НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи і методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Децентралізована система координації рою автономних дронів для розвідки в умовах бойових дій»

Виконав
студент IV курсу, групи КІ-12
Лисенко Вадим Олегович _____

Керівник:
доцент кафедри ШІ, к.ф.-м.н., доцент,
Тимошенко Юрій Олександрович _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ, к.т.н., доцент,
Комариста Богдана Миколаївна _____

Рецензент:
Доцент кафедри Інформаційної безпеки, к.т.н., доцент
Гальчинський Леонід Юрійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент_____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«15» січня 2025 р.

ЗАВДАННЯ
на дипломну роботу студенту
Лисенко Вадим Олегович

1. Тема роботи «Децентралізована система координації рою автономних дронів для розвідки в умовах бойових дій», керівник роботи Тимошенко Юрій Олександрович, к.т.н., доцент кафедри штучного інтелекту, затверджені наказом по НН ІПСА від «26» травня 2025 р. № 1759-с.

2. Термін подання студентом роботи «09» червня 2025 року.

3. Вихідні дані до роботи: Система керування децентралізованим роєм дронів для знаходження об’єктів у зоні пошуку

4. Зміст роботи існуючі мультиагентні системи, що використовуються для задач пошуку й розвідки; архітектура системи з використанням автономних дронів; тестування системи в умовах часткової втрати агентів; графічні матеріали.

5. Перелік ілюстративного матеріалу: електронна презентація PowerPoint

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, доцент, к. е. н.		

7. Дата видачі завдання «03» лютого 2025 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Збір та аналіз літературних джерел	14.04.2025-25.05.2025	Виконано
2	Огляд мультиагентних систем	26.04.2025-30.04.2025	Виконано
3	Аналіз підходу до пошуку об’єктів	01.05.2025-03.05.2025	Виконано
4	Розробка архітектури системи	04.05.2025-07.05.2025	Виконано
5	Реалізація контролера дрона	08.05.2025-15.05.2025	Виконано
6	Побудова симуляції	16.05.2025-20.05.2025	Виконано
7	Проведення тестування	21.05.2025-28.05.2025	Виконано
8	Оформлення пояснювальної записки	29.05.2025-9.06.2025	Виконано

Студент _____ Вадим ЛИСЕНКО

Керівник дипломної роботи _____ Юрій ТИМОШЕНКО

РЕФЕРАТ

Дипломна робота: 64 с., 5 рис., табл., 15 посилань, додаток.

РОЇ ДРОНІВ, BOIDS, ДЕЦЕНТРАЛІЗОВАНА СИСТЕМА, КООРДИНАЦІЯ, ПОШУК ЦІЛЕЙ, РОЗВІДКА, АВТОНОМІЯ

Об'єкт дослідження – процес пошуку об'єктів у просторі засобами мультиагентної системи.

Предмет дослідження – децентралізовані алгоритми координації автономних агентів під час виконання розвідувальних місій.

Метою роботи є розробка децентралізованої системи координації рою безпілотних літальних апаратів (БПЛА), здатної до ефективного пошуку об'єктів у заданій зоні з урахуванням обмежень на зв'язок, сенсорні можливості та стійкість до втрат агентів.

Результати роботи. У результаті роботи

виконано огляд існуючих мультиагентних систем і практичних реалізацій роїв дронів;

досліджено підходи до реалізації пошуку в мультиагентній системі;

запропоновано архітектуру системи з автономними агентами;

реалізовано логіку поведінки дронів на основі моделі Boids і PID-регуляторів;

проведено тестування в умовах часткової втрати агентів і різної кількості дронів.

Результати підтверджують ефективність децентралізованого підходу, а також демонструють зростання відмовостійкості при збільшенні кількості агентів. Отримана система може слугувати основою для подальших розробок у галузі автономної робототехніки, моніторингу та безпекових технологій.

.

ABSTRACT

Bachelor’s thesis: 64 p., 5 figures, 1 table, 15 references, appendix

Object of study – the process of object search in space using a multi-agent system.

Subject of study – decentralized coordination algorithms for autonomous agents performing reconnaissance missions.

The goal of the work is to develop a decentralized coordination system for a swarm of unmanned aerial vehicles (UAVs), capable of effectively searching for objects within a given area, taking into account communication limitations, sensor constraints, and resistance to agent failures.

The work includes:

an overview of existing multi-agent systems and practical implementations of drone swarms;

an analysis of approaches to object search in MAS environments;

the design of a system architecture based on autonomous drone agents;

implementation of swarm behavior using the Boids model and PID control algorithms;

testing of system performance under partial agent failure and with varying swarm sizes.

The results confirm the effectiveness of the decentralized approach and demonstrate improved fault tolerance with larger numbers of agents. The developed system can serve as a foundation for further research in the fields of autonomous robotics, environmental monitoring, and security applications.

ЗМІСТ

ВСТУП

Сучасні виклики у сфері безпеки, оборони та моніторингу навколишнього середовища потребують оперативного збору розвідувальної інформації в умовах невизначеності, динаміки та обмеженого доступу. Одним із перспективних підходів до розв’язання таких задач є використання мультиагентних систем, зокрема роїв автономних безпілотних літальних апаратів (БПЛА), які здатні до самостійної координації та прийняття рішень без централізованого керування.

Актуальність теми обумовлена потребою в створенні гнучких, масштабованих і відмовостійких систем, які можуть ефективно працювати в умовах часткової втрати зв'язку, виходу з ладу окремих агентів або змін середовища. У цьому контексті рої безпілотних літальних апаратів (БПЛА) виступають як перспективне та багатообіцяюче рішення, здатне радикально змінити підходи до ведення розвідки, спостереження та виявлення об’єктів противника. Ройові системи, натхненні природною поведінкою колективів (птахів, риб, комах), дозволяють реалізувати такі системи з високим рівнем автономності та гнучкої поведінки. Використання групи автономних БПЛА дозволяє швидко охопити великі території, проводити паралельне сканування місцевості з різних ракурсів і оперативно виявляти цілі, залишаючись при цьому малопомітними та важковразливими для противника. Саме це робить технології колективної автономної взаємодії БПЛА — так звані «рої дронів» — надзвичайно актуальними для військової сфери.

Об'єктом дослідження є процес пошуку об’єктів у просторі з використанням мультиагентної системи.

Предметом дослідження є децентралізовані алгоритми координації автономних агентів у складі рою БПЛА під час виконання розвідувальних завдань.

Метою роботи є розробка децентралізованої системи координації рою дронів, здатної здійснювати пошук об’єктів у динамічному середовищі, забезпечуючи адаптивну поведінку та стійкість до часткових втрат агентів.

Для досягнення цієї мети необхідно виконати такі завдання:

- проаналізувати існуючі рішення у сфері мультиагентних систем і роїв БПЛА;
- дослідити підходи до реалізації пошуку в мультиагентних системах;
- розробити архітектуру децентралізованої системи з дронами у якості агентів;
- дослідити можливі середовища симуляції
- реалізовати програмну модель у середовищі симуляції;
- провести тестування поведінки системи в умовах часткової відмови агентів.

Структура дипломної роботи складається з чотирьох основних розділів. У першому розділі наведено огляд існуючих мультиагентних систем, архітектур ройової взаємодії та прикладів реалізації роїв БПЛА. У другому розділі розглянуто підходи до вирішення задач пошуку в MAS, проаналізовано типи стратегій покриття території та моделі координації. У третьому розділі описано архітектуру розробленої системи, алгоритм роботи, та логіку симуляції. Четвертий розділ містить результати тестування у середовищі симуляції та загальні висновки щодо досягнутих результатів.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ МУЛЬТИАГЕНТНИХ СИСТЕМ

1.1 Загальні принципи мультиагентних систем

Мультиагентні системи (Multi-Agent Systems, MAS) є класом розподілених систем, що складаються з множини агентів, які можуть взаємодіяти між собою та з навколишнім середовищем для досягнення індивідуальних або колективних цілей. Агентами у такій системі можуть бути автономні пристрої (наприклад, роботи або дрони), програмні сутності або навіть біологічні організми (в моделюванні). Базовими властивостям окремого агента в MAS є: автономність, тобто здатність діяти незалежно від централізованого контролю. Взаємодія — агенти можуть обмінюватися інформацією або координувати дії для досягнення спільної мети. Адаптивність — здатність змінювати поведінку залежно від змін у середовищі або поведінки інших агентів. Розподілене прийняття рішень — рішення приймаються локально, але з урахуванням загальної ситуації через взаємодію.

Завдявки вищенаведеним властивостям окремих агентів, система набуває декілька ключових переваг. Основною перевагою є стійкість до відмов, оскільки вихід з ладу окремих агентів не призводить до зупинки усієї системи і не зупиняє виконання задачі. Також це забезпечує масштабованість, додавання нових агентів не потребує зміни архітектури. Паралельність дій агентів дозволяє значно швидше дослідження великих територій.

При вирішенні задач пошуку та виявлення об'єктів у просторі мультиагентні системи використовуються як альтернатива або доповнення до централізованих підходів. Вона дозволяє досягати цілей навіть у ситуаціях, коли комунікація обмежена, або відсутня централізована інфраструктура.

1.2 Історія розвитку мультиагентних систем

Історія розвитку мультиагентних систем охоплює декілька ключових етапів, що сформували сучасні підходи до створення ройових систем дронів.

У 1960–1970-х роках, коли починали формуватись основи розподілених обчислень та штучного інтелекту, виникали перші ідеї автономних агентів, які можуть діяти незалежно в межах великої системи. Одним із перших прикладів вважається концепція “Software Agents” — програмних процесів, що діють у мережевому середовищі й приймають рішення на основі локальної інформації.

У 1980-х роках почали активно вивчатися принципи колективної поведінки у природі: мурах, бджіл, птахів, риб. Саме ці дослідження стали підґрунтям для створення моделей ройового інтелекту. Зокрема, у 1986 році був представлений один із перших алгоритмів, натхненних поведінкою мурах — мурашиний алгоритм (в ориг. Ant Colony Optimization або ACO), а у 1995 році — метод рою частинок (в ориг. Particle Swarm Optimization або PSO), заснований на поведінці зграй птахів.

У 1987 році Крейг Рейнольдс запропонував модель "Boids" (від англійського bird-oid object), яка імітувала польоти пташиних зграй за трьома простими правилами: уникнення зіткнень, вирівнювання з сусідами та тяжіння до центру групи. Ця модель стала базовою для численних досліджень ройової поведінки.

