ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем
(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття рівня вищої освіти «бакалавр»
(рівень вищої освіти)

Система управління ройовим комплексом. Розробка протоколів обміну даними та їх реалізація на основі мікросервісної архітектури.
(тема кваліфікаційної роботи українською мовою)

Swarm complex control system. Data exchange protocols development and their implementation based on microservice architecture
(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач ВО денної форми навчання
спеціальності 123 – Комп'ютерна інженерія .
(шифр і назва напряму підготовки, спеціальності)
Освітня програма «Комп'ютерна інженерія»
(назва освітньої програми)
Ісаєв Михайло Андрійович
(прізвище, ім'я, по-батькові)

Керівник к. ф-м. н., доц. Шпінарева І. М. _____
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент Розновець О.І.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№___ від «___»_____ 2025 р.
Завідувач кафедри
_____ _Євгеній МАЛАХОВ_
(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № ____
протокол № __ від «___» ____ 2025 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)
Голова ЕК
_____ _Світлана АНТОЩУК_
(підпис) (ім'я, прізвище)

Одеса – 2025

АНОТАЦІЯ

Метою даної кваліфікаційної роботи є розробка та реалізація протоколів обміну даними між дронами та дронами і мікросервісами у роєвому комплексі, що забезпечить гарантовану злагоджену роботу рою, оперативне реагування на зміни умов та динамічну адаптацію його складу.

У роботі використано апаратне забезпечення Raspberry Pi Zero 2 W з камерою OV5647, що забезпечує компактність, енергоефективність та достатню обчислювальну потужність для обробки даних у реальному часі. Програмне забезпечення базується на операційній системі Raspberry Pi OS, фреймворку GStreamer для передачі відеопотоків, технології Fast DDS для обміну телеметричними даними та IDL для серіалізації даних.

Результати роботи демонструють, що розроблена комунікаційна підсистема забезпечує стабільний та ефективний обмін даними між дронами, що є критичним для функціонування роєвих систем БПЛА. Система відповідає вимогам до надійності, швидкості та безпеки передачі даних, а також демонструє стійкість до перебоїв у зв'язку та динамічних змін у складі рою. Практична цінність роботи полягає у можливості застосування розробленої системи у різних сферах, таких як моніторинг великих територій, екологічні дослідження, точне землеробство та швидке реагування у військових і рятувальних операціях.

Подальші дослідження можуть бути спрямовані оптимізацію протоколів для підвищення ефективності, впровадження додаткових механізмів безпеки та розширення функціональності системи для підтримки більш складних сценаріїв використання.

Таким чином, розроблена у цій роботі комунікаційна підсистема є актуальним і перспективним рішенням для розвитку автономних технологій у сфері БПЛА, що має значне наукове і практичне значення.

ABSTRACT

The purpose of this qualification work is to develop and implement protocols for data exchange between drones and drones in a swarm complex, which will ensure guaranteed coordinated operation of the swarm, rapid response to changing conditions, and dynamic adaptation of its composition.

The work uses Raspberry Pi Zero 2 W hardware with an OV5647 camera, which provides compactness, energy efficiency, and sufficient computing power for real-time data processing. The software is based on the Raspberry Pi OS operating system, the GStreamer framework for video streaming, Fast DDS technology for telemetry data exchange, and IDL for data serialization.

The results demonstrate that the developed communication subsystem provides stable and efficient data exchange between drones, which is critical for the functioning of UAV swarm systems. The system meets the requirements for reliability, speed, and security of data transmission, and demonstrates resistance to communication interruptions and dynamic changes in the composition of the swarm.

The practical value of the work lies in the possibility of applying the developed system in various fields, such as monitoring large territories, environmental research, precision farming, and rapid response in military and rescue operations.

Further research may focus on protocol optimization for efficiency, implementation of additional security mechanisms, and expansion of system functionality to support more complex usage scenarios.

Thus, the communication subsystem developed in this work is a relevant and promising solution for the development of autonomous technologies in the field of UAVs, which has significant scientific and practical significance.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

CSI – це специфікація Альянсу мобільних промислових процесорі, яка визначає інтерфейс між камерою та головним процесором

I2C – послідовна шина даних для зв'язку інтегральних схем

IDL – Interface Definition Language – мова опису інтерфейсів

SSH – Secure Shell – мережевий протокол прикладного рівня, що дозволяє проводити віддалене управління комп'ютером і тунелювання TCP-з'єднань

БПЛА – безпілотні літальні апарати

НС – надзвичайні ситуації

Пайплайн – ланцюг послідовно з'єднаних елементів для обробки потокових даних

ВСТУП

У сучасному світі автономні безпілотні літальні апарати відіграють ключову роль у виконанні широкого спектра завдань, таких як моніторинг територій, пошуково-рятувальні операції, аграрні роботи, доставка вантажів та військові місії. Особливу увагу привертають ройові системи БПЛА, де скоординована взаємодія групи дронів забезпечує високу адаптивність і стійкість до відмов. Ройові технології дозволяють підвищити ефективність виконання складних завдань у динамічних і непередбачуваних умовах завдяки децентралізованому управлінню та розподілу функцій між агентами. Однак успішна робота таких систем залежить від ефективного обміну інформацією між дронами та наземними сервісами, що має відповідати високим вимогам до надійності, швидкості та безпеки передачі даних у реальному часі.

Розробка комунікаційних підсистем для ройових комплексів стикається з низкою викликів. Серед них: гетерогенність трафіку, що вимагає чіткої пріоритизації відеоданих і критичних команд управління; обмеження пропускної здатності бездротових каналів, характерних для технологій типу Wi-Fi; стійкість до перебоїв у зв’язку через зовнішні фактори, такі як погодні умови чи відстань; обмежені обчислювальні ресурси дронів (наприклад, ARM-архітектура на Raspberry Pi); необхідність точної синхронізації часу для обробки даних; а також потреба в децентралізованій взаємодії без центрального управління. Ці проблеми підкреслюють актуальність створення інноваційних рішень, які забезпечать стабільну та ефективну роботу ройових систем. Історично розвиток таких технологій бере початок із досліджень розподілених систем і штучного інтелекту, але сучасні вимоги до автономності та масштабованості потребують нових підходів до комунікаційних протоколів.

Практична цінність таких розробок очевидна: від моніторингу великих територій і екологічних досліджень до точного землеробства в аграрному секторі

та швидкого реагування у військових і рятувальних операціях. За даними дослідження [1], впровадження ройових систем може скоротити час виконання складних місій на 40–85% порівняно з поодинокими БПЛА, що робить цю тему перспективною для подальшого розвитку інформаційних технологій.

Метою роботи є розробка та реалізація протоколів обміну даними між дронами та дронами і мікросервісами у роєвому комплексі, що забезпечить гарантовану злагоджену роботу рою, оперативне реагування на зміни умов та динамічну адаптацію його складу [2].

Для досягнення поставленої мети необхідно виконати наступні завдання:

1) проаналізувати предметну область, зокрема особливості управління роєм дронів у НС;

2) обрати та обґрунтувати протоколи обміну повідомленнями між дронами та дронами і мікросервісами у рої дронів у НС;

3) визначити семантику протоколу обміну повідомленнями;

4) розробити схему безпеки з механізмами аутентифікації та доступу;

5) побудувати інтерфейси взаємодії з підтримкою асинхронного та синхронного обміну даними;

6) реалізувати компонент потокової передачі відео;

7) забезпечити синхронізацію часу між учасниками системи;

8) провести тестування.

Таким чином, створення масштабованої та стійкої комунікаційної підсистеми для ройових систем БПЛА є актуальним завданням, що має практичне значення для розвитку автономних технологій.

1 ПОСТАНОВКА ЗАДАЧІ СТВОРЕННЯ КОМУНІКАЦІЙНОЇ ПІДСИСТЕМИ ДЛЯ РОЙОВОГО КОМПЛЕКСУ

1.1 Введення в предметну область

У ройових системах керування автономними безпілотними літальними апаратами ефективний обмін інформацією між елементами комплексу відіграє ключову роль у забезпеченні злагодженої поведінки рою [2], оперативного реагування на зміни в середовищі та динамічної реконфігурації його складу. Розробка протоколів обміну даними між дронами та дронами і мікросервісами має враховувати високу частоту оновлення даних, обмеження пропускної здатності бездротових каналів зв'язку, а також суворі вимоги до надійності, латентності та безпеки передачі інформації.

У рамках даної роботи передбачається створення комунікаційної підсистеми для ройового комплексу, яка включає наступні основні компоненти.

1) Мікросервісна архітектура. Усі відповідні сервіси розгортаються у рамках мікросервісної архітектури на кожному дроні рою.

2) Передача відеоданих. Відеопотоки з камер дронів передаються у реальному часі з використанням фреймворку GStreamer і кодеку H.264. Це вимагає не лише ефективного стиснення та передачі даних, але й реалізації механізмів мультиплексування відеопотоків від кількох дронів для їхньої подальшої обробки на наземних сервісах, а також можливості перемикання між потоками або їхньої агрегації.

3) Телекомунікаційні повідомлення. Обмін даними, такими як навігаційні параметри (координати, швидкість, орієнтація), повідомлення про стан дрона (рівень заряду, статус систем), команди управління та аналітичні дані (наприклад, результати обробки відео чи виявлені об'єкти), здійснюється через технологію Fast DDS із серіалізацією у форматі IDL. Це забезпечує швидку, структуровану

та ефективну передачу даних у системі.

Це дозволяє забезпечити масштабованість системи (зокрема горизонтальне масштабування шляхом додавання нових інстансів), чітке розділення відповідальності між компонентами та їхнє незалежне оновлення чи модифікацію.

1.2 Проблематика

Реалізація обміну даними у ройовому середовищі пов’язана з низкою викликів, які необхідно врахувати та вирішити.

1) Гетерогенність трафіку. Одночасна передача відеопотоків великих обсягів і низьколатентних контрольних повідомлень вимагає чіткого розмежування профілів якості обслуговування (QoS) та пріоритизації даних. Важливо гарантувати, що передача відео не перешкоджає доставці критичних команд або телеметрії. Для цього можуть бути застосовані окремі віртуальні канали, пріоритетні черги або інші методи сегментації трафіку.

2) Децентралізована взаємодія. Дрони повинні мати можливість обмінюватися даними як між собою у режимі many-to-many. Це потребує впровадження ефективних механізмів виявлення (discovery) інших учасників рою та динамічного встановлення зв’язків, наприклад, через вбудовані функції Fast DDS.

3) Надійність при втраті з’єднання. Система має бути стійкою до тимчасових перебоїв у зв’язку. Для цього необхідно реалізувати буферизацію даних на дронах, механізми повторної передачі втрачених пакетів і відновлення сеансів передачі відео чи телеметрії після відновлення зв’язку.

4) Обмеження обчислювальних ресурсів. Оскільки дрони зазвичай оснащені процесорами з обмеженими обчислювальними можливостями

(наприклад, на базі ARM-архітектури), стек Fast DDS + IDL потребує оптимізації для роботи в умовах низького енергоспоживання та обмеженої пам’яті.

5) Синхронізація часу. Для коректної обробки даних, отриманих від різних дронів (зокрема відеопотоків і навігаційних параметрів), необхідна синхронізація часу між усіма учасниками системи з точністю до мілісекунд. Це критично для аналізу подій і координації дій у рої.

6) Управління конфігурацією. Система має підтримувати динамічне додавання нових дронів чи сервісів без необхідності перезавантаження всієї інфраструктури. Це вимагає гнучких механізмів реєстрації, підписки та управління конфігурацією в реальному часі.

7) Моніторинг і діагностика. Важливим є створення інструментів для відстеження стану системи, виявлення проблем (наприклад, втрати пакетів чи затримок) і збору статистичних даних для подальшого аналізу та оптимізації роботи рою.

1.3 Постановка завдання

Для досягнення мети визначено такі основні завдання.

1) Визначення семантики обміну повідомленнями.

а) Розробка детальних структур даних у форматі IDL для різних типів повідомлень (навігація, стан, команди, аналітика).

б) Визначення ієрархії тем (topics) у Fast DDS для логічного розділення потоків даних.

в) Налаштування QoS-профілів для кожної теми з урахуванням вимог до надійності, пріоритетності та латентності.

2) Розробка схеми реєстрації, підписки та аутентифікації – впровадження механізмів безпечного та контрольованого доступу дронів до системи, включаючи аутентифікацію за допомогою токенів і управління правами доступу до тем.

3) Розробка відеоадаптера.

а) Створення компонента для потокової передачі відео з дронів до відеопроцесингового адаптера з мінімальною затримкою.

б) Налаштування GStreamer для кодування, передачі та декодування відеопотоків, а також інтеграція з Fast DDS для передачі метаданих відео.

4) Налагодження роботи мікросервісів (телеметрії та відеоадаптера)

5) Тестування з використанням платформи Raspberry Pi.

а) Розгортка екземплярів програмного забезпечення дронів на тестових платах Raspberry Pi, які використовують Fast DDS для обміну телеметрією та GStreamer для передачі відеопотоків.

6) Забезпечення синхронізації часу – впровадження механізмів синхронізації часу між дронами для точної обробки часових міток у даних.

7) Тестування системи – реалізація інструментів для відстеження стану системи, виявлення аномалій (наприклад, затримок чи втрат даних).

2 ВИБІР АППАРАТНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі обґрунтовується вибір апаратного забезпечення для реалізації комунікаційної підсистеми ройового комплексу автономних безпілотних літальних апаратів. Враховуючи вимоги до системи, описані у вступі та першому розділі, зокрема необхідність ефективного обміну даними, стійкості до перебоїв у зв’язку, оптимізації для обмежених обчислювальних ресурсів і підтримки бездротових інтерфейсів обрано платформу Raspberry Pi Zero 2 W з камерою OV5647.

2.1 Вимоги до апаратного забезпечення

Для успішної реалізації проекту апаратне забезпечення має відповідати таким основним вимогам.

1) Обчислювальна потужність: достатня для обробки відеопотоків і телеметричних даних у реальному часі.

2) Енергоефективність: низьке енергоспоживання, що є критично важливим для автономних дронів з обмеженим джерелом живлення.

3) Підтримка бездротових інтерфейсів: наявність Wi-Fi для зв’язку з іншими дронами та наземними сервісами.

4) Підтримка камери: можливість підключення камери для захоплення відеоданих.

5) Компактність і легкість: малі розміри та вага для інтеграції в дрони.

6) Вартість: доступна ціна для масштабування на кілька дронів.

Ці вимоги випливають із завдань, визначених у підрозділі 1.3, зокрема необхідності тестування на платформі Raspberry Pi, а також із контексту розробки комунікаційної підсистеми для ройового комплексу.

2.2 Обґрунтування вибору Raspberry Pi Zero 2 W

Raspberry Pi Zero 2 W – це компактний і доступний одноплатний комп’ютер, який є оновленою версією Raspberry Pi Zero W. Завдяки своїм технічним характеристикам і функціональності, він є ідеальним вибором для вбудованих систем, таких як комунікаційні підсистеми ройових комплексів безпілотних літальних апаратів. Ось ключові характеристики та причини, чому ця платформа є оптимальною завдяки її відповідності наступним вимогам [2].

1) Вартість: доступна ціна для масштабування на кілька дронів. Raspberry Pi Zero 2 W є однією з найдоступніших плат у своєму класі, що сприяє масштабуванню системи на кілька дронів без значних витрат.

2) Обчислювальна потужність: оснащена чотири-ядерним процесором Broadcom BCM2710A1 (Arm Cortex-A53, 1 ГГц), що забезпечує достатню продуктивність для обробки відеоданих і телеметрії в реальному часі.

3) Енергоефективність: має низьке енергоспоживання, що дозволяє використовувати її в автономних системах з обмеженим джерелом живлення.

4) Підтримка бездротових інтерфейсів: вбудований Wi-Fi модуль (2.4 ГГц IEEE 802.11b/g/n) забезпечує зв’язок з іншими пристроями без додаткового обладнання.

5) Підтримка камери: наявність CSI-інтерфейсу дозволяє підключити камеру OV5647, яка забезпечує відеозапис у форматі 1080p при 30 кадрах на секунду.

6) Компактність і легкість: розміри плати становлять 65 мм × 30 мм, що робить її ідеальною для інтеграції в дрони.

Крім того, широка підтримка спільноти розробників і сумісність із програмним забезпеченням, таким як Raspberry Pi OS, спрощують розробку та інтеграцію компонентів системи.

2.3 Характеристики обраного апаратного забезпечення

Основні технічні характеристики Raspberry Pi Zero 2 W (рис. 2.1).

