НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри

_____ Сергій СТІРЕНКО
(підпис)

“__” _____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “ Комп’ютерні системи та мережі”
спеціальності 123 “ Комп’ютерна інженерія”

на тему: Модуль для організації зв'язку machine-to-machine для БЛА

Виконав : студент __4__ курсу, групи _ІВ-93_
(шифр групи)

_____Рустамов Арсен Сергійович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____доцент, к.т.н., Роковий О.П.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ст. викл. Виноградов Ю.М.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____доцент, к.т.н., Коган А.В._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“ Комп’ютерні системи та мережі ”

спеціальності 123 “ Комп’ютерна інженерія ”

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Сергій СТІРЕНКО
(підпис)

“__” _____2023 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Рустамова Арсена Сергійовича

1. Тема проєкту *Модуль для організації зв'язку machine-to-machine для БЛА*
керівник проєкту Роковий О.П., доцент, к.т.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 31 травня 2023 року №2102-с

2. Термін здачі студентом закінченого проєкту *22 червня 2023 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані*.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Розділ 1. Огляд існуючих архітектур комунікацій та зв’яку між БЛА.

Розділ 2. Налагодження віртуального середовища для тестування модуля для БЛА.

Розділ 3. Реалізація комунікації для рою БЛА.

Розділ 4. Розробка та тестування протоколу.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Виноградов Ю.М.		

7. Дата видачі завдання «15» грудня 2022 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*02.02.2023-03.02.2023*	
2.	*Вивчення та аналіз завдання*	*05.02.2023-15.03.2023*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2023-25.03.2023*	
4.	*Розробка структур окремих підсистем*	*25.03.2023-5.04.2023*	
5.	*Програмна реалізація системи*	*05.04.2023-15.04.2023*	
6.	*Оформлення пояснювальної записки*	*15.04.2023-12.05.2023*	
7.	*Захист програмного продукту*	*25.04.2023*	
8.	*Передзахист*	*23.05.2023*	
9.	*Захист*	*22.06.2023*	

Студент-дипломник _____ Рустамов Арсен
(підпис)

Керівник проєкту _____ Олександр Роковий
(підпис)

АНОТАЦІЯ

У даній роботі було детально розглянуто способи розробки програмного забезпечення для БЛА. Були розглянуті існуюча схема координації БЛА за допомогою наземної станції, плюси та мінуси данної схеми. За результатами досліджень була побудована схема зв’язку між БЛА без використання наземної станції. Данна схема була запротатипована за допомогою офіційної бібліотекі симуляції БЛА SITL. Програмний продукт був розроблений на мові Lua.

Ключові слова: БЛА, бездротовий зв’язок, Ardupilot, SITL.

ANNOTATION

In this paper, the developing of existing scheme for coordinating UAVs with the help of a ground station was reviewed. The pros and cons of this scheme were considered. Based on the results of the research, a scheme of communication between UAVs without the use of a ground station was built. This scheme was tested using the official UAV simulation library SITL. The software product was developed in the Lua language.

Keywords: UAV, wireless communication, Ardupilot, SITL.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467100.002 ТЗ***	Модуль для організації	3		
			зв’язку machine-to-machine			
			оптимізації для БЛА			
			Технічне завдання			
	А4	***ІАЛЦ.467100.003 ПЗ***	Модуль для організації	57		
			зв’язку machine-to-machine			
			оптимізації для БЛА			
			Пояснювальна записка			
	А4	***ІАЛЦ.467100.004 Д1***	Модуль для організації	1		
			зв’язку machine-to-machine			
			оптимізації для БЛА			
			Основні компоненти			
			модуля (Структурна схема)			
	А4	***ІАЛЦ.467100.005 Д2***	Модуль для організації	1		
			зв’язку machine-to-machine			
			оптимізації для БЛА			
			Алгоритм протоколу			
			(Функціональна схема)			
	А4	***ІАЛЦ.467100.006 Д3***	Модуль для організації	1		
			зв’язку machine-to-machine			
			оптимізації для БЛА			
			Алгоритм дій програмного			
			забезпечення (Принципова			
			схема)			
	А4	***ІАЛЦ.467100.007 Д4***	Модуль для організації	4		
			зв’язку machine-to-machine			
			оптимізації для БЛА			
			Текст програмного коду			

					ІАЛЦ.467100.001 ОА
Зм	***Лист***	***№ докум.***	***Підп***	***Дата***	
Розроб		Рустамов А.С.			*Модуль для організації зв'язку machine-to-machine для БЛА* Опис альбому
Перев		Роковий О.П.			

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. Ігоря Сікорського ФІОТ ІВ-93

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Модуль для організації зв'язку machine-to-machine для БЛА*»

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи бездротового зв’язку між БЛА без використання наземної станції.

Областю застосування цієї системи є проектування більш автономних БЛА які змозі виконувати скоординовані місії в умовах відсутності наземної станції, на приклад допомога в пошуку людей при великомасштабних пожежах.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка системи побудови бездротового зв’язку між БЛА без участі наземної станції.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є програмне забезпечення польотного контролера у відкритому доступі Ardupilot, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Забезпечувати взаємодію між БЛА без участі наземної станції
- Підтримувати бездротовий зв’язок на дистанції до 1000 метрів
- Використовувати доступне на ринку обладнання
- Мати можливість бути інтегрованим у систему БЛА без зміни прошивки

5.2. Вимоги до програмного забезпечення

- Ubuntu 22.04 LTS
- Clang 6.0 та вище
- Python 3.10 та вище

5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core (TM) i5-7400.
- ROM не менше ніж 128 ГБ.
- RAM не менше ніж 8 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	02.02.2023-03.02.2023
Вивчення та аналіз завдання	05.02.2023-15.03.2023
Розробка архітектури та загальної структури системи	15.03.2023-25.03.2023
Розробка структур окремих частин системи	25.03.2023-05.04.2023
Програмна реалізація системи	05.04.2023-15.04.2023
Виправлення помилок	20.04.2023-22.04.2023
Оформлення документації дипломної роботи	25.04.2023-25.05.2023

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Модуль для організації зв'язку machine-to-machine для БЛА*»

Київ – 2023

ЗМІСТ

ВСТУП

В рамках сучасних технологій та прогресу у галузі безпілотних літальних апаратів, важливим компонентом для розвитку та оптимізації роботи стає модуль організації зв'язку machine-to-machine. Він спрямований на поліпшення якості обміну даними між різними машинами безпосередньо, без потреби в участі людини. Це дозволяє створювати більш складні, ефективні та інтегровані системи управління БЛА. Machine-to-machine технології, які вже широко використовуються в різних галузях промисловості, включаючи телекомунікації, медицину, транспорт та енергетику, тепер набувають значущості і в сфері БЛА. Основною метою є забезпечення безперебійного, стабільного та високошвидкісного зв'язку між апаратами для ефективного виконання задач та оперативного реагування на зміни умов. У цьому контексті, розгляд модулів machine-to-machine для БЛА включає в себе аналіз технічних характеристик, особливостей протоколів зв'язку, а також переваг та недоліків різних технологій. Також ми розглянемо практичні аспекти їх впровадження, зокрема врахування специфіки використання БЛА, від потреб та вимог безпеки до енергоефективності та оптимізації роботи. В умовах воєнного стану, коли фізична розробка та тестування модуля для БЛА неможлива, ми можемо використовувати альтернативний підхід: симуляцію. Також це дозволяє нам провести велику кількість ітерацій, не піддаючи ризику дорогоцінне обладнання, і відтворити різноманітні польотні умови в безпечному та контрольованому середовищі. Використовуючи SITL, Ardupilot та MavLink в поєднанні, ми можемо ефективно розробити та протестувати наш модуль для організації зв'язку machine-to-machine для БЛА, незважаючи на обмеження фізичної розробки.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ АРХІТЕКТУР КОМУНІКАЦІЙ ТА ЗВ’ЯКУ МІЖ БЛА

1.1 Аналіз предметної області

БЛА (Безпілотні літальні апарати), також відомі як БЛА, стають все більш доступними та дешевими для більшості людей. Також збільшилась кількість різноманітного програмного забезпечення саме для БЛА, від великих проектів компаній до маленьких стартапів початківців. Всі ці люди захоплюються, вивчають та розробляють програмне забезпечення для БЛА. Для роботи програмного забечення на БЛА треба встановити модуль зв’язку, який в свою чергу буде приймати сигнали від наземної станції керування (у більшості випадків це людина яка віддає команди через умовний контроллер). Модуль для організації зв'язку machine-to-machine використовується для роя БЛА, щоб прискорити передачу данних між БЛА та наземною станцією керування. У цій роботі ми розглянемо саме способи передачі необхідної інформації та команд від наземної станції керування до БЛА, також доповнимо вже існуючий модуль Ardupilot новим функціоналом для комунікації роя БЛА з використання децентралізованої архітектури зв’язку.

1.2 Аналіз існуючих архітектур

Архітектура комунікації відіграє важливу роль у розумному керуванні та автономності роїв БЛА (Безпілотних Літаючих Апаратів). У залежності від різних сценаріїв місій існують різні архітектури комунікації. Централізована архітектура комунікації підходить для сценаріїв, де рой БЛА є невеликим, а завдання відносно простим. Кожен окремий БЛА потребує довгодіючого зв'язку з інфраструктурою. Децентралізована архітектура комунікації розширює зону покриття комунікації за допомогою мережі з кількома

проміжними вузлами. Спеціально виділений БЛА-шлюз відповідає за комунікацію БЛА з інфраструктурою (U-T-I – UAV to Infrastructure). Архітектура "single-group swarm Ad hoc network" підходить для рою з БЛА одного типу, тоді як архітектури "multi-group swarm Ad hoc network" та "multi-layer swarm Ad hoc network" можуть бути реалізовані з використанням різних типів БЛА.[2]

1.2.1 Централізована Архітектура

Централізована система складається з наземної станції керування та самих БЛА. Наземна станція отримує інформацію про розташування БЛА та відсилає команду кожному БЛА у рої окремо. У більшості випадків міссія заздалегідь запрограмована на борту кожного БЛА та всі команди в міссії виконуються одночасно на кожному БЛА, а назмена станція використовується для спостереження за системою. Такі рої БЛА вважаються напівавтономними, оскільки вони все ще потребують вказівок від наземної станції для виконання поставленого завдання. Така архітектура роїв БЛА є найбільш поширенною на данний момент (схематичне зображення централізовнаї архітектури можна побачити на рисунку 1.1).[6]

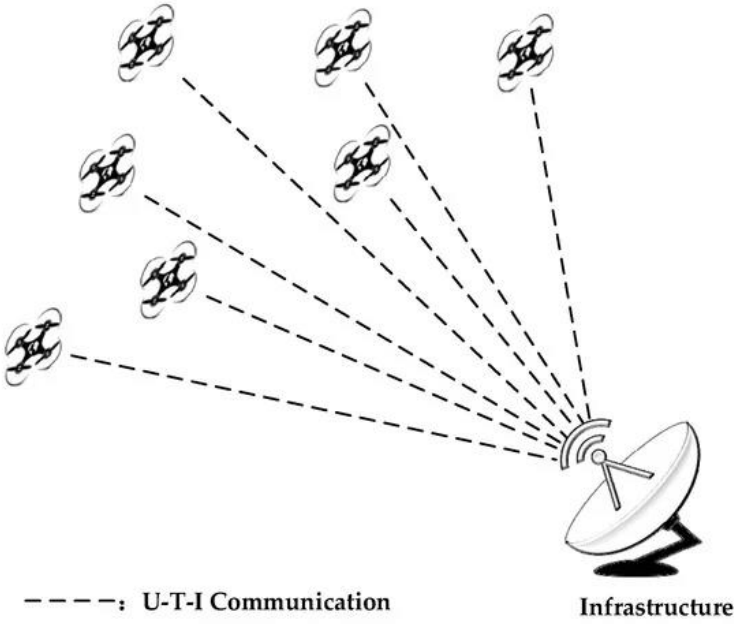


Рисунок 1.1 – Схема централізованої архітектури[2]

1.2.2 Децентралізована Архітектура

Децентралізована система також складається з наземної станції керування та самих БЛА, але один з рою БЛА буде працювати як шлюз для переді данних від наземної станції керування до інших БЛА у рої. Для цього на головному БЛА у рою треба буде встановити два приймача, один для зв'язку з іншими БЛА на малих потужностях і коротких відстанях, а інший - для зв'язку з інфраструктурою на великих потужностях і з великим радіусом досяжності. Із-за цього на інші БЛА вже не треба встановлювати тяжкі приймачі для великих відстаней зв’язку, бо вони будуть отримувати команди від головного БЛА який завжди знаходиться поряд. Але також є одне обмеження у цієї архітектури, а саме траекторія та швидкість кожного БЛА у рої повинна бути однакова, щоб зв’язок був постійним на стабільним (схематичне зображення децентралізовнаї архітектури можна побачити на рисунку 1.2).[14]

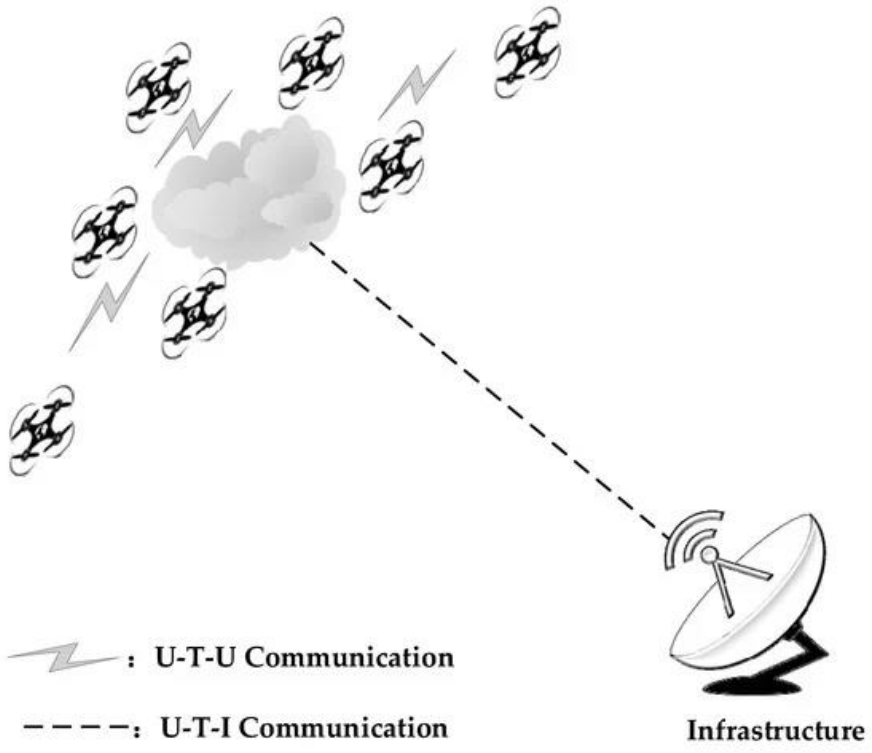


Рисунок 1.2 – Схема децентралізованої архітектури[2]

Також в залежності від заданної міссії архітектура внутрішньої комунікаціїї може приймати різні вигляди.

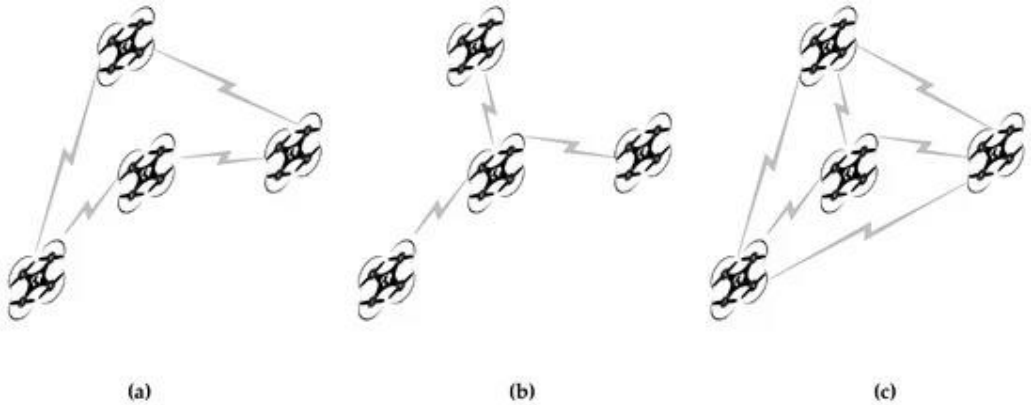


Рисунок 1.3 – Схема архітектур внутрішньої комунікаціїї[7]

Приклад різних архітектур внутрішньої комунікаціїї зображена на рисунку 1.3. БЛА в кільцевій архітектурі утворюють замкнутий контур зв'язку через двонаправлене з'єднання. [2]

1.2.3 Multi-Group UAV Network Architecture

Різні типи груп мають різне застосування в залежності від місії. Загалом, архітектура організована централізовано, як описано у Розділі 1.2.1. Відмінність полягає в тому, що внутрішньогрупові БЛА спілкуються один з одним у спеціальний спосіб. Архітектура внутрішньогрупового зв'язку така ж, як і архітектура зв'язку всередині рою, описана в Розділі 1.2.2. Міжгруповий зв'язок(G-T-G – Group to group), тобто зв'язок між групами, здійснюється через наземну станцію керування, тому головний БЛА все ще відповідає за зв'язок зі станцією керування. Слід звернути уваги на недоліки такої архітектури, а саме високу затримку для спілкування між групами БЛА, цю проблему вирішить наступна архітектура яку ми будем розглядати.

