МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри,
д-р техн. наук, проф.

_____ Ірина ЖУРАВСЬКА

« __ » _____ 202__ р.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

РОЗПОДІЛЕНА СИСТЕМА ДЕТЕКТУВАННЯ РУХОМИХ ОБ'ЄКТІВ ТА ОБМІНУ ІНФОРМАЦІЄЮ НА БАЗІ LORAWAN

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія»

Здобувач _____Даніїл БАСОВ
підпис

«__» _____ 202__ р.

Керівник канд. фіз.-мат. наук, доцент _____Сергій ПУЗИРЬОВ
підпис

«__» _____ 202__ р.

Миколаїв – 2024

Факультет	Комп’ютерних наук
Кафедра	Комп’ютерної інженерії
Рівень вищої освіти	Другий (магістерський)
Освітній ступень	Магістр
Спеціальність	123 Комп’ютерна інженерія
Освітня програма	Комп’ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри комп’ютерної інженерії
_____ Ірина ЖУРАВСЬКА
«___»_____ 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Басова Даніїла Євгенійовича
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи
Розподілена система детектування рухомих об'єктів та обміну інформацією на базі LoRaWAN

Затверджена наказом ректора ЧНУ ім. Петра Могили від 16.09.2024 № 236.

2. Строк представлення кваліфікаційної роботи «____» _____ 202__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Очікуваним результатом роботи є: розподілений апаратно-програмний комплекс, що використовує штучний інтелект для детектування рухомих об'єктів (у вигляді нейронної мережі) та протокол LoRaWAN для обміну інформацією.

4. Перелік питань, що підлягають розробці
– аналіз предметної області кваліфікаційної роботи;
– огляд існуючої наукової літератури на тему КМР;
– аналіз сучасних аналогічних систем та рішень;
– визначення вимог до розподіленої системи, що розробляється;
– розробка концептуального рішення розподіленої системи, виходячи з визначених вимог;
– вибір апаратних та програмних засобів розробки розподіленої системи;
– розробка апаратно-програмного забезпечення розподіленої системи;

– тестування працездатності розробленого апаратно-програмного забезпечення.

5. Перелік графічних матеріалів
Слайди презентації

6. Завдання до спеціальної частини

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи

Керівник роботи _____ Сергій ПУЗИРЬОВ
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ Данііл БАСОВ
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Дата видачі завдання «_17_» _____вересня_____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної магістерської роботи

Тема: Розподілена система детектування рухомих об'єктів та обміну інформацією на базі LoRaWAN

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КМР	01.09.2024	16.09.2024	*Виконано*
2.	Огляд літератури за темою роботи	17.09.2024	26.09.2024	*Виконано*
3.	Складання календарного плану КМР	27.09.2024	29.09.2024	*Виконано*
4.	Аналіз предметної області	30.09.2024	07.10.2024	*Виконано*
5.	Розробка проєктних рішень	08.10.2024	14.10.2024	*Виконано*
6.	Моделювання та конструювання АПЗ	15.10.2024	07.11.2024	*Виконано*
7.	Перевірка працездатності, тестування та апробація розробленого АПЗ, аналіз результатів тестування, розробка керівництва користувача	08.11.2024	21.11.2024	*Виконано*
8.	Відгук керівника КМР	21.11.2024	22.11.2024	*Виконано*
9.	Оформлення КМР та презентації	23.11.2024	27.11.2024	*Виконано*
10.	Попередній захист	28.11.2024	28.11.2024	*Виконано*
11.	Рецензування	29.11.2024	02.12.2024	*Виконано*
12.	Завершення оформлення КМР та презентації	03.12.2024	12.12.2024	*Виконано*
13.	Захист кваліфікаційної роботи	19.12.2024	19.12.2024	*Виконано*

Керівник роботи _____ Сергій ПУЗИРЬОВ
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ Даніїл БАСОВ
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

АНОТАЦІЯ

до кваліфікаційної магістерської роботи
«Розподілена система детектування рухомих об'єктів та обміну інформацією на базі LoRaWAN»
Здобувач гр. 605: Басов Даніїл Євгенійович
Керівник: канд. фіз.-мат. наук, доцент Пузирьов С. В.

Актуальність теми кваліфікаційної магістерської роботи. Останні декілька років характеризуються все зростаючим інтересом до безпілотних апаратів (дронів). Їх використання вже революціонізувало сферу національної безпеки. Незабаром постане питання їх активного цивільного використання, що робить актуальним попереднє узагальнення зроблених напрацювань та подальше удосконалення технологій та способів використання дронів.

Іншим технологічним напрямком, що зараз стрімко розвивається, є штучний інтелект. Переваги «розумних» технологій, здатних навчатися за допомогою алгоритмів машинного навчання, сприяють їх широкій інтеграції у все нові сфери людського життя. Поєднання штучного інтелекту та безпілотних апаратів є важливим та перспективним напрямом у технологічній сфері.

Об’єкт дослідження: технології та методи розробки розподілених систем обробки даних.

Предмет дослідження: розподілена система детектування рухомих об'єктів та обміну інформацією на базі LoRaWAN.

Мета: створення розподіленого апаратно-програмного комплексу, що використовує штучний інтелект (у вигляді нейронної мережі) та протокол LoRaWAN.

Дана кваліфікаційна робота була апробована на XXI Міжнародній науковій конференції «Ольвійський форум – 2024: стратегії країн Причорноморського регіону в геополітичному просторі». Матеріали доповіді були опубліковані у збірнику конференції.

Кваліфікаційна робота містить: перелік скорочень, вступ, чотири розділи, висновок, перелік джерел посилання та два додатки.

Вступ містить обґрунтування актуальності розробки обраної теми, об’єкт, предмет дослідження, мету та завдання які необхідно виконати для досягнення поставленої мети.

В першому розділі проведено аналіз предметної сфери, необхідної для розробки розподіленої системи, а також наведено специфікацію вимог до апаратно-програмного комплексу.

Другий розділ містить опис процесу проєктування апаратно-програмного комплексу, а саме: розробку концептуального рішення, вибір архітектури та моделі нейронної мережі, вибір апаратного забезпечення розподіленої системи та опис його підключення.

Третій розділ містить опис розробки апаратно-програмного комплексу, який включає в себе нейронну мережу, програму детектування, програмне забезпечення P2P-мережі та мережі LoRaWAN, а також базовий графічний інтерфейс клієнтського застосунку.

У четвертому розділі проведено тестування та перевірка працездатності ключових елементів розробленого АПЗ.

У висновку описано результати виконання кваліфікаційної роботи.

Додатки містять код програмного забезпечення та інформацію про апробацію кваліфікаційної магістерської роботи.

Кваліфікаційна робота містить 73 сторінки (без додатків), 64 рисунки, 37 джерел посилання, 2 додатки.

Ключові слова: розподілена система, комп’ютерний зір, детектування об’єктів, детектування рухомих об’єктів, LoRa, LoRaWAN, безпілотні апарати, рій безпілотних апаратів.

ABSTRACT

of the Master's Thesis
"Distributed system for detecting moving objects and exchanging information based on LoRaWAN"
Applicant: Basov Daniil Yevheniiovych
Supervisor: Cand. Sci. (Phys.-Math.), Docent Puzyrov S. V.

Relevance of the topic of the master's thesis. The last few years have been characterized by a growing interest in unmanned vehicles (drones). Their use has already revolutionized the national security sphere. Soon, the question of their active civilian use will arise, which makes it important to summarize the developments made and further improve the technologies and methods of using drones.

Another rapidly developing technological area is artificial intelligence. The advantages of smart technologies that can learn using machine learning algorithms contribute to their widespread integration into more and more areas of human life. The combination of artificial intelligence and unmanned vehicles is an important and promising area in the technological sphere.

Object of research: technologies and methods for developing distributed data processing systems.

Subject of research: a distributed system for detecting moving objects and exchanging information based on LoRaWAN.

Objective of the master's thesis: to create a distributed hardware and software system that uses artificial intelligence (in a form on neural network) and the LoRaWAN protocol.

This master's thesis was approbated on XXI International Scientific Conference "Olvian Forum – 2024: Strategies of the countries of the Black Sea region in the geopolitical space". The materials of the report were published in the conference proceedings.

The master's thesis contains a list of abbreviations, an introduction, four sections, a conclusion, a list of references and two appendices.

The introduction contains the main justifications for the relevance of the development of the chosen topic, the object, subject of research, the purpose and tasks to be performed to achieve the goal.

The first section provides an analysis of the subject field required for the development of a distributed system, as well as a specification of the requirements for the hardware-software complex.

The second section contains a description of the process of designing a hardware-software complex, namely: development of a conceptual solution, choice of an architecture and a model of a neural network, choice of distributed system hardware and a description of its connection.

The third section describes the development of the hardware-software complex, which includes a neural network, a detection program, P2P and LoRaWAN network software, and a basic graphical interface of the client application.

In the fourth section, testing and performance verification of the key elements of the developed hardware-software complex was carried out.

The conclusion describes the results of the master's thesis.

The appendices contain the software code and information on the approbation of the master's thesis.

The master's thesis contains 73 pages (without appendices), 64 figures, 37 references, 2 appendices.

Keywords: distributed system, computer vision, object detection, moving object detection, LoRa, LoRaWAN, unmanned vehicles, drone swarm.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

КМР – кваліфікаційна магістерська робота
АПЗ – апаратно-програмне забезпечення
ПЗ – програмне забезпечення
ОС – операційна система
ШІ – штучний інтелект

LoRa – Long Range
LoRaWAN – Long Range Wide Area Network
CNN – Convolutional Neural Network
FPGA – Field-Programmable Gate Array
CUDA – Compute Unified Device Architecture
P2P – Peer-to-Peer
YOLO – You Look Only Once
ISM – Industrial, Scientific and Medical

ВСТУП

Актуальність теми кваліфікаційної роботи. Останні декілька років характеризуються все зростаючим інтересом до безпілотних апаратів (дронів). Їх використання вже революціонізувало сферу національної безпеки. Незабаром постане питання їх активного цивільного використання, що робить актуальним попереднє узагальнення зроблених напрацювань та подальше удосконалення технологій та способів використання дронів.

Іншим технологічним напрямком, що зараз стрімко розвивається, є штучний інтелект. Переваги «розумних» технологій, здатних навчатися за допомогою алгоритмів машинного навчання, сприяють їх широкій інтеграції у все нові сфери людського життя. Поєднання штучного інтелекту та безпілотних апаратів є важливим та перспективним напрямом у технологічній сфері.

Об’єктом дослідження є технології та методи розробки розподілених систем обробки даних.

Предметом дослідження є розподілена система детектування рухомих об'єктів та обміну інформацією на базі LoRaWAN.

Метою даної кваліфікаційної роботи є створення розподіленого апаратно-програмного комплексу, що використовує штучний інтелект (у вигляді нейронної мережі) та протокол LoRaWAN.

Відповідно до поставленої мети визначено такі завдання:

– аналіз предметної області кваліфікаційної роботи;

– огляд існуючої наукової та технічної літератури на тему;

– аналіз сучасних аналогічних систем та рішень;

– розробка концептуального рішення розподіленої системи, виходячи з визначених вимог;

– вибір апаратних та програмних засобів розробки розподіленої системи;

– розробка апаратно-програмного забезпечення розподіленої системи;

– тестування працездатності розробленого апаратно-програмного забезпечення.

Дослідження і розробка розподіленої системи детектування рухомих об'єктів та обміну інформацією на базі технології LoRaWAN має велике *практичне значення*. Запропонована розподілена система поєднує потужні можливості штучного інтелекту з такими сильними сторонами технології LoRaWAN, як велика дальність сигналу, висока завадостійкість та низьке енергоспоживання. Таке поєднання зробить можливим полегшення та збільшення ефективності виконання задач у сферах, де потрібен автоматичний візуальний аналіз на великих площах та відстанях, а саме: у аграрній сфері, сфері охорони природи, під час пошуково-рятувальних операціях, наукових дослідженнях та інших видах моніторингу.

Дана кваліфікаційна робота була апробована на XXI Міжнародній науковій конференції «Ольвійський форум – 2024: стратегії країн Причорноморського регіону в геополітичному просторі». Матеріали доповіді були опубліковані у збірнику конференції [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, НЕОБХІДНОЇ ДЛЯ РОЗРОБКИ РОЗПОДІЛЕНОЇ СИСТЕМИ

Запропонована розподілена система призначена для виконання двох головних функцій, а саме: *детектування рухомих об'єктів* з відеопотоку, отриманого з камери носія, та *передачі даних* про детектований об'єкт від одиниць системи до кінцевого користувача.

1.1 Особливості задачі детектування рухомих об’єктів

Першочерговою функцією запропонованої розподіленої системи є детектування рухомих об’єктів (англ. moving object detection, MOD). Ця задача розглядається в рамках окремої галузі комп'ютерних наук – теорії та технологій *комп’ютерного зору* (англ. computer vision), та відноситься до ширшого кола задач детектування об’єктів (англ. object detection).

Комп'ютерним зором називають напрям досліджень і технологій у сфері комп'ютерних наук, що мають на меті надання комп’ютерним системам високорівневого «розуміння» візуальних даних (зображень та відео). Ця галузь зосереджена на розробці алгоритмів, моделей і систем, що дозволяють програмам аналізувати, інтерпретувати й отримувати значущу інформацію з візуальних даних, подібно до того, як люди візуально сприймають та розуміють світ. Основними завданнями комп’ютерного зору є: класифікація зображень, детектування об'єктів, оцінка руху, відстеження об'єктів, сегментація зображень, реконструкція сцени, оцінка положення об'єкта, відновлення зображень та ін.

Детектування об'єктів передбачає ідентифікацію та локалізацію об'єктів, що представляють інтерес, на зображенні або відео. Воно є класичним і фундаментальним завданням комп’ютерного зору, та є основою для таких інших завдань, як створення підписів до зображень, відстеження та сегментація об’єктів. Технології детектування об'єктів тепер широко застосовуються в багатьох прикладних сферах, серед яких: автономне керування автомобілем, відеоспостереження, робототехніка тощо.

Детектування рухомих об’єктів – це різновид детектування об’єктів, чия мета полягає у отриманні відеопослідовності з нерухомої чи рухомої камери та виведення бінарних масок, які представляють рухомі об'єкти, для кожного кадру послідовності [2]. Суміжними з детектуванням рухомих об’єктів задачами є також сегментація об’єкту на відео (англ. video object segmentation, VOS) та сегментація руху (англ. motion segmentation) [3].

Головними методами, що використовуються для детектування рухомих об'єктів є: різниця послідовних кадрів (англ. consecutive frame difference), віднімання фону (англ. background subtraction) та оптичний потік (англ. optical flow). Методи різниці послідовних кадрів є простими в реалізації, але погано справляються зі складними випадками. Методи оптичного потоку є більш надійними, але є досі трудомісткими, щоб відповідати вимогам реального часу. Віднімання фону, який є найпопулярнішим методом виявлення рухомих об'єктів, пропонує найкращий компроміс між надійністю та вимогами реального часу [4].

Як правило, для вирішення задачі детектування об’єктів використовують або опорно-векторні машини (англ. support vector machine, SVM), або, що частіше – *штучні нейронні мережі*.

1.1.1 Штучні нейронні мережі

Штучна нейронна мережа – це обчислювальна система, яка складається з поєднаних між собою обчислювальних одиниць – нейронів, та яка здатна ітеративно навчатися, покращуючи свій результат.

Нейрони в штучних нейронних мережах організовані у декілька пов’язаних між собою шарів: шару входу, прихованих шарів та шару виходу (рис. 1.1). Нейрони мають числові параметри – *вагу*, яка вказує на силу зв’язку нейрона одного шару з нейроном наступного шару, та *зміщення* (англ. bias). Всі нейрони, крім нейронів шару входу, мають *функцію активації* нейрона, яка здійснює нелінійне перетворення над входами нейрона та його зміщенням, і формує кінцевий вихід нейрона.

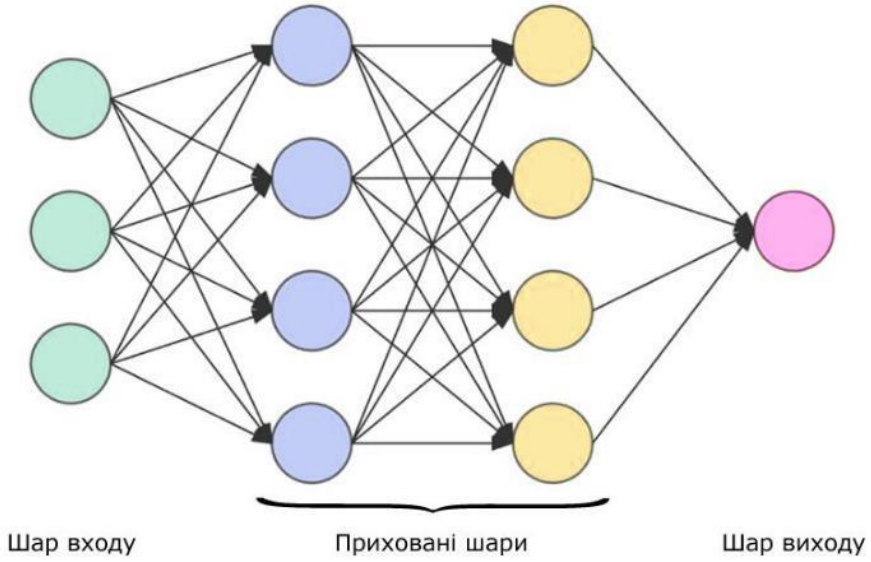


Рисунок 1.1 – Загальний приклад нейронної мережі

Нейронні мережі навчаються, ітеративно змінюючи свої параметри (ваги та зміщення) на основі вхідних даних та бажаного результату. Цей процес називається *навчанням*, або *тренуванням* нейронної мережі.

Зазвичай, спочатку випадковим чином ініціалізуються ваги та зміщення нейронної мережі. Далі виконується крок *прямого поширення*, під час якого вхідні дані проходять через всі шари мережі. Після цього отримані вихідні дані порівнюються з цільовими значеннями за допомогою функції витрат (англ. cost function), яка кількісно визначає різницю між прогнозованим і отриманим результатом. Мета навчання нейронної мережі полягає в тому, щоб мінімізувати значення функції витрат.

Наступним кроком застосовується *зворотне поширення*, під час якого параметри нейронної мережі корегуються відповідно до значення *градієнта* функції витрат. Розмір кроку корегування визначається обраним *темпом навчання*.

Далі, вище наведені кроки повторюються для визначеної підмножини навчальних даних (англ. batch). Цей процес називається *епохою*. Нейронна мережа навчається, проходячи визначену кількість епох, або до тих пір, доки функція витрат не буде достатньо мінімізована.

1.1.2 Архітектури нейронних мереж

На даний момент існує багато різних архітектур нейронних мереж: рекурентна нейронна мережа (англ. recurrent neural networks, RNN), згорткова нейронна мережа (англ. convolutional neural network, CNN), залишкова нейронна мережа (англ. residual neural network, ResNet), мережа довгої короткочасної пам'яті (англ. long short-term memory, LSTM), трансформер та ін.

Для задач комп’ютерного зору загалом, та детектування об’єктів зокрема найбільш часто застосовуються згорткові нейронні мережі [5].

Згорткова нейронна мережа – це архітектура нейронної мережі, що спеціалізована для обробки та аналізу структурованих сіткоподібних даних, таких як зображення та відео. Ключовою особливістю CNN-мереж є використання згорткових шарів, що застосовують фільтри або ядра (англ. kernel). Фільтри обчислюють скалярний добуток між вагами та вхідними значеннями, навчаючись таким чином автоматично виокремлювати просторові ознаки.

Згорткові нейронні мережі складаються зі згорткових шарів, функцій активації (наприклад, ReLU) та агрегувальних (англ. pooling) шарів, які зменшують просторові розміри та виокремлюють ключові ознаки. Наступними йдуть згладжувальний (англ. flatten) шар, який згортає просторові розмірності даних у розмірність каналу, та повністю з'єднані шари, що виконують фінальну класифікацію або регресію.

Перспективною архітектурою нейронних мереж є трансформер. Ця архітектура була винайдена для вирішення таких задач сфери обробки природної мови, як будування великих мовних моделей (англ. large language model, LLM), та невдовзі революціонізувала сферу штучного інтелекту (ШІ). Після успіху таких моделей нейронних мереж на основі трансформера, як GPT, почалася розробка моделей і у сфері комп’ютерного зору. Такі трансформери називаються зоровими (англ. Vision Transformer), та наразі існують наступні моделі: DETR, SMCA, Swin тощо [5; 6].

Зорові трансформери мають наступний принцип роботи. Спочатку зображення ділиться на менші фрагменти (англ. patch), які перетворюються на вектори. Далі разом із своєю позиційною інформацією фрагменти подаються до кодувальника (англ. encoder), який складається з шарів самоуваги (англ. self-attention) та шарів прямого поширення. За допомогою механізму самоуваги обчислюється як та наскільки сильно один фрагмент пов’язаний з іншими. На виході мережі генерується класифікаційний токен (англ. classification token, CLS), який агрегує дані з усіх фрагментів і робить висновок про ціле зображення.

1.2 Дрон як носій нейронної мережі

Нейронні мережі, які колись вважалися занадто вимогливими до обчислень для багатьох комп'ютерів, окрім найпотужніших, тепер можуть працювати на найрізноманітніших пристроях завдяки прогресу в апаратному забезпеченні та методах оптимізації. Носіями нейронних мереж можуть бути будь-які пристрої, що мають достатньо потужні або/та оптимізовані апаратні ресурси: відеокарти, процесори, спеціальні чипи для машинного навчання, тензорні процесори, програмовані вентильні матриці (англ. Field-Programmable Gate Array, FPGA). Таким чином, носіями нейронної мережі можуть бути суперкомп’ютери, хмарні високопродуктивні сервера, користувацькі персональні комп’ютери та ноутбуки, мобільні пристрої, одноплатні комп’ютери та вбудовані системи.

До останніх належать *безпілотні апарати*, також відомі як *дрони*. Дронами називають рухомі апарати, які працює без наявності людини (водія або пілота) на борту. Дрони можуть мати дистанційне керування або бути автономними апаратами.

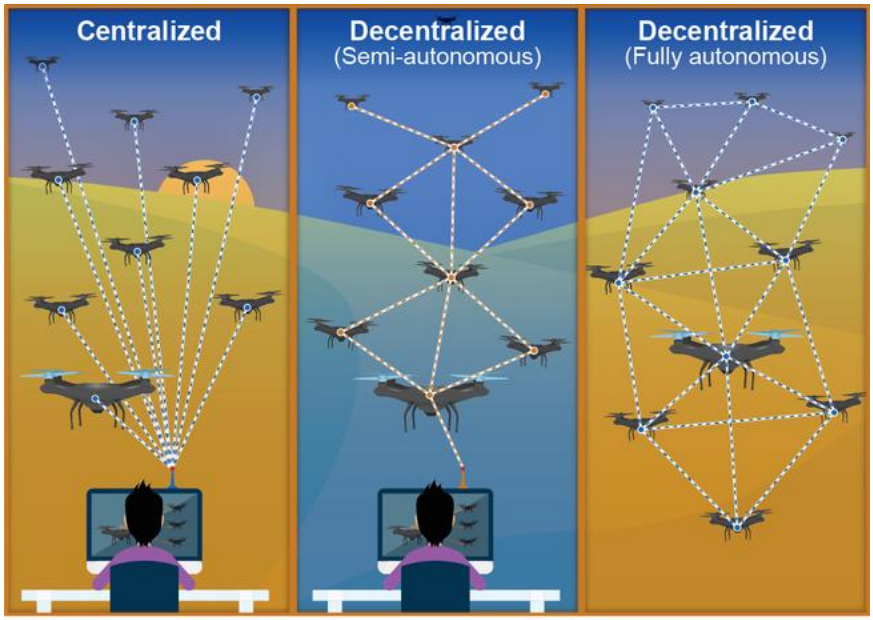
Використання ШІ у вигляді нейронних мереж набуло великого поширення у галузі розробки безпілотних апаратів. Найчастіше штучний інтелект використовується для вирішення наступних задач: детектування та відслідковування об’єктів, автономна навігація, планування шляху, координація дронів у рою, аналіз відео та зображень, кібербезпека [7].

Нейронні мережі можуть бути розташовані як на апаратному забезпеченні самого дрона, так і на віддаленому пристрої – сервері, з яким дрон має з’єднання. Останній варіант є більш застосованим через те, що таким чином відкидається необхідність забезпечення безпілотного апарата достатньою кількістю обчислювальних ресурсів.

Проте використання для роботи нейронної мережі апаратних ресурсів самого дрона є цілком можливим та має низку переваг, насамперед більшу швидкодію та автономність роботи. В якості апаратного забезпечення такі дрони використовують або одноплатні комп’ютери, такі як NVIDIA Jetson Nano [8, 9], або FPGA-матриці для більшої швидкодії [10].

1.3 Децентралізований рій дронів як розподілена система

Одним зі способів покращення ефективності безпілотних апаратів є їх об’єднання у так звані «рої». *Рій безпілотних апаратів* являє собою систему обміну інформацією між всіма дронами, яка дозволяє координувати їх роботу без прямого втручання людини. Рої дронів поділяються на *централізовані* та *децентралізовані* (рис. 1.2).



Source: GAO analysis (data). Sonar512/topvectors/stock.adobe.com (images). | GAO-23-106930

Рисунок 1.2 – Централізована та децентралізована організація керування дронами [11]

Централізовані рої дронів використовують таку систему керування, де один дрон чи зовнішній комп’ютер відповідальний за координацію та планування дій. У випадку децентралізованого рою дронів кожна одиниця зв’язується з іншою та приймає рішення самостійно [12].

Повністю централізовані підходи до управління дронами часто є високопродуктивними та ефективними, але є вразливими до єдиних точок відмови та мають погану масштабованість через комунікаційні обмеження. Повністю децентралізовані підходи, навпроти, не мають цих недоліків завдяки таким функціям, як дублювання і розпаралелювання, але можуть мати нижчу швидкість і ефективність [13].

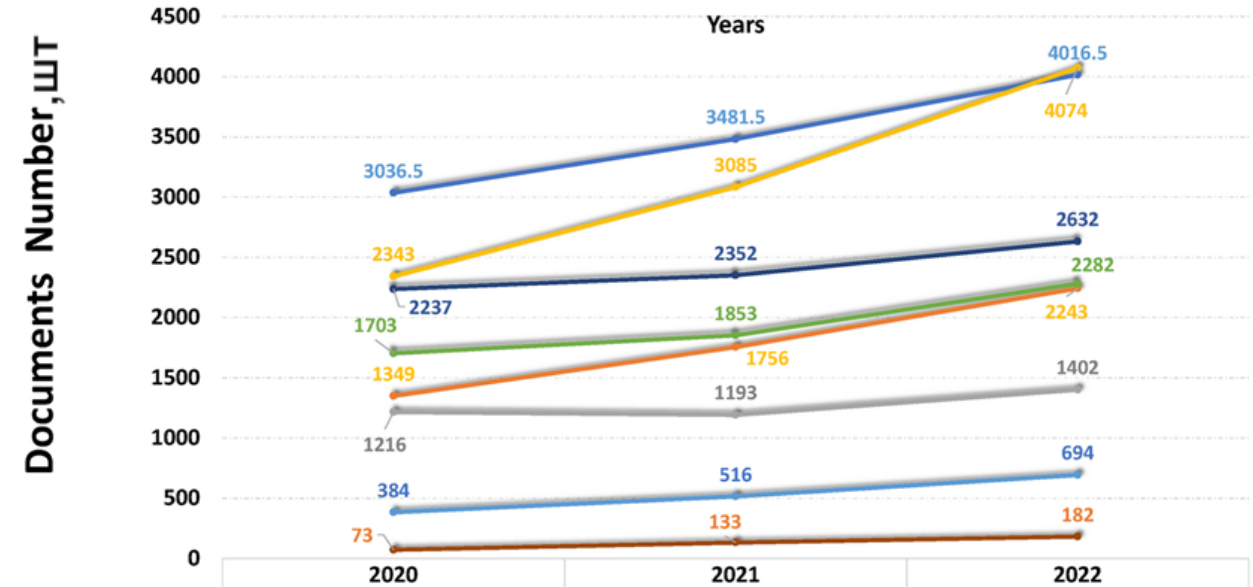
Децентралізований рій дронів може розглядатися як різновид **розподіленої системи**. У децентралізованому рої немає єдиного контролюючого органу, і кожен дрон працює автономно, приймаючи рішення на основі локальної інформації та безпосередньо спілкуючись з іншими дронами в рої. Така структура відображає ключові принципи розподілених систем, де кілька незалежних агентів працюють разом для досягнення спільної мети, не покладаючись на центральне керування.

