Міністерство освіти і науки України
Луцький національний технічний університет
Факультет архітектури, будівництва та дизайну
Кафедра прикладної математики та механіки

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА АДАПТИВНОГО АЛГОРИТМУ УПРАВЛІННЯ БПЛА НА ОСНОВІ ПІДКРІПЛЮВАЛЬНОГО НАВЧАННЯ

DEVELOPMENT OF AN ADAPTIVE UAV CONTROL ALGORITHM BASED ON REINFORCEMENT LEARNING

спеціальність 113 Прикладна математика
освітня програма Прикладна математика

Виконав: здобувач вищої освіти
Групи ПРМм-21
Закревський Віталій Леонідович
_____
(підпис)
Керівник:
к.т.н., доцент
Приходько Олексій Сергійович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
PhD, доцент
Гарант освітньої програми:
Самоненко Інга Вікторівна
_____
(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *архітектури, будівництва та дизайну*

Кафедра *прикладної математики та механіки*

Ступінь вищої освіти: *магістр*

Галузь знань: *11 Математика і статистика*

Спеціальність *113 Прикладна математика*

Освітня програма *Прикладна математика*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Мікуліч О.А.
«___» _____202__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Закревський Віталій Леонідович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Розробка адаптивного алгоритму управління БПЛА на основі підкріплювального навчання
Керівник роботи: *Приходько Олексій Сергійович*
затверджені наказом закладу вищої освіти від «25» січня 2025 р. № 45/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи
«___»_____ 202_ р.
3. Вихідні дані до роботи: *наукові публікації з питань теорії управління БПЛА, методів підкріплювального навчання; набори даних для навчання нейронних мереж (зокрема PyBullet з моделями БПЛА); математичні моделі динаміки квадрокоптерів; технічна документація бібліотек мови програмування Python; методичні вказівки до виконання кваліфікаційної роботи магістра.*
4. Зміст пояснювальної записки (перелік питань, що потрібно розробити):
Огляд літератури та аналіз проблематики за темою роботи, постановка завдань дослідження. Розробка математичної моделі квадрокоптера та формалізація задачі керування; вибір алгоритму підкріплювального навчання; реалізація алгоритму. Експериментальна перевірка та аналіз результатів. Висновки
5. Перелік графічного (ілюстративного) матеріалу: *Презентація роботи (слайди):*
Схема сил та моментів, що діють на БПЛА. Блок-схема архітектури нейронних мереж агента (Актор-Критик). Графік динаміки навчання алгоритму. Візуалізація порівняння траєкторій польоту. Графіки реакції системи на вітрове збурення та зміну мас.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
1 розділ	*Приходько О.С., доцент кафедри*		
2 розділ	*Приходько О.С., доцент кафедри*		
3 розділ	*Приходько О.С., доцент кафедри*		
4 розділ	*Приходько О.С., доцент кафедри*		
Висновки	*Приходько О.С., доцент кафедри*		

7. Дата видачі завдання «25» січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1.	*Обґрунтування теми*	*до 25.01.2025*	
2.	*Огляд літератури із досліджуваної проблеми*	*до 03.06.2025*	
3.	*Перший розділ*	*до 11.09.2025*	
4.	*Другий розділ*	*до 17.10.2025*	
5.	*Третій розділ*	*до 20.11.2025*	
6.	*Висновки та пропозиції*	*до 25.11.2025*	
7.	*Формування списку використаних джерел*	*до 29.11.2025*	
8.	*Формування додатків*	*до 03.12.2025*	
9.	*Оформлення ілюстративного матеріалу*	*до 09.12.2025*	
0.	*Нормоконтроль*	*до 13.12.2025*	
11.	*Інструментальна перевірка на академічний плагіат*	*до 22.12.2025*	*Показник запозичень тексту _____ %*
12.	*Представлення кваліфікаційної роботи магістра до захисту*	*до 26.12.2025*	

Здобувач вищої освіти _____ (Закревський В.Л.)
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (Приходько О.С.)
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Закревський В. Л. Розробка адаптивного алгоритму управління БПЛА на основі підкріплювального навчання. Рукопис.

Кваліфікаційна робота магістра ОП «Прикладна математика» спеціальності 113 «Прикладна математика». – Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається з вступу, чотирьох розділів, висновків і пропозицій, списку використаних джерел.

У роботі досліджено проблему створення автономних систем управління безпілотними літальними апаратами, здатних ефективно функціонувати в умовах невизначеності. Обґрунтовано використання методів глибокого підкріплювального навчання для задач безперервного керування. Розроблено та програмно реалізовано адаптивний алгоритм на основі методу Proximal Policy Optimization (PPO) із застосуванням стратегії доменної рандомізації, що забезпечує робастність системи до змін динаміки та зовнішніх збурень. Створено програмний комплекс мовою Python у середовищі PyBullet для навчання та тестування агента. Експериментальна перевірка підтвердила перевагу розробленого алгоритму над класичним ПІД-регулятором у сценаріях зі змінним корисним навантаженням та вітровим впливом.

Ключові слова: прикладна математика, БПЛА, навчання з підкріпленням, алгоритм PPO, адаптивне управління, доменна рандомізація, Python, PyBullet.

ABSTRACT

Zakrevskii V. L. Development of an adaptive UAV control algorithm based on reinforcement learning. Manuscript.

Master's thesis in the educational program «Applied Mathematics», specialty 113 «Applied Mathematics». – Lutsk National Technical University. Lutsk, 2025.

The master's thesis consists of an introduction, three chapters, conclusions and recommendations, and a list of references.

The thesis investigates the problem of creating autonomous control systems for unmanned aerial vehicles (UAVs) capable of functioning effectively under conditions of uncertainty. The application of deep reinforcement learning methods for continuous control tasks is substantiated. An adaptive algorithm based on the Proximal Policy Optimization (PPO) method was developed and implemented using a domain randomization strategy, which ensures system robustness to changes in dynamics and external disturbances. A software module in Python within the PyBullet environment was created for training and testing the agent. Experimental verification confirmed the advantage of the developed algorithm over the classical PID controller in scenarios with variable payload and wind influence.

Keywords: applied mathematics, UAV, reinforcement learning, PPO algorithm, adaptive control, domain randomization, Python, PyBullet.

ЗМІСТ

ВСТУП

В сучасному світі безпілотні літальні апарати (БПЛА) перестали бути винятково військовою технологією, стрімко інтегруючись у цивільні сфери та трансформуючи низку галузей. Від моніторингу сільськогосподарських угідь до доставки вантажів, від інспекції інфраструктури до участі в пошуково-рятувальних операціях – спектр застосувань БПЛА постійно розширюється. Цей стрімкий розвиток обумовлений їхньою здатністю виконувати завдання в небезпечних або важкодоступних місцях, забезпечувати високу точність даних та значно підвищувати ефективність багатьох процесів. Однак, незважаючи на вже існуючі досягнення, більшість сучасних систем управління БПЛА все ще стикаються з суттєвими обмеженнями, особливо коли мова йде про їхню автономність та здатність ефективно функціонувати в умовах невизначеності.

Традиційні підходи до управління БПЛА, засновані на класичних теоріях автоматичного регулювання, як правило, вимагають точних математичних моделей об'єкта управління та навколишнього середовища. Вони чудово працюють у контрольованих умовах, де динаміка апарату та характеристики зовнішнього середовища добре відомі та стабільні. Проте, реальний світ рідко буває таким ідеальним. Непередбачувані пориви вітру, зміна ваги корисного навантаження, зношування компонентів, раптові відмови датчиків або двигунів, а також перешкоди, що з'являються в польоті – все це є типовими викликами, які можуть істотно знизити продуктивність або навіть призвести до втрати керованості БПЛА. В таких ситуаціях попередньо запрограмовані контролери часто виявляються неефективними, оскільки їм бракує гнучкості та здатності адаптуватися до нових, непередбачених обставин.

Саме тут на передній план виходить концепція адаптивного управління. Адаптивні системи мають потенціал модифікувати свою поведінку та налаштовувати параметри управління в режимі реального часу, реагуючи на зміни у внутрішній динаміці БПЛА або у зовнішньому середовищі. Це критично важливо для підвищення надійності, безпеки та ефективності автономних польотів. Водночас, розробка таких систем є складною задачею,

оскільки вона вимагає інтеграції передових математичних методів, комп'ютерних наук та інженерних знань.

Останні роки ознаменувалися проривом у галузі штучного інтелекту, зокрема у підкріплювальному навчанні (Reinforcement Learning, RL), яке є потужною парадигмою для навчання агентів оптимальної поведінки шляхом взаємодії з динамічним середовищем. На відміну від традиційного керованого навчання, де система навчається на заздалегідь розмічених даних, RL дозволяє агенту самостійно відкривати найкращі стратегії дій, отримуючи "винагороду" за бажану поведінку та "покарання" за небажану. Цей підхід імітує процес навчання живих організмів, які методом проб і помилок покращують свої навички. Потенціал RL у вирішенні складних задач управління, де точні математичні моделі є недоступними або надто складними, зробив його надзвичайно привабливим для застосування в робототехніці та, зокрема, в управлінні БПЛА.

Метою даної магістерської роботи є розробка та реалізація адаптивного алгоритму управління БПЛА, заснованого на принципах підкріплювального навчання.

Для досягнення цієї глобальної мети було сформульовано низку конкретних завдань.

1. Провести детальний аналіз сучасних підходів до управління БПЛА та методів підкріплювального навчання, виявивши їхні переваги, недоліки та можливості для інтеграції.

2. Розробити математичну модель обраного типу БПЛА, яка буде використана в симуляційному середовищі для навчання та тестування алгоритму.

3. Формалізувати задачу управління БПЛА як Марковський процес вирішення (MDP), визначивши простір станів, дій та функцію винагороди.

4. Обрати та обґрунтувати конкретний алгоритм підкріплювального навчання, що найкраще підходить для вирішення поставлених задач, з урахуванням безперервності просторів станів та дій.

5. Розробити архітектуру адаптивного контролера на основі обраного RL-алгоритму, включаючи структуру нейронних мереж та механізми адаптації.

6. Реалізувати розроблений алгоритм у симуляційному середовищі, використовуючи сучасні програмні засоби.

7. Провести серію експериментів для оцінки ефективності, стійкості та адаптивності розробленого алгоритму в різних польотних сценаріях, включаючи умови зі збуреннями та змінами динаміки БПЛА.

8. Здійснити порівняльний аналіз продуктивності розробленого RL-контролера з традиційними методами управління.

Наукова новизна роботи полягає в розробці та застосуванні оригінальної конфігурації алгоритму підкріплювального навчання для адаптивного управління БПЛА, що дозволить підвищити стійкість системи до непередбачених внутрішніх і зовнішніх змін, а також впровадженні специфічної функції винагороди, оптимізованої для складних польотних задач. Практична цінність роботи полягає у створенні надійної основи для подальшої розробки повністю автономних та високоадаптивних систем управління БПЛА, які можуть бути застосовані в широкому спектрі реальних сценаріїв, від моніторингу до логістики, значно підвищуючи їхню ефективність та безпеку експлуатації. Структура даної роботи включає теоретичні основи, опис розробленого алгоритму, детальний опис експериментальної частини та аналіз отриманих результатів, а також висновки та перспективи подальших досліджень.

Апробація результатів дослідження. Ключові аспекти розробленої методики були представлені на Міжнародній науковій конференції «Actual Problems of Automation and Control» м. Луцьк, 26 листопада 2025 р. Також подана до друку наукова стаття (Наукові нотатки, ЛНТУ).

У процесі підготовки магістерської кваліфікаційної роботи застосовувалися технології штучного інтелекту як допоміжний інструментарій. Зокрема, для стилістичної правки та структурування тексту використано Gemini 3. Автор несе повну відповідальність за зміст роботи: усі наукові положення та висновки є результатом самостійного дослідження, а згенеровані матеріали пройшли ретельну верифікацію на предмет точності та відсутності плагіату

РОЗДІЛ 1.
ОГЛЯД ЛІТЕРАТУРИ

Глибоке розуміння сучасних підходів до управління БПЛА та передових методів підкріплювального навчання є фундаментальним для успішної реалізації поставлених у даній роботі завдань. Цей розділ присвячений детальному огляду існуючих наукових досліджень та розробок, які закладають теоретичну та практичну основу для проектування адаптивного алгоритму управління. Безпілотні літальні апарати, зокрема квадрокоптери, що є поширеним типом для досліджень та комерційного використання, є складними нелінійними динамічними системами з багатьма ступенями свободи. Їхня динаміка описується комплексом рівнянь, що враховують сили тяжіння, підйомну силу, опір повітря, моменти інерції та реактивні моменти від пропелерів.

Розробка точної математичної моделі БПЛА є першим кроком у проектуванні будь-якої системи управління, оскільки вона дозволяє симулювати поведінку апарату та проводити тестування в контрольованому середовищі. Ці моделі зазвичай базуються на законах Ньютона-Ейлера і описують практично будь-які поєднання як поступального, так і обертального руху апарату. Наразі можна стверджувати про значні досягнення у сфері БПЛА, що призвели до їхнього широкого практичного розгортання у багатьох областях, включаючи мобільний зв'язок, спостереження, військові застосування та рятувальні місії [1]. У таких сценаріях, як рятувальні операції у важкодоступних регіонах або надання комунікаційних послуг у зонах лиха, БПЛА-мережі ad-hoc є незамінними [2]. Мережі Ad-hoc безпілотних літальних апаратів є типом бездротової комунікаційної мережі, в якій БПЛА формують мережу без необхідності інфраструктури.