1) Процесор. Broadcom BCM2710A1, чотири-ядерний Cortex-A53 (ARMv8) 64-бітний SoC @ 1 ГГц.

2) Пам’ять. 512 МБ LPDDR2 SDRAM.

3) Бездротові інтерфейси. 2.4 ГГц IEEE 802.11b/g/n Wi-Fi, Bluetooth 4.2.

4) Інтерфейси. Mini HDMI, USB 2.0 OTG, CSI для камери, 40-pin GPIO.

5) Живлення. 5 В через micro USB.

6) Операційна система. Raspberry Pi OS, що підтримує необхідні бібліотеки та інструменти для розробки.

Для захоплення відеоданих використовується камера OV5647, яка підключається до CSI-інтерфейсу плати. Камера має роздільну здатність 5 Мп і підтримує відеозапис у форматі 1080p при 30 кадрах на секунду, що відповідає потребам передачі відеопотоків, описаних у підрозділі 1.1. Фактична роздільна здатність відео, бітрейт та інші характеристики відеопотоку описуються у підрозділі 5.2.

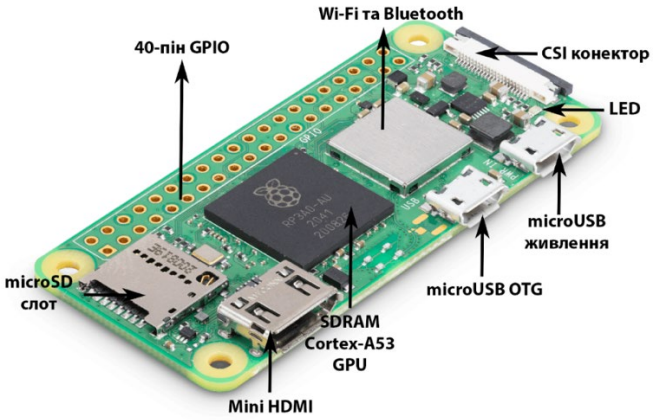

Рисунок 2.1 – Raspberry Pi Zero 2 W

2.4 Порівняння з альтернативними платформами

Для обґрунтування вибору розглянемо альтернативні платформи для вбудованих систем, таких як комунікаційні підсистеми ройових комплексів безпілотних літальних апаратів.

1) Arduino. Має меншу обчислювальну потужність і не підтримує операційну систему, що ускладнює реалізацію складних завдань обробки даних.

2) BeagleBone. Забезпечує кращу продуктивність, але має більші розміри та вище енергоспоживання, що не є оптимальним для дронів.

3) NVIDIA Jetson. Пропонує високу продуктивність для завдань машинного навчання, але є дорожчою і споживає більше енергії, що не завжди виправдано для даної задачі.

Порівняльний аналіз зазначений в таблиці (додаток А) показує, що Raspberry Pi Zero 2 W найкраще відповідає вимогам комунікаційної підсистеми рою БПЛА за наступними критеріями.

1) Баланс продуктивності та енергоспоживання. Завдяки процесору ARM Cortex-A53 (1 GHz) та низькому споживанню (0,3–1 Вт) забезпечує достатню швидкість для обробки потокових даних та мережевих протоколів.

2) Розміри та вага. Найкомпактніша Linux-платформа серед розглянутих, що критично для мобільних та аеродинамічних характеристик БПЛА.

3) Вартість. Орієнтовно $15–20 за саму плату, доступні численні аксесуари (корпуси, антени, корпуси із вбудованим живленням).

4) Екосистема та підтримка. Велика спільнота, підтримка безлічі мов та бібліотек (Python, C/C++, Node.js), що дозволяє швидко реалізувати та налагодити системи передачі телеметрії, маршрутизації й обробки даних.

Таким чином, Raspberry Pi Zero 2 W є оптимальним вибором з точки зору балансу між продуктивністю, енергоефективністю, вартістю та підтримкою спільноти (Додаток А).

2.5 Конфігурація апаратного забезпечення

Для впровадження та тестування системи використовуються дві плати Raspberry Pi Zero 2 W, які слугують основою для реалізації ройового комплексу автономних безпілотних літальних апаратів. Ці плати обрані завдяки їхній компактності, енергоефективності та підтримці бездротових інтерфейсів, що робить їх оптимальним вибором для мобільних автономних систем. Конфігурація апаратного забезпечення включає два основні вузли: «Дрон» з камерою та «Матка» рою, кожен з яких має специфічний набір компонентів, необхідних для виконання своїх функцій (рис. 2.2).

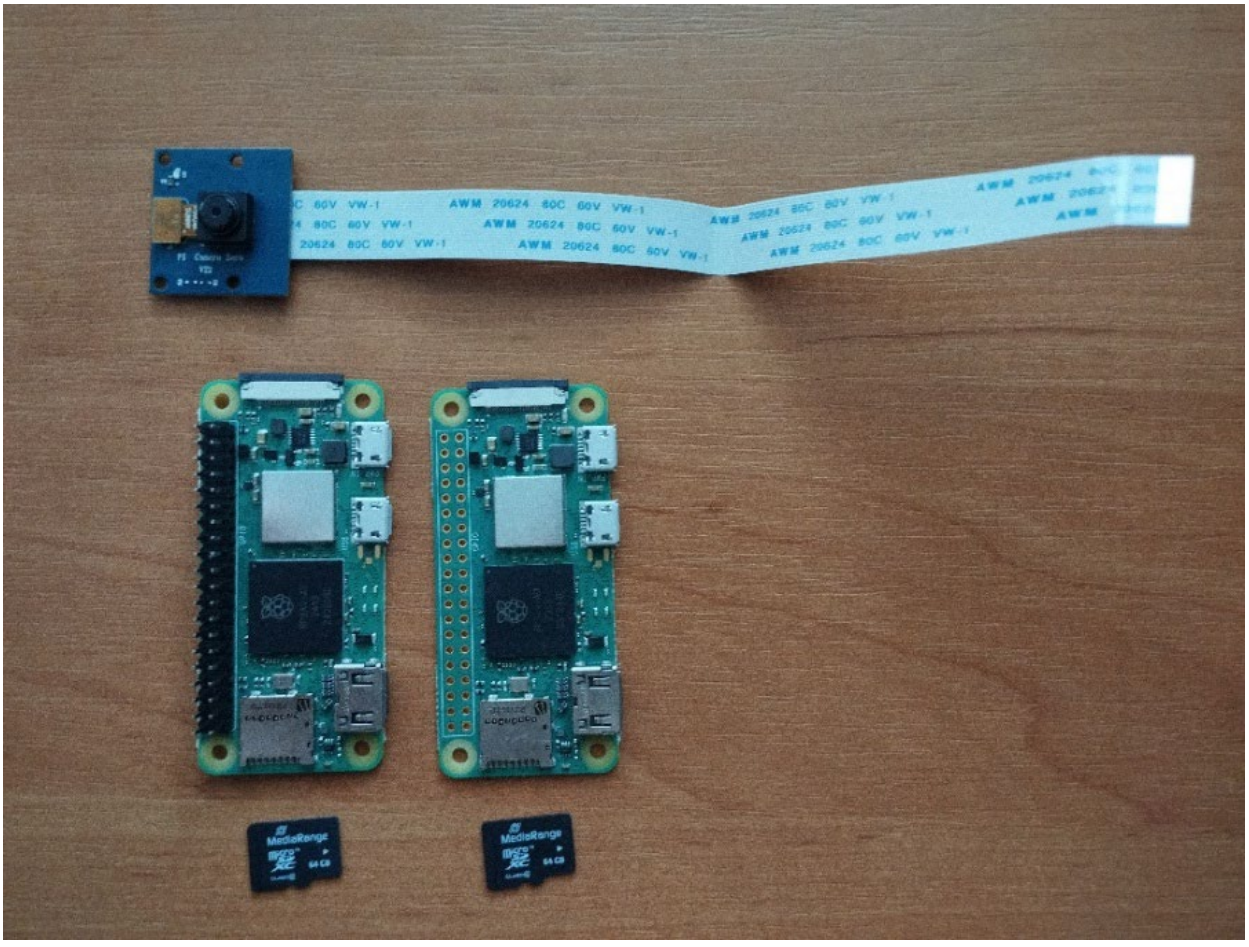

Рисунок 2.2 – Апаратне забезпечення тестових плат Raspberry Pi Zero 2 W

2.5.1 Плата «Дрон» з камерою

«Дрон» є автономним агентом рою, відповідальним за збір та передачу відеоданих і телеметрії. Його апаратне забезпечення складається з таких компонентів.

– Raspberry Pi Zero 2 W. Ця плата виступає як основний обчислювальний модуль дрона. Завдяки чотириядерному процесору ARM Cortex-A53 (1 ГГц) і 512 МБ оперативної пам’яті, вона забезпечує достатню продуктивність для обробки відеопотоків і телеметричних даних у реальному часі. Вбудований Wi-Fi модуль (2.4 ГГц IEEE 802.11b/g/n) дозволяє дронам обмінюватися даними з «Маткою» та іншими дронами в рої без потреби в додатковому обладнанні.

– Камера OV5647. Камера підключається до CSI-інтерфейсу плати Raspberry Pi і відповідає за отримання відеоданих. Вона має роздільну здатність 5 Мп і підтримує запис відео у форматі 1080p при 30 кадрах на секунду, що відповідає вимогам для передачі високоякісних відеопотоків у реальному часі. Камера є ключовим компонентом для моніторингу та виконання завдань, пов’язаних із візуальним спостереженням.

– Акумулятор або інше джерело живлення. Для забезпечення автономності дрона використовується портативний акумулятор, що підключається через MicroUSB-порт плати. Акумулятор має напругу 5 В і ємність, яка є достатньою для підтримки роботи дрона протягом відповідної місії. Це джерело живлення забезпечує мобільність і незалежність дрона від стаціонарних джерел енергії.

– Засоби для зберігання даних. Карта пам’яті MEDIARANGE Micro-SDXC 64GB Class 10 (MR955), використовується для зберігання операційної системи Raspberry Pi OS, програмного забезпечення дрона та тимчасових даних, таких як буферизовані відеопотоки або телеметрія у разі перебоїв у зв’язку. Клас 10

забезпечує достатню швидкість запису (не менше 10 МБ/с), що є критичним для стабільної роботи системи та запису відеоданих.

2.5.2 Плата «Дрон-матка» рою

«Матка» виступає як центральний вузол рою, відповідальний за прийом, обробку та зберігання та передачу даних від дронів, а також за координацію їх дій. Апаратне забезпечення дрону «Матки» включає наступне.

– Raspberry Pi Zero 2 W. Аналогічно до дрона, «Матка» використовує цю плату як основний обчислювальний модуль. Її продуктивність дозволяє обробляти дані від кількох дронів одночасно, а вбудований Wi-Fi модуль забезпечує зв’язок з дронами та іншими вузлами системи.

– Акумулятор або інше джерело живлення. «Матка» також оснащена портативним акумулятором для забезпечення автономності. Це дозволяє розміщувати «Матку» в польових умовах або на мобільних платформах, що розширює можливості розгортання системи.

– Засоби для зберігання даних. Карта пам’яті MEDIARANGE Micro-SDXC 64GB Class 10 (MR955). Карта пам’яті використовується для зберігання операційної системи, програмного забезпечення «Матки», а також для архівування отриманих від дронів відеопотоків і телеметричних даних. Об’єм 64 ГБ забезпечує достатньо місця для тривалого зберігання даних під час місій.

2.5.3 Взаємодія компонентів

Обидві плати підключаються до однієї Wi-Fi мережі, що дозволяє їм обмінюватися даними в режимі реального часу. «Дрон» передає відеопотоки та телеметричні дані до «Матки» через Wi-Fi за допомогою відповідних протоколів.

2.6 Інтеграція з програмним забезпеченням

Апаратне забезпечення, зокрема плати Raspberry Pi Zero 2 W, інтегрується з програмним забезпеченням через операційну систему Raspberry Pi OS. Ця операційна система забезпечує стабільне та оптимізоване середовище для виконання необхідних програмних пакетів: GStreamer для передачі відеопотоків, Fast DDS для розподілу даних у реальному часі та IDL для серіалізації даних.

Raspberry Pi OS. Як основа, Raspberry Pi OS надає необхідні драйвери та бібліотеки для взаємодії з апаратними компонентами, такими як камера OV5647 і Wi-Fi модуль. Це гарантує, що програмне забезпечення може ефективно використовувати апаратні можливості Raspberry Pi Zero 2 W.

GStreamer. Цей мультимедійний фреймворк використовується для захоплення відео з камери OV5647 на «Дроні», його ефективного кодування та потокової передачі через Wi-Fi мережу до «Матки». Конфігурація пайплайнів GStreamer оптимізована для роботи з обмеженими обчислювальними ресурсами Raspberry Pi Zero 2 W, забезпечуючи при цьому передачу відео в реальному часі.

Fast DDS, виконуючи роль проміжного програмного забезпечення для комунікації в реальному часі, Fast DDS забезпечує обмін телеметричними даними, командами та іншими критичними повідомленнями між «Дроном» і «Маткою». Він підтримує децентралізовану взаємодію завдяки можливості many-to-many комунікації, що є важливим для функціонування рою. Профілі якості обслуговування (QoS) налаштовані для пріоритизації критичних повідомлень і забезпечення надійної доставки даних, навіть у разі перебоїв у мережі.

IDL Використовується для серіалізації структурованих даних, IDL гарантує, що телеметричні повідомлення та команди є компактними та ефективно передаються через мережу. Визначені схеми для різних типів даних забезпечують

коректну серіалізацію та десеріалізацію даних усіма компонентами системи.

Крім того, ця інтеграція вирішує ключові виклики.

– Гетерогенність трафіку керується шляхом розділення відеопотоків (GStreamer) і телеметрії/команд (Fast DDS).

– Децентралізована взаємодія забезпечується підтримкою Fast DDS many-to-many комунікації.

– Надійність при втраті з’єднання досягається завдяки можливостям Fast DDS щодо повторної передачі втрачених повідомлень.

– Обмежені обчислювальні ресурси мінімізуються за рахунок використання легких протоколів і ефективної серіалізації.

– Синхронізація часу підтримується функціями Fast DDS для роботи з часовими мітками.

Загалом, інтеграція апаратного та програмного забезпечення через Raspberry Pi OS, GStreamer, Fast DDS і IDL забезпечує надійну та ефективну платформу для роботи ройової системи.

3 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі обґрунтовується вибір програмного забезпечення для реалізації комунікаційної підсистеми ройового комплексу автономних безпілотних літальних апаратів. Вибір здійснюється з урахуванням вимог до системи, описаних у вступі та розділі 1, а також характеристик обраного апаратного забезпечення (Raspberry Pi Zero 2 W), розглянутого у розділі 3. Метою є забезпечення надійного, швидкого та безпечного обміну даними між дронами.

3.1 Вимоги до програмного забезпечення

Програмне забезпечення має відповідати таким ключовим вимогам, що випливають із завдань, визначених у підрозділі 1.3, та контексту роботи.

1) Сумісність з апаратним забезпеченням: підтримка Raspberry Pi Zero 2 W з обмеженими ресурсами (512 МБ ОЗП, чотириядерний процесор Cortex-A53).

2) Ефективність використання ресурсів: оптимізація для роботи на платформах з низьким енергоспоживанням та обмеженою обчислювальною потужністю.

3) Підтримка реального часу. Низька латентність передачі даних і відеопотоків для оперативного реагування рою.

4) Масштабованість. Можливість інтеграції нових дронів і сервісів у систему без значних змін.

5) Надійність і безпека. Стійкість до перебоїв зв’язку, механізми аутентифікації та захисту даних.

6) Гнучкість інтеграції. Сумісність з іншим програмним забезпеченням.

Ці вимоги враховують гетерогенність трафіку, децентралізовану взаємодію та необхідність синхронізації часу, зазначені в підрозділі 1.2.

3.2 Вибір операційної системи

Для Raspberry Pi Zero 2 W обрано Raspberry Pi OS як операційну систему з таких причин.

1) Сумісність. ОС оптимізована для апаратного забезпечення Raspberry Pi, що гарантує стабільну роботу на платі з CSI-інтерфейсом і Wi-Fi модулем.

2) Легкість. 64-бітна версія ефективно використовує обмежені ресурси (512 МБ ОЗП), що відповідає вимогам енергоефективності.

3) Підтримка. Широка спільнота розробників і регулярні оновлення забезпечують актуальність і безпеку.

