$

де пріорітет(j) – пріорітет БПЛА j;

ціна_ризику(j) – ціна ризику для БПЛА;

β_1 та β_2 – коефіцієнти зважування.

Іншим параметром, який впливає на вибір мурахи, є пам'ять колонії (тобто сліди феромонів, зазначені τ_{ij}, спочатку фіксувалися на значенні τ_0). Підводячи підсумок, правило ймовірності вибору БПЛА виражається так:

$$p_{rj}^k(t) = \begin{cases} \frac{(\tau_{rj})^{\alpha} \cdot (\eta_{rj})^{\beta}}{\sum_l (\tau_{rj})^{\alpha} \cdot (\eta_{rj})^{\beta}}, & \text{якщо } j \in \text{кандидати} \\ 0, & \text{у іншому випадку} \end{cases} \quad (3.15)$$

У цьому випадку α і β визначають відносну важливість феромона слід і видимість мурах.

Після цього необхідно визначити час посадки БПЛА. Цей крок полягає в тому, щоб призначити літакам час приземлення, дотримуючись двох обмежень:

1. Час посадки має бути в межах посадки окремого БПЛА $[e_i, l_i]$.
2. Необхідно дотримуватись інтервалу безпеки.

Потрібно використати дві евристики для визначення часу посадки літака. Перша евристика призначає цільовий час приземлення, якщо вона дотримується інтервалів безпеки між попередніми БПЛА, в іншому випадку вона призначає найбільш ранній час, який дотримується інтервалів безпеки. Його можна виразити так:

$$t_j = max\left(ta_j, max_{i \in O}\left(t_i + S_{ij}\right)\right). \quad (3.16)$$

У даній формулі O – набір літаків, яким раніше був призначений час посадки.

Друга евристика, яка використовується, стосується лише інтервалів безпеки між іншими БПЛА та дотримання часового вікна приземлення:

$$t_j = max\ (e_j, max_{i \in O}(t_i + S_{ij})). \quad (3.17)$$

Обидва вирази використовуються для призначення часу посадки БПЛА на одній злітно-посадковій смузі, що пояснює використання матриці безпеки (S_{ij}).

Шлях феромонів оновлюється в кінці кожної ітерації відповідно до наступного рівняння:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij}(t). \quad (3.18)$$

У рівнянні (3.18) слід виділити:

1. ρ – коефіцієнт випаровування (ρ < 1, щоб уникнути необмеженого накопичення слідів).

2. $\Delta\tau_{ij}$ – кількість сліду, залишеного на краю (i, j) колонією в кінці ітерації, де $\Delta\tau_{ij} = \frac{Q}{C}$, при найкращому рішенні, та 0 у іншому, Q – оновлювана константа, C – штрафна вартість найкращого рішення в ітерації t.

Можна підсумувати алгоритм колонії мурах для посадки БПЛА наступним чином:

1. Ініціалізація матриці феромонів.
2. Для кожного БПЛА – ініціалізація кандидатури посадки на кожне місце посадки.
3. Ініціалізація усіх БПЛА.
4. Виконання розрахунків згідно наведених вище формул, до посадки усіх БПЛА.
5. Утилізація феромонів.

3.4 Висновки

У розділі представлено метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму

В результаті роботи була сформована модель скоординованого планування оптимальної траєкторії кількох БПЛА, яка включає в себе моделювання можливих загроз для БПЛА, розподілення їх по типу прогнозованих та не прогнозованих, та можливі методи уникання цих загроз.

Також було описано можливі функції координації БПЛА між собою, в результаті чого було описано проблему вирішення повітряного трафіку БПЛА, та способи уникнення конфліктів у повітрі.

Для вирішення проблеми посадок БПЛА було описано алгоритм дій БПЛА, та проаналізовано можливі випадки та способи їх рішення.

В результаті досліджень, було вирішено основні проблеми контролю групи БПЛА у повітрі, для швидкого виконання місій із можливістю уникання загроз як в повітрі так і на посадковому майданчику.

4 ПРОГРАМНО-ТЕХНІЧНИЙ ЗАСІБ ДЛЯ КЕРУВАННЯ ГРУПОЮ БЕЗПІЛОТНИХ ЛІТАЛЬНИХ АПАРАТІВ НА ОСНОВІ НЕЧІТКОГО МУРАШИНОГО АЛГОРИТМУ

4.1 Застосування адаптивного ACO Max-Min для скоординованого перепланування траєкторії кількох БПЛА

4.1.1 Принципова модель ACO з вдосконаленими стратегіями

Паралельний механізм ACO в основному містить два основних процеси: адаптацію та кооперацію. У процесі адаптації рішення-кандидати продовжують перебудовувати свої структури на основі накопичення інформації. Перебуваючи на стадії співпраці, рішення-кандидати обмінюються інформацією, щоб створити кращі рішення.

Алгоритм ACO був натхненний спостереженням за поведінкою колоній мурашок в пошуку їжі, і на цьому мурахі часто можуть знайти найкоротший шлях між джерелом їжі та своїм гніздом. Принцип цього явища полягає в тому, що мурахи відкладають на землю хімічну речовину (звану феромоном), таким чином вони позначають шлях по феромонному сліду.

У цьому процесі використовується свого роду механізм позитивного зворотного зв’язку. Мураха, що зустрічає раніше прокладений слід, може виявити концентрацію сліду феромонів. Він з великою ймовірністю вирішує піти найкоротшим шляхом і підкріпити слід власним феромоном.

Чим більша кількість феромона знаходиться на певному шляху, тим вище ймовірність того, що мураха вибере цей шлях, і випробування феромонів на цьому шляху стане більш щільним. Нарешті, колонія мурах разом позначає найкоротший шлях, який має найбільшу кількість феромонів. Такий простий непрямий спосіб спілкування між мурахами насправді втілює механізм колективного навчання [62]. На рисунку 4.1, змоделюємо діаграму принципу, згідно з яким мурахи використовують феромон для встановлення найкоротшого шляху від гнізда до джерела їжі і назад.

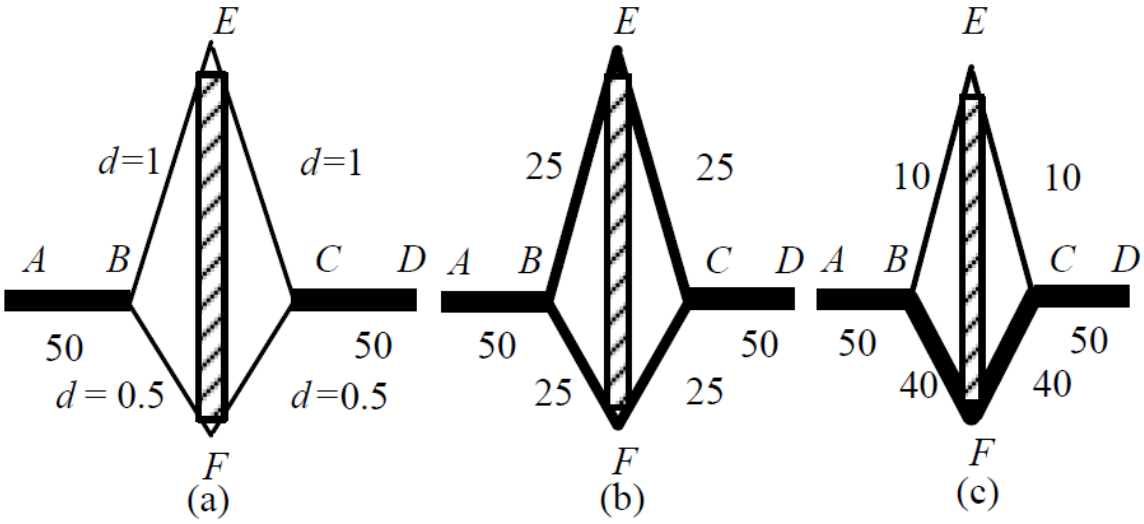


Рисунок 4.1 – Діаграма руху на прикладі трьох перешкод

Модель ACO вперше була застосована до TSP, яка полягає в пошуку найкоротшого замкнутого циклу, який проходить усі включені місця рівно один раз. У той час як планування траєкторії БПЛА полягає у пошуку оптимальної або неоптимальної безпечної траєкторії польоту, за якою БПЛА зможе виконати заздалегідь обумовлене завдання та уникнути ворожих загроз. Між TSP та плануванням траєкторії БПЛА є деякі спільні точки, і ACO є можливим способом вирішення проблеми планування траєкторії БПЛА у складних умовах бойового поля.

Для і-го транспортного засобу у формуванні N БПЛА, нехай m мурах буде в його початковій точці, мурахи виберуть наступні вузли мережі сітки відповідно до правила переходу. Після відходу мурахи від феромону, який може відчути наступна мураха як сигнал впливу на його дію, феромон, який залишив наступний, посилить початковий феромон. Таким чином, чим більше мурах проходить край сітки, тим вище ймовірність того, що край може бути обраний іншими мурахами. Цей процес може гарантувати, що майже всі мурахи пройдуть найкоротшим шляхом БПЛА.

Ключовими факторами в ACO, що впливають на поведінку мурах, є феромон τ і евристична бажаність η. Опишемо евристичну бажаність від вузла s до вузла u як:

$$\eta_{su} = \frac{1}{J_{su} \cdot d_{u,\text{ціль}}}. \quad (4.1)$$

Де J_{su} – загальна вартість краю (s, u), а d_u, ціль являє собою відстань від вузла u до цілі, яка використовується для того, щоб мурахи, розташовані у вузлі s, прагнули вибирати ті вузли, які знаходяться ближче до цілі.

Кількість феромонного сліду τ, що змушує мурах вибирати наступний вузол, у складається з двох частин: одна – це традиційний крайовий феромон τ^e, інша – точковий феромон τ^p, визначений для уникнення зіткнення. Колонії мурах не тільки залишають свій феромон на краях, які вони пройшли, але й відкладають феромон на цих вузлах на своєму шляху. Мурахи, які служать для i-го БПЛА, будуть прагнути вибирати ті краї, багатші на власний крайовий феромон τ_i^e і уникати вузлів з більшим точковим феромоном інших БПЛА. Таким чином, загальний феромон, який розглядають мурахи i-го БПЛА від вузла s до u, визначається:

$$\tau_{i,su} = \tau_{i,su}^{e} \cdot \frac{N-1}{\sum_{j \neq i} \tau_{j,u}^{p}}. \quad (4.2)$$

Визначимо ймовірність переходу від вузла s до вузла u для k-ї мурахи як:

$$p_{i,su}(t) = \begin{cases} \frac{[\tau_{i,su}(t)]^{\alpha}[\eta_{su}]^{\beta}}{\sum_{v \in \text{доступні}_k}[\tau_{i,su}(t)]^{\alpha}[\eta_{su}]^{\beta}}, \text{якщо } u \in \text{доступні}_k \\ 0, \text{ у іншому випадку} \end{cases}. \quad (4.3)$$

Де доступний k позначає допустиму область k-ї мурахи. α та β – це параметри, які контролюють відносну важливість сліду та видимість.

Після того як мурахи в алгоритмі побудують свої шляхи, значення сліду феромонів краю кожного ребра (s, u) і точкового феромона кожної точки u оновлюються відповідно до таких рівнянь:

$$\tau_{i,su}^{e}(t+1) = (1-\rho)\cdot\tau_{i,su}^{e}(t) + \Delta\tau_{i,su}^{e}, \quad (4.4)$$

$$\tau_{i,u}^{p}(t+1) = (1-\rho)\cdot\tau_{i,u}^{p}(t) + \Delta\tau_{i,u}^{p}. \quad (4.5)$$

Де ρ⊂ (0, 1) – параметр локального розпаду феромонів, який представляє швидкість випаровування сліду між часом t і t+1.

$$\Delta\tau_{i,su,k}^{e} = \begin{cases} \frac{Q}{J_{i,k}}, & \text{якщо } k-th \text{ мураха використовує } (s,u) \\ 0, & \text{у іншому випадку} \end{cases}, \quad (4.6)$$

$$\Delta\tau_{i,u,k}^{p} = \begin{cases} \frac{Q}{J_{i,k}}, & \text{якщо } k-th \text{ мураха використовує вітку } u \\ 0, & \text{у іншому випадку} \end{cases} \quad (4.7)$$

де Q є константою, а $J_{i,k}$ позначає вартість шляху k-ї мурахи.

4.1.2 Створення адаптивного Max-Min ACO

Щоб підвищити продуктивність мурашиної системи, усунути проблеми, пов’язані з ранньою стагнацією та прискорити швидкість конвергенції, в алгоритм ACO введимо наступні стратегії.

По-перше для m мурах, що обслуговують і-й БПЛА, на кожній ітерації будується загальна кількість m шляхів. Середня вартість цих шляхів становить:

$$J_{i,ave}(t) = \frac{1}{m}\sum_{k=1}^{m} J_{i,k}(t). \quad (4.8)$$

Ця формула діє тоді і тільки тоді, коли вартість шляху k-ї мурахи на t-й ітерації задовольняє $J_{i,k}(t) \le J_{i,min}(t)$, k-та мураха може оновити як свій крайовий, так і точковий феромон за допомогою раніше наведених рівнянь 4.6 та 4.7.

Наступним кроком, незалежно від вибору між найкращою ітерацією та глобальною найкращою мурахою для оновлення сліду феромонів може статися застій пошуку. Такої ситуації слід уникати.

Одним із способів досягнення цього є вплив на ймовірність вибору наступного компонента рішення, що безпосередньо залежить від слідів феромонів та евристичної інформації. Евристична інформація, як правило, залежить від проблеми та є статичною протягом усього застосування алгоритму.

Але обмеживши вплив феромонних слідів, можна легко уникнути відносних відмінностей між феромонними слідами під час використання алгоритму. Для досягнення цієї мети ACO накладає та явно обмежує τ_{max} і τ_{min} на мінімальний і максимальний сліди феромонів так, що для всіх слідів феромонів.

Після оновлення феромона в кінці ітерації до феромона буде застосовано наступну операцію на обох краях і точках:

$$\tau^{new}(t) = \begin{matrix} \tau_{min},\ \tau^{old}(t) < \tau_{min} \\ \tau^{old}(t),\ \tau_{min} \le \tau^{old}(t) \le \tau_{max}. \\ \tau_{max},\ \tau^{old}(t) > \tau_{max} \end{matrix} \quad (4.9)$$

4.1.3 Застосування адаптивного ACO Max-Min для скоординованого перепланування траєкторії кількох БПЛА

Блок-схема на рисунку 4.1 описує детальну процедуру застосування запропонованого адаптивного ACO Max-Min до одного БПЛА в практичному питанні перепланування скоординованої траєкторії кількох БПЛА. Мурахи і-го БПЛА побудували свої шляхи та завершили оновлення як крайових, так і точкових феромонних шляхів. Потім новий оновлений феромон передається до наступної ітерації. Тим часом точковий феромон передається іншим БПЛА. Координація повітряного простору кількох БПЛА, яка в основному спрямована на уникнення зіткнення, залежить лише від точкового феромона.

Тому через зв’язку точкового феромона кожного БПЛА, координація повітряного простору може бути встановлена на рівні планувальника траєкторії.

Узгодженим АСО для кожного БПЛА визначено ряд можливих маршрутів-кандидатів, але залишається вибрати, за якою траєкторією кожен БПЛА буде летіти.

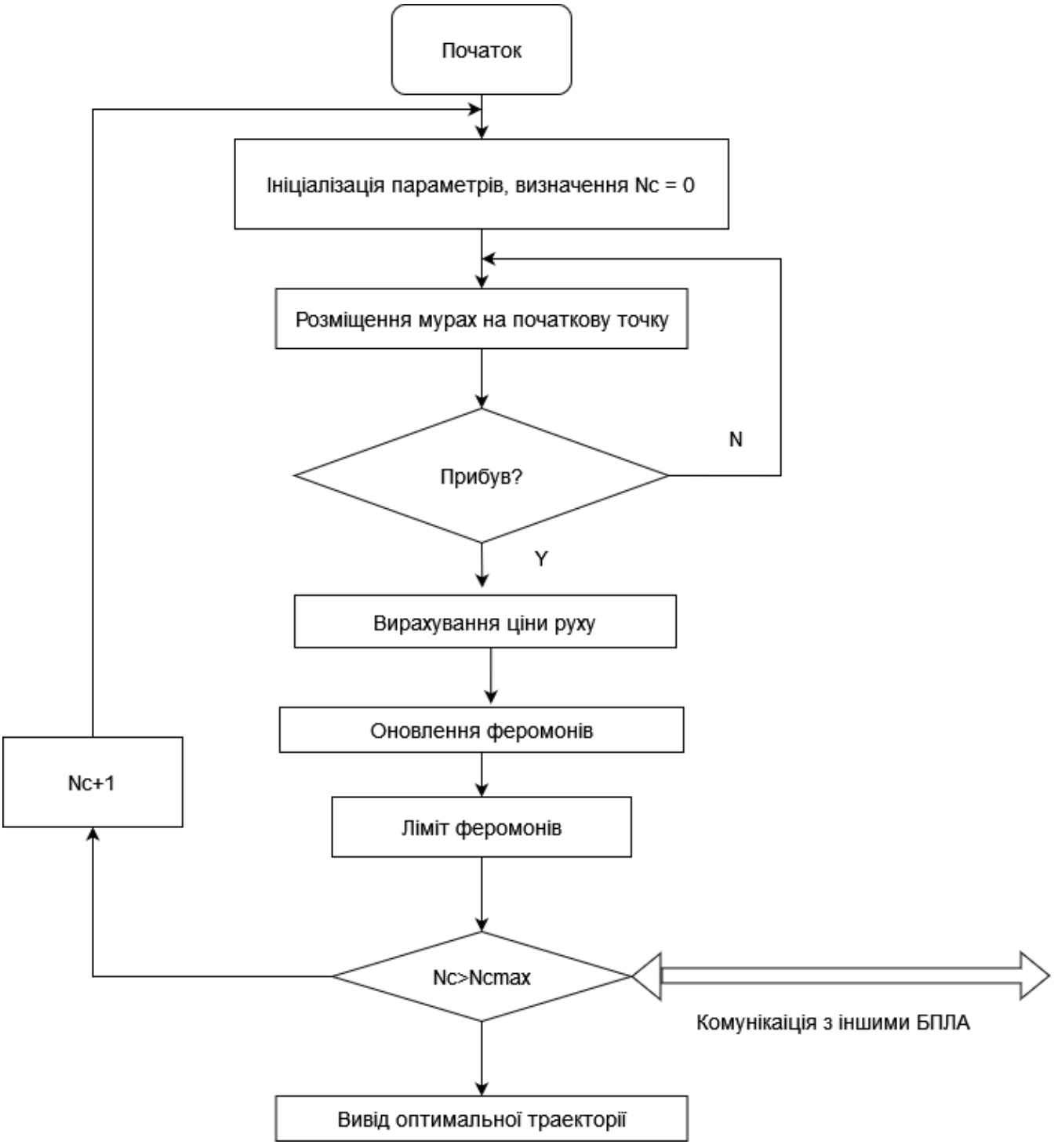


Рисунок 4.1 – Блок-схема адаптивного АСО Max-Min, застосованого до одного БПЛА з в системі кількох БПЛА

Тому з кожної субпопуляції вибирається чотири оптимальні траєкторії-кандидати для кожного БПЛА, а потім визначили оптимальний координаційний ЕТА за допомогою знаходження координаційної змінної T_a, яка мінімізує загальну вартість координації. Після отримання скоординованої командою ЕТА траєкторії, за якими будуть літати багато БПЛА, доступні також і швидкості польоту БПЛА. Повна процедура застосування ACO до проблеми координації кількох БПЛА показана на рисунку 4.2.