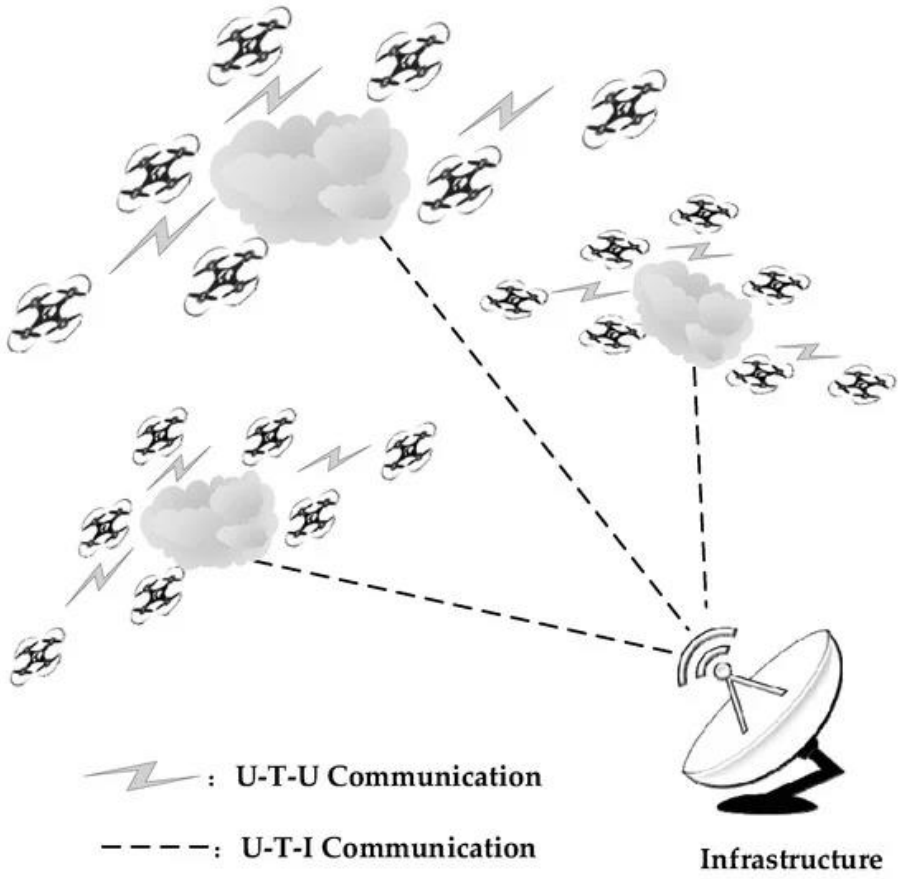


Рисунок 1.4 – Схема Multi-Group UAV Network Architecture[7]

Одним із прикладів використання такої архітектури є виконання завднань де группи знаходятся на великих відстаннях від одне одного, але повинні всі летіти в одне місце. Команда з наземної станції буде відсилатися головним БЛА, а вони в свою чергу передають її всії группі (схематичне зображення Multi-Group UAV Network Architecture можна побачити на рисунку 1.4).[14]

1.2.4 Multi-Layer UAV Ad Hoc Network Architecture

Multi-Layer UAV Ad Hoc Network Architecture це ще одна архітектура яка підходить для великої кількості БЛА, вона більш вдосконалена в порівнянні з Multi-Group UAV Network Architecture. Працює вона з допомогою окремих БЛА які комунікують між групами утворюючи окрему мережу яка є першим рівнем архітектури зв’язку. Архітектура внутрішнього зв’зку такаж сама як у Розділі 1.2.2. Різні типи груп БЛА покладаються на шлюзові БЛА для

забезпечення зв'язку G-T-G (Group to Group) , що становить другий рівень. Найближчий шлюзовий БЛА зв’язується з наземною станцією, що становить собою вже третій рівень архітектури. Зв'язок між будь-якими двома БЛА в цій архітектурі не вимагає ретрансляції інфраструктури. Взаємний зв'язок БЛА в одній групі відбувається на першому рівні. Зв'язок між БЛА в різних групах маршрутизується через головних БЛА. Пакети даних проходять через перший і другий рівні по черзі. Таким чином, в "Multi-Layer UAV Ad Hoc Network Architecture" відсутня єдинна точка відмови, і цей тип архітектури є надійним.[14]

1.3 Порівняння існуючих архітектур

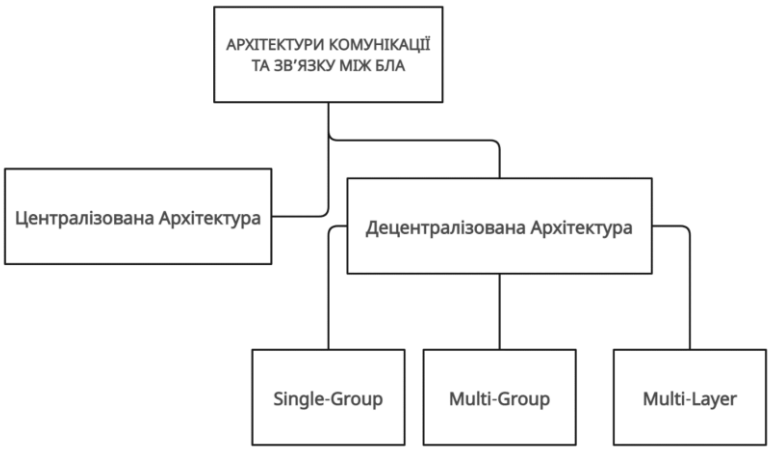


Рисунок 1.5 – Схематичне зображення всіх описанних архітектур[7]

Обговорюючи переваги та недоліки архітектур зв'язку роїв БЛА, ми часто фокусуємося на високих вимогах до покриття та забезпеченні зв'язку. Високе покриття відіграє важливу роль у зборі розвідданих та аналізі ситуацій. Лише за допомогою надійного зв'язку можна гарантувати комунікацію рою в реальному часі. Однак рої БЛА часто оперують у невідомому середовищі, де поява загроз та перешкод є непередбачуваною за часом і простором. Тому надзвичайно важливо, щоб члени рою могли відступити або приєднатися до нього, що ускладнює безперебійність зв'язку. Це переважно пов'язано зі

зниженням сигналу через його втрату при поширенні. Для забезпечення безперебійного з'єднання, з одного боку, відстань у спеціальній конфігурації мережі рою БЛА не повинна перевищувати чутливість приймача і повинна бути обмежена мінімальним співвідношенням сигнал/шум або рівнем сигналу прийому. З іншого боку, варто взяти на увагу природні властивості стадних птахів та риб. БЛА в рої мають бути здатні когнітивно реагувати на зміни навколишнього середовища, щоб адаптувати свою рухову поведінку до умов каналу зв'язку. Крім того, дослідники вже використовували потужні бездротові мережі 4G для підвищення стабільності з'єднання рою БЛА у широких зонах покриття. Впровадження нової бездротової мережі 5G наддасть ще більшу технічну підтримку для таких досліджень. Тож підсумуємо всі плюси та мінуси розглянених архітектур зв’язку між БЛА невеликою таблицею (таблиця 1.6)[2][6]

Таблиця 1.6 – Переваги та недоліки чотирьох описаних архітектур[2]

Функції	Централізована Архітектура	Single-Group	Multi-Group	Multi-Layer
Мульти-канальна комунікація	Ні	Так	Так	Так
Ретрансляція трафіку	Ні	Так	Так	Так
Різні види БЛА	Ні	Ні	Так	Так
Самоналаштування	Ні	Так	Ні	Так
Обмежена зона покриття	Так	Так	Так	Ні
SPOF (Єдина точка відмови)	Так	Ні	Так	Ні

Надійність	Так	Ні	Ні	Так

1.4 Аналіз існуючих видів зв’язку

1.4.1 Радіозв’язок

Контролери для БЛА використовують бездротову технологію, що дозволяє керувати ними на відстані. За допомогою пульта дистанційного керування контролер надсилає радіосигнал на БЛА, віддаючи йому команди. Варто зазначити, що кожен контролер для БЛА, оснащений передавачем. Передавач у контролері випромінює радіосигнал, який потім приймається приймачем БЛА. Це основний метод, за допомогою якого БЛА встановлюють зв’язок з контролером. Щоб забезпечити надійний і безперебійний зв’язок, передавач і приймач повинні працювати на одній частоті. Зазвичай БЛА керуються за допомогою нижчих частот, оскільки вони мають здатність ефективніше проникати крізь перешкоди.[1]

1.4.2 Wi-Fi

Пройшли ті часи, коли Wi-Fi з’єднання обмежувалося лише комп’ютерами. Завдяки технологічному прогресу пристрої тепер можуть бути легко інтегровані з Wi-Fi. БЛА також прийняли цю технологічну революцію, і в результаті деякі з них тепер керуються за допомогою Wi-Fi. БЛА з підтримкою Wi-Fi в першу чергу слугують для потокової передачі відео на смартфони, ПК або планшети[1]. У технології безпілотників використовуються дві основні категорії частот Wi-Fi: 2,4 ГГц і 5 ГГц. Кожна з цих частот має свої переваги, і вибір залежить від конкретних вимог, таких як бажана відстань польоту та якість відео.

Частота 2,4 ГГц добре підходить для зв'язку з БЛА на великі відстані, оскільки дозволяє сигналу долати більшу відстань. Однак діапазон 2,4 ГГц широко використовується різними бездротовими пристроями, такими як мікрохвильові печі та камери відеоспостереження, що призводить до переповненості спектру. Використання частоти 2,4 ГГц збільшує ймовірність виникнення перешкод, що потенційно може призвести до нестабільного з'єднання. Крім того, якість зображення на екрані контролера може бути не такою чіткою при використанні цієї частоти.[3]

Частота 5,0 ГГц забезпечує швидшу передачу даних порівняно з частотою 2,4 ГГц. Це забезпечує високошвидкісний зв'язок між БЛА і контролером, що призводить до більш реального відображення зображення на екрані. Крім того, на частоті 5,0 ГГц менше перешкод, що забезпечує більш стабільне з'єднання. Однак варто зазначити, що частота 5,0 ГГц має меншу здатність проникати крізь тверді об'єкти, такі як стіни. Тим не менш, якщо робота БЛА обмежена в межах вашої видимості, це обмеження може не викликати занепокоєння. Для усунення будь-яких обмежень дальності, пов'язаних з частотою 5,0 ГГц, можна використовувати ретранслятори.[3]

1.4.3 GPS

БЛА використовують технологію глобального позиціонування як додатковий засіб зв'язку з контролером. Ця технологія дозволяє контролеру надавати безпілотнику інструкції щодо бажаного маршруту або шляху, яким він має слідувати. Запрограмувавши БЛА відповідним чином, він може здійснювати навігацію і летіти у вказаному напрямку за командою системи GPS. БЛА з підтримкою GPS використовують GPS-приймач для встановлення зв'язку щонайменше з чотирма супутниками. Цей зв'язок відбувається за допомогою радіосигналів, які поширюються зі швидкістю світла. Система GPS відіграє вирішальну роль у точному визначенні місцезнаходження безпілотника з точністю від метрів до сантиметрів.[1][4]

ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі були розглянуті різні архітектури комунікацій та зв’язку між БЛА. Існує 2 вида архітектур, а саме Централізовні та Децентралізовані для передачі даних між наземною станцією керування та БЛА. Децентралізовані в свою чергу поділяються на Single-group, Multi-group та Multi-layer. У кожної архітектури є своє призначення, переваги та недоліки, а які саме ми роздивилися у попередніх розділах.

Також було розібрано види зв’язку які використовують для з’єдання та передачі команд з контролера на БЛА. Зазвичай сучасні БЛА використовують усі три перелічені види зв’язку задля виконання поставленних місій, але також можливі ситуації коли БЛА може справно працювати умовно без Wi-Fi модуля, з заздалегідь завантаженими картами по GPS. Правда в тому, що у такому випадку буде відсутня функціональність повернення до відправної точки, якщо вона змінилася. Може бути і навпаки, що Wi-Fi модуль присутній а GPS ні, але і в цьому випадку є свої недоліки, тому сучасні БЛА поєднують у собі всі можливі види зв’язку.

РОЗДІЛ 2. НАЛАГОДЖЕННЯ ВІРТУАЛЬНОГО СЕРЕДОВИЩА ДЛЯ ТЕСТУВАННЯ МОДУЛЯ ДЛЯ БЛА

2.1 Огляд існуючих рішень

З приводу військового стану запровадженого з 24 лютого 2022 року управління СБУ зазначає що вимоги пункту 4 розділу II Правил використання повітряного простору, а саме – “польоти безпілотних повітряних суден масою до 20 кг включно виконуються без подання заявок та отримання дозволів на використання повітряного простору” під час воєнного стану не діють, а запуск БЛА та легких літаків відбувається виключно з погодження Служби безпеки України.

Тому в мене не були можливості розробляти та тестувати модулі зв’яку на фізичних БЛА. На допомогу приходять симуляції БЛА. Наразі на ринку є такі варіанти[11]:

1. **Gazebo**

Широко визнаний як основний симулятор в галузі робототехніки і слугує основою для більшості симуляторів в цій галузі. Він особливо популярний серед інженерів з робототехніки завдяки широкій підтримці різних віртуальних датчиків, сумісності з ROS (Robot Operating System), інтеграції з Ardupilot і PX4 (популярним програмним забезпеченням для автопілотів), попередньо створеним моделям готових БЛА, високому рівню кастомізації, можливості одночасного моделювання декількох систем і використанню складного фізичного рушія. Як результат, Gazebo широко використовується для розробки цілого ряду безпілотних систем, включаючи VTOL (Vertical Take-Off and Landing), літальних апаратів і квадрокоптерів.

Однак основним недоліком Gazebo є відносно низька точність і обмежені можливості для віртуальних світів і об’єктів. Gazebo не пропонує просунутих функцій, таких як складні сценарії для агентів, анімація, ригінг моделей або фотореалістичний рендеринг. Отже, Gazebo не ідеальний вибір для завдань, пов’язаних з комп’ютерним зором або розробкою автономії, які вимагають більш реалістичних і детальних середовищ моделювання.[10]

2. jMAVSim

jMAVSim – це простий симулятор, розроблений виключно для використання з PX4 і в першу чергу використовується як інструмент швидкої валідації при розробці БЛА. Він має певні обмеження з точки зору моделювання фізики, візуального реалізму та загальної динаміки симуляції. Діапазон доступних датчиків також обмежений, в основному зосереджений на базових датчиках PX4. Незважаючи на ці обмеження, jMAVSim може слугувати корисним інструментом для швидкого прототипування та перевірки комунікаційних і базових програмних функцій як підтвердження концепції. Однак, коли інженери заглиблюються в процес розробки, вони часто виявляють, що можливостей jMAVSim недостатньо для їхніх потреб, що спонукає їх шукати більш просунуті симулятори.

3. Airsim

Airsim – це плагін для Unreal Engine 4, що надає спрощену модель динаміки польоту для симуляції БЛА. Він може взаємодіяти з системами автопілота Ardupilot та PX4. Однією з найсильніших сторін Airsim є винятковий візуальний реалізм, який забезпечує Unreal Engine 4, особливо в поєднанні з можливістю створювати власні середовища. Ця комбінація дозволяє розробляти високодеталізовані та візуально вражаючі віртуальні середовища.

Хоча фізична симуляція БЛА в Airsim дещо спрощена, це компенсується широким спектром варіантів симуляції датчиків, таких як камери і LiDAR, які можна використовувати в режимі реального часу для оцінки складних поведінкових систем. Однак важливо зазначити, що висока візуальна точність Airsim пов’язана з підвищеними обчислювальними вимогами порівняно з іншими варіантами моделювання БЛА.

4. AuterionSim

Auterion sim – це платне хмарне рішення для симуляції, призначене для комерційних платформ БЛА на базі PX4. Цей симулятор пропонує карти реальних локацій, забезпечуючи реалістичний просторовий контекст. Хоча візуальна складова Auterion гірша ніж у його конкурентів, він перевершує PX4 і MAVLink, маючи інтегровану підтримку камер.