4) Наявність пакетів. Репозиторії включають необхідні бібліотеки (GStreamer, Fast DDS, IDL), що спрощує розробку.

5) Альтернативи, такі як Ubuntu Server, відкинуті через більші вимоги до ресурсів, що не відповідає характеристикам Raspberry Pi Zero 2 W.

3.3 Вибір технологій для передачі даних

Для телекомунікаційних повідомлень (навігація, стан, команди) обрано Fast DDS та IDL.

1) Fast DDS. Реалізація стандарту DDS (Data Distribution Service), що забезпечує децентралізовану передачу даних за моделлю publish-subscribe. Підтримує QoS-профілі для пріоритизації трафіку, механізми виявлення учасників (discovery) і стійкість до перебоїв зв’язку, що відповідає вимогам підрозділу 1.2.

2) IDL, який забезпечує відповідність визначених даних. Зменшує навантаження на мережу та процесор, що критично для дронів з обмеженими ресурсами.

Fast DDS ідеально підходить для взаємодії в рої та інтеграції з новими

сервісами. IDL оптимізує передачу структурованих даних (навігація, стан), зменшуючи розмір повідомлень порівняно з JSON.

3.4 Вибір фреймворку для відеопотоків

Для передачі відеоданих з камери OV5647 на платформі Raspberry Pi Zero 2 W обрано GStreamer як основний мультимедійний фреймворк. Нижче наведене детальне обґрунтування вибору та порівняння з альтернативою (FFmpeg), а у таблиці 4.1 наведене узагальнене порівняння.

GStreamer на Raspberry Pi використовує графічний процесор VideoCore через OpenMAX IL (у старіших версіях MMAL), що дозволяє виконувати кодування з використанням H.264 із мінімальним залученням центрального процесора. Це особливо важливо для БПЛА, де обчислювальні ресурси й енергія обмежені. За допомогою тонкого налаштування параметрів буферів (queue, rtph264pay, rtpjitterbuffer) можна досягти кінцевої затримки менше 100 мс для одного каналу відео 480p 15-25 кадрів в секунду. Таке значення задовольняє вимоги реального часу, окреслені в розділі 1.1. Використання GStreamer дозволяє динамічно змінювати пайплайни під час роботи (за допомогою gst_element_set_state), додавати чи видаляти різні фрейм-обробні елементи (наприклад, детектори руху, накладення тексту) без перезапуску всього потоку. Крім того, доступне до використання мультиплексування декількох відеопотоків у єдиний RTP-multicast потік (Додаток Б). Отже GStreamer забезпечує ідеальний баланс між апаратним прискоренням, низькою латентністю, гнучкістю пайплайнів та простою інтеграцією з Fast DDS. Ці переваги роблять його оптимальним вибором для побудови високопродуктивної та енергоефективної системи передачі відеопотоків з дронів.

3.5 Інструменти розробки та тестування

У процесі розробки та тестування проекту використано набір інструментів, які забезпечують ефективну роботу з кодом, віддалене керування пристроями та налаштування системи. До основних інструментів належать SSH (через PowerShell у операційній системі Windows), VS Code та Vim. Кожен із них виконує специфічні функції, що сприяють реалізації проекту.

3.5.1 SSH (через PowerShell у Windows)

SSH (Secure Shell) застосовується для віддаленого доступу до пристроїв зокрема, із операційної системи Windows через PowerShell. Цей інструмент дозволяє підключатися до апаратного забезпечення, у даному випадку, Raspberry Pi, для виконання команд, передачі файлів та керування системою. Завдяки SSH забезпечується наступне.

– Віддалене налаштування та запуск програм.

– Безпечна передача даних між локальним комп’ютером і віддаленим пристроєм.

– Моніторинг роботи системи в реальному часі.

Використання SSH є ключовим для управління розподіленими системами, що потребують віддаленого доступу.

3.5.2 VS Code

VS Code (Visual Studio Code) – це сучасний редактор коду, який використовується для написання, редагування та налагодження програмного забезпечення. У проекті він слугує основним інструментом розробки завдяки

таким можливостям.

– Підтримка різних мов програмування, таких як Python або C++

– Інтеграція з розширеннями, наприклад, для віддаленої роботи через SSH.

– Вбудовані засоби для налагодження та тестування коду.

– Зручна робота з системами контролю версій, зокрема Git.

Цей інструмент значно підвищує продуктивність розробки завдяки гнучкості та широкому функціоналу.

3.5.3 Vim

Vim – це текстовий редактор, який використовується для швидкого редагування файлів безпосередньо на віддалених пристроях через термінал. Його основні переваги в контексті проекту наступні.

– Мінімальні вимоги до ресурсів, що важливо для роботи на пристроях із обмеженими можливостями;

– Оперативне внесення змін до конфігураційних файлів або коду.

– Інтеграція з SSH для редагування файлів у віддаленому середовищі.

– Застосовується для правок коду, налаштувань, доповнюючи основну розробку в VS Code.

Комбінація SSH, VS Code та Vim забезпечує повноцінний цикл розробки та тестування: від написання коду до його розгортання та налаштування на пристроях. SSH відповідає за віддалений доступ, VS Code – за основну розробку, а Vim – за швидкі зміни в терміналі. Ці інструменти оптимально підходять для потреб проекту, забезпечуючи ефективність і зручність роботи.

4 НАЛАШТУВАННЯ КОМУНІКАЦІЙНОЇ ПІДСИСТЕМИ

У цьому розділі описано налаштування комунікаційної підсистеми ройового комплексу автономних безпілотних літальних апаратів на базі платформи Raspberry Pi Zero 2 W. Налаштування охоплює: налаштування середовища та системи, інсталяцію та налаштування операційної системи, встановлення необхідних пакетів та бібліотек, налаштування апаратного забезпечення.

4.1 Налаштування системи

Для забезпечення роботи системи на платформах Raspberry Pi Zero 2 W необхідно виконати підготовку апаратного та програмного середовища. Налаштування включає встановлення необхідних бібліотек і пакетів, а також конфігурацію камери.

4.1.1 Інсталяція та налаштування операційної системи

Операційна система Raspberry Pi OS є офіційно підтримуваною для платформи Raspberry Pi Zero 2 W. Вона базується на 32-бітній або 64-бітній версіях Linux і оптимізована для роботи з апаратним забезпеченням Raspberry Pi. Інсталяція операційної системи здійснюється на microSD-карту за допомогою інструменту Raspberry Pi Imager, що забезпечує базове програмне середовище для подальшої роботи з пристроєм. Нижче наведено детальну послідовність кроків для завантаження та встановлення Raspberry Pi OS.

1) Завантаження інсталятору Raspberry Pi Imager. Інсталятор Raspberry Pi OS Imager (рис. 4.1) доступний на офіційному веб-сайті Raspberry Pi [3].

2) Встановлення Raspberry Pi Imager. Завантажений файл запускається,

після чого виконуються інструкції інсталятора для завершення процесу встановлення програми на персональний комп'ютер. Після цього, натискання кнопки Finish відкриває Raspberry Pi Imager (рис. 4.2).

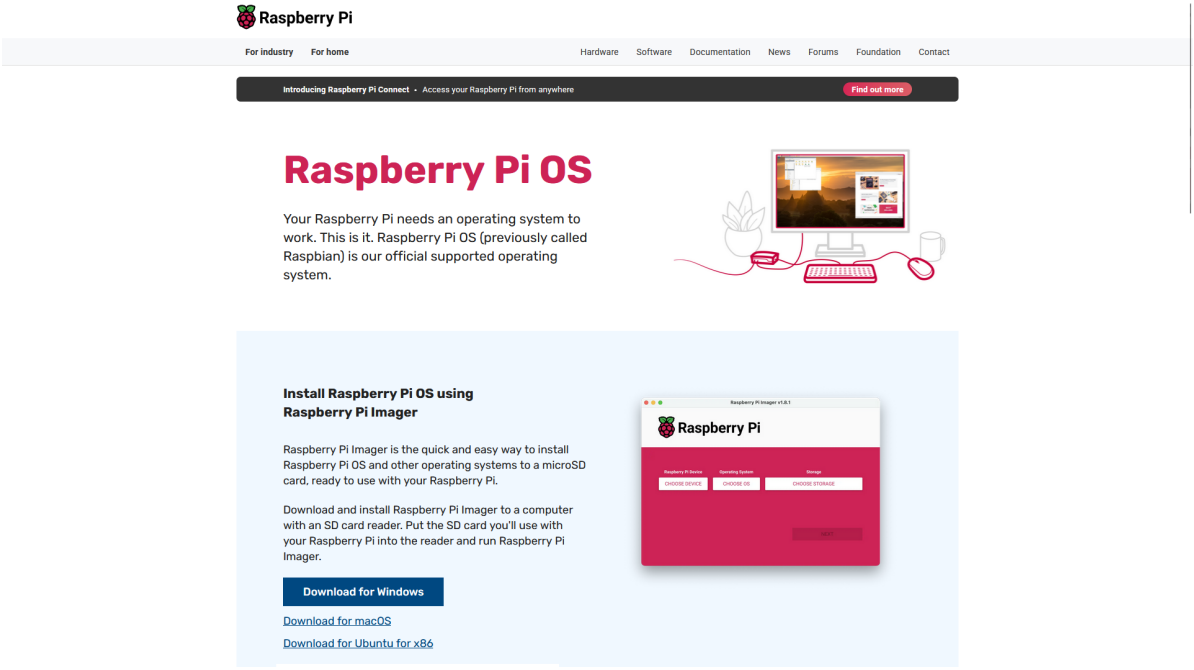

Рисунок 4.1 – Сторінка для завантаження Raspberry Pi OS Imager

3) Встановлення Raspberry Pi Imager. Завантажений файл запускається, після чого виконуються інструкції інсталятора для завершення процесу встановлення програми на персональний комп'ютер. Після цього, натискання кнопки Finish відкриває Raspberry Pi Imager (рис. 4.2).

4) Підготовка microSD-карти. MicroSD-карта (рекомендується об'єм 32 ГБ або більше) під'єднується до комп'ютера через кард-рідер або інший пристрій. У рамках цього проекту використано карту об'ємом 64 ГБ, як зазначено в підрозділі 3.5. Для правильного функціонування системи потрібно відформатувати MicroSD-карту у формат FAT32 (рис. 4.3), у даному випадку, вбудованими засобами операційної системи Windows.

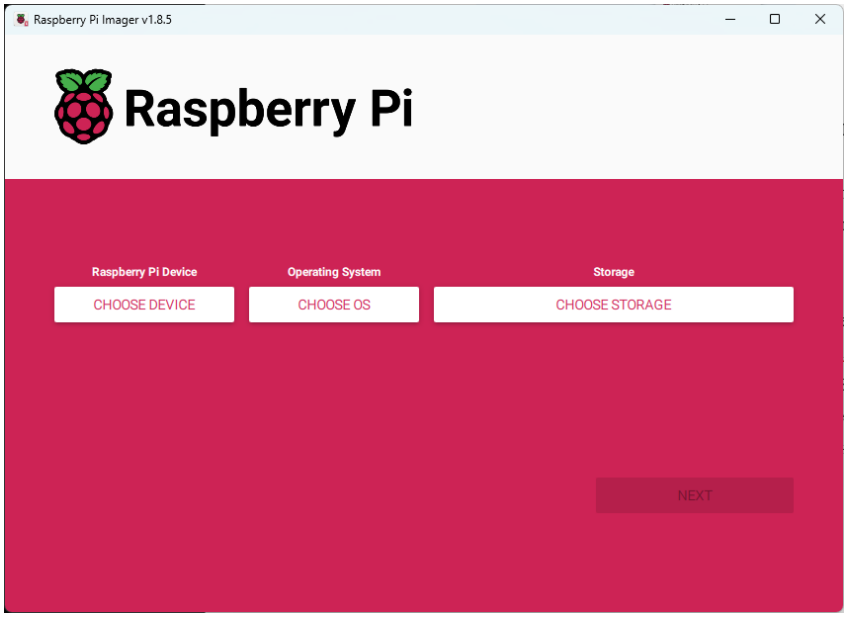

Рисунок 4.2 – Програма Raspberry Pi Imager

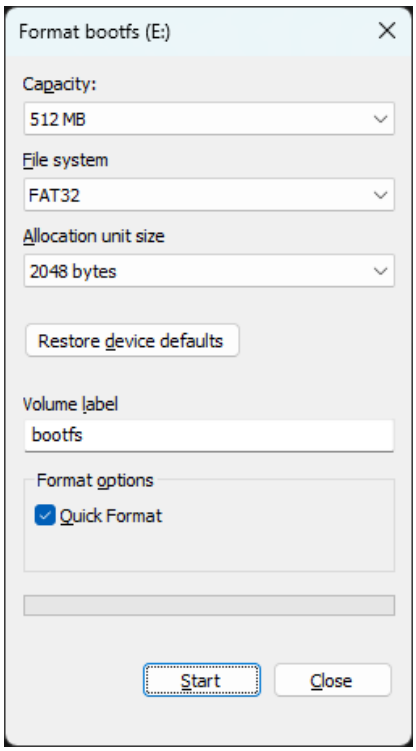

Рисунок 4.3 – Форматування MicroSD-картки

5) Вибір пристрою, операційної системи та носія. У вікні Raspberry Pi Imager (рис. 4.2) активуються наступні попередні налаштування.

– CHOOSE DEVICE. Вибір пристрою, у даному випадку – Raspberry Pi Zero 2 W (рис 4.4).

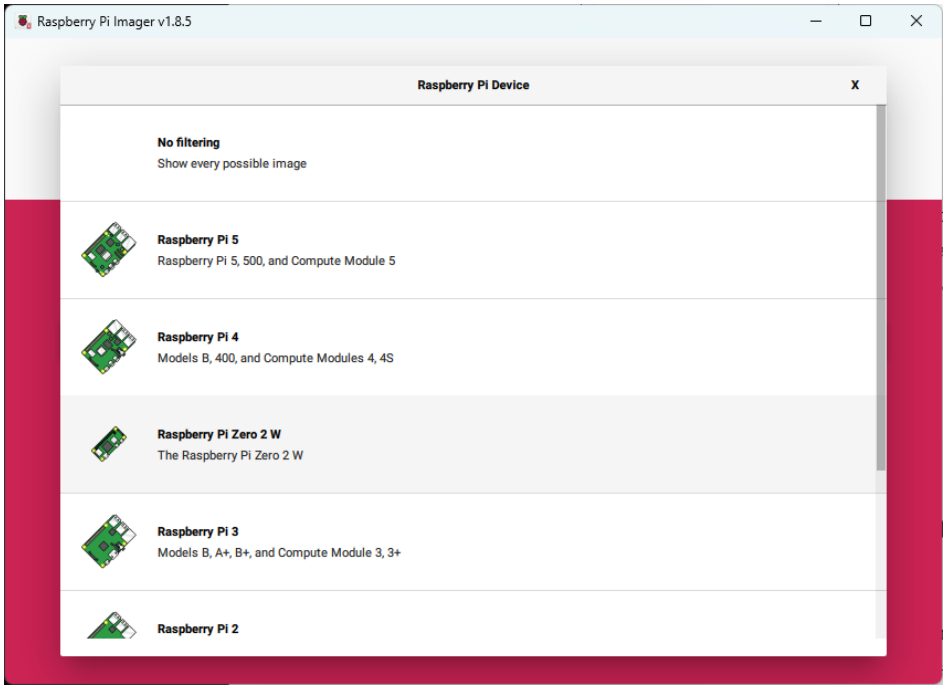

Рисунок 4.4 – Вибір пристрою у Raspberry Pi Imager

– CHOOSE OS. Вибір версії операційної системи (рис. 4.5), у даному випадку – Raspberry Pi OS (64-bit) як основна операційна система для інсталяції.

– CHOOSE STORAGE. Вибір носія для встановлення операційної системи (рис. 4.6), у даному випадку – microSD-карту як цільовий носій для запису операційної системи.

6) Налаштування параметрів системи. Після вибору пристрою, версії операційної системи та носія потрібно налаштувати додаткові параметри системи для забезпечення коректної роботи.

