НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Автоматизованих систем обробки інформації і управління
(повна назва кафедри)

«На правах рукопису»
УДК 004.424

До захисту допущено:
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Інженерія програмного забезпечення комп'ютеризованих систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «МЕТОД ПОБУДОВИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ БЕЗПІЛОТНОГО ВАНТАЖНОГО ЛІТАЛЬНОГО АПАРАТУ»

Виконав:
студент VI курсу, групи ІП-91мн
Волков Ілля Андрійович

Науковий керівник:
д.т.н, професор кафедри АСОІУ
Сидоров Микола Олександрович

Рецензент:
Посада, науковий ступінь, вчене звання,
Прізвище, ім'я, по батькові

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Автоматизованих систем обробки інформації і управління

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-наукова програма - «Інженерія програмного забезпечення комп'ютеризованих систем»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Волкову Іллі Андрійовичу

1. Тема дисертації «МЕТОД ПОБУДОВИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ БЕЗПІЛОТНОГО ВАНТАЖНОГО ЛІТАЛЬНОГО АПАРАТУ», науковий керівник дисертації Сидоров Микола Олександрович, д.т.н, професор кафедри АСОІУ, затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом дисертації *20 травня 2021р.*

3. Об'єкт дослідження *Архітектура програмного забезпечення екосистеми вантажного БПЛА*

4. Перелік завдань, які потрібно розробити *дослідження обраної предметної області, визначення проблематики, аналіз існуючих методів розв'язання задачі, розробка власного підходу до вирішення задачі, моделювання та конструювання програмного забезпечення*

5. Орієнтовний перелік графічного (ілюстративного) матеріалу *схема архітектури ПЗ, схема баз даних, креслення графічного інтерфейсу, схема варіантів використання, схема бізнес-процесу*

6. Орієнтовний перелік публікацій *наукова стаття, тези*

7. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Графічний	доц. Ліщук К.І.		

8. Дата видачі завдання *11 березня 2021р.*

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	*Вивчення рекомендованої літератури*	*20.03.2021*	
2	*Аналіз існуючих методів розв'язання задачі*	*30.03.2021*	
3	*Постановка та формалізація задачі*	*05.04.2021*	
4	*Аналіз вимог до програмного забезпечення*	*05.04.2021*	
5	*Алгоритмізація задачі*	*10.04.2021*	
6	*Моделювання програмного забезпечення*	*12.04.2021*	
7	*Обґрунтування використовуваних технічних засобів*	*15.04.2021*	
8	*Розробка архітектури програмного забезпечення*	*20.04.2021*	
9	*Подання дисертації на попередній захист*	*23.04.2021*	
10	*Розробка програмного забезпечення*	*05.05.2021*	
11	*Налагодження програми*	*10.05.2021*	
12	*Виконання графічних документів*	*15.05.2021*	
13	*Оформлення пояснювальної записки*	*15.05.2021*	
14	*Подання дисертації рецензенту*	*16.05.2021*	
15	*Подання дисертації на основний захист*	*20.05.2021*	

Студент Ілля ВОЛКОВ

Науковий керівник Микола СИДОРОВ

РЕФЕРАТ

Дана дисертація розглядає один з можливих способів автоматизації процесу кур'єрської доставки - доставку посилок кінцевому отримувачу за допомогою безпілотних апаратів, які б могли виконувати такі завдання у дуже короткий час, незалежно від завантаженості доріг та систем громадського транспорту. Така система дозволяє суттєво знизити час доставки вантажу кінцевому користувачу, зменшити обсяг інфраструктури, необхідної для підтримки її працездатності та скоротити кількість обслуговуючого персоналу. Також повна автоматизація процесу доставки дозволяє знизити вплив людського фактору на якість обслуговування. На даний момент вже реалізовано кілька таких систем, проте жодна з них не є повністю безпечною та не відповідає всім вимогам до системи автоматичної кур'єрської доставки. Також жодна з цих розробок не має архітектури, що повністю покриває функціонал екосистеми безпілотного літального апарату. Також в даній роботі була досліджена низка готових архітектурних рішень, призначених для побудови аналогічного програмного забезпечення, взятих зі схожих наукових досліджень. Проте жодна з них також не відповідає всім поставленим вимогам до даного програмного забезпечення, або має суттєві недоліки, що перешкоджає її програмній реалізації, впровадженню або застосуванню на практиці. У зв'язку з усіма вище переліченими факторами було прийняте рішення про розробку власної архітектури для реалізації програмного комплексу екосистеми вантажного безпілотного літального апарату. А для підтвердження працездатності цієї архітектури і доведення її ефективності було розроблене відповідне програмне забезпечення із застосуванням запропонованого методу розробки.

Метою даного наукового дослідження є розробка методу побудови програмного забезпечення екосистеми вантажного безпілотного літального апарату, такого, що покращить та удосконалить існуючі підходи до

програмування БПЛА як з точки зору процесу їх впровадження, так і з точки зору використання кінцевого продукту.

Основні задачі, які були виконані під час проведення даного дослідження:

- вивчення і аналіз готових впроваджених програмних продуктів-аналогів з метою виявлення їх основних переваг та недоліків;
- вивчення і аналіз аналогічних наукових досліджень з метою дослідження шляхів вирішення основних задач побудови програмного забезпечення БПЛА;
- створення власного методу побудови програмного забезпечення вантажного БПЛА, враховуючи результати попереднього дослідження предметної області;
- написання програмного забезпечення на основі даного методу, аналіз його основних переваг та недоліків, та доведення його ефективності.

Об’єктом даного наукового дослідження є архітектура програмного забезпечення екосистеми вантажного БПЛА та підходи до реалізації даної архітектури.

Предметом дослідження є методи та способи побудови програмного забезпечення екосистеми вантажного БПЛА.

Під час проведення даного дослідження був використаний метод systematic mapping study (систематичний огляд літератури) для вивчення і аналізу предметної області даного дослідження з текстових джерел інформації та метод case study (метод ситуативного аналізу) для аналізу розробленого методу побудови програмного забезпечення.

Наукова новизна отриманого методу побудови програмного забезпечення полягає у тому, що в ньому вперше БПЛА розглядається як актор екосистеми безпілотних літальних апаратів і вперше для цієї екосистеми була розроблена архітектура програмного забезпечення. Також вперше було введене саме поняття екосистеми безпілотних літальних апаратів.

Практичне значення отриманих результатів полягає у тому, що був розроблений простий, ефективний та комплексний підхід до вирішення задачі з побудови програмного забезпечення вантажного БПЛА, який досить легко може бути застосований для вирішення комерційних задач із адресної доставки малогабаритних вантажів.

Результати цього дослідження були представлені на VI Всеукраїнській науково-практичній конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2021).

Дана дисертація складається з реферату, вступу, основної частини що поділяється на 4 розділи, висновків та додатків що включають в себе лістинг програмного коду та графічні матеріали. Основна частина даної роботи містить 105 сторінок, 28 рисунків, 18 таблиць та 19 посилань.

БПЛА, ЕКОСИСТЕМА БПЛА, ДОСТАВКА ВАНТАЖУ, МЕТОД ПОБУДОВИ ПЗ, АВТОНОМНІ БЕЗПІЛОТНИКИ.

РЕФЕРАТ

Данная диссертация рассматривает один из возможных способов автоматизации процесса курьерской доставки - доставку посылок конечному получателю с помощью беспилотных аппаратов, которые могли бы выполнять такие задачи в очень короткое время, независимо от загруженности дорог и систем общественного транспорта. Такая система позволяет существенно снизить время доставки груза конечному пользователю, уменьшить объем инфраструктуры, необходимой для поддержания ее работоспособности и сократить количество обслуживающего персонала. Также полная автоматизация процесса доставки позволяет снизить влияние человеческого фактора на качество обслуживания. На данный момент уже реализовано несколько таких систем, однако ни одна из них не является полностью безопасной и не отвечает всем требованиям к системе автоматической курьерской доставки. Также ни одна из этих разработок не имеет архитектуры, полностью покрывает функционал экосистемы беспилотного летательного аппарата. Также в данной работе была исследована ряд готовых архитектурных решений, предназначенных для построения аналогичного программного обеспечения, взятых из похожих научных исследований. Однако ни одна из них также не соответствует всем предъявляемым требованиям к данному программному обеспечению, или имеет существенные недостатки, препятствует ее программной реализации, внедрению или применению на практике. В связи со всеми вышеперечисленными факторами было принято решение о разработке собственной архитектуры для реализации программного комплекса экосистемы грузового беспилотного летательного аппарата. А для подтверждения работоспособности этой архитектуры и доведение ее эффективности было разработано соответствующее программное обеспечение с применением предложенного метода разработки.

Целью данного научного исследования является разработка метода построения программного обеспечения экосистемы грузового беспилотного летательного аппарата, такого, что улучшит и усовершенствует существующие подходы к программированию БПЛА как с точки зрения процесса их внедрения, так и с точки зрения использования конечного продукта.

Основные задачи, которые были выполнены при проведении данного исследования:

- изучение и анализ готовых внедренных программных продуктов-аналогов с целью выявления их основных преимуществ и недостатков;
- изучение и анализ аналогичных научных исследований с целью исследования путей решения основных задач построения программного обеспечения БПЛА;
- создание собственного метода построения программного обеспечения грузового БПЛА, учитывая результаты предыдущего исследования предметной области;
- написание программного обеспечения на основе данного метода, анализ его основных преимуществ и недостатков, и доведение его эффективности.

Объектом данного научного исследования является архитектура программного обеспечения экосистемы грузового БПЛА и подходы к реализации данной архитектуры.

Предметом исследования являются методы и способы построения программного обеспечения экосистемы грузового БПЛА.

При проведении данного исследования был использован метод systematic mapping study (систематический обзор литературы) для изучения и анализа предметной области данного исследования из текстовых источников информации и метод case study (метод ситуационного анализа) для анализа разработанного метода построения программного обеспечения.

Научная новизна полученного метода построения программного обеспечения заключается в том, что в нем впервые БПЛА рассматривается как актер экосистемы беспилотных летательных аппаратов и впервые для этой экосистемы была разработана архитектура программного обеспечения. Также впервые было введено само понятие экосистемы беспилотных летательных аппаратов.

Практическое значение полученных результатов заключается в том, что был разработан простой, эффективный и комплексный подход к решению задачи по построению программного обеспечения грузового БПЛА, который достаточно легко может быть применен для решения коммерческих задач с адресной доставки малогабаритных грузов.

Результаты этого исследования были представлены на VI Международной научно-практической конференции молодых ученых и студентов «Информационные системы и технологии управления» (ИСТУ-2021).

Данная диссертация состоит из реферата, введения, основной части которая делится на 4 раздела, заключения и приложений, включающих в себя листинг программного кода и графические материалы. Основная часть данной работы содержит 105 страниц, 28 рисунков, 18 таблиц и 19 ссылок.

БПЛА, ЭКОСИСТЕМА БПЛА, ДОСТАВКА ГРУЗОВ, МЕТОД ПОСТРОЕНИЯ ПО, АВТОНОМНЫЕ БЕСПИЛОТНИКИ.

ABSTRACT

This dissertation considers one of the possible ways to automate the courier delivery process - delivery of parcels to the final recipient using unmanned aerial vehicles, which could perform such tasks in a very short time, regardless of the congestion of roads and public transport systems. This system can significantly reduce the time of delivery of goods to the end user, reduce the amount of infrastructure needed to maintain its efficiency and reduce the number of service personnel. Also, full automation of the delivery process reduces the impact of the human factor on the quality of service. Currently, several such systems have been implemented, but none of them is completely secure and does not meet all the requirements for an automatic courier system. Also, none of these developments has an architecture that fully covers the functionality of the unmanned aerial vehicle ecosystem. Also in this work, a number of ready-made architectural solutions designed to build similar software, taken from similar research. However, none of them also meets all the requirements for this software, or has significant shortcomings that prevent its software implementation, implementation or application in practice. In connection with all the above factors, it was decided to develop its own architecture for the implementation of the software package of the ecosystem of cargo unmanned aerial vehicles. And to confirm the efficiency of this architecture and prove its effectiveness, appropriate software was developed using the proposed method of development.

The purpose of this research is to develop a method for building software ecosystems of unmanned aerial vehicles, one that will improve and enhance existing approaches to UAV programming both in terms of the process of their implementation and in terms of use of the final product.

The main tasks that were performed during this study:

- study and analysis of ready-implemented software products-analogues in order to identify their main advantages and disadvantages;

- study and analysis of similar research to explore ways to solve the main problems of building UAV software;
- creation of own method of construction of the software of the cargo UAV, taking into account results of preliminary research of subject area;
- writing software based on this method, analyzing its main advantages and disadvantages, and proving its effectiveness.

The object of this research is the software architecture of the cargo UAV ecosystem and approaches to the implementation of this architecture.

The subject of the research is the methods and ways of building the software of the cargo UAV ecosystem.

During this study, the method of systematic mapping study was used to study and analyze the subject area of this study from textual sources of information and the method of case study to analyze the developed method of software construction.

The scientific novelty of the obtained method of software construction is that for the first time the UAV is considered as an actor in the ecosystem of unmanned aerial vehicles and for the first time a software architecture was developed for this ecosystem. Also, for the first time, the very concept of the unmanned aerial vehicle ecosystem was introduced.

The practical significance of the obtained results is that a simple, effective and comprehensive approach for solving the problem of building a UAV software was developed, which can easily be used to solve commercial problems of targeted delivery of small cargo.

The results of this study were presented at the VI All-Ukrainian scientific-practical conference of young scientists and students "Information Systems and Control Technologies" (ISCT-2021).

This dissertation consists of an abstract, introduction, main part divided into 4 sections, conclusions and appendices that include a list of program code and graphics. The main part of this work contains 105 pages, 28 figures, 18 tables and 19 references.

UAV, UAV ECOSYSTEM, CARGO DELIVERY, SOFTWARE CONSTRUCTION METHOD, AUTONOMOUS DRONES.

ЗМІСТ

РЕФЕРАТ **4**

РЕФЕРАТ **7**

ABSTRACT **10**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ **15**

ВСТУП **16**

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. АНАЛІЗ ВИМОГ ДО МЕТОДУ ПОБУДОВИ ПЗ БПЛА **21**

1.1 Загальні положення 21

1.2 Огляд літератури з використанням методу Systematic Mapping Study 22

1.3 Огляд літератури 31

1.4 Порівняльний аналіз існуючих наукових досліджень ПЗ БПЛА 35

1.5 Аналіз вимог до методу побудови ПЗ БПЛА 41

1.6 Висновки до розділу 47

2 МЕТОД ПОБУДОВИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЕКОСИСТЕМИ БПЛА **48**

2.1 Моделювання та аналіз програмного забезпечення 48

2.2 Архітектура програмного забезпечення 51

2.3 Підхід до реалізації модулів програмного забезпечення 54

2.4 Формулювання методу побудови програмного забезпечення 63

2.5 Висновки до розділу 67

3 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЕКОСИСТЕМИ БПЛА **69**

3.1 Налагодження модуля літального апарату 69
3.2 Налагодження модуля автопілоту 72
3.3 Тестування платформи літального апарату 73
3.4 Конструювання модуля керування БПЛА 74
3.5 Тестування модуля керування БПЛА 79
3.6 Конструювання модулю планування задач БПЛА 80
3.7 Конструювання модуля обслуговування клієнтів 82
3.8 Налагодження і тестування програмного комплексу 84
3.9 Висновки до розділу 85

4 CASE STUDY 86

4.1 Кейс 1: Служба кур'єрської доставки 86
4.2 Кейс 2: Компанія по боротьбі зі шкідниками 89
4.3 Використання інших методів побудови ПЗ БПЛА 90
4.4 Висновки до розділу 91

ВИСНОВКИ 92

ПЕРЕЛІК ПОСИЛАНЬ 93

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ 95

ДОДАТОК Б БІЗНЕС-ПРОЦЕС ДОСТАВКИ ВАНТАЖУ АКТОРОМ БЛПА 103

ДОДАТОК В МЕТОД ПОБУДОВИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 104

ДОДАТОК Г АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 105

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Безпілотний літальний апарат (БПЛА) – це такий літальний апарат що здатен злітати, здійснювати політ і приземлятись без пілота, присутнього на борту.

Екосистема безпілотного літального апарату - це апаратно-програмний комплекс, що дозволяє ставити задачі для автоматичного виконання одному або системі БПЛА і відслідковувати їх виконання.

ВСТУП

На сьогоднішній день важливу роль в сфері логістики відіграють поштові сервіси. Вони обробляють і доставляють щодня десятки тисяч листів і посилок. І ніколи в історії людства даний процес не був настільки швидким і дешевим. Проте, у вік цифрових технологій та тотальної автоматизації процесів, доставку кінцевому адресату все ще виконують звичайні листоноші, як і сотні років тому, які тепер називаються кур'єрами. Одним з можливих способів автоматизації даного процесу є доставка посилок кінцевому отримувачу за допомогою безпілотних апаратів, які б могли виконувати такі завдання у великих містах в дуже короткий час, незалежно від завантаженості доріг та систем громадського транспорту. На даний момент подібні сервіси доставки є тільки у великих містах США і носять більше експериментальний характер. Єдиною компанією яка на даний момент досягла значних успіхів у цій сфері є Amazon зі своїм сервісом PrimeAir. Компанія заявляє, що вона може доставити будь-який вантаж вагою до 2.5 кг у будь-яку точку міста, де працює сервіс, менш ніж за 30 хвилин. Для порівняння: у м. Києві середній час кур'єрської доставки наразі складає від 2-х годин до повного робочого дня (за даними з різних сайтів кур'єрських служб). І та оперативність, з якою ці самі служби доставляють посилки досягається лише за допомогою побудови розвиненої мережі складів. Економія часу, що досягається за допомогою сервісу кур'єрської доставки за допомогою БПЛА, дуже значна. До того ж, для її обслуговування потрібно набагато менше складів – місць базування безпілотників, які можуть бути розташовані в приміських районах, оскільки для них не має значення, як швидко можна дістатись клієнта за допомогою звичних засобів пересування (автомобілів, громадського транспорту тощо). Додатковим бонусом є те, що такий сервіс доставки не потребує великої кількості персоналу. Основна кількість обслуговуючого персоналу в ньому задіяна саме для сервісу і ремонту безпілотників, а не для стояння в заторах на шляху до адресата

доставки. Проте технології компанії Amazon залишаються закритими і недоступними. В той же час поки не було розроблене повністю автономне програмне забезпечення, яке б реалізувало функції системи доставки. Це стало мотивацією для пошуку готових методів реалізації даного програмного забезпечення або програмне забезпечення з відкритим вихідним кодом, яке можна було б пристосувати для реалізації подібних проектів.

То що ж очікується від такого програмного забезпечення?

Узагальнено, процес кур'єрської доставки складається з декількох етапів: зі створення клієнтом замовлення, забирання посилки кур'єром, доставки посилки у місце призначення і передачі її клієнту. Даний процес нічим не відрізняється і для безпілотників – кур'єрів, за винятком способів реалізації кожного з етапів доставки. Розглянемо кожну задачу окремо.

Для вирішення задачі із обслуговування клієнтів необхідний окремий застосунок, який повинен зберігати дані про користувача та його посилку, відображати статус доставки та надавати можливість для її оплати.

Для вирішення задач з доставки знадобиться програма обслуговування БПЛА, що надсилає команди для безпілотного літального апарату. У спрощеному варіанті це можуть бути як просто координати початкової точки, де безпілотник має підхопити посилку, і кінцевої, куди він має цю посилку доставити. У більш складному це може бути повноцінний маршрут із GPS-координат, складений з урахуванням всіх перешкод і обмежень на шляху (високі будівлі, лінії електропередач тощо). Дана програма керування безпілотниками може бути як самостійним компонентом ПЗ, так і інтегрованою в клієнт-серверний застосунок, зазначений вище.

В свою чергу, БПЛА має бути оснащений системою автопілоту. В загальному випадку, автопілот – це програма або пристрій, що веде транспортний засіб по певній, заданій йому траєкторії. У випадку з БПЛА ця програма має не лише вести літальний апарат за заданим маршрутом за рахунок керування аеродинамічними елементами (гвинти, рулі, елерони тощо), а й яка повинна, зчитуючи показники датчиків, з високою точністю

відстежувати своє місцезнаходження та орієнтацію в просторі, автоматично стабілізувати літальний апарат в процесі польоту та повертати його на базу у випадку виникнення нештатних ситуацій.

Останнім компонентом програмного забезпечення є програма керування БПЛА, встановлена на літальному апараті. Саме вона відповідальна за комунікацію із наземним програмним комплексом та надсилання керуючих команд безпосередньо до програми-автопілоту. Також вона може вирішувати низку додаткових задач, які не входять до функціоналу автопілоту, але без яких реалізація кур'єрської доставки безпілотним літальним апаратом є неможлива. Серед них – уникнення перешкод в процесі польоту, точна посадка, підйом вантажу БПЛА та їм подібні.

В наступних розділах в даній науковій роботі будуть детально розглянуті й проаналізовані декілька найбільш значущих робіт інших науковців, які вивчали й проектували аналогічне програмне забезпечення, виділені їх переваги та недоліки, та буде проведений аналіз вдалих підходів до реалізації програмного забезпечення вантажного безпілотного літального апарату.

Якщо коротко підсумувати результати проведеного дослідження, то наразі існують надійні системи автоматичного пілотування безпілотників, проте вони вимагають втручання оператора для виконання кожного нового польотного завдання та не вміють у своїй більшості взаємодіяти з додатковим корисним навантаженням у вигляді різноманітних сенсорів, датчиків, камер тощо.

Для прикладу, автори архітектури під назвою MULTIDRONE описують метод побудови та архітектури ПЗ для задачі автоматичної повітряної зйомки за допомогою БПЛА. Вони спроміглися зробити її досить гнучкою та розширюваною, та отримали дуже високий ступінь контролю за літальними апаратами. Ще одним плюсом є можливість керування безпілотниками через LTE та WIFI, що ліквідує обмеженість радіусу їх дії від розташування

наземної станції керування. Проте її мінуси також суттєві: висока складність реалізації архітектури, велика кількість операторів необхідних для роботи та надмірна кількість відповідальності функціоналу наземної станції роблять її не найкращим вибором.

В іншому підході автори розробили дуже надійну і продуману архітектуру. Проте вона ще складніша в реалізації за попередню, та реалізує лише базовий функціонал для планування польотних місій. Для реалізації сервісу доставки безпілотниками його недостатньо.

Ще в одному підході до побудови програмного забезпечення для БПЛА автори дуже вдало поєднали простоту реалізації із легким масштабуванням. Проте використання LAN для зв'язку з безпілотниками робить її обмеженою в радіусі дії, а надмірна відповідальність наземної станції є поганим архітектурним підходом. Останнім недоліком є відсутність зручного способу взаємодії із кінцевим користувачем.

Звісно, не існує певно такого випадку, коли б можна було б просто взяти чиєсь готове рішення та без змін застосувати його до іншої, відмінної від початкової, прикладної задачі. Проте переробка вищезазначених підходів до написання програмного забезпечення вантажного безпілотника була б надзвичайно витратною як за часовими ресурсами розробників, так і за грошовими, адже за час роботи розробників платитиме замовник, який і буде займатись організацією процесу реалізації й впровадження даного програмного комплексу.

Виходячи з того, що даний програмний комплекс має потенційно велику користь для сервісів кур'єрської доставки після його впровадження та для їх клієнтів, та з того, що жоден з розглянутих в даній роботі продуктів та проектів – аналогів не задовольняє усім вимогам до програмного забезпечення такого роду, було прийнято рішення про початок наукового дослідження існуючих методів побудови програмного забезпечення безпілотних літальних апаратів.

За мету в даному науковому дослідженні була поставлена розробка методу побудови програмного забезпечення екосистеми вантажного безпілотного літального апарату, такого, що покращить та удосконалить існуючі підходи до програмування БПЛА як з точки зору процесу їх впровадження, так і з точки зору використання кінцевого продукту.

