

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТРАНСПОРТНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут управління, технологій та правових наук
Кафедра інформаційних технологій

МАГІСТЕРСЬКА РОБОТА

на тему:

СИСТЕМА ВИЗНАЧЕННЯ КООРДИНАТ АВТОНОМНОГО ПОЛЬОТУ
БПЛА ЗА ДОПОМОГОЮ ШТУЧНОГО ІНТЕЛЕКТУ

Спеціальність: 121 «Інженерія програмного забезпечення»
ОПП «Інженерія програмного забезпечення»

Розробив: Микита ГРЕБЕНЮК
(ім'я, ПРІЗВИЩЕ)

(підпис)

Керівник: к.т.н., доцент Олена ГРІНЕНКО
(науковий ступінь, вчене звання) (ім'я, ПРІЗВИЩЕ)

(підпис)

Нормоконтроль: к.т.н., доцент Олена ГРІНЕНКО
(науковий ступінь, вчене звання) (ім'я, ПРІЗВИЩЕ)

(підпис)

Допущений до захисту
« ____ » _____ 2025р.

В.о. завідувача кафедри: д.т.н., професор Олег НІКОНОВ
(науковий ступінь, вчене звання) (ім'я, ПРІЗВИЩЕ)

(підпис)

Рецензент: к.т.н., доцент Владислав ЯСКЕВИЧ
(науковий ступінь, вчене звання) (ім'я, ПРІЗВИЩЕ)

(підпис)

Київ - 2025

Міністерство освіти і науки України
 Національний транспортний університет
 Навчально-науковий інститут управління, технологій та правових наук
 Кафедра інформаційних технологій

Рівень вищої освіти - другий (магістерський)
 Галузь знань 12 «Інформаційні технології»
 Спеціальність 121 «Інженерія програмного забезпечення»
 ОПП «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри
 інформаційних технологій
 д.т.н., проф. _____ Олег НІКОНОВ

«_____» _____ 2025 р.

**Завдання
 на магістерську роботу здобувачу вищої освіти
 Микиті ГРЕБЕНЮКУ**

(ім'я, прізвище)

1. Тема роботи: «Система визначення координат автономного польоту БПЛА за допомогою штучного інтелекту»

керівник роботи: к.т.н., доцент Олена ГРІНЕНКО

Затверджені наказом від «17» жовтня 2025 року № 1031

2. Строк подання здобувачем роботи 19 грудня 2025 року

3. Вихідні дані до роботи: Система визначення координат автономного польоту БПЛА, дані з бортових сенсорів

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області автономної навігації БПЛА (огляд та аналіз методів визначення координат і траєкторії руху БПЛА), дослідження методів комп'ютерного зору та штучного інтелекту для задач локалізації і навігації (алгоритм визначення координат і траєкторії руху БПЛА на основі злиття даних відео- та інерціальних сенсорів), опис програмного продукту (структури інформаційної бази, функцій та сценаріїв його використання).

5. Перелік графічного матеріалу: презентація магістерської роботи та скріншоти функціонування програмного продукту

6. Консультанти роботи: к.т.н., доцент Олена ГРІНЕНКО

7. Календарний план виконання роботи

№ з/п	Найменування етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітки
1	Огляд та аналіз джерел тематичної області	17.10.25 - 22.10.25	
2	Огляд та аналіз шляхів розв'язання проблем роботи	23.10.25 - 28.10.25	
3	Опис власного підходу до вирішення проблеми	29.10.25 - 04.11.25	
4	Визначення вимог до програмного продукту та його структури	05.11.25 - 07.11.25	
5	Опис та реалізація основних функцій програмного продукту	08.11.25 - 28.11.25	
6	Оформлення пояснювальної записки	29.11.25 - 05.12.25	
7	Захист		

8. Дата видачі завдання «17» жовтня 2025 року

Здобувач вищої освіти

(підпис)

Гребенюк М.С

(прізвище та ініціали)

Керівник роботи

(підпис)

Грінченко О.О.

(прізвище та ініціали)

РЕФЕРАТ

Тема «Система визначення координат автономного польоту БПЛА за допомогою штучного інтелекту»

Магістерська робота: 97 сторінок, 36 рисунків, 11 таблиць, 34 джерела, 3 додатки

Магістерська робота присвячена дослідженню та розробці системи визначення просторових координат та траєкторії руху БПЛА на основі даних відеопотоку та сенсорів руху з використанням методів комп'ютерного зору та штучного інтелекту.

Метою роботи є вивчення предметної області автономної навігації БПЛА, формалізація задачі визначення координат та траєкторії руху в умовах обмеженої або відсутньої супутникової навігації, а також розробка алгоритмічної та програмної основи відповідної системи. Дослідження зосереджене на аналізі методів візуальної одометрії, локалізації за маркерами та підходів сенсорної ф'юзії, а також на побудові математичної моделі руху БПЛА та алгоритму оцінювання просторового положення у часі.

У процесі виконання роботи було проведено аналіз вимог до системи, розроблено структурну та функціональну модель програмного продукту, що забезпечує обробку візуальних даних, формування траєкторії руху та візуалізацію результатів.

Результатом роботи є система визначення координат та траєкторії руху БПЛА, яка формує структуровані результати обробки у вигляді часових рядів координат. Розроблена система має потенціал для подальшого розвитку та застосування більш складних моделей штучного інтелекту.

Ключові слова: безпілотний літальний апарат, автономна навігація, оцінювання траєкторії, комп'ютерний зір, штучний інтелект, візуальна одометрія

ABSTRACT

Topic «System for determining the coordinates of the autonomous UAV flight using artificial intelligence»

Master's Thesis: 97 pages, 36 figures, 11 tables, 34 references, 3 appendices

The master's thesis is dedicated to the research and development of a system for determining the spatial coordinates and trajectory of UAV motion based on video stream data and motion sensors using computer vision and artificial intelligence methods.

The goal of the work is to study the subject area of autonomous UAV navigation, formalize the problem of determining coordinates and trajectory in conditions of limited or absent satellite navigation, and develop the algorithmic and software foundations of the corresponding system. The research focuses on the analysis of visual odometry methods, marker-based localization, and sensor fusion approaches, as well as on building a mathematical model of UAV motion and an algorithm for estimating spatial position over time.

During the course of the work, system requirements were analyzed, input/output data formats were defined, and a structural and functional model of the software product was developed, ensuring visual data processing, trajectory formation, and result visualization.

The outcome of the work is a UAV coordinate and trajectory determination system that generates structured processing results in the form of coordinate time series.

The developed system has potential for further development, including integration of inertial sensors, improvement of trajectory filtering methods, and application of more advanced artificial intelligence models.

Keywords: unmanned aerial vehicle, autonomous navigation, trajectory estimation, computer vision, artificial intelligence, visual odometry

ЗМІСТ

ВСТУП	7
1 ОГЛЯД ТА АНАЛІЗ ТЕМАТИЧНОЇ ОБЛАСТІ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ НАВІГАЦІЇ БПЛА	11
1.1 Аналіз задач навігації та локалізації безпілотних літальних апаратів	11
1.2 Огляд методів комп'ютерного зору та штучного інтелекту для відстеження руху і реконструкції траєкторії	17
1.3 Аналіз існуючих програмних рішень та середовищ симуляції для моделювання польотів БПЛА	24
2 АНАЛІЗ ТА ШЛЯХИ РОЗВ'ЯЗАННЯ ПРОБЛЕМИ ВИЗНАЧЕННЯ ТРАЄКТОРІЇ ПОЛЬОТУ БПЛА	31
2.1 Постановка задачі та математична модель руху БПЛА	31
2.2 Розробка алгоритму оцінювання координат та траєкторії руху БПЛА за даними відео і сенсорів руху	44
2.3 Структурна та функціональна модель програмного модуля визначення координат і траєкторії руху БПЛА	48
3 ОПИС ПРОГРАМНОГО МОДУЛЯ ВИЗНАЧЕННЯ КООРДИНАТ І ТРАЄКТОРІЇ РУХУ БПЛА	54
3.1 Вимоги до програмного модуля визначення координат і траєкторії руху БПЛА	54
3.2 Структура інформаційної бази програмного модуля визначення координат і траєкторії руху БПЛА	58
3.3 Опис основних функцій програмного модуля визначення координат і траєкторії руху БПЛА	68
ВИСНОВКИ	78
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	82
ДОДАТКИ	87

ВСТУП

Актуальність теми дослідження. Безпілотні літальні апарати, або дрони, сьогодні знаходять широке застосування у найрізноманітніших сферах - від сільського господарства та логістики до використання в умовах війни. Виконання багатьох небезпечних чи дорогих завдань покладається на БПЛА завдяки їхній мобільності та гнучкості.

Однак для успішного автономного виконання місій дрону необхідно інтелектуально навігувати - тобто самостійно орієнтуватися в просторі, визначати своє місцеположення, будувати безпечний маршрут до цілі та уникати зіткнень.

Точне визначення координат і траєкторії руху є важливим для автономної навігації і обходу перешкод, особливо в умовах, де відсутній надійний сигнал GPS (наприклад, в приміщеннях, густо забудованих містах чи лісах). Обмежені можливості традиційних засобів навігації (наприклад, покладення лише на GPS або прості датчики) серйозно гальмують застосування дронів у складних умовах, таких як GPS-недоступні зони.

Сучасні тенденції розвитку безпілотних систем характеризуються впровадженням методів штучного інтелекту та комп'ютерного зору для підвищення автономності навігації.

Інтеграція передових датчиків, обчислювальних технологій та алгоритмів ШІ суттєво розширила можливості БПЛА, дозволяючи виконувати дедалі складніші завдання в автономному режимі. Це і є суттю інтелектуальних систем навігації - поєднання різноманітних сенсорів (камери, лідара, інерціальних блоків, супутникових приймачів тощо) з алгоритмами обробки даних та прийняття рішень, що дає змогу дрону самостійно орієнтуватися в просторі.

Мета роботи - розробити підхід і програмний модуль визначення координат та траєкторії руху БПЛА за даними відеопотоку та сенсорів руху з використанням методів штучного інтелекту.

Для досягнення поставленої мети в роботі необхідно розв'язати такі **завдання:**

- Проаналізувати задачі навігації та локалізації БПЛА й обґрунтувати вимоги до визначення координат/траєкторії в умовах ненадійного або відсутнього GPS.
- Оглянути методи комп'ютерного зору та ШІ, придатні для оцінювання переміщення і відновлення траєкторії (візуальна одометрія/SLAM, візуально-інерціальні підходи тощо).
- Сформулювати постановку задачі та обґрунтувати математичну модель руху БПЛА, що використовується для оцінювання просторового стану.
- Розробити алгоритм оцінювання координат і траєкторії руху БПЛА на основі злиття даних відеопотоку та сенсорів руху.
- Побудувати структурну та функціональну модель програмного модуля визначення координат і траєкторії.
- Визначити вимоги до програмного модуля, описати структуру інформаційної бази та реалізувати основні функції обробки даних і формування результатів.
- Описати результати роботи модуля та особливості реалізації, а також окреслити практичні сценарії застосування.

Об'єктом дослідження є процес автономної навігації безпілотного літального апарата в тривимірному просторі.

Предметом дослідження є методи та алгоритми визначення координат БПЛА з використанням моделей штучного інтелекту та обробки сенсорної інформації.

Методи дослідження. У роботі використовуються методи математичного моделювання, теорії керування, чисельної оптимізації, а також методи машинного навчання і навчання з підкріплення. Для аналізу та перевірки алгоритмів застосовуються комп'ютерні симуляційні середовища для моделювання польоту БПЛА. Також використовуються методи

порівняльного аналізу та узагальнення результатів сучасних наукових досліджень.

Наукова новизна роботи полягає в систематизації та узагальненні підходів до визначення координат автономного польоту БПЛА з використанням штучного інтелекту, а також у формуванні концептуальної структури системи навігації, що поєднує математичну модель руху апарата, сенсорні дані та інтелектуальні алгоритми обробки інформації.

Практичне значення отриманих результатів полягає в можливості використання запропонованих підходів під час розробки програмного забезпечення для автономних безпілотних літальних апаратів, зокрема в задачах навігації в умовах обмеженої або відсутньої супутникової навігації. Матеріали роботи можуть бути використані в навчальному процесі та при подальших наукових дослідженнях у галузі автономних робототехнічних систем.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків. У першому розділі виконано огляд і аналіз предметної області навігації та локалізації БПЛА, розглянуто підходи комп'ютерного зору й штучного інтелекту для оцінювання руху та реконструкції траєкторії, а також проаналізовано наявні програмні рішення і середовища моделювання.

У другому розділі сформульовано постановку задачі, подано модель руху та розроблено алгоритм визначення координат і траєкторії за даними відеопотоку й сенсорів руху, а також побудовано структурну і функціональну моделі програмного модуля. У третьому розділі визначено вимоги до програмного модуля, описано структуру інформаційної бази, реалізовано та охарактеризовано основні функції обробки даних, візуалізації й експорту результатів.

Особистий внесок магістра. Магістерська робота є самостійним завершеним дослідженням автора.

Публікації: дослідження були апробовані:

– Ткаченко О., Гребенюк М. Система визначення координат при автономному польоті БПЛА. *Управління та адміністрування в умовах протидії гібридним загрозам національній безпеці*: збірник матеріалів VI Міжнародної науково-практичної конференції (м. Київ, 26 листопада, 2025 р.) Київ: НТУ, 2025. [33].

– результати дослідження були подані та прийняті до друку в науковій статті Ткаченко О.А., Гребенюк М.С. (2025). Деякі аспекти програмного забезпечення визначення координат при автономному польоті БПЛА. *Цифрова платформа: інформаційні технології в соціокультурній сфері*, 9(2) [34].

1 ОГЛЯД ТА АНАЛІЗ ТЕМАТИЧНОЇ ОБЛАСТІ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ НАВІГАЦІЇ БПЛА

1.1 Аналіз задач навігації та локалізації безпілотних літальних апаратів

Навігація безпілотних літальних апаратів є однією зі складових їх функціонування, оскільки саме вона забезпечує здатність апарата визначати власне положення у просторі, орієнтуватися відносно заданих координат, коригувати траєкторію руху та виконувати поставлені завдання з високою точністю. Поняття навігації БПЛА охоплює сукупність методів, алгоритмів і технічних засобів, які дозволяють апарату автономно або напіваавтономно здійснювати політ у складному та динамічному середовищі.

На відміну від пілотованої авіації, де значна частина навігаційних рішень приймається людиною, безпілотні системи потребують значно вищого рівня автоматизації, стійкості до зовнішніх впливів та здатності працювати в умовах обмеженої або спотвореної інформації.

Історичне становлення навігації безпілотних літальних апаратів тісно пов'язане з розвитком авіації та систем керування в цілому. Перші спроби створення апаратів без пілота датуються початком XX століття, коли з'явилися примітивні радіокеровані літальні засоби, навігація яких зводилася до елементарного дистанційного управління без зворотного зв'язку.

У цей період поняття навігації фактично не існувало у сучасному розумінні, оскільки апарати не мали можливості самостійно визначати своє місцеположення або адаптувати маршрут. Подальший розвиток електроніки, гіроскопічних приладів і систем стабілізації у середині XX століття заклав підґрунтя для створення інерціальних навігаційних систем, які стали першим кроком до автономності польоту.

У другій половині XX століття розвиток навігації БПЛА відбувався переважно в межах військових програм, де безпілотні апарати розглядалися як засоби розвідки та мішені для тренування протиповітряної оборони. Саме в

цей період були закладені основи інерціальної навігації, що базувалася на використанні акселерометрів і гіроскопів для обчислення координат апарата шляхом інтегрування параметрів руху.

Хоча такі системи мали значну похибку, що накопичувалася з часом, вони дозволяли БПЛА виконувати польоти без постійного зовнішнього управління. Поява супутникових навігаційних систем наприкінці XX століття стала революційним етапом у розвитку навігації, оскільки відкрила можливість точного глобального позиціонування у реальному часі.

На межі XX і XXI століть навігація безпілотних літальних апаратів почала активно інтегрувати супутникові системи, такі як GPS, що суттєво підвищило точність визначення координат і дозволило реалізувати складні маршрути польоту.

У цей період відбувся перехід від суто військового використання БПЛА до цивільних застосувань, зокрема в геодезії, сільському господарстві, моніторингу інфраструктури та екологічному спостереженні. Навігаційні системи стали комбінованими, поєднуючи інерціальні датчики, супутникові приймачі та алгоритми фільтрації, що дозволяло зменшувати похибки та підвищувати надійність роботи.

Подальший розвиток навігації БПЛА був тісно пов'язаний із прогресом обчислювальної техніки та штучного інтелекту. З'явилися методи візуальної навігації, які використовують дані з камер для визначення положення апарата відносно об'єктів навколишнього середовища.

Такі підходи стали особливо важливими в умовах, де супутникові сигнали недоступні або ненадійні, наприклад у міській забудові чи в приміщеннях. Алгоритми одночасної локалізації та побудови карти дозволили БПЛА не лише орієнтуватися в просторі, але й формувати цифрове уявлення про навколишнє середовище, що стало важливим етапом у напрямку повної автономності.

Особливого значення навігація безпілотних літальних апаратів набула в умовах повномасштабної війни, де вона стала важливим фактором ефективності бойових дій.

Військові конфлікти сучасного типу характеризуються активним застосуванням засобів радіоелектронної боротьби, що суттєво ускладнює використання супутникових навігаційних систем. У таких умовах розвиток навігації БПЛА набув нового імпульсу, спрямованого на підвищення стійкості до глушіння, підміни сигналів та інших форм впливу.

Це призвело до активного впровадження альтернативних методів навігації, зокрема інерціальних, візуальних та комбінованих систем, які здатні працювати в умовах інформаційної невизначеності.

Повномасштабна війна стала причиною швидкого технологічного розвитку навігаційних рішень для БПЛА, оскільки потреба у точному та надійному позиціонуванні безпілотників безпосередньо впливає на успішність розвідки, коригування вогню та ударних операцій.

Навігація в бойових умовах потребує не лише точності, але й адаптивності, оскільки середовище постійно змінюється, а маршрути польоту повинні коригуватися в реальному часі. Це і зумовило активне використання алгоритмів машинного навчання, які дозволяють БПЛА аналізувати навігаційні дані та приймати рішення на основі накопиченого досвіду.

Важливою особливістю розвитку навігації БПЛА у воєнний час є зростання ролі локальних навігаційних систем, що не залежать від глобальної інфраструктури. Такі системи можуть базуватися на зіставленні зображень місцевості, аналізі рельєфу або використанні попередньо створених цифрових карт. У поєднанні з інерціальними датчиками вони дозволяють безпілотникам виконувати завдання навіть у разі повної втрати супутникового сигналу. Це суттєво підвищує живучість апаратів та знижує їхню вразливість до технічних засобів противника.

Наразі навігація безпілотних літальних апаратів розглядається не лише як технічна функція, але й як стратегічний ресурс, що визначає перевагу на

полі бою. Інтеграція навігаційних систем із засобами зв'язку, управління та обробки даних формує єдиний інформаційний простір, у межах якого БПЛА стають елементами складних мережево-центричних систем.

Навігація БПЛА охоплює широкий спектр завдань, пов'язаних з автономним переміщенням дрона у просторі. До підзадач належать:

- локалізація - визначення поточного місцеположення та орієнтації апарата;
- планування маршруту - прокладання оптимальної траєкторії руху до цільової точки;
- стабілізація і керування - підтримання стійкого польоту та керування рухом згідно з запланованим шляхом;
- виявлення та уникнення перешкод - своєчасне виявлення перепон на шляху та обхід їх для запобігання зіткнень.

Для успішної автономної навігації дрон має орієнтуватися в навколишньому середовищі й водночас контролювати власний рух так, щоб досягнути мети швидко й безпечно.

Навігаційна система БПЛА зазвичай базується на комплексі різномірних датчиків і алгоритмів, що надають інформацію про параметри руху. Глобальні координати визначаються за допомогою супутникових навігаційних систем (GPS, ГЛОНАСС тощо), тоді як для відносної локалізації та орієнтації застосовуються бортові сенсори: інерціальні вимірювальні блоки (IMU), барометри, магнітометри, оптичні камери, лідари тощо.

Кожен тип сенсорів має свої переваги й обмеження. GPS є основним засобом навігації завдяки глобальному покриттю і простоті використання, забезпечуючи типову точність порядку кількох метрів.

Проте за відсутності або погіршення сигналу GPS (в приміщеннях, під водою, у міській забудові чи під дією глушіння) доводиться використовувати альтернативні методи локалізації [1]. Візуальні сенсори (камери, лідари, інфрачервоні камери) можуть давати високу точність і детальну інформацію про оточення, однак залежать від умов довкілля (освітлення, наявність

текстур) та потребують значних обчислювальних ресурсів. До того ж, високоякісні лідари є дорогими й важкими, що важливо для дронів з обмеженою вантажопідйомністю. Інерціальні датчики (акселерометри й гіроскопи у складі IMU) забезпечують автономне відстеження руху на коротких інтервалах, але їхні покази схильні до накопичення похибки (дрейфу) з часом [1].

Радіочастотні методи локалізації (наприклад, за сигналами Wi-Fi або Bluetooth) працюють лише на малій відстані та зазвичай піддаються шумам й інтерференції сигналу. Зважаючи на обмеження кожного окремого виду сенсорів, для забезпечення високої точності позиціювання в умовах недоступності GPS перспективними є спеціалізовані технології, зокрема ультраширококустовий радіозв'язок (або UWB). Технологія UWB здатна забезпечити дуже точну (до кількох сантиметрів) локалізацію на малих дистанціях, стійку до мультипроменевих спотворень та перешкод при невеликому енергоспоживанні.

Ба більше, інтеграція UWB з традиційним GPS розглядається як ефективна комбінація. GPS надає необмежений радіус дії, але з обмеженою точністю, тоді як UWB покриває невеликі відстані з високою прецизійністю. Дрони, оснащені UWB-маяками, уже застосовуються для відносної навігації в роях (для підтримання строю без зіткнень) та для польотів усередині приміщень, де сигнал GPS недоступний. [1]

Сучасні системи навігації БПЛА реалізують інтегрований підхід, поєднуючи дані від різних датчиків шляхом сенсорної ф'юзії (злиття сенсорної інформації) для підвищення надійності й точності [2].

Для інтеграції різномірних даних широко застосовуються алгоритми фільтрації, зокрема фільтр Калмана, який дозволяє об'єднувати GPS-дані з показами інерціальної навігаційної системи і тим самим суттєво підвищувати точність оцінки положення. Наприклад, GPS може надавати абсолютні координати, тоді як камера та лідар - детальну картину оточення для обходу перешкод, а IMU - безперервне відстеження руху між оновленнями сигналу

GPS. Така комбінація компенсує недоліки окремих методів: візуальна одометрія здатна поступово виправляти дрейф IMU, а супутникова навігація - усувати накопичену похибку відносних способів навігації.

