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

Радіотехнічний факультет

Кафедра прикладної радіоелектроніки

До захисту допущено:

Зав. кафедри

_____ Андрій МОВЧАНЮК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньою програмою «Інтелектуальні технології радіоелектронної техніки», за спеціальністю 172 «Телекомунікації та радіотехніка»

на тему: Опорно-поворотний пристрій наведення антени БПЛА

(проєкт)

Виконав:
студент IV курсу, групи РЕ-з21

Куций Едуард Юрійович
Прізвище, ім'я, по батькові

Керівник: асист. Лемеха Владислав Олександрович
Посада, науковий ступінь, вчене звання
Прізвище, ім'я, по батькові

Рецензент: доц., к.т.н. каф. РІ Пільтяй Степан Іванович
Посада, науковий ступінь, вчене звання,
Прізвище, ім'я, по батькові

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент Куций Едуард Юрійович

Київ – 2026року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Радіотехнічний факультет

Кафедра прикладної радіоелектроніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 172 «Телекомунікації та радіотехніка»

Освітня програма «Інтелектуальні технології радіоелектронної техніки»

ЗАТВЕРДЖУЮ

Зав. кафедри

_____Андрій МОВЧАНЮК

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студента

Куцого Едуарда Юрійовича

1.Тема проєкту «Опорно-поворотний пристрій наведення антени БПЛА»

2.Керівник проєкту асист. Лемеха Владислав Олександрович, затверджені наказом по університету від « 29 » травня 2026 р. №1989-с

Строк подання студентом проєкту 08 червня 2026 року

3. Вихідні дані до проєкту: Бортовий двовісний опорно-поворотний пристрій для БПЛА літакового типу; наведення спрямованої антени без використання GNSS. Діапазон азимуту 360 град. безперервно, кут місця -20...+95 град.; точність наведення не гірше +/-5 град.; час роботи без GNSS не менше 20 хв.; маса не більше 350 г; споживання не більше 15 Вт; інтерфейс MAVLink 2.0; робоча частота 5,8 ГГц (ISM); обов’язкова корекція наведення за метриками радіоканалу (RSSI, SNR, LQ).

4. Зміст пояснювальної записки: 1) Аналіз предметної галузі та вимоги до проєкту (огляд аналогів, вибір архітектури); 2) Дослідження поворотних механізмів та апаратної частини (вибір актуатора, конструкція PTU, виконавчий ланцюг наведення, узгодження радіочастотного тракту); 3) Практична частина реалізації наведення (математичний апарат, гібридний алгоритм INS+RF, три варіанти програмно-апаратної реалізації та їх порівняння).

6. Дата видачі завдання 14 квітня 2026 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Огляд аналогів та аналіз вимог до бортового ОПП наведення антени без GNSS	14.04.26-06.05.26	
2	Дослідження поворотних механізмів і вибір актуатора	07.05.26-10.05.26	
3	Розробка апаратної частини та виконавчого ланцюга наведення	11.05.26-18.05.26	
4	Розробка гібридного алгоритму наведення (INS + RF)	19.05.26-23.05.26	
5	Програмна реалізація алгоритму у трьох варіантах	24.05.26-29.05.26	
6	Верифікація алгоритму та аналіз граничних ситуацій	30.05.26-05.06.26	
7	Оформлення документації	06.06.26-08.06.26	

Студент _____ Едуард КУЦИЙ

Керівник _____ Владислав ЛЕМЕХА

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4	ДП.РЕз21.172.001	Завдання на дипломний проєкт	2	
2	А4	ДП.РЕ-з21.172.001.ПЗ	Пояснювальна записка	53	

				ДП.РЕз21.172.001		
	ПІБ	Підп.	Дата			
Розробн.	Куций Е.Ю.			Опорно-поворотний пристрій наведення антени БПЛА	Лист	Листів
Керівн.	Лемеха В.О.				1	1
					КПІ ім. Ігоря Сікорського Каф.ПРЕ ,Гр. РЕ-з21	
Н/контр.						
Зав.каф.	Мовчанюк А.В.					

РЕФЕРАТ

Дипломна робота: 3 розділи, 11 таблиць, 9 лістингів коду, 24 джерела, 4 додатки.

Мета роботи - дослідження та практична розробка бортового опорно-поворотного (антенно-поворотного) пристрою для безпілотного літального апарата (БПЛА) літакового типу, здатного автоматично наводити спрямовану антену на наземну станцію управління (НСУ) в умовах повної відсутності сигналів глобальних навігаційних супутникових систем (GNSS) [24].

Об'єкт дослідження - процес автоматичного наведення спрямованої антени бортового радіоканалу БПЛА на рухому наземну станцію.

Предмет дослідження - гібридний алгоритм наведення, що поєднує геометричний розрахунок за даними інерціальної навігаційної системи у системі координат NED та реактивне коригування за метриками якості радіоканалу (RSSI, SNR, LQ), а також варіанти його програмно-апаратної реалізації.

Методи дослідження: аналіз і синтез технічних рішень, порівняльний аналіз архітектур, математичне моделювання кінематики наведення, програмна реалізація та верифікація алгоритму у режимі симуляції.

Запропоновано гібридний контур наведення, у якому реактивне коригування за RF-метриками компенсує накопичення похибки інерціальної системи без GNSS. Жодне з розглянутих комерційних рішень такого активного RF-коригування не реалізує.

Практичне значення. Описано апаратну частину пристрою та виконавчий ланцюг наведення (від кута до PWM-сигналу двигуна), а алгоритм наведення реалізовано у трьох незалежних варіантах виконання (бортовий комп'ютер на Python, модуль ArduPilot мовою C++ на польотному контролері, застосунок Zephyr RTOS на мікроконтролері STM32) та обґрунтовано область застосування кожного від прототипу до серійного і сертифікованого виробу.

Ключові слова: БПЛА, опорно-поворотний пристрій, наведення антени, GNSS-незалежна навігація, MAVLink, RSSI, SNR, hill-climbing, ArduPilot, Zephyr RTOS, STM32.

ABSTRACT

The thesis addresses the design and practical implementation of an onboard antenna pan-and-tilt unit (PTU) for a fixed-wing unmanned aerial vehicle (UAV) that automatically points a directional antenna at a mobile ground control station (GCS) in a fully GNSS-denied environment.

A hybrid pointing algorithm is proposed: a geometric loop based on the inertial navigation system in the NED frame is corrected reactively by radio-link quality metrics (RSSI, SNR, LQ). The reactive loop performs hill-climbing in the antenna-angle space and compensates for inertial drift that cannot be bounded without GNSS. The hardware layer and the full actuation chain - from a commanded angle to the PWM servo pulse - are described and traced to the source code.

The same algorithm is implemented in three independent forms - a Python application on a companion computer, a C++ ArduPilot library running inside the flight controller, and a Zephyr RTOS application on a standalone STM32 microcontroller. The applicability of each form is justified.

Keywords: UAV, antenna pan-and-tilt unit, antenna pointing, GNSS-denied navigation, MAVLink, RSSI, SNR, hill-climbing, ArduPilot, Zephyr RTOS, STM32.

Пояснювальна записка
до дипломного проєкту
на тему: Опорно-поворотний пристрій наведення антени БПЛА

Київ – 2026 рік

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

Скорочення	Розшифрування
АРП	Автоматичне регулювання підсилення
АПП	Антенно-поворотний (опорно-поворотний) пристрій
БАК	Безпілотний авіаційний комплекс
БПЛА	Безпілотний літальний апарат
ДН	Діаграма направленості антени
ІВБ	Інерціальний вимірювальний блок
ІНС (INS)	Інерціальна навігаційна система
НСУ	Наземна станція управління
МЗВ	Максимальна злітна вага
ПІД	Пропорційно-інтегрально-диференціальний регулятор
РТОС (RTOS)	Операційна система реального часу
AGC	Automatic Gain Control
AHRS	Attitude and Heading Reference System - система визначення просторового положення
BLDC	Brushless DC motor - безколекторний двигун постійного струму
BOM	Bill of Materials
CC	Companion Computer - бортовий супутній комп'ютер
EKF	Extended Kalman Filter - розширений фільтр Калмана
FC	Flight Controller - польотний контролер
FOC	Field-Oriented Control - векторне керування двигуном
FPU	Floating-Point Unit - блок операцій з плаваючою комою
GCS	Ground Control Station - наземна станція управління
GNSS	Global Navigation Satellite System - глобальна навігаційна супутникова система
HPBW	Half-Power Beam Width - ширина пелюстки ДН за рівнем -3 дБ
LQ	Link Quality - відсоток успішно прийнятих пакетів радіоканалу
MAVLink	Micro Air Vehicle Link - протокол обміну даними компонентів БПЛА
NED	North-East-Down - система координат Північ-Схід-Вниз

PTU	Pan-and-Tilt Unit - двовісний механічний поворотний пристрій
PWM	Pulse-Width Modulation - широтно-імпульсна модуляція
RF	Radio Frequency - радіочастотний
RSSI	Received Signal Strength Indicator - рівень потужності прийнятого сигналу
SBC	Single-Board Computer - одноплатний комп'ютер
SNR	Signal-to-Noise Ratio - відношення сигнал/шум
SWaP	Size, Weight and Power - масогабаритні та енергетичні характеристики

ВСТУП

Надійний радіозв'язок між безпілотним літальним апаратом (БПЛА) літакового типу і наземною станцією управління (НСУ) є однією з ключових вимог до сучасних безпілотних авіаційних комплексів. Всеспрямована антена вирішує задачу зв'язку найпростішим способом, проте платить за це низьким підсиленням і вразливістю до завад. Спрямована патч-антена з вузькою діаграмою направленості дозволяє суттєво покращити дальність і завадостійкість каналу, але лише за умови, що антена постійно наведена на НСУ незалежно від маневрів БПЛА.

Особливістю задачі, що розглядається у цій роботі, є відсутність сигналів глобальних навігаційних супутникових систем (GNSS). У цих умовах БПЛА не може знати ні власних абсолютних координат, ні координат НСУ, тому алгоритм наведення повинен будуватись виключно на бортових сенсорах та на метриках якості радіоканалу.

Метою дипломної роботи є дослідження та практична розробка бортового опорно-поворотного пристрою для БПЛА літакового типу, здатного автоматично наводити спрямовану антену на НСУ в умовах відсутності GNSS, а також обґрунтування вибору обчислювальної платформи для його реалізації.

Для досягнення мети поставлено такі завдання:

1) проаналізувати вимоги до бортових АПП без GNSS та виконати огляд аналогів;
2) дослідити принципи роботи поворотних механізмів, обґрунтувати вибір актуатора та описати апаратну частину і виконавчий ланцюг наведення;
3) розробити гібридний алгоритм наведення, що поєднує інерціальний контур (INS) і контур радіочастотного автосупроводження (RF);
4) реалізувати алгоритм у трьох незалежних варіантах виконання на різних обчислювальних платформах і порівняти їх;
5) дослідити поведінку системи у граничних ситуаціях і виконати верифікацію алгоритму.

Об'єкт дослідження – процес автоматичного наведення спрямованої антени бортового радіоканалу БПЛА на рухому наземну станцію.

Предмет дослідження – гібридний алгоритм наведення та варіанти його програмно-апаратної реалізації.

Практичне значення полягає у тому, що запропонований гібридний INS+RF алгоритм та три варіанти його реалізації можуть бути безпосередньо застосовані під час створення бортових завадозахищених систем зв'язку безпілотних авіаційних комплексів, у тому числі для перспективних місій у середовищах без GNSS [23].

Робота складається з трьох розділів. У першому розділі проаналізовано предметну галузь, сформульовано вимоги до пристрою, виконано огляд аналогів і обґрунтовано гібридну архітектуру. У другому розділі досліджено поворотні механізми, обрано актуатор, описано апаратну частину, взаємодію її елементів та виконавчий ланцюг наведення з посиланням на код. Третій, практичний, розділ містить математичний апарат наведення (кватерніони, перетворення систем координат, метрику якості), три незалежні варіанти програмно-апаратної реалізації, їх порівняння та аналіз граничних ситуацій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ВИМОГИ ДО ПРОЄКТУ

1.1 Призначення і вимоги до бортового АПП

Бортовий опорно-поворотний пристрій забезпечує механічне наведення спрямованої антени у напрямку НСУ протягом усього польоту. Особливість умов застосування - відсутність GNSS – є первинним обмеженням, що визначає архітектуру всієї системи. Без GNSS алгоритм наведення повинен будуватись виключно на бортових сенсорах (ІВБ, барометр, датчик повітряної швидкості) та метриках якості радіоканалу.

Спрямована патч-антена прототипу має вузьку діаграму направленості з шириною головної пелюстки за рівнем половинної потужності (HPBW) близько 30 градусів, що відповідає приблизно одному відсотку площі сфери. Таке звуження пелюстки дає виграш у підсиленні та завадостійкості, але вимагає точного наведення: вихід осі антени за межі половини HPBW призводить до різкого падіння рівня прийнятого сигналу. Звідси випливає вимога до точності наведення на рівні кількох градусів.

Типовий сценарій застосування – політ носія літакового типу на середніх висотах з безперервним передаванням телеметрії та відеопотоку на наземну станцію, яка може бути як стаціонарною, так і рухомою (наприклад, на транспортному засобі). Антена має утримувати наведення під час розворотів, наборів і знижень, не вимагаючи від носія зміни курсу заради зв'язку. Саме ці умови визначають вимоги до діапазону осей, швидкодії приводів і допустимої похибки наведення.

За наявності GNSS задача наведення є тривіальною: знаючи власні координати та координати НСУ, бортовий обчислювач обчислює вектор напрямку і повертає антену математично. У середовищі без GNSS абсолютні координати недоступні, тому система може оцінити лише відносний напрямок, і то наближено – з двох взаємодоповнюючих джерел: інерціальної навігації (точна короткочасно, але з необмеженим дрейфом) та безпосереднього вимірювання якості радіоканалу

у різних напрямках (без накопичення похибки). Технічні вимоги до прототипу зведено у таблиці 1.1.

Вимога до точності наведення випливає з ширини головної пелюстки. Підсилення спрямованої апертурної антени наближено пов'язане з шириною пелюстки співвідношенням G ≈ 26000 / (HPBW_az * HPBW_el), де ширини задано у градусах; для пелюстки близько 30 градусів це дає підсилення порядку 14 ... 15 дБі. Відхилення осі на половину ширини пелюстки (15 градусів) знижує підсилення приблизно на 3 дБ, а на повну ширину – вже на 10 ... 12 дБ, що безпосередньо погіршує енергетику каналу. Тому допустима похибка наведення обрана на рівні кількох градусів – суттєво менше половини HPBW.