У 1990-х роках мультиагентні системи (MAS) оформились як окрема галузь досліджень в рамках штучного інтелекту. У 1995 році було проведено першу міжнародну конференцію з питань автономних агентів і мультиагентних систем (International Conference on Autonomous Agents and Multiagent Systems, AAMAS). З цього часу MAS почали активно

застосовуватись у симуляціях, іграх, економічних моделях, а згодом — і в робототехніці.

У цей період з’являються перші спроби створити фізичні роботизовані мультиагентні системи, зокрема за допомогою простих наземних роботів (наприклад, Khepera або ePuck). Однак через обмеження у вартості та енергоспоживанні застосування у повітряному середовищі ще залишалось обмеженим.

З 2010-х років починається активний розвиток ройових систем безпілотних літальних апаратів (БПЛА) — автономних дронів, здатних спільно вирішувати завдання без централізованого керування. Це стало можливим завдяки мініатюризації апаратів, зростанню обчислювальних потужностей та покращенню енергозбереження. Одними з перших реальних роїв дронів стали “Intel Shooting Star Drones”, ройова система з централізованим керуванням для світлових шоу (2015), та Проєкти DARPA OFFSET – серія досліджень у США для створення роїв дронів, здатних працювати у міських бойових умовах (з 2017 р.).

1.3 Аналіз реалізацій мультиагентних систем для пошуку та ідентифікації цілей

Упродовж останніх десятиліть було запропоновано та реалізовано низку мультиагентних систем, як в академічному середовищі, так і в рамках дослідницьких проєктів державного або комерційного рівня. Розглянемо найбільш відомі приклади систем, у яких агенти спільно виконують задачі пошуку, розвідки або ідентифікації об’єктів у просторі.

1.3.1 Swarmanoid — гетерогенна мультиагентна система

Swarmanoid (2006–2011) – це проєкт, що фінансувався Європейським союзом і ставив на меті дослідження нових підходів до проектування та імплементації самоорганізуючих роботів. Результатом дослідження стала мультиагентна система з різними типами агентів, кожен відповідальний за різні задачі. «Очі» – літаки, або пари дронів, що оглядають місцевість зверху. Основною задачею цього типу агентів є збір карт місцевості та пошук цілі. «Руки» – невеликі роботи, що здатні чіплятися за стелю та підніматись використовуючи тягові мотузки. Основною задачою цих ботів є доставка предметів до точок, недоступних знизу. «Ноги» – колісні роботи. Здатні пересуватись горизонтально по землі. Основними задачами є доставка «Рук» до цілей, вказаних «Очіма», та пересування об’єктів горизонтально. Здатні зчеплятись один з одним для збільшення потужності та з об’єктами для його пересування.

Дана система оперує на рівні задач - агенти отримують декларативні команди, наприклад: «знайти книгу», «справа підійдіть до полиці». Декларативне планування означає, що агенти самі вирішують, як виконати ці команди, без детальної маршрутизації від центра, покладаючись на локальні алгоритми. «Очі» будують мережу для пошуку, «Руки» формують коаліції з «Ногами» для підйому чи маніпуляції, а «Ноги» самотушки розраховують маршрут та формують транспортні ланцюги.

1.3.2 Kilobot swarm — масштабоване ройове керування

Система Kilobot була розроблена дослідниками з Гарвардського університету у рамках досліджень з ройової робототехніки. Метою проєкту було створення масштабованої, недороговартісної та фізично реалізованої платформи, яка б дозволила емпірично досліджувати поведінку рою з великої кількості агентів (від десятків до тисяч) в лабораторних умовах. Перші публікації щодо системи датуються 2011 роком, а повномасштабне використання (зокрема експерименти зі 1024 роботами) реалізовано у 2014 році.

Кожен окремий агент системи Kilobot є мінімалістичним роботом діаметром близько 3 см, який пересувається за допомогою двох вібраційних моторів. Ці мотори дозволяють роботу рухатись по гладкій поверхні, змінювати напрямок та обертатися, хоч і з обмеженою маневровістю. Взаємодія між агентами здійснюється за допомогою інфрачервоного зв’язку на коротких відстанях (до 10 см), який використовується для передавання повідомлень про положення та сигнали координації.

Система Kilobot демонструє фундаментальні принципи децентралізованої координації. Роботи не мають централізованого керування і функціонують за однаковими локальними правилами, які реалізовані у вигляді простих алгоритмів. Кожен агент здатен визначати відносну позицію сусідів, передавати та отримувати сигнали, а також виконувати дії відповідно до вбудованої логіки. Такі принципи дозволяють реалізовувати поведінку рою, у якій складна глобальна динаміка досягається через взаємодію простих агентів.

Оскільки система Kilobot була створена з фокусом на масштабованість та дослідницьку доступність, значна увага приділялась уніфікації комунікації та мінімізації вартості. Усі роботи програмуються одночасно через

інфрачервоний контролер, який також дозволяє здійснювати синхронізацію часу та координувати початок експериментів. Така інфраструктура забезпечує просте розгортання великої кількості агентів одночасно.

Попри свою простоту, система Kilobot стала одним із найвідоміших практичних роїв у світі робототехніки. Її активно використовують у дослідницьких лабораторіях для вивчення проблем координації, самоорганізації, локалізації та формування патернів. Багато експериментів на цій платформі демонструють, як складна поведінка може виникати з простих локальних правил, що робить систему Kilobot надзвичайно цінною для тестування нових алгоритмів керування в мультиагентних системах.

1.3.3. DARPA OFFSET – рої для міських бойових операцій

Програма OFFSET (OFFensive Swarm-Enabled Tactics), започаткована Агентством передових оборонних дослідницьких проєктів США (DARPA) у 2016 році. Основною метою було створення системи, яка б дозволяла керувати сотнями автономних повітряних і наземних дронів для здійснення спостереження, розвідки, блокування противника, виявлення цілей і логістичної підтримки в урбанізованій місцевості.

Особливістю цієї програми стало поєднання фізичних платформ (наземних роботів і безпілотних літальних апаратів) із віртуальними агентами в єдиному операційному середовищі. Завдяки цьому можливо було не лише випробовувати тактики в симуляції, але й оперативно реалізовувати їх у реальних умовах. Архітектура системи передбачала ієрархічно-децентралізований підхід, тобто кожен дрон функціонував як автономний агент, що мав локальну свободу дій, проте координувався вищим рівнем керування, що задавав стратегічні цілі та правила поведінки. Центральна

роль відводилась людино-орієнтованому керуванню – оператор через спеціальний інтерфейс (у тому числі з використанням віртуальної та доповненої реальності) міг задавати задачі, які рій виконував за допомогою попередньо розроблених «тактик».

Для забезпечення масштабованості і гнучкості керування, DARPA створила відкриту платформу, яка дозволяла розробникам, університетам та стартапам інтегрувати нові апаратні модулі, програмні рішення та алгоритми. Протягом кількох років було проведено серію польових випробувань. Важливим аспектом реалізації системи стала побудова масштабованої моделі взаємодії, у якій автономні агенти могли ділитися інформацією про середовище, координувати дії в реальному часі та адаптувати поведінку відповідно до змін ситуації. Також активно впроваджувались методи навігації в умовах слабкого GPS-сигналу та локалізації за допомогою бортових сенсорів.

1.4 Класифікація роїв безпілотних літальних апаратів

1.4.1 Класифікація БПЛА

Автономний безпілотний літальний апарат, або БПЛА – це роботизована повітряна платформа, здатна виконувати польотні завдання без постійного втручання оператора. Для цього такі системи використовують бортові сенсори, обчислювальні ресурси та алгоритми. За аеродинамічною схемою можна виділити розділення безпілотних літальних апаратів на класи, представлені нижче.

БПЛА з фіксованим крилом працюють з тим же принципом, що і літаки, тобто створюють підйомну силу завдяки постійно розгорнутим крилам та вертикальній швидкості. Такий вид БПЛА характеризується

високою швидкістю та енергоефективністю, але потребують розбігу або спеціального запуску. Проте володіють меншою маневреністю і гірше підходять для скоординованої дії у рої.

Мультикоптери, або роторні БПЛА – клас, що включає у себе квадрокоптери, гексакоптери, тощо. Такі літальні апарати здатні злітати вертикально, зависати у місці, забезпечуючи точне позиціонування. Є ідеальними для огляду об’єктів, операцій у міських умовах, розвідки на малих висотах. До того ж такі БПЛА мають низьку вартість та дуже поширені.

Конвертоплани, або гібридні БПЛА поєднують характеристики фіксованого крила та роторної системи. Забезпечують гнучкість у режимах вертикального злету та горизонтального польоту. Використовуються там, де важлива як автономність, так і мобільність, однак мають значно більшу вартість і тому не є найбільш ефективним вибором для проведення операцій в умовах бойових дій.

Варто також зазначити ***орнітоптери***, що імітують політ птахів шляхом махання крилами. Вони менш поширені, мають значно більшу вартість і використовуються в основному з ціллю маскування у природному середовищі. Не підходять для цілей дипломної роботи.

За функціональним призначенням поділяються на розвідувальні, ударні, рятувальні, супроводжувальні.

1.4.2 Класифікація роїв БПЛА

Рої безпілотних літальних апаратів (БПЛА) — це сукупність автономних або напівавтономних дронів, які взаємодіють між собою

відповідно до певних правил з метою виконання спільного завдання. Класифікація роїв може здійснюватися за різними ознаками:

За типом керування – централізовані та децентралізовані. Централізовані рої БПЛА характеризуються керуванням з одного, головного, вузла. У децентралізованих системах кожен безпілотний літальний апарат приймає рішення локально. Варто також зазначити, що існують також системи з ієрархією, тобто команди віддаються від провідного агента до підлеглих.

Найбільний інтерес становлять децентраліовані системи, оскільки вони демонструють високу гнучкість, масштабованість та стійкість до відмов.

1.5 Постановка задачі

Аналіз сучасних мультиагентних систем та ройових підходів до виконання задач пошуку й моніторингу показав, що найбільш перспективними є децентралізовані системи з локальним обміном інформацією між агентами, здатні до самоорганізації та стійкі до часткових відмов.

Зважаючи на обмеженість ресурсів окремих агентів (наприклад, дальність зв'язку, обмежене поле зору, обмеження батареї), необхідно забезпечити:

- розподілений пошук об'єктів без централізованого керування;
- ефективну координацію агентів у просторі;
- динамічну перебудову поведінки у відповідь на події (виявлення цілі, втрата агента тощо);
- мінімізацію часу виявлення об'єкта навіть при частковому виході дронів з ладу.

Таким чином, задачею дослідження є розробка та моделювання децентралізованої системи координації рою автономних дронів, здатної здійснювати ефективний пошук об’єктів у заданій зоні з урахуванням обмежень зв’язку, сенсорики та стійкості до відмов.

