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

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____Олександр КОВАЛЬ
«_____»_____2025р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Програмне забезпечення для автономної системи прийняття рішень в реальному часі»

Виконав (-ла):
студент (-ка) IV курсу, групи _ТВ-12__
_Нересниця Олег Юрійович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
___ст. викл. Сарибога Г. В._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:

_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (ка)_____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер - фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ В.о. завідувача кафедри
_____ Олександр КОВАЛЬ (підпис)
«____»_____2025р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Нересниці Олегу Юрійовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи
_Програмне забезпечення для автономної системи прийняття рішень в реальному часі_____
_____
керівник роботи_____ ст. викл. Сарибога Г. В._____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від "__"_____202__року №____
2. Строк подання студентом роботи :_____
3. Вихідні дані до роботи:__Мова програмування Python, Мова програмування Kotlin_____
4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити):__ Аналіз існуючих систем та вимог;Проектування архітектури системи як набору взаємодіючих модулів з незалежною логікою роботи;Розробити модуль прийняття рішень в реальному часі та модуль слідування траєкторії;Розробити користувацький інтерфейс у вигляді мобільного застосунку для комунікації з дроном;Протестувати роботу системи на основі прототипу.
5. Перелік ілюстративного матеріалу:_ Опис використаних інструментів; структура програмного забезпечення; блок-схема алгоритму; взаємодія користувачів з системою._____
_____
6. Дата видачі завдання «29» _вересня__2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	29.09.2024	
2	Дослідження предметної області	14.01.2025 - 27.01.2025	
3	Постановка вимог до проєктування системи	28.01.2025 - 01.02.2025	
4	Дослідження існуючих рішень	02.02.2025 - 10.02.2025	
5	Розробка програмного продукту	11.02.2025 – 24.04.2025	
6	Тестування	25.04.2025 – 10.05.2025	
7	Захист програмного продукту	12.05.2025	
8	Оформлення дипломної роботи	18.05.2025 – 28. 05.2025	
9	Передзахист	02.06.2025	
10	Захист	16.06.2025	

Студент _____ _Нересниця Олег_

(підпис) (ім'я, прізвище)

Керівник роботи _____ _ Ганна Сарибога_

(підпис) (ім'я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота була виконана на 50 аркушах. Вона містить: 2 додатка, перелік посилань на використані джерела та сторінки, 11 рисунків, 1 таблицю.

Метою роботи є реалізація повністю автономної поведінки, адаптивної до змінних умов за допомогою обробки потоків даних у реальному часі.

Для реалізації мети розроблено мобільний застосунок та бортовий модуль управління. Розроблена система для автономного управління дроном забезпечує автономність прийняття рішень у режимі реального часу. Програмне забезпечення складається з двох частин: мобільного застосунку для візуалізації стану польоту та взаємодії з користувачем та системи керування на борту дрона, яка виконує обчислення, обробку даних сенсорів та ухвалення рішень. Система дозволяє дрону адаптуватися до змін у середовищі — змінювати швидкість, ухилятись від перешкод та повертатися до початкової траєкторії.

Практичне значення отриманих результатів полягає у створенні функціональної системи, яка може бути використана в реальних польотних сценаріях для автономної навігації, уникнення перешкод та адаптації до змін навколишнього середовища. Завдяки модульності архітектури, система легко інтегрується з різними типами дронів, а мобільний застосунок забезпечує зручне керування та моніторинг, що дозволяє користувачу оперативно контролювати політ та параметри дрона.

Ключові слова: автономність, система управління, виконання місій, обхід перешкод, мобільний застосунок, моніторинг місій, PID контроллер.

ABSTRACT

Structure and volume of the thesis. The paper is 50 pages length and consists of 1 appendix, a list of source references, 11 images and 1 table.

The purpose of this work is to implement fully autonomous behavior that is adaptive to changing conditions by processing data streams in real time.

To achieve this goal, we developed a mobile application and an onboard control module. The developed system for autonomous control of the drone provides autonomous decision-making in real-time. The software consists of two parts: a mobile application for visualizing the flight status and interacting with the user and an onboard control system that performs calculations, processing of sensor data, and decision-making. The system allows the drone to adapt to changes in the environment by changing speed, dodging obstacles, and returning to its original trajectory.

The practical significance of the results obtained is the creation of a functional system that can be used in real flight scenarios for autonomous navigation, obstacle avoidance, and adaptation to environmental changes. Due to the modularity of the architecture, the system can be easily integrated with different types of drones, and the mobile application provides convenient control and monitoring, allowing the user to quickly control the flight and parameters of the drone.

Keywords: autonomy, control system, mission execution, obstacle avoidance, mobile application, mission monitoring, PID controller.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

MAVLink — легкий протокол зв’язку, який використовується для обміну даними між безпілотними апаратами та наземними станціями або іншими пристроями. Він забезпечує передачу команд, телеметрії, стану дрона і дозволяє керувати польотом у реальному часі.

ArduPilot — відкрите програмне забезпечення для автопілотування безпілотних літальних апаратів, наземної техніки та підводних роботів. Воно забезпечує автоматичне керування польотом, навігацію та виконання місій, підтримуючи широкий спектр апаратних платформ.

PX4 — відкрита платформа програмного забезпечення для автопілотування дронів і робототехніки, яка забезпечує управління польотом, навігацію та взаємодію з різними датчиками і апаратними засобами. PX4 підтримує широкий спектр апаратних платформ і широко використовується в дослідженнях та комерційних рішеннях.

REST (Representational State Transfer) — архітектурний стиль для створення веб-сервісів, який використовує стандартні HTTP-запити для взаємодії між клієнтом і сервером. REST дозволяє просто і ефективно обмінюватися даними у вигляді ресурсів через URL.

SDK (Software Development Kit) — набір інструментів, бібліотек і документації, який допомагає розробникам створювати програми для певної платформи або системи.

API (Application Programming Interface) — набір правил і протоколів, який дозволяє різним програмам взаємодіяти між собою, обмінюватися даними та використовувати функції одна одної.

MSP (Multiwii Serial Protocol) — це легкий і простий протокол зв’язку, розроблений для обміну даними між контролером польоту дрона і зовнішніми пристроями, такими як наземні станції, передавачі або монітори. Протокол використовується переважно для передачі телеметрії (дані про стан дрона: висота, швидкість, положення, заряд батареї тощо) і прийому команд керування (зміна режиму польоту або налаштувань).

UART (Universal Asynchronous Receiver/Transmitter) — це апаратний протокол послідовної асинхронної передачі даних, який використовується для обміну інформацією між пристроями. UART перетворює паралельні дані в послідовний бітовий потік для передачі і навпаки для прийому, забезпечуючи простий і надійний зв’язок без синхронізуючого сигналу.

ВСТУП

У наш час технології автономного управління активно впроваджуються в різні галузі, зокрема в системи безпілотних літальних апаратів (БПЛА). Завдяки здатності виконувати широкий спектр завдань — від аерофотозйомки та моніторингу навколишнього середовища до пошуково-рятувальних операцій і доставки вантажів — дрони стали незамінними інструментами як у цивільному, так і у військовому секторах.

Однак використання дронів на повну потужність вимагає не лише дистанційного керування, але й високого рівня автономності. Здатність дрона самостійно приймати рішення в реальному часі — ключова умова ефективності його роботи в умовах складного або динамічного середовища. Автономна система повинна вміти аналізувати вхідні дані від сенсорів, виявляти та уникати перешкод, а також адаптувати параметри польоту залежно від зовнішніх умов, зокрема погодних.

У зв’язку з цим розробка програмного забезпечення для автономної системи прийняття рішень у реальному часі є надзвичайно актуальною темою. Вона поєднує в собі завдання в галузі програмної інженерії, робототехніки, та обробки даних із сенсорів. Таке ПЗ має відповідати високим вимогам до надійності, швидкодії та адаптивності, що є викликом і водночас потужною мотивацією для досліджень у цій сфері.

Метою цього дослідження є розробка та впровадження програмного забезпечення для системи автономного керування дроном, яка здатна здійснювати політ за заданою траєкторією з динамічною реакцією на змінні умови середовища. Передбачається створення системи, яка в реальному часі аналізує дані з віддалеміра (датчика відстані) для виявлення перешкод і корекції траєкторії польоту, забезпечуючи тим самим безпечний обліт об'єктів або зміну маршруту

при необхідності.

Крім того, система повинна мати функціональність адаптації до змін погодних умов, таких як сильний вітер, дощ чи зниження температури. Для цього розглядається механізм зміни режиму польоту залежно від інформації, отриманої із зовнішніх датчиків або прогнозних моделей.

1 ЗАВДАННЯ ПОБУДОВИ АВТОНОМНОЇ СИСТЕМИ ПРИЙНЯТТЯ РІШЕНЬ В РЕАЛЬНОМУ ЧАСІ

1.1 Постановка задачі

Метою даного дослідження є створення автономної системи прийняття рішень в реальному часі, що включає підсистему керування дроном за заданою траєкторією з використанням PID-контроллера. Система повинна забезпечувати стабільний політ за маршрутом із можливістю оперативного втручання у випадках зміни зовнішніх умов або виникнення перешкод.

Прийняття рішень базується на обробці даних із двох типів датчиків: датчиків погодних умов та датчика вимірювання відстані до об’єктів навколо дрона. У разі виявлення перешкод застосовується алгоритм пропорційного керування, який коригує траєкторію польоту, виходячи з різниці між безпечною відстанню та фактичним значенням, отриманим від датчика відстані. Це дозволяє ефективно обходити перешкоди без різких маневрів, підтримуючи безпеку польоту.

Зміна режимів польоту залежно від погодних умов впливає на параметри управління дроном, насамперед на швидкість проходження заданої траєкторії. Система адаптує свої дії відповідно до рівня вітру, опадів та інших погодних факторів, що забезпечує надійність і стабільність польоту в умовах непередбачуваного середовища.

Кінцевими користувачами даного програмного продукту є фахівці у сфері безпілотних авіа систем, а також організації, які використовують дрони для виконання автоматизованих або ризикованих завдань у змінному середовищі.

1.2 Аналіз існуючих систем

Системи слідування місій є ключовими компонентами в управлінні безпілотними літальними апаратами та іншими автономними платформами. Вони забезпечують оператору можливість планувати маршрути, контролювати виконання завдань у реальному часі та отримувати телеметричні дані з бортових сенсорів. Розвиток таких систем є критичним для сфер, де потрібна автономна навігація — зокрема в геодезії, аграрному моніторингу, пошуково-рятувальних операціях та військових застосуваннях.

Серед великої кількості програмних рішень, що існують на сьогодні, особливе місце займають **ArduPilot Mission Control** та **QGroundControl** — два потужні інструменти, які широко використовуються як дослідниками, так і професіоналами. Вони дозволяють створювати складні сценарії місій, забезпечують надійне керування польотом і підтримують різноманітні типи апаратів.

ArduPilot — це платформа для автономного керування різними типами транспортних засобів (літальні апарати, наземні роботи, човни, підводні дрони тощо). Система управління місіями в ArduPilot реалізується через низку інтерфейсів, зокрема Mission Planner (Windows), MAVProxy (CLI) та сторонні мобільні додатки. На Рис. 1.1 Зображено приклад створення місії в Ardupilot Mission Control.