Проте, БПЛА-мережі ad-hoc стикаються з викликами, такими як нестабільний зв'язок та обмежений ресурс батареї, що ускладнює забезпечення надійних мереж. Крім того, топологія мережі постійно змінюється через рух БПЛА, що створює значні порушення в управлінні мережею та розподілі

ресурсів [3]. Традиційні методи управління, такі як ПІД-регулятори, є одними з найстаріших і найпоширеніших у промисловості завдяки своїй простоті реалізації та відносній ефективності в стабільних умовах. Проте, їхня продуктивність значно знижується за наявності нелінійностей, зовнішніх збурень або змін параметрів системи. Більш просунуті методи включають лінійно-квадратичні регулятори (LQR), керування зворотним кроком (Backstepping Control) або керування за допомогою ковзних режимів (Sliding Mode Control), які здатні впоратися з нелінійностями та забезпечити більшу робастність. Для досягнення ефективної маршрутизації від джерела до пункту призначення дослідники спочатку пропонували IP-базовані протоколи маршрутизації [4], які поділяються на проактивні та реактивні [5]. Проактивні протоколи, такі як описаний у [6] swarm control-based scheme, підтримують маршрути шляхом моніторингу змін топології. Проте, вони часто несуть високі контрольні накладні витрати. Реактивні протоколи, як-от [7] та [8], були запропоновані для зменшення накладних витрат, але все ще стикаються з труднощами у високо-мобільних середовищах. Дослідники використовували протоколи маршрутизації, засновані на топології, включаючи DSR, AODV та HRP, для мереж на основі БПЛА з метою вирішення цих проблем [9].

Паралельно з розвитком традиційного управління, останні десятиліття принесли революційні зміни у сфері штучного інтелекту, зокрема технології підкріплювального навчання (Reinforcement Learning - RL). RL – це область машинного навчання, де агент навчається приймати оптимальні рішення, взаємодіючи з динамічним середовищем. Цей процес навчання відбувається шляхом отримання "винагороди" або "покарання" за виконані дії. Ключовими поняттями є: агент, середовище, стан, дія, винагорода, політика та функція цінності. Існують різні типи RL-алгоритмів, які можна класифікувати як Model-based та Model-free. Серед класичних алгоритмів варто виділити Q-learning, SARSA та Monte Carlo методи, які добре працюють у дискретних просторах станів та дій. Однак, для задач управління БПЛА, де простір станів та простір дій є безперервними, класичні методи стають неефективними. Тут на допомогу приходять сучасні алгоритми, що використовують глибокі нейронні мережі, такі як Deep Q-Network (DQN), Deep Deterministic Policy

Gradient (DDPG), Actor-Critic (A2C/A3C), а також більш сучасні та ефективні алгоритми, як-от Proximal Policy Optimization (PPO) та Soft Actor-Critic (SAC), які пропонують кращу стабільність навчання та ефективність вибірки, що робить їх особливо привабливими для складних задач управління.

Нарешті, варто згадати застосування RL в управлінні роботами та БПЛА. Значна кількість досліджень демонструє успішне використання RL для вирішення широкого спектру задач. Зокрема, кластерні підходи були введені для покращення масштабованості та енергоефективності UAV ad hoc мереж. Прикладами є Energy-Efficient Clustering Algorithm (EECA) [10], Load-Balanced Clustering Algorithm (LBCA) [11], Hierarchical Clustering Algorithm (HCA) [12] та Swarm Intelligence-Based Clustering Algorithm (SICA) [13]. Крім того, фреймворки на основі RL все частіше використовуються для уникнення зіткнень у щільних ройових операціях, забезпечуючи безпечну навігацію у складному 3D-повітряному просторі [14]. Для вирішення суворих енергетичних обмежень БПЛА були розроблені RL-стратегії для оптимізації траєкторій польоту, потужності передачі та обчислювального розвантаження, що значно покращує термін служби мережі та операційну ефективність [15]. Ці роботи підкреслюють універсальність RL як потужного інструменту для цілісного управління мережами БПЛА [16]. Аналіз цих робіт дозволяє виявити переваги та недоліки різних підходів: RL-контролери демонструють високу адаптивність і можуть генерувати неінтуїтивні, але ефективні стратегії управління, проте їхнє навчання часто вимагає великої кількості взаємодій з середовищем, є обчислювально витратним та чутливим до налаштування гіперпараметрів та дизайну функції винагороди. Крім того, перенесення навчених політик з симуляції на реальні апарати (Sim-to-Real transfer) залишається суттєвим викликом через невідповідність моделей та невизначеність реального світу. Саме ці виклики та можливості визначають прогалини у поточних дослідженнях, які ця магістерська робота прагне заповнити, зосереджуючись на підвищенні адаптивності RL-контролерів до значних, непередбачених змін у динаміці БПЛА або у характеристиках середовища.

РОЗДІЛ 2.

ТЕОРЕТИЧНІ ОСНОВИ

2.1. Модель БПЛА

Для ефективної розробки та симуляції адаптивного алгоритму управління безпілотним літальним апаратом необхідно мати точну та повну математичну модель об'єкта управління. У даній роботі об'єктом дослідження є квадрокоптер – тип БПЛА, що вирізняється чотирма роторами, які розташовані попарно на перпендикулярних осях, і кожен з яких здатний створювати підйомну силу. Така конфігурація дозволяє апарату виконувати вертикальний зліт/посадку, зависати в повітрі та здійснювати маневри за рахунок зміни швидкості обертання роторів. Математична модель квадрокоптера є нелінійною та багатозв'язною, що робить задачу його управління досить складною, але водночас цікавою з точки зору застосування методів підкріплювального навчання.

Розглянемо квадрокоптер як тверде тіло, що рухається у тривимірному просторі. Для опису його руху ми будемо використовувати дві системи координат:

1. Інерціальна (глобальна) система координат $E = \{\vec{e}_x, \vec{e}_y, \vec{e}_z\}$, яка є нерухомою відносно Землі. Вісь $\vec{e}_z$ спрямована вертикально вгору.

2. Зв'язана (тілесна) система координат $B = \{\vec{b}_x, \vec{b}_y, \vec{b}_z\}$, жорстко пов'язана з центром мас квадрокоптера. Вісь $\vec{b}_x$ зазвичай спрямована вперед, $\vec{b}_y$ – вправо, а $\vec{b}_z$ – вниз (або вгору, залежно від конвенції). У даній моделі ми припускаємо, що вісь $\vec{b}_z$ спрямована вниз, щоб спростити опис моменту.

2.1.1. Кінематика квадрокоптера

Положення центру мас квадрокоптера в інерціальній системі координат описується вектором $p = [x, y, z]^T$. Швидкість руху центру мас квадрокоптера відносно інерціальної системи координат позначається як $V = [\dot{x}, \dot{y}, \dot{z}]^T$.

Орієнтація квадрокоптера (його кутове положення) описується кутами Ейлера:

ϕ (крен, roll) – обертання навколо осі $\vec{b}_x$

θ (тангаж, pitch) – обертання навколо осі $\vec{b}_y$

ψ (рискання, yaw) – обертання навколо осі $\vec{b}_z$.

Кутові швидкості квадрокоптера відносно зв'язаної системи координат позначаються як $\omega = [p, q, r]^T$, де $p = \dot{\phi}$, $q = \dot{\theta}$, $r = \dot{\psi}$ є кутовими швидкостями крену, тангажу та рискання відповідно. Зв'язок між кутовими швидкостями в зв'язаній системі координат та похідними кутів Ейлера задається кінематичними рівняннями:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

Зауважимо, що ця матриця має сингулярність при $\theta = \pm\pi/2$, що відповідає прямому вертикальному положенню апарата (гімбал лок). Для уникнення цієї проблеми в деяких випадках можуть використовуватися кватерніони, але для польотів, де тангаж не перевищує цих критичних значень, кути Ейлера є цілком прийнятними.

2.1.2. Динаміка квадрокоптера

Динаміка руху квадрокоптера описується двома основними рівняннями: рівнянням поступального руху та рівнянням обертального руху.

Рівняння поступального руху: На квадрокоптер діють дві основні сили:

1. Сила тяжіння: $F_g = mg\vec{e}_z$, де m – маса квадрокоптера, g – прискорення вільного падіння. У зв'язаній системі координат сила тяжіння перетворюється за допомогою матриці повороту R:

$$F_g^B = R^T \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix}$$

2. Сумарна підйомна сила від роторів: кожен ротор створює тягу F_i, пропорційну квадрату його кутової швидкості обертання ω_i: $F_i = k_f \omega_i^2$, де k_f – коефіцієнт тяги. Сумарна тяга $U_1 = \sum_{i=1}^{4} F_i$ спрямована вздовж осі $\vec{b}_z$

квадрокоптера (приймаючи, що $\vec{b}_z$ спрямована вниз). Таким чином, у зв'язаній системі координат повна тяга має вигляд:

$$F_T^B = \begin{bmatrix} 0 \\ 0 \\ -U_1 \end{bmatrix}.$$

Матриця повороту R від зв'язаної системи координат до інерціальної системи координат задається наступним чином:

$$R = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

З урахуванням сил, рівняння поступального руху в інерціальній системі координат має вигляд (за другим законом Ньютона):

$$m\ddot{p} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R\begin{bmatrix} 0 \\ 0 \\ -U_1 \end{bmatrix}$$

$$\ddot{x} = \frac{U_1}{m}(\sin\psi\sin\phi - \cos\psi\sin\theta\cos\phi)$$

$$\ddot{y} = \frac{U_1}{m}(-\cos\psi\sin\phi - \sin\psi\sin\theta\cos\phi)$$

$$\ddot{z} = \frac{U_1}{m}(\cos\theta\cos\phi) - g\ .$$

Ці рівняння описують прискорення центру мас квадрокоптера.

Рівняння обертального руху. Обертальний рух квадрокоптера викликається моментами, що створюються різницею тяги роторів та ефектом реактивного моменту. На квадрокоптер діють наступні моменти:

- Момент крену (U_2) створюється різницею тяги між роторами 1 і 3 (або 2 і 4);
- Момент тангажу (U_3) створюється різницею тяги між роторами 2 і 4 (або 1 і 3);
- Момент рискання (U_4) створюється різницею реактивних моментів, що виникають при обертанні роторів. Ротори, що обертаються за годинниковою стрілкою (CW), створюють реактивний момент в одному напрямку, а ротори, що обертаються проти годинникової стрілки (CCW),

– в протилежному. Змінюючи швидкість обертання цих пар, можна створити бажаний момент рискання.

Згідно з другим законом Ейлера для обертального руху, рівняння має вигляд: $I\dot{\omega}+\omega\times(I\omega)=\tau$ де I – тензор інерції квадрокоптера відносно центру мас, $\omega=[p,q,r]^T$ – вектор кутових швидкостей у зв'язаній системі координат, $\tau=[U_2,U_3,U_4]^T$ – вектор моментів, що діють на квадрокоптер.

Для симетричного квадрокоптера тензор інерції можна вважати діагональним:

$$I=\begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}$$

де I_x, I_y, I_z – моменти інерції відносно осей $\vec{b}_x, \vec{b}_y, \vec{b}_z$ відповідно. Припускаючи симетрію $I_x = I_y$.

Тоді рівняння обертального руху набувають вигляду:

$$I_x\dot{p}+(I_z-I_y)qr=U_2$$

$$I_y\dot{q}+(I_x-I_z)pr=U_3$$

$$I_z\dot{r}+(I_y-I_x)pq=U_4$$

Якщо $I_x = I_y$, то:

$$I_x\dot{p}+(I_z-I_x)qr=U_2$$

$$I_x\dot{q}+(I_x-I_z)pr=U_3$$

$$I_z\dot{r}=U_4$$

Кутові прискорення:

$$\dot{p}=\frac{U_2-(I_z-I_x)qr}{I_x}$$

$$\dot{q}=\frac{U_3-(I_x-I_z)pr}{I_x}$$

$$\dot{r}=\frac{U_4}{I_z}$$

2.1.3. Вхідні параметри (керуючі впливи) від роторів

Керуючі впливи U_1, U_2, U_3, U_4 формуються за рахунок зміни швидкостей обертання чотирьох роторів. Припустимо, ротори розташовані наступним

чином (див. рисунок 1): ротор 1 – передній, ротор 2 – правий, ротор 3 – задній, ротор 4 – лівий. Ротори 1 і 3 обертаються в одному напрямку (наприклад, проти годинникової стрілки), а ротори 2 і 4 – в іншому (за годинниковою стрілкою).

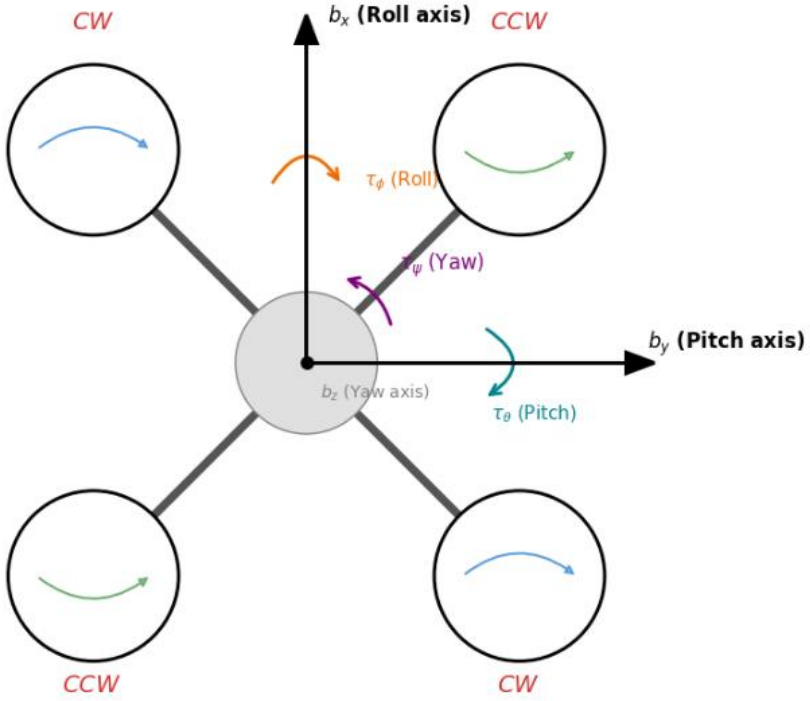

Рисунок 2.1 – Схематичне розташування роторів на квадрокоптері

Нехай l – відстань від центру мас квадрокоптера до центру кожного ротора. k_f – коефіцієнт тяги, k_m – коефіцієнт моменту опору. ω_i – кутова швидкість i-го ротора.

