НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет робототехніки та приладобудування
Кафедра комп'ютерно-інтегрованих технологій виробництва приладів

До захисту допущено:
Завідувач кафедри
Наталія СТЕЛЬМАХ
«12» 06 2026 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютерно-інтегровані системи та технології у приладобудуванні»
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»
на тему: «Автоматизована система супроводу рухомої цілі безпілотним літальним апаратом»

Виконала:
студентка IV курсу, групи ПБ-21
Перепечай Соломія Ігорівна
Керівник:
доцент, к.т.н,
Стельмах Наталія Володимирівна
Рецензент:
доцент, к.т.н, каф АСНК
Нечай Сергій Олексійович

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

Київ – 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет робототехніки та приладобудування
Кафедра комп'ютерно-інтегрованих технологій виробництва приладів

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»
Освітньо-професійна програма «Комп'ютерно-інтегровані системи та технології в приладобудуванні»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Наталія СТЕЛЬМАХ
«12» 06 2026 р.

ЗАВДАННЯ
на дипломну роботу студентці
Перепечай Соломії Ігорівні

1. Тема роботи «Автоматизована система супроводу рухомої цілі безпілотним літальним апаратом», керівник роботи Стельмах Наталія Володимирівна, доцент, к.т.н., затверджені наказом по університету від «26» 05 2026 р. № 1905-с
2. Термін подання студентом роботи 05 червня 2026 року.
3.Вихідні дані до роботи: безпілотний літальний апарат масою 2,5 кг має максимальне горизонтальне прискорення 5 м/с² та вертикальне 4 м/с². Його максимальна швидкість становить 18 м/с, крейсерська – 12 м/с. Кути крену і тангажу обмежені ±30°, максимальна кутова швидкість – 200°/с. Робоча висота польоту знаходиться в межах 30–150 м. На апарат діють аеродинамічні збурення у вигляді вітру до 6 м/с з випадковими поривами.
4. Зміст пояснювальної записки: Розділ 1. Аналіз сучасних технологій переслідування цілей дронами та методів оптимізації 1.1. Класифікація алгоритмів переслідування БПЛА та їхні функціональні можливості 1.2. Методи автоматичного переслідування рухомих об'єктів 1.3. Огляд математичних моделей для навігації та керування 1.4. Методи оптимізації в задачах стеження і навігації 1.5. Висновки до розділу 1. Розділ 2. Побудова математичної моделі переслідування та формалізація задачі оптимізації 2.1. Формалізація задачі переслідування рухомої цілі 2.2. Математична модель руху дрона і цілі 2.3. Обмеження, початкові та граничні умови функціонування автоматизованої системи супроводу рухомої цілі 2.4. Вибір методу оптимізації задачі автоматизованого супроводу рухомої цілі безпілотним літальним апаратом 2.5. Висновки до розділу 2. Розділ 3. Розробка та моделювання автоматизованої системи з оптимізацією траєкторії 3.1. Структура автоматизованої системи керування дроном 3.2. Реалізація моделі на мові програмування Python та моделювання в QGroundControl 3.3. Інтеграція оптимізаційного алгоритму в

систему навігації 3.4. Порівняння результатів переслідування з/без оптимізації Висновки до розділу 3.

5. Перелік графічного матеріалу: Структурно-функціональна сх автоматизованої системи оптичного супроводу та наведення БПЛА, Гра залежності кутового відхилення цілі від зміщення відносно центра кадру, Гра залежності оціненого бокового зміщення цілі від піксельного зміщення різній висоті польоту БПЛА, Параметрична схема ОК із вказаними фізич величин, Структурна схема автоматизованої системи взаємодії БПЛА середовщем у якому він перебуває, Траєкторія цілі у площині зображе вимірювання, еталон та оцінка після адаптивного фільтра Калмана, З координати North у часі без фільтрації та після адаптивного згладжува Презентація проєкту.

6. Дата видачі завдання 20 березня 2026 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз сучасних технологій переслідування цілей дронами та методів оптимізації	17.03.2026	виконано
2	Огляд математичних моделей для навігації та керування	23.03.2026	виконано
3	Побудова математичної моделі переслідування та формалізація задачі оптимізації	03.04.2026	виконано
4	Математична модель руху дрона і цілі (кінематика, динаміка)	10.04.2026	
5	Вибір та обґрунтування методу оптимізації	17.04.2026	
6	Розробка та моделювання автоматизованої системи з оптимізацією траєкторії	20.04.2026	
7	Реалізація моделі на мові програмування Python та моделювання в QGroundControl	14.05.2026	
8	Аналіз результатів: точність, енергоефективність, швидкодія	02.05.2026	
9	Оформлення дипломної роботи	20.05.2026	

Студентка _____ Соломія ПЕРЕПЕЧАЙ
(підпис)

Керівник _____ Наталія СТЕЛЬМАХ
(підпис)

АНТОТАЦІЯ

Під час виконання дипломної роботи «Автоматизована система супроводу рухомої цілі безпілотним літальним апаратом» було проведено аналіз методів відстеження та супроводу рухомих об’єктів, що дозволило обрати оптимальний підхід для реалізації математичної моделі. Робота складається з трьох розділів: аналітичного, математичного та розділу моделювання. Зміст роботи викладено викладено на 100 сторінках, містить 14 рисунків, 1 таблицю, 17 формул, 1 додаток та 27 літературних джерел.

У першому розділі виконано аналіз сучасних технологій переслідування рухомих цілей безпілотними літальними апаратами, розглянуто класифікацію алгоритмів супроводу, методи автоматичного стеження, математичні моделі навігації та керування, а також підходи до оптимізації в задачах супроводу.

У другому розділі здійснено формалізацію задачі переслідування рухомої цілі, побудовано математичну модель руху дрона і цілі, визначено обмеження, початкові та граничні умови функціонування системи, а також обґрунтовано вибір методу оптимізації.

У третьому розділі розроблено та реалізовано програмну модель автоматизованої системи супроводу з оптимізацією траєкторії, виконано моделювання процесу переслідування, інтеграцію алгоритму оптимізації в систему навігації та проведено порівняльний аналіз ефективності роботи системи.

Ключові слова: безпілотний літальний апарат, ціль, супровід, математична модель, алгоритм.

ANNOTATION

During the preparation of the diploma thesis “Automated System for Tracking a Moving Target by an Unmanned Aerial Vehicle”, an analysis of methods for detecting and tracking moving objects was carried out, which made it possible to choose the optimal approach for developing a mathematical model.

The thesis consists of three sections: analytical, mathematical, and simulation. The content of the thesis is presented on 100 pages and includes 13 figures, 1 table, 17 formulas, 1 appendix, and 27 references.

The first section analyzes modern technologies for pursuing moving targets using unmanned aerial vehicles. It considers the classification of tracking algorithms, methods of automatic target tracking, mathematical models of navigation and control, as well as optimization approaches in tracking tasks.

The second section formalizes the task of pursuing a moving target, develops a mathematical model of the drone and target motion, defines the constraints, initial and boundary conditions of the system operation, and justifies the choice of the optimization method.

The third section develops and implements a software model of the automated tracking system with trajectory optimization. It includes simulation of the pursuit process, integration of the optimization algorithm into the navigation system, and a comparative analysis of the system’s efficiency.

Keywords: unmanned aerial vehicle, target, tracking, mathematical model, algorithm.

ЗМІСТ

ВСТУП

Безпілотні літальні апарати сьогодні широко використовуються у багатьох сферах, зокрема для спостереження, швидкого аналізу територій, пошуку об’єктів і супроводу рухомих цілей. Особливо важливим є створення автоматизованих систем, які дозволяють дрону самостійно виявляти ціль, визначати її положення та підтримувати супровід у процесі руху.

Актуальність теми полягає в тому, що автоматизований супровід рухомої цілі потребує точного керування польотом, швидкої обробки інформації та вибору оптимальної траєкторії руху БПЛА. Для оптимізації роботи таких систем необхідно використовувати математичні моделі руху дрона і цілі, а також математичні та алгоритмічні підходи для знаходження найкращого значення цільової функції за певних обмежень, які дають змогу покращити точність стеження, швидкодію та енергоефективність.

Метою роботи є підвищення ефективності функціонування систем автоматизованого супроводу рухомих цілей безпілотними літальними апаратами на основі використання методів математичного моделювання та оптимізації.

Для досягнення поставленої мети в роботі необхідно проаналізувати сучасні технології переслідування цілей дронами, розглянути методи автоматичного супроводу рухомих об’єктів, дослідити математичні моделі навігації та керування, обґрунтувати вибір методу оптимізації, а також реалізувати модель системи у програмному середовищі та виконати порівняння результатів роботи з оптимізацією і без неї.

Об’єктом дослідження є процес автоматизованого супроводу рухомої цілі безпілотним літальним апаратом.

Предметом дослідження є методи математичного моделювання, керування та оптимізації траєкторії руху БПЛА в задачі переслідування цілі.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ ПЕРЕСЛІДУВАННЯ ЦІЛЕЙ ДРОНАМИ ТА МЕТОДІВ ОПТИМІЗАЦІЇ

1.1. Класифікація алгоритмів переслідування БПЛА та їхні функціональні можливості

Автоматизований супровід рухомої цілі безпілотним літальним апаратом ґрунтується на використанні спеціальних алгоритмів, які забезпечують виявлення об’єкта, визначення його координат, оцінювання параметрів руху, прогнозування подальшої траєкторії та обчислення і передачу сигналів на виконавчі механізми (двигуни, сервоприводи) для забезпечення заданої траєкторії польоту, стабілізації та виконання місії БЛА. У межах такої задачі дрон повинен не лише виявити ціль, а й підтримувати її супровід упродовж певного часу, адаптуючись до зміни положення об’єкта, умов середовища та власних динамічних обмежень. Саме тому алгоритми переслідування є однією з ключових складових автоматизованих систем керування БПЛА. Залежно від принципу роботи, джерел вхідної інформації та способу формування команд керування такі алгоритми можна класифікувати за кількома основними ознаками.

За способом отримання інформації про ціль алгоритми поділяються на **візуальні, координатні та комбіновані** [1]. Візуальні алгоритми використовують дані з камери або іншої оптичної системи для виявлення, розпізнавання та супроводу об’єкта. У цьому випадку ціль визначається за її положенням у кадрі, формою, контуром, кольоровими ознаками або іншими характеристиками зображення. Основною перевагою візуальних алгоритмів є можливість роботи в режимі реального часу без потреби у комплексі технічних засобів, систем та сигналів, що знаходяться поза безпілотником забезпечують його позиціонування, орієнтацію та навігацію в просторі. Вони дозволяють реалізовувати автономний супровід навіть у тих випадках, коли відсутній прямий доступ до координат цілі. Разом з тим ефективність таких алгоритмів суттєво залежить від якості зображення, рівня освітлення, погодних умов, фону, перекриття об’єкта іншими предметами, а також від обчислювальних можливостей бортової системи.

Координатні алгоритми базуються на використанні GPS, телеметрії, інерціальних датчиків або інших технологій — від фізичних датчиків до

програмних моделей, що забезпечують визначення місцезнаходження, орієнтації та швидкості об'єктів, що дає змогу визначати положення цілі в просторі у вигляді координат, швидкості та напряму руху. Такий підхід є доцільним у випадках, коли ціль обладнана передавачем координат або коли її положення може визначатися зовнішньою системою спостереження. Перевагою координатних алгоритмів є можливість визначення його точних координат та орієнтації у реальному часі та побудови траєкторії переслідування у глобальній системі координат. Недоліком є залежність від точності похибки GNSS, впливу оточення, РЕБ, наявності стабільного зв’язку та можливих затримок під час передавання даних.

Комбіновані алгоритми поєднують кілька джерел інформації, зокрема візуальні та координатні дані, що підвищує надійність і точність супроводу (рис. 1.1) [1]. Такий підхід є найбільш перспективним, оскільки дозволяє компенсувати недоліки окремих методів. Наприклад, при короткочасній втраті візуального контакту з ціллю система може використовувати прогнозовані координати об’єкта, а після повторного виявлення цілі уточнювати її положення за зображенням. Саме комбіновані алгоритми найчастіше застосовуються у сучасних інтелектуальних системах супроводу БПЛА, оскільки вони демонструють високу стійкість до не визначеностей параметрів та зовнішніх збурень, оскільки вони об'єднують переваги декількох методів.

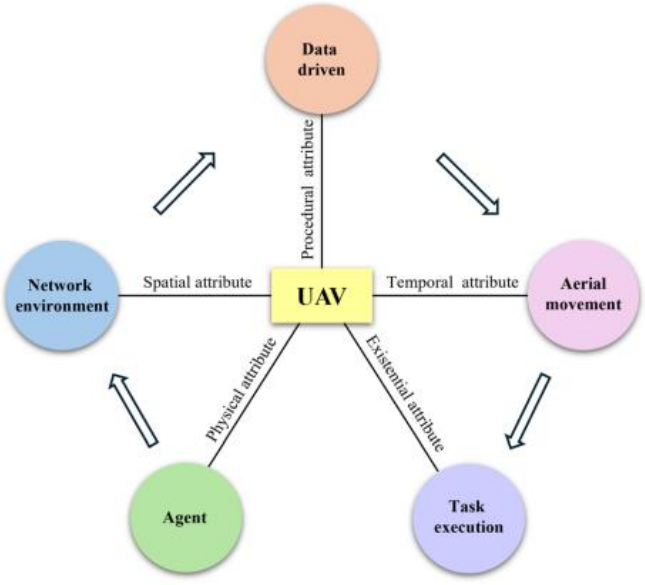


Рис. 1.1 Концептуальна схема БПЛА в сучасних технологічних умовах [1].

За методом формування траєкторії переслідування виділяють алгоритми прямого наведення, алгоритми прогнозування руху цілі та оптимізаційні алгоритми**.** Алгоритми прямого наведення орієнтують БПЛА безпосередньо на поточне положення цілі. Вони відносно прості в реалізації, не потребують складних обчислень і можуть бути ефективними в задачах із порівняно простим характером руху об’єкта. Проте в умовах, коли ціль змінює напрямок або швидкість, такі алгоритми можуть демонструвати недостатню точність, оскільки система реагує лише на поточний стан без урахування подальшого розвитку ситуації.

Алгоритми прогнозування руху цілі враховують не лише поточні координати об’єкта, а й швидкість, напрям, прискорення та характер зміни траєкторії. Це дозволяє оцінювати прогнозоване місцезнаходження цілі та формувати більш точну траєкторію переслідування. Застосування прогнозних моделей є особливо важливим у тих випадках, коли між вимірюванням стану цілі та виконанням керуючої дії існує затримка. У таких умовах орієнтація лише на поточне положення об’єкта може призвести до запізнення реакції БПЛА та збільшення похибки супроводу. Саме тому прогнозні алгоритми вважаються найбільш відповідними для задач автоматизованого переслідування рухомих цілей у реальному середовищі.

Оптимізаційні алгоритми забезпечують вибір найефективнішого маршруту або керуючої дії за певним критерієм, наприклад за мінімумом часу, енерговитрат, довжини траєкторії або похибки стеження. На відміну від простих алгоритмів наведення, вони дозволяють формалізувати задачу переслідування як задачу мінімізації або максимізації певного функціоналу якості. Це дає можливість враховувати одночасно кілька факторів: положення цілі, динамічні можливості БПЛА, перешкоди, ризики втрати об’єкта та обмеження системи керування. Саме оптимізаційні алгоритми є основою сучасних інтелектуальних систем навігації та супроводу, особливо в складних і динамічних умовах.

За способом керування рухом БПЛА алгоритми можна поділити на **реактивні та планувальні**. Реактивні алгоритми швидко реагують на зміну положення цілі та умов середовища, однак зазвичай працюють на основі поточних

даних без довгострокового планування. Їх перевага полягає у високій швидкості обробки та простоті реалізації, що робить їх зручними для локального керування та швидкої корекції траєкторії. Проте через відсутність прогнозу майбутніх станів такі алгоритми можуть генерувати неефективні або нестабільні траєкторії, особливо у випадку складного маневрування цілі.

Планувальні алгоритми, навпаки, враховують прогнозований рух цілі та дозволяють заздалегідь забезпечувати формування оптимальної траєкторії переслідування. У цьому випадку система не лише реагує на зміну ситуації, а й намагається передбачити подальший розвиток подій, визначаючи керуючі сигнали з урахуванням прогнозу. Такий підхід є більш складним з точки зору реалізації, однак забезпечує вищу точність стеження, зменшення кількості різких маневрів і покращення енергоефективності польоту. Саме тому в задачах автоматизованого супроводу рухомої цілі планувальні алгоритми мають суттєву перевагу над суто реактивними схемами.

Основними функціональними можливостями алгоритмів переслідування є **виявлення та ідентифікація цілі, визначення координат і параметрів руху об'єкта, прогнозування подальшого переміщення цілі, формування команд керування для БПЛА, корекція траєкторії польоту в реальному часі, а також забезпечення стійкості супроводу в умовах дії збурень та невизначеностей**. Крім того, сучасні алгоритми часто виконують додаткові функції, пов'язані з оцінюванням ризику втрати цілі, контролем допустимого положення об'єкта в полі зору камери, вибором безпечної дистанції до об'єкта та узгодженням даних від кількох сенсорів.

1.2 Методи автоматичного переслідування рухомих об'єктів

У задачах автоматизованого супроводу рухомих об'єктів безпілотним літальним апаратом застосовують сукупність методів, що забезпечують виявлення цілі, оцінювання її поточного стану, прогнозування подальшого руху та формування керуючих сигналів для корекції траєкторії польоту. Ефективність переслідування визначається точністю локалізації об'єкта, швидкістю виконання

алгоритмів обробки даних, стійкістю до зовнішніх збурень і здатністю системи працювати в умовах часткової невизначеності. Сучасні дослідження у цій галузі підкреслюють, що для стабільного супроводу необхідно одночасно забезпечити безперервну видимість цілі, безпеку польоту, плавність траєкторії та своєчасне реагування на зміни динаміки рухомого об’єктах[2].

Найбільш поширеним підходом є **візуальне переслідування**, при якому джерелом інформації про ціль виступає бортова камера БПЛА. У цьому випадку система виконує обробку відеопотоку, виявляє об’єкт у кадрі, визначає його положення та оцінює зміщення відносно центра зображення. На основі цієї інформації формуються керуючі сигнали, які забезпечують утримання цілі в полі зору та коригування просторового положення дрона. Перевагою такого підходу є автономність роботи та можливість функціонування без постійного надходження позиційних данних об'єкта. Разом із тим, точність візуального супроводу залежить від освітлення, масштабу об’єкта, наявності перекриттів, фону, погодних умов та обчислювальних можливостей бортової системи. У сучасних оглядових роботах саме зміна масштабу, оклюзії, швидкі маневри цілі та обмежене поле зору визначаються як головні проблеми візуального трекінгу для БПЛА.

Важливе місце займають **координатні методи переслідування**, у яких положення цілі визначається за допомогою супутникової навігації, телеметричних каналів, інерціальних датчиків або зовнішніх систем спостереження. За такого підходу траєкторія руху БПЛА формується на основі геопозиції об'єкта у просторі, що дозволяє виконувати супровід на значних відстанях та зменшує залежність від умов візуального спостереження. Проте координатні методи є чутливими до точності позиціонування, затримок передавання даних та можливих втрат зв’язку, тому на практиці вони часто застосовуються разом із засобами локального візуального контролю.

Для підвищення точності та стійкості переслідування застосовують **методи прогнозування руху цілі**, що базуються на оцінюванні її швидкості, напряму та характеру зміни траєкторії. Такі методи дозволяють забезпечити миттєвішу реакцію системи керування, оскільки БПЛА орієнтується не лише на поточне

положення об’єкта, а й на прогнозований стан у наступний момент часу. Особливо важливим це є у випадках нерівномірного або маневрового руху цілі. У сучасних дослідженнях прогнозування розглядається як один із ключових елементів систем переслідування, оскільки саме воно дозволяє підвищити надійність супроводу в мінливих умовах і при тимчасовій втраті візуального контакту.

Окрему групу становлять **методи візуального серво-керування** (visual servoing), у яких керування дроном здійснюється безпосередньо на основі параметрів зображення. У таких системах положення цілі в кадрі, її контур, розмір або інші візуальні ознаки використовуються як зворотний зв’язок для формування керуючих сигналів. Цей підхід забезпечує високу оперативність реагування та добре підходить для задач супроводу рухомих наземних або повітряних об’єктів. У recent працях показано, що поєднання visual servoing з адаптивним або ковзним керуванням підвищує стійкість системи до невизначеностей моделі, дії вітрових збурень і різких змін руху цілі[3].

У складніших сценаріях застосовують **комбіновані методи переслідування**, які інтегрують дані камер, навігаційних систем, прогнозних моделей і алгоритмів планування траєкторії. Такий підхід дозволяє компенсувати недоліки окремих методів: наприклад, при погіршенні візуального спостереження система може використовувати прогнозований рух цілі, а при накопиченні координатної похибки — уточнювати положення об’єкта за відеоданими. У сучасних науково-технічних розробках саме мультисенсорна інтеграція, кооперативне переслідування та поєднання трекінгу з алгоритмами оптимального планування траєкторії визначаються як перспективні напрями розвитку автоматизованих систем супроводу.

Одним із поширених підходів у задачах автоматичного супроводу рухомих об’єктів є **візуальний метод стеження**, який базується на обробці відеопотоку в реальному часі. У таких системах положення цілі визначається за зображенням, отриманим з бортової камери БПЛА, а подальше керування формується на основі зміщення об’єкта відносно центра кадру. У межах цього підходу широко застосовуються алгоритми **visual servoing**, у яких параметри зображення

безпосередньо використовуються як сигнали зворотного зв’язку для корекції траєкторії польоту. Перевагою візуального методу є можливість автономної роботи без постійної залежності від зовнішніх координатних джерел, а також висока оперативність формування керуючих сигналів у задачах супроводу наземних і повітряних цілей.

У сучасних системах візуальний супровід поєднується з алгоритмами трекінгу об’єкта, які дозволяють безперервно визначати положення цілі в послідовності кадрів, оцінювати зміну її координат і підтримувати об’єкт у полі зору камери. Такий підхід забезпечує швидке оновлення інформації про стан цілі та дає змогу оперативно реагувати на зміну її траєкторії руху. У сучасній фаховій літературі обгрунтовано, що поєднання візуального зворотного зв’язку з високошвидкісною обробкою зображення підвищує точність супроводу і покращує динамічні характеристики системи керування БПЛА.

Разом з тим ефективність візуального методу залежить від якості зображення, умов освітлення, наявності часткових перекриттів цілі, масштабу об’єкта та обчислювальних можливостей бортової системи. Саме тому в багатьох сучасних роботах візуальні методи доповнюються прогнозуванням руху цілі, компенсацією динамічних збурень і використанням нелінійних регуляторів. Така побудова системи дозволяє зменшити вплив затримок у каналі керування, підвищити стійкість супроводу та забезпечити можливість виконання керуючих сигналів у фізичному середовищі, що формуються для БПЛА.

З урахуванням сучасних тенденцій розвитку безпілотних систем візуальний супровід розглядається як один із базових підходів до побудови автоматизованих систем стеження за рухомими об’єктами. Це пояснюється тим, що такий метод дозволяє поєднати засоби комп’ютерного зору, алгоритми трекінгу, регулятори керування та елементи прогнозування в межах єдиної структури, придатної для роботи в реальному часі.

Таким чином, методи автоматичного переслідування рухомих об’єктів відрізняються способом отримання інформації про ціль, принципом формування керуючих сигналів і рівнем адаптації до умов середовища. Для задач супроводу

рухомої цілі безпілотним літальним апаратом найбільш доцільним є поєднання візуального стеження, прогнозування руху та алгоритмів корекції траєкторії, оскільки саме такий підхід забезпечує підвищення точності, швидкодії та надійності роботи системи в реальному часі.

1.3 Огляд математичних моделей для навігації та керування

У задачах навігації та керування безпілотними літальними апаратами математичні моделі використовують для опису руху БПЛА, оцінювання стану цілі, формування керуючих сигналів і побудови траєкторії польоту. У системах автоматизованого супроводу рухомих об’єктів моделі мають враховувати координати, швидкість, прискорення, орієнтацію апарата, а також вплив зовнішніх збурень, похибок вимірювання і затримок у каналах керування . У сучасних роботах з супроводу цілей БПЛА основну увагу приділяють поєднанню моделей руху, візуального зворотного зв’язку, прогнозування та оптимального керування [8].

До базових належать кінематичні моделі, у яких рух БПЛА і цілі описується через координати та їх зміну в часі без детального врахування сил і моментів. Такі моделі застосовують для аналізу траєкторії, обчислення відносного положення об’єктів, визначення похибки стеження та побудови спрощених законів наведення. Їх перевагою є простота реалізації та невисокі обчислювальні витрати, тому вони зручні на етапі первинного синтезу системи керування і при моделюванні руху цілі у дво- або тривимірному просторі [6]. У задачах visual servoing кінематичний опис часто пов’язують із координатами цілі в кадрі, що дозволяє безпосередньо використовувати геометричну похибку як сигнал зворотного зв’язку (Рис.1.2)[5].