В умовах відсутності GPS особливу роль відіграють технології Visual SLAM (Simultaneous Localization and Mapping, одночасна локалізація і картографування) та візуально-інерціальна одометрія, та дозволяють дрону будувати карту і оцінювати власний рух на основі даних камер та IMU [2].

Поєднання візуальної інформації з інерційними вимірюваннями дає змогу орієнтуватися в динамічному середовищі навіть без супутникових сигналів. У результаті інтелектуальна навігаційна система дрона здатна функціонувати навіть у складних умовах - при відсутності супутникового сигналу, поганому освітленні чи появі динамічних перешкод, оскільки сенсорна ф'юзія знижує дрейф та підвищує точність позиціонування. [3]

Для кращого розуміння процесу навігації розглянемо типовий цикл автономного польоту дрона. Спочатку бортові сенсори збирають дані про навколишнє середовище - наприклад, за допомогою камери чи лідара формується карта місцевості, зчитуються GPS-координати та визначається початкова позиція апарата [3].

На основі цієї інформації навігаційна система планує маршрут: обчислює оптимальний шлях від поточного місця до пункту призначення з урахуванням статичних перешкод (рельєф, будівлі). Під час руху дрон постійно моніторить оточення за допомогою сенсорів у реальному часі. Якщо на шляху виникають несподівані перепони (наприклад, рухомі об'єкти) або змінюються умови, бортовий комп'ютер коригує траєкторію - виконує локальне перепланування або ж облітає перешкоди на льоту, пріоритезуючи швидкість реакції над абсолютною оптимальністю маршруту.

Таким чином, поєднуються глобальне планування (для досягнення кінцевої мети) і локальне реагування (для маневрування в динамічному середовищі) . У результаті БПЛА слідує оновленому курсу, безперервно звіряючи фактичне положення з плановим та вносячи поправки за потреби.

Цикл «відчуття оточення - планування - дія» повторюється знову і знову, гарантуючи надійну та безпечну навігацію навіть у складних або непередбачуваних умовах.

Варто зазначити, що точність позиціювання та надійність навігаційної системи мають першорядне значення. Помилки у визначенні координат можуть призвести до порушення заборонених зон (наприклад, поблизу аеропортів) або до зіткнення з перешкодами, створюючи загрозу безпеці.

Тому до навігаційних систем висуваються вимоги високої точності, стійкості до різких змін умов середовища (освітлення, погоди, магнітних завад), роботи в режимі реального часу та достатнього радіусу дії для виконання місії [1].

Для групових польотів (роїв дронів) додається вимога масштабованості - здатності координувати значну кількість апаратів одночасно без втрати точності (підтримання строю, недопущення зіткнень між дронами) [3]. Досягнення усіх цих вимог можливе лише за рахунок поєднання багатьох технологій. Наприклад, використання GPS разом із локальними сенсорами та алгоритмами штучного інтелекту для аналізу сцени дозволяє створити інтегровану інтелектуальну навігаційну систему, здатну ефективно діяти в складних умовах.

Сучасні інтелектуальні навігаційні комплекси дійсно комбінують супутникову навігацію з бортовими сенсорами та методами AI, що уможливлює виконання місій навіть за несприятливих умов.

1.2 Огляд методів комп'ютерного зору та штучного інтелекту для відстеження руху і реконструкції траєкторії

Комп'ютерний зір є досить важливим у задачах навігації дрона, особливо коли йдеться про визначення траєкторії руху без зовнішніх орієнтирів (наприклад, у GPS-недоступному середовищі). Російсько-

українська війна наочно продемонструвала вразливість безпілотних літальних апаратів до засобів радіоелектронної боротьби.

Глушіння та спуфінг GPS-сигналів призводять до помилок позиціювання та втрати керованості [4, с. 7]. Це підсилює інтерес до оптичної навігації - підходу, що визначає положення дрона за зображеннями з бортових камер і є стійким до радіоперешкод.

Історично для автономної навігації БПЛА застосовували лідарні або радарні сенсори, проте завдяки розвитку технологій основний акцент змістився на візуальні методи, які використовують дешевші й легші камери.

Такий зсув зумовлений появою доступніших і досконаліших компонентів для дронів. Зорові сенсори також мають ту перевагу, що можуть пасивно збирати багату інформацію про оточення і не випромінюють власних сигналів - отже, не видають присутності дрона (на відміну від активних радарів) [5, с. 4703].

Використовується сценаріях, де помітність апарата небажана (наприклад, військові або приховані операції).

Одним із фундаментальних завдань комп'ютерного зору для навігації є оцінка власного руху дрона на основі аналізу зображень - тобто визначення, як змінюється положення і орієнтація апарата при переміщенні. Існують два тісно пов'язані підходи до такої візуальної одометрії: візуальна одометрія та візуальна одночасна локалізація і картографування.

Візуальна одометрія полягає у покроковому визначенні траєкторії руху за послідовністю кадрів з камери [4]. Іншими словами, VO дає оцінку его-руху дрона: на кожному кроці алгоритм прогнозує, наскільки і в якому напрямку змінилося положення камери відносно попереднього кадру.

Як правило, візуальна одометрія працює інкрементно, у режимі реального часу, і не будує глобальної карти місцевості. Це робить її швидшою та простішою за повноцінний SLAM. Зазвичай типовий VO включає: виявлення особливих точок на зображенні, зіставлення цих точок між

кадрами, оцінку перетворення камери (обчислення зміщення та повороту) та уточнення траєкторії.

Візуальна одометрія може слугувати проміжним етапом у складніших системах одночасної локалізації та картографування. Її недоліком є неминуче накопичення похибки з часом - навіть малі помилки у оцінках руху поступово додаються, що призводить до дрейфу траєкторії (поступового розходження розрахованого шляху з реальним положенням дрона).

Щоб зменшити дрейф, VO часто доповнюють даними інерціальних сенсорів, формуючи візуально-інерціальну одометрію: злиття сигналів камери та IMU допомагає визначити масштаб руху і підвищити точність відстеження [4, с. 24]. Прикладом є система VINS-Mono, яка об'єднує камеру і акселерометри/гіроскопи та демонструє високу точність позиціювання в режимі реального часу; подібні рішення вже знаходять застосування в комерційних дронах.

Visual SLAM вирішує дві задачі одночасно: локалізує дрон у просторі та будує карту навколишнього середовища. SLAM-алгоритм співставляє візуальні ознаки (ключові точки, контури тощо) між кадрами, щоб зрозуміти, де камера знаходиться відносно раніше побачених об'єктів, і паралельно додає ці об'єкти на карту.

На відміну від чистої одометрії, SLAM використовує глобальну оптимізацію траєкторії та спеціальні механізми корекції. Зокрема, завдяки детекції циклів SLAM здатен виправляти накопичений дрейф: коли дрон повертається в уже відвідану область, алгоритм впізнає знайомі орієнтири і коригує свою карту та траєкторію, «закриваючи» петлю [6].

Таким чином, сучасні SLAM-системи демонструють дуже високу точність - похибка відносно пройденого шляху може становити менше 1% саме завдяки замиканню циклів. SLAM є компонентом автономної навігації роботів - він дозволяє дрону одночасно орієнтуватися і досліджувати невідоме середовище. Ціна цієї функціональності - вищі обчислювальні витрати та складність алгоритму порівняно з одометрією: візуальний SLAM має значно

більшу затримку обробки (сотні мілісекунд) і потребує потужнішого апаратного забезпечення.

У малогабаритних дронах з обмеженими ресурсами повноцінне виконання SLAM в реальному часі часто є проблематичним без апаратного прискорення або спрощень.

Для вирішення задач VO і SLAM у комп'ютерному зорі розроблено два основні сімейства методів: традиційні (геометричні) та на основі глибокого навчання. Розглянемо спочатку традиційні підходи.

Більшість класичних алгоритмів VO/SLAM базуються на виявленні та відстеженні особливостей зображення - помітних точок, кутів, текстур на кадрах. Наприклад, відомі алгоритми виявлення ознак SIFT і SURF дозволяють надійно знаходити однакові точки на послідовних кадрах та зіставляти їх [4, с. 24].

За відповідностями між такими точками можна розрахувати відносне переміщення камери у просторі: алгоритм оцінює трансформацію (зсув і поворот), яка описує зміну положення камери між двома кадрами, використовуючи координати пар відповідних ознак. [4, с. 24]

Методи на основі ознак вразливі до хибних відповідностей, тому під час оцінки руху застосовуються процедури відфільтровування «викидів» на кшталт RANSAC. Feature-based SLAM - підхід, що опирається на такі точки, ліг в основу перших успішних систем: прикладами є MonoSLAM та PTAM, який запропонував розділити задачу на два паралельні потоки - відстеження ознак і побудову карти.

Значного прогресу досягнуто з появою алгоритму ORB-SLAM, який використовує швидкі бінарні дескриптори ORB для виділення ознак і включає повноцінний цикл локалізації, картографування та замикання циклів. ORB-SLAM2 став одним з еталонів візуального SLAM для дронів, продемонструвавши точне відстеження траєкторії в режимі реального часу на звичайному апаратному забезпеченні.

Його наступник, ORB-SLAM3, розширив можливості (підтримує візуально-інерціальний і мультимап SLAM) і вже знайшов застосування у комерційних безпілотноїках. [4, с. 24]

Натомість методи прямого відображення відмовляються від явного виділення дискретних точок і працюють безпосередньо з інтенсивностями пікселів у всьому зображенні.

Вони належать до так званих *appearance-based* підходів, оскільки визначають рух камери, намагаючись узгодити яскравість пікселів між кадрами без пошуку окремих ознак. Такі методи ефективні навіть на малотекстурованих або розмитих зображеннях.

Представник цього сімейства - алгоритм DSO. DSO виконує оптимізацію положення камери, намагаючись безпосередньо вирівняти інтенсивності (яскравість) пікселів на перекритих ділянках послідовних кадрів, відібраних у вигляді розрідженого набору точок. Інший приклад - попередній алгоритм LSD-SLAM, що продемонстрував можливість прямого побудування великомасштабної карти місцевості (*semi-dense картографування*) без використання детекторів ознак.

Прямі методи при правильних умовах уникають втрат інформації (адже не відкидають «невибрані» пікселі, як це роблять методи з ознаками), проте вони дуже чутливі до змін умов освітлення і експозиції камери. Сучасні системи часто комбінують обидва підходи: наприклад, гібридні методи поєднують переваги прямих і непрямих технік [4, с. 25].

Одним із прикладів є напівпряма візуальна одометрія), де побудова карти та оцінка руху виконуються паралельно - це дозволяє системі працювати в реальному часі навіть на обмежених обчислювальних ресурсах [4, с. 23]. Окрім того, до VO/SLAM можуть додавати відстеження інших геометричних примітивів (прямих ліній, площин), щоб покращити стійкість у середовищах з малою кількістю точкових особливостей.

Усуваючи слабкі місця (наприклад, нестачу текстур), такі удосконалення зменшують дрейф траєкторії навіть у складних умовах руху дрона.

Другий напрям - застосування штучного інтелекту, а саме глибокого навчання, для задач навігації за візуальними даними. Глибинні нейронні мережі сьогодні використовуються як для покращення окремих компонентів класичного SLAM, так і для побудови повністю нових систем.

Один із підходів - використання нейромереж для виділення та відбору ознак на зображеннях замість ручних детекторів на кшталт SIFT/ORB. Сучасні нейронні екстрактори ознак можуть бути навчені так, щоб забезпечувати надійніше співставлення кадрів при розмитті або швидкому русі, де класичні алгоритми нестабільні.

Ба більше, нейромережі здатні автоматично відфільтровувати менш інформативні чи помилкові ознаки, підвищуючи робастність одометрії. [6]

Інший підхід - пряме глибинне оцінювання руху: мережа «end-to-end» приймає послідовність зображень і напряму видає оцінку зміщення і повороту камери між кадрами. Такі моделі, відомі як Deep VO, навчаються на великій кількості відеопослідовностей з відомою траєкторією.

Вони здатні опановувати складні нелінійні залежності, недоступні класичним методам, і враховувати контекст сцени або повторювані візерунки, які можуть заплутати традиційний SLAM. Для прикладу за допомогою двох спарених нейронних мереж - однієї для оцінки карти глибини сцени, а другої для оцінки відносної позики (пози) камери - можна навчити систему, яка прогнозує послідовність переміщень дрона, нав'язуючи геометричні обмеження узгодженості між кадрами.

Такий підхід дозволив отримувати масштабно-сталі траєкторії дрона навіть у разі використання лише монокулярної камери, наближаючи точність одометрії до рівня, конкурентного класичним методам.

Хоча методи глибинного навчання помітно підвищують надійність відстеження руху (скажімо, зменшуючи дрейф на швидких маневрах або при

вібраціях платформи), повністю позбутися накопичуваної похибки поки що не вдається.

Тому на практиці нейронні підходи часто інтегрують із класичними: наприклад, застосовують глибоке навчання для детекції замикання петлі - автоматичного розпізнавання того, що дрон повернувся в уже відоме місце, або для побудови семантичних карт (виділення на відео категорій об'єктів, поверхонь тощо), що допомагає планувати траєкторію більш усвідомлено.

Використання нейромереж для семантичної сегментації сцени, зокрема, дає змогу відфільтровувати нерелевантні об'єкти і зосередитися на значущих орієнтирах, підвищуючи адаптивність та «інтелектуальність» навігаційної системи. [6]

Особливо перспективним вважається поєднання класичних геометричних алгоритмів зі сучасними методами машинного навчання, що дозволяє комбінувати точність математично обґрунтованих моделей із здатністю до адаптації й узагальнення, властивою нейронним мережам.

Такі гібридні рішення вже сьогодні демонструють найкращі результати і мають потенціал стати основою наступного покоління навігаційних систем, здатних працювати у непередбачуваних або складних умовах. [6]

Окрім завдань локалізації, методи комп'ютерного зору та ШІ широко застосовуються для виявлення і відстеження об'єктів у сцені, що також є важливим для навігації. Наприклад, щоб уникати зіткнень, дрон повинен завчасно побачити перешкоду (будівлю, дроти електропередач, інший літальний апарат) і оцінити її відносний рух.

Сучасні алгоритми глибинного навчання, наприклад сімейство моделей YOLO, здатні в реальному часі знаходити на зображенні об'єкти певних класів (людей, машини, дерева тощо), а спеціалізовані трекінгові алгоритми можуть супроводжувати (відстежувати) рухомі цілі. У сукупності ці технології дозволяють реалізувати на БПЛА нові функції, наприклад автоматичне переслідування обраного об'єкта чи обліт рухомих перешкод.

Завдяки впровадженню моделей на зразок YOLO на бортових комп'ютерах (навіть малих, як Raspberry Pi) дрони вже зараз можуть виявляти перешкоди та відстежувати об'єкти в режимі реального часу, розширюючи свої можливості у таких сферах, як доставка вантажів та безпека [7].

Інтелектуальна система навігації може, приміром, розпізнати наближення іншого дрона по відео і своєчасно скоригувати курс, щоб безпечно розминутися. Такі можливості істотно підвищують автономність і безпечність використання БПЛА у спільному повітряному просторі з іншими об'єктами.

Тож, методи комп'ютерного зору і штучного інтелекту наразі є складовою інтелектуальних навігаційних систем. Поєднання традиційних геометричних алгоритмів (візуальної одометрії, SLAM) з новітніми підходами глибинного навчання дозволяє дронам надійно відстежувати власний рух та відновлювати пройденої траєкторію навіть у складних умовах - при відмові GPS, швидких маневрах або за наявності візуальних перешкод.

Ці технології продовжують удосконалюватися, наближаючи нас до повністю автономних дронів, здатних орієнтуватися настільки ж упевнено, як це робить людина, а то й краще.

1.3 Аналіз існуючих програмних рішень та середовищ симуляції для моделювання польотів БПЛА

Розробка та тестування алгоритмів навігації безпілотних літальних апаратів тісно пов'язані з використанням програмних симуляторів, які стали невід'ємним інструментом сучасних досліджень у галузі автономних систем. Симуляційні середовища дають змогу відпрацьовувати польоти дронів у віртуальному просторі, що забезпечує безпечний, гнучкий та економічно доцільний підхід до перевірки інтелектуальних алгоритмів перед їх застосуванням у реальних умовах.

Особливої актуальності це набуває при використанні методів машинного навчання, оскільки збирання великого обсягу реальних польотних даних пов'язане з високими фінансовими витратами та ризиком пошкодження апаратури, тоді як якісна симуляція дозволяє отримувати практично необмежені набори даних для навчання моделей без фізичних втрат [8, с. 80]. Крім того, симулятори надають можливість тестувати роботу БПЛА в екстремальних або аварійних сценаріях, таких як сильні пориви вітру, відмови сенсорів чи нештатні ситуації, відтворення яких у реальному середовищі є складним або небезпечним.

На сьогодні існує значна кількість програмних платформ для моделювання польотів БПЛА, серед яких можна виділити як універсальні робототехнічні симулятори, так і спеціалізовані середовища, орієнтовані безпосередньо на дрони. Кожне з таких середовищ має власні особливості та обмеження, і жоден із симуляторів не здатен одночасно задовольнити всі потреби дослідників.

астина інструментів забезпечує високий рівень фізичної достовірності, проте має обмежені можливості з імітації сенсорів або масштабування на велику кількість апаратів; інші, навпаки, підтримують моделювання групового або ройового польоту, але використовують спрощені моделі динаміки чи зіткнень.

Таким чином, вибір симуляційного середовища завжди пов'язаний з компромісом між точністю фізичного моделювання, обчислювальною швидкістю та зручністю використання [8, с. 80].

Одним із найбільш поширених відкритих симуляторів робототехніки є Gazebo, який широко використовується для моделювання наземних, підводних і літальних роботів. Це середовище забезпечує реалістичну фізику руху та зіткнень і має модульну архітектуру, що дозволяє підключати різноманітні сенсори, зокрема камери, лідари, GPS та інерціальні вимірювальні блоки.

Тісна інтеграція Gazebo з фреймворком ROS дає змогу безпосередньо переносити розроблені в симуляції алгоритми на реальні апарати, що істотно скорочує цикл розробки. Gazebo ефективно використовується для відпрацювання автономних польотів у складних сценаріях, включно з моделюванням ройових систем або операцій зльоту та посадки на рухомі платформи [9].

Важливою перевагою цього симулятора є можливість підключення реальних автопілотів, таких як ArduPilot або PX4, у режимі Software-In-The-Loop, що дозволяє отримувати результати тестування, максимально наближені до поведінки реального дрона.

Іншим широко відомим середовищем є AirSim - фотореалістичний симулятор, розроблений компанією Microsoft для досліджень у сфері штучного інтелекту та автономних систем. AirSim побудований на ігровому рушії Unreal Engine, що забезпечує високий рівень деталізації віртуального середовища та фізично коректне освітлення.

Симулятор підтримує моделювання широкого спектра сенсорів, зокрема RGB- та стереокамер, глибинних сенсорів і лідарів, а також дозволяє імітувати погодні умови, такі як туман або дощ [8, с. 84]. Завдяки зручному програмному інтерфейсу AirSim активно використовується для навчання та тестування нейронних мереж, зокрема в задачах комп'ютерного зору та навігації за зоровими даними.

Висока якість візуальної інформації робить цей симулятор особливо цінним для розробки алгоритмів детекції об'єктів, оцінки глибини та SLAM. Разом з тим, фотореалістичність AirSim супроводжується значними вимогами до обчислювальних ресурсів, що може істотно знижувати швидкість симуляції на звичайних персональних комп'ютерах [8].

Для подолання цього обмеження були запропоновані рішення з паралельним запуском кількох інстанцій симулятора, зокрема розширення PRL4AirSim, орієнтоване на прискорення навчання алгоритмів навчання з

підкріплення. Хоча проєкт Project AirSim згодом було припинено, відкриту версію AirSim продовжує підтримувати спільнота розробників.

Загалом AirSim доцільно розглядати як ефективний інструмент для задач, у яких роль відіграє якість візуальних даних та реалістичність середовища, за умови наявності відповідного апаратного забезпечення.

Симулятор Flightmare є сучасним програмним середовищем, спеціально оптимізованим для навчання безпілотних літальних апаратів методами глибокого навчання з підкріплення та для досліджень, пов'язаних із високошвидкісними режимами польоту у задачах автономного дрон-рейсингу. Flightmare був розроблений у ETH Zurich і спроектований з урахуванням потреб наукових експериментів, де вимогами є висока продуктивність, масштабованість і контроль над фізичною моделлю [8, с. 85].

Архітектура цього симулятора принципово відрізняється від багатьох традиційних середовищ і складається з двох логічно та технічно розділених компонентів: фізичного рушія та рушія рендерингу, що можуть функціонувати як спільно, так і незалежно.

Графічний рушій Flightmare базується на платформі Unity і відповідає за генерацію візуальних даних. Він здатен створювати фотореалістичні або спрощені зображення навколишнього середовища, доповнені моделями шумів сенсорів, атмосферних ефектів і варіацій освітлення, що є особливо важливим для задач зорової навігації [8, с. 85].

Водночас цей рушій не прив'язаний жорстко до фізичної симуляції, що дозволяє відключати рендеринг або спрощувати графіку задля істотного підвищення швидкості обчислень. Такий підхід є важливим у задачах машинного навчання, де необхідно виконувати тисячі або мільйони симуляційних кроків за обмежений час.

Фізичний модуль Flightmare забезпечує детальний контроль над динамікою руху дрона і підтримує різні рівні складності моделі - від ідеалізованих рівнянь руху без шумів до розширених моделей, що враховують аеродинамічні ефекти, опір гвинтів, сили тертя та інші нелінійні фактори [8, с.

85]. Він дозволяє дослідникам гнучко обирати компроміс між фізичною достовірністю та обчислювальною ефективністю залежно від цілей експерименту.

Наприклад, для навчання політик керування часто запускають велику кількість паралельних фізичних симуляцій без графічного виводу, збираючи лише числові дані траєкторій і станів, тоді як для тестування зорових алгоритмів використовують реалістичний рендеринг за спрощеної фізики.

Завдяки такій модульній архітектурі Flightmare досяг високих показників продуктивності та став одним із провідних інструментів у дослідженнях автономного маневрування дронів на високій швидкості. За даними публікацій, симулятор здатний забезпечувати стабільну частоту оновлення фізичної моделі на рівні сотень герц навіть на персональних комп'ютерах середнього класу, що робить його придатним для навчання складних політик керування у режимі прискореної симуляції [8, с. 85].

