Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «**Розробка системи автоматизованого керування для безпілотного літального апарату**»

(тема роботи)

Виконав: студент 2-ого курсу групи 1АКІТР-23м

(шифр групи)

спеціальності 174 - Автоматизація,

комп'ютерно-інтегровані технології та робототехніка

(шифр та назва спеціальності)

Максим СТАНІСЛАВЕНКО

(Ім'я, ПРІЗВИЩЕ студента)

Керівник: к.т.н., доцент каф. АІІТ, _____

Ярослав КУЛИК

(науковий ступінь, вчене звання / посада, Ім'я, ПРІЗВИЩЕ керівника)

« 16 » грудня 2024 р.

Опонент: _____

(науковий ступінь, вчене звання / посада, Ім'я, ПРІЗВИЩЕ опонента)

« 18 » грудня 2024 р.

Допущено до захисту

Завідувач кафедри АІІТ

д.т.н., проф. Олег БІСІКАЛО

(науковий ступінь, вчене звання)

« 19 » грудня 2024 р.

Вінниця ВНТУ - 2024 рік

Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти - ІІ-ий(магістерський)
Галузь знань - 17 - Електроніка, автоматизація та електронні комунікації.
Спеціальність - 174 Автоматизація, комп`ютерно-інтегровані технології та робототехніка.
Інтелектуальні комп'ютерні системи.
Освітня програма - Інтелектуальні комп'ютерні системи.

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
“ 19 ” вересня 2024 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Станіславенку Максиму Михайловичу

1. Тема роботи: «Розробка системи автоматизованого керування для безпілотного літального апарату»
керівник роботи: к.т.н., доцент кафедри АІІТ Кулик Ярослав Анатолійович
затверджено наказом закладу вищої освіти від “ 17 ” 09 2024 року №310
2. Термін подання студентом роботи “ 16 ” 12 2024 року.
3. Вихідні дані до роботи: Apple MacMini на Apple Silicon M2 8GB RAM (як бортовий комп'ютер). Мова програмування бортового комп'ютера Kotlin, Python. Середовище симуляції Unreal Engine 5. Операційна система симуляції Windows/Linux. Ключові технічні характеристики комп'ютера симуляції Ryzen 5 5600, Nvidia RTX4070, 32GB RAM.
4. Зміст розрахунково-пояснювальної записки: Вступ. Аналіз предметної області та огляд існуючих аналогів. Вибір та огляд технологій для розробки бортового комп’ютера. Розробка програмного забезпечення для розв'язання поставлених задач. Тестування систем автопілота. Економічний розділ. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного (або графічного) матеріалу: Діаграми та схеми архітектури. Код налаштування Docker і Docker Compose. Програмний код реалізації модулів керування. Результати тестування системи.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Ярослав КУЛИК к.т.н., доцент каф. АІІТ	18.09.24	15.12.24
5	Володимир КОЗЛОВСЬКИЙ к.е.н., професор каф. ЕПтаВМ	20.10.24	20.11.24

7. Дата видачі завдання “ 18 ” вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз існуючих рішень та теоретична підготовка	23.09.24	Викон.
2	Вибір та огляд технологій розробки для розробки бортового комп`ютера	25.09.24	Викон.
3	Розробка архітектури програмної системи	30.09.24	Викон.
4	Реалізація програмних модулів системи	04.10.24	Викон.
5	Тестування та налагодження системи	09.10.24	Викон.
6	Підготовка економічного розділу	29.11.24	Викон.
7	Оформлення матеріалів до захисту МКР	01.12.24	Викон.
8	Попередній захист МКР	18.12.24	Викон.
9	Остаточний захист МКР	19.12.24	Викон.

Студент _____ Максим СТАНІСЛАВЕНКО
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ярослав КУЛИК
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.896

Станіславенко М. М. Розробка системи автоматизованого керування для безпілотного літального апарату. Магістерська кваліфікаційна робота зі спеціальності 174 - Автоматизація, комп`ютерно-інтегровані технології та робототехніка, освітньо-професійна програма - Інтелектуальні комп’ютерні системи. Вінниця: ВНТУ, 2024. 111 с.

На укр. мові. Бібліогр.: 21 назв; рис.: 26; табл.: 7.

Метою роботи є розробка системи автоматизованого керування для безпілотного літального апарату (БПЛА) на основі модульної архітектури із застосуванням технологій контейнеризації Docker та програмування на мові Kotlin. Система орієнтована на керування віртуальним дроном у середовищі симуляції AirSim, що дозволяє тестувати алгоритми керування та взаємодії модулів у безпечному віртуальному середовищі.

В даній роботі розглянуто аналіз предметної області та існуючих систем автоматизованого керування БПЛА, обґрунтування вибору технологій для побудови системи, засобів контейнеризації Docker і Docker Compose. Реалізовано основні компоненти системи, які включають модуль аналізу та передачі даних для інтеграції з AirSim, модуль навігації для обчислення базових маршрутів у межах карти симулятора, а також модуль керування польотом для виконання команд та забезпечення стабільного керування дроном. Проведено тестування системи у середовищі AirSim, що підтвердило працездатність архітектури та коректну взаємодію між модулями.

Ключові слова: автоматизоване керування, безпілотний літальний апарат, модульна архітектура, симуляція польоту, AirSim, Kotlin, Docker, RPC API, контейнеризація, навігація, керування польотом, аналіз сенсорних даних, алгоритми керування, kotlinx.serialization, мультиплатформеність, асинхронність, симуляційне середовище.

ABSTRACT

Stanislavenko M. M. Development of an Automated Control System for an Unmanned Aerial Vehicle. Master's Thesis in Specialty 174 – Automation, Computer-Integrated Technologies, and Robotics, Educational and Professional Program – Intelligent Computer Systems. Vinnytsia: VNTU, 2024. 111 p.

In Ukrainian language. Bibliography: 21 titles; fig.: 26; tabl.: 7.

The purpose of this work is to develop an automated control system for an unmanned aerial vehicle (UAV) based on a modular architecture using Docker containerization technologies and Kotlin programming language. The system is designed for controlling a virtual drone in the AirSim simulation environment, which allows testing control algorithms and module interactions in a safe virtual environment.

This work examines the analysis of the subject area and existing automated UAV control systems, the justification for the choice of technologies for system development, and the means of containerization with Docker and Docker Compose. The main system components have been implemented, including a data analysis and transfer module for integration with AirSim, a navigation module for calculating basic routes within the simulator map, and a flight control module for executing commands and ensuring stable drone operation. The system was tested in the AirSim environment, which confirmed the operability of the architecture and the correct interaction between modules.

Keywords: automated control, unmanned aerial vehicle, modular architecture, flight simulation, AirSim, Kotlin, Docker, RPC API, containerization, navigation, flight control, sensor data analysis, control algorithms, kotlinx.serialization, multiplatform, asynchrony, simulation environment.

ЗМІСТ

ВСТУП

Актуальність роботи. Розвиток технологій автоматизації відкриває нові горизонти у використанні безпілотних літальних апаратів (БПЛА). Завдяки автономності такі системи можуть виконувати завдання з мінімальною участю людини, що особливо важливо в умовах підвищеного ризику, наприклад, у зоні стихійних лих, військових операціях або моніторингу важкодоступних територій[1].

Зростання попиту на автономні системи керування БПЛА потребує інноваційних рішень у сфері їхньої архітектури, алгоритмів навігації та обробки даних. У цьому контексті дослідження модульних підходів до розробки систем автоматизованого керування, побудованих на сучасних платформах і технологіях, є надзвичайно актуальним.

Метою магістерської кваліфікаційної роботи є скорочення часу на розбору БПЛА при розробці системи автоматизованого керування на основі модульної архітектури із застосуванням контейнерів Docker, програмної логіки на Kotlin, Python, а також випробування її функціональності у симуляційному середовищі Unreal Engine 5 AirSim.

Задачі досліджень магістерської кваліфікаційної роботи:

1. Провести аналіз існуючих аналогів систем автоматизованого керування БПЛА та визначити ключові вимоги до їх функціоналу.

2. Обґрунтувати вибір платформ та інструментів для розробки системи (MacMini на Apple Silicon, Kotlin, Python, Docker, Unreal Engine 5 AirSim).

3. Розробити модульну архітектуру системи на основі контейнерів Docker із забезпеченням взаємодії модулів через API та спільні каталоги.

4. Реалізувати алгоритми керування польотом, навігації, обробки візуальних даних і прокладання маршруту.

5. Провести випробування системи у симуляційному середовищі та оцінити її ефективність за заданими критеріями.

Об’єктом дослідження є процес керування безпілотним літальним апаратом.

Предметом дослідження є Архітектура модульної системи керування, методи взаємодії між її компонентами та алгоритми, що забезпечують виконання завдань автономного польоту.

Методи дослідження. У роботі використовуються методи системного аналізу для визначення вимог до архітектури системи; методи об’єктно-орієнтованого програмування для реалізації функціоналу модулів; методи симуляції для тестування алгоритмів керування в Unreal Engine 5 AirSim; а також емпіричні методи для оцінки продуктивності системи.

Науково-технічний результат отриманих результатів дослідження полягає в тім:

1. Розроблено модульну систему автоматизованого керування БПЛА, яка інтегрує алгоритми польоту, навігації та обробки даних у контейнеризованому середовищі Docker.

2. Вперше використано поєднання Kotlin і Python для створення модульної системи з урахуванням специфіки роботи на платформі Apple Silicon.

3. Система протестована у симуляційному середовищі Unreal Engine 5 AirSim із використанням реалістичних сценаріїв.

Практична цінність Результати роботи можуть бути використані для створення реальних прототипів автономних безпілотників, а також для моделювання та тестування автоматизованих систем у різних сценаріях. Розроблена система може бути адаптована для виконання завдань у цивільній, промисловій та науковій сферах, таких як моніторинг довкілля, картографування територій або доставка вантажів.

Апробація та публікації матеріалів досліджень. Основні результати виконання магістерської кваліфікаційної роботи були опубліковані в матеріалах

Всеукраїнської науково-практичної Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи» (Вінниця, ВНТУ, 2023-2024 рр.) [2].

1 ЗАГАЛЬНІ ВІДОМОСТІ

1.1 Існуючі підходи до автоматизованого керування БПЛА

Системи автоматизованого керування БПЛА, які використовуються на ринку, можна умовно поділити на три основні категорії:

1. Рішення на основі відкритих платформ. ArduPilot — одна з найпоширеніших відкритих платформ, яка пропонує широкий набір функцій для керування літальними апаратами. Вона підтримує роботу з різними типами апаратів (літаки, коптери, наземні роботи) [3]. Основний недолік — обмеженість в адаптації до специфічних сценаріїв. PX4 — ще одна популярна платформа, що підтримує різні апаратні конфігурації. Платформа орієнтована на використання сучасних алгоритмів автономного керування, але вимагає глибоких знань для кастомізації [4].

2. Пропрієтарні системи керування. DJI Flight Control System — закрите рішення від DJI, яке забезпечує високий рівень стабільності, але обмежує доступ до кастомізації алгоритмів [5]. Ці системи переважно орієнтовані на масовий ринок. Intel Aero Platform — апаратно-програмна платформа для розробки автономних дронів, яка має підтримку глибокого навчання та інтеграцію з системами візуального аналізу [6].

3. Дослідницькі системи та платформи. ROS (Robot Operating System) — широко використовується в дослідницьких проектах для моделювання автономного керування БПЛА [7]. ROS підтримує модульну архітектуру, але має високий поріг входу через складність налаштування та інтеграції. Microsoft AirSim — середовище симуляції, яке дозволяє тестувати алгоритми керування та аналізу даних у віртуальному середовищі. AirSim не забезпечує реального керування апаратом, [8] але ідеально підходить для тестування моделей і сценаріїв.

1.2 Аналіз переваг та недоліків існуючих рішень

Аналіз переваг та недоліків існуючих рішень подано в таблиці 1.1

Таблиця 1.1 - Аналіз переваг та недоліків існуючих рішень

Рішення	Переваги	Недоліки
ArduPilot	Відкритий код, підтримка різних платформ	Складність кастомізації, обмежена продуктивність
PX4	Гнучкість, підтримка сучасних алгоритмів	Високий поріг входу, потреба в потужному обладнанні
DJI Systems	Простота використання, стабільність	Закритий код, низька адаптивність
Intel Aero	Підтримка AI, оптимізація для автономності	Висока вартість, обмежений функціонал
ROS	Модульність, широка дослідницька підтримка	Висока складність налаштування
AirSim	Реалістична симуляція, інтеграція з AI	Використовується тільки для моделювання

Аналіз існуючих систем автоматизованого керування БПЛА виявив такі ключові особливості:

1. PX4 — це потужна відкрита платформа для керування безпілотниками, яка широко використовується як у дослідницьких, так і в комерційних проектах. PX4 має модульну архітектуру, що дозволяє гнучко адаптувати систему під різні завдання та апаратні платформи. Основна робота PX4 полягає у виконанні контролю польоту, стабілізації дрона та забезпеченні автоматичних місій. Для інтеграції нових алгоритмів або сенсорів користувачу необхідно працювати із середовищем розробки PX4-Autopilot, що базується на C++. Код можна компілювати та тестувати за допомогою симулятора Gazebo або AirSim перед перенесенням на реальний апарат.

Щоб розширити функціонал PX4, наприклад, для обробки відео чи додавання нових алгоритмів навігації, рекомендується використовувати зовнішній бортовий комп'ютер, як Raspberry Pi або більш потужний NVIDIA Jetson. На таких пристроях можна запускати окремі сервіси, які спілкуються з PX4 через MAVLink протокол. Однак головна проблема роботи з PX4 полягає у складності налаштування ядра системи та інтеграції нових алгоритмів, оскільки будь-яка помилка може негативно вплинути на стабільність польоту. Крім того, платформа вимагає ретельного калібрування сенсорів і контролю за обчислювальними ресурсами, щоб уникнути перевантажень.

ArduPilot, на відміну від PX4, є більш орієнтованим на гнучкість у налаштуванні місій і доступності для розробників. ArduPilot підтримує широкий спектр апаратних контролерів і має глибоко оптимізований код для виконання завдань автопілоту, стабілізації та маршрутизації. Для роботи з ArduPilot можна використовувати стандартні інтерфейси, такі як Mission Planner для планування польотів і налаштування параметрів. У випадку розширення функціоналу, ArduPilot дозволяє впроваджувати власні скрипти на Lua, що полегшує створення нестандартних сценаріїв або нових алгоритмів поведінки дрона без глибоких змін у ядрі системи.

Якщо новий функціонал вимагає великих обчислювальних потужностей, наприклад, для обробки даних з камер чи лідара, ArduPilot також підтримує взаємодію через MAVLink-протокол з зовнішніми модулями. Це дозволяє винести важкі обчислення на окремий бортовий комп’ютер і передавати лише результати у систему керування. Однак основна складність у роботі з ArduPilot полягає в інтеграції нових сенсорів або нестандартного обладнання, оскільки це потребує розробки кастомних драйверів та глибокого тестування на надійність.

Таким чином, PX4 краще підходить для розробників, які готові працювати з низькорівневим кодом і кастомізувати ядро системи для складних завдань. ArduPilot є більш дружнім до користувачів завдяки скриптам і готовим

інструментам для планування місій, але для серйозних модифікацій також вимагає досвіду і ресурсів для інтеграції нового обладнання.

Впровадження кастомного функціоналу та його масштабування на платформах PX4 і ArduPilot є завданням, складність якого залежить від рівня змін, типу функціоналу та обраного підходу до розробки.

PX4 відрізняється більш жорсткою архітектурою, оскільки її кодова база побудована на модульному принципі і працює на RTOS (реальночасовій операційній системі). Це означає, що кожен новий модуль чи алгоритм потребує ретельної інтеграції, тестування та налаштування для забезпечення стабільної роботи у реальному часі. Наприклад, додавання нового алгоритму стабілізації чи нестандартного датчика потребує роботи з ядром системи, написання драйвера на C++ і підключення до системного менеджера сенсорів та контролю польоту. Через це впровадження змін є доволі складним і вимагає високого рівня технічної експертизи.

Масштабування функціоналу на PX4 залежить від його реалізації. Якщо функціонал побудований як зовнішній модуль (наприклад, обробка відео на Raspberry Pi або Jetson), то його можна досить просто масштабувати, додавши нові вузли, які спілкуються з PX4 через MAVLink. Однак інтеграція функціоналу безпосередньо у ядро PX4 для масштабування на різні платформи потребуватиме додаткової оптимізації під різні апаратні контролери та їх обчислювальні ресурси.

ArduPilot є більш гнучким для додавання кастомного функціоналу завдяки підтримці Lua-скриптів та відносно простішій архітектурі. Lua дозволяє додати нову логіку керування дроном без глибоких змін у системному коді, що значно знижує складність впровадження на початковому етапі. Наприклад, можна реалізувати кастомні поведінки, маршрути або реакції на дані сенсорів через прості скрипти. Це особливо зручно для швидкого прототипування нових функцій.

Проте для більш складного функціоналу, як-от інтеграція нового сенсора або зміна основного алгоритму стабілізації, знадобиться модифікація ядра ArduPilot на рівні C++. Це також вимагає досвіду, але структура ArduPilot є більш зрозумілою та добре документованою, що трохи полегшує роботу. Масштабування у випадку ArduPilot можливе через зовнішні обчислювальні модулі, які працюють з ArduPilot через MAVLink, або шляхом використання однієї конфігурації на різних платформах контролерів.

Загалом, PX4 має вищий поріг входження для кастомізації, але надає більшу гнучкість у впровадженні глибоких змін і оптимізації функціоналу. ArduPilot є більш доступним для впровадження менш критичних змін завдяки Lua-скриптам, але масштабування складніших рішень також потребує модифікації ядра та оптимізації під апаратні обмеження. У обох випадках складність зростає, якщо система вимагає високої продуктивності у реальному часі або використання нестандартного обладнання.

2. DJI Flight Control System — це закрита комерційна платформа для керування безпілотниками, що відзначається високою надійністю, стабільністю та простотою використання. Основною особливістю є обмежений доступ до внутрішнього програмного забезпечення та алгоритмів автопілота, що робить платформу менш гнучкою у порівнянні з відкритими рішеннями. Водночас DJI пропонує набір SDK, які дозволяють розробникам розширювати функціонал у рамках доступних інтерфейсів.

DJI Onboard SDK відкриває можливість підключати зовнішній бортовий комп'ютер до системи керування дроном через інтерфейс UART. Це дозволяє реалізовувати складні обчислювальні завдання, такі як обробка зображень, розпізнавання об’єктів, машинне навчання та автономна навігація. Зовнішній комп'ютер може надсилати команди автопілоту DJI, зберігаючи при цьому базовий функціонал керування дроном.

Payload SDK надає інструменти для інтеграції нових сенсорів, камер та іншого корисного навантаження на платформу DJI. Це дозволяє розширити

функціонал дрона для виконання специфічних завдань, таких як картографування місцевості за допомогою LiDAR чи моніторинг температури з інфрачервоною камерою. Payload SDK дозволяє взаємодіяти з бортовими системами, керувати підключеним обладнанням і отримувати дані з сенсорів.

Cloud API забезпечує можливість підключення дронів до хмарних сервісів для централізованого управління та автоматизації місій. Це рішення підходить для великих систем, де необхідно керувати кількома БПЛА одночасно, збирати дані у реальному часі та координувати їх дії на відстані. Використання хмарних технологій спрощує розгортання та масштабування інфраструктури для управління флотом дронів.