Для вирішення поставленої задачі необхідно:

- розробити архітектуру агентів та логіку їх взаємодії;
- реалізувати алгоритм пошуку об’єкта з використанням моделей поведінки;
- протестувати працездатність системи в умовах нормальної роботи та часткових відмов;
- проаналізувати вплив кількості агентів на ефективність пошуку.

Висновки до розділу 1

Проаналізовані приклади демонструють широкий спектр реалізацій мультиагентних систем — від простих лабораторних прототипів до масштабованих роїв для бойових задач. Незважаючи на різні рівні автономності, архітектурні підходи мають спільні риси:

Локальна взаємодія між агентами;

Відсутність єдиного центра керування;

Використання простих поведінкових правил як основи колективної інтелектуальної поведінки.

Усі ці реалізації підтверджують ефективність мультиагентних систем у задачах пошуку, виявлення цілей та адаптивної координації в складних умовах. Ці приклади можуть слугувати основою для подальших розробок, зокрема й у військовому застосуванні.

РОЗДІЛ 2 МАТЕМАТИЧНА МОДЕЛЬ СИСТЕМИ КЕРУВАННЯ ДЕЦЕНТРАЛІЗОВАНИМ РОЄМ ДРОНІВ

2.1 Огляд середовищ для моделювання

Для дослідження ройової поведінки в умовах симуляції використовуються різні інструменти моделювання. Вибір середовища визначається не лише точністю моделі, але й гнучкістю конфігурації, наявністю готових інструментів, підтримкою мов програмування та зручністю відлагодження логіки багатьох агентів. Нижче наведено розширений аналіз деяких ключових платформ.

2.1.1 Середовище симуляції Gazebo

«Gazebo» - це високоточне симуляційне середовище, орієнтоване на моделювання складної фізики, взаємодії з об'єктами, а також тісну інтеграцію з ROS (Robot Operating System). Gazebo підтримує сценарії з десятками агентів, симуляцію колізій, роботу зі складними картами та 3D-середовищами. Основними перевагами gazebo є детальна фізика та точне моделювання аеродинаміки, активна інтеграція з ROS для побудови розподілених систем, підтримка SLAM, комп’ютерного зору, розширена робота з сенсорами та модульна архітектура.

Проте Gazebo має декілька недоліків. По-перше, це висока складність, запуск простого багатодронового середовища потребує налаштування мережі, обміну топіками ROS і суттєвих обчислювальних ресурсів. Це може бути недоцільним недоцільним в рамках даної роботи.

2.1.2 Середовище розробки Unity

«Unity» – це потужне багатоплатформове середовище розробки 3D додатків, яке за останні десятиліття сформувалося з ігрового рушія у повноцінну платформу для візуального моделювання, симуляції та навчання автономних агентів. Завдяки своїй гнучкості, широкому інструментарію та підтримці машинного навчання, Unity все частіше використовують для симуляції робототехнічних систем, зокрема безпілотників.

До основних можливостей Unity для симуляції належить високореалістична 3D симуляція, підтримка мультиагентних сценаріїв із навчанням та можливість моделювання сенсорів – камер, lidar, gps, тощо. Завдяки повноцінному середовищу розробки, Unity дозволяє створити довільно складні сценарії з високим рівнем деталізації.

Основними недоліками є вбудована у рушій модель фізики (PhysX), що є менш точною порівняно з Gazebo чи WeBots, повноцінне моделювання фізики динаміки польоту БПЛА вимагає створення власного фізичного плагіна, або використання сторонніх модулів.

У контексті досліджень автономної поведінки рою БПЛА Unity може бути ефективним вибором, проте для реалізації фізично достовірного керування реальними параметрами дронів, а також у випадках обмеженого часу на реалізацію, інші середовища, як Webots є доцільнішими.

2.1.3 Середовище симуляції WeBots

«Webots» – це безкоштовне середовище моделювання з відкритим кодом, орієнтоване на освітні та дослідницькі задачі. Webots підтримує

велику кількість роботизованих моделей, включаючи мультикоптери. Користувач може реалізувати контроль логіки на Python, C, C++, Java або MATLAB, що робить платформу гнучкою та зручною для швидкого прототипування.

Основною перевагою WeBots є вбудована підтримка мультиагентної взаємодії. Симулятор дозволяє створювати декілька незалежних агентів, кожен з яких має власний контролер, сенсори, засоби комунікації, камери, інерційні сенсори, GPS, гіроскоп та інші пристрої. Це дозволяє моделювання рою БПЛА з повністю автономною поведінкою кожного окремого агента без потреби в централізованому керуванні. Webots також має вбудований фізичний рушій ODE, що забезпечує реалістичну симуляцію динаміки польоту, сили тяги, гравітації, тертя. Окрім цього, середовище підтримує візуалізацію камер, обробку зображень у реальному часі та емуляцію передачі даних між агентами.

WeBots є особливо зручним при роботі з БПЛА завдяки наявності готових моделей дронів з відкритим API для контролю тяги, сенсорних даних та розширення логіки польоту. Для створення рою достатньо підключити комунікаційні пристрої, задати правила локальної поведінки і система буде працювати без потреби в зовнішніх симуляційних фреймворках.

2.2 Біоінспіровані моделі колективної поведінки

Одним з найефективніших підходів до організації поведінки рою є використання біоінспірованих моделей — алгоритмів, що імітують колективну поведінку тварин, птахів, риб тощо. Основними особливостями таких моделей є властивість створювати імітацію складної поведінки у зграї на основі простих правил окремих агентів. Такі моделі і називаються

біоінспірованими, оскільки формувались на основі поведінки видів, окремі особини яких не мають здатності виконувати аналіз ситуації та приймати складні рішення. Проте з великої кількості агентів, кожен з яких виконує простий алгоритм дій створюється складна поведінка зграї

2.2.1 Мурашиний алгоритм

Мурашиний алгоритм (в оригіналі ant colony optimization або ACO) – це метод, натхненний поведінкою мурах під час пошуку найкоротших шляхів до джерела їжі. У рамках алгоритму кожен агент (мураха) формує маршрути до цілі, залишаючи умовний слід (феромон), інтенсивність якого залежить від якості знайденого маршруту. Інші агенти з вищою ймовірністю слідують стежками з більш насиченим феромоном, що створює ефект позитивного зворотного зв'язку. Основна формула ймовірності вибору шляху $i \to j$:

$$P_{ij} = \frac{{\tau_{ij}}^{\alpha} \cdot {\mu_{ij}}^{\beta}}{\sum_{k=0}^{N} {\tau_{ik}}^{\alpha} \cdot {\mu_{ik}}^{\beta}},$$

де τ_{ij} – рівень феромону на шляху (i,j), μ_{ij} – еврестична цінність, α, β - вагові коефіцієнти важливості феромону та евристики.

Мурашиний алгоритм добре підходить для задач оптимального охоплення територій і планування маршрутів в умовах з відомою топологією. Однак застосування в реальному часі у динамічному середовищі вимагає додаткових адаптацій.

2.2.2 Метод рою частинок

Метод рою частинок (в оригіналі Particle swarm optimization або PSO) – алгоритм, заснований на колективному навчанні частинок (агентів), які переміщуються в просторі рішень, враховуючи найкращі позиції, знайдені як самою частинкою, так і її сусідами.

Оновлення швидкості та положення відбувається за формулами:

$$V_i(t+1) = w \cdot V_i(t) + c_1 \cdot r_1 \cdot \left(p_{best_i} - x_i(t)\right) + \left(g_{best_i} - x_i(t)\right),$$

де $V_i(t)$ – швидкість частинки i у момент часу t, $x_i(t)$ – позиція частинки i у момент часу t, p_{best_i} – найкраща особиста позиція, g_{best_i} – найкраща глобальна позиція у всьому рої, w – інерційна складова, c_1, c_2 – коофіцієнти залучення, r_1, r_2 – випадкові числа в [0, 1].

У контексті дронів PSO потенційно застосовується для знаходження оптимальних позицій сканування або формування. Проте потреба в частій синхронізації та глобальній інформації може обмежити його ефективність у роях із обмеженим зв’язком.

2.2.3 Модель штучної зграї

Модель штучної зграї, також відома як Boids модель (від bird-oid object) – це модель, що базується на поведінці зграї птахів. Boids базується на трьох простих правилах, яких кожен агент намагається притримуватись на основі обмеженої інформації про систему, а саме – на позиціях та швидкостях сусідніх агентів. Ці правила включають у себе: вирівнювання,

згуртованість і відштовхування. Вирівнювання передбачає узгодження швидкості та напряму руху з сусідами, згуртованість – рух до центру мас локальної групи, а відштовхування – уникнення зіткнень з сусідами. Формально, положення агента оновлюється згідно остаточно сформованому векторі, що являє собою суму вищевказаних векторів:

$$\vec{V}_i(t+1) = w_1 \cdot A_i + w_2 \cdot C_i + w_3 \cdot S_i,$$

де A_i – вектор вирівнювання (alignment), C_i – вектор згуртованості (cohesion), S_i – вектор відштовхування (separation), w_1, w_2, w_3 – вагові коофіцієнти.

Кожен вектор формується на основі позицій та швидкостей найближчих сусідей за формулами:

Вектор вирівнювання:

$$\vec{A}_i = \frac{1}{N} * \sum_{j=0}^{N} V_j.$$

Вектор згуртовування:

$$\vec{C}_i = \sum_{j=0}^{N} p_j.$$

Вектор уникнення:

$$\vec{S}_i = -\sum_{j=0}^{N} \frac{p_j - p_i}{\left\| p_j - p_i \right\|^2}.$$

де V_j – швидкість сусіда j, p_j – позиція сусіднього агента,

У контексті дронів така модель реалізується через періодичне оновлення цільових векторів руху, які задаються на основі локально доступної інформації (наприклад, з GPS або віртуального сенсора відстані до сусідів). Врахування обмежень інерційності та фізичних характеристик дозволяє адаптувати модель до реального застосування.

2.3 Алгоритми пошуку цілі у визначеній області

При досягненні цільової точки, виникає задача пошуку цілі кількома агентами у визначеній області. Існує декілька основних підходів і алгоритмів для вирішення цієї задачі.

2.3.1 Стохастичні (випадкові) алгоритми

Стохастичні алгоритми є простими, гнучкими і широко використовуються в умовах обмеженої інформації або невизначеності. Їх основна ідея полягає в тому, щоб замість детермінованих маршрутів дозволити дронам досліджувати територію випадковим або напіввипадковим чином.

Основними методами є «Випадкова прогулянка» (або «Random walk»), Польоти леві та метод Монте-Карло.