Тоді керуючі впливи можна виразити через кутові швидкості роторів:

$$U_1 = k_f(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2),$$

$$U_2 = l \cdot k_f(\omega_4^2 - \omega_2^2),$$

$$U_3 = l \cdot k_f(\omega_3^2 - \omega_1^2),$$

$$U_4 = k_m(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2).$$

Знак у U_4 залежить від напрямку обертання роторів та конвенції осей. У даному випадку припускаємо, що ротори 1 і 3 обертаються в одному напрямку, а ротори 2 і 4 – в протилежному, що створює різницю реактивних моментів для керування рискою.

2.1.4. Параметри моделі квадрокоптера

Для повноцінної симуляції необхідно визначити конкретні параметри квадрокоптера:

- маса (m);
- моменти інерції (I_x, I_y, I_z);
- відстань від центру до ротора (l);
- коефіцієнт тяги ротора (k_f);
- коефіцієнт моменту опору ротора (k_m);
- прискорення вільного падіння (g).

Ці параметри є основою для налаштування симуляційного середовища та відображення реальної динаміки квадрокоптера.

2.1.5. Повна система рівнянь стану

Об'єднавши кінематичні та динамічні рівняння, ми отримуємо повну систему рівнянь стану, яка описує поведінку квадрокоптера. Вектор стану X можна визначити як:

$$X = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, p, q, r]^T,$$

де перші три компоненти – положення, наступні три – лінійні швидкості, далі три – кути орієнтації, і останні три – кутові швидкості.

Динаміка системи може бути записана як набір диференціальних рівнянь:

$$\dot{x} = \dot{x},$$

$$\dot{y} = \dot{y},$$

$$\dot{z} = \dot{z},$$

$$\ddot{x} = \frac{U_1}{m}(\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi),$$

$$\ddot{y} = \frac{U_1}{m}(\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi),$$

$$\ddot{z} = \frac{U_1}{m}(\cos\phi \cos\theta) - g.$$

Тут знаки для U_1 вказано відповідно до поширеної конвенції, де вісь $\vec{b}_z$ спрямована вгору, або сила тяги U_1 спрямована проти сили тяжіння. Якщо $\vec{b}_z$ спрямована вниз, то знаки будуть іншими, як у попередніх рівняннях. Тут ми

використовуємо конвенцію, де тяга протидіє гравітації, отже $R_{33} = \cos\phi\cos\theta$ є елементом, що відповідає за напрямок тяги в інерціальній Z-координаті). Кутові швидкості:

$$\dot{\phi} = p + \sin\phi\tan\theta q + \cos\phi\tan\theta r$$

$$\dot{\theta} = \cos\phi q - \sin\phi r$$

$$\dot{\psi} = \sin\phi / \cos\theta q + \cos\phi / \cos\theta r \ .$$

Кутові прискорення:

$$\dot{p} = qr\frac{I_y - I_z}{I_x} + \frac{U_2}{I_x}$$

$$\dot{q} = pr\frac{I_z - I_x}{I_y} + \frac{U_3}{I_y}$$

$$\dot{r} = pq\frac{I_x - I_y}{I_z} + \frac{U_4}{I_z} \ .$$

Припускаємо, що I_x, I_y, I_z – це головні моменти інерції, і тензор інерції є діагональним. Для симетричного квадрокоптера $I_x = I_y$, що спрощує останні три рівняння.)

Ця система рівнянь є основою для розробки симуляційного середовища. Вона дозволяє точно моделювати поведінку квадрокоптера під дією керуючих впливів і зовнішніх збурень. Для імітації реального середовища, до цієї моделі можуть бути додані фактори, такі як опір повітря (пропорційний квадрату швидкості), ефект ґрунту, шум датчиків та вітрові збурення, що підвищить реалістичність симуляції та дозволить навчати більш робастні контролери. У контексті підкріплювального навчання, саме ця динамічна модель буде слугувати "середовищем", з яким агент-контролер взаємодіятиме, отримуючи інформацію про стан та винагороду за свої дії.

2.2. Формалізація задачі управління як Марковського процесу вирішення (MDP)

Для того щоб застосувати методи підкріплювального навчання до задачі управління квадрокоптером, необхідно представити цю задачу у відповідній

математичній структурі. Марковський процес вирішення (Markov Decision Process, MDP) є стандартним фреймворком для моделювання задач послідовного прийняття рішень в умовах невизначеності. Цей підхід дозволяє формалізувати взаємодію між агентом (контролером) та середовищем (квадрокоптером та його оточенням), визначаючи цілі навчання через систему винагород. MDP описується кортежем (S, A, P, R, γ), де кожен елемент має своє специфічне значення в контексті нашої задачі. Цей ітеративний процес взаємодії можна візуально представити у вигляді стандартного циклу зворотного зв'язку "агент-середовище", як показано на рис. 2.2.

Рисунок 2.2 – Схема циклу взаємодії агента з середовищем у навчанні з підкріпленням

У контексті нашої задачі компоненти цього циклу мають значення. Агент – це наш адаптивний контролер на основі нейронних мереж, який навчається приймати рішення. Середовище – це симуляція динаміки квадрокоптера, що реагує на дії агента. У кожен момент часу t агент отримує Стан s_t (положення, швидкість БПЛА), виконує Дію a_t (команди до двигунів), після чого середовище переходить у новий стан s_{t+1} і повертає агенту Винагороду r_t, яка оцінює якість виконаної дії. Метою агента є максимізація сумарної винагороди.

Простір станів (S). Простір станів S визначає повний набір інформації, доступної агенту в кожен момент часу для прийняття рішення. Стан має задовольняти Марковській властивості, тобто поточний стан s_t повинен містити всю необхідну інформацію для передбачення майбутніх станів s_{t+1}, без необхідності знати повну історію попередніх станів та дій. У задачі управління квадрокоптером простір станів повинен містити як інформацію про власну динаміку апарату, так і інформацію, що стосується поставленого завдання (наприклад, досягнення цільової точки).

На основі динамічної моделі, описаної в розділі 2.1, ми можемо визначити основний вектор стану, що описує динаміку БПЛА, як 12-вимірний вектор: $s_{drone} = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, p, q, r]^T$ де $[x, y, z]$ – положення центру мас в інерціальній системі координат, $[\dot{x}, \dot{y}, \dot{z}]$ – лінійні швидкості, $[\phi, \theta, \psi]$ – кути Ейлера (крен, тангаж, рискання), а $[p, q, r]$ – кутові швидкості у зв'язаній системі координат.

Однак, для виконання конкретних завдань, таких як слідування траєкторії або досягнення цільової точки, цього вектору недостатньо. Агент повинен знати, куди йому потрібно рухатися. Тому простір станів доповнюється інформацією про ціль. Наприклад, для задачі досягнення точки $p_{target} = [x_t, y_t, z_t]$, ефективніше надавати агенту не абсолютні координати цілі, а вектор відносної позиції, оскільки це робить політику більш узагальнюючою та інваріантною до зсуву: $s_{target} = [x_t - x, y_t - y, z_t - z]^T$. Таким чином, повний вектор стану для задачі досягнення цілі може бути визначений як:

$$s = [s_{drone}, s_{target}]^T = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, p, q, r, x_t - x, y_t - y, z_t - z]^T .$$

Це 15-вимірний вектор стану. Важливо зазначити, що простір станів є безперервним, що вимагає використання RL-алгоритмів, здатних працювати з функціональними апроксиматорами, такими як нейронні мережі. Для покращення стабільності навчання нейронних мереж, компоненти вектору стану зазвичай нормалізуються до певного діапазону (наприклад, [-1, 1]) перед подачею на вхід мережі.

Простір дій (A).

Простір дій A визначає набір усіх можливих керуючих впливів, які агент може застосувати в будь-якому стані. Для квадрокоптера дії безпосередньо пов'язані з керуванням тягою чотирьох роторів. Існує два основні підходи до визначення простору дій:

1. Низькорівневий простір дій: Агент безпосередньо керує швидкостями обертання роторів. Дія може бути представлена як 4-вимірний вектор, що відповідає квадратам кутових швидкостей кожного ротора: $a = [\omega_1^2, \omega_2^2, \omega_3^2, \omega_4^2]^T$. Цей підхід надає агенту повний контроль над динамікою, але є більш складним для навчання, оскільки агент повинен самостійно вивчити складні взаємозв'язки між роторами для стабілізації та маневрування.

2. Високорівневий простір дій: Агент генерує бажані значення для сумарної тяги та кутів крену, тангажу і рискання, а окремий низькорівневий контролер (наприклад, ПІД) перетворює ці бажані значення у команди для роторів. Це спрощує задачу для RL-агента, але обмежує його гнучкість.

У контексті розробки повністю адаптивного алгоритму, низькорівневий підхід є більш привабливим, оскільки він дозволяє агенту знайти оптимальні стратегії управління без обмежень, що накладаються попередньо розробленими контролерами. Таким чином, ми визначаємо простір дій як 4-вимірний безперервний вектор, де кожна компонента відповідає нормованій тязі одного з роторів, наприклад, $a = [a_1, a_2, a_3, a_4]^T$, де $a_i \in [0,1]$ – нормоване значення, яке потім масштабується до реального діапазону тяги двигуна. Як і простір станів, простір дій є безперервним, що є ще однією причиною для вибору сучасних RL-алгоритмів, таких як PPO, DDPG або SAC.

Функція переходу (P).

Функція переходу $P(s' \mid s, a)$ визначає ймовірність переходу в стан s' з стану s при виконанні дії a. У детермінованому середовищі ця ймовірність дорівнює 1 для одного конкретного наступного стану. У стохастичному середовищі, де присутні випадкові збурення (наприклад, вітер), функція переходу описує розподіл ймовірностей по всіх можливих наступних станах.

У нашому випадку функція переходу не є відомою в явній формі. Вона неявно задана складною системою нелінійних диференціальних рівнянь, що

описують динаміку квадрокоптера (розділ 2.1), а також будь-якими змодельованими зовнішніми факторами. RL-алгоритми, що не вимагають моделі (model-free), такі як ті, що ми розглядаємо, не потребують знання цієї функції. Замість цього, вони навчаються, взаємодіючи з середовищем (симулятором) і отримуючи зразки переходів (s_t, a_t, r_t, s_{t+1}), що дозволяє їм опосередковано вивчати динаміку системи.

Функція винагороди (R).

Функція винагороди $R(s,a,s')$ є найважливішим елементом у формулюванні задачі, оскільки вона кількісно визначає мету навчання для агента. Мета агента – максимізувати сумарну дисконтовану винагороду протягом епізоду. Правильний дизайн функції винагороди є критичним: вона повинна заохочувати бажану поведінку (досягнення цілі, стабільний політ, енергоефективність) та штрафувати небажану (зіткнення, відхилення від курсу, нестабільність).

Для задачі досягнення цільової точки, функція винагороди на кожному кроці *t* може бути сконструйована як зважена сума кількох компонентів:

$$R_t = w_{dist} R_{dist} + w_{stable} R_{stable} + w_{effort} R_{effort} + R_{terminal}.$$

де w – вагові коефіцієнти, що визначають пріоритетність кожної складової.

Винагорода за наближення до цілі (R_{dist}). Це щільна винагорода, яка заохочує агента рухатися в напрямку цілі. Вона може бути визначена як від'ємна евклідова відстань до цілі: $R_{dist} = -\| p - p_{target} \|_2$. Таким чином, чим ближче агент до цілі, тим менший штраф він отримує.

Штраф за нестабільність (R_{stable}). Цей компонент штрафує за надмірні кутові відхилення та кутові швидкості, що сприяє плавному та стабільному польоту:

$$R_{stable} = -(\alpha_1(\phi^2 + \theta^2) + \alpha_2(p^2 + q^2 + r^2))$$

де α_1, α_2 – коефіцієнти для балансування штрафів за кути та кутові швидкості.

Штраф за зусилля керування (R_{effort}). Цей компонент штрафує за великі керуючі дії, що заохочує енергоефективні рішення та зменшує знос двигунів:

$$R_{effort} = -\beta \sum_{i=1}^{4} a_i^2$$

де β – ваговий коефіцієнт.

Термінальна винагорода/штраф ($R_{terminal}$). Це велика винагорода або штраф, що надається в кінці епізоду.

1. Успіх. Якщо квадрокоптер досягає цілі (наприклад, відстань до цілі менша за деякий поріг ε), агент отримує велику позитивну винагороду (наприклад, +500), і епізод завершується.

2. Невдача. Якщо квадрокоптер зазнає аварії (наприклад, перевертається, виходить за межі допустимої зони польоту, або час епізоду вичерпано), агент отримує великий негативний штраф (наприклад, -500), і епізод завершується.

Коефіцієнт дисконтування (γ).

Коефіцієнт дисконтування $\gamma \in [0,1]$ визначає важливість майбутніх винагород відносно поточних. Мета агента – максимізувати суму дисконтованих винагород: $\sum_{t=0}^{\infty} \gamma^t R_{t+1}$. Значення γ, близьке до 1 (наприклад, 0.99), означає, що агент є "далекоглядним" і сильно враховує майбутні винагороди, що є важливим для задач з віддаленою ціллю. Значення, близьке до 0, робить агента "короткозорим", що може бути корисним для задач, де важлива лише миттєва реакція.

Таким чином, формалізувавши задачу управління квадрокоптером як MDP, ми створюємо чітку математичну основу, на якій можна будувати та навчати алгоритм підкріплювального навчання. Агент, спостерігаючи за станом s, вибирає дію a, отримує винагороду R і переходить у новий стан s', і цей процес повторюється, доки агент не навчиться оптимальної політики $\pi(a \mid s)$, що максимізує очікувану сумарну дисконтовану винагороду.