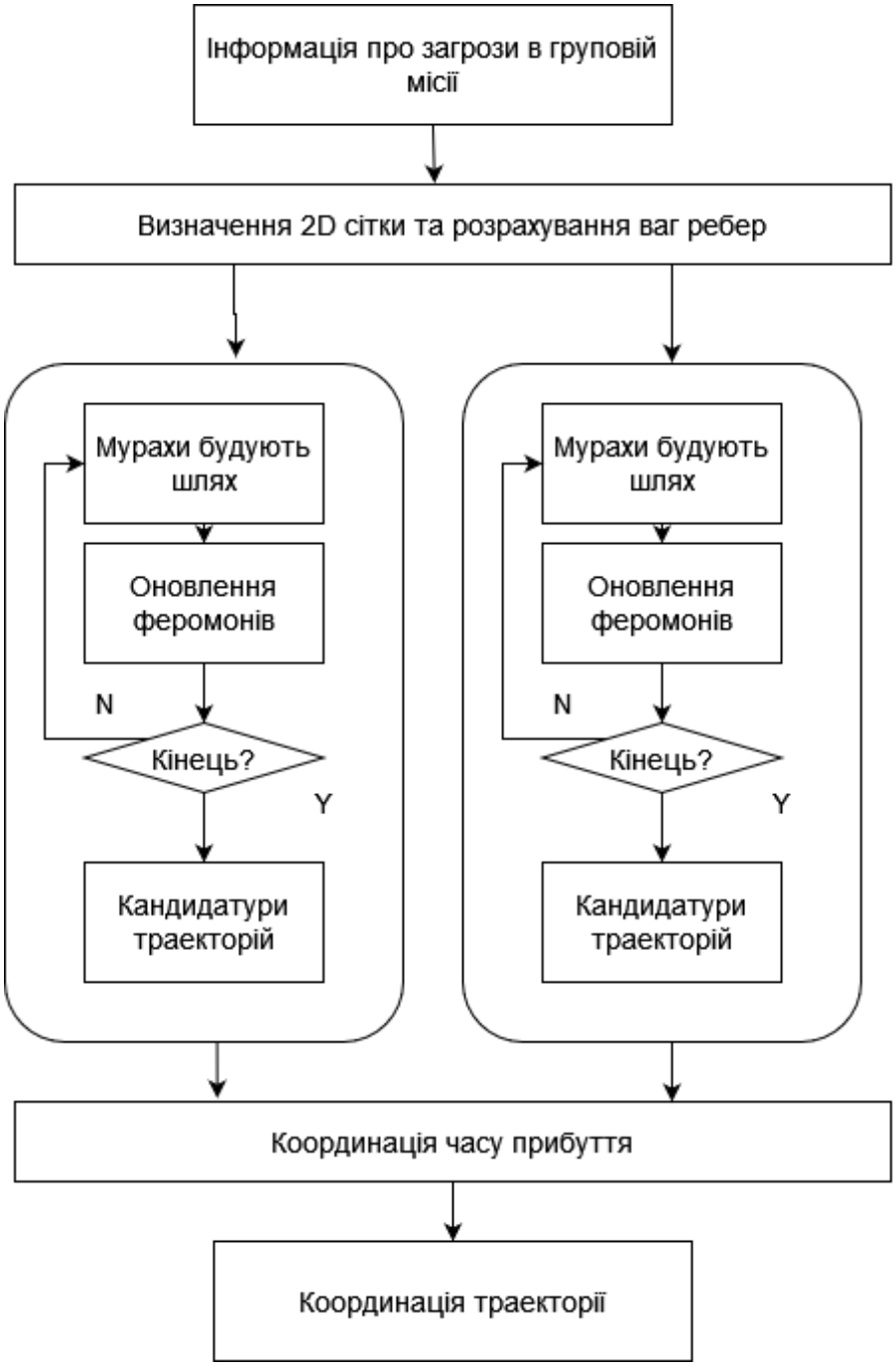


Рисунок 4.2 – Застосування ACO до проблеми координації кількох БПЛА

Дані етапи зображені на продемонстрованих рисунках, можна описати у вигляді текстового алгоритму із детальним поясненням кроків та необхідних дій.

Етапи програмування адаптивного алгоритму ACO Max-Min для вирішення перепланування траєкторії можна описати так:

1. Побудова двовимірної сітки, що охоплює область місії, і розрахуглк вартістості країв.

2. Ініціалізація параметрів алгоритму, включаючи α, β, ρ, Q, τ_{max}, τ_{min}, а також кількість мурах m і кількість ітерацій Nc_{max}.

3. Ініціалізація крайових та точкових феромонів для кожної субпопуляції та поміщення мурах у відповідну початкову точку.

4. Для кожного БПЛА мурахи вибирають вузол відповідно до рівняння 3.7 до досягнення цілі, а потім нарешті будуються деякі можливі траєкторії.

5. Обчислення значення функції вартості J_i,k k-ї мурахи, що належить і-му БПЛА, та оновлення точкових та крайніх феромонів відповідно до рівнянь (4.4) – (4.9).

6. Передання феромону до наступного обчислення ітерації та передання точкового феромону іншим БПЛА; потім повернення до кроку 4, поки він не задовольнить кінцевій умові $Nc > Nc_{max}$.

7. Вибір кілька можливих маршрутів-кандидатів для кожного БПЛА та визначення оптимальної скоординованої приблизно ETA для команди.

Коли спливаючі загрози виявлені, а вихідна траєкторія знаходиться в небезпеці, екстрені дії багаторазових БПЛА будуть вжиті відповідно до таких кроків:

1. Визначення інформації про спливаючі загрози, включаючи розташування, діапазон загрози та рівень загрози.

2. Обчислення вартісті загроз для країв, для яких створюються спливаючі загрози, а потім оновлення значення функції вартості кожного ребра в двохвимірній сітці.

3. Спрямовування кожного БПЛА на сусідній і досить безпечний вузол, який буде прийнятий за нову відправну точку перепланування траєкторії.

4. Коли багато БПЛА рухаються як крок 3, щойно перепланована траєкторія розраховується відповідно до процедури, показаної на рисунку 4.1 та рисунку 4.2.

4.2 Програмно-технічна реалізація мурашиного алгоритму для контролю визначеного БПЛА

4.2.1 Обгрунтування вибору технічних та програмних засобів розробки

Для створення програмно технічного засобу було вибрано мову програмування Python версії 3.10. Дана мова програмування підходить як для високорівневого програмування, так і оперування низькорівневим, для обробки даних, та передавання тієї чи іншої технічної інформації.

Основні математичні функції та додаткові функції для роботи із числами та графами містяться в імпортованих модулях. Наприклад, для роботи із математичними функціями був імпортований модуль:

```
import math
```

Для опрацювання додаткових математичних функцій та функцій роботи із числами, в тому числі основних алгоритмів сортування, та обробки числових масивів, використовується модуль numpy:

```
import numpy as np
```

Враховуючи специфіку проекту та обробку статистичних даних, які отримуються в процесі розробки програми, та в її результатах, для виводу тестів та результатів використовується модуль побудови графів mathplotlib:

```
import matplotlib.pyplot as plt
```

Результатом тестів, та плаваючого аналізу даних, враховуючи тестові набори даних маршрутів будуть різні графіки, виконані у тому чи іншому вигляді, на різних ресурсах, використовуючи результати роботи програми, зокрема масиви, даних, що основані на графах та ребрах, оптимальних маршрутів БПЛА.

Для програмування технічного засобу, а саме, БПЛА та контроллер для керування ним, було обрано технічний фреймворк, із можливістю налаштування автопілота БПЛА PX4.

4.2.2 Опис основних параметрів, змінних та функцій алгоритму

Враховуючи специфікацію алгоритму, та із раніше наведених формул, описів та блок схем можна визначити основні параметри, які будуть впливати на роботу алгоритму, та визначати основну роботу БПЛА.

Знаючи специфіку мови Python, та об’єктно орієнтованої структури проекту, першочергово визначається конструктор основного класу алгоритму, у ньому і визначмо основні змінні, також врахуємо оптимізований варіант алгоритму для конкретної задачі із БПЛА.

Першочерговим параметром роботи є кількість робочих БПЛА у групі, тобто кількість мурах для проходження графіка:

self.ants = ants

При постановці задачі для, БПЛА із розрахунку оптимального алгоритму, потрібно встановити точку відправлення, місце на карті, де починається рух мурах:

self.init_location = init_location

Аналогічним чином, вибираються орієнтовні кінцеві точки маршруту, список можливих вузлів, до яких мураха може перейти при внутрішньому використанні, список можливих місць, до яких мураха може пройти, мінус ті вузли, які вже відвідали:

self.possible_locations = possible_locations

Визначений маршрут, який був чи буде розрахований за допомогою алгоритму, певний список доріг, який оновлюється мітками вузлів, які пройшов мураха:

self.route = []

Знаючи початкову та орієнтовну кінцеву точку маршруту, потрібно буде розраховувати довжину кожного маршруту, загальну відстань, пройдену по кроках на маршруті:

self.distance_traveled = 0.0

Встановивши початковий стан БПЛА та його маршруту, потрібно буде розрахувати карту феромонів, для кожного обходу між кожним вузлом:

self.pheromone_map = pheromone_map

У оптимізованому ж варіанті алгоритму краще використати більш детальний варіант опису маршрутів мурах, а саме використати матрицю феромонів:

self.pheromone_matrix = None

Розраховану евристичну матрицю:

self.heuristic_matrix = None

Та матрицю розраховних ваг маршрутів:

self.probability_matrix = None

Для внутрішнії розрахунків, у циклах програми, при проходженні маршрутів, потрібно використати і інші відомості про стан мурах та їх участь у роботі феромонів, а саме distance_callback – що є функцією для обчислення відстані між двома вузлами, alpha – параметр з алгоритму ACO для контролю впливу кількості феромонів під час вибору шляху, beta – параметр від ACO, який контролює вплив відстані до наступного вузла в виборі шляху.

Також необхідно визначити параметр який буде запам’ятовувати перший можливий прохід БПЛА по маршруту, та розпилювати феромони, необхідно це для наступних порівнянь.

Встановивши першочергові необхідні змінні для обробки даних можна описати, основні методи, які впливають на розрахунку траєкторії та вибір оптимального шляху.

Основним методом є вибір оптимального шляху для БПЛА, визначення даної функції має наступний вигляд:

def _pick_path(self)

Даний метод має декілька під визначених функцій, зокрема перевірку першого проходження, визначення наступного маршруту на даних із існуючих, якщо маршрут не перший.

Перший тестовий прохід визначається випадковим чином із масиву доступних локацій:

return random.choice(self.possible_locations)

Даний метод реалізує алгоритм вибору шляху ACO, обчислює привабливість кожного можливого переходу з поточного розташування, а потім випадковим чином вибирає наступний шлях на основі його привабливості.

Оновлення феромонів мурах відбувається у постійному циклі, поки не вибереться найкращий маршрут, важливо уточнити, що даний метод містить багато функцій, частина із яких технічні, такі як оновлення матриць, побудова нових маршрутів, на основі старих, видалення маршрутів і тому подібне, усі дані функції будуть наведені у коді програми із відповідними описами.

Потрібно ще детальніше розібрати основні функції роботи програмно-технічного засобу. Важливо звернути увагу на наступні важливі функції:

1. Оновлення карти феромонів.
2. Визначення дистанції.
3. Прив’язка мурах до вибраного шляху.

Оновлення карти феромонів – self.pheromone_map, відбувається шляхом зменшення значень, що містяться в ньому, за допомогою алгоритму ACO. Додаються значення феромонів від усіх мурах із ant_updated_pheromone_map.

Визначення дистанції використовує distance_callback для повернення відстані між вузлами, якщо відстань не була обчислена раніше, потім вона заповнюється в distance_matrix і повертається, якщо відстань була викликана раніше, в кінці кінців її значення повертається з distance_matrix.

Прив’язка мурах відбувається перед оновленням карти феромонів, але після визначення дистанції, зокрема, поточне значення феромонів ділиться на вибрану дистанцію, і з цього формується коофіцієнт феромону. Коефіцієнт феромону new_pheromone_value, використовується пізніше для оновлення карти феромонів.

Після цього запускається основний цикл програми, із застосуванням наведених минулих функцій, та автоматизованих методів розрахунку та оновлення поточних та майбутніх матриць даних.

Основний цикл у вигляді блок схеми та встановленої послідовності дій був наведений у пункті 4.1, але можна сформувати більш детальну послідовність дій, яка може бути застосована для формування функції:

1. Запуск багато потокових мурах, виклик руху мурах у новому потоці.
2. Оперативна пауза, поки мурахи закінчать рух, перш ніж переходити до зміни спільних ресурсів.
3. Оновлення карти феромонів із вмістом феромонів мурахи на своєму маршруті.
4. Якщо ще не було жодних шляхів, заповнення їх для порівняння пізніше.
5. Якщо є коротший шлях, то збереження його для повернення.
6. Зменшення поточних значення феромонів і додання всіх значень феромонів, були помічені під час обходу.
7. Помітка, про завершення першого проходження мурах.
8. Скидання стану усіх мурах, до початку наступної ітерації.
9. Скидання оновляємої карти феромонів, для наступного проходження.
10. Переведення найкоротшого шляху до ідентифікаторів вузла.

4.2.3 Визначення та налаштування БПЛА

Визначення класу літального апарату було зазначено у другому розділі, типом пристрою, що досліджується є квадротор, БПЛА із чотирма моторами, та всіма необхідними датчиками для дослідження навколишнього середовища на загрози, та засоби комунікації між іншими БПЛА.

Для експериментального зображення БПЛА на анімованому графі, необхідно визначити віртуальний об’єкта даного БПЛА, та сформувати клас із необхідними методами.

Основними параметрами класу квадротору є його координати в просторі, а саме відносно осей x, y, z.

Основними методами є побудова матриці транспортування, де буде зображено рух БПЛА:

def transformation_matrix(self)

Та метод оновлення позиції БПЛА на матриці:

def update_pose(self, x, y, z, roll, pitch, yaw)

4.2.4 Генерація маршруту

Генерацію маршруту можна поділити на 2 етапи, а саме: генератор траєкторії, який використовується для побудови оптимальних маршрутів згідно алгоритму, та побудова траєкторія БПЛА у трьох вимірній моделі представлення руху БПЛА.

Перший варіант, побудови маршруту являє собою сукупність математичних функцій із розрахунком координат БПЛА в залежності від стартової позиції та кінцевого місця призначення, назва методу конструктора, із параметрами має вигляд:

def __init__(self, start_pos, des_pos, T, start_vel=[0,0,0], des_vel=[0,0,0], start_acc=[0,0,0], des_acc=[0,0,0])

Другий етап є проектування трьох вимірного простора для експериментального представлення руху БПЛА. Для цього початкові дані симуляції задаються сталими параметрами. А функції описуються з точки зору симуляційного процесу.

Першочерговим є розрахунок необхідної тяги та круговий момент для квадродвигуна слідувати за траєкторією, описаною наборами коефіцієнтів, які були задані початковими даними, або розрахованими внаслідок заданого алгоритму.

Після цього розраховується позиція БПЛА з урахуванням набору коофіцієнтів квінтичного поліному та часу необхідного для обчислення позиції, заголовок відповідної функції має вигляд:

def calculate_position(c, t):

Де відповідно c – це квантум, а t – час для обчислення. Сама функція повертає позицію.

Інші ж функції відповідають за побудову матриці простору, та рух і взаємодію БПЛА із цією матрицею.

Разом із використанням цих функцій та графічного модуля можна відобразити рух БПЛА у трьох вимірному просторі, згідно мурашиного алгоритму, з наперед заданими даними (рисунок 4.3).

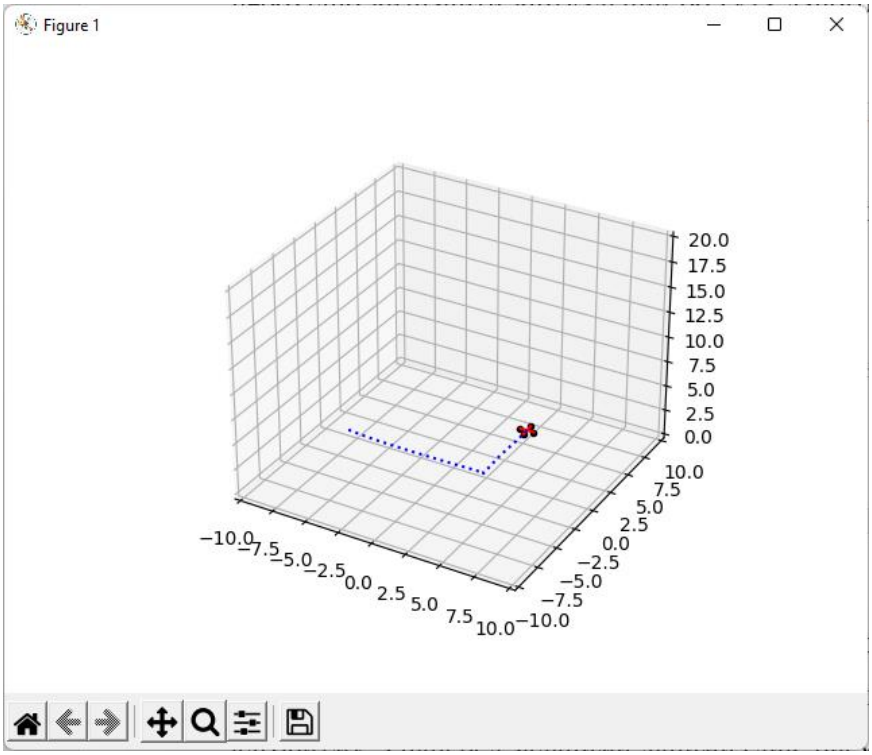


Рисунок 4.3 – Графічна симуляція руху БПЛА у трьох вимірному просторі

4.3 Використання MAVSDK та PX4

Використання MAVSDK дозволяє змінювати прошивку дронів, та їх контролерів, а PX4 має потужні інструменти для керування та налаштування автопілоту БПЛА, із користувацькою програмою.

Також за необхідності ресурс місить візуальний інтерпретатор польоту, але ця можливість не використовувалась наразі у роботі.

Для інсталяції даного SDK було виконано у менеджері встановлень Python наступний рядок коду:

pip3 install mavsdk

Для використання даного модулю необхідно лише імпортувати його у проект:

from mavsdk import System

Після цього будуть доступні команди для керування симуляційним дроном, або підключеним дроном. В результаті можна передавати дані координат, та траєкторій, інший дронів, які були розразовані внаслідок виконаного алгоритму.

PX4 має схожий функціонал, але працює на мові програмування C++, проте також має таблиці даних координат, та може працювати із сторонніми модулями.

Для встановлення модифікованої прошивки використовувалась програма QGroundControl зображена на рисунку 4.4.

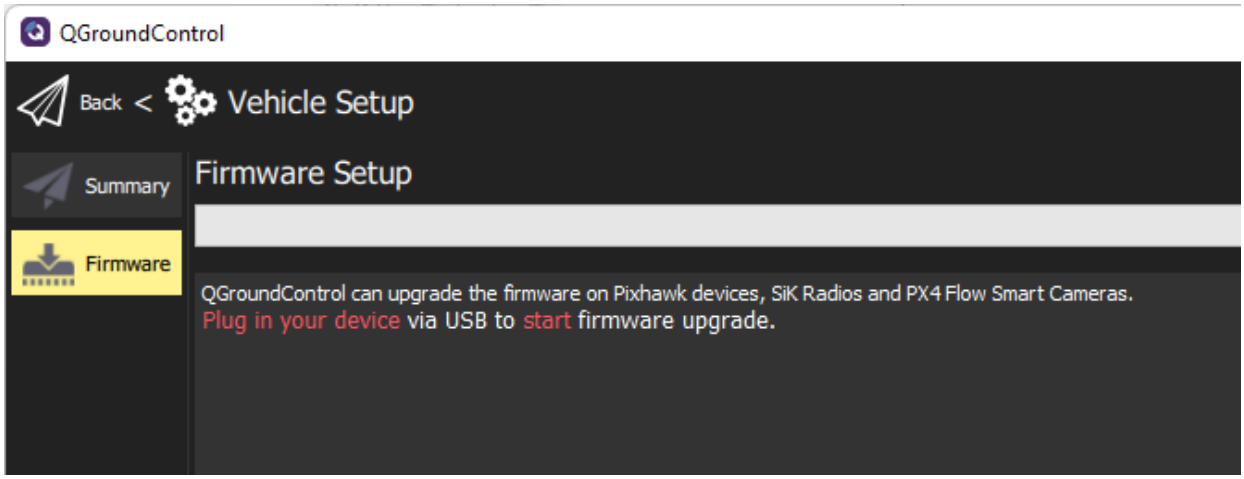


Рисунок 4.4 – Інтерфейс програми встановлення прошивки QGroundControl

4.4 Імітаційні експерименти

Використовуючи інформацію із минулих розділів, де приводились приклади стимуляційного польоту БПЛА (рисунок 4.3) та описані формули розрахунків основних координат, та поведінкових дій БПЛА. Можна приступити до математичних симуляцій польоту БПЛА.