Основною проблемою DJI Flight Control System є її закритість, що обмежує можливість зміни низькорівневих алгоритмів польоту та стабілізації. Доступ до ядра системи відсутній, а весь додатковий функціонал реалізується як зовнішні модулі, що взаємодіють з автопілотом через API. Це створює певні затримки у передачі команд і потребує додаткових обчислювальних ресурсів на зовнішньому обладнанні.

Впровадження нестандартного функціоналу, наприклад, використання унікальних сенсорів чи алгоритмів керування, є можливим, але вимагає роботи з SDK для адаптації програмного забезпечення під обмеження платформи DJI. Це також може потребувати значного часу для тестування і валідації, щоб забезпечити стабільність і надійність нових рішень у польоті.

Таким чином, DJI Flight Control System забезпечує високу стабільність та надає зручні інструменти для розробки, але обмежує можливість глибокої кастомізації. Розширення функціоналу реалізується через зовнішні модулі, підключені до автопілота, що робить платформу менш гнучкою, але досить зручною для швидкого впровадження інновацій.

Впровадження штучного зору та інших функцій на основі AI у DJI Flight Control System є можливим, але воно реалізується не безпосередньо у контролері дрона DJI, а через зовнішні обчислювальні модулі, які взаємодіють із системою керування дроном. Закритість DJI автопілота не дозволяє вносити зміни у базові алгоритми керування або працювати з внутрішніми сенсорами безпосередньо. Проте, за допомогою інструментів DJI Onboard SDK та Payload SDK можна інтегрувати AI-функціонал у роботу дрона.

Для реалізації штучного зору, наприклад, обробки зображень для виявлення об'єктів, побудови карти місцевості або автономної навігації, використовується зовнішній бортовий комп'ютер. Приклади таких модулів включають NVIDIA Jetson, Raspberry Pi або подібні пристрої з достатньою обчислювальною потужністю для роботи нейронних мереж і алгоритмів машинного навчання. Бортовий комп'ютер підключається до дрона через інтерфейс UART або USB та взаємодіє з автопілотом DJI через Onboard SDK.

3. Дослідницькі платформи, такі як ROS і Microsoft AirSim, надають інструменти для тестування й моделювання складних сценаріїв. ROS добре підходить для багатокомпонентних систем і широко застосовується у наукових дослідженнях, але його складність і висока вимога до ресурсів є суттєвим бар'єром для широкого впровадження. AirSim, у свою чергу, дозволяє моделювати навколишнє середовище й випробовувати алгоритми без реального апарату, однак не забезпечує прямого впливу на фізичні системи.

Microsoft AirSim – це платформа для симуляції, розроблена Microsoft Research, призначена для навчання, тестування та розробки систем автономного керування дронами, наземними транспортними засобами та іншими роботизованими системами. AirSim працює на основі Unreal Engine від Epic Games, що дозволяє створювати реалістичні та деталізовані середовища для симуляцій. Основна мета платформи – надати розробникам можливість

тестувати алгоритми комп'ютерного зору, машинного навчання, автономної навігації та контролю в умовах, максимально наближених до реального світу.

Основні компоненти та можливості AirSim:

1. Підтримка різних типів транспортних засобів. AirSim дозволяє моделювати не тільки дрони (квадрокоптери), але й наземні транспортні засоби, такі як автомобілі. Користувач може переключатися між різними режимами симуляції та додавати кастомні транспортні засоби.

2. Високоякісна фізична модель. AirSim використовує PhysX, фізичний рушій Unreal Engine, для моделювання фізики польоту та руху. Це дозволяє відтворити реалістичні умови, зокрема аеродинаміку, гравітацію, тертя, інерцію та інші фізичні властивості.

3. Сенсори AirSim підтримує широкий спектр віртуальних сенсорів, які можуть бути використані для отримання даних у симуляції: Камери (RGB, глибини, сегментації, стереокамери); LiDAR – лазерний сенсор для побудови 3D-карт місцевості; IMU – інерціальний вимірювальний блок; GPS – координатна система для навігації; Барометр, магнітометр та інші сенсори.

4. API для інтеграції AirSim надає крос-платформний API для керування дронами та транспортними засобами. API підтримує такі мови програмування, як Python та C++, що дозволяє розробникам писати сценарії для керування польотом, збору даних та реалізації AI-функцій. Через API можна: Керувати рухом дрона (висота, курс, швидкість); Збирати дані з сенсорів; Виконувати автоматизовані місії; Впроваджувати алгоритми автономної навігації.

5. Машинне навчання та штучний інтелект AirSim є ідеальним середовищем для розробки та навчання AI-моделей. Симуляція дозволяє збирати великі обсяги синтетичних даних для тренування алгоритмів комп'ютерного зору, таких як обробка зображень, виявлення об'єктів та ухилення від перешкод. Платформа підтримує роботу з нейронними мережами через TensorFlow, PyTorch та інші фреймворки.

6. Підтримка реалістичних середовищ AirSim дозволяє створювати кастомні карти та середовища завдяки інтеграції з Unreal Engine. Можна моделювати різні типи ландшафтів, погодні умови, час доби та інші фактори, що впливають на роботу дронів. Такі реалістичні умови корисні для тестування алгоритмів в умовах, максимально наближених до реального світу.

7. Інтеграція з хмарними сервісами AirSim може бути інтегрований із хмарними платформами, такими як Microsoft Azure, для обробки великих обсягів даних, отриманих у симуляції. Це дозволяє автоматизувати тренування AI-моделей та масштабувати обчислення.

ROS (Robot Operating System) – це відкритий фреймворк для розробки програмного забезпечення для робототехнічних систем. Хоча назва вказує на "операційну систему", фактично ROS є середовищем для створення, керування та інтеграції модулів, які взаємодіють у реальному часі. Він надає інструменти, бібліотеки та API для полегшення розробки складних робототехнічних додатків, зокрема систем керування для БПЛА.

ROS підтримує архітектуру на основі вузлів (nodes) та топіків (topics), що дозволяє розділяти функціональність на окремі незалежні компоненти. Вузли – це окремі процеси, які відповідають за виконання конкретної задачі (наприклад, обробка сенсорних даних або керування приводами). Комунікація між вузлами здійснюється через топіки, де інформація передається у вигляді повідомлень. Це забезпечує модульність та можливість паралельного виконання різних задач, що є ключовим для робототехнічних систем.

Однією з основних переваг ROS є підтримка широкого спектра сенсорів і пристроїв, а також готові пакети для обробки даних і керування роботами. Наприклад, ROS включає бібліотеки для роботи з камерами, лідаром, GPS, інерціальними датчиками тощо. Також наявні інструменти для реалізації алгоритмів навігації, планування маршруту, SLAM (одночасна локалізація та побудова карти) та машинного навчання. Ці пакети можна використовувати як основу для розробки власного функціоналу.

Для симуляції та тестування систем ROS тісно інтегрується з Gazebo, що дозволяє створювати реалістичні тривимірні середовища та тестувати алгоритми на віртуальних роботах. Завдяки цій інтеграції розробники можуть імітувати реальні умови роботи роботів і перевіряти ефективність алгоритмів перед запуском на фізичному обладнанні.

Однією з особливостей ROS є його модульність та гнучкість. Розробники можуть розширювати функціонал системи шляхом додавання нових вузлів, що виконують специфічні задачі. Це дозволяє інтегрувати нові алгоритми або технології без необхідності переробляти всю систему. Наприклад, для додавання системи штучного зору можна створити окремий вузол, який отримує дані з камери та виконує обробку зображень.

Попри свої переваги, ROS має певні обмеження. По-перше, він не підтримує жорстких реального часу обмежень, оскільки побудований на стандартних Linux-процесах. Це може стати проблемою для систем, які вимагають детермінованої поведінки та мінімальної затримки. Для таких задач часто використовується ROS 2, що має покращену підтримку реального часу завдяки використанню DDS (Data Distribution Service).

Іншою складністю є налаштування та інтеграція. ROS потребує значного досвіду для конфігурування середовища, оскільки воно працює на базі Linux, і більшість інструментів вимагає знання командного рядка. Крім того, велика кількість готових пакетів може бути застарілою або потребувати додаткового налаштування для роботи зі специфічним обладнанням.

Загалом, ROS є потужним інструментом для розробки робототехнічних систем завдяки своїй модульності, гнучкості та великій спільноті розробників. Він активно використовується для створення систем автономного керування БПЛА, оскільки дозволяє легко інтегрувати сенсори, алгоритми навігації та машинного навчання у єдину систему.

ROS і Docker з контейнерами на базі Linux виконують різні ролі в розробці програмних систем, але їх можна ефективно комбінувати для

створення масштабованих і модульних робототехнічних рішень. Порівняння цих технологій можна розглянути через їхні функціональні можливості, цілі використання та архітектурні особливості.

ROS є фреймворком, спеціалізованим для розробки та управління робототехнічними системами. Він надає інструменти для побудови складної архітектури, що складається з вузлів (nodes), які комунікують між собою через топіки (topics), служби (services) та дії (actions). ROS спрощує роботу з апаратними пристроями, сенсорами, алгоритмами навігації, обробкою даних і штучним інтелектом. Архітектура ROS дозволяє запускати вузли на різних пристроях та об'єднувати їх у єдину систему через мережеву інфраструктуру. Проте ROS працює переважно на рівні процесів, де кожен вузол є окремим процесом, що вимагає керування взаємодією та конфігурацією на рівні ОС.

Docker, на відміну від ROS, є платформою для контейнеризації, яка дозволяє ізолювати програмне забезпечення разом із його залежностями всередині контейнерів. Кожен контейнер – це легкий, незалежний екземпляр, який виконується на базі ядра Linux. Docker надає можливість стандартизувати розгортання додатків, забезпечуючи однакове середовище для роботи на різних платформах та пристроях. Контейнери працюють швидше, ніж віртуальні машини, оскільки не потребують окремого ядра операційної системи. Це робить Docker ефективним для модульного розгортання, масштабування та підтримки великих систем, включаючи робототехнічні рішення.

1.3 Висновки до розділу

У першому розділі проведено аналіз предметної області та визначено актуальність розробки системи автоматизованого керування безпілотним літальним апаратом. На основі проведеного дослідження зроблено наступні висновки:

1. Актуальність та значущість проблеми. Автоматизація керування безпілотними літальними апаратами є одним із ключових напрямів сучасних технологій, з огляду на їх широке застосування у різних сферах: від логістики та картографії до рятувальних операцій та моніторингу довкілля. Попит на універсальні системи керування, здатні адаптуватися до різних сценаріїв, є вкрай високим.

2. Огляд предметної області. Аналіз показав, що існуючі системи здебільшого зосереджені на вузькоспеціалізованих завданнях, таких як автономна навігація або обробка візуальних даних. Водночас інтегровані рішення, які б забезпечували комплексний підхід до керування польотом, обробки даних сенсорів та навігації, мають обмежене поширення.

3. Можливості симуляції Використання симуляційних середовищ, таких як AirSim, відкриває нові можливості для тестування і вдосконалення систем автопілота в умовах, наближених до реальних. Це дозволяє проводити експерименти без ризику пошкодження апаратури, що є особливо важливим на початкових етапах розробки.

4. Визначення напрямків роботи У розділі окреслено ключові вимоги до системи, серед яких модульність, використання сучасних мов програмування, таких як Kotlin, інтеграція із симулятором AirSim та застосування контейнерної архітектури для забезпечення масштабованості та гнучкості системи.

Аналіз предметної області підтвердив доцільність розробки системи автоматизованого керування безпілотним літальним апаратом із застосуванням сучасних інструментів і підходів. Окреслено основні проблеми, які необхідно вирішити, а також обґрунтовано вибір технологій для розробки системи. Результати даного розділу стали основою для визначення подальших етапів розробки, описаних у наступних розділах роботи.

Поєднання ROS і Docker надає значні переваги для розробки та розгортання робототехнічних систем. Кожен вузол ROS або групу вузлів можна ізолювати в окремий Docker-контейнер, що усуває конфлікти залежностей та

спрощує процеси інтеграції. Завдяки Docker, розробники можуть створювати стандартизовані середовища, які легко масштабуються за допомогою Kubernetes або інших інструментів оркестрації.

Загалом, Docker забезпечує інфраструктурну модульність, тоді як ROS надає функціональну модульність для робототехнічних додатків. Їхнє поєднання дозволяє створювати більш гнучкі, масштабовані та надійні системи для БПЛА та інших автономних роботів.

2 ВИБІР ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ БОРТОВОГО КОМП’ЮТЕРА

2.1 Вибір комп'ютера для керування БПЛА

Вибір бортового комп'ютера для безпілотного літального апарата (БПЛА) є критичним етапом розробки системи, оскільки від його характеристик залежить продуктивність, надійність і енергоефективність всієї системи. Основними вимогами до бортового комп’ютера є:

- компактність і мала маса, що забезпечує мінімальний вплив на льотні характеристики БПЛА;
- висока енергоефективність для зменшення споживання енергії з бортової батареї;
- висока продуктивність для обробки даних сенсорів, навігації та виконання задач реального часу;
- підтримка сучасних стандартів програмного забезпечення та інтерфейсів.

Зважаючи на ці вимоги, як бортовий комп’ютер для розробки було обрано Mac Mini на базі процесорів Apple Silicon M2 рисунок 2.1 та M4 рисунок 2.2, що забезпечують оптимальне співвідношення продуктивності, енергоефективності та вартості.

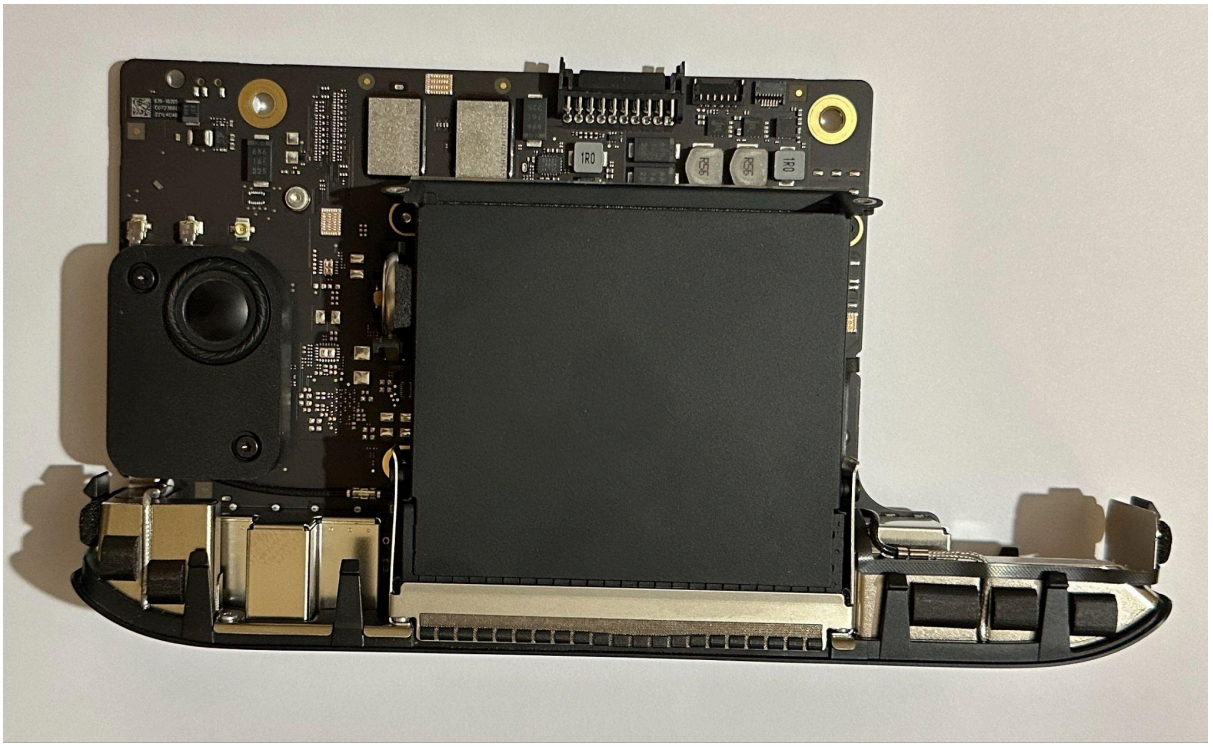

Рисунок 2.1 - Материнська плата Mac Mini M2

Рисунок 2.2 - Mac Mini M4 в розібраному стані

Особливості та переваги процесорів Apple Silicon. Процесори Apple Silicon побудовані на ARM-архітектурі, що поєднує високу енергоефективність

та обчислювальну потужність. Розглянемо ключові технічні характеристики моделей M2 та M4, а також їхні переваги в контексті використання на БПЛА таблиця 2.1.

Таблиця 2.1 - технічні характеристики моделей M2 та M4

Характеристика	M2	M4
Техпроцес	5 нм	3 нм
Кількість ядер CPU	8 (4 P + 4 E)	12 (6 P + 6 E)
Частота CPU	до 3.49 ГГц	до 3.7 ГГц
Кількість ядер GPU	10	18
Neural Engine	16 ядер, до 15.8 Тфлопс	20 ядер, до 24.5 Тфлопс
Оперативна пам'ять	до 24 ГБ	до 32 ГБ
Пропускна здатн пам’яті	100 ГБ/с	150 ГБ/с
Споживана потужність	до 25 Вт	до 30 Вт

Примітка: P — продуктивні ядра (Performance), E — енергоефективні ядра (Efficiency).

1. Енергоефективність:

- M2: забезпечує до 3 разів кращу продуктивність на ват порівняно з традиційними x86-процесорами [9]. При типовому навантаженні споживає 20–25 Вт, що дозволяє використовувати менші батареї або збільшити час автономної роботи;
- M4: завдяки техпроцесу 3 нм досягає ще більшої енергоефективності, зберігаючи низький рівень тепловиділення.

2. Продуктивність. Обидві моделі забезпечують високу обчислювальну потужність [10] завдяки збалансованому використанню продуктивних та енергоефективних ядер:

- Висока продуктивність CPU дозволяє виконувати ресурсоємні обчислення, включаючи аналіз даних сенсорів та навігаційні алгоритми;
- Потужний GPU підтримує задачі комп'ютерного зору [11], наприклад, розпізнавання об'єктів у реальному часі або побудову тривимірної карти місцевості;
- Neural Engine обробляє задачі машинного навчання зі швидкістю до 15.8 Тфлопс (M2) і до 24.5 Тфлопс (M4), що дозволяє реалізовувати інтелектуальні функції, такі як адаптивна навігація або розпізнавання перешкод [12].

3. Компактність і легкість Розміри Mac Mini M2 (197 × 197 × 36 мм), M4 (127 × 127 × 50 мм) та маса M2 (~1.2 кг в корпусі, ~0.3 кг без корпуса і блоку живлення) M4 (~0.67 кг в корпусі, ~0.15 кг без корпуса і блоку живлення) дозволяють інтегрувати пристрій у корпус середнього або великого дрона без значного впливу на льотні характеристики.

4. Сучасні інтерфейси Mac Mini підтримує:

- Thunderbolt 4/USB4 для швидкого підключення сенсорів або зовнішніх накопичувачів;
- Ethernet і Wi-Fi 6E для забезпечення зв’язку з наземною станцією;
- Bluetooth 5.3 для інтеграції периферійних пристроїв.

5. Стабільність та оптимізація. Apple Silicon забезпечує високу стабільність завдяки інтеграції апаратного та програмного забезпечення, що мінімізує ризик системних помилок.

6. Підтримка програмування. Моделі підтримують всі основні інструменти для розробки, зокрема Docker, Kotlin, Python, а також мають бібліотеки для комп’ютерного зору (OpenCV), машинного навчання (TensorFlow, PyTorch) та обробки сенсорних даних.