Метод випадкової прогулянки полягає у тому, що кожен агент обирає певний напрямок і рухається в ньому певний час, або до зіткнення з межами області. Після цього напрямок змінюється випадковим чином. Перевагою такого підходу є простота реалізації та обчислень, проте такий алгоритм має

низьку ефективність пошуку і не гарантує ефективне використання кількох агентів.

Польоти Леві – це метод, натхнений природньою поведінкою тварин при пошуку їжі. Він поєднує короткі рухи з рідкісними довгими "стрибками", що підвищує ймовірність швидкого знаходження розріджених цілей.

Метод Монте-Карло: використовується для оцінки імовірності знаходження цілі у певних ділянках, що дозволяє адаптувати випадковість на основі частково накопиченої інформації.

Перевагами стохастичного підходу є мінімальні вимоги до обчислювальних ресурсів та комунікації між агентами. До того ж такі алгоритми непогано працюють в невідомих або динамічних середовищах. Проте стохастичні алгоритми не гарантують повне покриття області та мають значну дисперсію часу до знаходження цілі.

2.3.2 Багатоагентне навчання з підкріпленням

Багатоагентне навчання з підкріпленням – область машинного навчання, де декілька агентів навчаються приймати рішення шляхом взаємодії з середовищем і один з одним. Кожен агент отримує винагороди на основі успіхів та адаптує свою стратегію. Існує безліч підходів такої реалізації пошуку, проте кожен має спільні переваги і недоліки. Такий підхід гарантує адаптивну поведінку, та можливість кооперації агентів без ручного програмування стратегій. Проте такий підхід потребує значних обчислювальних ресурсів.

2.3.3 Лінійне сканування

Лінійне сканування – це простий детермінований алгоритм, що полягає у покритті зони пошуку шляхом прямолінійного проходу з фіксованим інтервалом. Такий алгоритм є ефективним у відкритих рівнинних зонах, де немає складного рельєфу або перешкод. Математично, площина пошуку розбивається на паралельні траєкторії довжиною L з міжряддям D. Кожен агент відповідає за свою смугу, рухаючись по маршруту

$$x(t) = x_0 + V_x, \qquad y(t) = y_0 + k \cdot D,$$

де x_0 – початкова координата, V_x – швидкість руху уздовж траєкторії, k – номер відповідного агента.

Обчислення початкових координат та визначення номеру агента залежить від реалізації. У випадку роботи з роєм дронів, для формування початкової координати достатньо використовувати координати приблизної позиції об’єкта та розмір зони пошуку, що може бути передана дрону в залежності від ситуації. Для визначення номеру агента буде доречно використовувати початкові координати агента. Це забезпечить стійкість системи до втрати агенту, а також прибере потребу в пересуванні до початкової точки, що є неоптимальною для конкретного дрона. Для визначення значення D буде доречним використати зону покриття відповідного агента, що може бути визначена в залежності від висоти польоту дрону та кута огляду камери.

2.3.4 Спіральний пошук

Спіральний пошук – метод, що полягає в дослідженні локального регіону навколо ймовірної позиції об'єкта спірально, тобто по колу з поступовим збільшенням радіусу. Кожен агент описує спіраль Архімеда, або логарифмічну траєкторію. Тобто траєкторія обчислюється за формулою:

$$r(\theta) = a + b\theta,$$

де a – початковий радіус, b – швидкість розширення, θ – кут в полярній системі координат.

Для реалізації такого підходу в контексті мультиагентної системи кожному агенту може бути надано різний початковий радіус та/або різна швидкість розширення. Цей метод дозволяє знаходження об’єкту без передачі точності наданих цільових координат відносно справжнього положення об’єкту, проте може призвести до втрати зв’язку між агентами у результаті розльоту на надто велику відстань. До того ж зменшує швидкість переформування після знаходження об’єкту.

Висновки до розділу 2

У межах цієї дипломної роботи було обрано середовище симуляції WeBots через свою гнучкість, простоту конфігурації, підтримку мультиагентного керування та наявність вбудованих моделей дронів. У порівнянні з іншими сердовищами, WeBots має нижчій поріг входження, краще підходить для швидкої розробки алгоритмів керування.

В якості базовою логіки для переміщення рою дронів до цільової точки було обрано модель boids. Ця модель є несприйнятливою до втрати агентів, простою в реалізації та ефективною для базового узгодження траекторії переміщення рою.

В якості алгоритму пошуку було обрано координоване лінійне сканування, адаптоване до ройової структури. Кожен дрон відповідає за певну зону в межах прямолінійного шаблону. Такий підхід дозволяє зменшити ймовірність дублювання маршрутів, забезпечити стабільну реакцію навіть за втрати окремих агентів та реалізувати просту логіку на кожному дроні без потреби в централізованій карті. Таким чином, обраний алгоритм дозволяє зберігати баланс між ефективністю, стійкістю до відмов та простотою реалізації.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ СИСТЕМИ КЕРУВАННЯ РОЄМ РОЗВІДУВАЛЬНИХ ДРОНІВ

3.1 Загальна структура рою

Система моделювання рою дронів, реалізована в рамках цієї роботи, побудована за децентралізованою архітектурою. Кожен дрон функціонує як автономний агент, що здатний самостійно приймати рішення на основі сенсорних даних і обміну інформацією з сусідніми агентами. Рій не має центрального керуючого елемента: поведінка виникає з локальних взаємодій між агентами.

Загальна структура рою містить:

- агенти-дрони, які мають локальну модель поведінки та здатність змінювати фазу місії;
- середовище симуляції, що надає фізичну модель, сенсорні дані та канали зв’язку;
- схему взаємодії, яка базується на обміні повідомленнями між агентами через радіомодуль (в симуляції – emitter/receiver)

Кожен агент обробляє вхідні дані у два потоки.

Сенсорна інформація, тобто координати, отримані з GPS, орієнтація, отримана з IMU, зображення з камери;

Комунікаційна інформація – повідомлення від сусідів, що містять короткий опис інформації та координати.

Реакція на ці дані відбувається в межах певної фази місії, яка визначає логіку дій. Усі дрони перебувають у спільному середовищі, але приймають рішення локально, без глобальної синхронізації, що дозволяє системі бути стійкою до відмов.

При збільшенні кількості дронів змінюється лише кількість міжагентних взаємодій, але не потребується перебудова логіки чи централізованого обчислювального ядра.

3.2 Архітектура окремого дрона

Кожен дрон у симуляції є самостійним програмним агентом, що виконує повний цикл обробки даних, ухвалення рішень та руху. Його архітектура поділяється на кілька логічних компонентів:

Блок сенсорики отримує дані з віртуального GPS (координати), IMU (кут орієнтації), та камери;

Контролер фазової поведінки визначає поточну фазу місії (travel, search, regroup) і задає відповідну поведінку. Використовується модель скінченного автомата (FSM), яка реагує на дані з сенсорів та повідомлення;

Алгоритм руху відповідає за обчислення цільових векторів швидкості відповідно до поточної фази. У фазі «travel» дрон летить до зони пошуку, одночасно притримуючись boids логіки, у фазі «search» - виконує траєкторію огляду, а у фазі «regroup» -прямує до визначеної цілі;

Модуль обміну повідомленнями прослуховує вхідні повідомлення від інших агентів через recevier пристрій, обробляє їх вміст і виконує відповідні дії (зміна фази, зміна цільової точки, тощо). Також надсилає сигнали про виявлення цілі або власні координати найближчим сусідам через emitter пристрій.

Архітектура кожного дрона модульна, що дозволяє в майбутньому замінити окремі компоненти – наприклад, додати нові фази, інші алгоритми руху чи канали зв’язку, або змінити вже наявні без повного переписування

коду. Це робить систему адаптивною до нових сценаріїв і сприяє повторному використанню логіки в інших проєктах.

3.3 Модель фазової поведінки

Однією з ключових особливостей реалізованої системи є фазова структура поведінки дронів. Це дозволяє задати гнучку, ситуаційно залежну модель дій, яка змінюється в залежності від стану місії та отриманої інформації. Поведінка кожного дрона керується скінченним автоматом станів (Finite State Machine — FSM), де кожна фаза відповідає окремій логіці руху й обробки подій. У системі реалізовано три основні фази:

Фаза руху - дрон рухається до визначеної зони пошуку. У цій фазі застосовується базовий навігаційни алгоритм, що обчислює вектор до цільових координат. Усі дрони стартують одночасно та координовано переміщуються до області визначення координат;

Фаза пошуку - основна операційна фаза. Кожен дрон виконує траєкторію пошуку у вигляді прямолінійних проходів, зберігаючи орієнтацію, висоту та швидкість. Дрони не потребують глобальної карти, натомість координують свої дії через обмін повідомленнями, щоб уникати перетинів маршрутів;

Фаза перегрупування - активується, коли хоча б один агент виявив ціль (на основі камери). Усі дрони отримують координати об’єкта через повідомлення та спрямовують рух до визначеного центра з невеликим зсувом для створення «оточення» або збору даних з різних ракурсів.

Перехід між фазами відбувається або на основі внутрішньої події, як знаходження об’єкту на зображенні з камери чи досягнення координат або в

результаті отриманого сигналу від іншого дрона, наприклад повідомлення про виявлення цілі.

Модель FSM забезпечує високу адаптивність та простоту реалізації і дозволяє додавання нових фаз місії у майбутньому.

Таблиця 3.1 – Опис фаз місії

Фаза	*Умови активації*	*Дії агента*	*Комунікація*
Рух	Отримано координати цілі	Рух до зони пошуку, згуртована поведінка	Надсилання поточних координат
Пошук	Агент досяг зони пошуку	Активне сканування, розсередження	Надсилання поточних координат та координат об’єкту у випадку його знаходження
Перегрупування	Виявлено ціль або отримано повідомлення про виявлення цілі	Збір у точці події	Широкомовне повідомлення

3.4 Алгоритм обміну повідомленнями

У системі рою, реалізованій у даній роботі, дрони здійснюють міжагентну комунікацію через змодельований радіомодуль. В умовах симулятора Webots це реалізовано за допомогою пристроїв Emitter і Receiver, що дозволяють надсилати й отримувати повідомлення у вигляді рядків або масивів байтів. Комунікація має децентралізований характер, без загального каналу чи координаційного вузла. Кожен дрон періодично передає повідомлення з актуальним станом: необхідність переходу у наступну фазу

та координати (поточні координати дрона, або координати цілі). Типове повідомлення має формат <message>|<X>|<Y>|<Z>.