2.3. Вибір та обґрунтування алгоритму підкріплювального навчання

Після формалізації задачі управління квадрокоптером як Марковського процесу вирішення (MDP) наступним логічним кроком є вибір конкретного

алгоритму підкріплювального навчання для реалізації адаптивного контролера. Вибір алгоритму є критично важливим, оскільки він визначає ефективність навчання, стабільність поведінки навченого агента та обчислювальні вимоги. Враховуючи особливості нашої задачі, а саме безперервні простори станів та дій, високу динамічність системи та потребу в робастності, вибір має бути спрямований на сучасні алгоритми, здатні впоратися з цими викликами.

Традиційні алгоритми, такі як Q-learning або SARSA, є неефективними в даному контексті, оскільки вони призначені для роботи з дискретними просторами станів і дій та вимагають зберігання таблиці Q-значень, що неможливо при безперервних змінних. Алгоритми, що використовують апроксимацію функцій, такі як Deep Q-Network (DQN), успішно розширили Q-learning на задачі з високовимірними просторами станів (наприклад, зображення), але все ще обмежені дискретним простором дій. Для вирішення цієї проблеми були розроблені алгоритми класу "Актор-Критик" (Actor-Critic), які є природним вибором для задач з безперервним управлінням. У цій архітектурі:

– Актор (Actor), або мережа політики (Policy Network), відповідає за вибір дій. Він приймає на вхід стан s і видає дію a (або параметри розподілу ймовірностей для дій).

– Критик (Critic), або мережа цінності (Value Network), оцінює дії, згенеровані актором. Він приймає на вхід стан s (і іноді дію a) і видає оцінку, наприклад, функцію цінності стану $V(s)$ або функцію цінності пари стан-дія $Q(s,a)$.

Серед сучасних Actor-Critic алгоритмів для безперервного управління найбільш відомими є Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC) та Proximal Policy Optimization (PPO). DDPG та SAC є off-policy алгоритмами, що дозволяє їм бути дуже ефективними з точки зору використання даних завдяки replay buffer. Однак DDPG може бути чутливим до гіперпараметрів і нестабільним у навчанні. SAC, який базується на принципі максимізації ентропії, демонструє відмінну робастність та

ефективність дослідження, що робить його одним з передових алгоритмів. PPO, у свою чергу, є on-policy алгоритмом, але з покращеною ефективністю даних порівняно з попередниками, і він завоював популярність завдяки своїй стабільності, відносній простоті реалізації та високій продуктивності в широкому спектрі задач, особливо в робототехніці.

Для даної магістерської роботи як основний алгоритм було обрано Proximal Policy Optimization (PPO). Цей вибір обґрунтований кількома ключовими перевагами, що роблять його ідеальним кандидатом для розробки адаптивного контролера БПЛА.

1. Стабільність навчання. На відміну від стандартних методів градієнта політики, які можуть робити занадто великі кроки оновлення, що руйнують політику, PPO використовує механізм обмеження (clipping) для оновлень. Це гарантує, що нова політика не буде занадто сильно відрізнятися від старої, що забезпечує більш плавний та стабільний процес навчання.

2. Баланс між ефективністю та простотою. PPO є простішим в реалізації та налаштуванні порівняно з DDPG або SAC, але при цьому часто демонструє співставну або навіть кращу продуктивність. Він досягає кращої ефективності використання даних, ніж прості on-policy алгоритми (A2C), дозволяючи проводити кілька епох оптимізації на одній і тій же порції зібраних даних.

3. Природна підтримка безперервних просторів дій. Архітектура PPO дозволяє актору виводити параметри неперервного розподілу (зазвичай, Гауссового), з якого потім семплюється дія. Це ідеально підходить для управління тягою двигунів квадрокоптера.

4. Сприяння дослідженню. PPO зазвичай включає компонент ентропії в свою функцію втрат, що заохочує політику бути більш стохастичною і, таким чином, краще досліджувати простір дій, уникаючи передчасного збігання до субоптимальних локальних мінімумів.

Детальний опис алгоритму PPO.

PPO є алгоритмом класу Актор-Критик. Його архітектура складається з двох основних нейронних мереж.

Мережа Актора (Політики) $\pi_\theta(a|s)$. Ця мережа, з параметрами θ, приймає на вхід вектор стану s_t і видає на виході параметри розподілу ймовірностей для дій. Для безперервного простору дій це зазвичай середнє значення μ та стандартне відхилення σ для Гауссового розподілу. Фактична дія a_t семплюється з цього розподілу: $a_t \sim \mathcal{N}(\mu_\theta(s_t), \sigma_\theta(s_t))$.

Мережа Критика (Цінності) $V_\phi(s)$. Ця мережа, з параметрами ϕ, приймає на вхід вектор стану s_t і видає одне скалярне значення – оцінку очікуваної сумарної дисконтованої винагороди з цього стану, тобто $V_\phi(s_t)$.

Ключовою інновацією PPO є його цільова функція (objective function) для оновлення мережі актора. Замість прямого використання градієнта політики, PPO оптимізує "суррогатну" цільову функцію. Найпоширеніший варіант PPO використовує механізм кліпінгу (PPO-Clip).

На кожному кроці ітерації збирається набір траєкторій (s_t, a_t, r_t, s_{t+1}) шляхом взаємодії поточної політики $\pi_{\theta_{old}}$ з середовищем. Далі, для кожного часового кроку t розраховується перевага (Advantage) $\hat{A}_t$, яка показує, наскільки кращою була обрана дія a_t у стані s_t порівняно з очікуваною цінністю цього стану. Зазвичай для цього використовується Generalized Advantage Estimation (GAE): $\hat{A}_t = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}$ де $\delta_{t+l} = r_{t+l} + \gamma V_\phi(s_{t+l+1}) - V_\phi(s_{t+l})$ – це помилка часової різниці (TD error), а $\lambda \in [0,1]$ – параметр згладжування GAE.

Далі, розраховується відношення ймовірностей нової та старої політик: $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. Суррогатна цільова функція з кліпінгом для актора має вигляд:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\right]$$

де $\mathbb{E}_t$ означає усереднення по всіх часових кроках у зібраному батчі даних, а ε – це невеликий гіперпараметр (наприклад, 0.2), що визначає розмір "довірчого інтервалу". Функція $\text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)$ обмежує значення $r_t(\theta)$ в

межах $[1-\epsilon, 1+\epsilon]$. Оператор *min* гарантує, що оновлення політики буде песимістичним: воно бере мінімум між необмеженою цільовою функцією та її обмеженою версією. Це ефективно запобігає занадто великим змінам у політиці, які могли б дестабілізувати навчання.

Повна функція втрат, що оптимізується, зазвичай є комбінацією трьох компонентів: $L_t(\theta,\phi) = L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\phi) + c_2 S[\pi_\theta](s_t)$.

1. $L_t^{CLIP}(\theta)$ – цільова функція для актора (з мінусом, оскільки більшість оптимізаторів мінімізують функцію втрат).

2. $L_t^{VF}(\phi) = (V_\phi(s_t) - V_t^{targ})^2$ – функція втрат для критика (зазвичай середньоквадратична помилка), де V_t^{targ} – цільове значення, розраховане як сума винагород до кінця епізоду (Return) або за допомогою GAE: $V_t^{targ} = \hat{A}_t + V_\phi(s_t)$.

3. $S[\pi_\theta](s_t)$ – бонус за ентропію, що заохочує дослідження. c_1 та c_2 – коефіцієнти, що балансують важливість цих трьох компонентів.

Отже, обравши PPO, ми отримуємо надійну, стабільну та потужну основу для розробки адаптивного контролера. Його архітектура та механізми оптимізації добре підходять для складного та динамічного середовища, яким є управління квадрокоптером, і дозволяють очікувати на отримання якісних та робастних результатів.

РОЗДІЛ 3.

РОЗРОБКА АДАПТИВНОГО АЛГОРИТМУ

3.1. Архітектура алгоритму

Розробка адаптивного алгоритму управління на основі підкріплювального навчання вимагає проектування цілісної системи, що включає в себе не тільки ядро самого RL-алгоритму, але й компоненти для взаємодії з симуляційним середовищем та обробки даних. Цей розділ детально описує загальну архітектуру системи, структуру нейронних мереж, що лежать в основі агента, та схему їхньої взаємодії. Обраний алгоритм PPO (Proximal Policy Optimization) визначає ключові елементи цієї архітектури, зокрема, використання паралельних мереж Актора та Критика.

3.1.1. Загальна архітектура системи

Запропонована система складається з трьох ключових, логічно відокремлених компонентів: Середовище, Агент та Інтерфейс взаємодії. Така модульна структура дозволяє гнучко змінювати або вдосконалювати кожен компонент незалежно від інших.

Середовище (Environment). Цей компонент відповідає за симуляцію динаміки квадрокоптера та його оточення. Він реалізує математичну модель, описану в розділі 2.1, і є джерелом даних для навчання агента. Середовище виконує дві основні функції:

1. reset(). Повертає систему у початковий стан на початку кожного епізоду навчання. Це може бути фіксована стартова позиція або випадкова, що сприяє кращому узагальненню політики.

2. step(action). Приймає на вхід дію action, згенеровану агентом, і симулює один крок у часі. Після цього повертає кортеж (next_state, reward, done, info), де next_state – новий стан системи, reward – винагорода за виконану дію, done – булевий прапорець, що вказує на завершення епізоду, а info – додаткова діагностична інформація.

Агент (Agent). Це центральний компонент, що реалізує логіку алгоритму PPO. Він містить у собі нейронні мережі Актора та Критика і відповідає за прийняття рішень та процес навчання. Агент виконує наступні задачі:

1. Вибір дії. На основі поточного стану середовища, отриманого від інтерфейсу, Актор генерує дію.

2. Навчання. Періодично, після накопичення достатньої кількості даних (траєкторій), агент використовує їх для оновлення ваг своїх нейронних мереж (Актора та Критика) згідно з цільовою функцією PPO.

Інтерфейс взаємодії (Interaction Interface). Цей компонент є проміжним шаром, що організовує цикл взаємодії між Агентом та Середовищем. Він керує процесом збору даних (rollout), зберігає траєкторії (state, action, reward, next_state, done) у буфері та ініціює фазу навчання агента.

Схематично взаємодію компонентів можна представити наступним чином:

1. Інтерфейс ініціює новий епізод, викликаючи reset() у Середовища, і отримує початковий стан s_0.

2. Інтерфейс передає стан s_t Агенту.

3. Агент (його Актор) обробляє s_t і повертає дію a_t.

4. Інтерфейс передає дію a_t до Середовища через функцію step().

5. Середовище оновлює свій стан, розраховує винагороду і повертає $s_{t+1}, r_t, done$.

6. Інтерфейс зберігає цей перехід ($s_t, a_t, r_t, s_{t+1}, done$) і повторює цикл з кроку 2.

7. Після накопичення заданої кількості переходів, Інтерфейс передає зібрані дані Агенту для виконання кроку оптимізації.

3.1.2. Архітектура нейронних мереж

В основі PPO-агента лежать дві глибокі нейронні мережі: мережа Актора та мережа Критика. Для нашої задачі пропонується використовувати архітектуру з окремими мережами, оскільки це може підвищити стабільність навчання. Актор відповідає за вибір оптимальної дії в поточному стані, тоді як

Критик оцінює, наскільки "хорошим" є цей стан. Структура цих мереж, розроблена для вирішення задачі управління БПЛА, наведена на рис. 3.1.

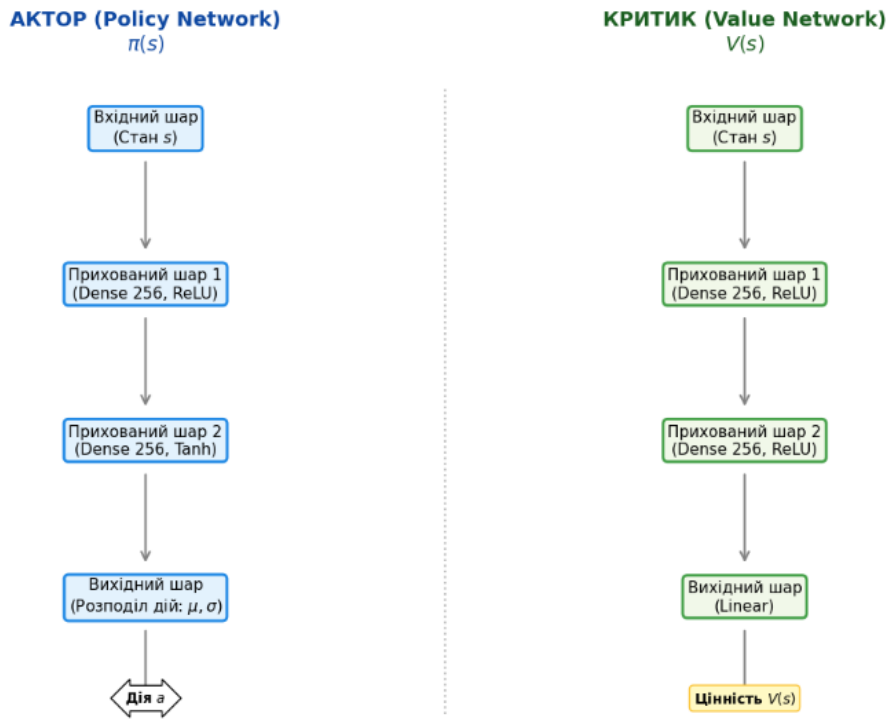

Рисунок 3.1 – Архітектура нейронних мереж Актора (зліва) та Критика (справа)

Мережа Актора (Policy Network).

Вхідний шар. Приймає вектор стану s, розмірність якого дорівнює 15 (12 параметрів динаміки БПЛА + 3 параметри відносної позиції до цілі), як визначено в розділі 2.2. Перед подачею на вхід дані стану нормалізуються для покращення збіжності.