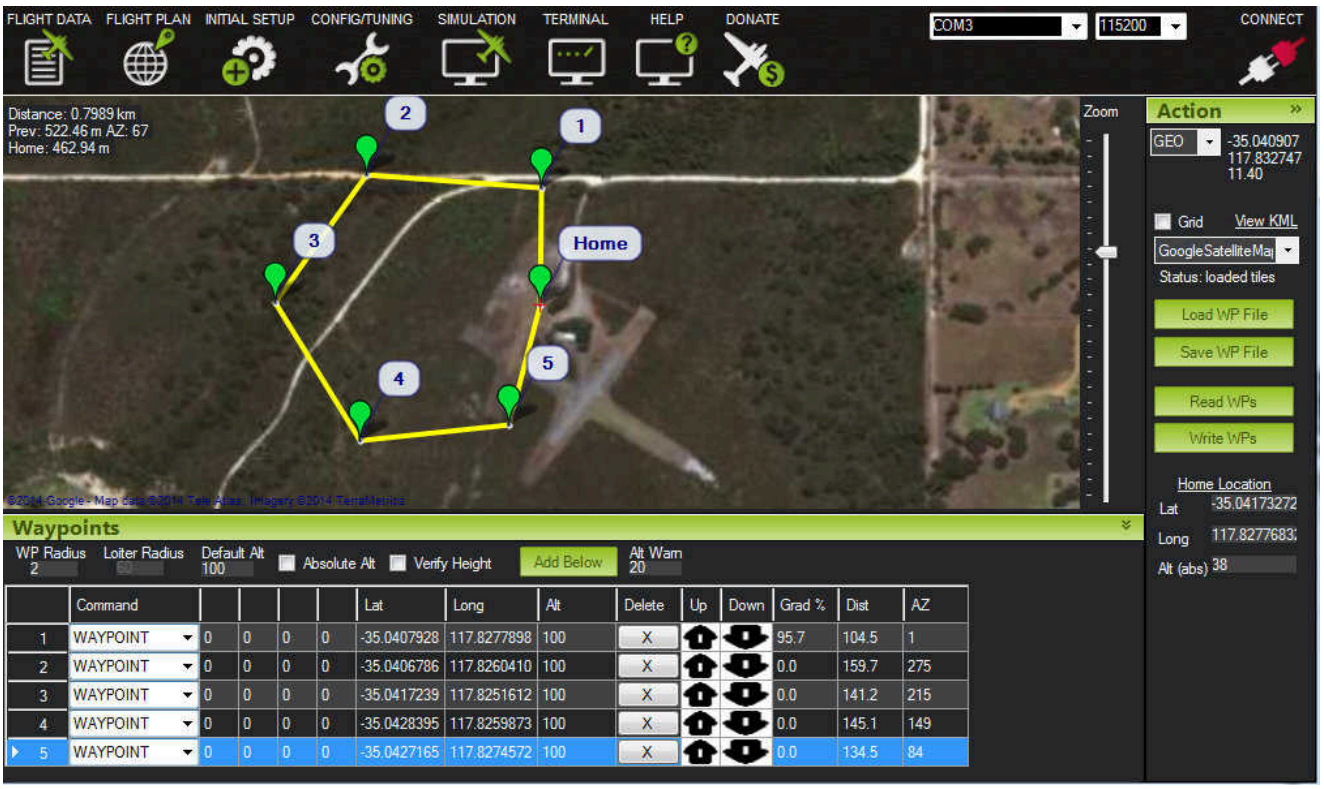


Рис. 1.1 Приклад створення місії в Ardupilot Mission Control

Основний функціонал ArduPilot Mission Control в себе включає:

- Створення та редагування місій з Waypoints.
- Моніторинг телеметрії у реальному часі: GPS, батарея, швидкість, висота.
- Автоматичне калібрування сенсорів та конфігурація PID-регуляторів.
- Підтримка скриптів LUA для кастомної логіки.
- Розширена підтримка сценаріїв з автопілотом (напр., сценарії пошуку та порятунку, аграрні місії тощо).

Дана система має основні переваги у вигляді підтримки широкого спектру транспортних засобів та високої гнучкості налаштувань. Крім переваг, дана система має даний ряд недоліків: високий поріг входу користувачів для початку використання, та потребує додаткового програмного забезпечення.

QGroundControl — це кросплатформенний ground station software з відкритим кодом, що надає зручний графічний інтерфейс для роботи з автопілотами на основі MAVLink (ArduPilot, PX4). Його основна мета — надати простий, візуально зрозумілий спосіб керування місіями.

Основний функціонал QGroundControl:

- Графічне планування місій: перетягування точок маршруту, налаштування висоти, швидкості, дій у точках.
- Інтеграція мап з підтримкою офлайн-режиму.
- Режим реального часу: телеметрія, стан сенсорів, HUD (Head-Up Display).
- Інструменти калібрування (компас, акселерометр, радіоканал).

Дана система має основні переваги у вигляді низького порогу входу користувачів для початку використання, працездатна на будь-якій платформі, та має візуалізацію місій в 3D. Крім переваг, дана система має даний ряд недоліків: обмежена підтримка систем керування окрім PX4, та обмежена можливість кастомізації місій. На Рис. 1.2 зображений інтерфейс QGroundControl для створення місій.

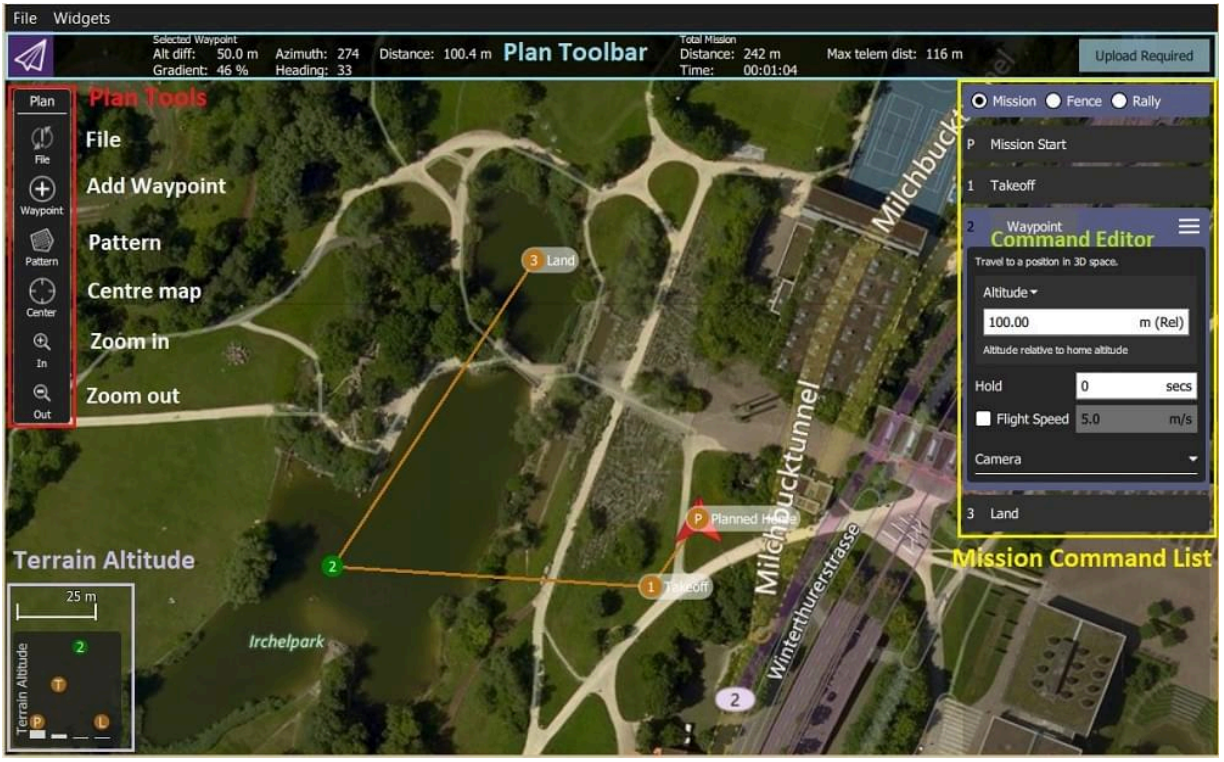


Рис. 1.2 Інтерфейс QGroundControl для створення місій.

Більш детальне порівняння представлено в таблиці 1.1.

Параметр	ArduPilot Mission Control	QGroundControl
Підтримка автопілотів	ArduPilot (APM, Pixhawk, Cube, Navio, etc.)	PX4 (оптимізований), обмежена підтримка ArduPilot
Тип планування місій	Табличне (waypoint list), можливість скриптів	Візуальне планування (drag-and-drop на карті)
Алгоритми управління	Підтримка EKF2/EKF3, LUA-скриптів, настроювані PID	Стандартний PX4 stack, без підтримки сторонніх скриптів
Підтримка LUA/скриптів onboard	Так (LUA scripting engine для onboard automation)	Ні (планування обмежене GUI і PX4 місіями)

Режим автопілоту	Понад 20 режимів (Auto, Guided, RTL, Loiter, Follow Me тощо)	Основні режими PX4 (Auto, Manual, RTL, Mission)
Інтеграція з телеметрією	MAVLink, серійний порт, 4G модем, SiK telemetry	MAVLink (через Wi-Fi, USB, SiK, LTE)
Підтримка розширених дій у місіях	Так (DO_JUMP, SET_SERVO, CONDITIONAL_DELAY, CAM_TRIGG_DIST)	Обмежена (підтримка лише базових waypoint-команд)
3D-моделювання місії / симуляція	Через сторонні симулятори (SITL + Mission Planner)	Вбудований 3D-перегляд місій, Terrain Following
Оновлення параметрів у польоті	Так, підтримка зміни параметрів у реальному часі	Частково (обмежено через інтерфейс)
Кросплатформеність	Mission Planner – лише Windows; MAVProxy – кросплатформений	Повна (Windows, macOS, Linux, Android, iOS)

Таблиця 1.1 – Таблиця аналізу існуючих систем

1.3 Аналіз алгоритмів слідування траєкторії

Розглянемо кілька алгоритмів слідування траєкторії, проаналізуємо їхні переваги, недоліки та застосування.

PID-регулятор. PID-регулятор — це класичний метод керування, який використовує три складові (P — пропорційна, I — інтегральна, D — диференціальна), щоб мінімізувати відхилення системи від заданого значення. Його завдання — стабілізувати систему та забезпечити плавне слідування за цільовою траєкторією або значенням.

Переваги ПІД-регулятора полягають насамперед у його простоті та універсальності. Його легко реалізувати як у програмному, так і в апаратному забезпеченні. Завдяки тому, що ПІД-регулятор не вимагає точного математичного опису системи, він підходить для широкого спектра завдань у керуванні, де точна модель недоступна або складна для побудови. У випадку з дронами це особливо корисно, оскільки дозволяє швидко налаштувати базову систему стабілізації. Пропорційна складова дозволяє швидко реагувати на зміну положення, інтегральна — компенсує постійну похибку, а диференційна — згладжує реакцію і зменшує коливання, що загалом забезпечує точне й стабільне слідування за цільовим значенням.

Недоліки ПІД-регулятора стають помітними при складних, нелінійних або швидкозмінних умовах. Оскільки він базується лише на поточній похибці та її похідних, він не враховує структуру системи чи зовнішні обмеження, як-от максимальні швидкості або навантаження. Це може призвести до перенасичення, особливо якщо неправильно налаштована інтегральна складова — у такому випадку система починає «накопичувати» помилку, що викликає нестійкість (так званий "integral windup"). Крім того, налаштування коефіцієнтів P, I та D зазвичай виконується вручну, що потребує досвіду і часу, особливо в складних або змінних умовах польоту. У високодинамічних середовищах або при виконанні агресивних маневрів, ПІД-регулятор може не забезпечити достатньої точності та швидкості реакції.

Застосування ПІД-регулятора охоплює переважно задачі стабілізації та базового керування. Зокрема, він активно використовується для регулювання кута нахилу (roll, pitch, yaw), підтримки висоти або позиції, а також для стабільного утримання дрона на місці у повітрі. При простих траєкторіях, наприклад, підйомі на задану висоту або русі вперед по прямій, ПІД-регулятор цілком справляється з поставленими завданнями.

MPC (Model Predictive Control, або модельно-прогнозне керування) — це метод керування, який на основі математичної моделі дрона прогнозує його майбутню поведінку та обирає оптимальні керуючі дії, мінімізуючи задану цільову функцію (помилку траєкторії або витрати енергії) з урахуванням обмежень на рух чи динаміку.

Переваги MPC полягають у його здатності враховувати як динаміку системи, так і обмеження на керуючі сигнали та стан. Однією з ключових переваг є здатність прогнозувати майбутні стани системи на основі поточного стану та моделі, завдяки чому MPC може приймати кращі рішення порівняно з класичними підходами, які реагують лише на поточну похибку. MPC також добре працює зі складними, багатовимірними системами, де класичні регулятори можуть бути неефективними. Завдяки вбудованій оптимізації, він дозволяє балансувати між точністю, енергоефективністю, плавністю руху та іншими критеріями керування.

Недоліки MPC переважно пов’язані з його обчислювальною складністю. Оскільки в кожен момент часу MPC розв’язує задачу оптимізації в реальному часі, це вимагає потужного процесора або добре оптимізованого коду. Для дронів із обмеженими ресурсами, реалізація MPC може бути складною або потребувати спрощення моделі. Крім того, точність та ефективність MPC залежать від якості моделі динаміки дрона. Якщо модель неточна або не враховує зовнішні збурення (наприклад, вітер), керування може погіршитися. Також налаштування MPC (вибір цільової функції, горизонту прогнозування, обмежень тощо) складніше і потребує глибшого розуміння системи порівняно з класичними підходами.

Застосування MPC охоплює складні задачі, де важливі точність, безпечне маневрування та дотримання обмежень. Наприклад, MPC використовується для слідування за складними траєкторіями у тривимірному просторі, польотів у тісних або динамічних середовищах, посадки на рухомі платформи, автономної навігації та взаємодії з іншими літаючими апаратами.

