МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ АЕРОНАВІГАЦІЇ ЕЛЕКТРОНІКИ ТА ТЕЛЕКОМУНІКАЦІЙ

КАФЕРДРА АВІОНКИ ТА СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

_____ Олена ТАЧИНІНА

“_____”_____2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

«БАКАЛАВР»

Тема: «Інтегрована система керування мобільною платформою з використанням хмарних технологій»

Виконавець: студент групи Бст-151-21-1-СУ Михайло СОЛОВЕЙ

Керівник: доцент, к.т.н. Юрій БЕЗКОРОВАЙНИЙ

Нормоконтролер: : доцент, к.т.н. Микола ДИВНИЧ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Факультет аеронавігації електроніки та телекомунікацій

Кафедра авіоніки та систем управління

Спеціальність 151 «Автоматизація комп'ютерно-інтегровані технології»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Олена ТАЧИНІНА

«_____» _____2025 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Солов'я Михайла Андрійовича

(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема кваліфікаційної роботи «Інтегрована система керування мобільною платформою з використанням хмарних технологій» затверджена наказом ректора від 20.05.2025 р. № 429/ст

2. Термін виконання роботи: з 19.05.2025р. по 14.06.2025р.

3. Вихідні дані до роботи (проєкту): мікроконтролер ESP32-WROOM-32, одноплатний комп'ютер Raspberry Pi 4B, GPS-модуль CT-U7, датчик навколишнього середовища BME680.

4. Зміст пояснювальної записки: аналітичний огляд сучасного стану робототехніки, методи та засоби розробки вбудованих систем, розробка автоматизованої системи управління рухомою плтформою.

5. Перелік ілюстративного матеріалу презентації: плата Arduino Uno R3, плата ESP32, одноплатний мікрокомп'ютер Raspberry Pi 4, плата на базі контролера STM32, блок-схема автоматизованої платформи, модуль навколишнього середовища BME680 , зображення модуля GPS GT-U7, базовий RC-шасі з заднім приводом, готовий прототипу.

6. Календарний план-графік

№ пор.	Завдання	Термін	Відмітка про виконання
1	Постановка задачі дослідження: формулювання мети, завдань, об’єкта і предмета дослідження	20.03-01.04	+
2	Аналітичний огляд існуючих рішень та опрацювання літературних джерел	02.04-10.04	+
3	Вибір та обґрунтування апаратних і програмних засобів	11.04-17.04	+
4	Розробка структурної, функціональної та принципової електричної схем системи	18.04-24.04	+
5	Розробка апаратної частини: підключення мікроконтролерів, сенсорів, модулів зв’язку, шасі тощо	25.05-01.05	+
6	Розробка програмного забезпечення: прошивка ESP32, скрипти на Raspberry Pi, обробка даних	02.05-08.05	+
7	Тестування прототипу на макеті та усунення виявлених помилок	09.05-17.05	+

7. Дата видачі завдання: «5» травня 2025 р.

Керівник кваліфікаційної роботи _____ Юрій БЕЗКОРОВАЙНИЙ

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Михайло СОЛОВЕЙ

(підпис здобувача вищої освіти) (П.І.Б.)

РЕФЕРАТ

Текстова частина роботи: 97 стор., 9 рис., 2 табл..

Об'єкт дослідження: процес збору, локального збереження та захищеної передачі сенсорних даних мобільною платформою в умовах нестабільного зв’язку.

Предмет дослідження: апаратно-програмна архітектура інтегрованої системи керування рухомою платформою з хмарними сервісами.

Мета роботи: розробка автономної та модульнуї системи керування мобільною платформою збору та передачі даних в польових умовах.

Методи дослідження: аналітичний огляд існуючих архітектур робототехнічних платформ, проєктування принципової електричної схеми, розробка прошивки для ESP32, програмування багатопотокового скрипту на Python для Raspberry Pi, польове тестування прототипу.

У роботі проведено аналітичний огляд розвитку робототехніки й методів передачі даних, розроблено архітектурну блок-схему інтегрованої системи та принципову електричну схеми, реалізовано прошивку ESP32 для опитування датчика BME680 із буферизацією та шифруванням на SD-карті, багатопотоковий скрипт на Raspberry Pi для агрегації даних і пакетної передачі в хмару, змонтовано й протестовано фізичний прототип із усуненням виявлених помилок.

Ключові слова: ІНТЕГРОВАНА СИСТЕМА, МОБІЛЬНА ПЛАТФОРМА, ХМАРНІ ТЕХНОЛОГІЇ, ESP32, RASPBERRY PI, GPS, BME680, KICAD, PYTHON, ARDUINO IDE.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

BME680 – мультисенсорний модуль для вимірювання температури, вологості, тиску та якості повітря.

CSV – Comma-Separated Values, текстовий формат для представлення табличних даних.

ESC – Electronic Speed Controller, електронний регулятор швидкості для керування двигунами.

ESP32 – високопродуктивний мікроконтролер з підтримкою Wi-Fi та Bluetooth.

GPIO – General-Purpose Input/Output, контакти загального призначення для вводу/виводу сигналів.

GPS – глобальна система позиціонування.

IDE – Integrated Development Environment, інтегроване середовище розробки.

IoT – Internet of Things, Інтернет речей.

IP – Internet Protocol, міжмережевий протокол для передачі даних.

JSON – JavaScript Object Notation, легкий текстовий формат обміну даними.

KUKA – німецький виробник промислових роботів.

OS – Operating System, операційна система.

Raspberry Pi – одноплатний комп’ютер для освітніх та дослідницьких цілей.

RX/RXD – Receive/Receive Data, лінія прийому даних в послідовних інтерфейсах.

SD-карта – карта пам’яті для зберігання даних.

SPI – Serial Peripheral Interface, послідовний периферійний інтерфейс.

TX/TXD – Transmit/Transmit Data, лінія передачі даних в послідовних інтерфейсах.

UART – універсальний асинхронний приймач-передавач.

USB – Universal Serial Bus, універсальна послідовна шина для підключення та живлення пристроїв.

Wi-Fi – технологія бездротового зв’язку.

ШІМ/PWM – Широтно-імпульсна модуляція / Pulse-Width Modulation, метод керування потужністю електричного сигналу.

ЗМІСТ

ВСТУП

Стрімкий розвиток інформаційних технологій та Інтернету речей (IoT) протягом останніх років відкриває нові можливості для створення автономних робототехнічних систем, здатних виконувати комплексні завдання у віддалених та екстремальних умовах без постійного доступу до мережі Інтернет. Сучасні рішення в галузі автоматизованих мобільних платформ зазвичай передбачають використання одно- та багатоплатформних апаратно-програмних комплексів на базі одноплатних комп'ютерів та високопродуктивних мікроконтролерів. Проте більшість із них розраховані на безперервне покриття Wi-Fi або стільникових мереж, що робить їх менш надійними у віддалених сільськогосподарських угіддях, гірських районах чи зонах надзвичайних ситуацій. Водночас існуючі розробки демонструють ефективність гібридних архітектур із локальною буферизацією даних на SD-карті та пакетним передаванням через супутникові канали, втім у них недостатньо уваги приділено забезпеченню енергоефективності, модульності та простоті інтеграції з хмарними сервісами аналітики.

Актуальність дослідження обумовлена необхідністю створення універсальної системи управління мобільною платформою, яка поєднувала б гнучкий механізм збору та локального збереження сенсорних даних, надійний протокол міжпристроєвого обміну через UART та I2C та автоматизовану передачу інформації в хмару за наявності зв'язку. Така платформа може значно підвищити ефективність точкового моніторингу ґрунтово-кліматичних параметрів у сільському господарстві, екологічного спостереження в важкодоступних районах і підтримки рятувальних операцій у зонах стихійних лих. Запозичивши кращі практики світових розробок і врахувавши потребу в енергоощадних режимах роботи, дослідження спрямоване на розв'язання критичних завдань у галузі «Автоматика та приладобудування».

Метою кваліфікаційної роботи є розробити автономну, енергоефективну та модульну систему управління мобільною платформою з використанням хмарних

технологій, яка забезпечить безперервний збір, локальне збереження й захищену передачу даних у середовищах із нестабільним або відсутнім інтернет-доступом.

Задачі, які необхідно виконати для досягнення поставленої мети:

- проаналізувати існуючі архітектури автоматизованих систем управління рухомими платформами та виявити їхні переваги й обмеження в умовах непостійного зв’язку;
- вибрати апаратні компоненти та обґрунтувати їхню сумісність із обраними комунікаційними інтерфейсами;
- розробити й реалізувати електричну схему підключення сенсорів, мікроконтролера й одноплатного комп’ютера з урахуванням енергетичної автономності;
- створити прошивку для ESP32, що забезпечує опитування датчиків, шифрування й буферизацію даних на SD-карті з передачею даних для подальшої оброки;
- розробити програмне забезпечення для Raspberry Pi, яке отримує дані BME680 через ESP32, перевіряє наявність інтернет-з’єднання і при його наявності надсилає накопичені дані на хмарний сервер;

Очікувані результати роботи включають побудову інтегрованої системи, що поєднує низькорівневу обробку сенсорних даних на ESP32, надійний обмін через UART, адаптивну енергоменеджмент-логіку та хмарну аналітику, а також розробку методики тестування в реальних польових умовах. Відповідність цих результатів сучасним трендам у робототехніці та хмарних рішеннях забезпечить практичну значущість і можливість подальшого масштабування розробленої платформи.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД СУЧАСНОГО СТАНУ РОБОТОТЕХНІКИ

1.1 Історія розвитку робототехніки

Розвиток робототехніки має глибокі історичні коріння та бере свій початок у давньому світі, де люди спробували створити механічні пристрої, які імітують людські дії, такі як хода, жести, міміка, мовлення, а також рухи тварин і небесних тіл. У багатьох випадках ці механізми виконували функції розваги, релігійного обряду або наукового дослідження, демонструючи дивовижну на той час точність і складність [22]. У працях давньогрецьких та римських інженерів згадуються автоматизовані пристрої, які здатні були виконувати базові рухи або демонструвати циклічні дії. Наприклад, в описах знайдені згадки про автоматичні двері, водяні годинники та рухомі фігури в храмах.

У III столітті до н.е. грецький вчений Герон Александрійський створив перші прототипи автоматів на парі. Серед його винаходів були театральні сцени, які приводились у рух за допомогою складної системи шківів, мотузок і важелів. Ці пристрої демонстрували основи програмованого руху, оскільки дія автоматів задавалась попередньо налаштованою послідовністю команд. Праці Герона вважаються фундаментальними для подальшого розвитку мехатроніки й кібернетики.

У Середньовіччі, незважаючи на загальний спад технічного прогресу в Європі, у Китаї та арабських країнах активно розвивались механізми, що слугували як інструментами точного вимірювання часу, так і демонстраціями інженерної майстерності. Китайський інженер Су Сун у XI столітті сконструював

астрономічний годинник з рухомими фігурами. Арабські вчені, такі як Аль-Джазірі, розробляли водяні автомати та складні механізми, які виконували обрядові або побутові дії, зокрема автоматичні крани, музичні інструменти, що самостійно грали, і навіть роботизованих слуг.

Поняття "робот" уперше з'явилося у XX столітті — в п'єсі "R.U.R." ("Розумні універсальні роботи") чеського драматурга Карела Чапека, опублікованій у 1920 році. Слово запропонував брат письменника — художник Йозеф Чапек. У цьому творі роботи були синтетичними істотами, створеними для виконання важкої праці. Саме ця концепція заклала підґрунтя для майбутнього наукового дискурсу про можливості й етику використання штучного розуму та автоматизованих систем у суспільстві.

У 1950-х роках починається епоха промислової робототехніки. Американський інженер Джордж Девол створив першого промислового робота під назвою UNIMATE, який у подальшому був удосконалений Джозефом Енгельбергером. UNIMATE був запрограмований на виконання повторюваних операцій, таких як маніпуляції з важкими деталями, точне позиціонування та автоматичне зварювання. Його конструкція включала гідравлічні приводи та систему управління на основі заздалегідь визначених команд, що забезпечувало високу точність та повторюваність дій.

UNIMATE використовувався на виробничих лініях General Motors для виконання зварювальних операцій, значно підвищуючи продуктивність і безпеку праці. Завдяки йому було усунуто потребу в ручному виконанні небезпечних завдань, що знижувало ризик травматизму серед працівників. Робот також міг працювати цілодобово без зниження ефективності, що позитивно вплинуло на рентабельність виробництва.

Цей прорив став відправною точкою для стрімкого розвитку робототехнічних рішень у промисловості. Згодом з’явилися інші моделі роботів, оснащені сенсорами, адаптивним програмуванням та гнучкими маніпуляторами. Роботи почали активно впроваджуватися в такі галузі, як електроніка, логістика,

харчова промисловість, а пізніше — в медицину та фармацевтику, де вимагалася висока точність і стерильність виконання завдань.

У 1960–1980-х роках робототехніка поступово розширювала сферу свого застосування: від обмеженого середовища заводів до мобільних роботів, які могли переміщуватись у просторі, збирати інформацію, взаємодіяти з оточенням. Важливою віхою цього періоду було створення роботів типу "BEAST" (Brain Emulating Adaptive System Technology), які демонстрували поведінку, наближену до біологічної, завдяки сенсорному зворотному зв’язку. Ці системи були попередниками сучасних автономних платформ, які здатні до адаптації в реальному середовищі.

Паралельно розроблялися перші комерційно доступні універсальні роботи, зокрема "Versatran", створений у 1960-х роках компанією AMF. Цей робот мав електрогідравлічний привід і міг виконувати різноманітні виробничі завдання, будучи однією з перших гнучких автоматизованих платформ. Разом із ним розвивались інші моделі, які поступово впроваджувались у виробничі лінії, склади та лабораторії.

Особливу роль у цьому періоді відіграв розвиток мікроелектроніки та напівпровідникових технологій. Завдяки зменшенню розмірів та вартості електронних компонентів стало можливим створення компактних, більш енергоефективних і надійних роботів. Це дало змогу інтегрувати мікропроцесори безпосередньо в керуючі системи роботів, що значно розширило їхні можливості, зробивши їх гнучкими до програмування та придатними для роботи в складних умовах.

Серед компаній, які мали значний вплив на розвиток електронної бази робототехніки, варто відзначити Intel, яка випустила перший комерційно успішний мікропроцесор 4004, Motorola — з її серією мікроконтролерів 6800, а також Texas Instruments, яка активно розвивала напрям вбудованих систем. Компанії General Electric та Westinghouse брали участь у створенні складних промислових автоматизованих систем. У сфері виробництва роботизованих платформ важливу роль відіграли Unimation, AMF, Cincinnati Milacron та Fanuc.

Їхні інженерні розробки стали основою для розвитку сучасних промислових роботів і автоматизованих виробничих ліній, що активно використовуються в автомобілебудуванні, електроніці та інших галузях.