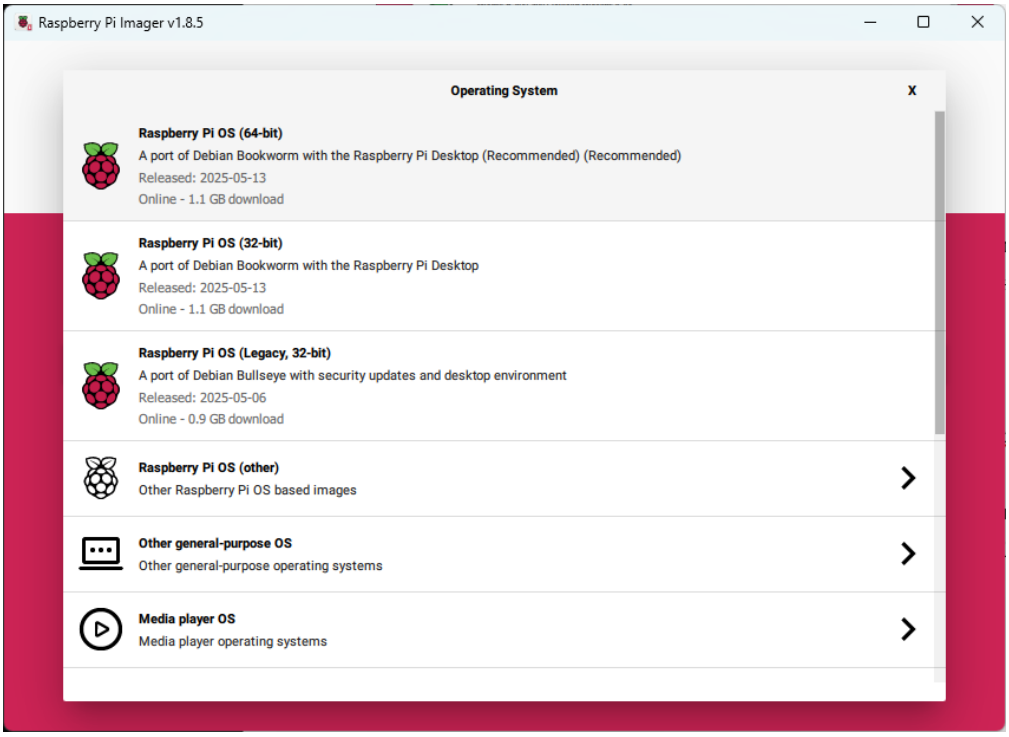

Рисунок 4.5 – Налаштування операційної системи у Raspberry Pi Imager

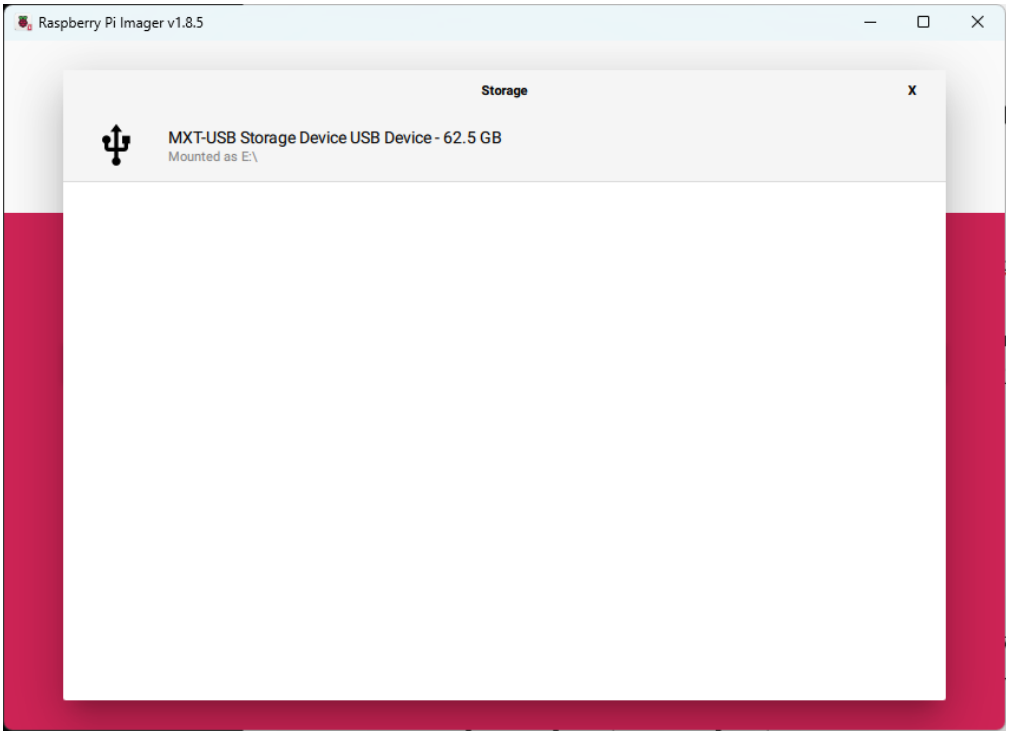

Рисунок 4.6 – Вибір носія у Raspberry Pi Imager

– Set hostname. Визначається як ім’я хоста для ідентифікації пристрою в мережі. На рисунку зображено приклад налаштування для одного з пристроїв, а саме той, який відіграє роль дрона з камерою (рис. 4.7).

– Set username and password. Визначається як логін та пароль користувача операційної системи Raspberry Pi OS для забезпечення безпеки та організації доступу (рис. 4.7).

– Configure wireless LAN. Визначаються параметри бездротової мережі – SSID, пароль, а також розташування роутера (UA для України) (рис. 4.7).

– Set local settings. Визначаються як налаштування часового поясу та розкладки клавіатури (рис. 4.7).

– Enable SSH. Визначається як опція з аутентифікацією за паролем для забезпечення віддаленого доступу до пристрою через SSH (рис. 4.8).

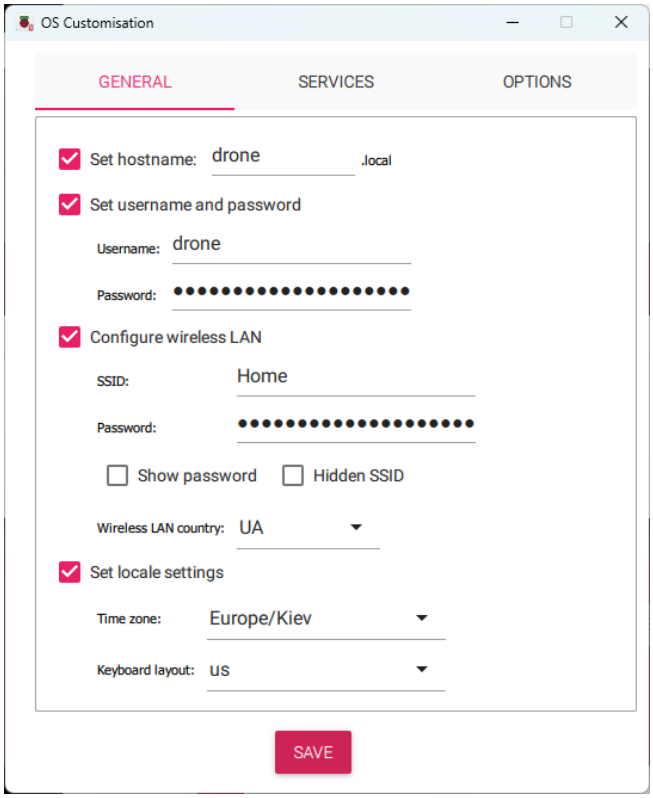

Рисунок 4.7 – Налаштування параметрів системи

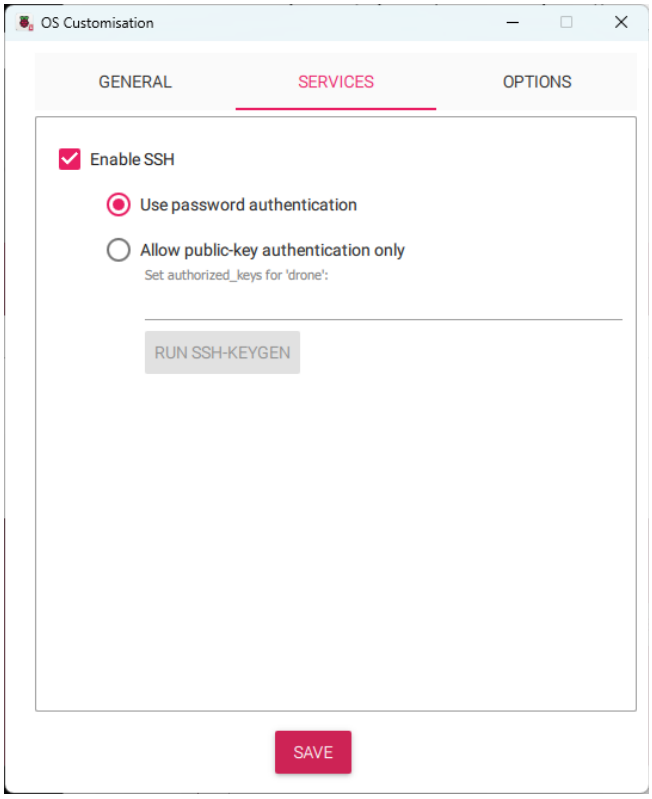

Рисунок 4.8 – Налаштування параметрів SSH

7) Запис операційної системи. Процес запису операційної системи на microSD-карту відбувається після налаштування параметрів системи. Завершення запису супроводжується верифікацією даних, після чого microSD-карта безпечно виймається з комп'ютера.

8) Підготовка до запуску Raspberry Pi Zero 2 W. MicroSD-карта розміщується в слоті Raspberry Pi Zero 2 W. Пристрій підключається до джерела живлення через MicroUSB-кабель. Завантаження пристрою відбувається з установленою операційною системою. Подальше налаштування віддаленого доступу описано в підрозділі 4.1.2.

Попереднє налаштування SSH та Wi-Fi під час інсталяції забезпечує можливість віддаленого керування пристроєм, що відіграє важливу роль у функціонуванні ройового комплексу дронів, де фізичний доступ до кожного

вузла може бути обмеженим.

Таким чином, описаний процес інсталяції Raspberry Pi OS створює основу для розгортання програмного забезпечення та інтеграції компонентів проекту.

4.1.2 Встановлення необхідних пакетів та бібліотек

Gstreamer [4] використовується для захоплення, кодування та передачі відеопотоків у реальному часі. Для його роботи необхідно встановити наступні пакети [5, 6].

- Gstreamer1.0-tools – базові утиліти для роботи з Gstreamer.
- Gstreamer1.0-plugins-base – основні плагіни.
- Gstreamer1.0-plugins-good – додаткові плагіни високої якості.
- Gstreamer1.0-plugins-bad – експериментальні плагіни.
- Gstreamer1.0-libav – підтримка кодеків FFmpeg.
- Libgstreamer1.0-dev – бібліотеки для розробки.
- Libgstrtspserver-1.0-dev – підтримка RTSP-серверів.

Fast DDS забезпечує обмін телеметричними даними між дронами та маткою. Необхідні наступні пакети [7].

- Libasio-dev – асинхронна бібліотека вводу-виводу.
- Libtinyxml2-dev – парсер XML для конфігурацій.
- Libssl-dev – криптографічні функції.
- Libfastrtps-dev – бібліотеки Fast DDS.
- Fastdds-tools – інструменти для генерації коду.

Для серіалізації даних у Fast DDS використовується IDL. Встановлюються наступний пакет: Fastdds_gen – компілятор IDL.

Усі пакети встановлюються за допомогою терміналу Raspberry Pi OS, до якого потрібне попереднє підключення через SSH (рис 4.9).

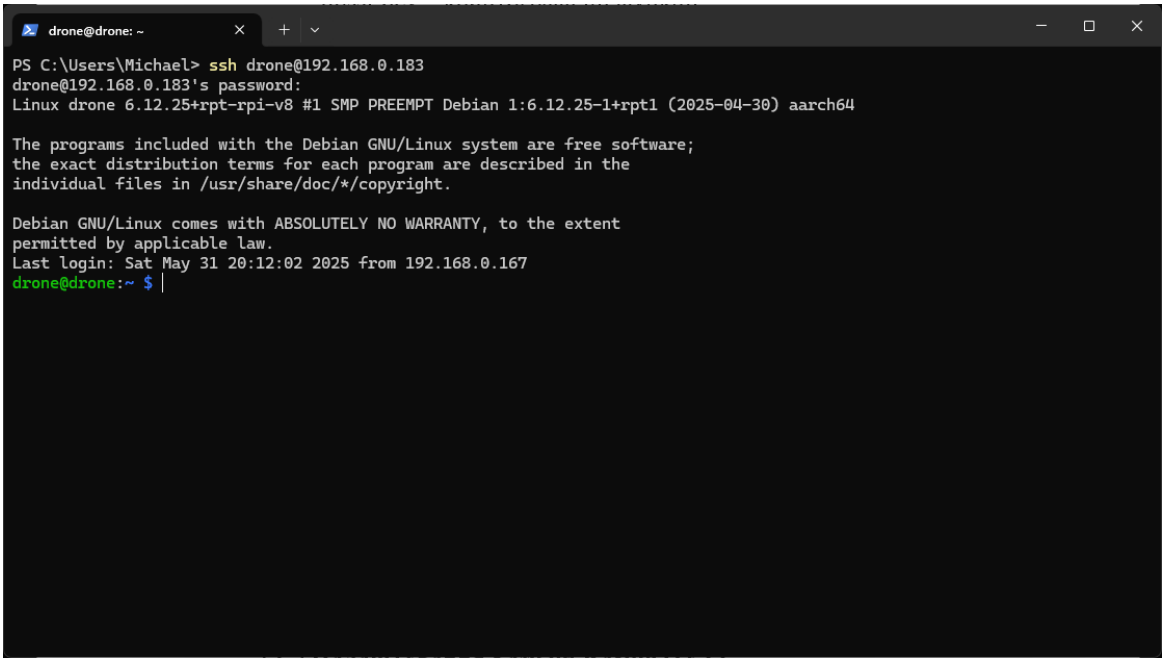

Рисунок 4.9 – Підключення до Raspberry Pi Zero 2 W через SSH

Для забезпечення коректної роботи системи необхідно встановити відповідні програмні пакети та бібліотеки, які є основою для функціонування ключових компонентів програмного забезпечення. Зокрема, ці компоненти включають GStreamer для обробки та передачі мультимедійних даних, Fast DDS для обміну телеметричними даними в реальному часі та IDL для серіалізації структурованих даних. Встановлення здійснюється за допомогою пакетного менеджера apt (рис 4.10), який є стандартним інструментом для операційних систем на базі Linux (Debian), таких як Raspberry Pi OS.

Першим кроком у процесі встановлення є оновлення списку доступних пакетів для забезпечення актуальності версій програмного забезпечення. Це досягається виконанням команди sudo apt update. Ця команда синхронізує локальний індекс пакетів із репозиторіями, дозволяючи системі отримати інформацію про доступні оновлення та нові пакети. Оновлення є важливим етапом, оскільки застарілі версії пакетів можуть призвести до несумісності або

помилок під час роботи системи. Після оновлення списку пакетів виконується встановлення необхідного програмного забезпечення за допомогою наступної команди [6, 7, 9] (лістинг 4.1).

```
sudo apt install -y gstreamer1.0-tools gstreamer1.0-plugins-base
gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-
libav libgstreamer1.0-dev libgstrtspserver-1.0-dev libasio-dev
libtinyxml2-dev libssl-dev libfastrtps-dev fastdds-tools
fastdds_gen
```

Лістинг 4.1 – Команда встановлення необхідних програмних пакетів

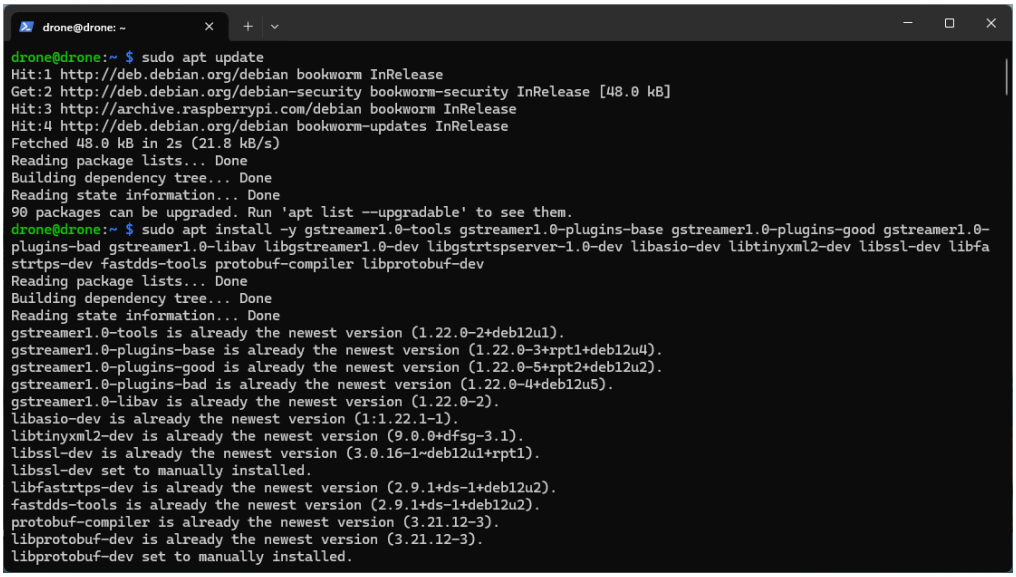

Рисунок 4.10 – Встановлення необхідних програмних пакетів та бібліотек

4.1.3 Налаштування камери

Для забезпечення коректного захоплення відеоданих із камери OV5647 на платформі Raspberry Pi Zero 2 W необхідно активувати CSI-інтерфейс (Camera Serial Interface) та відповідні апаратні інтерфейси, зокрема I2C. Цей етап є фундаментальним, оскільки відеопотоки слугують основним джерелом

інформації для подальшої обробки в рамках проекту. Активація CSI-інтерфейсу здійснюється через системну утиліту конфігурації raspi-config, що дозволяє адаптувати апаратне забезпечення до потреб роботи з камерою (рис. 4.11).