Приховані шари. Пропонується використовувати дві або три повнозв'язні (Dense) шари. Типова архітектура може бути [256, 256] нейронів. В якості функції активації для прихованих шарів рекомендується використовувати гіперболічний тангенс (Tanh), оскільки його вихідний діапазон [-1, 1] часто є корисним для стабілізації навчання в задачах управління.

Вихідний шар. Оскільки простір дій є безперервним, вихідний шар Актора генерує параметри для стохастичної політики. Для 4-вимірного простору дій (керування чотирма роторами) вихідний шар буде мати 8 нейронів:

– 4 нейрони для середнього значення (μ). Ці нейрони матимуть активацію Tanh, щоб їх вихід був у діапазоні [-1, 1]. Це значення потім буде масштабовано до реального діапазону керуючих сигналів двигунів.

– 4 нейрони для стандартного відхилення (σ). Ці нейрони будуть використовувати функцію активації softplus (або експоненту) для гарантії того, що стандартне відхилення завжди буде позитивним. Це значення визначає ступінь дослідження (exploration) агента.

Мережа Критика (Value Network).

Вхідний шар. Аналогічно до Актора, приймає нормалізований вектор стану *s* розмірністю 15.

Приховані шари. Архітектура може бути симетричною до архітектури Актора (наприклад, [256, 256] нейронів) з функцією активації Tanh або ReLU. Симетричність не є обов'язковою, але є поширеною практикою.

Вихідний шар. Складається з одного нейрона з лінійною функцією активації. Цей нейрон видає одне скалярне значення – оцінку функції цінності стану $V(s)$, яка представляє очікувану сумарну дисконтовану винагороду з поточного стану.

3.1.3. Схема взаємодії та потік даних

Процес навчання і взаємодії організований у вигляді ітеративного циклу, що складається з фази збору даних та фази оновлення.

Фаза 1. Збір даних (Rollout Phase).

1. Агент, використовуючи поточну політику $\pi_{\theta_{old}}$, взаємодіє з Середовищем протягом *N* часових кроків (наприклад, $N = 2048$).

2. На кожному кроці *t*:

– актор отримує стан s_t і видає параметри Гауссового розподілу $\mu(s_t)$ та $\sigma(s_t)$;

– дія a_t семплюється з цього розподілу: $a_t \sim \mathcal{N}(\mu(s_t), \sigma(s_t))$;

– дія передається в Середовище, яке повертає $s_{t+1}, r_t, done$.

– перехід $(s_t, a_t, r_t, s_{t+1}, V(s_t), \log\pi(a_t \mid s_t))$ зберігається в буфері. $V(s_t)$ – це оцінка критика, а $\log\pi(a_t \mid s_t)$ – логарифм ймовірності дії, які знадобляться для навчання.

Фаза 2. Оновлення (Update Phase).

1. Після заповнення буфера на N кроків, для всіх збережених переходів розраховуються переваги (Advantages) $\hat{A}_t$ та цільові значення для критика V_t^{targ} за допомогою методу GAE (Generalized Advantage Estimation).

2. Дані з буфера перемішуються і розбиваються на міні-батчі.

3. Протягом *K* епох (наприклад, *K* = 10):

– для кожного міні-батчу розраховуються градієнти функцій втрат для Актора та Критика;

– ваги нейронних мереж θ та ϕ оновлюються за допомогою оптимізатора (наприклад, Adam).

4. Після завершення *K* епох оновлення, буфер очищується, нова політика π_θ стає старою $\pi_{\theta_{old}}$, і система повертається до Фази 1.

Така архітектура створює міцний фундамент для реалізації адаптивного контролера, що здатний навчатися складній динаміці управління квадрокоптером та оптимізувати свою поведінку для досягнення поставлених цілей.

3.2. Стратегії навчання

Ефективність та успішність процесу навчання RL-агента значною мірою залежать від обраних стратегій навчання, які включають методи збору даних, механізми їхньої обробки та ретельний вибір гіперпараметрів. Розглянуті нижче стратегії спрямовані на забезпечення стабільної збіжності, покращення ефективності використання даних та підвищення узагальнюючої здатності навченої політики.

3.2.1. Методи збору даних (досвіду)

Збір даних, або досвіду (experience), є першим етапом у кожній ітерації циклу навчання. Оскільки PPO є on-policy алгоритмом, дані, зібрані за допомогою старої політики, використовуються для її оновлення, після чого вони стають нерелевантними і відкидаються. Це накладає певні вимоги на процес збору.

Паралельний збір даних. Для прискорення процесу збору даних та підвищення його різноманітності використовується підхід з паралельними середовищами (vectorized environments). Замість одного екземпляра симулятора, запускається декілька (наприклад, 8 або 16) незалежних копій середовища. Агент одночасно подає на вхід батч станів з усіх середовищ і отримує батч дій. Це дозволяє за той самий час зібрати значно більше різноманітного досвіду, що зменшує кореляцію між послідовними переходами і стабілізує навчання.

Довжина траєкторії (Rollout Length). На кожній ітерації агент збирає фіксовану кількість кроків взаємодії (rollout length) з середовища, наприклад, $N=2048$ кроків. Після цього дані збираються в один великий батч і використовуються для оновлення. Вибір довжини траєкторії є компромісом: довші траєкторії дозволяють точніше оцінити переваги (advantages), але вимагають більше пам'яті та затримують оновлення мережі. Коротші траєкторії дозволяють частіше оновлювати політику, але вносять більше шуму в оцінки.

Управління завершенням епізодів. Епізоди можуть завершуватися з двох причин: досягнення мети (успіх) або невдача (аварія, вихід за межі). Окрім цього, вводиться максимальна довжина епізоду (наприклад, 1000 кроків), щоб уникнути зациклення агента в субоптимальних станах. Якщо епізод обривається через досягнення максимальної довжини, а не через невдачу, це потрібно враховувати при розрахунку цільових значень для критика, щоб уникнути невірного "покарання" за хороший, але довгий епізод. Для цього при розрахунку цільового значення для останнього стану використовується оцінка критика (value bootstrapping), що імітує продовження епізоду.

3.2.2. Використання Replay Buffer (у контексті PPO)

Хоча класичний PPO є on-policy алгоритмом і не використовує replay buffer у тому сенсі, як це роблять off-policy алгоритми (наприклад, DDPG або SAC, де дані можуть зберігатися і використовуватися повторно протягом багатьох оновлень), він все ж використовує тимчасовий буфер для зберігання даних, зібраних протягом однієї фази збору (rollout).

Зібрані N кроків досвіду з усіх паралельних середовищ агрегуються в єдиний буфер. Потім, під час фази оновлення, агент проводить кілька епох (K) градієнтного спуску, використовуючи дані з цього буфера. На кожній епосі дані з буфера перемішуються і розбиваються на міні-батчі. Такий підхід, де одні й ті ж дані використовуються для кількох оновлень, значно підвищує ефективність використання даних порівняно з алгоритмами, що роблять лише одне оновлення на зібраний батч (наприклад, A2C). Це є однією з ключових переваг PPO, що дозволяє досягти кращого балансу між стабільністю on-policy методів та ефективністю off-policy методів.

3.2.3. Параметри навчання

Ретельний вибір гіперпараметрів є вирішальним для успішного навчання. Нижче наведено ключові параметри та їх обґрунтування для задачі управління БПЛА.

Швидкість навчання (Learning Rate). Це один з найважливіших параметрів. Занадто висока швидкість може призвести до нестабільності, а занадто низька – до дуже повільної збіжності. Рекомендується використовувати відносно низьку швидкість навчання, наприклад, 3×10^{-4}, з можливим використанням розкладу (learning rate schedule), де швидкість поступово зменшується протягом навчання. Це дозволяє робити більші кроки на початку, коли політика далека від оптимальної, і менші, більш точні кроки, коли вона наближається до збіжності.

Коефіцієнт дисконтування (γ). Цей параметр визначає, наскільки агент "цінує" майбутні винагороди. Для задач управління, де мета знаходиться на відстані і вимагає послідовності дій, використовується високе значення γ,

наприклад, 0.99. Це заохочує агента планувати свої дії на довгострокову перспективу.

Параметр кліпінгу PPO (ε). Цей параметр, як правило, встановлюється в діапазоні [0.1, 0.3], найчастіше використовується значення 0.2. Він контролює, наскільки сильно нова політика може відхилятися від старої на одній ітерації оновлення, що є ключовим для стабільності PPO.

Коефіцієнт GAE (λ). Параметр Generalized Advantage Estimation, зазвичай встановлюється близько до 1 (наприклад, 0.95). Він контролює зміщення (bias) та дисперсію (variance) в оцінці переваг. Значення, близьке до 1, зменшує зміщення, але може збільшити дисперсію, враховуючи інформацію з довшої послідовності кроків.

Розмір міні-батчу (Minibatch Size). Зібраний батч даних розміром N ділиться на менші міні-батчі для стохастичного градієнтного спуску. Типовий розмір міні-батчу – 64.

Кількість епох оновлення (Update Epochs). Це кількість разів, коли агент проходить по зібраному буферу даних для оновлення мереж. Зазвичай встановлюється в діапазоні від 4 до 10. Більша кількість епох покращує використання даних, але може призвести до перенавчання на поточному батчі та порушення стабільності, яку забезпечує PPO.

Коефіцієнти функції втрат:

– коефіцієнт критика (c_1) зазвичай встановлюється на 0.5, щоб збалансувати оновлення мереж Актора та Критика;

– коефіцієнт ентропії (c_2) має невелике позитивне значення (наприклад, 0.01) і заохочує дослідження, запобігаючи передчасному колапсу політики до детермінованої. Цей коефіцієнт також може зменшуватися з часом.

Застосування цих стратегій та ретельне налаштування параметрів дозволяють створити надійний та ефективний процес навчання, який здатний привести до розробки високопродуктивного та адаптивного контролера для БПЛА.

3.3. Адаптивність алгоритму

Основною метою даної роботи є не просто створення ефективного контролера, а розробка алгоритму, здатного до адаптації. Адаптивність означає здатність системи зберігати високу продуктивність або швидко відновлювати її при зміні умов функціонування, будь то внутрішні зміни в динаміці БПЛА або зовнішні збурення середовища. В контексті підкріплювального навчання адаптивність досягається не через явні математичні моделі цих змін, а через навчання робастної політики, яка узагальнює досвід, отриманий у різноманітних умовах. Нижче описані стратегії, що впроваджуються для забезпечення адаптивності розробленого алгоритму.

3.3.1. Навчання через Доменну Рандомізацію (Domain Randomization)

Доменна рандомізація є однією з найпотужніших технік для навчання робастних політик, які добре працюють у реальному світі, незважаючи на розбіжності між симуляцією та реальністю (sim-to-real gap). Суть методу полягає у навмисному внесенні варіативності в параметри симуляційного середовища під час навчання. Замість того, щоб навчати агента в одному ідеалізованому середовищі, ми навчаємо його в цілому ансамблі середовищ, де ключові параметри випадково змінюються на початку кожного епізоду.

Для нашої задачі управління БПЛА доменна рандомізація буде застосована до наступних параметрів.

Динамічні параметри БПЛА:

– маса (m). На початку кожного епізоду маса апарату буде випадково обиратися з певного діапазону, наприклад, $m \in [m_{base} - 15\%, m_{base} + 15\%]$. Це симулює зміну корисного навантаження (наприклад, доставка вантажу, встановлення додаткового обладнання);

– моменти інерції (I_x, I_y, I_z). Аналогічно до маси, моменти інерції будуть варіюватися, що відображає зміну розподілу маси;

– ефективність двигунів. Коефіцієнт тяги (k_f) для кожного з чотирьох двигунів може бути незначно змінений ($k_{f,i} = k_f \cdot (1 + \delta_i)$), де δ_i – невелика

випадкова величина. Це симулює невелику деградацію або нерівномірність роботи двигунів.

Параметри зовнішнього середовища:

– вітрові збурення. На кожному кроці симуляції до динаміки апарату може додаватися випадковий вектор сили, що імітує пориви вітру. Сила та напрямок цього вектора можуть змінюватися з часом, симулюючи турбулентність;

– початкові умови. Кожен епізод буде починатися з випадковими невеликими відхиленнями в початковому положенні, орієнтації та лінійних/кутових швидкостях. Це змушує агента навчатися стабілізувати апарат з будь-якого початкового стану, а не тільки з ідеального стану зависання.

Навчаючи агента в таких різноманітних умовах, ми змушуємо нейронну мережу вивчати не просто одну конкретну стратегію управління, а загальні принципи стабілізації та навігації, які є інваріантними до варіацій у динаміці та середовищі. Політика, навчена таким чином, стає менш чутливою до конкретних значень параметрів і демонструє значно кращу робастність та здатність до узагальнення, що і є суттю адаптивності.

3.3.2. Адаптація через функцію винагороди

Функція винагороди, описана в розділі 2.2, вже містить компоненти, що неявно сприяють адаптивності.

Штраф за зусилля керування (R_{effort}). Штрафуючи за великі керуючі дії, ми заохочуємо агента знаходити енергоефективні рішення. Коли, наприклад, маса БПЛА збільшується, агент, що намагається мінімізувати цей штраф, буде шукати більш плавні траєкторії замість різких маневрів, що є формою адаптивної поведінки.

Штраф за нестабільність (R_{stable}). Штраф за великі кутові відхилення та швидкості змушує агента активно протидіяти будь-яким збуренням (наприклад, вітру), щоб підтримувати стабільний політ. Таким чином, політика стає стійкою до зовнішніх впливів.

3.3.3. Потенційні напрямки для подальшого підвищення адаптивності

Хоча доменна рандомізація є основним інструментом у даній роботі, існують і більш просунуті методи, які варто згадати як перспективні напрямки.

Мета-навчання (Meta-Reinforcement Learning). Цей підхід спрямований на навчання "алгоритму навчання". Агент навчається на наборі різних завдань (наприклад, політ з різною масою) таким чином, щоб він міг дуже швидко (за кілька кроків) адаптуватися до нового, раніше не баченого завдання. Це дозволяє досягти адаптації в режимі реального часу, а не лише загальної робастності.