Водночас до недоліків Flightmare можна віднести більш складний процес початкового налаштування, порівняно меншу спільноту користувачів у порівнянні з Gazebo чи AirSim, а також відсутність великої кількості готових сцен і моделей довкілля, що змушує дослідника самостійно створювати необхідні віртуальні середовища у Unity.

Попри це, для наукових експериментів, де головними є швидкість симуляції, контроль над моделлю та можливість глибокої кастомізації, Flightmare вважається одним із найбільш доцільних рішень.

Іншим прикладом нетрадиційного підходу до симуляції є FlightGoggles - спеціалізований симулятор, розроблений у Массачусетському технологічному інституті для досліджень автономних польотів швидкісних дронів. Основна ідея FlightGoggles полягає у реалізації концепції «віртуальної реальності для дрона», яка поєднує реальний фізичний політ із віртуальним сприйняттям оточення [8, с. 85].

У цій системі справжній дрон літає в порожньому фізичному приміщенні, оснащеному системою захоплення руху (motion capture), яка з

високою точністю визначає його положення та орієнтацію у просторі. Відеопотік з камер дрона в реальному часі замінюється комп'ютерно згенерованим зображенням віртуального середовища.

Фактично це означає, що дрон фізично взаємодіє з реальним повітрям, інерцією та аеродинамікою, але «бачить» перед собою штучно створений світ, згенерований графічним рушієм Unity.

Такий підхід дозволяє надзвичайно точно відтворювати сигнали камер, лідарів та інерціальних сенсорів у замкненому контурі з реальним апаратним забезпеченням. Перевагою FlightGoggles є можливість тренувати та тестувати алгоритми автономного польоту в граничних режимах, зокрема при дуже високих швидкостях або складних маневрах: у разі помилки дрон фізично не зіштовхується з перешкодою, оскільки вона існує лише у віртуальному просторі.

Окрім вищезгаданих, існує чимало інших симуляційних інструментів. Webots - крос-платформний робототехнічний симулятор з графічним інтерфейсом, зручний для освітніх цілей і швидкого прототипування.

Він підтримує модель квадрокоптера та базові фізичні закони, хоча і не спеціалізується на дронах. Morse - мінімалістичний симулятор на базі Blender, який в минулому використовувався для роботів (нині проєкт припинено). Користувацькі симулятори: багато виробників дронів мають власні програми для тренування пілотів (наприклад, DJI Simulator), але вони більше орієнтовані на ручне керування, ніж на алгоритмічну автономність.

MATLAB/Simulink надає засоби моделювання динаміки БПЛА і синтезу алгоритмів керування - аж до автоматичної генерації коду контролера; хоча це середовище менш придатне для імітації реального середовища і сенсорів, його часто застосовують на етапі розробки теоретичних моделей та контролерів.

Хмарні симулятори (типу Microsoft Project AirSim чи середовищ Unity3D) починають пропонувати послуги масового паралельного моделювання польотів, що може пришвидшити навчання нейромереж у майбутньому.

Завершуючи огляд, зазначимо, що симуляція стала інструментом в розвитку інтелектуальних систем навігації БПЛА. Правильно підібраний симулятор дозволяє перевірити алгоритми у різних сценаріях, відібрати оптимальні параметри та виявити потенційні проблеми ще до підняття дрона в повітря.

При виборі симулятора дослідники враховують специфіку свого завдання. Якщо точність фізики - обирають рушії на кшталт Gazebo; якщо важливі фотореалістичні зображення - звертаються до AirSim. Для експериментів з групами дронів шукають ті, що підтримують багатокористувацькі сценарії; для навчання нейромереж - ті, що можуть запускатися паралельно та швидко.

Попри різноманіття доступних платформ, можна стверджувати, що жоден одиничний симулятор не є універсальним, і часто в дослідженнях використовують комбінацію інструментів.

Наприклад, алгоритм можуть спочатку перевірити в Simulink (на рівні моделі), потім у Gazebo (із реальним автопілотом у контурі), а фінальне доведення здійснити в AirSim для отримання синтетичних даних навчання. Таким чином, симуляційні середовища істотно прискорюють цикл розробки, дозволяючи більш комплексно підійти до створення «розумних» навігаційних систем для дронів.

2 АНАЛІЗ ТА ШЛЯХИ РОЗВ'ЯЗАННЯ ПРОБЛЕМИ ВИЗНАЧЕННЯ ТРАЄКТОРІЇ ПОЛЬОТУ БПЛА

2.1 Постановка задачі та математична модель руху БПЛА

У межах інтелектуальної навігаційної системи безпілотного літального апарата задача визначення траєкторії польоту є однією з базових. Вона полягає у побудові такого маршруту руху, який забезпечує гарантований перехід апарата з початкового стану до заданої цільової конфігурації з урахуванням фізичних, динамічних та середовищних обмежень. Формально рух БПЛА відбувається у тривимірному евклідовому просторі:

$$X \subset R^3,$$

який складається з допустимої для польоту області X_{free} та множини заборонених областей, що відповідають фізичним перешкодам:

$$X_{obs} \subset X, X_{free} = X \setminus X_{obs}.$$

Положення БПЛА в довільний момент часу $t \in [0, T]$ описується векторною функцією:

$$p(t) \in R^3,$$

яка визначає координати центра мас апарата в інерціальній системі координат. Таким чином, траєкторія польоту являє собою відображення:

$$p: [0, T] \rightarrow X_{free},$$

яке повинно задовольняти граничні умови:

$$p(0) = p_{start}, p(T) = p_{goal}.$$

На практиці траєкторія може задаватися у дискретному вигляді як скінченна послідовність опорних точок:

$$W = \{w_1, w_2, \dots, w_N\}, w_i \in R^3,$$

або у неперервному вигляді як гладка крива $p(t)$, що є більш зручною формою для аналітичного аналізу та оптимального керування. У будь-якому випадку вимогою є те, що на всьому проміжку часу траєкторія не повинна перетинати заборонені області:

$$p(t) \notin X_{obs}, \forall t \in [0, T].$$

З формальної точки зору задача планування траєкторії може бути розглянута як задача оптимального керування або оптимізаційна пошукова задача у просторі конфігурацій. У загальному вигляді вона формулюється як задача мінімізації функціонала якості:

$$\min_{u(t), T} J = \int_0^T L(x(t), u(t)) dt,$$

де $x(t)$ - вектор стану БПЛА, $u(t)$ - вектор керуючих впливів, а $L(\cdot)$ - щільність функціонала вартості, яка може відображати час польоту, довжину траєкторії, сумарні енерговитрати або їх комбінацію.

Мінімізація здійснюється за умов динаміки системи:

$$\dot{x}(t) = f(x(t), u(t)),$$

та граничних умов:

$$x(0) = x_{start}, x(T) = x_{goal}.$$

Таким чином, задача планування траєкторії нерозривно пов'язана з математичною моделлю руху конкретного БПЛА, оскільки будь-яка допустима траєкторія повинна бути фізично реалізованою з урахуванням обмежень на прискорення, швидкість, кути орієнтації та доступні керуючі дії.

Для подальшого аналізу розглянемо математичну модель руху квадрокоптера як одного з найбільш поширених типів БПЛА. Квадрокоптер є жорстким літальним апаратом з чотирма незалежно керованими гвинтами, який рухається у тривимірному просторі та має шість ступенів вільності - три поступальні та три обертальні.

Для коректного опису динаміки вводяться дві системи координат:

1) інерціальна система координат:

$$O_I X_I Y_I Z_I,$$

жорстко пов'язана із Землею;

2) тілесна (зв'язана з апаратом) система координат:

$$O_B X_B Y_B Z_B,$$

початок якої збігається з центром мас квадрокоптера, а вісь Z_B спрямована вздовж осі сумарної тяги пропелерів.

Стан динамічної системи квадрокоптера описується вектором:

$$x = (p \ v \ R \ \Omega),$$

де

$$p = (x, y, z)^T \in R^3$$

- координати центра мас у глобальній системі координат:

$$v = \dot{p}$$

- вектор лінійної швидкості:

$$R \in SO(3)$$

- матриця повороту, що задає орієнтацію апарата:

$$\Omega = (p, q, r)^T$$

- вектор кутової швидкості в тілесній системі координат.

Керування квадрокоптером зручно описувати через узагальнений вектор

$$u = (f, M),$$

де f - сумарна сила тяги, що створюється всіма чотирма гвинтами, а $M = (M_x, M_y, M_z)^T$ - вектор керуючих моментів, які відповідають за крен, тангаж та ролання.

Динаміка квадрокоптера описується системою рівнянь Ньютона-Ейлера:

$$\dot{p} = v, m\dot{v} = mg - fRe_3, \dot{R} = R\hat{\Omega}, J\dot{\Omega} + \Omega \times (J\Omega) = M,$$

де

$$g = (0, 0, -g)^T, e_3 = (0, 0, 1)^T, \hat{\Omega} = \begin{pmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{pmatrix}.$$

Перші два рівняння описують поступальний рух центра мас квадрокоптера. Прискорення визначається сумою сил тяжіння та тяги, причому множник Re_3 здійснює перетворення вектора тяги з тілесної системи координат у глобальну. Це означає, що нахил апарата безпосередньо впливає на напрям прискорення, дозволяючи створювати горизонтальні складові руху.

Третє рівняння описує кінематику орієнтації: матриця повороту змінюється відповідно до поточних кутових швидкостей. Оператор $\hat{\Omega}$ є скошеною матрицею, яка реалізує векторний добуток.

Четверте рівняння визначає обертальний рух апарата. Зміна кутової швидкості залежить не лише від прикладених керуючих моментів, а й від гіроскопічних ефектів, що виникають внаслідок обертання жорсткого тіла.

Наведена система рівнянь являє собою повну нелінійну динамічну модель квадрокоптера у векторно-матричній формі, яка без спрощень описує як поступальний, так і обертальний рух літального апарата у тривимірному просторі.

Ця модель є узагальненою і охоплює всі основні режими польоту, що мають практичне значення для автономної навігації БПЛА.

Зокрема, режим вертикального підйому або зависання реалізується за умови:

$$Re_3 = (0 \ 0 \ 1), f \approx mg,$$

коли вісь тяги квадрокоптера співпадає з вертикальною віссю інерціальної системи координат, а сумарна тяга компенсує силу тяжіння. У цьому випадку горизонтальні складові прискорення відсутні, і дрон або залишається нерухомим у просторі, або змінює висоту вздовж осі Z_I .

Поступальний політ у горизонтальній площині виникає внаслідок нахилу апарата, тобто при:

$$Re_3 \neq e_3,$$

що призводить до появи горизонтальної компоненти сили:

$$F_{hor} = -f(Re_3)_\perp,$$

яка забезпечує прискорення вздовж осей X_I, Y_I . Саме цей механізм лежить в основі керування швидкістю та напрямком руху квадрокоптера в просторі.

Обертальні рухи навколо осей апарата (крен, тангаж, ролання) реалізуються за рахунок керуючих моментів:

$$M = (M_x, M_y, M_z)^T,$$

які змінюють кутову швидкість відповідно до рівнянь Ейлера:

$$J\dot{\Omega} = M - \Omega \times (J\Omega).$$

Таким чином, наведена модель дозволяє відтворити всі маневри, які може виконувати реальний квадрокоптер, включно з агресивними розворотами, швидкісними прискореннями та складними тривимірними траєкторіями.

На практиці для задач планування траєкторії часто застосовують спрощені варіанти повної моделі, оскільки пряме використання нелінійної системи високої розмірності є обчислювально складним. Одним із поширених підходів є моделювання дрона як матеріальної точки з обмеженнями на прискорення:

$$\ddot{p}(t) = a(t), \| a(t) \| \leq a_{max}.$$

У такій моделі кутова динаміка ігнорується, а керування зводиться до вибору допустимого прискорення, що значно спрощує пошук траєкторії.

Інший поширений підхід - лінійаризація повної моделі навколо горизонтального режиму польоту, тобто при малих кутах нахилу:

$$\phi \approx 0, \theta \approx 0,$$

унаслідок чого матриця повороту апроксимується як:

$$R \approx I + O(\phi, \theta),$$

а система рівнянь стає лінійною і придатною для застосування класичних методів лінійного керування (LQR, PID, MPC у лінійній формі).

Проте важливо підкреслити, що не кожна математично побудована траєкторія є фізично здійсненою. Реальний дрон має обмеження:

$$0 \leq f(t) \leq f_{max}, \| \Omega(t) \| \leq \Omega_{max}, \| M(t) \| \leq M_{max}.$$

Отже, допустима траєкторія повинна належати області досяжності динамічної системи:

$$p(t) \in R(x_{start}),$$

де R - множина станів, які можуть бути досягнуті з початкового стану з урахуванням обмежень керування.

Після задання математичної моделі руху задача планування траєкторії формулюється як високорозмірна оптимізаційна задача, оскільки повний стан квадрокоптера містить щонайменше 12 змінних:

$$x = (p, v, R, \Omega).$$

Безпосередній пошук оптимальної траєкторії у цьому просторі є обчислювально складним, тому на практиці застосовують декомпозицію задачі та різні рівні абстракції.

У робототехніці та навігації БПЛА прийнято розрізняти глобальне та локальне планування шляхів.

Глобальне планування виконується у статичному середовищі та не залежить від часу. Його метою є знаходження шляху:

$$\gamma: [0,1] \rightarrow X_{free},$$

який мінімізує певний критерій, наприклад довжину:

$$J_{global} = \int_0^1 \left\| \frac{d\gamma(s)}{ds} \right\| ds.$$

При цьому дрон часто моделюється як точка або геометричне тіло простого типу (куля, диск), а динаміка руху не враховується явно.

Локальне планування, або уникнення перешкод у реальному часі, вирішує задачу корекції траєкторії:

$$p_{new}(t) = p(t) + \Delta p(t),$$

коли в середовищі з'являються несподівані або динамічні перешкоди.

Тут оптимальність глобального маршруту може бути пожертвована заради безпеки та стабільності польоту.

У практичних системах ці підходи поєднуються: глобальний план слугує еталоном, а локальний модуль забезпечує адаптацію в реальному часі. У випадку групового польоту рою БПЛА задача додатково ускладнюється необхідністю уникнення взаємних зіткнень:

$$\| p_i(t) - p_j(t) \| \geq d_{min}, \forall i \neq j,$$

та оптимізації колективного критерію, наприклад мінімізації часу виконання місії:

$$\min \max_i T_i.$$

Розглянемо дискретні пошукові алгоритми.

Класичні алгоритми A*, Dijkstra, D*, D* Lite працюють на графі:

$$G = (V, E),$$

де вершини V відповідають дискретним станам або позиціям дрона, а ребра E - допустимим переміщенням. Алгоритм A* мінімізує функцію:

$$f(n) = g(n) + h(n),$$

де $g(n)$ - вартість шляху від старту, $h(n)$ - евристична оцінка відстані до цілі.

Для тривимірних середовищ такі алгоритми часто дають ламані траєкторії, тому результати згладжуються за допомогою сплайнів:

$$p(t) = \sum_{i=0}^k c_i B_i(t),$$

де $B_i(t)$ - базисні функції (B-сплайни, поліноми).

Евристичні та оптимізаційні методи. Методи GA, PSO, ACO, DE розглядають траєкторію як параметризований об'єкт:

$$p(t; \theta),$$

і здійснюють пошук параметрів θ , що мінімізують багатокритеріальну функцію:

$$J = \alpha J_{length} + \beta J_{energy} + \gamma J_{risk}.$$

Вони добре підходять для складних задач з нетривіальними обмеженнями, проте зазвичай не гарантують глобального оптимуму і мають високу обчислювальну складність.

До класу геометричних та аналітичних методів планування траєкторій належать підходи, які будують маршрут руху БПЛА шляхом задання конкретних геометричних кривих або аналітичних залежностей у просторі.

Ці методи зазвичай не працюють безпосередньо з повною динамічною моделлю апарата, а оперують його геометричним образом у просторі, що дозволяє значно зменшити обчислювальну складність задачі.

Одним із найвідоміших представників цього класу є метод штучних потенційних полів. У цьому підході середовище моделюється за допомогою скалярного потенціального поля:

$$U(p) = U_{att}(p) + U_{rep}(p),$$

де U_{att} - привабливий потенціал цільової точки, а U_{rep} - сумарний відштовхувальний потенціал перешкод.

Привабливий потенціал зазвичай задається у квадратичній формі:

$$U_{att}(p) = \frac{1}{2} k_{att} \| p - p_{goal} \|^2,$$

тоді як відштовхувальний потенціал для кожної перешкоди O_i має вигляд:

$$U_{rep}(p) = \begin{cases} \frac{1}{2} k_{rep} \left(\frac{1}{d_0} \right)^2, & d(p, O_i) \leq d_0, \\ 0, & d(p, O_i) > d_0, \end{cases}$$

де $d(p, O_i)$ - відстань до перешкоди, а d_0 - радіус впливу.

Рух дрона визначається напрямком антиградієнта сумарного потенціалу:

$$\dot{p} = -\nabla U(p),$$

що забезпечує інтуїтивно зрозумілу поведінку: апарат «тягнеться» до цілі та «відштовхується» від перешкод. Основною перевагою APF є обчислювальна простота і можливість застосування в реальному часі як локального планувальника. Водночас суттєвим недоліком є наявність локальних мінімумів, у яких:

$$\nabla U(p) = 0, p \neq p_{goal},$$

що може призводити до застрягання дрона між перешкодами. Удосконалені варіанти APF намагаються усунути цю проблему шляхом введення часової залежності потенціалів, стохастичних збурень або комбінування з іншими алгоритмами.

Іншим напрямом геометричних методів є аналітична генерація траєкторій заданої форми, зокрема за допомогою поліномів або сплайнів. Якщо відома послідовність опорних точок $\{w_i\}$, то траєкторія може бути задана, наприклад, поліномом:

$$p(t) = \sum_{k=0}^n a_k t^k,$$

коефіцієнти якого підбираються з умов інтерполяції та гладкості:

$$p(t_i) = w_i, \dot{p}(t_i), \ddot{p}(t_i) \text{ неперервні.}$$

Кубічні сплайни та криві Безьє широко застосовуються для формування референсних траєкторій, які згодом відпрацьовуються системою керування. Такі методи забезпечують гладкість і фізичну реалізованість руху, проте вимагають попереднього визначення точок маршруту, які зазвичай отримують іншими алгоритмами (наприклад, A* або RRT).

Методи оптимального керування формують задачу планування траєкторії безпосередньо у термінах рівнянь руху БПЛА. У загальному вигляді задача має вигляд:

$$\min_{u(t)} \int_0^T (\|x(t) - x_{ref}\|_Q^2 + \|u(t)\|_R^2) dt,$$

за умов:

$$\dot{x}(t) = f(x(t), u(t)), u(t) \in U, x(t) \in X_{free}.$$

Одним з найбільш практичних підходів є прогнозоване керування. У MPC на кожному часовому кроці k розв'язується оптимізаційна задача на скінченному горизонті N :

$$\min_{\{u_{k+i}\}_{i=0}^{N-1}} \sum_{i=0}^{N-1} l(x_{k+i}, u_{k+i}) + l_f(x_{k+N}),$$

з урахуванням дискретної динаміки:

$$x_{k+1} = f_d(x_k, u_k),$$

та обмежень:

$$u_{min} \leq u_k \leq u_{max}.$$

Перевагою MPC є явне врахування динамічних обмежень і генерація гладких керуючих сигналів. Проте розв'язання нелінійної оптимізаційної задачі в реальному часі вимагає значних обчислювальних ресурсів, що обмежує застосування MPC на борту мультикоптерів.

Sampling-based алгоритми, зокрема Rapidly-Exploring Random Trees, є одним із найпопулярніших інструментів для планування у просторах високої розмірності. Алгоритм RRT поступово будує дерево:

$$T = (V, E),$$

де вершини V відповідають досяжним станам, а ребра E - допустимим переходам між ними.

На кожному кроці випадковим чином генерується точка:

$$x_{rand} \sim X,$$

визначається найближча вершина дерева:

$$x_{near} = \arg \min_{x \in V} \|x - x_{rand}\|,$$

і виконується розширення:

$$x_{new} = x_{near} + \eta \frac{x_{rand} - x_{near}}{\|x_{rand} - x_{near}\|}.$$

RRT має властивість ймовірнісної повноти, тобто:

$$\lim_{n \rightarrow \infty} P(\text{знайдено шлях}) = 1,$$

якщо такий шлях існує.

Модифікація RRT* забезпечує асимптотичну оптимальність, поступово мінімізуючи функціонал вартості:

$$J(\gamma) = \int_{\gamma} \| \dot{p}(s) \| ds.$$

Для БПЛА RRT є особливо привабливим, оскільки дозволяє враховувати кінематичні обмеження через обмеження кроку росту дерева.

Методи навчання з підкріплення розглядають навігацію як марковський процес прийняття рішень:

$$M = (S, A, P, R, \gamma),$$

де агент навчається політики:

$$\pi(a | s) = P(a_t = a | s_t = s),$$

максимізуючи очікувану сумарну винагороду:

$$\max_{\pi} E \left[\sum_{t=0}^{\infty} \gamma^t r_t \right].$$

Хоча RL дозволяє навчати дронів складній поведінці без явного планування, відсутність гарантій оптимальності та узагальнюваності змушує комбінувати його з класичними методами. Гібридні схеми, у яких глобальний маршрут будується RRT або A*, а локальне ухилення виконується RL-політикою, демонструють високу ефективність у складних середовищах, зокрема для ройових систем БПЛА.

Повертаючись до поставленої задачі планування траєкторії польоту безпілотного літального апарата, можна зробити принциповий і добре відомий у робототехніці висновок: не існує єдиного універсального алгоритму, який би оптимально розв'язував задачу навігації БПЛА в усіх можливих умовах і середовищах.

Це пояснюється як різноманіттям сценаріїв застосування дронів, так і складністю самої задачі, що поєднує геометричні, динамічні, обчислювальні та інформаційні аспекти.

Формально задача планування траєкторії може бути подана як пошук відображення:

$$p(t): [0, T] \rightarrow X_{free},$$

яке мінімізує деякий функціонал якості:

$$J(p) = \alpha J_{length} + \beta J_{smooth} + \gamma J_{energy} + \delta J_{safety},$$

де різні складові відповідають довжині маршруту, плавності, енерговитратам та запасу безпеки. У різних практичних ситуаціях вагові коефіцієнти $\alpha, \beta, \gamma, \delta$ змінюються, що автоматично змінює і «найкращий» алгоритм розв'язання.

Саме тому в реальних навігаційних системах застосовується композиція методів планування, де кожен алгоритм відповідає за певний рівень абстракції або за конкретну підзадачу.

У простих і добре структурованих середовищах (наприклад, при польоті над відкритою місцевістю з відомою картою) достатньо застосування графових алгоритмів пошуку, таких як A^* , які мінімізують довжину шляху:

$$J_{A^*} = \sum_{(i,j) \in \gamma} c_{ij},$$

де c_{ij} - вартість переходу між вузлами графа.