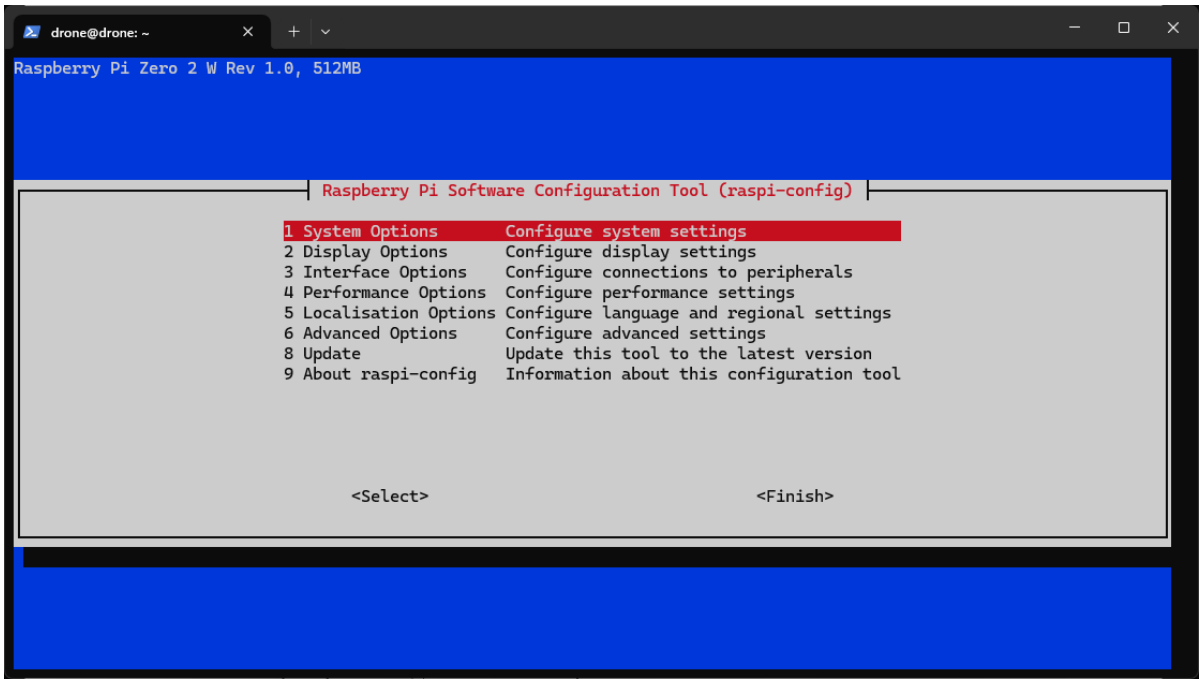

Рисунок 4.11 – Утиліта конфігурації raspi-config

У головному меню утиліти конфігурації потрібно обрати розділ Interface Options, який містить параметри для активації апаратних інтерфейсів (рис. 4.12), зокрема CSI та I2C. Цей крок є важливим для доступу до опцій, пов’язаних із камерою.

Після збереження конфігурації, система запропонує перезавантажити пристрій. Перезавантаження необхідне для ініціалізації драйверів і застосування внесених змін. Далі, рекомендується перевірити коректність роботи камери, зокрема працездатність апаратних інтерфейсів.

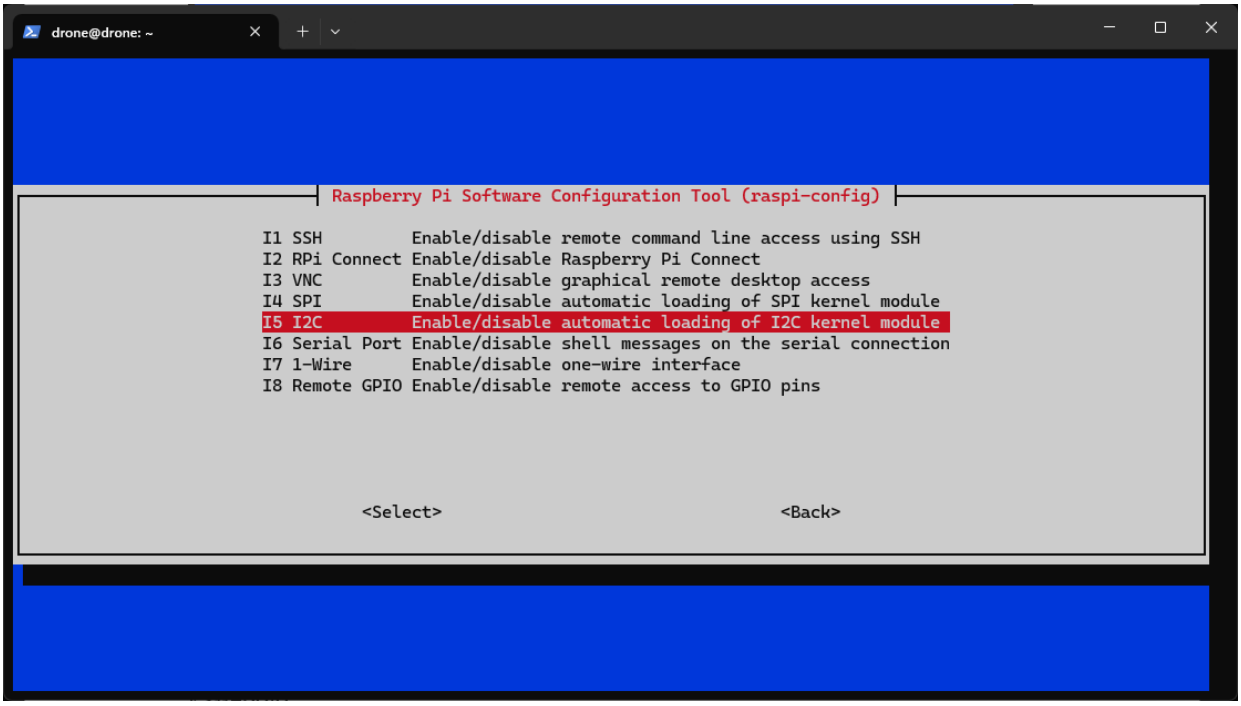

Рисунок 4.12 – Розділ Interface Options утиліти raspi-config

Але для забезпечення коректної роботи камери необхідно також додатково відредагувати файл конфігурації /boot/firmware/config.txt (рис. 4.13). Цей файл [8] містить параметри завантаження та налаштування апаратних компонентів Raspberry Pi, зокрема налаштування камери (лістинг 4.2).

```
dtparam=i2c_arm=on
dtparam=spi=on
dtoverlay=w1-gpio
enable_uart=1
start_x=1
camera_auto_detect=1
# dtoverlay=ov5647
```

Лістинг 4.2 – Фрагмент файлу конфігурації /boot/firmware/config.txt

У лістингу 4.2 зазначено параметр camera_auto_detect=1 [8], що дозволяє системі автоматично завантажувати драйвери для камери OV5647 при її

виявленні. Також, наступні параметри дозволяють підтримку інтерфейсів I2C і SPI: dtparam=i2c_arm=on та dtparam=spi=on [8]. Дані параметри автоматично налаштовані завдяки утиліті raspi-config, використання якої описано у даному підрозділі вище. Рядок # dtoverlay=ov5647 закоментовано, оскільки для камери OV5647 зазвичай достатньо автоматичного розпізнавання, але при необхідності є можливість зазначити конкретний пристрій камери.

Наступним кроком є перезавантаження Raspberry Pi Zero 2 W, для того, щоб система застосувала всі зміни, завантажила драйвери та підготувала апаратне забезпечення до роботи з камерою. Перезавантаження виконується командою sudo reboot.

Після перезавантаження пристрою, для перевірки коректності підключення камери використовуються наступні команди.

1) i2cdetect -y 10. Ця команда сканує I2C-шину з номером 10 (на Raspberry Pi шина для камери зазвичай має ідентифікатор 10) і виводить список адрес підключених пристроїв (рис 4.13). Якщо камера OV5647 підключена правильно, її адреса (наприклад, 0x36) з'явиться в таблиці у вигляді символів «UU». Це підтверджує фізичне з'єднання камери з платою.

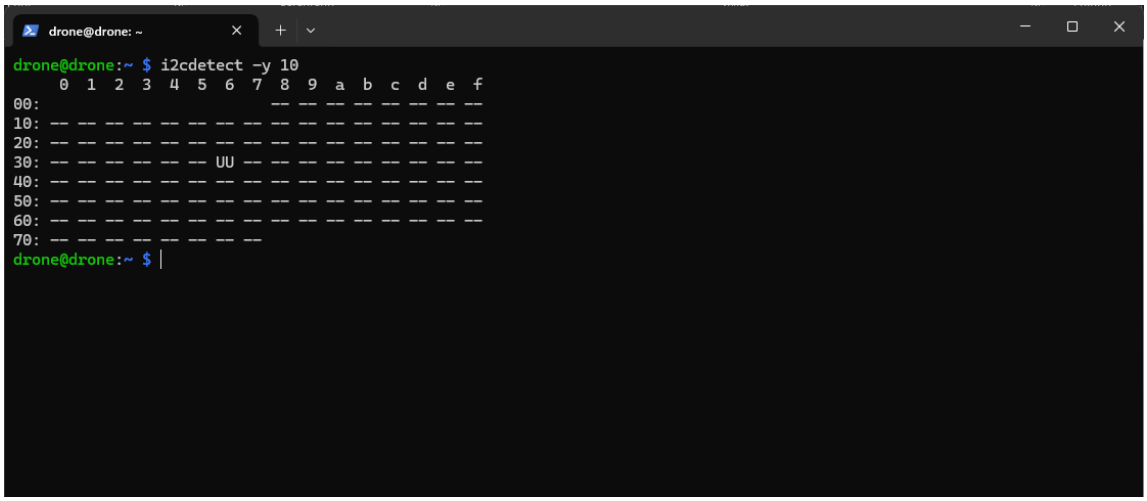

Рисунок 4.13 – Виконання команди i2cdetect -y 10

2) Dmesg | grep -i ov5647. Команда dmesg виводить системні повідомлення ядра, а grep -i ov5647 фільтрує їх, шукаючи записи, пов’язані з камерою OV5647 (опція -i ігнорує регістр символів). Якщо драйвери камери завантажено успішно, у виводі з’являться повідомлення про ініціалізацію OV5647, наприклад, про виявлення модуля або його реєстрацію в системі (рис. 4.14).

Для остаточної перевірки функціональності камери виконується команда libcamera-still -o test.jpg. Дана команда захоплює одне зображення з камери та зберігає його у файл test.jpg (рис. 4.15). Якщо команда виконується успішно і файл містить зображення, це означає, що камера OV5647 повністю налаштована і готова до роботи.

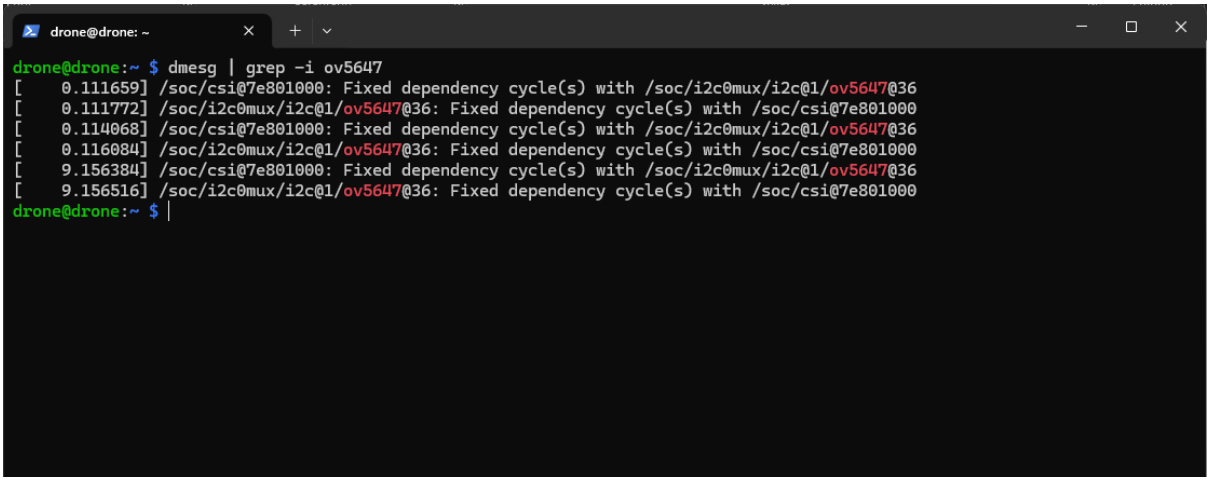

Рисунок 4.14 – Виконання команди dmesg | grep -i ov5647

Після захоплення зображення з камери (рис. 4.16) необхідно скопіювати файл для перегляду на персональному комп’ютері. Для цього використовується команда scp (лістинг 4.3), яка виконується у терміналі PowerShell з наступними параметрами.

– Drone – ім’я користувача на Raspberry Pi.
– 192.168.0.183 – IP-адреса пристрою.

– /home/drone/test.jpg – шлях до файлу на Raspberry Pi (рис. 4.17).