VFF.VFF (Vector Field Following) — це метод керування, при якому дрон рухається, слідуючи векторному полю, яке задає напрямок руху в кожній точці простору. Іншими словами, дрон орієнтується на вектори поля, щоб залишатися на траєкторії або рухатися вздовж заданого маршруту.

VFF має кілька переваг. По-перше, цей метод досить простий у реалізації та не вимагає складних розрахунків або точних моделей динаміки дрона. Він добре підходить для плавного слідування за криволінійними траєкторіями, оскільки векторне поле можна сконструювати так, щоб враховувати вигини і повороти шляху. Також VFF зазвичай більш стійкий до невеликих збурень і помилок, бо напрямок руху задається безпосередньо через вектори, що створює природну корекцію курсу.

Недоліки VFF пов’язані з тим, що метод вимагає правильного будування векторного поля — якщо поле погано сконструйоване, дрон може відхилятися від потрібної траєкторії. Крім того, VFF не завжди ефективний при різких змінах траєкторії або в умовах обмежених ресурсів дрона, бо не враховує динамічні обмеження системи (максимальну швидкість, прискорення тощо). Також цей метод менш точний, ніж більш складні оптимізаційні підходи, особливо при високих швидкостях або складних маневрах.

Щодо застосування, VFF часто використовується для простого та плавного слідування за заданими шляхами, наприклад, при патрулюванні території, обльоті перешкод або виконанні повторюваних маршрутів. Його можна застосовувати в автономних дронах середньої складності, де важлива надійність і простота реалізації, але немає потреби у складному обчислювальному керуванні.

1.4 Вибір рішення для побудови системи

Вибір способу реалізації системи автономного управління дроном ґрунтувався на поєднанні практичної ефективності, технологічної гнучкості та зручності використання. Кожен компонент системи був обраний з урахуванням специфічних вимог до обробки даних у реальному часі, простоти інтеграції та можливості масштабування.

Як основний алгоритм слідування маршруту було обрано PID-контроллер. Основною причиною цього вибору є його перевірена ефективність у системах керування з неперервним зворотним зв’язком. Простота реалізації, легка налаштованість і низька обчислювальна складність роблять PID хорошим кандидатом для вбудованих систем, що працюють у режимі реального часу.

Як засіб взаємодії між частиною дрона та мобільним застосунком було обрано Flask — легковагий фреймворк на Python. Його перевага полягає в простоті інтеграції в уже існуючий стек програмного забезпечення, низькому порозі входу, а також можливості швидко реалізувати REST-інтерфейс для передачі даних. Хоча Flask зазвичай асоціюється з веб-серверами, у даному випадку він використовується як локальний сервер між модулями системи, оскільки забезпечує швидке тестування, гнучкість та розширюваність.

У якості користувацького інтерфейсу було обрано мобільний застосунок на базі Android. Це рішення обумовлено потребою у зручному, доступному і портативному засобі взаємодії з дроном у реальних умовах.

2 АНАЛІЗ ІНСТРУМЕНТІВ РОЗРОБКИ

2.1 Середовища розробки

Вибір інструментів розробки для реалізації програмного забезпечення системи автономного керування дроном зумовлений прагненням забезпечити максимальну ефективність, стабільність і зручність у роботі. Саме тому основним середовищем розробки було обрано продукти компанії JetBrains — зокрема, PyCharm для модуля управління та Android Studio для розробки мобільного застосунку на Kotlin.

JetBrains — це міжнародна компанія, яка спеціалізується на створенні професійних інструментів для розробників. Її підхід базується на принципі "Develop with pleasure" — створення таких середовищ, які роблять процес розробки максимально продуктивним, зрозумілим і приємним.

2.1.1 Pycharm

PyCharm — це інтегроване середовище розробки для мови програмування Python, розроблене компанією JetBrains. Воно вирізняється багатим функціоналом, який охоплює весь цикл розробки програмного забезпечення: написання коду, його перевірку, тестування, налагодження, запуск та розгортання. PyCharm доступний у двох основних версіях: Community (безкоштовна) та Professional (комерційна, з розширеним функціоналом).

PyCharm забезпечує повну підтримку стилістичних рекомендацій PEP 8 — офіційного гайду щодо оформлення коду на Python. IDE автоматично перевіряє код на відповідність цим правилам у реальному часі, підкреслює порушення та пропонує варіанти виправлення. Крім того, PyCharm має вбудований інструмент

автоматичного форматування, який дозволяє привести код у відповідність до PEP 8.

Нижче наведені ключовий функціонал, який роблять PyCharm хорошим кандидатом для реалізації складних проєктів, таких, як система автономного керування дроном:

- Інтеграція з Git. PyCharm має вбудовану підтримку Git, GitHub, GitLab, Mercurial та інших VCS. Розробник може прямо з IDE виконувати всі основні операції: коміти, створення гілок, перегляд журналу змін, вирішення конфліктів злиття тощо. Інтерфейс зручний і дозволяє працювати з репозиторіями без необхідності відкривати окрему консоль або сторонні інструменти.
- Можливість віддаленого запуску та налагодження. PyCharm дозволяє працювати з кодом, розгорнутим на віддаленому пристрої, наприклад Raspberry pi. Завдяки функції "Deployment Configuration" можна синхронізувати локальний код із сервером через SSH/SFTP. Крім того, Professional-версія дозволяє налаштувати "Remote Interpreter" — тобто виконувати код безпосередньо на віддаленій машині, не встановлюючи всі залежності локально.

У рамках розробленої системи PyCharm допомагає швидко налаштовувати середовища (venv), відлагоджувати обробку протоколів MSP і тестувати логіку прийняття рішень на основі вхідних сенсорних даних. Його інструменти для перегляду стеку викликів, змінних і обробки виключень пришвидшують розробку і дозволяють ефективно локалізовувати баги, особливо у багатопоточному середовищі.

2.1.2 Android Studio

Android Studio — це інтегроване середовище розробки (IDE) для створення Android-додатків, розроблене компанією Google на базі платформи IntelliJ IDEA

від JetBrains. Воно забезпечує повний стек інструментів для створення, налагодження, тестування та розгортання Android-застосунків.

Нижче наведені ключові можливості Android Studio, які особливо корисні під час розробки мобільного клієнта для автономної системи:

- Інтеграція з Git. Android Studio має повноцінну інтеграцію з системами контролю версій — зокрема Git, GitHub та GitLab.
- Вбудований Android SDK. Android Studio постачається з інтегрованим Android Software Development Kit, який включає: Android Platform API, інструменти компіляції, SDK Manager для встановлення/оновлення компонентів, ADB (Android Debug Bridge) — консольний інструмент для взаємодії з пристроями.
- Запуск застосунку на підключеному фізичному пристрої. Android Studio дозволяє напряму запускати й налагоджувати застосунок на фізичному Android-пристрої через USB або Wi-Fi. Це суттєво прискорює цикл розробки, тестування та виправлення помилок — на реальному пристрої можна миттєво перевіряти, як виглядає й поводиться застосунок у реальних умовах.

2.2 Мови програмування та фреймворки

Підбір мов програмування та фреймворків здійснювався з урахуванням потреб реального часу, підтримки багатопоточності, а також сумісності з апаратною частиною дрона. Пріоритет надавався інструментам, що забезпечують стабільну роботу на обмежених ресурсах, мають зручні засоби для обробки сенсорних даних та керування потоками, а також дозволяють швидко інтегрувати алгоритми керування з апаратними інтерфейсами.

Також важливими критеріями були зручність розробки мобільного клієнта, можливість кросплатформної роботи та простота розгортання API для зв’язку між

модулями. Обрані технології дозволили швидко реалізувати прототип системи, забезпечити її масштабованість та гнучкість у подальшій розробці.

2.2.1 Python

Вибір мови програмування Python для реалізації модуля управління в автономній системі прийняття рішень ґрунтувався на її гнучкості, зрозумілості та наявності потужного інструментарію для роботи з обчислювальними процесами в реальному часі. Мова забезпечує зручну реалізацію класичних і сучасних алгоритмів керування, зокрема PID-регуляторів, а також дозволяє ефективно працювати з потоками, синхронізацією та обробкою зовнішніх подій через об’єкти типу Event і Lock. Завдяки бібліотекам, таким як numpy, scipy, threading, multiprocessing та іншим, Python значно спрощує реалізацію алгоритмів керування траєкторією, адаптації до показників сенсорів, а також маневрування в умовах змінного середовища.

Окремо варто зазначити сумісність Python з широким колом апаратного забезпечення, включаючи популярні обчислювальні платформи, як-от Raspberry Pi. Це дозволяє безпосередньо запускати код модуля управління на самій платформі дрона, без необхідності у складних компіляціях чи адаптаціях. Крім того, Python має чудову підтримку для мережевої взаємодії — як у форматі TCP/UDP-зв’язку, так і через HTTP/REST API, що спрощує інтеграцію з іншими модулями системи.

Бібліотеки типу NumPy, asyncio, pandas, а також спеціалізовані пакети для роботи з протоколами (наприклад, pyserial) дозволяють обробляти вхідні потоки, формувати логіку PID-регуляторів, здійснювати фільтрацію даних, обчислювати похибки та адаптувати траєкторію польоту в реальному часі. Крім того, Python використовується для формування REST API, через яке мобільний застосунок може взаємодіяти з бортовою логікою.

2.2.2 Flask

Фреймворк Flask було обрано як основу для реалізації серверної логіки через його простоту та гнучкість. Для даного проєкту, в якому необхідно забезпечити передачу даних між мобільним застосунком і модулем дрона, не потрібна важка інфраструктура або складна серверна архітектура — Flask ідеально підходить для побудови REST API, яке просто налаштовується та швидко інтегрується з клієнтською частиною.

Ще однією важливою перевагою Flask є сумісність з багатьма бібліотеками Python, що активно використовуються в цьому проєкті для обробки даних, логіки маршрутування та роботи з потоками. Крім того, Flask легко розгортається на пристроях з обмеженими ресурсами, таких як Raspberry Pi, що особливо важливо для обчислювального модуля дрона. Наявність великої спільноти та якісної документації також значно спрощує підтримку та доопрацювання серверної частини в процесі розробки та тестування.

Flask також дозволяє зручно реалізувати обробку запитів у форматі JSON, взаємодію з чергами подій і синхронізаційними механізмами, що є критично важливим для побудови системи з обробкою даних у реальному часі. Завдяки своїй архітектурі він легко масштабується, що дозволяє у майбутньому додавати нову функціональність без повного переписування серверної логіки.

2.2.3 Kotlin

Мова програмування Kotlin була обрана для реалізації мобільного застосунку завдяки її офіційній підтримці з боку Google для Android-розробки. Kotlin є сучасною, лаконічною мовою, яка дозволяє зменшити об’єм коду в порівнянні з Java, водночас зберігаючи повну сумісність з існуючими Android-бібліотеками. Це спрощує реалізацію інтерфейсів та роботу з API.

Крім того, Kotlin активно розвивається, має широку спільноту, а Android Studio надає повну підтримку цієї мови, включно з автодоповненням, відлагодженням і інструментами аналізу коду. Це дозволило забезпечити швидку і зручну розробку мобільного клієнта, який є ключовим компонентом системи користувацької взаємодії.

У проєкті Kotlin відповідає за реалізацію логіки підключення до REST API, відображення актуального стану дрона (висота, швидкість, координати), виведення повідомлень про події (наприклад, активація алгоритмів ухилення), а також керування польотом з мобільного пристрою. Підтримка корутин у Kotlin дозволяє обробляти мережеві запити асинхронно, не блокуючи інтерфейс користувача.

2.3 Betaflight

Betaflight — це програмне забезпечення, яке використовується для налаштування і керування параметрами польоту багатьох типів мультикоптерів і дронів. Воно виконує роль середовища конфігурації польотних контролерів, надаючи користувачам зручний інтерфейс для тонкої настройки апаратних та програмних параметрів. Завдяки Betaflight, пілоти можуть адаптувати дрони під свої індивідуальні потреби, покращуючи стабільність, керованість та загальну продуктивність польоту.

Основною функцією Betaflight є налаштування параметрів польотного контролера — пристрою, який відповідає за стабілізацію і управління дроном. Серед цих параметрів — чутливість сенсорів, швидкість реакції двигунів, калібрування гіроскопів і акселерометрів, а також налаштування PID-регуляторів, які визначають, як дрон реагує на відхилення від заданого курсу або положення. Betaflight дозволяє не лише змінювати ці налаштування, але й зберігати профілі для різних типів польотів, наприклад, для гонок чи аерофотозйомки.