У складніших середовищах, де можливі локальні перешкоди або неточності карти, глобальний маршрут доповнюється локальними алгоритмами уникнення перешкод, такими як Dynamic Window Approach (DWA) або штучні потенційні поля. Локальний планувальник у цьому випадку коригує команду руху:

$$v_{cmd} = v_{global} + \Delta v_{local},$$

де Δv_{local} визначається поточною сенсорною інформацією про оточення.

Важливу роль у композиції підходів відіграють методи згладжування траєкторії. Навіть якщо глобальний планувальник знаходить геометрично коректний шлях, він часто містить різкі злами, що є неприйнятними з точки зору динаміки БПЛА. Тому початкову ламану траєкторію $\{w_i\}$ перетворюють у гладку криву:

$$p(t) = \arg \min_p \left(\sum_i \|p(t_i) - w_i\|^2 + \lambda \int_0^T \|\ddot{p}(t)\|^2 dt \right),$$

де другий доданок мінімізує кривизну та забезпечує плавність руху. Такі методи часто називають trajectory smoothing і застосовують як постобробку результатів основного планування.

Обмежувальним фактором при цьому є математична модель руху дрона. Для того щоб траєкторія була фізично здійсненою, вона повинна задовольняти обмеження на прискорення:

$$\|\ddot{p}(t)\| \leq a_{max},$$

на кут нахилу апарата:

$$|\phi(t)| \leq \phi_{max}, |\theta(t)| \leq \theta_{max},$$

де для квадрокоптерів у безпечному режимі зазвичай:

$$\phi_{max}, \theta_{max} \approx 30^\circ - 40^\circ,$$

а також на швидкість зміни висоти:

$$|\dot{z}(t)| \leq v_{z,max}.$$

Ці обмеження означають, що надто різкі повороти або перепади висоти, знайдені геометричним планувальником, повинні бути автоматично скориговані. Наприклад, якщо глобальний маршрут містить кут повороту α , що перевищує допустиму кривизну:

$$\kappa = \frac{1}{R} > \kappa_{max},$$

навігаційна система замінює гострий злам дугою радіуса $R \geq R_{min}$, забезпечуючи поступовий маневр з урахуванням інерційних властивостей апарата.

Формально така корекція може бути подана як задача оптимізації:

$$\min_{p(t)} \int_0^T \|p(t) - p_{ref}(t)\|^2 dt,$$

за умов:

$$\ddot{p}(t) \in A, p(t) \in X_{free},$$

де $p_{ref}(t)$ - первісна (груба) траєкторія, а A - допустима область прискорень. Такі задачі часто зводяться до випуклого програмування, що дозволяє ефективно отримувати гладкі та контрольовані траєкторії.

У динамічних і непередбачуваних середовищах до цієї композиції методів додаються адаптивні та навчальні підходи, які дозволяють дрону реагувати на нові ситуації, не передбачені під час початкового планування.

Це особливо актуально при польотах у міському середовищі або при груповій взаємодії кількох БПЛА.

У рамках даної роботи основну увагу приділено теоретичному аналізу задачі планування траєкторії.

Було розглянуто математичний опис руху квадрокоптера, а також систематизовано основні класи методів визначення траєкторії - від геометричних і пошукових до оптимізаційних і навчальних.

2.2 Розробка алгоритму оцінювання координат та траєкторії руху БПЛА за даними відео і сенсорів руху

Задача оцінювання просторових координат і траєкторії руху БПЛА за даними відео та сенсорів руху належить до класу задач локалізації та оцінювання стану рухомого об'єкта. У практичній реалізації важливо забезпечити не лише формальну працездатність методу, а й відтворюваність результатів у реальних умовах (зміни освітлення, тремтіння камери, розмиття під час руху, пропуски вимірювань).

З огляду на вимогу отримання координат (X , Y , Z) або траєкторії в просторі, базовий алгоритм доцільно будувати так, щоб він мав абсолютний масштаб і не накопичував неконтрольованого дрейфу. Саме тому як найпростіший і реалістичний варіант пропонується підхід візуальної локалізації відносно фідуціального маркера ArUco відомого розміру, який широко застосовується для оцінювання пози (pose estimation) у задачах робототехніки та доповненої реальності [10].

Такий підхід належить до методів комп'ютерного зору, а вимогу “нейронна мережа або методи комп'ютерного зору” задовольняє саме за рахунок комп'ютерного зору.

Вхідні дані та узгодження систем координат. Вхідними даними алгоритму є відеопотік (або набір кадрів) з бортової/наземної камери, а за наявності - інерціальні дані IMU (акселерометр і гіроскоп), синхронізовані в часі. Для коректної інтерпретації координат необхідно зафіксувати системи відліку:

1) система координат маркера (умовно “світова” в рамках експерименту);

2) система координат камери;

3) за потреби - система координат БПЛА (корпуса).

У простішому демонстраційному сценарії вважається, що камера жорстко закріплена відносно БПЛА, тому координати камери однозначно характеризують і рух апарата з точністю до сталого зсуву/повороту між камерою та корпусом. Якщо ж камера є зовнішньою (наприклад, зйомка польоту з боку), тоді обчислені координати описують траєкторію саме камери у системі маркера, що також є валідною демонстрацією принципу.

Калібрування камери та модель проєкції. Для переходу від 2D-спостережень у зображенні до 3D-координат необхідні внутрішні параметри камери (матриця K) та коефіцієнти дисторсії. У загальному вигляді проєкційний зв'язок між 3D-точкою та її 2D-відображенням задається рівнянням перспективної проєкції:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R \mid t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

де (u,v) — координати точки на зображенні, (X,Y,Z) — координати точки в системі маркера, R - поворот і зсув, що задають перехід від об'єктної системи координат до камери.

У практичній реалізації ці параметри застосовуються у процедурі розв'язання задачі *PnP* (Perspective-n-Point), яка повертає вектори обертання та зсуву для заданих відповідностей 3D-2D. [12]

ArUco-маркер — це квадрат із чорним обрамленням та внутрішнім бінарним кодом, що забезпечує його ідентифікацію та контроль помилок.

На кожному кадрі відео виконується попередня обробка: перетворення у відтінки сірого, за потреби — легке згладжування для пригнічення шумів. Далі застосовується детектор маркерів, який повертає координати кутів маркера в площині зображення та його id.

Реалізаційно це підтримується модулем OpenCV ArUco, що включає як детекцію, так і інструменти для оцінювання пози по одному маркеру або по “дошці” з кількох маркерів. Оскільки геометрія маркера відома наперед (наприклад, квадрат зі стороною L), можна зафіксувати 3D-координати його кутів у системі маркера, наприклад:

$$(0,0,0), (L, 0,0), (L, L, 0), (0, L, 0),$$

тобто маркер розміщується у площині $Z=0$. Координати відповідних кутів у зображенні беруться з результатів детекції. Пара “3D кут — 2D кут” утворює набір відповідностей, достатній для розв’язання задачі PnP.

Функція розв’язання PnP (наприклад, *solvePnP*) повертає *rvec* та *tvec*, де *rvec* — компактне подання обертання (яке перетворюється в матрицю R через формулу Родрігеса), а *tvec* — вектор зсуву, що переводить точки з системи маркера до системи камери.

З практичної точки зору важливо правильно інтерпретувати ці величини: t описує положення маркера в координатах камери, якщо розглядати стандартну постановку “object-to-camera”.

Щоб отримати координати камери у системі маркера (що зазвичай і потрібно як траєкторія руху), використовується інверсія перетворення:

$$C = -R^T t,$$

де $C=(X,Y,Z)$ — координати камери в системі маркера. Саме послідовність таких координат для кожного кадру формує дискретну траєкторію руху.

Перевага цього підходу — абсолютний масштаб: якщо L задано в метрах (або сантиметрах), то й координати (X,Y,Z) будуть в тих самих одиницях. У порівнянні з чисто монокулярною одометрією без опорних об’єктів це істотно спрощує інтерпретацію результатів і зменшує дрейф.

На практиці детекція маркера може бути нестабільною через перекриття, вихід з поля зору, розмиття або зміну освітлення. Тому алгоритм має містити процедури обробки пропусків: якщо маркер не знайдено в кадрі, координата може бути позначена як відсутня, інтерпольована на короткому проміжку або оцінена прогнозом за попередньою швидкістю.

Для зменшення “тремтіння” траєкторії доцільно застосувати згладжування, наприклад експоненційне:

$$\hat{C}_k = \alpha C_k + (1 - \alpha) \hat{C}_{k-1},$$

де $\alpha \in (0,1)$ визначає компроміс між чутливістю та плавністю. Для демонстраційного програмного модуля цього достатньо, оскільки ціль — отримати читабельну 2D/3D-візуалізацію та файл результатів.

Дані гіроскопа та акселерометра можуть використовуватися для стабілізації оцінювання орієнтації та для короткочасного “продовження” траєкторії при втраті маркера. Найпростіший підхід — оцінювання орієнтації (кутів або кватерніона) за допомогою фільтра типу Madgwick, який поєднує вимірювання гіроскопа й напрямку гравітації з акселерометра та є обчислювально ефективним.

У рамках спрощеного алгоритму IMU можна використати двома способами. По-перше, як джерело “швидкої” орієнтації, щоб зменшити ривки у R , коли візуальна оцінка за маркером шумна.

По-друге, як засіб прогнозування: якщо маркер тимчасово зник, орієнтація з IMU дозволяє коректніше інтерпретувати зміну напрямку руху, а координату можна екстраполювати на короткому інтервалі до відновлення детекції.

При цьому слід підкреслити, що IMU не є обов’язковою для працездатності базового прототипу: візуальна локалізація по маркеру забезпечує координати та масштаб самостійно.

Після отримання послідовності координат $\{(X_k, Y_k, Z_k)\}$ виконується побудова траєкторії у вигляді графіка. Для звітності зручно показувати дві форми: 2D-проекцію на площину (наприклад, X-Y) та 3D-графік у просторі.

Паралельно координати зберігаються у файл CSV або JSON разом із номером кадру/часовою міткою, що забезпечує перевірюваність та можливість повторного аналізу.

Таким чином, алгоритм повністю відповідає вимогам: прийом відео/сенсорних даних, обробка методами комп'ютерного зору, оцінювання координат і траєкторії, графічне відображення і збереження результатів у файл.

У підсумку алгоритм функціонує як конвеєр:

- 1) зчитування кадру і (за наявності) IMU-даних;
- 2) попередня обробка зображення;
- 3) детекція ArUco та отримання 2D-координат кутів;
- 4) розв'язання PnP з урахуванням калібрування камери, отримання R, t ;
- 5) обчислення координат камери C в системі маркера та згладжування;
- 6) накопичення траєкторії;
- 7) візуалізація і експорт CSV/JSON.

Оскільки основні обчислення виконуються для невеликої кількості точок (кути маркера) і стандартних процедур, алгоритм є обчислювально легким та придатним для роботи на типовому ПК.

2.3 Структурна та функціональна модель програмного модуля визначення координат і траєкторії руху БПЛА

Програмний модуль визначення координат і траєкторії руху БПЛА призначений для обробки візуальних даних (відеопотоку або послідовності зображень) з метою оцінювання просторового положення платформи та формування траєкторії її переміщення.

У контексті поставленого завдання камера розглядається як сенсор, еквівалентний бортовій камері БПЛА: під час руху змінюється проєкція сцени

на матрицю, а ці зміни містять інформацію про переміщення та поворот платформи.

Основний принцип роботи модуля полягає у тому, що послідовні кадри з часовими мітками перетворюються на набір опорних ознак (візуальних маркерів або ключових точок), на основі яких оцінюється зміна положення між моментами часу, після чого формується послідовність координат (X, Y, Z) і траєкторія в просторі.

Таким чином, модуль реалізує повний цикл «вхідні дані → обробка → координати/траєкторія → візуалізація та збереження результатів», що відповідає функціональним вимогам технічного завдання.

Структурна модель програмного модуля відображає склад основних компонентів та логіку взаємодії між ними.

Як показано на рис. 2.1, обробка починається з підсистеми введення даних, яка завантажує послідовність зображень, пов'язані часові мітки та службову інформацію про сенсор.

Для практичної реалізації використовується формат «набір кадрів + файл відповідності timestamp-filename», що забезпечує відтворюваність обчислень і коректну синхронізацію результатів за часом.

Далі дані проходять етап попередньої обробки: приведення до відтінків сірого, масштабування (за необхідності), а також підготовка структур для покадрової обробки.

Попередня обробка потрібна для підвищення стійкості наступних алгоритмів до шуму та дрібних артефактів, а також для оптимізації швидкодії, оскільки більшість операцій комп'ютерного зору ефективніше виконувати в одноканальному представленні.

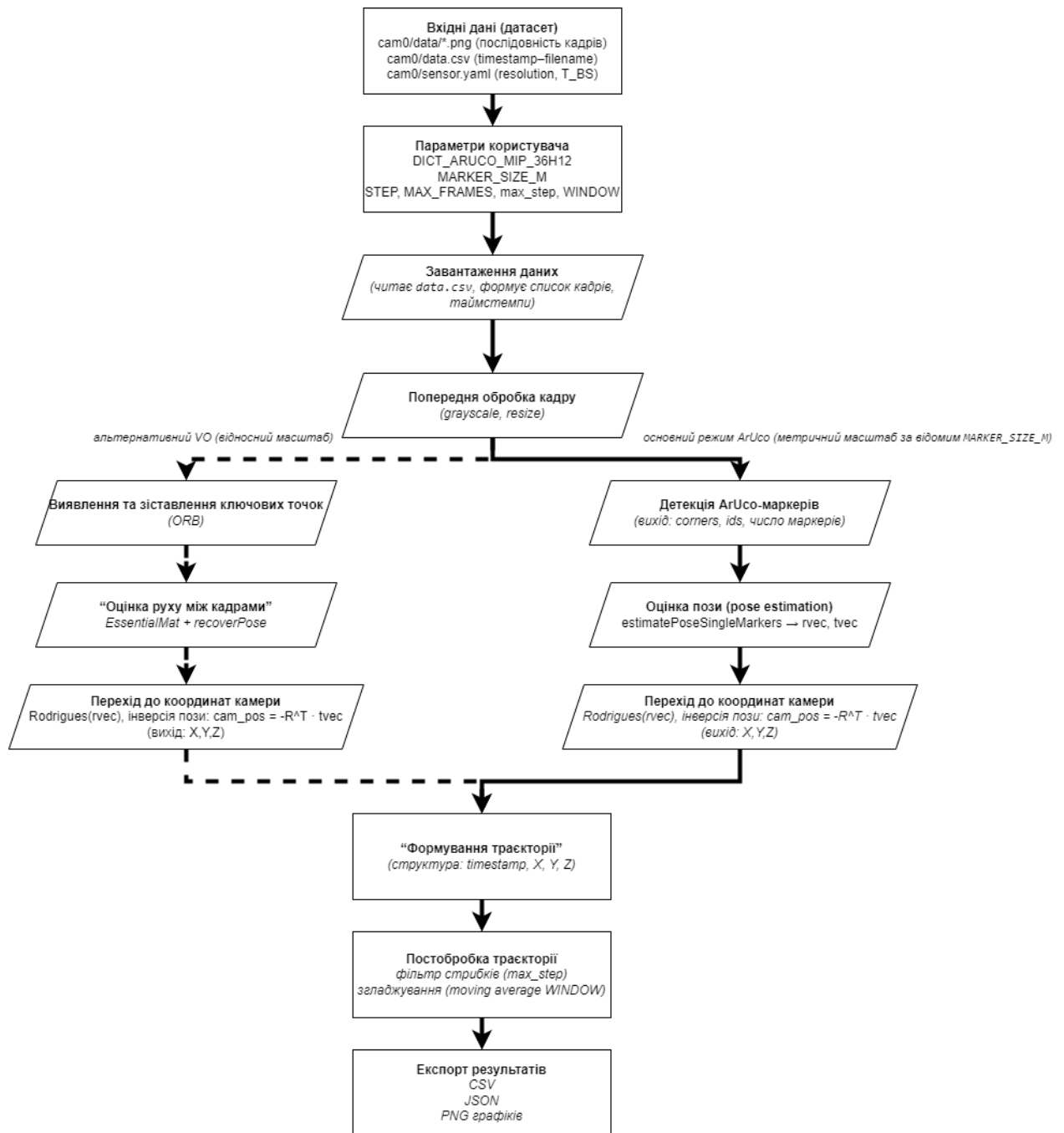


Рис. 2.1 - Структурна схема програмного модуля визначення координат і траєкторії руху БПЛА

Ключовим елементом структури є блок візуального аналізу, який може працювати в двох логічних режимах. Перший режим (основний для даної реалізації) ґрунтується на детекції маркерів типу ArUco у вхідних кадрах.

Саме цей підхід дозволяє отримувати координати в тривимірному просторі, оскільки маркер розглядається як відомий об’єкт зі стандартною

геометрією, а задача зводиться до оцінки пози (pose estimation) за перспективними спотвореннями на зображенні. У межах цього режиму модуль виконує детекцію маркерів заданого словника, визначає їхні кути в піксельних координатах, після чого переходить до оцінювання вектора обертання та вектора зсуву відносно камери.

Для підвищення стабільності обчислень у кадрі, де детектовано кілька маркерів, виконується вибір найбільш надійного екземпляра (наприклад, маркера з найбільшою площею проєкції), що зменшує вплив часткових перекриттів або погіршення видимості окремих міток.

Другий режим (альтернативний) базується на виявленні та зіставленні ключових точок між послідовними кадрами з подальшим відновленням відносного переміщення (візуальна одометрія).

Він дозволяє будувати траєкторію навіть без маркерів у сцені, однак зазвичай дає траєкторію у відносному масштабі. Наявність такої гілки в структурній схемі підкреслює універсальність модуля та можливість застосування на різних типах відеоматеріалів.

Після оцінки пози або переміщення між кадрами модуль переходить до побудови траєкторії. На цьому етапі формується часовий ряд точок у просторі: кожному timestamp відповідає трійка координат (X, Y, Z).

Для коректного накопичення координат застосовується узгоджена система координат, а результати приводяться до вигляду, зручного для подальшої обробки та інтерпретації.

Важливим структурним компонентом є блок постобробки траєкторії: він виконує відсікання аномальних стрибків (наприклад, якщо між сусідніми кадрами оцінено нереалістично велике переміщення) та, за потреби, згладжування координат простими методами (ковзне середнє).

Така постобробка не змінює принципового змісту траєкторії, однак істотно покращує її візуальну інтерпретацію і зменшує вплив випадкових похибок детекції.

Завершальними блоками структури є візуалізація та експорт: модуль будує 3D-графік траєкторії та 2D-проекції (на площини XY, XZ), а також зберігає результати у форматах CSV/JSON. Це забезпечує як наочну демонстрацію роботи (графіки), так і можливість подальшого аналізу чи інтеграції з іншими інструментами (табличні формати).

Функціональна модель програмного модуля відображає набір дій, які виконує система з точки зору користувача та зовнішнього середовища.

На відміну від структурного опису, де акцент робиться на компонентах і потоках даних, функціональна схема формалізує «що саме робить модуль» та «які результати отримуються».

Як показано на рис. 2.2, основним актором виступає користувач (оператор), який ініціює запуск програми та задає параметри обробки: шлях до набору даних, крок обробки кадрів, максимальну кількість кадрів, а також параметри, пов'язані з маркерами (словник і розмір маркера).

Другим джерелом взаємодії є файлове середовище, яке надає кадри та метадані і приймає результати у вигляді файлів і зображень графіків.

У межах моделі виділяються ключові прецеденти: завантаження вхідних даних; попередня обробка кадрів; виявлення маркерів і перевірка їх наявності; оцінка просторової пози; формування й постобробка траєкторії; візуалізація; експорт у файли; а також формування підсумкової статистики (наприклад, кількість кадрів, у яких маркери були успішно детектовані).

Логіка прецедентів передбачає, що побудова траєкторії є залежною від успішності етапу детекції (у маркерному режимі) або зіставлення ознак (у режимі візуальної одометрії), тобто система коректно обробляє ситуації, коли на окремих кадрах маркери відсутні чи не розпізнані.

У такому разі кадр пропускається або враховується лише в частині, яка не порушує цілісність послідовності даних. Завдяки цьому модуль залишається працездатним на реалістичних відеопотоках, де можливі розмиття, часткові перекриття об'єктів або змінні умови освітлення.

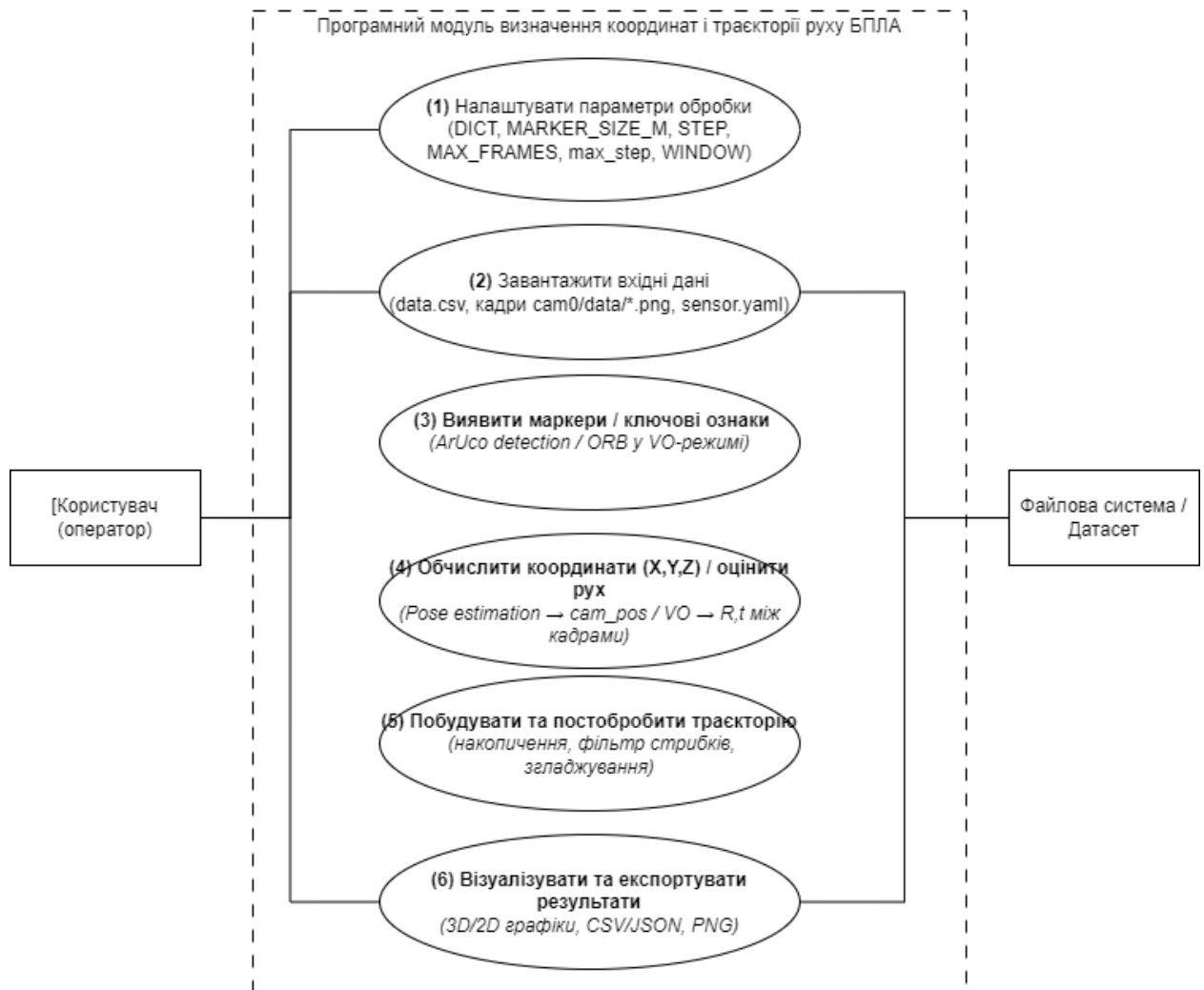


Рис. 2.2 - Функціональна модель (Use Case) програмного модуля визначення координат і траєкторії руху БПЛА.