Децентралізований рій безпілотних апаратів задовольняє наступні характеристики розподілених систем: автономність вузлів (дронів), відсутність єдиного централізованого контролера, взаємна координація вузлів, легка масштабованість, відмовостійкість та незалежність системи загалом від відмови одного вузла (дрона).

1.3.1 Використання штучного інтелекту у роях дронів

Організація дронів у рої та використання штучного інтелекту є двома перспективними напрямками досліджень у сфері безпілотних апаратів. І хоча досліджень щодо застосування машинного навчання для дронів у десятки раз більше за дослідження ройової поведінки (рис. 1.3), існують гібридні дослідження із використанням цих обох технологій, і навіть окремі комерційні

рішення [14].



	2020	2021	2022
Antenna x 0.5	3036.5	3481.5	4016.5
Aircraft Detection	1349	1756	2243
Remote Sensing	1216	1193	1402
AI	2343	3085	4074
IoT	384	516	694
Energy-Efficient	1703	1853	2282
Control	2237	2352	2632
Swarm Behavior	73	133	182

Рисунок 1.3 – Кількість досліджень у різних напрямках досліджень у сфері безпілотних апаратів на 2023 рік [7]

Машинне навчання у застосуванні до роїв дронів використовується для вирішення наступних задач: планування та дотримування оптимальної траєкторії руху дронів, зміна траєкторії через надзвичайні ситуації, автономне відслідковування цілей, розподіл комунікаційних ресурсів, керування живленням, класифікація зображень, моніторинг та прогнозування якості повітря, використання рою дронів як сенсорів для збирання даних тощо [15].

1.4 Застосування LoRaWAN для передачі даних

Використання децентралізованого рою дронів призводить до зменшення навантаження на комунікаційну складову через делегування певних операцій автономним дронам. Це дозволяє використовувати дрони для задач на більших відстанях. Але такі існуючі технології бездротового зв’язку, як Wi-Fi, Bluetooth

та стільниковий зв’язок, хоча і мають більшу пропускну здатність, але не можуть забезпечити роботу на дуже великих відстанях (10–15 км). Технологія, що здатна забезпечити таку велику відстань, на яку може бути передана інформація, називається LoRaWAN (рис. 1.4).

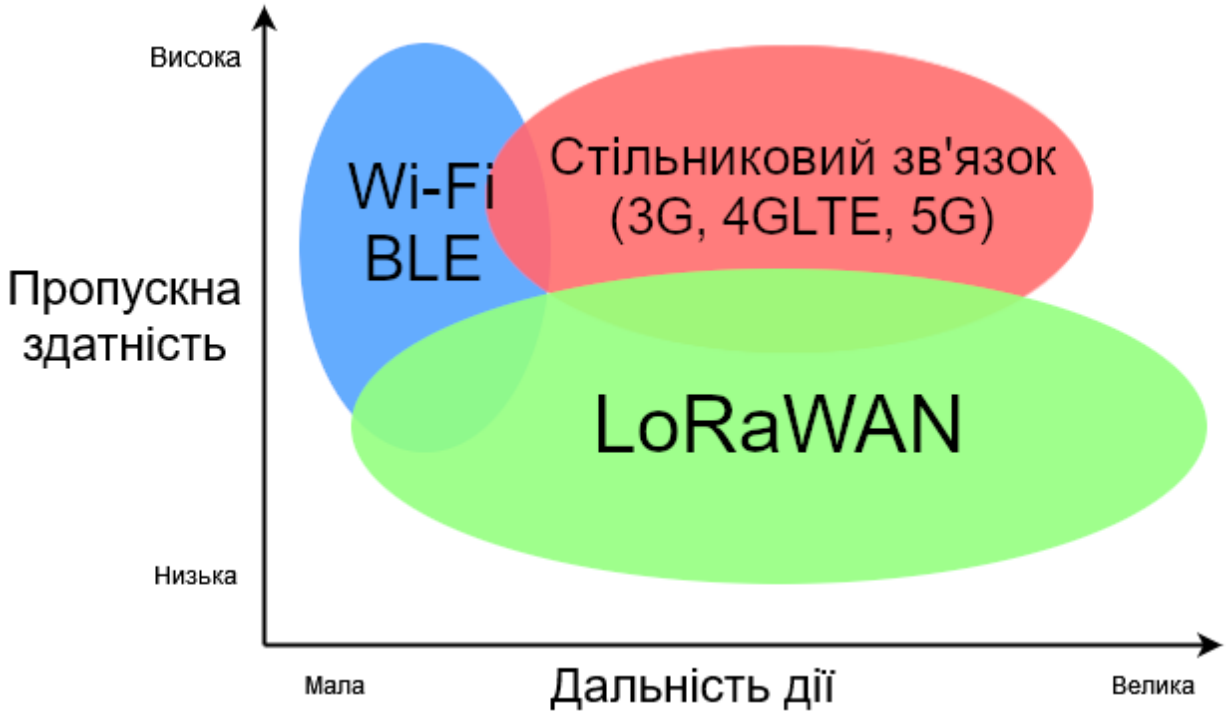


Рисунок 1.4 – Порівняння LoRaWAN з іншими технологіями бездротового зв’язку

LoRaWAN представляє собою технологію бездротового зв’язку для малопотужних (англ. low-power) глобальних мереж (LPWAN), яка була розроблена компанією Semtech. LoRaWAN здатна підтримувати зв'язок на великих відстанях: до 5 кілометрів у містах і до 15 кілометрів або більше в сільській місцевості (за умов прямої видимості) [16].

Власне, LoRaWAN – це відкритий та стандартизований комунікаційний протокол, який працює поверх сигналу, модульованого за допомогою методу LoRa. Таким чином, розглянутий за моделлю OSI, стек LoRaWAN має два рівні: фізичний та MAC-рівень. Фізичному рівню відповідає модуляція за методом LoRa, а MAC-рівню – протокол LoRaWAN (рис 1.5).

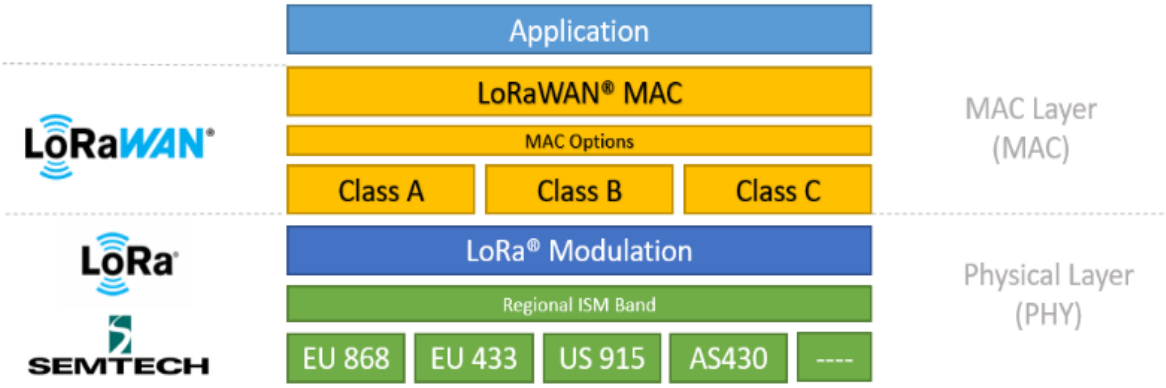


Рисунок 1.5 – Стек LoRaWAN у моделі OSI [16]

LoRa – це запатентований метод модуляції з розширеним спектром, який базується на модуляції Chirp Spread Spectrum (CSS). Chirp Spread Spectrum – це метод розширення спектру, який використовує широкосмугові лінійні частотно-модульовані chirp-імпульси для кодування інформації. Chirp-імпульсом називається розгортка частоти у відповідній смузі частот (125 кГц, 250 кГц і т.д.). Вигляд сигналу, модульованого за допомогою chirp-імпульсів, показаний на рис. 1.6.

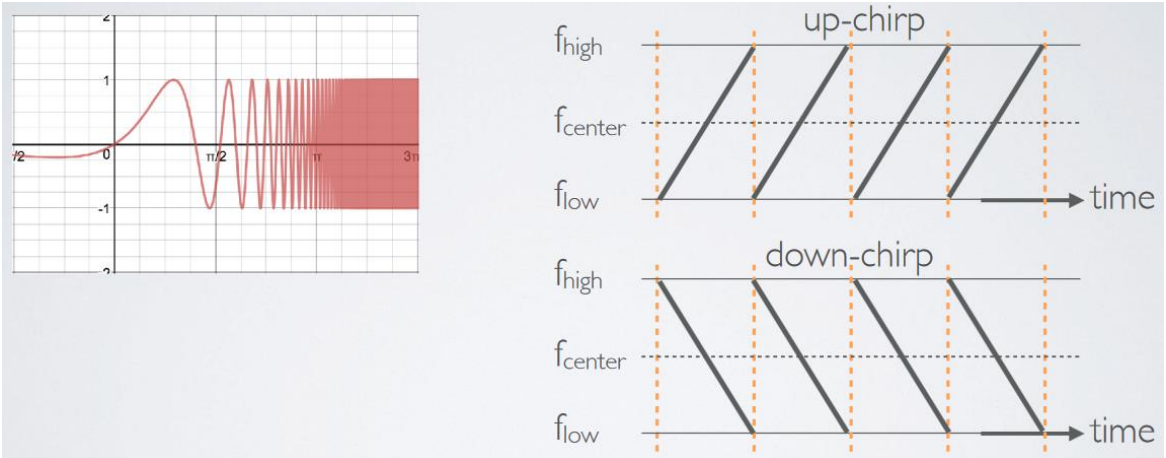


Рисунок 1.6 – Модуляція LoRa [17]

Для мереж LoRaWAN використовується неліцензовані радіочастотні діапазони менше 1 ГГц, такі як: 433 МГц, 868 МГц та 915 МГц. Саме поєднання низької частоти з модуляцією LoRa забезпечує ту велику відстань, на яку може бути передана інформація (5–15 км). При цьому особливість сигналу, модульованого таким чином, полягає у тому, що він має великий захист від завад.

Зворотною стороною є відносно низька швидкість передачі даних від 0,018 до 37,5 кбіт/с [18].

1.4.1 Архітектура LoRaWAN

Мережа LoRaWAN складається з наступних елементів: кінцевих пристроїв (вузлів), шлюзів, мережевого серверу, серверу приєднання та серверу застосунків (рис. 1.7).

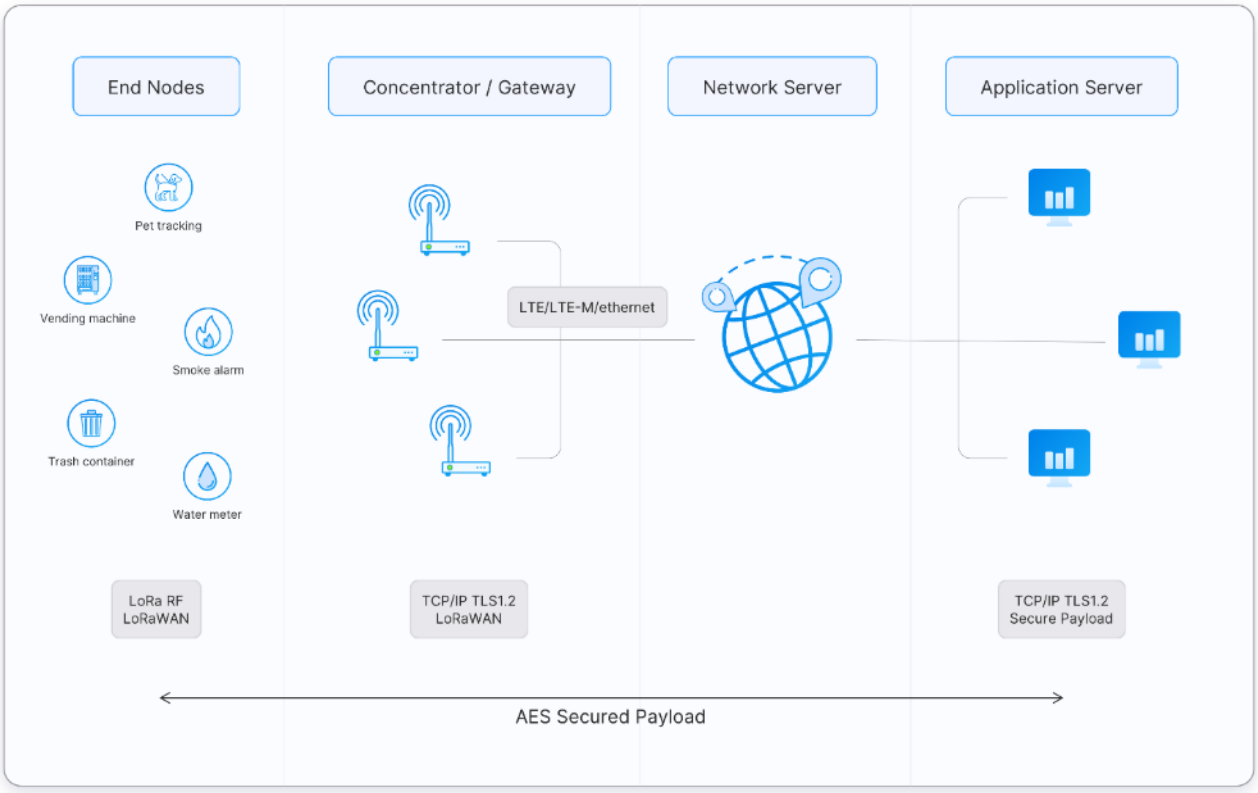


Рисунок 1.7 – Мережева архітектура LoRaWAN [19]

Кінцевим пристроєм, або вузлом (англ. node) називається сенсор чи актуатор (привід), який здійснює обмін модульованими за допомогою LoRa повідомленнями зі шлюзом.

Шлюз (англ. gateway) – це пристрій, який приймає LoRa-повідомлення від вузлів та передає їх до мережевого сервера через транзитну мережу, яка в свою чергу може використовувати будь-яку технологію бездротового зв’язку (Wi-Fi, Ethernet, стільникові мережі тощо). Шлюз та з’єднані з ним вузли утворюють мережеву топологію типу «зірка». При цьому фіксованого зв'язку вузлом та

конкретним шлюзом не існує: один і той самий датчик може обслуговуватися кількома шлюзами.

Мережевий сервер (англ. network server) призначений для керування всією мережею, динамічної зміни параметрів мережі для адаптації системи та встановлення безпечного 128-бітного з'єднання AES.

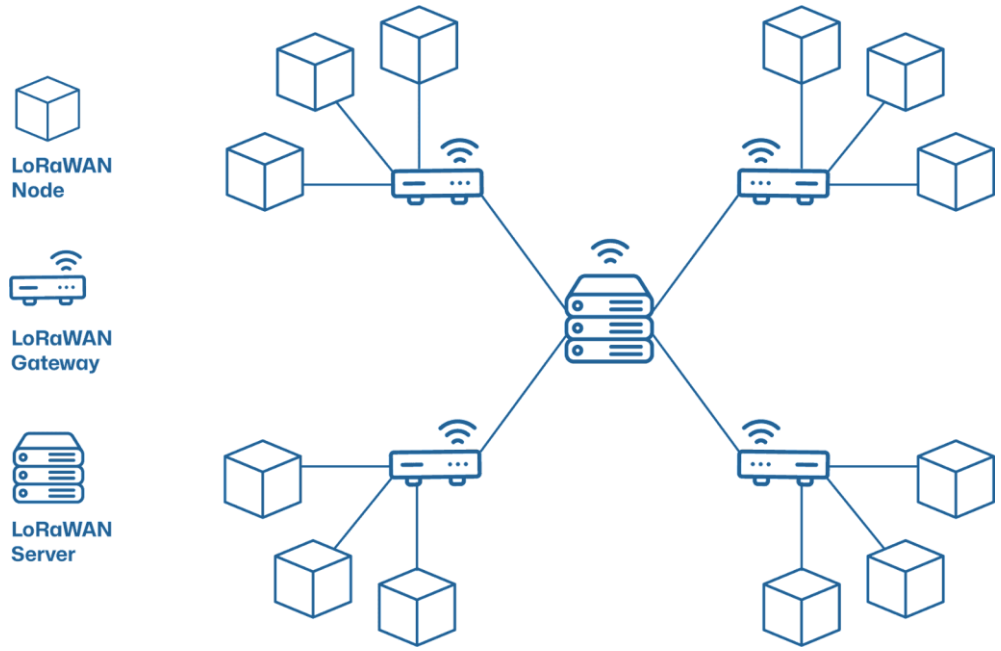


Рисунок 1.8 – Топологія LoRaWAN [20]

Мережі LoRaWAN мають топологію «зірка із зірок» (англ. stars-of-stars), яка зображена на рис. 1.8.

1.4.2 Класи пристроїв у LoRaWAN

Специфікація LoRaWAN визначає три типи пристроїв: «Клас А», «Клас В» і «Клас С». Класи пристроїв LoRaWAN відрізняються насамперед тим, як здійснюється двонаправлений зв'язок з сервером (рис 1.9).

Усі вузли LoRaWAN повинні підтримувати реалізацію «класу А». Пристрій «класу А» може передавати повідомлення в будь-який час. Після завершення передачі пристрій відкриває два коротких вікна для прийому повідомлень з сервера. Якщо за ці два проміжки часу нічого не було отримано, нова можливість прийняти повідомлення з’явиться лише після наступної відправки даних з вузла до сервера.

Пристрої «класу B» отримують від мережі спеціальний синхронізований за часом сигнал-маяк (англ. beacon), який надає пристрою часовий орієнтир. Відносно цього часового орієнтира пристрій налаштовує внутрішній годинник, і це дає змогу пристрою відкривати вікно прийому повідомлень у певний час, згідно за певним розкладом.

У пристроях «класу C» вікно прийому даних є постійно відкритим, окрім того часу, коли пристрій передає дані на сервер. Це дозволяє їх застосовувати для вирішення завдань, що вимагають одержання великого обсягу даних.

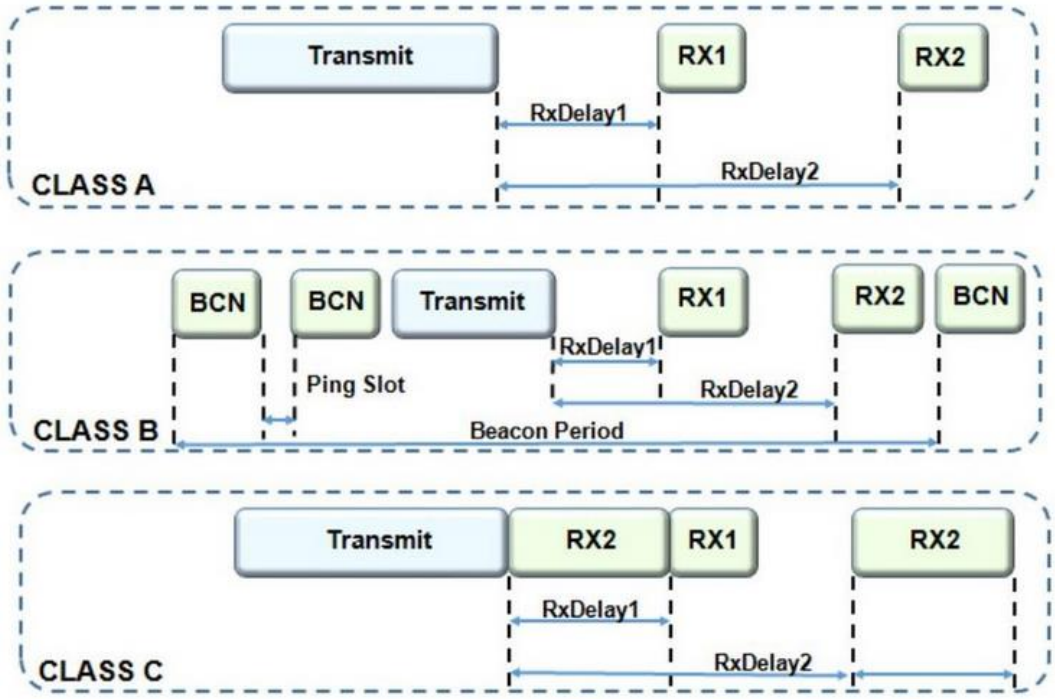


Рисунок 1.9 – Двонаправлений обмін даними класів пристроїв у LoRaWAN [21]

Через різницю у тому, як здійснюється обмін даними, класи пристроїв LoRaWAN мають різне енергоспоживання та затримку. «Клас A» є найбільш енергоефективним, але при цьому має найбільшу затримку. Пристроями такого класу є пасивні сенсори. «Клас C» майже не має затримки під час передачі даних, але. До «Класу C» відносяться пристрої, що потребують швидкодії та постійного підключення, такі як: сигналізації, ліхтарі тощо. Пристрої «Класу B» є достатньо енергоефективними для використання живлення від акумулятора, при цьому

маючи невелику затримку. Пристроями такого класу є, наприклад, комунальні лічильники.

1.5 Специфікація вимог до АПЗ

1 ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

1.1 *Призначення системи, для якої розробляється апаратно-програмне забезпечення*: Запропонована система призначена для детектування рухомих об'єктів з відеопотоку носіїв системи та передачі даних про детектований об'єкт кінцевому користувачеві.

2 ЗАГАЛЬНИЙ ОПИС

2.1 Сфера застосування. Сфери, де потрібен автоматизований аналіз візуальний даних на великих площах та відстанях: аграрна сфера, сфера охорони природи, пошуково-рятувальні операції, наукові дослідження та інші види моніторингу.

2.2 Характеристики користувачів. В проєкті передбачена всього одна роль користувачів. Розмежування прав доступу до різних функцій не передбачено. Очікується, що користувачі, що працюватимуть з зазначеним комплексом, мають базові знання в галузі інформаційних технологій.

2.3 Загальна структура і склад апаратно-програмного забезпечення (АПЗ). Системні вимоги. Апаратне забезпечення пристроїв розподіленої системи має включати наступні компоненти: пристрій керування – одноплатний комп’ютер, сумісну з ним камеру, модулі LoRa (трансивер або концентратор) для комунікації, а також GPS-модулі для отримання даних про місцеположення.

3 ВИМОГИ ДО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ. Для забезпечення роботи ПЗ з нейронною мережею необхідна наявність достатньо потужного одноплатного комп’ютера, який обладнаний графічним процесором з підтримкою ядер CUDA, процесором з тактовою частотою не менше 1 ГГц, обсягом оперативної пам’яті не менше 4 Гбайт та обсягом сховища не менше 64 Гбайт.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Архітектура програмної системи. Програмна складова має бути реалізована на основі об’єктно-орієнтованого підходу. Нейронна мережа має бути реалізована з використанням спеціалізованих фреймворків та бібліотек для машинного навчання.

4.2 Системне програмне забезпечення. Розроблений комплекс повинен коректно працювати на операційних системах (ОС) на базі ядра Linux.

4.3 Мова і технологія розробки ПЗ. У якості мови програмування буде застосуватися мова Python для реалізації нейронної мережі та мова C/C++ для реалізації підсистеми комунікації.

5 ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

5.1 Апаратний інтерфейс Для реалізації мережевого з’єднання апаратний інтерфейс має підтримувати стандарт LoRa. Для підключення модулів до одноплатного комп’ютера, він повинен мати інтерфейси USB, UART, SPI та CSI.

5.2 Комунікаційний протокол В якості основного комунікаційного протоколу для розподіленої системи використовується протокол LoRaWAN.

Висновки до розділу 1

У даному розділі був виконаний аналіз предметної області, пов’язаної з двома головними функціями запропонованої розподіленої системи, а саме: детектування рухомих об'єктів з відеопотоку та передачі даних про детектований об'єкт.

Було розглянуто особливості задачі детектування рухомих об’єктів відносно теорії та технологій комп’ютерного зору. Для реалізації цієї задачі було запропоновано застосувати штучну нейронну мережу та алгоритми віднімання фону. В якості носіїв нейронної мережі було вирішено використовувати безпілотні апарати. Розглянувши особливості збільшення ефективності роботи дронів за допомогою їх об’єднання рої, було прийнято рішення використовувати

елементи децентралізованого керування роєм дронів у запропонованій розподіленій системі.

Через велику автономність децентралізованого рою дронів було визначено доцільним використання технології LoRaWAN для передачі даних. LoRaWAN відрізняється від інших технологій бездротового зв’язку великою дальністю дії сигналу (до 15 км), низьким енергоспоживанням та високою стійкістю до завад. Таким чином, запропонована розподілена система буде використовувати для нейронну мережу, що працює на апаратному забезпечені безпілотних апаратів, для детектування рухомих об’єктів та технологію бездротового зв’язку LoRaWAN для передачі даних про детектований об’єкт на великі відстані.

У кінці розділу було наведено детальну специфікацію вимог до запропонованої розподіленої системи детектування рухомих об'єктів та обміну інформацією на базі технології LoRaWAN.

2 ПРОЄКТУВАННЯ РОЗПОДІЛЕНОЇ СИСТЕМИ

2.1 Розробка концептуального рішення розподіленої системи

Носіями розподіленої системи детектування рухомих об'єктів та обміну інформацією на базі LoRaWAN можуть бути мережі як статичних пристроїв (наприклад, мережа камер відеоспостереження у сільській місцевості), так і динамічних (об’єднані у рій безпілотні апарати). У даній кваліфікаційній роботі в якості носія системи розглядається рій дронів з елементами децентралізованого типу керування.

2.1.1 Мережева архітектура розподіленої системи

Сам рій дронів в цілому являє собою однорідну мережу, що має архітектуру P2P (від англ. Peer-to-Peer), що складається з взаємопов’язаних рівноправних вузлів, де в якості таких P2P-вузлів виступають групи дронів. У свою чергу, кожна така група представляє собою приватну мережу на базі протоколу LoRaWAN.

Вибір саме такої архітектури робить систему гнучкою, універсальною та більш захищеною. Так як, по-перше, P2P-мережа верхнього рівня (тобто на рівні всього рою дронів) може бути будь-якою в залежності від поставлених практичних завдань. А по-друге, наявність приватної мережі усередині груп дронів забезпечує їх ізоляцію.

Мережа LoRaWAN повинна складатися з притаманних їй елементів архітектури – вузлів, шлюзів, мережевого сервера та сервера застосунків. У типових мережах LoRaWAN ці елементи зазвичай є окремими пристроями з різними апаратними та програмними потужностями. Тому для реалізації приватної мережі LoRaWAN лише у межах групи дронів потрібно визначити співвідношення між реальними дронами та елементами мережевої архітектури. Від вирішення цієї проблеми також залежить і вибір апаратного забезпечення дронів.

Першим варіантом вирішення є спеціалізація кожного дрона під окремий архітектурний елемент (рис. 2.1а). Проте така надмірна спеціалізація робить систему чутливою до втрат, а також ускладнює виробництво.

Іншою, протилежною опцією є забезпечення кожного дрона необхідними ресурсами для реалізації будь-якого елемента (рис. 2.1б). У цього разі система буде дуже стійка до втрат, адже кожний дрон зможе замінити втрачений. Явним наслідком є значне збільшення коштовності такого рішення.

Запропонованим оптимальним рішенням є поділ дронів на дрони-вузли та керуючі дрони (рис. 2.1в). Перші є типовими вузлами LoRaWAN, задача яких передавати дані, отримувати та виконувати команди. Керуючі дрони повинні реалізовувати шлюз, мережевий сервер та сервер застосунків.