Інтерфейс Betaflight оснащений графічними та текстовими інструментами, які допомагають аналізувати телеметрію в режимі реального часу. Це дає змогу пілоту відстежувати стан дрона, змінювати параметри польоту «на льоту» і миттєво бачити результати корекцій. Також Betaflight підтримує оновлення прошивки польотного контролера, що дозволяє використовувати нові функції та покращення без необхідності заміни обладнання.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У цьому розділі наведено загальний опис архітектури програмного забезпечення, обчислювальних методів і моделей, використаних для реалізації автономної системи прийняття рішень в реальному часі. Розроблене програмне забезпечення розбите на функціональні підсистеми, кожна з яких виконує окремі завдання, пов’язані з обробкою даних, керуванням дроном та адаптацією до змін у зовнішньому середовищі.

3.1 Архітектура системи

Архітектура проєкту побудована за принципами модульності та слабкого зв’язку між компонентами, що забезпечує масштабованість, легкість в обслуговуванні, відмовостійкість та можливість подальшого розширення функціональності.

Система складається з двох основних компонентів: мобільного застосунку, модуля управління, що працює безпосередньо на Raspberry pi безпілотного літального апарата. Мобільний застосунок виконує роль клієнтського інтерфейсу: через нього користувач задає траєкторію, параметри місії та отримує телеметричні дані. Він розроблений на платформі Android з використанням мови Kotlin, що забезпечує нативну продуктивність, інтеграцію з сенсорами пристрою та сучасний UI.

Модуль управління є ядром автономної логіки системи. Він складається з кількох паралельно працюючих підсистем, зокрема: обробки сенсорних даних, керування польотом, аналізу стану та вибору алгоритму дій. Внутрішня комунікація між підсистемами організована за допомогою потоків та об’єктів синхронізації (зокрема, подій Event та блокувань Lock), що дозволяє уникати

конфліктів при доступі до спільних ресурсів і забезпечувати стабільну роботу в реальному часі.

Схема будови системи представлена на Рис. 3.1.

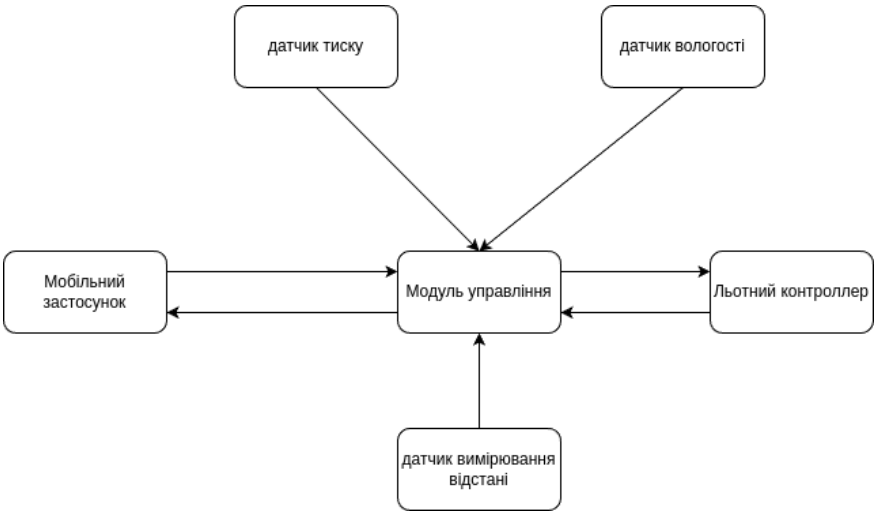


Рис. 3.1 Схема будови системи

3.2 Алгоритми системи

Система автономного керування дроном у реальному часі реалізує низку алгоритмів, які забезпечують стабільний політ, слідування заданій траєкторії та ухилення від перешкод. Кожен з цих алгоритмів є незалежною підсистемою, проте вони взаємодіють між собою за допомогою потокової архітектури та механізмів синхронізації, що забезпечує швидке реагування на зміну умов польоту та дозволяє приймати рішення без втручання оператора.

Алгоритм спілкування з льотним контролером побудований на основі протоколу MSP, що дозволяє передавати дані каналів управління та отримувати телеметричні дані між програмною частиною системи та апаратною платформою дрона. Вся комунікація реалізована через серійний порт з використанням

потокобезпечного механізму, що забезпечує коректний доступ до ресурсу при багатопоточній роботі.

Передача даних відбувається у вигляді спеціалізованих MSP-пакетів, структура яких включає фіксований заголовок (header), тіло повідомлення (body) та контрольну суму (checksum). Заголовок складається з трьох байтів: '$', 'M', '<', що сигналізує про початок повідомлення. Тіло містить довжину корисного навантаження, код операції (opcode) та власне дані. Контрольна сума обчислюється як XOR всіх байтів тіла і дозволяє перевірити цілісність пакета при прийомі.

Формування пакета виконується функцією send_msp_command, яка дозволяє відправити запит з корисним навантаженням (наприклад, байтами RC-команд). Після формування повного пакета (заголовок + тіло + контрольна сума), дані надсилаються в серійний порт із використанням mutex lock для уникнення конфлікту потоків.

На рівні прийому даних реалізований безперервний цикл читання з порту (функція continuous_read), який акумулює байти у внутрішній буфер та виконує пошук початку MSP-пакету. Якщо початок пакета знайдено, далі перевіряється його довжина, правильність контрольної суми та коректність структури. Успішно отримані пакети інкапсулюються в об’єкти класу Packet і передаються в чергу packets для подальшої обробки.

Аналіз отриманих пакетів здійснюється окремим потоком (continuos_parse), який послідовно витягує пакети з черги та, в залежності від їхнього коду операції (opcode), виконує необхідну логіку. Наприклад, пакети з кодом MSP_ATTITUDE містять інформацію про орієнтацію дрона в просторі (roll, pitch, yaw), яка перетворюється з байтів у відповідні значення в градусах і зберігається у черзі для подальшого використання іншими модулями системи. Аналогічно обробляються

дані висоти (MSP_ALTITUDE), канали керування (MSP_RC), які система може як отримувати, так і надсилати.

Передача команд на зміну каналів керування (так звані RC override) реалізується функцією send_rc_command, яка перетворює значення каналів керування (roll, pitch, yaw, throttle, aux) у байтове представлення та відправляє його в MSP-пакеті із відповідним кодом операції. Такий підхід дозволяє системі програмно контролювати рух дрона, симулюючи ручне керування оператором.

На початку роботи система також перевіряє наявність підключення до льотного контролера за допомогою регулярних ping-запитів типу MSP_ATTITUDE (функція wait_msp_connection). У разі отримання відповіді з правильним заголовком, система переходить до активного режиму роботи.

Таким чином, архітектура обміну повідомленнями побудована за принципом асинхронного запит-відповідь, де передача та прийом відокремлені в різних потоках, що дозволяє забезпечити стабільну роботу в реальному часі, навіть при високій частоті обміну даними. Протокол MSP обрано через сумісних із прошивкою Betaflight.

Алгоритм слідування траєкторії. Основою алгоритму слідування траєкторії є PID-регулятор, який використовується для обчислення керуючого впливу з урахуванням поточної позиції дрона, швидкості та заданої точки маршруту. Система регулярно порівнює поточне положення дрона, отримане з внутрішніх сенсорів, із задананими, і формує помилку позиціювання. Ця помилка подається на вхід PID-регулятора, який обчислює відповідні значення для rc override.

PID-контролер обчислює керуючий вплив на об’єкт за допомогою трьох складових — пропорційної (P), інтегральної (I) та диференційної (D). З математичної точки зору, PID-регулятор описується наступною формулою:

$$u(t) = Kp \cdot e(t) + Ki \cdot \int_0^t e(\tau)d\tau + Kd \cdot \frac{de(t)}{dt}$$

де:

u(t) — керуючий сигнал у момент часу t,

e(t)=r(t)−y(t) — похибка між заданим значенням r(t) та виміряним виходом системи y(t),

Kp — коефіцієнт пропорційної складової, яка реагує на поточну похибку,

Ki — коефіцієнт інтегральної складової, яка враховує накопичену похибку в часі,

Kd — коефіцієнт диференційної складової, яка враховує зміну похибки.

Пропорційна частина забезпечує миттєву реакцію, інтегральна — усуває сталу похибку, а диференційна — покращує стабільність, реагуючи на швидкість зміни похибки. PID-контролер забезпечує гнучке та точне регулювання у багатьох системах автоматичного керування, таких як стабілізація дронів, керування температурою або швидкістю.

Для кожної координатної осі використовується окремий регулятор, що дозволяє точно контролювати як горизонтальне, так і вертикальне положення дрона. Також реалізована логіка уповільнення перед досягненням точки, аби уникнути перерегулювання. Після досягнення точки траєкторії алгоритм переходить до наступної контрольної точки або завершує місію.

Алгоритм обходу перешкод. Алгоритм ухилення від перешкод є одним із ключових елементів системи автономного польоту. Його робота базується на безперервному моніторингу даних з інфрачервоного дальноміра. Якщо відстань до

об'єкта попереду стає меншою за критичний поріг, система призупиняє горизонтальний рух і активує процедуру сканування простору.

Для аналізу навколишнього середовища дрон виконує поворот на 180 градусів по горизонталі, паралельно зчитуючи значення з дальноміра на кожному кутовому кроці. На основі отриманих даних формується мапа перешкод навколо дрона. Якщо знаходиться сектор, де немає перешкод або відстань більша за встановлений поріг, система обирає цей напрямок як новий курс.

Після визначення кута безпечного напрямку виконується розрахунок необхідного часу польоту для обходу перешкоди. Використовуючи тригонометричні співвідношення, система обчислює довжину дуги обходу та відстань до перешкоди, а також час, необхідний для повернення на початкову траєкторію. Таким чином, ухилення виконується без втрати загального напрямку польоту. Схему алгоритму представлено на Рис. 3.2.

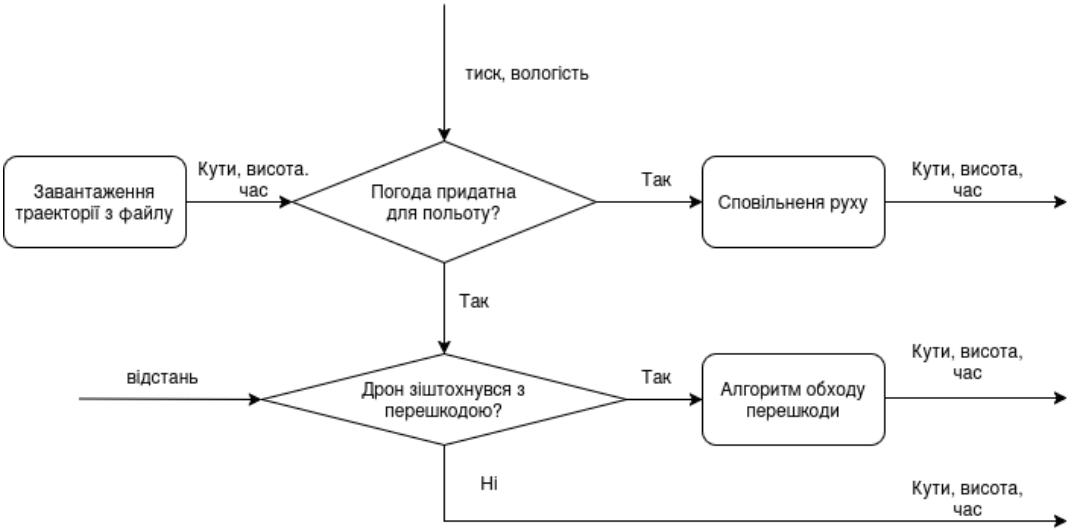


Рис. 3.2 Схема загального алгоритму прийняття рішення

Ці алгоритми є фундаментом автономної поведінки дрона та дають змогу не лише стабільно рухатись у просторі, а й самостійно приймати рішення у складних або змінних умовах навколишнього середовища. Така інтеграція обчислювальних

методів, математичних моделей керування та сенсорної обробки гарантує ефективну роботу системи в режимі реального часу.

3.3 Будова модуля управління

Модуль управління є центральним елементом програмної архітектури дрона, який відповідає за прийняття рішень, формування команд керування, моніторинг стану польоту, а також взаємодію з мобільним застосунком. Його програмна реалізація базується на модульно-багатопотоковій структурі з суворим розділенням відповідальностей між компонентами. Це забезпечує масштабованість, стійкість до збоїв та здатність працювати в реальному часі. На рис 3.3 зображена схема взаємодії компонентів модуля управління.