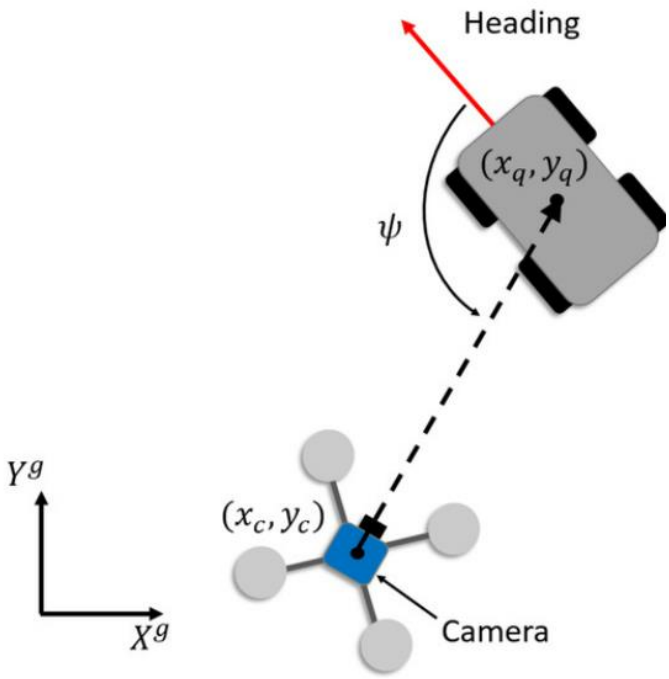


Рис.1.2 Визначення орієнтації цілі ψ, що спостерігається БПЛА[5].

Для точнішого опису функціонування пристрою використовують динамічні моделі, які враховують масу БПЛА, інерційні характеристики, сили, моменти та керуючі сигнали. На відміну від кінематичних, такі моделі дозволяють оцінити реальну реакцію літального апарата на команду керування, межі маневреності та вплив збурень. Це має значення у задачах супроводу, де траєкторія постійно коригується, а апарат повинен зберігати стійкість під час зміни швидкості, курсу або висоти, [7]. У статтях, присвячених visual servoing для БПЛА, саме динамічний опис системи використовується для побудови нелінійного керування та перевірки того, що сформовані керуючі сигнали не перевищують можливостей апарата.

Широко застосовуються моделі у просторі станів, у яких система описується вектором стану, вектором керування та вектором вихідних змінних. Такий підхід є забезпечує зручність для цифрової реалізації регуляторів, проектування спостерігачів стану та інтеграції алгоритмів оптимізації. Опис системи через змінні стану дозволяє формалізувати задачу супроводу як задачу переходу системи між станами з мінімізацією вибраного критерію: похибки стеження, часу зближення, енерговитрат або відхилення від заданої траєкторії. Для задач прогнозного керування цей метод моделювання динамічних систем, який базується на

використанні вектора стану для опису внутрішнього стану системи в будь-який момент часу, замість використання лише зв'язків між входом і виходом та вибирати керування за результатами оптимізації [8].

У задачах переслідування важливе місце займають моделі відносного руху БПЛА і цілі. У цьому випадку описується не абсолютне положення кожного об’єкта, а їх взаємне розташування: дальність, кут візування, відносна швидкість, зміщення цілі в кадрі або в локальній системі координат. Такий підхід спрощує задачу наведення, оскільки система керування працює безпосередньо з параметрами, які характеризують якість супроводу. У роботах із візуального серво-керування саме відносні координати цілі використовуються для формування сигналу керування, що дозволяє стабілізувати об’єкт у полі зору камери і зменшити похибку стеження [6].

Окрему групу становлять прогнозні моделі, призначені для оцінювання майбутнього положення рухомої цілі. Їх використання обумовлене тим, що між моментом вимірювання, обробкою інформації та виконанням команди існує затримка. Якщо керування формується лише за поточним станом об’єкта, це збільшує запізнення системи і може призвести до втрати цілі при різких маневрах. Прогнозна модель дозволяє оцінити майбутню траєкторію руху на основі попередніх спостережень, швидкості та напряму переміщення [8] У сучасних роботах для цього застосовують predictive control, імовірнісні підходи та методи навчання з підкріпленням, які поєднують прогноз стану з вибором керуючої дії [9].

Для підвищення точності навігації використовують оцінювальні моделі, що поєднують математичний опис руху з даними вимірювань. Реальні сигнали від GPS, IMU, камери або інших датчиків містять шум і похибки, тому прямих вимірювань недостатньо для стійкого супроводу. Поєднання моделі руху з алгоритмом оцінювання стану дає змогу згладжувати шум, уточнювати координати і відновлювати параметри руху цілі при короткочасній втраті спостереження [4], [10]. У системах переслідування це особливо важливо при оклюзіях, нестабільному освітленні та маневровому русі об’єкта [8].

У роботах з керування БПЛА значну увагу приділяють нелінійним моделям, оскільки рух апарата в реальних умовах не завжди може бути адекватно описаний лінійним наближенням. Нелінійність пов’язана з аеродинамічними ефектами, обмеженнями приводів, взаємозв’язком між кутовим і поступальним рухом, а також із насиченням керуючих каналів. Використання нелінійних моделей є доцільним у задачах активного переслідування, де БПЛА виконує часті маневри і працює під дією невизначених збурень [5]. У таких випадках застосовують адаптивне, робастне або ковзне керування, яке дозволяє зменшити вплив параметричної невизначеності та покращити стійкість системи [7].

У задачах побудови траєкторії переслідування використовують оптимізаційні моделі, у яких математичний опис руху БПЛА виступає основою для вибору найкращого керування за певним критерієм. Такими критеріями можуть бути мінімум похибки стеження, мінімум часу зближення, мінімум витрат енергії, збереження цілі в полі зору або дотримання обмежень безпеки польоту. Саме на основі таких моделей будуються алгоритми model predictive control, евристичного пошуку, еволюційної оптимізації та підкріплювального навчання [9]. Перспективність цих підходів пояснюється тим, що вони дозволяють одночасно враховувати динаміку апарата, прогноз руху цілі та обмеження середовища [8].

Отже, для підвищення точності навігації та ефективності управління БПЛА доцільно використовувати сукупність взаємопов’язаних математичних моделей. Кінематичні моделі застосовують для опису геометрії руху, динамічні — для врахування реальних властивостей апарата, моделі у просторі станів — для синтезу систем керування, прогнозні — для компенсації затримок і передбачення руху цілі, а оптимізаційні — для вибору раціональної траєкторії переслідування. Комплексне використання цих моделей створює теоретичну основу для побудови автоматизованої системи супроводу рухомої цілі безпілотним літальним апаратом.

1.4 Методи оптимізації в задачах стеження і навігації

У задачах стеження і навігації безпілотних літальних апаратів методи оптимізації використовуються для формування раціональної траєкторії польоту,

зменшення похибки супроводу цілі, скорочення часу реагування системи, зниження енерговитрат та врахування динамічних обмежень апарата. Для автоматизованого супроводу рухомої цілі оптимізація є необхідною, оскільки БПЛА повинен не лише зближуватися з об’єктом, а й підтримувати стійке спостереження, уникати перешкод, дотримуватись обмежень на швидкість і маневреність та забезпечувати придатність траєкторії до подальшого керування [11, 12]. У сучасних оглядах підкреслюється, що задача планування і стеження для БПЛА має багатокритеріальний характер, а тому оптимальна траєкторія визначається не одним параметром, а сукупністю показників: довжиною маршруту, часом руху, плавністю поворотів, безпекою, енергоспоживанням та якістю спостереження за ціллю [20].

Під час розв’язання задачі супроводу рухомої цілі важливо враховувати, що маршрут БПЛА не є сталим. На відміну від звичайної задачі прокладання шляху між двома фіксованими точками, у системі стеження кінцева точка змінюється в часі, а отже, траєкторія повинна постійно уточнюватися. Це означає, що оптимізація виконується як на етапі попереднього планування, так і під час польоту. Саме тому в сучасних роботах окремо розглядають глобальні алгоритми планування, локальні алгоритми перепланування, методи оптимального керування та комбіновані гібридні схеми. Такий підхід дозволяє поєднати переваги різних методів: глобальний пошук забезпечує структурований маршрут, а локальна корекція дозволяє адаптуватися до раптових змін у середовищі або поведінці цілі [18].

Одним із найпростіших підходів у задачах оптимізації є використання покрокової оптимізації. Її принцип полягає у виборі найкращого локального рішення на кожному кроці побудови траєкторії. Для БПЛА це може означати вибір напряму, який у поточний момент найшвидше зменшує відстань до цілі, скорочує кутову похибку спостереження або дозволяє швидко обійти найближчу перешкоду. Перевага такого підходу полягає у високій швидкодії та невеликій обчислювальній складності, що робить його корисним для роботи в реальному часі. Водночас покрокова оптимізація не гарантує глобально найкращого результату: локально

вигідний маневр може виявитися невигідним для всієї траєкторії, наприклад призвести до збільшення числа поворотів, втрати плавності шляху або погіршення умов спостереження за ціллю. Саме тому в сучасних роботах greedy-підхід найчастіше застосовують як допоміжний етап покращення маршруту після роботи іншого планувальника, а не як самостійний інструмент глобальної оптимізації (Рис.1.3) [16].

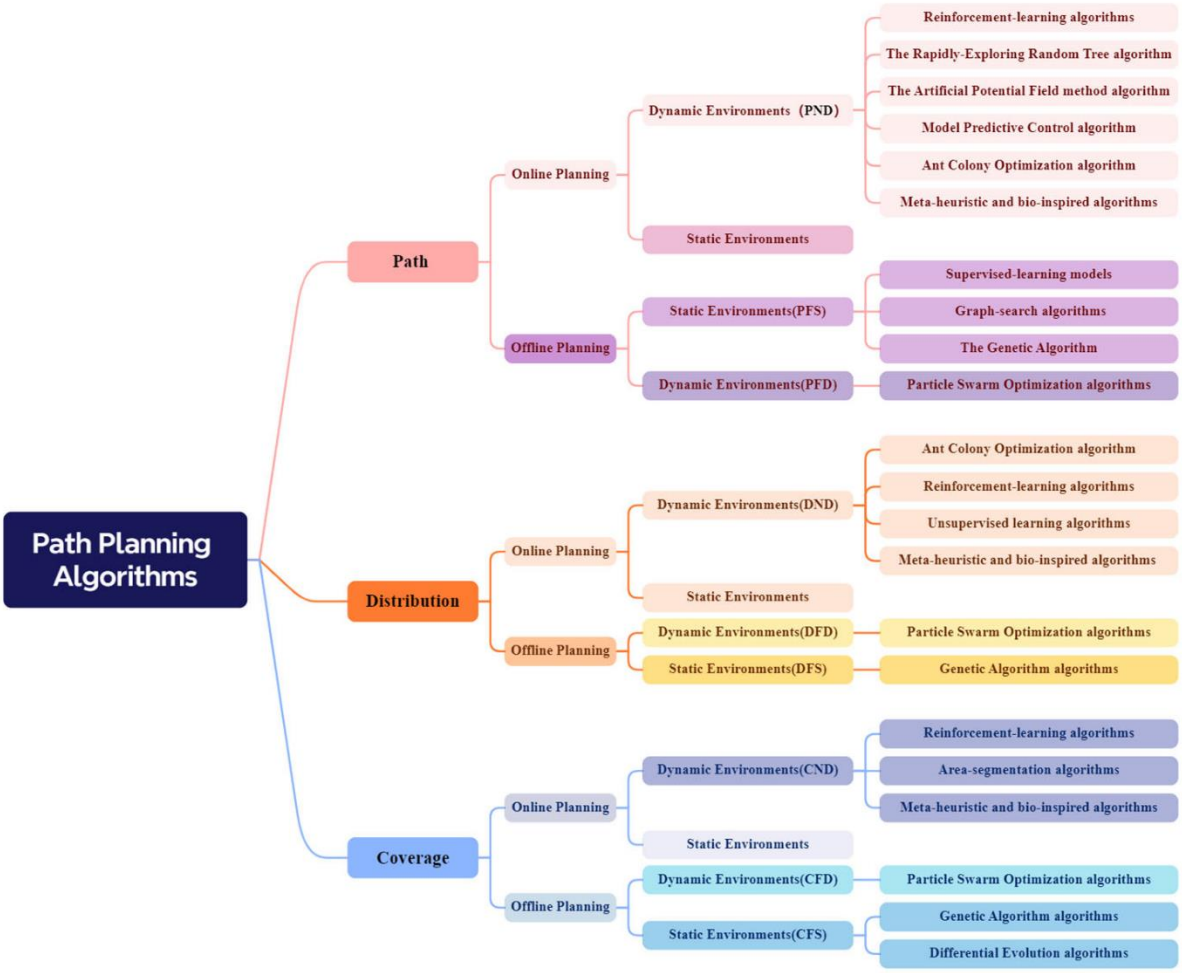


Рис.1.3 Сценарії місій БПЛА[12].

Практична цінність покрокової оптимізації для систем стеження полягає насамперед у локальній корекції траєкторії. Якщо ціль різко змінює напрям руху або на маршруті з’являється нова динамічна перешкода, повний перерахунок шляху може займати занадто багато часу. У такому випадку greedy-механізм дозволяє швидко перебудувати найближчий фрагмент траєкторії та зберегти безперервність супроводу. У статтях про сучасні модифікації RRT-подібних алгоритмів показано, що поетапне видалення вузлів і сегментів шляху, які

приносять найменшу користь дозволяє зменшити надлишковість маршруту, зробити його плавнішим та підвищити придатність до подальшого відстеження регулятором [19]. Отже, для задачі автоматизованого супроводу рухомої цілі алгоритм покрокової оптимізації доцільно розглядати як інструмент локального покращення, а не як основний засіб побудови всієї траєкторії.

Більш складним і гнучким підходом є використання генетичних алгоритмів. У таких методах кожна можлива траєкторія або набір параметрів маршруту подається як окреме рішення в популяції, після чого виконується ітеративний пошук кращих варіантів за допомогою відбору, схрещування та мутації. Генетичні алгоритми особливо корисні в багатокритеріальних задачах, де потрібно одночасно враховувати довжину шляху, витрати енергії, ризик зіткнення, плавність руху, безпечні відстані та якість супроводу цілі. Для задачі стеження це є важливим, оскільки маршрут повинен бути не лише коротким, а й таким, що забезпечує стабільне утримання цілі в полі зору та не створює надмірного навантаження на систему керування [17].

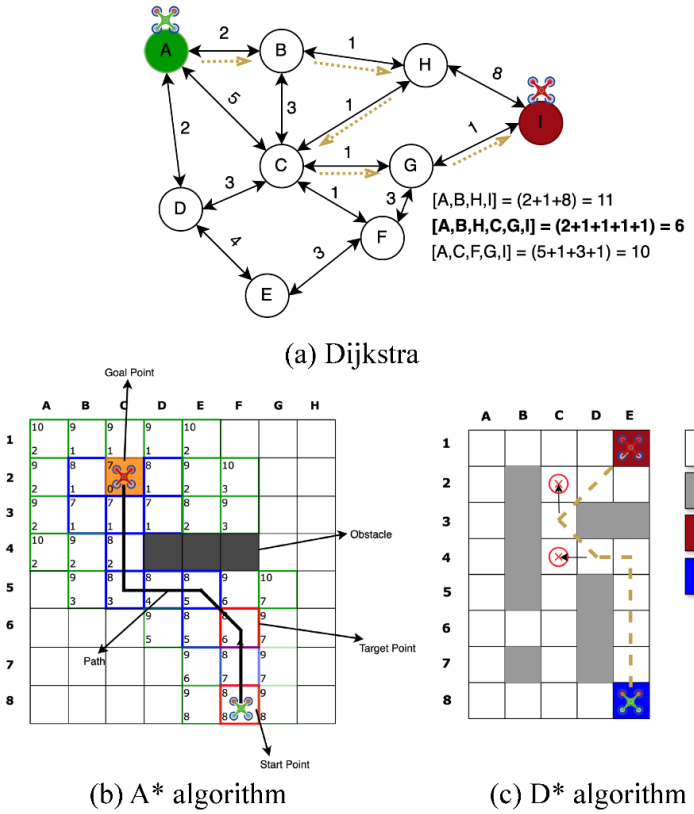
Перевагою генетичних алгоритмів є здатність працювати у великому просторі можливих рішень і знаходити компроміс між кількома критеріями. На відміну від локальних евристик, вони дозволяють досліджувати альтернативні варіанти маршруту і не настільки жорстко залежать від початкової конфігурації шляху. У сучасних дослідженнях генетичні алгоритми застосовують як до одиночних БПЛА, так і до багатокоптерних систем, де оптимізація охоплює не лише траєкторію, а й розподіл завдань між апаратами. Проте такі методи потребують більшого часу на обчислення, тому їх використання у високочастотному перепланування в реальному часі є обмеженим. Через це на практиці генетичні алгоритми частіше застосовують для глобального або напівглобального планування, після чого одержана траєкторія додатково уточнюється локальними алгоритмами [20].

Серед класичних методів оптимізації важливе місце займає динамічне програмування. Його застосування базується на принципі оптимальності, відповідно до якого оптимальний маршрут або послідовність керуючих дій

формується через оптимальні рішення на кожному попередньому кроці. Для систем автоматизованого супроводу це означає можливість оцінювати не лише поточний виграш від маневру, а й його вплив на подальше положення БПЛА відносно цілі. Такий підхід є корисним у тих випадках, коли короткочасне відхилення від прямого переслідування може забезпечити кращу оглядовість, менший ризик зіткнення або вигідніші умови для наступного маневру. У сучасних оглядах path planning саме динамічне програмування згадується як теоретично сильний інструмент для послідовної оптимізації рішень у часі, хоча його практичне використання часто обмежується високою розмірністю простору станів [20].

У системах навігації БПЛА значного поширення набули графові та евристичні алгоритми, зокрема A*, D*, Dijkstra, RRT і RRT*. Їх застосування є доцільним на етапі побудови базового маршруту, коли необхідно швидко знайти допустиму траєкторію в просторі з перешкодами. Алгоритм A* ефективний у дискретному середовищі, оскільки поєднує накопичену вартість шляху з евристичною оцінкою відстані до цілі. Це дозволяє відносно швидко отримувати оптимальний маршрут. Однак для БПЛА знайдений шлях часто потребує додаткового згладжування, оскільки ламана траєкторія не завжди відповідає кінематичним і динамічним обмеженням реального польоту. У свою чергу, RRT і RRT* краще працюють у безперервному просторі станів і можуть враховувати складні обмеження, але потребують подальшого покращення маршруту для зменшення надлишкових сегментів та підвищення плавності руху (Рис.1.4)[17].

Для задачі стеження за рухомою ціллю графові методи є корисними насамперед як засіб глобального планування, тоді як у реальному польоті вони часто поєднуються з локальними методами корекції. Одним із характерних прикладів є поєднання A* з методом штучних потенціальних полів. У таких схемах A* визначає загальний раціональний напрям руху, а APF виконує локальну адаптацію маршруту до динамічних перешкод або поточного положення цілі.



(a) Dijkstra

(b) A* algorithm

(c) D* algorithm

Рис.1.4 Порівняння алгоритмів пошуку оптимального маршруту та супроводу цілі: а) алгоритм Дейкстри; б) алгоритм A*; в) алгоритм D*. [15].

Подібні підходи показують хороші результати в середовищах з високим рівнем загроз, де необхідно одночасно враховувати небезпечні області, часові обмеження та потребу швидкого перепланування. Для автоматизованого супроводу це означає, що траєкторія може мати достатньо структурований глобальний характер, але при цьому залишатися адаптивною до локальних змін [16].

Особливе місце серед методів оптимізації посідає модельно-прогнозне керування. У MPC кожний часовий проміжок супроводжується визначенням оптимальних параметрів на скінченному горизонті прогнозу з урахуванням математичної моделі БПЛА, поточного стану системи та набору обмежень. Це дозволяє не лише вибрати бажану траєкторію, а й одразу сформувати керуючі сигнали, які є фізично реалізованими. Для задач автоматизованого супроводу рухомої цілі така властивість є принципово важливою, оскільки БПЛА повинен не просто “знати”, куди переміщатися, а й робити це без перевищення допустимих

швидкостей, прискорень, кутів повороту та інших обмежень польоту. Крім того, MPC дає можливість одночасно мінімізувати похибку стеження, обмежувати надлишкові керуючі сигнали та враховувати умови безпечного оминання перешкод [16].

У сучасних дослідженнях MPC часто використовується не лише для одиночного БПЛА, а й для групових систем. У такому випадку в оптимізаційну задачу вводяться додаткові складові: уникнення зіткнень між апаратами, узгодження розташування носіїв сенсорів, мінімізація “сліпих зон” та максимізація ймовірності видимості рухомої цілі. У статті про collaborative tracking in urban environment показано, що оптимізація траєкторій кількох БПЛА за критерієм максимальної видимості дозволяє покращити спостереження за об’єктом у міському середовищі, де огляд обмежується будівлями та іншими об’єктами [18]. Це особливо важливо для задачі супроводу, оскільки втрата прямої видимості цілі може призвести до погіршення точності стеження або повної втрати об’єкта.

Новим і перспективним напрямом є поєднання класичних методів оптимізації з підкріплювальним навчанням. У таких підходах нейромережева модель або агент навчання використовується для адаптивного налаштування параметрів оптимізації, вибору режиму руху або формування політики керування. Для БПЛА це означає можливість підвищити адаптивність системи до змін середовища, невизначеного руху цілі та часткової неповноти моделі. У роботі про trajectory planning and tracking with DRL and adaptive nonlinear MPC показано, що використання глибокого підкріплювального навчання для налаштування вагових коефіцієнтів у ANMPC дозволяє поліпшити точність відстеження траєкторії, швидкість реакції системи та стійкість у складних динамічних умовах [16]. Отже, сучасна тенденція полягає не у відмові від класичних оптимізаційних методів, а в їх інтелектуальному доповненні.

Гібридні підходи до оптимізації в задачах стеження і навігації БПЛА поєднують кілька рівнів обробки та прийняття рішень. На рівні глобального планування траєкторія може формуватися за допомогою алгоритмів A*, RRT*, генетичних або інших еволюційних методів, які дозволяють побудувати маршрут з

урахуванням загальної структури середовища та наявних обмежень. На рівні локальної корекції використовують potential field та споріднені підходи, що забезпечують швидке реагування на зміну положення цілі або появу нових перешкод. Формування керуючих сигналів доцільно покладати на MPC або інші прогнозні методи, оскільки вони дають змогу враховувати динамічні характеристики БПЛА та обмеження на керування. Додаткове використання елементів машинного навчання розширює можливості системи в умовах невизначеного або динамічного середовища, де класичні методи не завжди забезпечують достатню адаптивність. Така структура організації алгоритмів дозволяє розглядати задачу супроводу рухомої цілі як поєднання глобального планування, локального перепланування та безпосереднього керування польотом, що є характерним для сучасних автоматизованих систем БПЛА [12].

Висновки до розділу 1

У першому розділі було проведено аналіз сучасних технологій переслідування рухомих цілей безпілотними літальними апаратами, розглянуто класифікацію алгоритмів супроводу, методи автоматичного стеження, математичні моделі для навігації та керування, а також основні підходи до оптимізації траєкторії руху БПЛА. Встановлено, що задача автоматизованого супроводу рухомої цілі є комплексною і поєднує засоби комп'ютерного зору, навігації, прогнозування та формування керуючих сигналів у реальному часі.

У результаті аналізу алгоритмів переслідування визначено, що найбільш доцільними для задач супроводу рухомої цілі є візуальні та комбіновані підходи, оскільки вони дають змогу здійснювати безперервне стеження за об'єктом, визначати його положення в кадрі та використовувати ці дані для корекції траєкторії польоту. Візуальні алгоритми забезпечують автономність роботи системи, а комбіновані методи підвищують її стійкість і точність за рахунок поєднання даних комп'ютерного зору та навігаційної інформації.

Під час аналізу математичних моделей для навігації та керування встановлено, що для опису процесу супроводу доцільно використовувати кінематичні моделі руху БПЛА і цілі, моделі відносного положення об'єкта та апарата, а також елементи прогнозування для врахування зміни положення цілі в часі. Такі моделі створюють основу для формалізації задачі переслідування та подальшого синтезу алгоритму керування.

Розгляд методів оптимізації показав, що в задачах стеження і навігації найбільш ефективними є підходи, які поєднують глобальне планування, локальну корекцію та прогнозне формування керуючих сигналів. При цьому для практичної реалізації автоматизованого супроводу важливими є не лише точність і швидкодія, а й можливість роботи в реальному часі, простота інтеграції з програмними засобами та сумісність із системою керування польотом.

На основі проведеного аналізу для подальшої реалізації в роботі обрано **метод візуального супроводу рухомої цілі** з використанням алгоритмів комп'ютерного зору. Як основний інструмент обробки зображення

використовується **OpenCV Tracker**, який дозволяє виділяти ціль у кадрі, відстежувати її положення та визначати зміщення відносно центра зображення. Для реалізації алгоритмів керування та моделювання польоту використовується середовище **Python**, а для взаємодії з системою керування БПЛА та перевірки роботи алгоритму — **QGroundControl** і засоби обміну телеметричною інформацією. Такий підхід дає можливість поєднати візуальне виявлення і супровід цілі з формуванням навігаційних команд для безпілотного літального апарата.

Отже, за результатами розділу 1 визначено, що для задачі автоматизованого супроводу рухомої цілі доцільно використовувати візуальний метод стеження з елементами навігаційної корекції, реалізований у програмному середовищі Python із застосуванням OpenCV та інтеграцією з системою керування польотом БПЛА. Отримані результати є теоретичною основою для побудови математичної моделі переслідування, вибору структури системи керування та подальшого моделювання роботи автоматизованої системи супроводу в наступних розділах роботи.