Серед основних задач, що стоять перед даною роботою, слід виділити дослідження предметної області методів розробки програмного забезпечення вантажних БПЛА, пошук переваг та недоліків в існуючих підходах, створення власного підходу що має зберегти переваги досліджених та позбутися їх недоліків та написання програмного забезпечення на основі даного методу, аналіз його основних переваг та недоліків що має довести його ефективність.

Тобто, об’єктом даного дослідження є архітектура програмного забезпечення вантажного БПЛА, а предметом - методи та способи побудови такого програмного забезпечення.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. АНАЛІЗ ВИМОГ ДО МЕТОДУ ПОБУДОВИ ПЗ БПЛА

1.1 Загальні положення

БПЛА, або безпілотний літальний апарат – це апаратний пристрій, що здатен самостійно підтримувати себе у повітрі за допомогою різних рушіїв: гвинтів, крил, реактивних двигунів тощо за відсутності пілота на борту. Вони можуть мати конфігурації літака, гелікоптера, мультикоптера (апарату що не має крил і підтримує себе в повітрі виключно за допомогою трьох і більше однакових гвинтів) або ракети. За означенням, безпілотним літальним апаратом може вважатись навіть дитяча іграшка на радіокеруванні. Проте дане дослідження надалі стосуватиметься лише таких літальних апаратів, що підтримуються в повітрі як без наявності пілота на борту, так і без участі пілота-оператора який ним керує напряму. Це важливо для розуміння подальшого дослідження предметної області.

Програмне забезпечення БПЛА в даному контексті – це програмний комплекс, що здатен самостійно утримувати літальний апарат у повітрі, саджати його на землю і самостійно виконувати різноманітні польотні завдання (наприклад, переліт із точки А в точку Б по координатах) без участі оператора. При цьому такі завдання можуть ставитись як людиною, так і іншим компонентом даного програмного забезпечення автоматично.

Метод побудови програмного забезпечення БПЛА – це спосіб, у який програмний інженер, діючи за чітко визначеними кроками, дотримуючись інструкцій і рекомендацій даного методу, здатен самостійно, не запобігаючи до самостійного проектування системи, створити програмне забезпечення для керування безпілотним літальним апаратом. При цьому дане програмне забезпечення має бути повноцінним, відповідати низці мінімальних вимог до такого роду ПЗ, а процес його створення має бути очевидним. Даний метод має складатись з архітектури програмного забезпечення, кроків реалізації і випробування програмного забезпечення, переліку технологій що мають бути

застосовані в процесі його створення, а також рекомендацій які здатні спростити процес розробки.

1.2 Огляд літератури з використанням методу Systematic Mapping Study

Предметна область методів побудови програмного забезпечення для безпілотних літальних апаратів є досить обширною, потребує ретельного вивчення і містить велику кількість наукових статей, різноманітних публікацій та інших, менш авторитетних джерел інформації. Щоб дослідити і детально проаналізувати таку величезну кількість інформації, особливо такої, що міститься на просторах мережі Інтернет, необхідно витратити величезну кількість часових ресурсів. Одному досліднику на виконання такої задачі може не вистачити й цілого життя.

Для вирішення вищезазначеної проблеми та ретельного, ефективного дослідження предметної області в даному науковому дослідженні був використаний спеціальний метод для пошуку і фільтрації найбільш значущих для конкретного випадку джерел інформації метод під назвою Systematic Mapping Study.

Systematic Mapping Study (систематичний огляд літератури) [1] – це вид дослідження предметної області, що використовує метод під назвою Systematic Mapping Process для пошуку, збору та аналізу інформації пов’язаної з цією предметною областю.

При цьому метод Systematic Mapping Process є неупередженим, а його результати є повторюваними.

Далі будуть коротко описані всі етапи даного методу і результати, які мають бути отримані наприкінці кожного з них, а після цього у відповідності з цими етапами й буде проведення дослідження предметної області даної роботи.

На рисунку 1.1 зображена діаграма, яка відображає порядок виконання етапів методу Systematic Mapping Process. З лівої сторони, в прямокутниках,

зображені самі етапи, а з правої – результати що отримуються наприкінці кожного з них, в округлих прямокутниках.

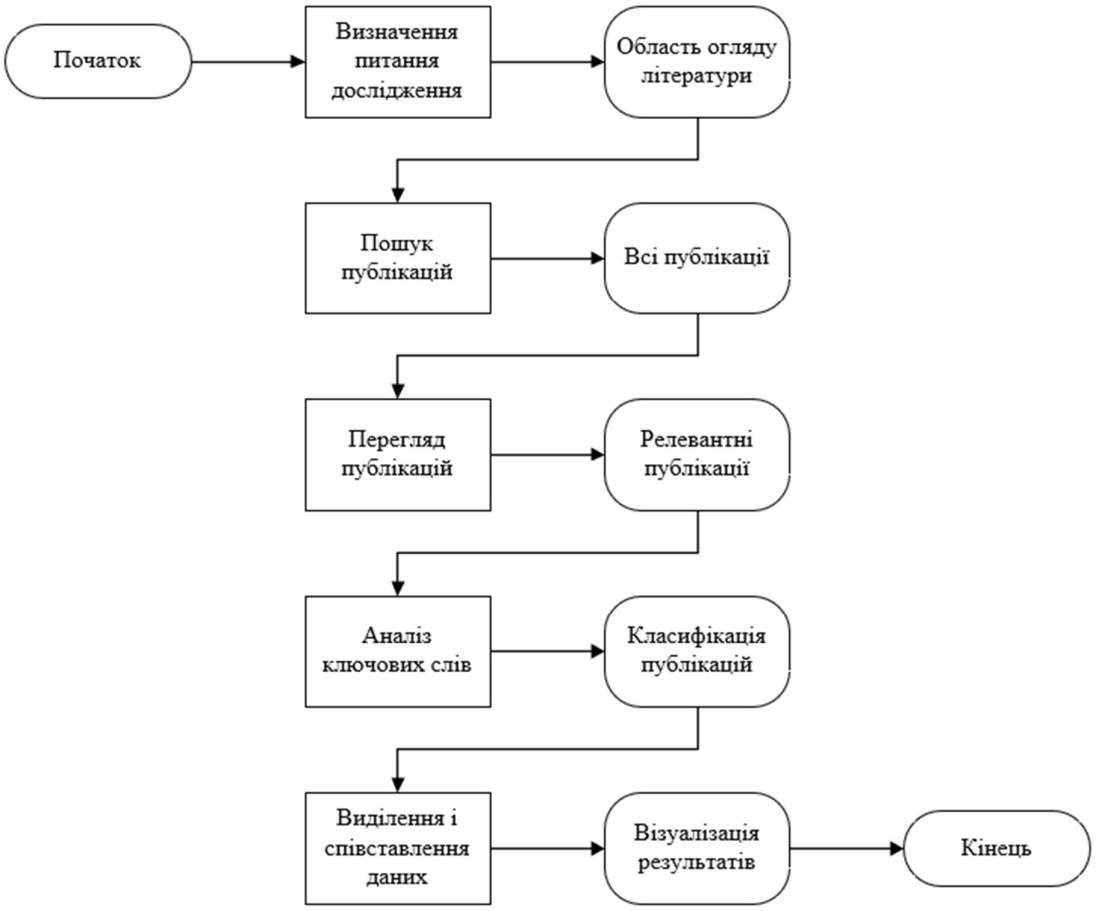


Рисунок 1.1 - Метод Systematic Mapping Process

Першим етапом даного методу є визначення питань дослідження. На даному етапі формуються основні питання, за якими в подальшому будуть відібрані матеріали що є актуальними для даного дослідження. Саме тому для постановки цих запитань необхідне знання предметної області, бо від цього етапу залежить перелік матеріалів на які можна буде опиратись в подальшому дослідженні.

Результатом першого етапу Systematic Mapping Process є перелік питань, що формують область огляду літератури яка нас цікавить.

Для предметної області методів побудови програмного забезпечення безпілотних літальних апаратів були поставлені запитання англійською

мовою, адже переважна кількість проведених досліджень в даній сфері була проведена іноземними дослідниками. Перелік цих запитань наведений у таблиці 1.1.

Таблиця 1.1 - Перелік питань дослідження

№	Питання
1.	What are the main methods of UAV software construction?
2.	What is the UAV ecosystem?
3.	What are the main approaches of implementation of UAV autopilot?
4.	What are the main characteristics of UAV software?
5.	What are the digital ecosystem models that can be applied to UAV ecosystem?

Наступним етапом методу є пошук публікацій. Пошук має проводитись у таких джерелах як бази даних Інституту інженерів електротехніки та електроніки (IEEE), інших організацій що займаються публікацією наукових робіт з програмної інженерії та пошуковий сервіс Google. При цьому при використанні останнього необхідно переходити лише на авторитетні сайти, такі як наукові журнали, інститути тощо.

Результатом даного етапу має стати перелік публікацій, в яких міститься потенційно корисна для вивчення цільової предметної області інформація.

Для реалізації даного етапу необхідно сформувати перелік пошукових запитів, які допоможуть знайти джерела в яких найбільш точно розкриваються питання, поставлені в першому етапі. Для цього з цих питань слід перш за все виокремити основні терміни що нас цікавлять. Ось перелік таких термінів:

- UAV ecosystem;
- UAV autopilot;
- Digital ecosystem;

- UAV software architecture;

Наступним кроком є формування пошукових запитів на основі цих ключових слів. Оскільки пошук по окремим ключовим словам може не дати належного результату, то слід дані ключові слова поєднувати в запитах за допомогою булевих операторів OR та AND. Перелік складених запитів наведений у таблиці 1.2.

Таблиця 1.2 - Пошукові запити

№	Запит
1.	“UAV ecosystem”
2.	“Digital ecosystem software” OR “Digital ecosystem UAV”
3.	“UAV autopilot” AND “UAV delivery system”
4.	“UAV software architecture”

Під час пошуку за цими пошуковими запитами були отримані результати, зображені у таблиці 1.3.

Таблиця 1.3 - Результати пошуку

Запит	Кількість результатів
“UAV ecosystem”	14 500
“Digital ecosystem software” OR “Digital ecosystem UAV”	2140
“UAV autopilot” AND “UAV delivery system”	7
“UAV software architecture”	1 480

В третьому етапі ці результати мають бути відфільтровані, тому що для перегляду всіх результатів з кроку 2 знадобиться дуже велика кількість часу. Для цього метод Systematic Mapping Process передбачає створення критеріїв включення за якими джерела включатимуться у результуючий перелік якщо всі вони виконуються і критеріїв виключення, за якими джерела

виключатимуться з нього, якщо виконується хоча б один з цих критеріїв. Дані критерії наведені у таблицях 1.4 та 1.5.

Таблиця 1.4 - Критерії включення

№	Критерій включення
1.	Мова дослідження - англійська або українською мовами
2.	Дослідження опубліковане авторитетним джерелом (наприклад IEEE Xplore)
3.	Публікація має бути не простроченою на поточний момент часу
4.	Публікація має щонайменше 3 посилання на інші публікації з авторитетних джерел
5.	Посилання на дослідження знаходиться серед перших 50 результатів пошукового запиту за релевантністю

Таблиця 1.5 - Критерії виключення

№	Критерій виключення
1.	Дослідження вже було знайдене іншим пошуковим запитом або в іншому джерелі
2.	Дослідження дублює раніше знайдене за своїм змістом
3.	Дослідження неможливо завантажити
4.	Дослідження стосується авіоніки або деталей розробки безпілотних літальних апаратів а не програмної інженерії

Наступний етап передбачає категоризацію наукових робіт за обраними критеріями. Це робиться для того щоб потім можна було проаналізувати добре досліджені та менш досліджені області галузі і зробити висновок про те, які публікації є найбільш корисними для цільового дослідження або які прогалини таким дослідженням слід заповнювати. Першим кроком даного етапу є складання критеріїв. Для даного дослідження пропонується поділити відфільтровані публікації за класифікацією, наведеною в роботі під назвою

«Requirements engineering paper classification and evaluation criteria: A proposal and a discussion» [2].

Таблиця 1.6 - Класифікація інженерних наукових робіт із запропонованого джерела

Вид наукової роботи	Короткий опис
Оціночне дослідження	Дослідження ґрунтується на результатах іншого, рішення запропоновані в попередньому дослідженні перевіряються на практиці, оцінюється їх ефективність та порівнюється із висновками попереднього.
Дослідження що пропонує спосіб вирішення задачі	Дослідження пропонує кардинально новий підхід до вирішення задачі або суттєво розширює існуючий. Ефективність підходу підтверджується на практиці або теоретично.
Перевірочне дослідження	Перевіряється метод вирішення задачі, що був запропонований раніше, але не був ніким реалізований до цього.
Філософське дослідження	Дослідження цілковито формує новий підхід до певної предметної області.
Власна думка	Висловлюється власна думка автора щодо певних речей. Аргументація не є ґрунтовною.
Власний досвід	Автор ділиться своїм власним досвідом із вирішення тої чи іншої проблеми.

Далі на даному етапі публікації аналізується анотація публікацій, вступи і висновки даних робіт. Роботи співвідносяться з обраними категоріями. Нижче наведена класифікація наукових робіт за вищезазначеним критерієм.

Таблиця 1.7 - Класифікація відібраних публікацій за видом наукової роботи

Вид наукової роботи	Публікації	Кількість
Оціночне дослідження	10	1
Дослідження що пропонує спосіб вирішення задачі	1, 2, 3, 6, 8, 9	6
Перевірочне дослідження	4, 5, 7	3
Філософське дослідження	-	0
Власна думка	11, 13	2
Власний досвід	12, 14	2

Для більш детального аналізу предметної області методу побудови програмного забезпечення вантажного БПЛА введемо ще один критерій – критерій за програмним забезпеченням. Дана класифікація необхідна для розуміння того, щодо якої конкретної частини програмного забезпечення ведеться дослідження, адже дослідження лише якоїсь частини даного ПЗ в даному випадку не підходить. Дана класифікація описана детальніше у таблиці 1.8.

Таблиця 1.8 - Класифікація наукових робіт за програмним забезпеченням

Вид наукової роботи	Короткий опис
Архітектура ПЗ БПЛА	Дослідження пропонує цілісну архітектуру побудови ПЗ БПЛА

Продовження таблиці 1.8

Метод побудови ПЗ БПЛА	Дослідження пропонує новий метод побудови ПЗ БПЛА
ПЗ автопілоту	Дослідження покриває лише особливості реалізації програми - автопілоту
Вирішення однієї задачі	Дослідження покриває лише особливості вирішення БПЛА однієї конкретної задачі (наприклад, автоматична посадка)
Патент на програмне забезпечення	Дана публікація є патентом на архітектуру ПЗ БПЛА

Результати класифікації наукових робіт за даним критерієм наведені у таблиці 1.9.

Таблиця 1.9 - Класифікація відібраних публікацій за програмним забезпеченням

Вид наукової роботи	Публікації	Кількість
Архітектура ПЗ БПЛА	3, 4, 8, 9,	4
Метод побудови ПЗ БПЛА	-	0
ПЗ автопілоту	11, 12	2
Вирішення однієї задачі	1, 2, 5, 6, 14	5
Патент на програмне забезпечення	7, 10, 13	3

Останнім етапом дослідження цільової області методом Systematic Mapping Study є виділення і співставлення даних, отриманих на передостанньому етапі.

Результати такого співставлення наведені на рисунку 1.2.

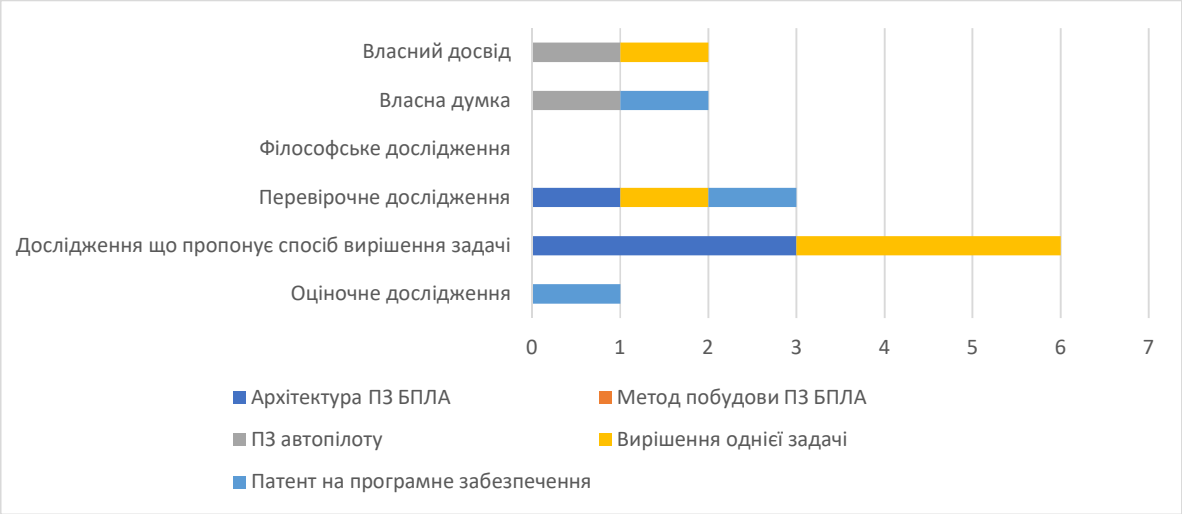


Рисунок 1.2 - Результати аналізу літератури методом Systematic Mapping Study

Які ж висновки ми можемо зробити після проведеного аналізу літератури методом Systematic Mapping Study? Серед всіх досліджених публікацій є доволі мало наукових статей, які пропонують нові методи вирішення задачі побудови програмного забезпечення безпілотного літального апарату. Всього 3 знайдені статті пропонують нову архітектуру, а не експлуатують вже запропоновані раніше рішення або вирішують якусь одну задачу пов'язану з управлінням безпілотниками. І немає жодної статті, яка б пропонувала комплексний метод для побудови такого програмного забезпечення із чітким описанням кроків, які необхідно виконати для його створення. Це ще раз підтверджує доцільність і актуальність проведення даного дослідження.

Підходи до програмування БПЛА з розглянутих наукових робіт будуть проаналізовані та порівняні у наступних підрозділах.

1.3 Огляд літератури

Розглянемо декілька найбільш значимих для даного дослідження робіт, що були відібрані в розділі вище і проаналізуємо їх.

Першою з розглянутих статей є «An Ecological Approach to the Supervisory Control of UAV Swarms» [3].

Основною метою даного дослідження є розробка архітектури системи управління БПЛА найбільш оптимальним шляхом, щоб затрачувати на це мінімальну кількість дій як з боку оператора, так і з боку самих безпілотників, оптимізуючи їх роботу.

З цього дослідження нам корисно те, що воно розбиває функції, що має виконувати програмне забезпечення БПЛА на різні рівні абстракції (Рисунок 1.2).

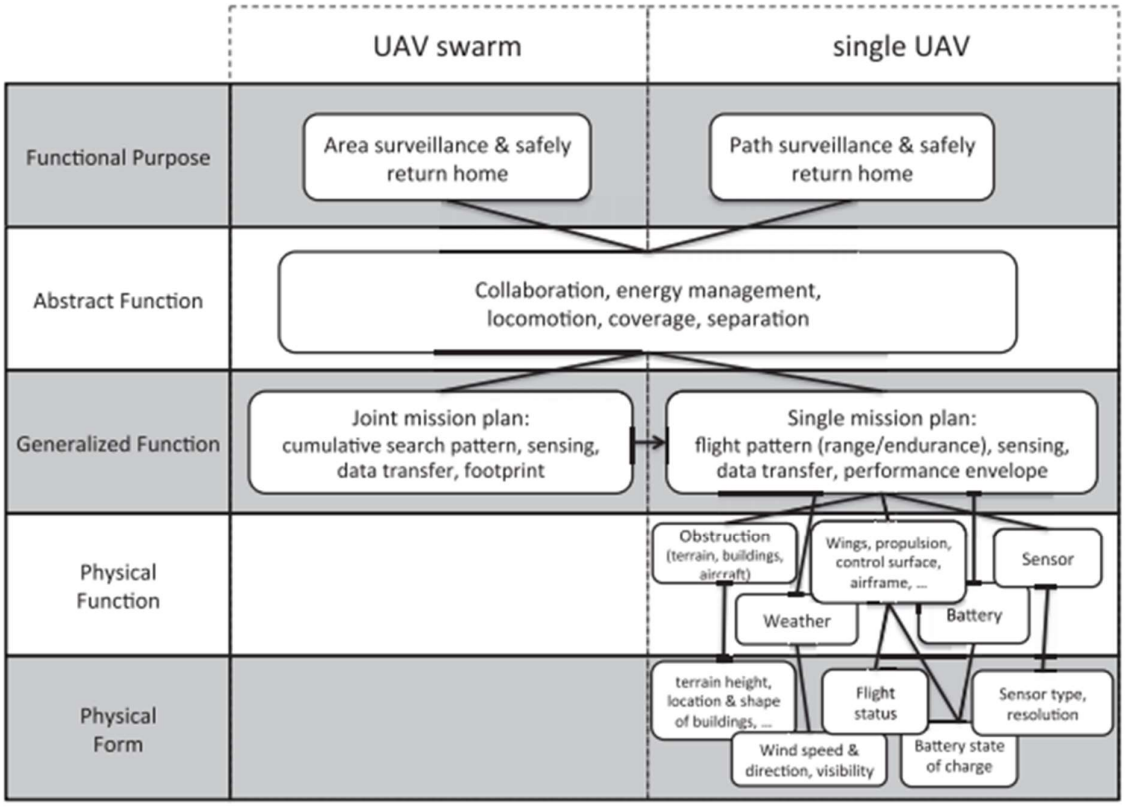


Рисунок 1.3 - Ієрархія абстракцій функцій БПЛА

Тут зображене порівняння рівнів абстракції флоту БПЛА та одного БПЛА, проте нам цікаві тут саме запропоновані рівні абстракції. Дивлячись

на дані рівні абстракції, можна легко виконати задачу декомпозиції ПЗ екосистеми БПЛА на різні модулі із чіткою сферою відповідальності, що буде застосовано в подальшій розробці.

Наприклад, верхній рівень можна виокремити в окремий модуль ПЗ, що ставитиме задачі для БПЛА, другий і третій – в окремий модуль, що відповідатиме за виконання даних завдань, а два останні – в модуль автопілоту що відповідає безпосередньо за контроль апаратного забезпечення.

Друга розглянута стаття під назвою «A Systematic Review of Unmanned Aerial Vehicle Application Areas and Technologies in the Aec Domain» [4] ставить за мету дослідити предметну область ПЗ БПЛА за застосовністю до різних сфер практичного застосування безпілотників.

З даної роботи ми можемо почерпнути, що повністю автономні БПЛА найбільш корисні у виконанні рутинних, повторюваних завдань, що мають низьку вірогідність виникнення помилок. Основними сферами їх застосування є: інфраструктура, спостереження з повітря, вантажні перевезення і дослідження різноманітних об’єктів та будівель на предмет пошкоджень, руйнувань.

Дана інформація важлива, тому що вона показує, які сфери застосування БПЛА наразі є найбільш важливими і значущими і в яких напрямках слід проводити дослідження.

Наступна робота називається «Unmanned Aircraft Systems» [5]. Вона детально описує предметну область безпілотних літальних апаратів і надає інформацію про область застосування БПЛА, види безпілотників, їх призначення, описує вимоги до БПЛА та межі їх застосування.

З даної роботи була почерпнута інформація про наступні необхідні функції вантажного БПЛА, а саме:

- уникнення перешкод, що дозволяє літати як в умовах міського ландшафту, так і уникати ліній електропередач що легко можуть пошкодити літальний апарат і призвести до його падіння;

- точна посадка, що необхідна для виконання деяких специфічних задач, таких як доставка вантажу;
- GPS-навігація, без якої автономний політ без участі пілоту є неможливим.