Самі математичні симуляції були виконані у середовищі MATLAB та за допомогою модуля Python – matplotlib.pyplot.

Для того, щоб дослідити доцільність та ефективність запропонованого адаптивного підходу Max-Min ACO до скоординованого перепланування траєкторії кількома БПЛА, було проведено серію імітаційних експериментів.

У цих імітаційних експериментах область місії має довжину 60 км і ширину 70 км з п’ятьма відомими ворожими загрозами. Інформація про ці небезпечні загрози наведена в таблиці 4.1.

Таблиця 4.1 – Інформація про відомі загрози

№	Локація (км)	Радіус загрози (км)	Оцінка загрози
1	52,32	10	2
2	36,26	6	1.2
3	22,48	8	1.6
4	26,56	12	1.4
5	30,30	9	2

У першому експерименті два БПЛА призначені для досягнення однієї мети із сусідніх вузлів. У цьому сценарії розглядалася лише координація повітряного простору, щоб перевірити ефективність уникнення зіткнення адаптивної моделі ACO Max-Min. Траєкторії, оптимізовані в цих експериментах, представлені на рисунку 4.5, і видно, що між двома сусідніми траєкторіями немає перекриття, таким чином досягається уникнення зіткнення.

На рисунку 4.6 показано еволюцію вартості двох траєкторій БПЛА, які сходяться після кількох ітерацій. Результати моделювання демонструють, що ACO, розглядаючи точковий феромон як коефіцієнт координації повітряного простору, можливий для створення траєкторій, які задовольняють уникнення зіткнення.

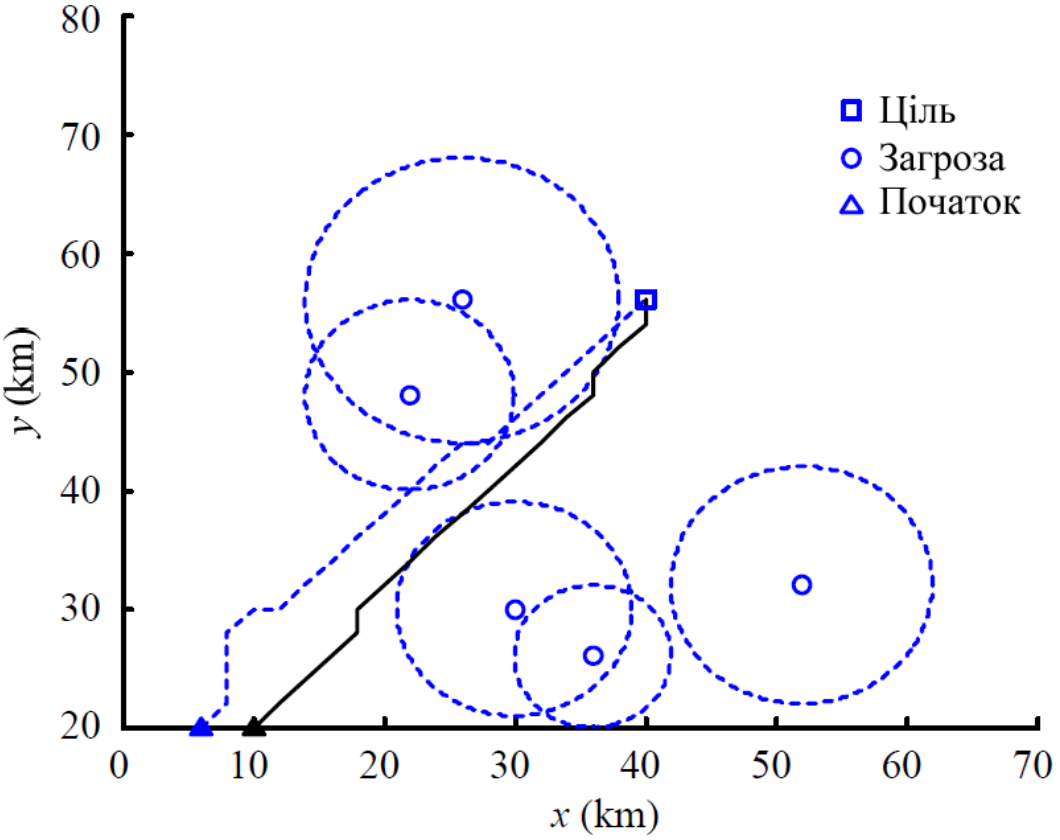


Рисунок 4.5 – Траєкторія двох БПЛА

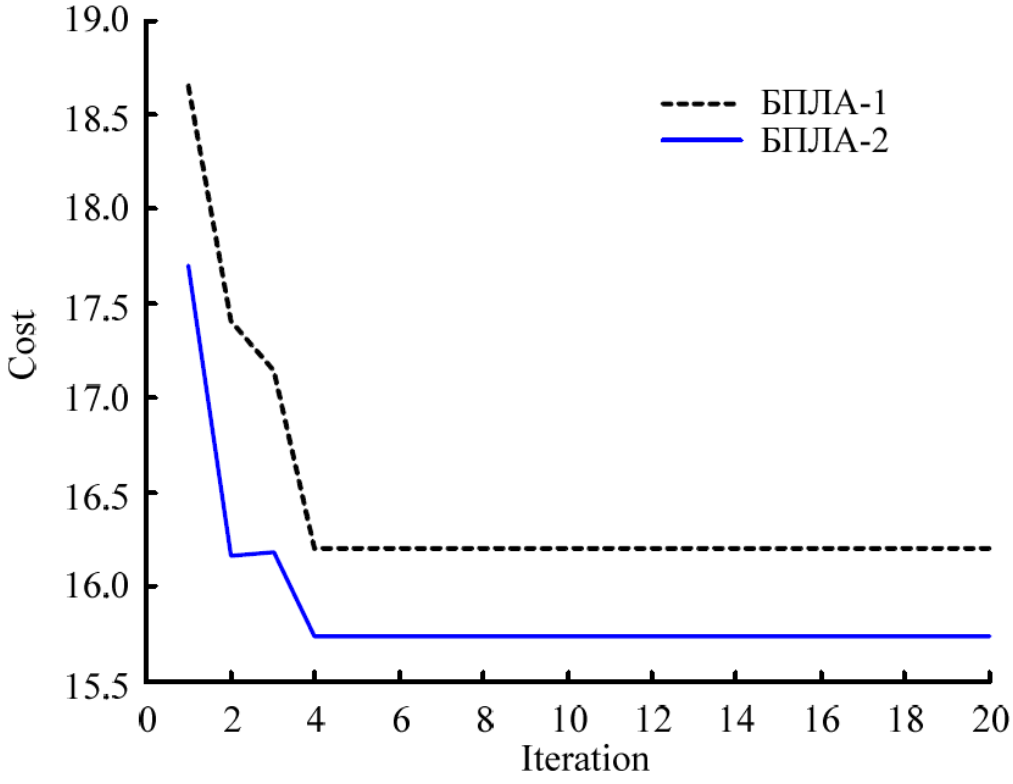


Рисунок 4.6 – Криві еволюції витрат на траєкторії двох БПЛА з використанням адаптивного ACO Max-Min

У другому експерименті повітряний бойовий порядок складається з трьох БПЛА і розташований на різних майданчиках, призначений для одночасної атаки різних цілей. У таблиці 4.2 показано початкові точки місії та цілі атаки.

Таблиця 4.2 – Початкові точки місії та атакуючі цілі

	БПЛА-1	БПЛА-2	БПЛА-3
Початкова позиція (км)	6,20	24,20	40,20
Ціль (км)	10,68	40,60	50,68

Швидкість польоту знаходиться між $V_{max} = 300\ m \cdot s^{-1}$ і $V_{min} = 200\ m \cdot s^{-1}$. Початкові параметри: $\alpha =$, $\beta = 2$, $\rho = 0,7$, $Q = 10$, $Nc_{max} = 20$, $m = 20$, $\tau_{max} = 10$ і $\tau_{min} = 0,1$.

Після повторного розрахунку кожен БПЛА отримує чотири кандидатські траєкторії. Визначений час прибуття ETA показано на рисунку 4.7, на якому T_a = 173 секунд. Відповідно до цього ETA, кожен БПЛА може вибрати свою оптимізовану траєкторію та належну швидкість польоту. Вибрана довжина траєкторії та швидкість польоту наведені в таблиці 4.3.

Таблиця 4.3 – час прибуття, довжина траєкторії та швидкість польоту для багаторазових БПЛА

	БПЛА-1	БПЛА-2	БПЛА-3
Розрахунковий час (с)		174	
Довжина траєкторії (км)	51.31	47.80	52.14
Швидкість польоту ($км \cdot с^{-1}$)	295	275	300

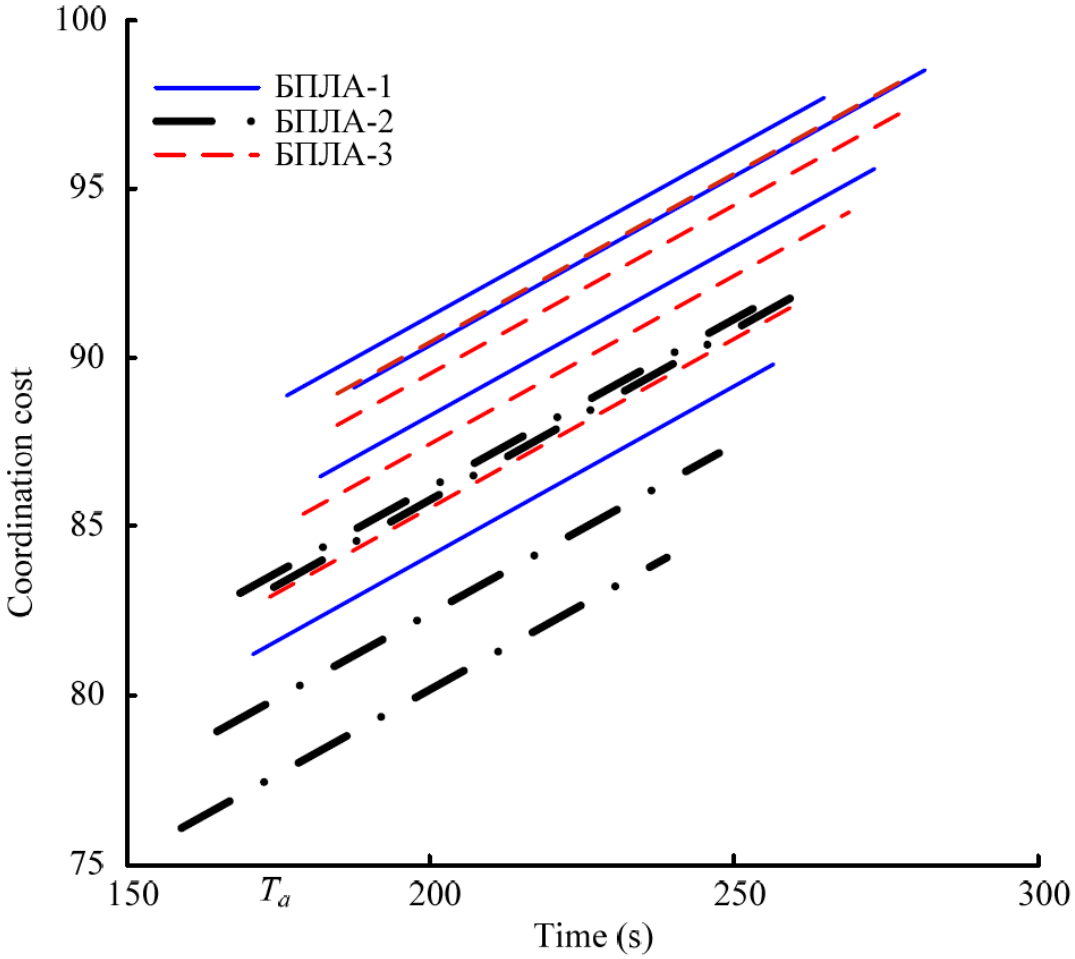


Рисунок 4.6 – Розв’язок ETA

На рисунку 4.7, показано дві спливаючі загрози, які виявляються кількома БПЛА. Інформація про ці спливаючі загрози зображена у таблиці 4.4.

Потім кожен БПЛА перенаправляється на сусідній захищений вузол, який приймається за нову відправну точку. Тим часом програма БПЛА, перепланує і генерує нові траєкторії для БПЛА.

Таблиця 4.4 – Інформація про спливаючі загрози

№	Локація (км)	Радіус загрози (км)	Оцінка загрози
1	12,40	6	2.5
2	40,40	6	2

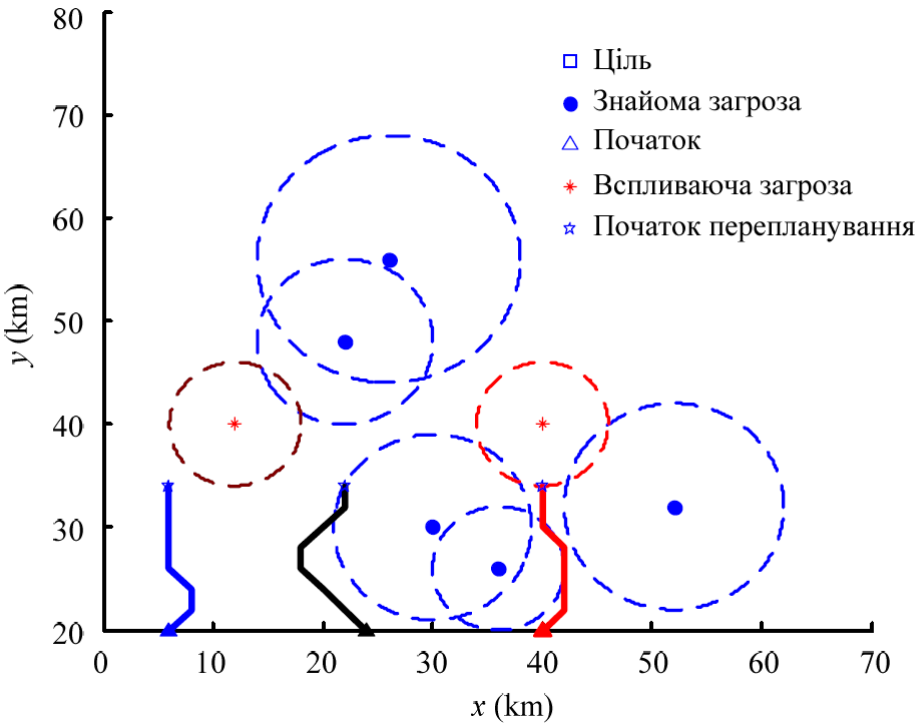


Рисунок 4.7 – Спливаючі загрози при руху БПЛА

Змінена траєкторія руху БПЛА, згідно мурашиного алгоритму зображена на рисунку 4.8.

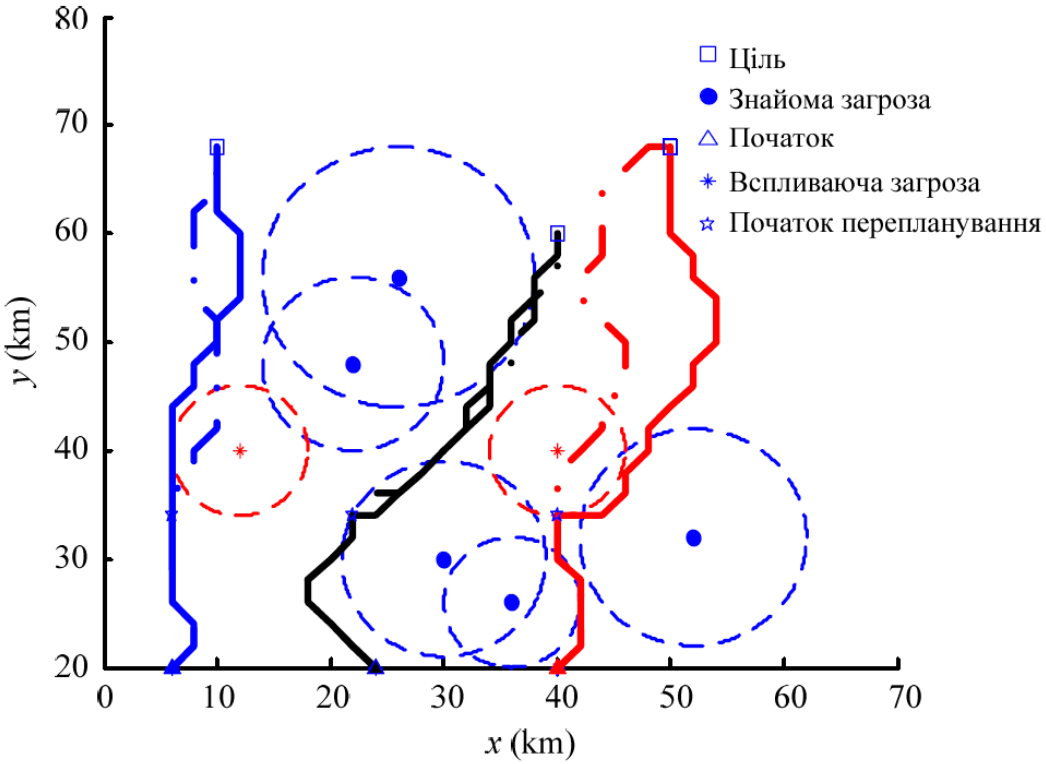


Рисунок 4.8 – Змінена траєкторія руху БПЛА

Експериментальні результати показують, що адаптивний алгоритм ACO Max-Min може ефективно вирішувати проблеми перепланування координованої траєкторії кількома БПЛА, а час прибуття також досить короткий.

4.5 Висновки

У даному розділі представлено реалізацію програмно-технічного засобу методу керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму.

Реалізація програмно-технічного засобу базується на використанні мови програмування Python із використанням допоміжних бібліотек numpy та math. Для трьох вимірного представлення моделі БПЛА, та його руху у просторі було використано модуль matplotlib.pyplot.

Технічна складова проекту та безпосередньо керування елементами БПЛА, було досягнуто за допомогою використання MAVSDK та фреймворку PX4, що дозволяє ефективно використовувати функції керування БПЛА та встановлення стороннього програмного засобу (в тому числі модифікованого) на апаратну частину пристрою.

В розділі описано основну логіку роботи алгоритму, яка адаптована, для програмної реалізації на системному рівні.

Особливої уваги приділяються основні параметри та функції, якими оперує програмно-технічний засіб та дозволяє реалізувати поставлений функціонал алгоритму на практичному рівні.

В результаті роботи було сформовано трьох вимірну модель симуляцію руху БПЛА у просторі згідно алгоритму, та проведено ряд математичних тестів, для затвердження ефективності ACO Max-Min алгоритму.

ВИСНОВКИ

У даній роботі за результатами практичних та теоретичних досліджень було створено та описано метод і систему керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму.