Кожен агент прослуховує ефір у радіусі своєї дії (визначений заздалегідь та може бути змінений у рамках симуляції) і зберігає релевантну інформацію, наприклад найперше отримане повідомлення про виявлення цілі. Для уникнення надмірного обміну повідомлення надсилаються лише з певною періодичністю.

Алгоритм обробки повідомлень полягає у наступному: отримати нове повідомлення через receiver; розпарсити дані, валідувати формат.

Якщо вказано зміну фази і поточна фаза не відповідає отриманій – змінити відповідну фазу, зберегти координати.

Такий підхід дозволяє забезпечити узгодженість поведінки без необхідності в централізованому диспетчері. Крім того, відсутність жорсткої синхронізації підвищує стійкість системи до втрат окремих повідомлень або перешкод у середовищі.

3.5 Реалізація у середовищі симуляції

Програмне забезпечення симуляційної системи побудовано у вигляді контролера агента, реалізованого мовою Python з використанням API симулятора Webots. Архітектура програмного рішення орієнтована на забезпечення автономної поведінки кожного дрона з урахуванням сенсорних даних, обробки координат, реалізації фаз місії, локальної координації рою та виявлення цілі.

Контролер дрона являє собою клас Mavic, що наслідує базовий клас Robot. Основними підсистемами даного контролера є:

Підсистема сенсорики - отримання даних з GPS, IMU, гіроскопа, камери. Ці дані використовуються для визначення положення, орієнтації та зображення оточення в реальному часі;

Підсистема керування польотом – реалізована на основі PD-регуляторів (simple_pid.PID). Включає окремі регулятори для координат X, Y, Z (висота) та yaw (обертання);

Комунікаційний модуль – забезпечує обмін повідомленнями між агентами через модулі emitter/receiver, використовуючи серіалізацію даних у форматі JSON;

Модуль поведінки Boids – відповідає за обчислення напрямку руху з урахуванням найближчих агентів у рої. Ураховуються три основні вектори: згуртування, відштовхування та вирівнювання;

Логіка переходу між фазами місії – вбудована FSM (finite state machine), яка оперує фазами travel, search, regroup.

Особливу увагу приділено модульності реалізації. Завдяки цьому можливо ізольовано змінювати параметри (наприклад, вагові коефіцієнти Boids або PID) та адаптувати поведінку системи без повної перебудови логіки. Ще однією перевагою запропонованої структури є її розширюваність: до системи легко додати нові стани місії, нові типи сенсорів (наприклад, LIDAR), або інтегрувати глибші моделі ухвалення рішень (reinforcement learning, нейронні мережі). Таким чином, архітектура програмного забезпечення дрона орієнтована на декомпозицію, адаптивність та автономність, що дозволяє її використання як у навчальних цілях, так і як базу для створення більш складних роботизованих систем.

3.6 Алгоритм симуляції місії та порядок виконання фаз

Симуляція місії відбувається у реальному часі, у вигляді циклічного виконання кроків симуляції (step) у тілі основної функції run(). Під час кожного кроку виконуються такі ключові дії:

Зчитування сенсорних даних: контролер дрона оновлює поточні координати, орієнтацію та висоту на основі GPS, IMU та гіроскопа. Отримані дані зберігаються у структурі self.current_pos;

Передача координат іншим агентам: через визначені інтервали часу (SEND_INTERVAL) дрон надсилає своє положення по каналу emitter, у рамках тестування було використано інтервал в одну секунду. Інші дрони отримують ці повідомлення та зберігають координати сусідів, що необхідно для Boids-логіки;

Обробка вхідних повідомлень: при отриманні даних з receiver виконується оновлення фази місії, координат цілі або вектора згуртування. Якщо дрон отримує повідомлення про знайдену ціль, він переходить до фази regroup;

Перехід між фазами: із travel до search: відбувається при досягненні зони пошуку. Контролер перевіряє відстань до цілі, і при перевищенні порогу, який рівний розміру зони пошуку (фактично – точності визначення координат об’єкту) з додаванням певної відстані, яка необхідна, щоб дрони не повертались назад для прийняття стартових позіцій пошуку, активується логіка пошуку. Із search до regroup: активується у випадку, якщо дрон виявляє об’єкт за допомогою обробки зображення. Для цього використовується функція CheckForObject(), яка перевіряє наявність об’єкту в камері;

Обчислення сигналів керування: після визначення поточної фази та бажаної позиції, виконуються розрахунки відхилень по координатах та кутах.

PID-регулятори формують управляючі впливи для досягнення стабільності, компенсації відхилення та підтримки курсу.

Використання Boids-логіки: при русі в рої враховуються положення інших дронів. Алгоритм формує вектор руху, що є результатом комбінації векторів відштовхування, згуртування та вирівнювання. Цей вектор коригує цільову траєкторію;

Реакція на аварійні ситуації: Дрон вимикається, якщо перевищено допустимі кути крену або тангажу, що може свідчити про перевертання дрона або нестабільність. Це зроблено в першу чергу для того, щоб втрачений дрон, що ще здатний відправляти сигнали про свою позицію, не впливав на розрахунок Boids векторів та не пересувався по непередбаченій траєкторії у випадку перевороту.

Таким чином, симуляція являє собою дискретну, подієво-орієнтовану систему, що переходить між станами у відповідь на внутрішні та зовнішні стимули. Основу функціонування складає децентралізована поведінкова модель, у якій агенти самостійно реагують на події, синхронізуються з іншими агентами та адаптуються до змін у середовищі.

Висновки до розділу 3

У цьому розділі було реалізовано децентралізовану систему керування роєм автономних дронів для виконання розвідувальних місій[1]. Сформована архітектура передбачає відсутність централізованого керування: кожен агент функціонує автономно, реагуючи на сенсорні дані та повідомлення сусідів.

[1]Тут і нижче використані такі інструменти штучного інтелекту як чат-бот з генеративним штучним інтелектом ChatGPT, Copilot виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

Було впроваджено модель фазової поведінки на основі скінченного автомата, що забезпечує гнучке перемикання між етапами місії — рухом, пошуком та перегрупуванням. Комунікація реалізована через симульовані радіомодулі, що дозволяє координувати дії без глобальної синхронізації. Реалізовано програмну модель на Python у середовищі Webots з використанням PID-регуляторів для стабілізації польоту та моделі Boids для підтримання ройової поведінки. Такий підхід забезпечує стійкість до відмов окремих агентів і високий ступінь адаптивності системи, з можливістю масштабування й модифікації без перебудови базової логіки.

РОЗДІЛ 4 ТЕСТУВАННЯ СИСТЕМИ В УМОВАХ СИМУЛЯЦІЇ

4.1 Взаємодія з наглядачем (supervisor)

Для моделювання умов розвідки в динамічному середовищі в симуляційній системі було реалізовано спеціального агента – наглядач (supervisor). Його роль полягає у періодичному створенні об’єктів-цілей у випадкових координатах, імітуючи появу нових джерел розвідувального інтересу на території. Така поведінка дозволяє перевірити здатність рою реагувати на змінні події без заздалегідь відомих сценаріїв.

Supervisor розміщує червоний об’єкт у межах визначеного прямокутного простору. Колір обрано для простоти ідентифікації об’єкту з камери дронів за допомогою RGB-зображення, отриманого з камери, що вмонтована в кожного з дронів. Для спрощення розпізнавання використовується фільтр на основі порогових значень інтенсивності червоного каналу, що дозволяє уникнути складної обробки зображень в реальному часі.

Після розміщення об’єкта supervisor передає приблизні координати цілі всім агентам рою через вбудований комунікаційний канал (emitter/receiver Webots API), що імітує централізовану розвіддану або сигнал тривоги. Отримавши координати, дрони переходять у фазу travel та прямують у бік заданої області, використовуючи логіку координації на основі моделі Boids.

Важливо, що supervisor не бере участі у безпосередньому керуванні польотом дронів — його завданням є лише стимулювання поведінки рою, що відповідає реальним умовам, коли розвідувальні дані можуть надходити ззовні (від супутника, наземного радара або іншого розвідника). Таким чином контролер формує сценарії з динамічно змінною ситуацією та дозволяє замірювати час пошуку цілей роєм, що використовується як метрика оцінки рою.

4.2 Тестування роботи системи в нормальних умовах

Для оцінки ефективності системи було проведено серію симуляцій з різною кількістю дронів у рої. Було проведено експерименти для кожної фази місії. Для кожної симуляції було виконано наступні умови:

Кожен дрон починає зі стартової точки, при цьому стартова точка кожного дрону знаходиться на однаковій відстані від інших стартових точок;

Цільові точки операції формуються випадковим чином за допомогою supervisor.

При тестуванні фази «travel» основним критерієм оцінки є збереження формування в процесі польоту та при досягнення області пошуку. На рисунках 3.1 та 3.2 можна побачити початкові та кінцеві позиції дронів під час тестування travel фази