У результаті таких змін з’явилися перші автономні транспортні засоби, дослідницькі платформи для вивчення поверхні планет (зокрема, місії NASA з роботами типу Lunokhod та Sojourner), а також роботи-помічники для медичних потреб, які відкрили нові горизонти для хірургії, діагностики та догляду за пацієнтами. Розширення сфер застосування роботів стало передумовою для формування багатьох сучасних напрямів робототехніки, включно з сервісною, мобільною, космічною та медичною робототехнікою.

У другій половині XX століття з’явилося чимало нових типів роботів, які суттєво вплинули на формування сучасної індустрії. Винайдені у 1970–1980-х роках багатофункціональні промислові роботи, такі як PUMA (Programmable Universal Machine for Assembly) від компанії Unimation, започаткували еру гнучкої автоматизації на виробництві. Роботи FANUC у Японії, а також KUKA у Німеччині стали стандартами точності та надійності у виконанні складних технологічних процесів.

Паралельно із промисловими платформами активно розвивались наукові й дослідницькі прототипи. У Стенфордському університеті було створено Stanford Cart — автономний мобільний робот, здатний орієнтуватися у складному середовищі за допомогою відеокамер. У Японії компанія WABOT представила першого гуманоїдного робота, який міг ходити, розмовляти та грати на музичних інструментах. Усі ці досягнення заклали технологічну основу для наступних хвиль розвитку автономних, інтелектуальних та сервісних роботів.

З настанням XXI століття робототехніка зазнає стрімкого розквіту, зумовленого технологічними проривами в галузях штучного інтелекту, комп'ютерного зору, обробки природної мови, сенсорних технологій і бездротових мереж. Поява інтернету речей (IoT), хмарних обчислень, 5G-зв'язку та розширених обчислювальних потужностей дала змогу створювати розумні,

взаємопов'язані системи, здатні до автономного прийняття рішень і взаємодії з іншими пристроями в режимі реального часу.

Сучасні роботи використовують алгоритми глибинного навчання для розпізнавання об’єктів, мови, жестів та емоцій, що відкриває нові горизонти у взаємодії між людиною і машиною. Наприклад, компанії Boston Dynamics, Hanson Robotics, SoftBank Robotics та iRobot активно розробляють автономні мобільні та соціальні платформи, здатні до самоорієнтації, комунікації з користувачем і адаптації до змін середовища. Успішним прикладом є робот Atlas від Boston Dynamics, який демонструє високий рівень динаміки, координації та стійкості при виконанні складних маневрів. А робот Sophia від Hanson Robotics імітує людську поведінку, здатен до ведення діалогу й емоційної взаємодії.

Сьогодні роботи застосовуються у військовій справі (розвідка, розмінування, автономні бойові системи), медицині (роботи-хірурги, асистенти, системи реабілітації), освіті (роботизовані навчальні помічники), логістиці (роботи-кур’єри, сортувальні системи), аграрному секторі (роботи для аналізу ґрунту, автоматичні системи поливу та збору урожаю), сфері обслуговування (роботи-офіціанти, консультанти) та побуті (пилососи, газонокосарки, охоронні системи) [21]. Наприклад, автономні дрони використовуються для розвідки та моніторингу територій, медичні роботи допомагають у проведенні хірургічних операцій з мінімальним втручанням, а мобільні платформи з сенсорами — для аналізу стану довкілля, збору даних про забруднення та прогнозування кліматичних змін.

Роботи поступово переходять із розряду лабораторних експериментів і прототипів у повсякденне життя, стаючи елементом інфраструктури сучасного світу. Вони забезпечують підвищення ефективності, безпеки, точності у виконанні завдань, які раніше вимагали значного людського ресурсу, і водночас породжують нові виклики — етичні, правові та соціальні, що потребують комплексного аналізу та регулювання.

1.2. Сфери застосування колісних рухомих платформ

Колісні рухомі платформи є одним із найпоширеніших типів мобільних роботів, що активно використовуються в різних галузях завдяки простоті конструкції, високій ефективності переміщення та енергоекономічності. Їхнє поширення зумовлено низкою технічних і практичних переваг: відносно просте управління, легкість у виробництві, висока швидкість пересування по рівних поверхнях та адаптивність до навігації в обмеженому просторі.

Типова колісна платформа складається з шасі (основи), на якому розміщуються електродвигуни, акумуляторна батарея, система керування (мікроконтролер або одноплатний комп'ютер), а також датчики для виявлення перешкод, навігації та збору даних. Колеса можуть бути парними (двоколісні або чотириколісні конфігурації) або омнінаправленими для підвищення маневреності. Для управління рухом використовується диференціальне керування або механізми із серводвигунами, що забезпечують точність позиціонування. Програмне забезпечення, яке контролює платформу, дозволяє реалізувати різні алгоритми навігації — від простих сценаріїв руху по заданій траєкторії до адаптивного реагування на зміну навколишнього середовища.

З моменту своєї появи колісні платформи зазнали значних удосконалень. Перші зразки мали обмежену функціональність та базувалися на простих схемах управління без інтеграції з сенсорами або зовнішніми пристроями. З часом з'явилися системи з повноцінними мікроконтролерами, модулями бездротового зв'язку та GPS-навігацією. Поліпшення матеріалів дозволило зменшити вагу конструкцій та підвищити надійність при експлуатації у складних умовах. Сучасні моделі включають модульну архітектуру, що дозволяє легко змінювати конфігурацію платформи відповідно до завдань, а також підтримують зв'язок з хмарними сервісами, базами даних і системами штучного інтелекту для прийняття автономних рішень у режимі реального часу.

Особистим прикладом такого застосування є воркшоп, який я організував 10 березня 2024 року, присвячений програмуванню та збиранню робототехнічних систем на базі Arduino. У рамках цього заходу учасники створили просту триколісну роботизовану платформу, яка рухалася по чорній лінії, використовуючи датчики для слідкування за траєкторією. Під час воркшопу було продемонстровано базові принципи диференціального керування, взаємодії мікроконтролера з сенсорами та основи алгоритмів автономного руху. Це стало практичним прикладом використання колісних платформ у навчальному процесі та розкриття їхнього потенціалу як інструменту STEM-освіти.

Одна з ключових сфер застосування таких платформ — логістика та складські системи. Автономні мобільні роботи (AMR) та автоматизовані транспортні засоби (AGV) широко використовуються на великих підприємствах і логістичних хабах для транспортування вантажів, сортування товарів і оптимізації внутрішньої логістики [22]. Компанії Amazon, Alibaba та DHL вже інтегрували десятки тисяч таких платформ у свої ланцюги постачання. Наприклад, логістичні роботи Kiva (придбані Amazon) здатні самостійно переміщувати товарні стелажі до операторів, значно підвищуючи продуктивність. Варто також згадати українську компанію "Нова пошта", яка активно модернізує свої логістичні процеси, впроваджуючи автоматизовані сортувальні лінії та мобільні платформи для покращення швидкості й точності обробки посилок на терміналах, демонструючи успішне поєднання роботизації та цифровізації в національному масштабі.

Іншою важливою галуззю є сільське господарство. Колісні платформи тут використовуються для моніторингу стану ґрунту, рослин, збору врожаю, автоматизованого поливу та внесення добрив. Вони оснащуються сенсорами для вимірювання вологості, кислотності ґрунту, температури, а також камерами для комп’ютерного зору, що дозволяє точно визначати стадії росту рослин. Такі рішення сприяють розвитку точного землеробства та підвищенню врожайності при зниженні витрат на ресурси. Саме цій галузі присвячено мій дипломний проєкт, в якому розглядається створення та використання мобільної платформи з

сенсорами для збору екологічних даних у полі, з подальшою передачею їх у хмарне середовище після повернення платформи до лабораторії.

У сфері медицини колісні платформи виконують функції допоміжного транспорту в лікарнях та реабілітаційних центрах. Наприклад, платформи типу TUG використовуються для доставки медикаментів, зразків аналізів або їжі між відділеннями. Також створюються роботи-супроводжувачі для осіб з обмеженими можливостями та роботи для телеприсутності лікарів. Важливим прикладом є проєкт Trauma Pod, ініційований DARPA — це мобільна автономна роботизована платформа, розроблена для використання на полі бою з метою надання медичної допомоги пораненим солдатам без безпосередньої участі людини. Trauma Pod поєднує можливості діагностики, стабілізації стану та підготовки до евакуації, що робить його революційним рішенням у сфері військової медицини.

У освітній і науковій діяльності колісні платформи використовуються як навчальні інструменти для вивчення основ програмування, алгоритмів автономної навігації та взаємодії з сенсорними системами. Шкільні та університетські лабораторії часто застосовують платформи на базі Arduino, Raspberry Pi або ESP32 для реалізації студентських проектів і дипломних робіт.

У промисловості такі платформи виконують роль мобільних інспекційних пристроїв для моніторингу стану обладнання, перевірки важкодоступних зон та збору даних у небезпечних середовищах. Вони можуть автономно переміщуватись по цехах, об'їжджаючи перешкоди та передаючи інформацію на центральні сервери або у хмару для подальшого аналізу.

Оборонна промисловість також активно використовує колісні платформи для розвідки, спостереження, доставки спорядження, розмінування та охорони територій. Вони оснащуються камерами нічного бачення, сенсорами руху та GPS-модулями, що дозволяє їм діяти в автономному режимі навіть у складних умовах. Особливого значення такі платформи набули в умовах повномасштабної війни Росії проти України, де безпілотні наземні транспортні засоби застосовуються для евакуації поранених, доставки боєприпасів та засобів зв’язку, а також для виявлення та знешкодження мін. Активно впроваджуються ініціативи українських

розробників, які створюють малогабаритні модульні платформи, здатні працювати у зоні бойових дій. Прикладами таких рішень є наземні дрони типу Ratel S, Camel, Lys, які успішно використовуються для виконання місій з доставки спорядження, відеоспостереження та ведення розвідки. Деякі з цих платформ мають броньовані шасі, захищені канали зв'язку та можуть керуватись як дистанційно, так і автономно за заданими маршрутами. Використання таких технологій суттєво знижує ризики для особового складу та підвищує ефективність військових операцій у зонах підвищеної небезпеки.

Ще однією перспективною галуззю є сфера обслуговування та побут. Колісні платформи у вигляді роботів-офіціантів, кур'єрів та прибиральників (наприклад, Roomba, робот-газонокосарка) вже стали частиною повсякденного життя в багатьох країнах. У поєднанні з хмарними технологіями та штучним інтелектом вони здатні навчатися, планувати маршрути, обходити перешкоди та адаптуватися до середовища користувача. Яскравим прикладом є використання роботів-офіціантів у корейських ресторанах, таких як "KT Robot Cafe" у Сеулі, де автономні платформи на базі системи AI обслуговують клієнтів, доставляючи страви до столів. Вони взаємодіють із клієнтами через сенсорні екрани, слідкують за картою приміщення та запам'ятовують маршрути, мінімізуючи ризик зіткнення з людьми чи меблями.

Таким чином, колісні рухомі платформи стали невід'ємним елементом сучасної автоматизації у найрізноманітніших сферах — від індустріальних об'єктів до домашнього вжитку. Їхня універсальність, простота в реалізації та здатність до інтеграції з передовими технологіями забезпечують постійне зростання їхньої ролі в інтелектуальних системах майбутнього.

1.3. Огляд існуючих автоматизованих систем управління рухомими платформами

Автоматизовані системи управління мобільними платформами на сьогоднішній день формують одну з найдинамічніших галузей сучасної

робототехніки. Завдяки розвитку мікроелектроніки, сенсорних технологій, обчислювальних платформ і методів обробки даних стало можливим створювати інтелектуальні пристрої, здатні до автономної навігації, адаптації до оточення та виконання складних завдань. Такі системи застосовуються в різноманітних сферах — від наукових досліджень і освіти до промисловості, медицини, сільського господарства та оборонного комплексу.

На архітектурному рівні автоматизовані системи зазвичай складаються з кількох ключових компонентів: обчислювального модуля (контролера), системи сенсорів, виконавчих механізмів, модуля зв’язку, блоку живлення та, за потреби, інтерфейсів для взаємодії з оператором або іншими системами. Контролер — це центральний елемент системи, який відповідає за прийняття рішень, обробку сенсорних даних, генерацію команд для приводів, а також зв’язок із зовнішніми сервісами чи платформами. Найпоширенішими рішеннями є мікроконтролери (STM32, Arduino, ESP32), одноплатні комп’ютери (Raspberry Pi, Jetson Nano), а також вбудовані системи на базі ARM-процесорів із підтримкою Linux або RTOS.

Сенсорна система може включати різноманітні типи датчиків залежно від призначення платформи: ультразвукові та інфрачервоні датчики для виявлення перешкод; LiDAR для створення 3D-карти оточення; GPS — для глобального позиціонування; IMU (інерційні вимірювальні одиниці) — для оцінки положення і руху; камери — для реалізації комп’ютерного зору. Кожен тип сенсорів має свої переваги та обмеження, і часто вони комбінуються для забезпечення вищої надійності та точності.

Залежно від умов експлуатації, мобільні платформи можуть мати різну конструкцію. Колісні платформи — найбільш поширений варіант завдяки простоті реалізації, хорошій енергоефективності та високій стабільності при русі на рівній поверхні.

Одноколісні роботи — нестабільні конструкції, які потребують постійного балансування за допомогою складних алгоритмів (ПІД-регулятори, LQR).

Двоколісні самобалансувальні платформи — використовуються в навчальних закладах та стартапах; зазвичай мають низьку вартість і підтримку

Bluetooth/Wi-Fi.

Чотири- та шестиколісні платформи — мають велику вантажопідйомність, стійкість і здатність до пересування по нерівному рельєфу; часто використовуються у промисловості та логістиці.

Безколісні рішення, зокрема гусеничні та крокуючі платформи, мають переваги в умовах важкодоступної місцевості. Наприклад, гусеничні платформи використовуються у військових застосуваннях, під час розмінування або у рятувальних операціях. Їх недоліком є складніша система управління рухом і більші енергетичні витрати.

Важливою складовою є рівень автоматизації, який визначається здатністю системи до самостійного прийняття рішень і ступенем взаємодії з оператором. Найпростіші системи здатні лише повторювати задані послідовності дій, не реагуючи на зміни в оточенні. Більш складні використовують реактивні стратегії, які базуються на простій обробці сенсорних сигналів (наприклад, уникнення перешкод шляхом об'їзду). Сучасні високорівневі платформи працюють на основі SLAM (Simultaneous Localization and Mapping), комп'ютерного зору, нейронних мереж, що дозволяє їм адаптувати поведінку в реальному часі, формувати карту оточення і планувати оптимальні маршрути.

Найбільш гнучким і широко використовуваним фреймворком у розробці таких систем є Robot Operating System (ROS). Його перевага полягає в модульності, що дає змогу розподіляти обробку задач між кількома вузлами, використовувати готові бібліотеки навігації, обробки сенсорних даних, інтеграції з апаратними модулями. ROS підтримує великий вибір платформ і датчиків, а також має активну спільноту, що пришвидшує розробку проєктів. Серед найпоширеніших навчальних та дослідницьких платформ, які підтримують ROS, можна виділити TurtleBot (I, II, III), Jackal, Husky, а також більш спеціалізовані рішення від Clearpath Robotics.