Використання Mac Mini з процесорами M2 та M4 як бортового комп’ютера забезпечує оптимальне поєднання продуктивності, енергоефективності та компактності, що робить його ідеальним рішенням для сучасних автономних БПЛА. Завдяки високій обчислювальній потужності ці процесори здатні обробляти дані з багатьох сенсорів одночасно, включаючи камери, лідари, інерційні вимірювальні пристрої (IMU) та GPS. Це дозволяє не лише забезпечувати стабільність польоту, а й виконувати такі завдання, як побудова маршруту в реальному часі, уникнення перешкод та адаптація до змінних умов середовища.

Інтеграція Neural Engine відкриває широкі можливості для використання штучного інтелекту, включаючи застосування алгоритмів машинного навчання для розпізнавання об’єктів, аналізу місцевості, ідентифікації цілей або прийняття рішень у складних сценаріях. Потужний GPU, разом із підтримкою сучасних бібліотек, дозволяє реалізовувати алгоритми комп’ютерного зору, такі як SLAM (одночасна локалізація та побудова карти), аналіз потокового відео та виявлення небезпечних зон.

Підтримка сучасних інтерфейсів, таких як Thunderbolt 4 і USB4, дозволяє без зусиль інтегрувати Mac Mini з різноманітним обладнанням: додатковими сенсорами, контролерами або модулями зв’язку. Високошвидкісний Ethernet і Wi-Fi 6E забезпечують стабільний зв’язок із наземними станціями, дозволяючи оперативно передавати телеметрію, відеопотоки або дані для віддаленого керування. Крім того, підтримка Bluetooth 5.3 спрощує підключення периферійних пристроїв, таких як пульти управління або додаткові модулі збору даних.

Завдяки підтримці платформ Docker, Kotlin, Python і популярних бібліотек, розробка і впровадження програмного забезпечення для Mac Mini стає більш гнучким і масштабованим процесом. Можливість контейнеризації дозволяє ізолювати різні модулі системи, спрощуючи інтеграцію, тестування та

обслуговування. Наприклад, окремий контейнер може бути виділений для обробки відеоданих, інший — для навігаційних алгоритмів, а третій — для задач оптимізації маршрутів.

Компактність і мала маса Mac Mini, особливо в моделях без корпусу, дозволяють інтегрувати пристрій навіть у дрони з обмеженим простором і вимогами до ваги. Низьке тепловиділення та висока енергоефективність зменшують вимоги до систем охолодження та дозволяють оптимізувати використання енергоресурсів, продовжуючи час польоту БПЛА. Усе це робить Mac Mini універсальною платформою, здатною адаптуватися до широкого спектру завдань — від базових автономних польотів до складних місій із використанням штучного інтелекту та обробки великих обсягів даних.

2.2 Використання Kotlin для реалізації алгоритмів керування

У системах керування безпілотними літальними апаратами (БПЛА) мова програмування Kotlin була обрана як основний інструмент для реалізації алгоритмів. Це рішення базується на її сучасних можливостях, високій продуктивності та зручності у використанні.

1. Продуктивність і сучасний синтаксис. Kotlin, як мова програмування, що компілюється у JVM (Java Virtual Machine), забезпечує високу продуктивність та ефективність виконання коду в реальному часі [13]. Сучасний синтаксис Kotlin сприяє написанню чистого, зрозумілого та безпечного коду, що особливо важливо для складних систем керування. Завдяки оптимізації виконання, алгоритми обробки даних сенсорів і навігаційних команд працюють із мінімальними затримками, що критично важливо для реального часу.

2. Кросплатформність. Kotlin є кросплатформною мовою, що забезпечує легке перенесення алгоритмів між різними операційними системами: Алгоритми, розроблені для бортового комп’ютера на базі macOS, можуть бути легко протестовані на Windows або Linux, що спрощує розробку й тестування.

Використання Kotlin Multiplatform дозволяє адаптувати окремі компоненти системи для інших платформ без переписування коду [14].

3. Простота інтеграції. Kotlin повністю сумісний із Java, що відкриває доступ до великої кількості бібліотек і фреймворків для роботи з сенсорами, API та іншими модулями: Бібліотеки Java для обробки даних (наприклад, Apache Commons Math) або роботи з протоколами зв'язку (MQTT, HTTP, gRPC) легко інтегруються у Kotlin-проекти. Зручна робота з асинхронним виконанням задач через корутини Kotlin дозволяє створювати складні алгоритми керування з низькими затримками.

4. Швидкість розробки Kotlin відзначається зручним і лаконічним синтаксисом, що прискорює процес написання коду: Скорочення коду на 30-40% у порівнянні з Java дозволяє зменшити час розробки без втрати функціональності. Інструменти для тестування, наприклад, JUnit чи KotlinTest, дозволяють швидко перевіряти коректність алгоритмів на різних етапах розробки.

5. Можливості для розробки модульної архітектури Kotlin ідеально підходить для створення модульних систем: Код алгоритмів легко розбивається на окремі класи або функції, які можуть бути перевикористані в різних компонентах системи. У поєднанні з контейнерною архітектурою Docker це дозволяє ізолювати алгоритми керування, обробки даних і виконання команд у різних модулях.

Використання Kotlin у системі керування БПЛА надає можливість створення надійних, масштабованих та ефективних алгоритмів. Кросплатформність, продуктивність, легкість інтеграції з бібліотеками та швидкість розробки роблять Kotlin ідеальним вибором для основної логіки системи.

2.3 Застосування контейнерної архітектури на основі Docker

Контейнерна архітектура є сучасним підходом до розробки програмного забезпечення, що забезпечує ізоляцію компонентів, гнучкість у розробці та розгортанні системи. У проекті керування безпілотним літальним апаратом (БПЛА) застосування контейнерів на основі Docker та Docker Compose дозволяє створити модульну, легко масштабовану та переносиму систему [15].

1. Модульність. Docker дозволяє розділити функціональність системи на окремі модулі, кожен із яких виконується в окремому контейнері. Наприклад:

- Контейнер для виконання команд польоту;
- Контейнер для обробки даних сенсорів (GPS, камери, IMU);
- Контейнер для планування маршрутів;
- Контейнер для взаємодії з наземною станцією.

Цей підхід дозволяє незалежно розробляти, тестувати та оновлювати кожен модуль без впливу на інші компоненти системи.

2. Взаємодія через API Контейнери спілкуються між собою через стандартизовані інтерфейси (REST API або gRPC) [16], що забезпечує:

- Ізоляцію функціональності: кожен модуль має чітко визначену роль і взаємодіє з іншими через API;

- Гнучкість: система може бути легко розширена шляхом додавання нових модулів або заміни існуючих;

- Простоту тестування: кожен модуль може бути протестований окремо, використовуючи його API.

3. Переносимість. Контейнери забезпечують однакове середовище виконання незалежно від апаратного чи програмного забезпечення. Це дозволяє:

- Тестувати систему в симуляторі на базі Windows (Unreal Engine 5 AirSim) і розгортати її на бортовому комп'ютері на macOS;

- Забезпечити сумісність із будь-якою платформою, яка підтримує Docker (наприклад, Linux-сервери).

4. Масштабованість. Контейнерна архітектура спрощує масштабування системи:

- Додавання нових контейнерів для додаткових функцій, таких як обробка даних з нових сенсорів або інтеграція штучного інтелекту, не потребує суттєвих змін у наявних модулях;

- Контейнери можуть бути розгорнуті у кластері, що дозволяє розподіляти обчислювальні ресурси між кількома фізичними машинами.

5. Зменшення залежностей. Кожен контейнер має власне середовище виконання із залежностями (бібліотеками, інструментами), що зменшує конфлікти між модулями:

- Наприклад, обробка зображень може використовувати Python із TensorFlow, а керування польотом — Kotlin із Ktor, без конфліктів між середовищами.

- Контейнери можуть використовувати різні версії одних і тих самих бібліотек без взаємного впливу.

6. Швидке розгортання Docker дозволяє створити контейнерні образи, які можна швидко розгортати на будь-якому пристрої:

- Оновлення системи зводиться до заміни контейнера новішою версією, що значно скорочує час розгортання.

- Стандартизовані образи забезпечують передбачувану роботу системи у будь-якому середовищі.

7. Використання Docker Compose для управління варіаціями модулів. Для зручного управління контейнерами використовується Docker Compose. Цей інструмент дозволяє створювати набори (паки) конфігурацій, які відповідають різним сценаріям використання БПЛА. Приклади наборів модулів: Базовий пакет (виконує мінімальні функції керування польотом і навігації) використовується для тестування основних алгоритмів; Розширений пакет (включає обробку візуальних даних і аналіз сенсорів) призначений для виконання складних завдань, таких як обхід перешкод; Пакет симуляції (для

тестування в симуляторі AirSim) інтеграція із симулятором через спеціальний API.

Переваги Docker Compose:

- Гнучкість налаштувань: кожен пакет може бути адаптований під конкретні завдання.

- Простота використання: запуск всієї системи виконується однією командою;

- Зручність у розробці: різні конфігурації можуть бути швидко протестовані.

Docker у цій системі працює як базова платформа для ізоляції та взаємодії модулів. Кожен контейнер має власний конфігураційний файл Dockerfile, який визначає операційну систему, залежності, інструменти та команду запуску. Налаштування мережі забезпечується через Docker Network, що дозволяє контейнерам спілкуватися між собою на віртуальних каналах. Це дозволяє чітко розмежувати трафік між модулями та захистити систему від зовнішніх втручань. За потреби можна налаштувати порти для зовнішнього доступу, наприклад, для інтеграції з наземною станцією або віддаленого моніторингу.

Спільні каталоги між контейнерами забезпечуються через Docker Volumes або Bind Mounts. Це дозволяє модулю обробки сенсорних даних зберігати результати, які можуть бути доступні для інших контейнерів, таких як модуль аналізу або логування. Наприклад, дані з GPS можуть бути збережені у спільному каталозі, де їх обробить модуль планування маршруту.

Використання Docker Compose дозволяє централізовано керувати цими налаштуваннями. У конфігураційному файлі docker-compose.yml можна визначити, які порти мають бути відкриті, які каталоги спільні між контейнерами, і які залежності має система. Наприклад, можна налаштувати, щоб модуль обробки камер використовував GPU для обчислень через параметри доступу до апаратного забезпечення.

Унікальні особливості Docker включають мультиплатформеність, яка забезпечує підтримку всіх основних операційних систем, включаючи Windows, macOS і Linux. Контейнери мають швидкість запуску, що дозволяє почати роботу за лічені секунди завдяки використанню готових образів. Інтеграція з DevOps-процесами дає змогу автоматизувати розробку та доставку програмного забезпечення за допомогою інструментів, таких як Jenkins, Kubernetes або GitLab CI/CD. Ресурсна ефективність досягається завдяки використанню ядра хостової ОС, що зменшує навантаження порівняно з віртуальними машинами. Можливість створення локальних реєстрів образів спрощує зберігання приватних контейнерів, а ізольованість середовищ дозволяє уникнути конфліктів між різними проєктами чи бібліотеками. Крім того, контейнери можуть працювати в офлайн-режимі, що є важливим для автономних систем, таких як БПЛА.

Контейнеризація має перспективи впровадження у масове виробництво БПЛА. Використання Docker дозволяє стандартизувати налаштування бортових комп'ютерів, забезпечуючи єдине середовище для всіх пристроїв. Образ контейнера містить усі необхідні компоненти для запуску програмного забезпечення на бортовому комп'ютері, що гарантує однакову роботу на кожному апараті незалежно від партії виробництва. Завдяки цьому підходу виробники можуть скоротити витрати часу та ресурсів на конфігурацію систем.

Автоматизація налаштувань через скрипти чи CI/CD-пайплайни дозволяє швидко підготувати бортовий комп'ютер до роботи, використовуючи такі інструменти, як Jenkins, GitLab CI/CD чи GitHub Actions. Ці платформи забезпечують централізоване управління процесами інтеграції та розгортання, включаючи тестування, створення образів і їхнє автоматичне завантаження на пристрої. Наприклад, Jenkins дозволяє створювати складні пайплайни для автоматизації, а GitLab CI/CD інтегрується безпосередньо з репозиторіями коду, спрощуючи процес оновлення для розробників. Масштабоване тестування можливе за допомогою симуляторів, таких як AirSim, де контейнери можуть

бути перевірені ще до встановлення на реальний пристрій. Швидке оновлення прошивки здійснюється шляхом заміни лише відповідного контейнера без впливу на інші частини системи, що значно підвищує ефективність виробничих процесів.

Для ефективного налаштування бортового комп'ютера у великих масштабах використовуються попередньо налаштовані образи контейнерів. Завантаження стандартизованих образів на кожен пристрій через централізовану систему оновлення забезпечує однаковість роботи. Контейнеризація кожної специфічної функції, такої як керування польотом, обробка сенсорів або навігація, дозволяє вибірково додавати чи змінювати модулі залежно від конфігурації конкретного БПЛА.

Мережеве розгортання є критично важливим у масштабних виробництвах, оскільки воно дозволяє автоматизувати процеси розгортання програмного забезпечення одночасно на велику кількість пристроїв. Для досягнення цього використовуються централізовані системи розгортання, такі як Ansible, Kubernetes або інструменти CI/CD, що забезпечують доставку оновлень на всі пристрої одночасно. Наприклад, для оновлення прошивки або контейнерного образу на 100 бортових комп'ютерах БПЛА, система автоматизації створює завдання з використанням Ansible Playbook або Kubernetes Deployment. Після цього контейнерні образи завантажуються у централізований реєстр, а далі – на кожен бортовий комп'ютер через мережевий доступ. Цей процес може контролюватися з хмарної платформи, забезпечуючи моніторинг прогресу та виявлення помилок у реальному часі. Таке рішення дозволяє знизити час на оновлення системи та мінімізувати ризик відмови, оскільки кожен пристрій отримує перевірений та однаковий образ програмного забезпечення. Крім того, розподілені завдання дозволяють одночасно масштабувати систему для великих партій БПЛА, гарантуючи ефективність і швидкість розгортання. Інтеграція з наземними станціями через контейнери дозволяє не лише розгортати оновлення, але й здійснювати моніторинг стану

систем у реальному часі. Це забезпечує надійний контроль за роботою системи на всіх етапах життєвого циклу БПЛА.

Перспективи розвитку системи контейнерної архітектури для БПЛА включають декілька ключових напрямків. По-перше, це інтеграція з хмарними сервісами для забезпечення ще більшої гнучкості та віддаленого управління. Наприклад, контейнери можуть бути розгорнуті на базі хмарних платформ, таких як AWS, Azure або Google Cloud, для синхронізації даних і оптимізації роботи в режимі реального часу.

По-друге, розвиток штучного інтелекту дозволить використовувати контейнери для виконання більш складних завдань, таких як автономна навігація, аналіз даних із сенсорів у польоті або координація групових польотів декількох БПЛА. Контейнеризація у цьому контексті забезпечить можливість швидкого оновлення алгоритмів і тестування нових моделей без необхідності втручання у базовий код.

По-третє, збільшення підтримки апаратного прискорення, зокрема для роботи з нейронними мережами або обробки великих обсягів даних, розширить можливості контейнерів. Використання таких технологій, як NVIDIA GPU Cloud (NGC), дозволить оптимізувати продуктивність системи для найвимогливіших обчислювальних завдань.

Зрештою, розвиток стандартів у галузі БПЛА та контейнеризації сприятиме підвищенню сумісності між різними платформами та виробниками. Крім того, додаткові дослідження в області безпеки контейнерів дозволять захистити автономні системи від потенційних загроз та забезпечити їх стабільну роботу у критично важливих умовах. У перспективі система зможе інтегрувати більш динамічні підходи до оркестрації та автоматизації, використовуючи передові інструменти для моніторингу, аналізу та самовідновлення системи в умовах реальних польотів.

2.4 Випробування у середовищі Unreal Engine 5 AirSim

Для тестування алгоритмів керування безпілотним літальним апаратом (БПЛА) було обрано симуляційне середовище Unreal Engine 5 AirSim [17]. Це потужний інструмент для моделювання реальних умов польоту, яке забезпечує високу точність симуляції та інтеграцію з програмними компонентами.

Основні можливості Unreal Engine 5 AirSim:

1. Реалістична фізика та моделювання середовища AirSim дозволяє:

- Точно моделювати поведінку дрона, враховуючи аеродинаміку, вібрації, вплив вітру, гравітації та інших фізичних параметрів;

- Створювати реалістичне середовище з урахуванням ландшафту, перешкод, освітлення та погодних умов (дощ, туман, вітер);

- Випробовувати алгоритми в умовах, максимально наближених до реальних.

2. Моделювання сенсорів Симулятор підтримує широкий спектр сенсорів, які можуть бути використані для тестування:

- Камери (RGB, глибина, сегментація);

- Лідари;

- IMU (інерційний вимірювальний блок);

- GPS.

3. Сценарії тестування Unreal Engine 5 AirSim дозволяє створювати складні сценарії, такі як:

- Автономне слідування маршруту з обходом перешкод;

- Робота в умовах обмеженої видимості або завадів сигналу GPS;

- Аварійні ситуації, такі як відмова одного з двигунів.

4. Інтеграція з програмними компонентами Середовище підтримує прямий обмін даними через API (Python, C++, REST API), що дозволяє інтегрувати алгоритми, написані на Kotlin, для:

- Відправки команд польоту;

- Отримання даних із сенсорів у реальному часі. Аналізу та адаптації поведінки дрона під час тестування.

Переваги використання AirSim для випробувань:

1. Безпека та економічність. Тестування алгоритмів проводиться у віртуальному середовищі, що виключає ризик пошкодження реального обладнання. Зменшується вартість тестування, оскільки відсутня потреба в польотних випробуваннях на початкових етапах.

2. Гнучкість налаштувань. Можливість швидкої зміни середовища, погодних умов та сценаріїв без фізичних затрат. Налаштування параметрів фізики та поведінки сенсорів для перевірки граничних умов роботи системи.

3. Розширення можливостей алгоритмів Використання симуляції для навчання системи прийняття рішень в умовах складних або рідкісних сценаріїв. Інтеграція алгоритмів машинного навчання, наприклад, для аналізу зображень або прогнозування траєкторій.

Інтеграція з системою керування БПЛА:

1. Керування польотом Модуль, написаний на Kotlin, взаємодіє з API AirSim для виконання команд:

- Зліт, посадка, зависання;

- Рух за заданими координатами;

- Обхід перешкод.

2. Обробка даних сенсорів AirSim генерує дані з сенсорів, які обробляються у відповідних контейнерах:

- Аналіз зображень із камер для розпізнавання об'єктів або визначення перешкод;

- Використання GPS для перевірки алгоритмів навігації.

Симуляційне середовище Unreal Engine 5 AirSim є потужним інструментом для розробки та перевірки алгоритмів керування БПЛА. Воно забезпечує безпечні умови для тестування, дозволяє швидко адаптувати сценарії до нових вимог і дає змогу створювати реалістичні умови для випробувань.

Завдяки інтеграції з системою, побудованою на Docker, процес тестування стає простішим і більш ефективним.

2.5 Висновки до розділу

У ході аналізу предметної області та огляду існуючих аналогів і систем було зроблено наступні висновки:

1. Переваги симуляційного підходу. Використання середовища AirSim для моделювання автоматизованого керування безпілотним літальним апаратом є обґрунтованим вибором. AirSim надає широкий спектр функцій для імітації роботи сенсорів, створення сценаріїв польотів і обробки даних в умовах, максимально наближених до реальних.

2. Обмеження існуючих систем. Аналіз показав, що більшість існуючих платформ орієнтовані на вузькоспеціалізовані завдання, такі як обробка візуальних даних або реалізація автономних маршрутів. Водночас, їх інтеграція у вигляді модульної системи, здатної забезпечувати гнучке керування, залишається обмеженою. Це створює нішу для розробки більш універсального рішення.