Системна ідентифікація в циклі. Можна розробити гібридний підхід, де окремий модуль (наприклад, рекурентна нейронна мережа) намагається в режимі реального часу оцінити поточні параметри системи (наприклад, масу або силу вітру) і подавати цю оцінку як додатковий вхід до політики RL-агента. Це дозволить агенту явно враховувати зміни в динаміці та приймати більш обґрунтовані рішення.

Таким чином, комбінація робастного навчання через доменну рандомізацію та ретельно розробленої функції винагороди дозволяє створити алгоритм управління, який не просто виконує завдання в ідеальних умовах, а демонструє високий ступінь адаптивності до реалістичних викликів. Навчена політика буде здатною впоратися зі змінами корисного навантаження, протистояти поривам вітру та компенсувати невеликі відхилення в роботі апаратури, що є ключовим кроком на шляху до створення по-справжньому автономних БПЛА.

РОЗДІЛ 4.

РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТИ

4.1. Вибір програмного забезпечення

Для успішної реалізації розробленого адаптивного алгоритму управління БПЛА та проведення всебічних експериментів було обрано комплекс сучасних, відкритих та широко використовуваних програмних інструментів. Вибір кожного компонента стеку технологій ґрунтувався на критеріях гнучкості, продуктивності, наявності якісної документації та активної спільноти розробників. Основним середовищем розробки є мова програмування Python завдяки її багатій екосистемі бібліотек для наукових обчислень та машинного навчання.

4.1.1. Мова програмування та основні бібліотеки

Python обрано як основну мову програмування завдяки її простоті, читабельності та величезній кількості бібліотек, спеціалізованих для вирішення завдань у галузі штучного інтелекту. Python дозволяє швидко прототипувати ідеї та легко інтегрувати різні компоненти системи.

NumPy – фундаментальна бібліотека для наукових обчислень в Python. Вона буде використовуватися для всіх векторних та матричних операцій, таких як маніпуляції з векторами станів та дій, розрахунки винагород та обробка даних у буфері.

Matplotlib/Seaborn. Ці бібліотеки будуть використовуватися для візуалізації результатів експериментів, зокрема для побудови графіків кривих навчання (залежність винагороди від епізоду), траєкторій польоту БПЛА, а також для аналізу зміни параметрів стану з часом.

4.1.2. Симулятор середовища

Симуляційне середовище є серцем експериментальної частини, оскільки саме в ньому відбувається навчання та тестування RL-агента. Безпечне та швидке симулювання є критично важливим, оскільки навчання RL-агентів вимагає мільйонів кроків взаємодії. Для даної роботи було обрано PyBullet як основний фізичний рушій для симуляції. Це Python-модуль для фізичного

рушія Bullet Physics SDK, який підтримує симуляцію динаміки твердих тіл, виявлення зіткнень, моделювання з'єднань та багато іншого. Переваги PyBullet для нашої задачі:

– висока продуктивність. PyBullet є одним з найшвидших доступних симуляторів, що дозволяє значно прискорити процес навчання;

– простота інтеграції. Він надає простий та інтуїтивно зрозумілий Python API для створення середовищ, завантаження моделей (у форматах URDF, SDF) та керування симуляцією;

– гнучкість, тобто дозволяє легко реалізувати динамічну модель квадрокоптера, описану в розділі 2.1, та впроваджувати доменну рандомізацію (зміну маси, інерції, симуляцію вітру) програмним шляхом.

Для стандартизації взаємодії між агентом та середовищем симулятор буде реалізовано у вигляді кастомного середовища, що відповідає інтерфейсу OpenAI Gym (тепер Gymnasium). Цей стандартний API (з методами step, reset та іншими) є де-факто стандартом у спільноті розробників RL і забезпечує сумісність з більшістю фреймворків.

4.1.3. Фреймворк для підкріплювального навчання

Замість реалізації алгоритму PPO з нуля, що є трудомістким і схильним до помилок процесом, було вирішено використовувати спеціалізований фреймворк. Вибір зупинено на Stable Baselines3 (SB3). Stable Baselines3 – це набір надійних реалізацій сучасних алгоритмів підкріплювального навчання на базі PyTorch. Його переваги:

– SB3 пропонує високоякісні, добре протестовані та оптимізовані реалізації популярних алгоритмів, включаючи PPO;

– фреймворк надає простий та уніфікований інтерфейс для навчання та тестування агентів. Для навчання агента PPO на власному Gym-сумісному середовищі достатньо кількох рядків коду;

– SB3 дозволяє легко налаштовувати архітектуру нейронних мереж (Актора та Критика), змінювати гіперпараметри навчання та інтегрувати власні політики або компоненти;

– SB3 легко інтегрується з такими інструментами, як TensorBoard, що дозволяє в реальному часі відстежувати процес навчання (графіки винагороди, функції втрат, ентропії тощо).

4.1.4. Фреймворк для глибокого навчання

Основою для Stable Baselines3 є PyTorch. Хоча безпосередня робота з PyTorch буде мінімізована завдяки використанню SB3, розуміння його основ є важливим, особливо для налаштування архітектури нейронних мереж.

PyTorch є один з провідних фреймворків для глибокого навчання, відомий своєю гнучкістю та динамічним обчислювальним графом. Він забезпечує всі необхідні інструменти для побудови нейронних мереж, обчислення градієнтів та оптимізації моделей. Використання PyTorch "під капотом" SB3 гарантує високу продуктивність обчислень, особливо при використанні графічних процесорів (GPU).

Таким чином, обраний стек програмного забезпечення (Python, PyBullet, OpenAI Gym, Stable Baselines3, PyTorch) є потужною, гнучкою та сучасною комбінацією для реалізації та дослідження адаптивного алгоритму управління БПЛА. Він дозволяє зосередитися на аспектах моделювання, дизайну винагород та аналізу результатів, покладаючись на перевірені та оптимізовані інструменти для реалізації складних обчислювальних компонентів.

4.2. Навчання алгоритму

Процес навчання є центральним етапом реалізації, де розроблений агент шляхом ітеративної взаємодії з симуляційним середовищем навчається виконувати поставлені задачі. Цей процес вимагає ретельної підготовки, постійного моніторингу та, за необхідності, оптимізації параметрів для досягнення найкращих результатів. Нижче описано процедуру навчання, що використовується в даній роботі.

4.2.1. Підготовка до навчання

Перед запуском основного циклу навчання необхідно виконати кілька підготовчих кроків.

1. Ініціалізація середовища. Створюється екземпляр розробленого Gym-сумісного середовища на базі PyBullet. Для прискорення збору даних використовується векторизована обгортка DummyVecEnv або SubprocVecEnv з бібліотеки Stable Baselines3, яка дозволяє одночасно запускати кілька паралельних симуляцій. Це не тільки прискорює процес, але й покращує стабільність навчання за рахунок збору більш різноманітного досвіду на кожному кроці.

2. Ініціалізація моделі агента. Створюється модель PPO з використанням класу PPO з Stable Baselines3. На цьому етапі задаються всі ключові гіперпараметри, визначені в розділі 3.2, а також архітектура нейронних мереж Актора та Критика. Приклад ініціалізації:

```
model = PPO(
policy='MlpPolicy',  # Використання багатошарового перцептрону
env=vec_env,         # Векторизоване середовище
learning_rate=3e-4,  # Швидкість навчання
n_steps=2048,        # Довжина траєкторії для одного оновлення
batch_size=64,       # Розмір міні-батчу
n_epochs=10,         # Кількість епох оновлення на зібраних даних
gamma=0.99,          # Коефіцієнт дисконтування
gae_lambda=0.95,     # Параметр для GAE
clip_range=0.2,      # Параметр кліпінгу PPO
ent_coef=0.01,       # Коефіцієнт ентропії
verbose=1,           # Рівень деталізації логів
tensorboard_log="./ppo_tensorboard/"  # Шлях для збереження логів TensorBoard)
```

3. Налаштування моніторингу. Навчання RL-агента може тривати від кількох годин до кількох днів, тому важливо мати інструменти для відстеження прогресу в реальному часі. Stable Baselines3 має вбудовану інтеграцію з TensorBoard. При ініціалізації моделі вказується шлях для збереження логів, що дозволяє візуалізувати ключові метрики під час навчання.

4.2.2. Процедура ітеративного навчання

Основний процес навчання запускається викликом методу model.learn(). Цей метод інкапсулює ітеративний цикл, описаний в розділі 3.1.

1. Запуск. Викликається метод model.learn(total_timesteps=N), де N – це загальна кількість кроків взаємодії з середовищем, після якої навчання припиняється (наприклад, 1,000,000 або 5,000,000).

2. Цикл "Збір-Оновлення". Всередині методу learn() виконується безперервний цикл:

– фаза збору (Rollout), в якій агент взаємодіє з паралельними середовищами протягом n_steps кроків, збираючи траєкторії (s, a, r, s', done);

– фаза оновлення (Update). Після збору даних агент розраховує переваги (advantages) та цільові значення, а потім протягом n_epochs ітеративно оновлює ваги своїх нейронних мереж на міні-батчах, використовуючи оптимізатор Adam.

3. Збереження моделі. Протягом навчання важливо періодично зберігати проміжні версії моделі (чекпоінти). Це дозволяє відновити навчання у разі збою, а також зберегти найкращу модель для подальшого аналізу. Stable Baselines3 надає зручні інструменти (callbacks), які дозволяють автоматично зберігати модель, наприклад, кожні 10,000 кроків або при досягненні нового рекорду середньої винагороди.

Навчання проводиться з використанням графічного процесора (GPU), що значно прискорює обчислення, пов'язані з прямим та зворотним поширенням сигналу в нейронних мережах під час фази оновлення.

4.2.3. Моніторинг прогресу навчання

За допомогою TensorBoard відстежуються ключові метрики, що дозволяють оцінити якість та динаміку процесу навчання:

– rollout/ep_rew_mean (середня винагорода за епізод). Найважливіший показник. Його стабільне зростання свідчить про те, що агент навчається виконувати завдання все краще;

– train/loss (загальна функція втрат). Включає в себе втрати політики, втрати функції цінності та ентропію. Її зниження вказує на те, що процес оптимізації проходить коректно;

– train/policy_loss та train/value_loss. Втрати для Актора та Критика відповідно. Дозволяють окремо аналізувати, як навчається кожна з мереж;

– train/entropy_loss (показник ентропії політики). Висока ентропія означає високий рівень дослідження (exploration). Очікується, що з часом, коли агент знаходить хорошу стратегію, ентропія буде поступово зменшуватися;

– rollout/ep_len_mean (середня довжина епізоду). Зростання цього показника (до досягнення максимального ліміту) зазвичай є позитивним знаком, оскільки свідчить про те, що агент довше уникає невдач (аварій).

Аналіз цих графіків дозволяє вчасно виявити проблеми, такі як дивергенція навчання, застрягання в локальному оптимумі або неправильно налаштована функція винагороди. Найважливішою метрикою, що характеризує успішність навчання, є середня сумарна винагорода, отримана агентом за епізод (rollout/ep_rew_mean). Її динаміка відображає здатність агента покращувати свою стратегію з часом. Результат процесу навчання візуалізовано на рис. 4.1.

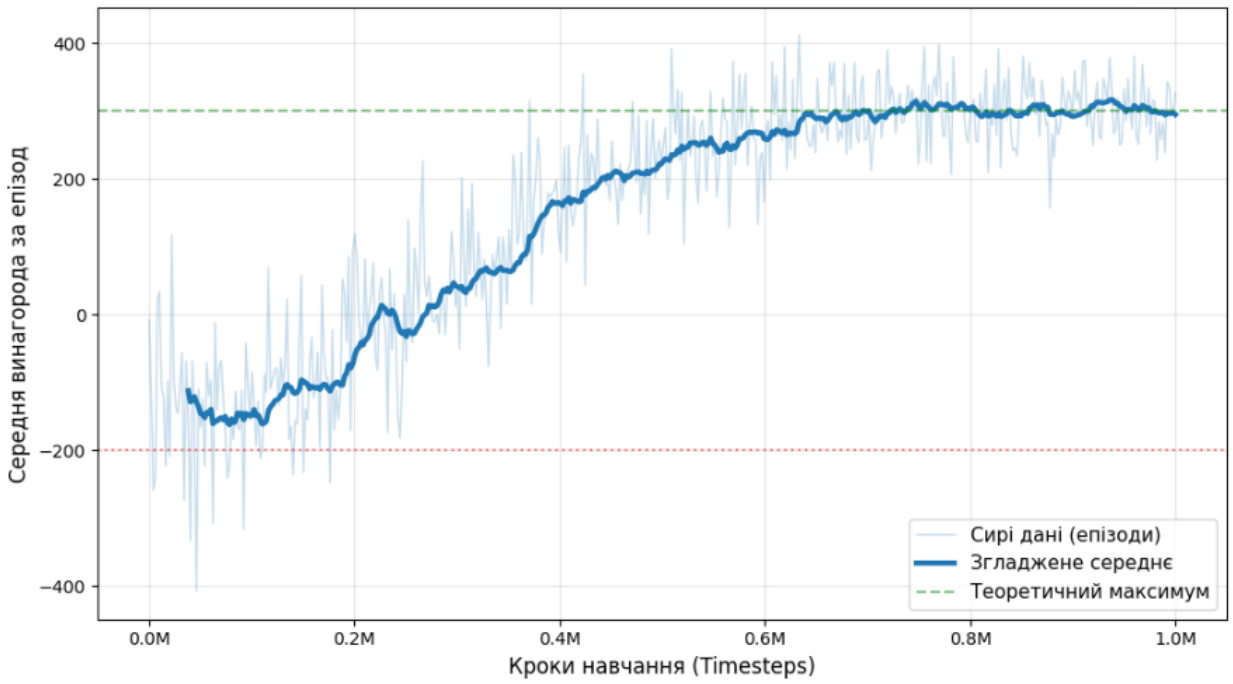

Рисунок 4.1 – Крива навчання агента: залежність середньої винагороди від кількості кроків навчання

Як видно з наведеного графіка, процес навчання можна розділити на кілька етапів. На початковому етапі (приблизно до X кроків) спостерігається стрімке зростання винагороди, що свідчить про те, що агент засвоює базові принципи стабілізації та уникнення падінь. Надалі крива стає більш пологою,

що відповідає етапу точного налаштування політики для мінімізації помилок та оптимізації енерговитрат. Вихід кривої на "плато" (стабільний рівень) сигналізує про збіжність алгоритму та завершення процесу навчання.