– D:\UNI\Diploma\Drone\ - місце збереження.

```
scp drone@192.168.0.183:/home/drone/test.jpg D:\UNI\Diploma\Drone\
```

Лістинг 4.3 – Команда scp

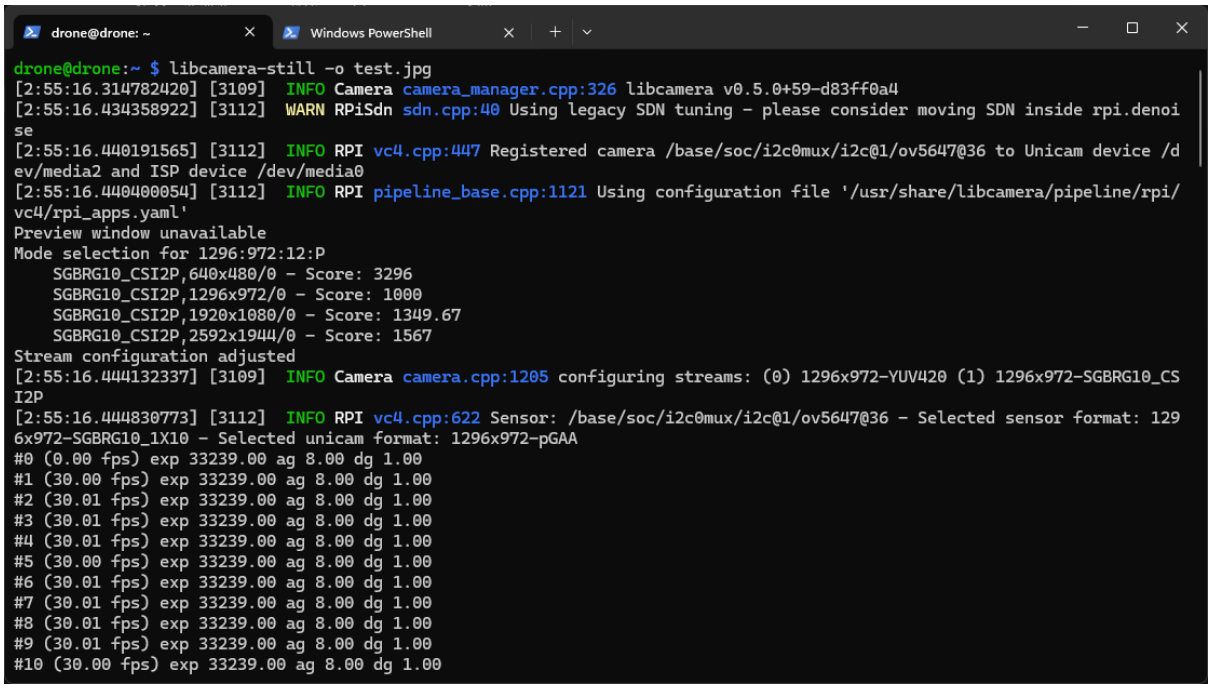

Рисунок 4.15 – Виконання команди libcamera-still -o test.jpg

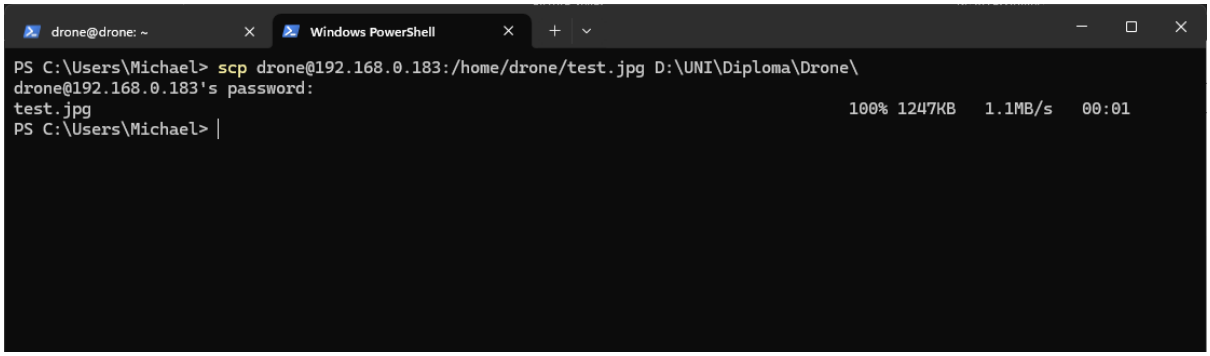

Рисунок 4.16 – Копіювання тестового зображення за допомогою команди scp

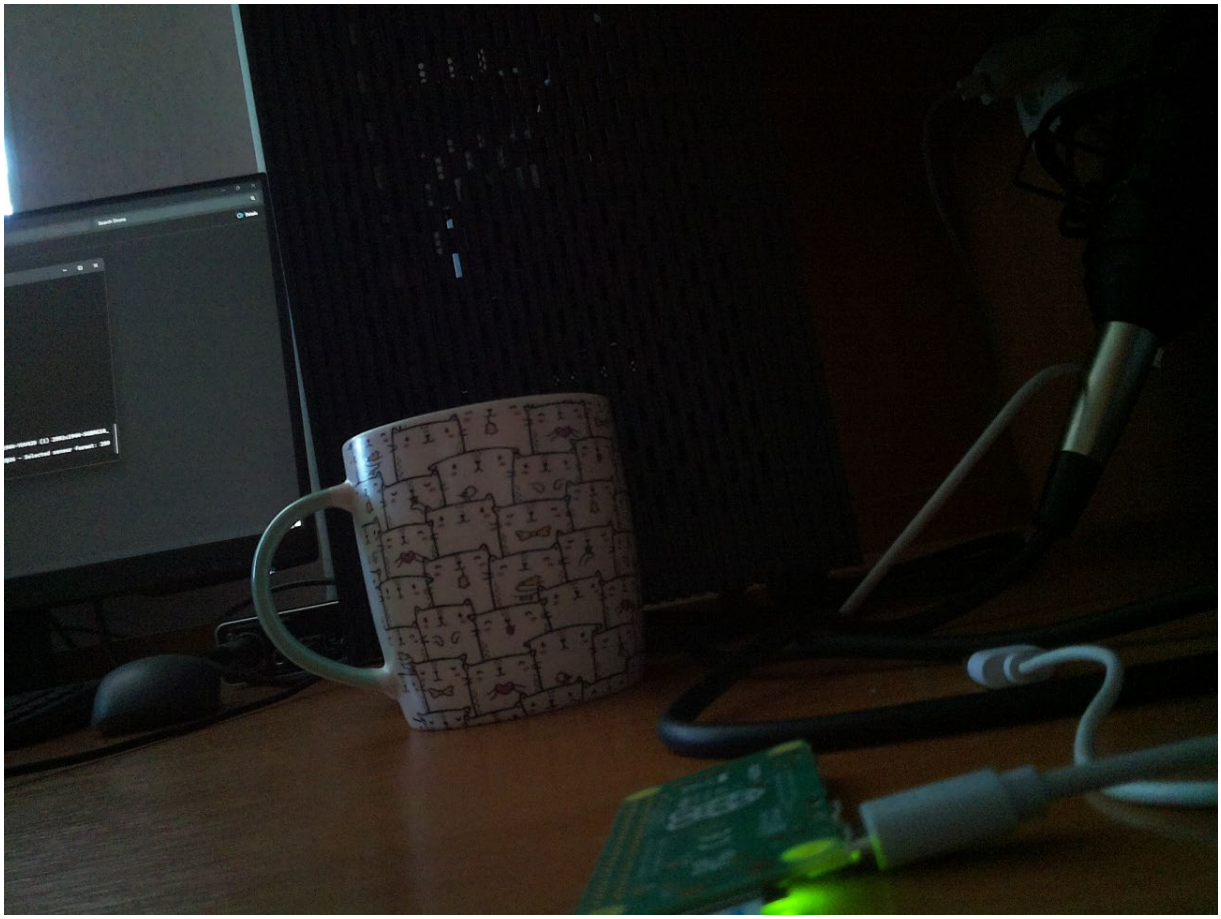

Рисунок 4.17 – Тестове зображення test.jpg

Виконання описаних кроків забезпечує коректне налаштування камери OV5647 для захоплення відео. Після цього система готова до компіляції та запуску програм, які використовуватимуть відеопоток із камери. Цей процес є основою для подальшої розробки й тестування на платформі Raspberry Pi.

5 ПРОГРАМНА РЕАЛІЗАЦІЯ

У цьому розділі описано програмну реалізацію комунікаційної підсистеми ройового комплексу автономних безпілотних літальних апаратів на базі платформи Raspberry Pi Zero 2 W, компіляцію програмного коду, опис основних компонентів-мікросервісів системи (GStreamer для передачі відеопотоків і Fast DDS для обміну телеметричними даними) а також тестування системи відповідно до завдань, визначених у підрозділі 1.3.

5.1 Опис реалізації передачі телеметричних даних

Для визначення структури даних, які передаються між учасниками системи (дроном і «маткою» рою), використовується IDL. Він дозволяє описати структури даних у нейтральному до мови програмування форматі, що спрощує їхню серіалізацію та десеріалізацію при обміні через Fast DDS. У даному контексті, файл telemetry.idl (лістинг 5.1) містить опис модуля swarm, який включає структуру Telemetry.

```
module swarm {
    struct Vector3 {
        float x;  float y; float z; };
    struct Telemetry {
        string drone_id;  string name;
        int64 timestamp;
        Vector3 position;  Vector3 velocity;
        float battery_level;
        float temperature;
        boolean camera_status;
        string motors_status;
        float cpu_usage;
        float memory_usage;
        string firmware_version;    };};
```

Лістинг 5.1 – Зміст файлу опису телеметричних даних telemetry.idl

Ця структура охоплює всі необхідні поля для телеметричних даних дрону, такі як ідентифікатор дрону, мітка часу, позиція, швидкість, рівень заряду батареї, температура, статус камери, статус двигунів, використання CPU та пам’яті, а також версія прошивки.

5.1.1 Опис реалізації програми для дрону

Код drone_telemetry.cpp (додаток В) є складовою частиною системи обміну телеметричними даними в ройовому комплексі автономних безпілотних літальних апаратів. Його основна функція полягає у генерації телеметричних даних дрону та їх відправленні до «матки» рою через технологію Fast DDS. Нижче наведено покроковий опис роботи коду.

Для реалізації функціоналу в коді використовуються наступні бібліотеки.

– Бібліотеки Fast DDS (<fastdds/dds/...>): забезпечують створення учасників домену, тем, видавців та записувачів даних, а також налаштування якості обслуговування (QoS).

– TelemetryPubSubTypes.h та telemetry.h: згенерований файл та визначення структури Telemetry для роботи з телеметричними даними.

– Стандартні бібліотеки C++ [9] (<chrono>, <thread>, <iostream>, <random>) для роботи з часом, потоками, виводом повідомлень та генерацією випадкових чисел.

Учасник домену (DomainParticipant) є ключовим об’єктом у Fast DDS, який об’єднує всі інші компоненти в межах одного домену. У коді створюється учасник з ідентифікатором 0 та стандартними налаштуваннями QoS. У разі невдачі створення учасника програма завершується з повідомленням про помилку. Для передачі телеметричних даних тип Telemetry реєструється в учаснику домену через клас TelemetryPubSubType. Записувач даних (DataWriter) дозволяє відправляти екземпляри типу Telemetry через задану тему. Його

створення відбувається з використанням стандартного QoS. Це дозволяє системі DDS розпізнавати структуру даних.

Тема (Topic) визначає тип і назву даних, що передаються. У коді створюється тема з назвою «telemetry_topic» та типом «swarm::Telemetry». Якщо створення теми не вдається, програма завершується з помилкою. Видавець (Publisher) відповідає за відправлення даних у систему DDS. Він створюється з налаштуваннями за замовчуванням. Записувач даних (DataWriter) дозволяє відправляти екземпляри типу Telemetry через задану тему. Його створення відбувається з використанням стандартного QoS.

Для симуляції телеметричних даних використовується генератор випадкових чисел («std::mt19937») [9] з різними розподілами.

- Позиція: від 0.0 до 100.0 (x, y, z).
- Швидкість: від -5.0 до 5.0 (x, y, z).
- Рівень заряду батареї: від 0.0% до 100.0%.
- Температура: від 20.0°C до 40.0°C.
- Використання CPU та пам’яті: від 0.0% до 100.0%.
- Статус камери: випадкове булеве значення (true/false).
- Статус двигунів: «OK» або «Warning» залежно від випадкового значення.

У основному циклі роботи кожну секунду генеруються та відправляються нові телеметричні дані.

- Мітка часу фіксує поточний момент у мілісекундах.
- Усі параметри заповнюються випадковими значеннями в межах заданих діапазонів.
- Фіксована версія прошивки встановлюється як «v1.0.0».
- Дані відправляються через writer->write(&telemetry), після чого виводиться повідомлення про успішну відправку.

Код drone_telemetry.cpp реалізує базову функціональність для генерації та передачі телеметричних даних у ройовій системі. Завдяки Fast DDS

забезпечується швидкий і надійний обмін даними, а використання випадкових значень дозволяє симулювати реальну поведінку дрону для тестування.

5.1.2 Опис реалізації програми для «матки» рою

Код womb_telemetry.cpp (додаток В) є частиною системи обміну телеметричними даними в ройовому комплексі автономних безпілотних літальних апаратів (БПЛА). Його основне призначення – приймання та обробка телеметричних даних від дрону за допомогою технології Fast DDS. Нижче наведено детальний покроковий опис роботи коду.

Для реалізації функціоналу код використовує наступні бібліотеки.

- Бібліотеки Fast DDS (<fastdds/dds/...>) забезпечують створення учасника домену, тем, підписників і читачів даних, а також налаштування якості обслуговування (QoS).
- «telemetryPubSubTypes.h»: згенерований файл, який визначає типи даних для телеметрії.
- Стандартні бібліотеки C++ (<iostream>, <thread>, <chrono>): використовуються для виведення повідомлень, роботи з потоками та управління часом.

Клас TelemetryListener успадковується від DataReaderListener і відповідає за обробку вхідних телеметричних даних. Основний метод on_data_available викликається щоразу, коли надходять нові дані. Цей метод отримує дані через об’єкт DataReader, перевіряє їхню валідність і виводить ключові параметри: ідентифікатор дрону, мітку часу, координати позиції, рівень заряду батареї та використання процесора.

Учасник домену (DomainParticipant) — це центральний об’єкт у Fast DDS, який об’єднує всі компоненти в межах одного домену. У коді він створюється з ідентифікатором 0 та стандартними налаштуваннями QoS. Якщо створення

учасника не вдається, програма завершується з повідомленням про помилку.

Тип даних Telemetry реєструється в учаснику домену через об’єкт TypeSupport, який базується на класі TelemetryPubSubType. Це необхідно для того, щоб система DDS могла розпізнавати структуру телеметричних даних. Тема (Topic) визначає тип і назву даних, які передаються в системі. У коді створюється тема з назвою «telemetry_topic» і типом «swarm::Telemetry».

Підписник (Subscriber) відповідає за приймання даних у системі DDS. Він створюється з налаштуваннями QoS за замовчуванням. Якщо створення підписника не вдається, програма видає повідомлення про помилку та завершується. Читач даних (DataReader) забезпечує доступ до телеметричних даних із заданої теми. Він створюється з використанням стандартного QoS та об’єкта слухача (TelemetryListener).

Після ініціалізації всіх компонентів програма виводить повідомлення про готовність до приймання даних і переходить у основний цикл. Обробка даних відбувається асинхронно в методі on_data_available слухача, коли надходять нові телеметричні дані. Файл womb_telemetry.cpp реалізує базову функціональність для приймання та обробки телеметричних даних у ройовій системі БПЛА. Використання технології Fast DDS забезпечує швидкий і надійний обмін даними, а клас TelemetryListener дозволяє ефективно обробляти отриману інформацію, виводячи її у зрозумілому форматі.

5.1.3 Генерація коду з IDL-файлу

Для генерації C++ коду з файлу IDL використовується інструмент fastddsgen, який є частиною пакету Fast DDS. Цей інструмент автоматично створює C++ класи, що відповідають описаним у IDL структурам, а також необхідні для серіалізації та десеріалізації даних (лістинг 5.2). Виконання цієї команди призводить до створення кількох файлів, зокрема telemetry.cxx і

telemetryPubSubTypes.cxx. Ці файли містять C++ код, який реалізує структуру Telemetry та забезпечує її інтеграцію з Fast DDS для подальшого використання у програмах дрону і «матки» рою.

```
fastddsgen telemetry.idl
```

Лістинг 5.2 – Команда генерації C++ коду Fast DDS

Генерація коду з IDL є важливим етапом, оскільки вона забезпечує типобезпечність і ефективну серіалізацію даних, що передаються між компонентами системи. Використання згенерованого коду спрощує розробку, усуваючи необхідність вручну створювати код для серіалізації, що може бути складним і схильним до помилок.

5.1.4 Компіляція C++ коду

Програмна реалізація системи передбачає компіляцію C++ коду, який забезпечує обмін телеметричними даними через технологію Fast DDS. Цей процес складається з двох основних етапів: генерації C++ коду з файлу IDL та компіляції програм для дрону і «матки» рою. Нижче наведено детальний опис кожного з етапів, що забезпечує повне розуміння процесу створення виконуваних файлів для системи.

Після генерації C++ коду з IDL-файлу наступним етапом є компіляція програм для дрону (лістинг 5.3) і «матки» рою (лістинг 5.4). Ці програми, написані на мові C++, використовують згенерований код для роботи з Fast DDS, забезпечуючи обмін телеметричними даними.

Компіляція виконується за допомогою компілятора g++. Для забезпечення підтримки сучасних можливостей мови C++ застосовується стандарт C++17 [9],

що дозволяє використовувати новітні конструкції та покращує читабельність і якість коду.

```
g++ -o womb_telemetry womb_telemetry.cpp telemetry.cxx
telemetryPubSubTypes.cxx -std=c++17 -lfastrtps -lfastcdr
```

Лістинг 5.3 – Компіляція програми для «матки» рою

```
g++ -o drone_telemetry drone_telemetry.cpp telemetry.cxx
telemetryPubSubTypes.cxx -std=c++17 -lfastrtps -lfastcdr
```

Лістинг 5.4 – Компіляція програми для дрону

Програма для «матки» рою womb_telemetry.cpp (додаток В) і програма для дрону drone_telemetry.cpp (додаток В) компілюються окремо, але обидві потребують включення згенерованих файлів telemetry.cxx і telemetryPubSubTypes.cxx, а також бібліотек Fast DDS (параметри -lfastrtps і -lfastcdr). Ці бібліотеки забезпечують базову функціональність для розподіленої комунікації та серіалізації даних.