3. Вибір технологій. Серед існуючих мов програмування Kotlin обрано через її сучасні можливості, включаючи підтримку корутин, асинхронного програмування та інтеграції з популярними бібліотеками для роботи з API і серіалізацією даних. Крім того, використання Docker для створення модульної системи забезпечує зручність розгортання і масштабованість.

4. Модульний підхід до системи. У результаті аналізу обрано архітектуру, побудовану на принципах модульності та використанні контейнерів. Кожен модуль відповідає за виконання окремих завдань (аналіз сенсорних даних, навігація, керування польотом), що спрощує розробку, тестування та обслуговування системи.

5. Вибір апаратного та програмного забезпечення Використання MacMini на Apple Silicon як бортового комп’ютера дозволяє ефективно реалізувати обчислювальні задачі, водночас середовище симуляції працює на високопродуктивному ПК з Ryzen 5 5600 та Nvidia RTX 4070, що забезпечує плавну і точну симуляцію.

Аналіз предметної області та огляд існуючих рішень підтвердили актуальність та обгрунтованість вибору запропонованих технологій і підходу до побудови системи автоматизованого керування безпілотним літальним апаратом. Виявлені обмеження існуючих систем стали основою для визначення вимог до власної архітектури, спрямованої на забезпечення гнучкості, модульності та ефективності роботи в умовах симуляції.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗВ'ЯЗАННЯ ПОСТАВЛЕНИХ ЗАДАЧ

3.1 Побудова архітектури системи

Архітектура системи автоматизованого керування безпілотного літального апарату ґрунтується на модульному підході, що забезпечує розподіл функцій між незалежними компонентами. Кожен модуль реалізований як окремий Docker-контейнер, який взаємодіє з іншими за допомогою REST API та спільних каталогів для обміну даними. Такий підхід дозволяє досягти гнучкості у масштабуванні, спрощення розробки й підтримки системи. Для демонстрації підходу в даній роботі наводиться приклад створення модуля системи на базі фреймворку Ktor у мові програмування Kotlin. Цей модуль, який слугуватиме основою для реалізації інших функціональних компонентів, ілюструє побудову та налаштування архітектури проекту.

Підготовка середовища розробки. Для створення модулів системи необхідно налаштувати середовище розробки, що включає:

- Інтегроване середовище розробки: IntelliJ IDEA (версія Community або Ultimate);

- Мова програмування: Kotlin (версія 1.9 або вище);

- Інструменти контейнеризації: Docker для локального тестування та розгортання;

- Зовнішні залежності: бібліотеки для роботи з сервером Ktor, обробки JSON, і асинхронної взаємодії через корутини.

Створення Ktor-проекту:

1. Запустіть IntelliJ IDEA та створіть новий проект:

- У меню виберіть File → New → Project;

- Оберіть шаблон Ktor рисунок 3.1.

2. Вкажіть основні параметри проекту:

- Name: FlightControlModule;

- Group: com.example;

- Build System: Gradle (Kotlin DSL);

- Ktor version: останню стабільну версію.

Додайте необхідні плагіни до проекту:

- ktor-server-core: для базових можливостей сервера;

- ktor-server-netty: реалізація сервера на основі Netty;

- ktor-serialization-kotlinx-json: для роботи з JSON рисунок 3.2;

- ktor-server-content-negotiation: для автоматичного узгодження форматів даних.

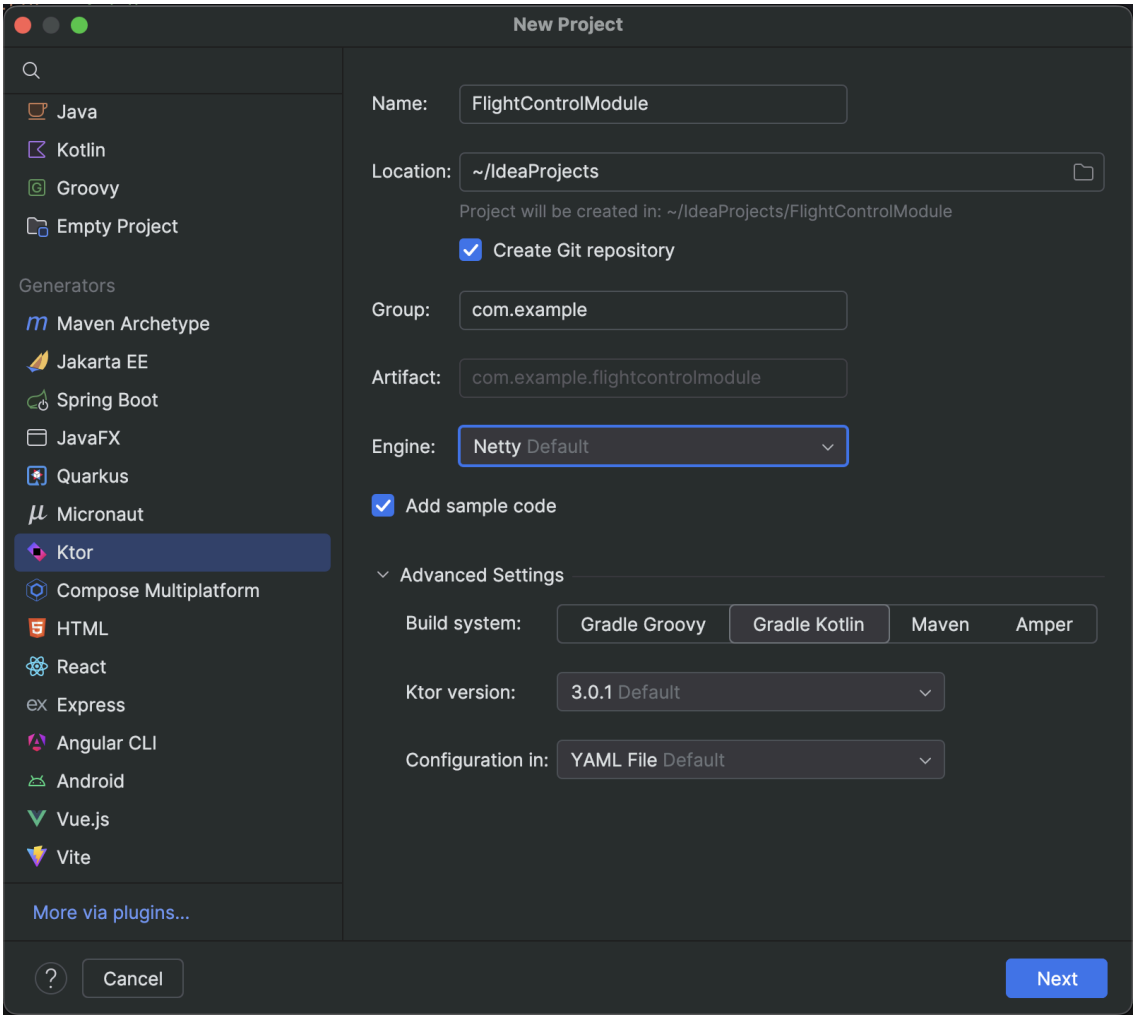

Рисунок 3.1 - Налаштування Ktor проекту

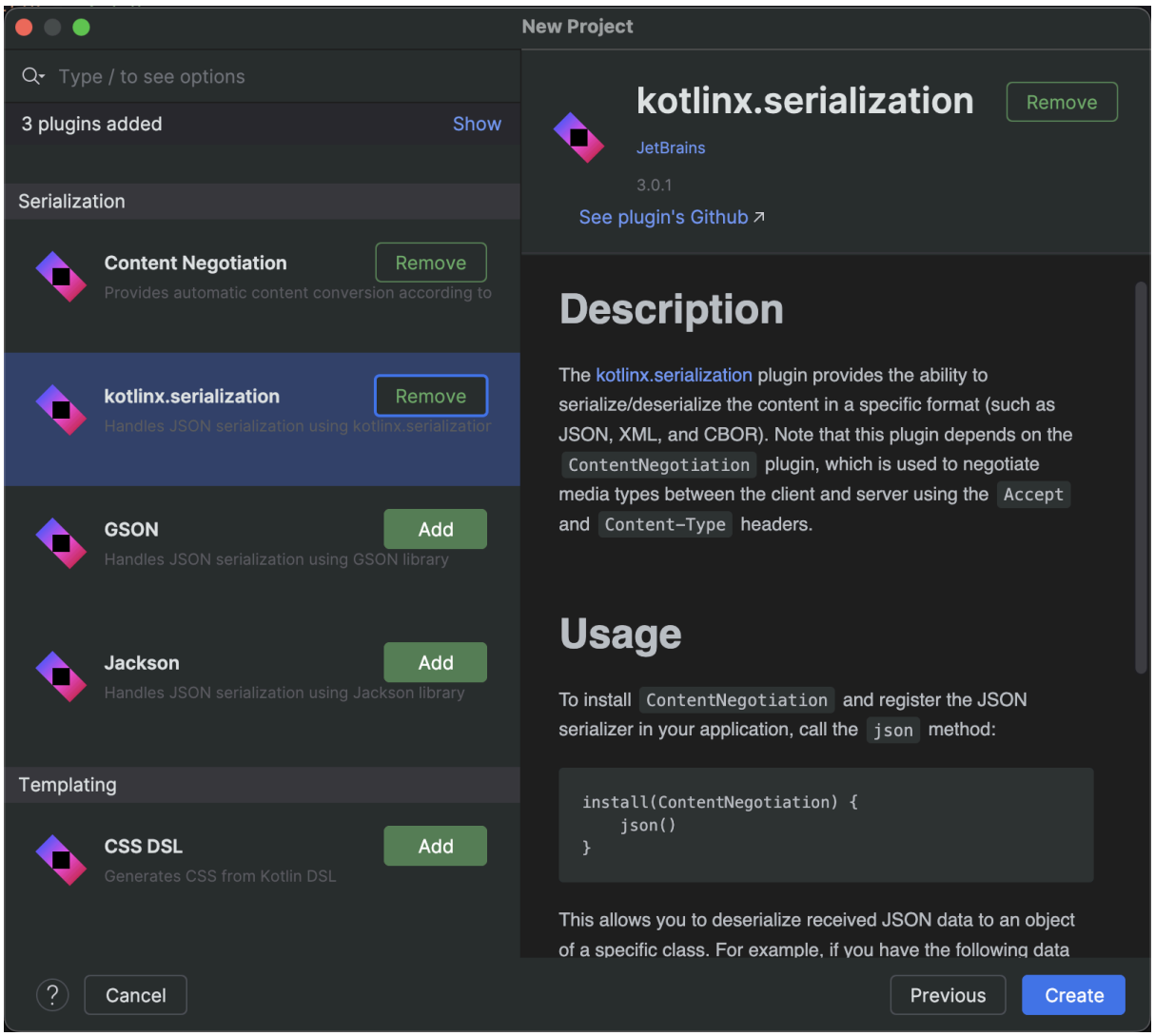

Рисунок 3.2 - Налаштування kotlinx.serialization

Створений проект матиме таку структуру як на рисунку 3.3

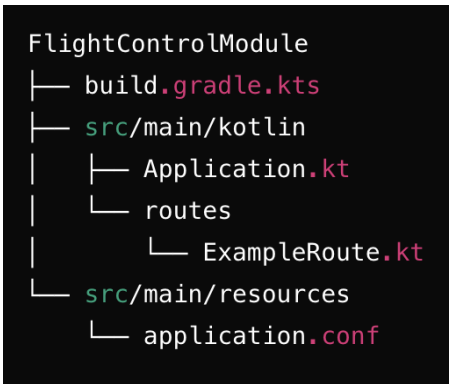

Рисунок 3.3 - Структура проекту

У файлі “build.gradle.kts” налаштовуються залежності як на рисунку 3.4

```
dependencies {
    implementation("io.ktor:ktor-server-core:2.x.x")
    implementation("io.ktor:ktor-server-netty:2.x.x")
    implementation("io.ktor:ktor-serialization-kotlinx-json:2.x.x")
    implementation("io.ktor:ktor-server-content-negotiation:2.x.x")
    testImplementation("io.ktor:ktor-server-tests:2.x.x")
    testImplementation("org.jetbrains.kotlin:kotlin-test-junit:1.9.x")
}
```

Рисунок 3.4 - Налаштування залежностей

У файлі “Application.kt” реалізується базова структура модуля як на рисунку 3.5

```
package com.example.flightcontrol

import io.ktor.server.application.*
import io.ktor.server.engine.*
import io.ktor.server.netty.*
import io.ktor.server.response.*
import io.ktor.server.routing.*

fun main() {
    embeddedServer(Netty, port = 8080, module = Application::module).start(wait =
}

fun Application.module() {
    routing {
        get("/") {
            call.respondText("Flight Control Module is running!")
        }
    }
}
```

Рисунок 3.5 - Файл “Application.kt”

Файл “application.conf” містить параметри налаштування як на рисунку 3.6

```
ktor {
    deployment {
        port = 8080
        watch = [ classes ]
    }
    application {
        modules = [ com.example.flightcontrol.ApplicationKt.module ]
    }
}
```

Рисунок 3.6 - Файл “application.conf”

Тестування модуля. Запуск проекту здійснюється через середовище IntelliJ IDEA. Після запуску сервер буде доступний за адресою “http://localhost:8080”, де можна побачити базове повідомлення про успішну роботу модуля.

Даний модуль є основою для розробки інших функціональних компонентів системи. У майбутньому він може бути розширений для виконання завдань, таких як обробка команд управління, взаємодія з симулятором AirSim та іншими модулями.

3.2 Реалізація створення Docker-образу модуля та його запуск у Docker

Одним із ключових аспектів архітектури системи є контейнеризація модулів. Використання Docker дозволяє створювати ізольовані середовища, що гарантує сумісність та стабільність під час роботи на різних платформах. У цьому розділі детально описується процес створення Docker-образу для модуля та його запуску [18].

1. Підготовка до контейнеризації. Перед контейнеризацією необхідно переконатися, що проєкт зібраний у вигляді виконуваного JAR-файлу. У випадку використання Gradle, це досягається за допомогою плагіна application або налаштувань для створення "fat JAR". Необхідно додати в “build.gradle.kts” наступне налаштування для створення "fat JAR" як на рисунку 3.7.

```
plugins {
    application
}

application {
    mainClass.set("com.example.flightcontrol.ApplicationKt")
}

tasks.named<Jar>("jar") {
    duplicatesStrategy = DuplicatesStrategy.EXCLUDE
    manifest {
        attributes["Main-Class"] = application.mainClass.get()
    }
    from(configurations.runtimeClasspath.get().map { if (it.isDirectory) it else
}
```

Рисунок 3.7 - Налаштування для створення "fat JAR"

Виконати збірку можна командою - “./gradlew jar”. Після успішного виконання команда створить файл “flightcontrol.jar” у каталозі “build/libs/”.

2. Створення Dockerfile Для контейнеризації модуля створюється файл Dockerfile у кореневій директорії проекту. Вміст Dockerfile зображено на рисунку 3.8.

```
# Використання офіційного образу OpenJDK як базового
FROM openjdk:17-jdk-slim

# Встановлення робочої директорії в контейнері
WORKDIR /app

# Копіювання створеного JAR-файлу в контейнер
COPY build/libs/flightcontrol.jar app.jar

# Встановлення порту для контейнера
EXPOSE 8080

# Команда для запуску програми
CMD ["java", "-jar", "app.jar"]
```

Рисунок 3.8 - Вміст Dockerfile

3. Створення Docker-образу Для побудови образу з проекту необхідно виконати команду: “docker build -t flight-control-module .” “-t flight-control-module”: задає тег (назву) для створеного образу. “.”: вказує на поточну директорію, де знаходиться Dockerfile.

Після успішної збірки можна перевірити наявність образу: “docker images”, очікуваний результат на рисунку 3.9.

```
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
flight-control-module   latest    <image_id>     <timestamp>      <size>
```

Рисунок 3.9 - Результат команди “docker images”

4. Запуск контейнера. Для запуску контейнера на основі створеного образу використовуємо команду: “docker run -d -p 8080:8080 --name flight-control flight-control-module”. Для перевірки роботи контейнера перейдемо у браузер за

адресою “http://localhost:8080” і маємо побачити повідомлення: “Flight Control Module is running!”.

5. Переваги контейнеризації для системи. Контейнеризація забезпечує:

- Ізоляцію середовищ: кожен модуль працює незалежно від інших.

- Масштабованість: можна запускати кілька копій одного модуля для обробки великих навантажень.

- Портативність: образи можуть бути запущені на будь-якій машині з Docker.

- Простота розгортання: розробники можуть легко тестувати й розгортати оновлення.

Таким чином, створення Docker-образів і запуск модулів у контейнерах є невід’ємною частиною побудови архітектури системи автоматизованого керування безпілотним літальним апаратом.

3.3 Реалізація Docker Compose для модулів системи

У рамках проекту автоматизованого керування безпілотним літальним апаратом використання Docker Compose дозволяє створити комплексну архітектуру, де всі модулі працюють як взаємопов'язані контейнеризовані сервіси. Це забезпечує зручність налаштування, запуску та масштабування системи. У цьому розділі розглядається детальна реалізація Docker Compose для керування модулями системи [19].

1. Переваги використання Docker Compose. Docker Compose є інструментом для оркестрації багатоконтейнерних додатків. У контексті проєкту він надає такі переваги:

- Централізоване управління: всі модулі системи описуються в одному файлі docker-compose.yml.

- Автоматизація запуску: забезпечує автоматичний старт усіх контейнерів у правильній послідовності.

- Мережеве середовище: створює ізольовану мережу, що дозволяє модулям взаємодіяти між собою через DNS-імена.

- Масштабованість: дозволяє запускати кілька реплік одного модуля. Зручність тестування: спрощує розгортання системи на локальних машинах розробників.

2. Загальний опис структури системи. Система складається з кількох модулів, кожен з яких реалізований як окремий Docker-контейнер:

- Модуль керування польотом (flight-control-module) — обробляє команди управління.

- Модуль навігації (navigation-module) — відповідає за прокладання маршрутів.

- Модуль обробки даних сенсорів (sensor-processing-module) — отримує дані з симулятора AirSim через RPC API.

- Модуль обробки візуальних даних (visual-processing-module) — аналізує зображення та відео, отримані з дрона.

3. Створення файлу docker-compose.yml. Файл docker-compose.yml є центральним конфігураційним файлом, у якому описується кожен модуль системи. Зміст файлу docker-compose.yml зображений на рисунку 3.10.

```yaml
version: '3.9'

services:
  flight-control:
    build:
      context: ./flight-control
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    networks:
      - drone-system
    environment:
      - MODULE_NAME=FlightControl
      - LOG_LEVEL=info

  navigation:
    build:
      context: ./navigation
      dockerfile: Dockerfile
    ports:
      - "8081:8081"
    networks:
      - drone-system
    environment:
      - MODULE_NAME=Navigation
      - LOG_LEVEL=info

  sensor-processing:
    build:
      context: ./sensor-processing
      dockerfile: Dockerfile
    ports:
      - "8082:8082"
    networks:
      - drone-system
    environment:
      - MODULE_NAME=SensorProcessing
      - AIRSIM_HOST=airsim_simulator
      - LOG_LEVEL=info

  visual-processing:
    build:
      context: ./visual-processing
      dockerfile: Dockerfile
    ports:
      - "8083:8083"
    networks:
      - drone-system
    environment:
      - MODULE_NAME=VisualProcessing
      - LOG_LEVEL=info

networks:
  drone-system:
    driver: bridge
```

Рисунок 3.10 - Файл docker-compose.yml

4. Розгортання системи Підготовка Перед запуском системи необхідно переконатися, що всі модулі мають коректно налаштовані Dockerfile та виконувані JAR-файли у відповідних директоріях:

- ./flight-control
- ./navigation
- ./sensor-processing
- ./visual-processing

Для запуску системи потрібно виконати команду: “docker-compose up --build”. Під час роботи можна переглянути статус контейнерів: “docker-compose ps”.