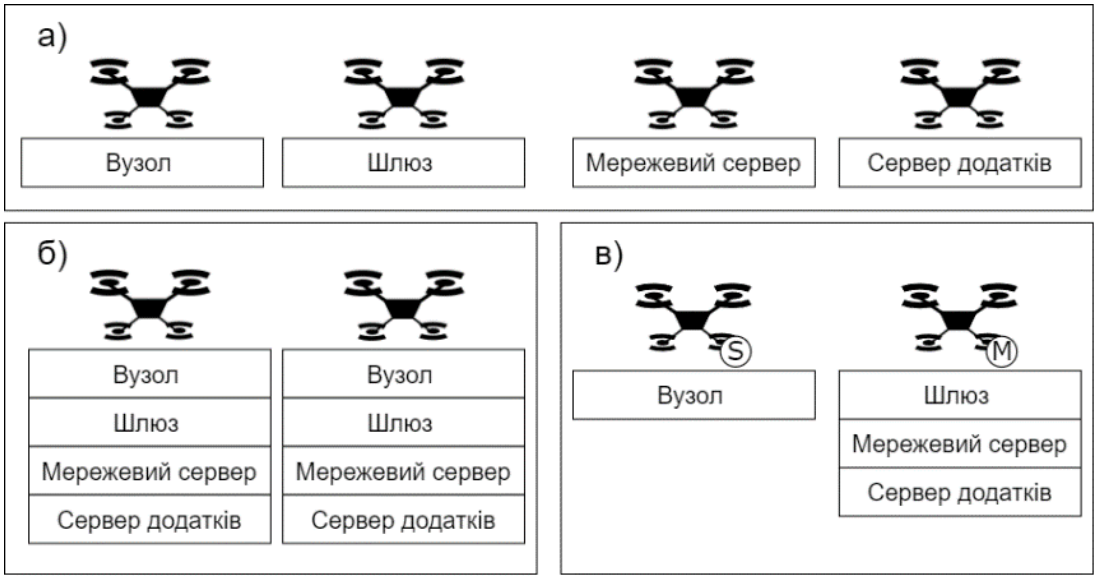


Рисунок 2.1 – Дрони та архітектура LoRaWAN: а – кожен дрон підтримує лише один елемент архітектури; б – кожен дрон підтримує будь-який елемент; в – поділ дрони на керуючі та дрони-вузли

Поєднання шлюзу та мережевого сервера у одному пристрої не є незвичним рішенням. Шлюзи LoRaWAN від таких компаній, як Milesight та Robustel мають вбудований мережевий сервер [22; 23]. Поєднання мережевого сервера та сервера застосунків також є відомою практикою [24].

Таким чином, керуючі дрони є центральним елементом у групах дронів. Вони обслуговують приватну мережу LoRaWAN і саме вони представляють

свою групу у P2P-мережі верхнього рівня. Така система також зберігає стійкість до втрат, адже у випадку втрати чи виходу з ладу керуючого дрона дрони-вузли, що належать до мережі його групи, перейдуть до мережі іншої.

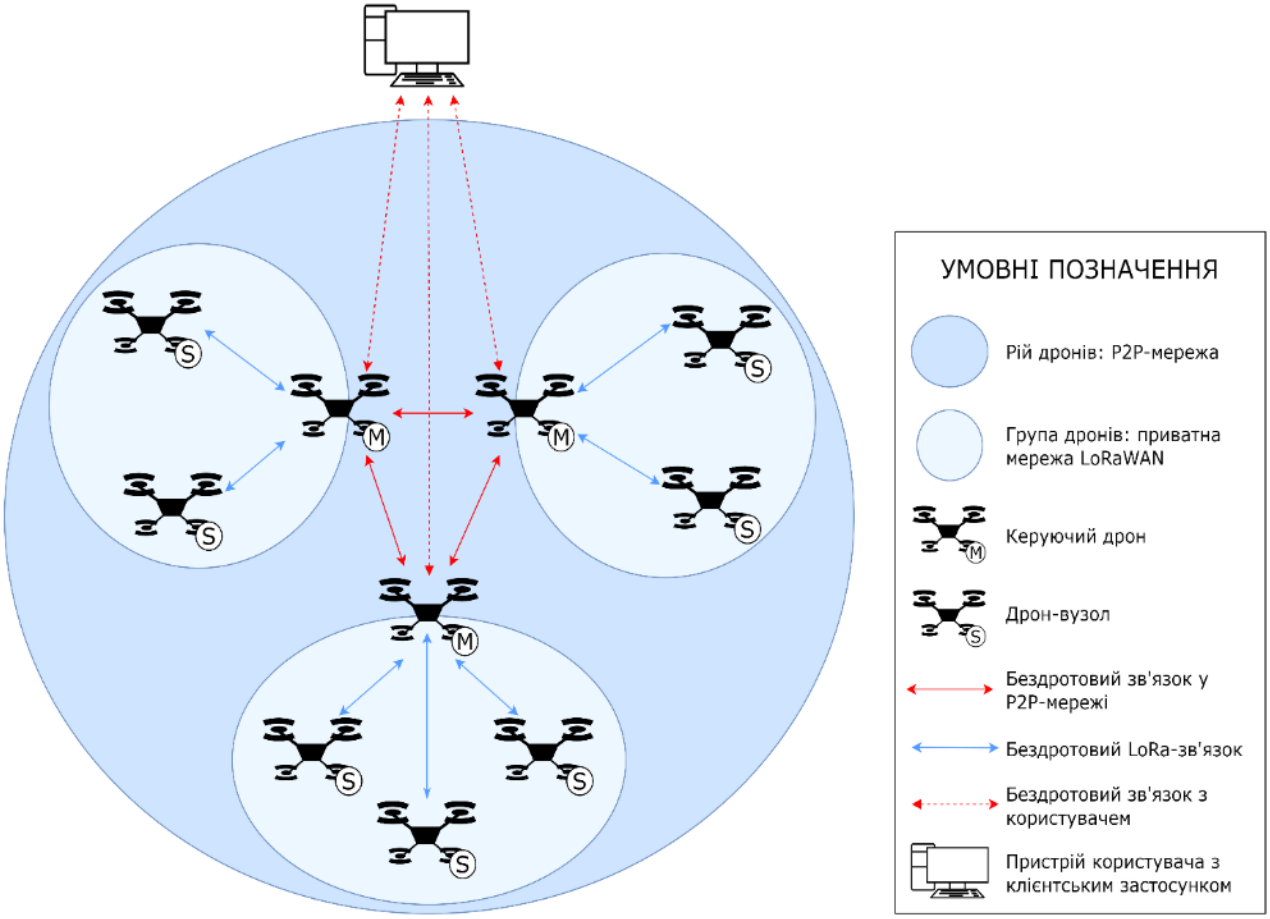


Рисунок 2.2 – Архітектура мережі розподіленої системи

Діаграма остаточно визначеної мережевої архітектури розподіленої мережі показана на рис. 2.2.

2.1.2 Рух даних у розподіленій систем

Кожен дрон у системі має одноплатний комп’ютер, камеру, модулі зв’язку LoRaWAN та GPS-датчик.

Кожен дрон спостерігає за своєю визначеною територію за допомогою камери. Далі відеопотік оброблюється за допомогою нейронної мережі, натренованої для детектування певних типів рухомих об’єктів. Якщо нейронна мережа знайшла об’єкт, дані про об’єкт (тип об’єкта, відсоток схожості)

передаються до основної програми. Програма оброблює дані на зручний для LoRa формат, додає координати дрона, що знайшов об’єкт, та передає їх далі. Якщо це дрон-вузол, то дані передаються до керуючого дрона (який є шлюзом LoRaWAN) за допомогою бездротового зв’язку LoRA. Якщо ж об’єкт знайшов керуючий дрон, то дані передаються в межах його власного пристрою.

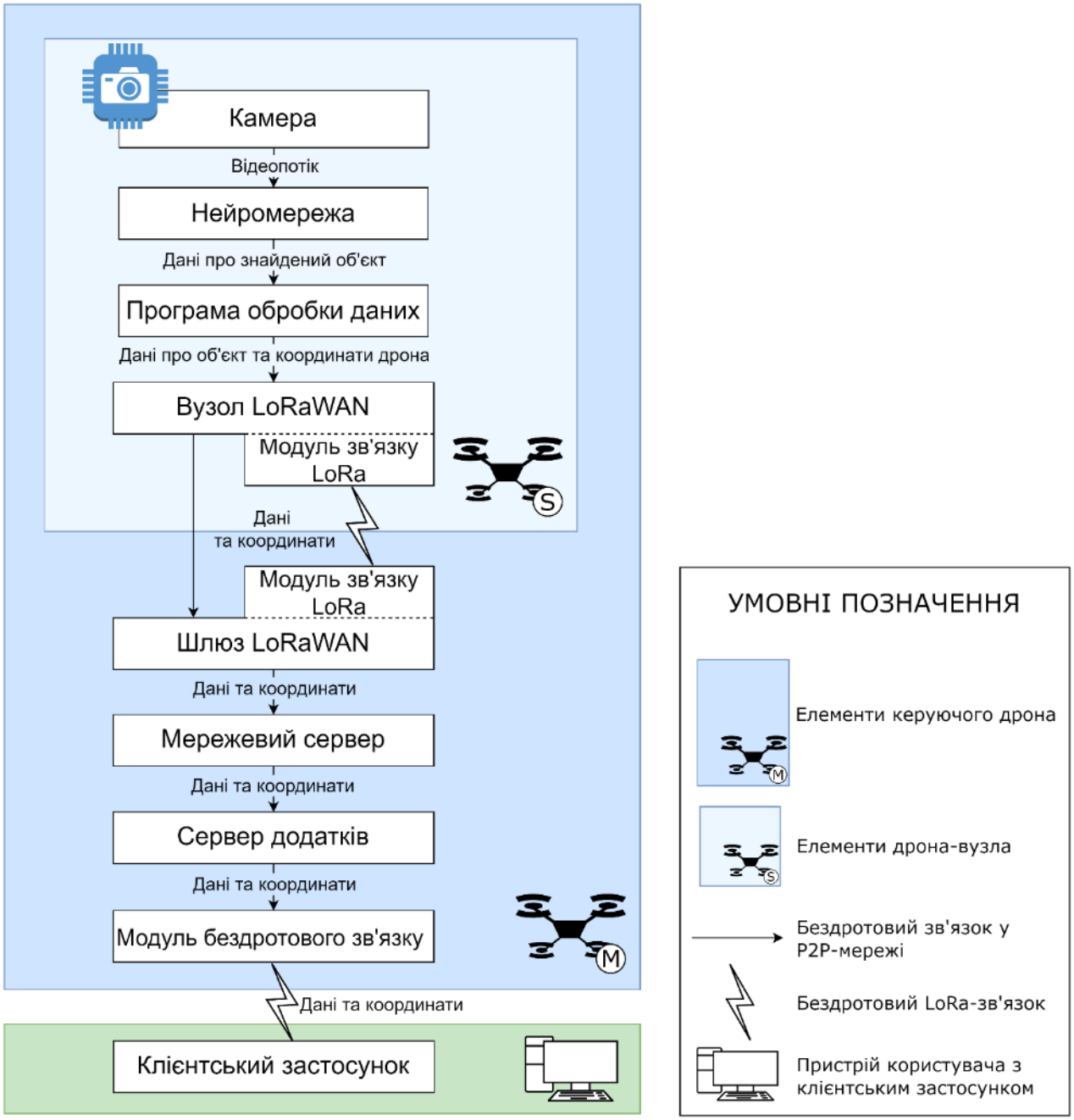


Рисунок 2.3 – Рух даних про детектований об’єкт у розподіленій системі

Далі, дані шлюзу отримує мережевий сервер, який в свою чергу відправляє їх до сервера застосунків. Кінцевий користувач взаємодіє з цими даними за

допомогою клієнтського застосунку, що здійснює підключення до сервера застосунків.

Процес відправки даних про детектований об’єкт зображений на рис. 2.3.

Команди користувача, визначені цілями застосування рою дронів, мають такий же шлях, але у оберненому напрямку.

2.1.3 Інтерфейс клієнтського застосунку

Кінцевий користувач взаємодіє з роєм дронів за допомогою клієнтського застосунку. Даний застосунок підключається до сервера застосунків, який надсилає дані про детектований об’єкт та координати дрона, що помітив об’єкт.

Головне вікно клієнтського застосунку має інтерактивну карту, на яку проєціюються місцеположення дронів відповідно до їх GPS-координат. Кожен дрон зображений за допомогою піктограм, які є різними для дронів-вузлів та керуючих дронів (рис. 2.4).

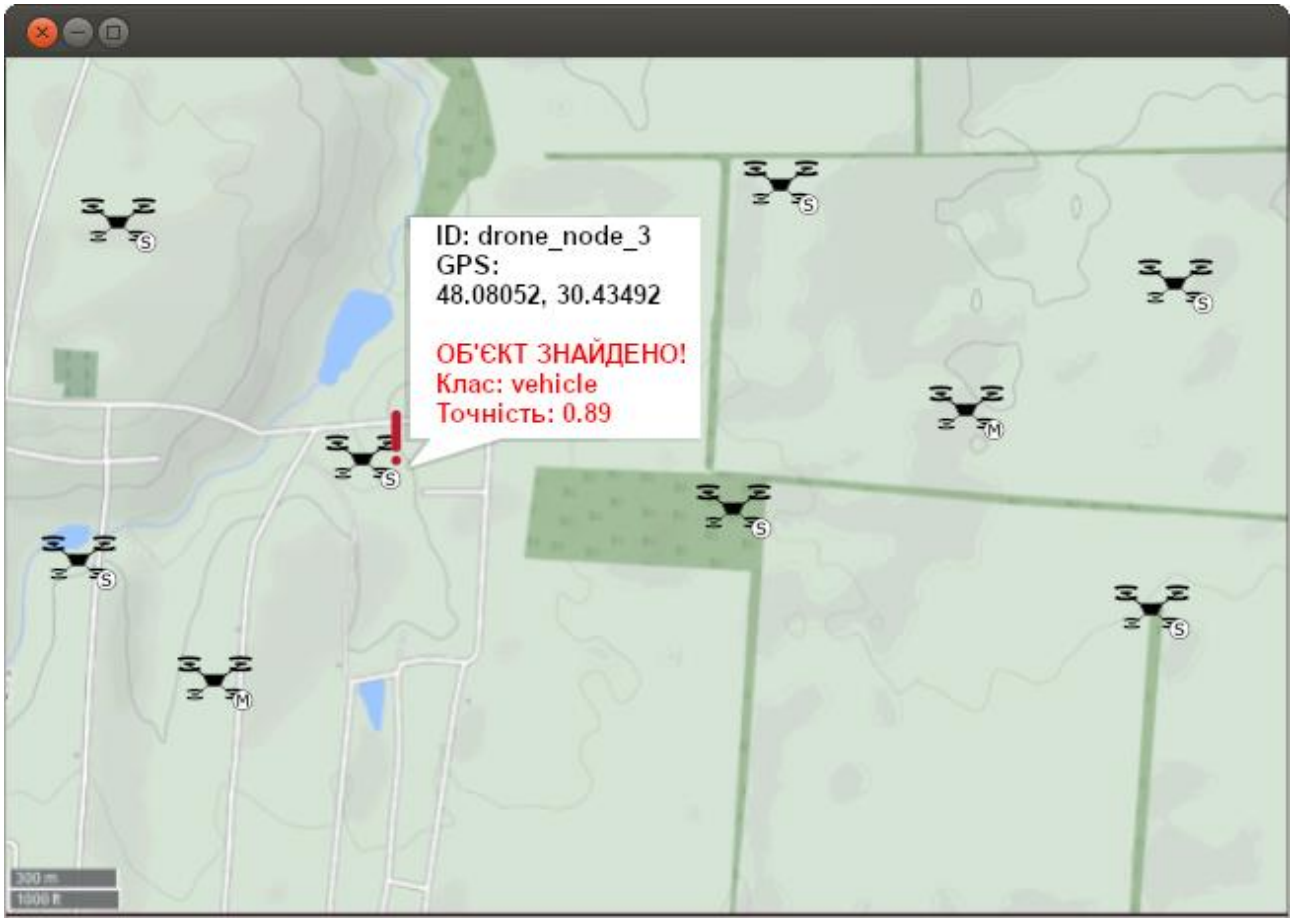


Рисунок 2.4 – Головне вікно клієнтського застосунку

При натисканні на піктограму дрона з’являється невелике вікно з його ID, координатами та приналежності до групи. Якщо дрон детектує об’єкт, то інформація про нього (клас об’єкта та точність детектування) також з’являється у вікні.

2.2 Вибір архітектури та моделі нейронної мережі

Як було зазначено раніше в першому розділі кваліфікаційної магістерської роботи (КМР), для задач детектування об’єктів найбільш часто використовуються такі архітектури, як згортокові нейронні мережі та зорові трансформери. Спираючись на сучасні дослідження ефективності цих архітектур, їх можна схарактеризувати наступним чином. Згорткові мережі є більш швидкими, а також потребують менше ресурсів. Зорові трансформери можуть досягати кращих результатів, але потребують великих датасетів та багато часу на тренування, а також мають меншу швидкодію. На даний момент зорові трансформери все ще не можуть задовольняти вимоги до детектування в режимі реального часу [5].

Зважаючи більшу швидкість згорткових мереж, було вирішено використовувати нейронну мережу саме такої архітектури для задач детектування об’єктів у реальному часі. Серед моделей згорткових мереж: Fast R-CNN, YOLO та ін.

YOLO – модель згорткової нейронної мережі, яка була винайдена у 2015 для задач детектування об’єктів у реальному часі. Назва моделі YOLO походить від назви її принципу «Ти дивишся тільки один раз» (англ. You Look Only Once). Цей принцип полягає у рішенні завдання детектування об’єктів за один прохід по мережі. Попередніх методи детектування об’єктів застосовували або ковзаючі вікна з класифікатором, класифікатор, який потрібно було запускати сотні або тисячі разів для кожного зображення, або двоетапні процеси, що включають пропозиції регіонів і класифікацію.

З 2015 року YOLO зазнала 11 версій, кожна з яких покращувала точність та швидкодію нейронної мережі. На даний момент YOLO залишається одним з найкращих виборів для детектування об’єктів у реальному часі [5]. Саме тому було вирішено використовувати YOLO для розподіленої мережі.

2.3 Вибір апаратної основи розподіленої системи

2.3.1 Вибір одноплатного комп’ютера

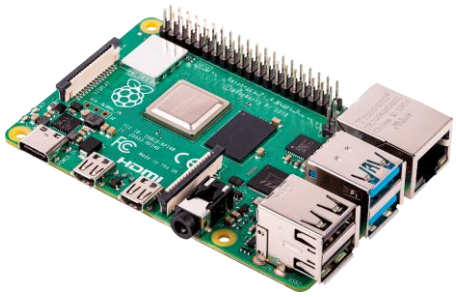
Одиниці запропонованої розподіленої системи повинні використовувати в якості своєї апаратної основи пристрій, достатньо потужний для коректної роботи нейронної мережі, та який має форм-фактор для розміщення на дроні. Таким пристроєм може бути або одноплатний комп’ютер, або FPGA-матриця.

Одноплатний комп'ютер – це повноцінна комп'ютерна система, всі компоненти (процесор, пам'ять, інтерфейси вводу/виводу) якої знаходяться на одній друкованій платі. FPGA-матриця – напівпровідниковий пристрій, що складається з конфігурованих логічних блоків, які дозволяють переналаштувати логіку і функціонал пристрою у будь-який момент.

Не зважаючи на те, що FPGA-матриці можуть бути більш продуктивними для задач штучного інтелекту за одноплатні комп’ютери, останні є більш доступними, простими у користуванні та швидкими у налаштуванні. Тому було вирішено використовувати саме одноплатний комп’ютер в якості основи апаратної частини розподіленої системи детектування рухомих об'єктів та обміну інформацією.

Відомими брендами одноплатних комп’ютерів є: Raspberry Pi, BeagleBone, Asus Tinkerpad, NVIDIA Jetson. В якості доступних одноплатних комп’ютерів для прототипування, що можуть бути використані для основи апаратної частини розподіленої системи, було обрано дві моделі: Raspberry Pi 4 Model B 4GB та NVIDIA Jetson Nano Developer Kit (далі Jetson Nano). Основні характеристики цих моделей одноплатних комп’ютерів наведено у табл. 2.1.

Таблиця 2.1 – Порівняння технічних характеристик одноплатних комп’ютерів Raspberry Pi 4 Model B 4GB та NVIDIA Jetson Nano Developer Kit

Модель	Raspberry Pi 4 Model B 4GB [25]	NVIDIA Jetson Nano Developer Kit [26]
Зовнішній вигляд		
Оперативна пам’ять	4 GB LPDDR4	4 GB 64-bit LPDDR4 25.6 GBps
Центральний процесор	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) @ 1.5GHz	Quad-core ARM A57 @ 1.43 GHz
Графічний процесор	Broadcom VideoCore VI (32-bit)	NVIDIA Maxwell w/ 128 CUDA cores @ 921 Mhz
Роздільна здатність та стандарт відеокодування	1080p 30 fps (H.264)	4K 30 fps, 4x 1080p 30 fps, 9x 720p 30 fps (H.264/H.265)
Роздільна здатність та стандарт відеодекодування	1080p 60 fps (H.264), 4K 60 fps (H.265);	4K 60 fps, 2x 4K 30 fps, 8x 1080p 30 fps, 18x 720p 30 fps (H.264/H.265)
Інтерфейси вводу/виводу	2 × USB 3.0 2 × USB 2.0 1 × 40-pin GPIO Header (J6)	4 × USB 3.0, 1 × USB 2.0 Micro-B 1 × 40-pin GPIO Header (J6)
Мережеві інтерфейси	Gigabit Ethernet, Wi-Fi 802.11ac	Gigabit Ethernet, M.2 Key E
Сховище	Micro-SD	Micro-SD
Вартість(станом на 05.12.2024)	3414 грн [27]	9597 грн [28]

Raspberry Pi 4 Model B 4GB є більш доступною та дешевою моделлю, до того ж існує багато спеціалізованих модулей LoRaWAN для Raspberry Pi. Натомість, NVIDIA Jetson Nano є дорожчим, але має низку переваг, які є

важливими для підтримки роботи нейронної мережі розподіленої системи. Цими перевагами є наявність графічного процесора з 128 ядрами CUDA та більша якість кодування та декодування відео.

Також важливим є початкова орієнтація Jetson Nano на підтримку застосунків, що використовують ШІ. Jetson Nano поставляється з програмним набором JetPack SDK, який містить середовище з прискоренням графіки на базі ОС Linux. JetPack SDK має вбудований набір інструментів для роботи з ядрами CUDA – NVIDIA CUDA Toolkit, бібліотеки cuDNN і TensorRT, а також має підтримку більшості популярних фреймворків для машинного навчання, таких як TensorFlow і PyTorch, а також інструментів для комп'ютерного зору й робототехніки, зокрема OpenCV і ROS.

2.3.2 Вибір камери

Щоб забезпечити матеріал для роботи нейронної мережі, апаратна частина розподіленої системи повинна мати відеокамеру. Головними факторами при виборі камери є її сумісність з одноплатним комп’ютером NVIDIA Jetson Nano, достатня роздільна здатність та доступність на місцевому ринку.

В якості відеокамери для Jetson Nano було вибрано модель IMX477-160 12.3MP виробника Waveshare Electronics (рис. 2.5). Технічні характеристики камери IMX477-160 12.3MP наведено у Таблиці 2.2.

Рисунок 2.5 – Зовнішній вигляд відеокамери IMX477-160 12.3MP [29]

Таблиця 2.2 – Характеристики відеокамери IMX477-160 12.3MP [29]

Матриця	MX477
Роздільна здатність	12.3 мегапікселів, 4056 × 3040
Довжина діагоналі CMOS	7.9 мм
Розмір пікселя	1.55 мкм (H) × 1.55 мкм (V)
Діафрагма	2.2
Фокусна відстань	5.52
Поле зору (FOV):	160°(D) 118°(H) 87°(V)
Спотворення:	<16%
Робоча напруга:	3.3 В
Інтерфейс підключення камери	SCI/MIPI
Формат вихідного відео	RAW12/1 /8, COMP8
Вартість (станом на 05.12.2024) [30]	3 466 грн

Відеокамера IMX477-160 12.3MP призначена для одноплатних комп’ютерів Raspberry Pi Compute Module та Jetson Nano за умови інсталяції відповідного драйвера.

2.4 Вибір модулів LoRaWAN

2.4.1 Різновиди модулів LoRaWAN

Майже всі модулі LoRaWAN мають своєю основою чипи, створені або компанією Semtech, або іншими виробниками за її ліцензією. Semtech має два головних сімейства чипів LoRaWAN: 12xx та 13xx. В залежності від типу чипу модулі діляться на трансивер та концентратор.

Трансивером LoRa називають базовий приймально-передавальний радіомодуль, реалізований за допомогою чипів Semtech сімейства 12xx. До цього сімейства належать чипи серій 126x, 127x та 128x. Чипи сімейства 12xx

реалізують модуляцію LoRa. Трансивери використовуються для вузлів та шлюзів мережі LoRaWAN.

Концентратор LoRa – це багатоканальний високопродуктивний приймально-передавальний радіомодуль, призначений для прийому багатьох пакетів LoRa одночасного. Концентратори використовують чипи сімейства 13xx, що представляють собою цифрову мікросхему базового діапазону (англ. Digital Baseband Chip). Концентратори використовуються для шлюзів мережі LoRaWAN, отримують сигнали від великої кількості вузлів.

2.4.2 Вибір радіочастоти для LoRaWAN

LoRaWAN використовує неліцензовані діапазони радіочастот. Такі діапазони називаються ISM (від англ. industrial, scientific and medical). ISM-діапазони наведені у Регламенті радіозв’язку – основному документі Міжнародного союзу електрозв'язку, що визначає порядок користуванням радіопристроями. Кожна країна має власні, визначені законодавством цієї країни, ISM-діапазони та відповідні норми та правила їх використання.

LoRaWAN використовує лише певні ISM-діпазони, що мають частоту нижче 1 ГГц. Такі ISM-діапазони визначені LoRa Alliance у специфікації LoRaWAN. До них входять: американський US902–928MHz, європейські EU863–870MHz та EU433MHz, китайські CN779–787 MHz і CN470–510MHz та інші [31].

Згідно до чинного законодавства України, пристрої на базі LoRaWAN відносяться до «радіообладнання, випромінювальних пристроїв, експлуатація яких здійснюється за принципом загальної авторизації (без внесення до реєстру присвоєнь радіочастот загальних користувачів)», мають категорію «Телеметрія та радіодистанційне керування / ETSI EN 300 220» і виділений радіочастотний діапазон 868,0–868,6 МГц [32]. Цьому діапазону відповідає LoRaWAN-діапазон EU863–870MHz. Саме тому при виборі модулів LoRa потрібно обирати лише ті, що працюють у цьому діапазоні радіочастот.

2.4.3 Вибір моделей модулів LoRaWAN

На вибір модулів LoRaWAN для розподіленої системи мають вплив такі фактори: наявність підтримки радіочастотного діапазону 868 МГц, ціна, а також використання таких інтерфейсів підключення, які сумісні з одноплатним комп’ютером Jetson Nano. Різним є вибір модулів LoRaWAN для дронів-вузлів та керуючих дронів. Дрони-вузли повинні мати лише трансивер LoRa, а керуючі дрони повинні ще мати концентратор LoRa.

В якості LoRa-трансивера для дронів-вузлів було вирішено використовувати плату Wio-E5 mini (рис. 2.6). Ця плата поєднує чіп Semtech SX126X з мікроконтролером STM32WLE5JC. Wio-E5 mini має вбудовану прошивку з підтримкою AT команд, яка підтримує протокол LoRaWAN та може реалізовувати всі класи вузлів мережі LoRaWAN (A, B та C). Дана плата працює на радіочастоті EU863-870MHz та має інтерфейси підключення USB type C та UART. Wio-E5 mini має два інтерфейси для підключення антени – SMA-K та IPEX [33].