У командах компіляції є наступні параметри.

- Параметр -o вказує ім’я вихідного виконуваного файлу (womb_telemetry для «матки» рою і drone_telemetry для дрону).
- Параметр -std=c++17 активує підтримку стандарту C++17 [9].
- Параметри -lfastrtps і -lfastcdr підключають бібліотеки Fast DDS і Fast CDR (сервіс серіалізації), необхідні для роботи з розподіленими даними.

Виконання зазначених команд призводить до створення виконуваних файлів womb_telemetry і drone_telemetry, які можна запускати на відповідних пристроях («матці» і дроні) для забезпечення обміну телеметричними даними.

Використання стандарту C++17 під час компіляції дозволяє застосовувати сучасні можливості мови, такі як std::string_view, std::optional та структуроване

прив’язування, що покращує якість коду. Fast DDS, як реалізація стандарту DDS, забезпечує швидкий і надійний обмін даними в розподілених системах, що є ключовим для роботи ройових комплексів.

5.2 Опис реалізації передачі відеопотоків

Система складається з двох ключових компонентів: передачі відеопотоків за допомогою GStreamer і обміну телеметричними даними через Fast DDS. Для паралельного запуску передачі та отримання телеметрії та відеопотоків застосовуються скрипти drone_start.sh (додаток Г) та womb_start.sh (ДОДАТОК) відповідно.

Для передачі відеопотоків з дрону на «матку» рою використовується фреймворк GStreamer, який забезпечує гнучке створення пайплайнів для обробки мультимедійних даних. GStreamer обрано завдяки його підтримці апаратного прискорення на платформах, таких як Raspberry Pi, а також можливості точного налаштування параметрів для оптимізації продуктивності.

Скрипт drone_start.sh (додаток Г) запускає пайплайн Gstreamer [4], який захоплює відео з камери дрону, кодує його у формат H.264 і передає на «матку» рою (лістинг 5.5).

```
gst-launch-1.0 libcamerasrc ! \
    video/x-raw,width=640,height=480,framerate=10/1 ! \
    videoconvert ! queue ! \
    x264enc bitrate=1000 key-int-max=10 ! \
    h264parse ! queue ! \
    rtph264pay ! queue ! \
    udpsink host=192.168.0.45 port=5000
```

Лістинг 5.5 – Пайплайн Gstreamer для передачі відеопотоків

В даному пайплайні використовуються наступні параметри пайплайну Gstreamer[4].

- Libcamerasrc – захоплення відео з камери Raspberry Pi.
- Video/x-raw – налаштування роздільної здатності 640x480 та частоти 10 кадрів/с.
- X264enc – кодування у формат H.264 з бітрейтом 1000 кбіт/с.
- Rtph264pay – пакетування у RTP.
- Udpsink – передача на IP-адресу матки (192.168.0.45:5000).

Скрипт womb_start.sh (ДОДАТОК Г) запускає пайплайн Gstreamer [4], який отримує відеопоток з дрону, розкодовує його та кожну хвилину зберігає окремий файл відео (лістинг 5.6).

```
gst-launch-1.0 udpsrc port=5000 ! \
    application/x-rtp, encoding-name=H264 ! \
    rtph264depay ! \
    h264parse ! \
    queue ! \
    splitmuxsink location=/home/womb/Videos/video_%04d.mp4 \
    max-size-time=60000000000 \
    muxer=mp4mux
```

Лістинг 5.6 – Пайплайн Gstreamer для отримання відеопотоків

В даному пайплайні використовуються наступні параметри пайплайну Gstreamer [4].

- Udpsrc – отримання UDP-пакетів на порту 5000.
- Rtph264depay – витягнення H.264 з RTP.
- splitmuxsink – збереження відео у 1-хвилинні MP4-файли.

6 ТЕСТУВАННЯ СИСТЕМИ

Тестування комунікаційної підсистеми ройового комплексу автономних безпілотних літальних апаратів є критичним етапом для підтвердження її працездатності, надійності та відповідності вимогам, визначеним у підрозділі 1.3 пояснювальної записки. На поточному етапі розробки системи реалізована паралельна передача телеметричних даних через Fast DDS та відеопотоків через GStreamer від дрону до «матки» рою, як описано в підрозділах 6.1 і 6.2. Цей розділ описує базове тестування функціональності передачі телеметричних даних і відеопотоків, включаючи методику тестування, сценарії, результати та аналіз відповідності вимогам. Тестування проводилося на апаратній платформі Raspberry Pi Zero 2 W з камерою OV5647, використовуючи операційну систему Raspberry Pi OS, як зазначено в розділах 3 і 4.

6.1 Мета та завдання тестування

Метою тестування є перевірка коректності роботи комунікаційної підсистеми в частині передачі телеметричних даних і відеопотоків між дроном і «маткою» рою, а також оцінка її продуктивності в умовах, що імітують реальну експлуатацію. Завдання тестування включають наступне.

1) Перевірка стабільності та коректності передачі телеметричних даних (ідентифікатор дрону, мітка часу, позиція, швидкість, рівень заряду батареї, температура, статус камери, статус двигунів, використання CPU і пам’яті, версія прошивки) від дрону до «матки» через Fast DDS.

2) Перевірка якості та безперервності передачі відеопотоків від камери OV5647 дрону до «матки» через GStreamer, а також коректності збереження відеофайлів у форматі MP4.

3) Оцінка затримки (латентності) передачі даних і відеопотоків.

4) Перевірка стійкості системи до короткочасних перебоїв у зв’язку (моделювання відключення Wi-Fi).

5) Аналіз відповідності результатів тестування вимогам, визначеним у підрозділі 1.3, зокрема щодо реального часу, надійності та ефективності використання ресурсів.

6.2 Методика тестування

Тестування проводилося на двох платах Raspberry Pi Zero 2 W, налаштованих як дрон і «матка» рою, відповідно до конфігурації, описаної в підрозділі 3.5. Обидві плати були підключені до однієї Wi-Fi мережі (2.4 ГГц IEEE 802.11b/g/n) з IP-адресами 192.168.0.183 для дрону та 192.168.0.45 для «матки». Камера OV5647 на дроні була налаштована через CSI-інтерфейс, як описано в підрозділі 5.1.3. Використовувалися програми та скрипти, описані в розділі 6.

- Для передачі телеметрії – програми drone_telemetry.cpp (додаток В) і womb_telemetry.cpp (додаток В) для генерації, відправлення та приймання телеметричних даних через Fast DDS.
- Для передачі відеопотоків – скрипти drone_start.sh (додаток Г) і womb_start.sh (додаток Г) для захоплення, кодування, передачі та збереження відеопотоків через GStreamer.

Тестування включало виконання кількох сценаріїв, які імітують базові операції ройової системи. Для оцінки продуктивності використовувалися наступні інструменти.

- Логування – виведення телеметричних даних через std::cout у програмі womb_telemetry.cpp для перевірки їхньої коректності.
- GStreamer-логи – аналіз логів виконання пайплайнів GStreamer для оцінки стабільності відеопотоків.

– Таймери – вбудовані мітки часу в телеметричних даних для вимірювання затримки.

– Перевірка файлів – аналіз збережених MP4-файлів на «матці» за допомогою медіаплеєра для оцінки якості відео.

6.3 Перевірка передачі телеметричних даних

Програма drone_telemetry.cpp генерує телеметричні дані з частотою 1 Гц (одне повідомлення за секунду) і відправляє їх на «матку» через Fast DDS на тему telemetry_topic. Програма womb_telemetry.cpp приймає дані та виводить їх у консоль (рис. 6.1).

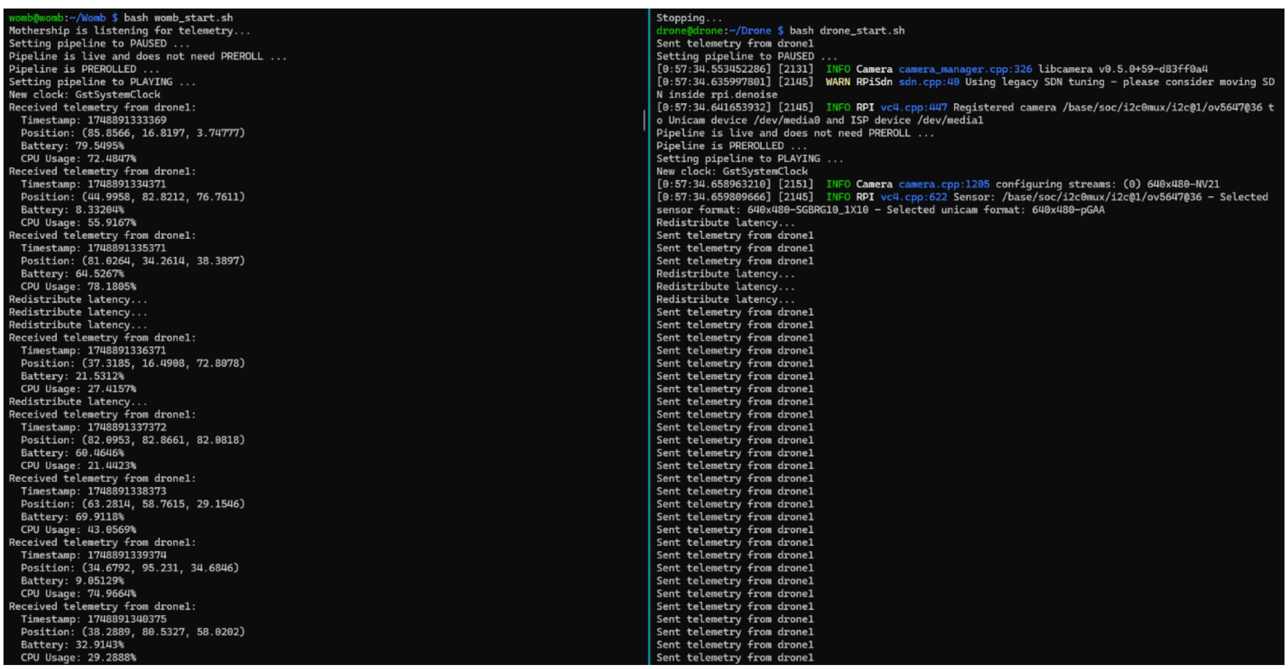

Рисунок 6.1 – Передача телеметричних даних

Передача телеметричних даних через Fast DDS є стабільною, з низькою латентністю та без втрат даних, що відповідає вимогам реального часу, зазначеним у підрозділі 1.3.

6.4 Перевірка передачі відеопотоків

Скрипт drone_start.sh запускає GStreamer-пайплайн для захоплення відео з камери OV5647 (роздільна здатність 640x480, 10 кадрів/с, кодек H.264) і передачі його через UDP на «матку» (IP 192.168.0.45, порт 5000). Скрипт womb_start.sh приймає відеопотік і зберігає його у вигляді 1-хвилинних MP4-файлів у директорії /home/womb/Videos/.

Відеопотік успішно передавався протягом 5 хвилин, створено 5 MP4-файлів тривалістю 1 хвилина кожен (рис 6.2). Перегляд властивостей файлів (рис. 6.3) підтвердив їхню коректність. Роздільна здатність 640x480, частота 10 кадрів/с, а перегляд у медіапрогравачі, відсутність помітних артефактів чи пропущених кадрів (рис. 6.4).

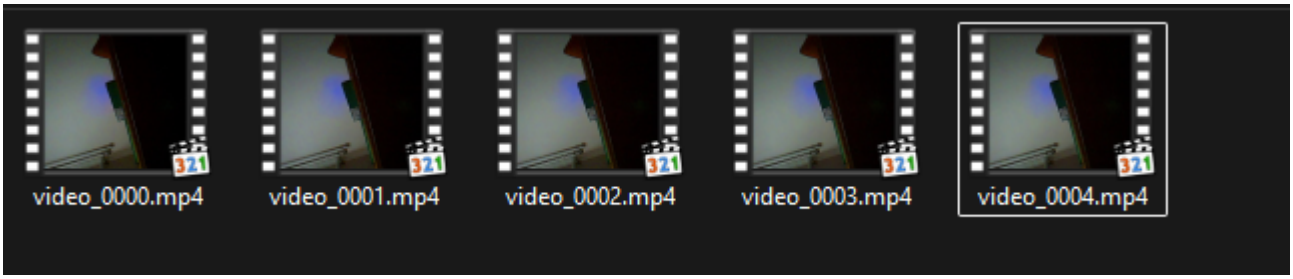

Рисунок 6.2 – Збережені файли відеопотоку

Передача відеопотоків через GStreamer є стабільною, з низькою латентністю та високою якістю збереженого відео, що відповідає вимогам реального часу та ефективності, зазначеним у підрозділі 1.3.

Базове тестування комунікаційної підсистеми ройового комплексу підтвердило її працездатність у частині передачі телеметричних даних через Fast DDS і відеопотоків через GStreamer. Система забезпечує стабільність і ефективне використання ресурсів, що відповідає вимогам, визначеним у підрозділі 1.3.

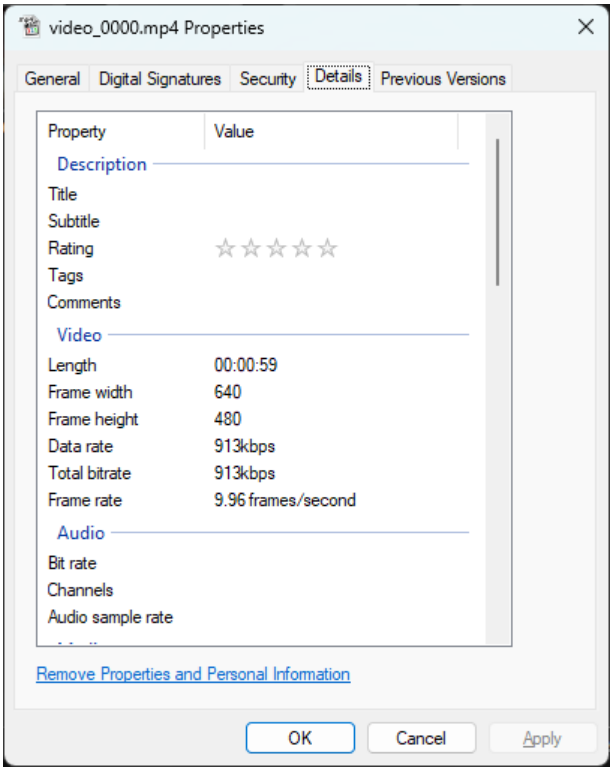

Рисунок 6.3 – Перегляд властивостей відеофайлів

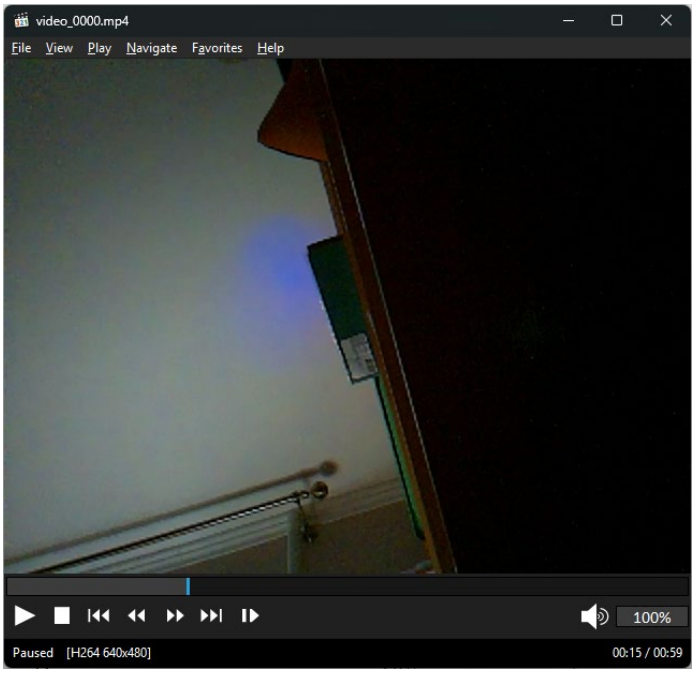

Рисунок 6.4 – Перегляд кадру відеофайлу

Стійкість до перебоїв у зв’язку є достатньою для базового функціонування, але потребує вдосконалення шляхом впровадження буферизації та оптимізації GStreamer-пайплайнів. Подальше тестування має включати сценарії з кількома дронами, моделювання перевантаження мережі та інтеграцію з мікросервісами, для забезпечення масштабованості та повної відповідності вимогам ройової системи.