Для перевірки досяжності дальності 300 кілометрів орієнтовно оцінено енергетику радіолінії за рівнянням втрат у вільному просторі. Прийнята потужність дорівнює сумі потужності передавача та підсилень антен за вирахуванням втрат поширення:

$$P_r = P_t + G_t + G_r - FSPL, \quad FSPL = 20 \cdot lg(d) + 20 \cdot lg(f) + 32{,}44 \text{ [дБ]},$$

де дальність d береться у кілометрах, частота f – у мегагерцах. Робочу частоту обрано у діапазоні 5,8 ГГц (ISM 5,725 ... 5,875 ГГц). Вибір зумовлений вимогою вузької головної пелюстки: за фіксованої ширини пелюстки потрібна фізична апертура спрямованої антени прямо пропорційна довжині хвилі, тож чим вища частота, тим компактнішою і легшою є патч-антена тієї самої спрямованості. Для пелюстки 30 градусів і підсилення близько 15 дБі апертура на 5,8 ГГц складає лише близько 12 см, тоді як на 915 МГц вона сягала б близько 0,76 м, що є неприйнятним для бортового підвісу масою до 350 г. Саме тому 5,8 ГГц є оптимальним компромісом для реалізації вузькоспрямованого променя у масогабаритному бюджеті носія, а частота модема має бути узгоджена з цим діапазоном. Платою за вищу частоту є більші втрати у вільному просторі. За потужності передавача близько 1 Вт, підсилення бортової патч-антени близько 15 дБі, всеспрямованої антени НСУ близько 3 дБі на дальності 300 км запас за відношенням сигнал/шум становить близько 4 дБ відносно типового рівня шумів приймача, тобто приблизно на 16 дБ менше, ніж було б на 915 МГц. Цей запас додатний, але невеликий, тому

для впевненого зв'язку на гранично великій дальності передбачено підвищення потужності передавача до 2 ... 4 Вт або звуження смуги приймача телеметрії; на менших дальностях запас зростає квадратично зі зменшенням відстані. Саме точність наведення, а не енергетика, лишається визначальним чинником на гранично великих дальностях, оскільки вихід осі за межі вузької пелюстки знецінює будь-який запас потужності.

Таблиця 1.1. Технічні вимоги до прототипу АПП

Параметр	Вимога	Обґрунтування
Діапазон азимуту	360 град. безперервно	Будь-який курс БПЛА відносно НСУ
Діапазон кута місця	-20 ... +95 град.	Підфюзеляжне кріплення PTU
Точність наведення	не гірше +/-5 град.	HPBW = 30 град., зона нечутливості 12 град.
Час роботи без GNSS	не менше 20 хв.	Тактичний клас ІВБ, дрейф 3 град./год
Маса АПП	не більше 350 г	10 % від МЗВ БПЛА 3,5 кг
Споживання	не більше 15 Вт	Бюджет живлення бортової мережі
Інтерфейс	MAVLink 2.0	Сумісність з ArduPilot та PX4
Корекція за SNR/RSSI	обов'язково	Компенсація дрейфу INS без GNSS
Зона нечутливості	гістерезис 12/6 град.	Уникнення надлишкового рискання
Робоча частота	5,8 ГГц (ISM)	Вузький промінь патч-антени у бюджеті 350 г; модем і антена на одній частоті

1.2 Огляд аналогів і вибір архітектури

Аналіз комерційних рішень показав, що жодне з них не реалізує активного коригування наведення за параметрами радіоканалу. Поширені контролери підвісів BaseCam SimpleBGC 32, Storm32 BGC, а також спеціалізовані системи на кшталт UAV Navigation POLAR-500 використовують виключно підхід на основі інерціальних вимірювальних блоків (IMU-based) [14]. При накопиченні дрейфу ІВБ

похибка наведення зростає без компенсації, що для вузької пелюстки антени швидко призводить до втрати каналу. Узагальнення розглянутих рішень наведено у таблиці 1.2.

Таблиця 1.2. Огляд наявних рішень наведення антен для БПЛА

Рішення	Принцип	Корекція за RF
BaseCam SimpleBGC 32	стабілізація за IMU	немає
Storm32 BGC	стабілізація за IMU	немає
UAV Navigation POLAR-500	наведення за координатами/IMU	немає
Наземні tracker-антени (на GPS)	наведення за телеметрією GPS	немає, потребує GNSS
Запропоноване рішення	гібрид INS + RF	так (hill-climbing за Q)

Порівняльний аналіз показав, що геометричний і реактивний підходи є взаємодоповнюючими: геометричний забезпечує стійкість за відсутності сигналу, реактивний – автоматичне відстеження рухомих цілей і компенсацію дрейфу INS. Оптимальна система об'єднує обидва контури, тому для цього проєкту обрано гібридну архітектуру: інерціальний INS-контур (мова C, мікроконтролер STM32, частота 200 Гц) та RF-контур радіочастотного атосупроводження. Обидва контури взаємодіють через стандартні повідомлення MAVLink, що забезпечує модульність і незалежну роботу у разі відмови одного з контурів. Порівняння підходів наведено у таблиці 1.3.

Таблиця 1.3. Порівняння підходів до наведення без GNSS

Критерій	INS	RF	Гібридний
Робота без сигналу	Так	Ні	Так
Похибка при дрейфі INS	5 град. за 30 хв	Немає	близько 2 град. (компенс. RF)
Відстеження рухомої НСУ	потребує координат	автоматично	обидва методи
Стійкість до завад (РЕБ)	висока	середня	висока
Знос PTU	малий	високий	мінімальний

На підставі проведеного аналізу задачу сформульовано так: необхідно розробити алгоритм наведення спрямованої антени на рухому НСУ без використання GNSS, який поєднує інерціальний контур на основі INS та контур радіочастотного автосупроводження на основі метрик радіоканалу, забезпечує точність наведення не гірше +/-5 градусів і використовує стандартний протокол MAVLink 2.0, причому реалізація має допускати перенесення між обчислювальними платформами різного класу.

2 ДОСЛІДЖЕННЯ ПОВОРОТНИХ МЕХАНІЗМІВ ТА АПАРАТНОЇ ЧАСТИНИ

2.1 Порівняльний аналіз типів актуаторів

Досліджено три класи актуаторів для двовісного поворотного пристрою. Серводвигуни з PWM- або UART-керуванням прості та дешеві, проте мають люфт редуктора у межах 0,3 ... 1,5 градуса, що для вузької пелюстки є неприйнятним джерелом похибки. Крокові двигуни працюють у розімкненому контурі і за різких маневрів ризикують пропустити кроки, втративши абсолютну позицію. Безколекторні gimbal-двигуни з прямим приводом, векторним керуванням (FOC) і 14-розрядним магнітним енкодером AS5048A забезпечують нульовий механічний люфт і плавне відпрацювання малих кутових корекцій [7].

Для цього проєкту обрано саме безколекторні двигуни з прямим приводом, оскільки нульовий люфт є критичним для точності наведення при малих кутових корекціях. Контролер двигуна (наприклад, Storm32 BGC або реалізація на базі SimpleFOC) замикає контур положення за сигналом енкодера і відпрацьовує заданий кут.

Векторне керування (FOC) подає на обмотки струми так, щоб вектор магнітного поля статора залишався ортогональним до поля ротора, що дає максимальний момент на одиницю струму і плавний хід без пульсацій навіть на дуже малих швидкостях, потрібних для точного доведення осі [6, 8, 11]. Абсолютний 14-розрядний магнітний енкодер AS5048A забезпечує роздільну здатність близько 0,022 градуса на оберт, що на порядки менше за допустиму похибку наведення і знімає питання втрати позиції після вимкнення живлення. Порівняння трьох класів актуаторів зведено у таблиці 2.1.

Таблиця 2.1. Порівняння типів актуаторів для PTU

Тип	Люфт	Контур	Придатність
Серводвигун (редуктор)	0,3 ... 1,5 град	замкнений (внутр.)	обмежена - люфт
Крокoвий двигун	немає, але крок	розімкнений	ризик пропуску кроків

BLDC прямий привід + FOC	відсутній	замкнений за енкодером	оптимальна

2.2 Конструкція двовісного поворотного пристрою

Двовісний поворотний пристрій (PTU, Pan-and-Tilt Unit) складається з двох рам і двох двигунів. Зовнішня рама обертається навколо вертикальної осі (азимут, AZ) і повертає весь механізм ліворуч-праворуч. Внутрішня рама нахиляє антену навколо горизонтальної осі (кут місця, EL) – угору і вниз. На внутрішній рамі закріплено прямокутну патч-антену. Третя вісь (крен) не використовується, оскільки патч-антена симетрична відносно осі візування і крен не впливає на якість каналу.

Порядок осей обрано азимут-кут місця (азимут зовнішньою рамою, кут місця внутрішньою), що є класичним рішенням для антенних опор. Такий порядок має відому особливість – зону невизначеності поблизу зеніту (keyhole), де для відстеження цілі прямо над пристроєм потрібна дуже велика швидкість азимутальної осі. Для бортового застосування ця зона не критична, бо НСУ перебуває переважно нижче або на рівні горизонту відносно апарата, тож кут місця рідко наближається до +90 градусів. Двигуни розташовано безпосередньо на осях обертання, що зменшує момент інерції і потрібний момент приводів під час швидких маневрів носія.

2.3 Склад апаратної частини та взаємодія елементів

Апаратна частина побудована за трирівневою схемою: сенсорний рівень (ІВБ, барометр, датчик повітряної швидкості, магнітні енкодери осей PTU, радіомодем як джерело RF-метрик), обчислювальний рівень (польотний контролер і, залежно від варіанта, бортовий супутній комп'ютер або окремий мікроконтролер) та виконавчий рівень (два двигуни PTU і закріплена на ньому патч-антена). Перелік компонентів наведено у таблиці 2.2.

Таблиця 2.2. Склад апаратної частини бортового АПП

Елемент	Призначення	Приклад
Польотний контролер (FC)	AHRS, MAVLink, керування підвісом	Pixhawk 6C, Cube Orange
Бортовий комп'ютер (CC)	Виконання алгоритму (варіант 1)	Raspberry Pi 4 / Zero 2 W
Мікроконтролер (варіант 3)	Автономне виконання алгоритму	STM32F411 (NUCLEO-F411RE)
Радіомодем	Канал зв'язку і джерело RF-метрик	Цифровий канал 5,8 ГГц з MAVLink і RSSI/SNR (Doodle Labs Mesh Rider, Microhard pMDDL клас)
Двовісний PTU	Механічне наведення антени	BLDC gimbal, Storm32 BGC
Патч-антена	Спрямований радіоканал (HPBW близько 30 град.)	Патч-решітка 5,8 ГГц
Енкодери осей	Зворотний зв'язок за положенням	AS5048A, 14 розрядів
Живлення	Окремі BEC для CC і для двигунів	BEC 5 В / 3 А

Взаємодія елементів організована навколо польотного контролера як центрального вузла обміну. Радіомодем уводить кадри RADIO_STATUS у послідовний канал FC, звідки вони надходять до обчислювача алгоритму. Обчислювач, визначивши потрібні кути, надсилає команду MOUNT_CONTROL, яку FC перетворює на сигнал керування двигунами PTU. Фактичні кути повертаються повідомленнями MOUNT_STATUS, а орієнтація корпусу безперервно надходить повідомленнями ATTITUDE. Така топологія забезпечує модульність: відмова одного контуру не блокує інший, а заміна обчислювача не змінює логіки обміну. Послідовність потоку даних на кожному циклі керування така:

1) радіомодем приймає пакети від НСУ і формує RADIO_STATUS з рівнями rssi, noise, lq;

2) обчислювач агрегує метрики у єдине значення якості Q і оновлює оцінку напрямку;
3) кінцевий автомат за значенням Q визначає бажані кути у земній системі координат;
4) блок компенсації переводить кути у зв'язану систему за поточною орієнтацією (ATTITUDE);
5) команда MOUNT_CONTROL надходить на драйвер підвісу, який формує сигнал двигунів;
6) двигуни доводять осі до заданих кутів, а MOUNT_STATUS повертає фактичне положення для контролю.

2.4 Осі повороту і відповідність каналам керування

Кожна вісь PTU керується незалежно і відображається на окреме поле команди наведення. Вісь PAN (азимут, горизонтальна площина) відповідає полю input_c повідомлення MOUNT_CONTROL і має діапазон 0 ... 360 град. (через ковзний контакт або програмне загортання у межах +/-180 град.). Вісь TILT (кут місця) відповідає полю input_a і має діапазон -90 ... +90 град. (фізично часто -30 ... +90 град.). Кути передаються у центи-градусах. Відповідність зведено у таблиці 2.3.

Таблиця 2.3. Відповідність осей PTU каналам керування

Двигун	Керований кут	Поле / канал	Діапазон
PAN (азимут)	напрямок у горизонт. площині	input_c; SERVO9 (func 6)	0 ... 360 град.
TILT (кут місця)	кут над горизонтом	input_a; SERVO10 (func 7)	-90 ... +90 град.
ROLL	не використовується	input_b = 0	-

2.5 Виконавчий ланцюг: за рахунок чого і як повертається PTU

Поворот PTU – це кінець єдиного виконавчого ланцюга, який починається від бажаного напрямку на НСУ і закінчується механічним моментом на валу двигуна. Ланцюг складається з трьох ланок: формування бажаних кутів у земній системі координат (виконує кінцевий автомат, підрозділ 3.3); перетворення цих кутів у зв'язану систему координат корпусу з урахуванням орієнтації апарата; перетворення кута у фізичний сигнал керування двигуном. Усі три ланки явно реалізовано у коді кожного з трьох варіантів.

Друга ланка – компенсація положення – віднімає курс корпусу для азимуту і тангаж для кута місця, після чого загортає азимут у діапазон (-180; +180] і обмежує кут місця діапазоном осі. У коді це функція перетворення систем координат:

```
def world_to_body(world_az_deg, world_el_deg, attitude):
    yaw_deg   = math.degrees(attitude.yaw)
    pitch_deg = math.degrees(attitude.pitch)
    body_az = _wrap_deg(world_az_deg - yaw_deg)      # віднімаємо курс
    body_el = world_el_deg - pitch_deg               # віднімаємо тангаж
    body_el = max(-90.0, min(90.0, body_el))         # обмеження осі
    return body_az, body_el
```

Завдяки цій ланці корпус БПЛА ніколи не рискає для пошуку НСУ: якщо апарат повертається на +30 град. за курсом, PTU миттєво відпрацьовує -30 град. за азимутом, утримуючи промінь на тому самому напрямку у земній системі.