У першому розділі було досліджено предметну область, описані доступні типи БПЛА, їх схеми контролю та сфери застосування. В тому числі було визначено основні необхідні технічні характеристики БПЛА, для їх оптимальної координаційної роботи при спільних місіях. Також було проаналізовано доступні методи рішення поставленої задачі, коротко описані основні недоліки наявних алгоритмів, та технологій. В результаті аналізу було зроблено висновок, що наявні алгоритми та технології мають багато недоліків у вигляді низької швидкодії, поганої координації, чи проблем у виявленні супутніх загроз. В розділі представлено подальші кроки для вирішення вказаних проблем та сформовано постановку задачі.

У другому розділі детально описується вибір типу БПЛА, для проведення досліджень, наведені його технічні характеристики, та специфікації роботи. В тому числі була сформована математична модель скоординованого планування оптимальної траєкторії руху БПЛА, яка включає в себе модель та принципи роботи вибраного БПЛА, включно із основними його механізмами, та модель роботи оптимізованого асинхронного алгоритму для корегування БПЛА при виконанні місій.

У третьому розділі було представлено метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму. Було описано метод скоординованого планування оптимального маршруту БПЛА для завершення місій, описано можливі загрози та способи їх обійти. Метод зображає обмеження умов при координації БПЛА при перепланування їх маршруту та описує схему їх координації за феромонами мурах для вирішення цих проблем. Результатом даних експериментальних досліджень, було отримано покращену швидкодію координації БПЛА на маршруті, вдале реагування на

прогнозовані та спливаючі перешкоди із оптимальним варіантом посадки на злітну полосу чи пересічну місцевість.

У четвертому розділі представлено реалізацію програмно-технічного засобу керування групи безпілотних апаратів методом нечіткого мурашиного алгоритму.

Реалізація містить у собі модифікований мурашиний алгоритм із паралельно асинхронними колоніями мурах, в якому продемонстрована принципова модель адаптивного Max-Min ACO, для швидкого та безпечного виконання місій групами БПЛА. У розділі представлено блок-схеми Max-Min ACO, де представлено основний алгоритм бій БПЛА при тих чи інших умовах, включаючи комунікацію БПЛА між собою, та реагування на групи феромонів на маршруті.

В розділі описано обумовлення вибору технологій розробки, зокрема Python 3.10, додаткових програмних бібліотек matplotlib.pyplot та numpy, які використовуються для математичних обчислень та побудови трьох вимірного простору для тестування змодельованого БПЛА. Описано основні функції та їх призначення, які були створенні в процесі виконання роботи, для реалізації поставлених задач. Також було описано використання стороннього SDK та платформи PX4 для суміжної імплементації програми у технічному та системному варіанті на практичній основі.

В результаті виконання було змодельований тестовий трьох вимірний простір для симуляції руху БПЛА, за допомогою утиліти QGroundControl, була створена можливість імплементації технічної прошивки для обраного БПЛА на основі мікрокомп’ютера Odroid-XU4 та проведено ряд математичих тестів у MATLAB для підтвердження ефективності алгоритму.

Наукова новизна отриманих результатів:

− набув подальшого розвитку метод суть роботи якого полягає в тому, щоб представити ієрархічну модель прийняття рішень для спільного планування місій безпілотників для місії розвідки шляхом інтеграції розподілу місії та планування маршруту;

− набула подальшого розвитку інформаційна технологія керування групою безпілотних літальних апаратів, модель реалізації якої буде відбуватись без критично-довгих навчань та з відносно очікуваним результатом.

Практична цінність проекту полягає в розробленому програмно-технічному продукті, який дозволяє організувати оптимальну систему керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму та виконання місій цими БПЛА у найкоротший термін та з високою ефективністю.

Під час виконання досліджень було опубліковано статтю на тему «Метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму» в збірнику наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук АПКН-2021». Хмельницький – 2021. С.117-119.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Ковальчук П.С., Лисенко С.М.. Метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму. *Збірник наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021»*. Хмельницький – 2021. – 117-119с..

2. Drone technology. URL: https://builtin.com/drones (дата звернення: 25.02.2022).

3. Chai Xue, Wang Ganglin and Wu Zhe, The decision making algorithm based on inverse-design method and its application in the UAV autonomous flight control system design. *2010 2nd International Conference on Advanced Computer Control*. 2010. pp. 169-173, doi: 10.1109/ICACC.2010.5487041.

4. Q. Sun and H. Xu, Adaptive Control Method of UAV Intelligent Rudder Based on Hybrid Genetic Algorithm. *2021 33rd Chinese Control and Decision Conference (CCDC)*. 2021. pp. 4753-4757. doi: 10.1109/CCDC52312.2021.9601687

5. N. Muchiri, S. Kimathi, A review of applications and potential applications of uav. *Proceedings of Sustainable Research and Innovation Conference*. pp. 280–283. 2016.

6. Minh T. Nguyen, Cuong V. Nguyen, Linh H. Truong, Anh M. Le, Toan V. Quyen, Antonino Masaracchia, and Keith A. Teague. Electromagnetic field based wpt technologies for uavs. *A comprehensive survey. Electronics*. pp.461. 2020.

7. M. Hassanalian, A. Abdelkefi. Classifications,applications, and design challenges of drones. *A review. Progress in Aerospace Sciences*, 91:99–131, 2017.

8. C. Kanellakis, G. Nikolakopoulos. Survey on computer vision for uavs. Current developments and trends. *Journal of Intelligent & Robotic Systems*. 87(1):141–168, 2017.

9. G. Cai, Kai-Yew Lum, BenMChen, Tong H Lee. A brief overview on miniature fixed-wing unmanned aerial vehicles. *IEEE ICCA 2010*. pp. 285–290. IEEE, 2010.

10. A. Noordin, MA Mohd Basri, M. Zaharuddin, AF Z. Abidin. Modelling and psofine-tuned pid control of quadrotor uav. *Int. J. Adv. Sci.Eng. Inf. Technol*, 7(4):1367–1373, 2017.

11. G. Szafranski, R. Czyba. Differentapproaches of pid control uav type quadrotor. *Differentapproaches of pid control uav type quadrotor*. pp. 12–15. 2011.

12. O. Mofid, S. Mobayen. Adaptive slidingmode control for finite-time stability of quad-rotor uavswith parametric uncertainties. *ISA transactions*, 72:1–14. 2018.

13. Y. Chen, S. Liu, C. Xiong, Y. Zhu and J. Wang. Research on UAV Flight Tracking Control Based on Genetic Algorithm optimization and Improved bp Neural Network pid Control. *2019 Chinese Automation Congress (CAC)*. 2019. pp. 726-731. doi: 10.1109/CAC48633.2019.8996179.

14. R. Andrade, G. V Raffo, J. E Normey-Rico. Model predictive control of a tilt-rotor uav forload transportation. *In 2016 European Control Conference(ECC)*. pp. 2165–2170. IEEE, 2016.

15. C. Paliotta, E. Lefeber, K. Ytterstad Pettersen, J. Pinto, M. Costa, et al. Trajectory tracking andpath following for underactuated marine vehicles. *IEEE Transactions on Control Systems Technology*. 27(4):1423– 1437. 2018.

16. S. Keyworth, S. Wolfe, UAVS for land use applications: UAVs in the civilian airspace institution of engineering and technology. *IET Seminar on UAVs in the Civilian Airspace*, 2013. pp. 1-13, doi: 10.1049/ic.2013.0071.

17. B. Yan, C. Wu. Research on Taxi Modeling and Taking-Off Control for UAV. *2014 Seventh International Symposium on Computational Intelligence and Design, 2014*. pp. 108-111. doi: 10.1109/ISCID.2014.34.

18. W. Rui, Z. Zhou and S. Yanhang. Robust Landing Control and Simulation for Flying Wing UAV. *2007 Chinese Control Conference.* 2007. pp. 600-604. doi: 10.1109/CHICC.2006.4346934.

19. I. Karakostas, I. Mademlis, N. Nikolaidis, I. Pitas. Shot Type Feasibility in Autonomous UAV Cinematography. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019*. pp. 1937-1941. doi: 10.1109/ICASSP.2019.8683014.

20. S. -I. Nishida, K. Nishigaki, T. Homma, M. Miura and K. Sakurama. Study of a new type of UAV with vertical fins. *2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. 2017. pp. 809-814. doi: 10.1109/AIM.2017.8014117.

21. H. -J. Lee, H. -c. Liu, G. Jeong and J. Lee. Newly proposed hybrid type multi-DOF operation motor for multi-copter UAV systems. *2016 IEEE Conference on Electromagnetic Field Computation (CEFC)*. 2016. pp. 1-1. doi: 10.1109/CEFC.2016.7815933.

22. T. Jiang, C. Du, S. Guo and T. Yin. Microgrid fault diagnosis model based on Weighted Fuzzy Neural Petri Net. *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. 2020. pp. 2361-2365. doi: 10.1109/ITNEC48623.2020.9084926.

23. ODROID-XU4. URL: https://www.hardkernel.com/shop/odroid-xu4-special-price/ (дата звернення: 14.03.2022).

24. F. Chen, R. Jiang, K. Zhang, B. Jiang, and G. Tao. Robust backstepping slidingmode control and observer-based fault estimation for a quadrotor uav. *IEEE Transactions on Industrial Electronics*. 63(8):5044–5056. 2016.

25. M. Kamel, M. Burri, and R. Siegwart. Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles. *IFAC-PapersOnLine*. 50(1):3463–3469. 2017.

26. Ma and S. M. Jiao. Research on the attitude control of quad-rotor UAV based on active disturbance rejection control. *2017 3rd IEEE International Conference on Control Science and Systems Engineering (ICCSSE)*. 2017. pp. 45-49. doi: 10.1109/CCSSE.2017.8087892.

27. X. Hu and J. Liu. Research on UAV Balance Control Based on Expert-fuzzy Adaptive PID. *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*. 2020. pp. 787-789. doi: 10.1109/AEECA49918.2020.9213511.

28. A. S. Holtsov, R. M. Farhadi, V. I. Kortunov and A. Mohammadi. Comparison of the UAV adaptive control with the robust control based on mu-

synthesis. *2016 4th International Conference on Methods and Systems of Navigation and Motion Control (MSNMC)*. 2016. pp. 18-21, doi:10.1109/MSNMC.2016.7783096.

29. ZHAO L, WANG Q W. Design of an attitude and heading reference system based on distributed filtering for small UAV. *Mathematical Problems in Engineering*. 2013.

30. M. M. Alobaedy, A. A. Khalaf and I. D. Muraina. Analysis of the number of ants in ant colony system algorithm. *2017 5th International Conference on Information and Communication Technology (ICoIC7)*. 2017. pp. 1-5, doi: 10.1109/ICoICT.2017.8074653.

31. W. Zheng et al.. Database Query Optimization Based on Parallel Ant Colony Algorithm. *2018 IEEE 3rd International Conference on Image*. 2018, pp. 653-656, doi: 10.1109/ICIVC.2018.8492789.

32. Z. Yang, Y. Yu, K. Zhang, H. Kuang and W. Wang. An improved ant colony algorithm for MapReduce-based fleet assignment problem. *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. 2017. pp. 104-108. doi: 10.1109/IAEAC.2017.8053986.

33. B. Zhao. Convergence Analysis for Ant Colony Algorithm. *2015 8th International Symposium on Computational Intelligence and Design (ISCID)*. 2015. pp. 362-365. doi: 10.1109/ISCID.2015.99.

34. J. Tang, Y. Zhao and F. Li. On the use of ant colony algorithm with weighted penalty strategy to optimize path searching. *2020 International Conference on Intelligent Computing, Automation and Systems (ICICAS)*. 2020. pp. 309-312. doi: 10.1109/ICICAS51530.2020.00070.

35. S. L. Gonzaga de Oliveira and L. Martins Silva. Experiments with pseudoperipheral vertex finders for heuristics for bandwidth reduction evolved by an ant colony hyperheuristic approach. *2020 39th International Conference of the Chilean Computer Science Society (SCCC)*. 2020. pp. 1-5. doi: 10.1109/SCCC51225.2020.9281206.

36. K. Ming. Solving path planning problem based on ant colony algorithm. *2017 29th Chinese Control And Decision Conference (CCDC)*. 2017. pp. 5391-5395. doi: 10.1109/CCDC.2017.7979455.

37. A. Byerly and A. Uskov. A new parameter adaptation method for Genetic Algorithms and Ant Colony Optimization algorithms. *2016 IEEE International Conference on Electro Information Technology (EIT)*. 2016. pp. 0668-0673. doi: 10.1109/EIT.2016.7535319.

38. CHEN F. Aircraft taxiing route planning based on multiagent system., *Electronic and Automation Control Conference*. 2016. DOI:10.1109/IMCEC.2016.7867448.

39. UO Y. H., ZHOU J., LIU Y. Y.. Distributed RISE control for spacecraft formation reconfiguration with collision avoidance. *Journal of the Franklin Institute*. 2019. 356(10):5332–5352.

40. ZHENG Z. X., GUO J., GILL E.. Distributed onboard mission planning for multi-satellite systems. *Aerospace Science and Technology*. 2019. 89: 111–122.

41. ZHEN Z. Y., ZHU P., XUE Y. X., et al. Distributed intelligent self-organized mission planning of multi UAV for dynamic targets cooperative search-attack. *Chinese Journal of Aeronautics*. 2019. 32(12): 2706–2716.

42. V. P. Karamchedu. A Path from Device-to-Device to UAV-to-UAV Communications. *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*. 2020. pp. 1-5. doi: 10.1109/VTC2020-Fall49728.2020.9348841.

43. S. ur Rahman, G. -H. Kim, Y. -Z. Cho and A. Khan. Positioning of UAVs for throughput maximization in software-defined disaster area UAV communication networks. *Journal of Communications and Networks, vol. 20, no. 5*, pp. 452-463. Oct. 2018. doi: 10.1109/JCN.2018.000070.

44. R. Jangra and R. Kait. Analysis and comparison among Ant System; Ant Colony System and Max-Min Ant System with different parameters setting. *2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT)*. 2017. pp. 1-4. doi: 10.1109/CIACT.2017.7977376

45. M. I. Mohamed, G. El-Saady and A. M. Yousef. Performance Analysis of Genetic Algorithm and Ant Colony Optimization Dependent on PID Controller for Matrix Converter. *2021 International Conference on Electronic Engineering (ICEEM)*. 2021. pp. 1-6. doi: 10.1109/ICEEM52022.2021.9480640.

46. W. Jun and Y. Tian. Fault Tolerant Control of Quadrotor UAV Based on Support Vector Machine. *2019 5th International Conference on Control Science and Systems Engineering (ICCSSE)*. 2019. pp. 10-13. doi: 10.1109/ICCSSE.2019.00010.

47. S. K. Çalık. UAV path planning with multiagent Ant Colony system approach. *2016 24th Signal Processing and Communication Application Conference (SIU)*. 2016. pp. 1409-1412. doi: 10.1109/SIU.2016.7496013.

48. Z. Li and R. Han. Unmanned Aerial Vehicle Three-dimensional Trajectory Planning Based on Ant Colony Algorithm. *2018 37th Chinese Control Conference (CCC)*. 2018. pp. 9992-9995. doi: 10.23919/ChiCC.2018.8484099.

49. Y. Sun, J. Chen and C. Du. Path Planning of UAVs Based on Improved Ant Colony System. *2020 IEEE International Conference on Progress in Informatics and Computing (PIC)*. 2020. pp. 396-400. doi: 10.1109/PIC50277.2020.9350789.

50. J. Li, Y. Xiong and J. She. An improved ant colony optimization for path planning with multiple UAVs. *2021 IEEE International Conference on Mechatronics (ICM)*. 2021. pp. 1-5. doi: 10.1109/ICM46511.2021.9385695.

51. X. Ji. 2-OptACO: An Improvement of Ant Colony Optimization for UAV Path in Disaster Rescue. *2017 International Conference on Networking and Network Applications (NaNA)*. 2017. pp. 225-231. doi: 10.1109/NaNA.2017.16.

52. T. Zaza and A. Richards. Intelligent foresight for UAV routing problems. *2016 UKACC 11th International Conference on Control (CONTROL)*. 2016. pp. 1-6. doi: 10.1109/CONTROL.2016.7737644.

53. H. Li, Y. Chen, Z. Chen and H. Wu. Multi-UAV Cooperative 3D Coverage Path Planning Based on Asynchronous Ant Colony Optimization. *2021 40th Chinese Control Conference (CCC)*. 2021. pp. 4255-4260. doi: 10.23919/CCC52363.2021.9549498.

54. C. Xiang, P. Hao and X. Zhang. The Path Planning Study of Multi-task Logistics UAVs Under Complex Low Airspace. *2021 33rd Chinese Control and Decision Conference (CCDC)*. 2021. pp. 5238-5242. doi: 10.1109/CCDC52312.2021.9601885.

55. J. Luo, Z. Wang, Z. Zuo and P. Deng. Research on Dynamic Task Planning of UAV Based on Ant Colony Algorithm. *2019 IEEE 9th Annual*

International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER). 2019. pp. 1515-1519. doi: 10.1109/CYBER46603.2019.9066732.

56. T. Chen. Research on Improved Potential Field Ant Colony Algorithm for UAV Path Planning. *2021 33rd Chinese Control and Decision Conference (CCDC)*. 2021. pp. 535-539. doi: 10.1109/CCDC52312.2021.9602445.

57. Y. Sun, J. Chen, C. Du and Q. Gu. Path planning of UAVs based on improved Clustering Algorithm and Ant Colony System Algorithm. *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*. 2020. pp. 1097-1101. doi: 10.1109/ITOEC49072.2020.9141728.

58. X. Chen, R. Xu and J. Zhao. Multi-Objective Route Planning for UAV. *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*. 2017. pp. 1023-1027. doi: 10.1109/ICISCE.2017.215.

59. C. Ze-ling, W. Qi and Y. Ye-qing. Research on Optimization Method of Multi-UAV Collaborative Task Planning. *2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*. 2018. pp. 1-6. doi: 10.1109/GNCC42960.2018.9018868.

60. Y. Zhu, R. Ma and Q. Yihong. Improvement of Ant Colony Method Track Planning Based on Artificial Potential Field Method. *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2020. pp. 1394-1398. doi: 10.1109/ICARCV50220.2020.9305370.

61. Y. Zhu, R. Ma and Q. Yihong. Improvement of Ant Colony Method Track Planning Based on Artificial Potential Field Method. *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2020. pp. 1394-1398. doi: 10.1109/ICARCV50220.2020.9305370.

62. W. Qing, H. Chen, X. Wang and Y. Yin. Collision-free Trajectory Generation for UAV Swarm Formation Rendezvous. *2021 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2021. pp. 1861-1867, doi: 10.1109/ROBIO54168.2021.9739428.