А також були складені відповідні вимоги до програмного забезпечення літального апарату.

Наступна розглянута робота називається «Software Ecosystems A Sytematic Literature Review» [6]. В ній автори досліджують релевантні наукові джерела на предмет застосування підходів до розробки програмного забезпечення з точки зору їх екосистем. Дане дослідження чітко визначає сфери застосування екосистем програмного забезпечення. На жаль, даний підхід наразі не застосовується до побудови ПЗ БПЛА, хоча він є цілком застосовним до даної сфери.

Виходячи з результатів проведеного дослідження предметної області, можна зробити чіткий висновок, що предметна область методів побудови ПЗ БПЛА розвинена досить мало. Запропонованих іншими дослідниками архітектур такого програмного забезпечення представлено досить мало, більшість публікацій стосуються вирішення однієї конкретної задачі в ПЗ БПЛА.

Натомість існує дуже багато інструкцій з побудови такого програмного забезпечення, що носить ненауковий характер. Дане дослідження на такі джерела опиратись не може, проте деякі програмні рішення та підходи будуть використані, перевірені на практиці, і включені до даного методу побудови ПЗ після доведення їх ефективності. Проте дані інструкції з побудови ПЗ також стосуються вирішення конкретних задач, тому їх одних також не вистачає для створення методу побудови ПЗ. Це та прогалина, покрити яку і взято за мету дослідження.

Якщо бігло оглянути всі знайдені підходи до створення програмного забезпечення безпілотників, то можна зробити висновок, що в загальному дане програмне забезпечення складається з трьох основних елементів:

програмного забезпечення самого літального апарату, автопілоту та програмного забезпечення наземної станції керування.

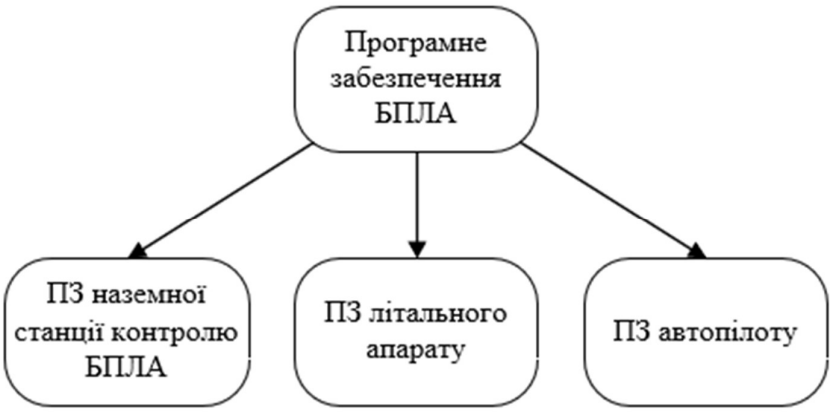


Рисунок 1.4 - Структура програмного забезпечення БПЛА

Програмне забезпечення наземної станції контролю безпілотного літального апарату призначено для того, щоб була змога керувати літальним апаратом, поки він знаходиться в повітрі і слідкувати за виконанням польотного завдання. В разі якої-небудь позаштатної ситуації саме воно надає змогу екстрено втрутитись у роботу БПЛА. При цьому концептуально не має жодного значення, хто саме керує даною наземною станцією – оператор-людина чи інша програма. Обидва підходи мають право на життя і активно наразі застосовуються.

Програмне забезпечення літального апарату встановлюється і запускається безпосередньо на самому літальному апараті і відповідає за спілкування з наземною станцією, зчитування даних з датчиків та керування корисним навантаженням безпілотника (камерами, серводвигунами тощо).

Програмне забезпечення автопілоту також встановлюється на літальному апараті, але воно завжди виокремлюється з попереднього в окремий модуль, адже воно має прямий доступ до апаратного забезпечення БПЛА що підтримує його в повітрі. І у більшості архітектур наземна станція

не має прямого доступу до керування автопілотом, а здійснює це через програму-прошарок.

1.4 Порівняльний аналіз існуючих наукових досліджень ПЗ БПЛА

В даному розділі будуть розглянуті три найбільш детальні архітектури з наукових робіт, обраних методом Systematic Mapping Study. Критерієм відбору саме цих робіт стала їх детальність, ґрунтовність та завершеність, оскільки кожен підхід описує архітектуру повноцінного програмного комплексу безпілотного літального апарату.

Першим дослідженням в цьому списку є робота під назвою «A Multiple-UAV Software Architecture for Autonomous Media Production» [8]. Дана робота представляє спосіб створення програмного забезпечення безпілотного літального апарату для виконання задач із повітряної зйомки різноманітних наземних об’єктів. Основною метою даної публікації було створення програмного забезпечення системи БПЛА для повітряної зйомки, що здатна обслуговувати повітряний флот літальних апаратів за забезпечити захист від нештатних ситуацій що можуть виникнути в процесі експлуатації даної системи. Як зазначають дослідники у висновках до даної роботи, це їм цілком вдалось.

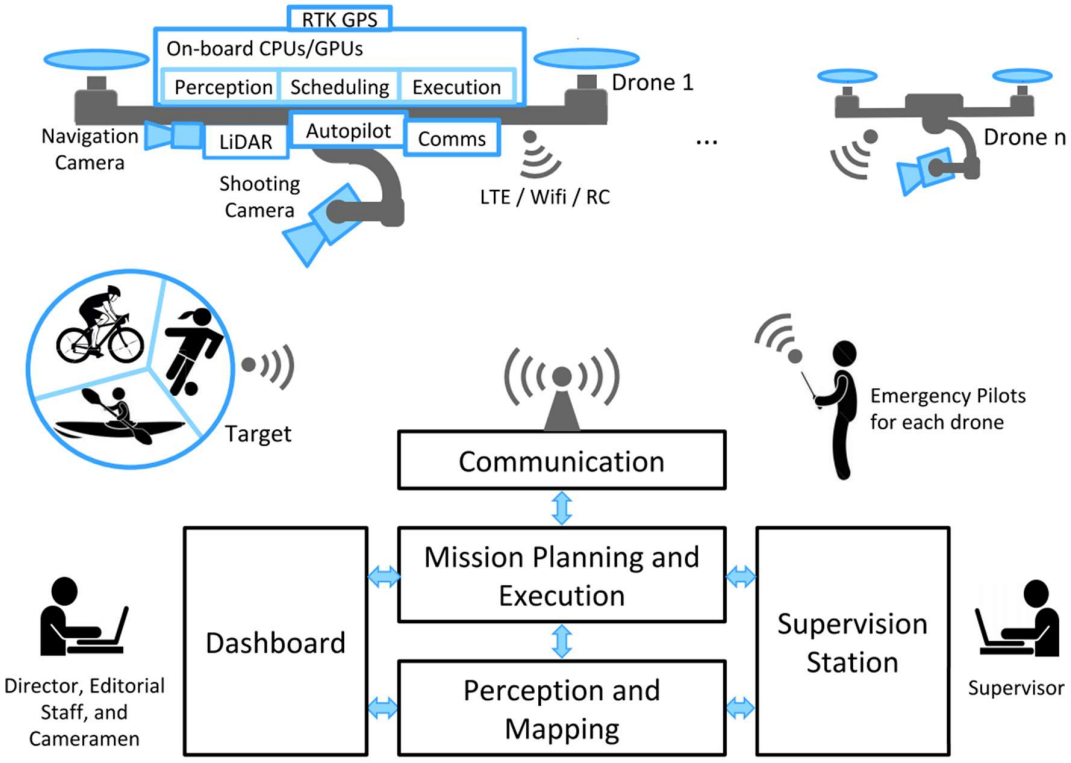


Рисунок 1.5 - Архітектура з дослідження A Multiple-UAV Software Architecture for Autonomous Media Production

Коротко розглянемо дану архітектуру. Дослідники пропонують будувати програмне забезпечення безпілотників з окремих програмних модулів, що є добре, спрощує розробку і підтримку такого ПЗ а також дозволяє розподіляти виконання задач між літальним апаратом та наземною станцією керування.

Програмне забезпечення самого літального апарату є досить вдало продуманим. Воно складається окремо з автопілоту та з модуля виконання супутніх завдань: спілкування з наземною станцією, обробка даних з камер та датчиків, планування та виконування польотних завдань. Таке рішення дозволяє виокремити автопілот в окремий програмний продукт або використати вже готовий модуль. Це суттєва перевага.

Обмін даними між літальним апаратом і наземною станцією здійснюється шляхом бездротової передачі даних через мережу Інтернет (Wifi / LTE). Дане рішення також є дуже правильним, адже таким чином

розташування наземної станції відносно літального апарату не має жодного значення. Вона може знаходитись хоч в іншій країні. Головне – забезпечити доступ до якісного Інтернет-з'єднання обом акторам даної схеми.

Проблеми виникають із запропонованим авторами даного дослідження архітектурою ПЗ наземної станції. Вона складається аж з п'яти окремих модулів, та потребує неперервної присутності аж двох операторів одразу: того що ставить польотні завдання безпілотним апаратам, і того, що верифікує процес виконання цих завдань. З цього одразу випливає суттєвий мінус даного підходу – персонал має надмірний контроль за діями флоту БПЛА і має в напівручному режимі контролювати весь процес зйомок. До того ж, звідси виникає ще одна, менш очевидна, але більш суттєва проблема. Дана розробка потребує програмування одразу аж двох складних і багатофункціональних графічних інтерфейсів для наземної станції, що суттєво збільшує час розробки даного програмного забезпечення, а останнє є досить критичним, адже в наш час якісні спеціалісти що цим займаються, коштують доволі дорого.

З цього робимо висновки, що дана архітектура має низку вдалих рішень щодо побудови програмного забезпечення самого літального апарату, проте через велику складність реалізації ПЗ наземної станції керування і залучення додаткового персоналу, дана архітектура не може стати повноцінною основою для побудови ПЗ вантажного БПЛА.

Наступною розглянутою роботою є дослідження під назвою «A Modular Software Architecture for UAVs» [9]. Основною метою даного дослідження було проектування системи управління БПЛА, що була б незалежною від апаратного забезпечення і дозволяла б швидку розробку такого програмного забезпечення.

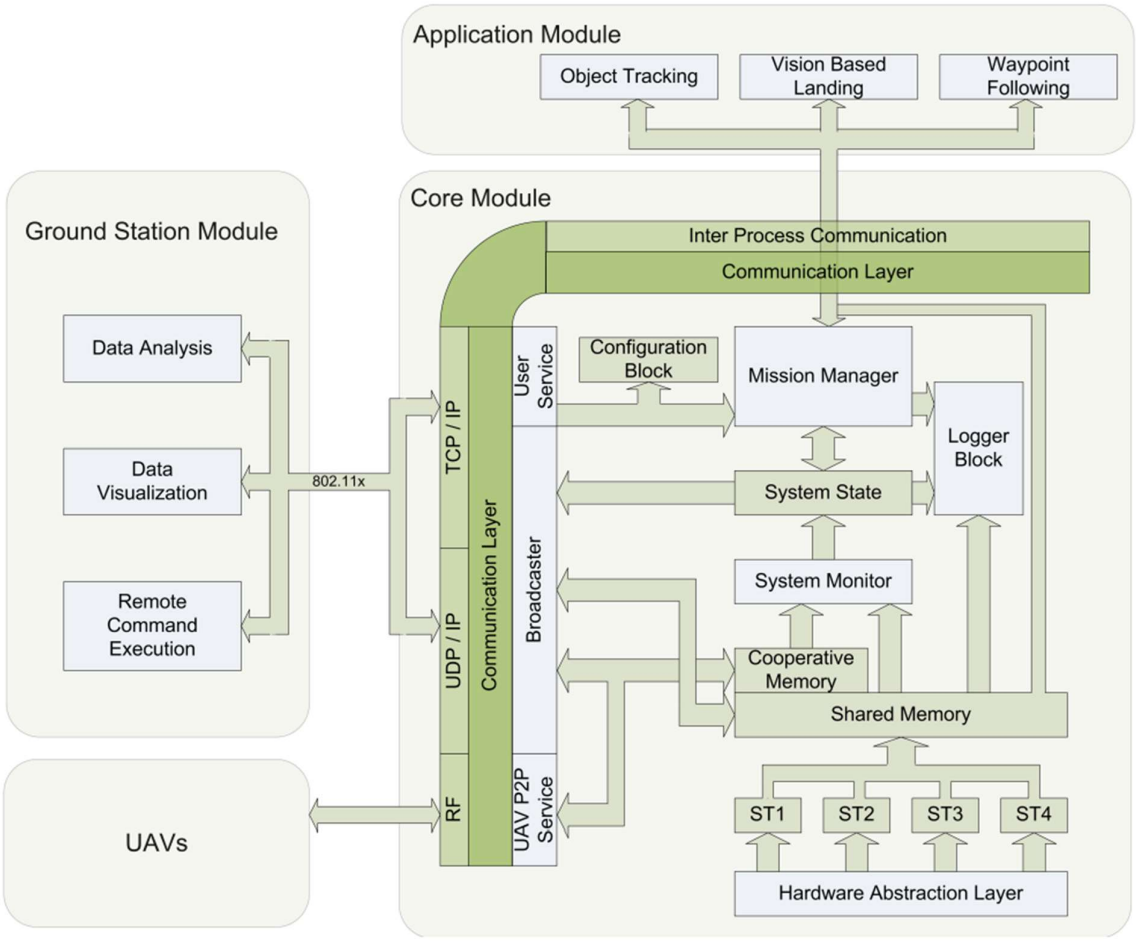


Рисунок 1.6 - Архітектура з дослідження A Modular Software Architecture for UAVs

Дана архітектура складається з трьох основних модулів. Модуль під назвою Application Module є аналогом модуля виконання супутніх завдань з попереднього завдання, і виконує ті самі задачі, тому додатково розписувати його особливості немає сенсу.

Модуль що має назву Core Module і є основним модулем в даному програмному забезпеченні. Узагальнено, даний модуль є аналогом модуля автопілотування, що так само працює із апаратним забезпеченням БПЛА напряму, проте не повноцінним. Для підтримання себе в польоті йому потрібен Application Module, оскільки Core Module в даній архітектурі не обробляє самостійно дані з GPS, акселерометра та барометра, що є життєво необхідним для повноцінного автопілоту. До того ж, дана архітектура

передбачає обмін даними із наземною станцією по стандарту 802.11x (Wifi), що суттєво обмежує радіус дії такого безпілотника відносно наземної станції і вимагає встановлення потужного і, відповідно, затратного за електроенергією Wifi-модуля. Врешті-решт, самостійна реалізація даного модуля є задачею далеко нетривіальною через його надмірну складність, хоч вона і обумовлена відповідними вимогами до надійності даного ПЗ.

Останнім модулем даної архітектури є модуль наземної станції. І одразу можна сказати, що через нього вся дана архітектура не підходить для задачі розробки ПЗ вантажного БПЛА який має виконувати функції адресної доставки вантажу. А все через те, що його функціонал лише самий базовий і ніяк не піддається розширенню. Це суто задачі із переміщення по чітко визначеному маршруту.

Висновок: дана архітектура хоч і є продуманою і надійною, проте її реалізація надто складна і не надає функціоналу, необхідного і достатнього для виконання цілей даного дослідження.

Наступною буде розглянута архітектура з дослідження «A HARDWARE/SOFTWARE ARCHITECTURE FOR UAV PAYLOAD AND MISSION CONTROL» [10]. Його основною метою було проектування програмного забезпечення для БПЛА з можливістю контролю виконання польотних завдань та підтримки корисного навантаження літального апарату (камер, датчиків тощо).

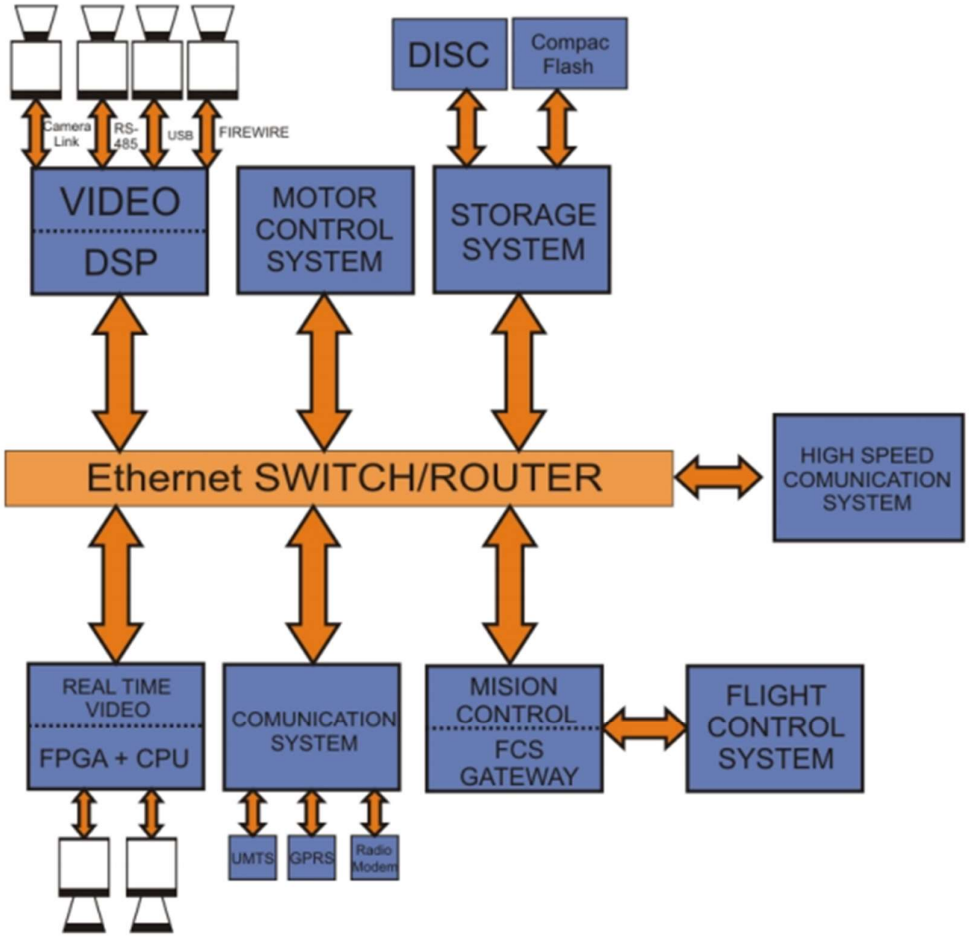


Рисунок 1.7 - Архітектура з дослідження A HARDWARE/SOFTWARE ARCHITECTURE FOR UAV PAYLOAD AND MISSION CONTROL

Дана архітектура складається з трьох основних підсистем. Перша з них – це програмно забезпечення наземної станції (те саме ПЗ що відповідає за контроль і виконання польотних завдань), контролеру польоту і корисного навантаження (його архітектура зображена на рисунку 1.6), який і є основним компонентом даної системи, і інфраструктури обміну даними між ними. Оскільки дане дослідження зосереджує увагу саме на контролері польоту і корисного навантаження, то проаналізуємо його основні переваги і недоліки.

Перш за все, даний елемент програмного забезпечення БПЛА має розгалужену модульну структуру, що дозволяє виокремити обробку кожного потоку вхідних даних (зображення з камери, дані датчика відстані та подібні) в окремий модуль. Це однозначна перевага над іншими архітектурами, адже

вона дозволяє легко розширювати та змінювати функціонал безпілотника, та окремо тестувати роботу кожного модуля.

Але далі, за таким плюсом йде низка суттєвих недоліків. По-перше, всі модулі даного ПЗ передбачають їх встановлення і запуск на самому літальному апараті. Це потребує значної обчислювальної потужності на борту БПЛА, що збільшує його вартість, і підвищує енергоспоживання, що зменшує його автономність. По-друге, вся логіка обробки даних і планування польотів по суті лежить на одному пристрої, і попри модульність даного програмного забезпечення і простоти тестування, може призводити до чисельних збоїв в роботі. По-третє, всі модулі даного ПЗ обмінюються між собою даними через локальну мережу, що не є найефективнішим методом передачі даних, особливо якщо це стосується даних якісного відео-формату. І останнім, та найважливішим недоліком даної архітектури, безперечно є відсутність будь-якої системи взаємодії із кінцевим користувачем, тому що ПЗ наземної станції цією архітектурою хоч і передбачається, проте його функціонал є недостатнім для програмного забезпечення безпілотного літального апарату.

Висновок: використання цієї архітектури в рамках даного дослідження теж недоцільне через невідповідність архітектури ПЗ вимогам даного дослідження.

1.5 Аналіз вимог до методу побудови ПЗ БПЛА

Розробляючи метод побудови програмного забезпечення вантажного безпілотного літального апарату, необхідно визначити нефункціональні вимоги до даного методу а також функціональні вимоги до самого програмного забезпечення, яке має бути результатом застосування методу розробки ПЗ.

1.5.1 Розробка нефункціональних вимог до методу побудови програмного забезпечення вантажного БПЛА

Основними нефункціональними вимогами до методу побудови ПЗ БПЛА є наступні:

- структурованість: метод зі створення ПЗ має бути описаний в порядку виконання етапів розробки, та кожен наступний етап розробки має логічно випливати з попереднього;
- детальність: метод має бути описаний з достатнім рівнем деталізації, щоб забезпечувати реалізацію робочого програмного забезпечення;
- повнота: метод має бути описаний в достатньому обсязі, щоб у розробника не виникало питань про архітектурні особливості системи що реалізується;
- простота: опис методу має бути достатньо простим і наочним, щоб не потребувати роз'яснень кваліфікованому спеціалісту, що реалізує програмне забезпечення за даним методом.

1.5.2 Розробка функціональних вимог програмного забезпечення вантажного БПЛА

Розглядаючи варіанти використання ПЗ БПЛА, можливі два основні підходи до інтерпретації ролі самого літального апарату в ньому.

Першим, традиційним підходом, є опис програмного забезпечення безпілотного літального апарату в ролі окремої підсистеми. І все подальше проектування програмного забезпечення в даному випадку відштовхується саме від програмування даної підсистеми.

Дане дослідження пропонує поглянути на ПЗ БПЛА з іншого боку, а саме – розглядати літальні апарати як актори екосистеми ПЗ БПЛА. Адже по своїй суті безпілотник є своєрідним клієнтом системи, що його обслуговує і супроводжує.

В даному випадку поняття «екосистеми програмного забезпечення безпілотного літального апарату» має на увазі апаратно-програмний комплекс, що дозволяє відслідковувати і ставити задачі для автоматичного виконання одному або системі БПЛА, які є акторами-клієнтами даного комплексу.

Все подальше проектування і розробка проводилась в рамках контексту поняття «екосистеми ПЗ БПЛА».

В архітектурі екосистеми ПЗ БПЛА мають бути задіяні два основні підсистеми – це підсистема обслуговування клієнтів, та підсистема контролю БПЛА.

До підсистеми обслуговування клієнтів належать наступні варіанти використання: реєстрація, вхід в систему, замовлення доставки та перегляд статусу замовлення. Дана підсистема взаємодіє з єдиним актором – клієнтом служби доставки. Всі вище перелічені варіанти використання підсистеми належать саме йому.

Під реєстрацією клієнта служби доставки мається на увазі додавання його персональних даних в систему, щоб дозволити в подальшому в цю систему заходити. Персональні дані користувача в даному випадку необхідні для ідентифікації власника товару що доставляється.

Процес замовлення доставки - це створення самого замовлення, додавання всіх необхідних даних до системи для того щоб система безпомилково доставила товар на місце призначення. Також при цьому відбувається доставка вантажу підсистемою контролю БПЛА.

Процес перегляду статусу замовлення є простим за своєю суттю – клієнт отримує інформацію про місцезнаходження посилки. При цьому відбувається моніторинг доставки вантажу підсистемою контролю БПЛА.

Підсистема контролю БПЛА має забезпечувати управління і контроль літальними апаратами. Вона взаємодіє з двома акторами – з адміністратором, що за потрібності налагоджує її роботу та БПЛА, та з самими літальними апаратами, що цей вантаж доставляють.