Базове тестування комунікаційної підсистеми ройового комплексу автономних безпілотних літальних апаратів підтвердило її працездатність у частині передачі телеметричних даних через Fast DDS та відеопотоків через GStreamer. Система була протестована на платформах Raspberry Pi Zero 2 W з камерою OV5647, що забезпечило реальні умови для оцінки її функціональності. Результати демонструють стабільність, низьку латентність і ефективне використання ресурсів, що відповідає ключовим вимогам реального часу, надійності та енергоефективності, визначеним у підрозділі 1.3 пояснювальної записки. Зокрема, передача телеметричних даних відбувалася з частотою 1 Гц без втрат, а відеопотоки передавалися з роздільною здатністю 640x480 при 10 кадрах/с, що є достатнім для базових операцій моніторингу в реальному часі.

ВИСНОВКИ

У даній кваліфікаційній роботі виконано розробку та впровадження комунікаційної підсистеми для роєвого комплексу автономних безпілотних літальних апаратів, що забезпечує ефективний, швидкий і надійний обмін даними між дронами. Робота охопила повний цикл створення системи – від теоретичного аналізу предметної області до практичної реалізації та тестування.

Усі поставлені завдання, визначені у вступі та розділі 1 успішно виконано. Проведено всебічний огляд особливостей управління роєм дронів у надзвичайних ситуаціях, що дало змогу сформулювати ключові вимоги до комунікаційної підсистеми. Зокрема, враховано потребу в надійності, швидкості передачі даних, стійкості до перебоїв у зв’язку та підтримці децентралізованої взаємодії. Обрано та обґрунтовано вибір технології Fast DDS для передачі телеметричних даних і фреймворк GStreamer для відеопотоків. Ці рішення забезпечують високу продуктивність, низьку латентність і оптимальне використання ресурсів. Створено асинхронний обмін даними за патерном publish-subscribe у Fast DDS, що забезпечує гнучкість і масштабованість системи. Налаштовано GStreamer для захоплення, кодування та передачі відеопотоків із камери OV5647 у реальному часі з роздільною здатністю 640x480 та частотою 10 кадрів/с. Розроблена комунікаційна підсистема базується на апаратному забезпеченні Raspberry Pi Zero 2 W з камерою OV5647, що забезпечує компактність, енергоефективність і достатню обчислювальну потужність. Програмне забезпечення, побудоване на операційній системі Raspberry Pi OS, інтегрує Fast DDS для телеметрії та GStreamer для відео, що дозволяє системі відповідати вимогам реального часу, надійності та безпеки.

За результатами кваліфікаційної роботи опубліковані тези доповіді на двадцять другій всеукраїнській конференції студентів і молодих науковців «Інформатика, інформаційні системі та технології» [2].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adoni WYH, Fareedh JS, Lorenz S, Gloaguen R, Madriz Y, Singh A, Kühne TD. Intelligent Swarm: Concept, Design and Validation of Self-Organized UAVs Based on Leader–Followers Paradigm for Autonomous Mission Planning. Drones. 2024; 8(10):575. DOI: https://doi.org/10.3390/drones8100575
2. Ісаєв М. А., Шпінарева І. М. Система управління ройовим комплексом. Розробка протоколів обміну даними та їх реалізація на основі мікросервісної архітектури // Інформатика, інформаційні системи та технології: тези доповідей XXII Всеукраїнської коференції студентів і молодих науковців. - Одеса, 25 квітня 2025 р. - Одеса, 2025. - С. 196-197.
3. Raspberry Pi Zero 2 W product brief [Електронний ресурс] – Режим доступу: https://datasheets.raspberrypi.com/rpizero2/raspberry-pi-zero-2-w-product-brief.pdf
4. Raspberry Pi OS [Електронний ресурс] – Режим доступу: https://www.raspberrypi.com/software/
5. GStreamer: gst-launch tool documentation [Електронний ресурс] – Режим доступу: https://gstreamer.freedesktop.org/documentation/tools/gst-launch.html
6. Gstreamer: Installing on Linux [Електронний ресурс] – Режим доступу: https://gstreamer.freedesktop.org/documentation/installing/on-linux.html
7. eProsima Fast DDS Documentation [Електронний ресурс] – Режим доступу: https://fast-dds.docs.eprosima.com/en/latest/
8. Raspberry Pi: config.txt [Електронний ресурс] – Режим доступу: https://www.raspberrypi.com/documentation/computers/config_txt.html
9. C++ Standard Library [Електронний ресурс] – Режим доступу: https://en.cppreference.com/w/cpp/standard_library.html

ДОДАТОК А

Порівняння апаратних платформ

Таблиця А.1 – Порівняння апаратних платформ

Платформа	Обчислювальна потужність	Операційна система	Оперативна пам’ять	Споживана потужність	Розміри	Орієнтовна ціна	Підтримка спільноти
Arduino Uno / Mega	8-/16-/32-MHz AVR / ARM Cortex-M (Mega 2560)	Нативна (без ОС)	2–8 КБ	≈ 50–100 мВт	Дуже компактні	$10–$40	Дуже широка
BeagleBone Black	1 × 1 GHz ARM Cortex-A8	Linux (Angstrom, Debian)	512 МБ	≈ 500–700 мВт	86×53 мм	$50–$75	Широка
Jetson Nano / Xavier NX	4 × 1.43 GHz ARM A57 + 128–384 ядер CUDA	Linux (Ubuntu)	4–8 ГБ	≈ 5–10 Вт	69×45 мм	$90–$400	Інтенсивна (NVIDIA)
Raspberry Pi Zero 2 W	*4 × 1 GHz ARM Cortex-A53*	*Linux (Raspbian)*	*512 МБ*	*≈ 0.3–1 Вт*	*65×30 мм*	*$15–$20*	*Дуже широка*

ДОДАТОК Б

Порівняння фреймворків Gstreamer та Ffmpeg

Таблиця Б.1 - Порівняння фреймворків Gstreamer та Ffmpeg

Критерій	GStreamer	FFmpeg
Апаратне прискорення	Повна підтримка апаратного кодування H.264 через OpenMAX IL / MMAL, що знижує навантаження CPU до < 30 % при 1080p@30fps.	Є підтримка кодеків через h264_omx, проте менш стабільна: більші затримки й іноді падіння кадрів.
Латентність	Мінімальна буферизація (pipelines із відключеними буферами), забезпечує реальну затримку < 100 мс при передачі RTP.	За замовчуванням більша затримка через внутрішню буферизацію; оптимізація до < 150 мс потребує ручного тонкого налаштування.
Гнучкість пайплайнів	Потужний опис потоків через gst-launch-1.0 або API: легко додавати, видаляти та змінювати елементи (детектори кадрів, обробники метаданих, мультиплексори).	Командний рядок ffmpeg підтримує складні фільтри, але побудова динамічних пайплайнів у програмі складніша – потрібна робота з C-API або додаткові бібліотеки.

Продовження таблиці Б.1

Критерій	GStreamer	FFmpeg
Підтримка мережевих протоколів	Нативна підтримка RTP/RTSP, SRT, UDP, TCP; плагіни для мультикасту й FEC.	Підтримка RTP/RTSP/RTMP; мультиплексування потоків реалізується через складніші скрипти, менше плагінів для FEC.
Інтеграція з DDS	Через плагін appsink можна напряму передавати бінарні дані відео в Fast DDS, синхронізувати часові мітки.	Потрібно реалізовувати власні колбек-функції в C/C++ для вивантаження кожного фрейму в DDS-медіатор.
Спільнота та документація	Активний розвиток у спільноті Raspberry Pi, велика кількість готових прикладів.	Розгалужена, але приклади оптимізації апаратного прискорення на RPi зустрічаються рідше.
Розмір та залежності	~30 МБ плагінів; встановлюється разом із libgstreamer1.0-*.	~50 МБ бібліотек; залежить від libx264, libfdk-aac, libomx-* тощо.

ДОДАТОК В

Програми передачі та отримання телеметричних даних

```
#include <fastdds/dds/domain/DomainParticipantFactory.hpp>
#include <fastdds/dds/domain/DomainParticipant.hpp>
#include <fastdds/dds/topic/Topic.hpp>
#include <fastdds/dds/publisher/Publisher.hpp>
#include <fastdds/dds/publisher/DataWriter.hpp>
#include <fastdds/dds/publisher/qos/DataWriterQos.hpp>
#include "telemetryPubSubTypes.h"
#include "telemetry.h"
#include <chrono>
#include <thread>
#include <iostream>
#include <random>

using namespace eprosima::fastdds::dds;
using namespace swarm;

int main() {
    // Створення DomainParticipant
    DomainParticipant* participant =
DomainParticipantFactory::get_instance()->create_participant(0,
PARTICIPANT_QOS_DEFAULT);
    if (participant == nullptr) {
        std::cerr << "Failed to create participant" << std::endl;
        return 1;
    }

    // Реєстрація типу
    TypeSupport type(new TelemetryPubSubType());
    type.register_type(participant);

    // Створення Topic
    Topic* topic = participant->create_topic("telemetry_topic",
"swarm::Telemetry", TOPIC_QOS_DEFAULT);
    if (topic == nullptr) {
        std::cerr << "Failed to create topic" << std::endl;
        return 1;
    }
```

Лістинг В1 – Код програми drone_telemetry.cpp

```
 // Створення Publisher
    Publisher* publisher =
participant->create_publisher(PUBLISHER_QOS_DEFAULT);
    if (publisher == nullptr) {
        std::cerr << "Failed to create publisher" << std::endl;
        return 1;
    }

    // Створення DataWriter
    DataWriter* writer = publisher->create_datawriter(topic,
DATAWRITER_QOS_DEFAULT);
    if (writer == nullptr) {
        std::cerr << "Failed to create data writer" << std::endl;
        return 1;
    }

    // Ініціалізація генератора випадкових чисел
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> pos_dist(0.0, 100.0);    //
Позиція від 0 до 100

    std::uniform_real_distribution<> vel_dist(-5.0, 5.0);     //
Швидкість від -5 до 5
    std::uniform_real_distribution<> battery_dist(0.0, 100.0); //
Батарея від 0 до 100%
    std::uniform_real_distribution<> temp_dist(20.0, 40.0);    //
Температура від 20 до 40°C
    std::uniform_real_distribution<> usage_dist(0.0, 100.0);   //
Використання CPU/пам'яті від 0 до 100%
    std::uniform_int_distribution<> bool_dist(0, 1);           //
Для булевих значень (0 або 1)
    std::uniform_int_distribution<> status_dist(0, 1);         //
Для статусу двигунів (0 або 1)

    // Основний цикл
    swarm::Telemetry telemetry;
    telemetry.drone_id("drone1");

    while (true) {
        telemetry.timestamp(
            std::chrono::duration_cast<std::chrono::milliseconds>(
                std::chrono::system_clock::now().time_since_epoch()
            ).count()
        );
```

Лістинг В1, аркуш 2

```
        // Генерація випадкових значень для позиції
        telemetry.position().x(pos_dist(gen));
        telemetry.position().y(pos_dist(gen));
        telemetry.position().z(pos_dist(gen));

        // Генерація випадкових значень для швидкості
        telemetry.velocity().x(vel_dist(gen));
        telemetry.velocity().y(vel_dist(gen));
        telemetry.velocity().z(vel_dist(gen));

        // Генерація випадкових значень для інших параметрів
        telemetry.battery_level(battery_dist(gen));
        telemetry.temperature(temp_dist(gen));
        telemetry.camera_status(bool_dist(gen) == 1);
        telemetry.motors_status(status_dist(gen) == 0 ? "OK" :
"Warning");
        telemetry.cpu_usage(usage_dist(gen));
        telemetry.memory_usage(usage_dist(gen));
        telemetry.firmware_version("v1.0.0"); // Фіксоване значення

        writer->write(&telemetry);

        std::cout << "Sent telemetry from " << telemetry.drone_id()
<< std::endl;

        std::this_thread::sleep_for(std::chrono::seconds(1));
    }

    // Очистка
    participant->delete_contained_entities();
    DomainParticipantFactory::get_instance()->delete_participant(pa
rticipant);
    return 0;
}
```

Лістинг В1, аркуш 3

```
#include <fastdds/dds/domain/DomainParticipantFactory.hpp>
#include <fastdds/dds/domain/DomainParticipant.hpp>
#include <fastdds/dds/topic/Topic.hpp>
#include <fastdds/dds/subscriber/Subscriber.hpp>
#include <fastdds/dds/subscriber/DataReader.hpp>
#include <fastdds/dds/subscriber/DataReaderListener.hpp>
#include <fastdds/dds/subscriber/qos/DataReaderQos.hpp>
```

Лістинг В2 - Код програми womb_telemetry.cpp

```
#include "telemetryPubSubTypes.h"
#include <iostream>
#include <thread>
#include <chrono>

using namespace eprosima::fastdds::dds;
using namespace swarm;

class TelemetryListener : public DataReaderListener {
public:
    void on_data_available(DataReader* reader) override {
        Telemetry telemetry;
        SampleInfo info;

        if (reader->take_next_sample(&telemetry, &info) ==
ReturnCode_t::RETCODE_OK &&
            info.instance_state == ALIVE_INSTANCE_STATE)
        {
            std::cout << "Received telemetry from " <<
telemetry.drone_id() << ":\n";
            std::cout << "  Timestamp: " << telemetry.timestamp()
<< "\n";
            std::cout << "  Position: (" <<
telemetry.position().x() << ", " << telemetry.position().y() << ",
" << telemetry.position().z() << ")\n";
            std::cout << "  Battery: " << telemetry.battery_level()
<< "%\n";
            std::cout << "  CPU Usage: " << telemetry.cpu_usage()
<< "%\n";
        }
    }
};

int main() {
    DomainParticipant* participant =
DomainParticipantFactory::get_instance()->create_participant(0,
PARTICIPANT_QOS_DEFAULT);
    if (!participant) {
        std::cerr << "Failed to create participant\n";
        return 1;
    }

    TypeSupport type(new TelemetryPubSubType());
    type.register_type(participant);
```

Лістинг В2, аркуш 2

```
    Topic* topic = participant->create_topic("telemetry_topic",
"swarm::Telemetry", TOPIC_QOS_DEFAULT);
    if (!topic) {
        std::cerr << "Failed to create topic\n";
        return 1;
    }

    Subscriber* subscriber =
participant->create_subscriber(SUBSCRIBER_QOS_DEFAULT);
    if (!subscriber) {
        std::cerr << "Failed to create subscriber\n";
        return 1;
    }

    TelemetryListener listener;
    DataReader* reader = subscriber->create_datareader(topic,
DATAREADER_QOS_DEFAULT, &listener);
    if (!reader) {
        std::cerr << "Failed to create data reader\n";
        return 1;
    }

    std::cout << "Mothership is listening for telemetry...\n";
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(1));
    }


    participant->delete_contained_entities();
    DomainParticipantFactory::get_instance()->delete_participant(pa
rticipant);
    return 0;
}
```

Лістинг В2, аркуш 3

ДОДАТОК Г

Скрипти для запуску обміну телеметрією та відеопотоків

```
#!/bin/bash

./drone_telemetry &
TELEMETRY_PID=$!
gst-launch-1.0 libcamerasrc ! \
    video/x-raw,width=640,height=480,framerate=10/1 ! \
    videoconvert ! queue ! \
    x264enc bitrate=1000 key-int-max=10 ! \
    h264parse ! queue ! \
    rtph264pay ! queue ! \
    udpsink host=192.168.0.45 port=5000 &
GST_PID=$!
trap "echo 'Stopping...'; kill $TELEMETRY_PID $GST_PID; wait"
SIGINT SIGTERM
wait
```

Лістинг Г.1 – Скрипт для дрону drone_start.sh

```
#!/bin/bash
./womb_telemetry &
TELEMETRY_PID=$!
gst-launch-1.0 udpsrc port=5000 ! \
    application/x-rtp, encoding-name=H264 ! \
    rtph264depay ! \
    h264parse ! \
    queue ! \
    splitmuxsink location=/home/womb/Videos/video_%04d.mp4 \
    max-size-time=60000000000 \
    muxer=mp4mux
GST_PID=$!
trap "echo 'Stopping...'; kill $TELEMETRY_PID $GST_PID; wait"
SIGINT SIGTERM

wait
```

Лістинг Г.2 – Скрипт для «матки» рою womb_start.sh

Olga Roznovets

Digitally signed by Olga Roznovets
Date: 2025.06.16 16:12:08 +03'00'