Отже, наведені структурна та функціональна моделі демонструють цілісну організацію програмного модуля і пояснюють, яким чином реалізується перетворення вхідних візуальних даних у координати та траєкторію руху. Структурна схема (рис. 2.1) відображає послідовність блоків обробки та варіативність методів оцінювання руху, тоді як функціональна схема (рис. 2.2) визначає ролі, сценарії роботи та перелік ключових функцій, які забезпечують отримання, візуалізацію та збереження результатів. Такий опис створює основу для подальшого розгляду вимог до програмного продукту, а також для детального аналізу реалізації та експериментальних результатів у наступних розділах.

3 ОПИС ПРОГРАМНОГО МОДУЛЯ ВИЗНАЧЕННЯ КООРДИНАТ І ТРАЄКТОРІЇ РУХУ БПЛА

3.1 Вимоги до програмного модуля визначення координат і траєкторії руху БПЛА

Програмний модуль визначення координат і траєкторії руху БПЛА призначений для обробки даних бортових сенсорів візуального типу (відеопотоку або послідовності зображень) з метою оцінювання просторового положення рухомої платформи та побудови траєкторії її переміщення у тривимірному просторі.

У межах поставленої задачі камера розглядається як основне джерело інформації про зміни положення, а програмний модуль реалізує повний цикл обчислювальної обробки: завантаження вхідних даних, попередню підготовку кадрів, застосування методів комп'ютерного зору для оцінювання руху, формування координат (X, Y, Z) у часі, візуалізацію отриманої траєкторії та збереження результатів у файли.

Вимоги до модуля сформульовані таким чином, щоб вони були перевірюваними на контрольному наборі даних і відповідали як функціональним, так і якісним (нефункціональним) критеріям.

Функціональні вимоги визначають набір можливостей, які модуль зобов'язаний забезпечувати під час виконання.

По-перше, модуль має приймати на вхід візуальні дані у вигляді відео або набору зображень; у реалізованому сценарії базовим є режим роботи з послідовністю кадрів, що дозволяє зберегти відтворюваність експериментів і спростити повторні запуски з однаковими параметрами.

Для коректного формування траєкторії кожному кадру повинна відповідати часова мітка, тому модуль має підтримувати завантаження метаданих (відповідність «timestamp-filename») з табличного файлу. По-друге,

модуль повинен виконувати попередню обробку кадрів перед застосуванням алгоритмів комп'ютерного зору.

До попередньої обробки належать перетворення зображення у відтінки сірого, масштабування (за потреби) та підготовка зображення до стійкої детекції ознак. По-третє, модуль має реалізовувати процедуру виявлення опорних візуальних елементів, за якими оцінюється положення або зміщення між кадрами.

У базовому режимі такими елементами виступають штучні візуальні маркери (ArUco), які дають можливість оцінити позу камери відносно маркера і тим самим отримати просторові координати (X , Y , Z).

Для підвищення універсальності допускається наявність альтернативного режиму, в якому положення оцінюється за ключовими точками природної сцени (візуальна одометрія), однак у такому випадку слід враховувати, що масштаб траєкторії може бути відносним.

Вимога щодо оцінювання координат і побудови траєкторії є ключовою: модуль повинен обчислювати просторові координати (X , Y , Z) для тих моментів часу, коли алгоритм виявлення ознак або маркерів спрацював коректно, та формувати послідовність цих координат у часі.

Для маркерного режиму необхідною умовою є визначення масштабу через відомий розмір маркера, що задається параметром і використовується під час оцінки пози.

Якщо точний розмір маркера відомий, координати можуть інтерпретуватися в метричній системі; якщо параметр заданий наближено, траєкторія зберігає геометричну форму, але масштаб є оцінковим.

При побудові траєкторії модуль повинен забезпечувати накопичення точок у вигляді часових рядів, де кожному timestamp відповідає трійка координат.

Також модуль має коректно обробляти пропуски: якщо на окремих кадрах маркери не детектуються або ознаки не можуть бути зіставлені, такі

кадри не повинні руйнувати послідовність, а мають або пропускатися, або оброблятися у спосіб, що не порушує цілісності результату.

Окремою функціональною вимогою є постобробка траєкторії з метою підвищення її стійкості та наочності. У реальних даних можливі випадкові похибки детекції, які проявляються як різкі стрибки координат між сусідніми точками. Тому модуль повинен підтримувати принаймні один механізм стабілізації: відсікання аномальних переміщень за порогом максимально допустимого кроку та/або згладжування координат за ковзним середнім.

Такі операції не змінюють логіку побудови траєкторії, але зменшують вплив випадкових помилок і підвищують інформативність візуалізації. Після формування та постобробки траєкторії модуль повинен здійснювати візуалізацію результатів у вигляді 3D-графіка, а також (за можливості) 2D-проекцій на площини, що спрощує аналіз компоненти руху по окремих осях.

Візуалізація є обов'язковою частиною функціоналу, оскільки забезпечує наочну перевірку коректності та є одним із критеріїв оцінювання роботи.

Вимоги до експорту результатів формалізують, у якому вигляді модуль повинен зберігати дані для подальшого використання. Модуль має зберігати обчислені координати та траєкторію у файл табличного формату (CSV) та у файл структурованого формату (JSON).

Мінімально необхідними полями є часова мітка та координати X, Y, Z; додатково доцільно зберігати технічні атрибути, що характеризують надійність кадру (наприклад, кількість детектованих маркерів або ідентифікатор обраного маркера). Для повторюваності експериментів модуль повинен також зберігати графіки в окремі файли (наприклад, у PNG), щоб результати можна було включити у звітні матеріали без необхідності повторного запуску під час підготовки документації.

Наявність обох форматів (CSV і JSON) забезпечує сумісність як з табличними редакторами, так і з програмними засобами подальшого аналізу.

Оскільки модуль працює з даними файлів, у вимогах до вхідних даних доцільно зафіксувати мінімальну структуру набору для тестування. Вхідний

набір повинен містити каталог із послідовністю кадрів у поширеному растровому форматі (наприклад, PNG або JPG), файл відповідності кадрів часовим міткам (CSV), а також конфігураційний файл сенсора, який містить принаймні роздільну здатність зображень.

У разі відсутності повних параметрів камери (внутрішньої калібровки) допускається використання апроксимації матриці внутрішніх параметрів на основі роздільної здатності, проте в цьому випадку необхідно вказати, що точність оцінки координат може зменшуватися і залежить від якості апроксимації.

Для маркерного режиму окремою умовою є присутність маркерів у кадрі протягом суттєвої частини послідовності, достатній контраст маркера відносно фону та прийнятний рівень розмиття. Такі умови прямо впливають на стійкість детекції і, відповідно, на повноту траєкторії.

Нефункціональні вимоги описують умови, за яких модуль має бути придатним до використання. З погляду програмної платформи модуль реалізується мовою Python та має виконуватися в середовищі Windows або Linux; у поточній конфігурації цільовою платформою є Windows із запуском у середовищі розробки PyCharm.

Залежності повинні бути обмежені поширеними бібліотеками науково-технічного стеку: OpenCV (з модулем для роботи з ArUco), NumPy, Pandas, Matplotlib та PyYAML. Вимога до відтворюваності передбачає, що при однакових вхідних даних і параметрах модуль формує однакові вихідні файли, а всі налаштування (розмір маркера, крок обробки, пороги фільтрації) задаються явно.

Вимога до надійності означає, що модуль коректно обробляє часткові відмови: якщо окремі кадри не зчитуються або на них не детектуються маркери, програма не припиняє роботу аварійно, а продовжує обробку з пропуском проблемних кадрів та формуванням звіту про кількість успішно оброблених точок.

Критерії приймання (перевірки виконання вимог) формулюються як набір ознак, що підтверджують працездатність модуля на тестовому наборі. Модуль вважається реалізованим коректно, якщо при запуску на контрольному датасеті він формує файл CSV із часовими мітками та координатами X, Y, Z, формує еквівалентний файл JSON, зберігає графічні матеріали (3D-траєкторія та, за наявності, 2D-проекції) і завершує роботу без критичних помилок.

Додатковим позитивним показником є виведення у консоль статистики обробки, зокрема кількості кадрів у наборі, кількості кадрів із успішною детекцією маркерів та кількості збережених точок траєкторії.

Таким чином, перелік вимог охоплює як функціональні аспекти (обробка даних, оцінка координат, побудова траєкторії, візуалізація, експорт), так і нефункціональні (платформа, залежності, відтворюваність, стійкість), що створює формальну основу для подальшого опису інформаційної бази та реалізованих функцій програмного модуля.

3.2 Структура інформаційної бази програмного модуля визначення координат і траєкторії руху БПЛА

Інформаційна база програмного модуля в межах даної роботи реалізована у вигляді файлово організованого сховища даних, яке об'єднує вхідний набір зображень (або відеопотік, представлений послідовністю кадрів), метадані сенсорів і результати обчислень.

Такий підхід дозволяє чітко розділити дані, що надходять на обробку, та артефакти, які формуються модулем у процесі виконання алгоритмів, і тим самим забезпечує відтворюваність експерименту при повторних запусках з однаковими параметрами.

Крім того, файлова організація дає змогу швидко перевіряти коректність вхідних файлів, виконувати порівняння результатів між різними режимами

обробки та зручно включати вихідні матеріали (графіки, таблиці координат, JSON-структури) до пояснювальної записки. У межах магістерської роботи це важливо також з точки зору документування: усі проміжні та кінцеві результати можуть бути збережені у стандартизованих форматах і використані як доказ працездатності програмного модуля.

Загальна організація даних інформаційної бази подана на рис. 3.1, де відображено ієрархію каталогів та логічне групування вхідних і вихідних компонентів.

Вхідні дані розміщуються у гілці, що відповідає набору сенсорів камери, зокрема у каталозі *mav0/cam0*, який містить підкаталог із кадрами (*data*) та файл відповідності імен кадрів часовим міткам (*data.csv*).

Метадані сенсора камери (зокрема роздільна здатність зображень і матриця зовнішніх параметрів відносно базової системи координат) зберігаються у конфігураційному файлі *sensor.yaml* у тому ж каталозі, що забезпечує локальність і цілісність опису джерела даних.

Вихідні результати формуються у окремих каталогах результатів, зокрема *outputs_aruco*, де зберігаються файли координат у форматах CSV/JSON та зображення графіків траєкторії; за наявності альтернативного режиму обробки (наприклад, візуальної одометрії) результати можуть додатково групуватися в окремому каталозі *outputs_vo*.

Така декомпозиція є зручною з погляду експлуатації: вихідні файли не змішуються з вхідним датасетом, а структура каталогів однозначно відображає, які саме результати отримано конкретним методом або режимом, що спрощує аналіз і підготовку матеріалів до представлення.

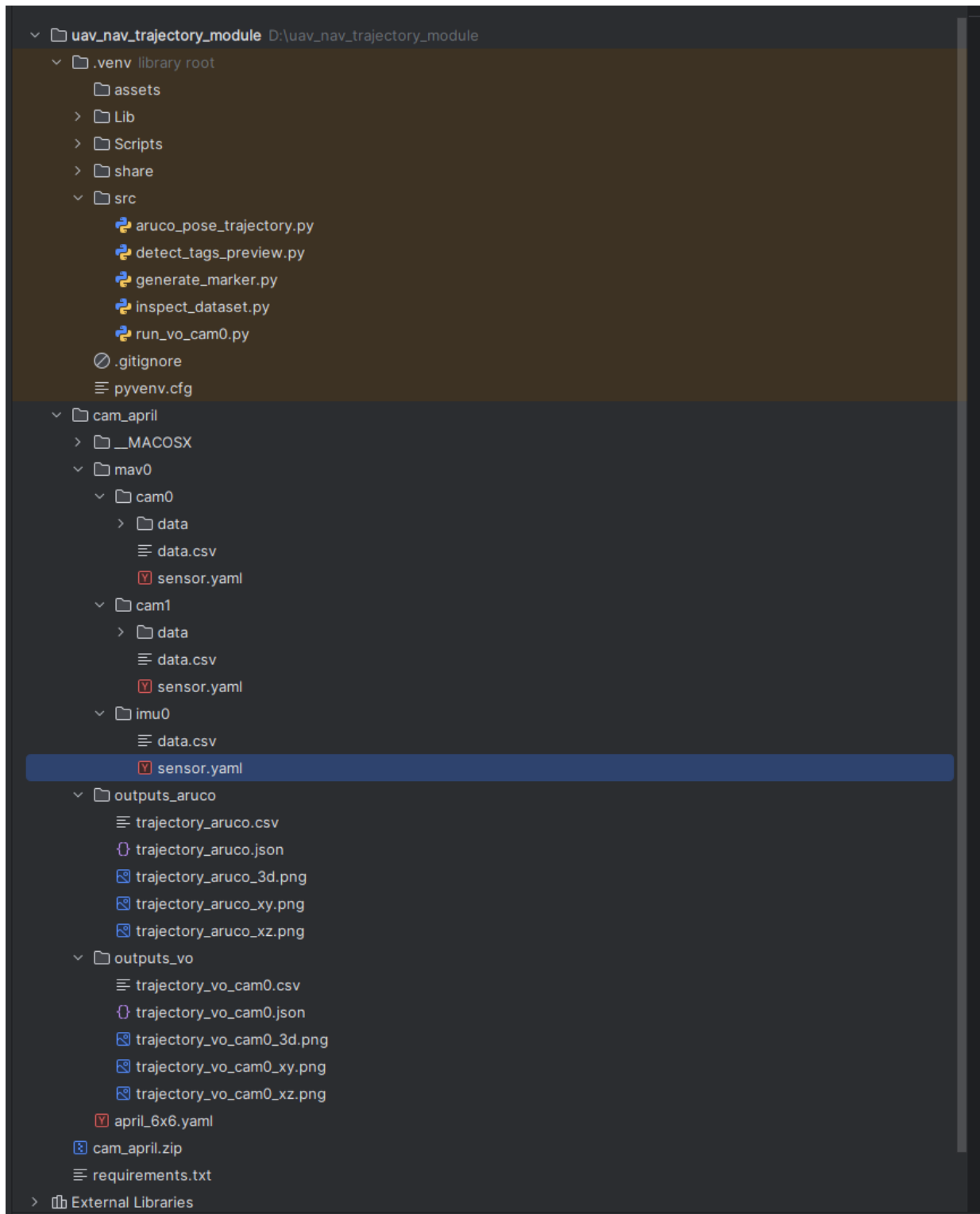


Рис. 3.1 - Структура інформаційної бази програмного модуля (файлова організація даних).

Після подання загальної структури інформаційної бази (див. рис. 3.1) доцільно деталізувати склад вхідних даних, які безпосередньо використовуються програмним модулем для відновлення координат і формування траєкторії руху.

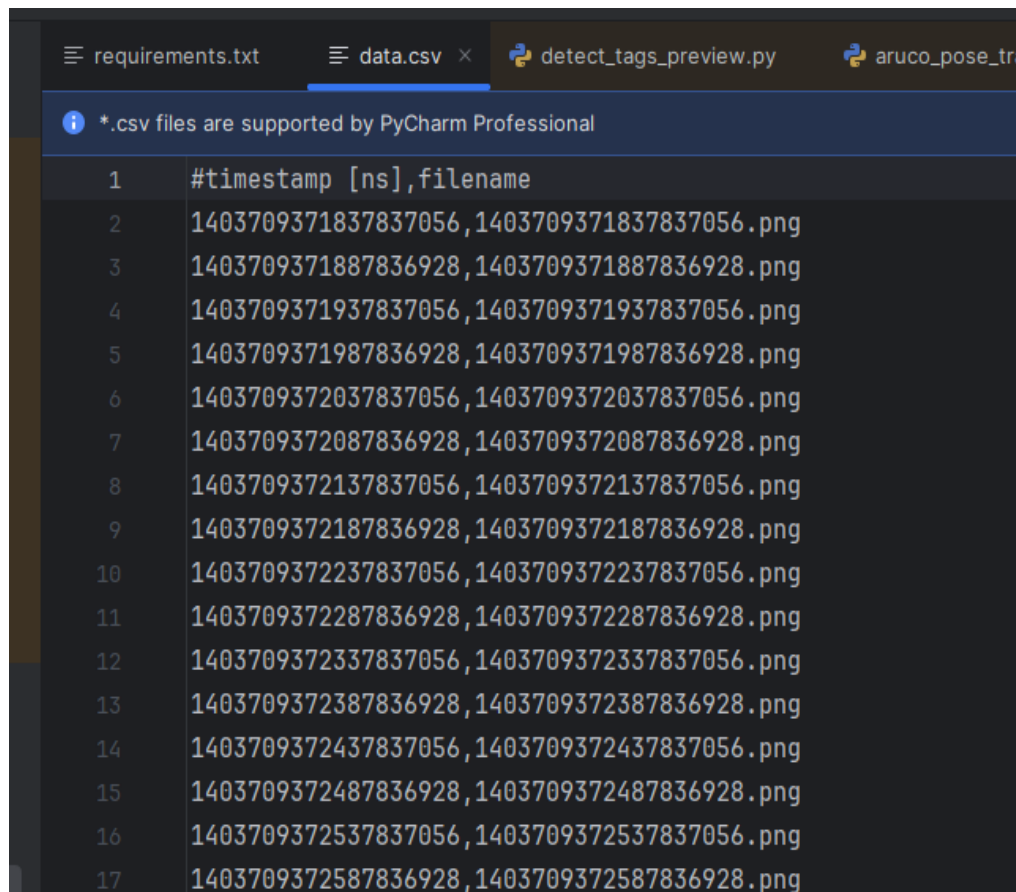
Вхідний набір включає послідовність кадрів камери у вигляді окремих зображень у каталозі *data/*.png*, службову таблицю відповідності кадрів часовим міткам *data.csv*, а також конфігураційний файл сенсора *sensor.yaml*, що містить мінімально необхідні метадані (зокрема параметри роздільної здатності та матрицю зовнішніх параметрів відносно базової системи координат). Узагальнена характеристика цих компонентів подана в табл. 3.1.

Табл. 3.1 - Опис вхідних даних інформаційної бази програмного модуля

Файл/ката лог	Формат	Призначення	Ключові атрибути/п оля	Примітки (умови коректності)
mav0/cam0 /data/*.png	PNG (растрові і кадри)	Вхідні зображення, що утворюють послідовність спостережень для оцінювання руху/пози	Вміст кадру (візуальна сцена, маркери), роздільна здатність кадру	Кадри мають бути впорядковані та доступні за іменами, що вказані в <i>data.csv</i> ; для маркерного режиму бажано, щоб маркери були присутні в кадрі значну частину часу, з достатнім контрастом і без критичного розмиття
mav0/cam0 /data.csv	CSV	Зв'язок часових міток із файлами кадрів; забезпечує прив'язку координат до часу	timestamp_n s, filename	Часова мітка повинна бути у наносекундах (ціле число); filename має відповідати реальній назві файлу в <i>data/</i> ; пропуски/невідповідності призводять до втрати кадрів під час обробки
mav0/cam0 /sensor.yaml	YAML	Метадані сенсора камери для коректної інтерпретації зображень і узгодження систем координат	resolution: [w, h], T_BS (4×4)	За відсутності повної внутрішньої калібровки допускається використання наближених внутрішніх параметрів камери на основі resolution; матриця T_BS зберігається для узгодження систем координат та подальшого розширення (наприклад, інтеграції IMU)

Безпосереднім “каркасом” часової організації експерименту виступає файл *data.csv*, який задає відповідність між часовими мітками та назвами кадрів. Фрагмент такого файлу наведено нижче як приклад структури записів.

У першому стовпці (timestamp_ns) подано часову мітку кадру у наносекундах, у другому (filename) — ім'я файлу зображення, що зберігається у каталозі data. Саме зв'язок «timestamp → кадр» використовується для формування траєкторії як часового ряду: кожній обробленій парі “часова мітка — кадр” відповідає обчислена точка траєкторії, яка далі зберігається у вихідних файлах і може бути проаналізована або візуалізована у часовій шкалі. (рис. 3.2)



	#timestamp [ns],filename
1	
2	1403709371837837056,1403709371837837056.png
3	1403709371887836928,1403709371887836928.png
4	1403709371937837056,1403709371937837056.png
5	1403709371987836928,1403709371987836928.png
6	1403709372037837056,1403709372037837056.png
7	1403709372087836928,1403709372087836928.png
8	1403709372137837056,1403709372137837056.png
9	1403709372187836928,1403709372187836928.png
10	1403709372237837056,1403709372237837056.png
11	1403709372287836928,1403709372287836928.png
12	1403709372337837056,1403709372337837056.png
13	1403709372387836928,1403709372387836928.png
14	1403709372437837056,1403709372437837056.png
15	1403709372487836928,1403709372487836928.png
16	1403709372537837056,1403709372537837056.png
17	1403709372587836928,1403709372587836928.png

Рисунок 3.2 - Фрагмент файлу data.csv (відповідність часових міток і кадрів)

Як видно з наведеного прикладу, імена файлів кадрів узгоджені з часовими мітками, що спрощує перевірку цілісності датасету та дозволяє однозначно ідентифікувати кожний кадр. На практиці це дає змогу формувати траєкторію не лише як геометричну криву в просторі, а і як послідовність координат, прив'язаних до конкретних моментів часу, що є необхідною умовою для подальших порівнянь, фільтрації та аналізу результатів.