4.2.4. Оптимізація гіперпараметрів

Хоча початкові гіперпараметри обираються на основі загальних рекомендацій та попереднього досвіду, для досягнення максимальної продуктивності може знадобитися їхня тонка настройка. Процес оптимізації є емпіричним і включає в себе проведення серії коротких навчальних сесій з різними значеннями ключових параметрів (наприклад, learning_rate, gamma, ent_coef) та порівняння кривих навчання для вибору найкращої конфігурації для фінального, тривалого навчання.

4.3. Експериментальні сценарії

Для всебічної оцінки продуктивності, робастності та адаптивності розробленого на основі PPO контролера було розроблено серію експериментальних сценаріїв. Ці сценарії імітують різноманітні польотні завдання та умови, від базових маневрів до складних ситуацій зі збуреннями та зміною динаміки. Кожен сценарій розроблений для тестування специфічних аспектів поведінки навченого агента. Тестування проводиться на навченій моделі в режимі оцінювання (без подальшого навчання та дослідження), що дозволяє об'єктивно оцінити її фінальну продуктивність.

4.3.1. Базові сценарії

Ця група сценаріїв призначена для перевірки здатності агента виконувати фундаментальні задачі управління, що є основою для більш складних місій.

Сценарій 1. Стабілізація та зависання (Hovering)

Мета: перевірити здатність агента утримувати квадрокоптер у стабільному горизонтальному положенні у заданій точці простору.

Умови: квадрокоптер ініціалізується з невеликими випадковими відхиленнями від цільової точки (x, y, z) та з ненульовими початковими лінійними та кутовими швидкостями.

Оцінюється середньоквадратичне відхилення (RMSE) від цільової позиції протягом фіксованого проміжку часу. Також аналізуються коливання кутів крену та тангажу. Успішним вважається сценарій, якщо агент швидко гасить початкові збурення і стабілізує апарат з мінімальними осциляціями.

Сценарій 2. Досягнення цільової точки (Waypoint Navigation)

Мета: оцінити здатність агента керувати БПЛА з початкової точки до заданої кінцевої точки в тривимірному просторі.

Умови: квадрокоптер стартує з точки А і повинен досягти точки Б, яка знаходиться на значній відстані.

Ключовими метриками оцінки є:

1. Час досягнення цілі: час, за який БПЛА входить у цільову зону (сферу малого радіуса навколо цілі).

2. Точність: фінальна відстань до цілі.

3. Плавність траєкторії: аналіз зміни швидкостей та прискорень.

Експеримент повторюється для різних початкових та кінцевих точок для перевірки узагальнюючої здатності.

Сценарій 3. Слідування траєкторії (Trajectory Tracking)

Мета: перевірити здатність агента точно слідувати заздалегідь визначеній динамічній траєкторії (наприклад, коло, вісімка, синусоїда).

Умови: на кожному кроці часу цільова точка для агента оновлюється відповідно до заданої траєкторії.

Вимірюється середня та максимальна помилка слідування (відстань від поточної позиції БПЛА до найближчої точки на ідеальній траєкторії). Візуалізуються реальна та бажана траєкторії для якісного аналізу.

4.3.2. Сценарії з перешкодами

Ця група сценаріїв тестує здатність агента до реактивного планування та уникнення зіткнень, що є критично важливим для автономних польотів. Для цього необхідно буде розширити простір станів, додавши інформацію про перешкоди (наприклад, через лідари або датчики відстані).

Сценарій 4. Уникнення статичних перешкод

Мета: оцінити здатність агента досягати цільової точки, облітаючи при цьому нерухомі перешкоди (наприклад, стовпи, стіни).

Умови: у симуляційному середовищі розміщуються кілька об'єктів простої форми на шляху до цілі.

Вимірюється відсоток успішних спроб (досягнення цілі без зіткнення). Аналізується мінімальна відстань до перешкод під час польоту та оптимальність обраної траєкторії обльоту.

4.3.3. Адаптивні сценарії

Це ключова група експериментів, розроблена для прямої демонстрації адаптивності навченого контролера, який тренувався з використанням доменної рандомізації.

Сценарій 5. Зміна маси корисного навантаження

Мета: перевірити, як агент справляється з раптовою зміною маси, що не була точно представлена під час навчання, але знаходиться в межах або трохи за межами діапазону рандомізації.

Умови: квадрокоптер виконує завдання зависання або слідування траєкторії. Під час польоту його маса програмно змінюється (наприклад, збільшується на 20%, імітуючи підхоплення вантажу).

Аналізується поведінка системи одразу після зміни маси: максимальне просідання по висоті, час відновлення стабільного польоту. Порівнюється продуктивність адаптивного агента з агентом, навченим без доменної рандомізації.

Сценарій 6. Протидія постійному вітру

Мета: оцінити стійкість контролера до постійних зовнішніх збурень.

Умови: під час виконання завдання досягнення цільової точки на квадрокоптер починає діяти постійна сила, що імітує вітер певного напрямку та сили.

Вимірюється здатність агента компенсувати знос і досягти цілі. Аналізується статична помилка позиції під час зависання в умовах вітру (наскільки БПЛА відхиляється від цілі, щоб компенсувати вітер).

Сценарій 7. Імітація часткової відмови двигуна

Мета: перевірити робастність алгоритму до внутрішніх збоїв.

Умови: під час стабільного польоту ефективність одного з двигунів програмно знижується на 10-15%. Це симулює часткову деградацію або пошкодження.

Аналізується здатність агента перерозподілити тягу між рештою двигунів для збереження стабільності та керованості. Вимірюється час відновлення та здатність продовжувати виконання завдання, хоч і з можливою втратою точності.

Проведення цих експериментів дозволить отримати повну картину можливостей розробленого алгоритму, кількісно оцінити його переваги та продемонструвати, що навчання з використанням доменної рандомізації дійсно призводить до створення по-справжньому адаптивного та надійного контролера.

4.4. Порівняльний аналіз

Для об'єктивної оцінки ефективності розробленого адаптивного контролера на основі підкріплювального навчання (далі – RL-контролер) було проведено його порівняльний аналіз з традиційним методом управління. В якості базового контролера (baseline) було обрано добре налаштований ПІД-регулятор (Пропорційно-Інтегрально-Диференціальний), який є промисловим стандартом для управління квадрокоптерами.

ПІД-регулятор був реалізований у вигляді каскадної структури: зовнішній контур керував положенням та швидкістю, генеруючи бажані кути крену та тангажу, а внутрішній контур стабілізував кутову орієнтацію. Коефіцієнти ПІД-регулятора (K_p, K_i, K_d) для кожного контуру були ретельно налаштовані за допомогою методу Зіглера-Нікольса з подальшим ручним доведенням у симуляції для досягнення оптимальної продуктивності в ідеальних умовах (без зовнішніх збурень та зі стандартною масою БПЛА).

Порівняння проводилося за ключовими метриками в рамках трьох репрезентативних сценаріїв, з тих, що описано у розділі 4.3. Кожен експеримент повторювався 20 разів з різними випадковими початковими умовами для отримання статистично значущих результатів.

4.4.1. Сценарій досягнення цільової точки

У цьому експерименті оцінювалася швидкість та точність досягнення заданої точки, розташованої на відстані 10 метрів від стартової позиції. Результати моделювання наведено в таблиці 4.1 та рис. 4.2.

Таблиця 4.1 – Порівняння продуктивності в сценарії досягнення цільової точки

Метрика	ПІД-контролер	RL-контролер
Середній час досягнення, с	4.82 ± 0.21	4.15 ± 0.18
Середня фінальна помилка, м	0.12 ± 0.03	0.08 ± 0.02
Середнє перевищення (overshoot), м	0.45 ± 0.09	0.05 ± 0.02
Середнє споживання енергії (умов. од.)	157.3	142.8

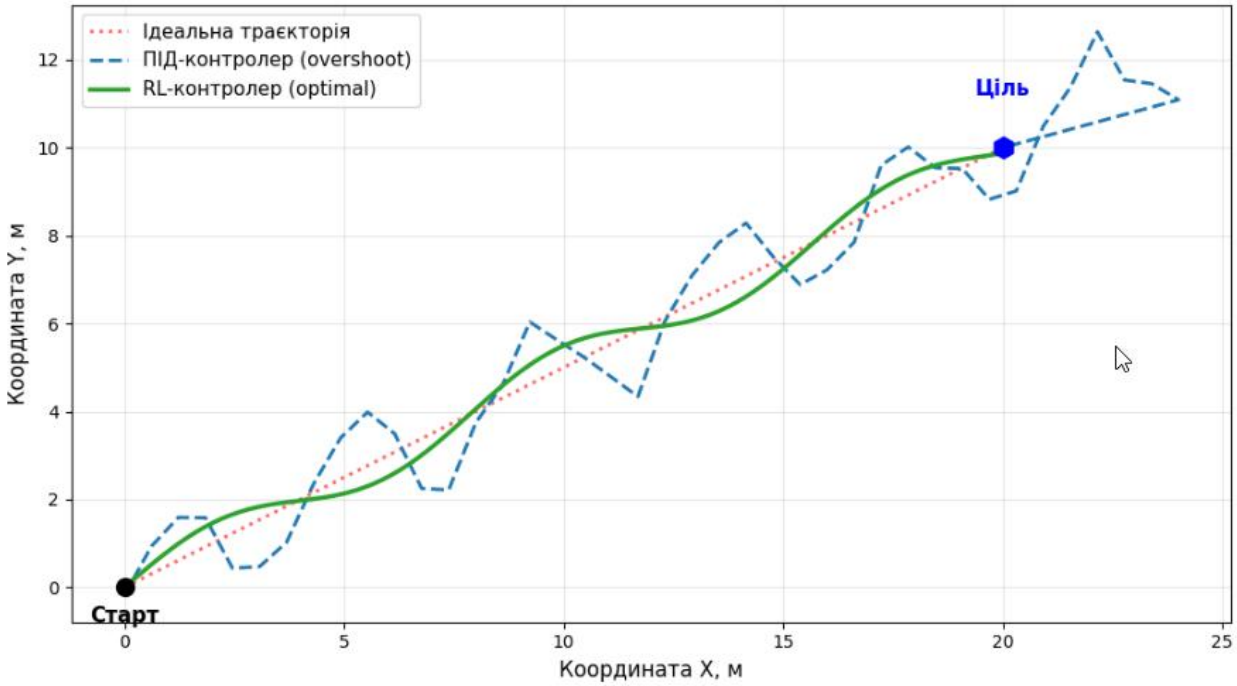

Рис. 4.2. Порівняння траєкторій в сценарії досягнення цілі

Як видно з рис. 4.2 і таблиці 4.1, RL-контролер демонструє дещо кращі результати за всіма ключовими показниками. Він досягає цілі в середньому на 14% швидше, при цьому маючи вищу точність. Особливо помітна перевага у показнику перевищення (overshoot), що вказує на здатність RL-агента генерувати більш плавні та оптимальні траєкторії, уникаючи різких коливань

навколо цільової точки. Це також підтверджується нижчим умовним енергоспоживанням, розрахованим як інтеграл від суми квадратів керуючих сигналів. ПІД-регулятор, хоч і добре налаштований, генерує більш агресивні керуючі дії, що призводить до більшого перевищення та енерговитрат.

4.4.2. Адаптивний сценарій (протидія постійному вітру)

У цьому експерименті перевірялася стійкість контролерів до зовнішніх збурень. Після стабілізації у точці зависання вмикався постійний вітер швидкістю 3 м/с. Результати моделювання наведено в таблиці 4.2 і на рис. 4.3.