Третя ланка – власне поворот – реалізована по-різному залежно від варіанта. У варіантах на Python та ArduPilot програма не торкається сигналу двигуна напряму: вона лише передає кут, а перетворення кута на широтно-імпульсний сигнал (PWM) виконує драйвер підвісу всередині польотного контролера [12]. Команда формується так:

```
def send_mount_control(self, pan_deg, tilt_deg):
    # кути у центи-градусах; input_a = tilt(pitch), input_c = pan(yaw)
    self.conn.mav.mount_control_send(
        target_system, target_component,
```

```
        int(tilt_deg * 100),   # input_a (кут місця)
        0,                     # input_b (крен, не використ.)
        int(pan_deg  * 100),   # input_c (азимут)
        0)
```

Далі польотний контролер (через AP_Mount, параметри MNT1_TYPE = 1, SERVO9_FUNCTION = 6 для азимуту, SERVO10_FUNCTION = 7 для кута місця) генерує на відповідних виходах PWM-імпульси, які надходять на драйвер двигуна. Двигун обертає раму доти, доки фактичний кут за енкодером не зрівняється із заданим – це і є момент, за рахунок якого повертається механізм.

У варіанті на Zephyr PWM-сигнал формує сам мікроконтролер. Накладка devicetree для плати NUCLEO-F411RE призначає азимутний сигнал на таймер PWM3 (вивід PA6), а кут місця – на PWM4 (вивід PB6); період імпульсів 20 мс (50 Гц). Перетворення кута на тривалість імпульсу виконує метод драйвера:

```
void Servo::set_angle(float angle_deg) {
  if (_wrap_360) {                // вісь азимуту загортається у 360 град.
    float a = fmodf(angle_deg + 180.0f, 360.0f);
    if (a < 0.0f) a += 360.0f;  angle_deg = a - 180.0f;
  }
  angle_deg = clamp(angle_deg, _angle_min_deg, _angle_max_deg);
  float t = (angle_deg - _angle_min_deg) /
        (_angle_max_deg - _angle_min_deg);        // нормування 0..1
  uint32_t pulse_us = _pulse_min_us +
    (uint32_t)((_pulse_max_us - _pulse_min_us) * t);  // 1000..2000 мкс
  pwm_set(_pwm_dev, _channel, _period_ns, pulse_us * 1000U, 0);
}
```

Отже, фізичний поворот PTU відбувається за рахунок широтно-імпульсного сигналу 50 Гц із тривалістю імпульсу 1000 ... 2000 мкс, що подається на драйвер двигуна. Драйвер за цим сигналом задає опорний кут, виконує векторне керування (FOC) обмотками безколекторного двигуна і, замикаючи контур за 14-розрядним енкодером, створює момент, який доводить раму до заданого кута з нульовим

люфтом. Уся послідовність від кута до імпульсу явно присутня у коді: перетворення систем координат — у спільному ядрі pointing_core.py та його C++-порті pointing_core.hpp, формування команди наведення — в адаптері companion_mavlink.py (для бортового комп'ютера) чи в модулі ArduPilot AP_AntPoint.cpp, а безпосереднє формування PWM — у застосунку Zephyr main.c.

2.6 Електроживлення та правила монтажу

Бортовий обчислювач живиться від окремого стабілізатора 5 В / 3 А, а двигуни PTU – від окремого стабілізатора, розрахованого на піковий струм приводів (близько 2 А на двигун). Розділення шин живлення запобігає просіданню напруги обчислювача під час пускових струмів двигунів. Лінії TX/RX між модемом і польотним контролером перехрещуються, усі вузли мають спільну землю, коаксіальний кабель антени роблять якомога коротшим, а вісь азимуту повинна забезпечувати повний оберт на 360 град. через ковзний контакт або програмне загортання кута.

2.7 Узгодження радіочастотного тракту і робоча частота

Окремої уваги потребує узгодження радіочастотного тракту, оскільки контур радіочастотного автосупроводження вимірює якість каналу через ту саму антену, якою працює модем. Радіомодем як джерело RF-метрик, патч-антена, фідер і роз'єм мають належати одному частотному діапазону – у цьому проєкті 5,8 ГГц. Антена є частотно-вибірковим елементом: її резонанс задається фізичними розмірами випромінювача (довжина близько половини довжини хвилі у діелектрику), тож антена, розрахована на інший діапазон, для частоти модема виявляється сильно неузгодженою, прийнятого сигналу фактично немає, а метрика Q втрачає сенс. Тому вимога однакової частоти модема й антени є не рекомендацією, а умовою працездатності всього GNSS-незалежного контуру [20]. Узгодження тракту контролюється за коефіцієнтом відбиття S11 нижче -10 дБ (КСХ менше 2) на робочій частоті при хвильовому опорі 50 Ом.

З цього випливає важлива властивість архітектури: робоча частота є спільним параметром модема й антени, який можна змінювати лише синхронно. Перехід в

інший діапазон означає одночасну заміну обох компонентів узгодженою парою (модем, антена та фідер разом), а не підстроювання одного під інший. Така синхронна заміна утворює модульність на рівні апаратного радіочастотного блоку: front-end виступає окремим змінним модулем, тоді як решта системи лишається незмінною. Принципово, що алгоритм наведення від частоти не залежить зовсім – він оперує нормованою метрикою Q, у якій абсолютний рівень сигналу прибрано автоматичним регулюванням підсилення, тому той самий програмний код працює на будь-якому діапазоні без змін. Саме програмний шар є по-справжньому переносимим і модульним, а апаратний радіочастотний тракт – модульним лише у сенсі заміни цілісної узгодженої пари.

Водночас слід відверто визнати обмеження такого підходу: робоча частота не є програмованим параметром. На відміну від порогів автомата, ваг метрики чи періодів пробних відхилень, які задаються параметрами ArduPilot або опціями Kconfig і змінюються без переробки заліза, резонансна частота патч-антени зафіксована у металі та діелектрику конструкції. Її неможливо перелаштувати під час польоту чи навіть між польотами без фізичної заміни антени. Через це система позбавлена частотної адаптивності: вона не може у відповідь на постановку завад перейти у вільний від перешкод діапазон, що було б цінним для роботи в умовах радіоелектронної боротьби. Це є платою за простоту, малу масу й низьку вартість фіксованого патч-рішення, і про цю відсутність гнучкості варто пам'ятати при оцінюванні завадозахищеності системи.

Часткова адаптивність, проте, досяжна вже у поточній архітектурі. У межах обраного діапазону 5,8 ГГц цифровий модем підтримує канальну гнучкість і псевдовипадкову зміну частоти (frequency hopping), що дозволяє оминати завантажені чи навмисно заглушені піднесучі без жодної зміни антени, оскільки вузькосмугове перестроювання у межах смуги антени лишається в зоні її узгодження. Повна ж частотна адаптивність, тобто перехід між діапазонами, потребує заміни як модема на програмно-визначуваний радіоприймач (SDR) з широким діапазоном перестроювання, так і антени на широкосмугову або реконфігуровану (наприклад, патч із підстроюванням резонансу варакторними чи

PIN-діодними елементами). Логічним продовженням цього шляху є перехід від механічного ротатора до фазованої антенної решітки з електронним керуванням променем, для якої на 5,8 ГГц крок елементів (близько половини довжини хвилі, тобто близько 2,6 см) є цілком реалізовним; саме там залучення програмованої логіки (FPGA) стає виправданим. Такий повністю адаптивний тракт принципово можливий, але коштує суттєвого зростання складності, споживання й масогабаритних показників, тому для прототипу обрано фіксований узгоджений тракт, а адаптивність винесено у напрями подальшого розвитку.

3 ПРАКТИЧНА ЧАСТИНА РЕАЛІЗАЦІЇ НАВЕДЕННЯ

Практична частина охоплює обидва контури гібридного алгоритму – інерціальний (на основі сенсорів) і радіочастотного автосупрводження (на основі радіочастотних метрик), їх поєднання у кінцевому автоматі, а також три незалежні варіанти програмно-апаратної реалізації одного й того самого алгоритму.

3.1 Реалізація розрахунку, що базований на сенсорах

Інерціальний контур реалізує класичну послідовність числення шляху (dead reckoning) [1] і будується на двох операціях: оцінюванні просторової орієнтації корпусу за показами інерціального блоку та перетворенні бажаного напрямку на НСУ із земної системи координат у зв'язану систему координат PTU. Нижче ці операції розписано через кватерніони і матриці напрямних косинусів, оскільки саме вони складають математичну основу контуру.

3.1.1 Системи координат

Використано дві праві ортогональні системи. Земна система NED (n) має осі: x на північ, y на схід, z униз. Зв'язана з корпусом система (b) має осі: x уперед (за віссю фюзеляжу), y на праве крило, z униз. Орієнтація корпусу - це поворот, що переводить систему n у систему b; його зручно задавати кутами Ейлера (крен phi, тангаж theta, курс psi) за послідовністю поворотів 3-2-1 (спершу навколо z на psi, потім навколо y на theta, потім навколо x на phi) або одиничним кватерніоном [5, 23].

3.1.2 Кватерніон орієнтації та його кінематика

Орієнтацію подано одиничним кватерніоном q = q0 + q1 i + q2 j + q3 k, для якого виконується умова нормування:

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1 \qquad (3.1)$$

де q_0 — скалярна частина кватерніона; q_1 , q_2 , q_3 — його векторні складові.

Перевага кватерніона перед кутами Ейлера - відсутність виродження (gimbal lock) при тангажі близько 90 градусів і менша обчислювальна вартість інтегрування

[1]. Зміна кватерніона у часі під дією кутової швидкості корпусу omega_b = (p, q, r), виміряної гіроскопами, описується кінематичним рівнянням:

$$\dot{q} = \frac{1}{2} q \otimes [0, \omega_b]^T \quad (3.2)$$

де знак ⊗ позначає добуток кватерніонів, а [0, omega_b] - чисто векторний кватерніон, складений з компонент кутової швидкості. У дискретному вигляді на кроці dt кватерніон інтегрується і перенормовується, що і виконує модуль числення.

3.1.3 Комплементарний фільтр Магоні (AHRS)

Інтегрування лише гіроскопів накопичує дрейф, тому застосовано нелінійний комплементарний фільтр Магоні на групі SO(3) [2, 3]. Він порівнює напрямок виміряного вектора прискорення вільного падіння (а за наявності магнітометра – і вектора магнітного поля) з тим, який передбачає поточна оцінка орієнтації, і формує вектор корекції як векторний добуток виміряного та оціненого ортів:

$$e = v_{meas} \times v_{est} \quad (3.3)$$

після чого кутова швидкість, що подається на інтегратор, коригується пропорційно-інтегральним законом:

$$\omega_{corr} = \omega_{gyro} + K_p \cdot e + K_i \cdot \int e \, dt \quad (3.4)$$

Пропорційна складова прибирає високочастотну похибку, інтегральна – компенсує сталий дрейф гіроскопів. Саме цей фільтр (клас MahonyAHRS у спільному ядрі pointing_core) дає стійку оцінку кватерніона орієнтації на частоті 200 Гц.

3.1.4 Матриця напрямних косинусів

Для перетворення векторів між системами координат з кватерніона будується матриця напрямних косинусів (DCM) [23]. Матриця C_n^b переводить вектор із земної системи у зв'язану і має вигляд:

$$C_n^b = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 + q_0 q_3) & 2(q_1 q_3 - q_0 q_2) \\ 2(q_1 q_2 - q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 + q_0 q_1) \\ 2(q_1 q_3 + q_0 q_2) & 2(q_2 q_3 - q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

(3.5)

Зворотне перетворення (із зв'язаної у земну) виконує транспонована матриця C_b^n = (C_n^b)ᵀ, оскільки DCM ортогональна. За потреби кути Ейлера відновлюються з кватерніона співвідношеннями:

$$\varphi = \operatorname{atan2}\big(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)\big)$$
$$\theta = \operatorname{asin}\big(2(q_0 q_2 - q_3 q_1)\big) \qquad (3.6)$$
$$\psi = \operatorname{atan2}\big(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2)\big)$$

3.1.5 Розрахунок кутів наведення і перетворення координат

Нехай оцінений напрямок на НСУ у земній системі заданий ортом r^n (його дає або геометричне числення, або контур радіочастотного автосупроводження). Бажані кути наведення у земній системі - азимут і кут місця - обчислюються як:

$$\mathrm{Az} = \operatorname{atan2}(r_{\mathrm{E}}^{\mathrm{n}}, r_{\mathrm{N}}^{\mathrm{n}}), \qquad \mathrm{El} = \operatorname{atan2}\big(-r_{\mathrm{D}}^{\mathrm{n}}, \sqrt{(r_{\mathrm{N}}^{\mathrm{n}})^2 + (r_{\mathrm{E}}^{\mathrm{n}})^2}\big) \qquad (3.7)$$

де $r_{\mathrm{N}}^{\mathrm{n}}, r_{\mathrm{E}}^{\mathrm{n}}, r_{\mathrm{D}}^{\mathrm{n}}$ — проєкції орта напрямку на НСУ на осі North, East, Down земної системи координат.

Щоб антена була наведена на цей напрямок незалежно від орієнтації корпусу, орт переводять у зв'язану систему множенням на DCM, а вже з нього обчислюють кути осей PTU (азимут осі pan і кут місця осі tilt):

$$r^{\mathrm{b}} = C_{\mathrm{n}}^{\mathrm{b}} \cdot r^{\mathrm{n}} \qquad (3.8)$$

$$\mathrm{pan} = \operatorname{atan2}\big(r_{\mathrm{y}}^{\mathrm{b}}, r_{\mathrm{x}}^{\mathrm{b}}\big), \qquad \mathrm{tilt} = \operatorname{atan2}\left(-r_{\mathrm{z}}^{\mathrm{b}}, \sqrt{(r_{\mathrm{x}}^{\mathrm{b}})^2 + \big(r_{\mathrm{y}}^{\mathrm{b}}\big)^2}\right) \qquad (3.9)$$

Це і є повне перетворення систем координат. У бортовому коді для зменшення обчислень застосовано його лінеаризований еквівалент для малих кутів крену, у якому азимут осі виходить відніманням курсу, а кут місця - відніманням тангажа (pan ≈ Az - psi, tilt ≈ El - theta), що повністю узгоджується з функцією world_to_body з підрозділу 2.5. Отриманий азимут осі загортається у діапазон (-180; +180], а кут місця обмежується фізичним діапазоном осі.

3.1.6 Числення швидкості та положення

Маючи оцінку орієнтації, прискорення з акселерометрів переводять із зв'язаної у земну систему і віднімають вектор сили тяжіння, після чого інтегруванням отримують швидкість і переміщення:

$$a^{\mathrm{n}} = C_{\mathrm{b}}^{\mathrm{n}} \cdot a^{\mathrm{b}} - g^{\mathrm{n}}, \qquad v^{\mathrm{n}} = \int a^{\mathrm{n}}\,\mathrm{d}t\,, \qquad p^{\mathrm{n}} = \int v^{\mathrm{n}}\,\mathrm{d}t \quad (3.10)$$

Це числення шляху дає відносне переміщення апарата за відсутності GNSS і використовується для випереджувальної (feed-forward) оцінки швидкості зміни напрямку на НСУ. Через подвійне інтегрування похибок акселерометра координатна оцінка дрейфує приблизно квадратично за часом, тому абсолютну прив'язку періодично уточнює RF-контур радіочастотного автосупроводження.