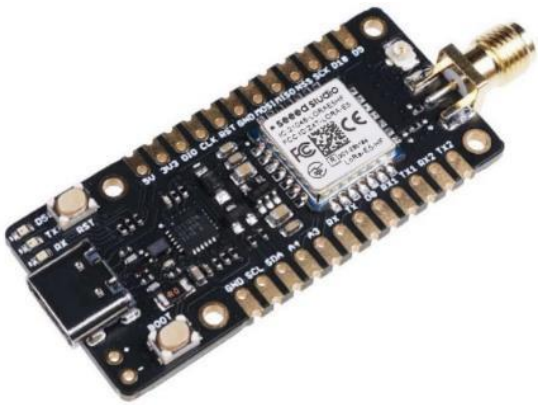


Рисунок 2.6 – Плата Wio-E5 mini [33]

Також дрон-вузол має GPS-модуль Grove-GPS (Air530). Даний модуль підтримує багаторежимне супутникове позиціонування та навігацію з підтримкою більше 6 супутників одночасно [34]. Grove-GPS (Air530) має високу точність позиціонування – похибка позиціонування в межах 10 метрів, та низьке енергоспоживання – всього 31 мкА. Даний GPS-модуль має інтерфейс підключення UART. Антена модуля має інтерфейс U.FL (рис. 2.7).

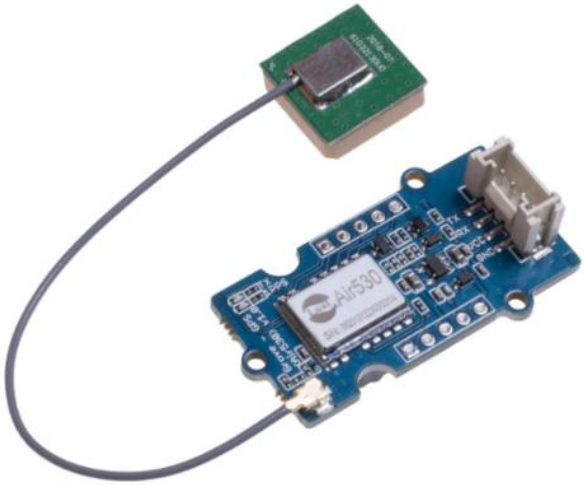


Рисунок 2.7 – Модуль Grove-GPS (Air530) [34]

В якості концентратора LoRaWAN для керуючого дрона було вирішено використовувати концентратор RAK2287 на основі чипу SX1302 (рис. 2.8). Даний чип відрізняється від SX1301 більшою потужністю та кращим температурним режимом.

Рисунок 2.8 – Концентратор RAK2287 [35]

Концентратор RAK2287 має форм-фактор mPCIe та інтерфейси підключення SPI або USB. Через те, що Jetson Nano має 4 USB-порти, то є доцільним використати один з них для концентратора RAK2287. Для цього потрібно застосувати перехідну плату RAK mPCIe to USB Board. Також у комплекті з концентратором йде дві антени: для бездротового зв’язку LoRa та для GPS. Даний концентратор працює на радіочастоті EU863-870MHz [35].

2.5 Підключення апаратної частини

Підключення апаратної частини зображено на рисунках 2.9 та 2.11. Модулі дрона-вузла підключені до одноплатного комп’ютера Jetson Nano наступним чином:

– Камера IMX477-160 12.3MP: до SCI-порту;

– LoRa-трансивер Wio-E5 mini: до порту USB за допомогою кабелю USB Type C;

– GPS-модуль: до пінів UART (рис 2.10).

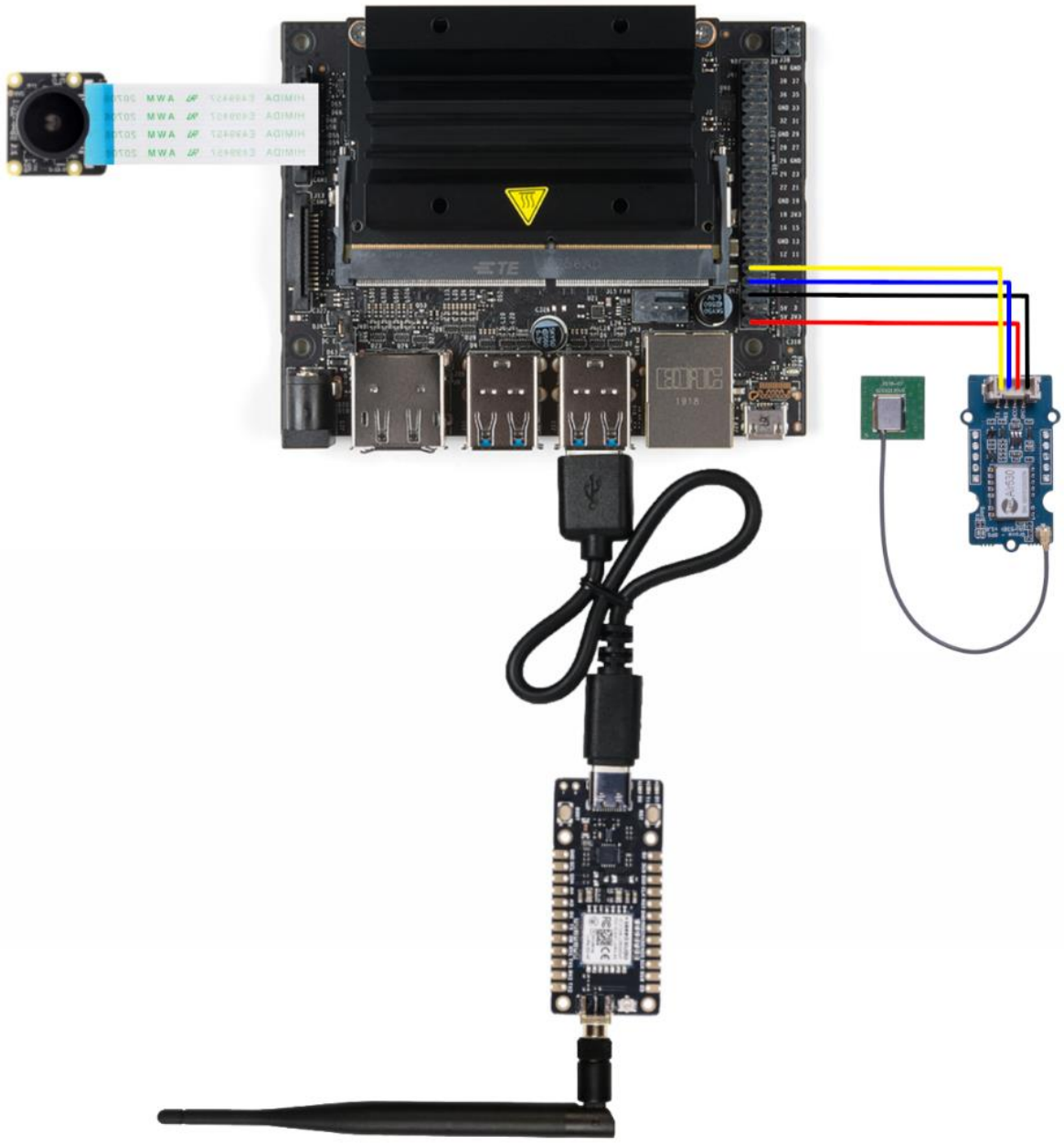


Рисунок 2.9 – Діаграма підключення модулів дрона-вузла

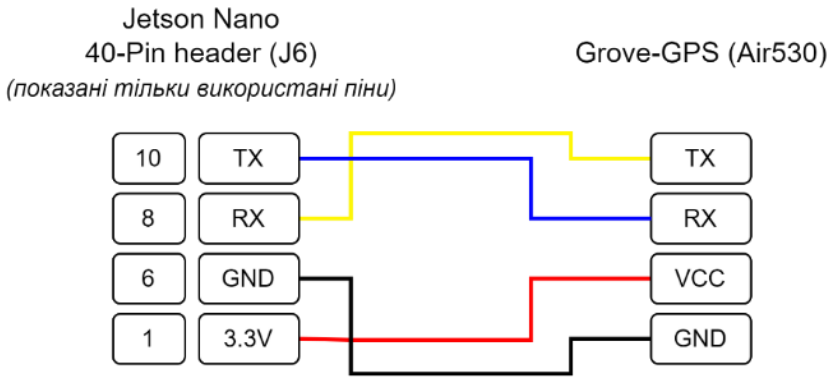


Рисунок 2.10 – Підключення Grove-GPS (Air530) до пінів Jetson Nano

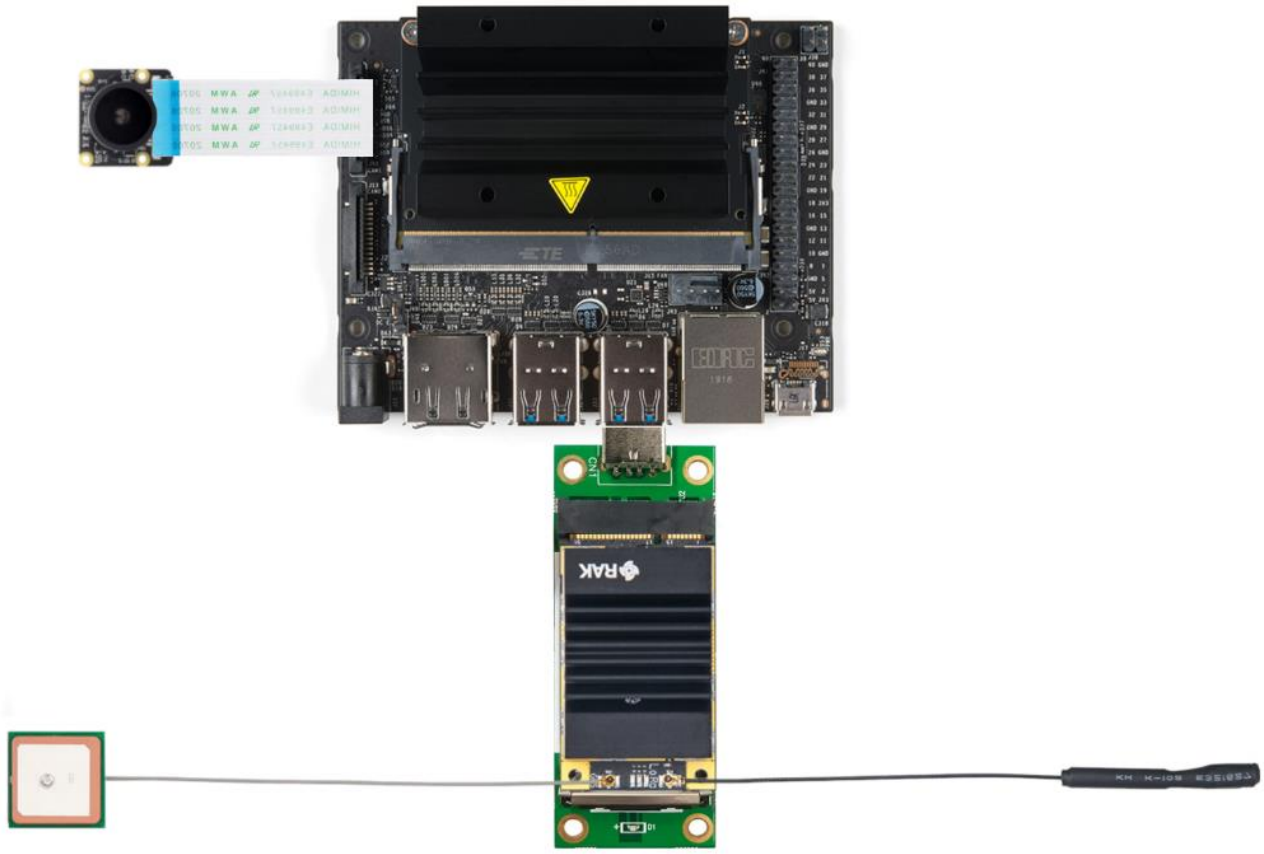


Рисунок 2.11 – Діаграма підключення модулів керуючого дрона

Модулі керуючого пристрою підключені до одноплатного комп’ютера Jetson Nano наступним чином:

– Камера IMX477-160 12.3MP: до SCI-порту;

– Концентратор RAK2287 (з LoRa- та GPS-антенами): до порту USB за допомогою перехідної плати RAK mPCIe to USB Board.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи було описано проєктування розподіленої системи детектування рухомих об'єктів та обміну інформацією на базі LoRaWAN. В якості носія запропонованої системи було обрано рій безпілотних апаратів.

Спочатку було розроблено концептуальне рішення розподіленої системи. Для цього, по-перше, було спроєктовано мережеву архітектуру, що представляє собою дворівневу мережу (рівень рою дронів – P2P-мережа, рівень груп дронів – приватна мережа LoRaWAN). Для забезпечення елементів архітектури LoRaWAN пристрої були поділені на дрони-вузли та керуючі дрони.

По-друге, було визначено принцип роботи пристроїв та рух даних у системі. Кожен дрон спостерігає за визначеною територією та оброблює відеопотік за допомогою нейромережі. При детектуванні об’єкту дрони-вузли передають керуючим дронам дані про детектований об’єкт. Кінцевий користувач взаємодіє з роєм за допомогою клієнтського застосунку. По-третє, було визначено архітектуру та модель нейронної мережі – YOLO, а також спроєктовано інтерфейс клієнтського застосунку.

Далі, для пристроїв розподіленої системи було вибрано відповідно апаратне забезпечення. В якості апаратної основи для всіх пристроїв було вирішено використовувати одноплатний комп’ютер Jetson Nano та сумісну з ним камеру IMX477-160 12.3MP. Дрони-вузли використовують плату Wio-E5 mini в якості LoRa-трансивера та GPS-модуль Grove-GPS (Air530). Керуючі дрони мають підключений через інтерфейс USB LoRA-концентратор RAK2287, що також має вбудований GPS-датчик. У кінці розділу було наведено діаграми підключення апаратного забезпечення для дронів-вузлів та керуючих дронів.

3 РОЗРОБКА АПАРАТНО-ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РОЗПОДІЛЕНОЇ СИСТЕМИ

3.1 Вибір програмних засобів розробки нейронної мережі

Для розробки нейронної мережі та програми обробки її результатів було вирішено доцільним використовувати мову програмування Python, фреймворк машинного навчання PyTorch, бібліотеки Ultralytics та OpenCV.

Python – це високорівнева інтерпретована мова програмування, що має динамічну типізацію та автоматичне керування пам'яттю (збирання сміття). Центральною ідею Python є використання максимально простого синтаксису, достатньо схожого на псевдокод. Python має потужну стандартну бібліотеку та широку екосистему сторонніх бібліотек та фреймворків, серед яких PyTorch та Tensorflow – для машинного навчання, та OpenCV – для комп’ютерного зору. Через ці особливості Python є однією з найпопулярніших мов програмування у сферах машинного навчання, аналізу даних та наукових розрахунків.

PyTorch – це фреймворк для машинного навчання з відкритим вихідним кодом. PyTorch є інтерфейсом-«обгорткою» бібліотеки Torch, що написана мовою програмування C++. Головними особливостями PyTorch є обчислення даних у виді тензорів (матрично-подібних структур даних) з прискоренням за допомогою графічних процесорів, та використання автоматичного диференціювання на основі плівки (англ. tape-based autograd) при глибокому навчанні. PyTorch відрізняється від свого головного конкурента, Tensorflow більш мінімалістичним, інтуїтивним та зручним інтерфейсом, що побить процес прототипування більш простим та швидким.

Використання прискорення на графічному процесорі фреймворком PyTorch забезпечує більш швидше навчання нейронних мереж. Основа цього процесу – використання технології CUDA від NVIDIA. CUDA представляє собою платформу та відповідний програмний інтерфейс, що дозволяє

використовувати певні графічні процесори для виконання загальних паралельних розрахунків.

Ultralytics – це бібліотека машинного навчання для Python, яка містить найсучасніші моделі нейронних мереж сімейства YOLO, впроваджує нові функції та інструменти для підвищення продуктивності та гнучкості мереж, а також зручної роботи з ними. Дана бібліотека використовує PyTorch в якості бекенду, що забезпечує швидке навчання та роботу нейронних мереж за допомогою прискорення на графічному процесорі. В даній КМР бібліотека Ultralytics використовується для реалізації моделі YOLOv11.

OpenCV – це бібліотека з відкритим вихідним кодом для комп'ютерного зору та обробки зображень. Вона надає інструменти для таких завдань, як розпізнавання та обробка зображень, виявлення об'єктів, аналіз відео та машинне навчання. Основа «бекенду» бібліотеки OpenCV, як у PyTorch, написана мовою програмування C++, але інтерфейси OpenCV мають Python, Java та MATLAB. Запропонована розподілена система використовує OpenCV для обробки відеопотоку та фільтрування рухомих об’єктів від статичних (за допомогою алгоритму віднімання фону).

3.1.1 Вибір датасету для навчання нейронної мережі

Датасет – це набір даних для тренування чи тестування нейронної мережі. Тренувальні та тестові датасети для детектування об’єктів складаються з зображень або відео, які містять анотовані об’єкти. Анотація зазвичай має вигляд прямокутної рамки, що позначає межі об’єкту. Іноді використовуються маски – повний контур об’єкта. У випадку, якщо задачею є класифікація об’єктів, то анотація буде також містити назву об’єкта.

На вибір датасету для нейронної мережі детектування рухомих об’єктів впливають декілька факторів. По-перше, в якості матеріалу даних шуканого датасету повинні бути відеофрагменти. По-друге, такі відеофрагменти повинні бути зняті за допомогою розташованої на безпілотному апараті камери.

Ці вимоги задовольняє датасет VisDrone. Цей датасет був зібраний командою AISKYEYE в Лабораторії машинного навчання та інтелектуального аналізу даних Університету Тяньцзіня, Китай. Даний датасет складається з 288 відеокліпів, сформованих з 261 908 кадрів, і 10 209 статичних зображень, знятих встановленими на дронах камерами в 14 різних містах Китаю. Матеріал датасету має різні: навколишнє середовище (міське і сільське), об'єкти (люди, транспортні засоби тощо) та щільність сцен (малолюдні і переповнені). Також датасет містить такі важливі атрибути, як видимість сцени, клас об'єкта та оклюзія (перекриття об’єктів одне одним). [36].

VisDrone поділяється на 5 частин, призначених для різних задач: VisDrone-DET (детектування об'єктів на зображеннях), VisDrone-VID (детектування об'єктів на відео), VisDrone-SOT (відстеження одного об'єкта), VisDrone-MOT (відстеження декількох об'єктів) та DroneCrowd (підрахунок людей у натовпі). Приклад даних датасету VisDrone-VID показано на рис. 3.1.

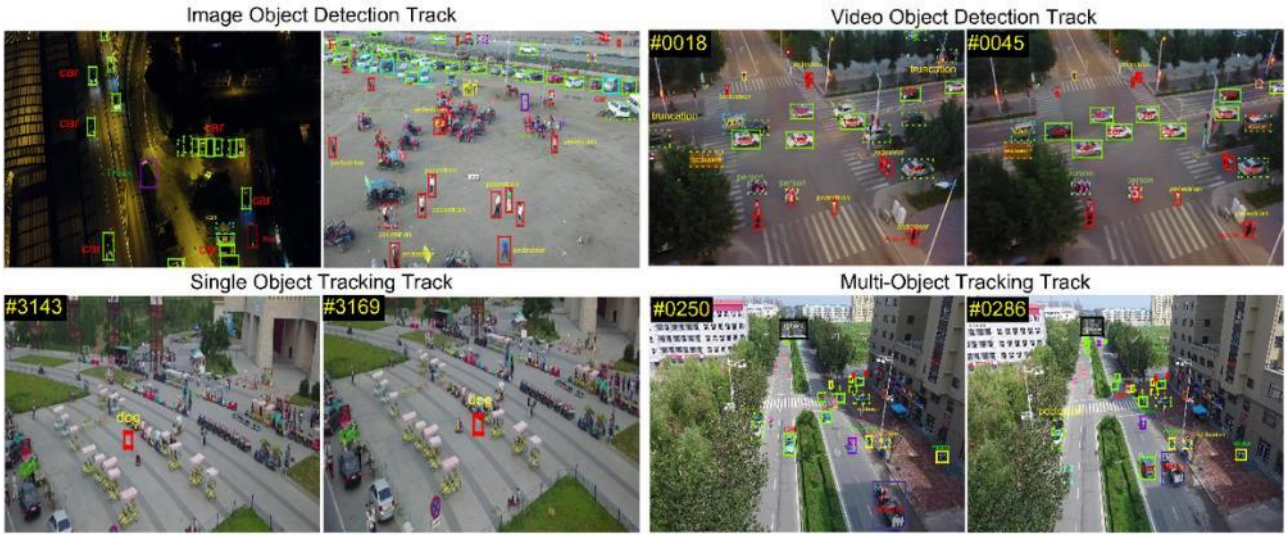


Рисунок 3.1 – Датасет для детектування об'єктів на відео VisDrone-VID [36]

Саме другий датасет VisDrone-VID буде використовуватися для навчання нашої нейронної мережі для детектування рухомих об’єктів. Він складається з тренувального датасету trainset (7,53 Гбайт), датасету валідації valset (1,49 Гбайт), та тестових датасетів testset-dev (2,14 Гбайт) і testset-challenge (2,70 Гбайт).

3.2 Розробка нейронної мережі та програми детектування

Нейронна мережа розподіленої системи представляє собою реалізацію моделі згорткової мережі YOLOv11 за допомогою бібліотеки Ultralytics для мови програмування Python. Для розробки нейронної мережі на Jetson Nano було встановлено наступні версії програмного забезпечення та бібліотек:

– Python 3.11;
– PyTorch 2.5.1+cu121;
– CUDA Toolkit 12.3;
– OpenCV 4.9;
– Ultralytics 8.3.30.

Використання бібліотеки Ultralytics на Jetson Nano має певні технічні нюанси. Встановлена за допомогою менеджера пакетів pip, бібліотека Ultralytics не буде працювати на Jetson Nano через те, що дана апаратна платформа базується на архітектурі ARM64. Тому для установки сумісної з Jetson Nano версії (тієї, що використовує JetPack 4) необхідно скористатися програмним забезпеченням Docker.

```
$ t=ultralytics/ultralytics:latest-jetson-jetpack4 && sudo docker
pull $t && sudo docker run -it --ipc=host --runtime=nvidia $t
```

Рисунок 3.2 – Команда Docker для установки Ultralytics на Jetson Nano

У Linux-консолі Jetson Nano за допомогою команди Docker, показаної на рис. 3.2, встановлюється сумісна з JetPack 4 бібліотека Ultralytics. Встановлена таким чином бібліотека має всі необхідні залежні компоненти, включаючи Torch та Torchvision.

3.2.1 Навчання нейронної мережі

Навчання нейронної мережі відбувається за допомогою скрипту, зображеного на рис. 3.3. Для того, щоб нейронна мережа YOLO могла використовувати завантажений датасет VisDrone2019-VID в якості матеріалу

для навчання потрібно, по-перше, налаштувати конфігураційний файл датасету, а по-друге – конвертувати анотації датасету у формат YOLO.

Конфігураційний файл датасету visdrone2019-vid.yaml (зміст файлу наведено у Додатку А) містить інформацію про шляхи до датасету, а також про класи об’єктів у датасеті. Конвертація анотацій відбувається за допомогою скрипту vis2yolo.py (рис. 3.4). Даний скрипт конвертує обмежувальні рамки у формат “YOLO xywh”, та перетворює анотації на файли з мітками labels.txt.

train_nn.py

```python
from ultralytics import YOLO

# Завантаження PyTorch-моделі YOLOv11n
model = YOLO("yolo11n.pt")

# Конвертація моделі на формат TensorRT для Jetson Nano
model.export(format="engine")
trt_model = YOLO("yolo11n.engine")

# Навчання моделі на датасеті VisDrone2019-VID
results = model.train(data="visdrone2019-vid.yaml", epochs=100, imgsz=480, amp=False)
```

Рисунок 3.3 – Скрипт навчання нейронної мережі train_nn.py

vis2yolo.py

```python
import os
import sys


# Конвертує анотації датасету VisDrone у формат YOLO
def convert(dir):
    from PIL import Image
    from tqdm import tqdm

    # Конвертація обмежувальних рамок
    def convert_box(size, box):
        dw = 1. / size[0]
        dh = 1. / size[1]
        return (box[0] + box[2] / 2) * dw, (box[1] + box[3] / 2) * dh, box[2] * dw, box[3] * dh

    (dir + '/labels').mkdir(parents=True, exist_ok=True)
    pbar = tqdm((dir + '/annotations').glob('*.txt'), desc=f'Converting {dir}')
    for f in pbar:
        img_size = Image.open((dir + '/images' + '/' + f.name).with_suffix('.jpg')).size
        lines = []
        with open(f, 'r') as file:
            for row in [x.split(',') for x in file.read().strip().splitlines()]:
                if row[4] == '0':
                    continue
                cls = int(row[5]) - 1
                box = convert_box(img_size, tuple(map(int, row[:4])))
                lines.append(f"{cls} {' '.join(f'{x:.6f}' for x in box)}\n")
                with open(str(f).replace(f'{os.sep}annotations{os.sep}', f'{os.sep}labels{os.sep}'), 'w') as fl:
                    fl.writelines(lines)
```

Рисунок 3.4 – Скрипт конвертації анотацій у формат YOLO vis2yolo.py (повний код наведено в Додатку А)

Після закінчення навчання мережі з’являється директорія *runs*. За шляхом “runs\detect\train\weights” знаходяться файли best.pt та last.pt. Дані файли містять навчені моделі мережі, які містять значення параметрів: last.pt – останньої епохи, best.pt – значення епохи, при якій мережа показала найкращий результат. Саме останній файл треба використовувати у подальшому.

3.2.2 Основна програма детектування

Натренована нейронна мережа застосовується до відеопотоку з камери дрона у основній програмі детектування camera_detection.py (рис. 3.5). Програма імпортує модель нейронної мережі best.pt. Далі за допомогою OpenCV реалізується алгоритм віднімання фону та розпочинається запис відео.

camera_detection.py

```
# Основний цикл детектування
while True:
    # Зчитування кадрів з відеопотоку
    success, frame = vcap.read()
    # Зупинити зчитування, якщо відопотік закінчився
    if not success:
        break
    ...
    # Застосувати віднімання фону до кадру відеопотоку
    mask = back_sub.apply(frame)
    # Застосувати нейромережу до кадру відеопотоку
    results = model.track(frame, persist=True)

    # Фільтрація рухомих об'єктів
    moving_objects = []
    for box in results[0].boxes:
        obj_id = box.id  # Унікальний ID об'єкта
        x1, y1, x2, y2 = map(int, box.xyxy[0])  # Координати обмежувальної рамки
        conf = box.conf  # Точність детектування об'єкта
        cls = box.cls  # Клас об'єкта
        # Перевірка на рух
        if mask[y1:y2, x1:x2].mean() > 25:
            moving_objects.append((obj_id, x1, y1, x2, y2, conf, cls))
            # Виведення повідомлення про детектування об'єкта
            pu.print_result(pu.format_result(box), pu.classNames)
            # Створення обмежувальної рамки з об'єктом
            annotated_frame = results[0].plot()
            # Збереження кадру з об'єктом
            cv2.imwrite(f"{int(obj_id)}_{int(cls[0])}_%.2f.jpg" % float(conf), annotated_frame)
            # Збереження даних про о'єкт
            pu.write_result(pu.format_result(box))
```

Рисунок 3.5 – Основний цикл програми детектування camera_detection.py (повний код програми наведено в Додатку А)

У основному циклі програми camera_detection.py до кожного послідовного відеокадру застосовується алгоритм віднімання фону, а потім – нейронна мережа. Далі, фільтруються всі детектовані об’єкти за допомогою порівняння маски переднього плану з обмежувальними рамками об’єктів. Якщо обмежувальна маска збігається з переднім планом, то такий об’єкт не вважається статичним та додається до масиву рухомих об’єктів.