РОЗДІЛ 2
ПОБУДОВА МАТЕМАТИЧНОЇ МОДЕЛІ ПЕРЕСЛІДУВАННЯ ТА ФОРМАЛІЗАЦІЯ ЗАДАЧІ

2.1 Формалізація задачі переслідування рухомої цілі

У даному підрозділі виконується формалізація задачі переслідування рухомої цілі безпілотним літальним апаратом у межах автоматизованої системи оптичного супроводу та наведення. Необхідність формалізації зумовлена тим, що задача переслідування є багатокомпонентною і поєднує процеси спостереження, оцінювання координат цілі, аналізу поточного стану БПЛА, формування навігаційної команди та корекції траєкторії польоту в режимі реального часу. Для подальшої побудови математичної моделі потрібно чітко визначити, які величини є вхідними, які параметри описують стан системи, які дії виконує система керування та яким має бути кінцевий результат супроводу.

У практичному змісті задача переслідування рухомої цілі полягає в тому, щоб безпілотний літальний апарат, використовуючи дані бортової камери, міг виявити об’єкт, утримувати його в полі зору, визначати його положення відносно власного місцеположення та формувати такі команди керування, які забезпечують стійке наведення на ціль. При цьому важливо, щоб система працювала не лише в ідеальних умовах, а й за наявності шумів відеопотоку, нестабільного освітлення, змін масштабу об’єкта, короткочасної втрати супроводу та похибок під час переходу від координат у кадрі до команд автопілота. Саме тому формалізація задачі повинна охоплювати не лише геометричний бік визначення положення цілі, а й логіку роботи всієї системи в цілому [21].

У межах даної роботи система переслідування розглядається як сукупність кількох функціонально пов’язаних блоків. Першим блоком є камера та модуль комп’ютерного зору, який отримує відеопотік і забезпечує візуальний супровід об’єкта. Другим блоком є підсистема оцінювання поточного стану БПЛА, яка використовує телеметричні дані автопілота, зокрема координати, висоту, швидкість, курс, крен і тангаж. Третім блоком є модуль математичного перетворення координат, у якому виконується перехід від координат об’єкта в кадрі до локальних зміщень у системі NED та далі до глобальних географічних координат. Четвертий блок формує траєкторну точку наведення і корекцію висоти.

П’ятий блок реалізує передавання команд до автопілота через MAVLink у режимі GUIDED. Така структура дає змогу розглядати задачу переслідування не як окремий алгоритм трекінгу, а як замкнений контур взаємодії між спостереженням, оцінюванням та керуванням (Рис.2.1).

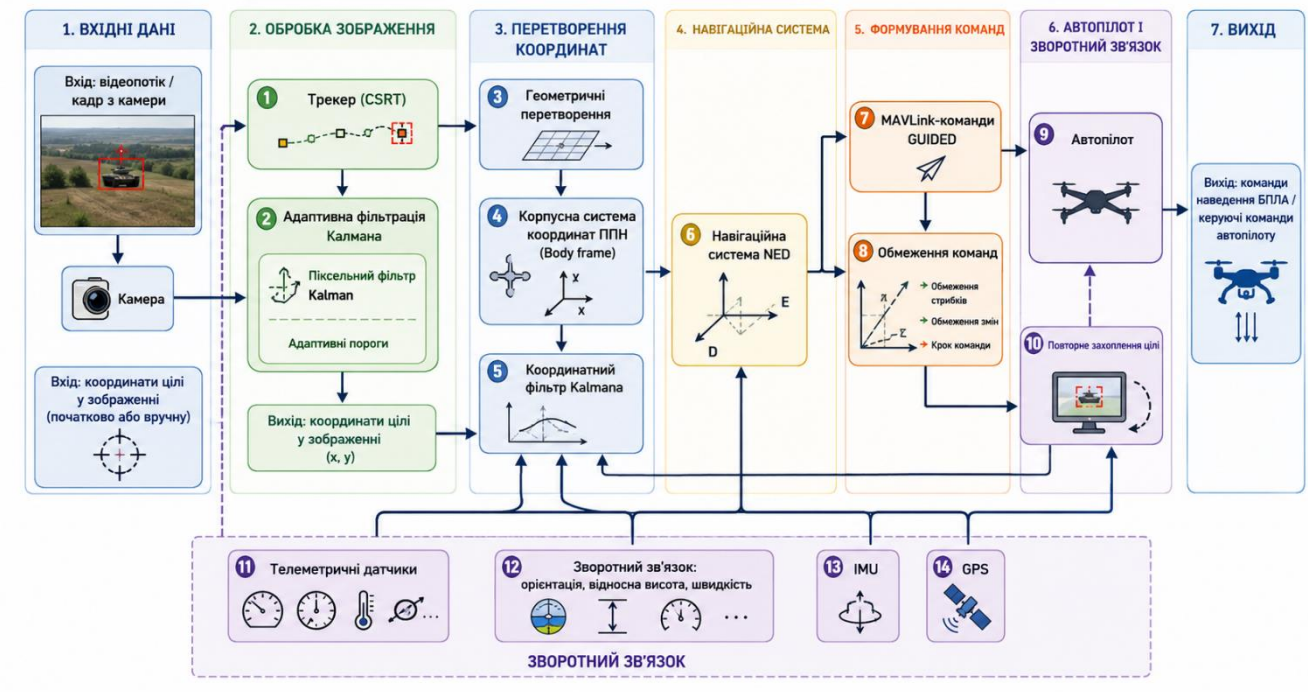


Рис.2.1 Структурно-функціональна схема оптичного супроводу та наведення БПЛА.

З позиції теорії керування задачу переслідування можна розглядати як задачу формування такої послідовності керуючих дій, за якої відносне положення цілі та БПЛА змінюється відповідно до заданої мети супроводу. У найпростішому випадку така мета полягає в утриманні об’єкта поблизу центра кадру. Проте в реальній системі цього недостатньо, оскільки положення цілі в центрі кадру ще не гарантує коректного просторового наведення. Необхідно, щоб на основі зображення система могла оцінити фактичне положення об’єкта відносно БПЛА, врахувати орієнтацію літального апарата, його висоту та навігаційний стан, а потім сформувати таку цільову точку, яка буде фізично реалізованою для автопілота. Отже, задача переслідування характеризується взаємодією інформаційної, геометричної та керуючої складових.

Для формалізації задачі важливо визначити вхідні дані системи. До них належать відеопотік з бортової камери, параметри камери, телеметричні дані БПЛА, а також інформація про початкове положення цілі, яка може задаватися оператором при виборі області супроводу. Відеопотік використовується для визначення положення об’єкта на зображенні. Телеметрія дозволяє оцінити поточний стан апарата, необхідний для просторового перетворення координат. Параметри камери забезпечують зв’язок між піксельними координатами та геометричними характеристиками напрямку на ціль. У поєднанні ці дані формують інформаційну основу для подальшого керування.

Вихідними даними системи є траєкторна точка наведення та команда для автопілота. Формально система на кожному кроці повинна сформувати таку команду, яка враховує поточне положення цілі, узгоджується з поточним станом БПЛА і не суперечить обмеженням системи керування. Це означає, що на виході системи не повинні виникати різкі стрибки координат або фізично нереалізовані вимоги до руху. Навпаки, команда наведення має змінюватися плавно, бути стійкою до перешкод і забезпечувати збереження цілі в полі зору. Отже, в задачі переслідування важливим є не тільки точне визначення положення об’єкта, а й коректне формування керуючого сигналу.

Особливістю задачі є те, що ціль і БПЛА одночасно перебувають у русі. Це означає, що система працює не з фіксованим об’єктом, а з об’єктом, стан якого змінюється в часі. Відповідно, кожне нове спостереження за ціллю не може розглядатися ізольовано від попередніх. Для стійкого супроводу необхідно враховувати зміну координат об’єкта між сусідніми кадрами, швидкість цього зміщення та характер руху цілі. У разі різкої зміни траєкторії або короткочасної втрати супроводу система повинна не втратити працездатності, а відновити коректне визначення положення об’єкта та продовжити наведення. Саме тому в задачу переслідування вводиться етап оцінювання стану і згладжування вимірювань, а також механізми повторного захоплення цілі.

У формалізованому вигляді задача переслідування включає кілька послідовних етапів. На першому етапі ціль вибирається або виявляється у

відеопотоці, після чого запускається її візуальний супровід. На другому етапі координати об’єкта в кадрі підлягають згладжуванню та перевірці на коректність. На третьому етапі виконується просторове перетворення координат, у результаті якого визначається положення цілі в локальній системі координат. На четвертому етапі формується траєкторна точка наведення, яка може бути додатково згладжена або обмежена за швидкістю зміни. На останньому етапі сформована команда передається автопілоту. Така послідовність відображає реальну логіку функціонування системи та є основою для побудови її математичної моделі.

Для задачі супроводу важливим є також визначення критерію якості. Якісний супровід означає, що ціль утримується в межах робочої області кадру, команди наведення змінюються плавно, а БПЛА рухається таким чином, щоб не втрачати об’єкт і не створювати нестійкого режиму польоту. Отже, при формалізації задачі доцільно враховувати кілька вимог одночасно: точність візуального супроводу, плавність формування траєкторної точки, стійкість до шуму та перешкод, узгодженість команд з можливостями автопілота і збереження фізичної реалізованості руху. У цьому полягає головна відмінність задачі автоматизованого переслідування від простої задачі розпізнавання або локалізації об’єкта на зображенні.

Окремо слід зазначити, що система працює в дискретному часі, тобто процес спостереження, оцінювання і керування відбувається покадрово. Кожний новий кадр дає чергове вимірювання положення цілі, після чого формується оновлена оцінка її стану та нова команда наведення. Такий підхід відповідає реальній роботі програмного модуля та дозволяє пов’язати задачу супроводу з дискретною моделлю стану, яка буде розглянута в наступному підрозділі. Саме покадровий характер обробки інформації зумовлює необхідність використання фільтрації, оскільки різниця між сусідніми вимірюваннями може бути викликана не лише рухом цілі, а й випадковими похибками відеоспостереження.

Таким чином, формалізація задачі переслідування рухомої цілі у даній роботі полягає у визначенні структури системи, встановленні складу її вхідних і вихідних даних, описі логіки переходу від відеоспостереження до навігаційної

команди та визначенні вимог до якості супроводу. У межах цієї постановки задача розглядається як замкнений контур, у якому результати спостереження за ціллю безпосередньо впливають на траєкторію польоту БПЛА. Така формалізація створює основу для подальшого опису математичної моделі руху дрона і цілі, що буде наведено в наступному підрозділі.

2.2 Математична модель руху дрона і цілі

Для побудови автоматизованої системи супроводу рухомої цілі необхідно задати математичну модель руху безпілотного літального апарата та об'єкта спостереження. Така модель повинна відображати зміну положення цілі у просторі, рух БПЛА відносно цілі, а також зв'язок між візуальними вимірюваннями та навігаційними параметрами системи. У межах даної роботи доцільно розглядати модель на двох рівнях: кінематичному та динамічному. Кінематичний рівень описує зміну координат і швидкостей без урахування причин їх виникнення, а динамічний — враховує вплив керуючих дій, інерційних властивостей системи та обмежень руху.

У загальному випадку рух цілі та БПЛА доцільно описувати в локальній системі координат NED, де вісь Nспрямована на північ, вісь E— на схід, а вісь D— вниз. Такий вибір системи координат є зручним, оскільки телеметричні параметри БПЛА, орієнтація літального апарата та обчислені зміщення цілі можуть бути узгоджені в межах єдиного просторового опису. Положення цілі в системі NED задається вектором

Далі задача переслідування формалізується через опис положення цілі у площині зображення. Нехай у поточний момент часу координати центра цілі в кадрі задаються як u та v. Якщо розмір кадру становить $W \times H$, тоді центр кадру визначається співвідношеннями (Формула 2.1.):

$$c_x = \frac{W}{2}, c_y = \frac{H}{2}, \tag{2.1}$$

де W - ширина зображення в пікселях; H - висота зображення в пікселях;

На основі цих величин визначається відхилення цілі від центра кадру, яке є початковою характеристикою похибки супроводу. Фізично це означає, що система оцінює, наскільки змістився об’єкт відносно оптичної осі камери, і саме це зміщення надалі використовується як основа для формування керуючих дій. Якщо ціль утримується поблизу центра кадру, то режим супроводу можна вважати стабільним. Якщо ж відхилення збільшується, система повинна змінити траєкторію польоту БПЛА таким чином, щоб знову наблизити об’єкт до центральної області зображення [21].

Для переходу від координат у пікселях до просторової постановки задачі використовується модель камери. Застосовується модель тонкої лінзи, за якою фокусні відстані у пікселях визначаються через розміри кадру та кути поля зору камери (Формула 2.2):

$$f_x = \frac{\left(\frac{W}{2}\right)}{tan\left(\frac{HFOV}{2}\right)}, \quad f_y = \frac{\left(\frac{H}{2}\right)}{tan\left(\frac{VFOV}{2}\right)}, \tag{2.2}$$

де f_x, f_y - фокусні відстані камери в пікселях відповідно по осях x та y;*HFOV, VFOV*- горизонтальний та вертикальний кут огляду камери;

Після цього нормовані координати зображення записуються як (Формула 2.3):

$$x_{img} = \frac{u - c_x}{f_x}, \quad y_{img} = \frac{v - c_y}{f_y}, \tag{2.3}$$

де c_x, c_y - координати головної точки (центра зображення) в пікселях; x_{img} , y_{img} - нормовані координати точки цілі у площині зображення;

Після цього формується v_cam вектор візування у системі камери (Формула 2.4):

$$v\_cam = \left[x_{img}, y_{img}, 1\right]^T, \tag{2.4}$$

який нормується до одиничної довжини. Саме такий підхід використано у функції pixel_to_ned_offset(...)(Додаток А).

Цей етап є принципово важливим, оскільки саме тут відбувається перехід від площини зображення до просторового опису напряму на ціль. Іншими словами, система перестає працювати лише з піксельним зміщенням і переходить до геометричного опису напряму на об’єкт відносно камери [22].

Оскільки камера встановлена на БПЛА з певною орієнтацією, отриманий вектор візування необхідно перевести спочатку в корпусну систему координат, а потім у локальну систему NED. Цей перехід виконується послідовно: спочатку застосовується матриця вирівнювання осей, потім ураховуються монтажні кути камери, а далі — орієнтація самого БПЛА, яка визначається креном, тангажем і курсом. У результаті формується вектор візування в системі NED [23].

У реалізації перехід від системи камери до корпусної системи виконується у два етапи. Спочатку застосовується R_{align} матриця вирівнювання осей (Формула 2.5):

$$R_{align} = [001][100][010] \tag{2.5}$$

і формується проміжний вектор (Формула 2.6):

$$v_{body0} = R_{align} \cdot v_{cam}. \tag{2.6}$$

Далі враховується R_{mount} монтаж камери відносно корпусу за допомогою матриці (Формула 2.7):

$$R_{mount} = Rz(\psi m) \cdot Ry(\theta m) \cdot Rx(\varphi m), \tag{2.7}$$

де φm, θm, ψm — монтажні кути roll, pitch, yaw камери (Формула 2.8):

$$v_{body} = R_{mount} \cdot v_{body0}. \tag{2.8}$$

Після цього вектор переводиться у систему NED з урахуванням орієнтації БПЛА (Формула 2.9, Формула 2.10):

$$R_{body2ned} = Rz(yaw) \cdot Ry(pitch) \cdot Rx(roll), \tag{2.9}$$

$$v_{ned} = R_{body2ned} \cdot v_{body}. \quad (2.10)$$

Отримані величини мають безпосередній фізичний зміст: вони показують, на яку відстань у локальній системі координат зміщена ціль відносно БПЛА (Рис. 2.2. та Рис .2.3.) [24].

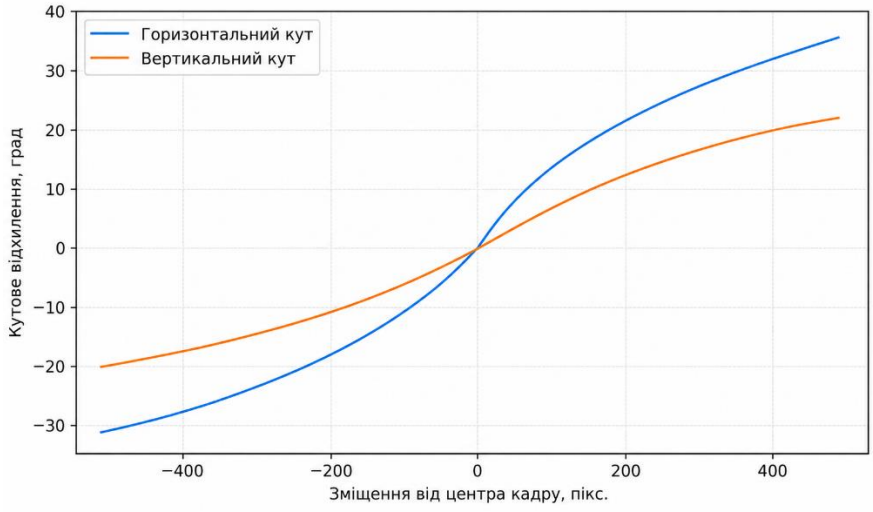


Рис. 2.2 Залежність кутового відхилення від зміщення цілі у пікселях відносно центра кадру.

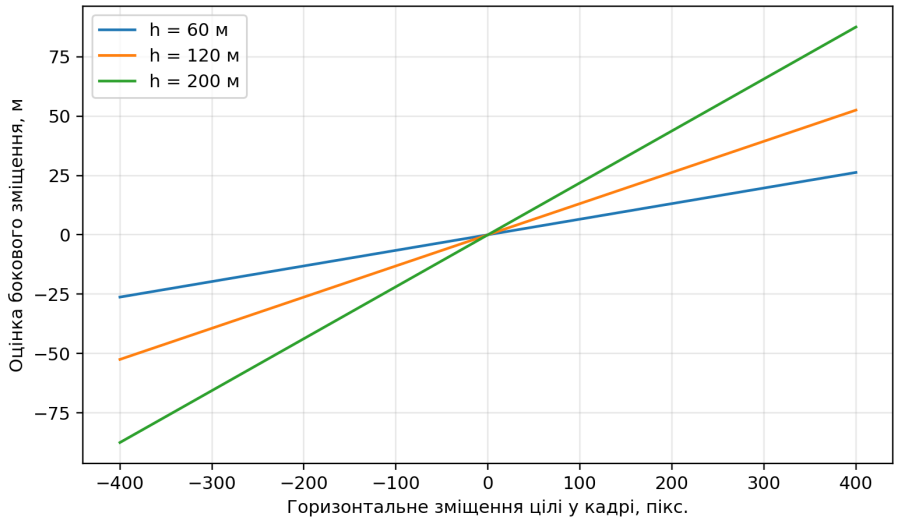


Рис. 2.3 Залежність кутового відхилення від зміщення цілі у пікселях відносно центра кадру.

Після визначення локальних зміщень у NED виконується перехід до географічних координат. Для цього використовується сферичне наближення Землі з радіусом R ≈ 6378137 м, а зміни широти й довготи визначаються за формулами.

Нехай відома відносна висота БПЛА над землею h (AGL). Припускаючи, що ціль лежить на площині землі (z=0 у NED), точку перетину променя v_{ned} з цією площиною знаходимо за параметром t. Якщо вісь Down у NED співнапрямлена з гравітацією, то компонента v_ned,z повинна бути додатною (Формула 2.11):

$$t = \frac{h}{v_{ned,z}}, \quad \text{за умови } v_{ned,z} > 0. \tag{2.11}$$

Тоді зміщення у метрах у NED (NED=North East Down) (Формула 2.12):

$$N = t \cdot v_{ned,x}, \quad E = t \cdot v_{ned,y}, \tag{2.12}$$

де t - параметр перетину променя візування з площиною землі; h - відносна висота БПЛА над поверхнею землі; $v_{ned,z}$ - складова вектора напрямку на ціль по вертикальній осі z у системі координат NED; $v_{ned,y}$ - складова вектора напрямку на ціль по вертикальній осі y у системі координат NED; $v_{ned,x}$ - складова вектора напрямку на ціль по вертикальній осі x у системі координат NED;

Якщо прямий перетин недоступний, у коді застосовано резервний алгоритм: визначається дальність до цілі за кутом вниз, далі формується проєкція у корпусній системі та виконується поворот на курс БПЛА. Після цього дальність додатково обмежується в інтервалі між мінімальною та максимальною допустимими відстанями. Це дозволяє уникати некоректних траєкторних точок.

Для переходу від локальних зміщень до географічних координат використовується наближення сферичної Землі з радіусом R ≈ 6378137 м. Тоді зміни широти та довготи визначаються як (Формула 2.13, Формула 2.14):

$$d_{lat} = \frac{N}{R}, \tag{2.13}$$

$$d_{lon} = \frac{E}{\left(R \cdot cos(lat0)\right)}, \quad (2.14)$$

де d_{lat} - зміна широти в радіанах; d_{lon} - зміна довготи в радіанах; $cos(lat0)$ - поправка, яка враховує зміну довжини одного градуса довготи залежно від широти; $lat0$ - початкова широта БПЛА або точки відліку;

Підсумкові координати цілі (Формула 2.15, Формула 2.16):

$$lat = lat0 + degrees(d_{lat}), \quad (2.15)$$

$$lon = lon0 + degrees(d_{lon}), \quad (2.16)$$

де lat - підсумкова широта цілі; lon - підсумкова довгота цілі.

Саме ці обчислення реалізовані у функції ned_offsets_to_global(...).

Таким чином, задача переслідування набуває вигляду задачі послідовного оновлення глобальної траєкторної точки наведення, яка формується на основі нового спостереження за об’єктом і передається автопілоту [25].

Важливою складовою формалізації є врахування шумів і нестабільності вимірювань. Координати цілі, отримані від трекера, не є абсолютно точними: вони можуть змінюватися стрибкоподібно, залежати від якості зображення, масштабу об’єкта, освітлення та часткових перекриттів. Саме тому в системі вводиться етап оцінювання стану, який базується на використанні фільтра Калмана (Формула 2.17).

$$[x, y, vx, vy]^T, \quad (2.17)$$

де враховуються не лише координати цілі, а й швидкість їх зміни. Це дозволяє перейти від безпосередньо виміряних координат до більш стабільної оцінки положення об’єкта. Такий підхід є доцільним, оскільки в задачі супроводу важлива не тільки точність вимірювання, а й плавність зміни траєкторної точки та керуючих дій [26].

Перший фільтр Калмана використовується для згладжування координат цілі у площині зображення. Після оновлення трекера формується вимірювання z_meas, далі обчислюється прогноз, і якщо відстань між прогнозом та виміряною координатою перевищує адаптивний поріг, стан фільтра скидається. Якщо ж розбіжність допустима, виконується стандартна корекція. Такий підхід пригнічує різкі стрибки координат та покращує стабільність положення маркера супроводу.

Другий фільтр Калмана використовується для згладжування координат цільової точки в системі NED. Після обчислення локальних зміщень формується вектор z_ne, до якого застосовується аналогічна логіка: прогноз, перевірка похибки, за потреби скидання стану або корекція. Після цього додатково застосовується функція обмеження швидкості зміни вектора limit_vector_step(...), яка не дозволяє цільовій точці змінюватися надто різко між кадрами. Саме це забезпечує плавніше наведення БПЛА.

Шум вимірювання та пороги спрацювання не є сталими. Вони адаптуються залежно від якості зображення, що визначається за дисперсією оператора Лапласа на області цілі. Якщо зображення більш чітке, шум вимірювання зменшується, а довіра до вимірювання зростає. Якщо ж якість падає, фільтр автоматично стає менш чутливим до нестабільних значень. Це робить систему адаптивною до умов спостереження.

Керуючі впливи формуються за нормованими помилками a_x та a_y, що відповідають зміщенню цілі відносно центра кадру. На основі відфільтрованих координат у NED обчислюється траєкторна точка, а висота команди змінюється відносно зафіксованої базової висоти при захопленні цілі. Висота обмежується допустимими межами. Після цього координати переводяться у широту та довготу, і команда надсилається автопілоту з фіксованим часовим інтервалом у режимі GUIDED.

У межах побудованої математичної моделі доцільно виділити дві групи змінних: вимірювані та керовані. До вимірюваних належать координати цілі в кадрі, телеметричний стан БПЛА, параметри орієнтації та висота польоту. До керованих належать траєкторна точка наведення, бажана висота, а також команда

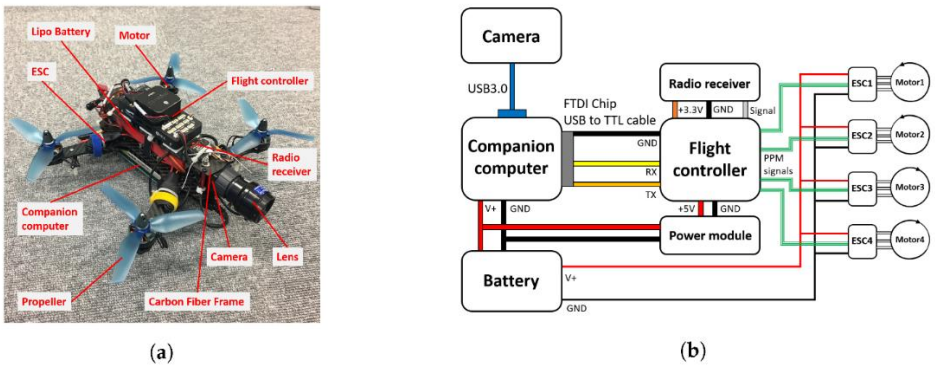
зміни просторового положення апарата. Такий поділ дозволяє чітко відокремити параметри, що надходять із сенсорів, від параметрів, які формуються системою керування.