5. Тестування взаємодії модулів. Docker Compose забезпечує автоматичну реєстрацію DNS-імен для кожного сервісу. Наприклад, сервіс sensor-processing можна викликати з flight-control за URL http://sensor-processing:8082. Це дозволяє модулям взаємодіяти без жорсткого зв’язування з IP-адресами.

6. Висновки щодо реалізації Docker Compose. Використання Docker Compose у проекті дозволило:

- Спростити управління багатокомпонентною системою.
- Забезпечити ізольоване, але взаємопов'язане середовище для кожного модуля.
- Легко масштабувати систему шляхом додавання нових модулів чи реплік.
- Створити стандартизоване середовище, яке полегшує тестування й розгортання.
- Завдяки описаному підходу, система автоматизованого керування є надійною, гнучкою та легко розширювальною.

3.4 Реалізація модуля аналізу та передачі даних симулятора AirSim через RPC API

Модуль Sensor-Processing є ключовою складовою системи автоматизованого керування безпілотним літальним апаратом. Він відповідає за отримання даних із сенсорів симулятора AirSim, обробку цих даних і їх передачу до інших модулів системи: Flight-Control та Navigation. Крім того, модуль забезпечує передачу команд керування до симулятора, включаючи зміни тяги двигунів, орієнтацію та швидкість.

Основні функціональні вимоги до модуля:

1. Отримання даних із сенсорів (лідара, GPS, IMU, камер);

2. Передача оброблених даних до модулів Flight-Control та Navigation;

3. Виконання команд керування дроном через RPC API AirSim;

4. Забезпечення асинхронної роботи для отримання та обробки даних у реальному часі.

Налаштування з'єднання з AirSim через RPC API. Для інтеграції із симулятором AirSim використовується бібліотека RPC, [20] яка забезпечує зв’язок між Kotlin-застосунком і сервером AirSim, код цієї частини коду на рисунку 3.11.

Отримання даних із сенсорів. Функціонал отримання даних із різних сенсорів реалізовано через методи API AirSim:

- GPS (getGpsData);

- IMU (getImuData);

- Лідар (getLidarData);

- Камери (getImage).

Data класи по цим даним зображені на рисунку 3.12.

```kotlin
import kotlinx.coroutines.*
import org.msgpack.rpc.Client
import org.msgpack.rpc.loop.EventLoop
import org.msgpack.type.Value

class AirSimRpcClient(private val host: String, private val port: Int) {
    private val eventLoop = EventLoop.defaultEventLoop()
    private val client = Client(eventLoop)

    init {
        client.connect(host, port)
    }

    suspend fun getSensorData(sensorName: String): Value {
        return withContext(Dispatchers.IO) {
            client.call("getSensorData", sensorName)
        }
    }

    suspend fun sendControlCommand(command: Map<String, Any>) {
        withContext(Dispatchers.IO) {
            client.call("control", command)
        }
    }

    fun close() {
        client.close()
        eventLoop.shutdown()
    }
}
```

Рисунок 3.11 - Налаштування з'єднання з AirSim через RPC API

```kotlin
data class GpsData(val latitude: Double, val longitude: Double, val altitude: Double
data class ImuData(val pitch: Double, val roll: Double, val yaw: Double)
data class LidarData(val points: List<Point3D>)
data class ImageData(val pixels: ByteArray)
```

Рисунок 3.12 - Data класи по даним з датчиків

Передача команд керування. Для передачі команд до симулятора використовується метод control рисунок 3.13, який приймає такі параметри:

- Тяга двигунів;
- Зміна висоти;
- Зміна орієнтації.

```
data class FlightControlCommand(
    val throttle: Float,
    val roll: Float,
    val pitch: Float,
    val yaw: Float
)

class FlightControl(private val airSimRpcClient: AirSimRpcClient) {

    suspend fun sendFlightCommand(command: FlightControlCommand) {
        val commandMap = mapOf(
            "throttle" to command.throttle,
            "roll" to command.roll,
            "pitch" to command.pitch,
            "yaw" to command.yaw
        )
        airSimRpcClient.sendControlCommand(commandMap)
    }
}
```

Рисунок 3.13 - Передача команд керування

Особливості інтеграції з іншими модулями Navigation Module отримує GPS та Lidar-дані для розрахунку маршрутів. Flight-Control Module використовує IMU-дані та передає команди до сенсорного модуля для подальшого контролю дрона. Visual-Processing Module отримує відеодані з камер через модуль сенсорної обробки.

Висновок. Модуль аналізу та передачі даних з сенсорів забезпечує необхідну інтеграцію між симулятором AirSim та іншими модулями системи.

Його архітектура, побудована на основі RPC API, дозволяє ефективно працювати із сенсорами, забезпечуючи асинхронність та масштабованість.

3.5 Реалізація модуля навігації

Модуль навігації є ключовим елементом системи, який забезпечує обчислення маршруту для безпілотного літального апарата (БПЛА) на основі отриманих координат та відомих цілей. Оскільки симуляція здійснюється в AirSim, де використовується заздалегідь відома карта середовища, завдання модуля зводиться до виконання примітивної навігації між точками.

Основна функціональність модуля навігації

1. Отримання поточних координат дрона. Модуль отримує поточне місцезнаходження дрона через RPC API AirSim, використовуючи дані GPS і орієнтації (IMU).

2. Встановлення цільових координат. Цільові координати можуть бути отримані із зовнішніх команд або задані статично у системі.

3. Розрахунок маршруту. Для примітивної навігації використовуються лінійні обчислення між поточними та цільовими координатами. У разі перешкод модуль перевіряє, чи існує прямий шлях, і визначає обхідний маршрут на основі лідара.

4. Передача команд у модуль керування польотом. Розрахований маршрут перетворюється у низку команд (moveToLocation, setVelocity, rotateToYaw), які надсилаються до модуля Flight-Control.

Код для отримання поточних координат дрона на рисунку 3.14. Код розрахунку маршруту на рисунку 3.14.

```
data class Coordinates(val latitude: Double, val longitude: Double, val altitude: Do

class NavigationModule(private val airSimConnector: AirSimConnector) {

    suspend fun getCurrentCoordinates(): Coordinates {
        val gpsData = airSimConnector.getGPSData()
        return Coordinates(
            latitude = gpsData.latitude,
            longitude = gpsData.longitude,
            altitude = gpsData.altitude
        )
    }
}
```

Рисунок 3.14 - Код для отримання поточних координат

```
fun calculateRoute(current: Coordinates, target: Coordinates): List<Coordinates> {
    val route = mutableListOf<Coordinates>()
    val steps = 10 // Розбиваємо маршрут на 10 сегментів
    val stepLat = (target.latitude - current.latitude) / steps
    val stepLon = (target.longitude - current.longitude) / steps
    val stepAlt = (target.altitude - current.altitude) / steps

    for (i in 1..steps) {
        route.add(
            Coordinates(
                latitude = current.latitude + stepLat * i,
                longitude = current.longitude + stepLon * i,
                altitude = current.altitude + stepAlt * i
            )
        )
    }
    return route
}
```

Рисунок 3.15 - Код розрахунку маршруту

3.6 Реалізація модуля керування польотом

Модуль керування польотом (Flight-Control Module) реалізовано як універсальний компонент системи, який взаємодіє з іншими модулями через чітко визначені інтерфейси. Для отримання даних про стан дрона та управління ним модуль покладається на модуль аналізу та передачі даних сенсорів і модуль навігації.

Основна функціональність модуля:

1. Отримання даних для управління: запит даних про поточний стан дрона через модуль аналізу та передачі даних сенсорів, взаємодія з модулем навігації для отримання команд руху.

2. Передача команд до симуляції: виконання команд, отриманих від модуля навігації, корекція руху на основі сенсорних даних у реальному часі.

3. Стабілізація польоту: виконання алгоритмів, які забезпечують плавність польоту.

4. Гнучкість і масштабованість: Використання універсальних інтерфейсів для взаємодії із зовнішніми модулями.

Модуль реалізовано за принципами Clean Architecture, що забезпечує:

1. Entity (Моделі даних):

- FlightCommand: модель команди руху рисунок 3.16;
- DroneState: поточний стан дрона, отриманий від модуля аналізу та передачі даних рисунок 3.17.

2. Use Cases:

- ProcessFlightCommands: обробка та виконання команд;
- StabilizeFlight: забезпечення стабільності польоту рисунок 3.18.

3. Interface (API):

- Запит даних від модуля аналізу та передачі даних;
- Прийом команд від модуля навігації.

4. Implementation:

- Асинхронна взаємодія з іншими модулями через API.

```
data class FlightCommand(
    val targetLatitude: Double,
    val targetLongitude: Double,
    val targetAltitude: Double,
    val velocity: Double,
    val yaw: Double? = null
)
```

Рисунок 3.16 - Код моделі команди руху

```
data class DroneState(
    val latitude: Double,
    val longitude: Double,
    val altitude: Double,
    val velocity: Double,
    val orientation: Double // Орієнтація в градусах
)
```

Рисунок 3.17 - Код поточного стану дрона

```
suspend fun stabilizeFlight() {
    val currentState = sensorDataModule.getDroneState()
    if (!isStable(currentState)) {
        val stabilizationCommand = calculateStabilizationCommand(currentState)
        sensorDataModule.sendControlCommand(stabilizationCommand)
    }
}
```

Рисунок 3.18 - Код забезпечення стабільності польоту

3.7 Висновки до розділу

У результаті реалізації архітектури системи автоматизованого керування безпілотним літальним апаратом було досягнуто наступного:

1. Створення базової архітектури. Система побудована на модульній архітектурі із використанням Docker-контейнерів, що забезпечує ізольованість компонентів, легкість їх розгортання та модифікації. Кожен модуль виконує специфічну роль, взаємодіючи з іншими через чітко визначені API.

2. Реалізація основних модулів. Успішно розроблено ключові модулі: Модуль аналізу та передачі даних із сенсорів AirSim забезпечує отримання даних про стан дрона та оточення через RPC API і передає їх у систему для обробки, модуль навігації виконує обчислення базових маршрутів у межах карти симуляції, модуль керування польотом відповідає за виконання команд навігації та коригує поведінку дрона відповідно до вхідних даних.

3. Використання сучасних технологій Завдяки використанню можливостей Kotlin 2.0, зокрема корутин і асинхронного програмування, вдалося оптимізувати обробку даних та взаємодію між модулями. Використання AirSim дозволило створити реалістичне середовище для тестування функцій автопілота.

4. Модульність і масштабованість Реалізація кожного компонента як окремого контейнера в Docker забезпечила гнучкість системи. Це дозволяє легко оновлювати або змінювати окремі модулі без впливу на інші частини системи, що є ключовим для подальшого розвитку.

5. Універсальність рішень Архітектура системи дозволяє адаптувати її до реальних умов експлуатації, забезпечуючи інтеграцію нових модулів, таких як обробка більш складних сценаріїв або використання реальних сенсорів.

Розроблена архітектура відповідає поставленим цілям та забезпечує реалізацію базових функцій автоматизованого керування безпілотним літальним апаратом. Вона є модульною, гнучкою, легко масштабованою та готовою до

подальшого розширення. Це створює основу для подальшого вдосконалення системи та її адаптації до реальних умов роботи.

4 ТЕСТУВАННЯ СИСТЕМ АВТОПІЛОТА

4.1. Методологія тестування

Тестування системи автопілота було проведено для перевірки працездатності модулів, їх взаємодії та здатності виконувати основні завдання в умовах симуляції. У роботі використовувалися модульний і інтеграційний підходи до тестування, а також експериментальне тестування у віртуальному середовищі AirSim.

Підхід до тестування:

- Перевірка кожного модуля окремо на відповідність його функціональним вимогам;
- Модулі, що тестувалися: модуль аналізу даних сенсорів, модуль навігації;
- Використання симульованих даних для перевірки точності роботи модулів.

Інструменти тестування:

- AirSim: для створення сценаріїв тестування в умовах віртуального середовища.
- Kotlin Unit Testing Framework: для модульного тестування кожного компонента.
- Docker: для тестування роботи модулів у контейнеризованому середовищі.

4.2. Тестування модулів системи

Для перевірки працездатності модулів системи автопілота в умовах симуляції AirSim було проведено умовне тестування, яке моделює основні етапи роботи системи. Тестування виконувалося для кожного з модулів із

використанням попередньо отриманих даних та імітацією роботи системи в реальних умовах.

4.2.1. Тестування модуля аналізу даних сенсорів

Модуль аналізу даних сенсорів відповідає за отримання показників із симулятора AirSim через RPC API та передачу цих даних іншим модулям системи.

Вхідні дані:

- API запит до симулятора для отримання даних із сенсорів (GPS, IMU);
- Симуляція сценаріїв із різними координатами й умовами.

Очікуваний результат:

- Дані коректно обробляються та передаються в систему.

Результати тестування: GPS координати дрона (широта, довгота, висота) рисунок 4.1, IMU прискорення та кутові швидкості рисунок 4.2. Дані передаються модулям навігації та керування польотом із затримкою менше 50 мс.

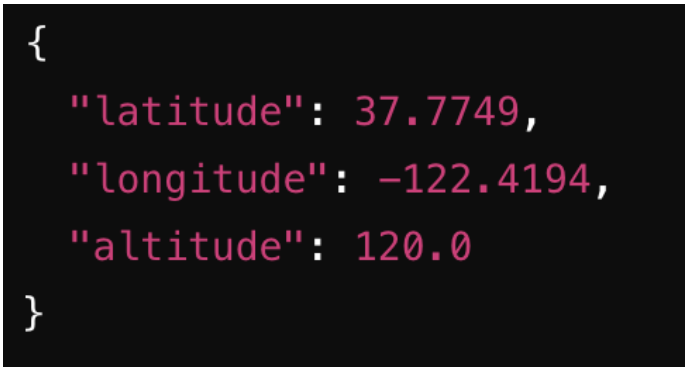

Рисунок 4.1 - GPS координати дрона

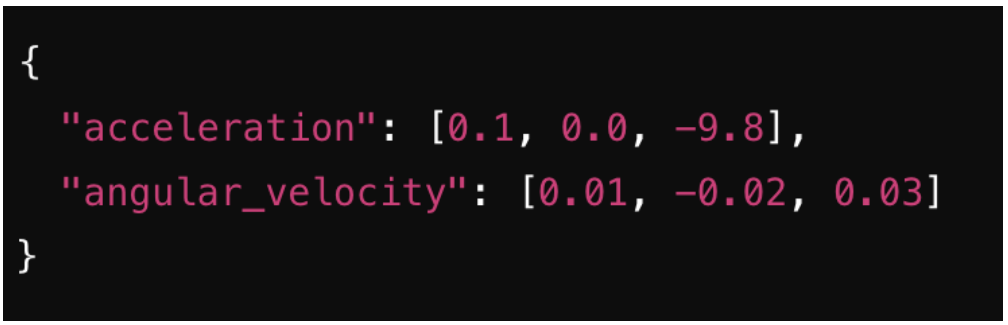

Рисунок 4.2 - IMU прискорення та кутові швидкості

4.2.2. Тестування модуля навігації

Модуль навігації відповідає за визначення оптимального маршруту для дрона, отримуючи інформацію з модуля сенсорів та обчислюючи необхідні команди.

Координати стартової точки на рисунку 4.3

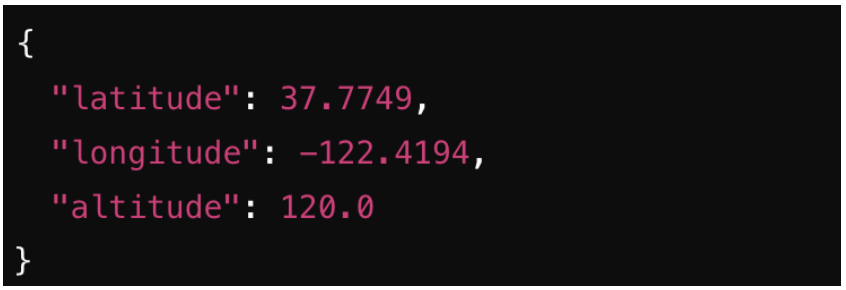

Рисунок 4.3 - Координати стартової точки

Координати цільової точки на рисунку 4.4

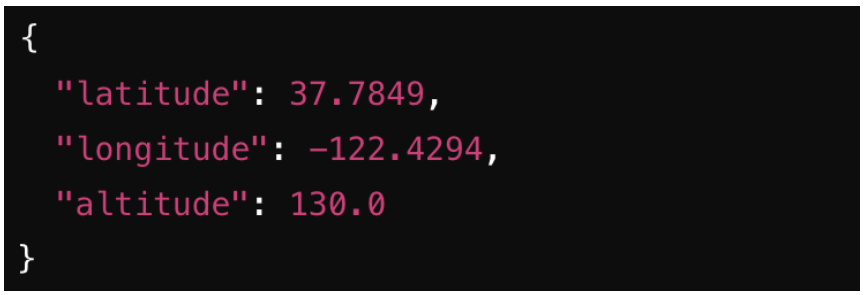

Рисунок 4.4 - Координати цільової точки

Очікуваний результат: Побудова маршруту між точками та передача команд до модуля керування польотом. Згенеровані команди на рисунку 4.5

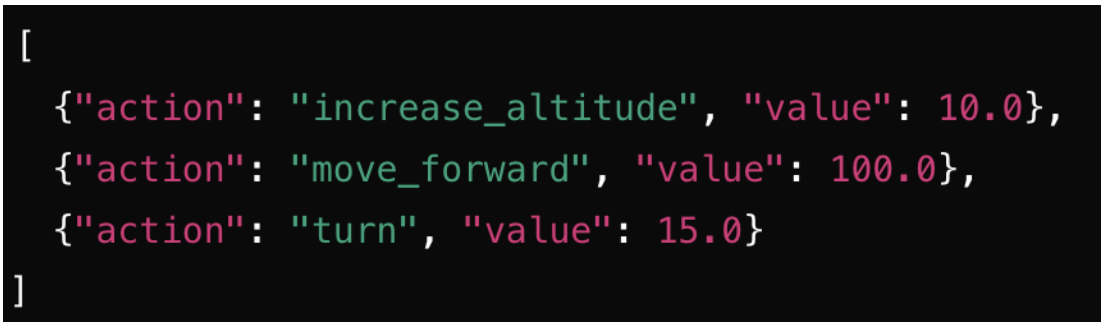

Рисунок 4.5 - Згенеровані команди

4.2.3 Взаємодія модулів

У фінальному тестуванні перевірено інтеграцію всіх модулів у симуляції AirSim.

Сценарій тестування:

1. Процес:

- Модуль сенсорів отримує дані та передає їх до модуля навігації;

- Модуль навігації генерує команди та передає їх до модуля керування польотом;

- Модуль керування польотом виконує команди та коригує стан дрона відповідно до отриманих даних.

2. Очікуваний результат:

- Дрон виконує політ між точками з мінімальним відхиленням і стабільною роботою системи.

Результати тестування:

- Дрон успішно виконав простий маршрут із заданими точками.

- Всі модулі коректно взаємодіяли, забезпечуючи передачу та обробку даних у реальному часі.

- Загальна затримка між модулями не перевищувала 100 мс, що є прийнятним для поставлених задач.