Загальна модель варіантів використання представлена на рисунку 1.7.

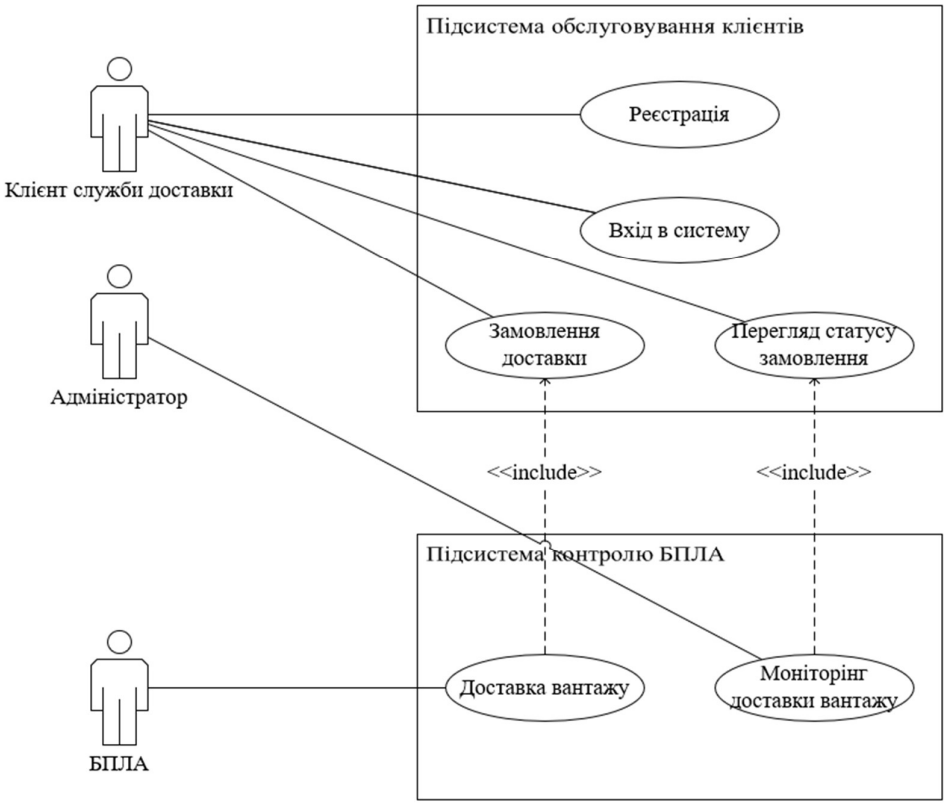


Рисунок 1.8 - Модель варіантів використання

Таблиця 1.10 - Функціональні вимоги

№	Назва функції	Код	Призначення функції	Актор	Пріоритет
1.	Реєстрація	REQ1	Реєстрація	Клієнт	головний
2.	Авторизація	REQ2	Вхід в систему	Клієнт	головний
3.	Створення замовлення на доставку	REQ3	Клієнт створює замовлення на доставку конкретної посилки	Клієнт	головний

Продовження таблиці 1.10

4.	Вибір локації доставки	REQ4	Клієнт обирає точку доставки	Клієнт	головний
5.	Оплата доставки	REQ5	Клієнт сплачує доставку	Клієнт	другорядний
6.	Перегляд статусу замовлення	REQ6	Клієнт відслідковує статус доставки вантажу	Клієнт	головний
7.	Перегляд місцезнаходження замовлення	REQ7	Клієнт відслідковує місцезнаходження вантажу	Клієнт	другорядний
8.	Моніторинг доставки вантажу	REQ8	Адміністратор відслідковує статус доставки вантажу	Адміністратор	головний
9.	Підбір посилки	REQ9	БПЛА підбирає посилку в її місцезнаходженні	БПЛА	головний
10.	Доставка посилки клієнту	REQ10	БПЛА прилітає за координатами доставки	БПЛА	головний
11.	Ідентифікація клієнта	REQ11	БПЛА ідентифікує наявність клієнта і залишає посилку	БПЛА	головний
12.	Повернення на базу	REQ12	БПЛА повертається в точку вильоту	БПЛА	головний

Таблиця 1.11 - Варіанти використання

№	Варіант використання	Код	Опис
1.	Реєстрація	UC1	Додавання даних користувача в систему
2.	Вхід в систему	UC2	Авторизація клієнта в системі
3.	Замовлення доставки	UC3	Додавання всіх даних, необхідних для доставки в систему
4.	Перегляд статусу замовлення	UC4	Отримання клієнтом статусу доставки
5.	Доставка вантажу	UC5	Безпосередньо процес доставки вантажу
6.	Моніторинг доставки вантажу	UC6	Перевірка стану і місцезнаходження БПЛА

Таблиця 1.12 - Матриця залежності між функціональними вимогами та варіантами використання

	UC1	UC2	UC3	UC4	UC5	UC6
REQ1	■					
REQ2		■				
REQ3			■			
REQ4			■			
REQ5			■			
REQ6				■		
REQ7				■		
REQ8					■	
REQ9						■
REQ10						■

Продовження таблиці 1.12

REQ11						
REQ12						

1.6 Висновки до розділу

В даному розділі була досліджена предметна область методів побудови програмного забезпечення БПЛА, описані основні прогалини які наразі існують в даній області і обґрунтована актуальність проведення даного дослідження.

Наступним кроком був проведений ґрунтовний аналіз досліджень попередників, були підмічені вдалі і невдалі рішення. На основі результатів цього дослідження були сформовані вимоги до методу побудови програмного забезпечення БПЛА та архітектури даного ПЗ.

І врешті, було введене інноваційне поняття екосистеми програмного забезпечення БПЛА, з точки зору якого далі буде розроблений метод для його побудови.

2 МЕТОД ПОБУДОВИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЕКОСИСТЕМИ БПЛА

Перш ніж сформулювати метод побудови програмного забезпечення екосистеми вантажного БПЛА, необхідно спроектувати дане програмне забезпечення, і відповідно до архітектури даного ПЗ, покроково описати всі етапи його реалізації.

2.1 Моделювання та аналіз програмного забезпечення

Для проектування даного програмного забезпечення необхідно дослідити варіанти його головних акторів, а саме, БПЛА та клієнта сервісу доставки.

В даному програмному забезпеченні реалізуються два основні бізнес-процеси - процес доставки вантажу, що здійснюється літальним апаратом, та процес створення замовлення, що здійснюється клієнтом.

Спершу розглянемо поетапно процес доставки вантажу (див. рисунок 2.1).

а) Першим етапом доставки є приліт БПЛА на точку підбору вантажу. Для цього безпілотник, отримуючи дані про маршрут польоту, надсилає команди автопілоту що дозволяють виконати переміщення відповідно маршруту;

б) Наступним етапом є ідентифікація та підбір вантажу. Безпілотник, аналізуючи дані з камери, знаходить потрібний вантаж, приземляється над ним та підбирає вантаж, механічно фіксуючи його в себе на борту;

в) Далі, отримуючи дані про маршрут до точки доставки вантажу, БПЛА летить за цим маршрутом в дану точку;

г) Зчитуючи дані з камери, літальний апарат ідентифікує клієнта доставки;

д) Ідентифікувавши власника вантажу, безпілотник приземляється, залишає вантаж та повертається в точку вильоту.

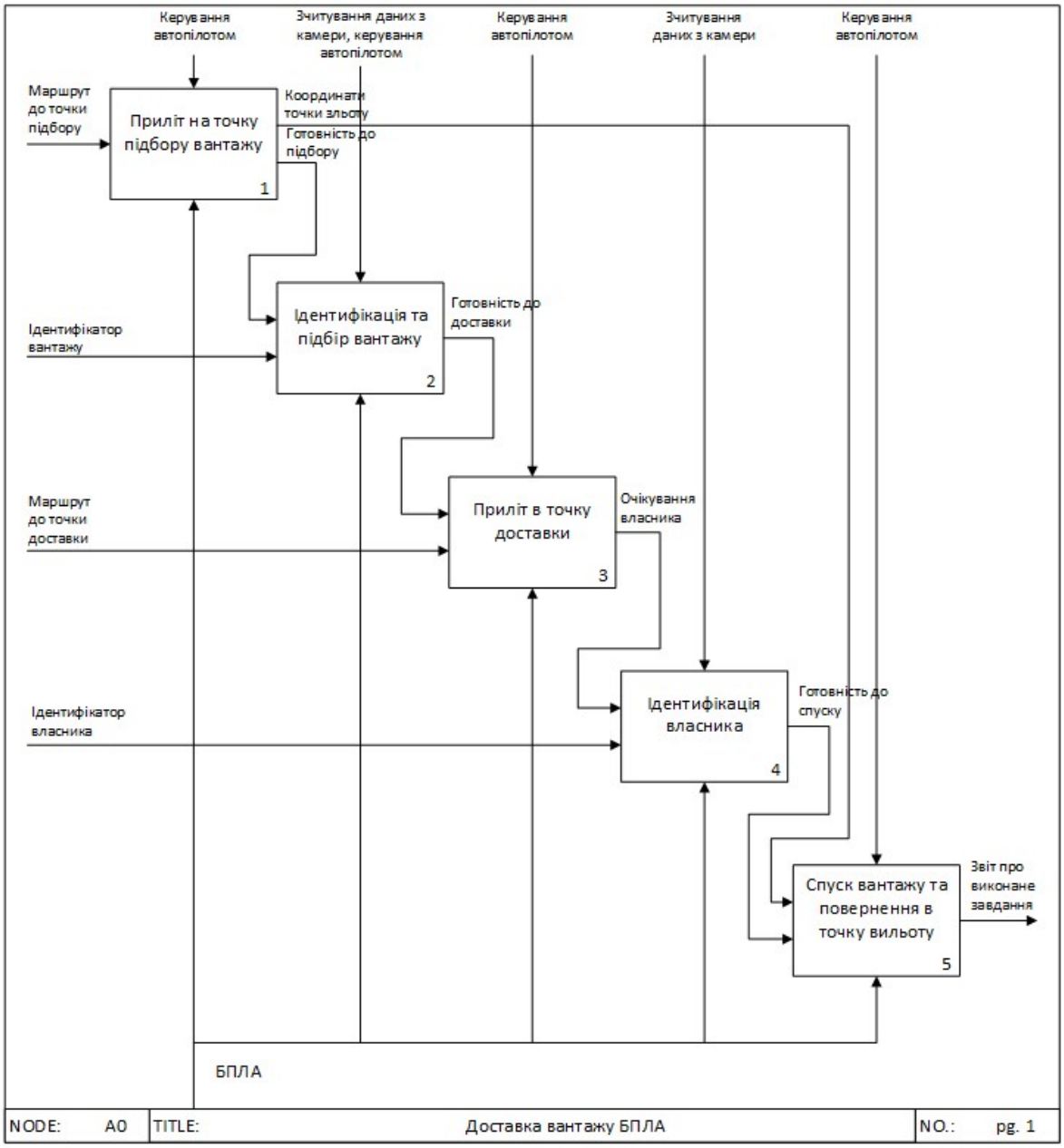


Рисунок 2.1 - IDEF0 модель процесу доставки вантажу БПЛА

Далі розглянемо процес створення замовлення клієнтом (див. рисунок 2.2).

а) Увійшовши в систему, клієнт обирає зі списку вантаж що має бути йому доставлений;

б) Далі клієнт вводить дані про доставку: адресу, бажаний час доставки та фотографує своє обличчя для подальшої ідентифікації БПЛА.

в) Після цього він обирає точку на мапі, куди необхідно доставити вантаж;

г) Передостаннім кроком клієнт підтверджує введені раніше дані і процес доставки розпочинається;

д) Останнім кроком клієнт перенаправляється на сторінку, де він може відстежувати місцезнаходження свого вантажу.

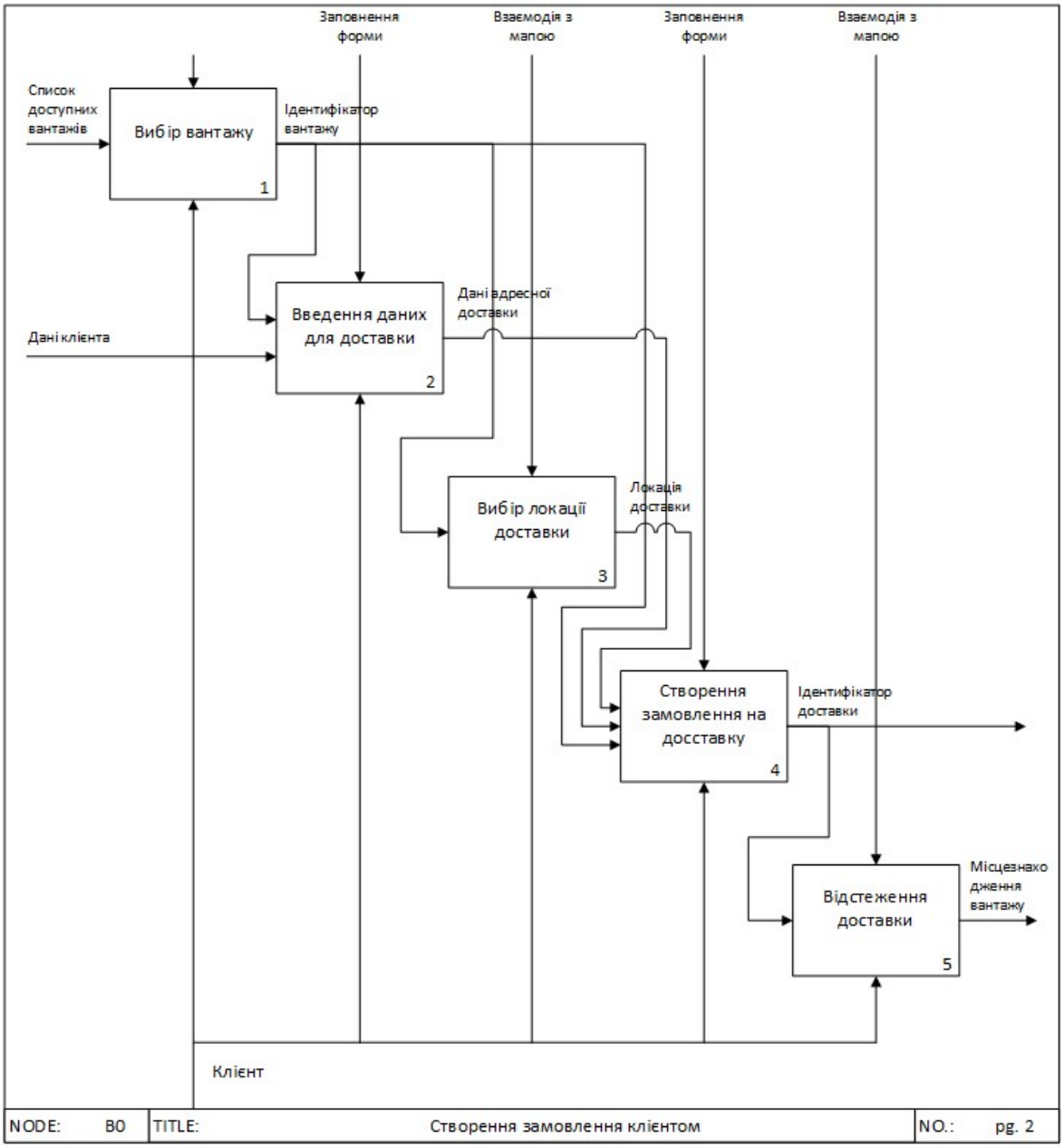


Рисунок 2.2- IDEF0 модель процесу створення замовлення клієнтом

2.2 Архітектура програмного забезпечення

Відповідно до сформульованих функціональних вимог до програмного забезпечення вантажного БПЛА, а також до бізнес-процесів, які дане ПЗ має реалізовувати, була розроблена архітектура, зображена на рисунку 2.3.

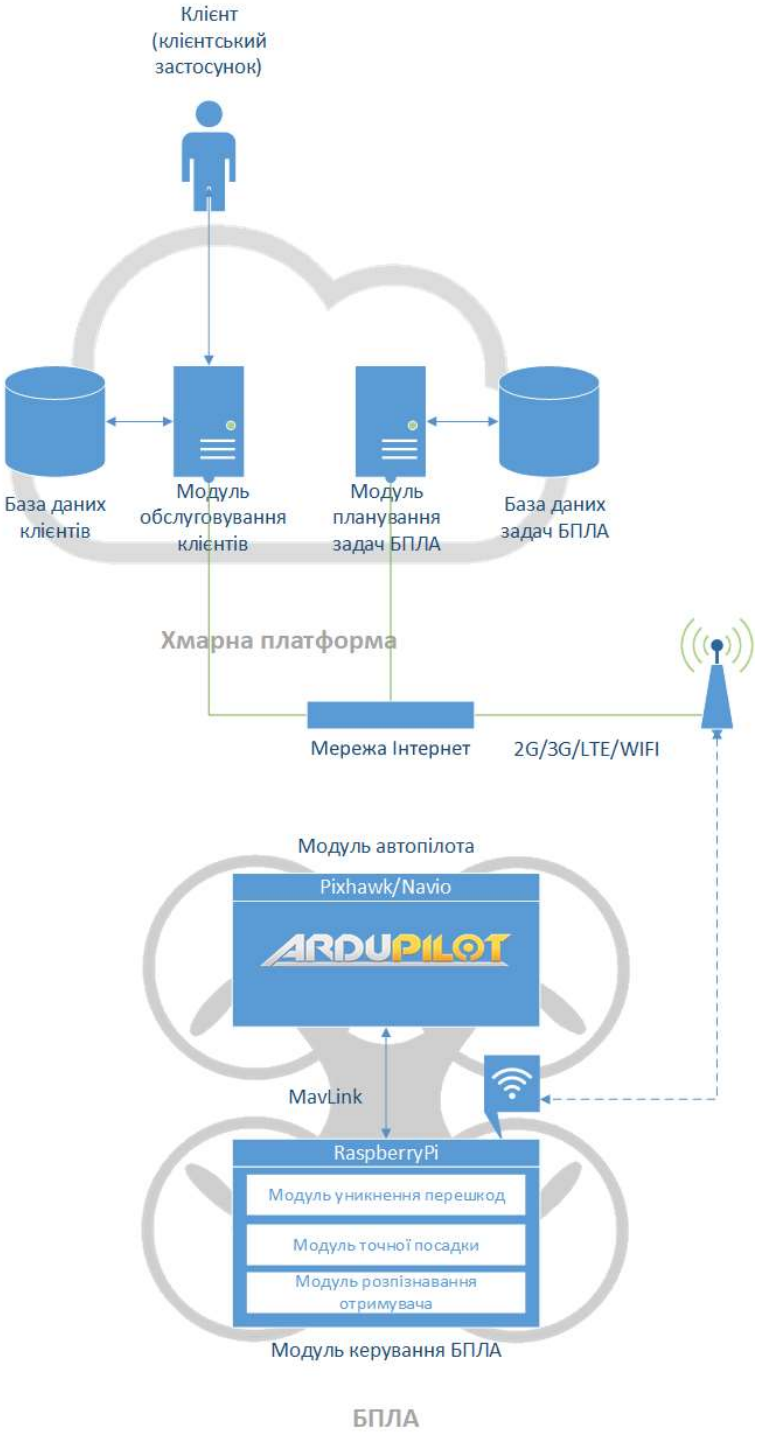


Рисунок 2.3 - Архітектура ПЗ екосистеми вантажного БПЛА

Запропонована архітектура програмного забезпечення екосистеми вантажного БПЛА складається з 4 окремих програмних модулів: з модуля обслуговування клієнтів, з модуля планування задач БПЛА, з модуля керування БПЛА та модуля автопілоту.

Призначенням модуля обслуговування клієнтів є взаємодія з клієнтом, відображення йому актуальної інформації та збереження даних пов'язаних з його замовленнями на доставку. Його основними функціональними задачами є наступні:

- зберігання даних клієнта;
- створення замовлень на доставку;
- відслідковування статусу замовлень та відображення його клієнту.

Модуль клієнта має власну базу даних для збереження даних користувача та замовлень, взаємодіє з клієнтом через користувацький інтерфейс.

Призначенням модуля планування задач БПЛА, в свою чергу, є постановка задач літальному апарату, передача йому всіх необхідних даних для їх виконання та контроль процесу виконання цих задач. Також саме цей модуль може містити функціонал адміністрування безпілотників (надавати певному користувачу з відповідними правами можливість змінювати параметри налаштування літальних апаратів). Основними функціональними задачами даного модулю є:

- вибір БПЛА для виконання замовлення;
- планування маршруту;
- передача команд модулю керування БПЛА та відслідковування їх виконання.

Даний модуль також має свою власну базу даних, де мають зберігатись дані про наявні безпілотники та їх вантажі.

Основна роль модуля керування БПЛА – це прийом задач від модуля планування задач та їх безпосереднє виконання шляхом обробки вхідних

даних, виконання відповідних алгоритмів та надсиланням команд до модуля автопілоту. Відповідно, його задачі:

- реалізація основних команд з доставки вантажів (забрати посилку, доставити посилку по маршруту, повернутись на базу);
- перетворення цих команд на послідовність простих команд для автопілоту;
- зчитування і обробка даних з додаткових датчиків.

Даний модуль також має модульну структуру всередині, де кожен модуль відповідає за свою власну, виокремлену з-поміж інших, функцію. А саме:

а) Модуль уникнення перешкод відповідає за зчитування даних з датчиків відстані, їх обробку, та посилання команд зі зміни курсу, якщо поряд надто близько була помічена перешкода.

б) Модуль точної посадки відповідає за приземлення безпілотника точно над вантажем, щоб була можливість його підхопити в автоматичному режимі.

в) Модуль розпізнавання отримувача має задачу з розпізнавання клієнта – отримувача посилки за допомогою камери на останньому етапі бізнес-процесу доставки вантажу.

Оскільки структура модуля керування БПЛА також модульна, то її можна легко розширювати і видозмінювати відповідно до задач, що ставляться перед літальним апаратом. Це суттєвий плюс, оскільки даний елемент програмного забезпечення не має чіткої прив’язки до сервісу із доставки вантажів. Його можна використовувати також і для інших задач.

Роль модуля автопілоту полягає здебільшого у переведенні простих управляючих команд, переданих модулем керування, у команди двигунам та забезпеченні стабільності польоту БПЛА. Список його задач:

- стабілізація під час польоту;
- ідентифікація власного положення в просторі;

- виконання простих польотних завдань (зліт, посадка, переміщення по GPS-координатах тощо).

Важливим елементом даної архітектури є спосіб обміну даними між модулями програмного забезпечення. Оскільки модулі обслуговування клієнтів та планування задач БПЛА мають бути доступними як для клієнтів, так і для БПЛА з будь-якої локації, то з цього випливає один єдиний можливий варіант розгортання даних програмних модулів - у хмарному середовищі. Для таких середовищ є характерним використання мережевих протоколів для обміну даними. Оскільки обмін даними цілком може бути двостороннім, то має застосовуватись протокол WS або WSS. Саме тому даний протокол і використовується для взаємодії програми обслуговування клієнтів з програмою планування задач БПЛА і взаємодії останньої з модулем керування безпілотника. Це забезпечує глобальну доступність даного сервісу.

Модуль керування БПЛА може взаємодіяти з модулем планування задач різними способами, проте найкращим рішенням є 2g/3g/LTE зв’язок, тому що він не накладає обмежень на радіус дії літальних апаратів, недорогий, швидкий та доступний майже в будь-якому населеному пункті.

Модулі керування БПЛА та автопілоту взаємодіють між собою за допомогою протоколу MavLink [11]. MavLink – це спеціалізований бінарний протокол для обміну даними між автопілотами та програмами, що ними керують. Він був вибраний, оскільки він гарно задокументований і розроблений спеціально для вирішення даної задачі, а також є дуже компактним, що безумовно є його перевагою, адже модулі автопілоту зазвичай не мають великої обчислювальної здатності.