Симулятор є дуже цінним для ряду спеціалізованих завдань і слугує загальним інструментом перевірки систем, які взаємодіють з БЛА з підтримкою PX4. Значною перевагою Auterion sim є те, що він працює в Інтернеті, що зменшує типові обчислювальні обмеження, пов'язані з локальним моделюванням.

5. Flightgear

Flightgear – це симулятор з відкритим вихідним кодом, який використовує JSBSim для забезпечення високореалістичної динаміки польоту. Він також пропонує можливості інтеграції з PX4, що дозволяє безперешкодно взаємодіяти з системою автопілота. Хоча Flightgear не може похвалитися найбільш візуально вражаючою графікою в порівнянні з іншими варіантами, його перевага полягає у використанні JSBSim, що забезпечує точну і реалістичну аеродинамічну симуляцію.

Як результат, Flightgear забезпечує відповідний баланс для оцінки та вдосконалення нових платформ безпілотників, за умови, що модельований БЛА є достатньо точним.

6. JSBSim

JSBSim – це не окремий симулятор, а модель динаміки польоту з відкритим вихідним кодом, яка забезпечує реалістичну аеродинамічну симуляцію для різних застосувань. Вона слугує моделлю польоту для кількох симуляторів, згаданих вище, і широко відома своїми точними можливостями аеродинамічного моделювання польоту. JSBSim дозволяє моделювати нестандартні планера, пропонуючи гнучкість і можливості кастомізації. У травні 2022 року було випущено плагін, який дозволяє інтегрувати JSBSim з Unreal Engine, розширюючи його використання до платформи Unreal Engine.

7. X-Plane

X-Plane – це вдосконалений симулятор, який пропонує інтеграцію з Ardupilot. Він вирізняється винятковою візуальною реалістичністю і великою колекцією доступних транспортних засобів, що робить його привабливим вибором для створення вражаючих демонстрацій можливостей автопілота. Хоча підтримка датчиків, окрім базових датчиків керування польотом, обмежена, можливість створювати власні середовища з високою візуальною достовірністю робить X-Plane привабливим варіантом для розробки.

8. RealFlight

Realflight – це симулятор, який пропонує унікальну функцію, що дозволяє користувачам розробляти і тестувати власні конструкції транспортних засобів, що робить його придатним для тестування апаратного

дизайну за допомогою візуального підходу. Хоча візуальна точність Realflight є пристойною, вона може не відповідати вимогам більш просунутих програм комп'ютерного зору. Хоча в першу чергу RealFlight орієнтований на спільноту RC (Remote Control), інтеграція з Ardupilot розширює можливості його використання і для тестування поведінки.

9. MATLAB & Simulink

MATLAB і Simulink є широко використовуваними інструментами в галузі фізичного проектування та калібрування контролерів. Хоча вони відрізняються від традиційних симуляторів, згаданих у цьому списку, MATLAB і Simulink пропонують численні цінні ресурси, включаючи UAV Toolbox, який полегшує розробку користувацьких реалізацій для PX4 і Ardupilot. Крім того, MATLAB і Simulink дозволяють моделювати вхідні дані датчиків, налаштовувати ПІД-регулятор і генерувати код для широкого спектру розширених налаштувань безпілотника.

10. Webots

Webots – це симулятор з відкритим вихідним кодом, який має схожість з Gazebo, пропонуючи платформу для загального розвитку робототехніки. Хоча він підтримує інтеграцію з Ardupilot (але не з PX4), його візуальна точність дещо обмежена. Проте Webots чудово підходить для розробки кастомних транспортних засобів і може похвалитися надійною підтримкою датчиків. Крім того, він надає спрощені фізичні моделі для різних типів транспортних засобів, що сприяє прискоренню процесу розробки.

11. SCRIMMAGE

SCRIMMAGE (Simulating Collaborative Robots in Massive Multi-Agent Game Execution) – це симулятор, спеціально розроблений для розробки поведінки рою БЛА. Він дозволяє користувачам моделювати велику кількість БЛА, що працюють у спрощених середовищах з мінімальною візуальною деталізацією, зосереджуючись на оптимізації багатоагентних взаємодій. Будучи універсальною платформою для моделювання, SCRIMMAGE полегшує гнучке тестування алгоритмів мобільної робототехніки та пропонує підтримку інтеграції з Ardupilot та PX4.

2.2 ROS/Gazebo Simulation Environment

Я обрав саме Gazebo, бо він підтримує всі необхідні для моєї праці інстурменти, плагіни та фреймворки, а саме Arducopter, Ardupilot, python-dronekit, SITL, MAVLink, MAVProxy, ROS та Qgroundcontrol. Також не можна не зазначити, що Gazebo має один з найреалістичнх фізичних движків та велику кількість готових моделей БЛА та світів де їх можна тестувати.

Середовище моделювання розробки ROS/Gazebo складається з трьох основних компонентів:

- Ardupilot, програмне забезпечення автопілота з відкритим вихідним кодом, що використовується для керування поведінкою БЛА.
- ROS (Robot Operating System) – набір програмних фреймворків, які слугують проміжним програмним забезпеченням для допомоги програмістам у розробці додатків для роботів.

Gazebo – середовище 3D-симуляції, яке дозволяє візуально та тривимірно моделювати кіберфізичні системи. Ці системи включають наземні ровери, БЛА, а також різні інші об’єкти, такі як перешкоди та елементи навколишнього середовища. Разом вони утворюють так званий світ Gazebo.[5]

2.2.1 Ardupilot

Для того, щоб розпочати програмне моделювання одного транспортного засобу, необхідним компонентом є програмне забезпечення з відкритим вихідним кодом Ardupilot. Розроблене спільнотою DIY Drones, Ardupilot дозволяє керувати різними безпілотними транспортними засобами, включаючи БЛА.

Ardupilot пропонує різноманітні режими польоту, які поділяються на ручні та автоматичні, з можливістю налаштування параметрів. Наприклад, БЛА можна керувати в режимі керованого польоту, коли Ardupilot автоматично регулює всі три осі на основі заданого положення. Крім того, інші режими польоту дозволяють БЛА підтримувати потрібну висоту, досягати певної позиції, рухатися по круговій траєкторії з фіксованим радіусом, повертатися до точки старту, ініціювати посадку та інше.

Для зв'язку з безпілотником використовується протокол MAVLink. Цей протокол забезпечує зв'язок між апаратами за допомогою обміну пакетами. Пакети, представлені у вигляді рядків мовою C, складаються з окремих бітів, які виконують певні функції в комунікації. Кожен пакет містить заголовок, корисне навантаження та контрольну суму і передається послідовними каналами зв'язку. MAVLink дозволяє обмінюватися попередньо визначеними командами, такими як повідомлення серцебиття (ID 0), що використовується для моніторингу стану зв'язку з БЛА, та повідомлення set_mode (ID 11), що використовується для визначення певного режиму польоту. Крім того, можна створювати користувацькі повідомлення.[5]

2.2.2 ROS

ROS (Robot Operating System) – не є власне операційною системою, а радше набором фреймворків і бібліотек з відкритим кодом, призначених

для розробки програмного забезпечення в робототехніці. Її мета – спростити і прискорити створення прототипів програмного забезпечення для робототехніки, сприяючи повторному використанню програмного забезпечення і забезпечуючи сумісність між різними інструментами в середовищі ROS. Вона вирішує такі завдання, як збір даних у реальному часі з датчиків у кіберфізичних системах, комунікація між роботами за моделлю «publish/subscribe», обробка команд користувача та відповідних дій.

ROS надає широкий спектр функціональних можливостей, включаючи апаратну абстракцію, драйвери пристроїв, бібліотеки, візуалізатори, можливості передачі повідомлень та управління пакетами. У ROS компоненти робота представлені у вигляді вузлів мережі, які взаємодіють один з одним за допомогою анонімного та асинхронного механізму publish/subscribe. Така модель проектування значно скорочує зусилля на розробку і сприяє гнучкості та модульності системи.

Крім того, ROS включає в себе інструменти для визначення фізики робота. Для опису фізичних параметрів робота можна використовувати файли у форматі Unified Robot Description Format (URDF). Це полегшує інтеграцію з 3D-симуляторами, такими як Gazebo, покращуючи симуляцію поведінки та взаємодії робота.[5]

2.2.3 Gazebo

Використання візуального симулятора корисно для оцінки продуктивності конкретних алгоритмів. Gazebo, наприклад, дозволяє створювати реалістичні сценарії, що охоплюють кіберфізичні системи та навколишнє середовище. Цей симулятор має комплексну динамічну та кінематичну фізику, а також гнучкий фізичний рушій. Інтеграція Gazebo з ROS полегшується набором плагінів Gazebo, які підтримують численні роботи і датчики, що зазвичай використовуються в польових умовах. Ці

плагіни відповідають тому ж інтерфейсу повідомлень, що і решта середовища ROS, що забезпечує бездоганну сумісність між моделюванням, записаними даними та апаратним забезпеченням. Важливо, що розробники можуть створювати додатки безпосередньо в середовищі моделювання, а потім розгортати їх на фізичних роботах з мінімальними змінами коду або взагалі без них. Наукова спільнота широко використовує Gazebo для моделювання літальних апаратів БЛА.[5]

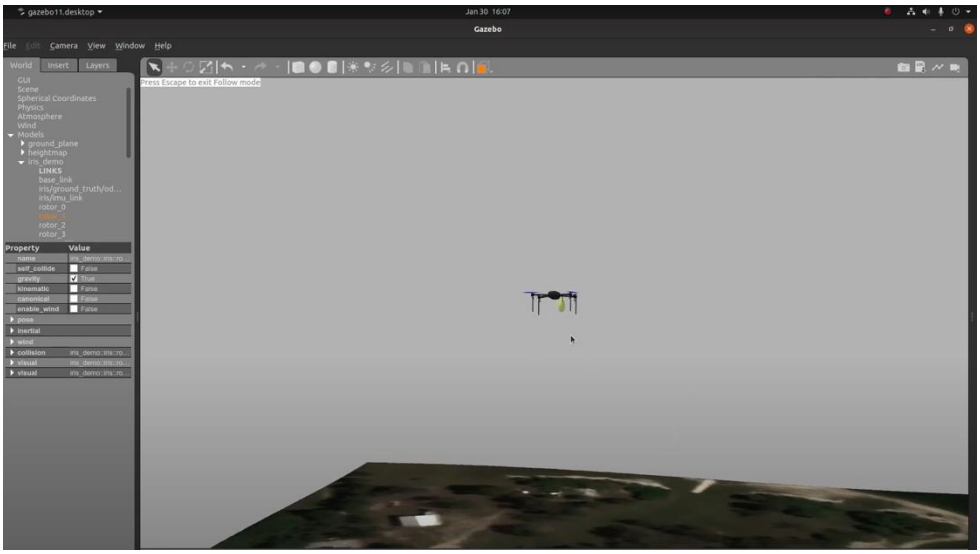


Рисунок 2.1 – Візуалізація виконання місії БЛА

2.2.4 SITL

Симулятор SITL (software in the loop) дозволяє запускати літак, коптер або ровер без будь-якого обладнання. Це збірка коду автопілота за допомогою звичайного компілятора C++, що дає вам нативний виконуваний файл, який дозволяє протестувати поведінку коду без апаратного забезпечення.

SITL дозволяє запускати ArduPilot на вашому комп'ютері безпосередньо, без будь-якого спеціального обладнання. Він використовує той факт, що ArduPilot – це портативний автопілот, який може працювати на дуже широкому спектрі платформ. Ваш ПК – це просто ще одна платформа, на якій можна створити і запустити ArduPilot.

При роботі в SITL дані з датчиків надходять з моделі динаміки польоту в симуляторі польоту. ArduPilot має широкий спектр вбудованих симуляторів транспортних засобів і може взаємодіяти з декількома зовнішніми симуляторами. Це дозволяє тестувати ArduPilot на найрізноманітніших типах транспортних засобів.[9]

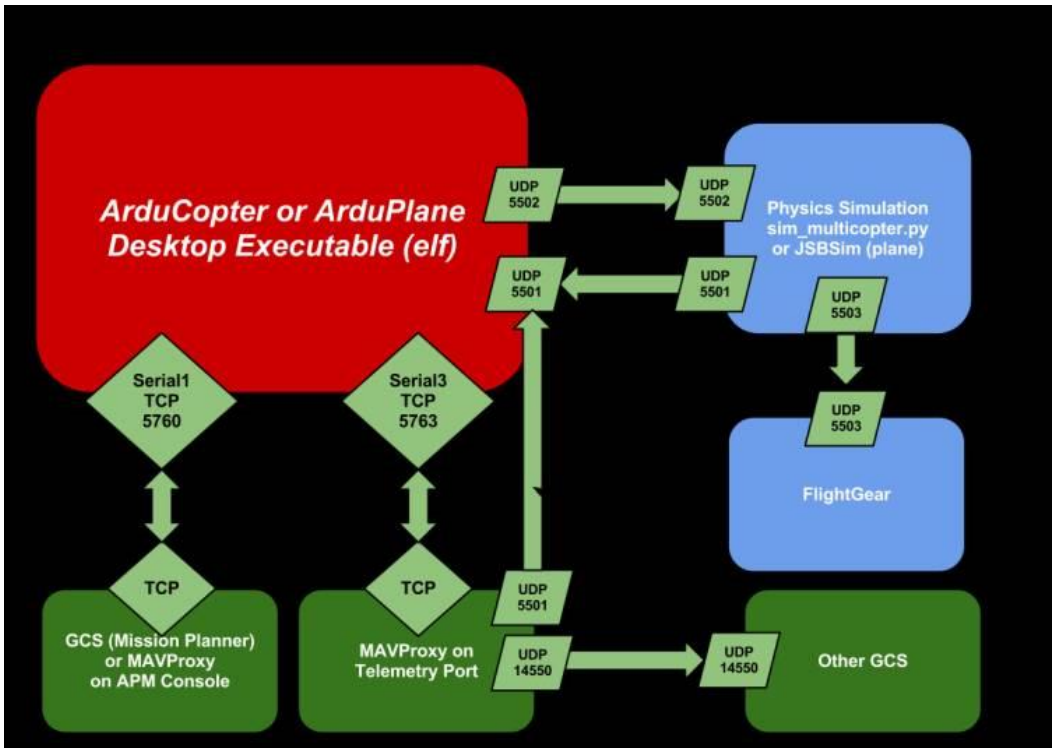


Рисунок 2.2 – Архітектура SITL[9]

2.2.5 MAVlink protocol

MAVLink або Micro Air Vehicle Link – це протокол інформаційної взаємодії з БЛА або малими безпілотними апаратами (літаючими, плаваючими, плаваючими, повзаючими тощо), які за текстом називаються MAV (Micro Air Vehicle). MAVLink поширюється під LGPL ліцензією у вигляді модуля для python (є зручна обгортка DroneKit) і генератора бібліотек під різні мови, зокрема header-only C/C++ бібліотеки.

Протокол описує інформаційну взаємодію між системами, такими як MAV і GCS (Ground control station) – станція наземного управління, а так само

їх складовими частинами – компонентами. Базовою сутністю MAVLink є пакет, що має такий формат (рисунок 2.3 – пакет MAVLink):

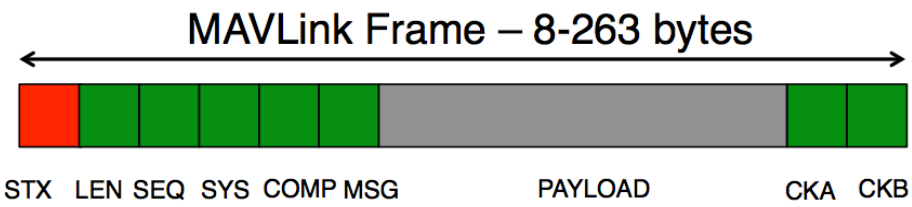


Рисунок 2.3 – пакет MAVLink[8]

Перший байт пакета (STX) – це символ початку повідомлення: 0xFD для версії v2.0, 0xFE для версії v1.0, 0x55 для версії v0.9. LEN – довжина корисного навантаження (повідомлення). SEQ – містить лічильник пакета (0-255), який допоможе нам виявити втрату повідомлення. SYS (System ID) – ідентифікатор системи, що відправляє, а COMP (Component ID) – ідентифікатор компонента, що відправляє. MSG (Message ID) – тип повідомлення, від нього залежить, які дані лежатимуть у корисному навантаженні пакета. PAYLOAD – корисне навантаження пакета, повідомлення, розміром від 0 до 255 байт. Два останні байти пакета – CKA і CKB, нижній і верхній байт, відповідно, містять контрольну суму пакета.