У кінці циклу зберігаються кадр з детектованим рухомим об’єктом та дані про нього (ID, клас об’єкта, точність детектування від 0 до 1, а також GPS-координати дрона, що знайшов об’єкт) для подальшої передачі. Функція зберігання даних та інші допоміжні функції були винесені у файл project_utils.py, код якого наведено у Додатку А.

3.3 Розробка програмного забезпечення P2P-мережі

Для розробки програмного забезпечення P2P-мережі верхнього рівня було вирішено використовувати мову програмування C++. Це компільована мова програмування зі статичною типізацією. Її Головною особливістю є поєднання об’єктно-орієнтованого програмування з елементами низькорівневого програмування. C++ надає розробнику інструменти прямого доступу до пам’яті та апаратних і системних ресурсів, що забезпечує високу продуктивність програмного коду. Саме через ці переваги C++ було обрано для розробки мережевих застосунків, які працюють у реальному часі.

З’єднання за допомогою P2P-мережі реалізується за допомогою написаного мовою програмування C++ застосунку (рис. 3.6). Даний застосунок реалізовує клієнтський застосунок для операційних систем на базі ядра Linux, який може з’єднуватись за допомогою протоколу TCP з іншими подібними застосунками для прийому та відправлення даних.

Головним файлом проекту є P2Pmain.cpp, який об’єднує два одночасних потоки (рис. 3.7). Перший потік (потік прийому) діє як сервер, а другий потік (потік відправки) – як клієнт, що працює в рамках одного загального процесу.

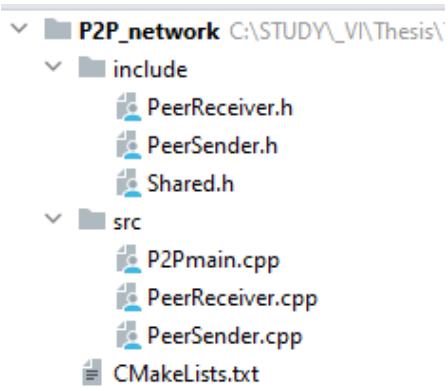


Рисунок 3.6 – Зміст директорії проєкту P2P_network

```
// Потік отримання даних
void StartThePeerReceiver(string PeerReceiverName,string PeerReceiverPort){
    std::unique_ptr<PeerReceiver> rec(new PeerReceiver(PeerReceiverName,PeerReceiverPort));
    rec->StartReceiver();
}

// Потік відправлення даних
void StartThePeerSender(){
    sleep(1);
    std::unique_ptr<PeerSender> send(new PeerSender());
    //PeerSender* send = new PeerSender();
    send->RegisterPeer();
    send->FileDownload();
}

int main(){
    string PeerReceiverName ="";
    string PeerReceiverPort ="";
    cout << "Введіть ваше ім'я хоста \n";
    cin >> PeerReceiverName;
    cout << "Введіть ваш номер порту \n";
    cin >> PeerReceiverPort;

    thread PeerReceiver(StartThePeerReceiver,PeerReceiverName,PeerReceiverPort);
    thread StartTheSender(StartThePeerSender);
    PeerReceiver.join();
    StartTheSender.join();
    return 0;
```

Рисунок 3.7 – Головний файл P2P-застосунку P2Pmain.cpp
(повний код наведено в Додатку А)

Клас потоку прийому відповідальний за прийняття та надання даних (у виді файлів) іншим користувачам у мережі. Цей клас реалізований у файлах з назвою PeerReceiver. Клас потоку відправки відповідальний за підключення до інших користувачів у мережі та передачу команд на завантаження певних файлів. Даний клас реалізований у файлах з назвою PeerSender.

3.4 Розгортання приватної мережі LoRaWAN

Для побудови приватної мережі LoRaWAN було вирішено доцільним використовувати мережеве забезпечення з відкритим вихідним кодом ChirpStack. Програмне забезпечення ChirpStack використовується для побудови та керування серверною частиною мережі LoRaWAN, а саме: мережевим сервером та сервером застосунків. Окрім наведених серверів ChirpStack включає в себе шлюзовий міст, завдяки якому шлюзи мережі LoRaWAN з’єднуються з мережевим сервером. З останній, четвертій версії у Chirpstack було об’єднано мережевий сервер та сервер застосунків, що полегшує розгортання мережі та взаємодію з нею [24].

3.4.1 Створення вузла LoRaWAN на основі модуля Wio-E5 mini

Для ініціалізації вузла мережі LoRaWAN, потрібно налаштувати підключений до дрона-вузла модуль Wio-E5 mini. Даний модуль використовує AT-команди для його програмування. Щоб застосувати AT-команди до модуля потрібно використати програму чи бібліотеку для взаємодії з serial-портами. Було вирішено в якості такої бібліотеки використати бібліотеку для Python під назвою pyserial.

Код програми налаштування вузла на базі модуля Wio-E5 mini lora_wioE5.py наведено на рис. 3.8. За допомогою цієї програми перевіряється підключення модуля через порт USB до Jetson Nano, отримується та виводиться ID модуля, а також модуль програмується на використання радіочастоти EU868 МГц.

Так як дрони-вузли мають GPS-модуль Grove-GPS (Air530), то для його коректної роботи потрібно спочатку ініціалізувати порт його підключення. Даний модуль підключено до пінів 8 та 10, що є інтерфейсом UART. У Jetson Nano йому відповідає порт “/dev/ttyTHS1”. Для налаштування даного порту для його використання модулем Grove-GPS (Air530) застосовується програма grove_gps.py (рис. 3.9).

lora_wioE5.py

```python
# Налаштування вузла з Wio-E5 mini
def set_node():
    # Створення порта для Wio-E5 mini (USB-підключення)
    # Через обмеження Jetson Nano, необхідно використовувати baudrate = X * 1.02
    port = serial.Serial(port='/dev/ttyUSB0', baudrate=9600 * 1.02, bytesize=8, parity='N', stopbits=1, timeout=1,
                         xonoff=False, rtscts=False, dsrdtr=False)

    # Перевірка підключення
    test_cmd = "AT\r"
    port.write(test_cmd.encode())
    test_msg = port.read(64)
    print(test_msg)  # +AT: OK

    # Отримання ID модуля Wio-E5 mini
    id_cmd = "AT+ID\r"
    port.write(id_cmd.encode())
    module_id = port.read(64)
    print(module_id)

    # Налаштування частоти EU868 МГц
    dr_cmd = "AT+DR=EU868\r"
    port.write(dr_cmd.encode())
    dr = port.read(64)
    print(dr)
```

Рисунок 3.8 – Код програми налаштування вузла lora_wioE5.py

(повний код наведено в Додатку А)

grove_gps.py

```python
# Grove GPS (Air530) підключений до пінів UART: 8 - RX; 10 - TX
import serial
import time
import gps_api  # Бібліотека для роботи з Grove GPS (Air530)


# Створення порта для UART-підключення
def initialize_port():
    serial_port = serial.Serial(
        port="/dev/ttyTHS1",
        baudrate=115200,
        bytesize=serial.EIGHTBITS,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
    )
    time.sleep(1) # Час на ініціалізацію порта
    return serial_port
```

Рисунок 3.9 – Код програми налаштування GPS-модуля grove_gps.py

(повний код наведено в Додатку А)

Після налаштування модулів Wio-E5 mini та Grove-GPS (Air530) вузол вважається готовим для під’єднання до Chirpstack-сервера приватної мережі LoRaWAN.

3.4.2 Налаштування шлюзу мережі LoRaWAN

Для реалізації шлюзу LoRaWAN керуючі дрони використовують концентратор RAK2287. Для налаштування даного модулю необхідно встановити ПЗ від виробника за допомогою команд, показаних на рис. 3.10. Остання команда установки має параметр, який виключає Chirpstack з переліку ПЗ, для того, щоб не встановлювати стару, третю версію Chirpstack.

```
git clone https://github.com/RAKWireless/rak_common_for_gateway.git
cd ~/rak_common_for_gateway
sudo ./install.sh --chirpstack=not_install
```

Рисунок 3.10 – Команди для установки ПЗ для RAK2287

Після виконання цих команд у консолі з’явиться меню, де потрібно обрати опцію «8» з модулем RAK2287, підключеним за допомогою інтерфейсу USB (рис. 3.11).

```
Please select your gateway model:
*         1.RAK2245
*         2.RAK7243/RAK7244 no LTE
*         3.RAK7243/RAK7244 with LTE
*         4.RAK2247(USB)
*         5.RAK2247(SPI)
*         6.RAK2246
*         7.RAK7248 no LTE (RAK2287 SPI + raspberry pi)
*         8.RAK7248 with LTE (RAK2287 SPI + LTE + raspberry pi)
*         9.RAK2287 USB
*         10.RAK5146 USB
Please enter 1-10 to select the model:
```

Рисунок 3.11 – Вибір установки ПЗ для RAK2287 з USB-підключенням

Далі потрібно встановити ПЗ для роботи з RAK2287 на Linux-системі Jetson Nano за допомогою команд, показаних на рис. 3.12. Після цього у директорії packet_forwarder потрібно перейменувати файл “global_conf.json.sx1250.EU868.USB” (рис. 3.13) на “global_conf.json”.

```
wget https://github.com/Lora-net/sx1302_hal/archive/V2.0.1.tar.gz
tar -zxvf V2.0.1.tar.gz
cd sx1302_hal-2.0.1
sudo make
cd packet_forwarder
```

Рисунок 3.12 – Команди установки для роботи з RAK2287 на Linux

```
ubuntu@ubuntu:~/sx1302_hal-2.0.1/packet_forwarder$ ls -l
total 380
-rw-rw-r-- 1 ubuntu ubuntu   2489 Dec 18  2020 Makefile
-rw-rw-r-- 1 ubuntu ubuntu  17939 Dec 18  2020 PROTOCOL.md
-rw-rw-r-- 1 ubuntu ubuntu   5603 Dec 18  2020 global_conf.json.sx1250.AS923.USB
-rw-rw-r-- 1 ubuntu ubuntu   4281 Dec 18  2020 global_conf.json.sx1250.CN490
-rw-rw-r-- 1 ubuntu ubuntu   5776 Dec 18  2020 global_conf.json.sx1250.CN490.USB
-rw-rw-r-- 1 ubuntu ubuntu   5841 Dec 18  2020 global_conf.json.sx1250.EU868
-rw-rw-r-- 1 ubuntu ubuntu   5777 Dec 18  2020 global_conf.json.sx1250.EU868.USB
-rw-rw-r-- 1 ubuntu ubuntu   4907 Dec 18  2020 global_conf.json.sx1250.US915
-rw-rw-r-- 1 ubuntu ubuntu   4863 Dec 18  2020 global_conf.json.sx1250.US915.USB
-rw-rw-r-- 1 ubuntu ubuntu   4026 Dec 18  2020 global_conf.json.sx1255.CN490.full-duplex
-rw-rw-r-- 1 ubuntu ubuntu   4604 Dec 18  2020 global_conf.json.sx1257.EU868
drwxrwxr-x 2 ubuntu ubuntu   4096 Dec 18  2020 inc
-rwxr-xr-x 1 root   root   267800 Sep  8 12:50 lora_pkt_fwd
drwxr-xr-x 2 root   root     4096 Sep  8 12:50 obj
-rw-rw-r-- 1 ubuntu ubuntu  12017 Dec 18  2020 readme.md
drwxrwxr-x 2 ubuntu ubuntu   4096 Dec 18  2020 src
ubuntu@ubuntu:~/sx1302_hal-2.0.1/packet_forwarder$
```

Рисунок 3.13 – Конфігураційні файли концентратора RAK2287

Після цього шлюз на базі концентратора RAK2287 може розпочати свою роботу за допомогою консольної команди sudo ./lora_pkt_fwd.

3.4.3 Налаштування серверу мережі LoRaWAN. Підключення вузлів та шлюзів до серверу

Наступним кроком є налаштування серверу Chirpstack четвертої версії. Спочатку треба завантажити з віддаленого репозиторію директорію з конфігураційними файлами та файлами установки серверу Chirpstack за допомогою програного забезпечення Docker (рис. 3.14).

```
git clone https://github.com/chirpstack/chirpstack-docker.git
cd chirpstack-docker
```

Рисунок 3.14 – Завантаження директорії з ПЗ Chirpstack

У директоріях configuration/chirpstack та configuration/chirpstack-gateway-bridge було змінено файли конфігурації мережевого серверу та шлюзового міста так, щоб Chirpstack використовував радіочастотний діапазон EU868. Зміст цих конфігураційних файлів chirpstack.toml та chirpstack-gateway-bridge.toml наведено у Додатку А.

Далі, сервер запускається за допомогою команди docker-compose up у терміналі ОС. Вузли та шлюзи LoRaWAN підключаються до серверу Chipstack

за допомогою веб-інтерфейсу за адресою localhost:8080. Спочатку створюється спільний профіль вузлів drone_node (рис. 3.15), у якому вказується радіочастота EU868 та версія стандарту LoRaWAN (Wio-E5 mini підтримує версії 1.0.2 і 1.0.3).

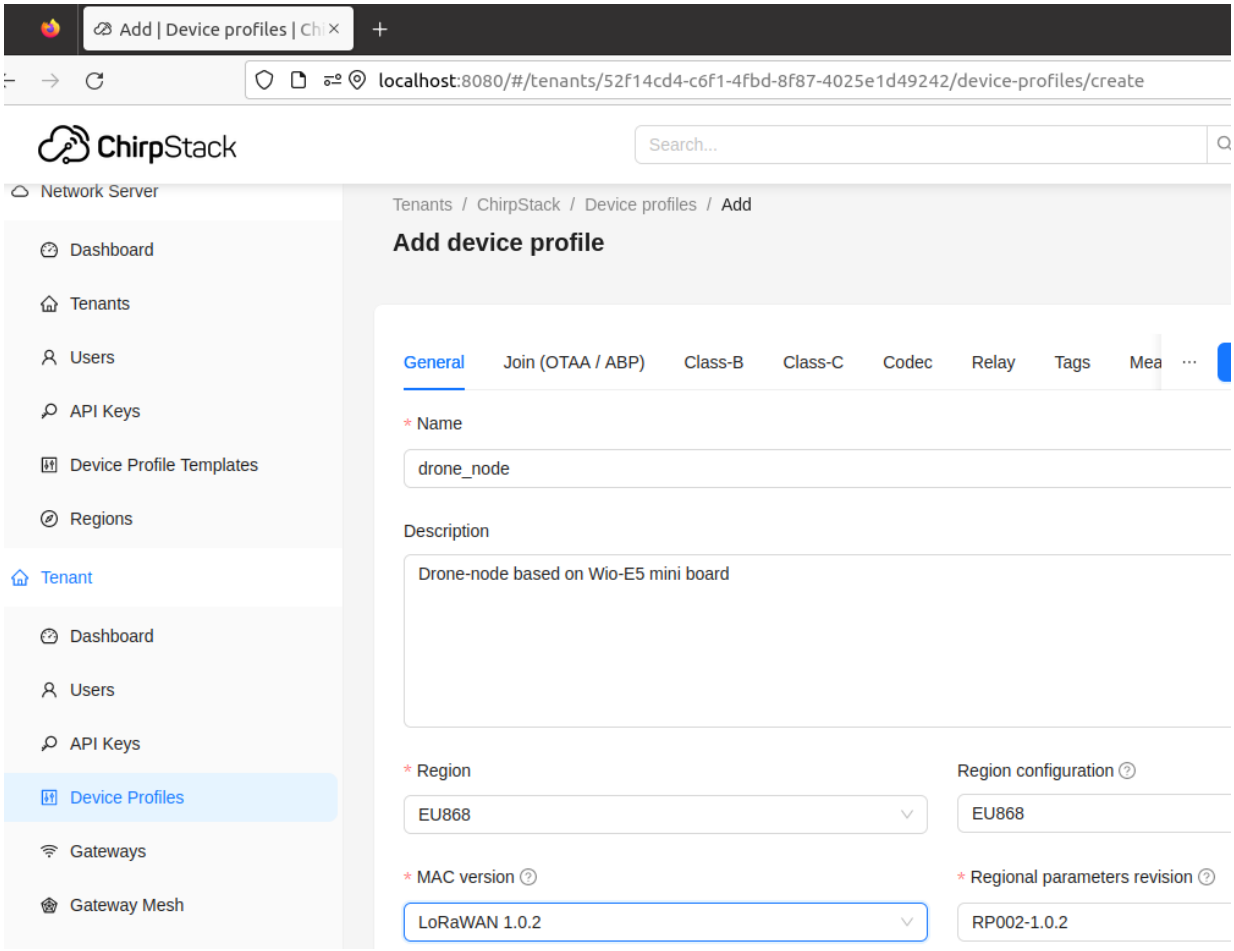


Рисунок 3.15 – Створення профіля кінцевого пристрою drone_node для дрона-вузла у Chirpstack

Наступним кроком, за допомогою меню “Gateways/Add gateway” до мережі Chirpstack додаються керуючі дрони, які в свою чергу є шлюзами мережі LoRaWAN (рис. 3.16). При додаванні шлюзу потрібно вказати ідентифікатор EUI64, який належить самому концентратору LoRa.

Для підключення вузлів мережі LoRaWAN потрібно спочатку створити застосунок, що працює поверх приватної мережі LoRaWAN. Для цього за допомогою меню “Applications/Add application” було створено застосунок під назвою “distributed_system_app”.

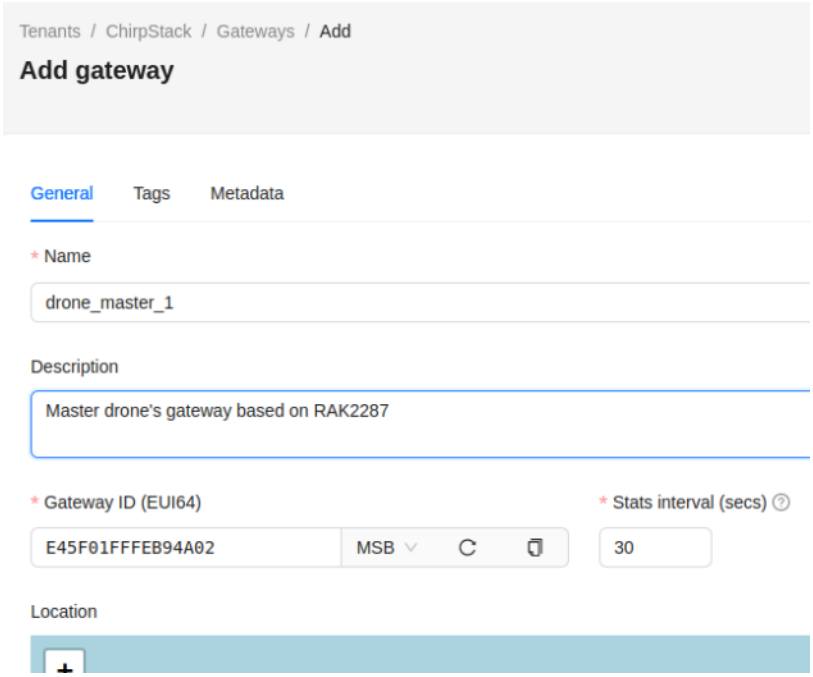


Рисунок 3.16 – Додання шлюзу LoRaWAN (керуючий дрон) у Chirpstack

Тільки після створення застосунку distributed_system_app до мережі можуть бути додані вузли. Під час додавання вузлів необхідно вказати ідентифікатор EUI пристрою LoRaWAN (у нашому випадку – це Wio-E5 mini) та створений раніше профіль drone_node (рис. 3.17).

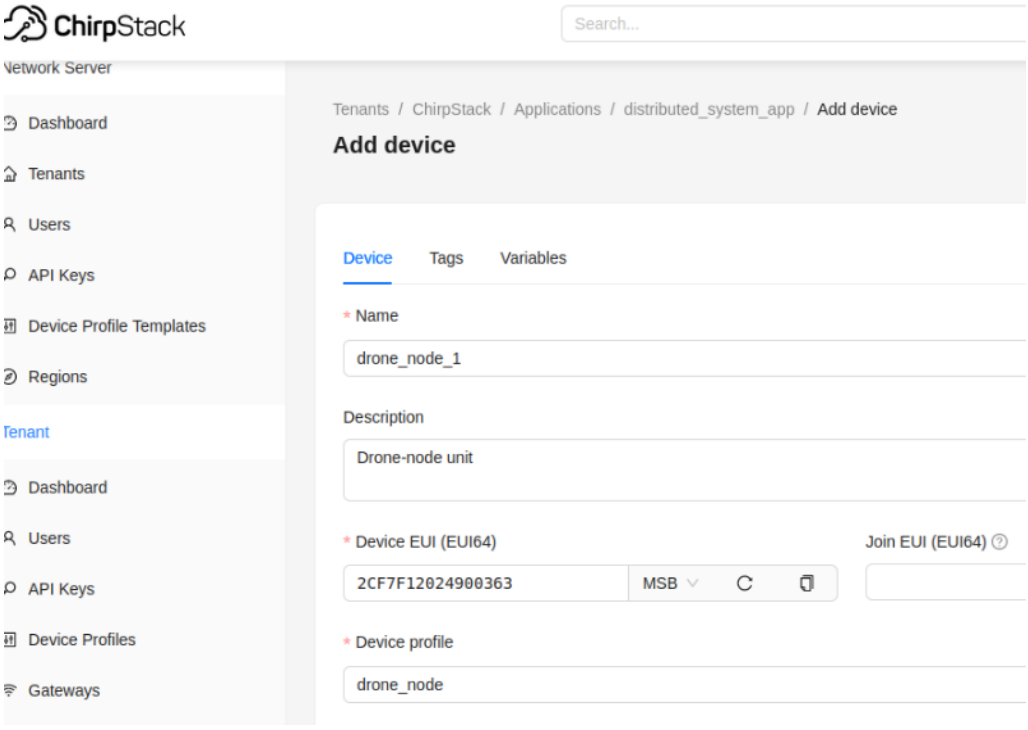


Рисунок 3.17 – Додання вузла до застосунку за профілем drone_node

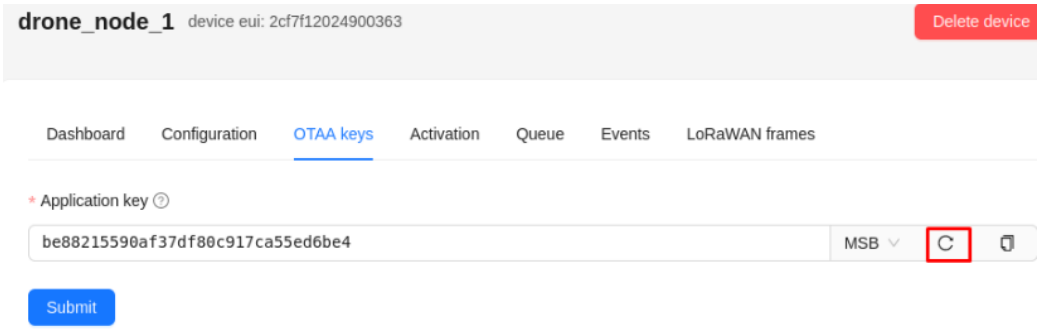


Рисунок 3.18 – Генерація ключа для вузла

Останнім кроком генерується ключ застосунку формату OTAA для доданого вузлу (рис. 3.18).

3.5 Базовий графічний інтерфейс клієнтського застосунку

Для створення базового графічного інтерфейсу користувача використовувались фреймворк Qt, мови програмування C++ і QML, та програмне середовище розробки Qt Creator.

QT являє собою фреймворк для створення графічних інтерфейсів користувача та кросплатформених застосунків мовою програмування C++. QT має багато модулів, у тому числі – QT Quick для більш гнучкої розробки графічних інтерфейсів, який використовує мову програмування QML. Для полегшення написання QT-програм використовувалось спеціалізоване середовище розробки Qt Creator.

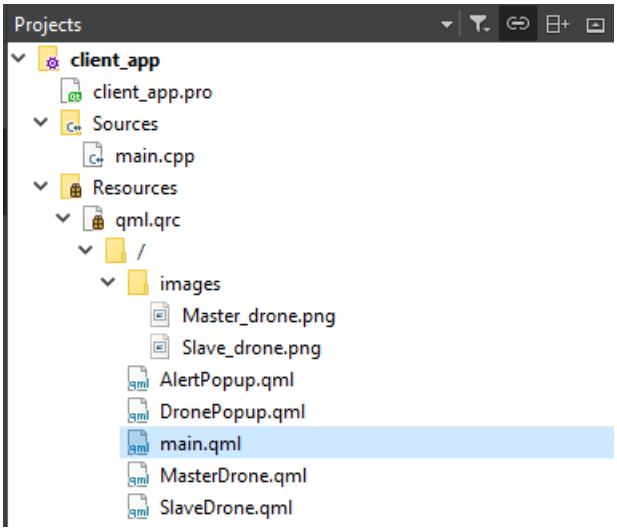


Рисунок 3.19 – Структура проєкту з інтерфейсом клієнтського застосунку

Проєкт з QT-інтерфейсом містить головні файли main.cpp та main.qml, файли графічних об’єктів QML – MasterDrone.qml (керуючий дрон), SlaveDrone.qml (дрон-вузол), DronePopup.qml (вікно з інформацією про дрон) та AlertPopup.qml (вікно з повідомленням про детектований об’єкт), а також піктограми (рис. 3.19).

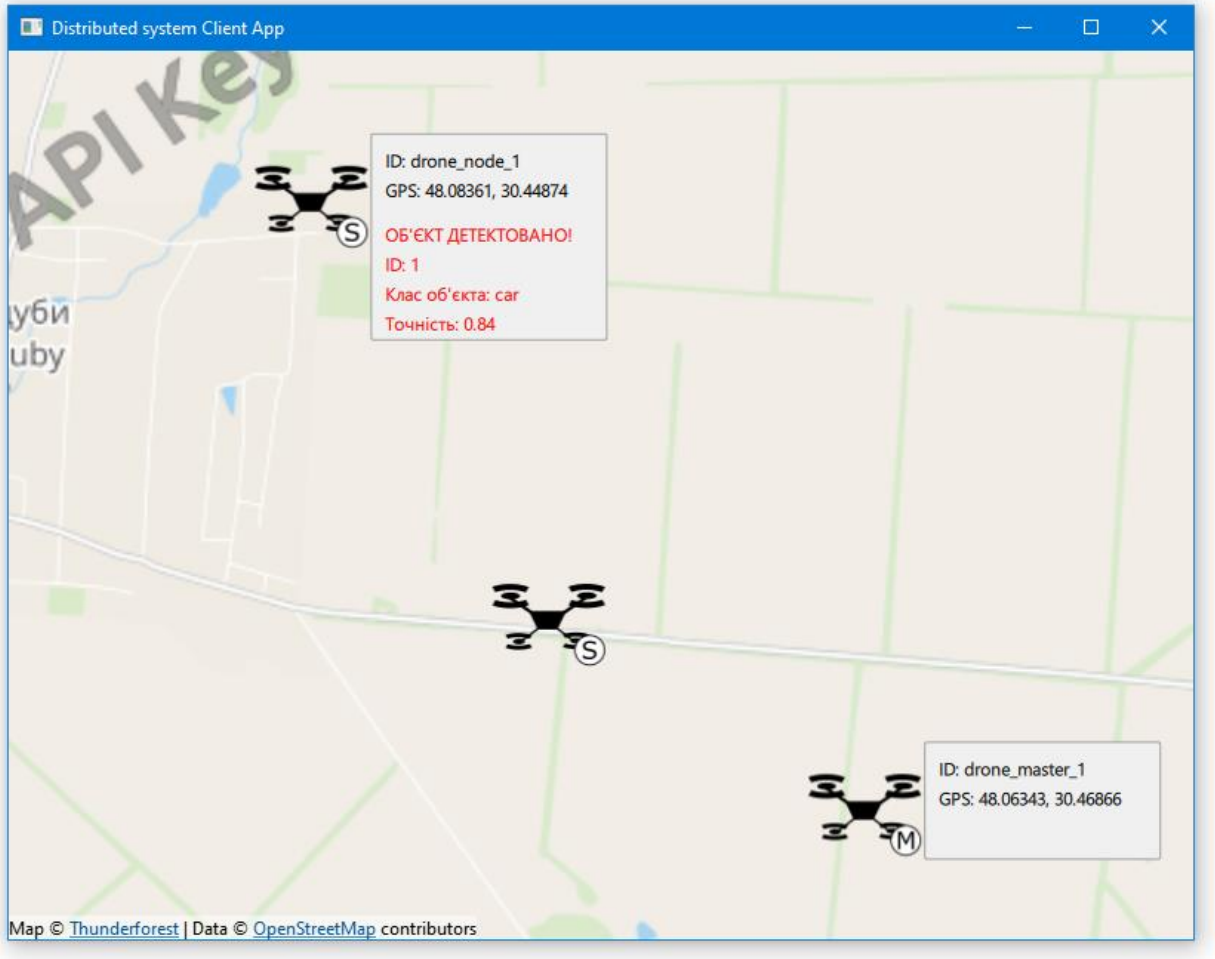


Рисунок 3.20 – Графічний інтерфейс клієнтського QT-застосунку

Графічний інтерфейс клієнтського застосунку використовує інтерактивну карту з відкритим кодом OpenStreetMap, на яку наносяться піктограми з дронами, місцеположення яких співпадає з їх GPS-координатами. При натисканні на піктограму показується вікно з інформацією про дрон (ID та координати). Якщо дрон детектував об’єкт, то інформація про об’єкт (ID, клас, точність детектування від 0 до 1) також з’являється у вікні (рис. 3.20).