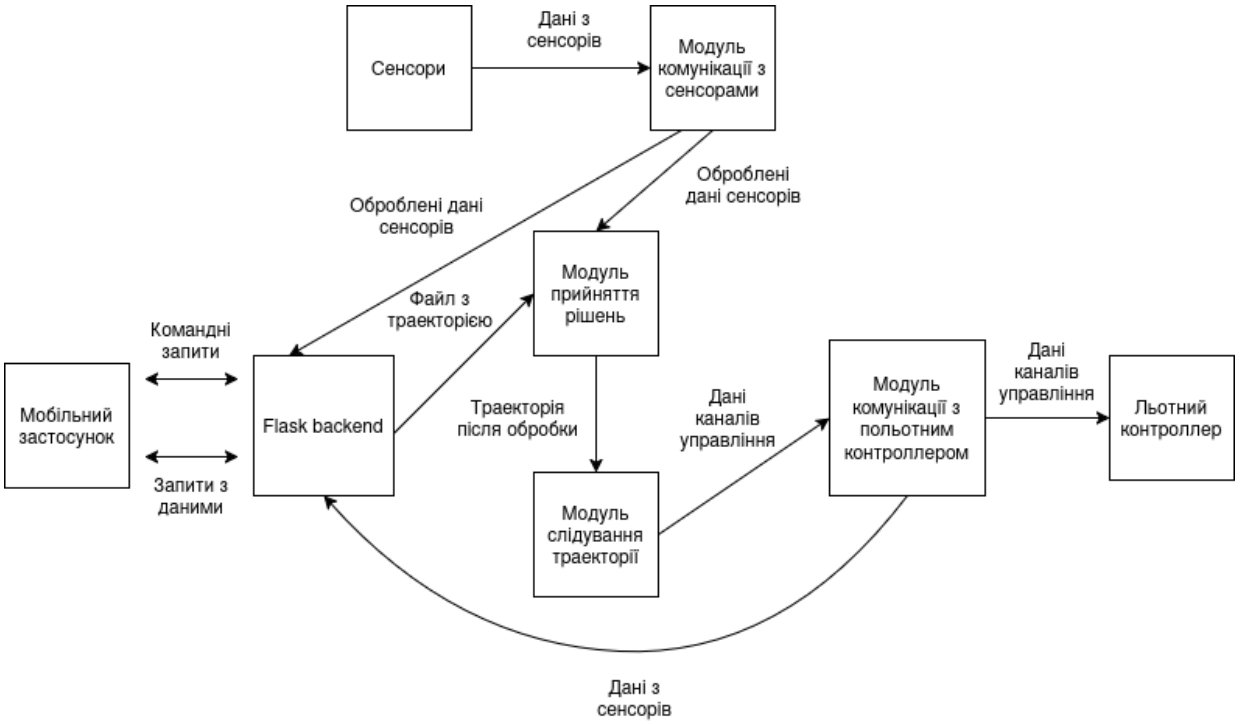


Рис. 3.3 Схема взаємодії компонентів модуля управління

Слід розпочати з найнижчого рівня системи а саме – модуль комунікації з льотним контроллером. Модуль управління безпосередньо взаємодіє з льотним контролером на базі прошивки Betaflight через серійний порт, використовуючи

протокол MSP (Multiwii Serial Protocol). Цей протокол дозволяє надсилати команди до контролера (наприклад, значення каналів RC) та отримувати від нього телеметрію (орієнтацію, висоту, дані IMU).

Паралельно працює окремий потік parser (continuos_parse), який витягує пакети з черги packets, інтерпретує їх залежно від типу (opcode), перетворює байтові масиви у фізичні значення (наприклад, int16 в кутові значення roll/pitch/yaw у градусах) та поміщає отриману інформацію у спеціалізовані черги (наприклад, msg_att_queue для орієнтації, msg_alt_queue для висоти).

Цей підхід дозволяє ізолювати обробку даних від телеметрії та забезпечити безпечний доступ до актуального стану дрона з боку інших потоків, зокрема модулів контролю та ухилення від перешкод.

Основний рух дрона у просторі контролюється PID-регуляторами, реалізованими у вигляді окремого контролера, що постійно отримує цільову траекторію з мобільного застосунку, а також актуальні дані з msg_att_queue, msg_alt_queue. Для кожної координатної осі окремо розраховується похибка, яку обробляє PID-регулятор з відповідними коефіцієнтами пропорційної, інтегральної та диференційної складових.

На основі цього контролер формує команду на канали RC (Roll, Pitch, Throttle), яка надсилається у MSP-пакеті через функцію send_rc_command. Частота оновлення таких команд — приблизно 30 Гц. Це дозволяє забезпечити стабільну траєкторію з мінімальними коливаннями навіть у наявності турбулентностей.

Модуль перешкод отримує вхідні дані від інфрачервоного сенсора. У випадку виявлення перешкоди на критично близькій відстані активується алгоритм обходження перешкод, який використовує модуль слідування траекторії для початку сканування та руху в обхід перешкоди.

3.4 Структурна схема модуля управління

Модуль управління, реалізований на платформі Raspberry Pi, виступає центральним обчислювальним вузлом системи автономного керування дроном. Він забезпечує координацію між сенсорними модулями, льотним контролером та комунікаційними інтерфейсами. У структурній схемі саме Raspberry Pi є точкою з'єднання всіх периферійних пристроїв — як цифрових, так і аналогових — через стандартизовані протоколи передачі даних.

Для зчитування даних із сенсорів, таких як ультразвуковий далекомір або сенсор глибини, використовується інтерфейс GPIO, що працює в режимі тригер-відлуння. Показники відстані фіксуються в мікросекундах і конвертуються в метри.

Комунікація між Raspberry Pi та льотним контролером реалізується через послідовний порт UART. Для обміну даними, використовується протокол MSP, розроблений для передачі команд RC, отримання телеметрії, даних про орієнтацію, висоту, а також для відправки команди ручного керування (override). Raspberry Pi надсилає пакети MSP з обчисленими параметрами керування, а у відповідь отримує телеметричні дані, що необхідні для прийняття рішень у реальному часі.

Крім того, для локального та віддаленого контролю, Raspberry Pi підтримує комунікацію з мобільним застосунком через Wi-Fi, де Flask-сервер обробляє HTTP-запити. Таким чином, структурна схема передбачає використання кількох протоколів — UART (MSP) для критично важливої комунікації з льотним контролером, GPIO для сенсорних входів та TCP/IP для високорівневого керування, що забезпечує модульність та розділення відповідальностей між фізичними компонентами системи. На Рис 3.4 зображена структурна схема модуля управління.

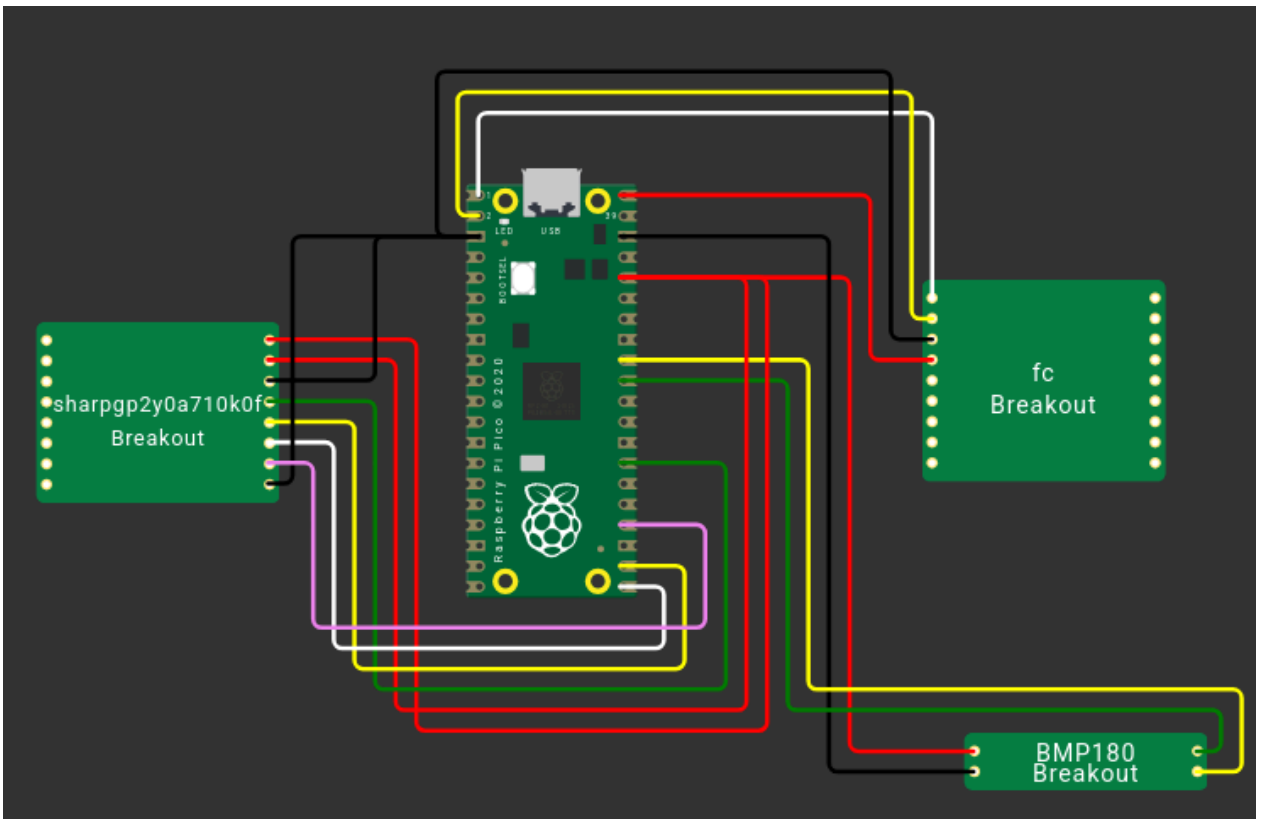


Рис. 3.4 Структурна схема модуля управління

3.5 Будова мобільного застосунку

Мобільний застосунок системи автономного керування дроном розроблений відповідно до сучасних принципів Android-архітектури з використанням компонентів Activity, Fragment та Navigation Component. Застосунок реалізує інтерфейс користувача, а також забезпечує двосторонній обмін інформацією з модулем управління на Raspberry Pi через Flask-сервер. На Рис. 4.1 зображена схема будови мобільного застосунку.

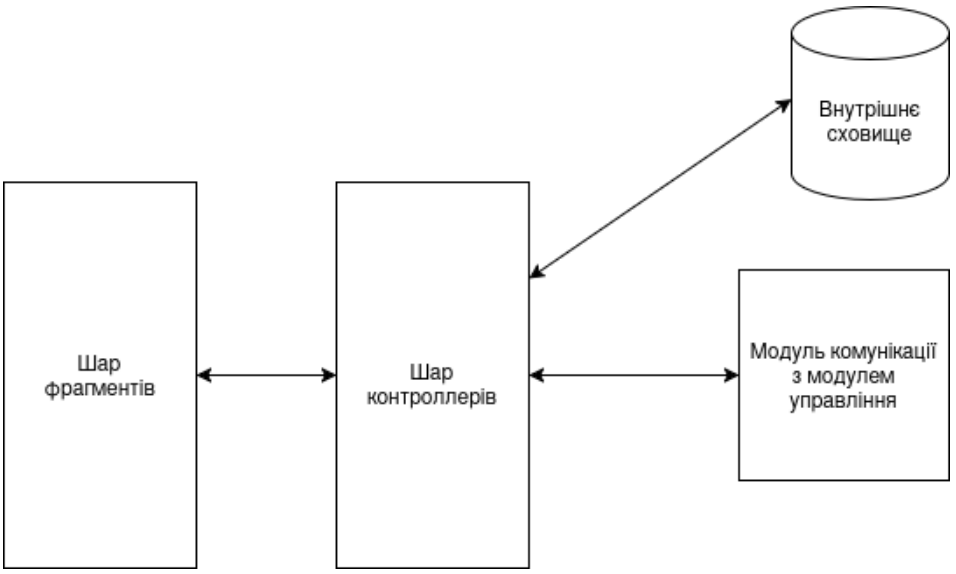


Рис. 4.1 Схема будови мобільного застосунку

Завантаження застосунку починається з Activity StartActivity. Його головна задача — автоматичне виявлення з’єднання із Wi-Fi мережею, створеною дрон-сервером. У цьому Activity реалізовано періодичну перевірку підключення до IP-адреси дрона (Flask-сервера), після чого здійснюється автоматичний перехід до MainActivity. Для цього використовується об’єкт Handler з Runnable, що кожні 2 секунди перевіряє стан Wi-Fi адаптера, порівнює локальну IP-адресу з очікуваною, і при успішному з’єднанні викликає redirectToMainActivity().

MainActivity виступає контейнером для двох основних функціональних фрагментів: CardsFragment і ControlsFragment. Управління навігацією реалізовано за допомогою NavHostFragment та BottomNavigationView, що забезпечує модульність інтерфейсу та легкість розширення системи.

У CardsFragment відображається колекція попередньо визначених маршрутів. Фрагмент використовує RecyclerView з GridLayoutManager для презентації RouteCard-об’єктів, кожен з яких містить інформацію про назву маршруту та ілюстрацію.