Отже, математична модель руху дрона і цілі в даній роботі поєднує кінематичний опис просторового положення БПЛА та об’єкта, геометричне перетворення координат зображення в локальну систему координат, а також динамічний опис зміни стану системи під впливом керуючих сигналів і фільтрації. Кінематична частина моделі дозволяє визначати положення цілі та БПЛА у просторі, а динамічна — враховувати реальний характер руху апарата, згладжування вимірювань та обмеження на формування команд.

2.3 Обмеження, початкові та граничні умови функціонування автоматизованої ситеми супроводу рухомої цілі

У цьому підрозділі задаються обмеження, початкові та граничні умови, в межах яких працює автоматизована система супроводу рухомої цілі безпілотним літальним апаратом. Їх урахування є необхідним для побудови реалістичної математичної моделі, оскільки саме вони визначають допустимі режими польоту, межі зміни керуючих параметрів та умови стійкої роботи алгоритму наведення.

У роботі розглядається безпілотний літальний апарат масою 2,5 кг. Така маса є характерною для малих БПЛА спостережного типу і визначає його інерційні властивості, тобто здатність реагувати на зміну керуючих сигналів. Для цього апарата максимальна швидкість польоту становить 18 м/с, а крейсерська — 12 м/с. Отже, у процесі супроводу доцільно орієнтуватися переважно на крейсерський режим, а максимальну швидкість використовувати лише за необхідності швидкого зближення з ціллю або корекції траєкторії Рис.2.4 [12, 27].



(a) (b)

Рис.2.4 БПЛА: (a) платформа; (b) схема підключення [28].

Обмеження за прискоренням також мають суттєве значення. Максимальне горизонтальне прискорення БПЛА становить 5 м/с², а вертикальне — 4 м/с². Це означає, що система не повинна формувати такі траєкторні точки, для досягнення яких апарату потрібно було б перевищити власні динамічні можливості. Інакше команди наведення будуть математично коректними, але фізично нереалізованими. Саме тому зміна траєкторної точки має бути плавною та узгодженою з динамікою руху апарата [29].

До важливих обмежень належать також кути орієнтації БПЛА. Кути крену і тангажу обмежені значеннями ±30°, а максимальна кутова швидкість становить 200°/с. Це визначає допустиму інтенсивність повороту та зміни просторового положення апарата. У задачі супроводу такі обмеження є принциповими, оскільки саме вони визначають, наскільки швидко БПЛА може відреагувати на зміну положення цілі в кадрі без втрати стійкості польоту.

Робоча висота польоту БПЛА знаходиться в межах 30–150 м. Нижня межа пов’язана з вимогами безпеки та необхідністю мати достатній простір для маневрування, а верхня — з точністю візуального супроводу. При збільшенні висоти одна й та сама похибка визначення координат цілі в кадрі відповідає більшому зміщенню в метрах, що погіршує точність наведення. Отже, вибраний діапазон висот є компромісом між безпекою польоту, якістю спостереження та точністю формування траєкторної точки.

Під час польоту на БПЛА діють аеродинамічні збурення, зокрема вітер до 6 м/с з випадковими поривами. Для легкого апарата такі впливи є відчутними,

оскільки можуть змінювати фактичну траєкторію руху, викликати коливання та погіршувати стабільність супроводу. Тому в моделі необхідно враховувати, що реальний рух БПЛА може відхилятися від розрахункового, а алгоритм наведення повинен бути стійким до таких зовнішніх впливів.

Початкові умови визначають стан системи в момент початку супроводу. До них належать початкові координати БПЛА, висота польоту, швидкість, орієнтація апарата та початкове положення цілі в кадрі. Саме в цей момент ініціалізується трекер, задається базова висота та формується початковий стан алгоритму супроводу.

Граничні умови визначають допустимі межі функціонування системи під час переслідування. До них належить обмеження швидкості, прискорення, кутів орієнтації, кутових швидкостей і висоти встановленими межами, а також збереження цілі в полі зору камери. У разі порушення цих умов система повинна обмежувати зміну траєкторії або коригувати режим роботи.

2.4 Вибір методу оптимізації задачі автоматизованого супроводу рухомої цілі безпілотним літальним апаратом

У цьому підрозділі виконується вибір методу оптимізації, який доцільно використати в задачі автоматизованого супроводу рухомої цілі безпілотним літальним апаратом. Необхідність такого вибору пов’язана з тим, що в процесі переслідування система повинна не лише визначати положення об’єкта, а й формувати такі керуючі сигнали, які забезпечують стійкий, плавний і фізично реалізований рух БПЛА. У загальному випадку задача супроводу може розв’язуватися різними методами, зокрема на основі прямого наведення, евристичних алгоритмів, прогнозного керування, генетичних алгоритмів, методів динамічного програмування або пошуку шляху. Однак вибір конкретного підходу повинен узгоджуватися з особливостями побудованої системи, доступними вхідними даними, обчислювальними ресурсами та вимогами до роботи в реальному часі.

Для даної роботи основною особливістю системи є те, що інформація про ціль надходить із відеопотоку бортової камери, а координати об’єкта формуються в результаті візуального супроводу та подальшого геометричного перетворення. Це означає, що вхідні дані мають дискретний характер, залежать від якості зображення і можуть містити випадкові похибки, короткочасні стрибки або затримки. За таких умов використання складних глобальних алгоритмів оптимізації, які потребують великих обчислювальних витрат або повної апріорної інформації про траєкторію цілі, є малоефективним. У системі супроводу, що працює в реальному часі, пріоритетним стає не глобальний пошук ідеальної траєкторії на всьому часовому інтервалі, а швидке та стійке формування поточної траєкторної точки наведення.

З урахуванням цього для задачі автоматизованого супроводу доцільно використовувати підхід, який поєднує **оцінювання стану цілі** та **локальну оптимізацію траєкторної точки наведення**. У межах реалізованої системи таку функцію виконує адаптивна фільтрація Калмана в поєднанні з обмеженням швидкості зміни цільової точки. По суті, у даній роботі оптимізація не зводиться до класичного пошуку найкоротшого маршруту, а полягає у виборі такого поточного положення траєкторної точки, яке забезпечує мінімізацію похибки супроводу, згладжування випадкових коливань координат та узгодження команди з динамічними можливостями БПЛА.

Вибір саме такого підходу пояснюється кількома причинами. По-перше, у задачі супроводу рухомої цілі система повинна працювати в режимі реального часу, тобто з мінімальною затримкою між отриманням нового кадру і формуванням команди наведення. Методи глобальної оптимізації, наприклад генетичні алгоритми або динамічне програмування, можуть бути ефективними для планування маршруту, однак вони потребують більшого часу обчислення і не завжди придатні для покадрового оновлення траєкторії. По-друге, положення цілі змінюється безперервно, а отже, навіть оптимальна траєкторія, обчислена для попереднього стану системи, швидко втрачає актуальність. По-третє, через шум відеоданих і нестабільність трекінгу система повинна мати вбудований механізм

згладжування вимірювань, а не лише механізм геометричного перетворення координат.

Адаптивна фільтрація Калмана є доцільною саме тому, що вона дозволяє оцінювати стан цілі не тільки за поточним вимірюванням, а й з урахуванням попередньої історії зміни координат. У результаті зменшується вплив шуму, пригнічуються різкі стрибки положення об’єкта, а траєкторна точка формується більш плавно. Додаткове обмеження швидкості зміни вектора наведення забезпечує узгодження між новою оцінкою положення цілі та фізичними можливостями БПЛА. Таким чином, обраний метод виконує одразу дві функції: підвищує точність оцінювання та забезпечує локальну оптимізацію процесу наведення.

Сутність обраного підходу полягає в тому, що система не намагається одразу вивести БПЛА в точку миттєво оціненого положення цілі. Натомість формується згладжена і обмежена траєкторна точка, яка змінюється поступово і враховує попередній стан системи. Це особливо важливо для малих БПЛА, оскільки різка зміна команд призводить до нестійкого руху, перевищення допустимих кутів орієнтації, зростання похибки та погіршення якості відеоспостереження. Отже, критерієм оптимальності в даній роботі є не лише мінімізація просторової похибки між БПЛА і ціллю, а й забезпечення плавності, стійкості та фізичної реалізованості траєкторії наведення.

На відміну від методів прямого наведення, які реагують лише на поточне положення об’єкта, обраний підхід має прогнозний характер, оскільки оцінювання стану цілі виконується з урахуванням її швидкості зміни. Це підвищує стійкість супроводу в умовах тимчасових збоїв трекінгу, короткочасної втрати цілі або дії зовнішніх збурень. У поєднанні з перевіркою стрибків координат і механізмом повторного захоплення такий метод дозволяє зменшити вплив нестабільних вимірювань на контур керування.

Висновки до розділу 2

У другому розділі було виконано формалізацію задачі переслідування рухомої цілі безпілотним літальним апаратом та побудовано математичну основу автоматизованої системи супроводу. Визначено структуру системи, її вхідні та вихідні дані, а також встановлено зв’язок між візуальним спостереженням за ціллю, перетворенням координат і формуванням команд наведення для БПЛА.

У розділі розглянуто математичну модель руху дрона і цілі, яка дає змогу описати зміну їхнього положення в часі та врахувати особливості процесу супроводу. Також задано основні обмеження системи, зокрема допустимі швидкість, прискорення, кути орієнтації, висоту польоту та вплив зовнішніх збурень. Це дозволило узгодити математичний опис із реальними фізичними можливостями БПЛА.

На основі проведеного аналізу обґрунтовано доцільність використання методу, що забезпечує стійке формування траєкторної точки наведення в реальному часі з урахуванням шумів вимірювання та динамічних можливостей апарата. Отримані результати є основою для подальшої програмної реалізації системи супроводу, її моделювання та оцінювання ефективності роботи.

РОЗДІЛ 3
РОЗРОБКА ТА МОДЕЛЮВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ З ОПТИМІЗАЦІЄЮ ТРАЄКТОРІЇ

3.1 Структура автоматизованої системи керування дроном

Автоматизована система керування дроном для супроводу рухомої цілі побудована за принципом замкненого контуру керування, у якому команди формуються на основі поточного стану системи, передаються до автопілота, реалізуються у вигляді руху БПЛА, а потім коригуються за рахунок зворотного зв’язку (Рис 3.1).

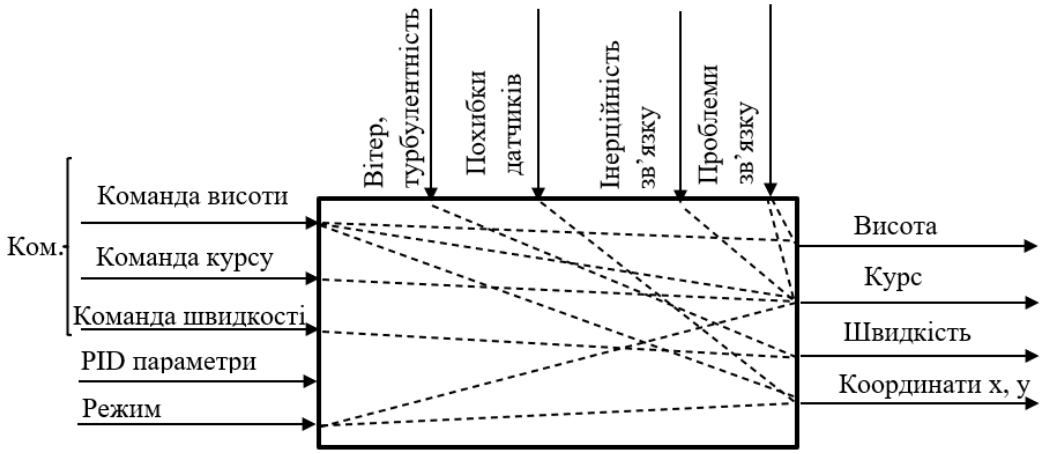


Рис.3.1 Параметрична схема ОК із вказаними фізичних величин

Такий підхід дає змогу враховувати не лише положення цілі, а й реальний стан дрона, вплив зовнішніх збурень та точність виконання команд (Рис.3.2.).

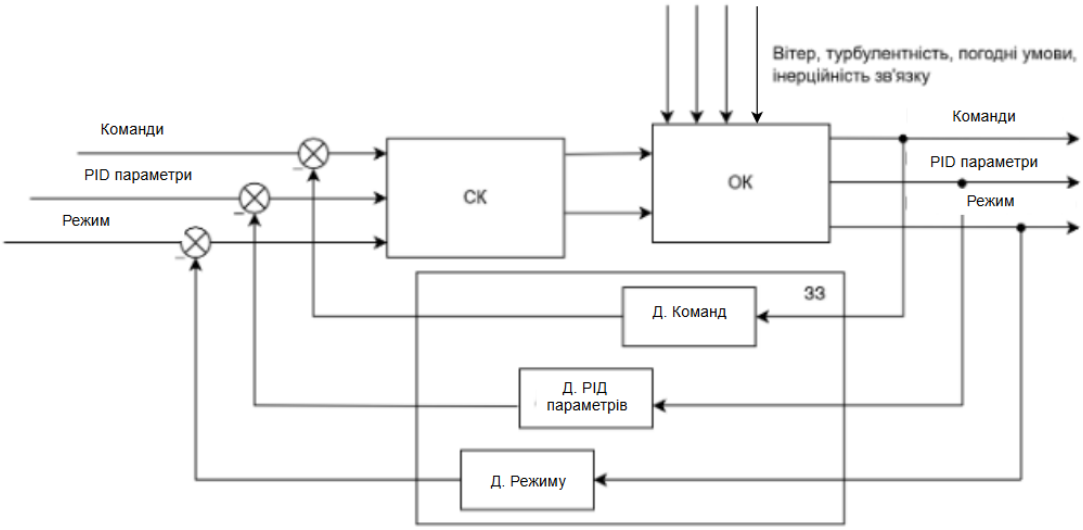


Рис.3.2 Структурна схема автоматизованої системи взаємодії робота-БПЛА з середовщем у якому він перебуває

На поданій схемі основними блоками є СК — система керування та ОК — об’єкт керування. Блок системи керування виконує аналіз вхідної інформації, формує керуючі сигнали та передає їх до об’єкта керування. Блок об’єкта керування відповідає самому безпілотному літальному апарату, який реалізує ці команди у вигляді зміни курсу, висоти, швидкості та просторового положення. Крім основного контуру, схема враховує вплив зовнішніх факторів, таких як вітер, турбулентність, погодні умови та інерційність зв’язку, які можуть викликати відхилення від розрахованої траєкторії.

На вході системи використовуються команди**,** PID-параметри та режим роботи. Команди задають напрям руху та бажану траєкторію, PID-параметри визначають характер реакції системи на похибку, а режим роботи встановлює загальну логіку функціонування автопілота. Через зворотні зв’язки система отримує інформацію про фактичне виконання команд, поточний режим і стан керування, що дає можливість постійно коригувати роботу контуру.

У програмній реалізації роль системи керування виконує модуль, написаний мовою Python. Він об’єднує обробку відеопотоку, роботу трекера, фільтрацію координат і передавання команд автопілоту через MAVLink. У коді задаються параметри шумів вимірювання, пороги помилок, характеристики фільтра Калмана, параметри камери, монтажні кути та інтервали надсилання команд. Усе це формує внутрішнє налаштування системи керування та забезпечує її адаптацію до умов супроводу.

Початковою ланкою системи є модуль комп’ютерного зору, який працює з відеопотоком із камери. Після вибору цілі в кадрі запускається трекер CSRT, що визначає положення об’єкта в послідовності кадрів. Якщо виникає втрата супроводу або різкий стрибок координат, система намагається повторно знайти об’єкт за шаблоном. Це дозволяє підтримувати більш стійкий режим спостереження навіть у випадку нестабільного зображення.

Для зменшення впливу шумів у системі використовується фільтрація Калмана. Один фільтр застосовується до координат цілі в кадрі, інший — до координат траєкторної точки в системі NED. Це дає змогу згладжувати випадкові

відхилення, уникати різких стрибків і формувати більш плавні керуючі дії. Додатково в коді реалізовано адаптивне налаштування параметрів фільтра залежно від якості зображення, що підвищує стійкість системи в реальних умовах.

Важливою частиною структури є блок приймання телеметрії. Через MAVLink зчитуються повідомлення про режим роботи, координати БПЛА, висоту, швидкість, крен, тангаж і курс. Ці дані використовуються для оцінювання поточного стану дрона та подальшого перетворення координат цілі з площини зображення у просторові координати. Саме завдяки цьому система враховує не лише візуальне положення цілі, а і фактичний стан літального апарата.

Після цього працює модуль математичних перетворень, який переводить координати цілі з кадру в локальну систему координат NED, а потім у глобальні географічні координати. У коді для цього реалізовані функції просторових поворотів, перетворення піксельних координат у локальні зміщення та обчислення глобальної цільової точки. Саме цей модуль забезпечує зв’язок між системою комп’ютерного зору та навігаційним контуром керування.

На основі відфільтрованих координат цілі система формує керуючі сигнали. Для цього обчислюються нормовані похибки зміщення об’єкта відносно центра кадру, після чого задається нова траєкторна точка наведення та коригується висота. Щоб уникнути нестабільного руху, у коді введено обмеження на швидкість зміни вектора наведення. Це означає, що система не передає різкі команди, а формує більш плавний і фізично реалізований рух БПЛА.

Передавання команд автопілоту виконується через функції встановлення режиму GUIDED та надсилання цільової точки у глобальних координатах. Крім того, в програмі передбачено службові команди керування: увімкнення або вимкнення автопілота, примусове встановлення GUIDED-режиму, виведення діагностичної інформації, скидання супроводу та перемикання режиму відображення. Ці елементи дають змогу не лише автоматизувати супровід, а й забезпечити контроль системи з боку оператора.

3.2 Реалізація моделі на мові програмування Python та моделювання в QGroundControl

Реалізація моделі виконана мовою **Python (https://pypi.org/project/opencv-python/)** з використанням бібліотек **OpenCV (https://pypi.org/project/opencv-python/)**, **NumPy (https://numpy.org/)** та **pymavlink (https://pypi.org/project/pymavlink/)**. Програма поєднує кілька основних частин: захоплення відеопотоку, трекінг цілі, фільтрацію координат, перетворення координат у систему NED, обмін телеметрією з автопілотом та надсилання команд наведення. У коді ці частини реалізовані як окремі логічні блоки, що працюють у межах одного циклу обробки кадру.

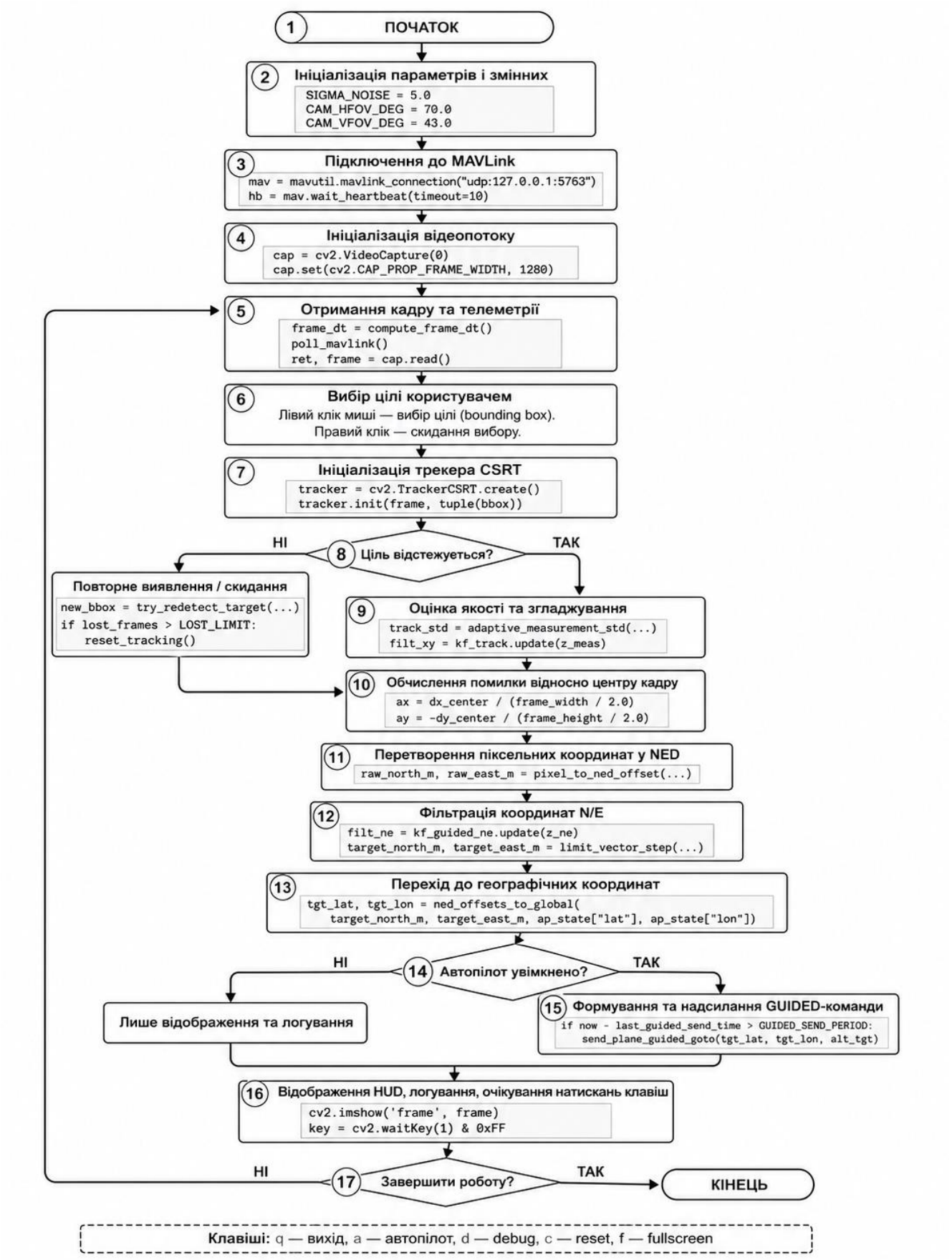


Рис.3.3 Блок-схема алгоритму автоматизованого супроводу рухомої цілі БПЛА.

На початку програми задаються службові константи, які визначають роботу всієї системи. До них належать параметри шуму вимірювання SIGMA_NOISE, поріг похибки THRESHOLD_ERROR, межі для адаптивної фільтрації TRACK_MEAS_STD_MIN, TRACK_MEAS_STD_MAX, NE_MEAS_STD_MIN, NE_MEAS_STD_MAX, обмеження на дальність точки наведення GUIDED_MIN_RANGE_M, GUIDED_MAX_RANGE_M, допустима швидкість зміни координат MAX_NE_STEP_M_PER_SEC, а також параметри камери CAM_HFOV_DEG, CAM_VFOV_DEG і кути її встановлення CAM_MOUNT_ROLL_DEG, CAM_MOUNT_PITCH_DEG, CAM_MOUNT_YAW_DEG. Окремо задаються параметри керування в режимі GUIDED, зокрема GUIDED_FORWARD_DIST, GUIDED_SIDE_MAX, GUIDED_ALT_MAX_DELTA і GUIDED_SEND_PERIOD. Саме ці константи визначають поведінку системи під час супроводу цілі.

Для згладжування координат у програмі створено клас KalmanFilter. У ньому реалізовані методи predict(), update(), set_dt(), set_measurement_std() та reset_state(). Стан фільтра задається у вигляді вектора з чотирьох компонент: двох координат і двох швидкостей. У коді використовуються два екземпляри цього класу: kf_track для фільтрації положення цілі в кадрі та kf_guided_ne для згладжування координат цільової точки в системі NED. Така побудова дозволяє спочатку стабілізувати положення об’єкта на зображенні, а потім окремо згладжувати просторові координати, що передаються автопілоту.

Для роботи з часовими інтервалами використовується функція compute_frame_dt(), яка визначає фактичний інтервал між сусідніми кадрами. Це значення далі передається у фільтр Калмана і впливає на матрицю переходу стану. Крім цього, функції adaptive_measurement_std() та adaptive_threshold() дозволяють змінювати параметри шуму вимірювання та пороги перевірки залежно від якості спостереження. Для обмеження різких змін навігаційної точки застосовується функція limit_vector_step(), яка не

дозволяє новому значенню координат змінюватися швидше, ніж задано в параметрах системи. Це безпосередньо реалізує вимогу плавного наведення БПЛА.

Підключення до автопілота виконується через `mavutil.mavlink_connection("udp:127.0.0.1:5763")`. Після цього програма очікує сигнал `HEARTBEAT`, який підтверджує наявність з'єднання. Якщо підключення успішне, у змінних `mav.target_system` і `mav.target_component` задаються ідентифікатори цільової системи. Поточний стан автопілота зберігається в словнику `ap_state`, де окремо записуються висота, широта, довгота, швидкість, крен, тангаж, курс і режим роботи. Це дозволяє в кожний момент часу використовувати не абстрактні, а реальні телеметричні дані для побудови команди наведення.