Код головного файлу main.qml наведено у Додатку А.

Висновки до розділу 3

У даному розділі було описано процес розробки та налаштування апаратно-програмного забезпечення розподіленої системи.

Для розробки нейронної мережі та програми детектування було вирішено використовувати мову програмування Python, фреймворк PyTorch, бібліотеки Ultralytics та OpenCV. Нейронна мережа навчається за допомогою скрипту train_nn.py, а потім імпортується у файл детектування camera_detection.py. У ньому вхідний відеопотік з камери оброблюється нейронною мережею для детектування об’єктів, та знайдені рухомі об’єкти фільтруються від статичних за допомогою алгоритму віднімання фону. У кінці циклу зберігаються кадр з детектованим об’єктом та дані про нього (ID, клас, точність детектування, а також GPS-координати дрона, що знайшов об’єкт).

Застосунок обміну даними у P2P-мережі розроблено за допомогою мови програмування C++. Він представляє собою програму, що об’єднує два одночасних потоки: для прийому та для відправки даних у виді файлів.

Для розгортання мережі LoRaWAN застосовувалось програмне забезпечення Chirpstack. Було описано налаштування вузлів та шлюзів за допомогою відповідного програмного забезпечення (у тому числі власних скриптів lora_wioE5.py та grove_gps.py). Далі, було детально описано розгортання серверу Chirpstack та додання шлюзів і вузлів до нього.

Графічний інтерфейс користувача розроблено із використанням фреймворку QT та мов програмування C++ і QML. Графічний інтерфейс представляє собою інтерактивну карту, на яку наносяться піктограми з дронами, місцеположення яких співпадає з їх GPS-координатами. При натисканні на піктограму показується вікно з інформацією про дрон (ID та координати) та детектований об’єкт (ID, клас, точність детектування від 0 до 1).

4 ТЕСТУВАННЯ КОМПОНЕНТІВ РОЗПОДІЛЕНОЇ СИСТЕМИ

4.1 Тестування навчання нейронної мережі

Навчання мережі може здійснюватися на самому одноплатному комп’ютері Jetson Nano, або спочатку на більш потужних пристроях, і лише потім навченою переноситись до сховища Jetson Nano.

Тестове навчання мережі YOLOv11 виконувалось на персональному комп’ютері, що має відеокарту NVIDIA GTX 1650 та 16 Гбайт оперативної пам’яті. NVIDIA GTX 1650 має 896 ядер CUDA та 4 Гбайт відеопам’яті.

При навчанні були обрані такі параметри:

– epochs=100 (Кількість епох – 100);

– imgsz=480 (Розмір зображення – 480; значення за замовчуванням – 640);

– amp=False (Не використовувати режим «змішаної точності», так як у ході тестування було визначено, що він конфліктує з відеокартою NVIDIA GTX 1650, що призводить до неправильного навчання мережі).

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     99/100      5.48G      1.333     0.9022     0.8796        191        480: 100%|██████████| 405/405 [02:15<00:00,  2.99it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 18/18 [00:07<00:00,  2.56it/s]
                   all        548      38759      0.366      0.275      0.259      0.146
  0%|          | 0/405 [00:00<?, ?it/s]
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100      4.71G      1.329     0.8968     0.8803        307        480: 100%|██████████| 405/405 [02:09<00:00,  3.13it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 18/18 [00:07<00:00,  2.56it/s]
                   all        548      38759      0.365      0.275       0.26      0.146

100 epochs completed in 5.246 hours.
Optimizer stripped from runs\detect\train\weights\last.pt, 5.4MB
Optimizer stripped from runs\detect\train\weights\best.pt, 5.4MB

Validating runs\detect\train\weights\best.pt...
Ultralytics 8.3.30 🚀 Python-3.11.1 torch-2.5.1+cu121 CUDA:0 (NVIDIA GeForce GTX 1650, 4096MiB)
YOLO11n summary (fused): 238 layers, 2,584,102 parameters, 0 gradients, 6.3 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 18/18 [00:12<00:00,  1.40it/s]
                   all        548      38759      0.366      0.275      0.261      0.147
            pedestrian        520       8844      0.372      0.273      0.258      0.102
                people        482       5125      0.475      0.164      0.203     0.0672
               bicycle        364       1287      0.138     0.0606     0.0432      0.014
                   car        515      14064      0.568      0.692      0.683      0.453
                   van        421       1975      0.391      0.314      0.297      0.198
                 truck        266        750      0.385      0.257      0.236      0.145
             tricycle        337       1045      0.305      0.215      0.166      0.084
       awning-tricycle        220        532      0.206       0.12     0.0824     0.0479
                   bus        131        251      0.449      0.371       0.38       0.26
                 motor        485       4886      0.367      0.289      0.258     0.0981
Speed: 0.2ms preprocess, 4.3ms inference, 0.0ms loss, 4.0ms postprocess per image
Results saved to runs\detect\train

Process finished with exit code 0
```

Рисунок 4.1 – Вивід в консолі при закінченні навчання нейронної мережі

Як видно з рис. 4.1, навчання нейронної мережі YOLOv11 з такими обраними параметрами зайняло 5,246 годин (що становить 5 годин 14 хвилин 46 секунд). Відеокарта GTX 1650 має в 7 разів більше ядер CUDA, ніж графічний процесор одноплатного комп’ютера Jetson Nano (896 проти 128), тому навчання нейронної мережі засобами Jetson Nano зайняло би в декілька разів більше часу. Останній спосіб не є доцільним, хоча і є цілком можливим (особливо при менших значеннях параметрів навчання).

Рисунок 4.2 – Результати роботи мережі на датасеті валідації VisDrone

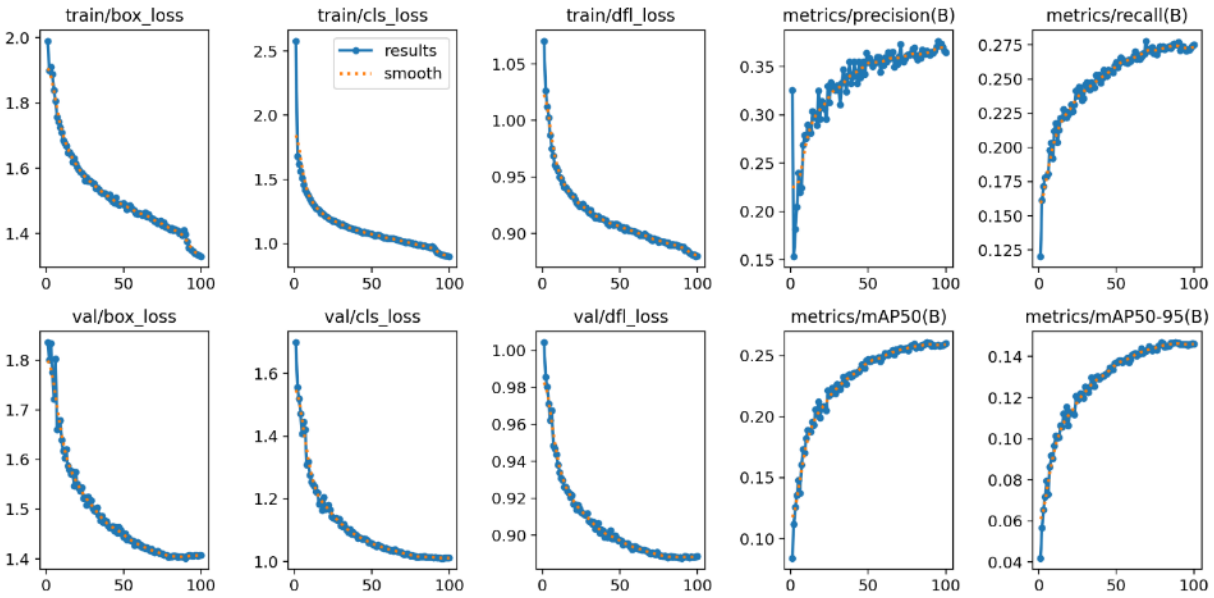


Рисунок 4.3 – Згенеровані графіки показників ефективності мережі

Після навчання нейронної мережі автоматично генеруються графіки показників ефективності мережі (рис. 4.3), Excel-файл з результатами під час кожної епохи та скріншоти перших двох перевірок мережі на датасеті валідації (приклад зображено на рис. 4.2).

4.2 Перевірка працездатності нейронної мережі та програми детектування

Після навчання нейронної мережі натреновану модель з найкращими значеннями параметрів за шляхом “runs/detect/train/weights/best.pt” було використано у програмі детектування camera_detection.py. Для симуляції відеопотоку з камери дрона було використано відеофайл test_video.mp4, яке являє собою завантажене відео “aerial view on car driving through autumn forest road scenic a” з відеохостингу YouTube [37]. На даному відео зображено рухомий автомобіль, знятий за допомогою безпілотного апарату.

Для тестування під час роботи програми детектування виводилось вікно з відео. Приклад роботи програми детектування зображено на рис. 4.4 (кадри з часовим проміжком 0,5 секунди). Під час виконання обробки відео було збережено кадр з детектованим об’єктом та дані про нього у форматі «ID об’єкта; клас об’єкта; точність детектування; ID дрона; широта; довгота», де символом розділення даних є символ табуляції (рис. 4.5).

Рисунок 4.4 – Робота програми детектування на відеофайлі test_video.mp4

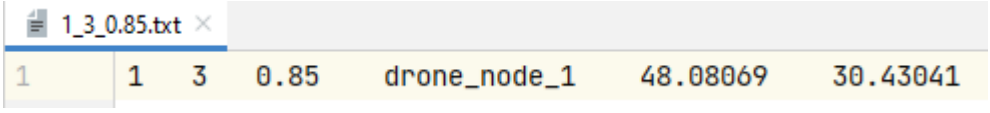


Рисунок 4.5 – Збережені дані про детектований об’єкт

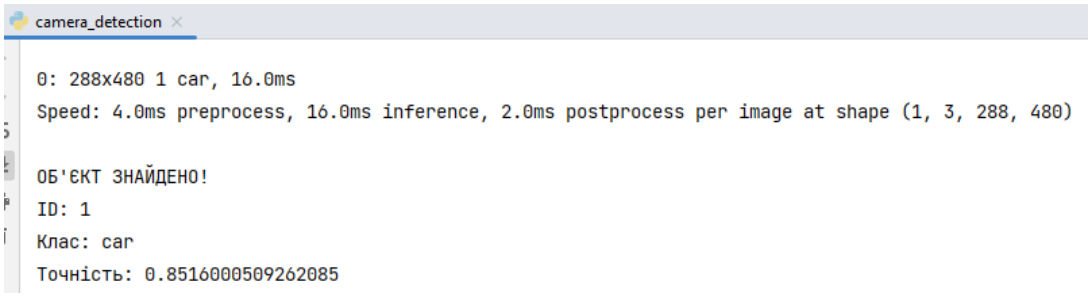


Рисунок 4.6 – Вивід у консолі при обробці останнього кадру відеофайлу

На рис. 4.6 показано вивід у консолі для останнього кадру відеофайлу. Перші два рядки є логами роботи нейронної мережі YOLOv11. Вони містять інформацію про розмір кадру, ID об’єкту, його клас та швидкість роботи мережі. Як зазначає другий рядок, 4 мілісекунд зайняла попередня обробка відеокадру, 16 мс робота власне нейронної мережі та 2 мс постобробка кадру. Загальний час роботи нейронної мережі на детектування об’єкту на кадрі становить 22 мс, що є гарним результатом для системи, що працює у реальному часі.

4.3 Тестування приватної мережі LoRaWAN

Для тестування приватної мережі LoRaWAN було вибрано LWN Simulator. Це програмне забезпечення представляє собою симулятор вузлів LoRaWAN із вбудованим веб-інтерфейсом. LWN Simulator симулятор дозволяє взаємодіяти або з реальною інфраструктурою LoRaWAN, або з ad-hoc мережевою інфраструктурою, такою як Chirpstack.

Для тестування використовується LWN Simulator версії 1.0.2, так як в ході виконання кваліфікаційної магістерської роботи було виявлено, що останні версії симулятора не можуть підключитися до мережевого сервера Chirpstack. Команди установки та запуску LWN Simulator показані на рис. 4.7. Важливо зазначити, що даний симулятор потребує встановлений пакет мови програмування Go версії 1.16 чи новіше.

```
wget https://github.com/UniCT-ARSLab/LWN-Simulator/archive/refs/tags/v1.0.2.tar.gz
tar -xvzf v1.0.2.tar.gz
cd LWN-Simulator-1.0.2
make install-dep
make run
```

Рисунок 4.7 – Команди установки та запуску LWN Simulator