Програмний стек інерціального контуру оформлено як апаратно-незалежне ядро pointing_core (Python-версія pointing_core.py та ідентичний за результатами C++-порт pointing_core.hpp): клас MahonyAHRS реалізує фільтр AHRS, а клас PointingController — числення шляху, матрицю C_n^b і кути PTU, SNR-контур із зоною нечутливості, ПІД-корекцію [13] та кінцевий автомат; задачу реального часу на мікроконтролері STM32 виконує адаптер застосунку Zephyr (main.c). Контур працює на частоті 200 Гц, що значно перевищує темп оновлення RF-метрик і забезпечує плавне відпрацювання маневрів. Ключова перевага контуру - він працює навіть за тимчасової відсутності радіосигналу, утримуючи антену на останньому відомому напрямку; ключова вада - похибка оцінки орієнтації необмежено зростає у часі через дрейф ІВБ. Саме цю ваду усуває контур радіочастотного автосупроводження.

3.2 Реалізація наведення за радіочастотними метриками

Контур радіочастотного автосупроводження реалізує принципово інший підхід – наведення безпосередньо за якістю радіоканалу (reactive RF-metric-driven beam steering). Замість обчислення положення НСУ система вимірює якість каналу у різних напрямках і рухає антену туди, де сигнал кращий. Це алгоритм підйому на пагорб (hill-climbing) у просторі кутів антени.

Інтегральна метрика якості Q обчислюється як зважена сума трьох нормованих складових – рівня прийнятого сигналу RSSI, відношення сигнал/шум SNR та відсотка успішно прийнятих пакетів LQ:

$$Q = w_1 \cdot \mathrm{RSSI}_{\mathrm{norm}} + w_2 \cdot \mathrm{SNR}_{\mathrm{norm}} + w_3 \cdot \mathrm{LQ} \qquad (3.11)$$

де кожна складова нормована у діапазон [0; 1], а типові вагові коефіцієнти w1 = 0,3, w2 = 0,4, w3 = 0,3. Більша вага надається SNR, оскільки у завадовій обстановці він є надійнішим показником, ніж сирий RSSI [15, 20]. Складові згладжуються ковзним вікном, що пригнічує шум одиничних пакетів.

Сирі значення з повідомлення RADIO_STATUS приходять у власних одиницях радіомодема, тому перетворення у фізичні величини є модемозалежним і узгоджується з обраним радіоканалом. Для застарілих модемів сімейства SiK діапазону 915 МГц (наприклад, RFD900x) перетворення у децибел-міліват виконувалося співвідношенням rssi[дБм] = raw/1,9 - 127; обраний для прототипу цифровий канал діапазону 5,8 ГГц подає RSSI та SNR безпосередньо у дБм і дБ через інтерфейс стану лінії, який бортовий обчислювач відображає у те саме поле метрик. Незалежно від модема відношення сигнал/шум обчислюється як різниця рівнів сигналу і шуму, а кожна складова масштабується до діапазону [0; 1] лінійним нормуванням між нижньою і верхньою межами:

$$x_{\mathrm{norm}} = \mathrm{clip}\left(\frac{(x - x_{\mathrm{min}})}{(x_{\mathrm{max}} - x_{\mathrm{min}})}, 0, 1\right) \qquad (3.12)$$

де x — поточне значення метрики; $x_{\mathrm{min}}, x_{\mathrm{max}}$ — нижня та верхня межі діапазону нормування; clip — операція обмеження результату відрізком [0; 1].

Для рівня сигналу взято межі від -90 до -30 дБм, для SNR - від 0 до 28 дБ; LQ обчислюється безпосередньо як частка успішно прийнятих пакетів у ковзному вікні. Така нормалізація відіграє роль автоматичного регулювання підсилення (AGC): метрика Q відображає переважно якість наведення, а не абсолютну дальність, що важливо для роботи в широкому діапазоні відстаней – від кількох кілометрів до сотень кілометрів.

Алгоритм підйому на пагорб працює з градієнтом метрики за кутом. Оскільки поблизу максимуму діаграма направленості апроксимується параболою $G(\Delta) \approx$

$G_{\max} - 12 \cdot \left(\frac{\Delta}{\mathrm{HPBW}}\right)^2$, метрика має виражений пік при нульовому відхиленні Δ і монотонно спадає при його зростанні. Контур оцінює знак похідної $\frac{\mathrm{d}Q}{\mathrm{d}\theta}$ скінченною різницею: він зміщує вісь на пробний крок і порівнює нову якість з попередньою.

$$\Delta Q = Q(\theta + \delta) - Q(\theta) \qquad (3.13)$$

Якщо $\Delta Q > 0$ — вісь рухається у напрямку пробного кроку δ, інакше — у протилежний бік.

Початкова оцінка напрямку отримується пусковим скануванням: на старті PTU виконує повний оберт за азимутом на 360 градусів, фіксуючи Q кожні 10 градусів; кут з максимальним Q дає початкову оцінку з невизначеністю близько +/-30 градусів. У режимі стеження контур періодично робить пробні відхилення +/-5 градусів і відстежує пік за наведеним правилом, що дозволяє безперервно супроводжувати навіть рухому НСУ, не накопичуючи похибки. Поблизу самого піка градієнт малий, тому вісь дрібно коливається навколо оптимуму (характерний для конічного сканування дизер), що добре видно на результатах моделювання у підрозділі 3.11. Цей контур реалізовано мовою Python (модулі metrics, estimator, state_machine), а у портах – відповідними модулями мовою C++.

3.3 Гібридне поєднання контурів і кінцевий автомат

Гібридна архітектура поєднує обидва контури. Інерціальний контур дає швидку випереджувальну (feed-forward) компенсацію руху корпусу: щойно апарат змінює тангаж, крен чи курс, PTU негайно повертається у протилежний бік, утримуючи промінь на тому самому напрямку у земній системі координат, ще до того, як Q встигне погіршитись. Контур радіочастотного автосупроводження повільніше, але без накопичення похибки уточнює абсолютний напрямок і компенсує дрейф INS. Принциповим обмеженням є заборона на рискання корпусу: для пошуку НСУ корпус БПЛА ніколи не повертається, рухаються лише дві осі PTU.

Для обміну даними між компонентами обрано MAVLink 2.0 - відкритий, широко підтримуваний протокол екосистем ArduPilot і PX4 [4, 12, 16]. Він надає всі потрібні повідомлення без розробки власних драйверів (таблиця 3.1).

Таблиця 3.1. Повідомлення MAVLink, що використовуються

Повідомлення	Напрям	Призначення
HEARTBEAT (#0)	НСУ <-> БПЛА	Виявлення присутності та стану вузлів
RADIO_STATUS (#109)	модем -> СС	Джерело RF-метрик: rssi, remrssi, noise, rxerrors
MOUNT_CONTROL (#157)	СС -> FC -> PTU	Команда кутів повороту (центи-градуси)
MOUNT_STATUS (#158)	PTU -> FC -> СС	Зчитування фактичних кутів PTU
MOUNT_CONFIGURE (#156)	СС -> FC	Перемикання підвісу у режим MAVLink-наведення
ATTITUDE (#30)	FC -> СС	Орієнтація корпусу для компенсації
STATUSTEXT (#253)	СС -> FC -> НСУ	Журналювання змін станів автомата

Логіка наведення оформлена як кінцевий автомат із шести станів. У стані BOOT перевіряється готовність радіо і справність приводів. У стані SCAN виконується повний оберт з пошуком піка Q. У стані TRACK напрямок утримується, а пробні відхилення відстежують пік. За погіршення Q автомат переходить до SEARCH_NARROW (коливання +/-15 градусів навколо останнього напрямку), потім до SEARCH_WIDE (+/-90 градусів) і за потреби повторює повне сканування. За повної втрати зв'язку понад заданий час автомат переходить у FAILSAFE: центрує PTU і передає керування штатному захисту польотного контролера. Порогові параметри автомата наведено у таблиці 3.2.

Таблиця 3.2. Порогові параметри автомата (типові значення)

Параметр	Значення	Призначення
Q_high	0,75	Зв'язок стабільний, дій не потрібно

Q_low	0,45	Зв'язок граничний, запуск SEARCH_NARROW
Q_critical	0,20	Канал майже втрачено, SEARCH_WIDE
t_low	3 с	Витримка перед ескалацією з TRACK
t_failsafe	10 с	Час до переходу у FAILSAFE
probe_step	5 град.	Крок пробного відхилення у TRACK
probe_period	2 с	Період пробних відхилень
delta_narrow / delta_wide	15 / 90 град.	Півширина пошуку у режимах SEARCH

Послідовність ініціалізації зв'язку відбувається у чіткому порядку. Бортовий обчислювач під'єднується до польотного контролера послідовним каналом і встановлює швидкість обміну; надсилає повідомлення MOUNT_CONFIGURE, що переводить підвіс у режим наведення за MAVLink; підписується на потоки RADIO_STATUS, MOUNT_STATUS та ATTITUDE із заданою частотою; після підтвердження потоків запускає кінцевий автомат у стані BOOT. Така послідовність гарантує, що до початку наведення усі джерела даних активні, а підвіс приймає кутові команди.

Переходи між станами визначаються поточним значенням Q і часовими витримками. З BOOT за готовності радіо і приводів автомат переходить у SCAN; із SCAN після завершення повного оберту – у TRACK (якщо знайдено достатній пік) або у SEARCH_WIDE. Зі стану TRACK за падіння Q нижче Q_low довше за t_low автомат переходить у SEARCH_NARROW, звідти за подальшого погіршення – у SEARCH_WIDE, а далі – у повторний SCAN. Відновлення Q повертає автомат у TRACK. Повна втрата зв'язку довше за t_failsafe переводить систему у FAILSAFE, де PTU центрується і керування передається штатному захисту польотного контролера. Завдяки витримкам і гістерезису між Q_high і Q_low автомат не реагує на короткочасні провали сигналу і не входить у надлишкові коливання.

3.4 Обґрунтування вибору обчислювальної платформи

Перш ніж розглядати конкретні реалізації, важливо чесно оцінити, який клас платформи взагалі потрібен. Обчислювальне навантаження алгоритму незначне: близько 5 повідомлень RADIO_STATUS на секунду, до 20 повідомлень ATTITUDE на секунду, контур керування близько 10 Гц і кілька операцій з плаваючою комою на крок. Тут немає ні цифрової обробки сигналів, ні жорстких часових обмежень жорсткіших за десятки мілісекунд, ні високошвидкісного введення-виведення. Одноплатний комп'ютер класу Raspberry Pi Zero 2 W витрачає на таке навантаження менше двох відсотків процесорного часу, а жорстку роботу реального часу (генерація PWM, опитування ІВБ на 1 кГц, фільтр Калмана) виконує польотний контролер [10].

Звідси випливає висновок: для прототипу, дослідної платформи або малосерійного виробу мова Python на Linux є цілком виправданим вибором. Водночас існують реальні підстави перейти до C++, мікроконтролера, спеціалізованої операційної системи чи навіть програмованої логіки. Таблиця 3.3 систематизує ці підстави.

Таблиця 3.3. Коли доцільно змінювати платформу

Вимога	Доцільне рішення
Сертифікація (DO-178C, MIL-STD, ASIL)	C/C++ на RTOS (NuttX, Zephyr, VxWorks); Python не сертифікується [22]
Серійне виробництво, важлива вартість BOM	Відмова від окремого SBC, порт C++ на вільне ядро FC або на STM32/ESP32; економія близько 35 дол. і кількох ват
Закриті, відтворювані, підписані образи ОС	Yocto / Buildroot замість Raspberry Pi OS; мінімальний аудитований образ [18, 21]
Реакція на радіоподії швидше за 1 мс	Bare-metal MCU або FPGA; затримки збирача сміття та планувальника Linux роблять субмілісекундну реакцію ненадійною

Фазована антенна решітка замість ротатора	FPGA або DSP; обробка комплексної основної смуги у реальному часі на частотах МГц
Захищена від злому, шифрована прошивка	MCU з захищеним завантаженням (STM32 з TrustZone) і підписаною прошивкою
Жорсткі вимоги до EMI та вібрації	Промисловий SBC або власна плата MCU замість споживчого Pi

Необхідно наголосити для чого жоден із цих засобів не потрібен. Програмована логіка (FPGA) для цього алгоритму недоцільна: тут немає паралельної математики чи високошвидкісної вибірки, а підйом на пагорб за метрикою якості з частотою 5 Гц не є тим навантаженням, де FPGA дає виграш. Так само недоцільно піднімати окрему прошивку RTOS на додатковому мікроконтролері лише заради кінцевого автомата – це додає тижні роботи і нічого не дає порівняно з виконанням тієї самої логіки на наявному польотному контролері.

Реалістичний шлях розвитку виробу – це послідовність етапів. Спочатку алгоритм відпрацьовується мовою Python на Raspberry Pi: доводиться працездатність і налаштовуються пороги у польоті. Під час продуктизації ключові модулі переписуються мовою C++ і виконуються або як модуль ArduPilot на самому польотному контролері без додаткового заліза, або на невеликому співпроцесорі Cortex-M4. Якщо Linux-обчислювач залишається, споживчу Raspberry Pi OS замінюють образом Yocto: мінімальна, доступна лише для читання, підписана коренева ФС з автоматичним відкотом. У разі переходу до фазованої антенної решітки механічний ротатор замінюють електронним керуванням променем, і ось тоді залучення FPGA стає виправданим. Саме цей аналіз і визначив три варіанти реалізації, описані далі.

3.5 Реалізація мовою Python на бортовому комп'ютері

Перший варіант реалізує RF-контур радіочастотного автосупроводження повністю мовою Python для бортового супутнього комп'ютера (Raspberry Pi). Це

еталонна реалізація алгоритму, призначена для швидкого ітерування і налагодження. Програмний пакет складається з апаратно-незалежного ядра pointing_core.py (клас MahonyAHRS — оцінка орієнтації; клас PointingController — нормалізація та фільтрація метрики Q, імовірнісна оцінка напрямку, геометричний контур і кінцевий автомат), адаптера введення-виведення MAVLink companion_mavlink.py та засобів симуляції без обладнання simulate_pointing.py й interactive_pointing.py.