Зчитування телеметрії виконується у функції `poll_mavlink()`. Вона обробляє повідомлення типів `HEARTBEAT`, `VFR_HUD`, `GLOBAL_POSITION_INT` та `ATTITUDE`. Через ці повідомлення система отримує режим автопілота, шляхову та повітряну швидкість, відносну висоту, координати БПЛА, а також кути крену, тангажу і курсу. Таким чином, у коді реалізовано повний канал зворотного зв'язку від автопілота до алгоритму супроводу.

Для переходу від координат цілі в кадрі до просторових координат у коді реалізовані функції `R_x()`, `R_y()`, `R_z()`, `pixel_to_ned_offset()` та `ned_offsets_to_global()`. Функції `R_x`, `R_y` і `R_z` формують матриці повороту навколо осей. Функція `pixel_to_ned_offset()` обчислює локальні зміщення цілі відносно БПЛА в системі NED на основі координат у кадрі, параметрів камери, висоти та орієнтації дрона. Функція `ned_offsets_to_global()` переводить ці зміщення в глобальні географічні координати. Саме цей фрагмент коду реалізує зв'язок між комп'ютерним зором і навігацією.

Робота з відеопотоком виконується через `cv2.VideoCapture(0)`. Після ініціалізації камери задаються параметри роздільної здатності, створюється вікно `frame`, а через `cv2.setMouseCallback()` підключається функція

mouse_callback(). Саме ця функція відповідає за вибір цілі: після натискання лівої кнопки миші формується початковий прямокутник bbox, ініціалізується трекер cv2.TrackerCSRT.create(), запускається фільтрація та вмикається автопілот. Якщо натиснути праву кнопку миші, виконується скидання супроводу через reset_tracking(). Таким чином, у програмі реалізовано простий, але функціональний інтерфейс взаємодії з оператором.

У головному циклі програми спочатку визначається часовий крок, опитуються MAVLink-повідомлення та зчитується новий кадр. Якщо супровід активний, викликається tracker.update(frame), після чого програма визначає нове положення об'єкта. Далі перевіряється, чи не відбувся різкий стрибок координат. Якщо стрибок перевищує поріг JUMP_LIMIT_PIXELS, поточне вимірювання вважається недостовірним. У такому разі запускається функція try_redetect_target(), яка виконує повторний пошук об'єкта за шаблоном. Це дає змогу зменшити кількість помилок супроводу при нестабільному зображенні.

Після цього визначаються сирі координати цілі та виконується їх згладжування через kf_track. На основі різкості зображення в області цілі обчислюється показник якості, який впливає на адаптивний вибір шуму вимірювання та порогів перевірки. Відфільтровані координати використовуються для побудови допоміжних величин ax та ay, які відображають зміщення цілі відносно центра кадру. Саме ці величини далі використовуються для визначення напрямку корекції польоту.

Якщо автопілот активний і доступні координати та орієнтація БПЛА, виконується розрахунок локальної цільової точки через pixel_to_ned_offset(). Якщо цей розрахунок неможливий, у коді передбачено резервний спосіб оцінювання точки наведення. Далі координати в системі NED проходять через другий контур фільтрації kf_guided_ne, після чого обмежуються функцією limit_vector_step(). На основі значення ay

додатково формується корекція висоти. У результаті обчислюються `tgt_lat`, `tgt_lon` і `alt_tgt` — координати точки, яка має бути передана автопілоту.

Передавання команди автопілоту виконується через `send_plane_guided_goto()`. У цій функції використовується MAVLink-повідомлення `mission_item_int_send`, у якому передаються широта, довгота та відносна висота цільової точки. Перед цим, за потреби, через `set_guided_mode()` виконується переведення автопілота в режим GUIDED. Надсилання команд виконується не на кожному кадрі, а з часовим інтервалом `GUIDED_SEND_PERIOD`, що дозволяє уникати надмірно частих оновлень і робить контур керування стабільнішим.

У програмі також реалізовані команди керування з клавіатури. Натискання **q** завершує роботу, **a** вмикає або вимикає автопілот, **g** примусово встановлює GUIDED-режим, **d** перемикає режим діагностики, **c** виконує скидання супроводу, а **f** вмикає або вимикає повноекранний режим. Ці команди полегшують тестування системи та дозволяють швидко змінювати режим її роботи під час моделювання.

Моделювання системи виконувалося в **QgroundControl (https://qgroundcontrol.com/)**, який використовувався для контролю режиму автопілота, приймання команд GUIDED та спостереження за реакцією БПЛА (Рис.3.3).

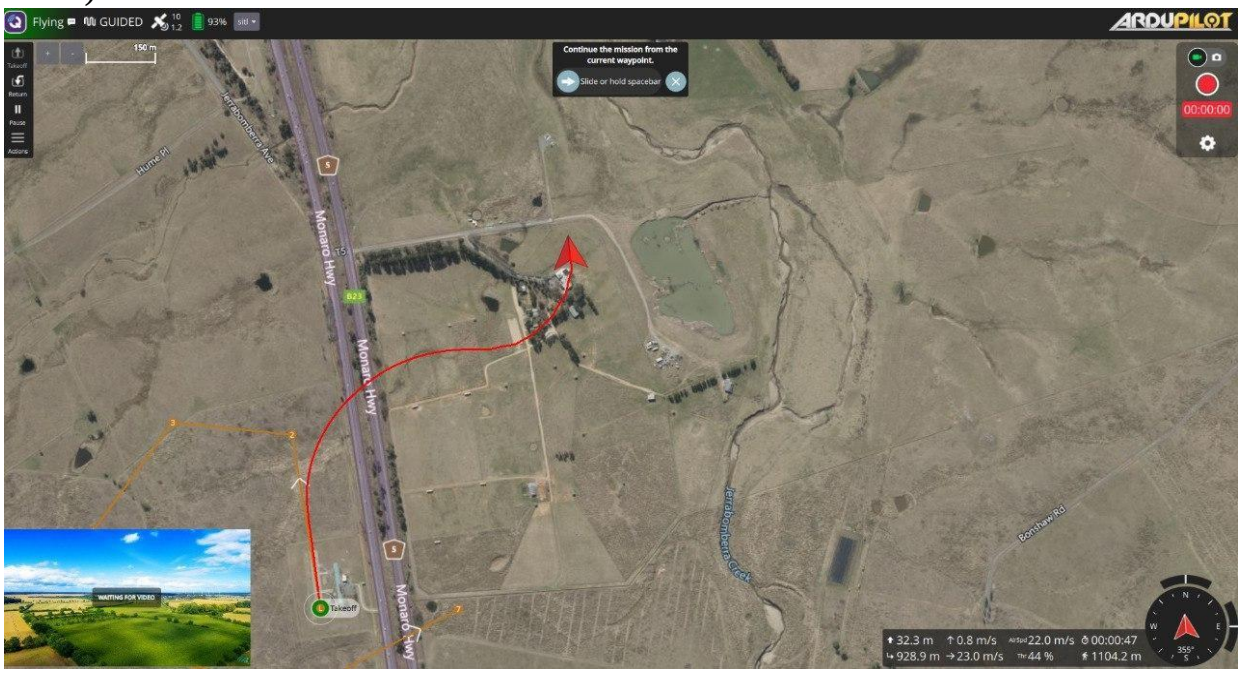


Рис.3.3 Моделювання системи в **QgroundControl.**

У поєднанні з програмним кодом на Python це дало можливість відтворити повний цикл роботи системи: вибір цілі, її супровід у кадрі, розрахунок координат точки наведення та передавання цієї точки автопілоту. Отже, програмна реалізація безпосередньо підтверджує можливість практичного використання побудованої математичної моделі в автоматизованій системі супроводу рухомої цілі.

3.3 Інтеграція оптимізаційного алгоритму в систему навігації

Інтеграція оптимізаційного алгоритму в систему навігації полягає у поєднанні візуального супроводу цілі, оцінювання її координат та формування траєкторної точки наведення для безпілотного літального апарата. У реалізованій системі оптимізація не зводиться до пошуку глобально найкоротшого маршруту, а виконується на рівні локального керування. Її основне завдання полягає у згладжуванні координат цілі, обмеженні різких змін траєкторної точки та формуванні таких команд, які відповідають динамічним можливостям БПЛА і можуть бути реалізовані в режимі реального часу. Це досягається через поєднання фільтрації Калмана, адаптивної зміни параметрів вимірювання та обмеження швидкості зміни вектора наведення.

У програмній реалізації оптимізаційний алгоритм інтегрований у систему навігації на двох рівнях. Перший рівень пов’язаний із координатами цілі в площині зображення. Після роботи трекера координати центра об’єкта не передаються далі безпосередньо, а спочатку проходять через фільтр `kf_track`. Для цього в коді використовується адаптивне налаштування дисперсії шуму вимірювання через функцію `adaptive_measurement_std()`, а також перевірка різниці між прогнозом і поточним вимірюванням. Якщо ця різниця перевищує допустимий поріг, фільтр скидає стан і перебудовує оцінку. Такий механізм дозволяє зменшити вплив випадкових стрибків координат, нестабільності трекера та короткочасних похибок спостереження.

Другий рівень оптимізації реалізується після переходу до локальної системи координат NED. Після визначення локальних зміщень цілі формується нова

траєкторна точка наведення, яка також не використовується безпосередньо. Для її обробки в коді застосовується другий фільтр Калмана `kf_guided_ne`, що працює з координатами North і East. Після цього до відфільтрованих значень застосовується функція `limit_vector_step()`, яка обмежує максимальну швидкість зміни координат траєкторної точки. Саме ця процедура є ключовим елементом локальної оптимізації, оскільки вона не дозволяє системі формувати надто різкі команди наведення, які могли б призвести до нестійкого руху БПЛА.

Інтеграція оптимізаційного алгоритму в навігаційний контур тісно пов'язана з використанням телеметричних даних автопілота. У коді постійно опитуються повідомлення MAVLink, з яких визначаються висота, координати, курс, крен, тангаж, швидкість польоту та активний режим роботи апарата. Ці дані використовуються не лише для геометричного перетворення координат, а і для узгодження команд наведення з поточним станом БПЛА. Зокрема, при формуванні цільової точки враховується поточна висота апарата, а також базова висота, зафіксована в момент захоплення цілі. Додаткова корекція по вертикалі обчислюється на основі нормованої похибки `ay`, що дозволяє змінювати висоту не довільно, а відповідно до зміщення об'єкта у кадрі.

Окрему роль в інтеграції алгоритму відіграє режим **GUIDED**, у якому автопілот приймає зовнішні навігаційні команди. У коді передбачено функцію `set_guided_mode()`, яка переводить систему в цей режим, а також функцію `send_plane_guided_goto()`, що передає нову цільову точку у вигляді MAVLink-команди. Отже, оптимізаційний алгоритм не є відокремленим програмним блоком, а безпосередньо вбудований у навігаційний цикл: координати цілі визначаються, фільтруються, обмежуються, перетворюються у глобальні координати, після чого передаються автопілоту для виконання.

Важливою особливістю інтегрованого підходу є те, що в системі поєднуються швидкодія та стійкість. З одного боку, обробка координат виконується для кожного нового кадру, що дає змогу оперативно реагувати на переміщення цілі. З іншого боку, через фільтрацію та обмеження швидкості зміни траєкторної точки усуваються різкі коливання команд. Це особливо важливо для

малих БПЛА, оскільки вони чутливі до надмірно швидких змін керування та до зовнішніх збурень. Таким чином, інтегрований оптимізаційний алгоритм забезпечує компроміс між точністю супроводу цілі та стабільністю польоту.

3.4 Порівняння результатів переслідування з/без оптимізації

У моделі використовувався кадр 1280×720, параметри камери HFOV = 70° і VFOV = 43°, частота обробки 30 кадр/с та відносна висота польоту 120 м. До еталонної траєкторії додавалися гаусів шум і поодинокі великі стрибки, що імітують нестабільність трекера.

Порівнювалися два варіанти: 1) безпосереднє використання виміряних координат без згладжування; 2) використання адаптивного фільтра Калмана з перевіркою інновації та м’яким скиданням стану. Такий сценарій не замінює повноцінний льотний експеримент, проте добре демонструє, як працює запропонована архітектура в умовах, наближених до реального відеосупроводу (Таблиця 3.1), (Рис. 3.4, рис. 3.5).

Таблиця 3.1. Порівняння показників для нефільтрованої та відфільтрованої траєкторії

Показник	Без фільтрації	Адаптивний KF
RMSE у площині зображення, пікс.	32.05	29.29
Середній крок між сусідніми оцінками, пікс.	34.95	4.34
RMSE після переходу до NED, м	4.21	3.85
Середній крок керуючої точки у NED, м	4.59	0.57

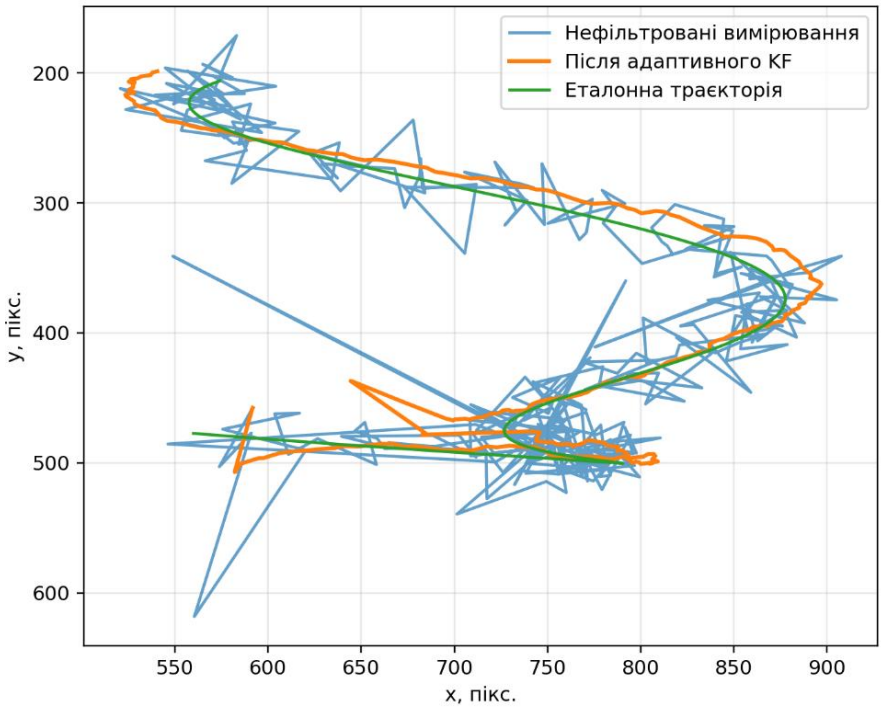


Рис. 3.4 Траєкторія цілі у площині зображення: вимірювання, еталон та оцінка після адаптивного фільтра Калмана.

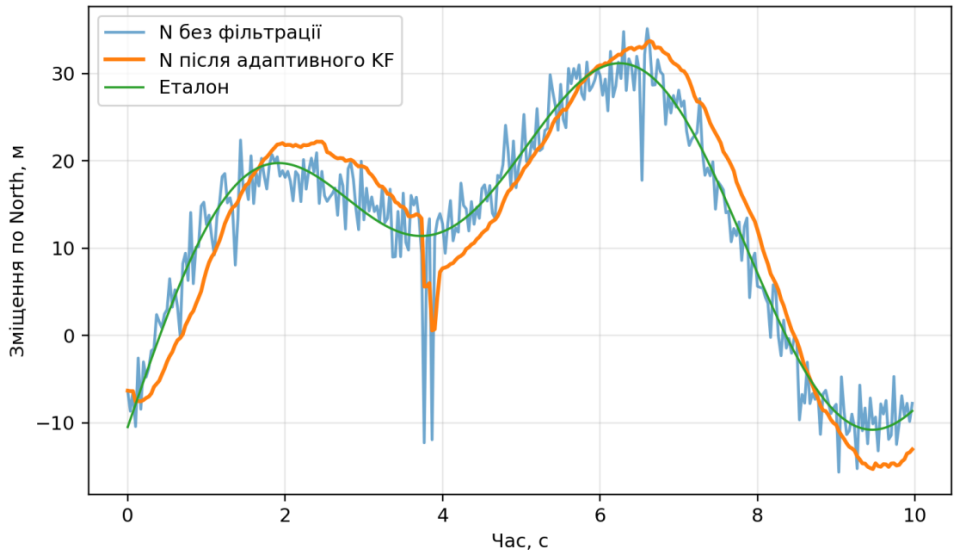


Рис. 3.5. Зміна координати North у часі без фільтрації та після адаптивного згладжування.

У проведеному експерименті RMSE у площині зображення зменшилася з 32.05 до 29.29 пікс., тобто приблизно на 8.6 %. Основний ефект виявився не лише у точності, а й у значному зменшенні ривковості: середній крок між сусідніми оцінками знизився з 34.95 до 4.34 пікс.

Після переходу до локальної системи NED середня зміна керуючої точки між сусідніми відліками зменшилася з 4.59 до 0.57 м, тобто більш ніж у 8.0 раза. Це є важливим саме для UAV, оскільки автопілот отримує фізично більш плавну команду наведення.

Висновки до розділу 3

У третьому розділі було розглянуто структуру автоматизованої системи керування дроном, реалізовано модель супроводу рухомої цілі на мові програмування Python та досліджено її роботу в середовищі QGroundControl. Показано, що побудована система об’єднує модуль комп’ютерного зору, блок фільтрації координат, модуль просторових перетворень, підсистему приймання телеметрії та контур формування керуючих команд для автопілота. Така структура забезпечує замкнений цикл керування, у якому результати візуального спостереження безпосередньо використовуються для наведення БПЛА на ціль.

У межах програмної реалізації було використано бібліотеки OpenCV, NumPy та pymavlink. Реалізовано вибір і супровід цілі у відеопотоці, згладжування координат за допомогою фільтра Калмана, перехід від координат у кадрі до локальної системи NED та формування команд наведення в режимі GUIDED. Додатково в програмі передбачено перевірку різких стрибків координат, повторне захоплення об’єкта, адаптивне налаштування параметрів фільтрації та обмеження швидкості зміни траєкторної точки, що підвищує стійкість роботи системи в реальному часі.

У результаті моделювання встановлено, що використання фільтрації та обмеження темпу зміни команд дозволяє підвищити точність супроводу, зменшити коливання траєкторії та зробити керування більш плавним. Це позитивно впливає як на стійкість системи, так і на енергоефективність польоту, оскільки зменшується кількість різких маневрів. Також підтверджено, що швидкодія реалізованого алгоритму є достатньою для роботи в режимі, близькому до реального часу, а інтеграція з QGroundControl дає змогу перевіряти роботу системи в умовах, наближених до практичного застосування.

ВИСНОВКИ

У роботі запропоновано технічне рішення автоматизованої системи супроводу рухомої цілі безпілотним літальним апаратом, що поєднує засоби комп'ютерного зору, математичну обробку координат та навігаційне керування дроном у режимі реального часу. Запропонований підхід базується на аналізі наукових джерел, сучасних методів візуального стеження, алгоритмів навігації та підходів до оптимізації траєкторії руху БПЛА.

Розроблене рішення забезпечує супровід рухомої цілі за даними бортової камери, визначення її положення у кадрі, перетворення координат у просторову систему та формування команд наведення для безпілотного літального апарата. Застосування фільтрації координат, обмеження швидкості зміни траєкторної точки та інтеграції з автопілотом забезпечує підвищення стійкості супроводу, зменшення впливу випадкових похибок і забезпечує плавне керування дроном. Це у свою чергу підвищує точність наведення та стабільність функціонування системи в реальному часі.

У теоретичній частині роботи було проаналізовано сучасні технології переслідування цілей безпілотними літальними апаратми, класифікацію алгоритмів супроводу, методи автоматичного стеження, математичні моделі навігації та керування, а також підходи до оптимізації в задачах супроводу рухомих об'єктів. Формалізовано задачу переслідування, побудовано математичні моделі руху дрона і цілі, визначено обмеження, початкові та граничні умови функціонування системи, а також обґрунтовано вибір методу локальної оптимізації траєкторної точки наведення.

У практичній частині роботи реалізовано програмну модель системи на мові Python із використанням бібліотек OpenCV, NumPy та pymavlink. Було розроблено алгоритм виявлення та супроводу цілі у відеопотоці, реалізовано фільтрацію координат із застосуванням фільтра Калмана, перетворення координат зображення у систему NED та формування команд наведення в режимі GUIDED. Для перевірки працездатності системи використано середовище QGroundControl, що дозволило

виконати моделювання процесу супроводу рухомої цілі та оцінити основні характеристики системи, зокрема точність, швидкодію та плавність керування.

Отже, поставлену мету досягнуто, а результати роботи підтверджують можливість практичної реалізації автоматизованої системи супроводу рухомої цілі безпілотним літальним апаратом. Запропоноване технічне рішення може бути використане як основа для подальшого вдосконалення безпілотних систем керування, а також для розширення їх функціональних можливостей у задачах моніторингу, спостереження та автоматизованого наведення.

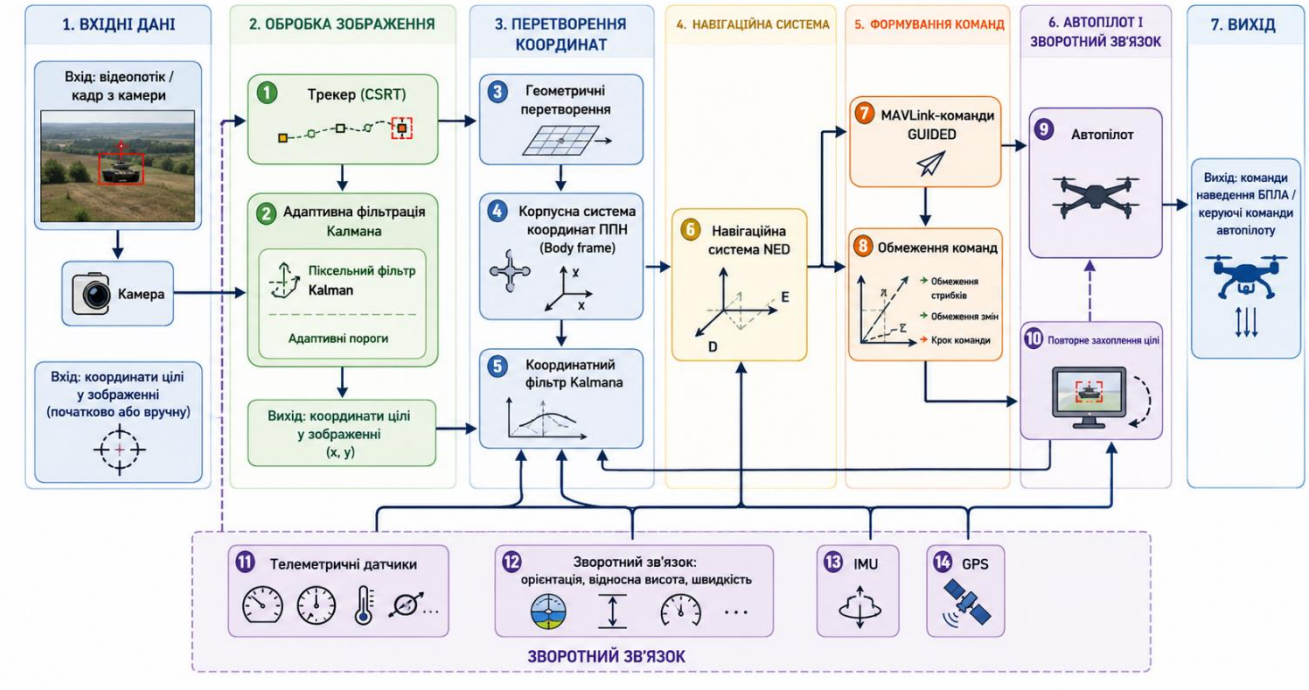
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