Окрім власне візуальних кадрів і таблиці відповідності часових міток, інформаційна база містить метадані сенсора, які необхідні для коректної інтерпретації даних та узгодження систем координат під час обчислень. У межах використаного набору даних такі метадані подані у файлі *sensor.yaml*, фрагмент якого наведено на рис. 3.3.

```

1  # General sensor definitions.
2  sensor_type: camera
3  comment: VI-Sensor cam0 (MT9M034)
4
5  # Sensor extrinsics wrt. the body-frame.
6  T_BS:
7    cols: 4
8    rows: 4
9    data: [0.0148655429818, -0.999880929698, 0.00414029679422, -0.0216401454975,
10           0.999557249008, 0.0149672133247, 0.025715529948, -0.064676986768,
11           -0.0257744366974, 0.00375618835797, 0.999660727178, 0.00981073058949,
12           0.0, 0.0, 0.0, 1.0]
13
14  # Camera specific definitions.
15  rate_hz: 20
16  resolution: [752, 480]
17

```

Рис. 3.3 - Фрагмент файлу *sensor.yaml* (метадані сенсора: *resolution*, *T_BS*)

Найбільш важливими для реалізованого програмного модуля є параметр *resolution*, що задає роздільну здатність кадрів, та матриця *T_BS*, яка описує положення (позу) сенсора відносно базової (корпусної) системи координат.

resolution використовується модулем як джерело інформації про геометрію зображення.

У реалізованій конфігурації, коли повні внутрішні параметри камери (матриця внутрішньої калібровки) можуть бути відсутні у файлі конфігурації, значення *resolution* застосовується для формування наближених внутрішніх параметрів (матриці K) на основі ширини та висоти кадру.

Така апроксимація дає можливість виконувати оцінювання пози для задачі побудови траєкторії й забезпечує працездатність модуля без проведення окремої процедури калібрування, що є важливим для спрощення відтворюваного експерименту на готовому датасеті.

T_{BS} (перетворення від *body-frame* до *sensor-frame*) відображає екструзійні параметри, тобто просторове розташування камери відносно базової системи координат носія.

У межах поточної реалізації цей параметр фіксується як складова інформаційної бази й може бути використаний для узгодження координат під час розширення модуля, зокрема при інтеграції даних IMU та реалізації візуально-інерціальної оцінки руху.

Наявність T_{BS} у наборі даних є важливою ознакою того, що інформаційна база передбачає можливість подальшої мультисенсорної обробки (камера + інерціальні датчики), навіть якщо базовий сценарій магістерської роботи використовує візуальні дані як основні.

Після обробки вхідних даних програмний модуль формує вихідні результати у стандартизованому вигляді, що дозволяє виконувати подальший аналіз, перевірку коректності та повторне використання траєкторії в інших програмних засобах.

Структура вихідних даних узагальнена в табл. 3.2. Результати обчислень зберігаються у двох взаємодоповнювальних форматах — табличному (CSV) та структурованому (JSON), а також супроводжуються графічними матеріалами (PNG), які відображають траєкторію у 3D та у вигляді 2D-проекцій.

Такий набір вихідних файлів забезпечує як машинну обробку координат (наприклад, для статистичного аналізу чи порівняння), так і швидко візуальну верифікацію отриманого результату.

Табл. 3.2 - Структура вихідних даних програмного модуля (результати обчислення траєкторії)

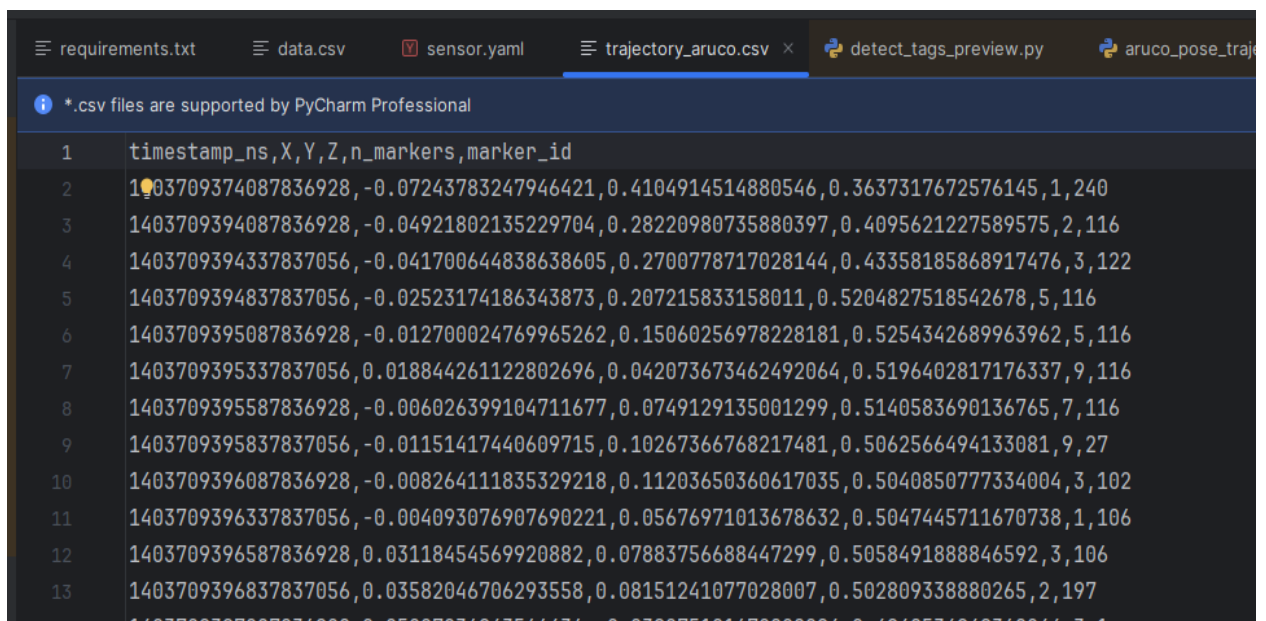
Файл/ каталог	Формат	Вміст (поля/ атрибути)	Призначення	Примітки
outputs_aruco/ trajectory_aruco.csv	CSV	timestamp_ns, X, Y, Z, n_markers, marker_id	Основний табличний результат: траєкторія як часовий ряд координат	Зручний для аналізу в табличних редакторах і для подальшої обробки; n_markers і marker_id використовуються для оцінювання надійності детекції
outputs_aruco/ trajectory_aruco.json	JSON	список об'єктів із полями timestamp_ns, X, Y, Z, (додатково n_markers, marker_id)	Структуроване подання траєкторії для програмної інтеграції	Зручний для підключення до інших модулів/сервісів та збереження разом із метаданими
outputs_aruco/ trajectory_aruco_3d.png	PNG	3D-графік траєкторії (X- Y-Z)	Візуальна перевірка форми траєкторії в просторі	Використовується як ілюстративний матеріал у звіті
outputs_aruco/ trajectory_aruco_xy.png	PNG	проєкція траєкторії на площину X-Y	Аналіз горизонтальної компоненти руху	Полегшує інтерпретацію траєкторії при складній 3D- геометрії
outputs_aruco/ trajectory_aruco_xz.png	PNG	проєкція траєкторії на площину X-Z	Аналіз вертикальної	Дозволяє оцінювати зміни

			компоненти руху	висоти/глибини відносно маркера
outputs_vo/*	CSV/JSON/PNG	аналогічні файли для VO-режиму (наприклад, trajectory_vo_cam0.*)	Альтернативний набір результатів (візуальна одометрія)	Масштаб може бути відносним; використовується для порівняння режимів і демонстрації універсальності підходу

Для прикладного підтвердження структури виходу на рис. 3.4 наведено фрагмент файлу trajectory_aruco.csv, який є основним табличним результатом маркерного режиму.

Поля X, Y, Z інтерпретуються як координати камери (або платформи з камерою) у системі координат обраного маркера; часові мітки timestamp_ns дозволяють розглядати траєкторію як часовий ряд.

Технічні атрибути n_markers та marker_id додані для контролю якості: кількість детектованих маркерів у кадрі та ідентифікатор використаного маркера дають змогу оцінювати стабільність детекції й пояснювати можливі локальні відхилення траєкторії при погіршенні умов спостереження.



	timestamp_ns,X,Y,Z,n_markers,marker_id
1	1403709374087836928,-0.07243783247946421,0.4104914514880546,0.3637317672576145,1,240
2	1403709394087836928,-0.04921802135229704,0.28220980735880397,0.4095621227589575,2,116
3	1403709394337837056,-0.041700644838638605,0.2700778717028144,0.43358185868917476,3,122
4	1403709394837837056,-0.02523174186343873,0.207215833158011,0.5204827518542678,5,116
5	1403709395087836928,-0.012700024769965262,0.15060256978228181,0.5254342689963962,5,116
6	1403709395337837056,0.018844261122802696,0.042073673462492064,0.5196402817176337,9,116
7	1403709395587836928,-0.006026399104711677,0.0749129135001299,0.5140583690136765,7,116
8	1403709395837837056,-0.01151417440609715,0.10267366768217481,0.5062566494133081,9,27
9	1403709396087836928,-0.008264111835329218,0.11203650360617035,0.5040850777334004,3,102
10	1403709396337837056,-0.004093076907690221,0.05676971013678632,0.5047445711670738,1,106
11	1403709396587836928,0.03118454569920882,0.07883756688447299,0.5058491888846592,3,106
12	1403709396837837056,0.03582046706293558,0.08151241077028007,0.502809338880265,2,197
13	1403709397087836928,-0.05027036013544474,0.030275181470828004,0.4048534040348244,3,1

Рис. 3.4 - Фрагмент файлу trajectory_aruco.csv (приклад вихідних координат траєкторії).

Узагальнюючи, сформована структура інформаційної бази безпосередньо відповідає реалізованим функціям програмного модуля та визначає послідовність його роботи.

На етапі запуску модуль виконує читання вхідних компонентів: із каталогу *data* завантажуються кадри, з файлу *data.csv* зчитуються часові мітки та імена зображень для формування коректного порядку обробки у часі, а з *sensor.yaml* отримуються метадані сенсора, необхідні для інтерпретації геометрії зображення та узгодження систем координат.

Далі ці дані використовуються у процедурі попередньої обробки і основних алгоритмах оцінювання руху/пози, в результаті чого формується послідовність просторових координат, тобто траєкторія у вигляді часового ряду.

Завершальним кроком є експорт результатів у стандартизовані вихідні файли, що зберігаються в окремих каталогах (*outputs_aruco*, за потреби *outputs_vo*) у форматах CSV/JSON та у вигляді графічних матеріалів.

Такий поділ на вхідний датасет, метадані та ізольовані каталоги результатів забезпечує відтворюваність експерименту й перевірюваність отриманих висновків у межах магістерської роботи.

Оскільки вхідні файли зберігаються незмінними, а параметри обробки задаються явно, повторний запуск модуля на тому самому наборі даних дозволяє отримати зіставні результати та виконати контроль правильності роботи алгоритмів. Наявність вихідних файлів у табличному й структурованому форматах спрощує подальший аналіз координат, а збережені графіки забезпечують швидку візуальну верифікацію траєкторії без необхідності повторного обчислення.

Таким чином, інформаційна база є не лише сховищем даних, а й основою для документування, тестування та демонстрації функціональних можливостей програмного модуля.

3.3 Опис основних функцій програмного модуля визначення координат і траєкторії руху БПЛА

У підрозділі 3.3 подано опис основних функцій програмного модуля визначення координат і траєкторії руху БПЛА та розкрито логіку його роботи на рівні етапів обробки даних і формування результатів.

Основна увага приділяється послідовності виконання операцій, взаємозв'язку між вхідними даними, алгоритмічними процедурами та вихідними артефактами, які утворюються в процесі обчислень.

Повні тексти програмних файлів, що реалізують наведені функції, подано у Додатку А; у цьому підрозділі вони розглядаються не як суцільний код, а як узгоджений набір процедур, що формує цілісний технологічний процес.

Такий підхід забезпечує прозорість реалізації, дозволяє перевірити відтворюваність експерименту та обґрунтувати отримані результати.

Загальна логіка функціонування програмного модуля відповідає типовій схемі обробки візуальних даних для задачі оцінювання положення та побудови траєкторії:

- ініціалізація параметрів і шляхів доступу;
- завантаження вхідних даних;
- попередня обробка кадрів;
- детекція опорних візуальних елементів (маркерів або ознак);
- оцінка пози/зміщення між кадрами;
- формування послідовності координат (траєкторії);
- постобробка (фільтрація аномальних стрибків і, за потреби, згладжування);
- візуалізація результатів;
- експорт у файли.

Вхідні компоненти та структура вихідних артефактів були описані в підрозділі 3.2, тому в межах даного підрозділу показано, яким чином ці дані

послідовно використовуються функціями модуля: зчитування кадрів та часових міток ініціює процес обчислень, параметри сенсора забезпечують коректну інтерпретацію зображень, а результати зберігаються у стандартизованих форматах для подальшого аналізу й перевірки. (табл 3.3)

Табл. 3.3 - Відповідність основних функцій програмного модуля реалізованим скриптам

Скрипт (файл)	Функціональне призначення	Вхідні дані	Вихідні артефакти (результат)
inspect_dataset.py	Перевірка структури датасету, коректності шляхів і наявності ключових файлів	Каталог датасету (cam_april), наявні підкаталоги mav0/cam0, data, файли data.csv, sensor.yaml	Діагностичний звіт у консолі (наявність/відсутність файлів, базова статистика), підтвердження готовності даних до обробки
detect_tags_preview.py	Тестова детекція маркерів на кадрі та підбір/визначення словника ArUco	Вибраний кадр із data/*.png, параметри детекції, перелік словників	Вивід у консоль (знайдений словник, кількість детекцій), візуальний перегляд результату детекції на кадрі
aruco_pose_trajectory.py	Основний режим (ArUco): оцінювання пози, обчислення координат (X,Y,Z), формування траєкторії, постобробка, експорт	data/*.png, data.csv, sensor.yaml, параметри (DICT_NAME, MARKER_SIZE_M, STEP, MAX_FRAME_S, max_step, WINDOW)	trajectory_aruco.csv, trajectory_aruco.json, графіки траєкторії (*.png) у каталозі outputs_aruco
run_vo_cam0.py	Альтернативний режим (VO): оцінка руху за ознаками (ORB), відновлення відносної траєкторії (масштаб відносний)	data/*.png, data.csv, (за потреби) параметри камери	Траєкторія VO у файлі(ax) результатів (CSV/JSON), графіки (*.png) у каталозі outputs_vo

generate_ marker.py	Допоміжний інструмент: генерація маркера/дошки для тестування детекції (за потреби)	Параметри генерації (тип словника, ID, розмір)	Зображення маркера/дошки у форматі PNG для подальших тестів
------------------------	--	---	---

Першим етапом практичного виконання є ініціалізація середовища обробки та параметрів запуску, які визначають режим роботи модуля й забезпечують відтворюваність результатів.

До ключових налаштувань належать вибір словника маркерів (*DICT_NAME*), метричний параметр розміру маркера (*MARKER_SIZE_M*), параметри дискретизації обробки кадрів (*STEP*) та обмеження кількості кадрів у експерименті (*MAX_FRAMES*).

Для підвищення стійкості траєкторії використовуються параметри постобробки, зокрема поріг відсікання різких стрибків координат (*max_step*) та параметр згладжування (*WINDOW*) за потреби.

Окремо задаються шляхи до датасету (*DATASET*) та каталогу збереження результатів (*OUT_DIR*), що дозволяє однозначно відокремити вхідні дані від вихідних артефактів і повторювати обчислення за однаковою конфігурацією.

Після ініціалізації модуль переходить до завантаження даних: зчитується файл *data.csv*, на основі якого формується впорядкований список кадрів із прив'язкою до часових міток, і для кожного запису будується повний шлях до зображення в каталозі *data*.

На цьому кроці також застосовується параметр *STEP*, який визначає проріджування послідовності кадрів і дає змогу керувати обсягом обробки без зміни структури датасету.

Далі кожен кадр завантажується для попередньої обробки та виконання алгоритмів детекції й оцінювання пози; часові мітки з *data.csv* забезпечують узгодження отриманих координат із часом, що критично для формування траєкторії як часового ряду (див. рис. 3.2).

На етапі попередньої обробки кожен кадр, визначений у переліку *data.csv*, завантажується з каталогу *data* та проходить мінімальну підготовку перед виконанням детекції.

Оскільки детектори маркерів та алгоритми виділення ознак є чутливими до шумів і різних форматів представлення зображення, кадр переводиться у відтінки сірого (*grayscale*), що зменшує обчислювальну складність і усуває вплив колірних компонент на результат детекції.

Додатково здійснюється контроль коректності завантаження: у випадку, якщо файл зображення недоступний або зчитування завершилося невдачею (*img is None*), кадр пропускається без аварійного завершення програми, що підвищує стійкість модуля до неповних або частково пошкоджених наборів даних. За потреби застосовується масштабування (*resize*) зображення, яке в практичному експерименті використовується як прийом для підвищення стабільності детекції дрібних маркерів на кадрі: збільшення зображення доцільне тоді, коли маркери займають невелику частку площі кадру та їх контури важче виділяються на початковій роздільній здатності.

Усі зазначені операції виконують роль стандартизації вхідних кадрів і створюють умови для стабільної роботи детектора без необхідності ускладнювати алгоритм додатковими фільтрами.

Після попередньої обробки кадру виконується детекція ArUco-маркерів, яка є ключовою операцією маркерного режиму визначення координат. Для цього застосовується об'єкт *ArucoDetector*, налаштований на конкретний словник маркерів, вибраний на етапі підготовки експерименту (*DICT_NAME*).

Функціонально *ArucoDetector* виконує пошук квадратних маркерів у зображенні, локалізує їхні контури та повертає геометрію знайдених маркерів. На виході детектора формуються два основних набори даних: *corners* та *ids*. Масив *corners* містить для кожного знайденого маркера координати його чотирьох кутових точок у площині зображення, що є необхідним для подальшого оцінювання пози маркера відносно камери. Масив *ids* містить

ідентифікатори (номери) маркерів, за якими можна однозначно визначити, який саме маркер виявлено на кадрі.

Вибір словника маркерів принципово важливий: якщо словник задано некоректно, маркери не будуть розпізнані, а кількість детекцій залишатиметься нульовою, тому в межах реалізації передбачено попередню перевірку та підбір словника за тестовими кадрами. Приклад успішної детекції маркерів із відображенням контурів та ідентифікаторів на кадрі наведено на рис. 3.5, що підтверджує працездатність детектора на вхідних даних.

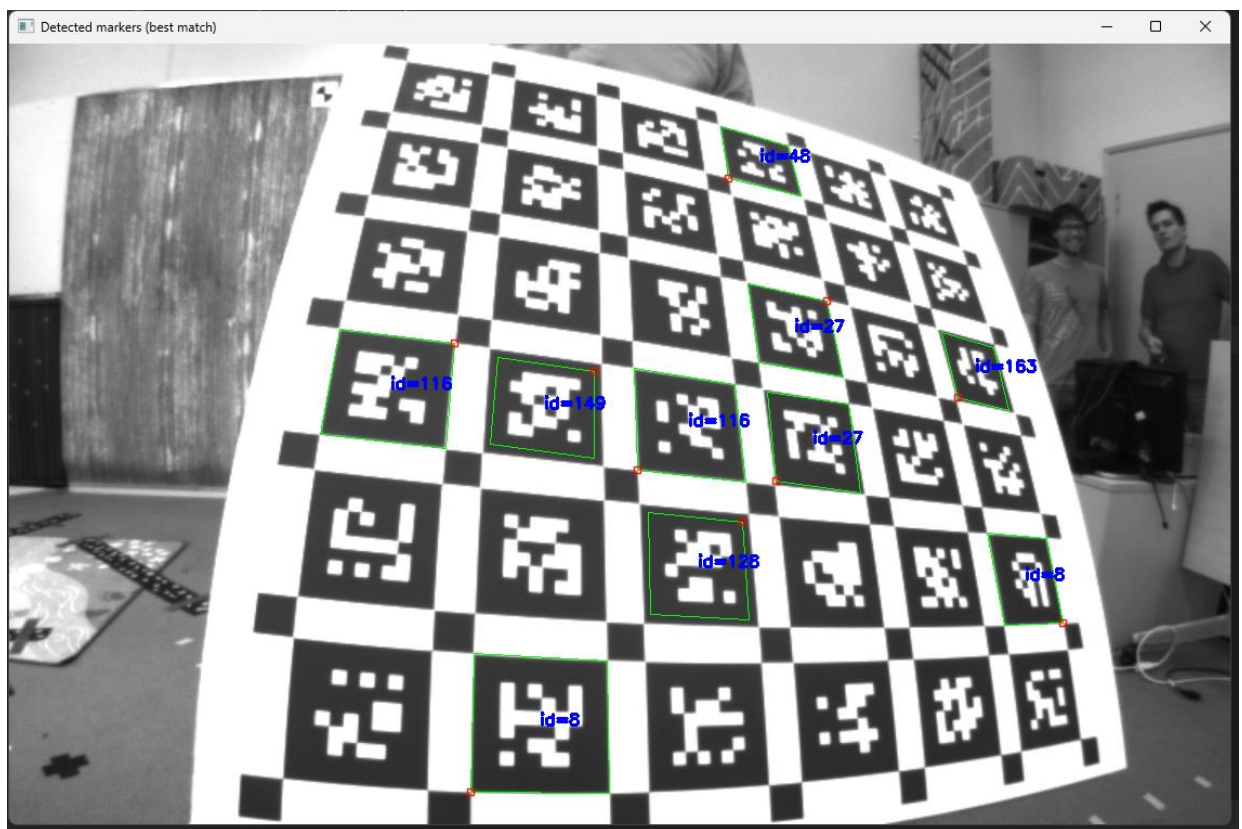


Рис. 3.5 - Приклад детекції ArUco-маркерів на вхідному кадрі (контури та ідентифікатори).

Після виявлення маркерів модуль переходить до етапу оцінювання пози (pose estimation), який забезпечує отримання просторових координат. Для кожного знайденого маркера на основі його кутових точок *corners*, відомого розміру маркера *MARKER_SIZE_M* та параметрів камери обчислюються

вектори обертання і переносу (*rvec*, *tvec*) за допомогою процедури *estimatePoseSingleMarkers*.

Вектор *tvec* відповідає просторовому переносу маркера відносно камери (у системі координат камери), а *rvec* задає його орієнтацію у вигляді вектора обертання. Для переходу до матричного представлення орієнтації використовується перетворення *Rodrigues*, яке будує матрицю повороту *R* на основі *rvec*. Оскільки у практичній задачі необхідно отримати положення камери (або платформи з камерою) відносно маркера, виконується інверсія оціненої пози: координати камери у системі маркера обчислюються як $cam_pos = -R^T \cdot tvec$.