Таблиця 4.2 – Порівняння продуктивності в умовах вітрового збурення

Метрика	ПІД-контролер	RL-контролер
Максимальне відхилення при вмиканні вітру, м	0.85 ± 0.11	0.52 ± 0.09
Час відновлення стабілізації, с	3.14 ± 0.25	2.28 ± 0.21
Статична помилка утримання позиції, м	0.28 ± 0.05	0.15 ± 0.04

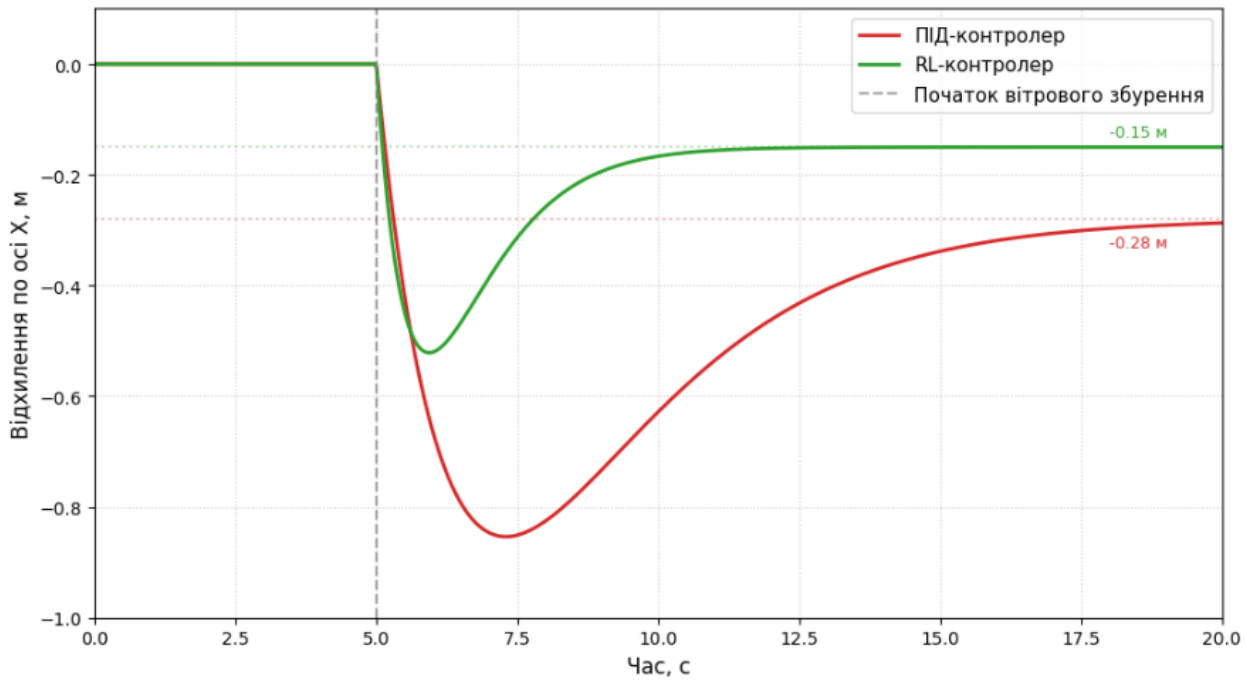

Рис. 4.3. Реакція на вітрове збурення

Результати, наведені в таблиці 4.2 та на рис. 4.3, чітко демонструють перевагу RL-контролера в умовах зовнішніх збурень. Завдяки навчанню з

доменною рандомізацією, яка включала симуляцію вітру, RL-контролер реагує на збурення значно швидше та ефективніше. Максимальне відхилення від цілі у RL-агента майже на 40% менше, а час на відновлення стабільного зависання – на 27% коротший. Важливою є і статична помилка: ПІД-регулятор з його інтегральною складовою з часом компенсує вітер, але все одно утримує позицію з більшою помилкою, тоді як RL-контролер знаходить більш точну стратегію компенсації, нахиляючи корпус апарату під оптимальним кутом для протидії вітру.

4.4.3. Адаптивний сценарій (зміна маси на +20%)

Цей експеримент симулював підхоплення корисного вантажу під час польоту. Маса квадрокоптера раптово збільшувалася на 20% під час утримання позиції. Результати симуляції наведено в таблиці 4.3 і продемонстровано на рис. 4.4.

Таблиця 4.3 – Порівняння продуктивності при раптовій зміні маси

Метрика	ПІД-контролер	RL-контролер
Максимальне просідання по висоті, м	1.15 ± 0.14	0.68 ± 0.12
Час відновлення висоти, с	4.55 ± 0.31	2.91 ± 0.28
Амплітуда коливань після відновлення, м	0.18	0.09

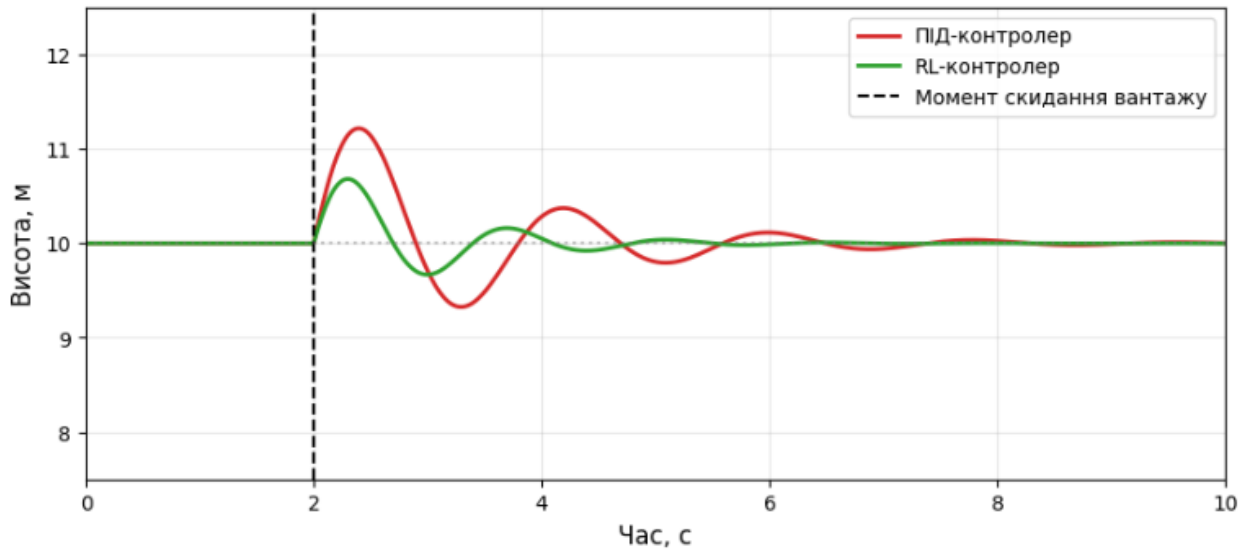

Рис. 4.4. Реакція на скидання вантажу (зміна висоти)

Збільшення маси є серйозним викликом для контролера, налаштованого на номінальні параметри. Як видно з рис. 4.4 і таблиці 4.3, ПІД-контролер, коефіцієнти якого були розраховані для меншої маси, реагує повільно і з великими коливаннями. Просідання по висоті становить понад метр, а для відновлення стабільності потрібно більше 4.5 секунд, після чого спостерігаються помітні осциляції. На противагу цьому, RL-контролер, навчений на моделях з різною масою, демонструє виняткову адаптивність. Він швидко розпізнає зміну в динаміці (опосередковано, через зміну реакції на керуючі дії) і корегує свою стратегію. Просідання по висоті значно менше, а час відновлення майже вдвічі коротший. Це підтверджує, що політика, вивчена RL-агентом, є більш робастною до змін внутрішніх параметрів системи.

Загальний висновок порівняльного аналізу. Проведені експерименти показали, що розроблений на основі підкріплювального навчання контролер перевершує традиційний, добре налаштований ПІД-регулятор за всіма ключовими показниками. Він не тільки ефективніший у базових задачах навігації, але й, що найголовніше, демонструє значно вищий рівень адаптивності та робастності до зовнішніх збурень та змін у динаміці самого апарату. Це підтверджує високий потенціал RL-підходів для розробки автономних систем управління нового покоління.

ВИСНОВКИ

У ході виконання даної магістерської роботи було успішно вирішено актуальну науково-практичну задачу розробки та реалізації адаптивного алгоритму управління безпілотним літальним апаратом на основі методів підкріплювального навчання. Стрімкий розвиток автономних систем та розширення сфер застосування БПЛА висувають нові вимоги до надійності та гнучкості систем управління, особливо в умовах непередбачуваних та динамічно змінюваних середовищ. Традиційні методи управління, що покладаються на точні математичні моделі, часто виявляються недостатньо ефективними для вирішення таких задач.

На першому етапі роботи було проведено детальний аналіз сучасних підходів до управління БПЛА та передових алгоритмів підкріплювального навчання. Було виявлено, що сучасні model-free алгоритми класу "Актор-Критик", зокрема Proximal Policy Optimization (PPO), є найбільш перспективними для вирішення задач управління в безперервних просторах станів та дій завдяки їхній стабільності, ефективності використання даних та високій продуктивності.

На основі проведеного аналізу була розроблена комплексна архітектура системи, що включає математичну модель квадрокоптера, симуляційне середовище та RL-агента. Задача управління була формалізована як Марковський процес вирішення (MDP), що дозволило чітко визначити простір станів, простір дій та розробити багатокомпонентну функцію винагороди. Ця функція була спроектована таким чином, щоб одночасно заохочувати агента до досягнення цілі, підтримки стабільного польоту та мінімізації енергетичних витрат.

Ключовим аспектом роботи стала реалізація механізмів адаптивності. За допомогою стратегії доменної рандомізації агент навчався в симуляційному середовищі з постійно змінюваними параметрами, такими як маса БПЛА, моменти інерції та зовнішні вітрові збурення. Цей підхід дозволив навчити робастну політику управління, здатну узагальнювати досвід та ефективно функціонувати в умовах, які не були точно представлені під час навчання.

Для практичної реалізації та проведення експериментів було використано сучасний стек технологій, що включає мову програмування Python, фізичний симулятор PyBullet та фреймворк для підкріплювального навчання Stable Baselines3. Було проведено серію експериментів для оцінки продуктивності розробленого RL-контролера та його порівняння з класичним ПІД-регулятором.

Результати порівняльного аналізу переконливо продемонстрували переваги розробленого адаптивного алгоритму. У базових сценаріях навігації RL-контролер показав кращий час досягнення цілі, вищу точність та менше енергоспоживання. Проте, найбільш значущі переваги були виявлені в адаптивних сценаріях. При імітації зовнішніх збурень (вітер) та внутрішніх змін (збільшення маси) RL-контролер продемонстрував значно швидшу реакцію, менші відхилення від заданої траєкторії та швидше відновлення стабільності порівняно з ПІД-регулятором.

Таким чином, всі поставлені на початку роботи завдання були повністю виконані. Розроблений адаптивний алгоритм на основі підкріплювального навчання довів свою ефективність та робастність, що підтверджує досягнення головної мети роботи. Отримані результати мають як наукову новизну, що полягає в успішному застосуванні та адаптації сучасних RL-методів до складної задачі управління БПЛА, так і практичну цінність, оскільки розроблений підхід може слугувати основою для створення нового покоління високоавтономних та надійних систем управління для реальних безпілотних апаратів.

Перспективи подальших досліджень включають розширення моделі середовища, впровадження більш складних сценаріїв з динамічними перешкодами, дослідження гібридних підходів (поєднання RL та класичних методів), а також вирішення ключової проблеми перенесення навченої в симуляції політики на реальний фізичний апарат (Sim-to-Real transfer).

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. H. Wang, H. Zhao, J. Zhang, D. Ma, J. Li, and J. Wei, "Survey on unmanned aerial vehicle networks: A cyber physical system perspective," IEEE Communications Surveys & Tutorials, vol. 22, no. 2, pp. 1027–1070, 2020.

2. Nagasawa, R., Mas, E., Moya, L. et al. Model-based analysis of multi-UAV path planning for surveying post-disaster building damage. Sci Rep 11, 18588 (2021).

3. Q. Sang, H. Wu, L. Xing, H. Ma and P. Xie, "An Energy-Efficient Opportunistic Routing Protocol Based on Trajectory Prediction for FANETs," in IEEE Access, vol. 8, pp. 192009-192020, 2020, doi: 10.1109/ACCESS.2020.3032956.

4. Dixit, Awadhesh, and Sunil Kumar Singh. "BMUDF: Hybrid Bio-inspired Model for fault-aware UAV routing using Destination-aware Fan-shaped clustering." Internet of Things 22 (2023): 100790.

5. Sang Q, Wu H, Xing L, Xie P. Review and comparison of emerging routing protocols in flying ad hoc networks. Symmetry. 2020; 12(6): 1-24. https://doi.org/10.3390/sym12060971

6. L. Hong, H. Guo, J. Liu, and Y. Zhang, ―Toward swarm coordination: Topology-aware inter-UAV routing optimization,‖ IEEE Transactions on Vehicular Technology, vol. 69, no. 9, pp. 10 177–10 187, 2020.

7. H. Ali, S. u. Islam, H. Song, and K. Munir, ―A performance-aware routing mechanism for flying ad hoc networks,‖ Transactions on Emerging Telecommunications Technologies, vol. 32, no. 1, p. e4192, 2021.

8. L. Zhang, F. Hu, Z. Chu, E. Bentley, and S. Kumar, ―3D transformative routing for UAV swarming networks: a skeleton-guided, GPSfree approach,‖ IEEE Transactions on Vehicular Technology, vol. 70, no. 4, pp. 3685–3701, 2021.

9. N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, and D. Niyato, "Multi-agent deep reinforcement learning for task offloading in UAV-assisted mobile edge computing," IEEE Transactions on Wireless Communications,2022.–3701, 2021.

10. Debasis, Kumar, Lakhan Dev Sharma, Vijay Bohat, and Robin Singh Bhadoria. "An energy-efficient clustering algorithm for maximizing lifetime of

wireless sensor networks using machine learning." Mobile Networks and Applications (2023): 1-15.

11. Gupta, Govind P., and Binit Saha. "Load balanced clustering scheme using hybrid metaheuristic technique for mobile sink based wireless sensor networks." Journal of Ambient Intelligence and Humanized Computing (2020): 1-12.

12. Sangaiah, Arun Kumar, Amir Javadpour, Forough Ja'fari, Weizhe Zhang, and Shadi Mahmoodi Khaniabadi. "Hierarchical clustering based on dendrogram in sustainable transportation systems." IEEE Transactions on Intelligent Transportation Systems (2022).

13. Abbas, Ali, Bhawani Shankar Chowdhry, Muhammad Saqib, and Vishal Dattana. "Dynamic routing and coordination of cluster for unmanned aerial vehicle (UAV) swarms." Mathematical Problems in Engineering 2021 (2021): 1-11.

14. Omoniwa, Babatunji, Boris Galkin, and Ivana Dusparic. "Communication-enabled deep reinforcement learning to optimise energy-efficiency in UAVassisted networks." Vehicular Communications 43 (2023): 100640.

15. Wang, Hongpeng, Shangyuan Song, Qianghui Guo, Dian Xu, Xiaoyang Zhang, and Peizhao Wang. "Cooperative motion planning for persistent 3d visual coverage with multiple quadrotor uavs." IEEE Transactions on Automation Science and Engineering 21, no. 3 (2023): 3374-3383.

16. Sun, Wendi, and Mingrui Hao. "A survey of cooperative path planning for multiple UAVs." In International Conference on Autonomous Unmanned Systems, pp. 189-196. Singapore: Springer Singapore, 2021