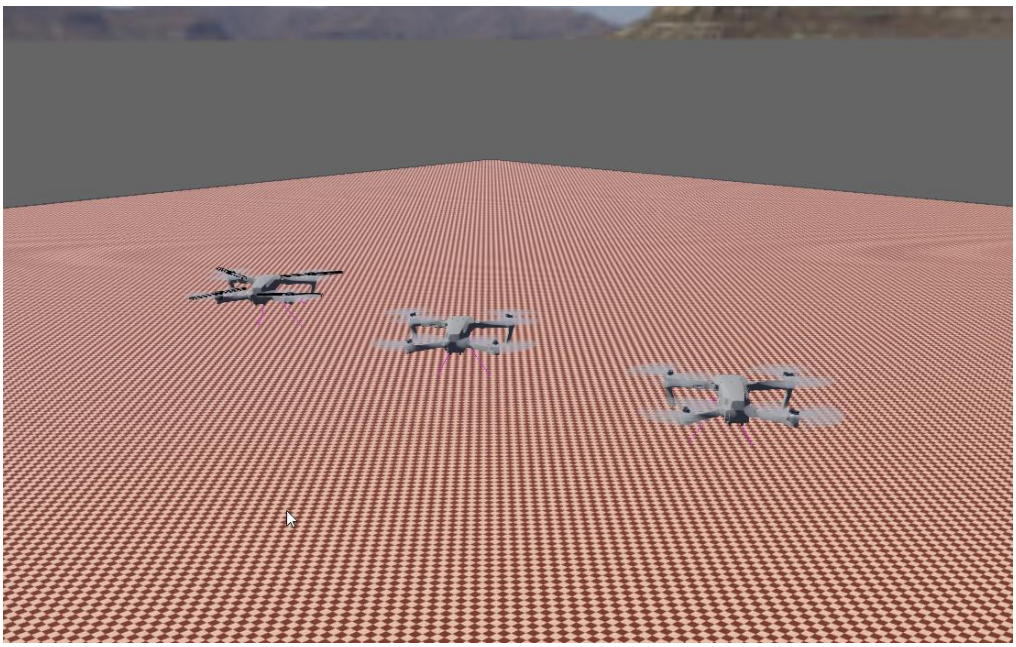

Рисунок 3.1 – Формація рою трьох дронів на початку тестування travel фази

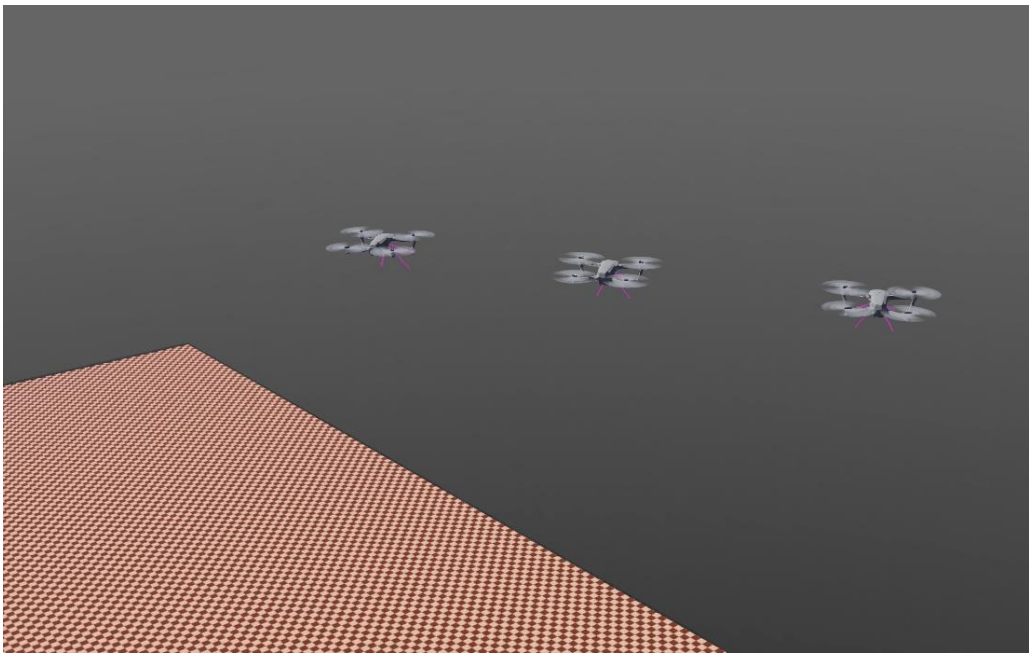

Рисунок 3.2 – Формація рою трьох дронів в кінці тестування travel фази

Під час пересування до зони пошуку дрони під час деяких симуляцій коливаються відносно один-одного, проте не наближуються ближче ніж на третину від параметру «safe_distance», який визначає дистанцію, за межами якої вектор відштовхування не враховується. На рисунку 3.2 можна побачити ідеальну ситуацію, коли дрони вирівнялись в одну лінію, перпендикулярну до напрямку руху. В реальності формація стабілізується після проходження відстані в приблизно 30 метрів, до цього є незначні коливання вздовж напрямку руху. Результати при подорожі до цільової точки з початкової не мають значних відмін від результатів при подорожі від першої цільової точки до другої (тобто з випадковою початковою формацією).

Для тестування фази «search» розроблено алгоритм, який полягає у тому, що supervisor генерує цільовий об’єкт на випадкових координатах і передає його приблизну позицію (погрішність менше за вказану зону пошуку). Після цього заміряв час від початку пошуку (ініціювання фази «search» дронами) до його закінчення (передачі координат об’єкта іншим

дронам). Також для тестування цієї фази було сформовано додаткові умови: Площа пошуку для фази «search» - прямокутник фіксованого розміру;

Кількість дронів варіюється від 1 до 10;

Для кожної симуляції проводиться 10 симуляцій із подальшим усередненням. Графік результатів зображено на рисунку 3.3

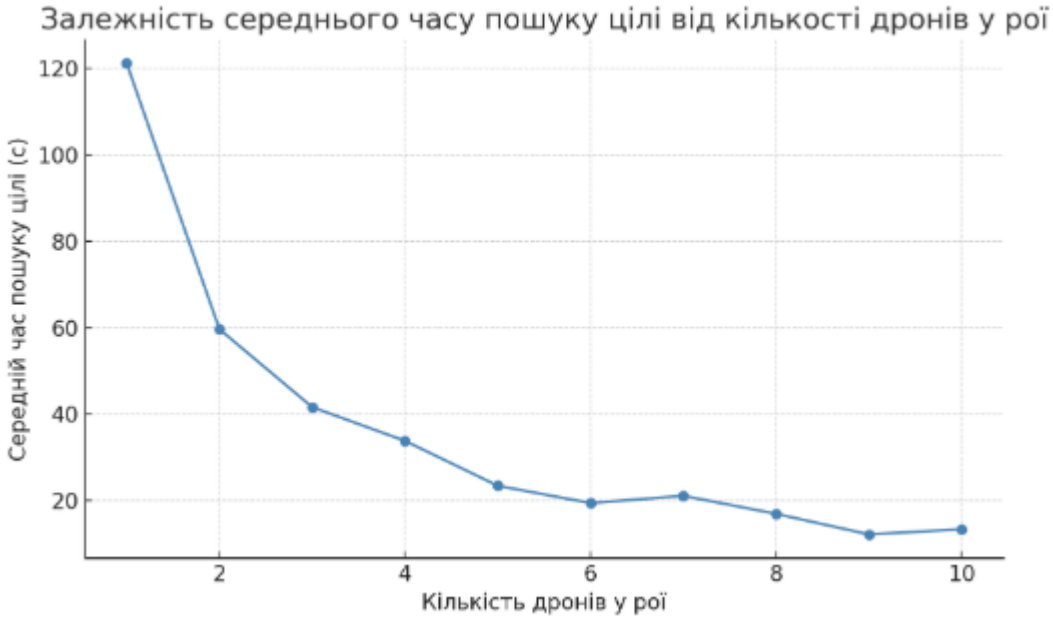

Рисунок 3.3 – Графік залежності середнього часу пошуку об’єкта від кількості агентів

Результати показали закономірне зменшення середнього часу пошуку при збільшенні кількості агентів. Це очікувано, оскільки розподіл зони між більшою кількістю дронів прискорює огляд території.

На графіку видно, що після певного порогу (8–9 дронів) спостерігається ефект насичення — додавання нових агентів не дає суттєвого виграшу в часі. Це пояснюється збільшенням кількості взаємодій і частковим перекриттям маршрутів.

Таким чином, результати тестування демонструють ефективність обраного підходу до організації фазової поведінки і підтверджують придатність реалізованої системи до масштабування в реальних умовах.

4.3 Тестування відмовостойкості розробленої системи керування

У межах дослідження функціональності запропонованої децентралізованої мультиагентної системи було проведено серію тестувань, спрямованих на перевірку її стійкості до втрати окремих агентів. Однією з ключових переваг децентралізованих систем є здатність продовжувати виконання задачі навіть у випадку часткових відмов, тому метою експериментів було оцінити, наскільки ефективно система справляється з пошуком об’єкта за таких умов. Базовим сценарієм для тестування було обрано випадок, у якому початкова кількість дронів становила 10 — це значення було визначене як достатнє для покриття зони пошуку при звичайному режимі роботи. У межах цього сценарію було виконано симуляції, під час яких певна частина дронів вимикалась на початку фази пошуку та в її процесі (імітація відмови або втрати зв’язку). В процесі кожного запуску фіксувався середній час виявлення цілі, розміщеної у випадковій позиції контролером типу supervisor. Для кожного рівня втрат (від 0% до 60%) проводилось кілька запусків, після чого обчислювалося середнє значення часу. Результати цього тесту представлені на графіку (рис. 4.4). Спостерігається зростання часу пошуку із збільшенням кількості недієздатних дронів. При втраті понад 50% агентів помітно значні погіршення у часі.

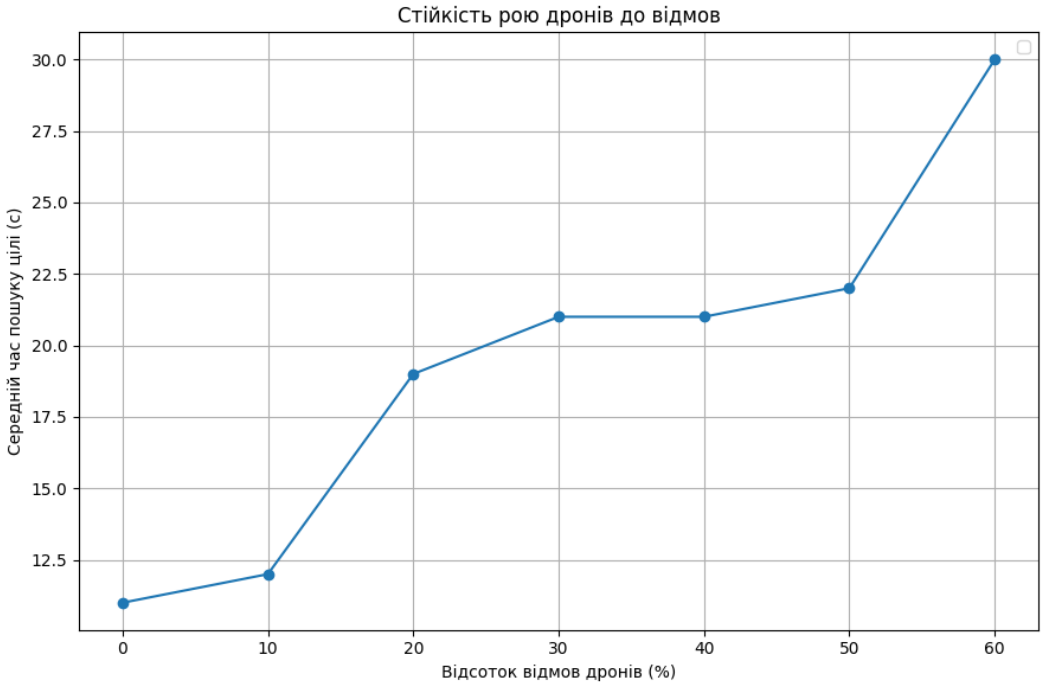

Рисунок 4.4 – залежність часу пошуку цілі від втрат агентів

Для оцінки можливості масштабування системи та підвищення її стійкості при збільшенні агентів, було проведено додатковий експеримент, у якому перевірялась поведінка рою з підвищеною початковою чисельністю — 10, 15 та 20 дронів. Метою цього тесту було з’ясувати, чи дозволяє більша кількість агентів частково компенсувати втрати і підтримувати стабільну ефективність пошуку. Для уникнення насичення, в умоваї даного експерименту було також розширено зону пошуку.