2.3 Підхід до реалізації модулів програмного забезпечення

Оскільки метод побудови програмного забезпечення вантажного БПЛА має низку нефункціональних вимог, таких як доступність та простота впровадження, то для його формулювання недостатньо однієї лише

архітектури. Тому далі описані підходи до реалізації окремих модулів даного програмного забезпечення, щоб забезпечити повноту і доступність описаного в наступних розділах метода.

2.3.1 Модуль автопілоту

Почнемо з модуля автопілоту. Програма-автопілот – це складний програмний продукт, реалізація якого потребує глибоких профільних знань у фізиці, аеродинаміці і багатьох інших суміжних сферах. Також її реалізація потребує тисячі годин налагодження і тестування. Цей процес є дуже дорогим і є можливим лише для великих компаній. Але існують вже готові протестовані рішення з відкритим вихідним кодом, які задовольняють всі вимоги до вантажного БПЛА. Наразі існують декілька таких програмних рішень. Всі вони абсолютно безкоштовні, і кожна з них має свою спільноту ентузіастів та розробників, що готові допомогти з питаннями по впровадженню даного ПЗ в свою власну систему. Це суттєво спрощує процес впровадження і налагодження даного компоненту програмного забезпечення.

Вибір програми-автопілоту здійснювався з-поміж двох найпопулярніших рішень, через причину, зазначену вище. Це проект PX4 Autopilot [12] та проект Ardupilot [13]. Дані проекти досить схожі в своєму функціоналі і для детального порівняння цих двох рішень потрібні глибокі знання з авіоніки і чималий досвід в конструюванні безпілотників. Такий аналіз не входить в рамки даного дослідження. Саме тому, для визначення найкращого з цих програм автопілотування були досліджені спеціалізовані форуми розробників даного ПЗ. Виявилось, що Ardupilot має низку вагомих переваг над проектом PX4 Autopilot. Ось основні з них:

- Ardupilot здатен працювати на платформі Linux, що значно розширює його сферу можливого застосування. Саме ця перевага і є вирішальною, оскільки більшість бібліотек з розпізнавання образів, застосування яких буде описано згодом, написані на мові програмування Python, для запуску якої потрібна повноцінна операційна система.

- Ardupilot розробляється на 3 роки довше за PX4 – починаючи з 2009-го. З цього можна зробити опосередкований висновок, що даний проект є більш довершеним і має більшу спільноту що ним цікавиться і розробляє на ньому свої власні проекти.
- Даний проект дуже ретельно тестується, кожна нова зміна завжди перевіряється у реальному світі на літальних апаратах різних типів, для запобігання випадкових падінь. Команда розробників Ardupilot випускає оновлення кожні 1-6 місяців, тоді як команда розробників PX4 іноді випускає їх щодня.
- Ardupilot має набагато більш досконалу систему стабілізації літального апарату у реальному світі, враховуючи сильний вітер та інші погодні умови, тоді як PX4 тестується здебільшого у закритих приміщеннях або в гарну погоду.

Саме тому Ardupilot і був обраний в якості модуля автопілоту програмного забезпечення екосистеми вантажного БПЛА.

2.3.2 Модуль керування БПЛА

Як вже було описано, основна задача модуля керування БПЛА є прийняття команд від модуля планування задач по WS(s)-протоколу, та передача управляючих команд автопілоту по протоколу MavLink. Це обумовлює спосіб організації даного програмного забезпечення – як клієнтського застосунку, в якому в ролі серверної частини і виступає модуль планування задач. З’єднання із сервером модуль керування встановлює, коли у безпілотника вмикається живлення і цей модуль запускається. Це забезпечить модуль планування задач актуальними даними про безпілотники, доступні йому для управління.

Розглянемо окремо особливості реалізації кожного з модулів системи керування БПЛА.

Для реалізації модуля уникнення перешкод оптимальним є метод, запропонований в дослідженні під назвою «Obstacle alert and collision

avoidance system development for UAVs with Pixhawk flight controller» [14]. Даний метод був вибраний для даного дослідження, тому що він, на відміну від багатьох інших, використовує доступні ультразвукові датчики відстані. Це дозволяє суттєво зменшити загальну вартість БПЛА, що ніяк не впливає на наукову складову даного дослідження, але суттєве для тих хто буде використовувати запропонований метод побудови ПЗ. З-поміж інших переваг, зазначених в наведеному дослідженні, слід відмітити сумісність даного підходу з автопілотом Ardupilot і протоколом Mavlink, що вдало вписується у розроблену архітектуру програмного забезпечення екосистеми вантажного БПЛА.

Загальний алгоритм, наведений у вищезазначеному дослідженні, зображений на рисунку 2.4.

Рисунок 2.4 - Узагальнений алгоритм уникнення перешкод

Більш детально особливості даного методу будуть описані на етапі конструювання програмного забезпечення.

Метод реалізації модуля точної посадки запозичений з іншого дослідження, що має назву «Accurate Landing of Unmanned Aerial Vehicles Using Ground Pattern Recognition» [15]. Необхідність застосування даного методу обумовлена тим, що точність посадки за GPS-координатами, що реалізована в автопілоті Ardupilot, є недостатньою для того, щоб підібрати вантаж. Саме тому тут необхідна точність посадки в кілька сантиметрів. Даний метод використовує зчитування даних з камери БПЛА, спрямованої вертикально вниз, що ідентифікує розташовані на землі ARUCO-маркери різних розмірів (див. рисунок 2.5).

Рисунок 2.5 - Приклади ARUCO-маркерів різних розмірів

Перевагою такого методу є можливість застосування при посадці кількох маркерів різного розміру, що дозволяє ідентифікувати більші з них з більшої висоти і, водночас, зробити точність посадки високою, ідентифікуючи менші маркери, поступово знижуючись до місця посадки.

Суть самого методу доволі проста: алгоритм розпізнавання образів ідентифікує та розшифровує всі ARUCO-маркери на зображенні, отриманому з камери БПЛА. Наступним кроком алгоритм визначає позиціонування маркера відносно зображення (координати маркера, кут повороту та кут нахилу, див. рисунок 2.6).

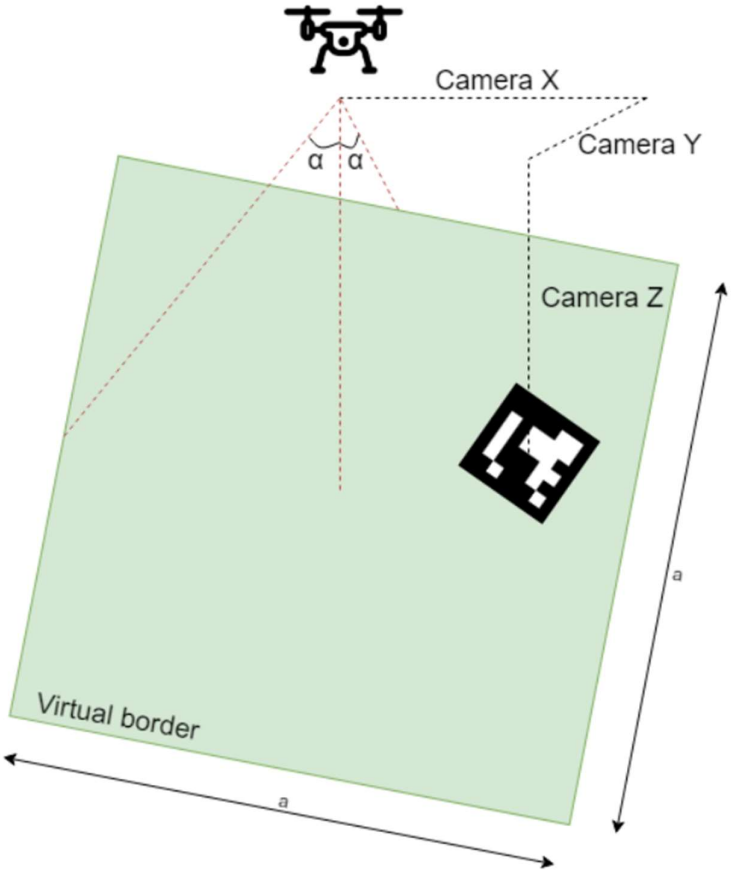


Рисунок 2.6 - Позиціонування ARUCO-маркера відносно зображення з камери БПЛА

Відповідно до отриманих даних, алгоритм дає команди на коригування положення літального апарату автопілоту. Якщо відхилення позиції маркера відносно центра зображення достатньо мале, алгоритм дає команду автопілоту на зниження на деяку висоту, і все починається з самого початку. Алгоритм зображений на рисунку 2.7.

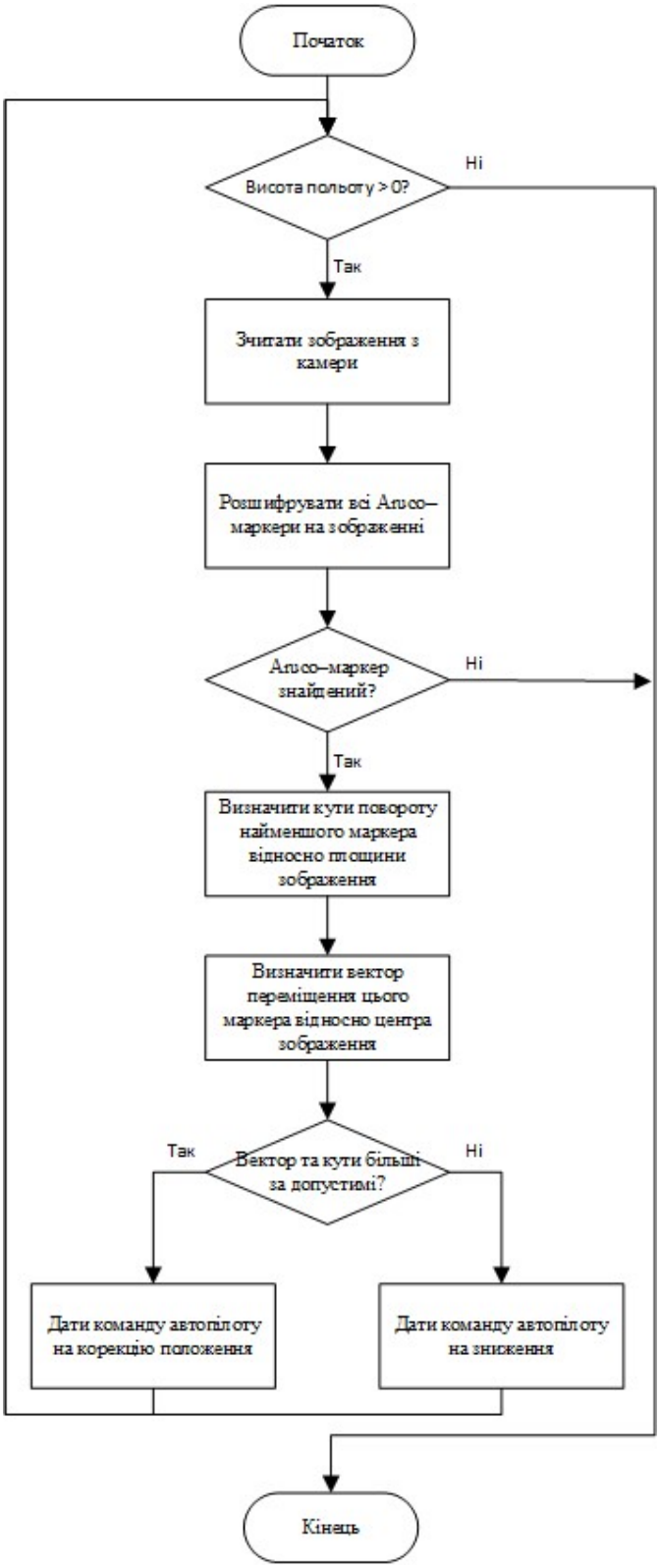


Рисунок 2.7 - Алгоритм точної посадки

Рисунок 2.8 - Зображення з борту БПЛА, отримане в процесі здійснення точної посадки

Реалізація останнього модуля програми керування БПЛА, модуля розпізнавання отримувача, базується на алгоритмі розпізнавання образів DeepFace [16]. Саме цей конкретний алгоритм був обраний для розпізнавання образів через декілька причин. По-перше він досить швидкий і, відповідно, не потребує великої обчислювальної потужності, що дуже важливо стосовно даного дослідження, оскільки цей алгоритм має здійснювати обчислення на борту БПЛА, де дана потужність є досить обмеженою. По-друге, бібліотека DeepFace є у відкритому доступі та має MIT-ліцензію, що дозволяє вільно використовувати її у таких дослідженнях як дане. Сам алгоритм зображений на рисунку 2.9.

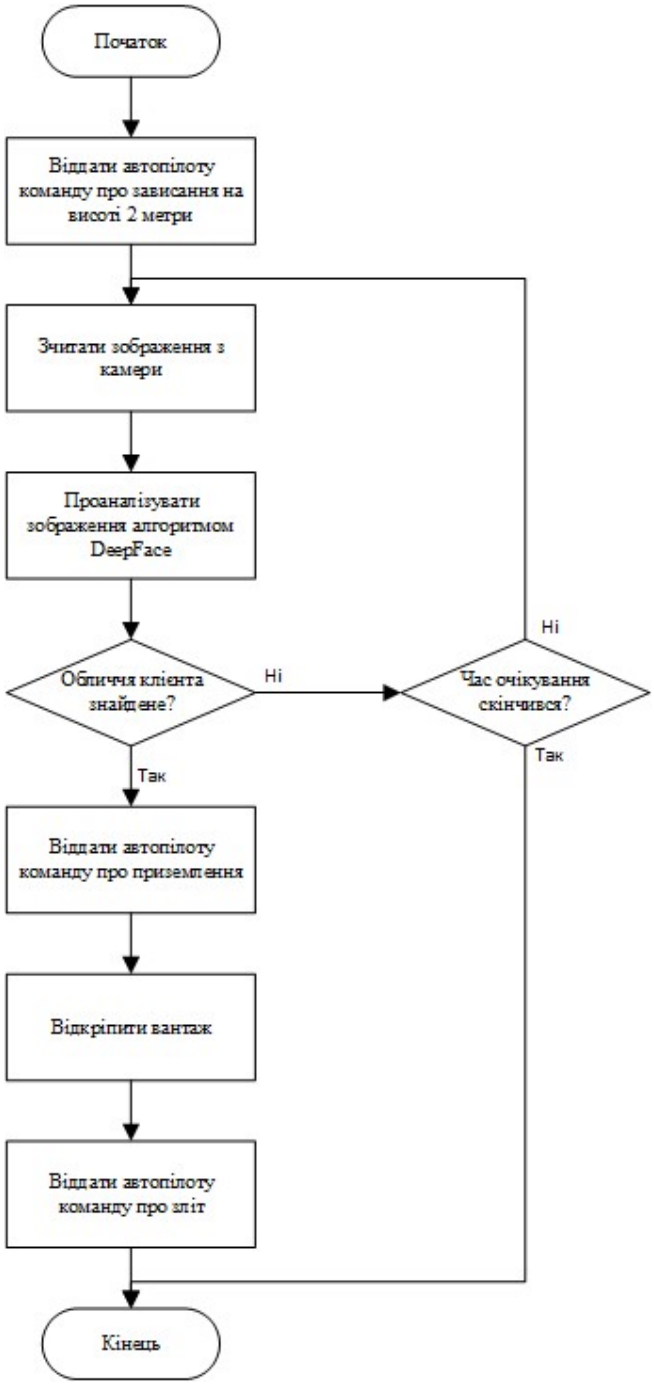


Рисунок 2.9 - Алгоритм роботи модулю розпізнавання отримувача

2.3.3 Модуль планування задач БПЛА

Модуль планування задач реалізується як звичайний серверний застосунок, тому особливих вимог до його реалізації немає. Єдиний нюанс – це те, що технологія, яка використовується при написанні даного модуля має підтримувати технологію WebSocket (протокол WS). Зараз більшість

високорівневих мов програмування мають утиліти для роботи з даним протоколом, але на це слід звернути увагу при її виборі.

2.3.4 Модуль обслуговування клієнтів

Модуль обслуговування клієнтів є звичайним клієнт-серверним застосунком з використанням HTTP(s)-протоколу обміну даними. Особливих умов для його реалізації також немає, єдиними вимогами до клієнтської частини є стандартні вимоги до користувацького інтерфейсу, до серверної – дотримання стандартів RESTful API.

2.4 Формулювання методу побудови програмного забезпечення

На даний момент вже сформульовані всі необхідні поняття та обрані всі необхідні підходи до розробки методів ПЗ щоб можна було сформулювати метод побудови програмного забезпечення екосистеми вантажного БПЛА. Модель бізнес-процесу даного методу представлена на рисунках 2.10 і 2.11. Розглянемо її детальніше.

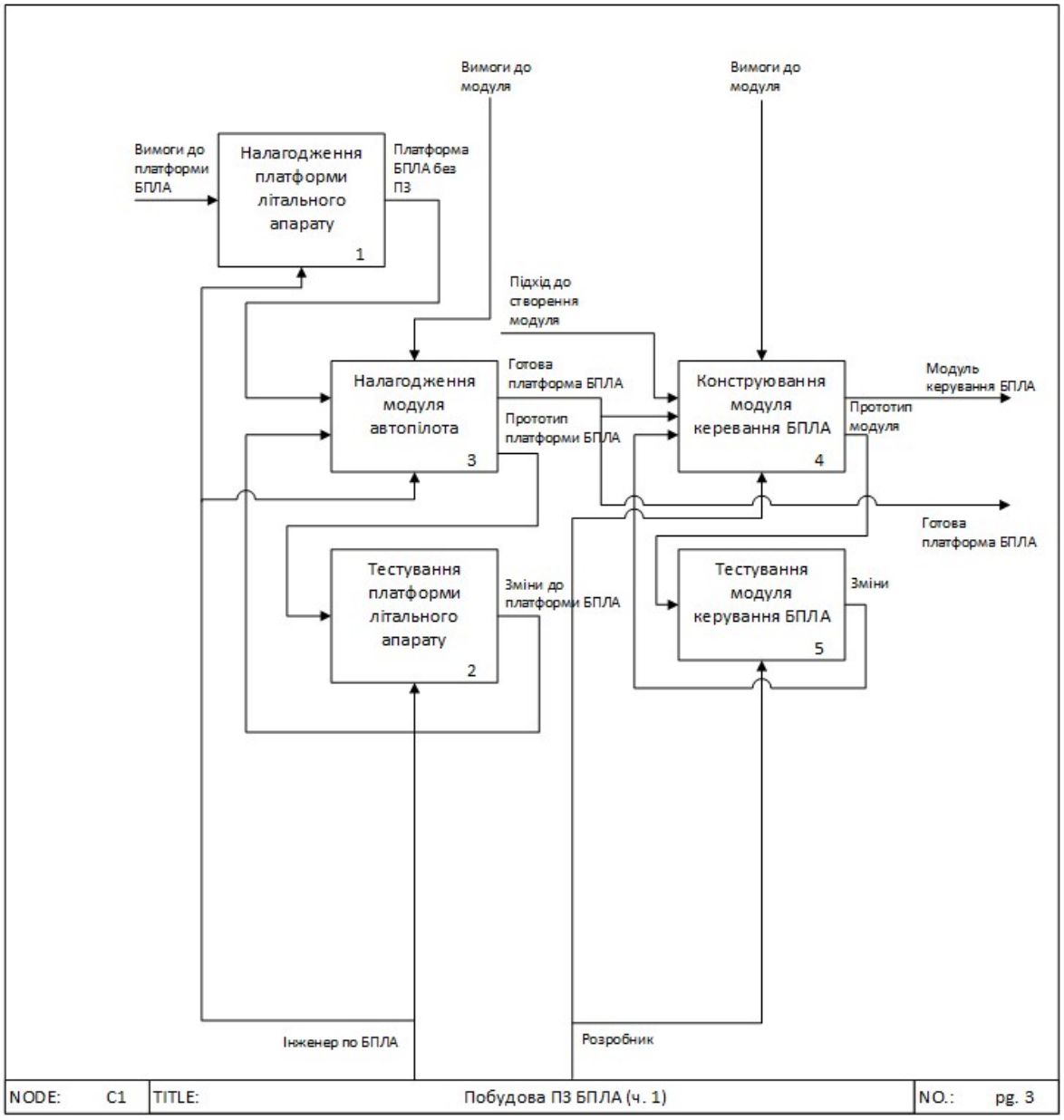


Рисунок 2.10 - IDEF0 модель процесу побудови ПЗ екосистеми вантажного БПЛА

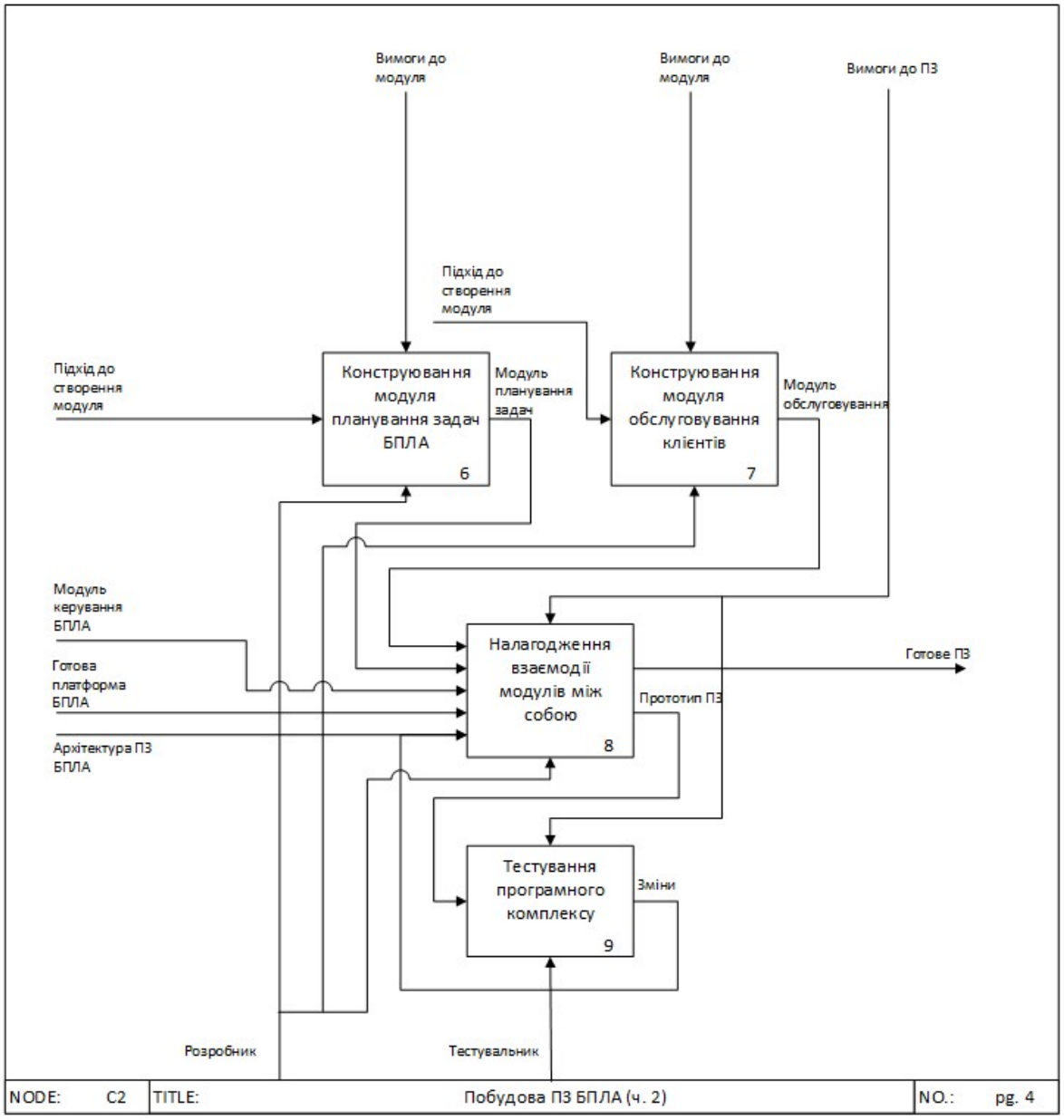


Рисунок 2.11 - IDEF0 модель процесу побудови ПЗ екосистеми вантажного БПЛА (продовження)

Першим етапом в побудові ПЗ БПЛА є налагодження платформи літального апарату. Даний процес включає в себе конструювання літального апарату відповідно до вимог до платформи БПЛА, та перевірка її функціональності. Вимоги до платформи безпілотного літального апарату є наступними:

- наявність системи двигунів та живлення на борту літального апарату, що надає можливості вертикального зльоту, посадки, утримання в повітрі та польоту в заданому напрямі та задовольняє вимогам ПЗ Ardupilot;
- наявність апаратного забезпечення, на яке можливо встановити ПЗ Ardupilot і яке здатне керувати двигунами;
- наявність на борту радіоприймача для керування літальним апаратом за допомогою радіокерування;
- наявність LTE та GPS модулів для геолокації та зв’язку з наземною станцією керування БПЛА;
- наявність 5-ти ультразвукових датчиків відстані (4 встановлені по боках і один знизу) для функціонування модуля уникнення перешкод;
- наявність апаратної платформи що підтримує встановлення OS Linux;
- наявність 2-х камер, встановлених по курсу і знизу для модуля розпізнавання отримувача і модуля точної посадки;
- наявність системи для захоплення і утримання вантажу.
- Результатом даного етапу має стати повністю зібрана і налагоджена платформа БПЛА.