Не менш значущим напрямком є використання доступних одноплатних комп'ютерів і мікроконтролерів. Arduino дозволяє швидко створювати прототипи систем з простими алгоритмами управління, підтримує численні датчики, легко

програмується, що робить його популярним у студентських і аматорських проєктах. ESP32, на відміну від Arduino, має вбудовані Wi-Fi та Bluetooth модулі, більші обчислювальні можливості та підтримує FreeRTOS, що дозволяє реалізувати багатозадачність. Raspberry Pi, як повноцінний комп'ютер, дає змогу реалізовувати проєкти з використанням Linux, OpenCV, ROS тощо. Такі платформи мають низьку вартість і відмінно підходять для навчання, прототипування, STEM-освіти та стартапів.

У промисловості ж домінують високотехнологічні автономні рішення. Яскравим прикладом є робот Spot від Boston Dynamics, який здатен до автономного пересування складною місцевістю, орієнтуючись за допомогою глибоких нейронних мереж та багатьох сенсорів. Clearpath Robotics спеціалізується на створенні платформ з відкритою архітектурою для наукових і промислових досліджень. Їхні роботи інтегруються з ROS, мають великий запас енергії, гнучкі шасі, потужні обчислювальні ресурси. Додатково варто згадати AgileX Robotics, які випускають серії Scout, Hunter, Bunker — роботизовані платформи для агросектору, охорони та логістики.

В умовах обмежених або недоступних каналів зв'язку (наприклад, у сільській місцевості, шахтах, пустелях) особливо важливою є можливість автономного збору та збереження даних. Такі системи мають зберігати інформацію на локальних носіях (SD-карти), а після повернення до бази — передавати ці дані на сервер для подальшого аналізу. У подібних умовах особливо актуальним є питання енергоефективності, оскільки обмежене живлення вимагає оптимального планування маршрутів, скорочення обчислювальних навантажень і використання режимів сну для модулів.

Ще один перспективний напрям — інтеграція з хмарними сервісами. Платформи можуть передавати зібрані дані в реальному часі або пакетно до серверів для аналізу, зберігання і візуалізації. Такі системи активно використовуються в агросекторі для моніторингу ґрунту, вологості, якості повітря; у логістиці — для відстеження місцезнаходження вантажів; у промисловості — для інспекцій важкодоступних місць.

Сучасні системи управління мобільними платформами демонструють вражаючу різноманітність — від простих дистанційно керованих пристроїв до інтелектуальних автономних систем, здатних до самонавчання, взаємодії з іншими пристроями та адаптації до умов навколишнього середовища. Основними викликами в цій галузі залишаються: підвищення точності навігації, енергоефективності, зниження вартості апаратного забезпечення, а також забезпечення кібербезпеки. Вивчення існуючих рішень є необхідною основою для подальшого проєктування власних систем, орієнтованих на специфічні вимоги — зокрема, дослідження навколишнього середовища в умовах відсутності комунікацій, інтеграцію з хмарними технологіями та реалізацію безпечного автономного руху.

1.4. Постановка задачі

Сучасні виклики в галузі автоматизованих мобільних платформ вимагають створення систем, здатних функціонувати в умовах відсутності постійного підключення до мережі, обмеженої енергетичної автономності та потреби в надійному зборі і захисті даних. Більшість існуючих рішень або покладаються на постійний доступ до інтернету (наприклад, для передачі даних або отримання команд), або мають занадто обмежений набір функцій, що не дозволяє ефективно застосовувати їх у реальних умовах, таких як польові дослідження, аграрна аналітика чи військове спостереження. У зв’язку з цим дана бакалаврська робота присвячена розробці вдосконаленої автоматизованої платформи, яка вирішує ці проблеми за рахунок гнучкої архітектури, здатної до автономної роботи, захищеного зберігання даних і дистанційної взаємодії через хмарні сервіси. Розроблювана система спрямована на практичне застосування в умовах, де традиційні технології управління мобільними платформами виявляються неефективними або взагалі непридатними.

У зв’язку з цим виникає необхідність у розробці автоматизованої системи управління рухомою платформою, здатної автономно виконувати місії зі збору

екологічних або інших параметричних даних у середовищі без мережевої інфраструктури, з подальшою передачею зібраної інформації до центрального серверу після повернення до зони зв’язку. Саме таку модель реалізовано у даному дипломному проєкті. Виходячи з поставленої задачі, у наступних розділах роботи буде розроблено концепт автоматизованої комп’ютерної системи управління рухомою платформою з детальним описом апаратної архітектури, програмного забезпечення та засобів зв’язку, необхідних для її функціонування.

У межах проєкту передбачено створення рухомої колісної платформи на базі чотириколісного шасі з заднім приводом, яка функціонує в межах наступного сценарію: команда на запуск надходить із лабораторії, розташованої в місті, до польової бази. Польова база оснащена інтернет-зв’язком через супутникову систему (наприклад Starlink) та є точкою запуску й прийому мобільної платформи. Після отримання команди рухома платформа починає маршрут збору даних, використовуючи вбудовані сенсори (температури, вологості тощо), і зберігає зібрану інформацію локально, на SD-карті, у зашифрованому вигляді.

Вибір саме чотириколісної конфігурації з заднім приводом зумовлений її стабільністю, рівномірним розподілом ваги та здатністю до руху по нерівній поверхні без втрати стійкості. У порівнянні з триколісними платформами, чотири колеса забезпечують кращу балансування та меншу ймовірність перекидання при маневруванні, особливо при наявності додаткового навантаження у вигляді сенсорного обладнання. Двоколісні рішення, хоча і маневреніші, вимагають складнішої системи стабілізації та мають обмеження у подоланні перешкод. Таким чином, обрана платформа поєднує високу надійність, простоту в керуванні та універсальність для використання в польових умовах.

Після завершення маршруту платформа повертається до бази, де відбувається автоматизоване з’єднання з Raspberry Pi для передавання зібраних даних, їх дешифрування та пересилання на сервер лабораторії. База також слугує пунктом зарядки акумулятора, зокрема завдяки інтегрованій сонячній панелі, що забезпечує додаткову енергетичну автономність. Таким чином, система формує замкнутий цикл: команда — виконання — повернення — звіт.

Для реалізації обробки та передачі даних використовується комбінація мікроконтролера ESP32 (відповідає за збір сенсорних даних у реальному часі) та одноплатного комп'ютера Raspberry Pi (координує маршрутизацію даних, шифрування, з'єднання зі Starlink і сервером). Обмін між ESP32 та Raspberry Pi організований через послідовний інтерфейс (UART), що забезпечує надійність та гнучкість системи. Архітектура розробки дає змогу масштабувати проєкт за рахунок додавання нових сенсорів, блоків шифрування або навіть розширення кількості платформ у мережі [2], [3], [8].

Таким чином, метою дипломної кваліфікаційної роботи є розробка адаптивної, автономної, енергоефективної та безпечної системи управління рухомою платформою, здатної до збору, збереження та захищеної передачі даних у розподіленому середовищі з обмеженим зв'язком. Цей підхід має практичне застосування в аграрному моніторингу, екологічних дослідженнях, оборонній галузі, а також у побудові IoT-систем нового покоління.

У першому розділі дипломної роботи було розглянуто історію розвитку робототехніки, сфери застосування колісних рухомих платформ, виконано аналітичний огляд сучасних систем управління мобільними платформами та сформульовано постановку задачі. У другому розділі буде проаналізовано структуру сучасних автоматизованих систем керування, проведено огляд мікроконтролерних платформ, одноплатних комп'ютерів, методів бездротового зв'язку та засобів комп'ютерного моделювання. Третій розділ присвячено безпосередньо розробці системи — від вибору апаратного забезпечення до створення електричної схеми та програмного забезпечення. У четвертому розділі буде змодельовано та протестовано роботу системи. Завершення роботи міститиме висновки та додатки., здатної до збору, збереження та захищеної передачі даних у розподіленому середовищі з обмеженим зв'язком. Цей підхід має практичне застосування в аграрному моніторингу, екологічних дослідженнях, оборонній галузі, а також у побудові IoT-систем нового покоління. — від простих прототипів до промислових і військових роботів із повною автономією. Їхній розвиток продовжується у напрямку підвищення автономності,

енергоефективності, інтелектуальності та здатності до співпраці з іншими пристроями й людьми. У контексті мого дипломного проєкту саме аналіз сучасних підходів до АСУРП є ключем до створення ефективної, адаптивної та стійкої до зовнішніх чинників системи управління колісною мобільною платформою.

РОЗДІЛ 2
МЕТОДИ ТА ЗАСОБИ РОЗРОБКИ ВБУДОВАНИХ СИСТЕМ

2.1. Структура сучасних систем автоматизованого управління

У типовій архітектурі автоматизованої системи управління рухомою платформою ключову роль відіграє набір взаємопов’язаних модулів, кожен з яких виконує певну функцію. Насамперед це сенсорні пристрої, що здійснюють зчитування параметрів навколишнього середовища, таких як температура, вологість, освітленість, відстань до об’єктів тощо. Вони підключаються до центрального вузла обробки даних, що виконує функцію аналізу отриманої інформації та прийняття рішень.

Роль обчислювального центру в системі може виконувати як мікроконтролер, так і одноплатний комп’ютер. У системах початкового рівня доцільно використовувати прості рішення на базі Arduino або ESP32. Для складніших задач — таких як передача даних у хмару, обробка великих обсягів інформації чи підтримка складних алгоритмів — використовують Raspberry Pi, STM32 або Jetson Nano.

Комунікація між елементами системи та обмін даними з віддаленим користувачем реалізується через відповідні модулі зв’язку. Залежно від потреб, це можуть бути модулі Wi-Fi, Bluetooth, LoRa або GSM. Одним із ключових завдань у цьому контексті є забезпечення захищеності даних при їхній передачі — особливо у разі роботи в агресивному середовищі чи за умов можливого зовнішнього втручання.

Виконавча частина системи включає у себе мотор-редуктори, серводвигуни,

реле та інші механізми, які забезпечують фізичне переміщення платформи, зміну її напрямку, активацію додаткових функцій тощо. Електроживлення всієї системи, у свою чергу, базується на акумуляторних батареях, блоках керування живленням, а в деяких випадках — і на альтернативних джерелах енергії, сонячні панелі наприклад.

Залежно від потреб проєкту та умов експлуатації, система може будуватись за принципами централізованої або розподіленої архітектури. Централізована модель передбачає, що весь контроль зосереджений в одному процесорному модулі. Такий підхід простіший у реалізації та дешевший для невеликих систем, проте менш стійкий до збоїв.

Натомість розподілені архітектури дають змогу рознести логіку керування між різними вузлами — наприклад, кожен сенсор або виконавчий механізм може мати власний мікроконтролер. Це значно підвищує гнучкість і масштабованість системи, дає змогу приймати рішення локально, що зменшує час реакції, а також забезпечує вищу надійність у разі виходу з ладу окремих компонентів.

Еволюція таких систем демонструє поступовий відхід від жорстко визначених алгоритмів та логічних контролерів до модульних, гнучких архітектур із вбудованими засобами машинного навчання. Завдяки розвитку інтернету речей (IoT), дедалі більше систем отримують можливість локальної обробки даних (edge computing), обмежуючи передавання лише до релевантних зведених даних або аналітики.

Особливо важливою тенденцією є підтримка віддаленого доступу: сучасна платформа може отримувати нові інструкції, оновлення прошивки або передавати повідомлення про помилки без фізичного втручання. Це критично важливо в умовах експлуатації у важкодоступних місцях — наприклад, у сільському господарстві, зоні бойових дій або на промислових об'єктах.

Крім апаратного забезпечення, ключову роль у функціонуванні сучасної автоматизованої системи відіграє програмне забезпечення. Саме воно забезпечує взаємодію між сенсорними модулями, виконавчими механізмами та

обчислювальними блоками, реалізуючи алгоритми прийняття рішень, обробки даних, виявлення перешкод та передавання інформації до віддаленого серверу. Розробка надійного, оптимізованого та масштабованого програмного забезпечення дозволяє досягти стабільної роботи всієї системи навіть за умов обмеженого живлення, зв'язку або в присутності перешкод. У контексті даного проєкту програмне забезпечення реалізується як на мікроконтролері ESP32, так і на одноплатному комп'ютері Raspberry Pi, де кожен компонент відповідає за свою частину задач: збирання даних, обробка, шифрування, збереження та передача. Тобто передбачено реалізацію гібридної архітектури, в якій ESP32 відповідає за збирання первинних даних, а Raspberry Pi забезпечує зберігання, попередню обробку та передачу у хмару через базову станцію зі Starlink. Такий підхід дає змогу поєднати переваги обох моделей керування — гнучкість та автономність локальної обробки з надійністю централізованої передачі та синхронізації.

2.2. Огляд сучасних мікроконтролерних платформ

У сучасних системах автоматизованого керування мобільними платформами мікроконтролери виступають центральною ланкою, що об'єднує апаратне забезпечення з програмною логікою. Їх компактність, енергоефективність, універсальність і доступність відкривають широкі можливості для реалізації різних задач — від зчитування даних з сенсорів і керування приводами до обміну інформацією з іншими компонентами системи або хмарними платформами, а також постійного моніторингу виконання операцій.

Основною перевагою мікроконтролерів є їх здатність працювати в реальному часі та виконувати індивідуальні функції з мінімальною затримкою. Це особливо важливо в умовах, коли необхідна миттєва реакція системи — наприклад, при виявленні перешкод на шляху рухомої платформи або у разі зміни умов навколишнього середовища. Мікроконтролери часто використовуються у поєднанні з одноплатними комп'ютерами, які виконують більш ресурсоємні задачі, зокрема аналіз даних, логування, шифрування та відправку інформації у мережу.

Для більш складних завдань, таких як попередня обробка даних, збереження інформації, передача в хмарні сервіси або обслуговування візуального інтерфейсу, до системи можуть інтегруватися одноплатні комп’ютери — наприклад, Raspberry Pi. Хоча вони не є мікроконтролерами за своєю природою, вони часто використовуються у поєднанні з ними, створюючи гібридні архітектури, які поєднують високу обчислювальну потужність з ефективністю апаратного управління.

У даному підрозділі здійснюється систематизований аналіз найбільш поширених і технічно доцільних мікроконтролерних платформ, що використовуються у вбудованих системах. Основну увагу зосереджено на таких рішеннях, як Arduino, ESP32 та STM32, з коротким оглядом альтернативних платформ — зокрема, Teensy, MSP430 тощо. Окремий акцент зроблено на платформі Raspberry Pi як представнику одноплатних комп’ютерів, що виконує функції центрального обчислювального вузла в комплексних автоматизованих системах. Розгляд платформ здійснюється з урахуванням їх технічних характеристик, функціональних можливостей, енергоспоживання, сумісності з периферійними пристроями та придатності до використання в умовах мобільної автоматизації.