Бібліотека MAVLink дає змогу кодувати та розкодувати пакети згідно з протоколом, але вона не регламентує, якими апаратними та програмними засобами дані буде надіслано – це можуть бути TCP/UDP-повідомлення, обмін через послідовний порт, та будь-що інше, що забезпечує двосторонній обмін. Бібліотека обробляє вхідні дані побайтово, додаючи їх у буфер і сама збирає з них пакет. Кожна система або компонент може одночасно обмінюватися даними за різними джерелами, тоді для кожного джерела призначається спеціальний ідентифікатор, званий channel (канал). MAVLink містить буфер на кожен канал.[8]

2.2.6 QgroundControl

QgroundControl – це програмне забезпечення наземної станції управління (GCS) з відкритим вихідним кодом, яке широко використовується в галузі безпілотних літальних апаратів (БЛА) і БЛА. Воно слугує комплексним інтерфейсом для управління та моніторингу роботи безпілотників. QgroundControl підтримує різні платформи БЛА, в тому числі ті, що використовують протокол зв'язку MAVLink, такі як Ardupilot і PX4.[12]

QgroundControl пропонує зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє операторам планувати та виконувати автономні польоти, контролювати рух безпілотника, відстежувати телеметричні дані в режимі реального часу та отримувати доступ до важливих параметрів та налаштувань польоту. Він надає широкий спектр функцій, включаючи навігацію по маршрутних точках, геозонування, керування камерами та ведення журналів телеметрії. (рисунок 2.4 – графічне зображення інтерфейсу QgroundControl)

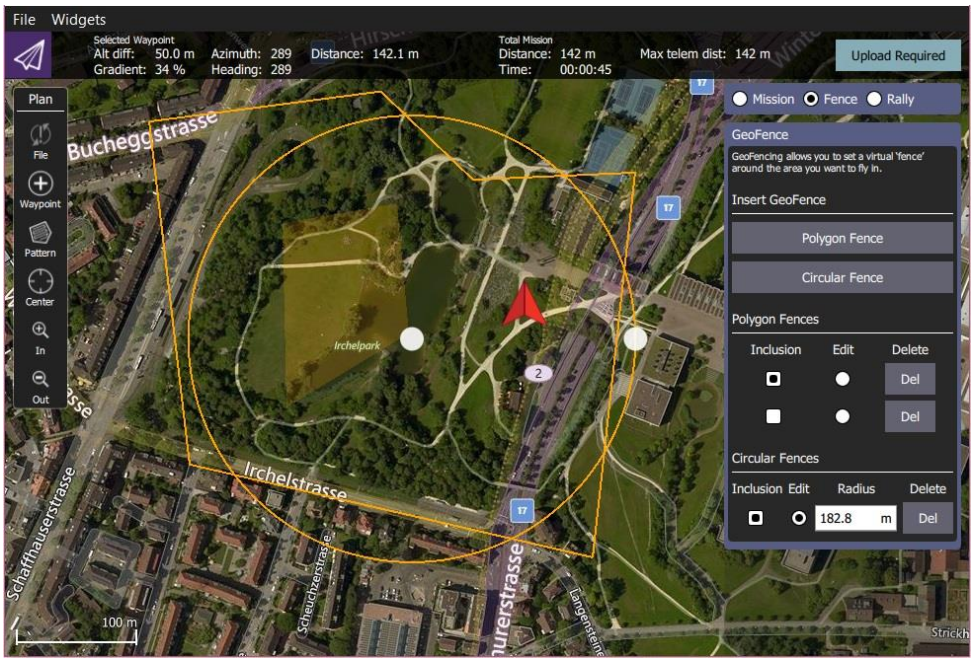


Рисунок 2.4 – графічне зображення інтерфейсу QgroundControl[24]

За допомогою QgroundControl користувачі можуть візуалізувати траєкторію польоту БЛА, переглядати відеопотоки з бортових камер у реальному часі та отримувати доступ до телеметричних даних, таких як висота, швидкість, стан акумулятора та інформація про GPS. Він також підтримує розширені функції, такі як планування місій, геотегування та оновлення прошивки для БЛА.[13]

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі ми ознайомилися з інструментами для налагодження віртуального середовища для тестування модуля для організації зв’язку machine-to-machine для БЛА. Я вважаю, що вибір такого набору (SITL, Ardupilot, ROS/Gazebo, MAVLink, QgroundControl) є найбільш оптимальним для поставленої задачі, а саме завдяки зрозумілим та доступним навіть для звичайної людини документаціям. Наявність відповідей на багато нетривіальних питань від небайдужих людей у мережі інтернет також була для мене великим ЗА при виборі саме цих інструментів.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ КОМУНІКАЦІЇ ДЛЯ РОЮ БЛА

3.1 Вибір операційної системи

Вибір операційної системи для програмування і симуляції безпілотних літальних апаратів (БЛА) залежить від ряду факторів, таких як вимоги до проекту, особисті навички та попередній досвід. Ось деякі з найпоширеніших операційних систем, які використовують програмісти:

- Linux: Linux є дуже популярною операційною системою серед розробників. Вона підтримує багато мов програмування та інструментів, що важливо для розробки БЛА. Також важливим є те, що Linux є відкритим програмним забезпеченням, тому можна налаштувати систему під свої власні потреби. Одним із зручностей Linux є можливість роботи з командним строкою, яка дозволяє виконувати складні задачі з високою точністю.
- Windows: Windows є популярним вибором через свою зручніть, простоту використання та широку підтримку програмного забезпечення. Однак, вона може бути менш гнучкою у термінах налаштування та кастомізації порівняно з Linux. Зазвичай Windows використовується для програмування БЛА, коли для цього використовуються конкретні програми, які працюють тільки на цій ОС.
- macOS: macOS також є популярною операційною системою для програмування, оскільки вона поєднує простоту використання Windows та потужність Unix-подібних систем, таких як Linux. Однак, для розробки БЛА на macOS може бути менше доступного програмного забезпечення порівняно з Linux або Windows.

Оскільки з macOS я ніколи не працював то цей варіант автоматично відпадає особисто для мене, а вся робота з Windows та обраним мною програмним забезпечення зводилася к дуже складній установці та

налагодженню програмного середовища через сторонні програми. Тому я вирішив обрати Linux, а саме останню LTS версію Ubuntu 22.04.[13]

Переваги Linux для програмування та симуляції БЛА:

- Відкритий код: Linux є відкритою операційною системою, що означає, що у вас є повний доступ до вихідного коду. Це дозволяє вам налаштувати систему за власними потребами, що може бути важливим для специфічних вимог БЛА.
- Підтримка ROS: ROS (Robot Operating System) є широко використовуваною платформою для розробки робототехніки, включаючи БЛА. Основна підтримка ROS реалізована на Linux.
- Керування ресурсами: Linux дозволяє краще контролювати ресурси комп'ютера, що може бути корисним для вимогливих симуляцій.
- Стабільність і безпека: Linux відомий своєю стабільністю і безпекою, що може бути важливим для довгострокових проектів.

Недоліки Linux для програмування та симуляції БЛА:

- Сумісність програмного забезпечення: Хоча багато професійних інструментів доступні на Linux, деяке програмне забезпечення, особливо комерційне, може бути доступне лише для Windows або macOS.
- User friendly: Про цей слоган можна забути при роботі з Linux, може знадобитися багато часу, щоб звикнути до його інтерфейсу та системи командної строки.
- Підтримка програмного забезпечення: Хоча підтримка обладнання в Linux з часом поліпшилася, деяке обладнання може бути недостатньо підтримане або взагалі не підтримуватися.[13]

3.2 Налаштування обраних інструментів

3.2.1 Налаштування SITL та середовища збірки (Linux/Ubuntu)

Спочатку треба встановити Git:

Git – це безкоштовна розподілена система контролю версій з відкритим вихідним кодом, яка використовується для керування кодовою базою ArduPilot.[16]

Прописуємо у терміналі команди:

sudo apt-get update

sudo apt-get install git

sudo apt-get install gitk git-gui

Клонуємо репозіторій Ardupilot:

“Клонування” – це термін git’a, що означає створення локальної копії віддаленого репозиторію (тобто того, що знаходиться на серверах GitHub).[16]

Розробники повинні клонувати основний репозиторій ArduPilot (якщо вони просто хочуть завантажити і скомпілювати останній код) або свій власний форк (якщо вони хочуть внести зміни до вихідного коду і потенційно відправити зміни назад).[16]

Прописуємо у терміналі команду:

git clone –recurse-submodules https://github.com/ArduPilot/ardupilot

Завантажуємо необхідні пакети:

Прописуємо у терміналі команди:

cd ardupilot

Tools/environment_install/install-prereqs-ubuntu.sh -y

. ~/.profile

3.2.1 Збірка Ardupilot

Збирати Ardupilot нам допоже waf. Waf — це портативна система побудови програмного забезпечення, що автоматизує компіляцію, тестування і встановлення програмних продуктів.[17]

Прописуємо у терміналі команди:

./waf configure –board sitl

./waf copter

Board можна вказати який завгодно, за замовчуванням встановлено sitl, але наприклад можна зібрати ArduCopter для Pixhawk2/Cube (--board CubeBlack) або для якогось SkyViper GPS.[16]

3.3 Тестування базового функціоналу

3.3.1 Демонстрація з використанням основного стеку Ardupilot

Запускаємо MavProxy:

Прописуємо у першому терміналі команду:

libraries/SITL/examples/follow-mavproxy.sh

Та запускаємо скрипт з 5 коптерами:

Прописуємо у другому терміналі команду:

libraries/SITL/examples/follow-copter.sh

У результаті маємо побачити на екрані таку картину:

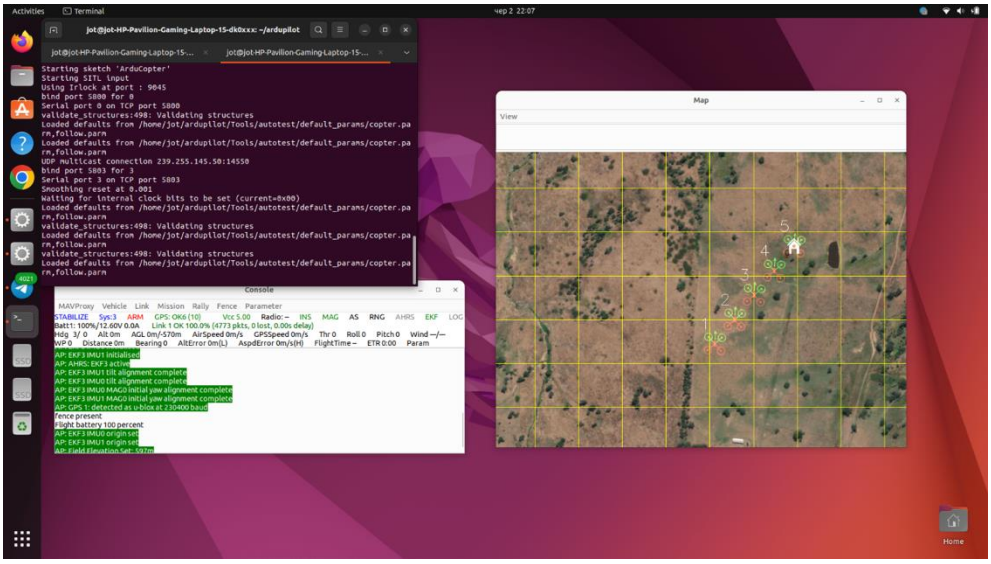


Рисунок 3.1 – Запуск MAVProxy та MAVLink Follow

Тепер в нас є п’ять дронів якими можна керувати, прописуючи команди у термінал.

3.3.2 Класична комунікація у рої БЛА за допомогою Follow

Загальний набір повідомлень MAVLink містить стандартні визначення, якими керує проект MAVLink. Визначення охоплюють функціональність, яка вважається корисною для більшості наземних станцій управління та автопілотів.[18]

Простими словами, MAVLink FOLLOW_TARGET передає координати головного БЛА підконтрольним, 10 разів за секунду (10 hz) , але передача координат відбувається через наземну станцію керування (у нашому випадку це MAVProxy). Тобто ми маємо реалізацію централізованої архітектури у чистому вигляді. Наочно це зображено на рисунку 3.2.

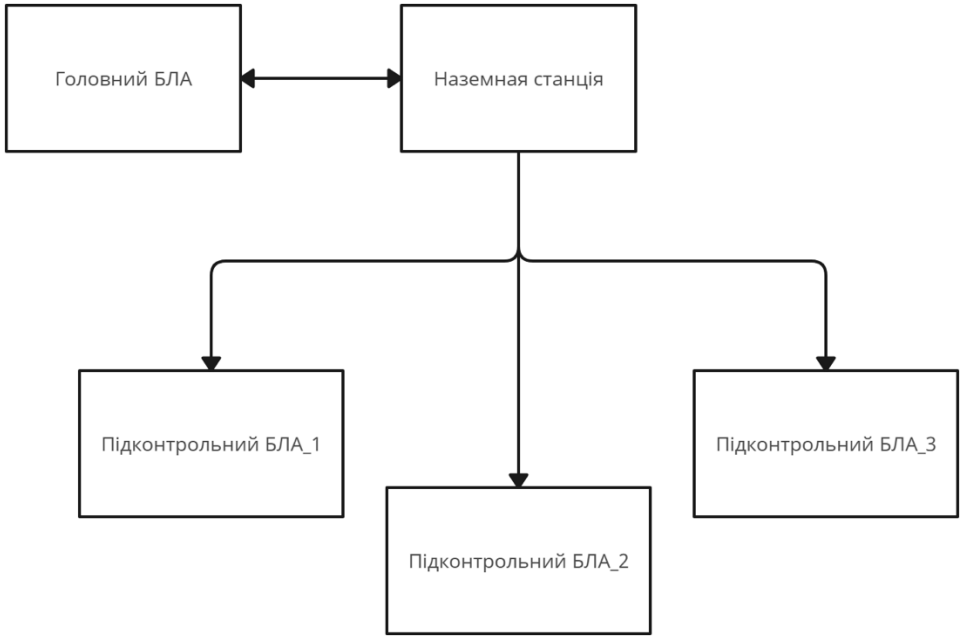


Рисунок 3.2 – Схематичне зображення комунікації рою БЛА з MAVLink Follow

Протестуємо Follow на практиці, для цього прописуємо у першому терміналі команди для головного БЛА:

mode guided

arm throttle

takeoff 5

Результат повиннен бути як на рисунку 3.3.

Це ми підняли головний БЛА у повітря, потім теж саме робимо для підконтрольних БЛА, але дописуючи кожен раз у терміналі “mode follow”. Результат повинен бути як на рисунку 3.4.