```
daniel@jetson:~/NanoProject/LWN-Simulator-1.0.2$ make run
2024/12/08 20:53:15 LWN Simulator is online...
2024/12/08 20:53:15 [WS]: Listen [ 0.0.0.0:8000 ]
2024/12/08 20:53:29 [WS]: Socket connected
2024/12/08 20:53:36 [SIM]: SETUP OK!
2024/12/08 20:53:36 [SIM]: START
2024/12/08 20:53:39 [SIM]: Status saved
```

Рисунок 4.8 – Початок роботи симулятора LWN Simulator

Після запуску веб-інтерфейс LWN Simulator доступний за локальною адресою localhost:8000 (рис. 4.8). Для підключення симулятора до працюючого мережевого серверу Chirpstack потрібно у меню “Gateway Bridge” вказати адресу та порт шлюзового мосту. У нашому випадку сервер Chirpstack також працює локально, тому його шлюзовий міст має адресу та порт localhost:1700 (рис. 4.9).

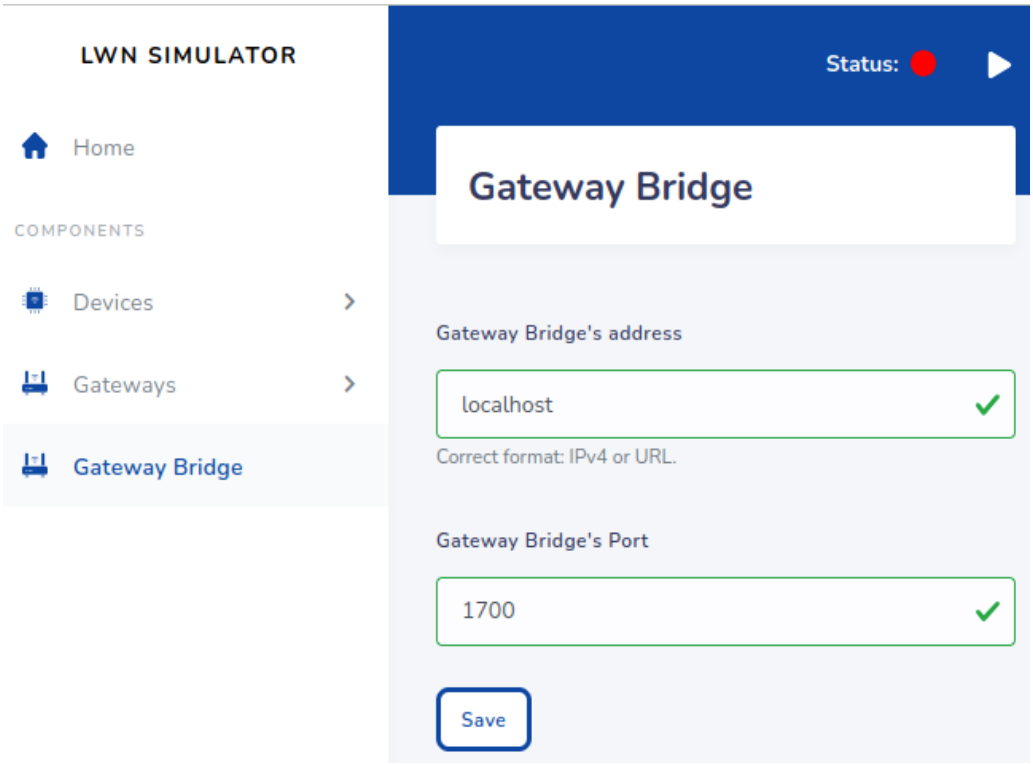


Рисунок 4.9 – Підключення LWN Simulator до шлюзового мосту Chirpstack

Після цього було створено віртуальний шлюз drone_master_1 та згенеровано MAC-адресу (рис. 4.10). При цьому можливо змінювати значення KeepAlive та GPS-координати шлюзу.

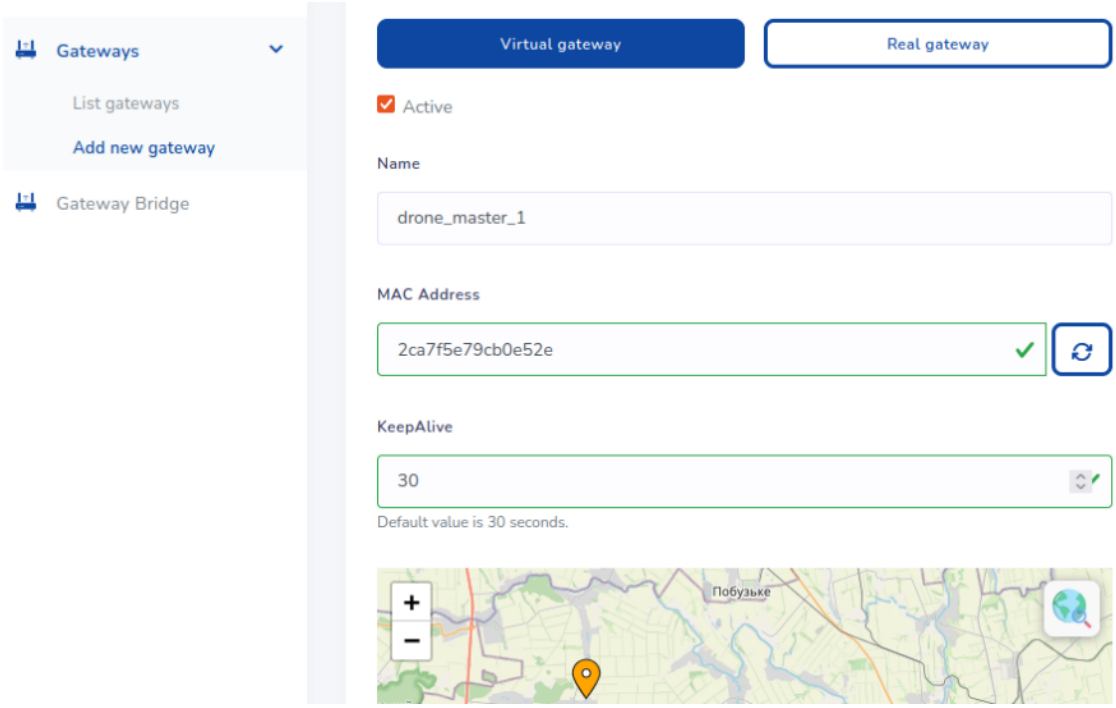


Рисунок 4.10 – Створення віртуального шлюзу

Віртуальний вузол drone_node_1 було створено за допомогою меню “Add new device” (рис. 4.11). Для вузла було згенеровано ідентифікатор DevEUI, вибрано регіон EU868 та згенеровано ключ активації OTAA.

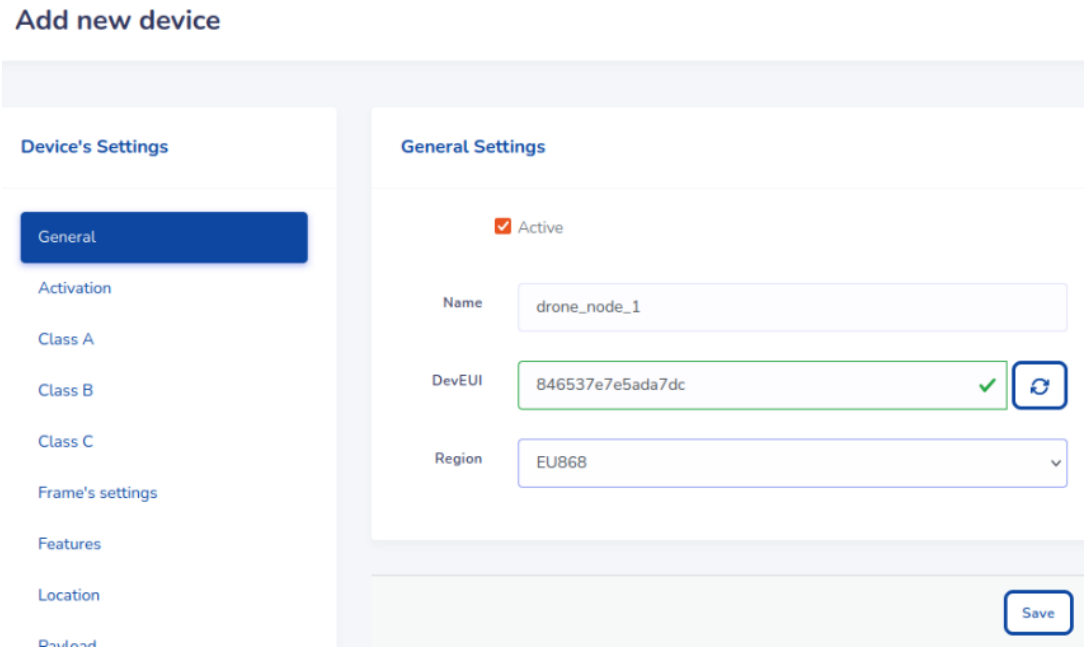


Рисунок 4.11 – Створення віртуального вузла

Для задач тестування в якості корисного навантаження вузла було використано дані про детектований об’єкт (рис. 4.12).

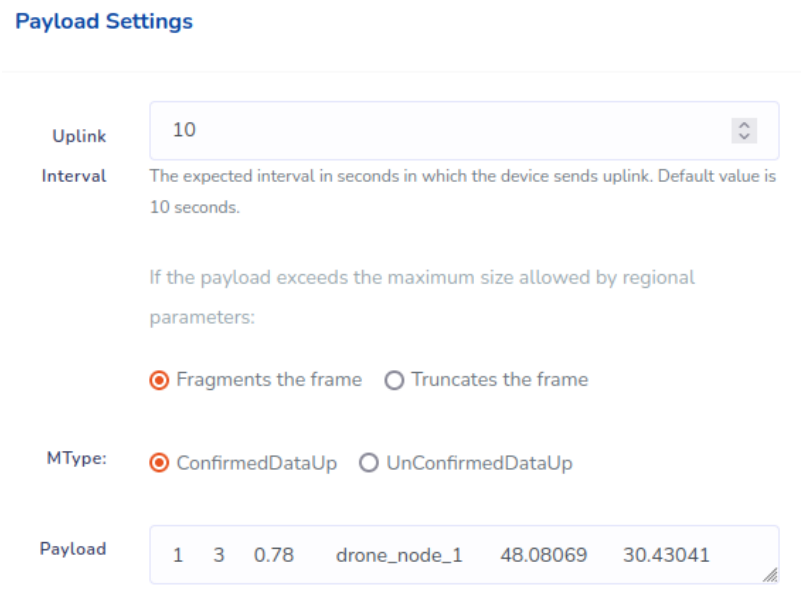


Рисунок 4.12 – Корисне навантаження вузла (дані про об’єкт)

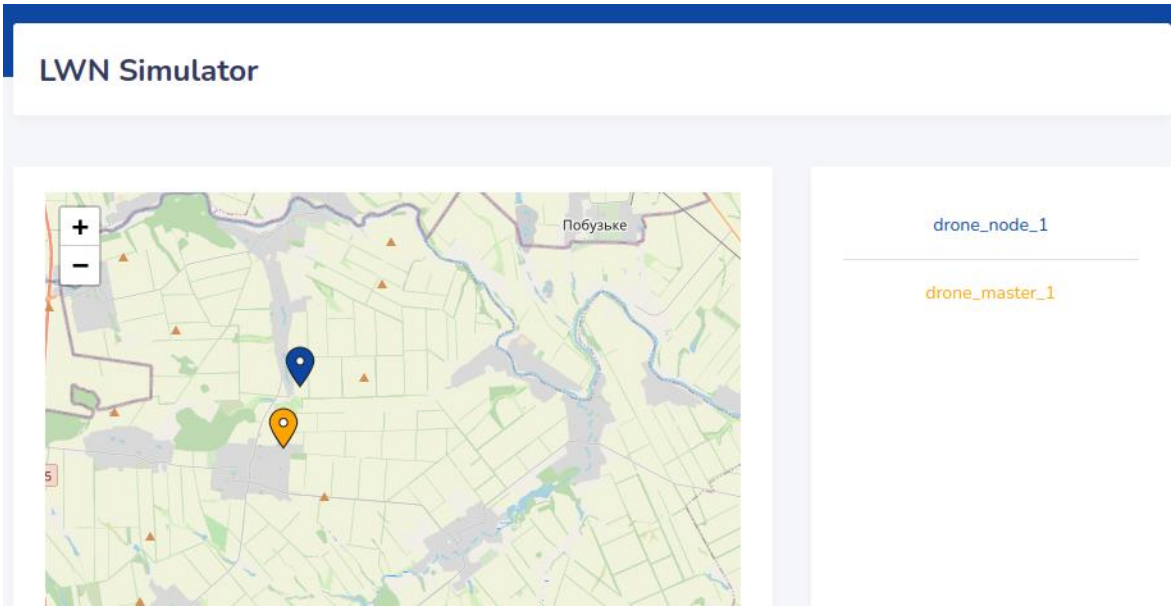


Рисунок 4.13 – Список віртуальних шлюзів та вузлів та їх геолокація

Після створення віртуальних шлюзів та вузлів у LWN Simulator (рис. 4.13) їх потрібно підключити до мережевого серверу Chirpstack. Для цього за допомогою веб-інтерфейсу Chirpstack було створено шлюзи та вузли за процедурою, показаною у третьому розділі КМР, із використанням відповідних MAC-адрес, ідентифікаторів та ключів активації, згенерованих у LWN Simulator.

Як видно з рис. 4.14, створені віртуальні шлюзи та вузли є підключеними до мережевого серверу Chirpstack та мають статус «Активні».

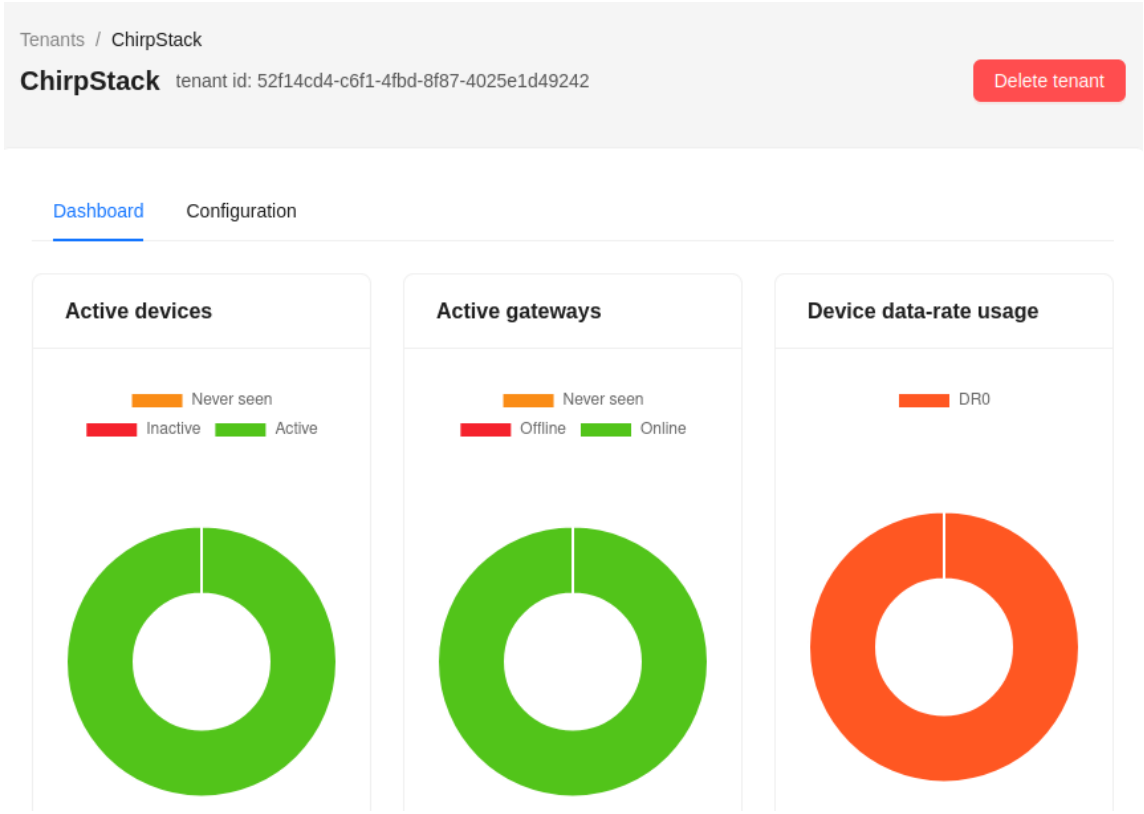


Рисунок 4.14 – Підключені до Chirpstack віртуальні вузли та шлюзи

У меню “LoRaWAN frames” вузла drone_node_1 показується лог повідомлень, що надсилаються до мережевого серверу (рис. 4.15). За допомогою меню “Events” можна побачити дані, які передає вузол (рис. 4.16). Дані, що передаються з вузла до серверу, мають кодування base64.

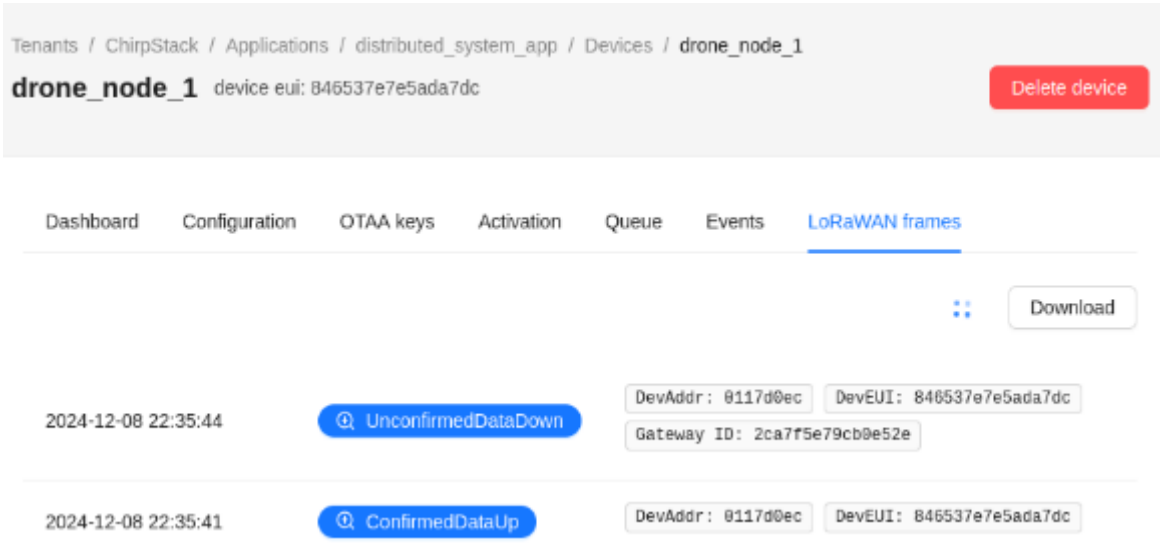


Рисунок 4.15 – Лог повідомлень вузла drone_node_1

Рисунок 4.16 – Отримані з вузла дані у кодуванні base64

Перевірити цілісність даних можна декодувавши рядок “data” за допомогою написаної функції decode_base64() з файлу project_utils.py (повний код наведено у Додатку А).

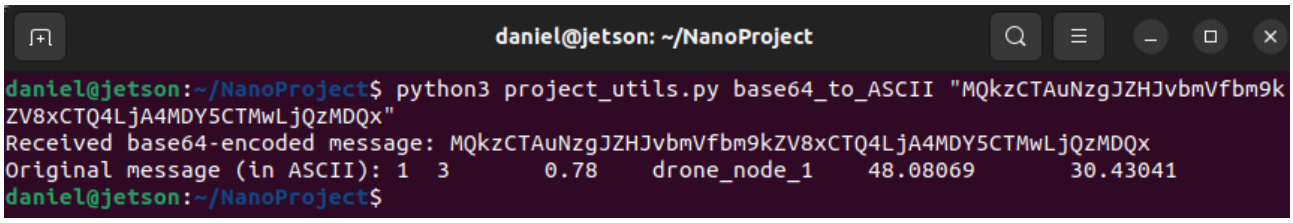


Рисунок 4.17 – Результат декодування base64-рядка

Як видно з рис. 4.17, дані з вузла були отримані без помилок, що свідчить про правильність налаштування та працездатність приватної мережі LoRaWAN.

4.4 Перевірка працездатності P2P-мережі

Для тестування працездатності застосунку P2P-мережі було створено дві директорії drone_master_1 та drone_master_2 (рис. 4.18), що представляють двох користувачів спільної P2P-мережі. Кожна директорія має виконуваний файл застосунку P2P, побудований на основі програмного коду проєкту P2P_network,

описаного у третьому розділі КМР. Також у кожній директорії є субдиректорія shared, в якій знаходяться файли для обміну між користувачами у P2P-мережі. drone_master_1 має у такій субдиректорії файл data1.txt, що містить дані про детектований об’єкт (рис. 4.19), а у drone_master_2 ця субдиректорія є пустою.

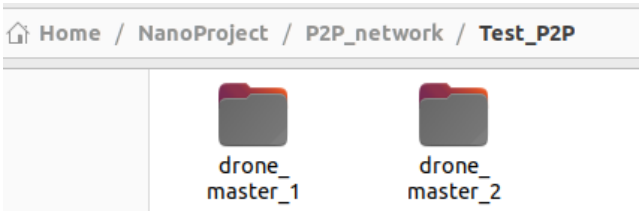


Рисунок 4.18 – Директорії для тестування P2P-застосунку

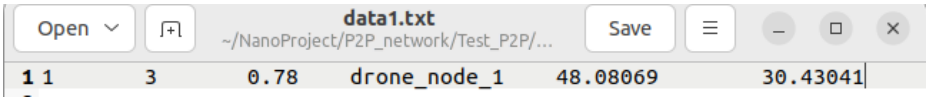


Рисунок 4.19 – Зміст файлу з даними data1.txt

З-під drone_master_1 було запущено файл застосунку та відкрито порт першого користувача localhost:2001 (рис. 4.20). Аналогічно з drone_master_2 було відкрито порт localhost:2002 та установлено з’єднання з першим користувачем (рис. 4.21).

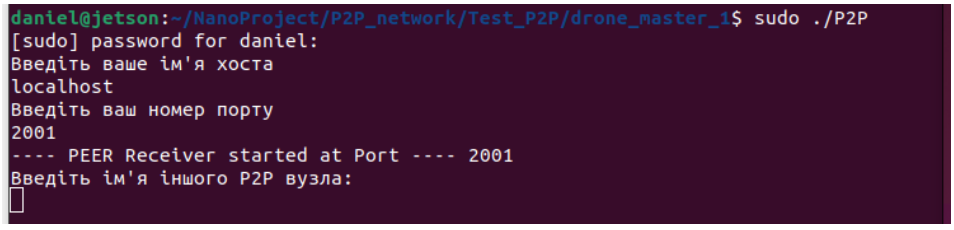


Рисунок 4.20 – Ініціалізація першого користувача P2P-мережі

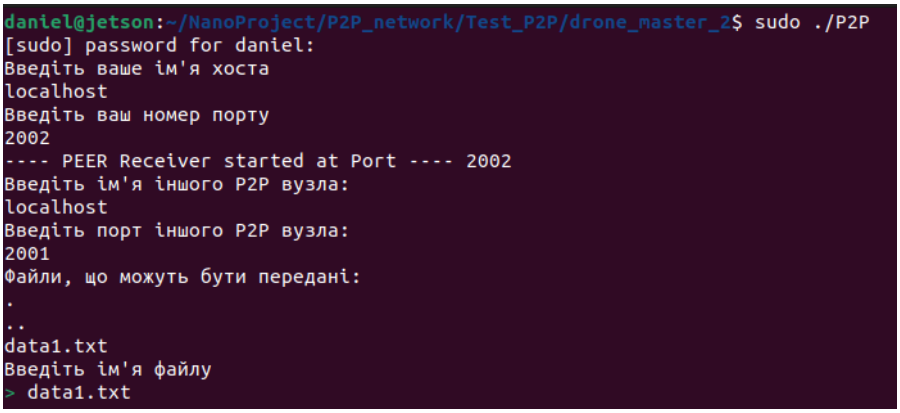


Рисунок 4.21 – Ініціалізація другого користувача P2P-мережі та підключення до першого

Як видно з рис. 4.21, при установленні з’єднання, застосунок першого користувача передав список файлів, доступних для завантаження, серед яких data1.txt. Після введення імені файлу другим користувачем, файл data1.txt було зчитано з сховища першого користувача та записано у сховище другого (рис. 4.22). При цьому, у першому застосунку було виведено повідомлення про завантаження файлу з локального сховища (рис. 4.23).

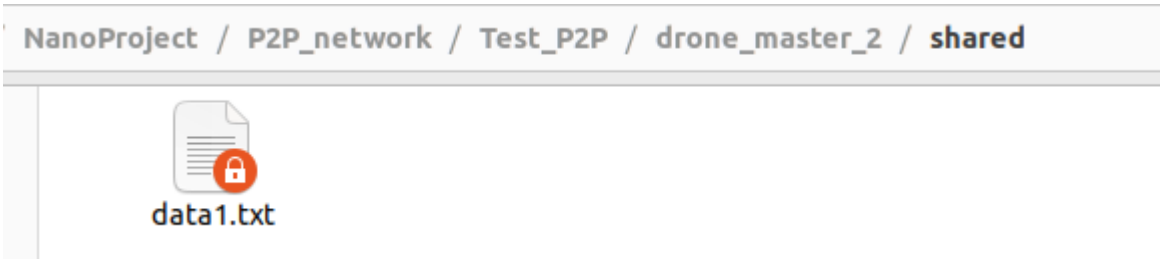


Рисунок 4.22 – Поява завантаженого файлу у сховищі другого користувача

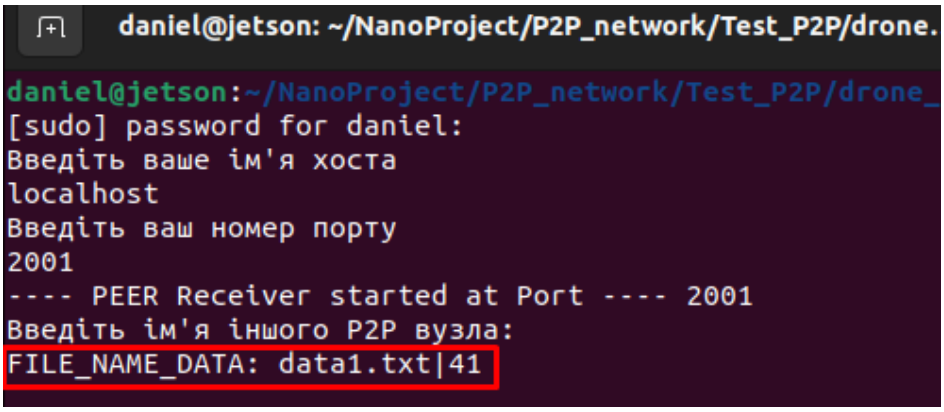


Рисунок 4.23 – Повідомлення про завантажений зі сховища файл

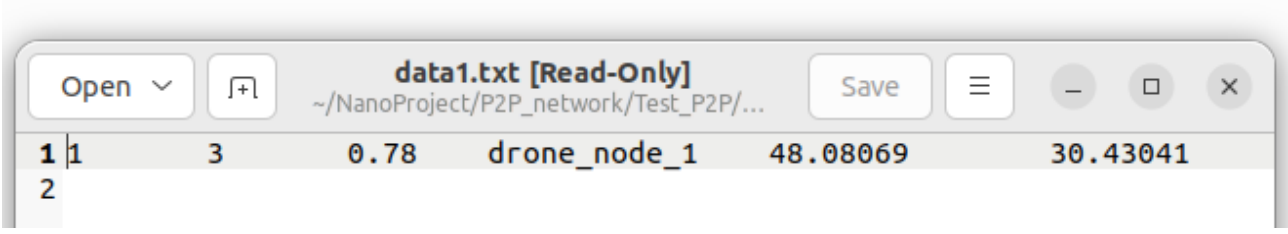


Рисунок 4.24 – Отриманий другим користувачем файл data1.txt

Як видно з рис. 4.24, файл з даними data1.txt був успішно переданий за допомогою застосунку обміну даними у P2P-мережі.

Висновки до розділу 4

У даному розділі було виконано тестування та перевірку працездатності ключових частин розподіленої системи: нейронної мережі, програми детектування об’єктів, приватної мережі LoRaWAN та P2P-мережі.

Тестування тренування нейронної мережі YOLOv11 було виконано за допомогою графічного процесора NVIDIA GTX 1650 та зайняло 5 годин 14 хвилин 46 секунд. Далі нейронну мережу було використано у програмі детектування. Загальний час роботи нейронної мережі на детектування об’єкту на кадрі становить 22 мс, що є гарним результатом для системи, що працює у реальному часі. Після детектування об’єкту, інформація про нього зберігається у файл для подальшої передачі.

Для тестування приватної мережі LoRaWAN було вибрано LWN Simulator. Було створено віртуальні вузли та шлюзи, які були підключені до мережевого серверу Chirpstack. Було проведено симуляцію роботи мережі LoRaWAN розподіленої системи, а саме: було здійснено передачу даних про детектований об’єкт від вузла до серверу. Наступним кроком було виконано тестування P2P-мережі на прикладі передачі даних (у виді файлів) між двома користувачами. У ході тестувань LoRaWAN та P2P мереж було доведено їх працездатність.

ВИСНОВКИ

Метою даної магістерської кваліфікаційної роботи була реалізація розподіленої системи детектування рухомих об'єктів та обміну інформацією на базі технології LoRaWAN.

Першим кроком було проведено аналіз предметної сфери, а саме: технологій детектування рухомих об’єктів за допомогою нейронних мереж, використання безпілотних апаратів як носіїв нейронних мереж, об’єднання дронів у рої та використання LoRaWAN для передачі даних. Було проаналізовано наукову літературу та технічні документації відповідних темі технологій. Здобута при цьому інформацію була використана при формуванні вимог до розподіленої системи. Хід та результати даного етапу виконання магістерської кваліфікаційної роботи було викладено у першому розділі.

Наступним кроком було виконано проєктування розподіленої системи. Під час розробки концептуального рішення було запропоновано дворівневу мережеву архітектуру (рівень рою дронів – P2P-мережа, рівень груп дронів – приватна мережа LoRaWAN). Для забезпечення елементів архітектури LoRaWAN пристрої були поділені на дрони-вузли та керуючі дрони. В якості моделі нейронної мережі було вирішено використовувати модель YOLOv11. Далі, було вибрано апаратне забезпечення для пристроїв розподіленої системи та наведено схеми його підключення. Дані кроки були детально описані у другому розділі магістерської кваліфікаційної роботи.

У третьому розділі був описаний етап розробки апаратно-програмного комплексу, який включає в себе нейронну мережу, програму детектування, програмне забезпечення P2P-мережі та мережі LoRaWAN, а також базовий графічний інтерфейс клієнтського застосунку. При розробці нейронної мережі та програми детектування використовувались фреймворки та бібліотеки мови програмування Python. Для розробки та налаштування мережевої частини використовувались мова програмування C++ та програмне забезпечення

Chirpstack для побудови LoRaWAN мережі. Для розробки користувацького інтерфейсу використовувався фреймворк Qt.

У четвертому розділі було виконано тестування та перевірку працездатності ключових частин розподіленої системи: нейронної мережі, програми детектування об’єктів, приватної мережі LoRaWAN та P2P-мережі. Результати тестування нейронної мережі підтвердили її швидкодію та пристосованість для роботи у реальному часі. А в ході тестувань мереж LoRaWAN та P2P було доведено їх правильна налаштованість та працездатність.

Дослідження і розробка розподіленої системи детектування рухомих об'єктів та обміну інформацією на базі технології LoRaWAN має велике практичне значення. Запропонована розподілена система поєднує потужні можливості штучного інтелекту з такими сильними сторонами технології LoRaWAN, як велика дальність сигналу, висока завадостійкість та низьке енергоспоживання. Таке поєднання зробить можливим полегшення та збільшення ефективності виконання задач у сферах, де потрібен автоматичний візуальний аналіз на великих площах та відстанях, а саме: у аграрній сфері, сфері охорони природи, під час пошуково-рятувальних операціях, наукових дослідженнях та інших видах моніторингу.

Розроблена розподілена система зберігає потенціал для подальшого удосконалення. Наступними кроками підвищення ефективності роботи розподіленої системи можуть бути: автоматизація налаштування сервера Chirpstack за допомогою використання його API, інтеграція зі станцією керування безпілотними апаратами (наприклад, QGroundControl), переведення певних Python-скриптів на мову програмування C++ для досягнення більшої швидкодії і т.д.

Дана кваліфікаційна робота була апробована на XXI Міжнародній науковій конференції «Ольвійський форум – 2024: стратегії країн Причорноморського регіону в геополітичному просторі». Матеріали доповіді були опубліковані у збірнику конференції [1].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Басов Д. Є., Пузирьов С. В. Розподілена система детектування рухомих об’єктів та обміну інформацією на базі LoRaWAN. *Ольвійський форум-2024: стратегії країн причорноморського регіону в геополітичному просторі*. Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2024. С. 138–141.

2. Yazdi M., Bouwmans T. New trends on moving object detection in video images captured by a moving camera: A survey. 2018. *Computer Science Review*. Vol. 28. P.157–177. DOI: 10.48550/arXiv.2404.12389.

3. Xie J., Yang C., Xie W., Zisserman A. Moving Object Segmentation: All You Need Is SAM (and Flow). 2024. *ArXiv*. Vol. abs/2404.12389. DOI: 10.48550/arXiv.2404.12389.

4. Chapel M., Bouwmans T. Moving Objects Detection with a Moving Camera: A Comprehensive Review. 2020. *ArXiv*. Vol. abs/2001.05238. DOI: 10.1016/j.cosrev.2020.100310.

5. Benali Amjoud A., Amrouch M. Object Detection Using Deep Learning, CNNs and Vision Transformers: A Review. 2023. *IEEE Access*. Vol. 11. DOI: 10.1109/ACCESS.2023.3266093.

6. Li Y., Miao N., Ma L., Feng S., Huang, X. Transformer for object detection: Review and benchmark. 2023. *Engineering applications of artificial intelligence*. Vol. 126. DOI: 10.1016/j.engappai.2023.107021.

7. Telli K., Kraa O., Himeur Y., Ouamane A., Boumehraz M., Atalla S., Mansoor W. A Comprehensive Review of Recent Research Trends on Unmanned Aerial Vehicles (UAVs). 2023. *Systems*. Vol. 11. DOI: 10.3390/systems11080400.

8. Pieczyński D., Ptak B., Kraft M., Piechocki M., Aszkowski, P. A fast, lightweight deep learning vision pipeline for autonomous UAV landing support with added robustness. 2024. *Engineering applications of artificial intelligence*. Vol. 131. DOI: 10.1016/j.engappai.2024.107864.

9. Skydio 2 Plus Enterprise drone. *Skydio*. URL: https://www.skydio.com/skydio-2-plus-enterprise (Last accessed: 05.12.2024).

10. Sadhu V., Anjum K., Pompili, D. On-Board Deep-Learning-Based Unmanned Aerial Vehicle Fault Cause Detection and Classification via FPGAs. 2023. *IEEE Transactions on Robotics*. Vol. 39. P. 3319–3331. DOI: 10.1109/TRO.2023.3269380.

11. Science & Tech Spotlight: Drone Swarm Technologies. *US: GAO*. URL: https://www.gao.gov/products/gao-23-106930 (Last accessed: 05.12.2024).

12. Chen R., Li J., Peng T. Decentralized UAV Swarm Scheduling with Constrained Task Exploration Balance. *Drones*. 2023. Vol. 7. No. 4:267. DOI: 10.3390/drones7040267.

13. Jamshidpey A., Wahby M., Heinrich M.K., Allwright M., Zhu W., Dorigo M. Centralization vs. decentralization in multi-robot coverage: Ground robots under UAV supervision. 2024. *ArXiv*. Vol. abs/2408.06553. DOI: 10.48550/arXiv.2408.06553.

14. Athena AI announcement for Teal 2. *TealDrones*. URL: https://tealdrones.com/press_releases/red-cat-and-athena-ai-announce-breakthrough-artificial-intelligence-and-computer-vision-capabilities-for-teal-2-military-grade-drone/ (Last accessed: 05.12.2024).

15. Ding Y., Yang Z., Pham V.Q., Zhang Z., Shikh-Bahaei M.R. Distributed Machine Learning for UAV Swarms: Computing, Sensing, and Semantics. 2023. *IEEE Internet of Things Journal*. Vol. 11. P. 7447–7473. DOI: 10.1109/JIOT.2023.3341307.

16. LoRa® and LoRaWAN®: A Technical Overview. *Semtech LoRa*. URL: https://www.katykoenen.com/wp-content/uploads/2024/07/LoRa_and_LoRaWAN-A_Tech_Overview-Downloadable.pdf (Last accessed: 05.12.2024).

17. LoRa documentation. *LoRa*. URL: https://lora.readthedocs.io/en/latest/#modulation-types-and-chirp-spread-spectrum (Last accessed: 05.12.2024).

18. Semtech FAQ. *Semtech*. URL: https://www.semtech.com/design-support/faq/P120 (Last accessed: 05.12.2024).

19. LoRaWAN Architecture. *The Things Network*. URL: https://www.thethingsnetwork.org/docs/lorawan/architecture (Last accessed: 05.12.2024).

20. Exploring protocols: LoRaWAN. *TagoIO*. URL: https://tago.io/blog/lorawan-benefits-applications (Last accessed: 05.12.2024).

21. De Carvalho Silva, J., Rodrigues, J., Alberti, A.M., Šolić, P., & Aquino, A.L. LoRaWAN — A low power WAN protocol for Internet of Things: A review and opportunities. 2017. *2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*. URL: https://api.semanticscholar.org/CorpusID:11505956 (Last accessed: 05.12.2024).

22. UG65 Semi-Industrial LoRaWAN® Gateway. *Milesight*. URL: https://www.milesight.com/iot/product/lorawan-gateway/ug65 (Last accessed: 05.12.2024).

23. LG5100 Industrial LoRaWAN Gateway. *Robustel*. URL: https://www.robustel.com/product/lg5100-industrial-edge-computing-lorawan-gateway/ (Last accessed: 05.12.2024).

24. Chirpstack v4.0.0 Changelog. *Chirpstack*. URL: https://www.chirpstack.io/docs/chirpstack/changelog.html (Last accessed: 05.12.2024).

25. Raspberry Pi 4 Model B specifications. *Raspberry Pi*. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/ (Last accessed: 05.12.2024).

26. Jetson Nano Developer Kit. *NVIDIA Developer*. URL: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (Last accessed: 05.12.2024).

27. Raspberry Pi 4 Model B 4GB. *Arduino в Україні*. URL: https://arduino.ua/prod3308-raspberry-pi-4-4gb (дата звернення: 05.12.2024).

28. Jetson Nano. *Arduino в Україні*. URL: https://arduino.ua/prod3564-nvidia-jetson-nano-developer-kit (дата звернення: 05.12.2024).

29. IMX477-160 12.3MP Camera. *Waveshare*. URL: https://www.waveshare.com/imx477-160-12.3mp-camera.htm (Last accessed: 05.12.2024).

30. Камера IMX477-160. *Arduino в Україні*. URL: https://arduino.ua/prod4863-imx477-160-12-3mp-camera-160-fov-applicable-for-jetson-nano-compute-module (дата звернення: 05.12.2024).

31. RP002-1.0.2 LoRaWAN® Regional Parameters. *LoRa Alliance*. URL: https://lora-alliance.org/wp-content/uploads/2020/11/RP_2-1.0.2.pdf (Last accessed: 05.12.2024).

32. ПЕРЕЛІК технічних характеристик та умов експлуатації радіообладнання, випромінювальних пристроїв, експлуатація яких здійснюється за принципом загальної авторизації: затв. постановою НКЕК від 03.07.2024 № 361. URL: https://zakon.rada.gov.ua/rada/show/z1175-24 (дата звернення 05.12.2024)

33. Wio-E5 mini. *SeeedStudio* URL: https://www.seeedstudio.com/LoRa-E5-mini-STM32WLE5JC-p-4869.html (Last accessed: 05.12.2024).

34. Grove - GPS (Air530). GPS location, Long-distance communication. *SeeedStudio*. URL: https://www.seeedstudio.com/Grove-GPS-Air530-p-4584.html (Last accessed: 05.12.2024).

35. RAK2287 Gateway Concentrator Module for LoRaWAN. *RAK*. URL: https://store.rakwireless.com/products/wislink-concentrator-module-sx1302-rak2287-lorawan?variant=39662118797510 (Last accessed: 05.12.2024).

36. Zhu P., Wen L., Du D., Bian X., Fan H., Hu Q., Ling H. Detection and Tracking Meet Drones Challenge. 2020. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. DOI: 10.1109/TPAMI.2021.3119563

37. Відео “aerial view on car driving through autumn forest road scenic a” користувача “Sophie Hodson”. *YouTube*. URL: https://youtu.be/5XVwL3cZhTo (Last accessed: 05.12.2024).

ДОДАТОК А

Код програмного забезпечення розподіленої системи

Лістинг 1. visdrone2019-vid.yaml

```
path: ../dataset # dataset root dir
train: VisDrone2019-VID-train/images # train images (relative to 'path') 6471 images
val: VisDrone2019-VID-val/images # val images (relative to 'path') 548 images
test: VisDrone2019-VID-test-dev/images # test images (optional) 1610 images

# Класи об'єктів
names:
  0: pedestrian
  1: people
  2: bicycle
  3: car
  4: van
  5: truck
  6: tricycle
  7: awning-tricycle
  8: bus
  9: motor
```

Лістинг 2. vis2yolo.py

```
import os
import sys


# Конвертує анотації датасету VisDrone у формат YOLO
def convert(dir):
    from PIL import Image
    from tqdm import tqdm