Саме компоненти вектора *cam_pos* інтерпретуються як просторові координати *X*, *Y*, *Z* у відліку від маркера. Масштаб отриманих координат визначається параметром *MARKER_SIZE_M*: якщо розмір сторони маркера задано у метрах, то й координати будуть отримані у метрах. Таким чином, маркерний режим дозволяє перейти від суто візуальної інформації кадру до метрично інтерпретованих координат (за умови відомого розміру маркера).

Для підвищення стабільності оцінювання пози в умовах, коли у кадрі може бути декілька маркерів, реалізовано вибір “найнадійнішого” маркера за геометричним критерієм. Практика показує, що маркер, який займає більшу площу в кадрі, зазвичай має більш чітко локалізовані кути, менше спотворений перспективою та дає стабільніші значення *rvec/tvec*.

Тому модуль додатково оцінює площу контуру кожного знайденого маркера (за *corners*) і обирає для розрахунку координат той, що має максимальну площу. Такий прийом зменшує “стрибки” координат, які можуть з’являтися при короткочасній нестабільності детекції окремих маркерів, та підвищує цілісність траєкторії у часовій послідовності.

На основі обчислених координат для кожного обробленого кадру формується траєкторія як упорядкована у часі послідовність точок. Для кожної точки зберігається часова мітка з *data.csv* та значення *X*, *Y*, *Z*, що дозволяє розглядати результат як часовий ряд просторового положення.

У процесі накопичення точок модуль додатково може фіксувати технічні атрибути, наприклад кількість детектованих маркерів у кадрі та ідентифікатор обраного маркера, що спрощує подальшу інтерпретацію якості детекції. Оскільки в реальних даних можливі поодинокі помилки детекції або оцінювання пози, траєкторія після формування проходить постобробку.

Основним елементом постобробки виступає фільтр аномальних стрибків за порогом *max_step*: якщо відстань між двома послідовними точками перевищує заданий поріг, така точка розглядається як потенційно некоректна та відкидається. Додатково, за потреби, може застосовуватися згладжування координат із використанням вікна *WINDOW*, що зменшує дрібні коливання та робить траєкторію більш наочною для візуального аналізу.

Після завершення цих процедур результати формуються у вигляді таблиці координат і експортуються у вихідні файли; приклад структури такого виходу подано в підрозділі 3.2 (див. рис. 3.4), що демонструє, які саме дані накопичуються та зберігаються як підсумок роботи програмного модуля.

Після формування та постобробки траєкторії модуль переходить до етапу візуалізації, який виконує дві практичні функції: по-перше, забезпечує швидку візуальну перевірку коректності результату, а по-друге, формує ілюстративні матеріали для подальшого використання у звітній документації.

Основним графічним представленням траєкторії є тривимірний графік, у якому послідовність точок з координатами *X*, *Y*, *Z* відображається як просторова крива.

Така форма подання дає змогу оцінити загальний характер руху, наявність різких розривів або аномальних відхилень, а також просторову узгодженість траєкторії після застосування фільтрації.

Приклад 3D-візуалізації траєкторії, відновленої в маркерному режимі, наведено на рис. 3.6.

Разом із 3D-графіком доцільно використовувати двовимірні проєкції траєкторії на окремі площини, оскільки вони спрощують аналіз компонент руху за окремими осями.

Зокрема, проєкція на площину X - Y дозволяє оцінити “плановий” характер переміщення (горизонтальну складову), тоді як проєкція X - Z або Y - Z є корисною для аналізу зміни висоти/глибини відносно системи координат маркера.

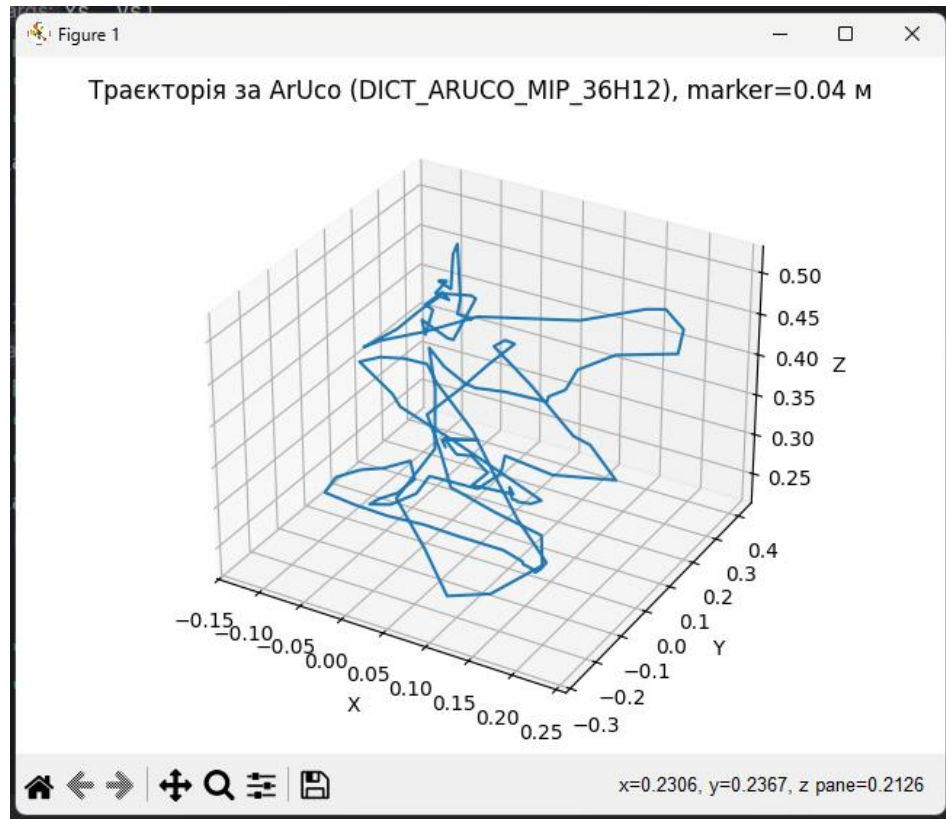


Рис. 3.6 - 3D-візуалізація траєкторії, відновленої за ArUco-маркерами.

У межах реалізації формуються відповідні 2D-графіки, які доповнюють 3D-подання та роблять результат більш інформативним для інтерпретації. Приклад 2D-проєкцій траєкторії подано на рис. 3.7.

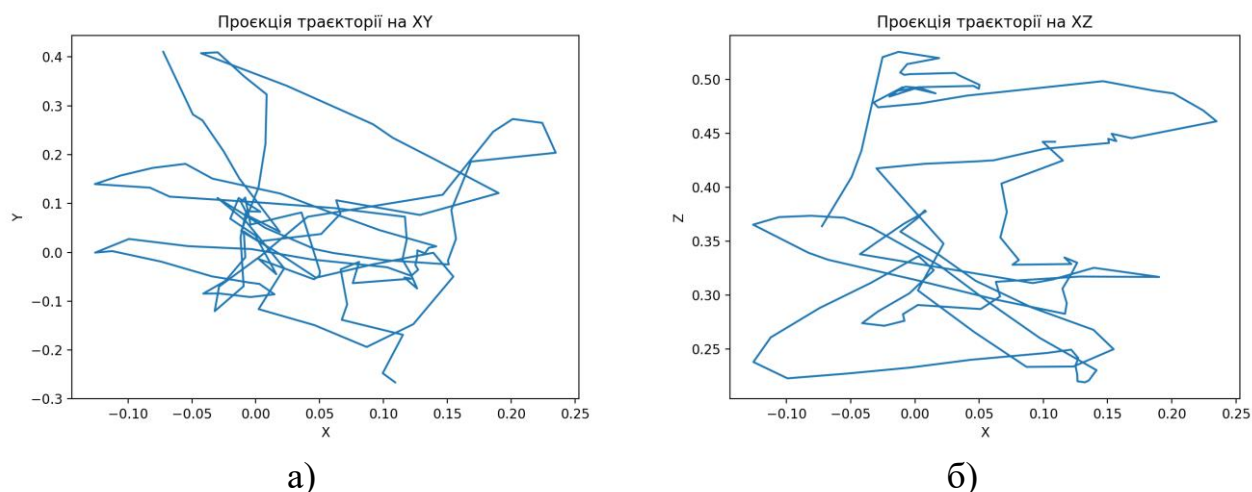


Рис. 3.7 - 2D-проекції траєкторії: (а) X-Y, (б) X-Z.

Після візуалізації модуль виконує експорт результатів, забезпечуючи їх збереження у форматах, придатних як для повторної програмної обробки, так і для документування.

Основним табличним результатом є файл CSV з часовими мітками та координатами, який зручний для аналізу у табличних процесорах і для розрахунків у середовищі Python.

Паралельно формується файл JSON, що подає траєкторію у структурованому вигляді та може бути використаний для інтеграції з іншими програмними компонентами або для передавання даних. Додатково зберігаються графічні матеріали у форматі PNG: 3D-траєкторія та її 2D-проекції, що дозволяє включати результати до пояснювальної записки без повторного запуску модулів візуалізації.

Усі артефакти маркерного режиму зберігаються в окремому каталозі результатів *outputs_aruco*, що забезпечує відокремлення вихідних файлів від вхідного датасету та підвищує зручність відтворення експерименту.

Узагальнена структура вихідних файлів і їх призначення наведені у табл. 3.2 (див. табл. 3.2), що дозволяє однозначно зіставити сформовані результати з вимогами до програмного модуля.

Окрім маркерного режиму, у програмному модулі реалізовано альтернативний підхід до відновлення траєкторії на основі візуальної одометрії (Visual Odometry, VO).

Метою VO-режиму є оцінювання переміщення камери між сусідніми кадрами без використання штучних маркерів, тобто лише за рахунок аналізу природних візуальних ознак сцени.

Загальна схема обчислень у цьому режимі включає послідовність операцій: виділення ключових точок і дескрипторів методом ORB → пошук відповідностей між двома кадрами → оцінювання матриці суттєвості (Essential Matrix) за парою наборів відповідних точок → відновлення відносної орієнтації та переносу функцією *recoverPose* → накопичення послідовних перетворень у вигляді траєкторії.

Таким чином, на кожному кроці модуль оцінює “крок руху” між кадрами, після чого послідовно інтегрує ці кроки у спільній системі координат, формуючи криву переміщення.

Принциповою особливістю VO-режиму є те, що отримана траєкторія має, як правило, відносний масштаб.

Це означає, що алгоритм здатний коректно відтворювати геометричну форму руху та зміну напрямку, однак абсолютні значення координат у метрах не гарантуються без додаткового джерела масштабу.

У маркерному режимі масштаб фіксується відомим розміром маркера (*MARKER_SIZE_M*), тоді як у VO-режимі для отримання метричного масштабу потрібні додаткові дані (наприклад, відома базова відстань, висота, або інтеграція з IMU).

Незважаючи на це, VO-режим є корисним для демонстрації універсальності модуля: він показує, що траєкторію можна відновлювати

навіть у сценаріях, де маркери відсутні або їх неможливо використати, а також дає можливість порівняти характер траєкторій, отриманих різними підходами.

Приклад траєкторії, сформованої у VO-режимі (у відносному масштабі), наведено на рис. 3.8.

За аналогією з маркерним режимом, результати VO-режиму також супроводжуються візуалізацією та збереженням файлів координат.

Візуалізація у 3D дозволяє швидко оцінити узгодженість траєкторії та наявність накопичених похибок, а 2D-проекції (за потреби) спрощують аналіз руху по окремих осях і зіставлення з результатами ArUco-режиму. З

береження результатів у стандартизованих форматах забезпечує можливість повторного аналізу й документування експериментів у межах магістерської роботи.

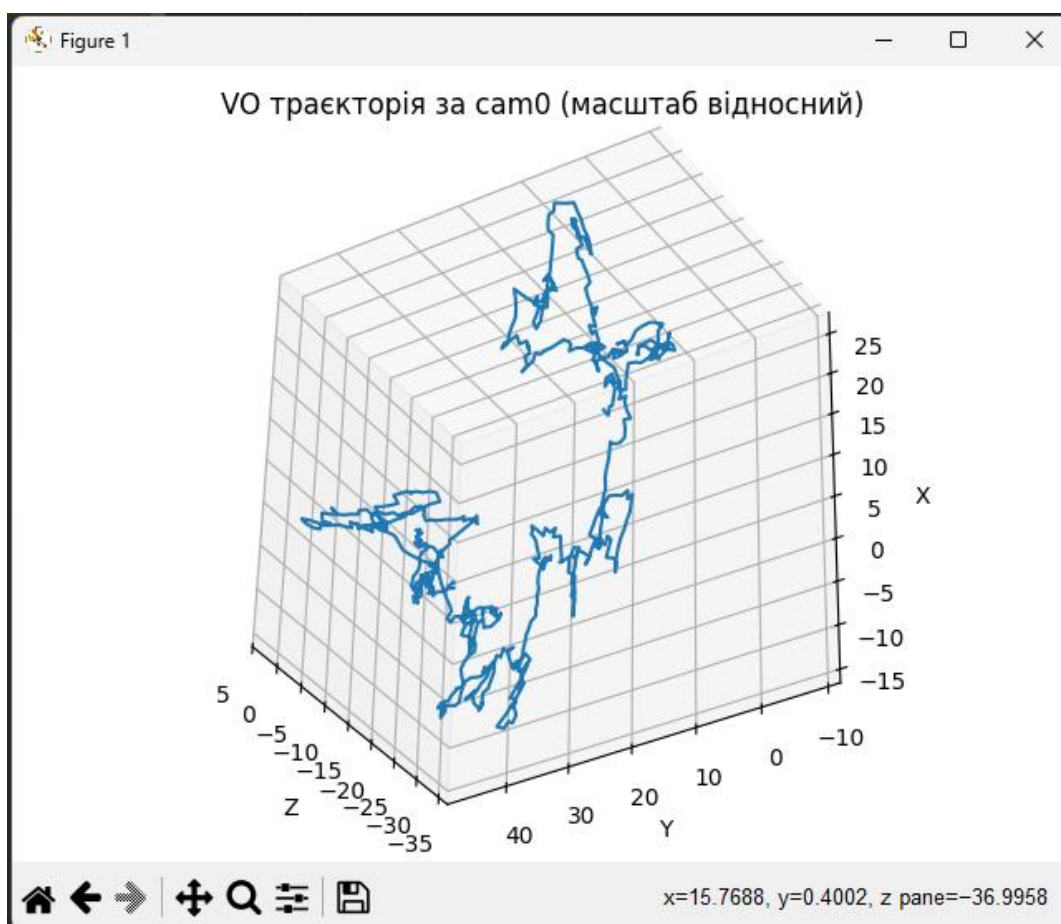


Рис. 3.8 - Приклад траєкторії у VO-режимі (відносний масштаб).

Отже, програмний модуль реалізує два взаємодоповнювальні підходи до визначення траєкторії руху: маркерний режим (ArUco), який дозволяє отримувати координати з метричною інтерпретацією за умови відомого розміру маркера, та альтернативний режим візуальної одометрії (VO), що працює за ознаками природної сцени й формує траєкторію у відносному масштабі.

В обох випадках обробка завершується формуванням траєкторії як послідовності координат у часі, її постобробкою, візуалізацією та експортом у файли, що підтверджує відповідність програмного модуля поставленим вимогам.

Наведений опис функцій і результати їх виконання створюють основу для подальшого узагальнення отриманих висновків та переходу до підсумкових положень роботи.

ВИСНОВКИ

У ході виконання магістерської роботи досягнуто поставленої мети — розроблено підхід і програмний модуль визначення просторових координат та траєкторії руху БПЛА на основі даних відеопотоку та сенсорів руху із застосуванням методів комп'ютерного зору й елементів штучного інтелекту.

Отримані результати підтверджують працездатність запропонованого рішення: модуль забезпечує послідовну обробку вхідних даних, формує часовий ряд координат (X , Y , Z), візуалізує траєкторію та зберігає результати у стандартизованих форматах для подальшого аналізу й документування.

У процесі дослідження було опрацьовано тематичну область інтелектуальних навігаційних систем БПЛА та уточнено роль задач локалізації й оцінювання руху в умовах, коли супутникова навігація є нестабільною або недоступною.

Проаналізовано основні вимоги до визначення траєкторії в реальному середовищі, зокрема необхідність узгодження систем координат, наявність шумів вимірювань, вплив якості зображення та динаміки руху на точність відновлення координат.

Окрему увагу приділено огляду сучасних підходів комп'ютерного зору, що застосовуються для відстеження переміщення між кадрами, а також способам підвищення надійності оцінювання положення за допомогою додаткових сенсорних даних (IMU).

На основі аналізу методів і практичних обмежень було сформульовано постановку задачі та обґрунтовано логіку математичної інтерпретації руху як послідовності просторових станів у часі.

Сформовано концепцію розв'язання проблеми, що передбачає перетворення сирих вхідних даних у координатну траєкторію через послідовність етапів: читання даних, попередня обробка, виділення опорних візуальних елементів, оцінювання пози/зміщення, накопичення координат, постобробка та контроль якості. Такий підхід забезпечив керованість

експерименту і дозволив чітко описати, які вхідні компоненти використовуються на кожному етапі та які артефакти формуються на виході.

Практичним результатом роботи стала розробка алгоритму оцінювання координат і траєкторії за відеоданими з можливістю розширення за рахунок даних сенсорів руху.

Реалізовано маркерно-орієнтований режим оцінювання пози, який дає змогу отримувати координати у метричному масштабі за умови відомого розміру маркера, а також альтернативний режим візуальної одометрії, що формує траєкторію за природними ознаками сцени у відносному масштабі.

Запропоновано механізми підвищення стабільності результатів, зокрема вибір найбільш надійного маркера при наявності кількох детекцій та постобробку траєкторії шляхом фільтрації аномальних стрибків і згладжування, що зменшує вплив випадкових похибок детекції або короточасних збоїв у відновленні пози.

У межах проєктування програмного продукту побудовано структурну та функціональну моделі модуля, визначено вимоги до його роботи та сформовано логічну організацію інформаційної бази.

Описано склад вхідних даних (кадри, часові мітки, метадані сенсора) і структуру вихідних результатів (координати траєкторії, допоміжні показники якості, графічні матеріали).

Реалізовано основні функції програмного модуля: завантаження і валідація вхідних файлів, попередня обробка кадрів, детекція візуальних елементів, оцінювання положення, формування траєкторії, її постобробка, візуалізація у 3D та у 2D-проекціях, а також експорт результатів у CSV/JSON і збереження графіків у форматі PNG.

Такий набір функцій забезпечує не лише демонстрацію працездатності алгоритму, а й повний цикл відтворюваного експерименту з можливістю повторного запуску й перевірки результатів.

Отже, виконана робота має прикладну цінність, оскільки запропонований програмний модуль може використовуватися як основа для

систем локалізації БПЛА в умовах обмеженої навігаційної інформації та як навчально-демонстраційний інструмент для аналізу траєкторії за відеоданими.

Подальший розвиток рішення доцільно спрямувати на повноцінну інтеграцію IMU (візуально-інерціальний підхід), використання реальної калібровки камери для підвищення точності, вдосконалення фільтрації траєкторії та тестування на різних датасетах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1 Kramarić L., Jelušić N., Radišić T., Muštra M. A Comprehensive Survey on Short-Distance Localization of UAVs. *Drones*. 2025. №9(3). С. 188.
- 2 Довбиш І., Муравйов О., Момот А., Богдан Г. Автономна навігація БПЛА: технології орієнтації та локалізації. *Прикладні питання математичного моделювання*. №8(1). С. 57-64.
- 3 Debnath D., Vanegas F., Sandino J., Hawary A. F., Gonzalez F. A review of UAV path-planning algorithms and obstacle avoidance methods for remote sensing applications. *Remote Sensing*, 2024, 16(21), 4019. URL: <https://www.mdpi.com/2072-4292/16/21/4019>
- 4 Бойко Х. А. Алгоритм оптичної навігації дрона: магістерська робота. Нац. ун-т «Києво-Могилянська акад.»; кер. А. М. Глибовець. Київ, 2025. 56 с.
- 5 Magrini G., Berlincioni L., Becattini F., Cultrera L., Pala P. Drone Detection with Event Cameras. *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2025. P. 4703-4714.
- 6 Кравчук А. Р., Ткачук А. Г., Добржанський О. О., Богоявленська Ю. В., Ткачук Д. Ю. Аналіз методів автономної навігації БПЛА в умовах відсутності GPS-сигналу. *Технічна інженерія*. 2025. №1(95). С. 235-242.
- 7 Розгортання моделей Ultralytics YOLO на пристроях Raspberry Pi. *Evo.net.ua*. URL: <https://evo.net.ua/deploying-ultralytics-yolo-models-on-raspberry-pi-devices/>
- 8 Nikolaiev M., Novotarskyi M. Comparative Review of Drone Simulators. *Information, Computing and Intelligent systems*. 2024. №4. Pp. 79-99.
- 9 Using SITL with Gazebo. *ArduPilot*. URL: <https://ardupilot.org/ardupilot/index.html>
- 10 Garrido-Jurado S., Muñoz-Salinas R., Madrid-Cuevas F. J., Marín-Jiménez M. J. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*. 2014. №47(6). P. 2280-2292.

11 Pose computation overview. OpenCV. URL: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html

12 Олійник, Р. Р. Автоматизована система розрахунку траєкторії руху об'єкта : дипломна робота ... бакалавра : 151 Автоматизація та комп'ютерно-інтегровані технології / Олійник Роман Романович. Київ, 2024. 64 с.

13 Петраш С. В., Запорожченко Ю. І. Підхід до визначення координат та параметрів руху об'єктів окремим радіотехнічним засобом, що працює в пасивному режимі. *Проблеми створення, випробування, застосування та експлуатації складних інформаційних систем*, 2015. №12. С. 93-100.

14 Миц А., Пуйда В., Цигилик Л. Визначення просторових координат об'єкта. 2009. URL: https://vlp.com.ua/files/09_19.pdf

15 Афтаназів І. С., Стоцько Р. З., Шевчук А. О., Строган О. І., Бойко О. О. Визначення координат та параметрів руху безпілотних літальних апаратів. *Системи озброєння і військова техніка*. 2022. №3 (71). С. 49-59.

16 Чернишев М. І., Куценко В. В. Оцінка точності визначення положення БПЛА різницево-далекомірним методом в рухомій системі пасивної радіолокації зенітних комплексів малої дальності. *Системи озброєння і військова техніка*. 2018. № 2. С. 61-66.