2.2.1. Мікроконтролер Arduino

Arduino — це одна з найпоширеніших мікроконтролерних платформ, яка знайшла застосування як у сфері освіти, так і в промислових розробках. Її популярність зумовлена відкритим вихідним кодом, модульною будовою, інтуїтивно зрозумілим інтерфейсом та низькою вартістю, що робить її ідеальним інструментом для створення прототипів систем автоматизації (рисю 2.2.1). Вперше Arduino була розроблена у 2005 році в Італії для потреб студентів технічних спеціальностей, проте з часом набула глобального визнання серед розробників робототехнічних систем.

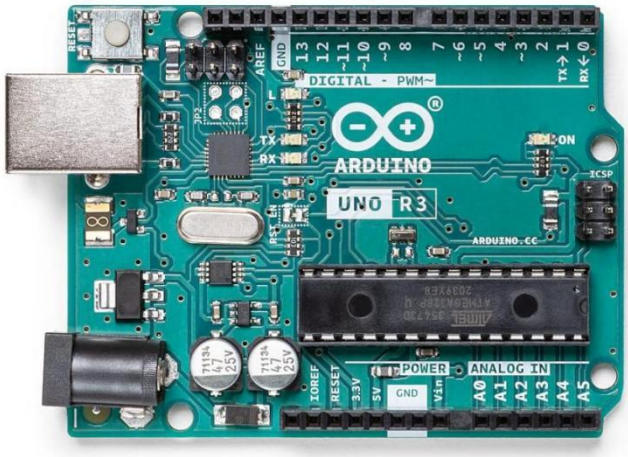


Рис. 2.2.1.1 – Плата Arduino Uno R3

Основою більшості плат Arduino є мікроконтролери серії AVR виробництва Atmel (тепер Microchip), зокрема ATmega328P та ATmega2560 [6]. Платформа має кілька версій — найпопулярніші з них це Arduino Uno, Mega, Nano, Leonardo та Due. Кожна з плат зазвичай містить набір цифрових та аналогових портів введення/виведення, USB-порт для завантаження програм, кварцовий резонатор, стабілізатор напруги, а також роз'єми для підключення розширень — так званих шилдів.

Однією з головних переваг Arduino є простота програмування. Вона використовує середовище Arduino IDE, в основі якого лежить спрощена версія мови C/C++. Завдяки широкій спільноті користувачів існує велика кількість готових бібліотек, що полегшують роботу з датчиками, приводами, комунікаційними модулями та іншими компонентами.

Arduino часто використовується в автоматизованих системах, зокрема для керування мобільними платформами, як головний або допоміжний контролер. Вона забезпечує надійне зчитування даних з аналогових та цифрових сенсорів, керування електродвигунами за допомогою драйверів типу L298N або DRV8833, а також підтримує основні комунікаційні протоколи, включно з UART, I2C та SPI.

Платформа легко інтегрується з популярними модулями бездротового зв'язку (HC-05/06, ESP8266), мобільного зв'язку (SIM800L), навігації (GPS), радіочастотної ідентифікації (RFID) та з модулями збереження даних (наприклад,

SD-карти). Це дозволяє створювати автономні системи збору та передачі інформації, які можуть реагувати на зміни середовища або накопичувати дані для подальшої обробки.

Типовими прикладами використання Arduino в мобільних платформах є реалізація алгоритмів слідування по лінії, уникнення перешкод, дистанційного керування, а також проведення вимірювань навколишнього середовища. Arduino здатна опрацьовувати сигнали з інфрачервоних або ультразвукових сенсорів, керувати сервоприводами, взаємодіяти з пультами управління та реєструвати дані.

Серед основних переваг Arduino можна виокремити:

- Простота освоєння та програмування;
- Низька вартість;
- Широка підтримка користувачів і документації;
- Швидкість створення прототипів;
- Сумісність із великою кількістю зовнішніх модулів.

Водночас платформа має і певні недоліки:

- Обмежені ресурси (оперативна пам’ять, тактова частота);
- Відсутність апаратної багатозадачності;
- Недостатній рівень захисту даних;
- Не підходить для складних обчислювальних задач чи роботи з великим обсягом даних.

Платформа Arduino розглядалась на початкових етапах даної кваліфікаційної роботи як один із варіантів для реалізації базових функцій управління та збору даних. Зокрема, вона активно використовувалася під час практичних експериментів і навчальних семінарів, де розроблялися спрощені версії керованих платформ для дослідження роботи сенсорів і виконавчих механізмів. Однак, з огляду на необхідність реалізації більш складної логіки, інтеграції із хмарними сервісами та обробки великого обсягу інформації, для основної системи було обрано поєднання ESP32 та Raspberry Pi. Проте Arduino залишається цінним інструментом для прототипування окремих вузлів системи, тестування периферії та освітніх цілей.

2.2.2. Мікроконтролер ESP32

Компанія Espressif Systems була заснована у 2008 році в місті Шанхай, Китай, і з самого початку орієнтувалася на розробку недорогих, енергоефективних мікроконтролерів з підтримкою бездротових технологій. Першим проривом компанії став чіп ESP8266, випущений у 2014 році, який швидко завоював популярність у спільноті розробників завдяки своїй низькій ціні, компактності, підтримці Wi-Fi та відкритому SDK. На той час це був один із перших модулів, який дозволяв реалізувати бездротові IoT-рішення без необхідності в окремому контролері, що значно знизило поріг входу для створення розумних пристроїв.

Наступним важливим кроком у розвитку Espressif стало випускання ESP32 (рис. 2.2.1) у 2016 році — більш потужного та функціонального спадкоємця ESP8266. Цей мікроконтролер має двоядерний 32-бітний процесор Tensilica Xtensa LX6 (або LX7 у новіших версіях), підтримку Wi-Fi та Bluetooth (у тому числі BLE), а також велику кількість периферії: GPIO, SPI, I2C, UART, ADC, DAC та інші. Завдяки цьому ESP32 став універсальним рішенням для широкого спектру застосувань: від розумного дому і промислової автоматизації до мобільних платформ і робототехніки. Espressif також активно підтримує відкриті програмні платформи, зокрема ESP-IDF (офіційний фреймворк для програмування ESP32), а також сумісність із Arduino, MicroPython і PlatformIO, що робить розробку доступною як для професіоналів, так і для ентузіастів.

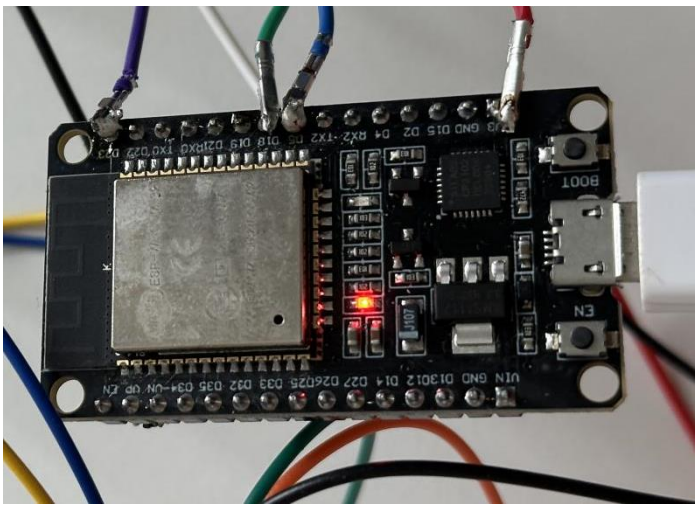


Рис. 2.2.2.1 – Плата ESP32

Серед усіх продуктів компанії особливої уваги заслуговує саме модель ESP32 — сучасне високопродуктивне вбудоване рішення, яке поєднує потужну обчислювальну архітектуру, енергоефективність і вбудовану підтримку бездротових технологій. Його архітектура базується на двоядерному 32-бітному процесорі Tensilica Xtensa LX6, який працює на частоті до 240 МГц [1]. Завдяки цьому ESP32 здатен виконувати паралельні обчислювальні завдання, що робить його особливо корисним у системах реального часу, де одночасно необхідно опрацьовувати вхідні сигнали, керувати пристроями та забезпечувати зв’язок.

Оперативна пам’ять обсягом до 520 КБ SRAM, а також підтримка зовнішньої Flash-пам’яті (до 4 МБ у більшості модулів) дозволяють створювати програми середньої складності без потреби в додатковій пам’яті. Деякі варіанти модулів також підтримують розширення через PSRAM, що корисно в проектах, пов’язаних з обробкою великих масивів даних або роботою з графікою, наприклад, при виведенні інформації на дисплей або при потоковій обробці відео з камер.

ESP32 має велику кількість універсальних пінів введення/виведення (GPIO), більшість з яких підтримує функції апаратного PWM, аналогового перетворення (ADC), цифрового перетворення (DAC), а також спеціалізованих протоколів зв’язку, таких як UART, SPI, I2C, I2S, CAN та RMT. Це дає змогу реалізувати складні конфігурації керування, а також інтегрувати в систему широкий спектр периферійних пристроїв — сенсори температури й вологості, GPS-модулі,

акселерометри, гіроскопи, RFID-рідери, а також виконавчі механізми, як-от сервомотори та реле.

Серед унікальних особливостей ESP32 варто відзначити підтримку до 18 аналогових входів із 12-бітною роздільною здатністю, а також два 8-бітні ЦАПи, які можуть бути використані для генерації аналогових сигналів. Проте точність АЦП у ESP32 обмежена внаслідок впливу шуму та нестабільності живлення, тому для точних вимірювань рекомендується використовувати зовнішні спеціалізовані АЦП, наприклад ADS1115.

Важливою перевагою ESP32 є вбудовані бездротові комунікаційні модулі — Wi-Fi та Bluetooth. ESP32 підтримує стандарти 802.11 b/g/n, включаючи режими SoftAP та STA, що дозволяє мікроконтролеру одночасно працювати як точка доступу та як клієнт. У багатьох IoT-системах це забезпечує автономну локальну взаємодію між пристроями без потреби у зовнішньому маршрутизаторі. Bluetooth підтримується у двох варіантах: класичному BR/EDR та BLE (Bluetooth Low Energy), що розширює можливості бездротового з’єднання з мобільними пристроями, сенсорами, пультами керування та іншими пристроями.

Високий рівень безпеки реалізується завдяки підтримці апаратного шифрування. ESP32 має спеціальні криптографічні блоки для роботи з алгоритмами AES, SHA-2, RSA, ECC, що забезпечують захист даних під час зберігання або передачі. Підтримка secure boot та flash encryption дає змогу захищати прошивку від несанкціонованого втручання, що критично важливо для промислових та комерційних рішень [1].

Ще одним суттєвим компонентом платформи є енергоефективність. У ESP32 реалізовано три режими сну: light sleep, deep sleep та hibernation. У режимі deep sleep споживання знижується до 10–20 мкА, що дозволяє пристроям працювати місяцями на акумуляторі або живитися від альтернативних джерел, таких як сонячні панелі. У цьому режимі зберігається лише мінімальний набір функцій, зокрема таймер або пробудження по GPIO, що дозволяє реалізовувати періодичне опитування середовища або відправку даних із заданим інтервалом.

Це є важливим фактором для застосування в бездротових сенсорних мережах, аграрних системах моніторингу або пристроях трекінгу.

Завдяки універсальності ESP32 широко застосовується в багатьох галузях: від побутових «розумних» пристроїв до складних мобільних платформ, що здійснюють навігацію та збір даних у польових умовах. Наприклад, у сільському господарстві мікроконтролер може використовуватись для моніторингу температури ґрунту, вологості повітря та інтенсивності освітлення з передачею даних до хмарної платформи для подальшого аналізу. В екологічному моніторингу — для автономного збору даних про рівень пилу, CO_2, або токсичних газів у повітрі, які потім надсилаються на сервер через Wi-Fi або зберігаються на SD-карті.

Інтерфейс SPI, реалізований у ESP32, дозволяє працювати з SD-картами, на яких може бути організовано зберігання даних у форматі файлової системи FAT. Це забезпечує простий доступ до інформації після зчитування, наприклад, через USB-кард-рідер або підключення до комп’ютера. Важливою перевагою є те, що ESP32 може самостійно працювати з файлами, читати, створювати або редагувати їх, що дозволяє реалізовувати буферизацію даних у випадках втрати зв’язку або для роботи в офлайн-режимі.

Платформа підтримує кілька мов програмування, зокрема C++ (через Arduino Core), C (через ESP-IDF — офіційний SDK), MicroPython, Lua, JavaScript (Espruino). Серед цих варіантів MicroPython вирізняється простотою у використанні, мінімальним часом розробки та зручним налагодженням. Завдяки інтерпретованому підходу можна швидко тестувати частини коду без перекомпіляції прошивки, що значно спрощує розробку у навчальних, дослідницьких або прототипних проєктах.

У рамках різних сценаріїв ESP32 здатен виконувати такі функції: періодичне опитування сенсорів, попередню обробку сигналів (наприклад, усереднення, фільтрацію, порогове визначення), збереження результатів у буфер або файл, передача через UART, Wi-Fi або Bluetooth, а також реагування на зовнішні події — натискання кнопок, зміни рівня напруги на піні, появу сигналу

на вхідному каналі. У комплексних системах ESP32 часто використовується як периферійний вузол, що передає дані до центрального модуля (наприклад, Raspberry Pi, серверу або хмарної платформи), де відбувається їх подальша обробка.

Окремо слід згадати розвинену екосистему, яка підтримує ESP32. Величезна кількість бібліотек, відкритих проєктів, навчальних матеріалів та інструментів розробки робить цю платформу доступною навіть для початківців. Зокрема, популярні середовища розробки — Arduino IDE, PlatformIO, Thonny (для MicroPython), Visual Studio Code з відповідними плагінами — забезпечують зручність і гнучкість при роботі.

Отже, незважаючи на численні переваги, ESP32 має певні обмеження. По-перше, точність вбудованого АЦП не є достатньою для задач, що потребують високоточних вимірювань. По-друге, при роботі з Wi-Fi та Bluetooth одночасно можливе зниження стабільності зв'язку, особливо за високих навантажень. По-третє, деякі модулі потребують стабільного живлення, і чутливі до перепадів напруги, тому у практичних застосуваннях рекомендується додаткове фільтрування та захист.

2.2.3. Мікрокомп'ютери Raspbarry Pi

У 2012 році було представлено Raspberry Pi — одноплатний комп'ютер, розроблений благодійною організацією Raspberry Pi Foundation для популяризації інформатики та програмування серед молоді. Завдяки доступній ціні, відкритому середовищу та підтримці Linux, пристрій швидко здобув популярність серед інженерів, хакерів, студентів і дослідників. Його почали використовувати не лише у школах, а й у різноманітних системах автоматизації, вбудованих проєктах та навіть у промислових рішеннях.

З моменту виходу першої моделі пройшло понад десять років, і кожне нове покоління Raspberry Pi приносило покращення в архітектурі, продуктивності, комунікаційних можливостях та енергоефективності. Перше покоління включало

моделі A та B на базі процесора ARM11 із частотою 700 МГц. Вони мали 256 або 512 МБ оперативної пам’яті, мінімальний набір портів (USB, Ethernet для Model B) і використовували SD-карту як накопичувач. Незважаючи на обмежені ресурси, ця плата дозволяла реалізовувати прості сервери, освітні середовища, мультимедійні центри.