Архітектурно реалізацію розбито на шість шарів: телеметрія MAVLink (приймання RADIO_STATUS, ATTITUDE, MOUNT_STATUS), агрегатор RF-метрик (нормалізація і згладжування Q), оцінювач напрямку, контролер наведення (кінцевий автомат), драйвер ротатора (формування MOUNT_CONTROL) і менеджер відмов та відновлення. Окремий модуль-симулятор генерує синтетичний потік RADIO_STATUS із заданою залежністю Q від кута, що дозволяє перевіряти переходи автомата без жодного обладнання та слугував основою імітаційної моделі підрозділу 3.10.

Перевага підходу – найшвидше ітерування з усіх трьох: контур керування працює на 10 Гц, а процесор з частотою близько 1 ГГц має понад 99 відсотків запасу; зміни запускаються однією командою без перекомпіляції. Зрілість бібліотек pymavlink і pyserial та режим симуляції додатково прискорюють розробку [4, 19]. Недоліки – додаткова вартість і маса супутнього комп'ютера, споживання близько 3 ... 5 Вт, окремий режим відмови (пошкодження SD-картки) і неможливість сертифікації інтерпретованого коду. Тому варіант оптимальний для етапу прототипу і дослідної експлуатації.

3.6 Реалізація мовою C++ як модуль ArduPilot

Другий варіант – повна переписка алгоритму мовою C++ у вигляді бібліотеки ArduPilot, що виконується всередині прошивки польотного контролера. Супутній комп'ютер не потрібен зовсім. Бібліотека (близько 530 рядків: головний клас, кінцевий автомат, агрегатор метрик, модуль перетворення систем координат) оформлена за зразком штатних бібліотек ArduPilot, вбудовується у дерево вихідних

кодів, реєструється у планувальнику і збирається системою waf. Сам алгоритм незмінний; змінюються лише джерела даних і шлях виведення: обробник RADIO_STATUS реалізовано у GCS_MAVLINK, орієнтація береться напряму з AHRS, а команди ротатора йдуть через AP_Mount [12].

Інтеграція з апстрімом виконується акуратно: бібліотека додається у каталог libraries, реєструється у таблиці планувальника апарата з частотою 10 Гц і отримує власні параметри (префікс ABT_) через стандартну підсистему параметрів ArduPilot, тож налаштування порогів і ваг доступне з наземної станції без перезбирання. Перехоплення повідомлень RADIO_STATUS реалізовано у спільному коді GCS_MAVLINK, що працює для будь-якого каналу телеметрії. Це робить рішення придатним і для PX4 за умови портування шару доступу до підвісу.

Перевагами є відсутність супутнього комп'ютера (економія 35 ... 100 доларів, 3 ... 5 Вт і одного режиму відмови), жорстке планування реального часу на NuttX або ChibiOS, прямий доступ до AHRS і драйвера підвісу, сертифіковний інструментарій (сумісність з DO-178C / DO-254), єдиний бінарний файл і менша поверхня атаки. Недоліками є повільніше ітерування (збирання і прошивання на кожну зміну), обмеженість журналювання та необхідність супроводжувати власну гілку ArduPilot. Варіант правильний за формою для серійного виробництва, сертифікованої авіоніки і мінімальної вартості BOM.

3.7 Реалізація на Zephyr RTOS на мікроконтролері STM32

Третій варіант – застосунок на операційній системі реального часу Zephyr, що виконує алгоритм на невеликому мікроконтролері (типовий пристрій – плата NUCLEO-F411RE на ядрі Cortex-M4F). Цей варіант не залежить від ArduPilot: мікроконтролер спілкується з сумісним з MAVLink модемом діапазону 5,8 ГГц напряму через MAVLink і керує двома сервоприводами через апаратні канали PWM. Код складається з головного модуля, агрегатора метрик, автомата, перетворення координат, мінімального парсера MAVLink, драйвера сервоприводів. Вибір Zephyr, а не bare-metal чи FreeRTOS, зумовлений наявністю HAL-драйверів,

опису апаратури через devicetree, системи збирання west/CMake та асинхронного UART-API в одному дереві.

Конфігурування виконується через Kconfig (prj.conf): пороги Q, витримки, крок і період пробних відхилень, частота контуру і вибір джерела орієнтації задаються опціями CONFIG_ABT_*, а прив'язка виводів PWM і UART – накладкою devicetree для конкретної плати. За замовчуванням компенсація положення вимкнена (придатно для ротатора на тринозі чи малого ровера), але вмикається опцією CONFIG_ABT_USE_FC_ATTITUDE, після чого орієнтація читається з польотного контролера повідомленнями ATTITUDE через UART. Завдяки статичній моделі пам'яті та невеликому образу (близько 30 кБ флеш понад ядро Zephyr) рішення вкладається навіть у недорогі мікроконтролери [9, 17].

Перевагами є справді мінімальне обладнання (один STM32, два сервоприводи, один модем), час завантаження у мілісекунди, детермінований планувальник, низьке споживання (близько 30 мА на 100 МГц), статична модель пам'яті та переносимість між десятками сімейств MCU зміною лише накладки плати. Недоліками є відсутність інтеграції з польотним контролером (дані орієнтації потрібно отримувати окремо – через UART до FC або власний ІВБ), необхідність самостійно підтримувати парсер MAVLink і таймінги PWM, менша спільнота та обмежені засоби налагодження. Варіант доцільний для виділеного антенного модуля і для ротаторів на боці наземної станції.

3.8 Порівняльний аналіз трьох підходів

Зведене порівняння трьох варіантів наведено у таблиці 3.4. Алгоритм у всіх трьох випадках ідентичний; різниця лежить виключно у площині платформи, інструментарію і компромісів між швидкістю розробки, вартістю, споживанням і придатністю до сертифікації.

Таблиця 3.4. Порівняння трьох варіантів реалізації

Критерій	Python	ArduPilot	Zephyr
Платформа	Raspberry Pi	польотний контролер	окремий STM32

Мова	Python 3.9+	C++	C++
Окреме залізо	так (SBC)	ні	так (MCU)
Швидкість ітерування	найвища	низька	середня
Споживання, додатк.	3 ... 5 Вт	близько 0	близько 0,1 Вт
Реальний час	м'який	жорсткий	жорсткий
Сертифіковність	ні	так	так
Обсяг коду, рядків	близько 720	близько 530	близько 800

Отже, три підходи не конкурують, а покривають різні стадії життєвого циклу виробу: Python обслуговує етап дослідження і налаштування; ArduPilot C++ – продуктизацію без додаткового заліза та сертифіковані застосування; Zephyr на STM32 – мінімальний автономний антенний контролер. Спільне ядро алгоритму гарантує перенесення результатів між варіантами без переробки логіки наведення.

Важливо, що в усіх трьох варіантах ядро алгоритму – формула метрики якості, кінцевий автомат, правило підйому на пагорб і перетворення систем координат – є спільним і відрізняється лише мовою та інтерфейсами до апаратури. Це дозволяє відпрацювати логіку один раз на швидкій Python-реалізації, а потім переносити її у продуктивні варіанти, маючи впевненість, що поведінка наведення збережеться. Така спільність ядра також спрощує супровід: виправлення в алгоритмі достатньо повторити у трьох невеликих, структурно однакових модулях.

3.9 Поведінка системи у граничних ситуаціях

Для всіх трьох реалізацій визначено і проаналізовано дев'ять граничних ситуацій з описом механізму виявлення і реакції системи (таблиця 3.5).

Аналіз граничних ситуацій виконано методом систематичного перебору відмов і збурень: для кожного джерела (канал зв'язку, привід, носій, ціль, живлення, інерціальні датчики) визначено спостережувану ознаку у даних телеметрії та закладену в алгоритм реакцію. Такий підхід забезпечує передбачувану поведінку системи не лише у штатному режимі стеження, а й за непередбачених умов експлуатації.

Таблиця 3.5. Реакція системи на граничні ситуації

Ситуація	Реакція системи
1. Повна втрата зв'язку (понад 10 с)	PTU центрується на останньому напрямку; FC виконує штатний failsafe (повернення, висіння або посадка); повторне сканування кожні 30 с
2. Несправність ротатора	виявлення за розбіжністю заданого і фактичного кута понад 5 с; припинення команд, щоб не перегріти двигуни; пасивний моніторинг
3. Багатопроменеве замирання	витримка t_low = 3 с не дає реагувати на короткочасні провали і запобігає коливанням ротатора
4. Різкий маневр БПЛА	випереджувальна компенсація PTU за кватерніоном AHRS до того, як Q встигне погіршитись
5. Швидкий рух НСУ	розширення вікна пробних відхилень з 15 до 30 град. і уповільнення періоду проб з 2 до 4 с
6. Завади / постановка перешкод	зростання шуму знижує SNR, метрика Q коректно зменшує оцінку якості; обробляється як деградація каналу
7. Втрата живлення ротатора	антена лишається у поточному положенні; зупинка оновлень MOUNT_STATUS виявляється, система переходить у пасивний моніторинг
8. Накопичення дрейфу INS	періодичні вузькі сканування (кожні 60 с у TRACK) скидають оцінку напрямку за виміром Q
9. Низький заряд / просідання живлення	за повідомленням про низький заряд PTU попередньо центрується і знижується частота оновлень

Окремо зазначимо принципове обмеження, спільне для всіх реалізацій: без зовнішньої прив'язки оцінка напрямку деградує з часом і пройденою відстанню, тому тривалі місії (понад 30 хвилин, понад 10 км) накопичують суттєву кутову невизначеність. Це фундаментальне обмеження роботи без GNSS, яке пом'якшується періодичним скануванням, але не усувається повністю. Так само система без додаткового спектрального обладнання не здатна відрізнити навмисні завади від природної втрати каналу.

ВИСНОВКИ

У результаті виконання дипломної роботи вирішено задачу дослідження та практичної розробки гібридного алгоритму наведення бортового опорно-поворотного пристрою для БПЛА літакового типу в умовах повної відсутності сигналів GNSS. Основні результати такі.

1) Проаналізовано вимоги до бортових АПП без GNSS і виконано огляд аналогів. Показано, що наявні комерційні рішення використовують виключно інерціальний підхід і не реалізують активного коригування наведення за параметрами радіоканалу, через що похибка наведення необмежено зростає з накопиченням дрейфу датчиків.
2) Обґрунтовано вибір безколекторних двигунів з прямим приводом для двовісного PTU як єдиного класу актуаторів, що забезпечує нульовий механічний люфт, критичний для наведення вузької пелюстки антени.
3) Описано апаратну частину та виконавчий ланцюг наведення. Показано, що фізичний поворот PTU відбувається за рахунок широтно-імпульсного сигналу 50 Гц (тривалість імпульсу 1000 ... 2000 мкс), який через драйвер двигуна задає опорний кут, а векторне керування з замиканням контуру за енкодером доводить раму до заданого положення. Уся послідовність від кута до імпульсу присутня у коді (перетворення систем координат, формування команди MOUNT_CONTROL і безпосереднє формування PWM).
4) Розроблено гібридний алгоритм наведення, що поєднує інерціальний контур на основі INS у системі координат NED та контур радіочастотного автосупроводження на основі метрик радіоканалу (RSSI, SNR, LQ) з підйомом на пагорб у просторі кутів. Контур радіочастотного автосупроводження компенсує дрейф INS, чого не роблять відомі аналоги; це і становить наукову новизну роботи.
5) Алгоритм реалізовано у трьох незалежних варіантах (Python на бортовому комп'ютері, бібліотека C++ для ArduPilot, застосунок Zephyr RTOS на

STM32), виконано їх порівняльний аналіз і обґрунтовано область застосування кожного: Python - для прототипу і дослідної експлуатації, ArduPilot C++ - для серійного і сертифікованого виробництва без додаткового заліза, Zephyr на STM32 - для мінімального автономного антенного контролера.

6) Узгоджено радіочастотний тракт системи: робочу частоту обрано у діапазоні 5,8 ГГц і зведено в єдиний діапазон для модема (джерела RF-метрик), патч-антени та фідера, оскільки контур радіочастотного автосупроводження вимірює якість каналу саме через робочу антену, а отже спільна частота є умовою працездатності GNSS-незалежного наведення. Вузький промінь вимагає високого підсилення у компактній апертурі, тому короткохвильовий діапазон 5,8 ГГц дозволяє вкласти патч-решітку з пелюсткою близько 30 градусів у масогабаритний бюджет носія (близько 12 см проти близько 0,76 м на 915 МГц) ціною меншого, але додатного запасу енергетики на гранично великій дальності. Показано, що робоча частота є спільним апаратним параметром, який змінюється лише синхронною заміною узгодженої пари «модем плюс антена»; це утворює модульність на рівні радіочастотного блоку, тоді як алгоритм наведення лишається повністю незалежним від частоти завдяки нормованій метриці Q. Окремо відзначено принципове обмеження - відсутність частотної адаптивності фіксованого патч-рішення (резонанс антени зафіксовано конструктивно і не перелаштовується програмно); часткову адаптивність забезпечує канальна гнучкість і зміна частоти у межах діапазону, а повна частотна адаптивність потребує переходу до SDR з широкосмуговою чи реконфігурованою антеною або до фазованої решітки.

Запропонований гібридний INS+RF алгоритм та три варіанти його реалізації можуть бути безпосередньо застосовані під час створення бортових завадозахищених систем зв'язку безпілотних авіаційних комплексів літакового типу, у тому числі для перспективних місій дослідження середовищ, де GNSS відсутній, а орієнтиром слугує наземна станція управління.