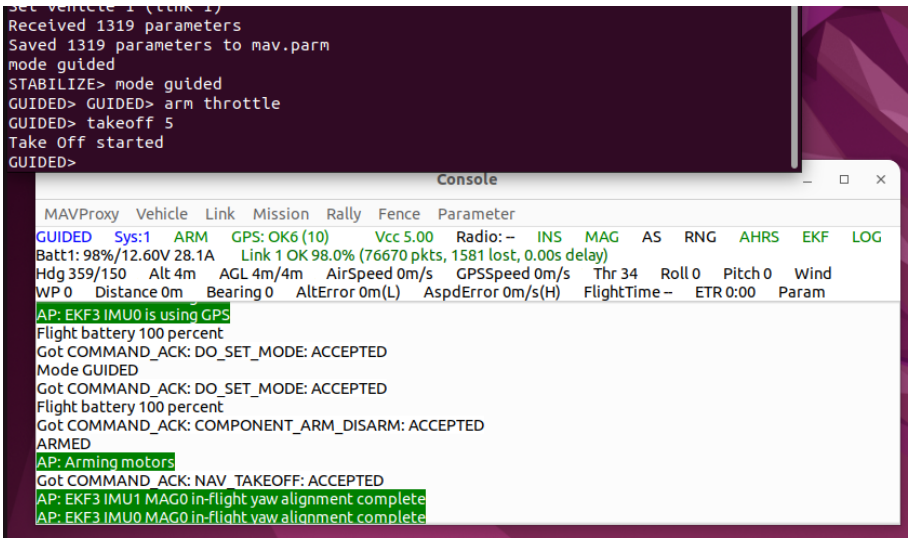


Рисунок 3.3 – Взліт головного БЛА

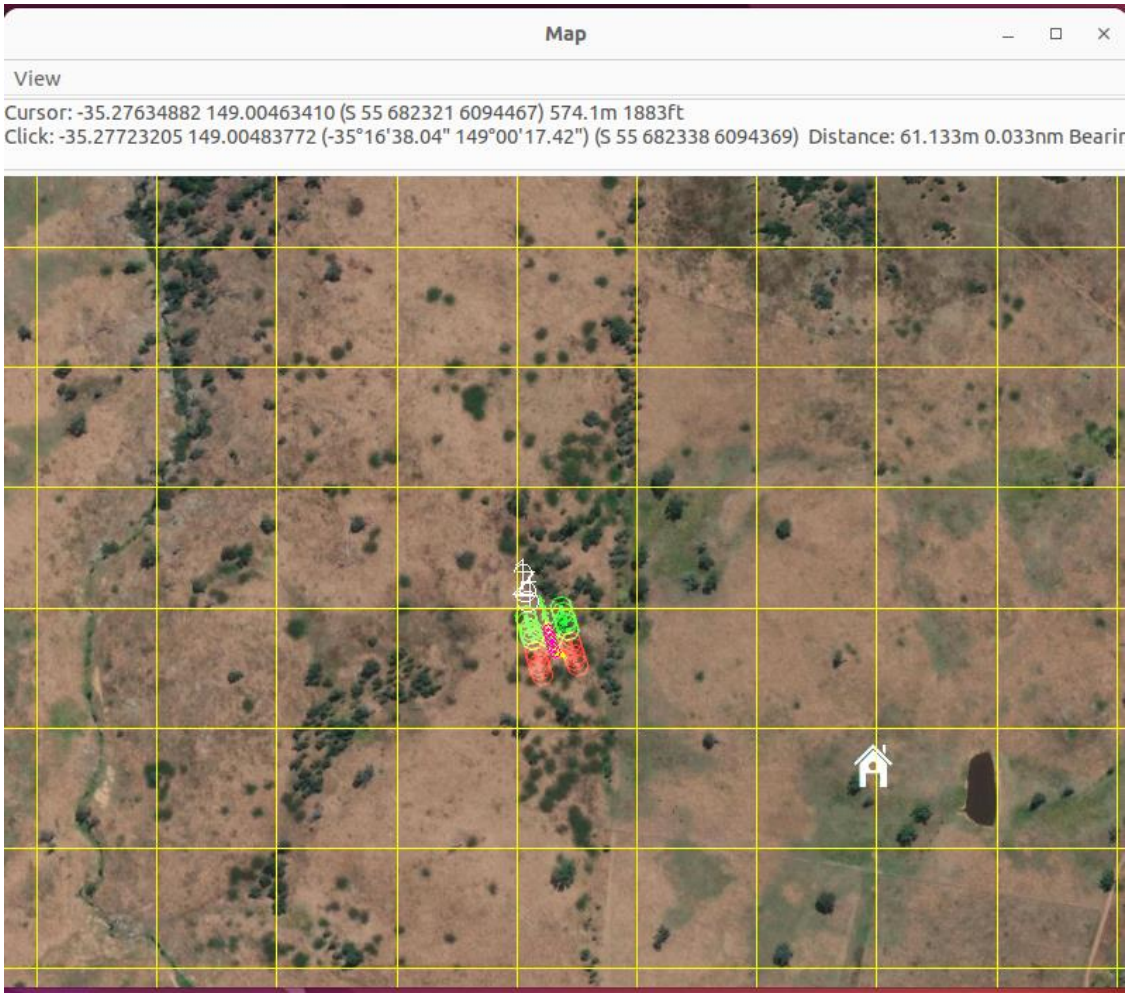


Рисунок 3.4 – Взліт та Follow для підконтрольних БЛА

Для перевірки Follow задамо координати в які повинен полетіти головний БЛА, а підконтрольні повинні слідувати за ним. Результат на рисунку3.5

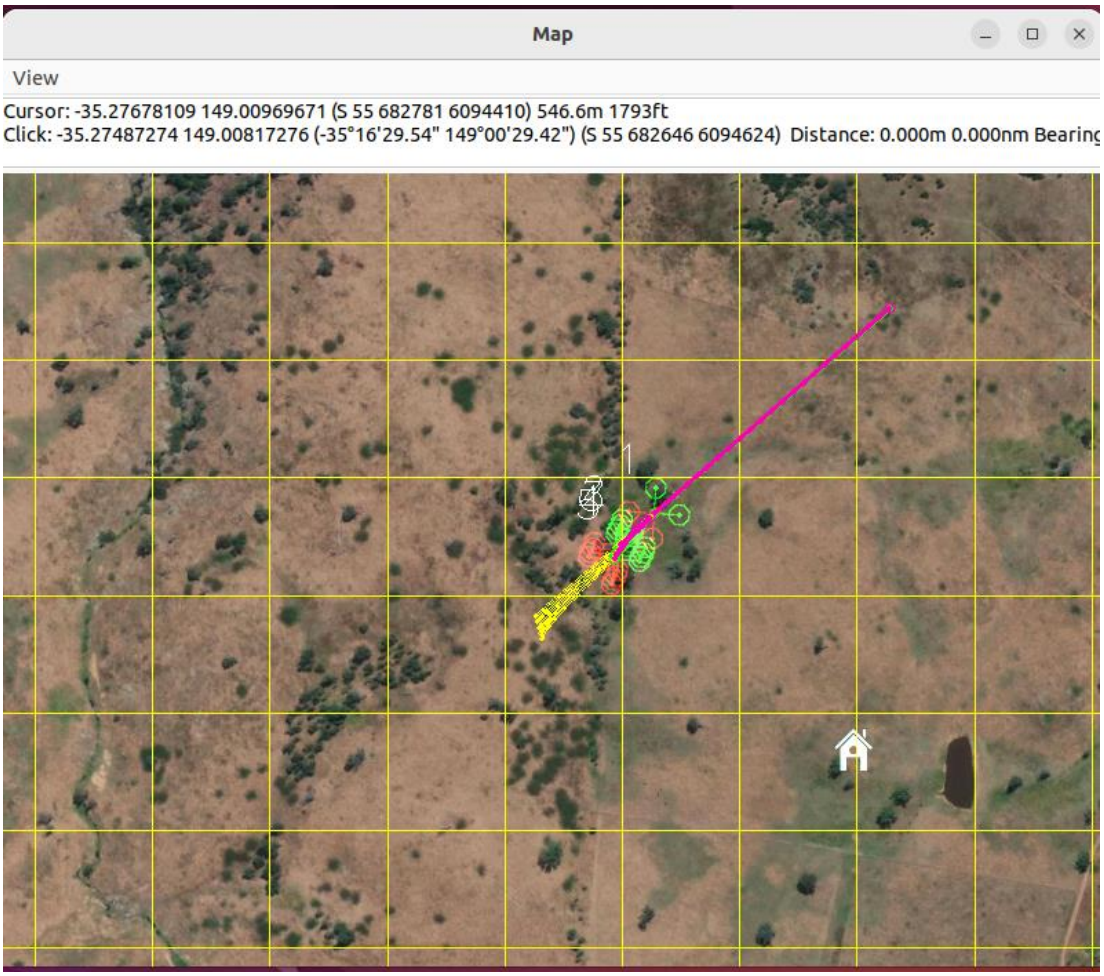


Рисунок 3.5 – Тестування Follow у реальному часі

Проведемо ще одну перевірку, коли підконтрольні БЛА стартують з різних координат. Результат на рисунку 3.6.

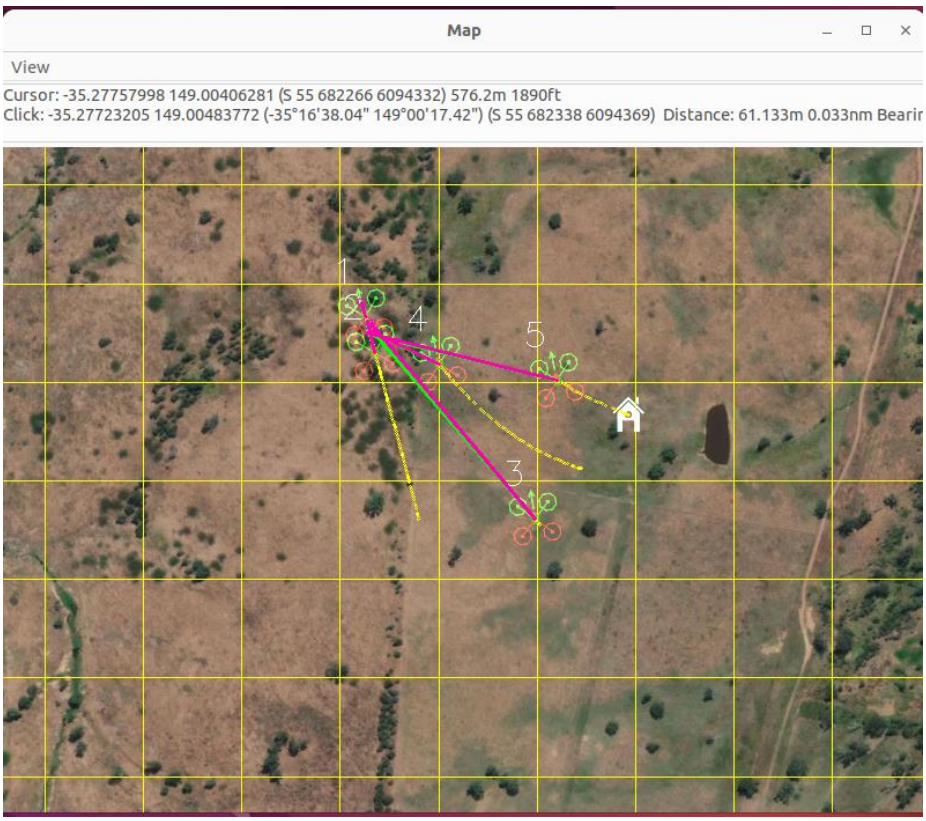


Рисунок 3.6 – Тестування Follow у реальному часі з різними початковими координатами у підконтрольних БЛА

Схематичний опис тестових сценаріїв можна побачити на рисунку 3.7.

Зазначу плюс та мінуси MAVLink Follow:

Плюси:

1)Є частиною стеку Ardupilot

2)Легко ділитися плануванням місії

3) Реалізовано за допомогою телеметрії UART2, що підтримується у всіх пілотних контролерах

Мінуси:

1) Одна станція наземного керування обмежена 254 БЛА (має 255 ID)
2) SPOF (Єдина точка відмови)

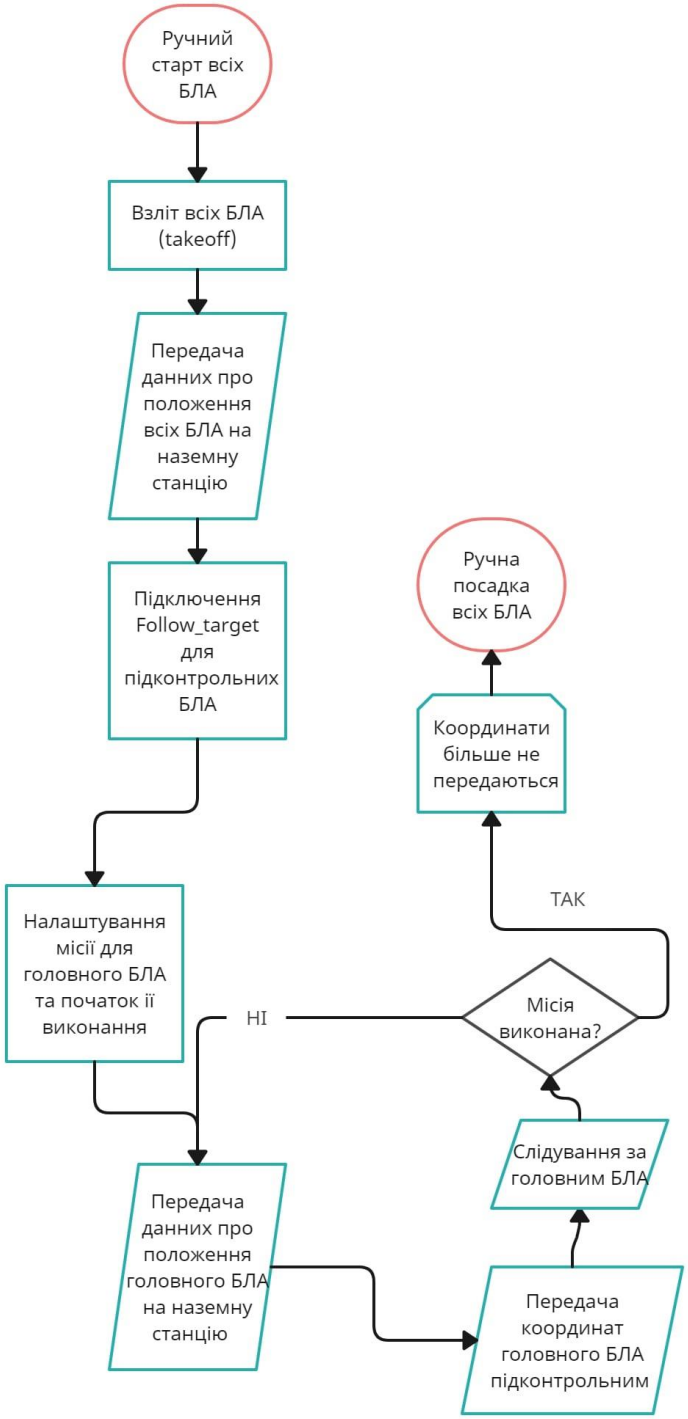


Рисунок 3.7 – Схематичне зображення тестового сценарію з MAVLink Follow

Цей сценарій є лише загальною рекомендацією і може вимагати адаптації для вашої конкретної конфігурації, моделі БЛА та потреб.

3.4 Польотний контролер

Польотний контролер - це пристрій, який контролює і стабілізує політ БЛА. Він розраховує і налаштовує складні параметри польоту, включаючи висоту, швидкість, напрямок і стабілізацію. Польотний контролер працює шляхом прийому інформації від різних датчиків, таких як гіроскопи, акселерометри, барометри, компаси. За допомогою цих даних контролер виробляє команди для регулювання швидкості обертання моторів БЛА, щоб забезпечити стабільний політ або виконати вказані маневри. Додатково, польотні контролери можуть мати розширені функції, такі як GPS навігація, автопілот, зміна польотних режимів, підтримка точки стабілізації (зупинки в повітрі), автоматичний поворот до точки відправлення.[13]

3.4.1 Концепція UART

UART (Universal Asynchronous Receiver/Transmitter) - це тип обладнання, який використовується для асинхронного обміну даними між різними пристроями через комп'ютерні шини або кабелі. Асинхронне в цьому контексті означає, що передача даних може відбуватися в будь-який момент часу, а не в певний проміжок часу. Коли дані передаються, вони супроводжуються сигналами "start" і "stop", які дозволяють приймачу визначити початок і кінець кожного байта. UART використовує два канали передачі: один для передачі даних (TX) і один для прийому даних (RX). Він може бути використаний для з'єднання різноманітних пристроїв, включаючи комп'ютери, модеми і мікроконтролери.[13]

Ardupilot HAL (Hardware Abstraction Layer) наразі визначає 8 UARTs. Сам HAL не встановлює ніяких конктретних ролей для цих UARTs, але інші частини ArduPilot припускають, що їм будуть призначені певні функції.[19]

Зазвичай у більшості польотних контролерів присутні декілька UARTs, які виконують ті чи інші призначенні їм функції, але у більш продвинутих контролерах UARTs наприклад може бути 8. Це зроблено для того щоб можна було підключати різну перефірію яка підтримує UART з’єднання для розширення функціоналу БЛА. Наприклад ми будем підключати RF модуль для побудови full-mesh з’єднання.

3.4.2 Pixhawk 2.4.8

Польотний контролер Pixhawk 2.4.8 підтримує протокол UART і має кілька портів UART, які можна використовувати для різних цілей. Наприклад, один з портів UART може бути використаний для з'єднання з GPS модулем. GPS модуль передає дані про своє місцезнаходження до Pixhawk по UART.

Інший порт UART може бути використаний для з'єднання з телеметрією, що передає дані про стан дрона і його положення на землі. Крім того, UART може бути використаний для з'єднання з різними сенсорами, такими як сенсори барометра або компаса, для збору даних про оточуюче середовище.

UART дозволяє Pixhawk обмінюватися даними з цими пристроями безпосередньо, без потреби в додатковому мікроконтролері або комп'ютері. Це полегшує інтеграцію різних компонентів системи і знижує загальну складність системи. На рисунку 3.8 можна побачити польотний контролер Pixhawk 2.4.8.

Pixhawk 2.4.8 може використовувати протокол MAVLink для взаємодії з наземною станцією керування, такою як QGroundControl або Mission Planner. Це дозволяє нам програмувати маршрути, налаштувати параметри польоту, переглядати дані телеметрії в режимі реального часу та виконувати інші завдання. Для програмного забезпечення, Pixhawk 2.4.8 використовує PX4 або ArduPilot. Обидва цих програмних стеки надають можливості для різноманітних режимів польоту.