Другим етапом побудови програмного забезпечення є налагодження модуля автопілоту. Він передбачає встановлення модуля автопілоту (Ardupilot) на відповідне програмне забезпечення та налаштування таким чином, щоб він був здатен керувати платформою літального апарату, сконструйованою на попередньому етапі. Результатом цього етапу є прототип платформи БПЛА з встановленим автопілотом, що має бути протестований на етапі 3. Тестування відбувається пілотом-оператором. Результатом етапів 2 і 3 має стати налагоджена польотна платформа.

Наступними етапами є конструювання модуля керування БПЛА та його тестування. Тестування даного модуля може здійснюватися за допомогою спеціально створених симуляторів для цього, але бажано проводити дане

тестування на створеній польотній платформі. Вхідними даними для етапу 4 є вимоги до даного модуля та розроблений підхід до його реалізації.

Для етапів конструювання модуля планування задач БПЛА та конструювання модуля обслуговування клієнтів наявність окремого етапу тестування не вимагається. Воно може мати місце але не є вирішальним, оскільки ризик відмови даних модулів не виражається у можливій втраті безпілотників і не є потенційно небезпечним, на відміну від попередніх етапів.

Останніми етапами є налагодження роботи всіх модулів програмного забезпечення екосистеми БПЛА разом та тестування даного програмного комплексу на відповідність вимогам до даного ПЗ. Результатом цих етапів має стати завершене програмне забезпечення що відповідає всім поставленим вимогам та необхідним стандартам.

2.5 Висновки до розділу

В даному розділі був сформульований метод побудови програмного забезпечення екосистеми вантажного БПЛА, змодельовані основні бізнес-процеси даного ПЗ, розроблена його архітектура та підходи до її реалізації. Запропонований підхід є кращим за підходи, запропоновані конкурентами, оскільки:

а) є найбільш повним, тому що розглядає ПЗ БПЛА з точки зору його екосистеми, а не окремих її елементів;

б) описаний найбільш детально;

в) спроектоване ПЗ є модульним, а тому розширюваним;

г) наявне окреме програмне забезпечення що взаємодіє з кінцевим користувачем;

д) радіус дії БПЛА в запропонованій архітектурі не залежить від розташування наземної станції керування, оскільки остання доступна для нього глобально через мережу Інтернет;

е) даний підхід є простим в своїй реалізації, оскільки використовує набір готових компонентів, бібліотек та алгоритмів, що були розроблені іншими людьми та чия ефективність була перевірена на практиці.

3 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЕКОСИСТЕМИ БПЛА

В даному розділі буде описаний приклад побудови програмного забезпечення екосистеми вантажного БПЛА відповідно до розробленого методу побудови даного програмного забезпечення для підтвердження його застосовності для поставленої задачі.

3.1 Налагодження модуля літального апарату.

Для того щоб задовільнити всі вимоги до платформи БПЛА, була зібрана апаратна платформа літального апарату, що складається з компонент, перелічених у таблиці 3.1.

Таблиця 3.1 - Перелік компонент для побудови БПЛА

Компонент	**Найменування**	**Призначення**	**Вартість**
Апаратна платформа для автопілоту	Navio2 kit	Набір датчиків та сенсорів для правильного функціонування програми-автопілоту Ardupilot	205$
Рама літального апарату	F550 Frame Full set	Рама літального апарату, що здатна вмістити на собі всі необхідні компоненти	35$
Апаратура радіо керування	Flysky FS T6 R6b	Пристрій радіо керування для тестування польотної платформи	52$

Продовження таблиці 3.1

Апаратна платформа для OS Linux	Raspberry PI3 B+	Linux платформа для встановлення та запуску модуля автопілоту та модуля керування БПЛА	43$
Двигуни	EMAX 2212 (6 шт.)	Двигуни для здійснення польоту апаратної платформи	57$
Регулятори швидкості	ESC 30A 2-4s (6 шт.)	Пристрої для регулювання кількості обертів для кожного двигуна	32$
Гвинти	8045 Props (10 шт.)	Гвинти для створення тяги на двигунах	7$
Акумуляторна батарея	4200 mAh 4s 60c	Батарея для живлення БПЛА	41$
Зарядний пристрій	iMAX B6	Зарядний пристрій для зарядки АКБ	24$
Модуль телеметрії	3DR 500mW 915 hz	Модуль телеметрії для зв'язку з наземною станцією для налагодження польотної платформи	21$
PPM Конвертер сигналу	Pixhawk PPM Encoder	Пристрій для конвертації сигналу радіоприймача в керуючий сигнал для автопілоту	7$

Продовження таблиці 3.1

Картка пам’яті	Kingston microsd 32GB	Картка пам’яті для запису образу ОС Linux для платформи Raspberry PI	4$
Роз’єми	XT60 (5 шт.)	Набір роз'ємів для з'єднання всіх елементів системи живлення БПЛА	4$
Стійка GPS	GPS mount	Стійка для кріплення GPS-антени на БПЛА	4$
Нижня камера БПЛА	Raspberry PI camera v2	Камера для точної посадки БПЛА	29$
Апаратна платформа для модуля уникнення перешкод	Arduino Nano	Апаратна платформа для встановлення модуля уникнення перешкод і під’єднання всіх датчиків	3$
Ультразвукові датчики відстані	HCSR-04 (5 шт.)	Датчики для визначення відстані до перешкод	5$
Фронтальна камера БПЛА	AX-1080p-v2 Camera	Камера для розпізнавання обличчя	37$
Серво-пристрій для вантажу	Servo claw	Пристрій для захоплення й утримання вантажу в польоті	22$

Сумарна вартість конструювання польотної платформи БПЛА з урахуванням витратних матеріалів склала близько 700$, що у 5-10 разів нижче за вартість готових подібних апаратів від відомих виробників.

При цьому, дана конструкція задовольняє всі вимоги до платформи БПЛА, які були перелічені в попередньому розділі і є повністю функціональної. Єдиним недоліком даної конструкції є те, що дана

платформа майже нездатна самостійно переміщувати вантажі, проте це не стосується програмної інженерії і цілком задовольняє задачу даного розділу – показати можливість реалізації програмного забезпечення екосистеми вантажного БПЛА за розробленим раніше методом.

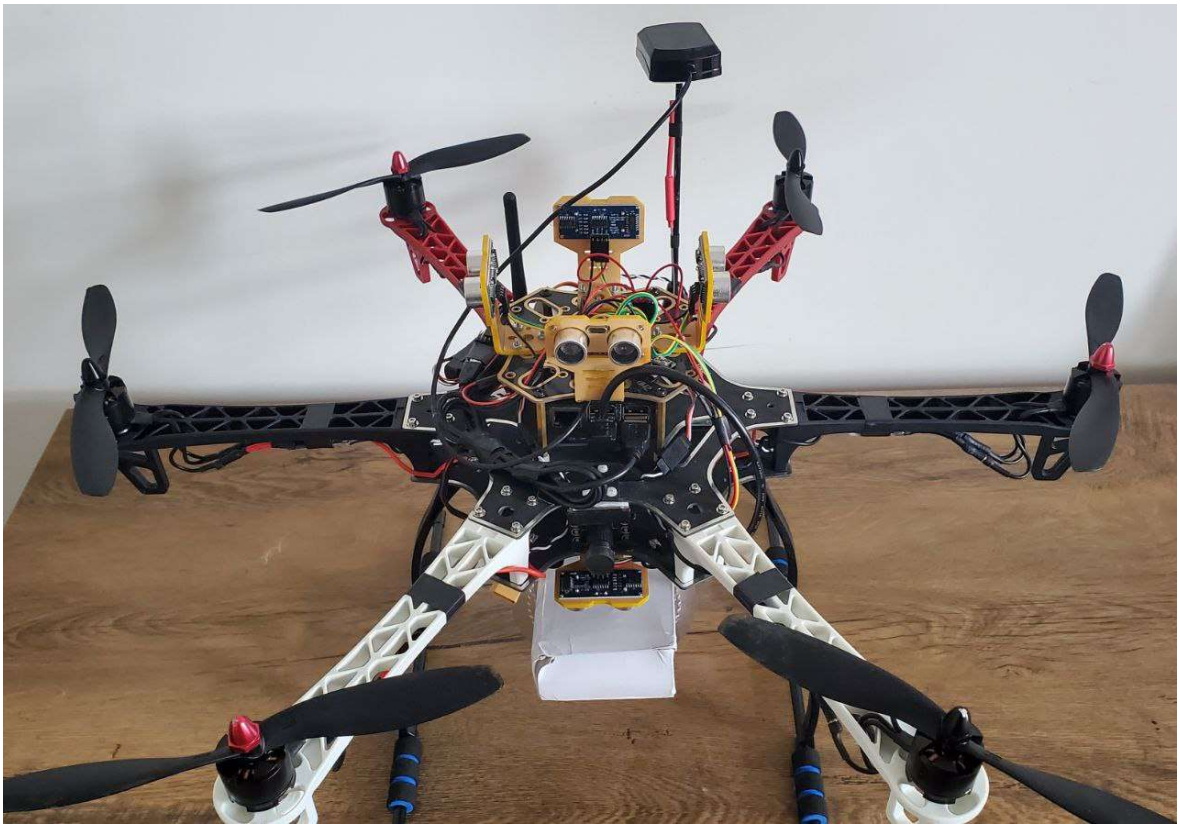


Рисунок 3.1 - Фото зібраної апаратної платформи вантажного БПЛА

3.2 Налагодження модуля автопілоту

Налагодження модуля автопілоту в рамках запропонованого методу полягає у встановленні програмного забезпечення Ardupilot на апаратну платформу та її налаштуванні таким чином, щоб вона була здатна до самостійного польоту під керуванням пілота-оператора.

Налаштування автопілоту Ardupilot даного апарату відбувалось за допомогою програмного забезпечення Mission Planner, що з’єднувався з модулем автопілоту по протоколу Mavlink. За допомогою даного ПЗ було проведене налаштування та калібрування літального апарату для нормального функціонування програми-автопілоту. Для коректного її

налаштування та функціонування, були відкалібровані датчики GPS, компасу та акселерометр. Також були задані важливі параметри кількості двигунів, їх розташування, потужності акумулятора таким чином, щоб автопілот міг самостійно піднімати БПЛА у повітря, утримувати його на місці та слідувати у наперед задані GPS-координати.

Рисунок 3.2 - Інтерфейс ПЗ Mission Planner

3.3 Тестування платформи літального апарату

Етап тестування платформи БПЛА здійснювався шляхом польотних випробувань літального апарату. Дані випробування здійснювались за допомогою пілота-оператора що управляв БПЛА.

Рисунок 3.3 - Платформа БПЛА в повітрі

Тестування і налагодження польотної платформи, як і очікувалось від методу побудови ПЗ, дали в результаті готову платформу БПЛА, що здатна самостійно переміщувати вантажі.

3.4 Конструювання модуля керування БПЛА

Під час проведення даного дослідження було вирішено трохи адаптувати розроблену раніше архітектуру БПЛА та винести модуль уникнення перешкод з модуля керування БПЛА в окрему складову архітектури. Таке рішення обумовлено тим, що обране апаратне забезпечення БПЛА не підходить для під’єднання одразу п’яти ультразвукових датчиків відстані. Тому даний модуль розроблявся окремо і розроблявся під платформу Arduino Nano.

3.4.1 Розробка модуля уникнення перешкод

Модуль уникнення перешкод розроблявся на мові С за алгоритмом, зображеним на рисунках 3.4 і 3.5. Загально алгоритм повністю повторює алгоритм, запропонований в методі розробки ПЗ екосистеми БПЛА, тільки даний алгоритм є більш точним, та спеціально підібраними параметрами, що підходять для даного розробленого літального апарату.

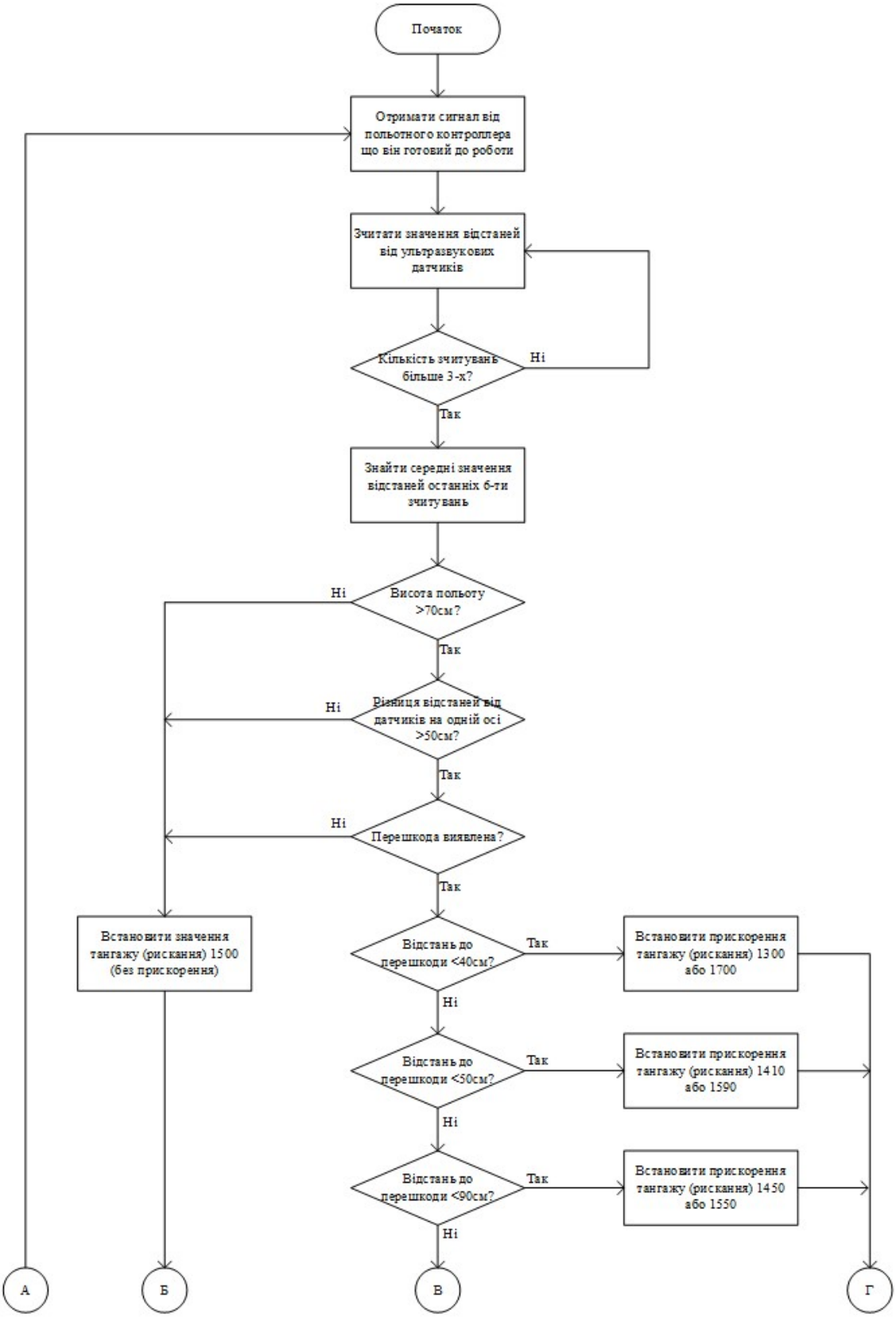


Рисунок 3.4 - Алгоритм уникнення перешкод

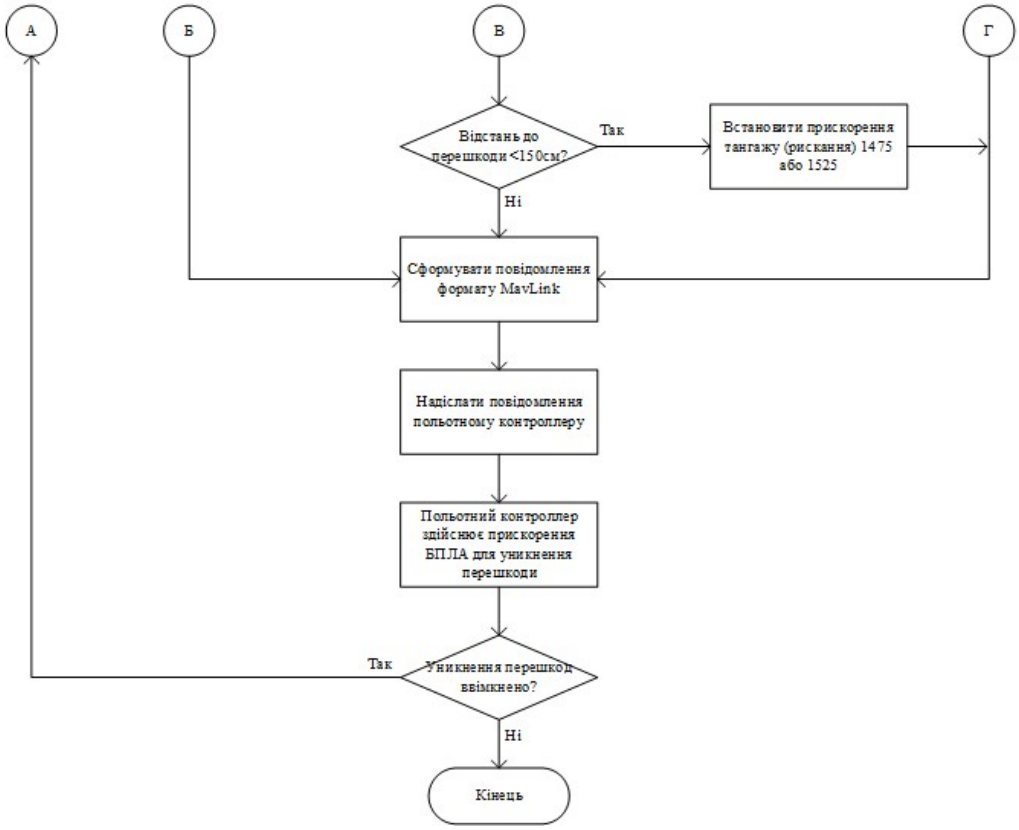


Рисунок 3.5 - Алгоритм уникнення перешкод (продовження)

Даний модуль має функціональну структуру, що відображена в таблиці 3.1.

Таблиця 3.2 - Функції модуля уникнення перешкод

Функція	**Опис**
setup	Встановлення з’єднання з автопілотом
loop	Цикл виконання програми
refreshCollisionDistances	Зчитування даних про відстань з датчиків
checkCollisionConditions	Перевірка умов можливого зіткнення
avoidCollisions	Уникнення перешкоди
calculatePitch	Підрахунок прискорення по осі тангажу для уникнення перешкоди

Продовження таблиці 3.2

calculateRoll	Підрахунок прискорення по осі рискання для уникнення перешкоди
calculateAcceleration	Підрахунок прискорення
compensateInertion	Компенсація прискорення, що було надане під час уникнення перешкоди
mavlinkSendHeartBeat	Відправка автопілоту команди про працездатність модулю
mavlinkSendRCOverride	Відправка автопілоту управляючої команди

3.4.2 Розробка модуля точної посадки

Алгоритм даного модуля повністю відповідає алгоритму, запропонованому в методі розробки ПЗ екосистеми БПЛА. Для розробки була використана мова програмування Python і модуль написаний в функціональному стилі. Далі в таблиці наведений опис основних функцій даного ПЗ.

Таблиця 3.3 - Функції модуля точної посадки

Функція	**Опис**
aruco_track	Отримати дані про місцезнаходження ARUCO-маркера на зображенні камери
calc_marker_angle	Обрахувати кут повороту БПЛА відносно ARUCO-маркера
calc_marker_location	Обрахувати зсув БПЛА відносно ARUCO-маркера
is_angle_valid	Перевірка, чи дозволяє кут повороту знижувати БПЛА, чи ні
move_vehicle_to	Переміщення літального апарату по координатах
check_ vehicle_landed	Перевірка, чи БПЛА приземлився
loop	Цикл виконання програми

3.4.3 Розробка модуля розпізнавання отримувача

Даний модуль також реалізований мовою програмування Python за раніше запропонованим алгоритмом, його основні функції наведені у таблиці 3.4.

Таблиця 3.4 - Функції модуля розпізнавання отримувача

Функція	**Опис**
get_image	Отримати дані з камери
compare_face	Порівняти обличчя з зображення камери із зображенням, що було завантажене в модуль розпізнавання отримувача
land	Посадка БПЛА на землю
release_cargo	Скидування вантажу
loop	Цикл виконання програми
go_home	Відправка команди про повернення БПЛА в точку вильоту

3.4.4 Організація роботи модулів

В програмі керування БПЛА принцип взаємодії модулів був обраний наступний: після надходження команди від системи обслуговування БПЛА, запускається відповідний програмний модуль. Після отримання результатів виконання відповідного модулю, програма керування пересилає результати системі обслуговування. Тобто, модулі запускаються у вигляді окремих підпрограм для виконання різних задач і не взаємодіють один з одним.

3.5 Тестування модуля керування БПЛА

Оскільки тестування модуля керування на реальному БПЛА може бути потенційно небезпечним і призвести як до руйнування самого літального апарату, так і до травм оточуючих, було прийнято рішення тестувати дане ПЗ

в програмі під назвою Gazebo Simulator [17]. Дане ПЗ призначене для симуляції поведінки різник літальних апаратів в попередньо створеному середовищі. Це дозволяє уникнути зайвих падінь при тестування ПЗ БПЛА. На рисунку 3.6 зображений фрагмент процесу тестування у даній програмі.