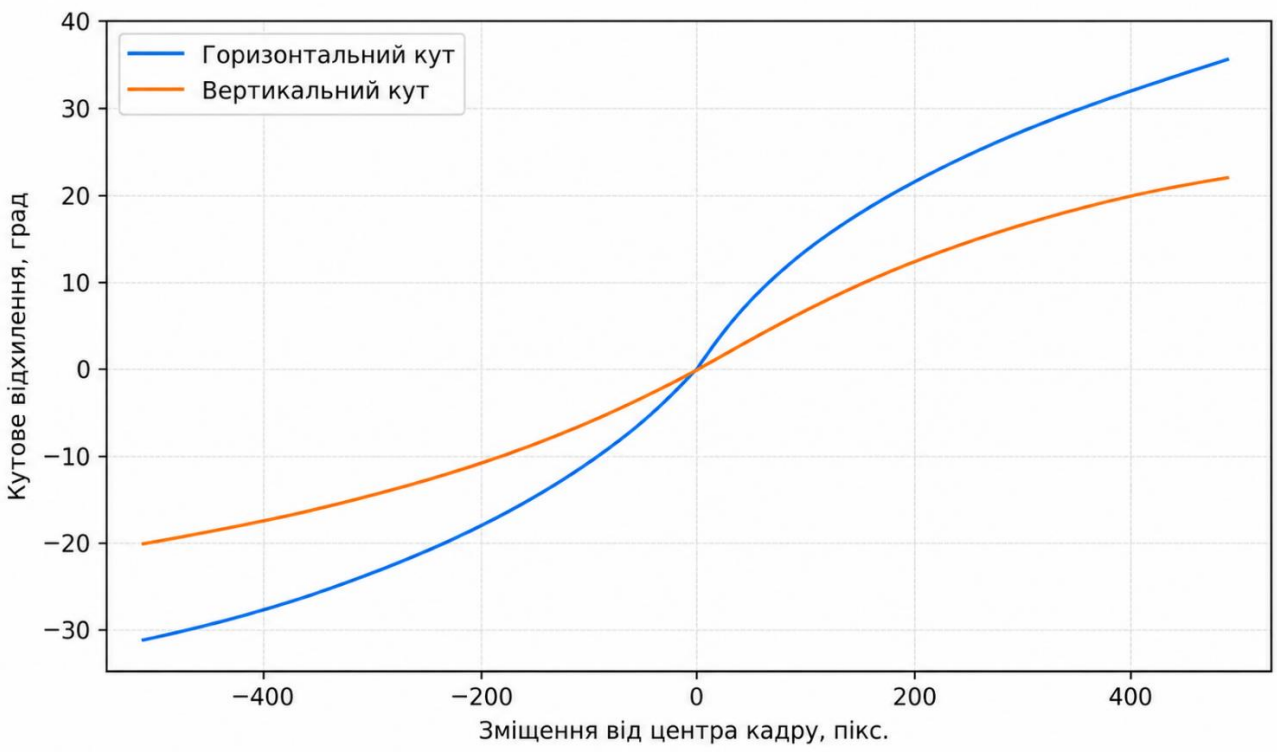
1. UAV target tracking: a survey – Режим доступу - https://link.springer.com/article/10.1007/s10462-025-11348-x?utm_source
2. A review of current studies on the unmanned aerial vehicle-based moving target tracking methods – Режим доступу - https://www.sciencedirect.com/science/article/pii/S2214914725000327?utm_source
3. Visual Servoing Using Sliding-Mode Control with Dynamic Compensation for UAVs' Tracking of Moving Targets – Режим доступу - https://www.mdpi.com/2504-446X/8/12/730?utm_source
4. Visual Servoing of a Moving Target by an Unmanned Aerial Vehicle – Режим доступу - https://www.mdpi.com/1424-8220/21/17/5708
5. Autonomous Target Tracking of UAV Using High-Speed Visual Servoing – Режим доступу - https://www.mdpi.com/2076-3417/9/21/4552
6. Online Predictive Visual Servo Control for Constrained Target Tracking of Fixed-Wing Unmanned Aerial Vehicles – Режим доступу - https://www.mdpi.com/2504-446X/8/4/136
7. Real-Time Visual Tracking of Moving Targets Using a Low-Cost Unmanned Aerial Vehicle with a 3-Axis Stabilized Gimbal System – Режим доступу - https://www.mdpi.com/2076-3417/10/15/5064
8. Vision-Based Navigation Techniques for Unmanned Aerial Vehicles: Review and Challenges – Режим доступу - https://www.mdpi.com/2504-446X/7/2/89
9. Dynamic Target Tracking and Following with UAVs Using Multi-Target Information: Leveraging YOLOv8 and MOT Algorithms – Режим доступу - https://www.mdpi.com/2504-446X/8/9/488
10. Autonomous UAV Chasing with Monocular Vision: A Learning-Based Approach – Режим доступу - https://www.mdpi.com/2226-4310/11/11/928
11. N. Stelmakh, Y. Yukhymenko, I. Rudkovskiy, and A. Lavrinenkov, "Object detection algorithm in a navigation system for a rescue drone", Informatyka, Automatyka, Pomiary W Gospodarce I Ochronie Środowiska, vol. 15, no. 2, pp. 32–36, 2025. DOI: 10.35784/iapgos.7090

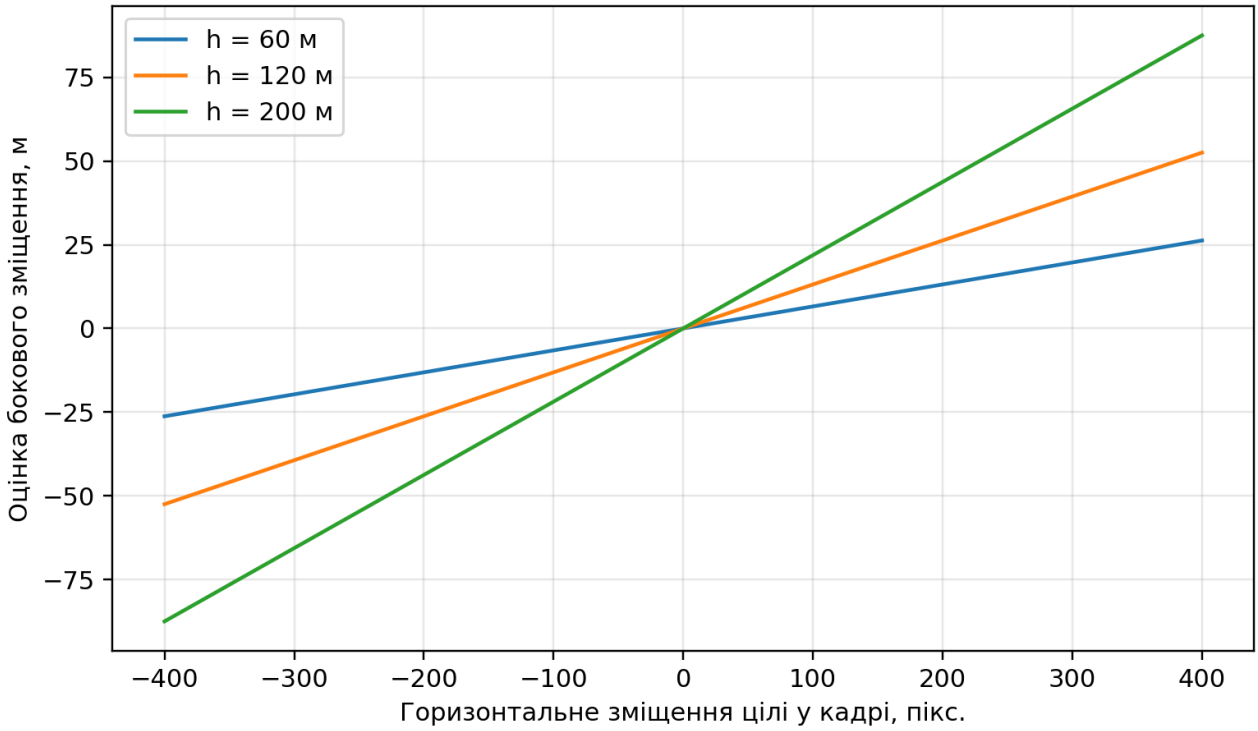
12. Ю. . Юхименко і Н. . Стельмах, «Біоінспірований метод виявлення фронтального наближення БПЛА в умовах обмежених обчислювальних ресурсів», Bull. Kyiv Polytech. Inst. Ser. Instrum. Mak., вип. 71(1), с. 143–149, Трав 2026. DOI: https://doi.org/10.20535/1970.71(1).2026.361935
13. Multi-UAVs collaborative tracking of moving target with maximized visibility in urban environment – Режим доступу - https://www.sciencedirect.com/science/article/abs/pii/S0016003222002927?
14. A Comprehensive Review of Path-Planning Algorithms for Unmanned Aerial Vehicles – Режим доступу - https://www.mdpi.com/2504-446X/10/1/11?
15. Advances in UAV Path Planning: A Comprehensive Survey – Режим доступу - https://www.mdpi.com/2504-446X/9/5/376
16. Multi-UAVs Collaborative Tracking – Режим доступу - https://www.sciencedirect.com/science/article/abs/pii/S0016003222002927?
17. Path Planning for Unmanned Aerial Vehicle: A-Star-Guided– Режим доступу - https://www.mdpi.com/2504-446X/9/8/545
18. Trajectory Planning and Tracking for UAVs with Deep Reinforcement Learning and Adaptive Nonlinear MPC – Режим доступу - https://www.sciencedirect.com/science/article/abs/pii/S0957417425042642
19. Unmanned Aerial Vehicle Logistics Distribution Path Planning Based on Improved Grey Wolf Optimization Algorithm – Режим доступ - https://www.mdpi.com/2073-8994/17/12/2178
20. A Collaborative Navigation Algorithm for Unmanned Aerial Vehicles – Режим доступу - https://www.mdpi.com/2504-446X/10/3/186
21. An Adaptive and Efficient Path Planning Algorithm for UAV Based on Rapidly Converging and Greedy-Optimized RRT* – Режим доступу - сторінка анотації ScienceDirect
22. A Survey on Multi-UAV Path Planning – Режим доступу - https://www.mdpi.com/2504-446X/9/4/263

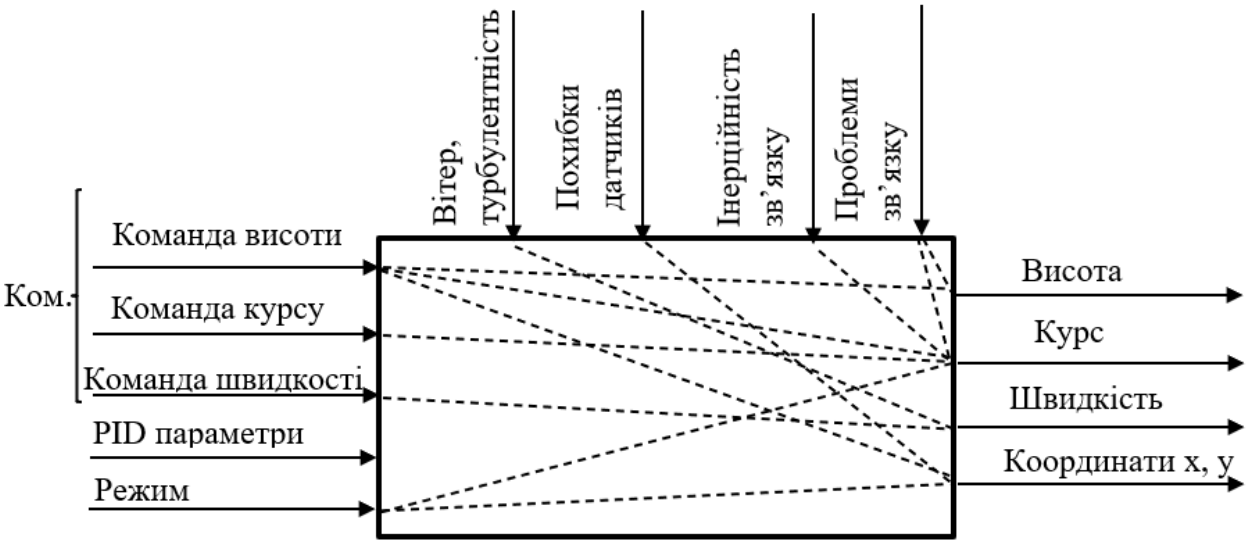
23. Multiple View Geometry in Computer Vision – Режим доступу - https://assets.cambridge.org/97805215/40513/frontmatter/9780521540513_frontmatter.pdf
24. A New Approach to Linear Filtering and Prediction Problems – Режим доступу - https://courses.cs.duke.edu/compsci527/cps274/fall11/papers/Kalman60.pdf
25. An Introduction to the Kalman Filter – Режим доступу - https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf
26. Autonomous Target Tracking of UAV Using High-Speed Visual Feedback – Режим доступу - https://www.mdpi.com/2076-3417/9/21/4552
27. Dynamic Target Tracking of Unmanned Aerial Vehicles Under Unpredictable Disturbances – Режим доступу - https://www.sciencedirect.com/science/article/pii/S209580992300276X
28. Visual Servoing of a Moving Target by an Unmanned Aerial Vehicle – Режим доступу - https://www.mdpi.com/1424-8220/21/17/5708
29. Visual Servoing Approach to Autonomous UAV Landing on a Moving Vehicle – Режим доступу - https://www.mdpi.com/1424-8220/22/17/6549
30. Yukhymenko, Y., & Stelmakh, N. (2025). Development of UAV behavior algorithm in case of control signal loss. КПІ ім. Ігоря Сікорського.

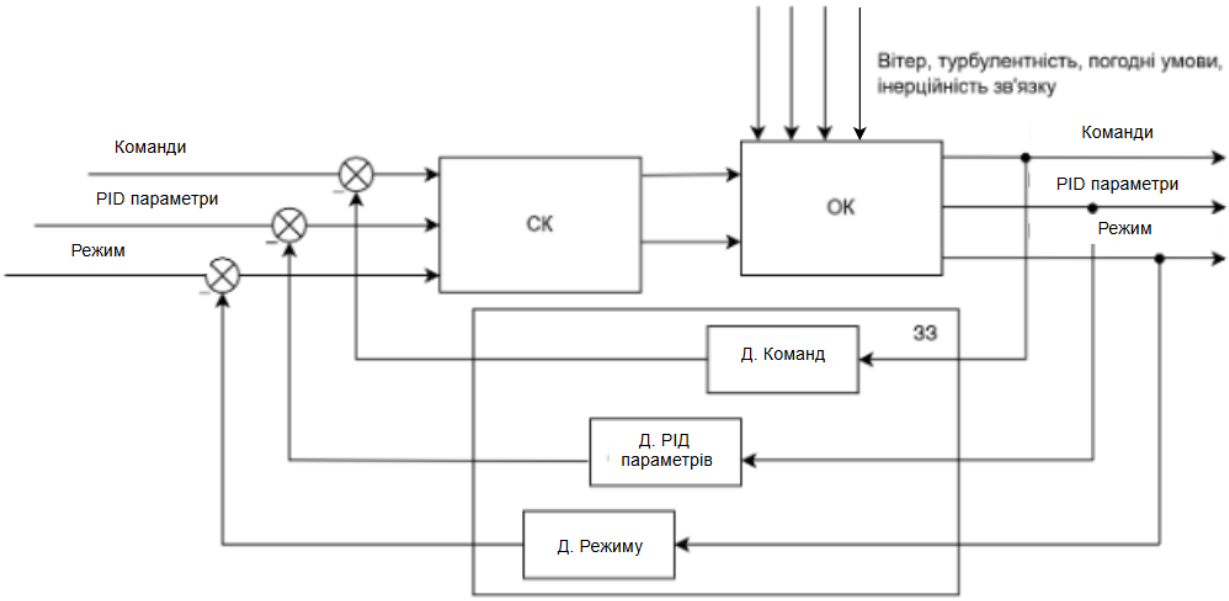
Додаток А

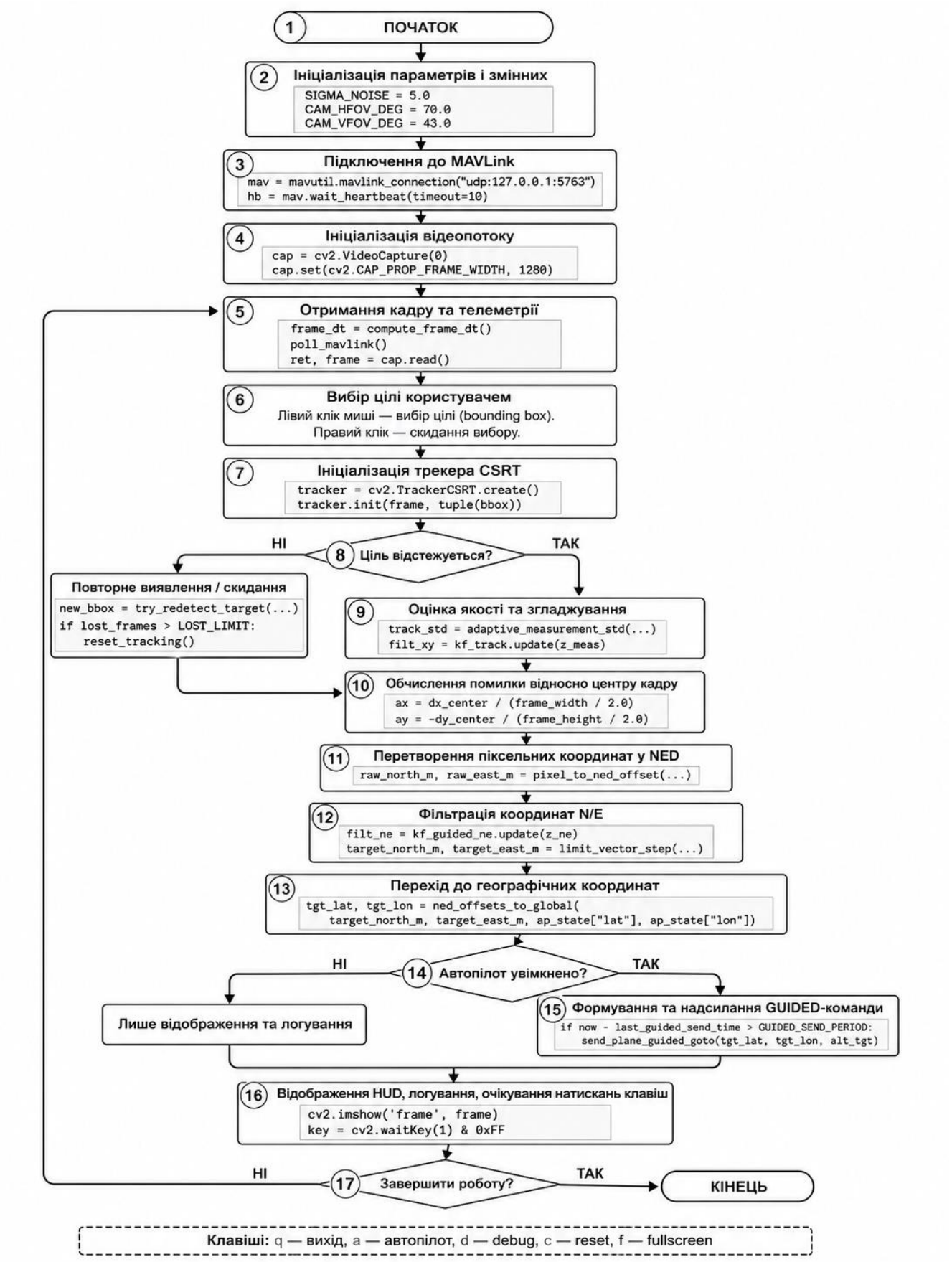

1. ВХІДНІ ДАНІ
Вхід: відеопотік / кадр з камери
Камера
Вхід: координати цілі у зображенні (початково або вручну)
2. ОБРОБКА ЗОБРАЖЕННЯ
1 Трекер (CSRT)
2 Адаптивна фільтрація Калмана
Піксельний фільтр Kalman
Адаптивні пороги
Вихід: координати цілі у зображенні (x, y)
3. ПЕРЕТВОРЕННЯ КООРДИНАТ
3 Геометричні перетворення
4 Корпусна система координат ППН (Body frame)
5 Координатний фільтр Kalmana
4. НАВІГАЦІЙНА СИСТЕМА
6 Навігаційна система NED
E
D
5. ФОРМУВАННЯ КОМАНД
7 MAVLink-команди GUIDED
8 Обмеження команд
Обмеження стрибків
Обмеження змін
Крок команди
6. АВТОПІЛОТ І ЗВОРОТНИЙ ЗВ'ЯЗОК
9 Автопілот
10 Повторне захоплення цілі
7. ВИХІД
Вихід: команди наведення БПЛА / керуючі команди автопілоту
11 Телеметричні датчики
12 Зворотний зв'язок: орієнтація, відносна висота, швидкість
13 IMU
14 GPS
ЗВОРОТНИЙ ЗВ'ЯЗОК

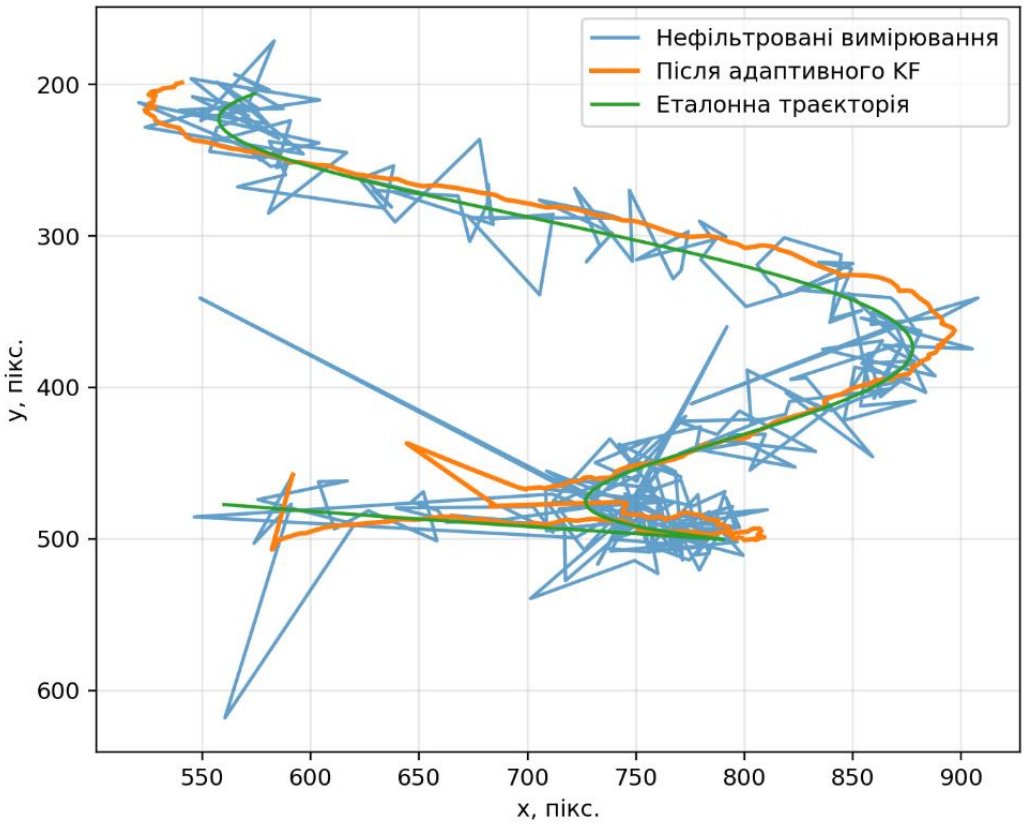

Горизонтальний кут
Вертикальний кут
Кутове відхилення, град
Зміщення від центра кадру, пікс.
40
30
20
10
0
−10
−20
−30
−400
−200
0
200
400

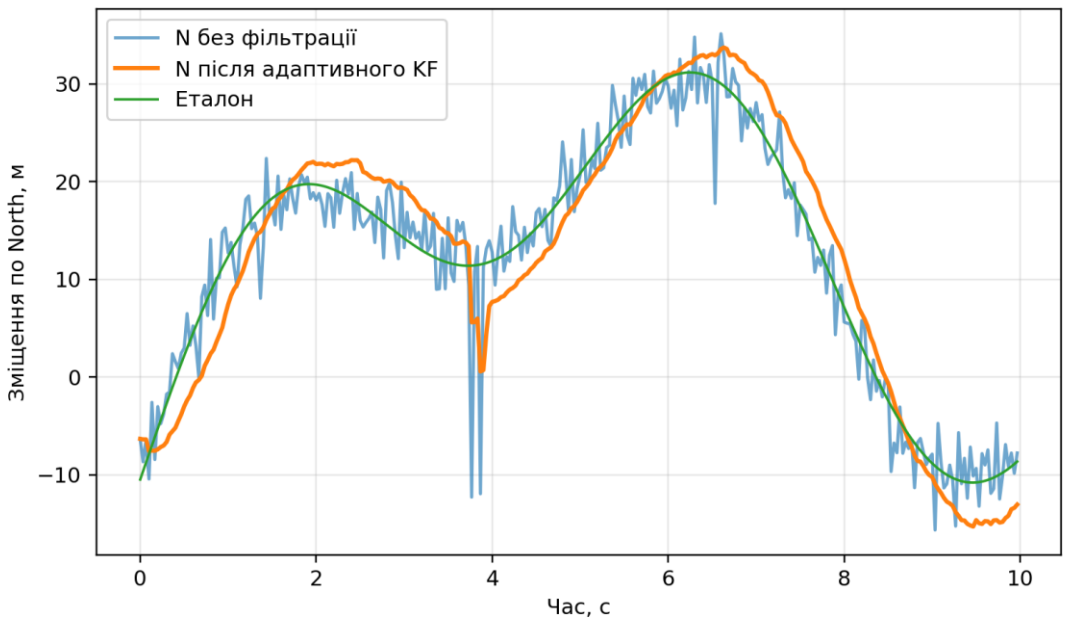

h = 60 м
h = 120 м
h = 200 м
Оцінка бокового зміщення, м
75
50
25
0
−25
−50
−75
−400
−300
−200
−100
0
100
200
300
400
Горизонтальне зміщення цілі у кадрі, пікс.