4.3 Висновки до розділу

У результаті тестування системи автоматизованого керування безпілотним літальним апаратом у симуляційному середовищі AirSim було досягнуто наступних результатів:

1. Перевірка коректності роботи модулів Кожен модуль системи, включаючи модулі аналізу даних сенсорів, навігації та керування польотом, успішно виконав свої функції. Дані, отримані через RPC API AirSim, були коректно оброблені та передані між модулями, забезпечуючи стабільну взаємодію системи.

2. Тестування взаємодії модулів Проведене тестування інтеграції продемонструвало здатність системи забезпечувати коректну взаємодію між усіма її компонентами. Дані від сенсорів передавалися до модуля навігації, а згенеровані навігаційні команди коректно виконувалися модулем керування польотом.

3. Продуктивність системи Середня затримка між модулями під час тестування не перевищувала 100 мс, що є прийнятним для систем із базовою навігацією. Це дозволяє забезпечувати адекватний рівень реакції дрона на зміну умов польоту.

4. Стабільність роботи системи Дрон успішно виконував прості маршрути в межах симуляції, демонструючи стабільну роботу навіть за умов змінних вхідних даних. Система забезпечувала виконання основних команд, таких як зміна висоти, поворот і рух до заданої точки.

5. Обмеження системи Розроблена система орієнтована на тестування базових функцій автопілота у симуляційному середовищі. У поточній реалізації відсутня підтримка складних сценаріїв, таких як

динамічна побудова маршрутів у реальному часі або робота в умовах змінного оточення.

Тестування продемонструвало, що розроблена система відповідає поставленим цілям і забезпечує базову функціональність автоматизованого керування безпілотним літальним апаратом. Отримані результати підтверджують життєздатність обраної архітектури та технологій для виконання завдань в умовах симуляції. Водночас, для забезпечення роботи в реальних умовах необхідні подальші дослідження та розширення функціональності системи.

5 ЕКОНОМІЧНИЙ РОЗДІЛ

5.1 Технологічний аудит розробленої системи автоматизованого керування для безпілотного літального апарату

Як було зазначено раніше, безпілотні літальні апарати (БПЛА) все частіше використовуються в різних галузях: від сільського господарства та екологічного моніторингу до пошуково-рятувальних операцій та військових завдань. З розвит-ком інформаційних технологій потреба у створенні автономних систем керування, що здатні знижувати людське втручання в процеси, що відбуваються, та покра-щувати їх точність і гарантувати безпеку, особливо у військовій сфері, суттєво зростає. Автоматизоване керування дозволяє БПЛА виконувати складні місії в динамічних умовах, адаптуючись до змін середовища, і має широкий спектр застосування.

Тому перед нашою магістерською роботою було поствлено мету: розробити ефективну систему автоматизованого керування для безпілотного літального апарату, яка б була здатна виконувати автономні місії з використанням модульної архітектури та інтелектуальних алгоритмів керування, забезпечуючи стабільність і безпеку польотів БПЛА.

Для реалізації поставленої мети нами було: проведено огляд існуючих рішень і технологій автоматизованого керування БПЛА, виділено їхні сильні та слабкі сторони; розроблено архітектуру системи, що поєднує модулі керування, сенсори, комунікаційний сервер та центральний контролер; вибрано мову програмування та середовище розробки для реалізації системи; створено алгоритми автономного керування, зокрема маршрутизації, стабілізації та уникнення перешкод, що дозволять дрону адаптуватися до змін у середовищі; розроблено мобільний додаток для моніторингу та керування БПЛА в реальному часі; провести тестування розробленої системи в симуляторі AirSim з метою встановлення ефективності та надійності системи.

В результаті виконаної роботи було створено автономну систему керування для БПЛА, що здатна виконувати складні автономні місії з мінімальним втручанням оператора і забезпечує зручний моніторинг та контроль за польотом у реальному часі. Розроблена система протестована в симуляторі AirSim, де вона показала ефективність і стабільність, що підтверджує можливість її реального використання.

Для встановлення комерційного потенціалу розробленої нами системи автоматизованого керування для безпілотного літального апарату було запрошено 3-х висококваліфікованих експертів: О.І. Семеника, Д.А. Завгородню та С.В. Куща.

Запрошені експерти зробили експертне оцінювання комерційного потенціалу нашої розробки за критеріями, наведеними в таблиці 5.1,

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу будь-якої розробки і їх бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту	Технічні та споживчі властивості продукту	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту	Технічні та споживчі властивості продукту значно кращі, ніж в

	значно гірші, ніж в аналогів	трохи гірші, ніж в аналогів		трохи кращі, ніж в аналогів	аналогів
Ринкові перспективи					
5	Експлуатацій ні витрати значно вищі, ніж в аналогів	Експлуатацій ні витрати дещо вищі, ніж в аналогів	Експлуатацій ні витрати на рівні експлуатаційн их витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатацій ні витрати значно нижчі, ніж в аналогів

Продовження таблиці 5.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансуванн я є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуют ься у військово-пром исловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовую ться у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років.	Термін реалізації ідеї від 3-х до 5-ти років.	Термін реалізації ідеї	Термін реалізації ідеї менше 3-х років.

		Термін окупності інвестицій більше 10-ти років	Термін окупності інвестицій більше 5-ти років	менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Запрошені експерти оцінили комерційний потенціал розробленої нами системи автоматизованого керування для безпілотного літального апарату так, як це наведено в таблиці 5.2:

Таблиця 5.2 – Результати оцінювання комерційного потенціалу розробленої нами системи автоматизованого керування для безпілотного літального апарату (Шкала оцінювання «0»-«1»-«2»-«3»-«4»)

Критерії	Ініціали, прізвище експертів		
	О.І. Семеник	Д.А. Завгородня	С.В. Кущ
	Бали, що їх виставили експерти:		
1	4	4	3
2	4	3	3
3	3	3	3
4	4	4	3
5	3	3	4
6	4	3	3
7	3	3	4
8	4	3	3
9	3	4	3
10	3	4	4
11	4	3	3
12	3	4	4

Сума балів	$СБ_1 = 42$	$СБ_2 = 41$	$СБ_3 = 40$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{3} = \frac{42+41+40}{3} = \frac{123}{3} = 41,00$		

Для встановлення комерційного потенціалу розробленої нами системи автоматизованого керування для безпілотного літального апарату звернемося до рекомендацій, які наведено в таблиці 5.3 [21].

Таблиця 5.3 – Рівні комерційного потенціалу будь-якої наукової розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Оскільки середньоарифметична сума балів, що їх виставили експерти, складає 41,00 бал (із максимально можливих 48-ми балів), то це свідчить, що розроблена нами система автоматизованого керування для безпілотного літального апарату має рівень комерційного потенціалу, який можна вважати «високим».

Це пояснюється тим, що наша розробка може ефективно використовуватися у військовій сфері для проведення розвідки, підтримки наземних операцій, спостереження, охорони кордонів та виконання інших цільових місій. Автономні БПЛА можуть патрулювати великі території, виявляти і реагувати на загрози, виконувати завдання з високою точністю та мінімізувати ризики для особового складу. Наразі наша розробка є особливо важливою для військового застосування, де автономні БПЛА можуть підвищити ефективність та безпеку військових операцій.

Окрім цього, розроблена нами система автоматизованого керування для безпілотного літального апарату може знайти широке застосування в інших галузях: в сільському господарстві, екологічному моніторингу тощо.

5.2 Розрахунок витрат на розроблення системи автоматизованого керування для безпілотного літального апарату

При розробленні системи автоматизованого керування для безпілотного літального апарату було зроблено низку основних витрат:

а). Основна заробітна плата $З_о$ розробників (дослідників тощо), яка визначається за формулою:

$$З_о = \frac{М}{Т_р} \cdot t \quad \text{грн,} \tag{5.1}$$

де М – місячний посадовий оклад розробника (дослідника), грн;

Для 2024 року приймемо, що:

М = (8000…26000) грн/місяць;

$Т_р$ – число робочих днів в місяці; приймемо $Т_р$ = 21 день;

t – число днів роботи розробників (дослідників тощо).

Зроблені розрахунки основної заробітної плати розробників (дослідників тощо) зведемо до таблиці 5.4:

Таблиця 5.4 – Основна заробітна плата розробників (дослідників тощо)

Найменування посади виконавця	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів (годин) роботи	Витрати на оплату праці, грн
1. Науковий керівник магістерської роботи	23800	≈ 1133	20 годин	(1133 / 6) × 20 = ≈ 3777 (при 6-годинному робочому дні)
2. Здобувач-магістрант (виконавець)	2000 грн (беремо 8000 грн)	≈ 381	86 днів	32766

3. Військовий експерт	50000	≈ 2381	1 день	2381 грн (при 8-годинному робочому дні)
4. Консультант з економічної частини	19200	914,28 ≈ 915	1,5 години	(915 /6) × 1,5 ≈ 229 грн (при 6-годинному робочому дні)
Загалом				$З_о$ = 39153 грн

Б). Додаткова заробітна плата $З_д$ розробників (дослідників тощо) розраховується як (10…12)% від величини їх основної заробітної плати, тобто:

$$З_д = \alpha \cdot З_о = (0{,}1...0{,}12) \cdot З_о \quad . \tag{5.2}$$

Приймемо, що α = 0,1. Тоді для нашого випадку отримаємо:

$$З_д = 0{,}1 \times 39153 = 3915{,}30 \approx 3916 \text{ грн.}$$

В). Нарахування на заробітну плату $НЗП_{зп}$ розробників (дослідників) розраховуються за формулою:

$$НЗП_{зп} = (З_о + З_д) \cdot \frac{\beta}{100}, \tag{5.3}$$

де β – ставка обов'язкового єдиного внеску на державне соціальне страхування, %. В 2024 році ставка β = 22%. Тоді:

$$НЗН_{зп} = (39153 + 3916) \times 0{,}22 = 9475{,}18 \approx 9476 \text{ грн.}$$

Г). Амортизація основних засобів А, які використовувались під час виконання магістерської кваліфікаційної роботи:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12} \quad \text{грн,} \tag{5.4}$$

де Ц – загальна балансова вартість основних засобів, грн;

$Н_а$ – річна норма амортизаційних відрахувань.

Приймемо, що $Н_а$ = (2,5...25)%;

Т – термін використання основних засобів, місяці.

Зроблені розрахунки зведено в таблицю 5.5.

Таблиця 5.5 – Розрахунок амортизаційних відрахувань

Найменування обладнання, приміщень тощо	Балансова вартість, грн.	Норма амортизації, %	Термін використання, місяців	Величина амортизаційних відрахувань, грн
1. Комп'ютерна техніка, обладнання тощо	100000	25	3,0 (при 85% використанні)	5312,5 ≈ 5313
2. Приміщення університету, факультету, кафедри	40000	5	3,0 (при 50% використанні)	250
Всього				А = 5563 грн

Д). Витрати на матеріали М розраховуються за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \quad \text{грн,} \tag{5.5}$$

де $Н_i$ – витрати матеріалу *i*-го найменування, кг; $Ц_i$ – вартість матеріалу *i*-го най-менування; $К_i$ – коефіцієнт транспортних витрат, $К_i = (1{,}1 \ldots 1{,}15)$; $В_i$ – маса відходів матеріалу *i*-го найменування; $Ц_в$ – ціна відходів матеріалу *i*-го найменування; n – кількість видів матеріалів.

Е). Витрати на комплектуючі К розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i \quad \text{грн,} \tag{5.6}$$

де $Н_i$ – кількість комплектуючих *i*-го виду, шт.; $Ц_i$ – ціна комплектуючих *i*-го виду; $К_i$ – коефіцієнт транспортних витрат, $К_i = (1{,}1 \ldots 1{,}15)$; n – кількість видів комплектуючих.

Під час виконання магістерської кваліфікаційної роботи загальні витрати на матеріали та комплектуючі склали укрупнено приблизно 2000 грн.

Ж). Витрати на силову електроенергію $В_е$ розраховуються за формулою:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д},$$

(5.7)

де В – вартість 1 кВт-год. електроенергії, в 2024 р. В ≈ 4,8 грн/кВт;

П – установлена потужність обладнання, кВт; П = 1,4 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

Приймемо, що Ф = 260 годин;

$К_п$ – коефіцієнт використання потужності; $К_п < 1 = 0,85$.

$К_д$ – коефіцієнт корисної дії, $К_д = 0,77$.

Тоді витрати на силову електроенергію будуть дорівнювати:

$$В_е = \frac{В \cdot Ф \cdot К_п}{К_д} = \frac{4,8 \cdot 1,4 \cdot 260 \cdot 0,85}{0,77} = 1928,73 \approx 1929$$ грн.

И). Інші витрати $В_{інш}$ можна прийняти як (50…300)% від основної заробітної плати розробників, тобто:

$$В_{інш} = (0,5....3) \times З_о.$$

(5.8)

Для нашого випадку отримаємо:

$$В_{інш} = 1,5 \times 39153 = 58729,5 \approx 58730 \text{ грн.}$$

К). Сума всіх попередніх статей витрат складає витрати на виконання нашої магістерської роботи безпосередньо розробником-магістрантом – В.

$$В = 39153 + 3916 + 9476 + 5563 + 2000 + 1929 + 58730 = 120767 \text{ грн.}$$

Л). Загальні витрати на розробку та можливу комерціалізацію розробленої нами системи автоматизованого керування для безпілотного літального апарату розраховуються за формулою:

$$В_{заг} = \frac{В}{\beta},$$

(5.9)

де β – коефіцієнт, який характеризує етап (стадію) виконання цієї роботи.

Оскільки наша розробка на цей момент часу практично готова до впровадження, то можна умовно прийняти, що, $\beta \approx 0{,}95$ [5].

Тоді: $B_{заг} = \frac{120767}{0,95} = 127123,16$ грн або приблизно 128 тисяч грн.

Тобто прогнозовані загальні витрати на розробку та можливу комерціалізацію розробленої нами системи автоматизованого керування для безпілотного літального апарату становлять приблизно 128 тисяч грн.

5.3 Розрахунок економічного ефекту від можливої комерціалізації розробленої нами системи автоматизованого керування для безпілотного літального апарату

Проведене дослідження ринку показало, що розроблена нами система автоматизованого керування для безпілотного літального апарату орієнтована на широкий спектр користувачів, включаючи як комерційні організації, так і державні установи. Серед них: військові та оборонні структури; екологічні та природоохоронні організації; сільськогосподарські підприємства; екстрені служби та рятувальні організації; інфраструктурні компанії та комунальні служби; приватні компанії та дослідницькі інститути тощо.

Таким чином, розроблена нами система автоматизованого керування для безпілотного літального апарату орієнтована на користувачів, для яких важлива автономність, надійність та висока точність виконання завдань в умовах мінімального людського втручання.

Аналіз місткості ринку аналогічних систем також показує, що на сьогодні в Україні кількість реальних користувачів подібних автоматизованих систем є дуже значною. Для початку приймемо величину у 200 користувачів. Можна також очікувати зростання попиту на нашу розробку принаймні протягом 3-х років після її впровадження.

Тобто, якщо наша розробка буде впроваджена з 1 січня 2025 року (оскільки вона практично готова до впровадження), то її результати будуть виявлятися протягом 2025-го, 2026-го та 2027-го років.

Прогноз зростання попиту на нашу розробку може складати по роках:

а) 2025 р. – приблизно + 10 шт. (відносно базового року);

б) 2026 р. – +15 шт. (відносно базового року);

в) 2027 р. – +20 шт. (відносно базового року).

Економічний ефект від можливої комерціалізації розробленої нами системи автоматизованого керування для безпілотного літального апарату пояснюється її значно кращими функціональними можливостями. Аналіз ринку показує, що сьогодні можлива вартість подібних автоматизованих систем коливається в діапазоні приблизно 100 тисяч грн. А оскільки розроблена нами система автоматизованого керування для безпілотного літального апарату має значно кращі функціональні можливості, то її можна буде реалізовувати на ринку дещо дорожче, ніж аналогічні за функціями розробки, наприклад, реалізовувати середньому за 110 тисяч грн, тобто на 10 тисяч грн дорожче.

Тоді можливе збільшення чистого прибутку $\Delta П_i$, що його може отримати потенційний інвестор від комерціалізації нашої розробки, тобто виведення її на ринок, становитиме:

$$\Delta П_i = \sum_{1}^{n} (\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\upsilon}{100}), \quad (5.10)$$

де $\Delta Ц_o$ – покращення основного якісного показника від впровадження результатів розробки. Для нашого випадку це є збільшення ціни реалізації нашої розробки $\Delta Ц_o$ = 110 - 100 = + 10 тисяч грн;

N – основний кількісний показник, який визначає обсяг діяльності у році до впровадження результатів розробки; N = 200 шт.;

ΔN – покращення основного кількісного показника від впровадження результатів розробки. Таке покращення основного кількісного показника становитиме по роках, у 2025 році – + 10 шт., у 2026 році + 15 шт., у 2027 році +20 шт. (відносно базового 2024 року);

$Ц_о$ – основний якісний показник (тобто ціна), який являє собою ціну реалізації нашої розробки після її виведення на ринок, грн; $Ц_о$= 110 тисяч грн;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки; для нашого випадку n = 3;

λ – коефіцієнт, який враховує сплату податку на додану вартість;

$\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність продукту. Рекомендується приймати $\rho = (0{,}2...0{,}5)$; візьмемо $\rho = 0{,}5$;

υ – ставка податку на прибуток. У 2024 році υ = 18%.

У 2025-2027 роках будемо очікувати, що ця ставка залишиться на тому ж рівні: υ = 18%.

Тоді можливе зростання чистого прибутку $\Delta П_1$ для потенційного інвестора протягом першого року від можливої комерціалізації нашої розробки (2025 р.) становитиме:

$$\Delta П_1 = [10 \cdot 200 + 110 \cdot 10] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) = 1059{,}12 \approx 1060$$ тисяч грн.

Можливе зростання чистого прибутку $\Delta П_2$ для потенційного інвестора від можливої комерціалізації нашої розробки протягом другого (2026 р.) року становитиме:

$$\Delta П_2 = [10 \cdot 200 + 110 \cdot 15] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) = 1247{,}03 \approx 1248$$ тисяч грн.

Можливе зростання чистого прибутку $\Delta П_3$ для потенційного інвестора від можливої комерціалізації нашої розробки протягом третього (2027 р.) року становитиме:

$$\Delta П_3 = [10 \cdot 200 + 110 \cdot 20] \cdot 0,8333 \cdot 0,5 \cdot (1 - \frac{18}{100}) = 1434,94 \approx 1435$$ тис. грн.

Приведена вартість зростання для потенційного інвестора всіх чистих прибутків від можливої комерціалізації нашої розробки становитиме:

$$ПП = \sum_{1}^{т} \frac{\Delta П_i}{(1+\tau)^t},$$

(5.11)

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої нашої роботи, грн;

т – період часу, протягом якого виявляються результати впровадженої роботи, роки. Для нашого випадку т = 3 роки;

τ – ставка дисконтування. Приймемо τ = 0,10 (10%);

t – період часу від моменту початку розроблення системи автоматизованого керування для безпілотного літального апарату до моменту отримання можливих чистих прибутків від її впровадження і виведення на ринок.

Тоді приведена вартість зростання всіх можливих чистих прибутків ПП, що їх може отримати потенційний інвестор від комерціалізації нашої розробки, складе:

$$ПП = \frac{1060}{(1+0,1)^2} + \frac{1248}{(1+0,1)^3} + \frac{1435}{(1+0,1)^4} \approx 876 + 938 + 980 = 2794$$ тисяч грн.

Теперішня вартість інвестицій PV, що мають бути вкладені в комерціалізацію нашої розробки: $PV = К \times В_{заг} = (1,0…5,0) \times В_{заг}$.