Практичні рекомендації щодо впровадження такі: на етапі науково-дослідної роботи доцільно використовувати реалізацію на Python з бортовим комп'ютером для швидкого налаштування порогів і ваг метрики у польоті; для серійного виробу - порт мовою C++ як модуль ArduPilot на польотному контролері без додаткового заліза; для виділеного антенного блоку чи наземного ротатора - застосунок на Zephyr RTOS. Подальші напрями розвитку: додавання другої осі пробних відхилень за кутом місця, характеризація реальної діаграми патч-антени з калібруванням порогів Q, перенесення образу ОС бортового обчислювача на Yocto для відтворюваних і підписаних збірок, перехід до частотно-адаптивного радіочастотного тракту (програмно-визначуваний радіоприймач у поєднанні з широкосмуговою чи реконфігурованою антеною) для протидії навмисним завадам, а у разі відмови від механічного ротатора - перехід до фазованої антенної решітки з електронним керуванням променем, що природно поєднується з частотною адаптивністю на діапазоні 5,8 ГГц.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Titterton D. H., Weston J. L. Strapdown Inertial Navigation Technology. - 2nd ed. - Stevenage : IET ; Reston : AIAA, 2004. - 558 p.
2. Mahony R., Hamel T., Pflimlin J.-M. Nonlinear Complementary Filters on the Special Orthogonal Group // IEEE Transactions on Automatic Control. - 2008. - Vol. 53, No. 5. - P. 1203-1218.
3. Madgwick S. O. H., Harrison A. J. L., Vaidyanathan R. Estimation of IMU and MARG Orientation Using a Gradient Descent Algorithm // Proc. IEEE Int. Conf. on Rehabilitation Robotics (ICORR). - Zurich, 2011. - P. 1-7.
4. Meier L., Tanskanen P., Fraundorfer F., Pollefeys M. MAVLink Micro Air Vehicle Communication Protocol, version 2.0 [Електронний ресурс]. - Режим доступу: https://mavlink.io (дата звернення: 20.05.2026).
5. Sola J. Quaternion Kinematics for the Error-State Kalman Filter [Електронний ресурс]. - arXiv:1711.02508. - 2017. - Режим доступу: https://arxiv.org/abs/1711.02508.
6. Skuric A., Bank H. S., Unger R. et al. SimpleFOC: A Field-Oriented Control Library for Brushless Motors [Електронний ресурс]. - Режим доступу: https://simplefoc.com (дата звернення: 20.05.2026).
7. ams-OSRAM. AS5048A 14-Bit Rotary Magnetic Position Sensor : Datasheet. - 2023. - 42 p.
8. T-Motor. GB36-2 Gimbal Motor : Specification. - 2023. - 6 p.
9. STMicroelectronics. STM32G474xE Reference Manual RM0440 [Електронний ресурс]. - Режим доступу: https://st.com (дата звернення: 20.05.2026).
10. InvenSense / TDK. ICM-42688-P Six-Axis MEMS MotionTracking Device : Product Specification Rev. 1.7. - 2022. - 76 p.
11. Morimoto S., Sanada M., Takeda Y. High-Performance Current-Regulator-Free Drive for PMSM // IEEE Transactions on Industry Applications. - 2003. - Vol. 39, No. 2. - P. 415-422.

12. ArduPilot Development Team. ArduPilot Documentation: Mount / Gimbal Control [Електронний ресурс]. - Режим доступу: https://ardupilot.org (дата звернення: 22.05.2026).
13. Astrom K. J., Hagglund T. PID Controllers: Theory, Design, and Tuning. - 2nd ed. - Research Triangle Park : ISA, 1995. - 343 p.
14. Yermakov A., Kuznietsov O. et al. Tracking Antenna Systems for Small UAVs: A Review // Sensors (MDPI). - 2021. - Vol. 21, No. 12. - Art. 4051.
15. ExpressLRS Team. CRSF Protocol Specification [Електронний ресурс]. - Режим доступу: https://github.com/ExpressLRS (дата звернення: 22.05.2026).
16. PX4 Development Team. PX4 Autopilot User Guide: Gimbal Configuration [Електронний ресурс]. - Режим доступу: https://docs.px4.io (дата звернення: 22.05.2026).
17. Zephyr Project. Zephyr RTOS Documentation, release 3.7 [Електронний ресурс]. - Режим доступу: https://docs.zephyrproject.org (дата звернення: 24.05.2026).
18. The Yocto Project. Yocto Project Reference Manual [Електронний ресурс]. - Режим доступу: https://docs.yoctoproject.org (дата звернення: 24.05.2026).
19. Dronecode Project. pymavlink: Python MAVLink Interface Library [Електронний ресурс]. - Режим доступу: https://pypi.org/project/pymavlink (дата звернення: 24.05.2026).
20. Doodle Labs. Mesh Rider Radio (5 GHz Band) : Datasheet. - 2023. - 16 p.
21. Raspberry Pi Ltd. Raspberry Pi Zero 2 W : Product Brief. - 2021. - 12 p.
22. RTCA. DO-178C: Software Considerations in Airborne Systems and Equipment Certification. - Washington : RTCA Inc., 2011.
23. Beard R. W., McLain T. W. Small Unmanned Aircraft: Theory and Practice. - Princeton : Princeton University Press, 2012. - 320 p.
24. Groves P. D. Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems. - 2nd ed. - Boston : Artech House, 2013. - 800 p.

ДОДАТОК А

Спільне ядро алгоритму наведення (pointing_core.py)

Алгоритм наведення винесено в апаратно-незалежне ядро, яке не виконує введення/виведення. Один і той самий код використовують усі програмні варіанти та засоби перевірки, тому імітаційне моделювання валідує саме той алгоритм, що завантажується на пристрій. Ядро містить фільтр орієнтації Магоні (клас MahonyAHRS), а також інерціальний контур, реактивний перебір-спостереження та кінцевий автомат, інкапсульовані у класі PointingController.

Структура та інтерфейс:

Виклик за один такт керування: ahrs.update(gyro, acc, mag, dt) повертає оцінку орієнтації у вигляді кватерніона; ctl.update(q_est, uav_pos, nsu_pos, measured_Q, dt) повертає словник із полями pan, tilt (командні кути) та state (стан автомата). Контур радіочастотного автосупроводження реалізовано методом збурення-спостереження: ядро не обчислює якість у гіпотетичних точках, а лише читає виміряну метрику Q, як і на реальному пристрої.

Лістинг А.1 — pointing_core.py

```
"""
pointing_core.py — апаратно-незалежне ЯДРО алгоритму наведення антени БПЛА.

Це той самий алгоритм, що описаний у розділі 3 пояснювальної записки. Ядро НЕ
виконує жодного введення/виведення (немає MAVLink, GPIO, PWM). Воно лише приймає
числові входи й повертає команди кутів — тому однаковий код можна:
  * прогнати в симуляторі для валідації (simulate_pointing.py / interactive_pointing.py);
  * запустити модульні тести (test_pointing_core.py);
  * обгорнути адаптером і виконати на бортовому комп'ютері (companion_mavlink.py);
  * портувати рядок-у-рядок у C++ (pointing_core.hpp) для ArduPilot та Zephyr.

Системи координат:
  world  — ENU (x=Схід, y=Північ, z=Вгору), НСУ та БПЛА задаються у world, метри;
  body   — зв'язана з корпусом (x=вперед); матриця q_to_R(q) переводить body->world;
  антена — нульова орієнтація променя збігається з body x; кути:
           pan  — азимут навколо body z, tilt — кут місця (підняття над площиною body xy).

Контур радіочастотного автосупроводження реалізовано методом «збурення-спостереження» (perturb & observe):
ядро НЕ обчислює Q у гіпотетичних точках, а лише читає ВИМІРЯНУ метрику Q (як на
реальному пристрої) і за її приростом вирішує, куди робити наступний пробний крок.
"""
from __future__ import annotations
import math
import numpy as np

  all   = ["q mul", "q norm", "q to R", "boresight body",
           "MahonyAHRS", "PointingController",
           "STATE_BOOT", "STATE_TRACK", "STATE_SEARCH", "STATE_FAILSAFE"]

# ---------------------------- кватерніонна математика ----------------------------
def q_mul(a, b):
    w1, x1, y1, z1 = a; w2, x2, y2, z2 = b
    return np.array([w1*w2 - x1*x2 - y1*y2 - z1*z2,
                     w1*x2 + x1*w2 + y1*z2 - z1*y2,
                     w1*y2 - x1*z2 + y1*w2 + z1*x2,
                     w1*z2 + x1*y2 - y1*x2 + z1*w2])

def q_norm(q):
    return np.asarray(q, float) / np.linalg.norm(q)

def q_to_R(q):
    """Матриця повороту body->world (стовпці — осі body у world)."""
    w, x, y, z = q
    return np.array([[1-2*(y*y+z*z),   2*(x*y-z*w),   2*(x*z+y*w)],
                     [2*(x*y+z*w),   1-2*(x*x+z*z),   2*(y*z-x*w)],
```

```
                    [2*(x*z-y*w),     2*(y*z+x*w), 1-2*(x*x+y*y)]])

def boresight_body(pan, tilt):
    """Орт променя антени у body за командними кутами pan/tilt (рад)."""
    return np.array([math.cos(tilt)*math.cos(pan),
                     math.cos(tilt)*math.sin(pan),
                     math.sin(tilt)])

# --------------------------- AHRS: фільтр Магоні ---------------------------
class MahonyAHRS:
    """Комплементарний фільтр орієнтації (розділ 3.1.3)."""
    def __init__(self, kp=2.0, ki=0.1):
        self.kp, self.ki = kp, ki
        self.q = np.array([1.0, 0.0, 0.0, 0.0])
        self.bias = np.zeros(3)

    def reset(self, q=None):
        self.q = np.array([1.0, 0, 0, 0]) if q is None else q_norm(q)
        self.bias = np.zeros(3)

    def update(self, gyro, acc, mag, dt):
        """gyro [рад/с], acc, mag — вектори у body; повертає оновлений кватерніон."""
        R = q_to_R(self.q)
        v_acc = R.T @ np.array([0.0, 0.0, 1.0])     # очікуваний орт «вгору» у body
        v_mag = R.T @ np.array([0.0, 1.0, 0.0])     # очікуваний орт «північ» у body
        acc = np.asarray(acc, float); mag = np.asarray(mag, float)
        acc = acc / (np.linalg.norm(acc) + 1e-12)
        mag = mag / (np.linalg.norm(mag) + 1e-12)
        e = np.cross(acc, v_acc) + np.cross(mag, v_mag)     # e = Σ (вимір × оцінка)
        self.bias = self.bias - self.ki * e * dt
        omega = np.asarray(gyro, float) - self.bias + self.kp * e   # ω_corr
        self.q = q_norm(self.q + 0.5 * q_mul(self.q, np.array([0.0, *omega])) * dt)
        return self.q

# --------------------------- стани кінцевого автомата ---------------------------
STATE_BOOT = "BOOT"
STATE_TRACK = "TRACK"
STATE_SEARCH = "SEARCH"
STATE_FAILSAFE = "FAILSAFE"

# --------------------------- контролер наведення ---------------------------
class PointingController:
    """
    Гібридний контролер: інерціальний контур (3.1.4–3.1.5) + реактивний
    перебір-спостереження за метрикою Q (3.2) + кінцевий автомат (3.3).

    Використання за один такт:
        ctl = PointingController(hpbw_deg=20)
        res = ctl.update(q_est, uav_pos, nsu_pos, measured_Q, dt)
        pan, tilt, state = res["pan"], res["tilt"], res["state"]
    """
    def __init__(self, hpbw_deg=20.0, q_track=0.75, q_low=0.35,
                 dither_deg=0.6, failsafe_s=1.5, tilt_limits_deg=(-30.0, 95.0)):
        self.hpbw = math.radians(hpbw_deg)
        self.q_track, self.q_low = q_track, q_low
        self.dither = math.radians(dither_deg)
        self.failsafe_s = failsafe_s
        self.tilt_min = math.radians(tilt_limits_deg[0])
        self.tilt_max = math.radians(tilt_limits_deg[1])
        self.reset()

    def reset(self):
        self.dpan = 0.0; self.dtilt = 0.0           # реактивні поправки
        self._axis = 0; self._sign = [1.0, 1.0]     # стан перебору-спостереження
        self._pending = None                        # яку вісь збурили й чекаємо результат
        self._Qbefore = [0.0, 0.0]                  # Q до збурення відповідної осі
        self.state = STATE_BOOT; self._lost = 0.0

    @staticmethod
    def geometric(q_est, uav_pos, nsu_pos):
        """Геометричні кути наведення з оцінки орієнтації та лінії на НСУ."""
        los = np.asarray(nsu_pos, float) - np.asarray(uav_pos, float)
        n = np.linalg.norm(los)
        los = los / n if n > 0 else los
        r = q_to_R(q_est).T @ los                   # напрямок на НСУ у body
        pan = math.atan2(r[1], r[0])
        tilt = math.atan2(r[2], math.hypot(r[0], r[1]))
        return pan, tilt, los

    def update(self, q_est, uav_pos, nsu_pos, measured_Q, dt):
        pan_g, tilt_g, _ = self.geometric(q_est, uav_pos, nsu_pos)

        # --- кінцевий автомат за ВИМІРЯНОЮ якістю Q ---
        if measured_Q >= self.q_track:
            self.state = STATE_TRACK; self._lost = 0.0
        elif measured_Q >= self.q_low:
            self.state = STATE_SEARCH; self._lost = 0.0
        else:
            self._lost += dt
            self.state = STATE FAILSAFE if self. lost > self.failsafe s else STATE SEARCH

        # --- контур радіочастотного автосупроводження: перебір-спостереження (perturb & observe) ---
        # Збурюємо одну вісь; наступного такту читаємо ВИМІРЯНУ Q: якщо погіршилось —
        # відкочуємо крок і міняємо напрям; якщо покращилось — крок лишаємо.
        if self.state in (STATE_TRACK, STATE_SEARCH):
            if self._pending is not None:
                ax = self._pending
                if measured_Q < self._Qbefore[ax] - 1e-3:        # крок зашкодив
```

```
                if ax == 0:
                    self.dpan -= self._sign[0] * self.dither  # відкотити
                else:
                    self.dtilt -= self._sign[1] * self.dither
                self._sign[ax] *= -1.0                        # змінити напрям
            self._pending = None
        else:
            ax = self._axis; self._axis ^= 1                  # нове збурення
            self._Qbefore[ax] = measured_Q
            if ax == 0:
                self.dpan += self._sign[0] * self.dither
            else:
                self.dtilt += self._sign[1] * self.dither
            self._pending = ax
    elif self.state == STATE_FAILSAFE:
        self.dpan *= 0.9; self.dtilt *= 0.9      # плавне центрування
        self._pending = None

    pan = pan_g + self.dpan
    tilt = min(max(tilt_g + self.dtilt, self.tilt_min), self.tilt_max)
    return {"pan": pan, "tilt": tilt, "state": self.state,
            "pan_geo": pan_g, "tilt_geo": tilt_g,
            "dpan": self.dpan, "dtilt": self.dtilt}
```

ДОДАТОК Б

Реалізація на бортовому комп’ютері (Python, MAVLink)

Варіант для бортового обчислювача (Raspberry Pi / Jetson). Адаптатор companion_mavlink.py є обгорткою вводу/виводу навколо ядра: він читає телеметрію орієнтації та положення з польотного контролера по MAVLink, бере виміряну якість каналу від радіомодема, викликає ядро й надсилає команду наведення на підвіс. Сам алгоритм не дублюється.

Структура:

pointing_core.py (ядро) і companion_mavlink.py (адаптер) розміщують в одній теці на бортовому комп’ютері.