RouteDetailFragment є центральним вузлом взаємодії з автоматизованим польотом дрона. Його основна функція — виведення інформації про орієнтацію дрона в реальному часі (yaw, pitch, roll), тривалість місії, а також контроль за її запуском і зупинкою.

Дані про орієнтацію отримуються через періодичні GET-запити до Flask-сервера, що розгорнутий на Raspberry Pi (точка доступу: /get_attitude). Сервер, у свою чергу, отримує ці значення з льотного контролера або обчислює їх з набору сенсорів.

Функції startMission() та stopMission() виконують HTTP POST-запити до відповідних endpoint-ів (/start_mission та /end_mission) та передають у заголовку унікальний ідентифікатор клієнта. Це дозволяє серверу забезпечити сесійну безпеку та керування доступом.

ControlsFragment призначений для реалізації прямого керування дроном за допомогою віртуальних джойстиків, розташованих на екрані. Його концепція базується на перетворенні жестів користувача у числові значення, що відповідають управляючим сигналам по каналах: throttle, pitch, roll і yaw.

В фрагменті присутнє використання двох віртуальних джойстиків, розташованих ліворуч і праворуч екрана, кожен з яких обробляє двовісний рух. Лівий джойстик відповідає за throttle і yaw, правий — за pitch і roll. Значення координат джойстиків нормалізується до певного діапазону (від 1000 до 2000) і формує JSON-об’єкт типу:

Цей об’єкт надсилається POST-запитом до Flask-серверу за маршрутом, наприклад, /update_rc. Передача відбувається у реальному часі в окремому потоці. На стороні сервера дані оброблюються, і надсилаються до льотного контролера через інтерфейс UART.

Таким чином, ControlsFragment відіграє роль віртуального пульта дистанційного керування, надаючи користувачеві змогу вручну маневрувати дроном без потреби у фізичному контролері.

RouteDetailFragment і ControlsFragment реалізують дві частини однієї системи: перший дозволяє запускати місії і відслідковувати стан дрона, другий — переходити в режим прямого керування.

4 Робота користувача з програмною системою

У межах поставленого завдання було створено веб-застосунок для визначення класу енергоефективності будівель. У цьому розділі наведено опис типових сценаріїв взаємодії користувача із системою та подано рекомендації щодо її використання.

4.1 Системні вимоги

Мобільний застосунок розроблений для платформи Android та орієнтований на стабільну роботу на сучасних мобільних пристроях, що відповідають наведеним нижче технічним вимогам. Вимоги визначено з урахуванням функціональності застосунку.

Мінімальні системні вимоги:

- Операційна система: Android 8.0 (Oreo) або новіша
- Процесор: ARM Cortex-A53 або еквівалент з тактовою частотою від 1.4 ГГц
- Оперативна пам’ять: 2 ГБ
- Вільне місце в пам’яті: щонайменше 50 МБ
- Дисплей: з роздільною здатністю не нижче 720x1280 px
- Підключення: модуль Wi-Fi (обов’язково для з’єднання з модулем керування)
- Інші вимоги: підтримка OpenGL ES 3.0 або новішої версії

Рекомендовані системні вимоги:

- Операційна система: Android 10 або новіша
- Процесор: Octa-core
- Оперативна пам’ять: 4 ГБ або більше

- Вільне місце в пам’яті: 50 МБ
- Дисплей: Full HD (1080x1920 px)
- Підключення: стабільне Wi-Fi з’єднання, бажано з можливістю ручного вибору мережі

Застосунок не потребує постійного доступу до Інтернету, але для успішної роботи повинен бути підключений до локальної Wi-Fi мережі дрона. У разі відсутності зв’язку з цільовим IP, застосунок повідомляє користувача та періодично перевіряє статус мережі до встановлення з’єднання.

Ці вимоги дозволяють забезпечити безперебійну роботу інтерфейсу, точну візуалізацію орієнтації дрона, а також стабільний обмін даними з модулем керування.

4.2 Стартовий фрагмент

Після запуску мобільного застосунку користувач потрапляє на стартовий екран, який автоматично виконує перевірку мережевого з’єднання з дроном. Візуально екран виглядає просто: на ньому розміщено повідомлення про поточний стан з’єднання, яке оновлюється в реальному часі.

Механізм перевірки працює без участі користувача: застосунок негайно після запуску перевіряє, чи увімкнено Wi-Fi на пристрої, і чи підключено його до очікуваної мережі дрона. Якщо Wi-Fi увімкнено, але з’єднання з неправильним пристроєм (IP-адреса не відповідає потрібній), користувач бачить повідомлення “Connected to wrong IP”, що дозволяє швидко зрозуміти проблему та змінити мережу. Якщо взагалі немає IP-адреси, з’являється повідомлення “WiFi connected but no IP address”.

Якщо все налаштовано коректно — Wi-Fi увімкнений, і пристрій підключено саме до модуля дрона — застосунок автоматично генерує унікальний

ідентифікатор користувача, зберігає його, і миттєво переходить до головного екрана, де доступні всі функції управління та взаємодії з дроном.

Окрім початкової перевірки, система кожні 2 секунди повторно перевіряє стан з’єднання. Це означає, що користувачу не потрібно перезапускати програму — якщо мережу було підключено або виправлено вручну, застосунок автоматично реагує і запускається далі. На рис 4.1 зображено приклад початкового екрану в стані очікування підключення до мережі Wi-Fi.

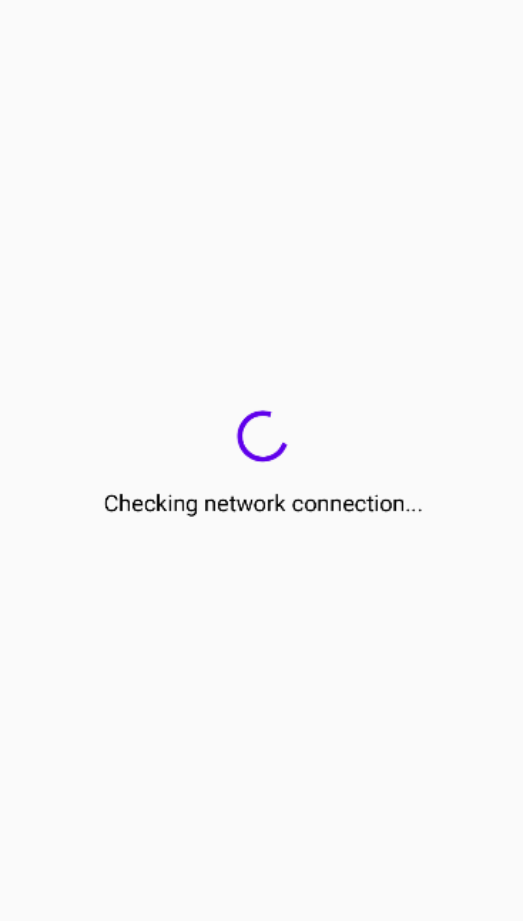


Рис. 4.1 Початковий екран в стані очікування підключення до мережі Wi-Fi

4.3 Вибір місії

Інтерфейс вибору місії є початковим екраном, з якого користувач починає взаємодію з системою автономного управління дроном. Після відкриття цього

фрагмента користувачу відображається список доступних місій. Кожна місія може мати візуальний індикатор маршруту. На Рис. 4.2 представлено зображеня меню вибору місії.

Рис. 4.2 Меню вибору місії

Користувач переглядає доступні місії й обирає одну з них для перегляду детальнішої інформації. При натисканні на будь-яку місію система автоматично переходить до екрану з детальним описом маршруту — RouteDetailFragment.

Після переходу до RouteDetailFragment користувач потрапляє на спеціалізований інтерфейс, призначений для перегляду та контролю виконання

автономної місії дрона. На Рис. 4.3 зображено екран керування місією під час її виконання.

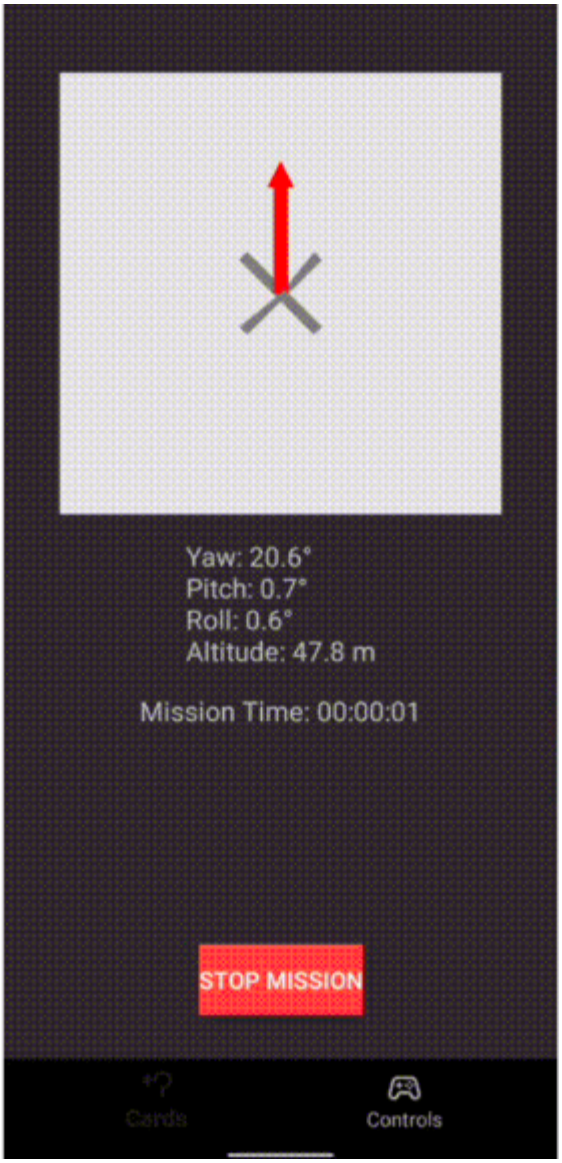


Рис. 4.3 Екран керування місією під час її виконання

Основні елементи та їх призначення:

- Візуалізація орієнтації дрона. У верхній частині екрана розташовано інтерактивну графічну область, яка реалізована на основі OpenGL через GLSurfaceView. У ній відображається 3D-модель, що динамічно змінює положення відповідно до отриманих з дрона даних про орієнтацію (yaw, pitch,

roll). Це дозволяє користувачу в реальному часі бачити, як саме орієнтований дрон у просторі під час польоту.

- Текстова інформація про положення:

Безпосередньо під 3D-візуалізацією розміщено текстове поле, де кожну секунду оновлюються числові значення yaw, pitch, roll. Ця інформація дає змогу точніше оцінити положення дрона.

- Кнопка управління місією. У нижній частині фрагмента знаходиться головна кнопка для запуску та зупинки місії. Її вигляд (колір і напис) змінюється залежно від поточного стану місії: зелена кнопка з написом «Start Mission» або червона кнопка з написом «Stop Mission». Натиснувши кнопку, користувач надсилає відповідний запит на сервер дрона, який або запускає, або завершує виконання місії. Успішність дії підтверджується коротким повідомленням, яке з’являється внизу екрана.
- Таймер виконання місії. Після запуску місії починається відлік часу. У полі з назвою "Mission Time" відображається тривалість виконання польоту.
- Повідомлення про помилки. Якщо під час запитів виникає помилка (втрачено з’єднання, неправильна IP-адреса або відповідь сервера відсутня), інтерфейс повідомляє про це — текстове поле змінюється на «Error fetching attitude», а спроби запиту тривають із кожною наступною ітерацією. Подібна поведінка застосовується й у випадку, якщо не вдається запустити або завершити місію — тоді користувач отримує повідомлення про помилку із зазначенням її причини.

Загалом, RouteDetailFragment надає користувачу повноцінний контроль над місією — від її ініціації до завершення — з одночасною візуалізацією, технічними показниками та часовим трекінгом.

4.4 Фрагмент ручного керування

Фрагмент ручного керування (ControlsFragment) є частиною інтерфейсу мобільного застосунку, яка надає користувачу можливість керувати дроном у режимі реального часу за допомогою сенсорного управління. Основним елементом цього фрагмента є два інтерактивні віртуальні джойстики, що відображаються на екрані. Кожен джойстик реагує на дотик користувача, дозволяючи змінювати положення дрона у просторі.

Інтерфейс реалізований так, що користувач може інтуїтивно керувати напрямком і швидкістю руху: при натисканні та переміщенні пальця по джойстику, його положення зміщується відносно центру, а після відпускання — автоматично повертається у вихідну позицію.

Під час руху джойстиків система розраховує координати зміщення від центру та нормалізує їх у діапазоні від 1000 до 2000. Ці координати формуються у вигляді JSON-пакету та передаються на дрон. У відповідь на отримані значення дрон виконує відповідні дії.