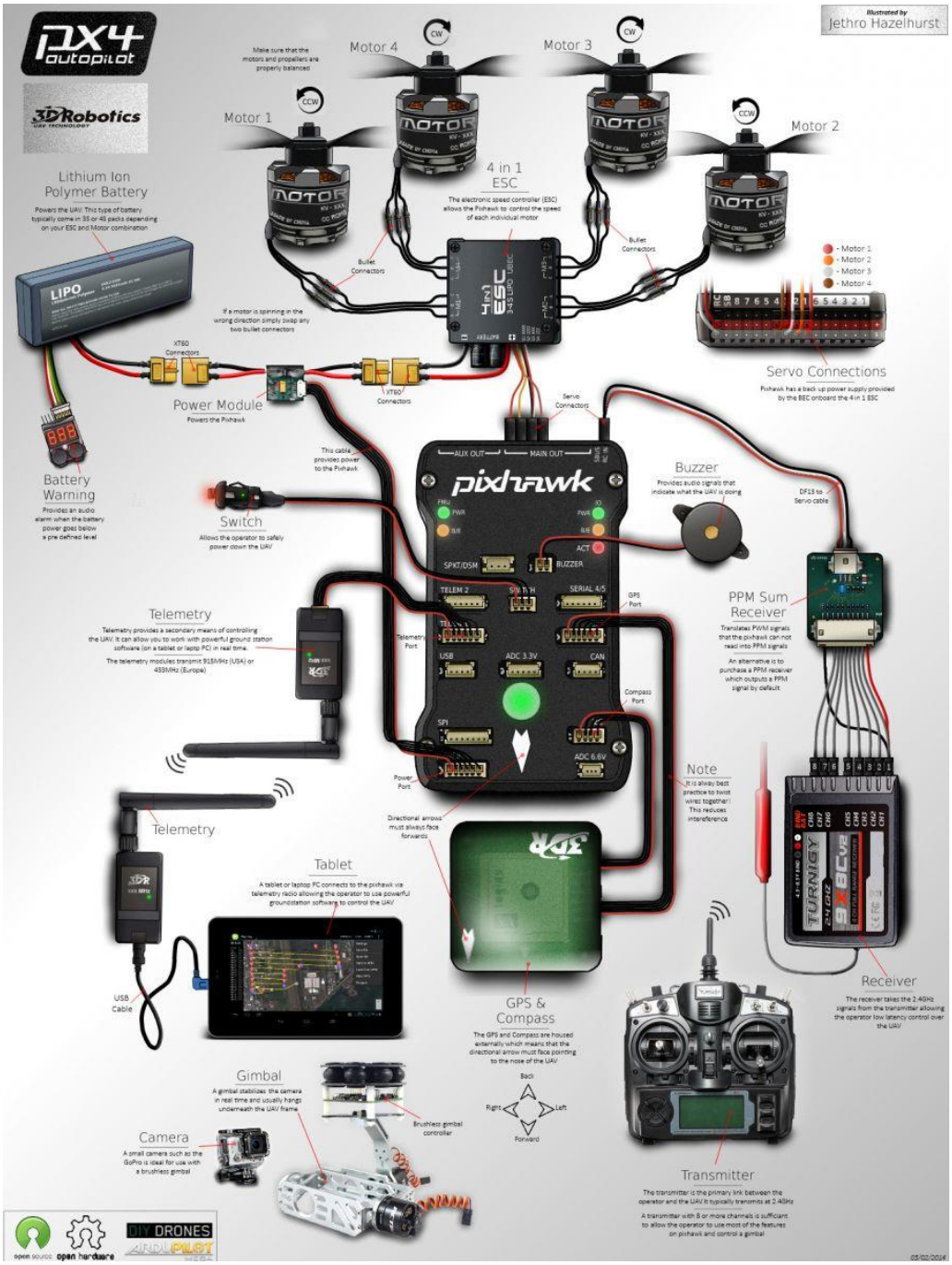


Рисунок 3.8 – Польотний контролер Pixhawk 2.4.8[20]

Саме на Pixhawk 2.4.8 можна буде протестувати розроблений модуль для організації зв'язку machine-to-machine для БЛА.

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі було обговорено реалізацію комунікації для рою безпілотних літальних апаратів (БЛА). Вибір операційної системи та налаштування вибраних інструментів були ключовими аспектами, що визначають успішність цієї реалізації. Було використано Linux/Ubuntu для SITL та середовища збірки, а також налаштовано Ardupilot. Тестування базового функціоналу, включаючи демонстрацію з використанням основного стеку Ardupilot та класичну комунікацію у рої БЛА за допомогою Follow, показало, що налаштування були виконані правильно. Нарешті, було розглянуто польотний контролер, включаючи концепцію UART і Pixhawk 2.4.8. Ці елементи разом створюють ефективну систему для реалізації комунікації між БЛА у рої.

РОЗДІЛ 4. РОЗРОБКА ТА ТЕСТУВАННЯ ПРОТОКОЛУ

4.1 Скриптинг на Lua

Lua - це потужна, ефективна та легка мова програмування, створена у 1993 році в Бразилії. Lua використовується для розширення програмного забезпечення, що потребує гнучкості або здатності до налаштування користувачами. Lua використовується в багатьох індустріях, включаючи відеоігри, вбудовані системи, веб-розробку та робототехніку.

В нашому випадку для реалізації machine-to-machine потрібен новий протокол та новий польотний контролер. Польотний контролер повинен мати вільні UART порти, щоб підключити пристрій для комунікації між БЛА. У Ardupilot HAL (Hardware Abstraction Layer) наразі вільні 3 UART порти до яких можна підключити свої пристрої, детальніше у таблиці 4.1.

Таблиця 4.1 – UARTs у Ardupilot та їх призначення[19]

ParamPrefix	Sim_vehicle Cmd Line	Def Role	Default Connection
SERIAL0_	- -uartA= or - -serial0=	Console	tcp:localhost:5760:wait
SERIAL1_	- -uartC= or - -serial1=	MAVLink	tcp:localhost:5762
SERIAL2_	- -uartD= or - -serial2=	MAVLink	tcp:localhost:5763
SERIAL3_	- -uartB= or - -serial3=	GPS	Simulated GPS
SERIAL4_	- -uartE= or - -serial4=	GPS	Simulated GPS
SERIAL5_	- -uartF= or - -serial5=		
SERIAL6_	- -uartG= or - -serial6=		
SERIAL7_	- -uartH= or - -serial7=		

Але стек Ardupilot дозволяє нам розширювати можливості польотного контролеру за допомогою драйверів, написаних на Lua. Скриптування надає безпечне середовище з обмеженнями, яке дозволяє додавати нові функції до автопілота без зміни основного коду. Скрипти зберігаються на SD-картці та виконуються паралельно з кодом управління польотами.

4.2 Hello World на Lua

Так як у нас замість польотного контролеру SITL, йому треба передати SERIAL порт з параметром та швидкість передачі даних як на рисунку 4.2:

```
SCR_ENABLE                                                    1
SERIAL7_PROTOCOL                                             28
SERIAL7_BAUD 9600
```

Рисунок 4.2 – Передача SERIAL порту та швидкості

Scr_enable 1 – включаємо скриптинг

SERIAL7_PROTOCOL 28 – 28 (Scripting) кажемо порту протокола для чого він буде застосовуватись [21]

SERIAL7_BAUD 9600 – 9600 біт/с це швидкість передачі даних

Щоб перевірити чи правильно все працює, напишемо на Lua скріпт який кожні 2 секунди буде виводити в консоль Hello world from Lua. Приклад скрипта на рисунку 4.3.

```lua
local function update()
    local ts = gps:time_week_ms(0)

    gcs:send_text(4, "Hello world from Lua. Time: " .. tostring(ts))

    return update, 2000
end

return update, 2000
```

Рисунок 4.3 – Тестовий скрипт для перевірки працездатності

Прописуємо у терміналі_1:

libraries/SITL/hello_world_demo/mavproxy.sh

Прописуємо у терміналі_2:

libraries/SITL/hello_world_demo/hellow_world_copter.sh

Маємо побачити у консолі MavProxy Hello world from Lua та час як на рисунку 4.4 (виділено помаранчевим для кращого бачення)

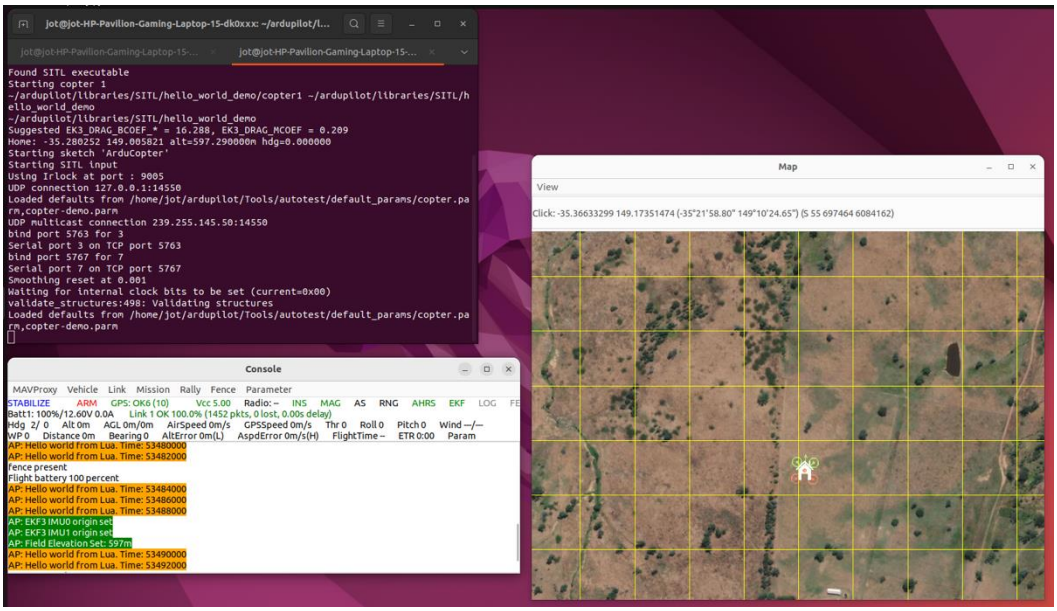


Рисунок 4.4 – Hello world from Lua у консолі MavProxy

4.3 Симуляція яка дозволяє БЛА обмінюватися повідомленнями

Так як в нас симуляція БЛА, то і радіо зв’язок треба як інструмент спілкування треба замінити на симуляцію спілкування. Для цього напишемо Python скрипт який буде симулювати поведінку напівдуплексної мережі. Приклад реалізації на рисунку 4.5.

```python
from socket import *

# Create a UDP socket
# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)

# Assign IP address and port number to socket
serverSocket.bind(('', 12000))

# simple simulation of RF half-duplex network
addresses = []

while True:
    # Receive the client packet along with the address it is coming from
    message, address = serverSocket.recvfrom(1024)

    # save it address list if not yet available
    if address not in addresses:
        print(str(address) + " in air")
        addresses.append(address)

    # send this message to all others except of sender
    for addr in addresses:
        if addr != address:
            serverSocket.sendto(message, addr)
```

Рисунок 4.5 – Скрипт для симуляції спілкування БЛА

Цей Python скрипт створює простий UDP сервер, який може приймати пакети від БЛА, зберігати їх адреси і пересилати прийняті повідомлення до всіх інших клієнтів, виключаючи відправника.

Алгоритм дій скрипту:

1)Створюємо новий UDP сокет, передаючи у socket() аргументи AF_INET (сімейство інтернет-адрес для IPv4) та SOCK_DGRAM (тип сокету для UDP)

2) Призначаємо сокету IP-адресу та порт. Пустий рядок ' ' вказує на те, що сервер буде слухати на всіх доступних інтерфейсах

3) Створюємо список адрес, які будуть зберігатися під час отримання пакетів

4) У нескінченному циклі сервер чекає на отримання пакета. Коли пакет приходить, він зберігає повідомлення та адресу відправника.

5) Додаємо адресу відправника до списку адрес.

6) Надсилаємо повідомлення кожному БЛА, крім відправника.

4.4 Дизайн та розробка протоколу machine-to-machine

Візьмемо за приклад обладнання під яке будемо розробляти протокол, а саме радіочастотний модуль APC220, який підтримує UART і може бути використаний для побудови бездротової мережі між БПЛА.

APC220 - це високоінтегрований напівдуплексний модуль прийомопередавача з низьким енергоспоживанням і високою швидкістю MCU та високопродуктивною радіочастотною мікросхемою.[23]

Міркування щодо протоколу:

- Повинен бути компактним, щоб вписатися в обмежену смугу пропускання 9600 бод 455 МГц

- Повинен бути простим в обслуговуванні

- Повинен справлятися з широко використовуваними цивільними частотами, на яких багато шуму і сміття

- Повинен працювати з енергозбереженням, пристрій може переходити в сплячий режим, якщо занадто довго неактивний

- Повинен працювати в напівдуплексному режимі - тільки один БЛА може надсилати дані у заданий часовий інтервал

Запропонований протокол:

- Магічний байт - щоб розбудити пристрій і визначити, що це наш пакет

- 7 байт - корисне (фактичне) навантаження

- байт контрольної суми CRC - для перевірки цілісності корисного навантаження

Для економії енергії ми будемо використовувати стандартну швидкість 9600 бод, що дасть нам реалістичні 960 байт/с[22]

Обмежимо наш протокол так, що кожен БЛА в одну секунду або відсилає дані або тільки отримує. Код якій керує протоколом викликається кожні 100 мсек, що гарантовано дає можливість хоча би раз бути викликаним протягом секунди. Стандартний MavLink протокол може сягати до 263 байт, але поширені повідомлення мають зазвичай 55-56 байт, що менш ніж 960 байт, що дозволяє навіть пересилати MavLink повідомлення від одного БЛА до іншого БЛА.

Для прототипу зафіксуємо розмір нашого повідомлення 9-байтами, формування повного пакету та його по-байтова відправка в порт займає 1-2 мсек згідно з замірами на SITL реалізації, що дає вдосталь часу на відсилку більш великого за розміром пакету, навіть враховуючи обмежені можливості процесора та накладні витрати на інтерпретатор Lua.

4.4.1 Схема протоколу

Зв'язок між БПЛА організовано за схемою з фіксованим часовим кроком. Повний цикл оновлення становить 10 секунд, коли тільки один БПЛА надсилає дані іншим щосекунди. Таким чином, ми можемо підтримувати до 10 БПЛА в цій схемі.

На рисунку 4.6 зображено схему з фіксованим кроком для БЛА 1 та БЛА 2:

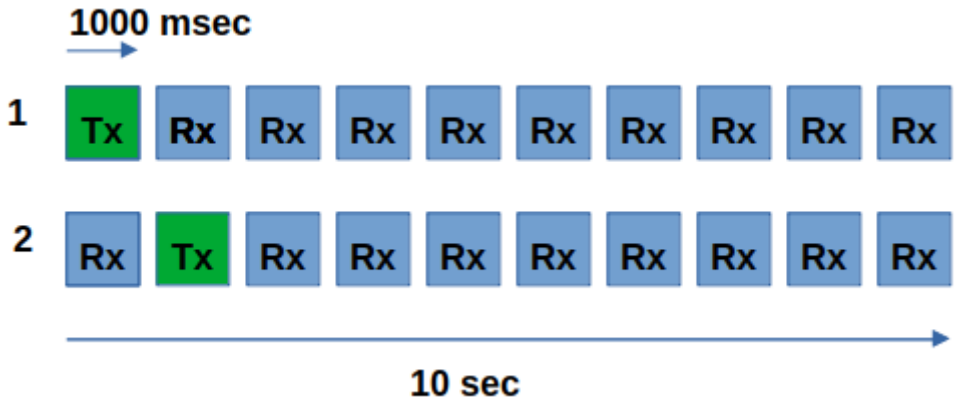


Рисунок 4.6 – Схема з фіксованим кроком для БЛА 1 та БЛА 2

- Протягом перших 1000 мс перший БЛА передає дані, тоді як інші намагаються лише приймати;
- Протягом наступних 1000 мс другий БЛА передає дані і так далі

Цикл оновлення Lua викликається раз на 100 мс, що дає нам достатню деталізацію для вимірювання часового кроку в секундах.

Ця схема піднімає дві проблеми:
- Як синхронізувати час на всіх БЛА?
- Як стабільно визначити, який БЛА є першим, а який другим?

4.4.2 Розрахунок кроку з фіксованим часом

Синхронізація часу здійснюється за допомогою GPS API, як описано у фрагменті коду: gps:time_week_ms(0)

Саме прив'язка Lua до GPS HW після ініціалізації GPS може надати глобально синхронізовану мітку часу у форматі мс. Синхронізувавши його на всіх БЛА, ми можемо легко розрахувати кількість часових кроків, використовуючи базову схему балансування навантаження на основі ділення за модулем (залишок від ділення на 10000 дає періодично 0 100 200 300 … 1000 1100 1200 1300 … 9900 потім знову 0 100 200 300 и так по колу).