Результати тестів наведені на графіку (рис. 4.5). Як видно з порівняльного аналізу, при однаковому відсотку втрат, рої з більшою початковою кількістю дронів демонструють менше зростання часу пошуку.

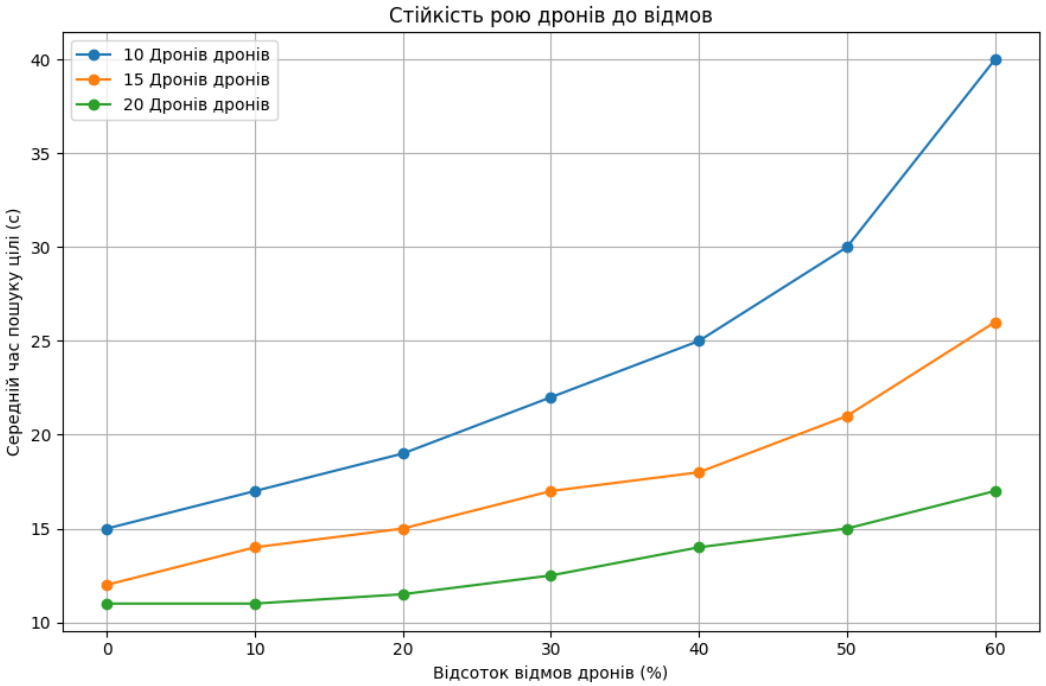

Рисунок 4.5 – Залежність часу пошуку цілі від відсотку втрат агентів для систем з різною кількістю дронів

4.4 Можливості подальшого розвитку системи

У межах подальшого вдосконалення розробленої системи слід звернути увагу на впровадження повноцінного алгоритму виявлення об’єкту, заснованого на нейронних мережах Наразі реалізовано лише імітацію знаходження об’єкту. Доречним буде інтеграція сучасних моделей глибокого навчання, як YOLOv5/YOLOv7, SSD, або застосування розподіленої обробки зображень.

Другим важливим напрямом є оптимізація маршруту пошуку. Замість статичної схеми пошуку може бути використано адаптивне зональне покриття на базі reinforcement learning.

Роширення фаз поведінки збільшить кількість сценаріїв використання системи в автономних місіях. Прикладом може бути фаза збереження позиції, що забезпечить покриття певної зони і подальшу координацію з бойовими дронами.

Висновки до розділу 4

Розділ присвячено тестуванню запропонованої системи в умовах симуляції. У ході випробувань підтверджено працездатність архітектури у штатному режимі: дрони синхронізовано переходять у фазу пошуку, покривають задану область і реагують на виявлення цілі. Система демонструє стабільну поведінку навіть при збільшенні кількості агентів. Проведено тестування відмовостійкості — при виведенні з ладу окремих дронів решта продовжують місію без суттєвого зниження ефективності, що підтверджує переваги децентралізованого підходу. Була виявлена кореляція між кількістю агентів та часом виявлення цілі: зі збільшенням кількості дронів час скорочується. Отримані результати свідчать про ефективність обраної архітектури та можливість її подальшого розвитку, зокрема через інтеграцію нових типів сенсорів, вдосконалення алгоритмів пошуку або застосування методів навчання з підкріпленням.

ВИСНОВКИ

У процесі виконання дипломної роботи було досліджено, спроєктовано та частково реалізовано симульовану децентралізовану систему координації рою безпілотних літальних апаратів (БПЛА) для виявлення об’єктів у зоні розвідки.

Отримані результати дозволяють зробити висновки, представлені нижче.

Обґрунтовано актуальність використання роїв дронів у бойових умовах, особливо в задачах автономної розвідки, де критичними є оперативність, масштабованість та стійкість до втрат окремих агентів.

Проаналізовано предметну область — класифікацію БПЛА, принципи ройової поведінки, біоінспіровані моделі, алгоритми пошуку, а також інструменти симуляції. Особливу увагу приділено моделі Boids як базовій для реалізації колективного руху.

Розроблено архітектуру системи, що складається з автономних агентів-дронів, кожен із яких має власну логіку руху, обробки сенсорних даних, взаємодії з іншими агентами та реагування на зовнішні події. Використано PID-регулятори для стабілізації польоту та комунікаційний обмін для узгодження дій.

Реалізовано основні етапи місії: рух до зони пошуку (travel), сканування місцевості (search), реакція на виявлення об'єкта (regroup). Підключено просту модель виявлення об’єкта за допомогою аналізу зображення з камери.

Імітовано роботу зовнішнього контролера-наглядача (supervisor), який забезпечує генерацію об’єктів у довільних координатах та ініціалізує фазу місії шляхом передачі координат дронам. Завдяки цьому проведено тестування системи.

Проведено тестування системи в умовах симуляції. Систему було оцінено в нормальних умовах без втрати агентів, а також було проведено оцінку відмовостойкості системи в умовах втрати випадкових агентів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Parker, L. E. ALLIANCE: An architecture for fault-tolerant multirobot cooperation // IEEE Transactions on Robotics and Automation. – 1998. – Vol. 14, No. 2. – P. 220–240.

2. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M. Swarm robotics: a review from the swarm engineering perspective // Swarm Intelligence. – 2013. – Vol. 7, No. 1. – P. 1–41.

3. Мірошниченко, В.Ю. Мультиагентні системи: навч. посіб. – Київ: НАУ, 2014. – 176 с.

4. Чижевський, Д.Ю. Основи децентралізованих роботизованих систем. – К.: КПІ ім. І. Сікорського, 2021. – 145 с.

5. Hussein, A., Saad, M., El-Khamy, S. Survey on swarm robotics: Communication, coordination and decision-making // 2020 37th National Radio Science Conference (NRSC). – IEEE, 2020. – P. 1–8.

6. Rubenstein, M., Cornejo, A., Nagpal, R. Programmable self-assembly in a thousand-robot swarm // Science. – 2014. – Vol. 345, Issue 6198. – P. 795–799.

7. Crazyswarm Project– URL: https://github.com/USC-ACTLab/crazyswarm (дата звернення: 10.06.2025).

8. Webots – Open-source robot simulator – URL: https://cyberbotics.com/

9. AirSim: Aerial Informatics and Robotics Platform [Електронний ресурс] – URL: https://github.com/microsoft/AirSim (дата звернення: 05.06.2025).

10. FireRS Project – Fire Remote Sensing [Електронний ресурс] – URL: https://firers.eu (дата звернення: 01.06.2025).

11. DARPA OFFSET Program Overview [Електронний ресурс] – URL: https://www.darpa.mil/program/offensive-swarm-enabled-tactics (дата звернення: 10.05.2025).

12. Камінський, Ю.І. Програмування роботизованих систем у Webots: метод. вказ. – К.: КПІ, 2020. – 84 с.

13. Kennedy, J., Eberhart, R. Particle swarm optimization // Proceedings of ICNN'95 - International Conference on Neural Networks. – Vol. 4. – 1995. – P. 1942–1948.

14. OpenAI. Simple-PID Python Library Documentation [Електронний ресурс] – URL: https://github.com/m-lundberg/simple-pid (дата звернення: 10.06.2025).

15. Мельник О.С. Мультиагентні системи в задачах розвідки: огляд підходів // Автоматизація та сучасні технології. – 2022. – №2. – С. 33–41.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
from controller import Robot
from simple_pid import PID
import sys
import math
import json
from controller import Camera
try:
    import numpy as np
except ImportError:
    sys.exit("Warning: 'numpy' module not found.")
import time


def clamp(value, value_min, value_max):
    return min(max(value, value_min), value_max)




class Mavic (Robot):
    # Constants, empirically found.
    K_VERTICAL_THRUST = 68.5  # with this thrust, the drone
lifts.
    # Vertical offset where the robot actually targets to
stabilize itself.
    K_VERTICAL_OFFSET = 1.5
    K_HORIZONTAL_OFFSET = 10
```

```
        K_VERTICAL_P = 3.0        # P constant of the vertical
PID.
        K_ROLL_P = 50.0           # P constant of the roll PID.
        K_PITCH_P = 30.0          # P constant of the pitch PID.
        SEND_INTERVAL = 0.5
        SEARCH_AREA = 50.0
        BRAKE_DISTANCE = 20
        SAFE_DISTANCE = 10

        MAX_YAW_DISTURBANCE = 0.4
        MAX_PITCH_DISTURBANCE = -1
        # Precision between the target position and the robot
position in meters
        target_precision = 0.5

        boids_weights_travel = [3, 1, 0]
        boids_weights_search = [0, 0, 0]

        def __init__(self):
            Robot.__init__(self)

            self.time_step = int(self.getBasicTimeStep())

            # Get and enable devices.
            self.camera = self.getDevice("camera")
            self.camera.enable(self.time_step)
            self.imu = self.getDevice("inertial unit")
            self.imu.enable(self.time_step)
            self.gps = self.getDevice("gps")
            self.gps.enable(self.time_step)
            self.gyro = self.getDevice("gyro")
```