17 Kutsenko V. V., Kovalenko S. H., Dobrowolski D. D. Parameters numerical values of errors distribution law in coordinate measuring process at the difference-distance measuring passive location method. *Наука і техніка Повітряних Сил Збройних Сил України*. 2017. № 1. С. 82-84. <https://doi.org/10.30748/nitps.2017.26.17>.

18 Пулькевич І. Г. Лінійні оператори кінематичних проєкційних відображень. *Геометричне моделювання, інженерна та комп'ютерна графіка*: зб. наук. праць Львівської міжн. наук.-метод. конф. 1994. № 57. С. 35.

19 Куценко Л. М., Запольський Л. Л., Сухарькова О. І. Геометричне моделювання мобільної гравітаційної установки для захисту безпілотників типу літака. *Прикладна геометрія та інженерна графіка*. 2018. № 94. С. 60-65.

20 Вірченко С.Г. (2018). Деякі питання комплексного динамічного формоутворення на прикладі проектування крила літака. Геометричне моделювання мобільної гравітаційної установки для захисту безпілотників типу літака. *Прикладна геометрія та інженерна графіка*. № 94. С. 20-25.

21 Мірошніченко А. С. Відстеження та прогнозування траєкторії руху об'єктів у просторі за допомогою цифрових сигналів: бакалавр. робота. Суми : Сумський державний університет, 2025. 149 с.

22 Коротка Л. І., Науменко Н. Ю. Розробка нейромережевого програмного модуля планування траєкторії руху об'єкту на площині. *Інженерні та освітні технології*. 2015. №2. С. 28-35.

23 Катаєва М. О. Математична модель побудови траєкторії руху сенсора по поверхні деталі. *Вимірювальна та обчислювальна техніка в технологічних процесах*. 2016. №1. С. 91-94.

24 Зозуля В. М., Рижков О. В., Добришкін Ю. М., Собора А. І. Розробка методики визначення швидкості руху та кута сходу авіаційних засобів ураження з використанням цифрових засобів відеореєстрації. *Наука і техніка Повітряних Сил Збройних Сил України*. 2016. №2. С. 96-98.

25 Пінчук, В. Алгоритм оцінювання параметрів руху БПЛА за даними супутникової навігаційної системи: дипл. робота... бакалавра. Київ, 2021. 48 с.

26 Бурдейний М. В. Обґрунтування точності визначення параметрів руху літальних апаратів у кінці активної ділянки траєкторії польоту. *Проблеми створення, випробування, застосування та експлуатації складних інформаційних систем*. 2013. №7. С. 101-107.

27 Горбов В. Ю. Робастно-оптимальна система керування квадрокоптером в режимі руху програмованою траєкторією для задач моніторингу місцевості. *Сучасні проблеми автоматики та електротехніки*. 2024. С. 34-36.

28 Герасимов С. В., Ізосімов Д. М., Рощупкін Є. С., Богдановський О. М. Оцінка параметрів руху повітряних об'єктів при об'єднанні результатів незалежних первинних вимірювань в активній

багатопозиційній системі радіолокації. *Системи озброєння і військова техніка*. 2010. №3. С. 110-113.

29 Афтаназів І. С., Строган О. І., Шевчук Л. І., Струтинська Л. Р. Уточнення координат ворожих БПЛА засобами кінематичного проектування. *Scientific Bulletin of UNFU*. 2024. №34(1). С. 53-60.

30 Ткач В. С. Розробка та моделювання алгоритму керування дроном з використанням бібліотеки PyDu (Python Dynamics): дипломна робота ... бакалавра. Київ, 2024. 98 с.

31 Сліпенький В. В. Метод машинного навчання для торкання рукою точок у просторі: магістерська робота. Харків:ХНУРЕ, 2025. 50 с.

32 Седінкін О. А., Деркач М. В., Скарга-Бандурова І. С., Матюк Д. С. Система для відстеження руху очей на основі машинного навчання. *Комп'ютерно-інтегровані технології: освіта, наука, виробництво*. 2024. №55. С. 199-205.

33 Ткаченко О., Гребенюк М. Система визначення координат при автономному польоті БПЛА. *Управління та адміністрування в умовах протидії гібридним загрозам національній безпеці*: збірник матеріалів VI Міжнародної науково-практичної конференції (м. Київ, 26 листопада 2025 р.). Київ: НТУ, 2025.

34 Ткаченко О.А., Гребенюк М.С. Деякі аспекти програмного забезпечення визначення координат при автономному польоті БПЛА. *Цифрова платформа: інформаційні технології в соціокультурній сфері*. 2025. № 9(2).

ДОДАТКИ

ДОДАТОК А . Програмна реалізація модуля визначення координат і траєкторії руху БПЛА (тексти програмних файлів)

Лістинг А.1 inspect_dataset.py — перевірка структури датасету

```
from pathlib import Path

DATASET = Path(r"D:\uav_nav_trajectory_module\cam_april")

cam0_dir = DATASET / "mav0" / "cam0"
cam0_data = cam0_dir / "data"
cam0_csv = cam0_dir / "data.csv"
cam0_yaml = cam0_dir / "sensor.yaml"

print("cam0 images:", len(list(cam0_data.glob("*.png"))))
print("cam0 data.csv exists:", cam0_csv.exists())
print("cam0 sensor.yaml exists:", cam0_yaml.exists())
print("first 3 images:", [p.name for p in
sorted(cam0_data.glob("*.png"))[:3]])
```

Лістинг А.2 detect_tags_preview.py — тест детекції та вибір словника

```
from pathlib import Path
import cv2
import pandas as pd

DATASET = Path(r"D:\uav_nav_trajectory_module\cam_april")
cam0_dir = DATASET / "mav0" / "cam0"
data_dir = cam0_dir / "data"
csv_path = cam0_dir / "data.csv"

aruco = cv2.aruco

# зберемо ВСІ словники, які є у твоєму OpenCV
dicts = []
for name in dir(aruco):
    if name.startswith("DICT_"):
        val = getattr(aruco, name)
        if isinstance(val, int):
            dicts.append((name, val))
dicts = sorted(dicts, key=lambda x: x[0])

print("Dictionaries found:", len(dicts))

df = pd.read_csv(csv_path, comment="#", header=None,
names=["timestamp_ns", "filename"])
```

```

df["path"] = df["filename"].apply(lambda f: str(data_dir / f))

params = aruco.DetectorParameters()
# можна трохи "полегшити" детекцію
params.cornerRefinementMethod = aruco.CORNER_REFINE_SUBPIX

best = {"count": 0, "dict": None, "path": None, "ids": None,
        "corners": None, "img": None}

# скануємо перші N кадрів з кроком, щоб було швидко
N = min(len(df), 3000)
STEP = 10

for i in range(0, N, STEP):
    img = cv2.imread(df.loc[i, "path"], cv2.IMREAD_GRAYSCALE)
    if img is None:
        continue

    # трохи підвищимо шанс детекції
    img2 = cv2.resize(img, None, fx=1.5, fy=1.5,
                      interpolation=cv2.INTER_LINEAR)

    for dname, did in dicts:
        dictionary = aruco.getPredefinedDictionary(did)
        detector = aruco.ArucoDetector(dictionary, params)
        corners, ids, _ = detector.detectMarkers(img2)
        cnt = 0 if ids is None else len(ids)

        if cnt > best["count"]:
            best.update({"count": cnt, "dict": dname, "path":
df.loc[i, "path"], "ids": ids, "corners": corners, "img": img2})

        if best["count"] >= 10: # достатньо маркерів – можна
зупинятись
            break

print("BEST:")
print("  count:", best["count"])
print("  dict :", best["dict"])
print("  frame:", best["path"])

# показуємо результат, якщо щось знайшли
vis = cv2.cvtColor(best["img"] if best["img"] is not None else
img, cv2.COLOR_GRAY2BGR)
if best["count"] > 0:
    aruco.drawDetectedMarkers(vis, best["corners"], best["ids"])
cv2.imshow("Detected markers (best match)", vis)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

Лістинг A.3 aruco_pose_trajectory.py — ArUco-режим: координати, траєкторія, експорт

```
import json
from pathlib import Path

import cv2
import yaml
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

DATASET = Path(r"D:\uav_nav_trajectory_module\cam_april")
OUT_DIR = DATASET / "outputs_aruco"
OUT_DIR.mkdir(parents=True, exist_ok=True)

# Те, що ти вже знайшов:
DICT_NAME = "DICT_ARUCO_MIP_36H12"

# !!! ВАЖЛИВО: реальний розмір сторони маркера (в метрах).
# Якщо не знаєш точний — постав 0.04 або 0.05 як "параметр моделі".
MARKER_SIZE_M = 0.04

MAX_FRAMES = 2000
STEP = 5

def load_approx_K(sensor_yaml_path: Path):
    y = yaml.safe_load(sensor_yaml_path.read_text(encoding="utf-8"))
    w, h = map(int, y.get("resolution", [752, 480]))
    f = 0.9 * max(w, h)
    cx, cy = w / 2.0, h / 2.0
    K = np.array([[f, 0, cx],
                  [0, f, cy],
                  [0, 0, 1]], dtype=np.float64)
    dist = np.zeros((5, 1), dtype=np.float64)
    return K, dist

def main():
    cam0_dir = DATASET / "mav0" / "cam0"
    data_dir = cam0_dir / "data"
    csv_path = cam0_dir / "data.csv"
    sensor_yaml = cam0_dir / "sensor.yaml"

    df = pd.read_csv(csv_path, comment="#", header=None,
names=["timestamp_ns", "filename"])
    df["path"] = df["filename"].apply(lambda f: str(data_dir /
f))
```

```

df = df.iloc[:, :STEP].reset_index(drop=True)
if MAX_FRAMES is not None:
    df = df.iloc[:MAX_FRAMES].reset_index(drop=True)

aruco = cv2.aruco
dict_id = getattr(aruco, DICT_NAME)
dictionary = aruco.getPredefinedDictionary(dict_id)

params = aruco.DetectorParameters()
params.cornerRefinementMethod = aruco.CORNER_REFINE_SUBPIX
detector = aruco.ArucoDetector(dictionary, params)

K, dist = load_approx_K(sensor_yaml)

rows = [] # timestamp, X,Y,Z, n_ids

for i in range(len(df)):
    img = cv2.imread(df.loc[i, "path"],
cv2.IMREAD_GRAYSCALE)
    if img is None:
        continue

    # трохи збільшимо зображення (підвищує детекцію на
дрібних маркерах)
    img2 = cv2.resize(img, None, fx=1.5, fy=1.5,
interpolation=cv2.INTER_LINEAR)

    corners, ids, _ = detector.detectMarkers(img2)
    if ids is None or len(ids) == 0:
        continue

    rvecs, tvecs, _ =
aruco.estimatePoseSingleMarkers(corners, MARKER_SIZE_M, K, dist)

    # 1) обрати маркер з найбільшою площею (найстабільніший)
    areas = []
    for c in corners:
        pts = c.reshape(4, 2).astype(np.float32)
        areas.append(cv2.contourArea(pts))
    best_i = int(np.argmax(areas))

    rvec = rvecs[best_i].reshape(3, 1)
    tvec = tvecs[best_i].reshape(3, 1)

    # 2) інвертувати (отримуємо позицію камери у системі
маркера)
    R, _ = cv2.Rodrigues(rvec)
    cam_pos = -R.T @ tvec # (3,1)

    X, Y, Z = float(cam_pos[0, 0]), float(cam_pos[1, 0]),
float(cam_pos[2, 0])
    ts = int(df.loc[i, "timestamp_ns"])

```

```

# (опційно) збережемо ще id маркера
marker_id = int(ids[best_i][0])
rows.append((ts, X, Y, Z, int(len(ids)), marker_id))

if len(rows) % 100 == 0:
    print("saved points:", len(rows))

# ===== ФІЛЬТР ВІД СТРИВКІВ (вставляти ТУТ) =====
filtered = []
max_step = 0.5 # підбери за потреби (0.3..1.0)

prev = None
for r in rows:
    _, x, y, z, *rest = r
    if prev is None:
        filtered.append(r)
        prev = (x, y, z)
        continue

    dx, dy, dz = x - prev[0], y - prev[1], z - prev[2]
    step = (dx * dx + dy * dy + dz * dz) ** 0.5

    if step <= max_step:
        filtered.append(r)
        prev = (x, y, z)

rows = filtered
# ===== КІНЕЦЬ ФІЛЬТРА =====

# ===== Згладжування траєкторії (moving average) =====
WINDOW = 5 # 3..9, підбери
if len(rows) >= WINDOW:
    df_tr = pd.DataFrame(rows, columns=["timestamp_ns", "X",
    "Y", "Z", "n_markers", "marker_id"])
    df_tr["X"] = df_tr["X"].rolling(WINDOW, center=True,
    min_periods=1).mean()
    df_tr["Y"] = df_tr["Y"].rolling(WINDOW, center=True,
    min_periods=1).mean()
    df_tr["Z"] = df_tr["Z"].rolling(WINDOW, center=True,
    min_periods=1).mean()
    rows = list(df_tr.itertuples(index=False, name=None))

    out_csv = OUT_DIR / "trajectory_aruco.csv"
    out_json = OUT_DIR / "trajectory_aruco.json"

    pd.DataFrame(rows, columns=["timestamp_ns", "X", "Y", "Z",
    "n_markers", "marker_id"]).to_csv(out_csv, index=False)
    with open(out_json, "w", encoding="utf-8") as f:
        json.dump(
            [{"timestamp_ns": a, "X": b, "Y": c, "Z": d,
            "n_markers": e, "marker_id": m}

```

```

        for a, b, c, d, e, m in rows],
        f, ensure_ascii=False, indent=2
    )

    xs = [r[1] for r in rows]
    ys = [r[2] for r in rows]
    zs = [r[3] for r in rows]

    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")
    ax.plot(xs, ys, zs)
    ax.set_title(f"Траєкторія за ArUco ({DICT_NAME}),"
marker={MARKER_SIZE_M} m")
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")
    plt.tight_layout()
    out_png = OUT_DIR / "trajectory_aruco_3d.png"
    plt.savefig(out_png, dpi=200)
    plt.show()

    plt.figure()
    plt.plot(xs, ys)
    plt.title("Проекція траєкторії на XY")
    plt.xlabel("X");
    plt.ylabel("Y")
    plt.tight_layout()
    plt.savefig(OUT_DIR / "trajectory_aruco_xy.png", dpi=200)
    plt.show()

    plt.figure()
    plt.plot(xs, zs)
    plt.title("Проекція траєкторії на XZ")
    plt.xlabel("X");
    plt.ylabel("Z")
    plt.tight_layout()
    plt.savefig(OUT_DIR / "trajectory_aruco_xz.png", dpi=200)
    plt.show()

    print("DONE")
    print("CSV:", out_csv)
    print("JSON:", out_json)
    print("PLOT:", out_png)

if __name__ == "__main__":
    main()

```

Лістинг A.4 run_vo_cam0.py — VO-режим: ORB → recoverPose

→ траєкторія

```
import os
import json
from pathlib import Path

import cv2
import yaml
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

DATASET = Path(r"D:\uav_nav_trajectory_module\cam_april")
OUT_DIR = DATASET / "outputs_vo"
OUT_DIR.mkdir(parents=True, exist_ok=True)

MAX_FRAMES = 2000      # можна зменшити для швидкого тесту (наприклад 500)
STEP = 2               # 1 = кожен кадр, 2 = кожен другий (швидше)

def load_camera_intrinsics(sensor_yaml_path: Path):
    y = yaml.safe_load(sensor_yaml_path.read_text(encoding="utf-8"))

    # Якщо intrinsics все ж є (на майбутнє)
    if isinstance(y, dict) and "intrinsics" in y:
        fx, fy, cx, cy = map(float, y["intrinsics"])
        K = np.array([[fx, 0, cx],
                      [0, fy, cy],
                      [0, 0, 1]], dtype=np.float64)
        dist_list = y.get("distortion_coefficients", [0, 0, 0, 0, 0])
        dist = np.array(dist_list, dtype=np.float64).reshape(-1, 1)
        if dist.shape[0] < 5:
            dist = np.vstack([dist, np.zeros((5 - dist.shape[0], 1))])
        return K, dist[:5], y.get("distortion_model", "unknown")

    # Твій випадок: є тільки resolution
    res = y.get("resolution", None)
    if res is None:
        raise RuntimeError("У sensor.yaml немає intrinsics і немає resolution.")

    w, h = int(res[0]), int(res[1])

    # Апроксимація фокусної: ~0.9 * max(w, h)
    f = 0.9 * max(w, h)
    cx, cy = w / 2.0, h / 2.0

    K = np.array([[f, 0, cx],
                  [0, f, cy],
                  [0, 0, 1]], dtype=np.float64)

    dist = np.zeros((5, 1), dtype=np.float64)
    return K, dist, "approx_from_resolution"
```

```

def load_cam0_frames(cam0_dir: Path):
    csv_path = cam0_dir / "data.csv"
    data_dir = cam0_dir / "data"

    # У EuRoC форматі перший рядок коментар: "#timestamp
    [ns],filename"
    df = pd.read_csv(csv_path, comment="#", header=None,
names=["timestamp_ns", "filename"])
    df["path"] = df["filename"].apply(lambda f: str(data_dir / f))
    return df

def orb_matches(img1, img2):
    orb = cv2.ORB_create(nfeatures=2500)

    k1, d1 = orb.detectAndCompute(img1, None)
    k2, d2 = orb.detectAndCompute(img2, None)

    if d1 is None or d2 is None or len(k1) < 20 or len(k2) < 20:
        return None, None

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)
    knn = bf.knnMatch(d1, d2, k=2)

    good = []
    for m, n in knn:
        if m.distance < 0.75 * n.distance:
            good.append(m)

    if len(good) < 40:
        return None, None

    pts1 = np.float32([k1[m.queryIdx].pt for m in good]).reshape(-1,
1, 2)
    pts2 = np.float32([k2[m.trainIdx].pt for m in good]).reshape(-1,
1, 2)
    return pts1, pts2

def main():
    cam0_dir = DATASET / "mav0" / "cam0"
    sensor_yaml = cam0_dir / "sensor.yaml"

    K, dist, model = load_camera_intrinsics(sensor_yaml)
    frames = load_cam0_frames(cam0_dir)

    # Візьмемо підвибірку
    frames = frames.iloc[:, :STEP].reset_index(drop=True)
    if MAX_FRAMES is not None:
        frames = frames.iloc[:, :MAX_FRAMES].reset_index(drop=True)

    # Початковий стан (світова система координат довільна)
    R_w = np.eye(3, dtype=np.float64)

```

```

t_w = np.zeros((3, 1), dtype=np.float64)

traj = [] # (i, timestamp, x,y,z)

prev = cv2.imread(frames.loc[0, "path"], cv2.IMREAD_GRAYSCALE)
if prev is None:
    raise RuntimeError("Не прочитав перший кадр cam0")

traj.append((0, int(frames.loc[0, "timestamp_ns"]), 0.0, 0.0,
0.0))

pairs_total = 0
pairs_ok = 0
pairs_skip = 0

for i in range(1, len(frames)):
    cur = cv2.imread(frames.loc[i, "path"], cv2.IMREAD_GRAYSCALE)
    if cur is None:
        continue

    pairs_total += 1

    pts1, pts2 = orb_matches(prev, cur)
    if pts1 is None:
        pairs_skip += 1
        prev = cur
        continue

    # Оцінка есенціальної матриці та відновлення руху
    E, mask = cv2.findEssentialMat(
        pts1, pts2, K,
        method=cv2.RANSAC, prob=0.999, threshold=1.0
    )
    if E is None:
        prev = cur
        continue

    _, R, t, mask_pose = cv2.recoverPose(E, pts1, pts2, K)

    scale = 1.0

    R_prev = R_w.copy()
    t_w = t_w + (R_prev @ (t * scale))
    R_w = R_w @ R

    x, y, z = float(t_w[0, 0]), float(t_w[1, 0]), float(t_w[2, 0])
    ts = int(frames.loc[i, "timestamp_ns"])
    traj.append((i, ts, x, y, z))

    pairs_ok += 1

    prev = cur

    if i % 100 == 0:
        print(f"Processed {i}/{len(frames)} frames, points in
traj={len(traj)}")

```

```

# ===== Експорт CSV =====
out_csv = OUT_DIR / "trajectory_vo_cam0.csv"
pd.DataFrame(traj, columns=["frame_idx", "timestamp_ns", "X", "Y",
"Z"]).to_csv(out_csv, index=False)

# ===== Експорт JSON =====
out_json = OUT_DIR / "trajectory_vo_cam0.json"
with open(out_json, "w", encoding="utf-8") as f:
    json.dump(
        [{"frame_idx": a, "timestamp_ns": b, "X": c, "Y": d, "Z":
e} for a, b, c, d, e in traj],
        f, ensure_ascii=False, indent=2
    )

# ===== Візуалізація 3D =====
xs = [p[2] for p in traj]
ys = [p[3] for p in traj]
zs = [p[4] for p in traj]

fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
ax.plot(xs, ys, zs)
ax.set_title("VO траєкторія за cam0 (масштаб відносний)")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
plt.tight_layout()
out_png = OUT_DIR / "trajectory_vo_cam0_3d.png"
plt.savefig(out_png, dpi=200)
plt.show()

# ===== 2D проєкції =====
plt.figure()
plt.plot(xs, ys)
plt.title("Проекція траєкторії на площину XY")
plt.xlabel("X")
plt.ylabel("Y")
plt.tight_layout()
plt.savefig(OUT_DIR / "trajectory_vo_cam0_xy.png", dpi=200)
plt.show()

plt.figure()
plt.plot(xs, zs)
plt.title("Проекція траєкторії на площину XZ")
plt.xlabel("X")
plt.ylabel("Z")
plt.tight_layout()
plt.savefig(OUT_DIR / "trajectory_vo_cam0_xz.png", dpi=200)
plt.show()

print(f"Frame pairs total: {pairs_total}")
print(f"Pairs OK (pose recovered): {pairs_ok}")
print(f"Pairs skipped (not enough matches): {pairs_skip}")
print(f"Trajectory points: {len(traj)}")

```

```

print("DONE")
print("CSV:", out_csv)
print("JSON:", out_json)
print("PLOT:", out_png)

if __name__ == "__main__":
    main()

```

Лістинг A.5 generate_marker.py — генерація маркерів/дошки

```

import os
import cv2

out_dir = os.path.join(os.path.dirname(__file__), "..",
                        "assets")
os.makedirs(out_dir, exist_ok=True)

aruco = cv2.aruco
dictionary = aruco.getPredefinedDictionary(aruco.DICT_4X4_50)

marker_id = 0
side_px = 1200

marker_img = aruco.generateImageMarker(dictionary, marker_id,
side_px)
cv2.imwrite(os.path.join(out_dir, f"aruco_id{marker_id}.png"),
marker_img)

print("Готово: assets/aruco_id0.png")

```