Ось як це виглядає у коді:

local ts = gps:time_week_ms(0)

local step = ts % 10000

Тепер у нас є крок з фіксованим часом, який можна використовути на БЛА зі стабільним ID. Кожен дрон має спеціальний параметр - SYSID_THISMAV

Цей параметр починається від 1 до 255, де 255 зарезервовано для наземної станції керування і не може бути використаний дронами.

Ми можемо отримати цей ідентифікатор у Lua наступним чином:

local id = math.floor(param:get("SYSID_THISMAV"))

Отже, як тільки ми визначили, що настав час передавати дані, ми можемо серіалізувати повідомлення з деяким корисним навантаженням - наприклад, рядком "ping" і відправити його через інтерфейс UART байт за байтом, як показано на рисунку 4.7.

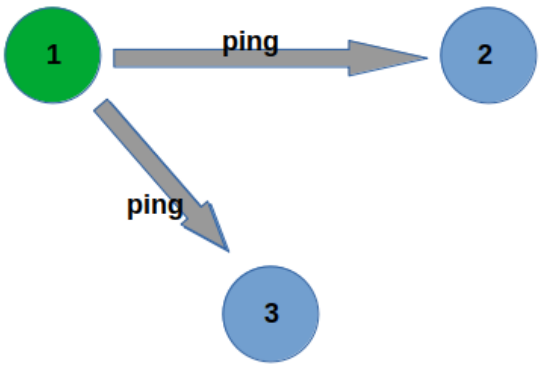


Рисунок 4.7 – Схематичне зображення передачі “ping” з першого БЛА іншим двум

4.4.3 Формат повідомлення

Тепер давайте розглянемо наш формат повідомлень, а потім подивимося на прикладі SITL, як БЛА надсилають та отримують дані.

Повідомлення між БЛА та БЛА - це, по суті, потік байт, закодований і надісланий через інтерфейс UART. Під кодуванням ми маємо на увазі

перетворення пакета в структуру даних з додаванням контрольних сум, ID БЛА та передачу його через канал байт за байтом.

Таблиця 4.8 – Опис формата повідомлення

Частина повідомлення	Байт	Опис
Заголовок	0	Магічне число - 15, якщо інше число, то це повідомлення пропускається
Заголовок	1	ID БЛА відправника з параметра SYSID_THISMAV, може бути використаний для визначення джерела повідомлення
Корисне навантаження	2-7	Фактичний фіксований розмір корисного навантаження
Контрольна сума	8	Контрольна сума CRC8 для перевірки цілісності корисного навантаження

Для спрощення демонстрації ми віддзеркалюємо відправлені та отримані повідомлення з БЛА на нашу консоль MavProxy за допомогою Lua API: gcs:send_text

У логах консолі MavProxy є два типи повідомлень:

- [крок] Відправлено з БЛА <ID> корисне навантаження:<повідомлення користувача>

- [крок] Отримано від БЛА <ID> корисне навантаження:<повідомлення користувача>

Таким чином, ми бачимо, що кожен БЛА робить тільки одну операцію передачі на своєму кроці, і ми бачимо, що БЛА 1 отримує дані від БЛА 2 і 3, БЛА 2 отримує дані від БЛА 1 і 3, а БЛА 3 отримує дані від БЛА 1 і 2.

4.5 Тестування протоколу machine-to-machine

Щоб перевірити працездатність протоколу я додав 3 БЛА на яких працює наш Lua драйве, нехай це будуть copter1, copter2 та copter3. До папки scripts яка у кожного БЛА своя, треба додати rf_driver.lua.

Тепер прописуємо у термінал_1:

libraries/SITL/UAV_full_mesh_demo/mavproxy.sh

Прописуємо у термінал_2:

libraries/SITL/UAV_full_mesh_demo/rf_sim.py

Прописуємо у термінал_3:

libraries/SITL/UAV_full_mesh_demo/full_mesh_copter.sh

І маємо побачити у себе на екрані теж саме як на рисунку 4.9

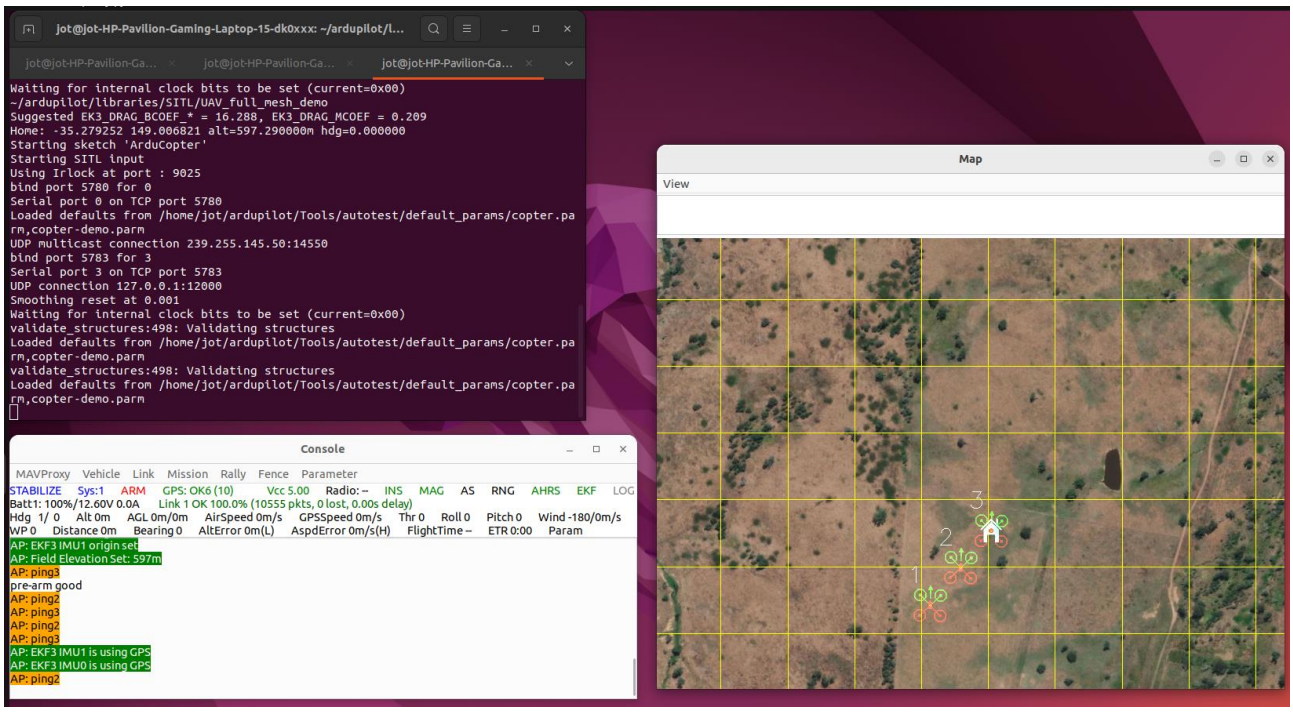


Рисунок 4.9 – Тестування протоколу (перший запуск)

Як ми бачимо є три БЛА, на рисунку 4.7 перед нами консоль UAV 1. Він надсилає дані з кроком 1000 і отримує дані від UAV 1 отримує повідомлення від UAV 2 та UAV 3 з їхніми кроками (ping у консолі).

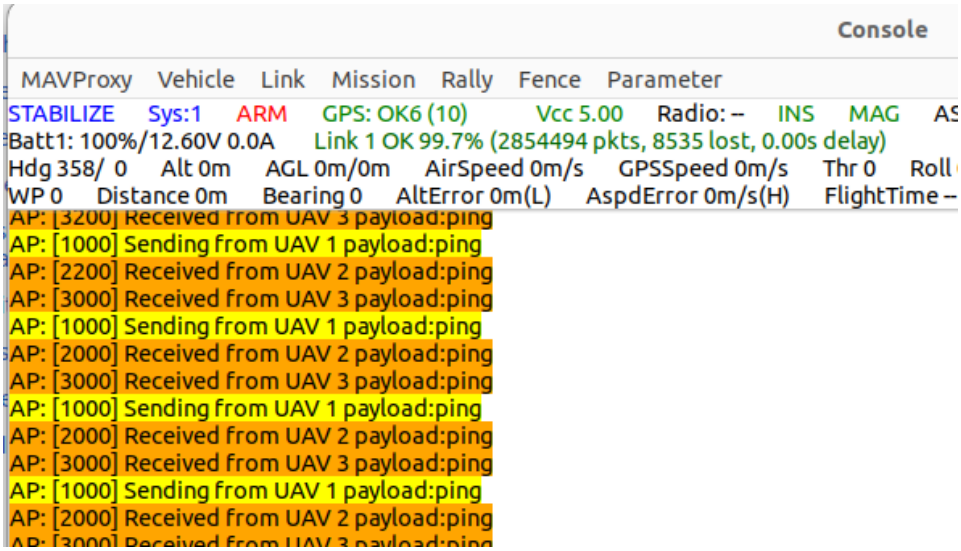


Рисунок 4.10 – Тестування протоколу для UAV 1

Змінимо у консолі Mavproxy UAV 1 на UAV 3 та очікуємо результат.

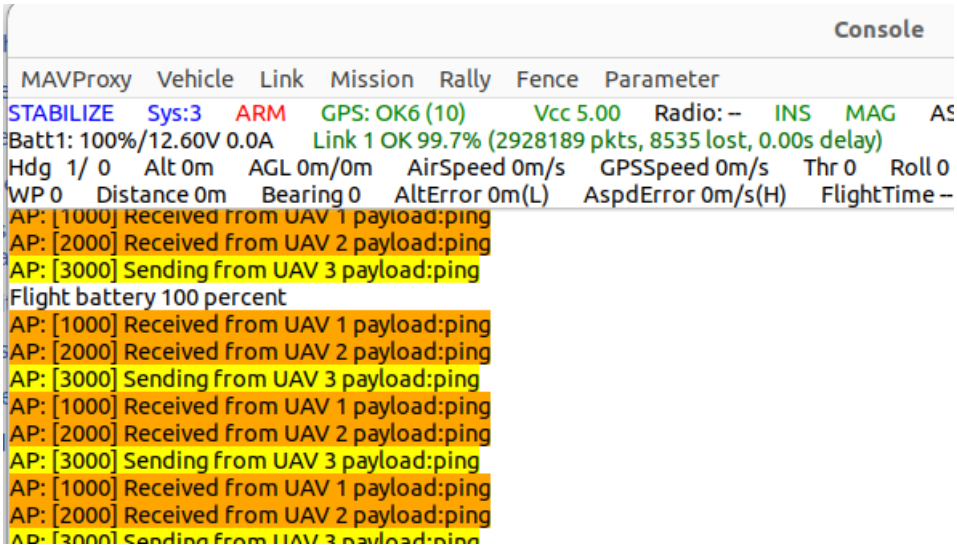


Рисунок 4.11 – Тестування протоколу для UAV 3

Тепер консоль показує дані з UAV 3, і як ми бачимо, UAV 3 надсилає дані з кроком 3000 мс, в той час як UAV 1 і 2 отримують дані з іншими кроками. Результат на рисунку 4.8.

ВИСНОВКИ ДО РОЗДІЛУ 4

Розділ 4 був присвячений розробці та тестуванню протоколу machine-to-machine, у контексті БЛА. На початкових етапах було проведено знайомство з мовою програмування Lua за допомогою розробки простого "Hello World" скрипту, що поклало основу для подальшої роботи з більш складними скриптами. Було створено симуляцію, яка дозволяє БЛА обмінюватися повідомленнями. Цей етап був важливим, оскільки він відтворював реальний сценарій комунікації між БЛА. У процесі дизайну та розробки протоколу machine-to-machine було створено схему протоколу, розраховано крок з фіксованим часом і встановлено формат повідомлень, що дозволило реалізувати ефективну взаємодію між БЛА. На заключному етапі проведено тестування розробленого протоколу. Результати тестування показали, що протокол machine-to-machine надійно працює в рамках розробленої симуляції.

ВИСНОВКИ

Існуючі рішення для побудови взаємодії між БЛА без використання наземної станції виявлено у вигляді наукових робіт, які викладали теоретичні основи побудови мережі між БЛА але не містили деталей реалізації та підходів. Було зроблено дослідження існуючих методів розробки програмного забезпечення для БЛА які знаходяться у відкритому доступі. Була реалізована симуляція мережі комунікації між БЛА на основі технології Ardupilot, яка є стеком з відкритим вихідним кодом для пілотних транспортних засобів. Розробка драйвера велася за допомогою симуляції БЛА SILT яка дозволяє писати програмне забезпечення без використання реального обладнання. Результати, отримані за допомогою розробленої системи, дають можливість перейти до наступного кроку - реалізації цієї схеми на реальних моделях БЛА, які мають відкрите апаратне забезпечення і підтримуються стеком Ardupilot. Розроблений протокол дозволяє обмінюватися різною інформацією між БЛА такими як GPS координати, висота, швидкість та інші данні які будуть потрібні для вирішення задач які потребуються для більш автономної координації БЛА без використання наземної станції керування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cast N. How Drones Communicate With the Controller? [Електронний ресурс] / Nick Cast // Remoteflyer Private Linited. – 2022. – Режим доступу до ресурсу: https://www.remoteflyer.com/how-drones-communicate-with-the-controller/.
2. Xi C. Review of Unmanned Aerial Vehicle Swarm Communication Architectures and Routing Protocols [Електронний ресурс] / C. Xi, T. Jun, L. Songyang // MDPI. – 2020. – Режим доступу до ресурсу: https://www.mdpi.com/2076-3417/10/10/3661.
3. Posea P. Do you need wifi/internet to fly a drone? [Електронний ресурс] / Paul Posea // Dronesgator. – 2023. – Режим доступу до ресурсу: https://dronesgator.com/need-wifi-to-fly-a-drone/.
4. Valdovinos J. M. LOW-COST UAV SWARM FOR REAL-TIME OBJECT DETECTION APPLICATIONS : дис. канд. техн. наук / Valdovinos Joel Miranda – San Luis Obispo, 2022. – 96 с.
5. ROS/Gazebo based simulation of co-operative UAVs : дис. канд. техн. наук / Bernardeschi Cinzia – Pisa, 2018. – 16 с.
6. Campion M. A Review and Future Directions of UAV Swarm Communication Architectures : дис. канд. техн. наук : EIT / Campion Mitch – Grand Forks, 2018. – 6 с.
7. Bęben K. ARCHITECTURE OF THE UAV SWARM COMMAND AND CONTROL SYSTEMS – AN OVERVIEW : дис. докт. техн. наук / Bęben Karol – Warsaw, 2021. – 18 с.
8. Рогачёв М. Разбираемся в MAVLink. Часть 1 [Електронний ресурс] / Михаил Рогачёв // mrogachev. – 2016. – Режим доступу до ресурсу: https://habr.com/ru/articles/312300/.

9. SITL Simulator (Software in the Loop) [Електронний ресурс] // ArduPilot. – 2023. – Режим доступу до ресурсу: https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html.
10. Gazebo documentation [Електронний ресурс] // Gazebo. – 2023. – Режим доступу до ресурсу: https://gazebosim.org/docs.
11. Top Simulation Tools for Drone Development [Електронний ресурс] // Adinkra Inc – Режим доступу до ресурсу: https://adinkratech.com/top-drone-simulation-solutions/.
12. QGroundControl documentation [Електронний ресурс] // QGroundControl. – 2023. – Режим доступу до ресурсу: https://docs.qgroundcontrol.com/master/en/.
13. ChatGPT [Електронний ресурс] // openai. – 2022. – Режим доступу до ресурсу: https://chat.openai.com/.
14. Communication Architectures [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://encyclopedia.pub/entry/990.
15. Setting up the Build Environment (Linux/Ubuntu) [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://ardupilot.org/dev/docs/building-setup-linux.html#building-setup-linux.
16. Building ArduPilot [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://github.com/ArduPilot/ardupilot/blob/master/BUILD.md.
17. Waf (build system) [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Waf_(build_system).
18. MAVLink Messages [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://mavlink.io/en/messages/common.html#messages.
19. UARTs and the Console [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://ardupilot.org/dev/docs/learning-ardupilot-uarts-and-the-console.html.
20. Польотний контролер Pixhawk 2.4.8 [Електронний ресурс] – Режим доступу до ресурсу: https://greenchip.com.ua/0-0-1546-0.html.

21. Complete Parameter List [Електронний ресурс] – Режим доступу до ресурсу: https://ardupilot.org/dev/docs/AP_Periph-Parameters.html#serial1-protocol.
22. Most common baud rates table [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://lucidar.me/en/serialib/most-used-baud-rates-table/.
23. APC220 ISM Transparent Transceiver Module [Електронний ресурс] // AppconWireless. – 2008. – Режим доступу до ресурсу: https://www.appconwireless.com/uploadfile/20180605223627.pdf.
24. Plan View - GeoFence [Електронний ресурс] – Режим доступу до ресурсу: https://docs.qgroundcontrol.com/master/en/PlanView/PlanGeoFence.html.