```
            self.gyro.enable(self.time_step)
            self.emitter = self.getDevice("emitter")
            # self.emitter.enable(self.time_step)
            self.receiver = self.getDevice("receiver")
            self.receiver.enable(self.time_step)
            self.emitter.setChannel(1)
            self.receiver.setChannel(1)

            self.front_left_motor = self.getDevice("front left
propeller")
            self.front_right_motor = self.getDevice("front right
propeller")
            self.rear_left_motor = self.getDevice("rear left
propeller")
            self.rear_right_motor = self.getDevice("rear right
propeller")
            self.camera_pitch_motor = self.getDevice("camera
pitch")
            self.camera_pitch_motor.setPosition(0.7)
            motors = [self.front_left_motor,
self.front_right_motor,
                      self.rear_left_motor,
self.rear_right_motor]
            for motor in motors:
                motor.setPosition(float('inf'))
                motor.setVelocity(1)

            self.current_pos = 6 * [0]  # X, Y, Z, roll, pitch,
yaw
            self.target_pos = [0, 0, 20, 0]
            self.target_index = 0
```

```
            self.last_time_sent = 0

            # PID controllers for x, y and z movement
            self.x_pid = PID(1, 0, 1, sample_time=None,
time_fn=self.getTime)
            self.x_pid.output_limits = (-0.2, 0.2)

            self.y_pid = PID(1, 0, 1, sample_time=None,
time_fn=self.getTime)
            self.y_pid.output_limits = (-0.2, 0.2)

            self.altitude_pid = PID(3, 0.09, 5,
sample_time=None, time_fn=self.getTime)
            self.altitude_pid.output_limits = (-5, 5)

            self.yaw_pid = PID(1, 0, 1, sample_time=None,
time_fn=self.getTime)
            self.yaw_pid.output_limits = (-0.5, 0.5)

            self.mission_phase = "travel"

            # boids logic
            self.w_sep, self.w_coh, self.w_align = [3, 1, 0]
# boids weights
            self.other_drones = np.array([])
            self.boids_vector = [0, 0]

        def setPosition(self, pos):
            self.current_pos = pos

        def convertToLocalFrame(self, x, y):
```

```
            resx = math.cos(self.current_pos[5]) * x +
math.sin(self.current_pos[5]) * y
            resy = math.sin(-self.current_pos[5]) * x +
math.cos(self.current_pos[5]) * y
            return resx, resy

        def setTargetPos(self, target_pos):
            for i in range(len(target_pos)):
                print(i)
                self.target_pos[i] = target_pos[i]

        def setTargetYaw(self, target_pos):
            self.target_pos[3] = math.atan2(target_pos[1] -
self.current_pos[1], target_pos[0] - self.current_pos[0])

        def countDisturbance(self):
            target_pos =
self.convertToLocalFrame(self.target_pos[0], self.target_pos[1])
            current_pos =
self.convertToLocalFrame(self.current_pos[0],
self.current_pos[1])

            pitch_disturbance = self.x_pid(target_pos[0] -
current_pos[0] + self.boids_vector[0])
            roll_disturbance = self.y_pid(target_pos[1] -
current_pos[1] + self.boids_vector[1])
            return roll_disturbance, pitch_disturbance

        def sendPosition(self):
            self.emitter.send(json.dumps({"pos":
self.current_pos[:3]}))
```

```
        def sendTarget(self, target):
            self.emitter.send(json.dumps({"found": target}))

        def checkReceiver(self):
            other_drones = []
            while self.receiver.getQueueLength() > 0:

                message = self.receiver.getString()
                message = json.loads(message)

                if ("pos" in message):
                    other_drones.append(message["pos"])

                elif ("target" in message):
                    self.w_sep, self.w_coh, self.w_align =
self.boids_weights_travel
                    self.mission_phase = "travel"
                    self.setTargetPos(message["target"])
                    self.setTargetYaw(message["target"])

                elif ("found" in message):
                    self.mission_phase = "regroup"
                    self.setTargetPos(message["found"])
                    self.setTargetYaw(self.target_pos)

                self.receiver.nextPacket()

            if other_drones:
                other_drones = np.array(other_drones)
                self.countBoidVector(other_drones)
```

```
        def countBoidVector(self, drones):
            separation = sum([(self.current_pos[:2] -
other_pos[:2]) / np.linalg.norm(self.current_pos[:2] -
other_pos[:2])**2 for other_pos in drones if
np.linalg.norm(self.current_pos[:2] - other_pos[:2]) <=
self.SAFE_DISTANCE])

            cohesion = np.mean(drones[:, 1], axis=0) -
self.current_pos[:2]
            cohesion[0] = 0

            if self.other_drones.size > 0: alignment =
np.mean(self.other_drones[:, :2] - drones[:, :2], axis=0) /
self.SEND_INTERVAL
            else: alignment = 0

            print(f"cohesion = {cohesion}")
            self.other_drones = drones

            boids_vector = self.w_sep * separation + self.w_coh
* cohesion + self.w_align * alignment
            self.boids_vector =
self.convertToLocalFrame(boids_vector[0], boids_vector[1])

        def CheckForObject(self):
            image = self.camera.getImage()
            width = self.camera.getWidth()
            height = self.camera.getHeight()
```

```
            rgba_array = np.frombuffer(image,
dtype=np.uint8).reshape((height, width, 4))

            rgb_array = rgba_array[:, :, :3]

            red_mask = (rgb_array[:, :, 2] > 200) &
(rgb_array[:, :, 1] < 80) & (rgb_array[:, :, 0] < 80)

            return np.any(red_mask)

        def computeSearchArea(self):
            search_square = np.array([[-self.SEARCH_AREA, -
self.SEARCH_AREA], [-self.SEARCH_AREA, self.SEARCH_AREA],
                                      [self.SEARCH_AREA,
self.SEARCH_AREA], [self.SEARCH_AREA, -self.SEARCH_AREA]])

            search_square = search_square -
np.array(self.target_pos[:2])

            search_square_turned = []
            for pos in search_square:
                pos = pos - np.array(self.target_pos[:2])
                pos = [pos[0] * math.cos(self.current_pos[5]) -
pos[1] * math.sin(self.current_pos[5]), pos[0] *
math.sin(self.current_pos[5]) + pos[1] *
math.cos(self.current_pos[5])]
                pos = pos + np.array(self.target_pos[:2])
                search_square_turned.append(pos)

            return search_square_turned
```

```
        def checkLeftNeighboor(self):
            current_pos_local =
self.convertToLocalFrame(self.current_pos[0],
self.current_pos[1])
            for pos in self.other_drones:
                pos_local = self.convertToLocalFrame(pos[0],
pos[1])
                print(f"pos = {pos}, pos_local = {pos_local}")
                if (pos_local[1] > current_pos_local[1]):
                    return pos_local[1]
            return None

        def switchToSearch(self):
            print("starting search")
            search_area = self.computeSearchArea()
            closest_point = search_area[0]
            visible_width = self.current_pos[2] *
math.tan(self.camera.getFov() / 2)

            search_area_local = []
            for pos in search_area:
                if np.linalg.norm(pos[:2] -
self.current_pos[:2]) < np.linalg.norm(closest_point[:2] -
self.current_pos[:2]):
                    closest_point = pos

            left_neighboor = self.checkLeftNeighboor()

            if left_neighboor == None:
                self.setTargetPos([closest_point[0] -
visible_width, closest_point[1]])
```

```
                print(f"target pos after starting search =
{self.target_pos}")
            else:
                self.setTargetPos([left_neighboor -
visible_width, closest_point[1]])

            self.mission_phase = "search"
            self.w_sep, self.w_coh, self.w_align =
self.boids_weights_search




        def run(self):
            roll_disturbance = 0
            pitch_disturbance = 0
            yaw_disturbance = 0
            # self.setTargetPos([20, 20, 50])
            # self.setTargetYaw(self.target_pos)
            # target altitude of the robot in meters

            while self.step(self.time_step) != -1:
                # Read sensors
                roll, pitch, yaw = self.imu.getRollPitchYaw()
                x_pos, y_pos, altitude = self.gps.getValues()
                roll_acceleration, pitch_acceleration, _ =
self.gyro.getValues()

                self.setPosition([x_pos, y_pos, altitude, roll,
pitch, yaw])
```

```
                if self.getTime() - self.last_time_sent >=
self.SEND_INTERVAL:
                    self.last_time_sent = self.getTime()
                    self.sendPosition()
                    # self.CheckForObject()



                #check for crash
                if abs(self.current_pos[3]) >= math.pi/2 or
abs(self.current_pos[4]) >= math.pi/2:
                    break

                #send coordinates
                if (self.receiver.getQueueLength() > 0):
self.checkReceiver()

                #check for reaching search area
                # if self.mission_phase == "travel" and
(abs(self.target_pos[0] - self.current_pos[0]) +
abs(self.target_pos[1] - self.current_pos[1])) <=
self.SEARCH_AREA + self.BRAKE_DISTANCE:
                #     self.switchToSearch()

                if self.mission_phase == "search":
                    if self.CheckForObject():
                        self.sendTarget(self.current_pos)
```

```
                # if reached target altitude, count x and y
disturbance
                if abs(self.target_pos[2] - altitude) <
self.K_VERTICAL_OFFSET:
                    roll_disturbance, pitch_disturbance =
self.countDisturbance()

                # look in the direction of travel
                yaw_input =
self.yaw_pid(math.atan2(math.sin(self.current_pos[5] -
self.target_pos[3]), math.cos(self.current_pos[5] -
self.target_pos[3])))

                #countint final roll, pitch and altitude inputs
                roll_input = self.K_ROLL_P * clamp(roll, -1, 1)
+ roll_acceleration - roll_disturbance
                pitch_input = self.K_PITCH_P * clamp(pitch, -1,
1) + pitch_acceleration + pitch_disturbance
                vertical_input = self.altitude_pid(altitude -
self.target_pos[2])

                #setting motor velocities
                front_left_motor_input = self.K_VERTICAL_THRUST
+ vertical_input - yaw_input + pitch_input - roll_input
                front_right_motor_input = self.K_VERTICAL_THRUST
+ vertical_input + yaw_input + pitch_input + roll_input
                rear_left_motor_input = self.K_VERTICAL_THRUST +
vertical_input + yaw_input - pitch_input - roll_input
                rear_right_motor_input = self.K_VERTICAL_THRUST
+ vertical_input - yaw_input - pitch_input + roll_input
```

```
self.front_left_motor.setVelocity(front_left_motor_input)
                  self.front_right_motor.setVelocity(-
front_right_motor_input)
                  self.rear_left_motor.setVelocity(-
rear_left_motor_input)

self.rear_right_motor.setVelocity(rear_right_motor_input)



     # To use this controller, the basicTimeStep should be set to
8 and the defaultDamping
     # with a linear and angular damping both of 0.5
     robot = Mavic()
     robot.run()
```