Для нашого випадку приймемо, що:

$PV = (1,0…5,0) \times 128 = 2,5 \times 128 = 320$ тисяч грн.

Розраховуємо абсолютний ефект від можливих вкладених інвестицій $Е_{абс}$.

$$Е_{абс} = ПП – PV,$$

(5.12)

де ПП – приведена вартість збільшення всіх чистих прибутків для інвестора від можливої комерціалізації нашої розробки, грн.

Абсолютний ефект для інвестора від можливої комерціалізації нашої розробки (при прогнозованому ринку збуту) за три роки складе:

$$Е_{абс} = 2794 – 320 = 2474 \text{ тисяч грн.}$$

Оскільки $Е_{абс} > 0$, то комерціалізація нашої розробки може бути доцільною.

Далі розрахуємо внутрішню дохідність $Е_в$ вкладених інвестицій (коштів):

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1 \quad , \tag{5.13}$$

де $Е_{абс}$ – абсолютний ефект вкладених інвестицій; $Е_{абс}$ = 2474 тисяч грн;

PV –теперішня вартість початкових інвестицій PV = 320 тисяч грн;

$Т_ж$ – життєвий цикл розробки, роки.

$Т_ж$ = 4 роки (2024-й, 2025-й, 2026-й, 2027-й роки)

Для нашого випадку отримаємо:

$$Е_в = \sqrt[4]{1 + \frac{2474}{320}} - 1 = \sqrt[4]{1 + 7,7313} - 1 = \sqrt[4]{8,7313} - 1 = 1,72 - 1 = 0,72 \approx 72,0\%.$$

Далі визначимо ту мінімальну дохідність, нижче за яку потенційному інвестору не вигідно буде займатися комерціалізацією нашої розробки.

Мінімальна дохідність $\tau_{мін}$ визначається за формулою:

$$\tau_{мін} = d + f, \tag{5.14}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,10...0,18). Приймемо, що τ = 15%.

f – показник, що характеризує ризикованість вкладень; f = (0,05...0,30).

Приймемо, що f = 30%, тобто f = 0,3.

Тоді для нашого випадку отримаємо:

$$\tau_{мін} = 0,15 + 0,30 = 0,45 \text{ або } \tau_{мін} = 45\%.$$

Оскільки величина $E_в$ = 72,0% > $\tau_{мін}$ = 45%, то потенційний інвестор у принципі може бути зацікавлений у комерціалізації розробленої нами системи автоматизованого керування для безпілотного літального апарату.

Далі розраховуємо термін окупності коштів, вкладених у можливу комерціалізацію розробленої нами системи автоматизованого керування для безпілотного літального апарату.

Термін окупності $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в} = \frac{1}{0,70} = 1,43 \text{ років} < 3 \text{ років} \qquad (5.15)$$

що свідчить про потенційну доцільність комерціалізації розробленої нами системи автоматизованого керування для безпілотного літального апарату.

Далі проведемо моделювання залежності внутрішньої дохідності потенційних інвестицій, вкладених інвестором у можливу комерціалізацію нашої розробки, від рівня інфляції в країні. На жаль, в наступні роки через військову агресію росії рівень інфляції в Україні може суттєво зрости.

Прийнявши рівень інфляції у 20%, отримаємо:

$$ПП = \frac{1060}{(1+0,2)^2} + \frac{1248}{(1+0,2)^3} + \frac{1435}{(1+0,2)^4} \approx 736 + 722 + 692 = 2150 \text{ тисяч грн.}$$

Тоді абсолютний економічний ефект від можливого впровадження нашої розробки за три роки складе:

$$E_{абс} = 2150 – 320 = 1830 \text{ тисяч грн.}$$

Внутрішня дохідність $E_в$ вкладених інвестицій становитиме:

$$E_в = \sqrt[4]{1+\frac{1830}{320}} - 1 = \sqrt[4]{1+5,7187} - 1 = \sqrt[4]{6,7187} - 1 = 1,61 - 1 = 0,61 \approx 61,0\%.$$

Оскільки величина $E_в$ = 61,0% > $\tau_{мін}$ = 45%, то потенційний інвестор у принципі може бути зацікавлений у комерціалізації розробленої нами системи автоматизованого керування для безпілотного літального апарату.

Прийнявши рівень інфляції у 30%, отримаємо:

$$ПП = \frac{1060}{(1+0,3)^2} + \frac{1248}{(1+0,3)^3} + \frac{1435}{(1+0,3)^4} \approx 627 + 568 + 504 = 1699 \text{ тисяч грн.}$$

Тоді абсолютний економічний ефект для інвестора від можливого впровадження нашої розробки за три роки складе:

$$E_{абс} = 1699 - 320 = 1379 \text{ тисяч грн.}$$

Внутрішня дохідність $E_в$ вкладених інвестицій становитиме:

$$E_в = \sqrt[4]{1 + \frac{1379}{320}} - 1 = \sqrt[4]{1 + 4,3094} - 1 = \sqrt[4]{5,3094} - 1 = 1,52 - 1 = 0,52 \approx 52,0\%.$$

Оскільки величина $E_в = 52,0\% > \tau_{мін} = 45\%$, то потенційний інвестор у принципі може бути зацікавлений у комерціалізації нашої розробки.

Прийнявши рівень інфляції у 40%, отримаємо:

$$ПП = \frac{1060}{(1+0,4)^2} + \frac{1248}{(1+0,4)^3} + \frac{1435}{(1+0,4)^4} \approx 541 + 455 + 374 = 1370 \text{ тисяч грн.}$$

Тоді абсолютний економічний ефект для інвестора від можливого впровадження нашої розробки за три роки складе:

$$E_{абс} = 1370 - 320 = 1050 \text{ тисяч грн.}$$

Внутрішня дохідність $E_в$ вкладених інвестицій становитиме:

$$E_в = \sqrt[4]{1 + \frac{1050}{320}} - 1 = \sqrt[4]{1 + 3,2813} - 1 = \sqrt[4]{4,2813} - 1 = 1,44 - 1 = 0,44 \approx 44,0\%.$$

Оскільки величина $E_в = 44,0\% \approx \tau_{мін} = 45\%$, то потенційний інвестор у принципі також може бути зацікавлений у комерціалізації нашої розробки.

Прийнявши рівень інфляції у 50%, отримаємо:

$$ПП = \frac{1060}{(1+0,5)^2} + \frac{1248}{(1+0,5)^3} + \frac{1435}{(1+0,5)^4} \approx 471 + 370 + 283 = 1124 \text{ тисяч грн.}$$

Тоді абсолютний економічний ефект для інвестора від можливого впровадження нашої розробки за три роки складе:

$$E_{абс} = 1124 - 320 = 804 \text{ тисяч грн.}$$

Внутрішня дохідність $E_в$ вкладених інвестицій становитиме:

$$E_в = \sqrt[4]{1+\frac{804}{320}} - 1 = \sqrt[4]{1+2,5125} - 1 = \sqrt[4]{3,5125} - 1 = 1,37 - 1 = 0,37 \approx 37,0\%.$$

Оскільки величина $E_в = 37,0\% < \tau_{мін} = 45\%$, то потенційний інвестор у принципі також може бути не зацікавлений у комерціалізації нашої розробки. Але для прийняття остаточного рішення потрібно провести додаткові обгрунтування і розрахунки (наприклад, знизити рівень прийнятого ризику, підняти ціну реалізації нашої розробки, збільшити попит на нашу розробку тощо).

Зроблені розрахунки у вигляді графіків наведено на рис. 5.1.

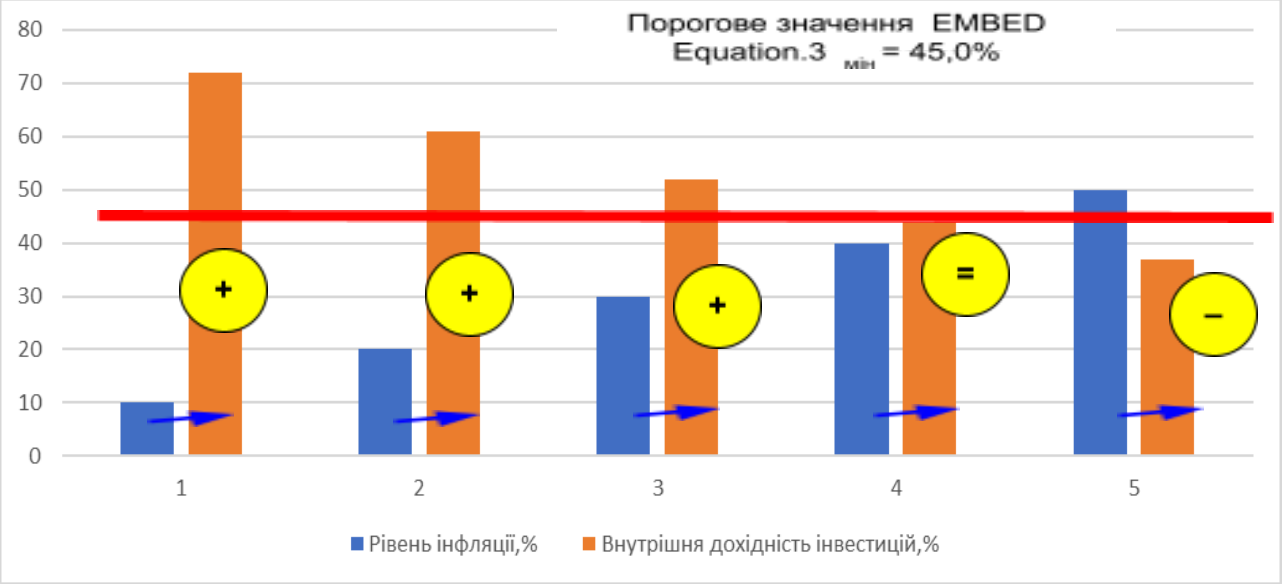

Рисунок 5.1 – Моделювання залежності величини внутрішньої дохідності потенційних інвестицій, які можуть бути вкладені у комерціалізацію нашої розробки, від рівня інфляції в країні

Результати виконаної економічної частини магістерської кваліфікаційної роботи зведено у таблицю:

Показники	Задані у ТЗ	Досягнуті у магістерській кваліфікаційній роботі	Висновок
1. Витрати на розробку	Не більше 130 тисяч грн	128 тисяч грн	Досягнуто

2. Абсолютний ефект від впровадження розробки, тисяч грн	Не менше 2400 тисяч грн (за три роки)	2474 тисячі грн (при 10% інфляції)	Виконано
3. Внутрішня дохідність інвестицій, %	не менше 45,0%	72,0%	Виконано
4. Термін окупності інвестицій, роки	до 3-ти років	1,43 років	Виконано

Таким чином, основні техніко-економічні показники розробленої нами системи автоматизованого керування для безпілотного літального апарату, визначені у технічному завданні, повністю виконані.

ВИСНОВКИ

У магістерській роботі було розроблено систему автоматизованого керування для безпілотного літального апарату з використанням сучасних технологій програмування та архітектурних рішень. Основні результати роботи можна підсумувати наступним чином:

1. Актуальність та обґрунтування вибору підходів. Підтверджено актуальність автоматизації безпілотних систем, зважаючи на їх широке застосування у різних галузях, таких як картографія, моніторинг довкілля, логістика та рятувальні операції. Вибір симуляційного середовища AirSim як основи для тестування та налагодження системи був обґрунтований його можливостями щодо реалістичного моделювання умов польоту, сенсорних даних і навігаційних задач.

2. Розробка архітектури системи. Запропоновано модульну архітектуру системи, реалізовану на базі Docker-контейнерів. Кожен модуль виконує свою специфічну роль: обробка даних сенсорів, навігація, управління польотом, а їх взаємодія забезпечується через API та спільні файли. Такий підхід гарантує гнучкість, масштабованість та можливість легкої модифікації окремих частин системи.

3. Вибір технологій. У роботі використано сучасні інструменти: Kotlin 2.0 для реалізації основної логіки, Docker для контейнеризації, AirSim для симуляції та налагодження, а також бібліотеки kotlinx.serialization для роботи з даними та RPC API AirSim для інтеграції з симулятором.

4. Розробка модулів. Реалізовано ключові модулі системи:

- Модуль обробки сенсорних даних, який через RPC API забезпечує передачу даних з AirSim до інших частин системи.

- Модуль навігації, що виконує розрахунок маршрутів на основі отриманих даних.

- Модуль керування польотом, який взаємодіє з іншими модулями та забезпечує виконання польотних команд відповідно до стану дрона.

5. Тестування системи Проведено тестування системи в симуляторі AirSim. Перевірено коректність роботи модулів, взаємодії між ними та виконання основних завдань: отримання даних, обробка навігаційної інформації, виконання польотних команд. Умовні результати тестування підтвердили працездатність системи в рамках поставлених задач.

6. Практичне значення Розроблена система може бути використана як основа для створення реального автопілота з адаптацією до апаратної платформи дрона. Крім того, її модульна структура дозволяє легко інтегрувати нові компоненти або змінювати існуючі.

7. У ході виконання економічної частини магістерської кваліфікаційної роботи було досягнуто наступних результатів. Витрати на розробку системи автоматизованого керування склали 128 тисяч гривень, що відповідає встановленому технічному завданню (не більше 130 тисяч гривень). Абсолютний ефект від впровадження розробки за три роки, з урахуванням 10% інфляції, оцінюється у 2474 тисячі гривень, що перевищує встановлений мінімум у 2400 тисяч гривень. Внутрішня дохідність інвестицій склала 72%, що значно перевищує пороговий показник у 45%. Термін окупності інвестицій скоротився до 1,43 років, що також є кращим за встановлений максимум у три роки. Ці результати підтверджують економічну доцільність розробки системи.

8. Система автоматизованого керування успішно інтегрована в симуляційне середовище AirSim, забезпечуючи коректну роботу з мінімальною затримкою передачі даних між модулями — менше 100 мс. Точність виконання маршрутів становить не більше 1-2% відхилення. Усі модулі працюють стабільно у контейнерному середовищі Docker із використанням виділених портів для взаємодії. Система демонструє готовність до подальшої адаптації для реальних умов застосування.

Робота продемонструвала ефективність запропонованих підходів до розробки автоматизованої системи керування. Отримані результати є важливим внеском у розвиток технологій автоматизації безпілотних літальних апаратів і можуть слугувати базою для подальших досліджень та вдосконалень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Безпілотні літальні апарати: Методи керування та майбутні виклики. URL: https://ieeexplore.ieee.org/document/10077453 (дата звернення 11.11.2024).

2. Станіславенко М. М., Кулик Я. А. БАЗОВА МОДЕЛЬ БОРТОВОГО КОМП'ЮТЕРА ДЛЯ БЕЗПІЛОТНОГО ЛІТАЛЬНОГО АПАРАТУ : Молодь в науці: дослідження, проблеми, перспективи, м. Вінниця, 22-23 червня 2024 р. Вінниця, 2024.

3. ArduPilot: Відкрите програмне забезпечення автопілота. URL: https://ardupilot.org/ardupilot/ (дата звернення 11.11.2024).

4. PX4 Автопілот: Посібник користувача. URL: https://docs.px4.io/main/en/ (дата звернення 11.11.2024).

5. DJI Onboard SDK: Керування польотом. URL: https://developer.dji.com/onboard-sdk/documentation/guides/component-guide-flight-control.html (дата звернення 11.11.2024).

6. Intel Aero: Про платформу Intel Aero. URL: https://github.com/intel-aero/meta-intel-aero/wiki/01-About-Intel-Aero (дата звернення 11.11.2024).

7. Система робототехнічної операційної системи (ROS). URL: https://www.ros.org/blog/ecosystem/ (дата звернення 11.11.2024).

8. AirSim: Симулятор автономних систем. URL: https://microsoft.github.io/AirSim/ (дата звернення 11.11.2024).

9. Споживання чипів Apple Silicon на Mac Mini. URL: https://support.apple.com/en-gb/103253 (дата звернення 11.11.2024).

10. Процесор Apple M2: Бенчмарки та специфікації. URL: https://www.notebookcheck.net/Apple-M2-Processor-Benchmarks-and-Specs.632312.0.html (дата звернення 11.11.2024).

11. Metal і TensorFlow: Офіційна документація. URL: https://developer.apple.com/metal/tensorflow-plugin/ (дата звернення 11.11.2024).

12. Neural Engine: Огляд технології. URL: https://apple.fandom.com/wiki/Neural_Engine (дата звернення 11.11.2024).

13. Kotlin: Офіційна документація. URL: https://kotlinlang.org/docs/home.html (дата звернення 11.11.2024).

14. Kotlin Multiplatform. URL: https://www.jetbrains.com/kotlin-multiplatform/ (дата звернення 11.11.2024).

15. Docker: Посібник для початківців. URL: https://docs.docker.com/get-started/ (дата звернення 11.11.2024).

16. gRPC проти REST: Порівняння технологій. URL: https://aws.amazon.com/compare/the-difference-between-grpc-and-rest/?nc1=h_ls (дата звернення 11.11.2024).

17. Colosseum: Відкрита документація. URL: https://codexlabsllc.github.io/Colosseum/ (дата звернення 11.11.2024).

18. Docker: Документація зі збірки. URL: https://docs.docker.com/build/ (дата звернення 11.11.2024).

19. Docker Compose: Посібник користувача. URL: https://docs.docker.com/compose/ (дата звернення 11.11.2024).

20. Colosseum Simulator: Симулятор від Codex Labs. URL: https://codexlabsllc.github.io/Colosseum/ (дата звернення 11.11.2024).

21. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. / Укладачі В.О. Козловський, О.Й. Лесько, В.В.Кавецький. Вінниця : ВНТУ, 2021. 42 с.

ДОДАТКИ

Додаток А (обов'язковий)

Технічне завдання на магістерську кваліфікаційну роботу

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

ЗАТВЕРДЖУЮ

Завідувач кафедри АІІТ

_____ д.т.н., проф. Олег БІСІКАЛО

«____» _____ 2024 року

ТЕХНІЧНЕ ЗАВДАННЯ

на магістерську кваліфікаційну роботу

РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОГО КЕРУВАННЯ ДЛЯ БЕЗПІЛОТНОГО ЛІТАЛЬНОГО АПАРАТУ

08-31.МКР.008.02.000 ТЗ

Керівник: к.т.н., доц. каф. АІІТ

_____ Ярослав КУЛИК

«____» _____ 2024 р.

Розробив студент гр. 1АКІТР-23м

_____ Максим СТАНІСЛАВЕНКО

«____» _____ 2024 р

Вінниця 2024

1. Назва та галузь застосування

Розробка системи автоматизованого керування для безпілотного літального апарату. У даній роботі розглядається створення системи,

орієнтованої на автоматизоване управління безпілотним літальним апаратом (БПЛА), яка реалізована на основі модульної архітектури. Основна мета системи — забезпечити стабільне та ефективне керування БПЛА в умовах симуляційного середовища AirSim. Система орієнтована на інтеграцію модулів для аналізу даних з сенсорів, навігації та виконання команд керування польотом за допомогою технологій контейнеризації (Docker) та програмування на мові Kotlin.

2. Підстави для проведення робіт

Підставою для виконання роботи є наказ №__ по ВНТУ від «__» _____ 2024р., та індивідуальне завдання на МКР, затверджене протоколом №__ засідання кафедри АІІТ від «__» _____ 2024р.