У 2015 році було випущено Raspberry Pi 2. Нова модель отримала чотириядерний процесор ARM Cortex-A7 і 1 ГБ оперативної пам’яті, що забезпечило суттєвий приріст швидкодії. Завдяки збереженню сумісності з периферією попереднього покоління, оновлення було плавним. Розробники змогли запускати складніші графічні середовища, писати багатопоточні програми, використовувати пристрій як легкий десктоп або сервер для локальної мережі.

У 2016 році вийшло третє покоління, Raspberry Pi 3 Model B, з новим чотириядерним процесором ARM Cortex-A53 (1.2 ГГц) і вбудованими модулями Wi-Fi та Bluetooth. Це стало важливою віхою, адже тепер для бездротової комунікації не потрібно було додаткового обладнання. Рік потому з’явилася модель 3 B+, яка додала підтримку 5 ГГц Wi-Fi, поліпшену теплову продуктивність і Ethernet із можливістю живлення через PoE за допомогою HAT-модуля. Плата стала популярною в проєктах, де важливою була бездротова мобільність або компактність.

Четверте покоління — Raspberry Pi 4, принесло суттєві зміни (рис. 2.2.3.1). Пристрій отримав нову SoC Broadcom BCM2711 з архітектурою ARM Cortex-A72, підтримку оперативної пам’яті LPDDR4 (2, 4 або 8 ГБ), а також розширений набір портів. З’явились два micro-HDMI, що дозволяють виводити зображення на два дисплеї з роздільною здатністю до 4K [2]. Порти USB 3.0 забезпечили вищу швидкість передачі даних, а гігабітний Ethernet і вбудований Wi-Fi 802.11ac розширили мережеві можливості. Живлення тепер здійснюється через USB-C — мінімальна потужність джерела становить 15 Вт (5В / 3А), що слід враховувати при розгортанні системи.

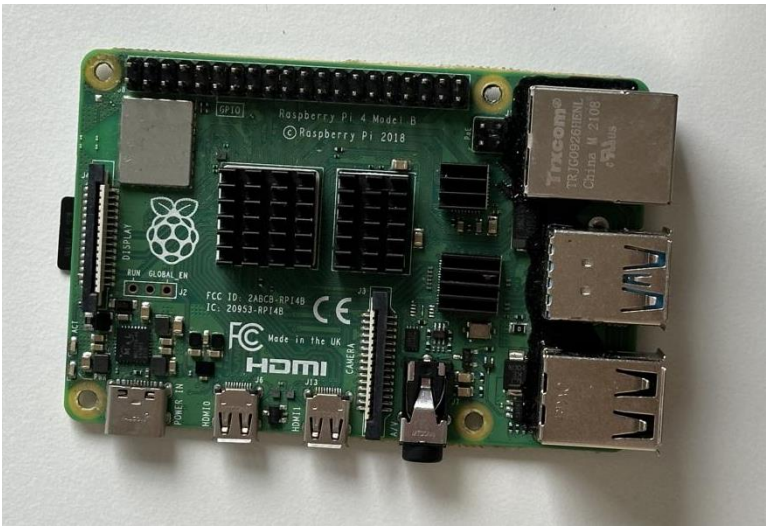


Рис. 2.2.3 – Одноплатний мікрокомп’ютер Raspberry Pi 4

У 2023 році світ побачив Raspberry Pi 5 — модель, яка принесла якісний стрибок у продуктивності. Новий чотириядерний процесор BCM2712 (ARM Cortex-A76, до 2.4 ГГц), LPDDR4X-пам’ять, покращений GPU (VideoCore VII), вища пропускна здатність USB та microSD, а також підтримка PCIe інтерфейсу через спеціальний роз’єм зробили пристрій придатним для ще складніших задач — як-от обробка потокового відео, комп’ютерний зір, розгортання моделей машинного навчання безпосередньо на пристрої (Edge AI). З’явилась офіційна підтримка активного охолодження, а також нові варіанти корпусів.

Для підключення сенсорів і зовнішніх пристроїв Raspberry Pi підтримує GPIO, UART, I2C, SPI, CSI (для камери) і DSI (для дисплея). У поєднанні з HAT-модулями, які встановлюються безпосередньо на плату, це дозволяє легко реалізовувати проєкти з керуванням моторами, збором телеметрії, GPS-навігацією, підключенням екранів і навіть з живленням через PoE.

У плані програмного забезпечення Raspberry Pi найбільш часто використовує Raspberry Pi OS — Debian-подібний дистрибутив, адаптований для плати. Також підтримуються Ubuntu Server, Arch Linux ARM, LibreELEC, DietPi, Kali Linux та інші. Можна запускати програми у фреймворках Flask, FastAPI, Node-RED, працювати з MQTT, InfluxDB, Grafana, SQLite, а також реалізовувати вебсервери, REST API, локальні бази даних або сценарії автоматизації. Файлова система зазвичай розташовується на microSD-карті (тип exFAT, ext4), але також підтримуються зовнішні SSD або HDD через USB 3.0.

Завдяки підтримці різноманітних мов програмування — Python, C/C++, JavaScript (Node.js), Go, Java — Raspberry Pi підходить як для освітніх цілей, так і для комерційних прототипів. Існує безліч готових бібліотек: для сенсорів (gpiozero, Adafruit), мережі (requests, paho-mqtt), візуалізації (matplotlib, plotly), а також для комп'ютерного зору (OpenCV) і машинного навчання (scikit-learn, tensorflow-lite).

Значною перевагою платформи є можливість повноцінного віддаленого доступу. Завдяки SSH, VNC або вебінтерфейсам, пристроєм можна керувати з іншого комп'ютера або навіть смартфона, що ідеально підходить для безголових (headless) конфігурацій — коли пристрій працює без підключеного монітора й клавіатури. Це дозволяє ефективно адмініструвати пристрої в польових умовах, на віддалених об'єктах або в автоматизованих системах.

Сфера застосування Raspberry Pi надзвичайно широка. Його використовують для збору екологічних даних, побудови станцій контролю якості повітря, створення інтелектуальних роботизованих систем, контролерів розумного дому, а також як шлюзи передачі даних у хмару через MQTT або HTTPS. Завдяки компактності, енергонезалежним варіантам живлення (через акумулятори або сонячні батареї) та підтримці системних інструментів, таких як cron, systemd або docker, Raspberry Pi чудово підходить для тривалого автономного функціонування.

Хоча ця платформа є дуже універсальною, вона має також свої обмеження. Енергоспоживання вище, ніж у мікроконтролерів, відсутність вбудованих механізмів енергозбереження на рівні заліза може ускладнити використання в енергокритичних умовах. Крім того, система потребує певного часу на завантаження, а живлення має бути стабільним, інакше можливе пошкодження файлової системи. Незважаючи на це, Raspberry Pi залишається однією з найуспішніших і наймасовіших одноплатних платформ у світі.

Raspberry Pi вирізняється балансом продуктивності та гнучкості: його процесор достатньо потужний для обробки великих обсягів даних, а набір інтерфейсів GPIO, I²C, SPI та UART дозволяє підключати різноманітні датчики й

виконавчі модулі. Повноцінна операційна система Linux відкриває можливості багатопоточності, розгортання мережевих сервісів та швидкої інтеграції з мовами високого рівня. Компактні розміри, низьке споживання енергії та підтримка популярних бібліотек для аналізу даних і візуалізації роблять Raspberry Pi ідеальною основою головного контролера: він забезпечує реалізацію складних алгоритмів навігації, збирання й передачі телеметрії без потреби в додаткових апаратних надбудовах.

2.2.4. Мікроконтроллери STM32

STM32 — це серія 32-бітних мікроконтролерів, створена швейцарсько-французькою компанією STMicroelectronics, одним із провідних світових виробників напівпровідникових компонентів, зі штаб-квартирою в Женеві. Ці мікроконтролери базуються на архітектурі ARM Cortex-M і охоплюють широкий діапазон застосувань — від простих пристроїв керування до складних промислових, медичних, автомобільних та побутових систем. Завдяки великій кількості моделей, високій енергоефективності, продуктивності та розвиненій периферії, STM32 утвердився як одне з наймасовіших і найпопулярніших сімейств мікроконтролерів на ринку (рис. 2.2.4.1).

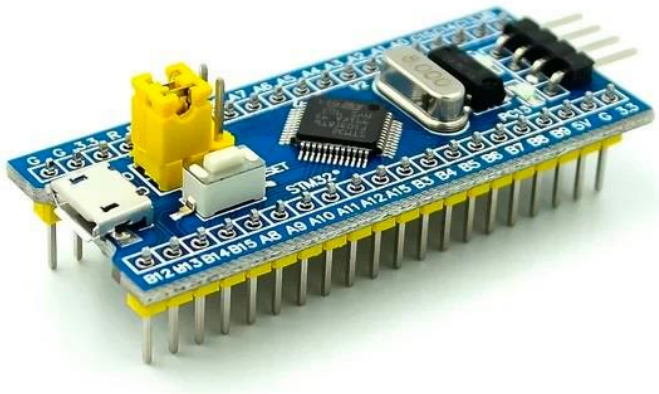


Рис. 2.2.4.1 – Плата на базі контролера STM32

Перші моделі STM32 з’явилися у 2007 році. З того часу лінійка постійно розширюється та вдосконалюється, охоплюючи процесори на основі ліцензованих

ядер ARM Cortex-M0, M0+, M3, M4, M7, а також новіші — M33 та M35P. Такий підхід надає розробникам змогу гнучко обирати мікроконтролери з оптимальним співвідношенням між ціною, обчислювальними можливостями, енергоспоживанням і функціональністю відповідно до конкретного проєкту.

Серед різноманітних серій STM32 варто виокремити кілька ключових. STM32F0 — це базова лінійка з ядром Cortex-M0, призначена для простих завдань із мінімальними ресурсами [18]. Її модернізований аналог STM32G0 зберігає сумісність, але має кращі аналогові характеристики та підвищену надійність. Серія STM32F1, а особливо модель STM32F103C8T6 (відомий як "Blue Pill"), поєднує доступність із функціональністю та є надзвичайно популярною в аматорських і навчальних проєктах.

STM32F3 відзначається наявністю цифрового сигнального процесора (DSP) і підтримкою апаратного FPU, що робить її доречною для задач обробки сигналів. Більш потужна лінійка — STM32F4 — має ядро Cortex-M4, високу тактову частоту, швидку Flash-пам’ять і підтримку багатьох периферійних інтерфейсів. Вона є стандартом у багатьох комерційних та промислових розробках.

На верхівці продуктивності — STM32H7, яка забезпечує тактову частоту до 480 МГц, великий об’єм пам’яті, підтримку графіки, криптографічних модулів, інтерфейсів Ethernet та дисплеїв. Для пристроїв із тривалим автономним живленням існують серії STM32L0, L1, L4 та L5, спеціально оптимізовані для ультранизького енергоспоживання без шкоди для функціональності.

STM32-платформи підтримують багатий набір периферійних інтерфейсів: GPIO, UART, SPI, I2C, CAN, USB, SDIO, ADC, DAC, PWM, а також інтерфейси для підключення дисплеїв (LTDC), камер (DCMI) та зовнішньої пам’яті (FSMC, QSPI, SDRAM). Кількість і типи доступних інтерфейсів залежать від конкретної серії та моделі. Так, згадана вище STM32F103 (Blue Pill) широко застосовується у прототипуванні завдяки простоті, доступності та великій спільноті користувачів, зокрема в середовищі Arduino.

Платформа STM32 має добре реалізовані функції енергозбереження. Більшість моделей підтримують декілька режимів низького енергоспоживання —

sleep, stop, standby, що дозволяє значно зменшити витрати енергії під час простою. Особливо ці можливості важливі для переносних, автономних або IoT-пристроїв, де важливо мінімізувати споживання при збереженні готовності до роботи. Моделі серій STM32L особливо ефективні в цьому аспекті.

Для розробки програмного забезпечення STMicroelectronics надає фреймворк STM32Cube, що включає графічний конфігуратор CubeMX, HAL-драйвери, бібліотеки CMSIS, приклади коду та інструменти для генерації проектів. Основним середовищем розробки є STM32CubeIDE, але також активно використовуються Keil uVision, IAR Embedded Workbench, PlatformIO та інші IDE. Частина моделей підтримує Arduino Core, що спрощує вхід у платформу для новачків. Для складніших проєктів передбачена підтримка RTOS (наприклад, FreeRTOS або Zephyr) із планувальниками задач, синхронізацією, таймерами та іншими засобами міжзадачної взаємодії.

Програмування мікроконтролерів STM32 зазвичай відбувається через інтерфейс SWD або JTAG, із використанням відлагоджувача ST-Link (вбудованого або зовнішнього). Крім того, більшість плат підтримують оновлення прошивки через UART або USB (DFU-режим). Для зберігання коду використовується вбудована Flash-пам’ять, обсяг якої може сягати кількох мегабайт, залежно від моделі. Також у багатьох версіях реалізована SRAM-пам’ять розміром до 1 МБ.

STM32 знаходить широке застосування у промисловій автоматизації, медичних приладах, системах HMI, автомобільній електроніці, аудіообробці, моніторингу навколишнього середовища тощо. Наприклад, у системах збору екологічних даних мікроконтролер може зчитувати інформацію з температурних, вологісних або газових сенсорів, виконувати попередню обробку, зберігати дані у пам’яті або передавати їх через інтерфейси UART, LoRa чи NB-IoT до центрального вузла. Це забезпечує довготривалу автономну роботу, стабільність та точність вимірювань навіть у складних польових умовах.

Попри численні переваги, STM32 вимагає глибшого розуміння вбудованих систем у порівнянні з простішими платформами на зразок Arduino або ESP32.

Наприклад, конфігурація тактування, робота з HAL або LL-бібліотеками, детальне налаштування регістрів — усе це потребує технічної підготовки. Крім того, офіційна документація хоч і велика за обсягом, іноді виявляється надто загальною або фрагментарною, що змушує звертатися до спільнот, форумів і сторонніх прикладів. Проте ці складності окуповуються можливістю створення масштабованих, надійних і гнучких рішень для будь-якого рівня складності.

У поєднанні з сильною спільнотою розробників, стабільною підтримкою та великою кількістю доступних ресурсів, STM32 залишається потужною платформою як для комерційних проєктів, так і для наукових досліджень та експериментальних розробок.

2.2.5. Інші мікроконтролерні платформи (Teensy, MSP430 та інші)

Окрім популярних платформ, таких як Arduino, ESP32, Raspberry Pi та STM32, у галузі вбудованих систем існує ще низка альтернатив, які мають специфічні переваги залежно від завдань. Серед них варто згадати мікроконтролери Teensy, MSP430, Atmel AVR, PIC, а також RISC-V-платформи.