Ком.
Команда висоти
Команда курсу
Команда швидкості
PID параметри
Режим
Вітер, турбулентність
Похибки датчиків
Інерційність зв'язку
Проблеми зв'язку
Висота
Курс
Швидкість
Координати x, y


Вітер, турбулентність, погодні умови, інерційність зв'язку
Команди
PID параметри
Режим
СК
ОК
Команди
PID параметри
Режим
ЗЗ
Д. Команд
Д. РІД параметрів
Д. Режиму


1 ПОЧАТОК
2 Ініціалізація параметрів і змінних
SIGMA_NOISE = 5.0
CAM_HFOV_DEG = 70.0
CAM_VFOV_DEG = 43.0
3 Підключення до MAVLink
mav = mavutil.mavlink_connection("udp:127.0.0.1:5763")
hb = mav.wait_heartbeat(timeout=10)
4 Ініціалізація відеопотоку
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
5 Отримання кадру та телеметрії
frame_dt = compute_frame_dt()
poll_mavlink()
ret, frame = cap.read()
6 Вибір цілі користувачем
Лівий клік миші — вибір цілі (bounding box).
Правий клік — скидання вибору.
7 Ініціалізація трекера CSRT
tracker = cv2.TrackerCSRT.create()
tracker.init(frame, tuple(bbox))
8 Ціль відстежується?
НІ
ТАК
Повторне виявлення / скидання
new_bbox = try_redetect_target(...)
if lost_frames > LOST_LIMIT:
reset_tracking()
9 Оцінка якості та згладжування
track_std = adaptive_measurement_std(...)
filt_xy = kf_track.update(z_meas)
10 Обчислення помилки відносно центру кадру
ax = dx_center / (frame_width / 2.0)
ay = -dy_center / (frame_height / 2.0)
11 Перетворення піксельних координат у NED
raw_north_m, raw_east_m = pixel_to_ned_offset(...)
12 Фільтрація координат N/E
filt_ne = kf_guided_ne.update(z_ne)
target_north_m, target_east_m = limit_vector_step(...)
13 Перехід до географічних координат
tgt_lat, tgt_lon = ned_offsets_to_global(
target_north_m, target_east_m, ap_state["lat"], ap_state["lon"])
14 Автопілот увімкнено?
НІ
ТАК
Лише відображення та логування
15 Формування та надсилання GUIDED-команди
if now - last_guided_send_time > GUIDED_SEND_PERIOD:
send_plane_guided_goto(tgt_lat, tgt_lon, alt_tgt)
16 Відображення HUD, логування, очікування натискань клавіш
cv2.imshow('frame', frame)
key = cv2.waitKey(1) & 0xFF
17 Завершити роботу?
НІ
ТАК
КІНЕЦЬ
Клавіші: q — вихід, a — автопілот, d — debug, c — reset, f — fullscreen


Нефільтровані вимірювання
Після адаптивного KF
Еталонна траєкторія
y, пікс.
x, пікс.
200
300
400
500
600
550
600
650
700
750
800
850
900


N без фільтрації
N після адаптивного KF
Еталон
Зміщення по North, м
Час, с
30
20
10
0
−10
0
2
4
6
8
10