З погляду користувача, цей фрагмент забезпечує простий, зручний та візуально зрозумілий спосіб управління дроном без потреби у зовнішньому контролері. Достатньо лише взаємодії з екраном смартфона. Інтерфейс не перевантажений і дозволяє зосередитися безпосередньо на керуванні польотом. На Рис. 4.4 зображено екран ручного керування до початку польоту.

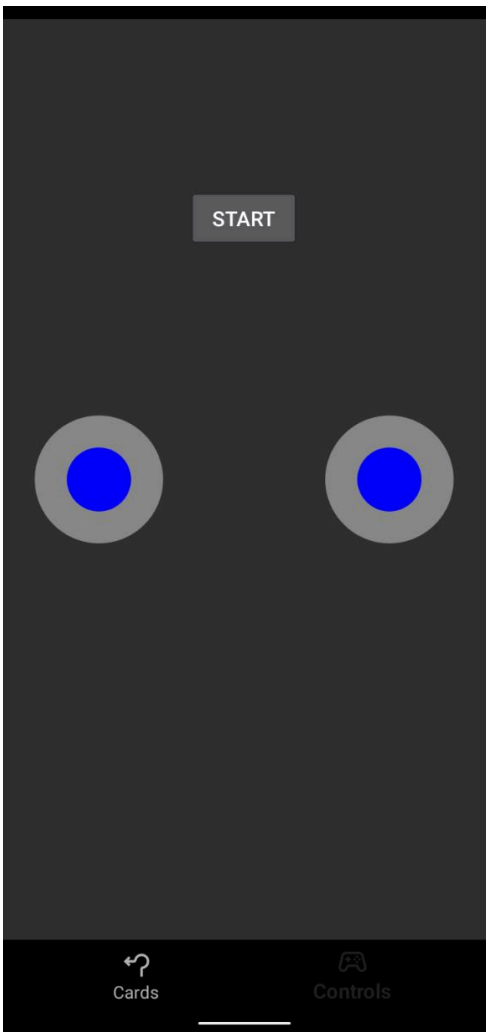


Рис. 4.5 Екран ручного керування до початку польоту

ВИСНОВКИ

У межах виконання дипломної роботи було проведено комплексне дослідження, спрямоване на розробку програмного забезпечення для автономної системи прийняття рішень у реальному часі.

Результатом роботи стала реалізація функціональної програмної системи, що складається з двох основних компонентів: модуля керування на базі Raspberry Pi та мобільного застосунку на платформі Android. Розроблена система забезпечує автономну навігацію дрона, ухилення від перешкод, адаптацію до змінних умов середовища та взаємодію з користувачем у режимі реального часу. Архітектура системи побудована за принципами модульності, що забезпечує гнучкість, масштабованість і можливість подальшого розширення функціональності.

Завдяки використанню сучасних інструментів і мов програмування (Python, Kotlin, Flask), було досягнуто високої ефективності обробки даних, стабільності роботи в багатопоточному середовищі та зручності користувацької взаємодії. Також впроваджено механізми обробки телеметричних даних, динамічного маршрутування та візуалізації стану польоту.

Практичне значення розробленої системи полягає в її застосуванні у сферах, що вимагають автономної поведінки: моніторинг об’єктів, пошуково-рятувальні місії, дослідження територій. Завдяки універсальній архітектурі програмне забезпечення може бути адаптоване для інших типів автономних платформ та розширене під додаткові алгоритми прогнозування, прийняття рішень і машинного навчання. Отримані результати демонструють потенціал для подальших досліджень і впроваджень у сфері інтелектуальних кібер-фізичних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Yang Y, Xiong X, Yan Y. UAV Formation Trajectory Planning Algorithms: A Review. *Drones*. 2023; 7(1):62.

2. Borenstein J, Koren Y. Real-Time Obstacle Avoidance for Fast Mobile Robots. *IEEE Transactions on Systems, Man, and Cybernetics*. 1989; 19(5):1179-1187.

3. Protic Z. *Modern Android Development with Kotlin Coroutines and Flow.* Journal of Mobile Development. 2022; 7(2):45-60.

4. Van Rossum G., Drake F.L. Python Reference Manual / G. van Rossum, F.L. Drake. – Python Software Foundation, 2011. – Version 3.8.

5. Documentations for the Flask Web Framework. URL: https://flask.palletsprojects.com/

ДОДАТОК А Лістинг розробленої системи

Код модуля управління:

PID.py:

```
class PID:
def __init__(self, kp, ki, kd, setpoint=0):
self.kp = kp
self.ki = ki
self.kd = kd
self.setpoint = setpoint
self.last_error = 0
self.integral = 0

def update(self, current_value, dt):
error = self.setpoint - current_value
self.integral += error * dt
derivative = (error - self.last_error) / dt if dt > 0 else 0
self.last_error = error
return self.kp * error + self.ki * self.integral + self.kd * derivative

def reset_error(self):
self.integral = 0
```

api.py:

```
from flask import Flask, request, jsonify
from threading import Lock, Event, Thread
import time
from diploma import start_communication, run_mission_control_loop, arm, disarm, put_rc_in_queue
from events.mission_stop_event import mission_stop_event
app = Flask(__name__)

trajectory = []
trajectory_lock = Lock()
start_time = None
```

```
# Store current drone state
current_state = {
'timestamp': 0.0,
'pitch': 0.0,
'roll': 0.0,
'yaw': 0.0,
'altitude': 0.0
}
state_lock = Lock()
mission_lock = Lock()
mission_thread = None

authorized_id = None
id_lock = Lock()

@app.before_request
def validate_client_id():
global authorized_id
client_id = request.headers.get("X-Client-ID")

# Allow first request to set the ID
with id_lock:
if authorized_id is None:
if client_id:
authorized_id = client_id
print(f"[INFO] Authorized client ID set to: {authorized_id}")
return
else:
return jsonify({"error": "Missing X-Client-ID in first request"}), 400

# For all other requests: check the ID
if client_id != authorized_id:
return jsonify({"error": "Unauthorized client ID"}), 403
```

```
@app.route('/start', methods=['GET'])
def start():
return jsonify("Prototype2")

# Endpoint to upload trajectory data
@app.route('/upload_trajectory', methods=['POST'])
def upload_trajectory():
global trajectory, start_time
if 'file' not in request.files:
return jsonify({'error': 'No file part in request'}), 400

file = request.files['file']
if file.filename == '':
return jsonify({'error': 'No selected file'}), 400

# Save file to disk
save_path = 'current_trajectory.csv'
file.save(save_path)
return jsonify({'status': 'Trajectory uploaded'}), 200

# Endpoint to get current desired point from trajectory
@app.route('/get_current_point', methods=['GET'])
def get_current_point():

#todo: HAVE TO RETURN PERCENTAGE
global trajectory
now = time.time() - start_time if start_time else 0

with trajectory_lock:
if not trajectory:
return jsonify({'error': 'No trajectory loaded'}), 404
```

```
point = None
for i, t in enumerate(trajectory):
if t['timestamp'] >= now:
point = t
break
if point is None:
point = trajectory[-1]

return jsonify(point), 200

# NEW: Endpoint to update drone's current state
@app.route('/update_rc', methods=['POST'])
def update_position():
global current_state, mission_status
if mission_status:
data = request.get_json()
required_keys = ['pitch', 'roll', 'yaw', 'throttle']
if not all(k in data for k in required_keys):
return jsonify({'error': 'Missing keys in position update'}), 400

put_rc_in_queue({
'pitch': data['pitch'],
'roll': data['roll'],
'yaw': data['yaw'],
'throttle': data['throttle']
})

return jsonify({'status': 'Current position updated'}), 200
else:
return jsonify({'status': 'Mission is not active'}), 400

@app.route('/disable_manual_control', methods=['POST'])
def disable_manual_control():
global mission_active
```

```
with mission_lock:
mission_active = False
disarm()
return jsonify({'status': 'Control Disabled'}), 200

@app.route('/enable_manual_control', methods=['POST'])
def enable_manual_control():
global mission_active
with mission_lock:
mission_active = True
arm()
return jsonify({'status': 'Control Enabled'}), 200

# NEW: Endpoint to get current drone position
@app.route('/get_current_position', methods=['GET'])
def get_current_position():
with state_lock:
return jsonify(current_state), 200

@app.route('/start_mission', methods=['POST'])
def start_mission():
global mission_active, mission_thread # Add this line
with mission_lock:
mission_active = True
mission_stop_event.clear()
arm()
mission_thread = Thread(target=run_mission_control_loop, args=(True,))
mission_thread.start()
return jsonify({'status': 'Mission started'}), 200

@app.route('/end_mission', methods=['POST'])
def end_mission():
global mission_active
with mission_lock:
```

```
if not mission_active:
return jsonify({'status': 'Mission is not active'}), 400
mission_active = False
mission_stop_event.set()
if mission_thread and mission_thread.is_alive():
mission_thread.join(1)
disarm()
return jsonify({'status': 'Mission ended'}), 200

@app.route('/mission_status', methods=['GET'])
def mission_status():
with mission_lock:
return jsonify({'status': mission_active}), 200

if __name__ == '__main__':
start_communication()
app.run(host='0.0.0.0', port=5000)

mission_control.py:
def run_mission_control_loop(from_request=False):
current_time = datetime.datetime.timestamp(datetime.datetime.now())
msg_waiting_start = datetime.datetime.timestamp(datetime.datetime.now())
counter = 0
log_buffer = ""
read_line_num = 0
homing_failsafe = False
drone_pitch = 0
drone_roll = 0
drone_yaw = 0
drone_altitude = 0
chan12_val = 0
chan1_val = 0
msg = None
arrived_at_start = False
```

```
pid_roll = PID(kp=1.3, ki=0.6, kd=0.1)
pid_pitch = PID(kp=1.4, ki=0.8, kd=0.12)
pid_yaw = PID(kp=1.2, ki=0.5, kd=0.08)
pid_altitude = PID(kp=0.035, ki=0.07, kd=0.005)

with open(f"result_{current_time}.csv", 'a+') as f:
f.write("roll, pitch, throttle, yaw, aux1, aux2, interval, pitch, roll\n")

while not mission_stop_event.is_set():
cycle_start_time = datetime.datetime.timestamp(datetime.datetime.now())
counter += 1
packet_flag_list = [False, False]

if (msg_rcin_queue.empty() == False):
msg = msg_rcin_queue.get()
chan11_val = msg[10]
chan12_val = msg[11]
rc_throttle_bypass = msg[3]
rc_rll_bypass = msg[0]
if from_request:
chan1_val = 2000
else:
chan1_val = msg[4]
chan8_val = msg[7]

if (msg_att_queue.empty() == False):
# att_que_size = mav_msg_att_queue.qsize()
msg_att = msg_att_queue.get()
drone_pitch = msg_att['pitch'] # update current drone pitch
drone_roll = msg_att['roll']
drone_yaw = msg_att['yaw']
packet_flag_list[0] = True

if (msg_alt_queue.empty() == False):
```

```
msg_alt = msg_alt_queue.get()
drone_altitude = msg_alt['alt']
packet_flag_list[1] = True

is_packets_updated = sum(packet_flag_list) > 0

if is_packets_updated:
if not homing_failsafe:
if chan1_val < 1900:
homing_failsafe = True

if homing_failsafe:
if chan1_val > 1900:
homing_failsafe = True

if chan1_val > 1900:
homing_failsafe = False

# if msg_no_obj:
# log_buffer += f"{msg_no_obj[0]}, {msg_no_obj[1]}, {msg_no_obj[3]}, {msg_no_obj[2]}, {msg_no_obj[4]}, {msg_no_obj[5]}, {interval}, {drone_pitch}, {drone_roll}\n"
# msg_waiting_start = datetime.datetime.timestamp(datetime.datetime.now())
# if counter % 100 == 0:
# with open(f"result_{current_time}.csv", 'a+') as f:
# f.write(log_buffer)
# log_buffer = ""
if is_packets_updated:
# print(homing_failsafe)
if not homing_failsafe:
with open('current_trajectory.csv') as f:
interval = datetime.datetime.timestamp(datetime.datetime.now()) - msg_waiting_start
lines = f.readlines()
line = lines[read_line_num]
line_elements = line.split(',')
desired_pitch = int(line_elements[1])
```

```
desired_roll = int(line_elements[0])
desired_yaw = int(line_elements[2])
desired_altitude = int(line_elements[3])

pid_pitch.setpoint = desired_pitch
pid_roll.setpoint = desired_roll
pid_yaw.setpoint = desired_yaw
pid_altitude.setpoint = desired_altitude

dt = interval if interval > 0 else 0.02 # fallback to 20ms

rc_pitch = 1500 + pid_pitch.update(drone_pitch, dt)
rc_roll = 1500 + pid_roll.update(drone_roll, dt)
rc_throttle = 1220 + pid_altitude.update(drone_altitude, dt)
if arrived_at_start:
rc_yaw = 1500 + pid_yaw.update(drone_yaw, dt)
else:
rc_yaw = 1500
print(pid_altitude.last_error, rc_throttle, counter)

rc_pitch = max(min(int(rc_pitch), 1600), 1400)
rc_roll = max(min(int(rc_roll), 1600), 1400)
rc_yaw = max(min(int(rc_yaw), 1600), 1400)
rc_throttle = max(min(int(rc_throttle), 1300), 1000)

rc_data = {'source': 'PID', 'overroll': rc_roll,
'overrpitch': rc_pitch,
'overrthrottle': rc_throttle,
'overryaw': rc_yaw, 'overaux1': chan1_val,
'overaux2': 1000,
'overaux3': msg[6],
'overaux4': chan5_val}
log_buffer = f"{rc_roll}, {rc_pitch}, {rc_yaw}, {rc_throttle}, {interval}, {drone_pitch}, {drone_roll}\n"
# print(log_buffer)
rc_output_queue.put(rc_data)
```

```
with open(f"result_{current_time}.csv", 'a+') as f:
f.write(log_buffer)
msg_waiting_start = datetime.datetime.timestamp(datetime.datetime.now())
if read_line_num + 1 < len(lines):
read_line_num += 1
if abs(drone_altitude - int(line_elements[3])) < 5:
arrived_at_start = True
if msg and homing_failsafe:
rc_data = {'source': 'PID', 'overroll': msg[0],
'overrpitch': msg[1],
'overrthrottle': msg[3],
'overryaw': msg[2], 'overaux1': msg[4],
'overaux2': msg[5],
'overaux3': msg[6],
'overaux4': msg[7]}
rc_output_queue.put(rc_data)
if counter != 0:
pid_yaw.reset_error()
pid_roll.reset_error()
pid_pitch.reset_error()
pid_altitude.reset_error()
arrived_at_start = False
counter = 0
```