Teensy — це серія мікроконтролерів, розроблена компанією PJRC. Основна перевага — висока продуктивність у компактному формфакторі та повна сумісність із Arduino IDE. Наприклад, Teensy 4.1 оснащений процесором ARM Cortex-M7 з частотою до 600 МГц, має 1 МБ оперативної пам’яті RAM, підтримку Ethernet, SD-карт, USB-хосту, та надзвичайно низьке енергоспоживання. Завдяки підтримці високошвидкісного аудіо, MIDI, обробки сигналів та DMA, платформа часто використовується у проєктах цифрового синтезу, звукових ефектів, HMI-інтерфейсах [20].

MSP430 — мікроконтролери від Texas Instruments, оптимізовані для наднизького енергоспоживання. Завдяки споживанню одиниць мікроампер у режимі сну, вони широко застосовуються в носимій електроніці, побутових приладах, сенсорних системах і медичних пристроях. Хоча продуктивність

MSP430 нижча порівняно з ARM-рішеннями, їх автономність і стабільність у режимах з мінімальним енергоспоживанням є головною перевагою [21].

Atmel AVR — класична серія 8-бітних мікроконтролерів, яка стала основою для оригінальних Arduino. Моделі, такі як ATmega328P, досі широко використовуються у навчальних проєктах та простих системах керування. Їх переваги — простота, велика кількість бібліотек та інструментів, але обмежена продуктивність і периферія [21].

PIC — мікроконтролери від Microchip, що мають велику історію застосування у промисловості. Вони доступні у 8-, 16- і 32-бітних версіях. Характеризуються низьким енергоспоживанням, стабільністю та довгим терміном життєвого циклу. Програмуються за допомогою MPLAB X IDE. Хоча екосистема менш популярна серед хобістів, PIC і досі актуальні в комерційних вбудованих продуктах 20.

RISC-V — новітній відкритий набір інструкцій (ISA), на базі якого створюються мікроконтролери нового покоління, як-от SiFive або GD32V. Їх перевага — відкритість архітектури, модульність, перспективи масштабування. RISC-V все активніше використовується в освіті, експериментах та відкритих проєктах [21].

Ці альтернативні мікроконтролери дають розробникам більшу гнучкість у виборі апаратної платформи, виходячи з вимог до енергоспоживання, продуктивності, периферії, бюджету та екосистеми підтримки.

У таблиці 2.2.5.1 наведена порівняльна таблиця основних мікроконтролерних платформ.

Таблиця 2.2.5.1

Порівняльна таблиця основних мікроконтролерних платформ

Платформа	**Архітектура**	**Тактова частота**	**Пам’ять (RAM/Flash)**	**Основні переваги**	**Основні недоліки**
Arduino (Uno)	AVR 8-біт	16 МГц	2 КБ / 32 КБ	Простота, Arduino IDE, велика спільнота	Обмежена продуктивність і периферія
ESP32	Xtensa 32-біт	до 240 МГц	до 520 КБ / до 16 МБ	Wi-Fi, Bluetooth, підтримка Python, низька ціна	Нестабільний АЦП, чутливість до живлення
Raspberry Pi 4	ARM Cortex-A72	до 1.5 ГГц	2–8 ГБ RAM / SD-карта	Повноцінна ОС, багато портів, потужність	Високе енергоспоживання, потрібне охолодження
STM32F4	ARM Cortex-M4	до 180 МГц	до 192 КБ / до 1 МБ	Гнучкість, стабільність, низьке енергоспоживання	Складність для новачків, конфігурація

2.3. Методи передачі даних

У контексті сучасних робототехнічних систем і мобільних платформ передача даних є не лише технічним засобом комунікації між вузлами, а й критичним фактором, що визначає продуктивність, надійність та безпеку всієї системи. Зі збільшенням автономності, складності архітектури та необхідності в обміні даними в реальному часі, вимоги до засобів передачі даних постійно зростають.

Вибір методу передачі даних залежить від конкретних параметрів системи: дальності зв’язку, енергоспоживання, стійкості до перешкод, швидкості обміну, а також сумісності з іншими модулями. Наприклад, Bluetooth забезпечує енергоефективний обмін на короткі відстані (до 100 м), що робить його ідеальним для локальної передачі між мікроконтролером і датчиками. Натомість Wi-Fi надає значно більшу швидкість передачі даних і кращу інтеграцію з інтернет-інфраструктурою, але вимагає більше енергії, що важливо для мобільних автономних пристроїв.

У випадках, де дротове підключення неможливе або небажане, часто використовуються радіомодеми, здатні працювати на більших відстанях з мінімальним енергоспоживанням. Проте вони мають обмежену пропускну здатність, що обмежує їх використання в додатках, які потребують передачі великих обсягів даних, як-от відео- або сенсорні потоки в реальному часі.

2.3.1. Bluetooth

Bluetooth є одним із найпоширеніших стандартів бездротової передачі даних на короткі відстані, що широко використовується в сучасних автоматизованих системах, включаючи мобільні роботизовані платформи. Його переваги полягають у низькому енергоспоживанні, простоті реалізації, широкій підтримці пристроїв і можливості передачі даних без необхідності прямої видимості між модулями. У контексті розроблюваної мобільної платформи, де ключовими обчислювальними вузлами є мікроконтролер ESP32 та одноплатний комп'ютер Raspberry Pi, технологія Bluetooth може розглядатися як один із потенційних методів для організації бездротового обміну даними між цими двома компонентами, особливо враховуючи їхню вбудовану підтримку даного стандарту.

Основні принципи роботи технології Bluetooth базуються на радіочастотному обміні даними в діапазоні 2.4 ГГц (ISM-діапазон), що не потребує ліцензування. Передача здійснюється за допомогою методу частотної перебудови (FHSS — Frequency Hopping Spread Spectrum), що передбачає зміну робочої частоти до 1600 разів на секунду [9]. Це зменшує ймовірність перешкод і підвищує надійність з’єднання. Bluetooth-пристрої організовуються в піко-мережі (piconet), де один пристрій виступає в ролі ведучого (master), а до нього можуть підключатися до семи відомих підлеглих (slaves). Можливе також об'єднання кількох піко-мереж у скеттер-мережу (scatternet), що забезпечує більшу гнучкість топології.

Існують три основні класи Bluetooth-пристроїв, що розрізняються потужністю передавача і, відповідно, дальністю дії: Клас 1 має потужність передавача 100 мВт (20 дБм) і максимальну відстань до 100 м; Клас 2 – 2.5 мВт (4 дБм) з дальністю до 10 м; Клас 3 – 1 мВт (0 дБм) з дальністю до 1 м. Для мобільних платформ, особливо тих, що працюють в енергозалежному середовищі, зазвичай обираються модулі класу 2 або класу 3, оскільки вони забезпечують баланс між дальністю зв’язку та енергоспоживанням [8].

З моменту своєї появи Bluetooth постійно вдосконалювався. Найбільш значущими для вбудованих систем стали такі версії: Bluetooth 2.0 + EDR (Enhanced Data Rate), що підтримує швидкість передачі до 3 Мбіт/с; Bluetooth 3.0 + HS (High Speed), який використовує Wi-Fi для високошвидкісної передачі, але споживає більше енергії; Bluetooth 4.0/4.2, що представив Bluetooth Low Energy (BLE) – надзвичайно енергоефективну версію, яка дозволяє довготривалу роботу на батарейках; та Bluetooth 5.0/5.2, де значно збільшено дальність (до 240 м) і швидкість (до 2 Мбіт/с для BLE), а також реалізовано підтримку одночасного підключення кількох пристроїв. Обидва мікроконтролери, ESP32 та Raspberry Pi 4B, що використовуються в даному проєкті, оснащені вбудованими модулями Bluetooth, що підтримують як класичний режим, так і BLE.

Розглядаючи Bluetooth як канал зв'язку між ESP32 та Raspberry Pi на одній мобільній платформі, можна виділити декілька потенційних сценаріїв використання. Наприклад, ESP32, відповідаючи за збір даних з сенсорів, міг би передавати ці дані або їх частину на Raspberry Pi для подальшої обробки, зберігання чи агрегації без необхідності фізичного дротового з'єднання. Raspberry Pi, у свою чергу, міг би надсилати конфігураційні команди або оновлення завдань на ESP32. Такий підхід може бути корисним для зменшення кількості дротів або як резервний канал зв'язку.

Однак, при виборі Bluetooth для внутрішньоплатформеного зв'язку між ESP32 та Raspberry Pi необхідно враховувати певні аспекти. По-перше, хоча BLE є енергоефективним, постійно активний канал Bluetooth (особливо в класичному режимі) може споживати більше енергії порівняно з дротовими інтерфейсами. По-

друге, швидкість передачі даних може бути нижчою, ніж у дротових аналогів, що може стати обмеженням при передачі великих масивів даних. По-третє, програмна реалізація Bluetooth-зв'язку є складнішою порівняно з прямим використанням UART.

Безпека Bluetooth реалізується на кількох рівнях і включає процеси сполучення (pairing), автентифікації та шифрування переданих даних. Під час сполучення пристрої обмінюються криптографічними ключами, після чого встановлюється захищений канал зв’язку. У сучасних версіях Bluetooth (починаючи з 4.2) використовуються AES-128 у режимі CCM (Counter with CBC-MAC), що забезпечує достатній рівень захисту при правильному налаштуванні. Однак, попри ці механізми, існує низка відомих вразливостей. До них належать такі загрози, як bluejacking – надсилання небажаних повідомлень; bluesnarfing – несанкціонований доступ до особистих даних пристрою; а також bluebugging – отримання повного контролю над Bluetooth-пристроєм. Такі атаки є особливо небезпечними у випадках, коли Bluetooth-модуль постійно активний або використовується в публічних чи незахищених середовищах. Для зниження ризиків рекомендується використовувати складні паролі або PIN-коди для сполучення, вимикати Bluetooth-модуль, коли він не використовується, відхиляти сполучення з невідомими пристроями та регулярно оновлювати мікропрограмне забезпечення. У випадку мобільної платформи, що працює в ізольованому середовищі, ризик таких атак на внутрішньоплатформенний зв'язок суттєво знижується, проте при сервісному підключенні зовнішніх пристроїв до платформи ці заходи безпеки залишаються актуальними.

Bluetooth, у даному дипломному проєкті, може розглядатися як допоміжний або опціональний канал: наприклад, для передачі некритичних даних, для тимчасового діагностичного підключення до одного з модулів без використання кабелю. Однак, для основного завдання передачі зібраних екологічних даних від ESP32 до Raspberry Pi перевага надається дротовому з'єднанню через його простоту, надійність та низькі накладні витрати.

2.3.2 Wi-Fi

Wi-Fi (Wireless Fidelity) — це технологія бездротової передачі даних, яка забезпечує високошвидкісний доступ до комп’ютерних мереж і базується на міжнародних стандартах IEEE 802.11. Завдяки своїй гнучкості, простоті інтеграції, відносно високій пропускній здатності та підтримці широкого спектра пристроїв, Wi-Fi став одним з основних засобів зв’язку у сфері автоматизації, зокрема в системах керування мобільними роботизованими платформами.

Розробка технології Wi-Fi розпочалася у 1985 році, коли Федеральна комісія США зі зв’язку (FCC) відкрила неліцензовані частотні діапазони ISM (Industrial, Scientific and Medical) для публічного використання. Перший стандарт IEEE 802.11, прийнятий у 1997 році, забезпечував швидкість передачі даних до 2 Мбіт/с. З того часу технологія зазнала значного розвитку. Стандарт IEEE 802.11a (1999) запропонував швидкість до 54 Мбіт/с у діапазоні 5 ГГц, характеризуючись вищою швидкістю, але меншим радіусом дії. Одночасно з'явився IEEE 802.11b (1999), що працював у діапазоні 2.4 ГГц зі швидкістю до 11 Мбіт/с, забезпечуючи кращу проникність сигналу, але нижчу швидкість. Стандарт IEEE 802.11g (2003) став компромісом, пропонуючи до 54 Мбіт/с у діапазоні 2.4 ГГц. Значний крок уперед був зроблений з появою IEEE 802.11n (2009), який підтримував швидкість до 600 Мбіт/с, працював у діапазонах 2.4 ГГц та 5 ГГц і впровадив технологію MIMO (Multiple Input, Multiple Output) для підвищення продуктивності. Наступним етапом став IEEE 802.11ac (2013), що працював переважно у діапазоні 5 ГГц і забезпечував теоретичну швидкість до 3.47 Гбіт/с, підтримуючи ширші канали та MU-MIMO (Multi-User MIMO). Найновішим поширеним стандартом є IEEE 802.11ax, відомий як Wi-Fi 6 (прийнятий у 2019 році), який працює в обох діапазонах (2.4 ГГц та 5 ГГц) і може досягати швидкостей до 9.6 Гбіт/с, пропонуючи покращену ефективність, особливо в умовах високої щільності клієнтських пристроїв [10].

Сучасні роботизовані системи найчастіше використовують стандарти 802.11n, 802.11ac та Wi-Fi 6 (802.11ax). Робота в діапазоні 2.4 ГГц забезпечує краще проникнення сигналу крізь фізичні перешкоди та більший радіус дії (до 100 метрів на відкритій місцевості), проте цей діапазон більш схильний до інтерференції від інших пристроїв, таких як Bluetooth-модулі чи мікрохвильові печі. Діапазон 5 ГГц, навпаки, пропонує вищу швидкість передачі даних та менший рівень завад, але має менший радіус дії, зазвичай до 30–50 метрів.

Завдяки відносно низькій затримці (в діапазоні 5-50 мс), Wi-Fi добре підходить для завдань, що вимагають передачі даних у реальному часі, наприклад, для потокового відео з камер спостереження, встановлених на мобільній платформі, або для передачі телеметричних даних. Висока пропускна здатність Wi-Fi дозволяє ефективно передавати дані з сенсорів, що генерують великі обсяги інформації (наприклад, LiDAR), а також координувати роботу кількох мобільних платформ у спільному робочому просторі, як це реалізовано в інтелектуальних складських системах або на автоматизованих виробничих лініях. Більшість сучасних одноплатних комп'ютерів, включаючи Raspberry Pi та ESP32 (у певних конфігураціях), мають вбудовану підтримку Wi-Fi, що значно спрощує їх інтеграцію в робототехнічні системи. Широка підтримка Wi-Fi з боку хмарних сервісів (AWS IoT, Azure IoT Hub, Google Cloud IoT) зробила цю технологію основою для реалізації концепції Інтернету речей (IoT) у промисловій автоматизації.

Існують також альтернативні режими роботи Wi-Fi, окрім традиційного підключення до точки доступу (Access Point). Наприклад, Wi-Fi Direct дозволяє створювати прямі з'єднання між пристроями без необхідності використання маршрутизатора, що може зменшити затримки та підвищити швидкість локальної взаємодії між компонентами платформи, такими як ESP32 та Raspberry Pi. Mesh Wi-Fi технології забезпечують гнучке мережеве покриття на великих територіях (склади, виробничі цехи) завдяки динамічному маршрутизуванню трафіку між численними вузлами. Варто також згадати стандарт IEEE 802.11ah (Wi-Fi HaLow), що працює в субгігагерцовому діапазоні (переважно 900 МГц) і характеризується

більшим радіусом дії (до 1 км), нижчим енергоспоживанням та кращою проникністю сигналу, що робить його перспективним для роботизованих IoT-систем на відкритих територіях.