Додаток Б

```
import math
import time
import cv2
import numpy as np
from pymavlink import mavutil

SIGMA_NOISE = 5.0
THRESHOLD_ERROR = 25.0
STD_ACC_MODEL = 0.1
TRACK_MEAS_STD_MIN = 2.0
TRACK_MEAS_STD_MAX = 14.0
NE_MEAS_STD_MIN = 4.0
NE_MEAS_STD_MAX = 40.0
TRACK_THRESHOLD_MIN = 18.0
TRACK_THRESHOLD_MAX = 60.0
GUIDED_NE_THRESHOLD_MIN_M = 35.0
GUIDED_NE_THRESHOLD_MAX_M = 180.0
MAX_NE_STEP_M_PER_SEC = 120.0
FRAME_DT_MIN = 1.0 / 120.0
FRAME_DT_MAX = 0.25
last_frame_time = None
last_frame_dt = 1.0 / 30.0

class KalmanFilter:
    def __init__(self, dt=1.0, u_x=0.0, u_y=0.0,
                 std_acc=STD_ACC_MODEL,
std_meas=SIGMA_NOISE):
        self.dt = max(FRAME_DT_MIN, min(FRAME_DT_MAX,
float(dt)))
        self.std_acc = float(std_acc)
        self.std_meas = float(std_meas)
        self.x = np.array([[float(u_x)], [float(u_y)],
[0.0], [0.0]], dtype=float)
        self.H = np.array(
            [[1, 0, 0, 0],
             [0, 1, 0, 0]],
            dtype=float
        )
        self.P = np.eye(4, dtype=float) * 500.0
        self._update_system_matrices()

    def _update_system_matrices(self):
        dt = self.dt
        dt2 = dt * dt
        dt3 = dt2 * dt
```

```
        dt4 = dt2 * dt2
        q = self.std_acc ** 2
        self.F = np.array(
            [[1, 0, dt, 0],
             [0, 1, 0, dt],
             [0, 0, 1, 0],
             [0, 0, 0, 1]],
            dtype=float
        )
        self.Q = q * np.array(
            [[dt4 / 4.0, 0, dt3 / 2.0, 0],
             [0, dt4 / 4.0, 0, dt3 / 2.0],
             [dt3 / 2.0, 0, dt2, 0],
             [0, dt3 / 2.0, 0, dt2]],
            dtype=float
        )
        self.R = np.eye(2, dtype=float) * (self.std_meas **
2)

    def set_dt(self, dt):
        self.dt = max(FRAME_DT_MIN, min(FRAME_DT_MAX,
float(dt)))
        self._update_system_matrices()

    def set_measurement_std(self, std_meas):
        self.std_meas = max(0.1, float(std_meas))
        self.R = np.eye(2, dtype=float) * (self.std_meas **
2)

    def predict(self):
        self.x = self.F @ self.x
        self.P = self.F @ self.P @ self.F.T + self.Q
        return self.x[:2].copy()

    def update(self, z):
        z = np.asarray(z, dtype=float).reshape(2, 1)
        y = z - (self.H @ self.x)
        S = self.H @ self.P @ self.H.T + self.R
        K = self.P @ self.H.T @ np.linalg.inv(S)
        self.x = self.x + (K @ y)
        I = np.eye(4, dtype=float)
        self.P = (I - (K @ self.H)) @ self.P
        return self.x[:2].copy()

    def reset_state(self, z):
```

```
            z = np.asarray(z, dtype=float).reshape(2, 1)
            self.x[0] = z[0]
            self.x[1] = z[1]
            self.x[2] = 0.0
            self.x[3] = 0.0
            self.P = np.eye(4, dtype=float) * 1000.0


def compute_frame_dt():
    global last_frame_time, last_frame_dt
    now = time.time()
    if last_frame_time is None:
        dt = last_frame_dt
    else:
        dt = now - last_frame_time
    last_frame_time = now
    dt = max(FRAME_DT_MIN, min(FRAME_DT_MAX, dt))
    last_frame_dt = dt
    return dt

def adaptive_measurement_std(quality, low_std, high_std):
    q = max(0.0, min(1.0, float(quality) / 100.0))
    return high_std - (high_std - low_std) * q

def adaptive_threshold(quality, low_thr, high_thr):
    q = max(0.0, min(1.0, float(quality) / 100.0))
    return high_thr - (high_thr - low_thr) * q

def limit_vector_step(prev_x, prev_y, new_x, new_y,
max_step):
    if prev_x is None or prev_y is None or max_step <= 0.0:
        return float(new_x), float(new_y)
    dx = float(new_x) - float(prev_x)
    dy = float(new_y) - float(prev_y)
    dist = math.hypot(dx, dy)
    if dist <= max_step or dist <= 1e-9:
        return float(new_x), float(new_y)
    scale = max_step / dist
    return float(prev_x + dx * scale), float(prev_y + dy *
scale)

tracking_on = False
tracker = None

bbox = None
```

```
initial_target_center = None  # (x0, y0)

CLICK_BBOX_W = 100
CLICK_BBOX_H = 100

frame_width = 640
frame_height = 480

autopilot_on = False                # будемо вмикати при
виборі цілі
lock_percent = 0                    # рівень захвату
debug_visible = False               # перемикаємо клавішею 'd'

quality_percent = 0

ax = 0.0   # -1..1 (ліва <0, права >0)
ay = 0.0   # -1..1 (вниз <0, вверх >0)

template_roi = None       # BGR
last_good_bbox = None     # останній стабільний bbox

prev_center = None
stagnant_frames = 0

lost_frames = 0
LOST_LIMIT = 60  # терпіння перед повним скиданням

smooth_bbox = None
initial_bbox_size = None  # (w0, h0)

last_speed = 0.0

kf_track = None
kalman_target_x = None
kalman_target_y = None
raw_target_x = None
raw_target_y = None

kf_guided_ne = None
raw_target_north_m = None
raw_target_east_m = None
kalman_target_north_m = None
kalman_target_east_m = None
```

```
JUMP_LIMIT_PIXELS = 120  # можна підкрутити за відчуттям

display_scale = 1.0
display_offset_x = 0
display_offset_y = 0
fullscreen_on = True  # стартуємо у повноекранному режимі

CAM_HFOV_DEG = 70.0   # горизонтальний кут огляду камери
(deg)
CAM_VFOV_DEG = 43.0   # вертикальний кут огляду (deg)

CAM_MOUNT_ROLL_DEG = 0.0
CAM_MOUNT_PITCH_DEG = -45.0
CAM_MOUNT_YAW_DEG = 0.0

GUIDED_FORWARD_DIST = 250.0        # м вперед по курсу
GUIDED_SIDE_MAX = 250.0            # макс. бічне зміщення
при |ax|=1
GUIDED_ALT_MAX_DELTA = 80.0        # макс. зміна висоти при
|ay|=1
GUIDED_SEND_PERIOD = 0.3           # сек між посиланням
команд

GUIDED_MIN_RANGE_M = 250.0   # мін. дальність точки
наведення (м) щоб не крутило кола
GUIDED_MAX_RANGE_M = 1500.0  # макс. дальність (м)
GUIDED_NE_SMOOTH_ALPHA = 0.35  # legacy EMA param (не
використовується після переходу на Kalman для N/E)
base_alt_lock = None               # базова висота при
захваті цілі
last_guided_send_time = 0.0        # час останньої команди
GUIDED

TARGET_SYSTEM_ID = 1       # ID SITL
TARGET_COMPONENT_ID = 1    # автопілот

ap_state = {
    "has_link": False,
    "alt": None,           # relative altitude, m (AGL)
    "groundspeed": None,   # m/s
    "mode": None,
    "system_status": None,
    "pitch": None,
    "roll": None,
    "yaw": None,
```

```
    "airspeed": None,
    "lat": None,           # deg
    "lon": None,           # deg
}

mav = None
try:
    mav = mavutil.mavlink_connection("udp:127.0.0.1:5763")
    print("MAVLink: waiting for HEARTBEAT on
udp:127.0.0.1:5763 ...")

    hb = mav.wait_heartbeat(timeout=10)
    if hb is None:
        print("MAVLink: no heartbeat within 10s, running
WITHOUT link")
        mav = None
    else:
        print(
            f"MAVLink: connected, remote
system={hb.get_srcSystem()}, "
            f"component={hb.get_srcComponent()}"
        )

        mav.target_system = TARGET_SYSTEM_ID
        mav.target_component = TARGET_COMPONENT_ID
        print(
            f"MAVLink: using
target_system={mav.target_system}, "
            f"target_component={mav.target_component}"
        )

        ap_state["has_link"] = True

except Exception as e:
    print("MAVLink: failed to open port 5763:", e)
    mav = None

def R_x(phi):
    c, s = math.cos(phi), math.sin(phi)
    return np.array([[1, 0, 0],
                     [0, c, -s],
                     [0, s, c]])

def R_y(theta):
    c, s = math.cos(theta), math.sin(theta)
```

```
    return np.array([[c, 0, s],
                     [0, 1, 0],
                     [-s, 0, c]])

def R_z(psi):
    c, s = math.cos(psi), math.sin(psi)
    return np.array([[c, -s, 0],
                     [s, c, 0],
                     [0, 0, 1]])

def pixel_to_ned_offset(u, v,
                        frame_w, frame_h,
                        hfov_deg, vfov_deg,
                        alt_agl,
                        roll_deg, pitch_deg, yaw_deg,
                        mount_roll_deg, mount_pitch_deg,
mount_yaw_deg):
    if alt_agl is None:
        return None, None

    cx = frame_w / 2.0
    cy = frame_h / 2.0
    hfov = math.radians(hfov_deg)
    vfov = math.radians(vfov_deg)

    fx = cx / math.tan(hfov / 2.0)
    fy = cy / math.tan(vfov / 2.0)

    x_img = (u - cx) / fx       # вправо +
    y_img = (v - cy) / fy       # вниз +

    v_cam = np.array([x_img, y_img, 1.0], dtype=float)
    v_cam /= np.linalg.norm(v_cam)

    R_align = np.array([[0, 0, 1],
                        [1, 0, 0],
                        [0, 1, 0]], dtype=float)
    v_body0 = R_align @ v_cam

    R_mount = (
        R_z(math.radians(mount_yaw_deg)) @
        R_y(math.radians(mount_pitch_deg)) @
        R_x(math.radians(mount_roll_deg))
    )
    v_body = R_mount @ v_body0
```

```
    roll = math.radians(roll_deg if roll_deg is not None
else 0.0)
    pitch = math.radians(pitch_deg if pitch_deg is not None
else 0.0)
    yaw = math.radians(yaw_deg if yaw_deg is not None else
0.0)

    R_body2ned = R_z(yaw) @ R_y(pitch) @ R_x(roll)
    v_ned = R_body2ned @ v_body

    dz = v_ned[2]
    if dz <= 1e-3:
        return None, None

    t = alt_agl / dz
    north = float(v_ned[0] * t)
    east = float(v_ned[1] * t)
    return north, east

def ned_offsets_to_global(north_m, east_m, lat0_deg,
lon0_deg):
    if lat0_deg is None or lon0_deg is None:
        return None, None

    R_earth = 6378137.0  # м
    d_lat = north_m / R_earth
    d_lon = east_m / (R_earth *
math.cos(math.radians(lat0_deg)))

    lat = lat0_deg + math.degrees(d_lat)
    lon = lon0_deg + math.degrees(d_lon)
    return lat, lon

def poll_mavlink():
    if mav is None:
        return

    global ap_state

    try:
        msg = mav.recv_match(blocking=False)
        while msg is not None:
            mtype = msg.get_type()
            ap_state["has_link"] = True  # щось прийшло -
```

```
лінк живий

            if mtype == "HEARTBEAT":
                try:
                    mode = mavutil.mode_string_v10(msg)
                except Exception:
                    mode = None
                ap_state["mode"] = mode
                ap_state["system_status"] =
msg.system_status

            elif mtype == "VFR_HUD":
                ap_state["groundspeed"] = msg.groundspeed
# m/s
                ap_state["airspeed"] = msg.airspeed       #
m/s

            elif mtype == "GLOBAL_POSITION_INT":
                ap_state["alt"] = msg.relative_alt / 1000.0
# м (AGL)
                ap_state["lat"] = msg.lat / 1e7
                ap_state["lon"] = msg.lon / 1e7

            elif mtype == "ATTITUDE":
                ap_state["roll"] = math.degrees(msg.roll)
                ap_state["pitch"] = math.degrees(msg.pitch)
                ap_state["yaw"] = math.degrees(msg.yaw)

            msg = mav.recv_match(blocking=False)

    except Exception as e:
        print("MAVLink error:", e)

def send_plane_guided_goto(lat_deg, lon_deg, alt_rel_m):
    global mav, ap_state

    if mav is None or not ap_state.get("has_link", False):
        return
    if lat_deg is None or lon_deg is None or alt_rel_m is
None:
        return

    try:
        mav.mav.mission_item_int_send(
            mav.target_system,
```

```
            mav.target_component,
            0,  # seq (неважливо для current=2)
            mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
# alt above home
            mavutil.mavlink.MAV_CMD_NAV_WAYPOINT,
# 16
            2,  # current=2 => guided goto
            0,  # autocontinue
            0, 0, 0, 0,  # param1..param4
            int(lat_deg * 1e7),  # x = LAT * 1e7   <--
ВИПРАВЛЕНО
            int(lon_deg * 1e7),  # y = LON * 1e7   <--
ВИПРАВЛЕНО
            float(alt_rel_m)
        )
    except Exception as e:
        print("MAVLink: failed to send plane guided goto:",
e)

def set_guided_mode():
    if mav is None or not ap_state["has_link"]:
        print("MAVLink: no link, cannot set GUIDED")
        return

    try:
        mapping = mav.mode_mapping()
        if mapping and "GUIDED" in mapping:
            mav.set_mode(mapping["GUIDED"])
            print(f"MAVLink: GUIDED set via mode_mapping
({mapping['GUIDED']})")
            return

        GUIDED_MODE_ID = 15
        mav.mav.command_long_send(
            mav.target_system,
            mav.target_component,
            mavutil.mavlink.MAV_CMD_DO_SET_MODE,
            0,

mavutil.mavlink.MAV_MODE_FLAG_CUSTOM_MODE_ENABLED,
            GUIDED_MODE_ID,
            0, 0, 0, 0, 0
        )
        print("MAVLink: GUIDED mode command sent (fallback
custom_mode=15)")
```

```
    except Exception as e:
        print("MAVLink: failed to set GUIDED:", e)

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)

cv2.namedWindow('frame', cv2.WINDOW_NORMAL)
cv2.setWindowProperty('frame', cv2.WND_PROP_FULLSCREEN,
cv2.WINDOW_FULLSCREEN)

def draw_dashed_line(img, pt1, pt2, color, thickness=1,
                     dash_length=8, gap_length=4,
alpha=0.5):
    x1, y1 = pt1
    x2, y2 = pt2
    length = math.hypot(x2 - x1, y2 - y1)
    if length == 0:
        return img

    overlay = img.copy()
    dx = (x2 - x1) / length
    dy = (y2 - y1) / length

    dist = 0.0
    while dist < length:
        start_x = int(x1 + dx * dist)
        start_y = int(y1 + dy * dist)
        end_dist = min(dist + dash_length, length)
        end_x = int(x1 + dx * end_dist)
        end_y = int(y1 + dy * end_dist)
        cv2.line(overlay, (start_x, start_y), (end_x,
end_y), color, thickness)
        dist += dash_length + gap_length

    return cv2.addWeighted(overlay, alpha, img, 1 - alpha,
0)

def display_to_frame_coords(x_disp, y_disp):
    return x_disp, y_disp

def reset_tracking():
    global tracking_on, tracker, bbox, template_roi,
last_good_bbox
```

```
    global prev_center, stagnant_frames, lost_frames
    global lock_percent, quality_percent, ax, ay,
initial_target_center
    global smooth_bbox, initial_bbox_size, last_speed
    global kf_track, kalman_target_x, kalman_target_y,
raw_target_x, raw_target_y
    global kf_guided_ne, raw_target_north_m,
raw_target_east_m
    global kalman_target_north_m, kalman_target_east_m
    global target_north_m, target_east_m
    global base_alt_lock, last_guided_send_time

    tracking_on = False
    tracker = None
    bbox = None
    template_roi = None
    last_good_bbox = None
    prev_center = None
    stagnant_frames = 0
    lost_frames = 0
    lock_percent = 0
    quality_percent = 0
    ax = ay = 0.0
    initial_target_center = None
    smooth_bbox = None
    initial_bbox_size = None
    last_speed = 0.0
    kf_track = None
    kalman_target_x = None
    kalman_target_y = None
    raw_target_x = None
    raw_target_y = None
    kf_guided_ne = None
    raw_target_north_m = None
    raw_target_east_m = None
    kalman_target_north_m = None
    kalman_target_east_m = None
    target_north_m = None
    target_east_m = None
    base_alt_lock = None
    last_guided_send_time = 0.0

target_north_m = None
target_east_m = None
```

```
def mouse_callback(event, x, y, flags, param):
    global bbox, tracking_on, tracker,
initial_target_center
    global frame_width, frame_height, template_roi,
last_good_bbox
    global prev_center, stagnant_frames, lost_frames
    global smooth_bbox, initial_bbox_size, last_speed
    global kf_track, kalman_target_x, kalman_target_y,
raw_target_x, raw_target_y
    global kf_guided_ne, raw_target_north_m,
raw_target_east_m
    global kalman_target_north_m, kalman_target_east_m
    global autopilot_on, base_alt_lock,
last_guided_send_time

    fx, fy = display_to_frame_coords(x, y)

    if event == cv2.EVENT_LBUTTONDOWN:
        w = CLICK_BBOX_W
        h = CLICK_BBOX_H

        x_min = int(fx - w / 2)
        y_min = int(fy - h / 2)

        if x_min < 0:
            x_min = 0
        if y_min < 0:
            y_min = 0
        if x_min + w > frame_width:
            x_min = frame_width - w
        if y_min + h > frame_height:
            y_min = frame_height - h

        bbox = (x_min, y_min, w, h)
        initial_target_center = (x_min + w // 2, y_min + h
// 2)

        tracking_on = True
        tracker = None
        template_roi = None
        last_good_bbox = bbox
        prev_center = None
        stagnant_frames = 0
        lost_frames = 0
        smooth_bbox = bbox
```

```
            initial_bbox_size = (w, h)
            last_speed = 0.0

            start_cx = x_min + w / 2.0
            start_cy = y_min + h / 2.0
            kf_track = KalmanFilter(dt=last_frame_dt,
u_x=start_cx, u_y=start_cy)
            kalman_target_x = int(start_cx)
            kalman_target_y = int(start_cy)
            raw_target_x = int(start_cx)
            raw_target_y = int(start_cy)
            kf_guided_ne = None
            raw_target_north_m = None
            raw_target_east_m = None
            kalman_target_north_m = None
            kalman_target_east_m = None

            base_alt_lock = ap_state["alt"] if ap_state["alt"]
is not None else 80.0
            last_guided_send_time = 0.0

            autopilot_on = True
            print("Target selected: AUTOPILOT ON")
            set_guided_mode()

        elif event == cv2.EVENT_RBUTTONDOWN:
            reset_tracking()
            autopilot_on = False
            print("Tracking reset, AUTOPILOT OFF")

cv2.setMouseCallback('frame', mouse_callback)

def try_redetect_target(frame, last_bbox, template,
base_threshold=0.25):
    global last_speed

    if last_bbox is None or template is None:
        return None

    x, y, w, h = [int(v) for v in last_bbox]

    if last_speed > 15:
        region_scale = 6.0
        scales = (0.5, 0.8, 1.0, 1.3, 1.7, 2.0)
        score_threshold = max(0.18, base_threshold - 0.05)
```

```
    elif last_speed > 5:
        region_scale = 4.5
        scales = (0.6, 0.8, 1.0, 1.2, 1.5)
        score_threshold = base_threshold
    else:
        region_scale = 3.0
        scales = (0.7, 1.0, 1.3)
        score_threshold = base_threshold

    search_w = int(w * region_scale)
    search_h = int(h * region_scale)

    cx_old = x + w // 2
    cy_old = y + h // 2

    sx = max(0, cx_old - search_w // 2)
    sy = max(0, cy_old - search_h // 2)
    ex = min(frame.shape[1], sx + search_w)
    ey = min(frame.shape[0], sy + search_h)

    search_roi = frame[sy:ey, sx:ex]
    if search_roi.size == 0:
        return None

    base_tmpl = template.copy()
    base_th, base_tw = base_tmpl.shape[:2]

    search_gray = cv2.cvtColor(search_roi,
cv2.COLOR_BGR2GRAY)

    best_val = -1.0
    best_loc = None
    best_w = None
    best_h = None

    for s in scales:
        tw = int(base_tw * s)
        th = int(base_th * s)
        if tw < 10 or th < 10:
            continue

        tmpl_resized = cv2.resize(base_tmpl, (tw, th),
interpolation=cv2.INTER_CUBIC)
        if search_roi.shape[1] < tw or search_roi.shape[0]
< th:
```

```
                continue

            tmpl_gray = cv2.cvtColor(tmpl_resized,
cv2.COLOR_BGR2GRAY)

            res = cv2.matchTemplate(search_gray, tmpl_gray,
cv2.TM_CCOEFF_NORMED)
            _, max_val, _, max_loc = cv2.minMaxLoc(res)

            if max_val > best_val:
                best_val = max_val
                best_loc = max_loc
                best_w = tw
                best_h = th

    if best_val < score_threshold or best_loc is None:
        return None

    nx = sx + best_loc[0]
    ny = sy + best_loc[1]

    cx_new = nx + best_w // 2
    cy_new = ny + best_h // 2
    jump = math.hypot(cx_new - cx_old, cy_new - cy_old)

    if jump > max(w, h) * 2.5:
        return None

    return (nx, ny, best_w, best_h)

while True:
    frame_dt = compute_frame_dt()
    poll_mavlink()

    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, 1)

    frame_height, frame_width, _ = frame.shape
    frame_center = (frame_width // 2, frame_height // 2)

    if cv2.getWindowProperty('frame', cv2.WND_PROP_VISIBLE)
< 1:
```

```
            break

        if bbox is None and not (tracking_on and tracker is not
None):
            cv2.putText(
                frame, "Click on target to start tracking",
                (20, 30),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.5, (0, 255, 0), 1
            )
            lock_percent = 0
            quality_percent = 0
            ax, ay = 0.0, 0.0

        if tracking_on and tracker is None and bbox is not
None:
            tracker = cv2.TrackerCSRT.create()
            tracker.init(frame, tuple(bbox))
            lost_frames = 0

        target_x = target_y = None
        raw_target_x = raw_target_y = None
        kalman_target_x = kalman_target_y = None
        dist_to_center = 0.0
        angle_to_center = 0.0
        dist_from_start = 0.0
        angle_from_start = 0.0

        cam_ang_horiz = 0.0
        cam_ang_down = 0.0

        target_north_m = None
        target_east_m = None
        raw_target_north_m = None
        raw_target_east_m = None
        kalman_target_north_m = None
        kalman_target_east_m = None

        if tracking_on and tracker is not None:
            success, tracked_bbox = tracker.update(frame)

            if success and last_good_bbox is not None:
                lx, ly, lw, lh = last_good_bbox
                prev_cx = lx + lw // 2
                prev_cy = ly + lh // 2
```

```
            tx_raw = int(tracked_bbox[0] + tracked_bbox[2]
/ 2)
            ty_raw = int(tracked_bbox[1] + tracked_bbox[3]
/ 2)
            jump = math.hypot(tx_raw - prev_cx, ty_raw -
prev_cy)

            if jump > JUMP_LIMIT_PIXELS:
                success = False

        if not success:
            lost_frames += 1
            new_bbox = try_redetect_target(
                frame,
                last_good_bbox if last_good_bbox is not
None else bbox,
                template_roi
            )
            if new_bbox is not None:
                bbox = new_bbox
                tracker = cv2.TrackerCSRT.create()
                tracker.init(frame, tuple(bbox))
                stagnant_frames = 0
                lost_frames = 0
                success, tracked_bbox =
tracker.update(frame)

        if success:
            lost_frames = 0

            x, y, w, h = [int(i) for i in tracked_bbox]
            target_x = x + w // 2
            target_y = y + h // 2
            raw_target_x = target_x
            raw_target_y = target_y

            if initial_bbox_size is not None:
                w0, h0 = initial_bbox_size
                min_w = int(w0 * 0.7)
                max_w_ = int(w0 * 1.3)
                min_h = int(h0 * 0.7)
                max_h_ = int(h0 * 1.3)
                w = max(min_w, min(max_w_, w))
                h = max(min_h, min(max_h_, h))
```

```
            if smooth_bbox is None:
                smooth_bbox = (x, y, w, h)
            else:
                px, py, pw, ph = smooth_bbox
                alpha = 0.5
                sx = int(px * (1 - alpha) + x * alpha)
                sy = int(py * (1 - alpha) + y * alpha)
                sw = int(pw * (1 - alpha) + w * alpha)
                sh = int(ph * (1 - alpha) + h * alpha)
                smooth_bbox = (sx, sy, sw, sh)

            x, y, w, h = smooth_bbox
            measured_x = x + w // 2
            measured_y = y + h // 2
            raw_target_x = int(measured_x)
            raw_target_y = int(measured_y)

            z_meas = np.array([[float(measured_x)],
[float(measured_y)]], dtype=float)
            track_std = adaptive_measurement_std(
                quality_percent,
                TRACK_MEAS_STD_MIN,
                TRACK_MEAS_STD_MAX
            )
            track_threshold = adaptive_threshold(
                quality_percent,
                TRACK_THRESHOLD_MIN,
                TRACK_THRESHOLD_MAX
            )
            if kf_track is None:
                kf_track = KalmanFilter(
                    dt=frame_dt,
                    u_x=measured_x,
                    u_y=measured_y,
                    std_acc=STD_ACC_MODEL,
                    std_meas=track_std
                )
                filt_xy = z_meas
            else:
                kf_track.set_dt(frame_dt)
                kf_track.set_measurement_std(track_std)
                pred_xy = kf_track.predict()
                error_dist = np.linalg.norm(pred_xy -
z_meas)
```

```
                if error_dist > track_threshold:
                    kf_track.reset_state(z_meas)
                    filt_xy = z_meas
                else:
                    filt_xy = kf_track.update(z_meas)

            target_x = int(float(filt_xy[0, 0]))
            target_y = int(float(filt_xy[1, 0]))
            kalman_target_x = target_x
            kalman_target_y = target_y

            margin = 2
            if (
                x <= margin or y <= margin or
                x + w >= frame_width - margin or
                y + h >= frame_height - margin
            ):
                reset_tracking()
            else:
                last_good_bbox = (x, y, w, h)

                roi = frame[y:y + h, x:x + w]
                if roi.size > 0:
                    gray = cv2.cvtColor(roi,
cv2.COLOR_BGR2GRAY)
                    lap = cv2.Laplacian(gray, cv2.CV_64F)
                    fm = lap.var()

                    TH_LOW = 30.0
                    TH_HIGH = 150.0

                    quality_norm = (fm - TH_LOW) / (TH_HIGH
- TH_LOW)
                    quality_norm = min(max(quality_norm,
0.0), 1.0)
                    quality_percent = int(quality_norm *
100)
                else:
                    quality_percent = 0
                    quality_norm = 0.0

                sub_w = max(10, int(w * 0.6))
                sub_h = max(10, int(h * 0.6))
                sx = x + (w - sub_w) // 2
                sy = y + (h - sub_h) // 2
```

```
                    sx = max(0, min(frame_width - sub_w, sx))
                    sy = max(0, min(frame_height - sub_h, sy))
                    new_template = frame[sy:sy + sub_h, sx:sx +
sub_w].copy()

                    if template_roi is None:
                        template_roi = new_template
                    else:
                        if quality_percent > 50:
                            h_old, w_old =
template_roi.shape[:2]
                            if (w_old, h_old) != (sub_w,
sub_h):
                                resized_old = cv2.resize(
                                    template_roi, (sub_w,
sub_h),

interpolation=cv2.INTER_CUBIC
                                )
                            else:
                                resized_old = template_roi
                            template_roi = cv2.addWeighted(
                                resized_old, 0.8,
                                new_template, 0.2, 0
                            )

                    tx_tmp = target_x
                    ty_tmp = target_y
                    cur_center = (tx_tmp, ty_tmp)
                    if prev_center is not None:
                        move = math.hypot(
                            cur_center[0] - prev_center[0],
                            cur_center[1] - prev_center[1]
                        )
                        last_speed = move
                        if move < 1.0:
                            stagnant_frames += 1
                        else:
                            stagnant_frames = 0
                    else:
                        last_speed = 0.0
                    prev_center = cur_center

                    if stagnant_frames >= 10:
                        new_bbox = try_redetect_target(
```

```
                            frame,
                            last_good_bbox,
                            template_roi,
                            base_threshold=0.3
                        )
                        if new_bbox is not None:
                            bbox = new_bbox
                            tracker = cv2.TrackerCSRT.create()
                            tracker.init(frame, tuple(bbox))
                            stagnant_frames = 0
                            success, tracked_bbox =
tracker.update(frame)
                            if success:
                                x, y, w, h = [int(i) for i in
tracked_bbox]

                                if initial_bbox_size is not
None:
                                    w0, h0 = initial_bbox_size
                                    min_w = int(w0 * 0.7)
                                    max_w_ = int(w0 * 1.3)
                                    min_h = int(h0 * 0.7)
                                    max_h_ = int(h0 * 1.3)
                                    w = max(min_w, min(max_w_,
w))
                                    h = max(min_h, min(max_h_,
h))

                                if smooth_bbox is None:
                                    smooth_bbox = (x, y, w, h)
                                else:
                                    px, py, pw, ph =
smooth_bbox
                                    alpha = 0.5
                                    sx = int(px * (1 - alpha) +
x * alpha)
                                    sy = int(py * (1 - alpha) +
y * alpha)
                                    sw = int(pw * (1 - alpha) +
w * alpha)
                                    sh = int(ph * (1 - alpha) +
h * alpha)
                                    smooth_bbox = (sx, sy, sw,
sh)
```

```
                                x, y, w, h = smooth_bbox
                                target_x = x + w // 2
                                target_y = y + h // 2
                                last_good_bbox = (x, y, w, h)

                    if quality_norm < 0.33:
                        box_color = (0, 0, 255)
                    elif quality_norm < 0.66:
                        box_color = (0, 255, 255)
                    else:
                        box_color = (0, 255, 0)

                    cv2.rectangle(frame, (x, y), (x + w, y +
h), box_color, 2)

                    cv2.line(frame, (target_x - 8, target_y),
                             (target_x + 8, target_y), (0, 255,
0), 1)
                    cv2.line(frame, (target_x, target_y - 8),
                             (target_x, target_y + 8), (0, 255,
0), 1)

                    if raw_target_x is not None and
raw_target_y is not None:
                        cv2.circle(frame, (raw_target_x,
raw_target_y), 3, (255, 0, 0), -1)

                    dx_center = target_x - frame_center[0]
                    dy_center = target_y - frame_center[1]
                    dist_to_center = math.hypot(dx_center,
dy_center)
                    angle_to_center = math.degrees(
                        math.atan2(-dy_center, dx_center)
                    )

                    if initial_target_center is not None:
                        dx_start = target_x -
initial_target_center[0]
                        dy_start = target_y -
initial_target_center[1]
                        dist_from_start = math.hypot(dx_start,
dy_start)
                        angle_from_start = math.degrees(
                            math.atan2(-dy_start, dx_start)
                        )
```

```
                if frame_width > 0 and frame_height > 0:
                    ax = dx_center / (frame_width / 2.0)
                    ay = -dy_center / (frame_height / 2.0)
                    ax = max(-1.0, min(1.0, ax))
                    ay = max(-1.0, min(1.0, ay))
                else:
                    ax, ay = 0.0, 0.0

                max_radius = math.hypot(frame_width / 2,
frame_height / 2)
                norm = min(dist_to_center / max_radius,
1.0)
                lock_percent = int((1.0 - norm) * 100)

                cam_ang_horiz = (dx_center / (frame_width /
2.0)) * (CAM_HFOV_DEG / 2.0)
                cam_ang_down = (dy_center / (frame_height /
2.0)) * (CAM_VFOV_DEG / 2.0)

                if (autopilot_on and
                        ap_state["lat"] is not None and
                        ap_state["lon"] is not None and
                        ap_state["yaw"] is not None):

                    alt_agl = ap_state["alt"] if
ap_state["alt"] is not None else 0.0
                    roll_deg = ap_state["roll"] if
ap_state["roll"] is not None else 0.0
                    pitch_deg = ap_state["pitch"] if
ap_state["pitch"] is not None else 0.0
                    yaw_deg = ap_state["yaw"] if
ap_state["yaw"] is not None else 0.0

                    raw_north_m, raw_east_m =
pixel_to_ned_offset(
                        target_x,
                        target_y,
                        frame_width,
                        frame_height,
                        CAM_HFOV_DEG,
                        CAM_VFOV_DEG,
                        alt_agl,
                        roll_deg,
                        pitch_deg,
```

```
                            yaw_deg,
                            CAM_MOUNT_ROLL_DEG,
                            CAM_MOUNT_PITCH_DEG,
                            CAM_MOUNT_YAW_DEG
                        )

                        if raw_north_m is None or raw_east_m is
None:
                            yaw_rad = math.radians(yaw_deg)
                            cx = frame_width / 2.0
                            cy = frame_height / 2.0
                            hfov = math.radians(CAM_HFOV_DEG)
                            vfov = math.radians(CAM_VFOV_DEG)
                            fx = cx / math.tan(hfov / 2.0)
                            fy = cy / math.tan(vfov / 2.0)
                            x_norm = (target_x - cx) / fx
                            y_norm = (target_y - cy) / fy
                            theta_x = math.atan(x_norm)
                            theta_y = math.atan(y_norm)
                            down0_deg = (-CAM_MOUNT_PITCH_DEG)
- pitch_deg
                            down_deg = max(1.0, min(89.0,
down0_deg + math.degrees(theta_y)))
                            range_m = alt_agl /
math.tan(math.radians(down_deg)) if alt_agl > 0.0 else
GUIDED_MIN_RANGE_M
                            range_m = max(GUIDED_MIN_RANGE_M,
min(GUIDED_MAX_RANGE_M, range_m))
                            body_x = range_m
                            body_y = range_m *
math.tan(theta_x)
                            raw_north_m = body_x *
math.cos(yaw_rad) - body_y * math.sin(yaw_rad)
                            raw_east_m = body_x *
math.sin(yaw_rad) + body_y * math.cos(yaw_rad)

                        raw_range = math.hypot(raw_north_m,
raw_east_m)
                        if raw_range > 1e-6:
                            clamped_range =
max(GUIDED_MIN_RANGE_M, min(GUIDED_MAX_RANGE_M, raw_range))
                            raw_north_m *= clamped_range /
raw_range
                            raw_east_m *= clamped_range /
raw_range
```

```
                    raw_target_north_m = float(raw_north_m)
                    raw_target_east_m = float(raw_east_m)

                    z_ne = np.array([[raw_target_north_m],
[raw_target_east_m]], dtype=float)
                    ne_std = adaptive_measurement_std(
                        quality_percent,
                        NE_MEAS_STD_MIN,
                        NE_MEAS_STD_MAX
                    )
                    ne_threshold = adaptive_threshold(
                        quality_percent,
                        GUIDED_NE_THRESHOLD_MIN_M,
                        GUIDED_NE_THRESHOLD_MAX_M
                    )

                    if kf_guided_ne is None:
                        kf_guided_ne = KalmanFilter(
                            dt=frame_dt,
                            u_x=raw_target_north_m,
                            u_y=raw_target_east_m,
                            std_acc=STD_ACC_MODEL,
                            std_meas=ne_std
                        )
                        filt_ne = z_ne
                    else:
                        kf_guided_ne.set_dt(frame_dt)

kf_guided_ne.set_measurement_std(ne_std)
                        pred_ne = kf_guided_ne.predict()
                        error_ne = np.linalg.norm(pred_ne -
z_ne)
                        if error_ne > ne_threshold:
                            kf_guided_ne.reset_state(z_ne)
                            filt_ne = z_ne
                        else:
                            filt_ne =
kf_guided_ne.update(z_ne)

                    limited_n, limited_e =
limit_vector_step(
                        kalman_target_north_m,
                        kalman_target_east_m,
                        float(filt_ne[0, 0]),
```

```
                        float(filt_ne[1, 0]),
                        MAX_NE_STEP_M_PER_SEC * frame_dt
                    )

                    target_north_m = limited_n
                    target_east_m = limited_e
                    kalman_target_north_m = target_north_m
                    kalman_target_east_m = target_east_m

                    if kf_guided_ne is not None:
                        kf_guided_ne.x[0, 0] =
target_north_m
                        kf_guided_ne.x[1, 0] =
target_east_m

                    base_alt = base_alt_lock if
base_alt_lock is not None else ap_state["alt"]
                    if base_alt is None:
                        base_alt = 80.0

                    alt_tgt = base_alt +
GUIDED_ALT_MAX_DELTA * ay
                    alt_tgt = max(30.0, min(alt_tgt,
1000.0))

                    tgt_lat, tgt_lon =
ned_offsets_to_global(
                        target_north_m, target_east_m,
                        ap_state["lat"], ap_state["lon"]
                    )

                    if tgt_lat is not None and tgt_lon is
not None:
                        if ap_state["mode"] != "GUIDED":
                            set_guided_mode()

                        now = time.time()
                        if now - last_guided_send_time >
GUIDED_SEND_PERIOD:
                            print(f"CUR :
lat={ap_state['lat']:.7f}, lon={ap_state['lon']:.7f},
alt={ap_state['alt']}")
                            print(f"TGT :
lat={tgt_lat:.7f}, lon={tgt_lon:.7f}, alt={alt_tgt:.1f},
N={target_north_m:.1f}, E={target_east_m:.1f}")
```

```
                                send_plane_guided_goto(tgt_lat,
tgt_lon, alt_tgt)
                                last_guided_send_time = now
                                print(
                                    f"GUIDED tgt:
lat={tgt_lat:.7f}, lon={tgt_lon:.7f}, "
                                    f"alt_rel={alt_tgt:.1f},
ax={ax:+.2f}, ay={ay:+.2f}, "
                                    f"mode={ap_state['mode']}"
                                )

                    frame = draw_dashed_line(
                        frame,
                        frame_center,
                        (target_x, target_y),
                        color=(0, 0, 255),
                        thickness=1,
                        dash_length=8,
                        gap_length=5,
                        alpha=0.5
                    )

                    if roi.size > 0:
                        preview_h = 160
                        preview_w = int(preview_h *
roi.shape[1] / roi.shape[0])
                        max_preview_w = 260
                        if preview_w > max_preview_w:
                            preview_w = max_preview_w
                            preview_h = int(preview_w *
roi.shape[0] / roi.shape[1])

                        preview = cv2.resize(
                            roi, (preview_w, preview_h),
                            interpolation=cv2.INTER_CUBIC
                        )

                        margin = 10
                        px = frame_width - preview_w - margin
                        py = frame_height - preview_h - margin
                        if px < 0:
                            px = 0
                        if py < 0:
                            py = 0
```

```
                    overlay = frame.copy()
                    cv2.rectangle(
                        overlay,
                        (px - 3, py - 3),
                        (px + preview_w + 3, py + preview_h
+ 3),
                        (0, 0, 0),
                        -1
                    )
                    frame = cv2.addWeighted(overlay, 0.35,
frame, 0.65, 0)

                    frame[py:py + preview_h, px:px +
preview_w] = preview

                    zoom_factor = 2.0
                    z_w = max(10, int(w / zoom_factor))
                    z_h = max(10, int(h / zoom_factor))
                    zx = target_x - z_w // 2
                    zy = target_y - z_h // 2
                    zx = max(0, min(frame_width - z_w, zx))
                    zy = max(0, min(frame_height - z_h,
zy))
                    zoom_roi = frame[zy:zy + z_h, zx:zx +
z_w]

                    if zoom_roi.size > 0:
                        mini_size = 70
                        mini = cv2.resize(
                            zoom_roi, (mini_size,
mini_size),
                            interpolation=cv2.INTER_CUBIC
                        )
                        mzx = px + preview_w - mini_size -
6
                        mzy = py + 6
                        overlay = frame.copy()
                        cv2.rectangle(
                            overlay,
                            (mzx - 2, mzy - 2),
                            (mzx + mini_size + 2, mzy +
mini_size + 2),
                            (0, 0, 0),
                            -1
```

```
                        )
                        frame = cv2.addWeighted(overlay,
0.4, frame, 0.6, 0)
                        frame[mzy:mzy + mini_size, mzx:mzx
+ mini_size] = mini
                        cv2.rectangle(
                            frame,
                            (mzx - 1, mzy - 1),
                            (mzx + mini_size + 1, mzy +
mini_size + 1),
                            (255, 255, 255),
                            1
                        )

            if not success and lost_frames > LOST_LIMIT:
                reset_tracking()

        ax_pct = int(ax * 100)
        ay_pct = int(ay * 100)

        if ap_state["has_link"]:
            ap_mode = ap_state["mode"] if ap_state["mode"] is
not None else "N/A"
            ap_alt = (
                f"{ap_state['alt']:.1f}m" if ap_state["alt"] is
not None else "--.-m"
            )
            gs = ap_state["groundspeed"]
            as_ = ap_state["airspeed"]
            ap_gs = f"{gs:.1f}m/s" if gs is not None else "--.-
m/s"
            ap_as = f"{as_:.1f}m/s" if as_ is not None else "--
.-m/s"
            ap_line1 = f"AP: LINK OK  MODE: {ap_mode}"
            ap_line2 = f"ALT: {ap_alt}  GS: {ap_gs}  AS:
{ap_as}"
        else:
            ap_line1 = "AP: NO LINK"
            ap_line2 = "ALT: --.-m  GS: --.-m/s  AS: --.-m/s"

        pitch = ap_state["pitch"]
        roll = ap_state["roll"]
        yaw = ap_state["yaw"]

        if pitch is not None and roll is not None and yaw is
```

```
not None:
        att_line = (
            f"PITCH: {pitch:+5.1f}deg  "
            f"ROLL: {roll:+5.1f}deg  "
            f"YAW: {yaw:+5.1f}deg"
        )
    else:
        att_line = "PITCH:   --.-deg  ROLL:   --.-deg  YAW:
--.-deg"

    hud_lines = [
        f"MODE HUD: {'AUTO' if autopilot_on else
'MANUAL'}",
        f"LOCK: {lock_percent:3d}%  QUAL:
{quality_percent:3d}%",
        f"AX: {ax_pct:+4d}%  AY: {ay_pct:+4d}%",
        ap_line1,
        ap_line2,
        att_line,
    ]

    hud_x0 = 10
    hud_y0 = 60
    hud_line_h = 16
    hud_font_scale = 0.45
    hud_thickness = 1

    max_w = 0
    for text in hud_lines:
        (tw, th), _ = cv2.getTextSize(
            text, cv2.FONT_HERSHEY_SIMPLEX, hud_font_scale,
hud_thickness
        )
        if tw > max_w:
            max_w = tw

    hud_box_w = max_w + 12
    hud_box_h = hud_line_h * len(hud_lines) + 16

    overlay = frame.copy()
    cv2.rectangle(
        overlay,
        (hud_x0 - 4, hud_y0 - 4),
        (hud_x0 - 4 + hud_box_w, hud_y0 - 4 + hud_box_h),
        (0, 0, 0),
```

```
            -1
        )
        frame = cv2.addWeighted(overlay, 0.5, frame, 0.5, 0)

        text_y0 = hud_y0 + hud_line_h
        for i, text in enumerate(hud_lines):
            cv2.putText(
                frame, text,
                (hud_x0, text_y0 + i * hud_line_h),
                cv2.FONT_HERSHEY_SIMPLEX,
                hud_font_scale, (0, 255, 255), hud_thickness,
cv2.LINE_AA
            )

        if debug_visible and last_good_bbox is not None and
initial_target_center is not None:
            lx, ly, lw, lh = [int(v) for v in last_good_bbox]

            if raw_target_north_m is not None and
raw_target_east_m is not None:
                raw_ne_line = (
                    f"Raw N/E: {raw_target_north_m:6.1f} m,
{raw_target_east_m:6.1f} m"
                )
            else:
                raw_ne_line = "Raw N/E: ---"

            if kalman_target_north_m is not None and
kalman_target_east_m is not None:
                kalman_ne_line = (
                    f"Kalman N/E: {kalman_target_north_m:6.1f}
m, {kalman_target_east_m:6.1f} m"
                )
            else:
                kalman_ne_line = "Kalman N/E: ---"

            debug_lines = [
                f"Last bbox: ({lx}, {ly}, {lw}, {lh})",
                f"Dist to center: {dist_to_center:.1f} px",
                f"Angle to center: {angle_to_center:.1f} deg",
                f"Dist from start: {dist_from_start:.1f} px",
                f"Angle from start: {angle_from_start:.1f}
deg",
                f"Sharpness: {quality_percent:3d} %",
                f"AX: {ax:+.2f}, AY: {ay:+.2f}",
```

```
            f"Raw center: {raw_target_x}, {raw_target_y}",
            f"Kalman center: {kalman_target_x},
{kalman_target_y}",
            f"Img dX: {cam_ang_horiz:+4.1f} deg, dY:
{cam_ang_down:+4.1f} deg",
            raw_ne_line,
            kalman_ne_line,
            f"Stagnant frames: {stagnant_frames}",
            f"Lost frames: {lost_frames}",
            f"Frame dt: {frame_dt:.3f} s",
        ]

        dbg_font_scale = 0.45
        dbg_thickness = 1
        dbg_line_h = 16

        dbg_x0 = 10
        dbg_y0 = hud_y0 + hud_box_h + 20

        dbg_max_w = 0
        for text in debug_lines:
            (tw, th), _ = cv2.getTextSize(
                text, cv2.FONT_HERSHEY_SIMPLEX,
dbg_font_scale, dbg_thickness
            )
            if tw > dbg_max_w:
                dbg_max_w = tw

        dbg_box_w = dbg_max_w + 12
        dbg_box_h = dbg_line_h * len(debug_lines) + 12

        overlay = frame.copy()
        cv2.rectangle(
            overlay,
            (dbg_x0 - 4, dbg_y0 - 4),
            (dbg_x0 - 4 + dbg_box_w, dbg_y0 - 4 +
dbg_box_h),
            (0, 0, 0),
            -1
        )
        frame = cv2.addWeighted(overlay, 0.5, frame, 0.5,
0)

        text_dbg_y0 = dbg_y0 + dbg_line_h
        for i, text in enumerate(debug_lines):
```

```
            cv2.putText(
                frame, text,
                (dbg_x0, text_dbg_y0 + i * dbg_line_h),
                cv2.FONT_HERSHEY_SIMPLEX,
                dbg_font_scale, (0, 255, 0), dbg_thickness,
cv2.LINE_AA
            )

    cv2.imshow('frame', frame)

    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        break
    elif key == ord('a'):
        autopilot_on = not autopilot_on
        print(f"AUTOPILOT {'ON' if autopilot_on else
'OFF'}")
        if autopilot_on:
            set_guided_mode()
    elif key == ord('g'):
        set_guided_mode()
    elif key == ord('d'):
        debug_visible = not debug_visible
        print(f"DEBUG {'ON' if debug_visible else 'OFF'}")
    elif key == ord('c'):
        reset_tracking()
        autopilot_on = False
        print("Tracking reset, AUTOPILOT OFF")
    elif key == ord('f'):
        fullscreen_on = not fullscreen_on
        cv2.setWindowProperty(
            'frame',
            cv2.WND_PROP_FULLSCREEN,
            cv2.WINDOW_FULLSCREEN if fullscreen_on else
cv2.WINDOW_NORMAL
        )

cap.release()
cv2.destroyAllWindows()
```