ДОДАТОК В Презентація

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

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ТЕМА: «Програмне забезпечення для автономної системи прийняття рішень в реальному часі»

Виконав: Нересниця Олег Юрійович, ТВ-12
Керівник: Сарибога Ганна Володимирівна, ст. викладач

Київ - 2025

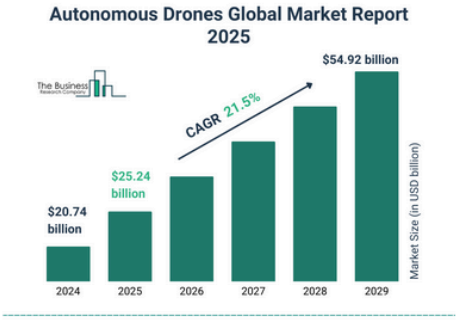
1/12

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

Актуальність теми

Сучасні тенденції розвитку автономних систем формують зростаючий попит на програмне забезпечення, здатне забезпечувати прийняття рішень у реальному часі.

- Попит на ПЗ для автономних систем стрімко зростає завдяки розвитку ШІ та автоматизації.
- Основні сфери застосування — оборона та аграрний сектор. Прогнозоване зростання ринку — 13% до 2031 року.
- Системи мають реагувати на змінні умови середовища й ухвалювати рішення без участі людини.
- Тема є міждисциплінарною — охоплює програмування, кібернетику, ШІ та прикладну математику.



2/12

Мета та основні завдання

Метою проектної роботи є створення програмного забезпечення для автономної системи прийняття рішень у реальному часі, що дозволяє безпілотний платформі реагувати на змінні умови середовища та адаптивно змінювати свою поведінку.

Завдання проекту включають:

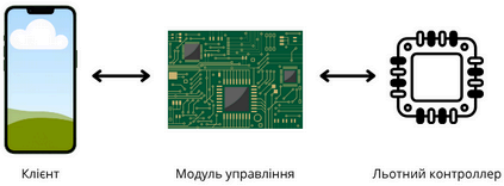
- Аналіз існуючих систем та вимог
- Проектування архітектури системи як набору взаємодіючих модулів з незалежною логікою роботи;
- Розробити модуль прийняття рішень в реальному часі та модуль слідування траєкторії
- Розробити користувацький інтерфейс у вигляді мобільного застосунку для комунікації з дроном
- Протестувати роботу системи на основі прототипу

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

Аналіз існуючих систем

Параметри аналізу	Ardupilot mission planner	QgroundControl	Розроблена система
Візуалізація польоту	Так	Так	Ні (може бути додано в застосунок)
Підтримка платформ	Ardupilot	PX4	Betaflight
Обхід перешкод	Частково (через зовнішні модулі)	Частково	Так
Зміна поведінки в реальному часі	Обмежено	Обмежено	Так
Оновлення параметрів у польоті	Так, підтримка зміни параметрів у реальному часі	Частково (обмежено через інтерфейс)	Так, підтримка зміни параметрів у реальному часі

Архітектура модуля управління системи прийняття рішень



Архітектура системи

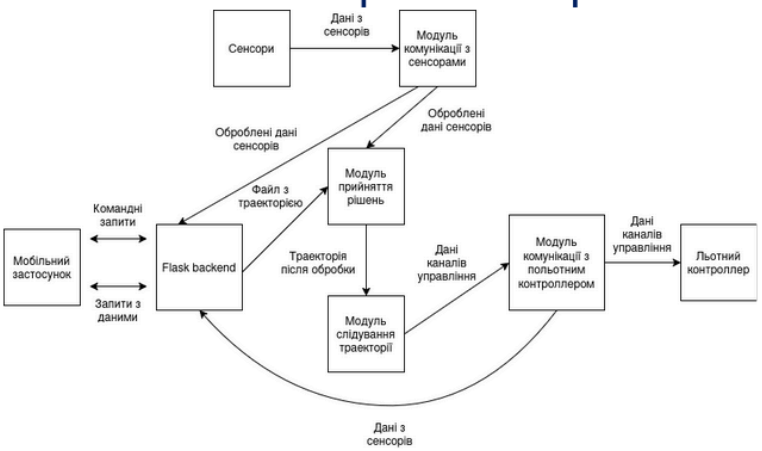
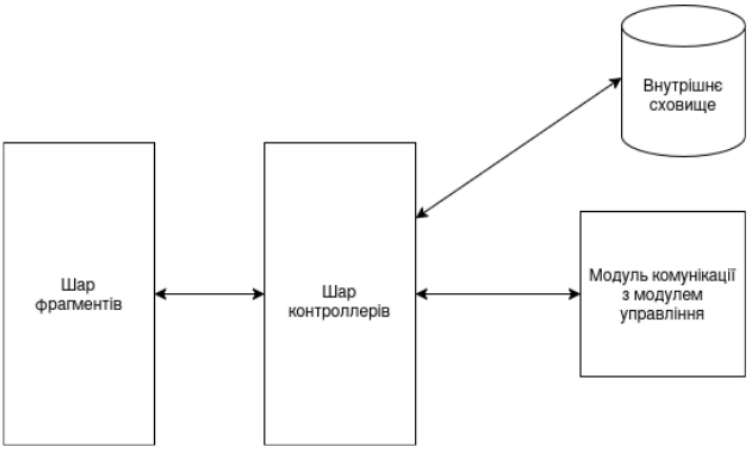
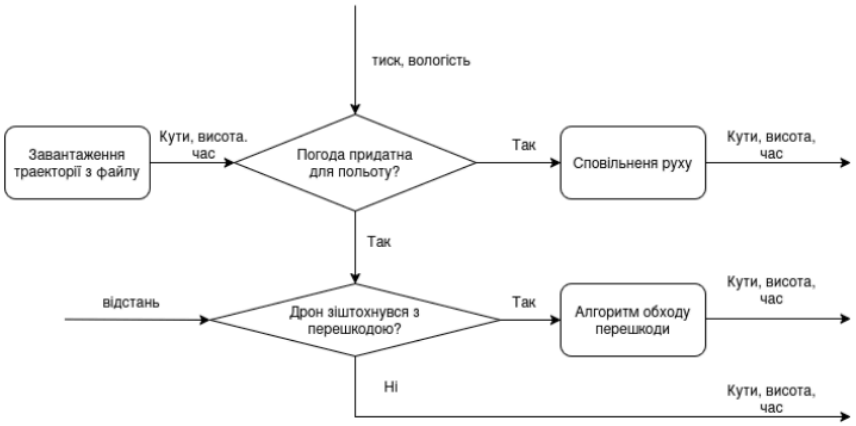


Схема роботи модуля управління

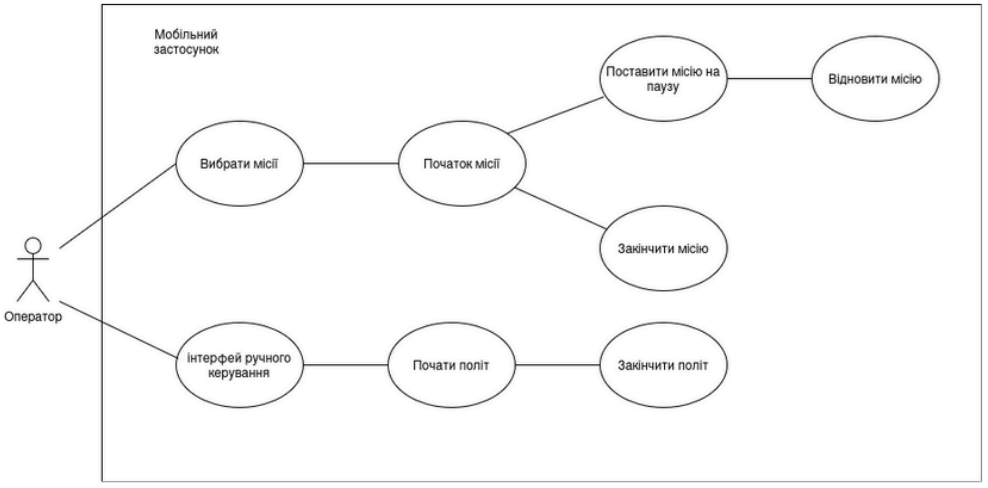
Архітектура мобільного застосунку



Алгоритм прийняття рішення



Use case діаграма

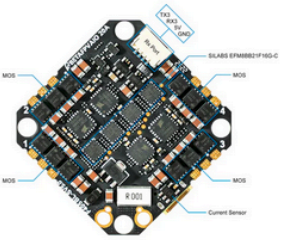


Засоби розробки модулів системи прийняття рішень



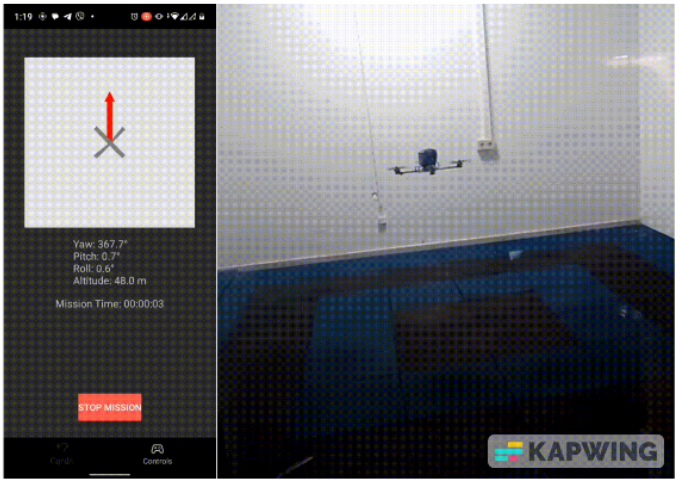


програмні

апаратні

Демонстрація роботи



Метрики працездатності системи

Метрика	Опис	Допустимий інтервал похибки
Частота оновлення стану	Частота оновлення положення, швидкості та орієнтації	≥ 20 Гц
Відхилення по висоті	Максимально допустиме відхилення від заданої висоти	≤ 0.3 м
Час реакції на перешкоду	Час між виявленням об'єкта і запуском алгоритму ухилення	≤ 100 мс
Точність слідування траєкторії	Середнє відхилення між передбаченою та фактичною траєкторією	≤ 1 м
Використання процесора	Середнє навантаження на обчислювальний модуль дрона	≤ 75%

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

Висновки

Було розроблено програмного забезпечення для автономної системи прийняття рішень у реальному часі, що дозволяє безпілотний платформі реагувати на змінні умови середовища та адаптивно змінювати свою поведінку.

- Проведено аналіз існуючих систем та вимог
- Спроектовано архітектуру системи як набору взаємодіючих модулів з незалежною логікою роботи;
- Розроблено модуль прийняття рішень в реальному часі та модуль слідування траєкторії
- Розроблено користувацький інтерфейс у вигляді мобільного застосунку для комунікації з дроном
- Протестовано роботу системи на основі прототипу