Незважаючи на численні переваги, Wi-Fi має певні обмеження, особливо для автономних мобільних платформ. Обмежений радіус дії є критичним фактором при роботі на великих відкритих просторах або в польових умовах, де відсутня розвинена інфраструктура точок доступу. Вихід платформи за межі зони покриття Wi-Fi призводить до втрати зв'язку та можливості керування в реальному часі. Високе енергоспоживання Wi-Fi модулів, особливо під час активної передачі великих обсягів даних, суттєво впливає на час автономної роботи мобільних пристроїв від акумуляторних батарей. Стабільність з'єднання може погіршуватися в середовищах із значною кількістю радіоперешкод, наявністю металевих конструкцій або щільних фізичних перепон, що призводить до втрат сигналу та затримок.

Питанням безпеки в мережах Wi-Fi приділяється значна увага. Сучасні системи використовують протоколи шифрування, такі як WPA2 (Wi-Fi Protected Access 2) з шифруванням AES, та новіший стандарт WPA3, який покращує захист від атак підбору паролів та забезпечує підвищену конфіденційність навіть у відкритих мережах завдяки OWE (Opportunistic Wireless Encryption). Додатково, для захисту даних на прикладному рівні, особливо при взаємодії з хмарними платформами, використовуються протоколи SSL/TLS, що гарантують цілісність та конфіденційність інформації під час передачі. При організації Wi-Fi зв'язку між ESP32 та Raspberry Pi на борту платформи, рекомендується використовувати надійні методи автентифікації та шифрування, навіть якщо мережа є локальною, для запобігання несанкціонованому доступу.

У контексті розроблюваної мобільної платформи, Wi-Fi може слугувати каналом зв'язку між мікроконтролером ESP32 та одноплатним комп'ютером Raspberry Pi. Обидва пристрої мають вбудовані Wi-Fi модулі. Це дозволяє організувати бездротовий обмін даними, наприклад, ESP32 може передавати зібрані з сенсорів дані на Raspberry Pi для агрегації, обробки або подальшої

передачі через потужніший канал зв'язку (як-от Starlink). Raspberry Pi, у свою чергу, може надсилати команди управління або конфігураційні параметри на ESP32. Такий підхід забезпечує гнучкість у розміщенні компонентів платформи, хоча й може бути менш енергоефективним та мати більші затримки порівняно з прямими дротовими з'єднаннями, такими як UART або SPI, для постійного потоку даних. Wi-Fi Direct є особливо цікавим варіантом для прямого зв'язку між ESP32 та Raspberry Pi, оскільки він не потребує зовнішньої точки доступу, дозволяючи створити ізольовану локальну мережу на борту платформи.

Тому спосіб передачі даних Wi-Fi є потужним інструментом для організації зв'язку в багатьох сценаріях, особливо в межах локального середовища з наявною інфраструктурою.

2.3.3. LORA

У контексті сучасних методів передачі даних для систем Інтернету речей (IoT), особливо для мобільних платформ та задач віддаленого збору інформації, технологія LoRa (Long Range) представляє значний інтерес. LoRa є запатентованою технологією модуляції радіосигналів фізичного рівня (PHY), розробленою компанією Semtech. Її функціонування базується на методі розширення спектра з використанням чирп-сигналів (Chirp Spread Spectrum, CSS), що забезпечує високу стійкість до радіозавад та дозволяє здійснювати передачу даних на значні відстані при мінімальному енергоспоживанні. LoRa використовує вільні від ліцензування ISM-діапазони частот (Industrial, Scientific, and Medical), такі як 868 МГц у Європі або 915 МГц у Північній Америці, що спрощує її впровадження.

На основі фізичного рівня LoRa функціонує мережевий протокол LoRaWAN (Long Range Wide Area Network). LoRaWAN є відкритим стандартом, що визначає протокол канального рівня (MAC layer) та загальну системну архітектуру. Цей стандарт розробляється та підтримується альянсом LoRa Alliance. Він регламентує, яким чином кінцеві пристрої використовують апаратні засоби LoRa

для взаємодії з мережевою інфраструктурою, охоплюючи аспекти форматування повідомлень, класифікації пристроїв, процедури підключення до мережі та механізми управління нею. Архітектура мережі LoRaWAN зазвичай будується за топологією "зірка-з-зірок" (star-of-stars). У такій конфігурації кінцеві вузли, що включають сенсори та виконавчі пристрої, передають інформацію на один або декілька шлюзів (gateways) [11]. Шлюзи, в свою чергу, виконують функцію ретрансляції отриманих повідомлень на центральний мережевий сервер (Network Server). Мережевий сервер відповідає за важливі функції, такі як дедуплікація пакетів даних, адаптивне регулювання швидкості передачі (Adaptive Data Rate, ADR), управління політиками безпеки та маршрутизація оброблених даних до відповідного сервера додатків (Application Server), де вони стають доступними для кінцевого користувача або подальшого аналізу.

Технологія LoRa/LoRaWAN характеризується низкою суттєвих переваг. По-перше, це велика дальність зв'язку, що може досягати декількох кілометрів у міських умовах і значно більше на відкритій місцевості, суттєво перевершуючи можливості Wi-Fi або Bluetooth. По-друге, надзвичайно низьке енергоспоживання кінцевих пристроїв LoRaWAN дозволяє їм функціонувати автономно від батарейних джерел живлення протягом тривалого часу, що є критичним для систем збору даних, де часта заміна або підзарядка елементів живлення ускладнена. Крім того, сигнали LoRa демонструють високу проникаючу здатність, що забезпечує надійність зв'язку в умовах щільної забудови або промислових об'єктів. Масштабованість системи, яка дозволяє підключати велику кількість пристроїв до одного шлюзу, та відносно низька вартість розгортання інфраструктури роблять LoRaWAN економічно привабливим рішенням. Важливим аспектом є також вбудовані механізми безпеки, що включають шифрування даних на мережевому та прикладному рівнях (зазвичай AES-128) для забезпечення конфіденційності та цілісності інформації [11].

Незважаючи на значні переваги, технологія LoRa/LoRaWAN має певні обмеження. До них належить, передусім, низька швидкість передачі даних, що є компромісом для досягнення великої дальності та низького енергоспоживання.

Типова швидкість передачі варіюється в діапазоні від 0.3 кбіт/с до приблизно 50 кбіт/с, що робить цю технологію неоптимальною для передачі великих обсягів даних, наприклад, мультимедійної інформації. Технологія оптимізована для передачі коротких пакетів даних, таких як показники сенсорів або командні повідомлення. Іншим обмежуючим фактором є регламентований робочий цикл (Duty Cycle) у ISM-діапазонах, який обмежує сумарний час, протягом якого пристрій може активно передавати дані. Це обмеження введене для запобігання перевантаженню радіоефіру і може впливати на максимальну частоту відправки повідомлень. Також, залежно від обраного класу пристрою LoRaWAN (A, B, або C), можуть виникати затримки при передачі даних у напрямку від сервера до кінцевого пристрою (downlink), оскільки найбільш енергоефективні пристрої класу A активують приймач лише на короткий час після власної передачі.

У контексті мобільних роботизованих платформ, призначених для збору даних, технологія LoRaWAN може знайти застосування для передачі телеметричної інформації, компактних даних з сенсорів або статусних повідомлень на пункт управління, особливо в умовах обмеженого або відсутнього покриття Wi-Fi чи стільникових мереж. Можливе також створення локальних приватних LoRaWAN-мереж, де мобільна платформа взаємодіє з розподіленою мережею стаціонарних сенсорів або сама виконує роль мобільного шлюзу.

При розгляді технології LoRa/LoRaWAN як основного засобу передачі даних для проєкту я зіткнувся з деякими обмеженнями, які роблять її неоптимальною для вирішення всіх завдань передачі даних.

По-перше, ключовим обмежуючим фактором є низька пропускна здатність LoRaWAN. Швидкість передачі даних, що зазвичай не перевищує 50 кбіт/с, є недостатньою для передачі великих обсягів інформації, які можуть генеруватися мобільною платформою. Наприклад, якщо платформа збирає дані з різноманітних сенсорів високої роздільної здатності, фотографічні матеріали, відеофрагменти (навіть низької якості), або здійснює детальне логування параметрів довкілля з високою частотою, сукупний обсяг даних швидко перевищить можливості LoRa.

По-друге, обмежений розмір корисного навантаження (payload size) пакетів

LoRaWAN є ще одним суттєвим недоліком для передачі об'ємних даних. Максимальний розмір корисного навантаження в одному пакеті LoRaWAN становить від декількох десятків до приблизно 242 байт, залежно від швидкості передачі та регіональних налаштувань. Це означає, що будь-яке повідомлення, що перевищує цей ліміт, повинно бути фрагментоване на безліч дрібних пакетів, що значно збільшує накладні витрати, складність програмного забезпечення для збірки повідомлень та загальний час передачі.

По-третє, регуляторні обмеження на робочий цикл (duty cycle), що діють в ISM-діапазонах, строго обмежують загальний час, протягом якого пристрій може здійснювати передачу. Для інтенсивного збору даних, що потребує частої відправки інформації, ці обмеження можуть стати критичним вузьким місцем, не дозволяючи передавати дані з необхідною періодичністю або в повному обсязі.

Таким чином, LoRa виявляє себе найкраще у випадках, коли необхідно передавати невеликі пакети даних з дуже низьким енергоспоживанням на відстані до декількох кілометрів, наприклад, для довготривалого моніторингу навколишнього середовища або віддалених об’єктів із обмеженим доступом до джерел живлення.

2.3.4. Інтерфейс SPI

SPI – це синхронний послідовний інтерфейс передачі даних, який широко використовується для зв'язку мікроконтролерів з периферійними пристроями на коротких відстанях, зокрема в межах однієї друкованої плати. Його архітектура базується на принципі "ведучий-ведений" (master-slave), де один пристрій (зазвичай мікроконтролер, у нашому випадку це може бути як Raspberry Pi, так і ESP32) виступає в ролі ведучого та ініціює всі сеанси зв'язку. Інші пристрої є веденими.

Основні характеристики та лінії зв'язку SPI:

- Інтерфейс SPI зазвичай використовує чотири основні логічні сигнали:

- SCLK (Serial Clock): Тактовий сигнал, що генерується ведучим пристроєм для синхронізації передачі даних.
- MOSI (Master Out Slave In): Лінія для передачі даних від ведучого до веденого.
- MISO (Master In Slave Out): Лінія для передачі даних від веденого до ведучого.
- CS (Chip Select) або SS (Slave Select): Сигнал вибору веденого пристрою. Ведучий активує цю лінію (зазвичай переводячи її в низький логічний рівень) для того веденого, з яким він має намір обмінюватися даними. Кожен ведений пристрій на шині SPI потребує окремої лінії CS [12].

Завдяки наявності окремих ліній для передачі даних у кожному напрямку (MOSI та MISO), SPI підтримує повнодуплексний режим зв'язку, дозволяючи одночасну передачу та прийом даних. Це є суттєвою перевагою для застосувань, що вимагають високої пропускної здатності.

Інтерфейс SPI має кілька значних переваг. По-перше, це висока швидкість передачі даних: SPI може працювати на швидкостях, що значно перевищують швидкості UART або стандартного режиму I2C, досягаючи десятків Мбіт/с. Це робить його ідеальним для передачі великих обсягів даних, таких як дані з АЦП, потокове аудіо/відео або обмін даними з картами пам'яті. По-друге, протокол SPI є відносно простим на апаратному рівні, не вимагаючи складних механізмів адресації чи підтвердження прийому даних на рівні самого протоколу, хоча це може бути реалізовано програмно. По-третє, повнодуплексний зв'язок, тобто можливість одночасної передачі та прийому даних, підвищує ефективність використання шини.

Також SPI пропонує гнучкість у виборі швидкості та формату даних, оскільки параметри зв'язку, такі як частота тактового сигналу та порядок біт, можуть бути налаштовані ведучим.

Поряд з перевагами, існують і певні недоліки та аспекти, які слід враховувати при використанні SPI. Більша кількість сигнальних ліній, порівняно з I2C (2 лінії) або UART (2-3 лінії), SPI потребує щонайменше 4 лінії для одного веденого пристрою, і кількість необхідних виводів мікроконтролера зростає з кожним додатковим веденим пристроєм через окремі лінії CS. Також варто

відзначити відсутність апаратного підтвердження прийому: на відміну від I2C, SPI не має вбудованого механізму підтвердження (ACK/NACK), тому перевірка успішності передачі даних покладається на програмну реалізацію. Крім того, стандартний SPI підтримує лише одного ведучого на шині. Ще може спостерігатися відсутність стандартизації на вищому рівні: хоча базовий механізм передачі біт є стандартним, специфічні команди та формати даних для конкретних SPI-пристроїв можуть сильно відрізнятися [12].

Інтерфейс SPI поєднує високу пропускну здатність та синхронний обмін, що робить його доцільним для підключення ESP32 до швидких багатоканальних сенсорів або карт пам’яті. Водночас кожен пристрій на шині вимагає власної лінії вибору (CS), а довгі траси без підсилювачів сигналу непридатні — це слід врахувати вже на стадії проєктування плати. Отже, SPI варто застосовувати там, де необхідні швидкі й надійні зв’язки всередині компактного модуля, залишаючи для віддалених або менш вимогливих датчиків більш прості інтерфейси на кшталт I²C чи UART.

2.3.5. Інтерфейс I2C

Інтерфейс I2C (Inter-Integrated Circuit), розроблений компанією Philips Semiconductors, нині NXP Semiconductors, являє собою синхронну, багатопровідну, багатоведену, пакетну та напівдуплексну послідовну шину, призначену для ефективної комунікації між мікросхемами на коротких відстанях, переважно в межах однієї друкованої плати. Для організації зв'язку I2C використовує всього дві двонаправлені лінії: SDA (Serial Data Line) для передачі даних та SCL (Serial Clock Line) для тактових сигналів, що синхронізують весь обмін. Особливістю цих ліній є їхня робота за принципом відкритого колектора або відкритого стоку, що означає, що мікросхеми можуть лише активно притягувати лінію до низького логічного рівня ("0"). Для встановлення високого логічного рівня ("1") обидві лінії обов'язково підтягуються до позитивного виводу живлення, наприклад +3.3В або +5В, через зовнішні підтягуючі резистори [13].

Номінал цих резисторів є критично важливим параметром, що залежить від напруги живлення, загальної ємності шини та бажаної швидкості передачі, оскільки саме вони забезпечують високий рівень на шині, коли жоден пристрій не активний.