ДОДАТОК А

(обов'язковий)

ЛІСТИНГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МЕТОД КЕРУВАННЯ ГРУПОЮ БЕЗПІЛОТНИХ ЛІТАЛЬНИХ АПАРАТІВ НА ОСНОВІ НЕЧІТКОГО МУРАШИНОГО АЛГОРИТМУ

Модуль «Реалізація модифікованого мурашиного алгоритму для керування БПЛА».

```
from threading import Thread


class ant_colony:
  class ant(Thread):
    def __init__(self, init_location, possible_locations, pheromone_map, distance_callback, alpha, beta, first_pass=False):
      """
      initialized an ant, to traverse the map
      init_location -> marks where in the map that the ant starts
      possible_locations -> a list of possible nodes the ant can go to
        when used internally, gives a list of possible locations the ant can traverse to _minus those nodes already visited_
      pheromone_map -> map of pheromone values for each traversal between each node
      distance_callback -> is a function to calculate the distance between two nodes
      alpha -> a parameter from the ACO algorithm to control the influence of the amount of pheromone when making a choice in _pick_path()
      beta -> a parameters from ACO that controls the influence of the distance to the next node in _pick_path()
      first_pass -> if this is a first pass on a map, then do some steps differently, noted in methods below
```

```
        route -> a list that is updated with the labels of the nodes that the ant has
traversed
        pheromone_trail -> a list of pheromone amounts deposited along the ants
trail, maps to each traversal in route
        distance_traveled -> total distance tranveled along the steps in route
        location -> marks where the ant currently is
        tour_complete -> flag to indicate the ant has completed its traversal
          used by get_route() and get_distance_traveled()
        """
        Thread.__init__(self)

        self.init_location = init_location
        self.possible_locations = possible_locations
        self.route = []
        self.distance_traveled = 0.0
        self.location = init_location
        self.pheromone_map = pheromone_map
        self.distance_callback = distance_callback
        self.alpha = alpha
        self.beta = beta
        self.first_pass = first_pass

        #append start location to route, before doing random walk
        self._update_route(init_location)

        self.tour_complete = False

    def run(self):
        """
        until self.possible_locations is empty (the ant has visited all nodes)
```

```
        _pick_path() to find a next node to traverse to
        _traverse() to:
          _update_route() (to show latest traversal)
          _update_distance_traveled() (after traversal)
      return the ants route and its distance, for use in ant_colony:
        do pheromone updates
        check for new possible optimal solution with this ants latest tour
      """
      while self.possible_locations:
        next = self._pick_path()
        self._traverse(self.location, next)

      self.tour_complete = True

    def _pick_path(self):
      """
      implements the path selection algorithm of ACO
      calculate the attractiveness of each possible transition from the current location
      then randomly choose a next path, based on its attractiveness
      """
      #on the first pass (no pheromones), then we can just choice() to find the next one
      if self.first_pass:
        import random
        return random.choice(self.possible_locations)

      attractiveness = dict()
      sum_total = 0.0
      #for each possible location, find its attractiveness (it's (pheromone amount)*1/distance [tau*eta, from the algortihm])
```

```
        #sum all attrativeness amounts for calculating probability of each route in the next step
        for possible_next_location in self.possible_locations:
          #NOTE: do all calculations as float, otherwise we get integer division at times for really hard to track down bugs
          pheromone_amount = float(self.pheromone_map[self.location][possible_next_location])
          distance = float(self.distance_callback(self.location, possible_next_location))

          #tau^alpha * eta^beta
          attractiveness[possible_next_location] = pow(pheromone_amount, self.alpha)*pow(1/distance, self.beta)
          sum_total += attractiveness[possible_next_location]

        #it is possible to have small values for pheromone amount / distance, such that with rounding errors this is equal to zero
        #rare, but handle when it happens
        if sum_total == 0.0:
          #increment all zero's, such that they are the smallest non-zero values supported by the system
          def next_up(x):
            import math
            import struct
            # NaNs and positive infinity map to themselves.
            if math.isnan(x) or (math.isinf(x) and x > 0):
              return x
            # 0.0 and -0.0 both map to the smallest +ve float.
            if x == 0.0:
              x = 0.0
            n = struct.unpack('<q', struct.pack('<d', x))[0]
```

```
        if n >= 0:
          n += 1
        else:
          n -= 1
        return struct.unpack('<d', struct.pack('<q', n))[0]

      for key in attractiveness:
        attractiveness[key] = next_up(attractiveness[key])
      sum_total = next_up(sum_total)

    #randomly choose the next path
    import random
    toss = random.random()

    cummulative = 0
    for possible_next_location in attractiveness:
      weight = (attractiveness[possible_next_location] / sum_total)
      if toss <= weight + cummulative:
        return possible_next_location
      cummulative += weight

def _traverse(self, start, end):
    """
    _update_route() to show new traversal
    _update_distance_traveled() to record new distance traveled
    self.location update to new location
    called from run()
    """
    self._update_route(end)
    self._update_distance_traveled(start, end)
```

```
        self.location = end

    def _update_route(self, new):
        """
        add new node to self.route
        remove new node form self.possible_location
        called from _traverse() & __init__()
        """
        self.route.append(new)
        self.possible_locations.remove(new)

    def _update_distance_traveled(self, start, end):
        """
        use self.distance_callback to update self.distance_traveled
        """
        self.distance_traveled += float(self.distance_callback(start, end))

    def get_route(self):
        if self.tour_complete:
            return self.route
        return None

    def get_distance_traveled(self):
        if self.tour_complete:
            return self.distance_traveled
        return None

    def __init__(self, nodes, distance_callback, start=None, ant_count=50, alpha=.5, beta=1.2, pheromone_evaporation_coefficient=.40, pheromone_constant=1000.0, iterations=80):
        """
```

nodes -> is assumed to be a dict() mapping node ids to values
that are understandable by distance_callback

distance_callback -> is assumed to take a pair of coordinates and return the distance between them
populated into distance_matrix on each call to get_distance()

start -> if set, then is assumed to be the node where all ants start their traversal
if unset, then assumed to be the first key of nodes when sorted()

distance_matrix -> holds values of distances calculated between nodes
populated on demand by _get_distance()

pheromone_map -> holds final values of pheromones
used by ants to determine traversals
pheromone dissipation happens to these values first, before adding pheromone values from the ants during their traversal
(in ant_updated_pheromone_map)

ant_updated_pheromone_map -> a matrix to hold the pheromone values that the ants lay down
not used to dissipate, values from here are added to pheromone_map after dissipation step
(reset for each traversal)

alpha -> a parameter from the ACO algorithm to control the influence of the amount of pheromone when an ant makes a choice

beta -> a parameters from ACO that controls the influence of the distance to the next node in ant choice making

pheromone_constant -> a parameter used in depositing pheromones on the map (Q in ACO algorithm)
used by _update_pheromone_map()

pheromone_evaporation_coefficient -> a parameter used in removing pheromone values from the pheromone_map (rho in ACO algorithm)
used by _update_pheromone_map()

ants -> holds worker ants
they traverse the map as per ACO
notable properties:
total distance traveled
route

first_pass -> flags a first pass for the ants, which triggers unique behavior

iterations -> how many iterations to let the ants traverse the map

shortest_distance -> the shortest distance seen from an ant traversal

shortets_path_seen -> the shortest path seen from a traversal (shortest_distance is the distance along this path)
"""
#nodes
if type(nodes) is not dict:
raise TypeError("nodes must be dict")

if len(nodes) < 1:

```
            raise ValueError("there must be at least one node in dict nodes")

        #create internal mapping and mapping for return to caller
        self.id_to_key, self.nodes = self._init_nodes(nodes)
        #create matrix to hold distance calculations between nodes
        self.distance_matrix = self._init_matrix(len(nodes))
        #create matrix for master pheromone map, that records pheromone amounts
along routes
        self.pheromone_map = self._init_matrix(len(nodes))
        #create a matrix for ants to add their pheromones to, before adding those to
pheromone_map during the update_pheromone_map step
        self.ant_updated_pheromone_map = self._init_matrix(len(nodes))

        #distance_callback
        if not callable(distance_callback):
            raise TypeError("distance_callback is not callable, should be method")

        self.distance_callback = distance_callback

        #start
        if start is None:
            self.start = 0
        else:
            self.start = None
            #init start to internal id of node id passed
            for key, value in self.id_to_key.items():
                if value == start:
                    self.start = key

            #if we didn't find a key in the nodes passed in, then raise
            if self.start is None:
```

```
            raise KeyError("Key: " + str(start) + " not found in the nodes dict passed.")

        #ant_count
        if type(ant_count) is not int:
          raise TypeError("ant_count must be int")

        if ant_count < 1:
          raise ValueError("ant_count must be >= 1")

        self.ant_count = ant_count
        #alpha
        if (type(alpha) is not int) and type(alpha) is not float:
          raise TypeError("alpha must be int or float")

        if alpha < 0:
          raise ValueError("alpha must be >= 0")

        self.alpha = float(alpha)

        #beta
        if (type(beta) is not int) and type(beta) is not float:
          raise TypeError("beta must be int or float")

        if beta < 1:
          raise ValueError("beta must be >= 1")

        self.beta = float(beta)

        #pheromone_evaporation_coefficient
```

```
        if (type(pheromone_evaporation_coefficient) is not int) and
type(pheromone_evaporation_coefficient) is not float:
            raise TypeError("pheromone_evaporation_coefficient must be int or
float")

        self.pheromone_evaporation_coefficient =
float(pheromone_evaporation_coefficient)

        #pheromone_constant
        if (type(pheromone_constant) is not int) and type(pheromone_constant) is
not float:
            raise TypeError("pheromone_constant must be int or float")

        self.pheromone_constant = float(pheromone_constant)

        #iterations
        if (type(iterations) is not int):
            raise TypeError("iterations must be int")

        if iterations < 0:
            raise ValueError("iterations must be >= 0")

        self.iterations = iterations

        #other internal variable init
        self.first_pass = True
        self.ants = self._init_ants(self.start)
        self.shortest_distance = None
        self.shortest_path_seen = None

    def _get_distance(self, start, end):
```

```
        """
        uses the distance_callback to return the distance between nodes
        if a distance has not been calculated before, then it is populated in
distance_matrix and returned
        if a distance has been called before, then its value is returned from
distance_matrix
        """
        if not self.distance_matrix[start][end]:
          distance = self.distance_callback(self.nodes[start], self.nodes[end])

          if (type(distance) is not int) and (type(distance) is not float):
            raise TypeError("distance_callback should return either int or float, saw:
"+ str(type(distance)))

          self.distance_matrix[start][end] = float(distance)
          return distance
        return self.distance_matrix[start][end]
      def _init_nodes(self, nodes):
        """
        create a mapping of internal id numbers (0 .. n) to the keys in the nodes
passed
        create a mapping of the id's to the values of nodes
        we use id_to_key to return the route in the node names the caller expects in
mainloop()
        """
        id_to_key = dict()
        id_to_values = dict()
        id = 0
        for key in sorted(nodes.keys()):
          id_to_key[id] = key
          id_to_values[id] = nodes[key]
```

```
            id += 1

        return id_to_key, id_to_values
    def _init_matrix(self, size, value=0.0):
        """
        setup a matrix NxN (where n = size)
        used in both self.distance_matrix and self.pheromone_map
        as they require identical matrixes besides which value to initialize to
        """
        ret = []
        for row in range(size):
            ret.append([float(value) for x in range(size)])
        return ret

    def _init_ants(self, start):
        """
        on first pass:
            create a number of ant objects
        on subsequent passes, just call __init__ on each to reset them
        by default, all ants start at the first node, 0
        """
        #allocate new ants on the first pass
        if self.first_pass:
            return [self.ant(start, self.nodes.keys(), self.pheromone_map,
self._get_distance,
                self.alpha, self.beta, first_pass=True) for x in range(self.ant_count)]
        #else, just reset them to use on another pass
        for ant in self.ants:
            ant.__init__(start, self.nodes.keys(), self.pheromone_map,
self._get_distance, self.alpha, self.beta)
```

```
    def _update_pheromone_map(self):
        """
        1) Update self.pheromone_map by decaying values contained therein via the
ACO algorithm
        2) Add pheromone_values from all ants from ant_updated_pheromone_map
        called by:
          mainloop()
          (after all ants have traveresed)
        """
        #always a square matrix
        for start in range(len(self.pheromone_map)):
          for end in range(len(self.pheromone_map)):
            #decay the pheromone value at this location
            #tau_xy <- (1-rho)*tau_xy  (ACO)
            self.pheromone_map[start][end]                    =                    (1-
self.pheromone_evaporation_coefficient)*self.pheromone_map[start][end]

            #then add all contributions to this location for each ant that travered it
            #(ACO)
            #tau_xy <- tau_xy + delta tau_xy_k
            #  delta tau_xy_k = Q / L_k
            self.pheromone_map[start][end]                                        +=
self.ant_updated_pheromone_map[start][end]

    def _populate_ant_updated_pheromone_map(self, ant):
        """
        given an ant, populate ant_updated_pheromone_map with pheromone
values according to ACO
        along the ant's route
        called from:
          mainloop()
```

```
            ( before _update_pheromone_map() )
            """
            route = ant.get_route()
            for i in range(len(route)-1):
              #find the pheromone over the route the ant traversed
              current_pheromone_value = float(self.ant_updated_pheromone_map[route[i]][route[i+1]])

              #update the pheromone along that section of the route
              #(ACO)
              #  delta tau_xy_k = Q / L_k
              new_pheromone_value = self.pheromone_constant/ant.get_distance_traveled()

              self.ant_updated_pheromone_map[route[i]][route[i+1]] = current_pheromone_value + new_pheromone_value
              self.ant_updated_pheromone_map[route[i+1]][route[i]] = current_pheromone_value + new_pheromone_value

          def mainloop(self):
            """
            Runs the worker ants, collects their returns and updates the pheromone map with pheromone values from workers
              calls:
              _update_pheromones()
              ant.run()
            runs the simulation self.iterations times
            """

            for _ in range(self.iterations):
              #start the multi-threaded ants, calls ant.run() in a new thread
```

```
            for ant in self.ants:
                ant.start()

            #wait until the ants are finished, before moving on to modifying shared resources
            for ant in self.ants:
                ant.join()

            for ant in self.ants:
                #update ant_updated_pheromone_map with this ant's constribution of pheromones along its route
                self._populate_ant_updated_pheromone_map(ant)

                #if we haven't seen any paths yet, then populate for comparisons later
                if not self.shortest_distance:
                    self.shortest_distance = ant.get_distance_traveled()

                if not self.shortest_path_seen:
                    self.shortest_path_seen = ant.get_route()

                #if we see a shorter path, then save for return
                if ant.get_distance_traveled() < self.shortest_distance:
                    self.shortest_distance = ant.get_distance_traveled()
                    self.shortest_path_seen = ant.get_route()

            #decay current pheromone values and add all pheromone values we saw during traversal (from ant_updated_pheromone_map)
            self._update_pheromone_map()

            #flag that we finished the first pass of the ants traversal
            if self.first_pass:
```

```
            self.first_pass = False

        #reset all ants to default for the next iteration
        self._init_ants(self.start)

        #reset ant_updated_pheromone_map to record pheromones for ants on next pass
        self.ant_updated_pheromone_map = self._init_matrix(len(self.nodes), value=0)

    #translate shortest path back into callers node id's
    ret = []
    for id in self.shortest_path_seen:
        ret.append(self.id_to_key[id])

    return ret
```