Покрокова інструкція запуску:

1. Встановити залежності: pip install pymavlink numpy.
2. З’єднати бортовий комп’ютер із польотним контролером (UART/USB) і ввімкнути потік ATTITUDE_QUATERNION та GLOBAL_POSITION_INT.
3. Задати координати наземної станції (НСУ) параметрами --nsu-lat / --nsu-lon.
4. Запустити: python companion_mavlink.py --device /dev/ttyAMA0 --baud 921600 --nsu-lat 50.45 --nsu-lon 30.52.
5. Перевірити у консолі стан автомата та кути pan/tilt; за потреби налаштувати нормування метрики Q під конкретний модем.

Лістинг Б.1 — companion_mavlink.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
companion_mavlink.py — АДАПТЕР для бортового комп'ютера (Raspberry Pi / Jetson).

Це «обгортка» вводу/виводу навколо спільного ядра pointing_core.py. Сам алгоритм
тут не дублюється — викликається PointingController/MahonyAHRS із ядра. Адаптер:
  1) читає телеметрію орієнтації та положення з польотного контролера по MAVLink;
  2) бере метрику якості Q від радіомодема (RSSI/SNR/LQ);
  3) викликає ЯДРО і отримує команди pan/tilt;
  4) надсилає команду наведення на підвіс (MOUNT_CONTROL / DO_MOUNT_CONTROL).

Залежності:  pip install pymavlink numpy
Запуск:      python companion_mavlink.py --device /dev/ttyAMA0 --baud 921600 \
                                         --nsu-lat 50.45 --nsu-lon 30.52
"""
import argparse, math, time
import numpy as np
from pymavlink import mavutil
import pointing_core as pc


def latlon_to_enu(lat, lon, alt, ref_lat, ref_lon):
    """Груба локальна ENU-проєкція відносно опорної точки (НСУ), метри."""
    R = 6378137.0
    de = math.radians(lon - ref_lon) * R * math.cos(math.radians(ref_lat))
    dn = math.radians(lat - ref_lat) * R
    return np.array([de, dn, alt])


def main():
    ap = argparse.ArgumentParser()
```

```
    ap.add_argument("--device", default="udp:127.0.0.1:14550")
    ap.add_argument("--baud", type=int, default=921600)
    ap.add_argument("--nsu-lat", type=float, required=True)
    ap.add_argument("--nsu-lon", type=float, required=True)
    ap.add_argument("--rate", type=float, default=50.0, help="частота контуру, Гц")
    args = ap.parse_args()

    m = mavutil.mavlink_connection(args.device, baud=args.baud)
    m.wait_heartbeat()
    print("Підключено до системи", m.target_system)

    ahrs = pc.MahonyAHRS(kp=2.0, ki=0.1)          # за наявності ATTITUDE_QUATERNION
    ctl = pc.PointingController(hpbw_deg=20.0, dither_deg=0.6)
    nsu = np.array([0.0, 0.0, 0.0])               # НСУ — початок локальної ENU
    uav = np.array([0.0, 0.0, 100.0])
    q_est = np.array([1.0, 0, 0, 0])
    dt = 1.0 / args.rate
    last = time.time()

    while True:
        # --- 1) телеметрія орієнтації (кватерніон прямо з ЕНС польотного контролера) ---
        msg = m.recv_match(type=["ATTITUDE_QUATERNION", "GLOBAL_POSITION_INT",
                                 "RADIO_STATUS"], blocking=False)
        if msg is not None:
            t = msg.get_type()
            if t == "ATTITUDE_QUATERNION":
                q_est = pc.q_norm([msg.q1, msg.q2, msg.q3, msg.q4])
            elif t == "GLOBAL_POSITION_INT":
                uav = latlon_to_enu(msg.lat*1e-7, msg.lon*1e-7, msg.relative_alt*1e-3,
                                    args.nsu_lat, args.nsu_lon)
            elif t == "RADIO_STATUS":
                # ВИМІРЯНА метрика якості каналу (нормування під ваш модем):
                rssi = msg.rssi / 255.0
                noise = msg.noise / 255.0
                main.measured_Q = max(0.0, min(1.0, rssi - 0.5 * noise))

        # --- 2) такт контуру з фіксованою частотою ---
        now = time.time()
        if now - last < dt:
            continue
        last = now
        Q = getattr(main, "measured_Q", 0.0)

        # --- 3) ВИКЛИК ЯДРА (той самий код, що в симуляторі й у тестах) ---
        res = ctl.update(q_est, uav, nsu, Q, dt)
        pan_deg = math.degrees(res["pan"])
        tilt_deg = math.degrees(res["tilt"])

        # --- 4) команда наведення на підвіс ---
        m.mav.command_long_send(
            m.target_system, m.target_component,
            mavutil.mavlink.MAV_CMD_DO_MOUNT_CONTROL, 0,
            tilt_deg,      # pitch (кут місця)
            0,             # roll
            pan_deg,       # yaw (азимут)
            0, 0, 0,
            mavutil.mavlink.MAV_MOUNT_MODE_MAVLINK_TARGETING)

        if int(now) != int(last - dt):
            print(f"стан={res['state']:8s} pan={pan_deg:7.1f} tilt={tilt_deg:6.1f} Q={Q:.2f}")


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

ДОДАТОК В

Реалізація як модуль ArduPilot (C++)

Ядро портовано у C++ (pointing_core.hpp) рядок-у-рядок; той самий заголовок використовують і модуль ArduPilot, і застосунок Zephyr. Коректність порту підтверджено: на детермінованому сценарії C++ та Python ядра дають байт-у-байт однаковий результат. Обгортка AP_AntPoint отримує орієнтацію від AP_AHRS і виводить кути на сервоканали підвісу.

Структура:

libraries/AP_AntPoint/ містить pointing_core.hpp, AP_AntPoint.h та AP_AntPoint.cpp.

Покрокова інтеграція, симуляція та прошивка:

1. Встановити середовище: git clone --recursive https://github.com/ArduPilot/ardupilot; Tools/environment_install/install-prereqs-ubuntu.sh -y.
2. Скопіювати теку AP_AntPoint/ у libraries/, поклавши поряд pointing_core.hpp.
3. Створити екземпляр AP_AntPoint у транспортному засобі та викликати update() зі scheduler на 50 Гц.
4. Програмна симуляція без заліза: Tools/autotest/sim_vehicle.py -v ArduPlane --console --map.
5. Прошивка: ./waf configure --board CubeOrange; ./waf plane; ./waf --target bin/arduplane --upload.

Лістинг В.1 — pointing_core.hpp (спільне ядро, C++)

```
// pointing_core.hpp — C++ ПОРТ спільного ядра алгоритму наведення антени БПЛА.
//
// Це той самий алгоритм, що й pointing_core.py, портований рядок-у-рядок.
// Заголовок не залежить від платформи (лише <cmath>, <array>), тому однаковий
// файл вмикається і в модуль ArduPilot (C++), і в застосунок Zephyr RTOS (C/C++).
// Жодного введення/виведення тут немає — лише обчислення.
//
// Системи координат — як у Python-версії:
//   world ENU (x=Схід, y=Північ, z=Вгору); body x=вперед; qToR: body->world;
//   промінь антени за нуля кутів збігається з body x; pan — азимут, tilt — кут місця.
#pragma once
#include <array>
#include <cmath>

namespace pointing {

using Vec3 = std::array<double, 3>;
using Quat = std::array<double, 4>;      // {w, x, y, z}
using Mat3 = std::array<std::array<double, 3>, 3>;

enum class State { BOOT, TRACK, SEARCH, FAILSAFE };
```

```
inline const char* state_name(State s) {
    switch (s) {
        case State::BOOT: return "BOOT";
        case State::TRACK: return "TRACK";
        case State::SEARCH: return "SEARCH";
        default: return "FAILSAFE";
    }
}

// ---------------------------- кватерніонна математика ----------------------------
inline Quat q_mul(const Quat& a, const Quat& b) {
    return {a[0]*b[0]-a[1]*b[1]-a[2]*b[2]-a[3]*b[3],
            a[0]*b[1]+a[1]*b[0]+a[2]*b[3]-a[3]*b[2],
            a[0]*b[2]-a[1]*b[3]+a[2]*b[0]+a[3]*b[1],
            a[0]*b[3]+a[1]*b[2]-a[2]*b[1]+a[3]*b[0]};
}
inline Quat q_norm(const Quat& q) {
    double n = std::sqrt(q[0]*q[0]+q[1]*q[1]+q[2]*q[2]+q[3]*q[3]);
    return {q[0]/n, q[1]/n, q[2]/n, q[3]/n};
}
inline Mat3 q_to_R(const Quat& q) {                 // body->world
    double w=q[0], x=q[1], y=q[2], z=q[3];
    return {{1-2*(y*y+z*z),   2*(x*y-z*w),   2*(x*z+y*w)},
             {2*(x*y+z*w),   1-2*(x*x+z*z),   2*(y*z-x*w)},
             {2*(x*z-y*w),     2*(y*z+x*w), 1-2*(x*x+y*y)}};
}
inline Vec3 mat_T_mul(const Mat3& R, const Vec3& v) {  // R^T * v
    return {R[0][0]*v[0]+R[1][0]*v[1]+R[2][0]*v[2],
            R[0][1]*v[0]+R[1][1]*v[1]+R[2][1]*v[2],
            R[0][2]*v[0]+R[1][2]*v[1]+R[2][2]*v[2]};
}
inline Vec3 mat_mul(const Mat3& R, const Vec3& v) {    // R * v
    return {R[0][0]*v[0]+R[0][1]*v[1]+R[0][2]*v[2],
            R[1][0]*v[0]+R[1][1]*v[1]+R[1][2]*v[2],
            R[2][0]*v[0]+R[2][1]*v[1]+R[2][2]*v[2]};
}
inline Vec3 cross(const Vec3& a, const Vec3& b) {
    return {a[1]*b[2]-a[2]*b[1], a[2]*b[0]-a[0]*b[2], a[0]*b[1]-a[1]*b[0]};
}
inline double dot(const Vec3& a, const Vec3& b) { return a[0]*b[0]+a[1]*b[1]+a[2]*b[2]; }
inline double norm3(const Vec3& a) { return std::sqrt(dot(a, a)); }
inline Vec3 boresight_body(double pan, double tilt) {
    return {std::cos(tilt)*std::cos(pan), std::cos(tilt)*std::sin(pan), std::sin(tilt)};
}

// ---------------------------- AHRS: фільтр Магоні ----------------------------
class MahonyAHRS {
public:
    Quat q{1, 0, 0, 0};
    Vec3 bias{0, 0, 0};
    double kp, ki;
    explicit MahonyAHRS(double kp_ = 2.0, double ki_ = 0.1) : kp(kp_), ki(ki_) {}
    void reset(const Quat& q0 = {1, 0, 0, 0}) { q = q_norm(q0); bias = {0, 0, 0}; }

    Quat update(Vec3 gyro, Vec3 acc, Vec3 mag, double dt) {
        Mat3 R = q_to_R(q);
        Vec3 v_acc = mat_T_mul(R, {0, 0, 1});
        Vec3 v_mag = mat_T_mul(R, {0, 1, 0});
        double na = norm3(acc) + 1e-12, nm = norm3(mag) + 1e-12;
        acc = {acc[0]/na, acc[1]/na, acc[2]/na};
        mag = {mag[0]/nm, mag[1]/nm, mag[2]/nm};
        Vec3 ea = cross(acc, v_acc), em = cross(mag, v_mag);
        Vec3 e = {ea[0]+em[0], ea[1]+em[1], ea[2]+em[2]};
        for (int i = 0; i < 3; ++i) bias[i] -= ki * e[i] * dt;
        Vec3 omega = {gyro[0]-bias[0]+kp*e[0], gyro[1]-bias[1]+kp*e[1], gyro[2]-bias[2]+kp*e[2]};
        Quat qd = q_mul(q, {0, omega[0], omega[1], omega[2]});
        for (int i = 0; i < 4; ++i) q[i] += 0.5 * qd[i] * dt;
        q = q_norm(q);
        return q;
    }
};

// ---------------------------- контролер наведення ----------------------------
struct PointResult { double pan, tilt, pan_geo, tilt_geo, dpan, dtilt; State state; };

class PointingController {
public:
    double hpbw, q_track, q_low, dither, failsafe_s, tilt_min, tilt_max;
    double dpan, dtilt;
    int axis; double sign[2]; int pending; double Qbefore[2];
    State state; double lost;

    explicit PointingController(double hpbw_deg = 20.0, double q_track_ = 0.75,
        double q_low_ = 0.35, double dither_deg = 0.6, double failsafe = 1.5,
        double tilt_min_deg = -30.0, double tilt_max_deg = 95.0)
        : hpbw(hpbw_deg*M_PI/180), q_track(q_track_), q_low(q_low_),
          dither(dither_deg*M_PI/180), failsafe_s(failsafe),
          tilt_min(tilt_min_deg*M_PI/180), tilt_max(tilt_max_deg*M_PI/180) { reset(); }

    void reset() {
        dpan = dtilt = 0; axis = 0; sign[0] = sign[1] = 1; pending = -1;
        Qbefore[0] = Qbefore[1] = 0; state = State::BOOT; lost = 0;
    }

    static void geometric(const Quat& q_est, const Vec3& uav, const Vec3& nsu,
                          double& pan, double& tilt) {
        Vec3 los = {nsu[0]-uav[0], nsu[1]-uav[1], nsu[2]-uav[2]};
        double n = norm3(los); if (n > 0) { los[0]/=n; los[1]/=n; los[2]/=n; }
```

```
        Vec3 r = mat_T_mul(q_to_R(q_est), los);
        pan = std::atan2(r[1], r[0]);
        tilt = std::atan2(r[2], std::hypot(r[0], r[1]));
    }

    PointResult update(const Quat& q_est, const Vec3& uav, const Vec3& nsu,
                       double measured_Q, double dt) {
        double pan_g, tilt_g; geometric(q_est, uav, nsu, pan_g, tilt_g);

        if (measured_Q >= q_track)      { state = State::TRACK;  lost = 0; }
        else if (measured_Q >= q_low)   { state = State::SEARCH; lost = 0; }
        else { lost += dt; state = (lost > failsafe_s) ? State::FAILSAFE : State::SEARCH; }

        if (state == State::TRACK || state == State::SEARCH) {
            if (pending >= 0) {
                int ax = pending;
                if (measured_Q < Qbefore[ax] - 1e-3) {
                    if (ax == 0) dpan -= sign[0]*dither; else dtilt -= sign[1]*dither;
                    sign[ax] *= -1.0;
                }
                pending = -1;
            } else {
                int ax = axis; axis ^= 1;
                Qbefore[ax] = measured_Q;
                if (ax == 0) dpan += sign[0]*dither; else dtilt += sign[1]*dither;
                pending = ax;
            }
        } else if (state == State::FAILSAFE) {
            dpan *= 0.9; dtilt *= 0.9; pending = -1;
        }

        double pan = pan_g + dpan;
        double tilt = tilt_g + dtilt;
        if (tilt < tilt_min) tilt = tilt_min;
        if (tilt > tilt_max) tilt = tilt_max;
        return {pan, tilt, pan_g, tilt_g, dpan, dtilt, state};
    }
};

} // namespace pointing
```