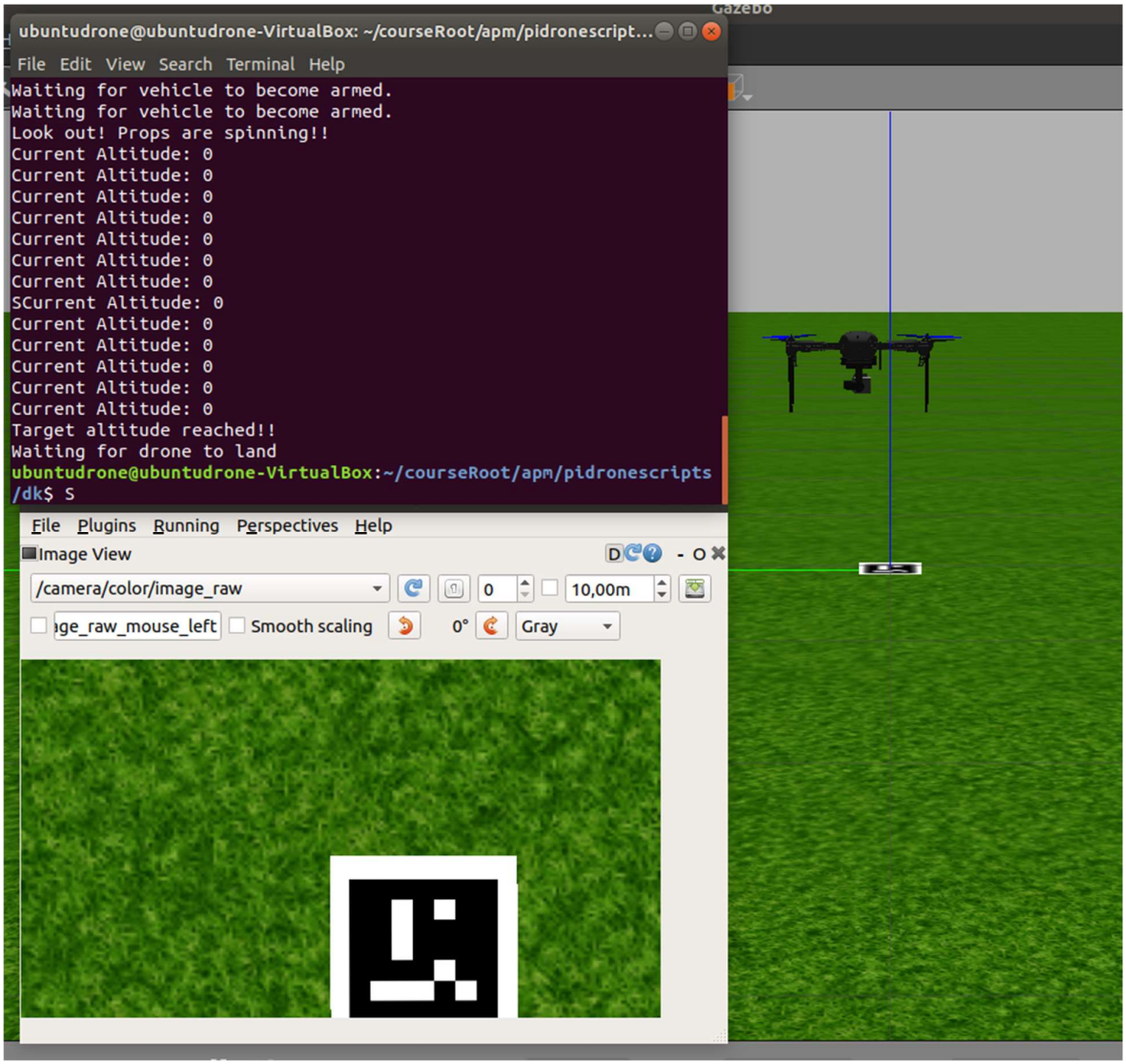


Рисунок 3.6 - Тестування функції точної посадки за допомогою Gazebo Simulator

3.6 Конструювання модулю планування задач БПЛА

Модуль планування задач БПЛА був сконструйований як серверний API-застосунок із використанням мови програмування Python та веб-фреймворка Django [18]. Оскільки постановка задач в даному модулі має

відбуватись асинхронно і очікування їх виконання від БПЛА може займати досить тривалий час, то була застосована спеціальна бібліотека для виконання таких задач.

Структура бази даних для даного застосунку має вигляд, зображений на рисунку 3.7.

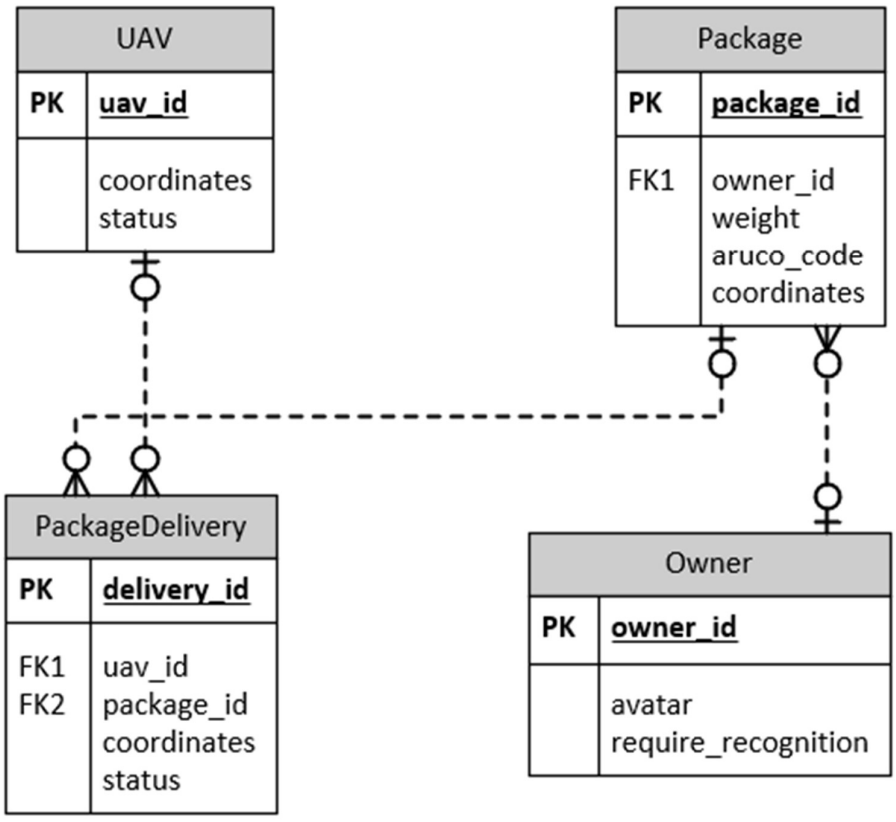


Рисунок 3.7 - Схема БД модуля планування задач БПЛА

API-ресурси модуля даного програмного забезпечення описані в таблиці 3.5.

Таблиця 3.5 - API модуля планування задач БПЛА

API-ресурс	**Призначення**
/api/deliveries	Додавання доставок модулем обслуговування клієнтів. На даному етапі ініціюється процес доставки вантажу модулем обслуговування БПЛА і розпочинається виконання бізнес-процесу даної доставки.
/api/owners	Додавання інформації модулем обслуговування клієнтів про отримувача посилки і сам вантаж
/api/uavs	Отримання інформації про доступні безпілотники для виконання замовлення
/api/deliveries/<delivery_id>/status	Отримання статусу виконання поточного замовлення

3.7 Конструювання модуля обслуговування клієнтів

Модуль обслуговування клієнтів БПЛА також реалізується як клієнт-серверний застосунок із використанням Python та Django. Нижче наведені діаграма бази даних для даного модуля та схема екранних форм та переходів між ними.

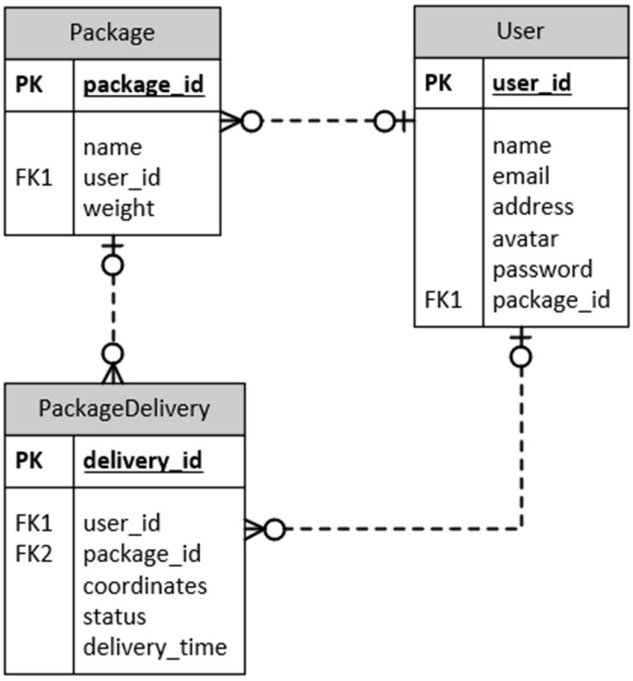


Рисунок 3.8 - Діаграма БД модуля обслуговування клієнтів

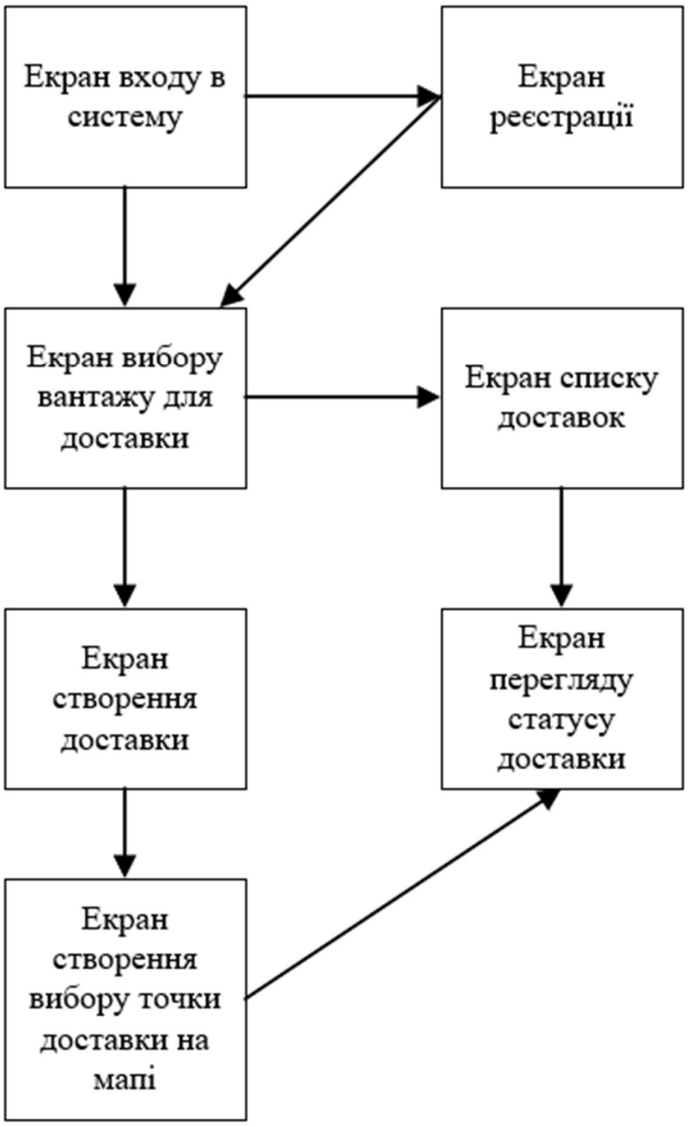


Рисунок 3.9 - Схема переходу між екранними формами модулю обслуговування клієнтів

3.8 Налагодження і тестування програмного комплексу

Налагодження і тестування програмного комплексу відбувається із застосуванням стандартних методик до тестування програмного забезпечення.

3.9 Висновки до розділу

В даному розділі був на конкретному прикладі описаний процес побудови програмного забезпечення екосистеми вантажного БПЛА із застосуванням методу запропонованого в розділі 2.

В цілому, процес розробки програмного забезпечення за запропонованим методом виявився простим і очевидним, результатом даного методу побудови ПЗ стало повноцінне і завершене програмне забезпечення для екосистеми вантажного БПЛА, що і було однією з цілей даного дослідження.

4 CASE STUDY

Для визначення застосовності методу побудови ПЗ екосистеми БПЛА необхідно провести його дослідження в реальних умовах, або в умовах, наближених до реальних. Оскільки проведення такого дослідження фізично не є можливим через низку причин (надмірна вартість і складність) для однієї людини, був застосований метод дослідження під назвою «тематичне дослідження» (або Case Study – англ.) [19].

Тематичне дослідження – це звіт про проблему, що містить реальну або гіпотетичну ситуацію, ціль якого – знайти спосіб вирішення проблем даної ситуації або довести ефективність вже запропонованого способу.

Оскільки реальне застосування та впровадження даного методу побудови ПЗ не є задачею даного дослідження, тому розберемо потенційну ефективність методу у наступних кейсах.

4.1 Кейс 1: Служба кур’єрської доставки

4.1.1 Опис ситуації

У місті-мільйоннику під назвою N працює служба із кур’єрської доставки ТОВ «Нові горизонти». Вона має у своєму розпорядженні 14 складів, що знаходяться приблизно на однаковому віддаленні один від одного і покривають однакові за площею області доставки. В кожному відділенні-складі працює по 10 людей, з них по 4 людини займаються обслуговуванням клієнтів, решта людей – кур’єри, що займаються доставкою посилок. Кожна з них доставляється окремо. Через численні затори в місті, середній час однієї доставки складає близько 2-х годин. Працівники компанії мають 8-годинний робочий день та зарплату у 300$/міс. Через низьку швидкість доставки компанія обслуговує малу кількість клієнтів і останнім часом взагалі не отримує прибутку. Всі вантажі, що доставляються кур’єрами мають малі габарити і вагу.

4.1.2 Задача до кейсу

Необхідно змоделювати заміну кур'єрів-людей на кур'єрів-безпілотників із застосуванням методу побудови ПЗ екосистеми БПЛА виходячи з умови, що 1 кур'єр замінюється на 1 безпілотник. Вважати собівартість перевезення вантажу людьми і безпілотниками рівними. Підрахувати вартість введення даного ПЗ в експлуатацію.

4.1.3 Питання до кейсу

- Через скільки місяців ТОВ «Нові горизонти» окупить вартість введення екосистеми БПЛА в експлуатацію?
- У скільки разів зміниться щоденна кількість доставлених посилок, враховуючи що безпілотнику на 1 доставку необхідно в середньому 30 хвилин?

4.1.4 Вирішення задачі

Спробуємо на даному прикладі застосувати метод побудови ПЗ БЛПА.

Спершу визначимо, яку кількість безпілотників обслуговуватиме дана система в цілому. Маємо 14 відділень і 6 кур'єрів-людей, кожного з яких замінюємо на безпілотника-кур'єра. Всього 84 безпілотника.

Визначимо, скільки ресурсів потрібно витратити на розробку даного програмного комплексу з нуля. У таблиці 4.1 наведена приблизна кількість годин, що має бути витрачена на кожен етап розробки даного ПЗ.

Таблиця 4.1 - Оцінка часу розробки ПЗ екосистеми БПЛА

Етап розробки ПЗ	**Час (години)**
Налагодження платформи літального апарату (на апарат)	40
Налагодження модуля автопілоту (на апарат)	20
Тестування платформи літального апарату (на апарат)	20
Конструювання модуля керування БПЛА	200
Тестування модуля керування БПЛА	100

Продовження таблиці 4.1

Конструювання модуля планування задач БПЛА	200
Конструювання модуля обслуговування клієнтів	200
Налагодження взаємодії модулів між собою	100
Тестування програмного комплексу	100

Як ми бачимо, на розробку програмного забезпечення має бути затрачено 900 людино-годин, а на налаштування однієї платформи БПЛА – 80 людино-годин. Для всіх апаратів ця цифра складає 6720 людино-годин. Оскільки метод побудови ПЗ БПЛА є простим у своїй реалізації і детально описаним, то припустимо що заробітна плата розробника складає 10$ за годину. Тоді вартість розробки ПЗ і налагодження всіх літальних апаратів складе 76200$.

Нехай вартість кожного літального апарату складатиме 2000$ (взята приблизна цифра), звідси отримаємо ціну на закупівлю платформ БПЛА у 164000$. Загальні витрати складають 240200$.

4.1.5 Відповіді на питання до кейсу

а) Через скільки місяців ТОВ «Нові горизонти» окупить вартість введення екосистеми БПЛА в експлуатацію?

Підрахуємо витрати компанії щомісяця до введення в експлуатацію екосистеми БПЛА. Маємо 14 відділень, 140 працівників і 300$ на зарплату кожному. Всього – 42000$/міс. Рештою витрат компанії нехтуємо.

Підрахуємо витрати компанії щомісяця після введення в експлуатацію екосистеми БПЛА. Кількість працівників у кожному відділенні складає 5 людей (6 кур’єрів замінюються одним оператором системи БПЛА). Всього на 14 відділень – 70 працівників із зарплатою у 21000$. Економія складає 21000$ щомісяця у порівнянні з ситуацією «до».

Виходить що вартість впровадження екосистеми БПЛА окупиться через 240200/21000 = 11.44 місяця.

б) У скільки разів зміниться щоденна кількість доставлених посилок, враховуючи що безпілотнику на 1 доставку необхідно в середньому 30 хвилин?

Оскільки кількість відділень і кур’єрів залишилась тою ж самою, то достатньо порівняти середній час доставки «до» і «після». Тому кількість доставлених посилок збільшиться у 4 рази.

4.2 Кейс 2: Компанія по боротьбі зі шкідниками

4.2.1 Опис ситуації

Компанія по боротьбі зі шкідниками займається обробкою полів і городів в складно доступних місцях, куди неможливо дістатись сільськогосподарською технікою, тому в даних цілях використовується ручна праця. Обробка полягає в обприскуванні посівів за допомогою аерозолю. Проблема полягає в непродуктивності такого підходу, оскільки ручна обробка займає велику кількість часу.

4.2.2 Питання до кейсу

- Чи можна застосувати метод побудови ПЗ екосистеми БПЛА в даному випадку?
- Які зміни необхідно внести в метод для його застосування в даній ситуації?
- Наскільки суттєво дані зміни відобразяться на складності впровадження даного ПЗ?

4.2.3 Відповіді на питання до кейсу

а) Чи можна застосувати метод побудови ПЗ екосистеми БПЛА в даному випадку?

В цілому розроблена архітектура ПЗ екосистеми БПЛА цілком може виконувати задачі боротьби зі шкідниками на полях, оскільки процес доставки вантажу не сильно відрізняється від даного. Не беручи до уваги

відмінності у конструкції літальних апаратів для цих задач, основна різниця полягає в модулі керування БПЛА, де стане непотрібним модуль розпізнавання отримувача, і в модулі обслуговування БПЛА, який буде містити трохи видозмінені завдання для самого безпілотника.

б) Які зміни необхідно внести в метод для його застосування в даній ситуації?

Процес обприскування полів для безпілотника можна розбити на наступні етапи:

- заправка безпілотника оприскувачем;
- приліт на місце обприскування;
- обприскування поля;
- повернення на базу.

Перший етап відповідає етапу підбору вантажу в оригінальному методі побудови ПЗ БПЛА і для нього використовуватиметься модуль точної посадки. Тут ніяких змін немає.

Другий етап аналогічно без змін.

Для третього етапу, обприскування поля, необхідно додати в модуль обслуговування БПЛА метод, який координати поля перетворюватиме на маршрут GPS-координат для безпілотника, який дозволить йому обробити поле по всій площі.

Останній етап також змін до ПЗ не потребує.

в) Наскільки суттєво дані зміни відобразяться на складності впровадження даного ПЗ?

Вищезазначені зміни є простими, і, завдяки модульній структурі спроектованого ПЗ, не потребують суттєвих зусиль. Найбільша різниця полягає в конструкції самого літального апарату і його налагодженні.

4.3 Використання інших методів побудови ПЗ БПЛА

Чи можна використовувати інші методи побудови ПЗ БПЛА для вирішення такої і подібних задач? Якщо брати до уваги переваги і недоліки

подібних методів побудови такого ПЗ, що були проаналізовані у попередніх розділах, то відповідь – однозначне ні, тому що вони не відповідають мінімальним вимогам ПЗ сервісу доставки, оскільки не мають інтерфейсу взаємодії із клієнтом та мають низку інших суттєвих мінусів, описаних у попередніх розділах, що значно ускладнюють їх впровадження.

4.4 Висновки до розділу

В даному розділі на прикладі двох кейсів була доведена практична корисність використання методу розробки ПЗ екосистеми вантажного БПЛА, а також його гнучкість в адаптації для інших практичних задач.

ВИСНОВКИ

Під час проведення даного наукового дослідження, була досліджена предметна область методів побудови програмного забезпечення БПЛА, були проаналізовані переваги та недоліки існуючих методів. Також було сформульоване нове поняття екосистеми БПЛА та був розроблений новий метод розробки ПЗ для неї. Даний метод був апробований на практиці, шляхом побудови програмного забезпечення з його використанням і, врешті решт, була доведена ефективність його використання на практиці.

Запропонований метод є абсолютно новим підходом до задачі побудови програмного забезпечення БПЛА що дозволяє робити це швидко та якісно і отримувати на виході гнучке, продумане та повноцінне програмне забезпечення екосистеми БПЛА.

ПЕРЕЛІК ПОСИЛАНЬ

1. Kai Petersen, Robert Feldt, Shahid Mujtaba, Michael Mattsson: Systematic Mapping Studies in Software Engineering [Електронний ресурс] – 2008. – Режим доступу до ресурсу: https://www.researchgate.net/publication/228350426_Systematic_Mapping_Studies_in_Software_Engineering.

2. Roel Wieringa, Nancy R. Mead, Colette Rolland, Neil A. M. Maiden: Requirements engineering paper classification and evaluation criteria: A proposal and a discussion [Електронний ресурс] – 2006. – Режим доступу до ресурсу: https://www.researchgate.net/publication/220428096_Requirements_engineering_paper_classification_and_evaluation_criteria_A_proposal_and_a_discussion.

3. Christian Fuchs, Clark Borst, Guido C. H. E. de Croon, M. M. (René) van Paassen and Max Mulder - An Ecological Approach to the Supervisory Control of UAV Swarms [Стаття] // Faculty of Aerospace Engineering, Delft University of Technology - 2014.

4. Gilles Albeaino, Masoud Gheisari, Bryan W. Franz - A Systematic Review Of Unmanned Aerial Vehicle Application Areas And Technologies In The Aec Domain [Стаття] // Journal of Information Technology in Construction - 2019.

5. Unmanned Aircraft Systems (UAS) [Стаття] // International Civil Aviation Organization.

6. Konstantinos Manikas and Klaus Marius Hansen - Software Ecosystems A Sytematic Literature Review [Стаття] // Department Of Computer Science (Diku) University Of Copenhagen - 2012.

7. European Commission Information Society and Media - Digital Business Ecosystems [Стаття] // Luxembourg: Office for Official Publications of the European Communities - 2007.

8. Ioannis Mademlis, Arturo Torres-González, Jesús Capitán, Rita Cunha, Bruno Guerreiro, Alberto Messina, Fulvio Negro, Cedric Le Barz, Tiago Gonçalves, Anastasios Tefas, Nikos Nikolaidis, Ioannis Pitas: A Multiple-UAV Software Architecture for Autonomous Media Production [Електронний ресурс] – 2019. – Режим доступу до ресурсу: https://novaresearch.unl.pt/en/publications/a-multiple-uav-software-architecture-for-autonomous-media-product.

9. Taygun Kekec, Baris Can Ustundag, Mehmet Ali Guney, Alper Yildirim, Mustafa Unel: A Modular Software Architecture for UAVs [Електронний ресурс] // Institute of Electrical and Electronics Engineers (IEEE) – 2013. – Режим доступу до ресурсу: https://core.ac.uk/display/19477579.

10. Enric Pastor, Juan Lopez and Pablo Royo: A hardware/software architecture for UAV payload and mission control [Електронний ресурс] // Institute of Electrical and Electronics Engineers (IEEE) – 2006. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/4106319.