Модуль «Реалізація методу оптимізації мурашиного алгоритму для керування БПЛА».

```
import numpy as np
import matplotlib.pyplot as plt
import time

import warnings

warnings.filterwarnings("ignore")


class AntColonyOptimizer:
```

```
        def __init__(self, ants, evaporation_rate, intensification, alpha=1.0, beta=0.0,
beta_evaporation_rate=0,
                     choose_best=.1):
            """
            Ant colony optimizer.  Traverses a graph and finds either the max or min
distance between nodes.
            :param ants: number of ants to traverse the graph
            :param evaporation_rate: rate at which pheromone evaporates
            :param intensification: constant added to the best path
            :param alpha: weighting of pheromone
            :param beta: weighting of heuristic (1/distance)
            :param beta_evaporation_rate: rate at which beta decays (optional)
            :param choose_best: probability to choose the best route
            """
            # Parameters
            self.ants = ants
            self.evaporation_rate = evaporation_rate
            self.pheromone_intensification = intensification
            self.heuristic_alpha = alpha
            self.heuristic_beta = beta
            self.beta_evaporation_rate = beta_evaporation_rate
            self.choose_best = choose_best

            # Internal representations
            self.pheromone_matrix = None
            self.heuristic_matrix = None
            self.probability_matrix = None

            self.map = None
            self.set_of_available_nodes = None
```

```
        # Internal stats
        self.best_series = []
        self.best = None
        self.fitted = False
        self.best_path = None
        self.fit_time = None

        # Plotting values
        self.stopped_early = False

    def __str__(self):
        string = "Ant Colony Optimizer"
        string += "\n--------------------"
        string += "\nDesigned to optimize either the minimum or maximum distance between nodes in a square matrix that behaves like a distance matrix."
        string += "\n--------------------"
        string += "\nNumber of ants:\t\t\t\t{}".format(self.ants)
        string += "\nEvaporation rate:\t\t\t{}".format(self.evaporation_rate)
        string += "\nIntensification factor:\t\t{}".format(self.pheromone_intensification)
        string += "\nAlpha Heuristic:\t\t\t{}".format(self.heuristic_alpha)
        string += "\nBeta Heuristic:\t\t\t\t{}".format(self.heuristic_beta)
        string += "\nBeta Evaporation Rate:\t\t{}".format(self.beta_evaporation_rate)
        string += "\nChoose Best Percentage:\t\t{}".format(self.choose_best)
        string += "\n--------------------"
        string += "\nUSAGE:"
        string += "\nNumber of ants influences how many paths are explored each iteration."
        string += "\nThe alpha and beta heuristics affect how much influence the pheromones or the distance heuristic weigh an ants' decisions."
```

```
            string += "\nBeta evaporation reduces the influence of the heuristic over time."
            string += "\nChoose best is a percentage of how often an ant will choose the best route over probabilistically choosing a route based on pheromones."
            string += "\n--------------------"
            if self.fitted:
                string += "\n\nThis optimizer has been fitted."
            else:
                string += "\n\nThis optimizer has NOT been fitted."
            return string


        def _initialize(self):
            """
            Initializes the model by creating the various matrices and generating the list of available nodes
            """
            assert self.map.shape[0] == self.map.shape[1], "Map is not a distance matrix!"
            num_nodes = self.map.shape[0]
            self.pheromone_matrix = np.ones((num_nodes, num_nodes))
            # Remove the diagonal since there is no pheromone from node i to itself
            self.pheromone_matrix[np.eye(num_nodes) == 1] = 0
            self.heuristic_matrix = 1 / self.map
            self.probability_matrix = (self.pheromone_matrix ** self.heuristic_alpha) * (
                    self.heuristic_matrix ** self.heuristic_beta)  # element by element multiplcation
            self.set_of_available_nodes = list(range(num_nodes))


        def _reinstate_nodes(self):
            """
```

```
        Resets available nodes to all nodes for the next iteration
        """
        self.set_of_available_nodes = list(range(self.map.shape[0]))

    def _update_probabilities(self):
        """
        After evaporation and intensification, the probability matrix needs to be updated.  This function
        does that.
        """
        self.probability_matrix = (self.pheromone_matrix ** self.heuristic_alpha) * (
                self.heuristic_matrix ** self.heuristic_beta)

    def _choose_next_node(self, from_node):
        """
        Chooses the next node based on probabilities.  If p < p_choose_best, then the best path is chosen, otherwise
        it is selected from a probability distribution weighted by the pheromone.
        :param from_node: the node the ant is coming from
        :return: index of the node the ant is going to
        """
        numerator = self.probability_matrix[from_node, self.set_of_available_nodes]
        if np.random.random() < self.choose_best:
            next_node = np.argmax(numerator)
        else:
            denominator = np.sum(numerator)
            probabilities = numerator / denominator
            next_node = np.random.choice(range(len(probabilities)), p=probabilities)
```

```
            return next_node

        def _remove_node(self, node):
            self.set_of_available_nodes.remove(node)

        def _evaluate(self, paths, mode):
            """
            Evaluates the solutions of the ants by adding up the distances between nodes.
            :param paths: solutions from the ants
            :param mode: max or min
            :return: x and y coordinates of the best path as a tuple, the best path, and the best score
            """
            scores = np.zeros(len(paths))
            coordinates_i = []
            coordinates_j = []
            for index, path in enumerate(paths):
                score = 0
                coords_i = []
                coords_j = []
                for i in range(len(path) - 1):
                    coords_i.append(path[i])
                    coords_j.append(path[i + 1])
                    score += self.map[path[i], path[i + 1]]
                scores[index] = score
                coordinates_i.append(coords_i)
                coordinates_j.append(coords_j)
            if mode == 'min':
                best = np.argmin(scores)
            elif mode == 'max':
```

```
            best = np.argmax(scores)
        return (coordinates_i[best], coordinates_j[best]), paths[best], scores[best]

    def _evaporation(self):
        """
        Evaporate some pheromone as the inverse of the evaporation rate.  Also evaporates beta if desired.
        """
        self.pheromone_matrix *= (1 - self.evaporation_rate)
        self.heuristic_beta *= (1 - self.beta_evaporation_rate)

    def _intensify(self, best_coords):
        """
        Increases the pheromone by some scalar for the best route.
        :param best_coords: x and y (i and j) coordinates of the best route
        """
        i = best_coords[0]
        j = best_coords[1]
        self.pheromone_matrix[i, j] += self.pheromone_intensification

    def fit(self, map_matrix, iterations=100, mode='min', early_stopping_count=20, verbose=True):
        """
        Fits the ACO to a specific map.  This was designed with the Traveling Salesman problem in mind.
        :param map_matrix: Distance matrix or some other matrix with similar properties
        :param iterations: number of iterations
        :param mode: whether to get the minimum path or maximum path
        :param early_stopping_count: how many iterations of the same score to make the algorithm stop early
```

```
        :return: the best score
        """
        if verbose: print("Beginning ACO Optimization with {} iterations...".format(iterations))
        self.map = map_matrix
        start = time.time()
        self._initialize()
        num_equal = 0

        for i in range(iterations):
            start_iter = time.time()
            paths = []
            path = []

            for ant in range(self.ants):
                current_node = self.set_of_available_nodes[np.random.randint(0, len(self.set_of_available_nodes))]
                start_node = current_node
                while True:
                    path.append(current_node)
                    self._remove_node(current_node)
                    if len(self.set_of_available_nodes) != 0:
                        current_node_index = self._choose_next_node(current_node)
                        current_node = self.set_of_available_nodes[current_node_index]
                    else:
                        break

                path.append(start_node)  # go back to start
                self._reinstate_nodes()
                paths.append(path)
```

```
            path = []

        best_path_coords, best_path, best_score = self._evaluate(paths, mode)

        if i == 0:
            best_score_so_far = best_score
        else:
            if mode == 'min':
                if best_score < best_score_so_far:
                    best_score_so_far = best_score
                    self.best_path = best_path
            elif mode == 'max':
                if best_score > best_score_so_far:
                    best_score_so_far = best_score
                    self.best_path = best_path

        if best_score == best_score_so_far:
            num_equal += 1
        else:
            num_equal = 0

        self.best_series.append(best_score)
        self._evaporation()
        self._intensify(best_path_coords)
        self._update_probabilities()

        if verbose: print("Best score at iteration {}: {}; overall: {} ({}s)"
                          "".format(i, round(best_score, 2), round(best_score_so_far,
2),
                                    round(time.time() - start_iter)))
```

```
                if best_score == best_score_so_far and num_equal == early_stopping_count:
                    self.stopped_early = True
                    print("Stopping early due to {} iterations of the same score.".format(early_stopping_count))
                    break

            self.fit_time = round(time.time() - start)
            self.fitted = True

            if mode == 'min':
                self.best = self.best_series[np.argmin(self.best_series)]
                if verbose: print(
                    "ACO fitted. Runtime: {} minutes. Best score: {}".format(self.fit_time // 60, self.best))
                return self.best
            elif mode == 'max':
                self.best = self.best_series[np.argmax(self.best_series)]
                if verbose: print(
                    "ACO fitted. Runtime: {} minutes. Best score: {}".format(self.fit_time // 60, self.best))
                return self.best
            else:
                raise ValueError("Invalid mode!  Choose 'min' or 'max'.")

        def plot(self):
            """
            Plots the score over time after the model has been fitted.
            :return: None if the model isn't fitted yet
            """
            if not self.fitted:
```

```
                print("Ant Colony Optimizer not fitted!  There exists nothing to plot.")
                return None
            else:
                fig, ax = plt.subplots(figsize=(20, 15))
                ax.plot(self.best_series, label="Best Run")
                ax.set_xlabel("Iteration")
                ax.set_ylabel("Performance")
                ax.text(.8, .6,
                        'Ants: {}\nEvap Rate: {}\nIntensify: {}\nAlpha: {}\nBeta: {}\nBeta Evap: {}\nChoose Best: {}\n\nFit Time: {}m{}'.format(
                            self.ants, self.evaporation_rate, self.pheromone_intensification, self.heuristic_alpha,
                            self.heuristic_beta, self.beta_evaporation_rate, self.choose_best, self.fit_time // 60,
                            ["\nStopped Early!" if self.stopped_early else ""][0]),
                        bbox={'facecolor': 'gray', 'alpha': 0.8, 'pad': 10}, transform=ax.transAxes)
                ax.legend()
                plt.title("Ant Colony Optimization Results (best: {})".format(np.round(self.best, 2)))
                plt.show()
        import numpy as np
        import matplotlib.pyplot as plt
        import time

        import warnings

        warnings.filterwarnings("ignore")


        class AntColonyOptimizer:
```

```
        def __init__(self, ants, evaporation_rate, intensification, alpha=1.0, beta=0.0,
beta_evaporation_rate=0,
                     choose_best=.1):
            """
            Ant colony optimizer.  Traverses a graph and finds either the max or min
distance between nodes.
            :param ants: number of ants to traverse the graph
            :param evaporation_rate: rate at which pheromone evaporates
            :param intensification: constant added to the best path
            :param alpha: weighting of pheromone
            :param beta: weighting of heuristic (1/distance)
            :param beta_evaporation_rate: rate at which beta decays (optional)
            :param choose_best: probability to choose the best route
            """
            # Parameters
            self.ants = ants
            self.evaporation_rate = evaporation_rate
            self.pheromone_intensification = intensification
            self.heuristic_alpha = alpha
            self.heuristic_beta = beta
            self.beta_evaporation_rate = beta_evaporation_rate
            self.choose_best = choose_best

            # Internal representations
            self.pheromone_matrix = None
            self.heuristic_matrix = None
            self.probability_matrix = None

            self.map = None
            self.set_of_available_nodes = None
```

```
        # Internal stats
        self.best_series = []
        self.best = None
        self.fitted = False
        self.best_path = None
        self.fit_time = None

        # Plotting values
        self.stopped_early = False

    def __str__(self):
        string = "Ant Colony Optimizer"
        string += "\n----------"
        string += "\nDesigned to optimize either the minimum or maximum distance between nodes in a square matrix that behaves like a distance matrix."
        string += "\n----------"
        string += "\nNumber of ants:\t\t\t\t{}".format(self.ants)
        string += "\nEvaporation rate:\t\t\t{}".format(self.evaporation_rate)
        string += "\nIntensification factor:\t\t{}".format(self.pheromone_intensification)
        string += "\nAlpha Heuristic:\t\t\t{}".format(self.heuristic_alpha)
        string += "\nBeta Heuristic:\t\t\t\t{}".format(self.heuristic_beta)
        string += "\nBeta Evaporation Rate:\t\t{}".format(self.beta_evaporation_rate)
        string += "\nChoose Best Percentage:\t\t{}".format(self.choose_best)
        string += "\n----------"
        string += "\nUSAGE:"
        string += "\nNumber of ants influences how many paths are explored each iteration."
        string += "\nThe alpha and beta heuristics affect how much influence the pheromones or the distance heuristic weigh an ants' decisions."
```

```
            string += "\nBeta evaporation reduces the influence of the heuristic over time."
            string += "\nChoose best is a percentage of how often an ant will choose the best route over probabilistically choosing a route based on pheromones."
            string += "\n--------------------"
            if self.fitted:
                string += "\n\nThis optimizer has been fitted."
            else:
                string += "\n\nThis optimizer has NOT been fitted."
            return string


        def _initialize(self):
            """
            Initializes the model by creating the various matrices and generating the list of available nodes
            """
            assert self.map.shape[0] == self.map.shape[1], "Map is not a distance matrix!"
            num_nodes = self.map.shape[0]
            self.pheromone_matrix = np.ones((num_nodes, num_nodes))
            # Remove the diagonal since there is no pheromone from node i to itself
            self.pheromone_matrix[np.eye(num_nodes) == 1] = 0
            self.heuristic_matrix = 1 / self.map
            self.probability_matrix = (self.pheromone_matrix ** self.heuristic_alpha) * (
                    self.heuristic_matrix ** self.heuristic_beta)  # element by element multiplcation
            self.set_of_available_nodes = list(range(num_nodes))


        def _reinstate_nodes(self):
            """
```

```
        Resets available nodes to all nodes for the next iteration
        """
        self.set_of_available_nodes = list(range(self.map.shape[0]))

    def _update_probabilities(self):
        """
        After evaporation and intensification, the probability matrix needs to be updated.  This function
        does that.
        """
        self.probability_matrix = (self.pheromone_matrix ** self.heuristic_alpha) * (
                self.heuristic_matrix ** self.heuristic_beta)

    def _choose_next_node(self, from_node):
        """
        Chooses the next node based on probabilities.  If p < p_choose_best, then the best path is chosen, otherwise
        it is selected from a probability distribution weighted by the pheromone.
        :param from_node: the node the ant is coming from
        :return: index of the node the ant is going to
        """
        numerator = self.probability_matrix[from_node, self.set_of_available_nodes]
        if np.random.random() < self.choose_best:
            next_node = np.argmax(numerator)
        else:
            denominator = np.sum(numerator)
            probabilities = numerator / denominator
            next_node = np.random.choice(range(len(probabilities)), p=probabilities)
```

```
            return next_node

        def _remove_node(self, node):
            self.set_of_available_nodes.remove(node)

        def _evaluate(self, paths, mode):
            """
            Evaluates the solutions of the ants by adding up the distances between nodes.
            :param paths: solutions from the ants
            :param mode: max or min
            :return: x and y coordinates of the best path as a tuple, the best path, and the best score
            """
            scores = np.zeros(len(paths))
            coordinates_i = []
            coordinates_j = []
            for index, path in enumerate(paths):
                score = 0
                coords_i = []
                coords_j = []
                for i in range(len(path) - 1):
                    coords_i.append(path[i])
                    coords_j.append(path[i + 1])
                    score += self.map[path[i], path[i + 1]]
                scores[index] = score
                coordinates_i.append(coords_i)
                coordinates_j.append(coords_j)
            if mode == 'min':
                best = np.argmin(scores)
            elif mode == 'max':
```

```
            best = np.argmax(scores)
        return (coordinates_i[best], coordinates_j[best]), paths[best], scores[best]

    def _evaporation(self):
        """
        Evaporate some pheromone as the inverse of the evaporation rate.  Also evaporates beta if desired.
        """
        self.pheromone_matrix *= (1 - self.evaporation_rate)
        self.heuristic_beta *= (1 - self.beta_evaporation_rate)

    def _intensify(self, best_coords):
        """
        Increases the pheromone by some scalar for the best route.
        :param best_coords: x and y (i and j) coordinates of the best route
        """
        i = best_coords[0]
        j = best_coords[1]
        self.pheromone_matrix[i, j] += self.pheromone_intensification

    def fit(self, map_matrix, iterations=100, mode='min', early_stopping_count=20, verbose=True):
        """
        Fits the ACO to a specific map.  This was designed with the Traveling Salesman problem in mind.
        :param map_matrix: Distance matrix or some other matrix with similar properties
        :param iterations: number of iterations
        :param mode: whether to get the minimum path or maximum path
        :param early_stopping_count: how many iterations of the same score to make the algorithm stop early
```

```
        :return: the best score
        """
        if verbose: print("Beginning ACO Optimization with {} iterations...".format(iterations))
        self.map = map_matrix
        start = time.time()
        self._initialize()
        num_equal = 0

        for i in range(iterations):
            start_iter = time.time()
            paths = []
            path = []

            for ant in range(self.ants):
                current_node = self.set_of_available_nodes[np.random.randint(0, len(self.set_of_available_nodes))]
                start_node = current_node
                while True:
                    path.append(current_node)
                    self._remove_node(current_node)
                    if len(self.set_of_available_nodes) != 0:
                        current_node_index = self._choose_next_node(current_node)
                        current_node = self.set_of_available_nodes[current_node_index]
                    else:
                        break

                path.append(start_node)  # go back to start
                self._reinstate_nodes()
                paths.append(path)
```

```
            path = []

        best_path_coords, best_path, best_score = self._evaluate(paths, mode)

        if i == 0:
            best_score_so_far = best_score
        else:
            if mode == 'min':
                if best_score < best_score_so_far:
                    best_score_so_far = best_score
                    self.best_path = best_path
            elif mode == 'max':
                if best_score > best_score_so_far:
                    best_score_so_far = best_score
                    self.best_path = best_path

        if best_score == best_score_so_far:
            num_equal += 1
        else:
            num_equal = 0

        self.best_series.append(best_score)
        self._evaporation()
        self._intensify(best_path_coords)
        self._update_probabilities()

        if verbose: print("Best score at iteration {}: {}; overall: {} ({}s)"
                          "".format(i, round(best_score, 2), round(best_score_so_far,
2),
                                    round(time.time() - start_iter)))
```

```
                if best_score == best_score_so_far and num_equal == early_stopping_count:
                    self.stopped_early = True
                    print("Stopping early due to {} iterations of the same score.".format(early_stopping_count))
                    break

            self.fit_time = round(time.time() - start)
            self.fitted = True

            if mode == 'min':
                self.best = self.best_series[np.argmin(self.best_series)]
                if verbose: print(
                    "ACO fitted. Runtime: {} minutes. Best score: {}".format(self.fit_time // 60, self.best))
                return self.best
            elif mode == 'max':
                self.best = self.best_series[np.argmax(self.best_series)]
                if verbose: print(
                    "ACO fitted. Runtime: {} minutes. Best score: {}".format(self.fit_time // 60, self.best))
                return self.best
            else:
                raise ValueError("Invalid mode!  Choose 'min' or 'max'.")

        def plot(self):
            """
            Plots the score over time after the model has been fitted.
            :return: None if the model isn't fitted yet
            """
            if not self.fitted:
```

```
            print("Ant Colony Optimizer not fitted!  There exists nothing to plot.")
            return None
        else:
            fig, ax = plt.subplots(figsize=(20, 15))
            ax.plot(self.best_series, label="Best Run")
            ax.set_xlabel("Iteration")
            ax.set_ylabel("Performance")
            ax.text(.8, .6,
                    'Ants: {}\nEvap Rate: {}\nIntensify: {}\nAlpha: {}\nBeta: {}\nBeta Evap: {}\nChoose Best: {}\n\nFit Time: {}m{}'.format(
                        self.ants, self.evaporation_rate, self.pheromone_intensification, self.heuristic_alpha,
                        self.heuristic_beta, self.beta_evaporation_rate, self.choose_best, self.fit_time // 60,
                        ["\nStopped Early!" if self.stopped_early else ""][0]),
                    bbox={'facecolor': 'gray', 'alpha': 0.8, 'pad': 10}, transform=ax.transAxes)
            ax.legend()
            plt.title("Ant Colony Optimization Results (best: {})".format(np.round(self.best, 2)))
            plt.show()
```

Модуль «Програмна реалізація опису типу БПЛА».

```
"""

Визначення класу літального апарату : квадротор

"""

from math import cos, sin
```

```
import numpy as np
import matplotlib.pyplot as plt

class Quadrotor():
    def __init__(self, x=0, y=0, z=0, roll=0, pitch=0, yaw=0, size=0.25, show_animation=True):
        self.p1 = np.array([size / 2, 0, 0, 1]).T
        self.p2 = np.array([-size / 2, 0, 0, 1]).T
        self.p3 = np.array([0, size / 2, 0, 1]).T
        self.p4 = np.array([0, -size / 2, 0, 1]).T

        self.x_data = []
        self.y_data = []
        self.z_data = []
        self.show_animation = show_animation

        if self.show_animation:
            plt.ion()
            fig = plt.figure()
            # для зупинки симуляції клавішою ESC
            fig.canvas.mpl_connect('key_release_event',
                    lambda event: [exit(0) if event.key == 'escape' else None])

            self.ax = fig.add_subplot(111, projection='3d')

        self.update_pose(x, y, z, roll, pitch, yaw)

    def update_pose(self, x, y, z, roll, pitch, yaw):
        self.x = x
        self.y = y
        self.z = z
```

```
            self.roll = roll
            self.pitch = pitch
            self.yaw = yaw
            self.x_data.append(x)
            self.y_data.append(y)
            self.z_data.append(z)

            if self.show_animation:
                self.plot()

        def transformation_matrix(self):
            x = self.x
            y = self.y
            z = self.z
            roll = self.roll
            pitch = self.pitch
            yaw = self.yaw
            return np.array(
                [[cos(yaw) * cos(pitch), -sin(yaw) * cos(roll) + cos(yaw) * sin(pitch) *
sin(roll), sin(yaw) * sin(roll) + cos(yaw) * sin(pitch) * cos(roll), x],
                 [sin(yaw) * cos(pitch), cos(yaw) * cos(roll) + sin(yaw) * sin(pitch)
                  * sin(roll), -cos(yaw) * sin(roll) + sin(yaw) * sin(pitch) * cos(roll), y],
                 [-sin(pitch), cos(pitch) * sin(roll), cos(pitch) * cos(yaw), z]
                 ])

        def plot(self):  # pragma: no cover
            T = self.transformation_matrix()

            p1_t = np.matmul(T, self.p1)
            p2_t = np.matmul(T, self.p2)
            p3_t = np.matmul(T, self.p3)
```

```
        p4_t = np.matmul(T, self.p4)

        plt.cla()

        self.ax.plot([p1_t[0], p2_t[0], p3_t[0], p4_t[0]],
                     [p1_t[1], p2_t[1], p3_t[1], p4_t[1]],
                     [p1_t[2], p2_t[2], p3_t[2], p4_t[2]], 'k.')

        self.ax.plot([p1_t[0], p2_t[0]], [p1_t[1], p2_t[1]],
                     [p1_t[2], p2_t[2]], 'r-')
        self.ax.plot([p3_t[0], p4_t[0]], [p3_t[1], p4_t[1]],
                     [p3_t[2], p4_t[2]], 'r-')

        self.ax.plot(self.x_data, self.y_data, self.z_data, 'b:')

        plt.xlim(-10, 10)
        plt.ylim(-10, 10)
        self.ax.set_zlim(0, 20)

        plt.pause(0.001)
```

Модуль «Генерація квінтичної поліноміальної траєкторії БПЛА».

```
"""
Генерація квінтичної поліноміальної траєкторії

"""

import numpy as np

class TrajectoryGenerator():
```

```
        def __init__(self, start_pos, des_pos, T, start_vel=[0,0,0], des_vel=[0,0,0],
start_acc=[0,0,0], des_acc=[0,0,0]):
            self.start_x = start_pos[0]
            self.start_y = start_pos[1]
            self.start_z = start_pos[2]

            self.des_x = des_pos[0]
            self.des_y = des_pos[1]
            self.des_z = des_pos[2]

            self.start_x_vel = start_vel[0]
            self.start_y_vel = start_vel[1]
            self.start_z_vel = start_vel[2]

            self.des_x_vel = des_vel[0]
            self.des_y_vel = des_vel[1]
            self.des_z_vel = des_vel[2]

            self.start_x_acc = start_acc[0]
            self.start_y_acc = start_acc[1]
            self.start_z_acc = start_acc[2]

            self.des_x_acc = des_acc[0]
            self.des_y_acc = des_acc[1]
            self.des_z_acc = des_acc[2]

            self.T = T

        def solve(self):
            A = np.array(
                [[0, 0, 0, 0, 0, 1],
```

```
    [self.T**5, self.T**4, self.T**3, self.T**2, self.T, 1],
    [0, 0, 0, 0, 1, 0],
    [5*self.T**4, 4*self.T**3, 3*self.T**2, 2*self.T, 1, 0],
    [0, 0, 0, 2, 0, 0],
    [20*self.T**3, 12*self.T**2, 6*self.T, 2, 0, 0]
   ])

b_x = np.array(
   [[self.start_x],
    [self.des_x],
    [self.start_x_vel],
    [self.des_x_vel],
    [self.start_x_acc],
    [self.des_x_acc]
   ])

b_y = np.array(
   [[self.start_y],
    [self.des_y],
    [self.start_y_vel],
    [self.des_y_vel],
    [self.start_y_acc],
    [self.des_y_acc]
   ])

b_z = np.array(
   [[self.start_z],
    [self.des_z],
    [self.start_z_vel],
    [self.des_z_vel],
    [self.start_z_acc],
```

```
            [self.des_z_acc]
           ])

        self.x_c = np.linalg.solve(A, b_x)
        self.y_c = np.linalg.solve(A, b_y)
        self.z_c = np.linalg.solve(A, b_z)
```