3. Мета та призначення роботи

Метою роботи є розробка системи автоматизованого керування для безпілотного літального апарату (БПЛА), що ґрунтується на модульній архітектурі та використовує сучасні технології контейнеризації Docker і програмування на мові Kotlin. Система повинна забезпечити інтеграцію основних функцій керування БПЛА, таких як обробка сенсорних даних, навігація, виконання команд польоту та стабільне управління дроном в умовах симуляційного середовища AirSim. Призначення роботи полягає в створенні універсальної та ефективної архітектури для автоматизованого керування БПЛА, яка може бути використана як у віртуальних середовищах для тестування та вдосконалення алгоритмів, так і в реальних умовах для реалізації автономних функцій БПЛА в різних галузях застосування. Ця система може стати основою для подальших досліджень і розробок у сфері автоматизованого керування в галузі безпілотних технологій.

4. Джерела розробки

4.1 AirSim – Документація для використання AirSim в симуляціях дронів. URL: https://microsoft.github.io/AirSim/ (дата звернення 01.10.2024).

4.2 Kotlin Documentation – Офіційна документація Kotlin. URL: https://kotlinlang.org/docs/home.html (дата звернення 01.10.2024).

4.3 Docker Documentation – Офіційна документація Docker. URL: https://docs.docker.com/ (дата звернення 01.10.2024).

5. Показники призначення

5.1 Мінімальні вимоги до симулятора:

- ОС Windows/Linux
- Процесор Ryzen 5 5600;
- Відеокарта Nvidia RTX4070;
- Об’єм оперативної пам’яті: 32Gb;
- Середовище симуляції: Unreal Engine 5.

5.2 Бортовий комп'ютер:

- Mac Mini;
- Процесор M2;
- Об’єм оперативної пам’яті: 8Gb;

5.3 Середовище розробки та запуску intellij idea.

6. Економічні показники

До економічних показників входять:

- витрати на розробку – не більше 130 тис. грн;
- абсолютний економічний ефект від впровадження розробки – не менше 2400 тисяч грн;
- внутрішня дохідність потенційних інвестицій – не менше 45%;
- термін окупності потенційних інвестицій – до 3-ох років;

7. Стадії розробки

7.1 Розділ 1 «ЗАГАЛЬНІ ВІДОМОСТІ» має бути виконаний до 10.09.2024.

7.2 Розділ 2 «ВИБІР ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ БОРТОВОГО КОМП’ЮТЕРА» має бути виконаний до 23.09.2024.

7.3 Розділ 3 «РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗВ’ЯЗАННЯ ПОСТАВЛЕНИХ ЗАДАЧ» має бути виконаний до 05.11.2024.

7.4 Розділ 4 «ТЕСТУВАННЯ СИСТЕМ АВТОПІЛОТА» має бути виконаний до 28.11.2024.

7.5 Економічний розділ має бути виконаний до 15.12.2024.

8. Порядок контролю та приймання

8.1 Попередній захист магістерської кваліфікаційної роботи провести до 04.12.2024.

8.2 Захист магістерської кваліфікаційної роботи провести до 19.12.2024.

Розробив студент групи 1АКІТР-23м _____ Максим СТАНІСЛАВЕНКО

Додаток Б - Ілюстративна частина

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

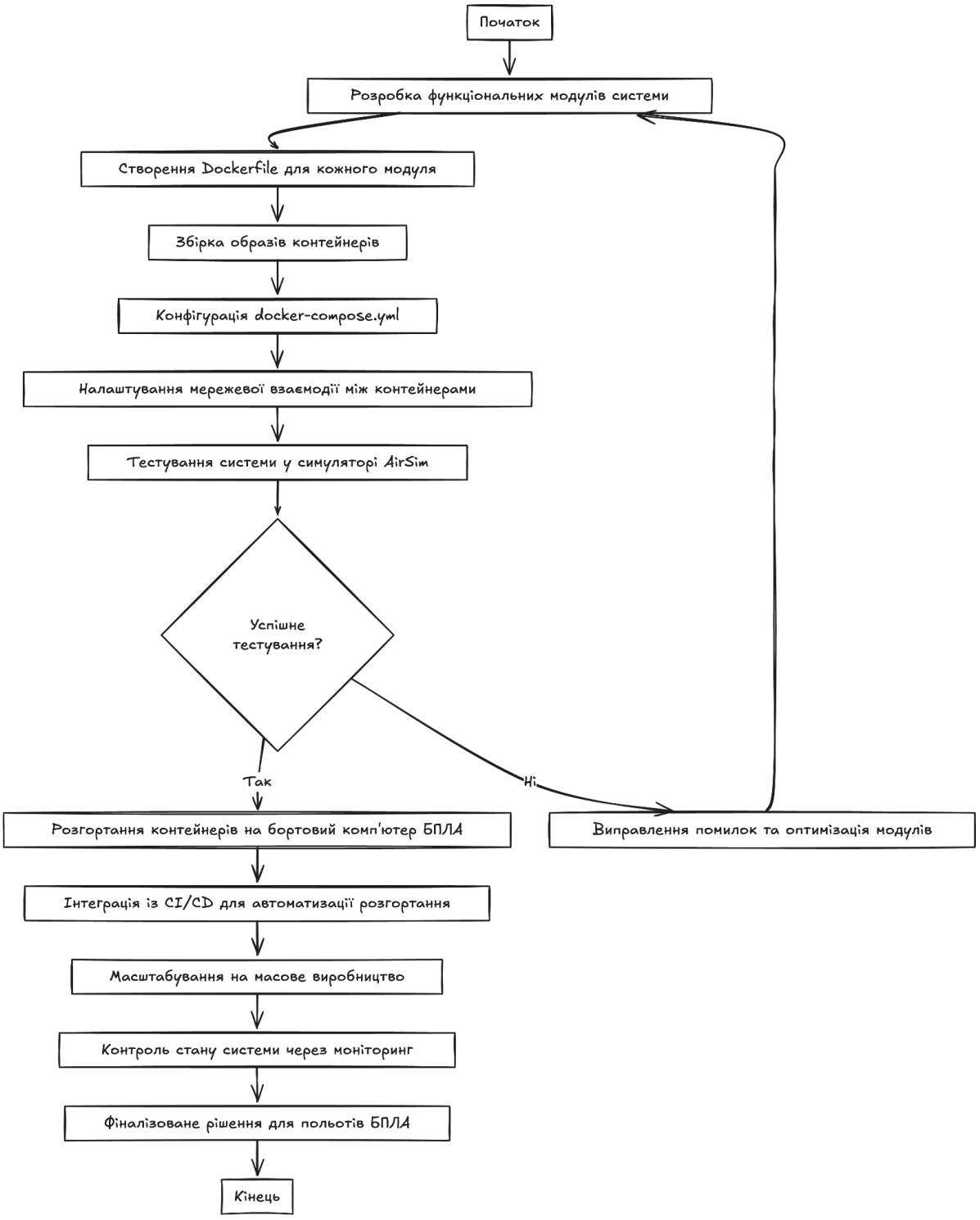

Рисунок Б.1 – Схема розробки функціональних модулів системи

Рисунок Б.2 – Титулка презентації

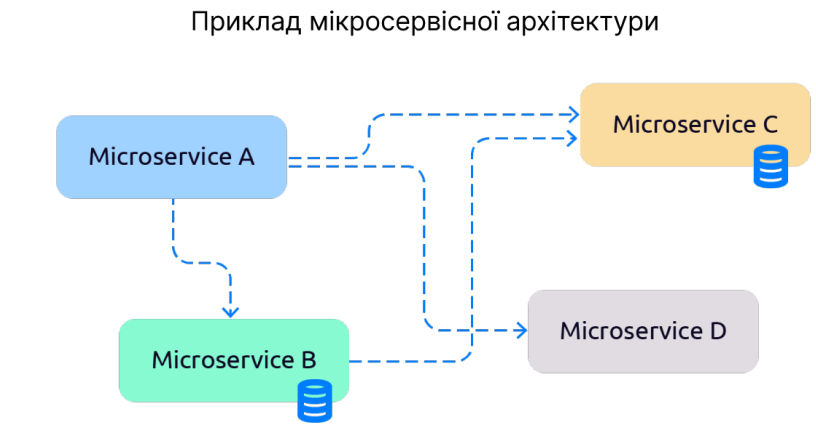

Рисунок Б.3 – Приклад мікросервісної архітектури

Зв’язок між компонентами БПЛА і бортовим комп’ютером

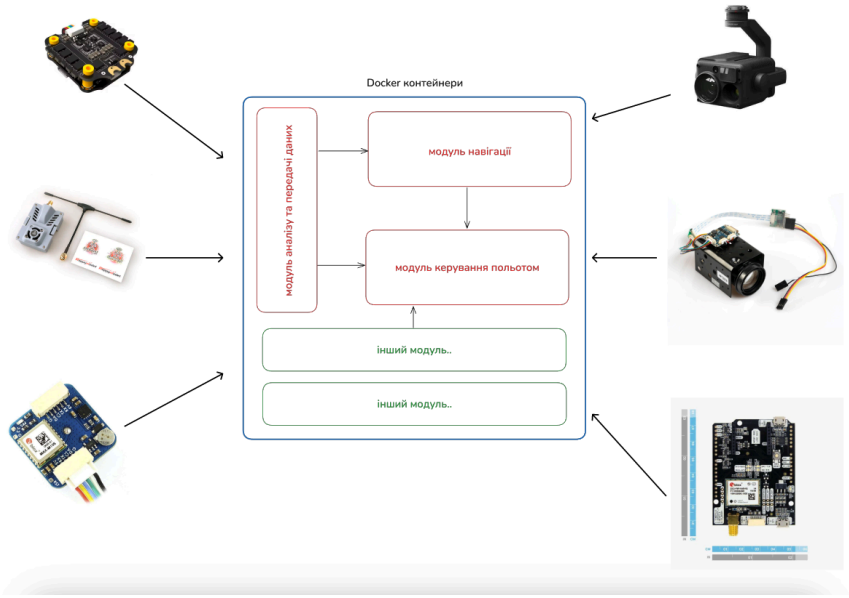

Рисунок Б.4 – Зв'язок між компонентами БПЛА і бортовим комп'ютером

Відображення контейнерів запущених на бортовому комп’ютері

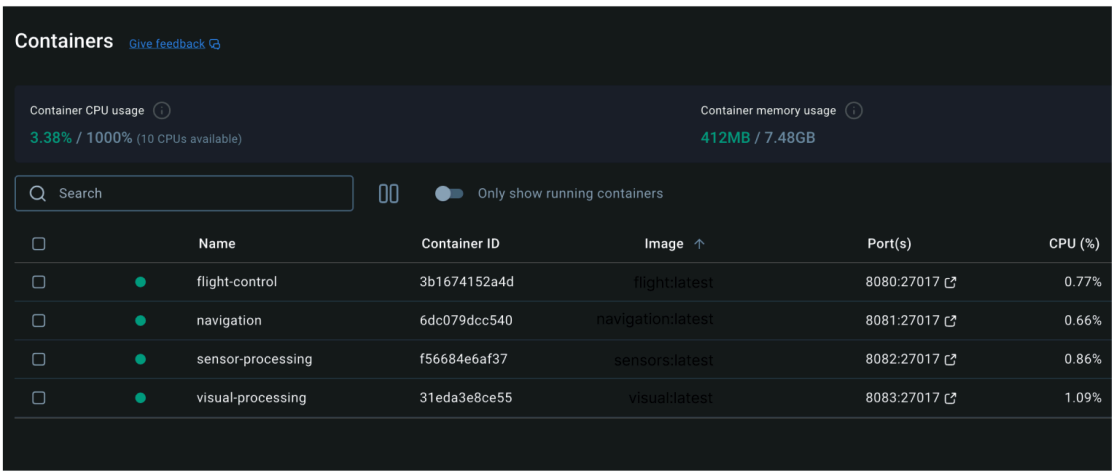

Рисунок Б.5 – Відображення контейнерів бортового комп’ютера

Apple Mac Mini в розібраному стані

Рисунок Б.6 – Apple Mac Mini в розібраному стані

Розміри бортового комп'ютера Apple Mac Mini M4

Рисунок Б.7 – Розміри бортового компютера Apple Mac Mini M4

Додаток В (обов’язковий)
Лістинг Docker файлів налаштувань

Вміст Dockerfile

```
# Використання офіційного образу OpenJDK як базового
FROM openjdk:17-jdk-slim

# Встановлення робочої директорії в контейнері
WORKDIR /app

# Копіювання створеного JAR-файлу в контейнер
COPY build/libs/flightcontrol.jar app.jar

# Встановлення порту для контейнера
EXPOSE 8080

# Команда для запуску програми
CMD ["java", "-jar", "app.jar"]
```

Вміст docker-compose.yml

```
version: '3.9'

services:
  flight-control:
    build:
      context: ./flight-control
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    networks:
      - drone-system
    environment:
```

```
    - MODULE_NAME=FlightControl
    - LOG_LEVEL=info

navigation:
  build:
    context: ./navigation
    dockerfile: Dockerfile
  ports:
    - "8081:8081"
  networks:
    - drone-system
  environment:
    - MODULE_NAME=Navigation
    - LOG_LEVEL=info

sensor-processing:
  build:
    context: ./sensor-processing
    dockerfile: Dockerfile
  ports:
    - "8082:8082"
  networks:
    - drone-system
  environment:
    - MODULE_NAME=SensorProcessing
    - AIRSIM_HOST=airsim_simulator
    - LOG_LEVEL=info

visual-processing:
  build:
    context: ./visual-processing
    dockerfile: Dockerfile
  ports:
    - "8083:8083"
```

```
    networks:
      - drone-system
    environment:
      - MODULE_NAME=VisualProcessing
      - LOG_LEVEL=info

networks:
  drone-system:
    driver: bridge
```

Додаток Г (обов’язковий).
Лістинг архітектури модулів і виконавчих частин

Сутності

```
data class FlightCommand(val action: String, val parameters: Map<String, Any>)
```

Use Cases

```
class ExecuteFlightCommandUseCase(private val rpcClient: RpcClient) {
    suspend fun execute(command: FlightCommand): Boolean {
        return rpcClient.sendCommand(command.action, command.parameters)
    }
}
```

Adapters

```
class FlightController(private val executeCommandUseCase:
ExecuteFlightCommandUseCase) {
    fun handleRequest(request: HttpRequest): HttpResponse {
        val command = parseRequest(request)
        return HttpResponse(executeCommandUseCase.execute(command))
    }
}
```

Налаштування з'єднання з AirSim через RPC API

```
import kotlinx.coroutines.*
import org.msgpack.rpc.Client
import org.msgpack.rpc.loop.EventLoop
import org.msgpack.type.Value

class AirSimRpcClient(private val host: String, private val port: Int) {
```

```
    private val eventLoop = EventLoop.defaultEventLoop()
    private val client = Client(eventLoop)

    init {
        client.connect(host, port)
    }

    suspend fun getSensorData(sensorName: String): Value {
        return withContext(Dispatchers.IO) {
            client.call("getSensorData", sensorName)
        }
    }

    suspend fun sendControlCommand(command: Map<String, Any>) {
        withContext(Dispatchers.IO) {
            client.call("control", command)
        }
    }

    fun close() {
        client.close()
        eventLoop.shutdown()
    }
}
```

Отримання даних із сенсорів

```
data class GpsData(val latitude: Double, val longitude: Double, val altitude: Double)
data class ImuData(val pitch: Double, val roll: Double, val yaw: Double)
data class LidarData(val points: List<Point3D>)
```

```
data class ImageData(val pixels: ByteArray)

class SensorProcessor(private val airSimRpcClient: AirSimRpcClient) {

    suspend fun getGpsData(): GpsData {
        val result = airSimRpcClient.getSensorData("gps") // запит до AirSim
        return GpsData(
            latitude = result["latitude"].asFloatValue().toDouble(),
            longitude = result["longitude"].asFloatValue().toDouble(),
            altitude = result["altitude"].asFloatValue().toDouble()
        )
    }

    suspend fun getImuData(): ImuData {
        val result = airSimRpcClient.getSensorData("imu")
        return ImuData(
            pitch = result["pitch"].asFloatValue().toDouble(),
            roll = result["roll"].asFloatValue().toDouble(),
            yaw = result["yaw"].asFloatValue().toDouble()
        )
    }

    suspend fun getLidarData(): LidarData {
        val result = airSimRpcClient.getSensorData("lidar")
        val points = result["points"].asArrayValue().map {
            Point3D(it[0].asFloatValue().toDouble(), it[1].asFloatValue().toDouble(),
it[2].asFloatValue().toDouble())
        }
        return LidarData(points)
```

```
    }

    suspend fun getCameraImage(cameraId: String): ImageData {
        val result = airSimRpcClient.getSensorData(cameraId)
        return ImageData(pixels = result["image"].asRawValue().byteArray)
    }
}
```

Передача команд керування

```
data class FlightControlCommand(
    val throttle: Float,
    val roll: Float,
    val pitch: Float,
    val yaw: Float
)

class FlightControl(private val airSimRpcClient: AirSimRpcClient) {

    suspend fun sendFlightCommand(command: FlightControlCommand) {
        val commandMap = mapOf(
            "throttle" to command.throttle,
            "roll" to command.roll,
            "pitch" to command.pitch,
            "yaw" to command.yaw
        )
        airSimRpcClient.sendControlCommand(commandMap)
    }
}
```

Асинхронна робота модуля

```
fun main() = runBlocking {
    val airSimClient = AirSimRpcClient("127.0.0.1", 41451)
    val sensorProcessor = SensorProcessor(airSimClient)
    val flightControl = FlightControl(airSimClient)

    launch {
        while (true) {
            val gpsData = sensorProcessor.getGpsData()
            println("GPS: $gpsData")

            val imuData = sensorProcessor.getImuData()
            println("IMU: $imuData")

            delay(1000)
        }
    }

    launch {
        flightControl.sendFlightCommand(FlightControlCommand(throttle = 0.5f, roll =
0f, pitch = 0f, yaw = 0f))
    }
}
```

Отримання поточних координат дрона

```
data class Coordinates(val latitude: Double, val longitude: Double, val altitude:
Double)

class NavigationModule(private val airSimConnector: AirSimConnector) {
```

```
    suspend fun getCurrentCoordinates(): Coordinates {
        val gpsData = airSimConnector.getGPSData()
        return Coordinates(
            latitude = gpsData.latitude,
            longitude = gpsData.longitude,
            altitude = gpsData.altitude
        )
    }
}
```

Розрахунок маршруту

```
fun calculateRoute(current: Coordinates, target: Coordinates): List<Coordinates> {
    val route = mutableListOf<Coordinates>()
    val steps = 10 // Розбиваємо маршрут на 10 сегментів
    val stepLat = (target.latitude - current.latitude) / steps
    val stepLon = (target.longitude - current.longitude) / steps
    val stepAlt = (target.altitude - current.altitude) / steps

    for (i in 1..steps) {
        route.add(
            Coordinates(
                latitude = current.latitude + stepLat * i,
                longitude = current.longitude + stepLon * i,
                altitude = current.altitude + stepAlt * i
            )
        )
    }
```

```
    return route
}
```

Додаток Д (обов'язковий).

Протокол перевірки

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка системи автоматизованого керування для безпілотного літального апарату

Тип роботи: магістерська кваліфікаційна робота.

Підрозділ: кафедра автоматизації та інтелектуальних інформаційних технологій, факультет інтелектуальних інформаційних технологій та автоматизації

Показники звіту подібності Turnitin

Оригінальність. 98.0% Схожість. 2.0%

Аналіз звіту подібності (відмітити потрібне):

■ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень

Особа, відповідальна за перевірку _____ Роман МАСЛІЙ

Ознайомлені з повним звітом подібності, який був згенерований системою Turnitin щодо роботи.

Автор роботи _____ Максим СТАНІСЛАВЕНКО
(підпис)

Керівник роботи _____ Ярослав КУЛИК
(підпис)