11. MAVLink Developer Guide [Електронний ресурс] // MAVLink. – Режим доступу до ресурсу: https://mavlink.io/en/.

12. PX4 Autopilot [Електронний ресурс] – Режим доступу до ресурсу: https://px4.io/.

13. Ardupilot [Електронний ресурс] – Режим доступу до ресурсу: https://ardupilot.org/.

14. Crist´obal Cuevas Garc´ıa: Obstacle alert and collision avoidance system development for UAVs with Pixhawk flight controller [Електронний ресурс] // Core – 2018. – Режим доступу до ресурсу: https://core.ac.uk/download/pdf/288501541.pdf.

15. Accurate Landing of Unmanned Aerial Vehicles Using Ground Pattern Recognition [Електронний ресурс] // MDPI. – Режим доступу до ресурсу: https://www.mdpi.com/2079-9292/8/12/1532.

16. Deepface [Електронний ресурс] // GitHub. – Режим доступу до ресурсу: https://github.com/serengil/deepface.

17. Gazebo [Електронний ресурс] – Режим доступу до ресурсу: http://gazebosim.org/.

18. Django [Електронний ресурс] – Режим доступу до ресурсу: https://www.djangoproject.com/.

19. Writing a Case Study Report in Engineering [Електронний ресурс] // UNSW Sydney. – Режим доступу до ресурсу: https://student.unsw.edu.au/writing-case-study-report-engineering.

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

```
#include <NewPing.h>
#include "libraries/mavlink/common/mavlink.h"

/* Initialization of NewPing objects to access data of HC-SR04 ultrasonic sensors in Arduino */
#define MAX_SONAR_PING_DISTANCE 300 // cm
NewPing sonarLeft(2, 3, MAX_SONAR_PING_DISTANCE);
NewPing sonarRear(4, 5, MAX_SONAR_PING_DISTANCE);
NewPing sonarRight(6, 7, MAX_SONAR_PING_DISTANCE);
NewPing sonarFront(8, 9, MAX_SONAR_PING_DISTANCE);
NewPing sonarBottom(10, 11, MAX_SONAR_PING_DISTANCE);

#define DISTANCES_BUFFER_LENGTH              5
#define DISTANCE_MAX_THRESHOLD               100 // cm
#define ALTITUDE_MIN_THRESHOLD               5 // cm
#define DISTANCES_AXIS_DIFF_MIN_THRESHOLD    50 // cm
#define COMPENSATION_TIME                    800 // ms
#define RC_PITCH_TRIM_VALUE                  1500
#define RC_ROLL_TRIM_VALUE                   1500
#define SEND_RC_DELAY                        4 // loop() calls

#define COLLISION_DIRECTIONS_COUNT 5

struct CollisionDirection {
  uint16_t distancesBuffer[DISTANCES_BUFFER_LENGTH] = {0};
  uint16_t averageDistance                    = 0;
  bool needCompensation                       = false;
  bool compensationActive                     = false;
  unsigned long currentCompensationTime       = 0;
} leftCollisionDirection, rearCollisionDirection, rightCollisionDirection, frontCollisionDirection,
bottomCollisionDirection;

CollisionDirection* collisionDirections[] = {&leftCollisionDirection, &rearCollisionDirection,
&rightCollisionDirection, &frontCollisionDirection, &bottomCollisionDirection};


void setup() {
  // Communication speed setup
  Serial.begin(57600);
  // initialize digital pin LED_BUILTIN as an output.
```

```
  pinMode(LED_BUILTIN, OUTPUT);
}

const unsigned long REFRESH_INTERVAL = 1000; // ms
unsigned long lastRefreshTime = 0;
bool isLEDTurnedOn = false;

void loop() {
  if (millis() - lastRefreshTime >= REFRESH_INTERVAL) {
    lastRefreshTime = millis();
    mavlinkSendHeartBeat();

    if (isLEDTurnedOn) {
      digitalWrite(LED_BUILTIN, LOW);   // turn the LED off
      isLEDTurnedOn = false;
    } else {
      digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on
      isLEDTurnedOn = true;
    }
  }
  refreshCollisionDistances();
  checkCollisionConditions();
  avoidCollisions();
}

void refreshCollisionDistances() {
  for (uint8_t i = 0; i < COLLISION_DIRECTIONS_COUNT; i++) {
    for (uint8_t j = DISTANCES_BUFFER_LENGTH - 1; j > 0; j--) {
      collisionDirections[i] -> distancesBuffer[j] = collisionDirections[i] -> distancesBuffer[j - 1];
    }
  }

  leftCollisionDirection.distancesBuffer[0] = sonarLeft.ping_cm();
  rearCollisionDirection.distancesBuffer[0] = sonarRear.ping_cm();
  rightCollisionDirection.distancesBuffer[0] = sonarRight.ping_cm();
  frontCollisionDirection.distancesBuffer[0] = sonarFront.ping_cm();
  bottomCollisionDirection.distancesBuffer[0] = sonarBottom.ping_cm();

  for (uint8_t i = 0; i < COLLISION_DIRECTIONS_COUNT; i++) {
    int distanceBufferSum   = 0;
    uint8_t validDistancesCount = 0;
```

```
   for (uint8_t j = 0; j < DISTANCES_BUFFER_LENGTH; j++) {
    if (collisionDirections[i] -> distancesBuffer[j] != 0  && collisionDirections[i] -> distancesBuffer[j] <
MAX_SONAR_PING_DISTANCE) {
     distanceBufferSum += collisionDirections[i] -> distancesBuffer[j];
     validDistancesCount += 1;
    }
   }
   if (validDistancesCount > 3) {
    collisionDirections[i] -> averageDistance = distanceBufferSum / validDistancesCount;
   } else {
    collisionDirections[i] -> averageDistance = 0;
   }
  }

  // Serial.print("Distances: left=");
  // Serial.print(leftCollisionDirection.averageDistance);
  // Serial.print(", rear=");
  // Serial.print(rearCollisionDirection.averageDistance);
  // Serial.print(", right=");
  // Serial.print(rightCollisionDirection.averageDistance);
  // Serial.print(", front=");
  // Serial.print(frontCollisionDirection.averageDistance);
  // Serial.print(", bottom=");
  // Serial.print(bottomCollisionDirection.averageDistance);
  // Serial.print("cm\n\r");
}

void checkCollisionConditions() {
  for (uint8_t i = 0; i < COLLISION_DIRECTIONS_COUNT; i++) {
   if (collisionDirections[i] -> averageDistance != 0  && collisionDirections[i] -> averageDistance <
DISTANCE_MAX_THRESHOLD) {
    collisionDirections[i] -> needCompensation = true;
   } else {
    collisionDirections[i] -> needCompensation = false;
   }
  }
}

uint16_t appliedPitch = 0;
uint16_t appliedRoll = 0;

void avoidCollisions() {
```

```
  uint16_t pitch = calculatePitch();
  uint16_t roll  = calculateRoll();

  compensateInertion(pitch, roll, appliedPitch, appliedRoll);

  if (pitch != appliedPitch || roll != appliedRoll) {
     appliedPitch = pitch;
     appliedRoll = roll;
     mavlinkSendRCOverride(appliedPitch, appliedRoll);
  }
}

uint16_t calculatePitch() {
  int16_t distancesLeftRightDiff = abs(frontCollisionDirection.averageDistance -
rearCollisionDirection.averageDistance);

  if (bottomCollisionDirection.averageDistance > ALTITUDE_MIN_THRESHOLD ||
bottomCollisionDirection.averageDistance == 0 ) {
    if (distancesLeftRightDiff > DISTANCES_AXIS_DIFF_MIN_THRESHOLD) {
      if (frontCollisionDirection.needCompensation == true && rearCollisionDirection.needCompensation == true) {
        if (frontCollisionDirection.averageDistance < rearCollisionDirection.averageDistance) {
          // The front sensor has a smaller distance
          return calculateAcceleration(frontCollisionDirection.averageDistance, true, RC_PITCH_TRIM_VALUE);
        } else {
          // The rear sensor has a smaller distance
          return calculateAcceleration(rearCollisionDirection.averageDistance, false, RC_PITCH_TRIM_VALUE);
        }
      } else if (frontCollisionDirection.needCompensation == true && rearCollisionDirection.needCompensation ==
false) {
        // Detects the front, but not the rear
        return calculateAcceleration(frontCollisionDirection.averageDistance, true, RC_PITCH_TRIM_VALUE);
      } else if (frontCollisionDirection.needCompensation == false && rearCollisionDirection.needCompensation ==
true) {
        // Detects the rear, but not the front
        return calculateAcceleration(rearCollisionDirection.averageDistance, false, RC_PITCH_TRIM_VALUE);
      }
    } else if (frontCollisionDirection.needCompensation == true && rearCollisionDirection.averageDistance == 0) {
      // Rear distance too large to track and returns 0
      return calculateAcceleration(frontCollisionDirection.averageDistance, true, RC_PITCH_TRIM_VALUE);
    } else if (frontCollisionDirection.averageDistance == 0 && rearCollisionDirection.needCompensation == true) {
      // Front distance too large to track and returns 0
      return calculateAcceleration(rearCollisionDirection.averageDistance, false, RC_PITCH_TRIM_VALUE);
```

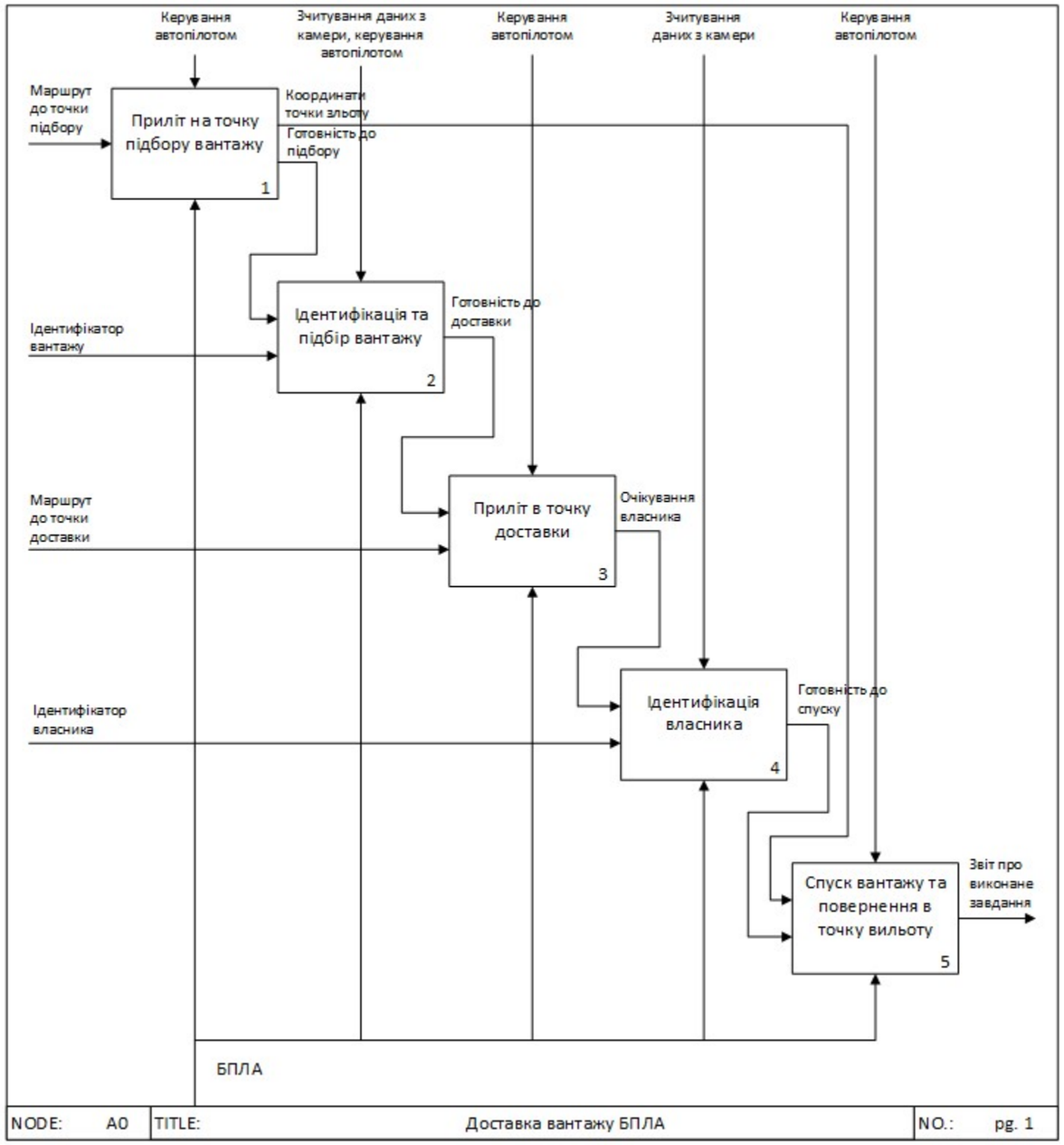
```
    }
  }
  return 0;
}

uint16_t calculateRoll() {
  int16_t distancesLeftRightDiff = abs(leftCollisionDirection.averageDistance -
rightCollisionDirection.averageDistance);

  if (bottomCollisionDirection.averageDistance > ALTITUDE_MIN_THRESHOLD ||
bottomCollisionDirection.averageDistance == 0 ) {
    if (distancesLeftRightDiff > DISTANCES_AXIS_DIFF_MIN_THRESHOLD) {
      if (leftCollisionDirection.needCompensation == true && rightCollisionDirection.needCompensation == true) {
        if (leftCollisionDirection.averageDistance < rightCollisionDirection.averageDistance) {
          // The left sensor has a smaller distance
          return calculateAcceleration(leftCollisionDirection.averageDistance, true, RC_ROLL_TRIM_VALUE);
        } else {
          // The right sensor has a smaller distance
          return calculateAcceleration(rightCollisionDirection.averageDistance, false, RC_ROLL_TRIM_VALUE);
        }
      } else if (leftCollisionDirection.needCompensation == true && rightCollisionDirection.needCompensation ==
false) {
        // Detects the left, but not the right
        return calculateAcceleration(leftCollisionDirection.averageDistance, true, RC_ROLL_TRIM_VALUE);
      } else if (leftCollisionDirection.needCompensation == false && rightCollisionDirection.needCompensation ==
true) {
        // Detects the right, but not the left
        return calculateAcceleration(rightCollisionDirection.averageDistance, false, RC_ROLL_TRIM_VALUE);
      }
    } else if (leftCollisionDirection.needCompensation == true && rightCollisionDirection.averageDistance == 0) {
      // Right distance too large to track and returns 0
      return calculateAcceleration(leftCollisionDirection.averageDistance, true, RC_ROLL_TRIM_VALUE);
    } else if (leftCollisionDirection.averageDistance == 0 && rightCollisionDirection.needCompensation == true) {
      // Left distance too large to track and returns 0
      return calculateAcceleration(rightCollisionDirection.averageDistance, false, RC_ROLL_TRIM_VALUE);
    }
  }
  return 0;
}

uint16_t calculateAcceleration(uint16_t distance, bool increase, uint16_t channelTrimValue) {
  uint16_t acceleration = 0;
```

```
  if (distance < 30) {
    acceleration = 200;
  } else if (distance < 90) {
    acceleration = 175;
  } else if (distance < 150) {
    acceleration = 150;
  }

  if (increase == true) {
    return channelTrimValue + acceleration;
  } else {
    return channelTrimValue - acceleration;
  }
}

void compensateInertion(uint16_t &pitch, uint16_t &roll, uint16_t &appliedPitch, uint16_t &appliedRoll) {
  if (appliedPitch > 1500 && frontCollisionDirection.compensationActive == false &&
rearCollisionDirection.compensationActive == false) {
    frontCollisionDirection.compensationActive = true;
  } else if (appliedPitch < 1500 && appliedPitch != 0 && rearCollisionDirection.compensationActive == false &&
frontCollisionDirection.compensationActive == false) {
    rearCollisionDirection.compensationActive = true;
  } else if (appliedPitch == 0 && frontCollisionDirection.compensationActive == true &&
frontCollisionDirection.currentCompensationTime == 0) {
    frontCollisionDirection.currentCompensationTime = millis();
  } else if (appliedPitch == 0 && rearCollisionDirection.compensationActive == true &&
rearCollisionDirection.currentCompensationTime == 0) {
    rearCollisionDirection.currentCompensationTime = millis();
  }

  if (appliedRoll > 1500 && leftCollisionDirection.compensationActive == false &&
rightCollisionDirection.compensationActive == false) {
    leftCollisionDirection.compensationActive = true;
  } else if (appliedRoll < 1500 && appliedRoll != 0 && rightCollisionDirection.compensationActive == false &&
leftCollisionDirection.compensationActive == false) {
    rightCollisionDirection.compensationActive = true;
  } else if (appliedRoll == 0 && rightCollisionDirection.compensationActive == true &&
rightCollisionDirection.currentCompensationTime == 0) {
    rightCollisionDirection.currentCompensationTime = millis();
  } else if (appliedRoll == 0 && leftCollisionDirection.compensationActive == true &&
leftCollisionDirection.currentCompensationTime == 0) {
```

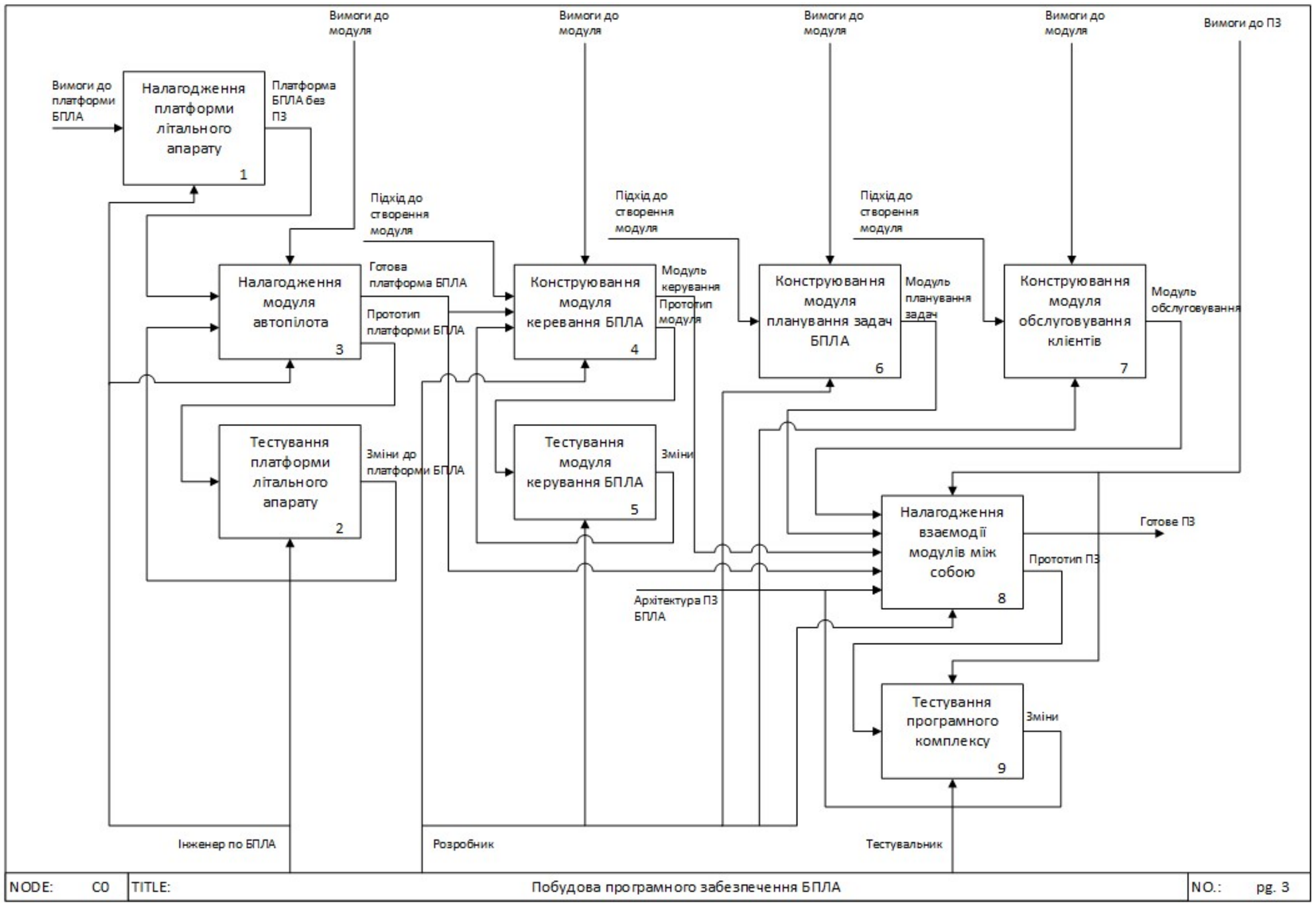
```
    leftCollisionDirection.currentCompensationTime = millis();
   }

   if (frontCollisionDirection.currentCompensationTime != 0) {
    if (frontCollisionDirection.currentCompensationTime + COMPENSATION_TIME > millis()) {
     pitch = RC_PITCH_TRIM_VALUE - 200;
    } else {
     appliedPitch = 0;
     frontCollisionDirection.compensationActive = false;
     frontCollisionDirection.currentCompensationTime = 0;
    }
   } else if (rightCollisionDirection.currentCompensationTime != 0) {
    if (rightCollisionDirection.currentCompensationTime + COMPENSATION_TIME > millis()) {
     roll = RC_ROLL_TRIM_VALUE + 200;
    } else {
     appliedRoll = 0;
     rightCollisionDirection.compensationActive = false;
     rightCollisionDirection.currentCompensationTime = 0;
    }
   } else if (rearCollisionDirection.currentCompensationTime != 0) {
    if (rearCollisionDirection.currentCompensationTime + COMPENSATION_TIME > millis()) {
     pitch = RC_PITCH_TRIM_VALUE + 200;
    } else {
     appliedPitch = 0;
     rearCollisionDirection.compensationActive = false;
     rearCollisionDirection.currentCompensationTime = 0;
    }
   } else if (leftCollisionDirection.currentCompensationTime != 0) {
    if (leftCollisionDirection.currentCompensationTime + COMPENSATION_TIME > millis()) {
     roll = RC_ROLL_TRIM_VALUE - 200;
    } else {
     appliedRoll = 0;
     leftCollisionDirection.compensationActive = false;
     leftCollisionDirection.currentCompensationTime = 0;
    }
   }
  }

  /* AUTOPILOT CONFIGURATION */
  uint8_t    system_id = 255;
  uint8_t    component_id = 0;
```

```
uint8_t    ap_base_mode = 0;
uint32_t   ap_custom_mode = 1;
uint8_t    ap_system_status = 0;

uint8_t target_system = 1;
uint8_t target_component = 0;

// Send heartbeat message to the flight controller
void mavlinkSendHeartBeat()
{
        mavlink_message_t msg;
        uint8_t buf[MAVLINK_MAX_PACKET_LEN];

        mavlink_msg_heartbeat_pack(system_id, component_id, &msg, MAV_TYPE_HEXAROTOR, MAV_AUTOPILOT_GENERIC, ap_base_mode, ap_custom_mode, ap_system_status);

        uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
        Serial.write(buf, len);
}

// Send acceleration command to the flight controller
void mavlinkSendRCOverride(uint16_t Pitch, uint16_t Roll) {
  mavlink_message_t msg;
        uint8_t buf[MAVLINK_MAX_PACKET_LEN];

  mavlink_msg_rc_channels_override_pack(system_id, component_id, &msg, target_system, target_component, Roll, Pitch, 0, 0, 0, 0, 0, 0);

  uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
  Serial.write(buf, len);

  Serial.print("Pitch: ");
  Serial.print(Pitch);
  Serial.print(",");
  Serial.print(" Roll: ");
  Serial.print(Roll);
  Serial.print("\n\r");
}
```

ДОДАТОК Б БІЗНЕС-ПРОЦЕС ДОСТАВКИ ВАНТАЖУ АКТОРОМ БЛПА



ДОДАТОК В МЕТОД ПОБУДОВИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



ДОДАТОК Г АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