Модуль «Симуляція рубу квадротора по трьох вимірній траекторії».

```
"""

Симуляція руху квадротору по 3д траекторії

"""

from math import cos, sin
import numpy as np
from Quadrotor import Quadrotor
from TrajectoryGenerator import TrajectoryGenerator
from mpl_toolkits.mplot3d import Axes3D

show_animation = True

# Simulation parameters
g = 9.81
m = 0.2
Ixx = 1
Iyy = 1
Izz = 1
T = 5
```

```
# Proportional coefficients
Kp_x = 1
Kp_y = 1
Kp_z = 1
Kp_roll = 25
Kp_pitch = 25
Kp_yaw = 25

# Derivative coefficients
Kd_x = 10
Kd_y = 10
Kd_z = 1


def quad_sim(x_c, y_c, z_c):
    """
    Calculates the necessary thrust and torques for the quadrotor to
    follow the trajectory described by the sets of coefficients
    x_c, y_c, and z_c.

    """
    x_pos = -5
    y_pos = -5
    z_pos = 5
    x_vel = 0
    y_vel = 0
    z_vel = 0
    x_acc = 0
    y_acc = 0
    z_acc = 0
    roll = 0
```

```
pitch = 0
yaw = 0
roll_vel = 0
pitch_vel = 0
yaw_vel = 0

des_yaw = 0

dt = 0.1
t = 0

q = Quadrotor(x=x_pos, y=y_pos, z=z_pos, roll=roll,
              pitch=pitch, yaw=yaw, size=1, show_animation=show_animation)

i = 0
n_run = 8
irun = 0

while True:
    while t <= T:
        des_x_pos = calculate_position(x_c[i], t)
        des_y_pos = calculate_position(y_c[i], t)
        des_z_pos = calculate_position(z_c[i], t)
        des_x_vel = calculate_velocity(x_c[i], t)
        des_y_vel = calculate_velocity(y_c[i], t)
        des_z_vel = calculate_velocity(z_c[i], t)
        des_x_acc = calculate_acceleration(x_c[i], t)
        des_y_acc = calculate_acceleration(y_c[i], t)
        des_z_acc = calculate_acceleration(z_c[i], t)

        thrust = m * (g + des_z_acc + Kp_z * (des_z_pos -
```

```
                                    z_pos) + Kd_z * (des_z_vel - z_vel))

roll_torque = Kp_roll * \
    (((des_x_acc * sin(des_yaw) - des_y_acc * cos(des_yaw)) / g) - roll)
pitch_torque = Kp_pitch * \
    (((des_x_acc * cos(des_yaw) - des_y_acc * sin(des_yaw)) / g) - pitch)
yaw_torque = Kp_yaw * (des_yaw - yaw)

roll_vel += roll_torque * dt / Ixx
pitch_vel += pitch_torque * dt / Iyy
yaw_vel += yaw_torque * dt / Izz

roll += roll_vel * dt
pitch += pitch_vel * dt
yaw += yaw_vel * dt

R = rotation_matrix(roll, pitch, yaw)
acc = (np.matmul(R, np.array(
    [0, 0, thrust.item()]).T) - np.array([0, 0, m * g]).T) / m
x_acc = acc[0]
y_acc = acc[1]
z_acc = acc[2]
x_vel += x_acc * dt
y_vel += y_acc * dt
z_vel += z_acc * dt
x_pos += x_vel * dt
y_pos += y_vel * dt
z_pos += z_vel * dt

q.update_pose(x_pos, y_pos, z_pos, roll, pitch, yaw)
```

```
            t += dt

        t = 0
        i = (i + 1) % 4
        irun += 1
        if irun >= n_run:
            break

    print("Done")


def calculate_position(c, t):
    """
    Calculates a position given a set of quintic coefficients and a time.

    Args
        c: List of coefficients generated by a quintic polynomial
           trajectory generator.
        t: Time at which to calculate the position

    Returns
        Position
    """
    return c[0] * t**5 + c[1] * t**4 + c[2] * t**3 + c[3] * t**2 + c[4] * t + c[5]


def calculate_velocity(c, t):
    """
    Calculates a velocity given a set of quintic coefficients and a time.

    Args
```

```
        c: List of coefficients generated by a quintic polynomial
            trajectory generator.
        t: Time at which to calculate the velocity

    Returns
        Velocity
    """
    return 5 * c[0] * t**4 + 4 * c[1] * t**3 + 3 * c[2] * t**2 + 2 * c[3] * t + c[4]


def calculate_acceleration(c, t):
    """
    Calculates an acceleration given a set of quintic coefficients and a time.

    Args
        c: List of coefficients generated by a quintic polynomial
            trajectory generator.
        t: Time at which to calculate the acceleration

    Returns
        Acceleration
    """
    return 20 * c[0] * t**3 + 12 * c[1] * t**2 + 6 * c[2] * t + 2 * c[3]


def rotation_matrix(roll, pitch, yaw):
    """
    Calculates the ZYX rotation matrix.

    Args
        Roll: Angular position about the x-axis in radians.
```

```
        Pitch: Angular position about the y-axis in radians.
        Yaw: Angular position about the z-axis in radians.

    Returns
        3x3 rotation matrix as NumPy array
    """
    return np.array(
        [[cos(yaw) * cos(pitch), -sin(yaw) * cos(roll) + cos(yaw) * sin(pitch) *
sin(roll), sin(yaw) * sin(roll) + cos(yaw) * sin(pitch) * cos(roll)],
         [sin(yaw) * cos(pitch), cos(yaw) * cos(roll) + sin(yaw) * sin(pitch) *
          sin(roll), -cos(yaw) * sin(roll) + sin(yaw) * sin(pitch) * cos(roll)],
         [-sin(pitch), cos(pitch) * sin(roll), cos(pitch) * cos(yaw)]
         ])


def main():
    """
    Calculates the x, y, z coefficients for the four segments
    of the trajectory
    """
    x_coeffs = [[], [], [], []]
    y_coeffs = [[], [], [], []]
    z_coeffs = [[], [], [], []]
    waypoints = [[-5, -5, 5], [5, -5, 5], [5, 5, 5], [-5, 5, 5]]

    for i in range(4):
        traj = TrajectoryGenerator(waypoints[i], waypoints[(i + 1) % 4], T)
        traj.solve()
        x_coeffs[i] = traj.x_c
        y_coeffs[i] = traj.y_c
        z_coeffs[i] = traj.z_c
```

```
    quad_sim(x_coeffs, y_coeffs, z_coeffs)


if __name__ == "__main__":
    main()
```

Модуль «Реалізація ACO для тестового набору даних».

```
"""

Оптимізований мурашиний алгоритм для БПЛА

"""

import sys
sys.path.append("/Users/pasha/Workspace/AntColonyOptimization/ant-colony-optimization")
import ant_colony
import math
from enum import Enum
import numpy as np

# встановлення тестових віток та локацій
test_nodes = {0: (0, 7), 1: (3, 9), 2: (12, 4), 3: (14, 11), 4: (8, 11), 5: (15, 6), 6: (6, 15), 7: (15, 9), 8: (12, 10), 9: (10, 7)}

# функція визначення дистанції між вітками
def distance(start, end):
    x_distance = abs(start[0] - end[0])
    y_distance = abs(start[1] - end[1])
```

```
    # c = sqrt(a^2 + b^2)
    import math
    return math.sqrt(pow(x_distance, 2) + pow(y_distance, 2))

# створення колоної
colony = ant_colony.ant_colony(test_nodes, distance)

# знаходження оптимального маршруту з ACO
answer = colony.mainloop()

print(answer)
```

ДОДАТОК Б

(обов'язковий)

ПРЕЗЕНТАЦІЯ ВИСТУПУ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Кафедра комп'ютерної інженерії та системного програмування

Ковальчук Павло Сергійович

Метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму

Науковий керівник – д. т. н, проф. Лисенко С.М.

Хмельницький - 2022

Мета і задачі дослідження

- **Метою кваліфікаційної роботи магістра** є розробка програмно-технічного засобу, методу та системи керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму для підвищення ефективності виконання групових завдань.
- **Об'єктом дослідження** є процес керування групою безпілотних літальних апаратів в тому числі процес оптимізації методу керування групи БПЛА з метою досягнення кінцевої точки призначення за найкоротший час.
- **Предметом дослідження** є математичні моделі, удосконалений метод та програмно-технічні засоби для реалізації системи та методу керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму.

Мета і задачі дослідження

- Поставлена мета досягається розв'язанням наступних **задач**:

1. Проведення аналізу та дослідження предметної області та доступних методів рішення задачі.
2. Розробити математичної моделі скоординованого планування оптимальної траєкторії руху БПЛА.

Мета і задачі дослідження

3. Спроектувати удосконалений ACO Max-Min алгоритм для розрахунку оптимального маршруту БПЛА з урахуванням уникнення спливаючих перешкод.
4. Розробити та дослідити програмно-апаратну реалізацію керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму та провести ряд імітаційних тестів.

Наукова новизна отриманих результатів

Наукова новизна отриманих результатів :

- набув подальшого розвитку метод суть роботи якого полягає в тому, щоб представити ієрархічну модель прийняття рішень для спільного планування місій безпілотників для місії розвідки шляхом інтеграції розподілу місії та планування маршруту;

- набула подальшого розвитку інформаційна технологія керування групою безпілотних літальних апаратів, модель реалізації якої буде відбуватись без критично-довгих навчань та з відносно очікуваним результатом

Практичне значення отриманих результатів

- **Практична цінність** проекту полягає в розробленому програмно-технічному продукті, який дозволяє організувати оптимальну систему керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму та виконання місій цими БПЛА у найкоротший термін та з високою ефективністю.

Актуальність дослідження

- Військова сфера
- Доставка
- Рятувальна допомога
- Фотозйомка та кіно
- Космічна сфера

Класифікація БПЛА

Багатороторний

- Два або більше ротори

Одноороторний

- Один ротор

З фіксованим крилом

- Із двигуном та суцільним крилом

Недоліки відомих методів керування скоординованою групою БПЛА

- **В результаті дослідження було виявлено, що відомі методи** (Лінійний квадратичний регулятор (LQR), лінеаризація зворотного зв'язку, адаптивний, надійний, оптимальний та нейронні мережі) мають певні недоліки:

Низька швидкодія.

Довге навчання алгоритму

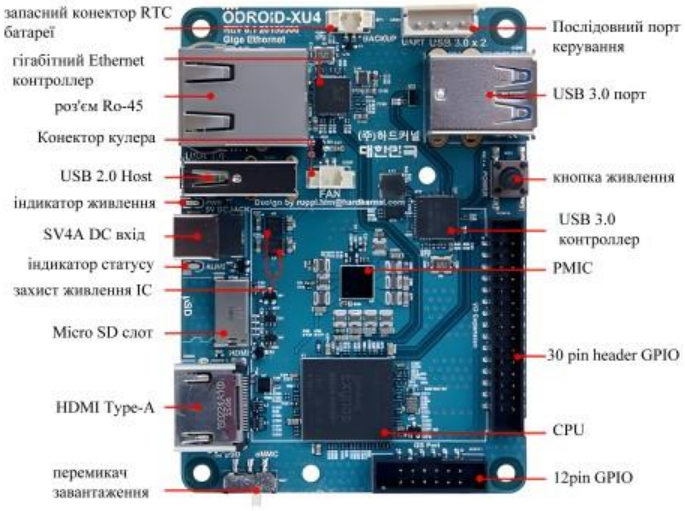
Низька ефективність в реагуванні на спливаючі перешкоди.

Вибір типу БПЛА. Квадротор.

Основні моделі польоту.

- (а) зависаючий рух і вертикальний рух завдяки збалансованим крутним моментам
- (б) різниця в тягі для створення моменту крену / кроку
- (в) різниця в крутному моменті для створення моменту повороту

Odroid-XU4 та каркас БПЛА



МАТЕМАТИЧНА МОДЕЛЬ СКООРДИНОВАНОГО ПЛАНУВАННЯ ОПТИМАЛЬНОЇ ТРАЄКТОРІЇ РУХУ БПЛА. ОНОВЛЕННЯ ФЕРОМОНІВ.

- Мурахи залишають свій феромон на краях під час кожної подорожі, при завершенні однієї ітерації. Сума феромона одного ребра визначається так:

$$\tau_{ij}(t+1) = \Delta\tau_{ij} + (1-\rho)\tau_{ij}(t) \qquad (1)$$

- де $\rho \in (0,1)$, 1-ρ – швидкість збереження попереднього феромона. ρ визначається як швидкість випаровування феромона.
- У MMAS лише найкраща мураха оновлює сліди феромонів і значення феромону привязане до неї.

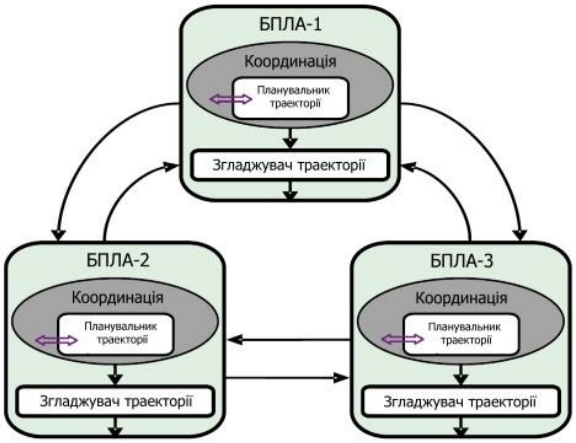
Правило руху мурах.

Мурахи переходять з одного міста (колонії) в інше відповідно до ймовірності. По-перше, міста, до яких здійснюється доступ, повинні бути поміщені в таблицю табу.

Потрібно визначити набір міст, до яких ніколи не звертався k-й мураха як дозволений k. По-друге, потрібно визначити видимий градус η_{ij}, $\eta_{ij} = \frac{1}{d_{ij}}$. Імовірність того, що k-й мураха вибере місто, визначається як:

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{k \in \text{доступні}_k}[\tau_{ik}(t)]^{\alpha}[\eta_{ik}]^{\beta}}, j \in \text{доступні}_k \\ 0, \text{ у інших варіантах} \end{cases} \quad (2)$$

МЕТОД КЕРУВАННЯ ГРУПОЮ БЕЗПІЛОТНИХ ЛІТАЛЬНИХ АПАРАТІВ НА ОСНОВІ НЕЧІТКОГО МУРАШИНОГО АЛГОРИТМУ



Координаційний механізм перепланування траєкторії багаторазових БПЛА. Комунікація.

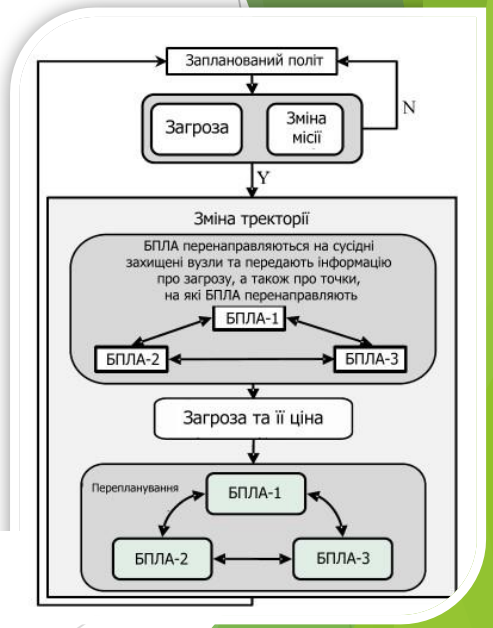
Загрози та їх вартість.

- Що стосується цих меж – це отримана радіолокаційна енергія від радіолокаційних місць, яка становить:

$$J_{ij,threat} = \int_0^{L_{ij}} \sum_{k=2}^{N_t} \frac{1}{((x-x_k)^2+((y-y_k)^2)^2} dl \qquad (3)$$

$$J_{ij,threat} = L_{ij} \sum_{k=1}^{N_t} W_k \left(\frac{1}{d_{0.1,k}^4} + \frac{1}{d_{0.3,k}^4}\frac{1}{d_{0.5,k}^4}\frac{1}{d_{0.7,k}^4}\frac{1}{d_{0.9,k}^4}\right) \qquad (4)$$

Вартість загрози



Проблема повітряного трафіку.

- У кожному вузлі мураха вибирає наступне ребро з імовірністю, що залежить від кількості феромонів, що залишилися на наступному ребрі. Якщо мураха конфліктує з іншою мурахою, вона не відкладає ніяких феромонів. Однак, коли конфлікту немає, він залишає за собою кількість феромонів, рівну:

$$\Delta\tau = \frac{n_a - n_{out}}{n_a} \cdot \frac{\tau_0}{s_{path}} \qquad (5)$$

- де n_{out} – це кількість «втрачених» мурах, τ_0 – початкова кількість феромонів, а s_{path} – оцінка шляху, яким йде мураха. Ця сума враховує кількість мурах, які нарешті знайшли дійсний шлях.

Вирішення задачі посадок БПЛА.

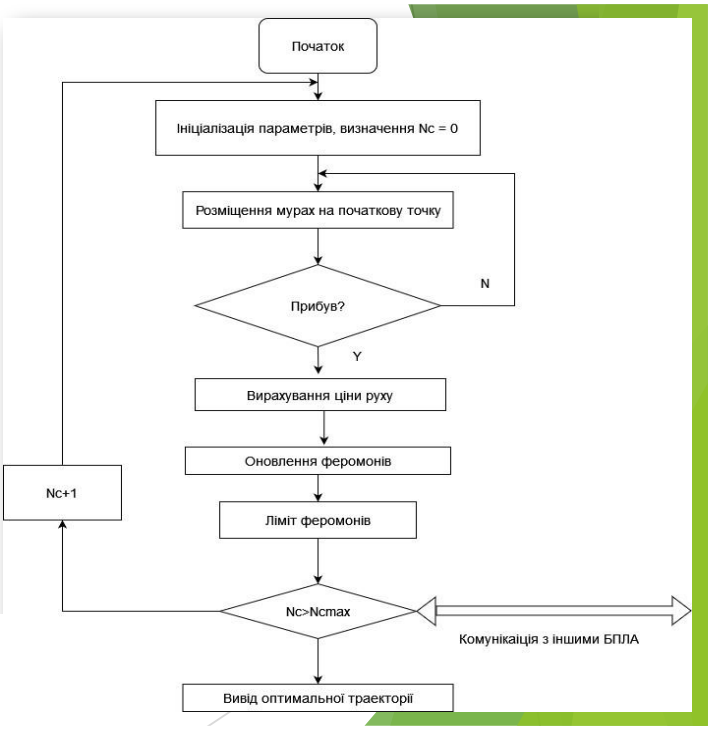
- Перший – це пріоритет БПЛА, такий як час найранішої посадки, цільовий час, штрафна вартість БПЛА. По-друге, вартість, розрахована після того, як ми призначили БПЛА час на посадку за алгоритмом призначення. Вагою цих двох параметрів є «інформаційна евристика» алгоритму мурашиної колонії.

$$\eta_{rj} = \left(\frac{1}{(\text{пріорітет}(j)+1}\right)^{\beta_1} \cdot \left(\frac{1}{(\text{ціна\_ризику}(j)+1}\right)^{\beta_2} \qquad (6)$$

- де пріорітет(j) – пріорітет БПЛА j;
- ціна_ризику(j) – ціна ризику для БПЛА;
- β_1 та β_2 – коефіцієнти зважування.