Лістинг В.2 — AP_AntPoint.h

```
// AP_AntPoint.h — модуль ArduPilot для наведення антени поверх спільного ядра.
// Підключає той самий pointing_core.hpp, що й симулятор та Zephyr-варіант.
#pragma once
#include <AP_AHRS/AP_AHRS.h>
#include <AP_Common/Location.h>
#include "pointing_core.hpp"

class AP_AntPoint {
public:
    AP_AntPoint() : _ctl(20.0 /*HPBW*/, 0.75, 0.35, 0.6) {}

    // Викликати з планувальника (наприклад, 50 Гц).
    // measured_Q — нормована якість каналу від радіомодема (0..1).
    void update(const Location &nsu_loc, float measured_Q, float dt);

    float pan_deg()  const { return _pan_deg; }
    float tilt_deg() const { return  tilt_deg; }

private:
    pointing::PointingController _ctl;
    float _pan_deg{0}, _tilt_deg{0};
};
```

Лістинг В.3 — AP_AntPoint.cpp

```
#include "AP_AntPoint.h"
#include <AP_HAL/AP_HAL.h>
#include <SRV_Channel/SRV_Channel.h>

extern const AP_HAL::HAL& hal;

void AP_AntPoint::update(const Location &nsu_loc, float measured_Q, float dt)
{
    AP_AHRS &ahrs = AP::ahrs();

    // 1) Орієнтація корпусу як кватерніон (NED->ENU узгодити з конвенцією ядра).
    Quaternion q; ahrs.get_quaternion(q);
    pointing::Quat q_est{q.q1, q.q2, q.q3, q.q4};

    // 2) Положення БПЛА та НСУ у локальній ENU відносно домашньої точки.
    Vector3f uav_ned; ahrs.get_relative_position_NED_home(uav_ned);
    Vector2f nsu_off = AP::ahrs().get_home().get_distance_NE(nsu_loc);
    pointing::Vec3 uav{uav ned.y, uav ned.x, -uav ned.z};      // NED->ENU
    pointing::Vec3 nsu{nsu_off.y, nsu_off.x, 0.0};

    // 3) ВИКЛИК ЯДРА — той самий алгоритм, що валідовано в симуляторі.
    auto r = _ctl.update(q_est, uav, nsu, measured_Q, dt);
    _pan_deg  = r.pan  * 180.0f / M_PI;
    _tilt_deg = r.tilt * 180.0f / M_PI;

    // 4) Вивід на сервоканали підвісу (азимут/кут місця).
    SRV_Channels::set_output_scaled(SRV_Channel::k_mount_pan,  _pan_deg);
```

```
    SRV_Channels::set_output_scaled(SRV_Channel::k_mount_tilt, _tilt_deg);
}
```

ДОДАТОК Г

Реалізація на Zephyr RTOS (STM32)

Варіант для мікроконтролера (наприклад, NUCLEO-F411RE). Застосунок вмикає те саме ядро pointing_core.hpp (компілюється як C++); адаптер main.c зчитує IMU через драйвери Zephyr, викликає ядро та формує ШІМ-сигнали приводів через devicetree.

Структура:

Тека проєкту містить CMakeLists.txt, prj.conf, app.overlay, pointing_core.hpp та src/main.c.

Покрокова збірка та прошивка:

1. Встановити Zephyr SDK і west (docs.zephyrproject.org).
2. Покласти pointing_core.hpp у теку проєкту (поряд із CMakeLists.txt).
3. Зібрати: west build -b nucleo_f411re . -- -DDTC_OVERLAY_FILE=app.overlay.
4. Прошити: west flash.
5. Перевірити без заліза на нативному симуляторі: west build -b native_sim . && ./build/zephyr/zephyr.exe.

Лістинг Г.1 — src/main.c

```
/* main.c — застосунок Zephyr RTOS: наведення антени поверх спільного ядра.
 * Ядро pointing_core.hpp компілюється як C++ (CONFIG_CPP=y); цей файл — адаптер
 * вводу/виводу: IMU -> AHRS+контролер (ЯДРО) -> ШІМ на два приводи.            */
#include <zephyr/kernel.h>
#include <zephyr/device.h>
#include <zephyr/drivers/sensor.h>
#include <zephyr/drivers/pwm.h>
#include "pointing_core.hpp"

static const struct device *imu = DEVICE_DT_GET_ONE(bosch_bmi160);
static const struct pwm_dt_spec pan  = PWM_DT_SPEC_GET(DT_NODELABEL(ptu_pan));
static const struct pwm_dt_spec tilt = PWM_DT_SPEC_GET(DT_NODELABEL(ptu_tilt));

/* 1000..2000 мкс відповідно до кута -90..+90 град */
static uint32_t angle_to_pulse_ns(float deg) {
    float u = 1500.0f + (deg / 90.0f) * 500.0f;       /* мкс */
    if (u < 1000) u = 1000; if (u > 2000) u = 2000;
    return (uint32_t)(u * 1000.0f);                    /* нс */
}

int main(void)
{
    static pointing::MahonyAHRS ahrs(2.0, 0.1);
    static pointing::PointingController ctl(20.0, 0.75, 0.35, 0.6);
    pointing::Vec3 uav{300, 120, 150}, nsu{0, 0, 0};   /* приклад; реально — з GNSS/телеметрії */
    const double dt = 0.02;                            /* 50 Гц */

    while (1) {
        struct sensor_value a[3], g[3];
        sensor_sample_fetch(imu);
        sensor_channel_get(imu, SENSOR_CHAN_ACCEL_XYZ, a);
        sensor_channel_get(imu, SENSOR_CHAN_GYRO_XYZ,  g);
        pointing::Vec3 acc{sensor_value_to_double(&a[0]), sensor_value_to_double(&a[1]), sensor_value_to_double(&a[2])};
        pointing::Vec3 gyro{sensor_value_to_double(&g[0]), sensor_value_to_double(&g[1]), sensor_value_to_double(&g[2])};
        pointing::Vec3 mag{0, 1, 0};                   /* за наявності магнітометра — реальні дані */

        pointing::Quat q = ahrs.update(gyro, acc, mag, dt);
        double Q = 0.9;                                /* ВИМІРЯНА якість каналу з UART-модема */
        auto r = ctl.update(q, uav, nsu, Q, dt);       /* ВИКЛИК ЯДРА */
```

```
        pwm_set_dt(&pan,  PWM_USEC(20000), angle_to_pulse_ns(r.pan  * 57.2958));
        pwm_set_dt(&tilt, PWM_USEC(20000), angle_to_pulse_ns(r.tilt * 57.2958));
        k_sleep(K_MSEC(20));
    }
    return 0;
}
```

Лістинг Г.2 — prj.conf

```
CONFIG_CPP=y
CONFIG_STD_CPP14=y
CONFIG_GLIBCXX_LIBCPP=y
CONFIG_SENSOR=y
CONFIG_PWM=y
CONFIG_LOG=y
```

Лістинг Г.3 — app.overlay

```
/ {
    pwmleds {
        compatible = "pwm-leds";
    };
    aliases { };
};

&timers3 { status = "okay"; };
&timers4 { status = "okay"; };

/ {
    ptu_pan:  ptu_pan  { compatible = "pwm-servo"; pwms = <&pwm3 1 PWM_USEC(20000) PWM_POLARITY_NORMAL>; };
    ptu_tilt: ptu_tilt { compatible = "pwm-servo"; pwms = <&pwm4 1 PWM_USEC(20000) PWM_POLARITY_NORMAL>; };
};
```

ДОДАТОК Д

Валідація алгоритму: тести та імітаційне моделювання

Коректність ядра перевірено трьома незалежними способами. По-перше, модульні тести pointing_core безпосередньо перевіряють властивості матриці повороту, геометричне наведення, збіжність фільтра Магоні, переходи кінцевого автомата та здатність контуру радіочастотного автосупроводження усувати незмінний перекіс монтажу антени. По-друге, крос-перевіркою встановлено, що C++ та Python реалізації ядра дають однаковий результат. По-третє, симулятори simulate_pointing.py та interactive_pointing.py проганяють ядро в замкненому контурі (фізика → виміряна Q → ядро → команди → фізика), тобто демонстрація є валідацією реального коду, а не його наближення.

Запуск перевірок:

1. Модульні тести ядра: python test_pointing_core.py (6 тестів).
2. Крос-перевірка C++/Python: g++ -std=c++14 -O2 test_core.cpp -o test_core; ./test_core > cpp.txt; python xcheck_python.py > py.txt; diff py.txt cpp.txt.
3. Імітаційне моделювання: python simulate_pointing.py (анімація) або python interactive_pointing.py (інтерактивно).

Лістинг Д.1 — test_pointing_core.py

```
"""
test_pointing_core.py — модульні тести ЯДРА алгоритму (pointing_core.py).

Запуск:
    python test_pointing_core.py        # власний раннер
    pytest test_pointing_core.py        # якщо встановлено pytest
Ці тести перевіряють САМЕ написаний код ядра, а не його копію.
"""
import math
import numpy as np
import pointing core as pc


def test rotation is orthonormal():
    q = pc.q_norm([0.5, 0.5, -0.3, 0.7])
    R = pc.q_to_R(q)
    assert np.allclose(R @ R.T, np.eye(3), atol=1e-9)
    assert abs(np.linalg.det(R) - 1.0) < 1e-9


def test_boresight_unit_and_angles():
    bs = pc.boresight_body(math.radians(30), math.radians(20))
    assert abs(np.linalg.norm(bs) - 1.0) < 1e-12
    assert abs(math.atan2(bs[1], bs[0]) - math.radians(30)) < 1e-9
    assert abs(math.atan2(bs[2], math.hypot(bs[0], bs[1])) - math.radians(20)) < 1e-9


def test_geometric_pointing_known_cases():
    q = np.array([1.0, 0, 0, 0])                        # корпус без повороту
    # БПЛА прямо над НСУ -> промінь униз, tilt = -90°
    pan, tilt, _ = pc.PointingController.geometric(q, [0, 0, 200], [0, 0, 0])
    assert abs(math.degrees(tilt) + 90) < 1e-6
    # БПЛА на схід від НСУ -> промінь на захід, pan = 180°, tilt = 0
    pan, tilt, _ = pc.PointingController.geometric(q, [100, 0, 0], [0, 0, 0])
    assert abs(abs(math.degrees(pan)) - 180) < 1e-6
    assert abs(math.degrees(tilt)) < 1e-6


def test_mahony_converges_to_true_attitude():
```

```
    # істинна орієнтація (крен 15°, тангаж -10°, курс 40°)
    def euler_q(r, p, y):
        cr, sr = math.cos(r/2), math.sin(r/2)
        cp, sp = math.cos(p/2), math.sin(p/2)
        cy, sy = math.cos(y/2), math.sin(y/2)
        return pc.q_norm([cr*cp*cy+sr*sp*sy, sr*cp*cy-cr*sp*sy,
                          cr*sp*cy+sr*cp*sy, cr*cp*sy-sr*sp*cy])
    qt = euler_q(math.radians(15), math.radians(-10), math.radians(40))
    Rt = pc.q_to_R(qt)
    acc = Rt.T @ np.array([0, 0, 1.0])                # ідеальні датчики
    mag = Rt.T @ np.array([0, 1.0, 0])
    ahrs = pc.MahonyAHRS(kp=3.0, ki=0.0)
    for _ in range(4000):                             # 20 с при 200 Гц
        ahrs.update([0, 0, 0], acc, mag, 0.005)
    # кут між оцінкою та істиною має зійти до ~0
    ang = 2*math.acos(min(1.0, abs(float(np.dot(ahrs.q, qt)))))
    assert math.degrees(ang) < 1.0


def test_fsm_transitions():
    ctl = pc.PointingController(failsafe_s=1.0)
    r = ctl.update([1,0,0,0], [300,0,200], [0,0,0], measured_Q=0.9, dt=0.05)
    assert r["state"] == pc.STATE_TRACK
    r = ctl.update([1,0,0,0], [300,0,200], [0,0,0], measured_Q=0.5, dt=0.05)
    assert r["state"] == pc.STATE_SEARCH
    for _ in range(40):                               # тривала відсутність сигналу
        r = ctl.update([1,0,0,0], [300,0,200], [0,0,0], measured_Q=0.0, dt=0.05)
    assert r["state"] == pc.STATE_FAILSAFE


def test_reactive_loop_corrects_mount_offset():
    """Контур радіочастотного автосупроводження має зменшити похибку, спричинену незмінним
    механічним зміщенням осі антени (інерціальний контур його «не бачить")."""
    hpbw = math.radians(20)
    mount_pan, mount_tilt = math.radians(5.0), math.radians(-4.0)   # перекіс монтажу
    q = np.array([1.0, 0, 0, 0])
    uav, nsu = np.array([300.0, 120.0, 200.0]), np.array([0.0, 0.0, 0.0])
    los = (nsu - uav) / np.linalg.norm(nsu - uav)

    def measure_Q(pan_cmd, tilt_cmd):
        bs = pc.boresight_body(pan_cmd + mount_pan, tilt_cmd + mount_tilt)  # реальний промінь
        bs_world = pc.q_to_R(q) @ bs
        delta = math.acos(max(-1.0, min(1.0, float(bs_world @ los))))
        return max(0.0, 1.0 - (delta / hpbw) ** 2), delta

    ctl = pc.PointingController(hpbw_deg=20, dither_deg=0.5)
    pan_g, tilt_g, _ = pc.PointingController.geometric(q, uav, nsu)
    Q0, d0 = measure_Q(pan_g, tilt_g)                 # лише геометрія (з перекосом)
    Q = Q0; last = None
    for _ in range(400):
        res = ctl.update(q, uav, nsu, Q, 0.05)
        Q, last = measure_Q(res["pan"], res["tilt"])
    Qf, df = measure_Q(res["pan"], res["tilt"])
    assert df < d0 * 0.4, (math.degrees(d0), math.degrees(df))   # похибка суттєво менша
    assert Qf > 0.9


def _run():
    tests = [v for k, v in sorted(globals().items()) if k.startswith("test ")]
    ok = 0
    for t in tests:
        try:
            t(); print(f"PASS  {t.__name__}"); ok += 1
        except AssertionError as e:
            print(f"FAIL  {t.__name__}: {e}")
        except Exception as e:
            print(f"ERROR {t.__name__}: {e}")
    print(f"\n{ok}/{len(tests)} тестів пройдено")
    return ok == len(tests)


if __name__ == "__main__":
    import sys
    sys.exit(0 if _run() else 1)
```