    # Конвертація обмежувальних рамок
    def convert_box(size, box):
        dw = 1. / size[0]
        dh = 1. / size[1]
        return (box[0] + box[2] / 2) * dw, (box[1] + box[3] / 2) * dh, box[2] * dw, box[3]
* dh

    (dir + '/labels').mkdir(parents=True, exist_ok=True)
    pbar = tqdm((dir + '/annotations').glob('*.txt'), desc=f'Converting {dir}')
    for f in pbar:
        img_size = Image.open((dir + '/images' + '/' + f.name).with_suffix('.jpg')).size
        lines = []
        with open(f, 'r') as file:
            for row in [x.split(',') for x in file.read().strip().splitlines()]:
                if row[4] == '0':
                    continue
                cls = int(row[5]) - 1
                box = convert_box(img_size, tuple(map(int, row[:4])))
                lines.append(f"{cls} {' '.join(f'{x:.6f}' for x in box)}\n")
                with                open(str(f).replace(f'{os.sep}annotations{os.sep}',
f'{os.sep}labels{os.sep}'), 'w') as fl:
                    fl.writelines(lines)
```

```
def convert_all():
    convert('dataset\VisDrone2019-VID-train')
    convert('dataset\VisDrone2019-VID-test-dev')
    convert('dataset\VisDrone2019-VID-val')

if __name__ == '__main__':
    # Застосування конвертації до всіх датасетів VisDrone2019-VID
    if len(sys.argv) < 2:
        convert_all()

    # Застосування конвертації до певного датасету VisDrone2019-VID
    elif sys.argv[1] == "train":
        convert('dataset\VisDrone2019-VID-train')
    elif sys.argv[1] == "test-dev":
        convert('dataset\VisDrone2019-VID-test-dev')
    elif sys.argv[1] == "val":
        convert('dataset\VisDrone2019-VID-val')
    else:
        print('Available options (dataset names): "train", "test-dev", "val"')
```

Лістинг 3. camera_detection.py

```
from ultralytics import YOLO
import cv2
import project_utils as pu
import sys

# Отримати ID дрону (1-й аргумент при запуску програми)
drone_id = sys.argv[1]

# Використати натреновану модель з найкращими значеннями параметрів
model = YOLO("runs/detect/train/weights/best.pt")

# Ініціалізація віднімання фону OpenCV
# MOG2 – на основі моделі гаусівського змішування
back_sub     =     cv2.createBackgroundSubtractorMOG2(history=500,     varThreshold=100,
detectShadows=True)

# Вибір джерела відео
# vcap = cv2.VideoCapture(0) #Запис відеопотоку з камери
# vcap.set(3, 640)
# vcap.set(4, 480)
vcap  =  cv2.VideoCapture("dataset/test_video.mp4")   #  Відеофайл  замість  камери  –  для
тестування

# Основний цикл детектування
while True:
    # Зчитування кадрів з відеопотоку
    success, frame = vcap.read()
    # Зупинити зчитування, якщо відопотік закінчився
    if not success:
        break
    # Вивести вікно з відео
    # cv2.imshow('Distributed system', frame)
    # Застосувати віднімання фону до кадру відеопотоку
    # Застосувати віднімання фону до кадру відеопотоку
    mask = back_sub.apply(frame)
    # Застосувати нейромережу до кадру відеопотоку
    results = model.track(frame, persist=True, verbose=False)
```

```
    # Фільтрація рухомих об'єктів
    moving_objects = []
    for box in results[0].boxes:
        obj_id = box.id  # Унікальний ID об'єкта
        x1, y1, x2, y2 = map(int, box.xyxy[0])  # Координати обмежувальної рамки
        conf = box.conf  # Точність детектування об'єкта
        cls = box.cls  # Клас об'єкта
        # Перевірка на рух
        if mask[y1:y2, x1:x2].mean() > 25:
            moving_objects.append((obj_id, x1, y1, x2, y2, conf, cls))
            # Виведення повідомлення про детектування об'єкта
            pu.print_result(pu.format_result(box), pu.classNames)
            # Створення обмежувальної рамки з об'єктом
            annotated_frame = results[0].plot()
            # Збереження кадру з об'єктом
            cv2.imwrite(f"{int(obj_id)}_{int(cls[0])}_%.2f.jpg"      %      float(conf),
annotated_frame)
            # Збереження даних про о'єкт
            pu.write_result(pu.format_result(box), drone_id)
            # Вивести вікно з відео
            # cv2.imshow('Distributed system', annotated_frame)

    # print(moving_objects)
    if cv2.waitKey(1) == ord('q'):
        break

vcap.release()
cv2.destroyAllWindows()
```

Лістинг 4. project_utils.py

```
import grove_gps as gps
import sys
import base64


# Класи об'єктів датасета VisDrone-VID
classNames = {
    0: "pedestrian",
    1: "people",
    2: "bicycle",
    3: "car",
    4: "van",
    5: "truck",
    6: "tricycle",
    7: "awning - tricycle",
    8: "bus",
    9: "motor",
}

# Перетворити інформацію про детектований об'єкт
# У масив типу [id, клас об'єкта, точність детектування]
def format_result(obj_box):
    obj_id = int(obj_box.id)
    obj_cls = int(obj_box.cls[0])
    obj_conf = float(obj_box.conf)
    formatted_result = [obj_id, obj_cls, obj_conf]
    return formatted_result
```

```
# Показати результат детектування об'єкта у консолі
def print_result(f_res, cls_names):
    print("\nОБ'ЄКТ ЗНАЙДЕНО!")
    print("ID: " + str(f_res[0]))
    print("Клас: " + cls_names[f_res[1]])
    print("Точність: " + str(f_res[2]))


# Зберегти дані про детектуваний об'єкт та координати дрона у txt-файл
# Формат даних у файлі: ID\tклас\tточність\tширота\tдовгота
def write_result(f_res, drone_id):
    obj = f_res
    drone_gps = gps.get_coordinates()
    with open(f"{int(obj[0])}_{int(obj[1])}_%.2f.txt" % float(obj[2]), 'w') as output:

print(f"{int(obj[0])}\t{int(obj[1])}\t%.2f\t{drone_id}\t{drone_gps[0]}\t{drone_gps[1]}"
% float(obj[2]), file=output)

def base64_to_ASCII(base64_message):
    msg_hex = base64.b64decode(base64_message).hex()
    msg_ascii = bytearray.fromhex(msg_hex).decode()
    print("Received base64-encoded message: " + str(base64_message))
    print("Original message (in ASCII): " + str(msg_ascii))

if __name__ == '__main__':
    if sys.argv[1] == "base64_to_ASCII":
        base64_to_ASCII(sys.argv[2])
```

Лістинг 5. P2Pmain.cpp

```
#include "PeerReceiver.h"
#include "PeerSender.h"
#include <thread>
#include <unistd.h>
#include <memory>
#include <exception>
using namespace std;


// Потік отримання даних
void StartThePeerReceiver(string PeerReceiverName,string PeerReceiverPort){
      std::unique_ptr<PeerReceiver>                                          rec(new
PeerReceiver(PeerReceiverName,PeerReceiverPort));
      rec->StartReceiver();
}

// Потік відправлення даних
void StartThePeerSender(){
      sleep(1);
      std::unique_ptr<PeerSender> send(new PeerSender());
      //PeerSender* send = new PeerSender();
      send->RegisterPeer();
      send->FileDownload();
}


int main(){
      string PeerReceiverName ="";
      string PeerReceiverPort ="";
      cout << "Введіть ваше ім'я хоста \n";
```

```
        cin >> PeerReceiverName;
        cout << "Введіть ваш номер порту \n";
        cin >> PeerReceiverPort;

        thread PeerReceiver(StartThePeerReceiver,PeerReceiverName,PeerReceiverPort);
        thread StartTheSender(StartThePeerSender);
        PeerReceiver.join();
        StartTheSender.join();
        return 0;
}
```

Лістинг 6. lora_wioE5.py

```
import sys
import serial

# Налаштування вузла з Wio-E5 mini
def set_node():
    # Створення порта для Wio-E5 mini (USB-підключення)
    # Через обмеження Jetson Nano, необхідно використовувати baudrate = X * 1.02
    port  =  serial.Serial(port='/dev/ttyUSB0',  baudrate=9600  *  1.02,  bytesize=8,
parity='N', stopbits=1, timeout=1,
                           xonoff=False, rtscts=False, dsrdtr=False)

    # Перевірка підключення
    test_cmd = "AT\r"
    port.write(test_cmd.encode())
    test_msg = port.read(64)
    print(test_msg)  # +AT: OK

    # Отримання ID модуля Wio-E5 mini
    id_cmd = "AT+ID\r"
    port.write(id_cmd.encode())
    module_id = port.read(64)
    print(module_id)

    # Налаштування частоти EU868 МГц
    dr_cmd = "AT+DR=EU868\r"
    port.write(dr_cmd.encode())
    dr = port.read(64)
    print(dr)

# Відправка повідомлення
# (за умов налаштованого вузла та наявності підключення до Chirpstack)
def send_message(msg):
    port  =  serial.Serial(port='/dev/ttyUSB0',  baudrate=9600  *  1.02,  bytesize=8,
parity='N', stopbits=1, timeout=1,
                           xonoff=False, rtscts=False, dsrdtr=False)
    dr_msg = 'AT+MSG="'+str(msg)+'"'
    port.write(dr_msg)


if __name__ == '__main__':
    if sys.argv[1] == "init":
        set_node()
    elif sys.argv[1] == "msg":
        send_message(sys.argv[2])
    else:
        print('Use "init" or "msg" + message as arguments"')
```

Лістинг 7. grove_gps.py

```
# Grove GPS (Air530) підключений до пінів UART: 8 - RX; 10 - TX
import serial
import time
import gps_api  # Бібліотека для роботи з Grove GPS (Air530)

# Створення порта для UART-підключення
def initialize_port():
    serial_port = serial.Serial(
        port="/dev/ttyTHS1",
        baudrate=115200,
        bytesize=serial.EIGHTBITS,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
    )
    time.sleep(1) # Час на ініціалізацію порта
    return serial_port

# Отримання координат у форматі (широта, довгота)
def get_coordinates():
    port = initialize_port()  # Використання створеного UART-порта
    gps = gps_api.GPS(port)  # Створення об'єкта класу GPS
    lat = gps.get_latitude()  # Широта
    lon = gps.get_longitude()  # Довгота
    coordinates = (lat, lon)  # Кортеж з координатами
    # coordinates = (48.08069, 30.43041) # для тестування
    return coordinates

# Закриття порта для UART-підключення
def close_port(serial_port):
    serial_port.close()
```

Лістинг 8. chirpstack.toml

```
# Logging.
[logging]
  # Options are: trace, debug, info, warn error.
  level="info"

# PostgreSQL configuration.
[postgresql]
  # PostgreSQL DSN.

dsn="postgres://chirpstack:<POSTGRESQL_CHIRPSTACK_PASSWORD>@$POSTGRESQL_HOST/chirpstack
?sslmode=disable"
  # Connection Options
  max_open_connections=10
  min_idle_connections=0

# Redis configuration.
[redis]
  # Server address or addresses.
  servers=["redis://$REDIS_HOST/"]
  # TLS enabled.
  tls_enabled=false
  # Redis Cluster.
  cluster=false
```

```
# Network related configuration.
[network]
  # Network identifier (NetID, 3 bytes) encoded as HEX (e.g. 010203).
  net_id="000000"
  # Enabled regions.
  enabled_regions=["eu868",]

# API interface configuration.
[api]
  # interface:port to bind the API interface to.
  bind="0.0.0.0:7070"

  # API Secret. (Generate with openssl rand -base64 32)
  secret="<API_SECRET>"

[integration]
  enabled=["mqtt", "postgresql"]

  [integration.mqtt]
    server="tcp://$MQTT_BROKER_HOST:7071/"
    json=true

  [integration.postgresql]

dsn="postgres://chirpstack_integration:<POSTGRES_CHIRPSTACK_INTEGRATION_PASSWORD>@$POST
GRESQL_HOST/chirpstack_integration?sslmode=disable"
```

Лістинг 9. chirpstack-gateway-bridge.toml

```
[integration.mqtt.auth.generic]
servers=["tcp://<NETWORK_SERVER_HOST>:7071"]

[integration.mqtt]
event_topic_template="eu868/gateway/{{ .GatewayID }}/event/{{ .EventType }}"
state_topic_template="eu868/gateway/{{ .GatewayID }}/state/{{ .StateType }}"
command_topic_template="eu868/gateway/{{ .GatewayID }}/command/#"

[backend]
type="basic_station"

  [backend.basic_station]
  bind=":3001"
  tls_cert=""
  tls_key=""
  ca_cert=""

  region="EU868"
  frequency_min=863000000
  frequency_max=870000000


  [[backend.basic_station.concentrators]]

    [backend.basic_station.concentrators.multi_sf]
    frequencies=[
      868100000,
      868300000,
      868500000,
      867100000,
```

```
      867300000,
      867500000,
      867700000,
      867900000,
    ]

    [backend.basic_station.concentrators.lora_std]
    frequency=868300000
    bandwidth=250000
    spreading_factor=7

    [backend.basic_station.concentrators.fsk]
    frequency=868800000
```

Лістинг 10. main.qml

```
import QtQuick
import QtQuick.Window
import QtQuick.Controls
import QtLocation
import QtPositioning

Window {
    width: 800
    height: 600
    visible: true
    title: "Distributed system Client App"

    Plugin {
        id: mapPlugin
        name: "osm"
    }

    Map {
        id: map
        anchors.fill: parent
        plugin: mapPlugin
        center: QtPositioning.coordinate(48.08361, 30.44874)
        zoomLevel: 14
        property geoCoordinate startCentroid

        PinchHandler {
            id: pinch
            target: null
            onActiveChanged: if (active) {
                map.startCentroid = map.toCoordinate(pinch.centroid.position, false)
            }
            onScaleChanged: (delta) => {
                map.zoomLevel += Math.log2(delta)
                map.alignCoordinateToPoint(map.startCentroid, pinch.centroid.position)
            }
            onRotationChanged: (delta) => {
                map.bearing -= delta
                map.alignCoordinateToPoint(map.startCentroid, pinch.centroid.position)
            }
            grabPermissions: PointerHandler.TakeOverForbidden
        }
        WheelHandler {
            id: wheel
            // workaround for QTBUG-87646 / QTBUG-112394 / QTBUG-112432:
```

```
            // Magic Mouse pretends to be a trackpad but doesn't work with PinchHandler
            // and we don't yet distinguish mice and trackpads on Wayland either
            acceptedDevices:      Qt.platform.pluginName      ===      "cocoa"      ||
Qt.platform.pluginName === "wayland"
                                ? PointerDevice.Mouse | PointerDevice.TouchPad
                                : PointerDevice.Mouse
            rotationScale: 1/120
            property: "zoomLevel"
        }
        DragHandler {
            id: drag
            target: null
            onTranslationChanged: (delta) => map.pan(-delta.x, -delta.y)
        }
        Shortcut {
            enabled: map.zoomLevel < map.maximumZoomLevel
            sequence: StandardKey.ZoomIn
            onActivated: map.zoomLevel = Math.round(map.zoomLevel + 1)
        }
        Shortcut {
            enabled: map.zoomLevel > map.minimumZoomLevel
            sequence: StandardKey.ZoomOut
            onActivated: map.zoomLevel = Math.round(map.zoomLevel - 1)
        }

        MapQuickItem {
            id: drone_node_1
            anchorPoint.x: image.width/4
            anchorPoint.y: image.height
            coordinate : QtPositioning.coordinate(48.08361, 30.44874)
            HoverHandler {
                id: hoverHandler1
            }
            TapHandler {
                id: tapHandler1
                acceptedButtons: Qt.LeftButton
                gesturePolicy: TapHandler.WithinBounds
                onTapped: {
                    popup.open()
                }
            }
            DragHandler {
                id: dragHandler1
                grabPermissions:          PointerHandler.CanTakeOverFromItems          |
PointerHandler.CanTakeOverFromHandlersOfDifferentType
            }

            sourceItem: Image {
                id: image
                source : "qrc:/images/Slave_drone.png"
            }
            Popup {
                  id: popup
                  x: 80
                  y: -20
                  width: 160
                  height: 120
                  closePolicy: Popup.CloseOnEscape | Popup.CloseOnPressOutsideParent
                  padding: 10
```

```
                Label {
                  text: "ID: drone_node_1"
                }
                Label {
                  y : 20
                  text: "GPS: 48.08361, 30.44874"
                }
                Label {
                  y : 50

                  text: "ОБ'ЄКТ ДЕТЕКТОВАНО!"
                  color : "red"
                }
                Label {
                  y : 70

                  text: "Клас об'єкта: car"
                  color : "red"
                }
                Label {
                  y : 90

                  text: "Точність: 0.84"
                  color : "red"
                }
              }
        }
        MapQuickItem {
            id: drone_node_2
            anchorPoint.x: image.width/4
            anchorPoint.y: image.height
            coordinate : QtPositioning.coordinate(48.08241, 30.42874)
            HoverHandler {
                id: hoverHandler2
            }
            TapHandler {
                id: tapHandler2
                acceptedButtons: Qt.LeftButton
                gesturePolicy: TapHandler.WithinBounds
                onTapped: {
                    popup2.open()
                }
            }
            DragHandler {
                id: dragHandler2
                grabPermissions:          PointerHandler.CanTakeOverFromItems          |
PointerHandler.CanTakeOverFromHandlersOfDifferentType
            }

            sourceItem: Image {
                id: image2
                source : "qrc:/images/Slave_drone.png"
            }
            Popup {
                  id: popup2
                  x: 80
                  y: -20
                  width: 160
                  height: 80
                  closePolicy: Popup.CloseOnEscape | Popup.CloseOnPressOutsideParent
```

```
                padding: 10

                Label {
                  text: "ID: drone_node_2"
                }
                Label {
                  y : 20
                  text: "GPS: "
                }

            }
        }
        MapQuickItem {
            id: drone_master_1
            anchorPoint.x: image.width/4
            anchorPoint.y: image.height
            coordinate : QtPositioning.coordinate(48.06343, 30.46866)
            HoverHandler {
                id: hoverHandler3
            }
            TapHandler {
                id: tapHandler3
                acceptedButtons: Qt.LeftButton
                gesturePolicy: TapHandler.WithinBounds
                onTapped: {
                    popup3.open()
                }
            }
            DragHandler {
                id: dragHandler3
                grabPermissions:          PointerHandler.CanTakeOverFromItems          |
PointerHandler.CanTakeOverFromHandlersOfDifferentType
            }

            sourceItem: Image {
                id: image3
                source : "qrc:/images/Master_drone.png"
            }
            Popup {
                  id: popup3
                  x: 80
                  y: -20
                  width: 160
                  height: 80
                  closePolicy: Popup.NoAutoClose
                  padding: 10

                  Label {
                    text: "ID: drone_master_1"
                  }
                  Label {
                    y : 20
                    text: "GPS: 48.06343, 30.46866"
                  }
            }
        }

    }

}
```

ДОДАТОК Б

Апробація кваліфікаційної магістерської роботи

Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
Національна академія наук України
Південний науковий центр НАН і МОН України
Інститут української археографії та джерелознавства
ім. М.С. Грушевського НАН України
Державний архів Миколаївської області
ДУ «Національний науковий центр радіаційної медицини НАМН України»
Донецький національний медичний університет
Technical University of Moldova (Moldova)
Jan Dlugosz University in Czestochowa (Poland)
Adam Mickiewicz University (Poland)
Leipzig University of Applied Sciences (Germany)
Rzeszow University of Technology (Poland)
Ca` Foscari University (Italy)

ОЛЬВІЙСЬКИЙ ФОРУМ – 2024:
стратегії країн Причорноморського регіону
в геополітичному просторі

XXI Міжнародна наукова конференція

ТЕЗИ

ТЕХНІЧНІ НАУКИ ТА ІНЖЕНЕРІЯ

20–23 червня 2024 р., м. Миколаїв, Україна

Миколаїв – 2024

ПІДСЕКЦІЯ: Комп’ютерна інженерія

По горінню обох діодів на IR-сенсорі можна зробити висновок, що пристрій працює коректно. Це свідчить про те, що обидва канали передачі даних активні, що, в свою чергу, означає належне функціонування пристрою.

Подальшим розвитком запропонованого рішення слід вважати підвищення швидкості реакції системи на засинання водія та покращення естетики й дизайну компонентів системи.

Список використаних джерел

1. ARDUINONANO Datasheet (PDF) – List of unclassifed manufacturers. URL: https://html.alldatasheet.com/htmlpdf/1424859/ETC/ARDUINONANO/66/1/ARDUINONANO.html (Last accessed: 28.04.2024).

2. Wörle J., Metz B., Thiele C., Weller G. Physiological indicators for detecting a driver falling asleep during highly automated driving. *Driver Distraction and Inattention* : Proc. of the Conf., Oct. 2018, Gothenburg, Sweden.. URL: https://www.researchgate.net/publication/328359189_Physiological_indicators_for_detecting_a_driver_falling_asleep_during_highly_automated_driving (Last accessed: 28.04.2024).

3. Основи мікроелектроніки з Arduino. URL: https://www.slideshare.net/ssuser74332f/arduino-76923628 (дата звернення: 28.04.2024).

УДК 004.75

Басов Д. Є.,
магістрант кафедри комп'ютерної інженерії,
Пузирьов С. В.,
канд. фіз.-мат. наук, доцент кафедри комп'ютерної інженерії,
ЧНУ імені Петра Могили, м. Миколаїв, Україна

РОЗПОДІЛЕНА СИСТЕМА ДЕТЕКТУВАННЯ РУХОМИХ ОБ'ЄКТІВ ТА ОБМІНУ ІНФОРМАЦІЄЮ НА БАЗІ LORAWAN

Останні декілька років характеризуються все зростаючим інтересом до безпілотних апаратів (дронів). Їх використання вже революціонізувало сферу національної безпеки. Незабаром постане питання їх активного цивільного використання, що робить актуальним попереднє узагальнення зроблених напрацювань та подальше

удосконалення технологій та способів використання дронів.

Одним зі способів покращення ефективності безпілотних апаратів є їх об’єднання у так звані «рої». Рій безпілотних апаратів являє собою систему обміну інформацією між всіма дронами, яка дозволяє координувати їх роботу без прямого втручання людини. Рої дронів поділяються на централізовані та децентралізовані. Централізовані рої дронів використовують таку систему керування, де один дрон чи зовнішній комп’ютер відповідальний за координацію та планування дій. У випадку децентралізованого рою дронів кожна одиниця зв’язується з іншою та приймає рішення самостійно [1]. Запропонована розподілена система детектування рухомих об’єктів та обміну інформацією може бути використана для керування децентралізованим роєм дронів.

Іншим технологічним напрямком, що зараз стрімко розвивається, є штучний інтелект. Переваги «розумних» технологій, здатних навчатися за допомогою алгоритмів машинного навчання, сприяють їх широкій інтеграції у все нові сфери людського життя.

Штучний інтелект має значний вплив на галузь розробки безпілотних апаратів. Найчастіше штучний інтелект використовується для вирішення наступних задач: детектування та відслідковування об’єктів, автономна навігація, планування шляху, координація дронів у рою, аналіз відео та зображень, кібербезпека.

Детектування об’єктів – одна з найголовніших задач сфери комп’ютерного зору (англ. *computer vision*), для вирішення якої застосовуються штучні нейронні мережі. Для детектування об’єктів що найчастіше використовуються наступні архітектури штучних нейронних мереж: згорткова нейронна мережа (англ. *convolutional neural network, CNN*), рекурентна нейронна мережа (англ. *recurrent neural networks, RNN*), трансформер та їх гібриди [2]. Нейронна мережа для детектування об’єктів приймає на вхід відеопотік та видає на виході положення знайденого об’єкта. У запропонованій розподіленій системі детектування рухомих об’єктів та обміну інформацією як архітектура штучної нейронної мережі використовується гібрид згорткової нейронної мережі та трансформера.

Обробка відеоінформації нейронною мережею здійснюється локально, тобто апаратними засобами самого безпілотного апарату. Для забезпечення достатньої кількості апаратних ресурсів на запис, збереження та обробку відео нейронною мережею використовується одноплатний комп’ютер NVIDIA Jetson Nano.

Інформація про детектований об’єкт передається іншим дронам у системі. Для комунікації між дронами використовується технологія LoRaWAN (від англ. *Long Range Wide Area Network*).

LoRaWAN – це технологія радіочастотної модуляції для малопотужних (англ. *low-power*) глобальних мереж (LPWAN), що була розроблена компанією Semtech. LoRaWAN забезпечує зв’язок на великі відстані: до п’яти кілометрів у містах і до 15 кілометрів або більше в сільській місцевості (в межах прямої видимості) [3].

Вибір технології LoRaWAN для вирішення проблем обміну інформацією зумовлений її перевагами над іншими технологіями бездротового зв’язку (рис. 1). Значна дії сигналу дозволяє координацію та комунікацію безпілотних апаратів на дуже великих відстанях. Низьке енергоспоживання збільшує тривалість польотів. А висока стійкість до завад, забезпечена технологією Chirp Spread Spectrum (CSS) [3] дозволяє безпілотним апаратам протидіяти сильному рівню шуму, створеному навмисно чи ні.

Рис.1. Порівняння LoRaWAN з іншими технологіями бездротового зв’язку

У цілому запропонована розподілена система детектування рухомих об’єктів та обміну інформацією складається з нейронної мережі для детектування об’єктів та підсистеми комунікації. Головними елементами апаратної частини є відеокамера для збирання інформації, одноплатний комп’ютер NVIDIA Jetson Nano для виконання розрахунків та збереження даних, та модуль зв’язку LoRaWAN для передачі даних.

Список використаних джерел

1. Chen R., Li J., Peng T. Decentralized UAV Swarm Scheduling with Constrained Task Exploration Balance. *Drones. 2023; 7(4):267.* DOI: https://doi.org/10.3390/drones7040267
2. Telli K., Kraa O., Himeur Y., Ouamane A., Boumehraz M., Atalla

S., Mansoor W. A Comprehensive Review of Recent Research Trends on Unmanned Aerial Vehicles (UAVs). *Systems 2023, 11, 400.* DOI: https://doi.org/10.3390/systems11080400

3. LoRa® and LoRaWAN®: A Technical Overview – Semtech LoRa. URL:https://loradevelopers.semtech.com/uploads/documents/files/LoRa_and_LoRaWAN-A_Tech_Overview-Downloadable.pdf (Last accessed: 30.04.2024).

УДК 004.67

Буряк М. Р.,
аспірант кафедри програмного забезпечення
комп'ютерних систем,
Чернівецький національний університем
ім. Ю. Федьковича, м. Чернівці, Україна

МЕТОДИ ТА ЗАСОБИ ОПТИМІЗАЦІЇ ВИКОРИСТАННЯ РЕСУРСІВ В KUBERNETES КЛАСТЕРІ

Технологія Kubernetes надає можливість гнучко управляти життєвим циклом контейнеризованих застосунків і виділяти необхідну кількість ресурсів для їх функціонування. Це можливо завдяки функціоналу, який реалізується в Kubernetes resources requests/limits, affinity/anti-affinity та labels, що дозволяє формувати запити і обмежувати необхідні ресурси. Компонент Kubernetes autoscaler забезпечує формування запиту до провайдера хмарних обчислень на виділення необхідної кількості вузлів з запланованою кількістю ресурсів, а Kubernetes scheduler є планувальником для розміщення застосунків на нових вузлах (рис. 1).

Разом із перевагами щодо динамічного виділення ресурсів, Kubernetes має недолік – неоптимізовані контейнеризовані застосунки можуть споживати більше ресурсів ніж необхідно, що призводить до деградації роботи застосунку, а також незапланованих витрат на інфраструктуру.

Для вирішення проблеми оптимізації ресурсів в Kubernetes дослідники найчастіше використовують наступні методи:

– прогнозування використання ресурсів;
– користувацькі планувальники Kubernetes із застосуванням різних алгоритмів планування;
– масштабування контейнеризованих застосунків.