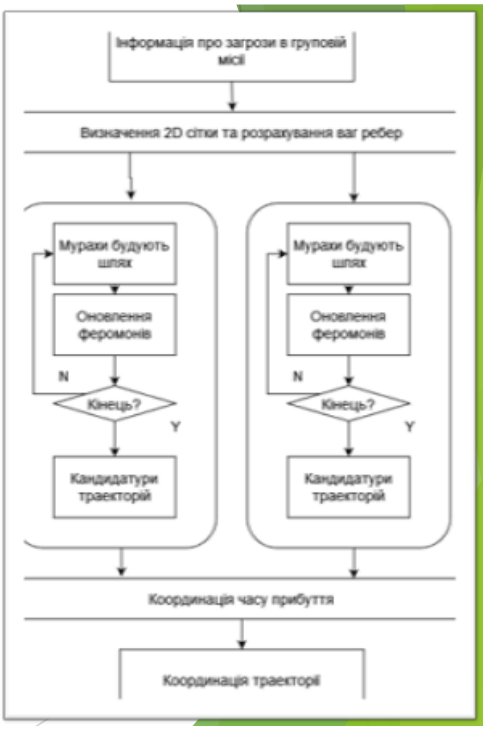
Реалізація методу керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму.

Адаптивний ACO Max-Min, застосований до одного БПЛА з в системі кількох БПЛА



Реалізація методу керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму.

Застосування ACO до проблеми координації кількох БПЛА.



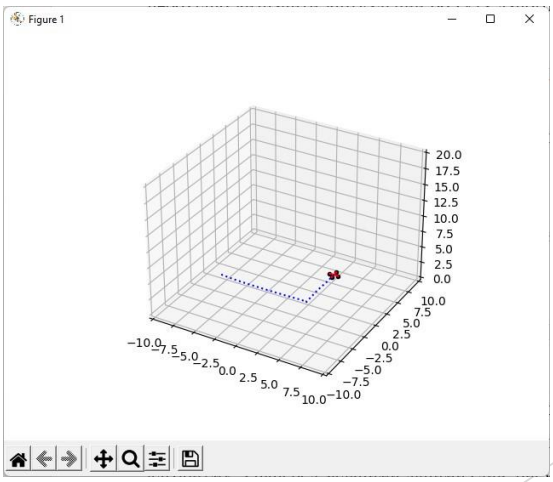
- Важливо звернути увагу на наступні важливі функції:

1. Оновлення карти феромонів.
2. Визначення дистанції.
3. Прив'язка мурах до вибраного шляху.

- Оновлення карти феромонів – self.pheromone_map , відбувається шляхом зменшення значень, що містяться в ньому, за допомогою алгоритму ACO. Додаються значення феромонів від усіх мурах із ant_updated_pheromone_map .

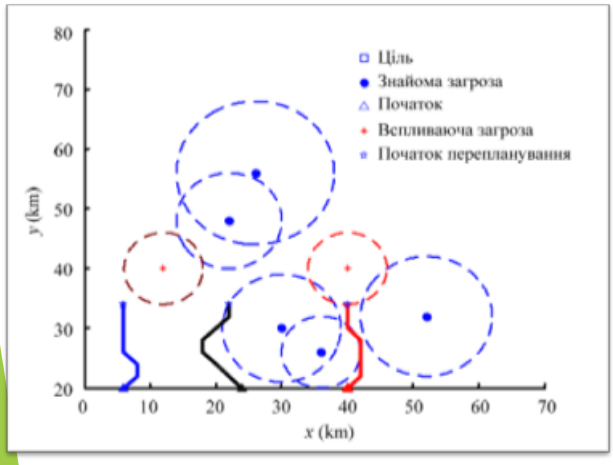
- Визначення дистанції використовує distance_callback для повернення відстані між вузлами, якщо відстань не була обчислена раніше, потім вона заповнюється в distance_matrix і повертається, якщо відстань була викликана раніше, в кінці кінців її значення повертається з distance_matrix.
- Прив'язка мурах відбувається перед оновленням карти феромонів, але після визначення дистанції, зокрема, поточне значення феромонів ділиться на вибрану дистанцію, і з цього формується коофіцієнт феромону. Коефіцієнт феромону new_pheromone_value, використовується пізніше для оновлення карти феромонів.

Графічна симуляція руху БПЛА у трьох вимірному просторі

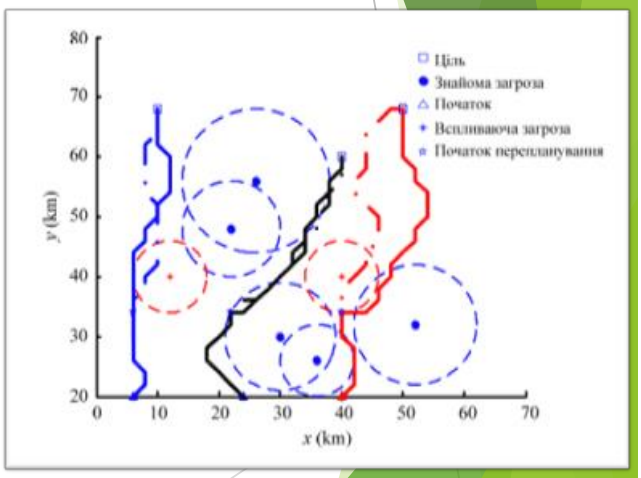


Результати імітаційного тестування.

Спливаючі загрози при руху БПЛА та змінена траєкторія руху БПЛА



Зіткнення із загрозою



Побудова нової траєкторії

Публікації за матеріалами дипломної роботи.

- Ковальчук П.С., Лисенко С.М.. Метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму. *Збірник наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021»*. Хмельницький – 2021. – 117-119с..

Висновки.

- У даній роботі за результатами практичних та теоретичних досліджень було створено та описано метод і систему керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму.
- У першому розділі було досліджено предметну область, описані доступні типи БПЛА, їх схеми контролю та сфери застосування. В тому числі було визначено основні необхідні технічні характеристики БПЛА, для їх оптимальної координаційної роботи при спільних місіях. Також було проаналізовано доступні методи рішення поставленої задачі, коротко описані основні недоліки наявних алгоритмів, та технологій. В результаті аналізу було зроблено висновок, що наявні алгоритми та технології мають багато недоліків у вигляді низької швидкодії, поганої координації, чи проблем у виявленні супутніх загроз. В розділі представлено подальші кроки для вирішення вказаних проблем та сформовано постановку задачі.

Висновки.

- У другому розділі детально описується вибір типу БПЛА, для проведення досліджень, наведені його технічні характеристики, та специфікації роботи. В тому числі була сформована математична модель скоординованого планування оптимальної траєкторії руху БПЛА, яка включає в себе модель та принципи роботи вибраного БПЛА, включно із основними його механізмами, та модель роботи оптимізованого асинхронного алгоритму для корегування БПЛА при виконанні місій.

Висновки.

- У третьому розділі було представлено метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму. Було описано метод скоординованого планування оптимального маршруту БПЛА для завершення місій, описано можливі загрози та способи їх обійти. Метод зображає обмеження умов при координації БПЛА при перепланування їх маршруту та описує схему їх координації за феромонами мурах для вирішення цих проблем. Результатом даних експериментальних досліджень, було отримано покращену швидкодію координації БПЛА на маршруті, вдале реагування на прогнозовані та спливаючі перешкоди із оптимальним варіантом посадки на злітну полосу чи пересічну місцевість.
- У четвертому розділі представлено реалізацію програмно-технічного засобу керування групи безпілотних апаратів методомнечіткого мурашиного алгоритму.

Висновки.

- Реалізація містить у собі модифікований мурашиний алгоритм із паралельно асинхронними колоніями мурах, в якому продемонстрована принципова модель адаптивного Max-Min ACO, для швидкого та безпечного виконання місій групами БПЛА. У розділі представлено блок-схеми Max-Min ACO, де представлено основний алгоритм бій БПЛА при тих чи інших умовах, включаючи комунікацію БПЛА між собою,та реагування на групи феромонів на маршруті.
- В розділі описано обумовлення вибору технологій розробки, зокрема Python 3.10, додаткових програмних бібліотек matplotlib.pyplot та numpy, які використовуються для математичних обчислень та побудови трьох вимірного простору для тестування змодельованого БПЛА. Описано основні функції та їх призначення, які були створенні в процесі виконання роботи, для реалізації поставлених задач. Також було описано використання стороннього SDK та платформи PX4 для суміжної імплементації програми у технічному та системному варіанті на практичній основі.

Висновки.

- В результаті виконання було змодельований тестовий трьох вимірний простір для симуляції руху БПЛА, за допомогою утиліти QGroundControl, була створена можливість імплементації технічної прошивки для обраного БПЛА на основі мікрокомп'ютера Odroid-XU4 та проведено ряд математичних тестів у MATLAB для підтвердження ефективності алгоритму.
- Наукова новизна отриманих результатів полягає у удосконаленому методі представлення ієрархічної модель прийняття рішень для спільного планування місій безпілотників для місії розвідки шляхом інтеграції розподілу місії та планування маршруту. Модель даної реалізації системи буде відбуватись без критично -довгих навчань та з відносно очікуваним результатом.

Висновки.

- Практична цінність проекту полягає в розробленому програмно-технічному продукті, який дозволяє організувати оптимальну систему керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму та виконання місій цими БПЛА у найкоротший термін та з високою ефективністю.

ДОДАТОК В

(обов'язковий)

ПУБЛІКАЦІЯ

Актуальні проблеми комп'ютерних наук

УДК 004.7.056.5

Ковальчук П. С., Лисенко С. М.

Хмельницький національний університет

МЕТОД КЕРУВАННЯ ГРУПОЮ БЕЗПІЛОТНИХ ЛІТАЛЬНИХ АПАРАТІВ НА ОСНОВІ НЕЧІТКОГО МУРАШИНОГО АЛГОРИТМУ

Запропоновано метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму. Суть методу полягає в тому, щоб представити ієрархічну модель прийняття рішень для спільного планування місій безпілотників для спільної місії розвідки шляхом інтеграції розподілу місії та планування маршруту. В дослідженні пропонується метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму. Метод включає в себе три етапи планування місії кількох БПЛА.

A method of controlling a group of unmanned aerial vehicles based on a fuzzy ant algorithm is proposed. The method should be applied to present a hierarchical decision-making model for joint planning of drone missions for a joint conversation mission by integrating mission allocation and route planning. The study proposes a method of controlling a group of unmanned aerial vehicles based on a fuzzy ant algorithm. The method includes three stages of planning the missions of several UAVs.

Безпілотний літальний апарат (БПЛА) – це новий тип бойової платформи з незалежними польотами та можливостями незалежного виконання, які викликали величезний інтерес та безпрецедентну увагу у всьому світі [1-3].

БПЛА демонструють видатну продуктивність та потенційні можливості застосування, включаючи розвідку [4-8], пошук джерел [3], виявлення цілей та багаторазову атаку цілі [9]. БПЛА приділяють все більшу увагу при виконанні спільних завдань, оскільки їхні польоти можна контролювати за допомогою робочої станції на землі.

Дослідження відомих методів керування групою безпілотних літальних апаратів показали низьку ефективність спільного планування місій польотів БПЛА [10-12].

Через обмеження сучасних апаратних та технічних умов, як розумно відправити БПЛА для виконання спільних завдань та вибрати найкращий маршрут розвідки для групи чи класу БПЛА, має велике значення для операцій [12].

У дослідженні пропонується удосконалений метод, суть роботи якого полягає в тому, щоб представити ієрархічну модель прийняття рішень для спільного

планування місій безпілотників для спільної місії розвідки шляхом інтеграції розподілу місії та планування маршруту.

В дослідженні пропонується метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму. Метод включає в себе три етапи планування місії кількох БПЛА.

Модель спільного планування БПЛА може забезпечити ефективну підтримку прийняття рішень щодо розвідки декількох БПЛА, що має велике значення для стратегічної та технологічної еволюції системи багатопілотних БПЛА. Модель ієрархічного прийняття рішень переважно включає триетапне планування.

У плануванні першого етапу здійснюється кластеризація цільової групи. З цією метою використовується алгоритм кластеризації, який об'єднує вихідні декілька об'єктів у нову цільову групу, тоді як вихідна цільова група вважається цільовою підгрупою БПЛА.

У плануванні другого етапу ґрунтується на основі нечіткого мурашиного алгоритму для планування глобального шляху між цільовими підгрупами.

У плануванні третього етапу алгоритм нечіткої колонії мурашок знову використовується для планування локального шляху всередині цільової підгрупи.

З метою підвищення ефективності зв'язку між БПЛА в рамках запропонованого методу передбачається спільна стратегія зв'язку, яка може мінімізувати кількість БПЛА, необхідних для зв'язку.

Таким чином запропоновано керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму, який надає можливість здійснити планування найкоротшого маршруту спільної місії БПЛА, а також обчислюється найкоротша відстань та час польоту.

Перелік посилань

1. L. Ruan et al., Energy-efficient multi-UAV coverage deployment in UAV networks: A game-theoretic framework, in China Communications, vol. 15, no. 10, pp. 194-209, Oct. 2018, doi: 10.1109/CC.2018.8485481.
2. C. Liu, G. Wang, Y. Yan, C. Bao, Y. Sun and Ding Fan, Pint-sized military UAV engine's tele-adjusting arm based on force feedback, 2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2010), 2010, pp. 205-209, doi: 10.1109/CAR.2010.5456566.
3. X. Yang, D. Lin, F. Zhang, T. Song and T. Jiang, High Accuracy Active Stand-off Target Geolocation Using UAV Platform, 2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP), 2019, pp. 1-4, doi: 10.1109/ICSIDP47821.2019.9172919.
4. Y. Kang, B. Park, A. Cho, C. Yoo and S. Koo, Flight test of flight control performance for airplane mode of Smart UAV, 2012 12th International Conference on Control, Automation and Systems, 2012, pp. 1738-1741.
5. P. Shao, C. Wu and S. Ma, Research on key problems in assigned-point recovery of UAV using parachute, 2013 IEEE International Conference of IEEE Region 10 (TENCON 2013), 2013, pp. 1-4, doi: 10.1109/TENCON.2013.6719061.

6. Y. Kang, B. Park, A. Cho, C. Yoo and S. Koo, Flight test of flying qualities for helicopter mode of smart UAV, 2011 11th International Conference on Control, Automation and Systems, 2011, pp. 879-882.
7. S. Wang, Y. Han, J. Chen, Z. Zhang, G. Wang and N. Du, A Deep-Learning-Based Sea Search and Rescue Algorithm by UAV Remote Sensing, 2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC), 2018, pp. 1-5, doi: 10.1109/GNCC42960.2018.9019134.
8. William Kucinski, Distributed Hydrogen-Fueled Propulsion for HALE UAVs, in So You Want to Design Engines: UAV Propulsion Systems , SAE, 2018, pp.29-38.
9. William Kucinski, So You Want to Design Engines: UAV Propulsion Systems, in So You Want to Design Engines: UAV Propulsion Systems , SAE, 2018, pp.i-xii.
10. William Kucinski, "Plasma Propelled UAVs and Dielectric Barrier Discharge, in So You Want to Design Engines: UAV Propulsion Systems , SAE, 2018, pp.47-56.
11. X. Gao, H. Jia, Z. Chen, G. Yuan and S. Yang, UAV security situation awareness method based on semantic analysis, 2020 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS), 2020, pp. 272-276, doi: 10.1109/ICPICS50287.2020.9201954.
12. C. Yoo, A. Cho, B. Park, Y. Kang, S. Shim and I. Lee, Collision avoidance of Smart UAV in multiple intruders, 2012 12th International Conference on Control, Automation and Systems, 2012, pp. 443-447.

UNICHECK

Ім'я користувача:
Кафедра КІ

ID перевірки:
1010993800

Дата перевірки:
29.04.2022 09:08:19 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
29.04.2022 09:08:41 EEST

ID користувача:
100005591

Назва документа: Метод та система керування групою безпілотних літальних апаратів на основі нечіткого му...

Кількість сторінок: 92 Кількість слів: 15275 Кількість символів: 111113 Розмір файлу: 7.18 MB ID файлу: 1010898932

1.01%
Схожість

Найбільша схожість: 0.58% з джерелом з Бібліотеки (ID файлу: 1010883069)

0.45% Джерела з Інтернету	14	Сторінка 94
0.78% Джерела з Бібліотеки	97	Сторінка 94

0.22% Цитат

Цитати	2	Сторінка 95

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	198

Fri Apr 29 08:16:06 EEST 2022, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 0.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 11%**

ID: 103212 Название: Метод та система керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму Добавлено в БД: 2022-04-29 Авторы: Ковальчук П.С. Руководители: Лисенко С.М. Консультанты: Оппоненты:	Документ		Суммарное совпадение по Базе Данных	
	Символы	Лексемы	Символы	Лексемы
	100383	839	363 (0%)	5 (1%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

Дипломник: Ковальчук Павло Сергійович

Тема: Метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг дипломної роботи:

Кількість листів креслень – ; кількість сторінок записки 89

1. Короткий зміст роботи та прийнятих рішень Представлена робота присвячена розробленню методу керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму

2. Висновок про відповідність роботи дипломному завданню Дипломна робота відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі було досліджено предметну область, описані доступні типи БПЛА, їх схеми контролю та сфери застосування, проаналізовані існуючі рішення. У другому розділі була сформована математична модель скоординованого планування оптимальної траєкторії руху БПЛА та модель роботи оптимізованого асинхронного алгоритму для корегування БПЛА при виконанні місій. У третьому розділі було описано метод скоординованого планування оптимального маршруту БПЛА для завершення місій, описано можливі загрози та способи їх обійти. У четвертому розділі представлено реалізацію програмно-технічного засобу керування групи безпілотних апаратів методом нечіткого мурашиного алгоритму.

4. Позитивні сторони роботи: В результаті виконаного наукового дослідження було розроблено програмно-технічний продукт, який дозволяє організувати оптимальну систему керування групою безпілотних літальних апаратів на основі

нечіткого мурашиного алгоритму та виконання місій цими БПЛА у найкоротший термін та з високою ефективністю.

5. Негативні сторони роботи: Оптимізація методу була здійснення для БПЛА типу квадротор, для інших БПЛА потрібні додаткові дослідження

6. Оцінка графічного оформлення та пояснювальної записки роботи: –

7. Відгук про роботу в цілому: В загальному робота виконана на достатньому рівні.

8. Інші зауваження: –

9. Оцінка дипломної роботи:

Розглянувши позитивні та негативні сторони представленої дипломної роботи вважаю, що робота заслуговує оцінки «добре» 4,25 (B)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Джулій В.М. , к.т.н., доцент, кафедри кібербезпеки Хмельницького національного університету

" 27 " квітня 2022р.

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Ковальчук Павло Сергійович
ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-20-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіатоповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

27.04.2022
дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА СИСТЕМНОГО ПРОГРАМУВАННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Метод керування групою безпілотних літальних апаратів на основі нечіткого мурашиного алгоритму

Автор: Ковальчук Павло Сергійович

Спеціальність: 123 – Комп'ютерна інженерія та програмування

Освітня програма: освітньо-наукова

Науковий керівник: Лисенко С.М., д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є незначними, законними і не є плагіатом, оскільки:

1) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 97 джерелами з бібліотек та 14 джерелами з мережі Інтернет (найбільша схожість: 0.58%) ;
2) запозичення розміщене в кваліфікаційній роботі з опублікованих тез в «Актуальні проблеми комп'ютерних наук АПКН-2021»
3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 1.01% і адресується до 10 першоджерел, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ С.М. Лисенко

Гарант ОП _____ О. С. Савенко

Завідувач кафедри КІСП _____ Т. О. Говорущенко