ДОДАТОК 1

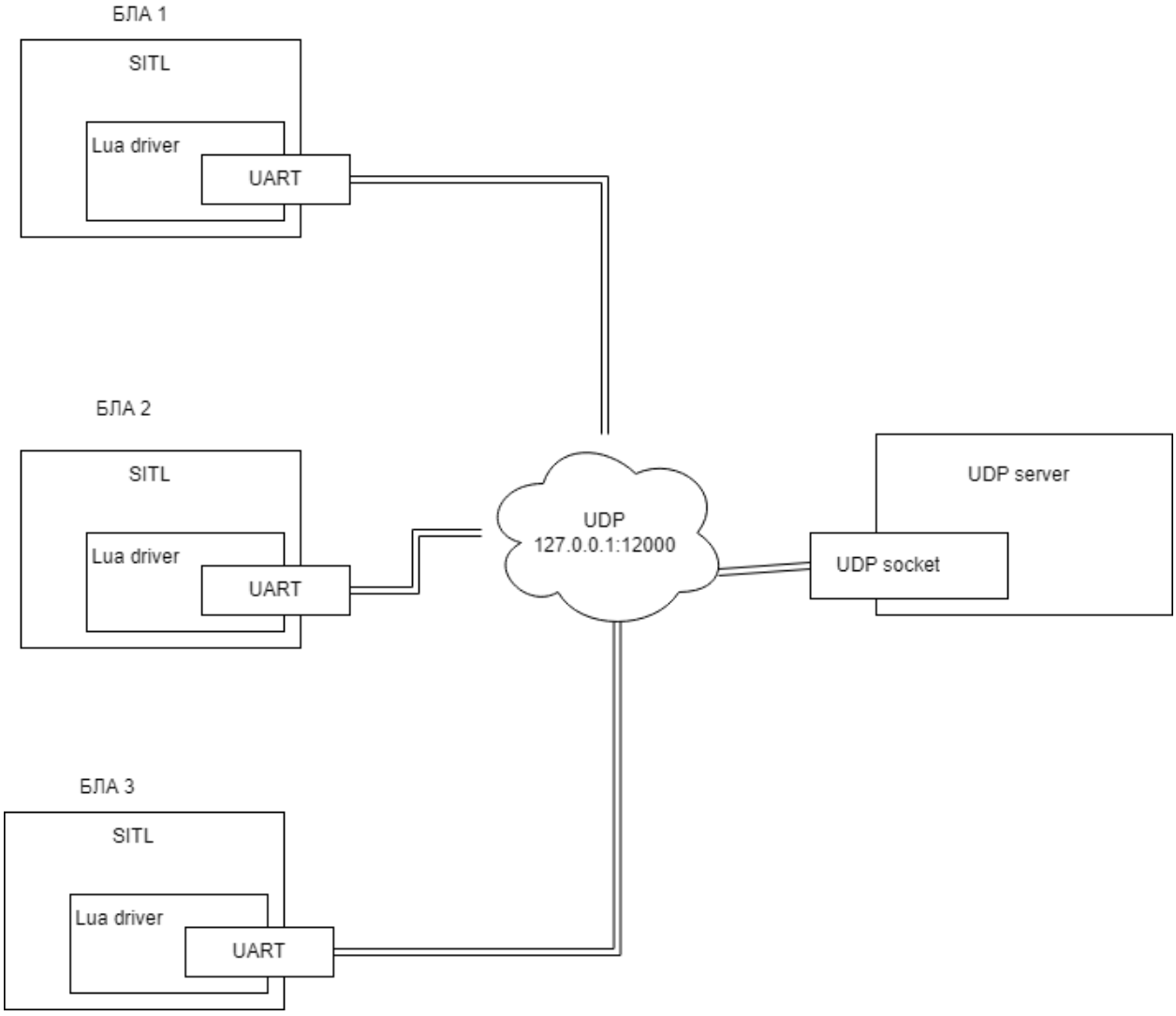
Модуль для організації зв'язку machine-to-machine для БЛА

Основні компоненти модуля (Структурна схема)

ІАЛЦ.467100.004 Д1

Аркушів 1

Київ 2023 р



					ІАЛЦ.467100.004 Д1
Зм.	*Арк.*	*№ докум.*	*Підп.*	*Дата*	
Розроб.		*Рустамов А.С.*			*Модуль для організації зв'язку machine-to-machine для БЛА* Основні компоненти модуля (структурна схема)
Перевір.		*Роковий О.П.*			
Н. контр.		*Виноградов Ю.М.*			
Затверд.					

Літ.	*Арк.*	*Аркушів*
	1	1

КПІ ім. Ігоря Сікорського

ФІОТ ІВ-93

ДОДАТОК 2

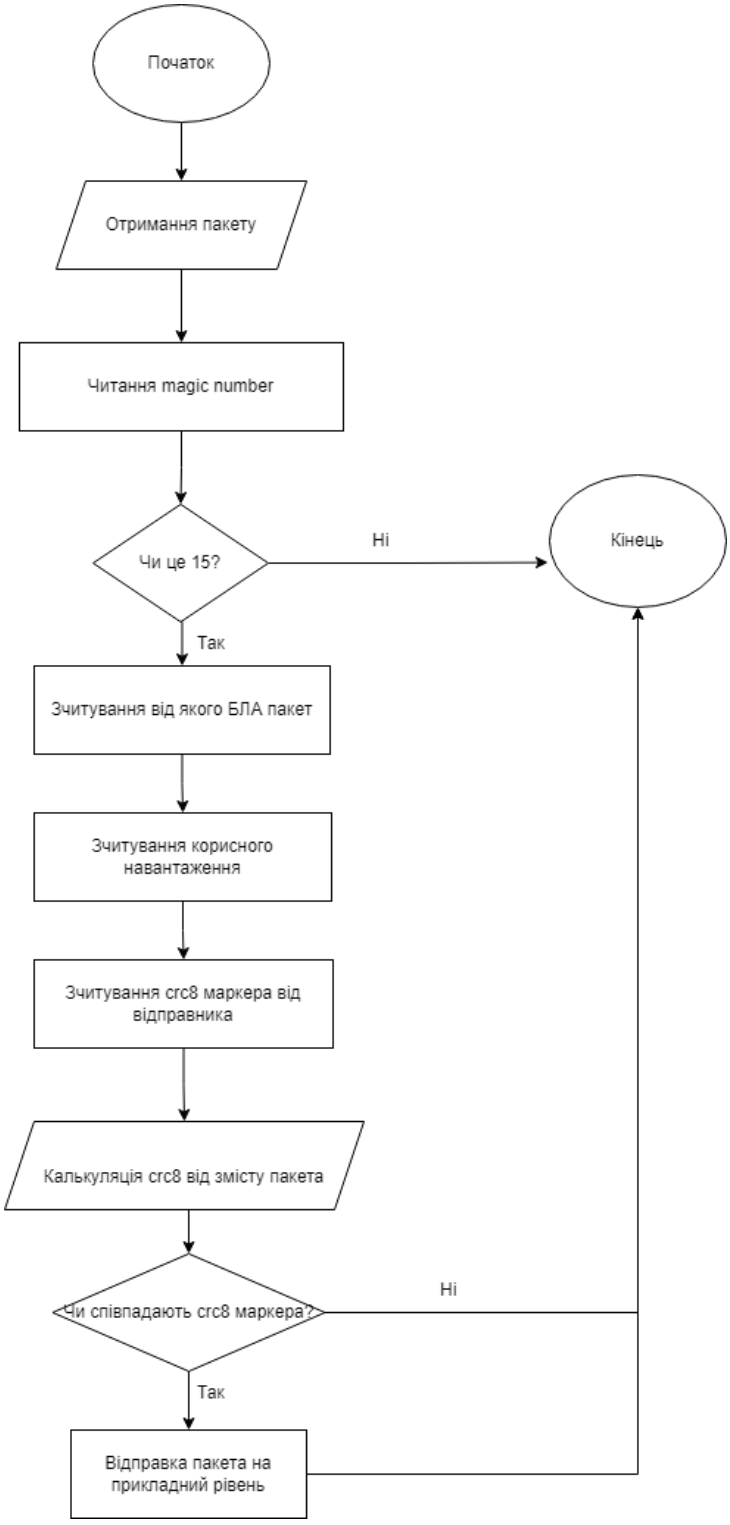
Модуль для організації зв'язку machine-to-machine для БЛА

Алгоритм протоколу (Функціональна схема)

ІАЛЦ.467100.005 Д2

Аркушів 1

Київ 2023 р



					ІАЛЦ.467100.005 Д2
Зм.	Арк.	№ докум.	Підп.	Дата	
Розроб.		Рустамов А.С.			*Модуль для організації зв'язку machine-to-machine для БЛА* Алгоритм протоколу (функціональна схема)
Перевір.		Роковий О.П.			
Н. контр.		Виноградов Ю.М.			
Затверд.					

Літ.	Арк.	Аркушів
	1	1

КПІ ім. Ігоря Сікорського

ФІОТ ІВ-93

ДОДАТОК 3

Модуль для організації зв'язку machine-to-machine для БЛА

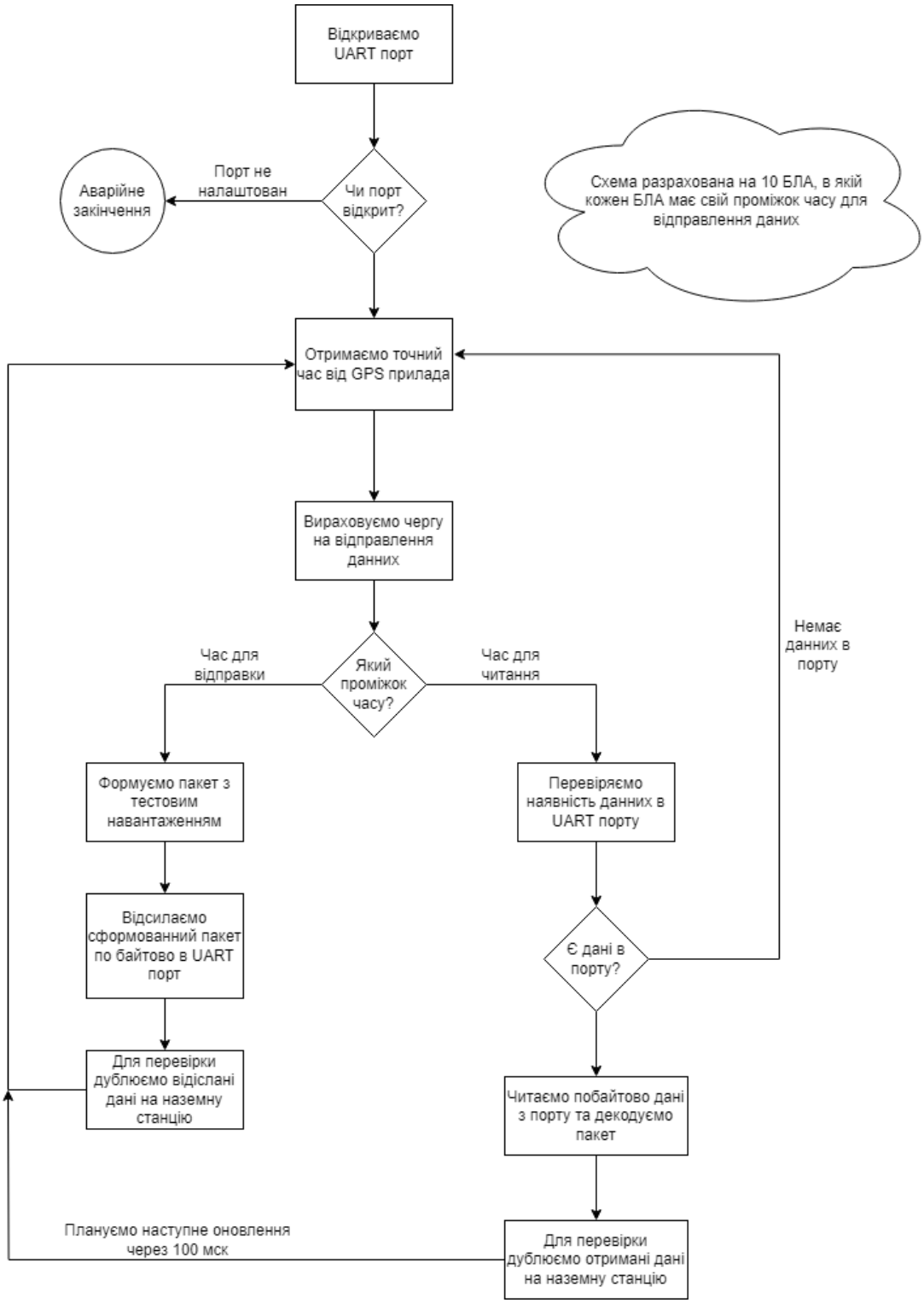
Алгоритм дій програмного забезпечення

(Принципова схема)

ІАЛЦ.467100.006 Д3

Аркушів 1

Київ 2023 р



Зм.	Арк.	№ докум.	Підп.	Дата	ІАЛЦ.467100.006 Д3		
Розроб.		Рустамов А.С.			*Модуль для організації зв'язку machine-to-machine для БЛА* Алгоритм дії програмного забезпечення (принципова схема)	Літ.	Арк. 1 / Аркушів 1
Перевір.		Роковий О.П.					
Н. контр.		Виноградов Ю.М.				КПІ ім. Ігоря Сікорського ФІОТ ІВ-93	
Затверд.							

ДОДАТОК 4

Модуль для організації зв'язку machine-to-machine для БЛА

Текст програмного коду

ІАЛЦ.467100.007 Д4

Аркушів 2

Київ 2023 р

```
-- Lua driver to communicate via 9600 baud half-duplex media

gcs:enable_high_latency_connections(true)

local port = serial:find_serial(0)

port:begin(9600)
port:set_flow_control(0)

local magic = 15
local is_ticked = false
local id = math.floor(param:get("SYSID_THISMAV"))

local function crc8(t)
   local c = 0
   for _, b in ipairs(t) do
      for i = 0, 7 do
         c = c >> 1 ~ ((c ~ b >> i) & 1) * 0x8C
      end
   end
   return c
end

local function encode_packet(payload)
   -- get fixed length packet from payload
   local payload_packet = {}
   local n_bytes = math.min(#payload, 6)
   for idx = 1, n_bytes do
      table.insert(payload_packet, string.byte(payload, idx))
   end
   while n_bytes < 6 do
      table.insert(payload_packet, 0)
      n_bytes = n_bytes + 1
   end

   -- form full packet:
   --   magic
   --   from id
   --   payload
   --   crc8
   local packet = {}
   table.insert(packet, magic)
   table.insert(packet, id)
   for idx = 1, #payload_packet do
      table.insert(packet, payload_packet[idx])
   end

   local crc = crc8(payload_packet)
   table.insert(packet, crc)
   return packet
end

local function decode_packet(buffer)
   -- check for magic byte
   if #buffer > 0 and buffer[1] == magic then
      local payload = {}
      local n_bytes = #buffer
      local crc_rx = buffer[n_bytes]

      for idx = 3, n_bytes - 1 do
         table.insert(payload, buffer[idx])
      end
      local crx = crc8(payload)
```

```
      if crx == crc_rx then
         return payload
      end
   end
   return {} -- empty payload means either not our packet or currupted
end

 local function update()
   local ts = gps:time_week_ms(0)
   local step = ts % 10000

   if step >= id*1000 and step < (id + 1)*1000 then
      if not is_ticked then
         -- this is our time slot to send whather we want
         local test_text_payload = "ping"

         local start_time = millis():toint()
         local tx_buffer = encode_packet(test_text_payload)

         for idx = 1, #tx_buffer do
            port:write(tx_buffer[idx])
         end
         local diff_time = millis():toint()
         gcs:send_text(5, string.format("[%s:%i:%i] Sending from UAV %i payload:%s", step, start_time, diff_time,
id, test_text_payload))
         is_ticked = true
      end
   else
      -- this is our time slots for receive data
      local rx_buffer = {}
      local n_bytes = port:available()
      while n_bytes > 0 do
         table.insert(rx_buffer, port:read())
         n_bytes = n_bytes - 1
      end
      local payload  = decode_packet(rx_buffer)
      if #payload > 0 then
         local msg = string.char(table.unpack(payload))
         gcs:send_text(4, string.format("[%s] Received from UAV %i payload:%s", step, rx_buffer[2], msg))
      end
      is_ticked = false
   end

   return update, 100
end

return update, 100
```