Кожна транзакція на шині I2C ініціюється ведучим (master) пристроєм. Процес обміну починається з генерації ведучим стартового стану (START condition) – переходу лінії SDA з високого рівня на низький, поки лінія SCL залишається високою. Після цього ведучий надсилає 7-бітну, або в деяких випадках 10-бітну, адресу веденого пристрою, з яким планується обмін. Ця адреса доповнюється восьмим бітом, який вказує напрямок передачі даних: "0" для запису даних веденому (Write) та "1" для читання даних від веденого (Read). Пристрій, що розпізнав свою адресу, повинен підтвердити її прийом, притягуючи лінію SDA до низького рівня протягом дев'ятого тактового імпульсу SCL, що називається бітом підтвердження (Acknowledge - ACK). Якщо лінія SDA залишається високою протягом цього дев'ятого такту, це інтерпретується як відсутність підтвердження (Not Acknowledge - NACK), що може свідчити про помилку, відсутність пристрою, його зайнятість або, у випадку читання ведучим, сигналом про завершення прийому даних. Сама передача даних відбувається байтами (по 8 біт), причому першим завжди передається старший біт (MSB). Кожен переданий або отриманий байт даних також повинен супроводжуватися бітом ACK або NACK від приймаючої сторони. Важливо, щоб дані на лінії SDA були стабільними протягом усього часу, поки лінія SCL знаходиться у високому рівні; зміна даних дозволена лише тоді, коли SCL перебуває в низькому рівні. Завершується транзакція генерацією ведучим стопового стану (STOP condition) – переходом лінії SDA з низького рівня на високий, поки SCL залишається високою, після чого шина вважається вільною. Ведучий також може використовувати повторний стартовий стан (Repeated START) замість стопового, щоб розпочати нову транзакцію, не звільняючи шину, наприклад, для зміни напрямку передачі або звернення до іншого пристрою [13].

I2C підтримує декілька режимів швидкості для обміну даними, серед яких Стандартний режим (Standard Mode, Sm) до 100 кбіт/с, Швидкий режим (Fast Mode, Fm) до 400 кбіт/с, та Швидкий режим плюс (Fast Mode Plus, Fm+) до 1 Мбіт/с. Існує також Високошвидкісний режим (High-Speed Mode, Hs-mode), що дозволяє передачу даних зі швидкістю до 3.4 Мбіт/с, однак його реалізація вимагає спеціальних апаратних рішень, таких як активні підтягуючі елементи або перемикання на струмові джерела під час Hs-передачі, а також використання спеціального "master code" для ініціації цього режиму. Менш поширений, але існуючий, Ultra Fast-mode (UFm) може досягати 5 Мбіт/с, проте він є однонаправленим [13]. Вибір швидкості залежить від можливостей підключених пристроїв та вимог до пропускної здатності, при цьому слід враховувати, що вищі швидкості накладають жорсткіші обмеження на максимальну ємність шини та якість сигналу, що, в свою чергу, залежить від довжини провідників, кількості пристроїв та правильності вибору підтягуючих резисторів.

Протокол I2C також передбачає розширені можливості, такі як підтримка декількох ведучих пристроїв на одній шині (multi-master). У цьому випадку використовується механізм арбітражу: якщо два або більше ведучих починають передачу одночасно, той, хто першим спробує передати логічну "1" (відпустивши лінію), в той час як інший ведучий передає "0" (активно притягуючи лінію до землі), програє арбітраж, припиняє свою передачу і переходить в режим очікування, дозволяючи іншому завершити транзакцію. Ще однією корисною функцією є розтягування тактового сигналу (clock stretching), коли ведений пристрій може тимчасово призупинити обмін даними, утримуючи лінію SCL в низькому стані після того, як ведучий її відпустив. Це дає повільним веденим пристроям необхідний додатковий час для обробки отриманих даних або підготовки відповіді.

Така архітектура надає інтерфейсу I2C значні переваги, серед яких головними є економічність використання виводів мікроконтролера – лише два незалежно від кількості підключених пристроїв (в межах адресного простору), що особливо цінно для систем з обмеженою кількістю GPIO. Також важливими

плюсами є простота підключення декількох пристроїв до однієї шини та наявність вбудованого механізму підтвердження прийому даних. Протокол добре підходить для керування та збору даних з відносно повільних пристроїв, де максимальна швидкість передачі не є критичним фактором. Однак, I2C має і певні обмеження. Швидкість передачі даних зазвичай нижча, ніж у альтернативних інтерфейсів, таких як SPI. Складність програмної реалізації може зростати при роботі з великою кількістю пристроїв, використанні 10-бітної адресації, у конфігураціях з кількома ведучими або при необхідності детальної обробки всіх можливих станів шини. Необхідність використання підтягуючих резисторів та їх правильний розрахунок є важливим аспектом при проєктуванні шини, оскільки неправильний вибір може призвести до помилок або нестабільної роботи.

На практиці, інтерфейс I2C є надзвичайно актуальним і широко використовується для підключення різноманітних сенсорів та допоміжних модулів як до мікроконтролерів типу ESP32, так і до одноплатних комп'ютерів, наприклад, Raspberry Pi. ESP32 зазвичай оснащений кількома апаратними I2C контролерами, які можуть бути гнучко призначені на різні GPIO піни, а робота з ними в середовищах розробки, таких як ESP-IDF або Arduino IDE, спрощується завдяки наявності спеціалізованих бібліотек, наприклад Wire.h для Arduino. Raspberry Pi також має апаратні I2C інтерфейси, доступні на GPIO роз'ємі (найчастіше I2C1 на пінах GPIO2 (SDA) та GPIO3 (SCL)), а доступ до них в операційній системі Raspberry Pi OS здійснюється через ядро Linux (у вигляді пристроїв /dev/i2c-X) та за допомогою утиліт командного рядка, таких як i2c-tools (що включають i2cdetect, i2cget, i2cset), або через програмні бібліотеки для мов типу Python, наприклад smbus2. Типовими пристроями, що підключаються через I2C, є інерційні вимірювальні блоки (IMU) для визначення орієнтації, датчики температури, вологості, тиску, освітленості, а також модулі реального часу (RTC), розширювачі портів вводу/виводу, аналого-цифрові та цифро-аналогові перетворювачі, EEPROM та Flash пам'ять, контролери для LCD та OLED дисплеїв, драйвери світлодіодів та мікросхеми керування живленням. При відлагодженні I2C комунікацій часто виникають проблеми, пов'язані з

неправильними або відсутніми підтягуючими резисторами, конфліктами адрес пристроїв, помилками в послідовності команд протоколу або відсутністю підтвердження від веденого пристрою. Для ефективної діагностики таких проблем незамінними інструментами є логічний аналізатор або осцилограф, які дозволяють візуально спостерігати за сигналами на лініях SDA та SCL.

Незважаючи на певні обмеження, особливо щодо максимальної швидкості, I2C залишається надзвичайно популярним та виправданим вибором для підключення широкого спектра периферійних пристроїв у вбудованих системах завдяки своїй простоті реалізації, мінімальній кількості необхідних сигнальних ліній та надійним вбудованим механізмам адресації й підтвердження.

2.3.6. UART

Універсальний асинхронний приймач-передавач (UART) є одним із фундаментальних та найпоширеніших послідовних інтерфейсів, що використовуються для обміну даними між мікроконтролерами та різноманітними периферійними пристроями або іншими обчислювальними модулями. Його популярність зумовлена простотою апаратної реалізації, низькою вартістю та достатньою надійністю для багатьох застосувань, особливо у вбудованих системах, до яких належить і розроблювана мобільна платформа. UART забезпечує послідовну передачу даних, тобто біти інформації передаються один за одним по одній лінії.

Для організації зв'язку через UART зазвичай використовуються дві основні сигнальні лінії: TX (Transmit – передача) та RX (Receive – прийом). Лінія TX одного пристрою з'єднується з лінією RX іншого, і навпаки, що дозволяє реалізувати повнодуплексний режим зв'язку, при якому дані можуть одночасно передаватися та прийматися. Окрім основних ліній TX та RX, деякі реалізації UART можуть використовувати додаткові сигнали для апаратного керування потоком даних (Hardware Flow Control), такі як RTS (Request To Send) та CTS (Clear To Send). Лінія RTS одного пристрою сигналізує іншому про готовність

приймати дані, а лінія CTS підтверджує, що інший пристрій готовий до прийому [14]. Це допомагає запобігти втраті даних через переповнення буфера приймача, особливо при високих швидкостях передачі або коли один із пристроїв може бути тимчасово зайнятий іншими завданнями. У контексті розроблюваної мобільної платформи, де обсяги даних та швидкість обробки є контрольованими, використання апаратного керування потоком може не бути критичним, але його наявність є перевагою для забезпечення максимальної надійності.

Ключовою особливістю UART є його асинхронний характер передачі. Це означає, що між передавачем та приймачем відсутній спільний тактовий сигнал для синхронізації. Замість цього, синхронізація кожного байта даних досягається за допомогою стартового біта, який передує інформаційним бітам, та одного або кількох стопових бітів, які завершують передачу байта. Для успішного обміну даними обидва пристрої повинні бути налаштовані на однакові параметри зв'язку: швидкість передачі (baud rate), кількість біт даних (зазвичай 8), наявність та тип біта парності (часто не використовується для спрощення) та кількість стопових бітів.

Незважаючи на узгоджені параметри, під час передачі даних через UART можуть виникати певні типи помилок. До найпоширеніших належать:

- Помилка кадрування (Framing Error): виникає, коли приймач не виявляє очікуваного стопового біта, що може свідчити про розсинхронізацію або шум на лінії.
- Помилка переповнення (Overrun Error): стається, якщо нові дані надходять до UART-приймача швидше, ніж центральний процесор встигає їх зчитати з буфера приймача.
- Помилка парності (Parity Error): якщо біт парності використовується для перевірки цілісності, ця помилка сигналізує про невідповідність отриманого біта парності розрахованому значенню, вказуючи на можливе спотворення одного з бітів даних. Обробка цих помилок залежить від вимог конкретного застосування. У деяких випадках, особливо при передачі некритичних даних або даних, де цілісність

перевіряється на вищому рівні протоколу (наприклад, контрольними сумами в JSON-пакетах), деякі з цих апаратних помилок можуть ігноруватися. В інших системах, виявлення помилок може ініціювати запит на повторну передачу даних або інші механізми відновлення для забезпечення надійності.

Серед переваг UART варто відзначити його простоту та надійність на коротких відстанях, що цілком достатньо для внутрішньоплатформенного зв'язку. Він забезпечує повнодуплексний обмін, що дозволяє одночасно передавати команди від Raspberry Pi до ESP32 (наприклад, для ініціації певних дій) та отримувати дані або статусні повідомлення від ESP32. Також UART часто використовується для відлагоджувальних цілей, надаючи прямий доступ до консолі пристрою.

Однак UART має і певні недоліки. Як асинхронний інтерфейс, він не має окремої лінії тактування, що робить його більш чутливим до розбіжностей у налаштуваннях швидкості між пристроями. На великих відстанях або при високих швидкостях передачі даних UART може бути вразливим до електромагнітних завад, що може призводити до помилок, як-от згадані помилки кадрування. У нашому випадку, коли зв'язок відбувається між компонентами, розташованими близько один до одного на одній платформі, цей недолік мінімізується. Максимальна стандартна швидкість у 115200 біт/с є достатньою для передачі структурованих даних з сенсорів у форматі JSON, як це реалізовано в проєкті, без створення значних затримок. Хоча теоретично можливі й вищі швидкості (до 1-4 Мбіт/с за спеціальних налаштувань), для забезпечення стабільності було обрано стандартне значення.

При реалізації UART-зв'язку між ESP32 та Raspberry Pi в рамках розроблюваної мобільної платформи, важливо враховувати специфіку використання UART на цих мікроконтролерах. Обидва пристрої оснащені кількома апаратними UART-портами. Наприклад, ESP32 зазвичай має три або чотири UART-контролери (UART0, UART1, UART2 тощо), що надає гнучкість у виборі порту для комунікації, залежно від інших використовуваних периферійних

пристроїв та їхніх підключень. Raspberry Pi також має декілька UART, проте основний (зазвичай /dev/ttyS0 на старіших моделях або /dev/serial0, що є символічним посиланням на основний UART, на новіших моделях) може бути за замовчуванням зайнятий системною консоллю Linux для виведення повідомлень завантажувача та системи. Для стабільного зв'язку з ESP32 може знадобитися або переналаштування цього порту (наприклад, відключення консолі через конфігураційні файли системи, такі як cmdline.txt та config.txt), або використання іншого доступного UART-інтерфейсу (наприклад, одного з додаткових UART, які можуть бути активовані через Device Tree Overlays та виведені на GPIO-піни). Правильне налаштування портів, їх ініціалізація в програмному забезпеченні обох пристроїв та узгодження логічних рівнів (зазвичай 3.3В для обох пристроїв, що спрощує пряме з'єднання) є ключовим для надійної передачі даних.

Таким чином, UART є ефективним та прагматичним рішенням для організації каналу передачі даних між ESP32 та Raspberry Pi в рамках розроблюваної мобільної платформи, забезпечуючи необхідний баланс між простотою реалізації, швидкістю передачі та надійністю для поставлених завдань.

2.3.7. Хмарні технології

Хмарні технології (cloud computing) стали невід’ємною частиною автоматизованих систем керування, особливо в сфері робототехніки. Вони відкривають нові можливості для обробки великих обсягів даних у режимі реального часу, підвищують ефективність взаємодії між компонентами системи та спрощують масштабування обчислювальних процесів. Ідея полягає в тому, що користувачі можуть отримувати доступ до серверів, баз даних, сховищ і аналітичних інструментів через інтернет, не маючи потреби у встановленні та технічному обслуговуванні локального обладнання. Такий підхід дозволяє створювати розподілені системи, де окремі пристрої — сенсори, контролери, роботизовані платформи — взаємодіють з хмарною інфраструктурою, передаючи дані для зберігання, обробки та аналізу.

Хмарні обчислення ґрунтуються на моделі розподілених систем, у яких ресурси централізовано розташовані на дата-центрах, а доступ до них здійснюється через інтерфейси прикладного програмування (API) або вебсервіси. Основними моделями хмарних сервісів є:

- IaaS (Infrastructure as a Service) — надання в оренду віртуалізованих обчислювальних потужностей (наприклад, Amazon EC2, Google Compute Engine);
- PaaS (Platform as a Service) — середовище для розробки й розгортання додатків без потреби адмініструвати інфраструктуру (наприклад, Google App Engine, Microsoft Azure);
- SaaS (Software as a Service) — готові до використання онлайн-сервіси (наприклад, Dropbox, Google Drive, Microsoft 365).

Хмарні середовища можуть бути публічними (загальнодоступні сервіси), приватними (локальні рішення для організацій) або гібридними, що поєднують елементи обох моделей. Залежно від потреб проєкту вибір відповідної архітектури впливає на рівень безпеки, швидкодію та вартість обслуговування системи.

У контексті мобільних роботизованих платформ хмарні технології виконують роль «віддаленого мозку» — вони зберігають дані, приймають або ініціюють команди, виконують аналітичну обробку й формують зворотний зв’язок для системи управління. Це особливо актуально в умовах, коли пристрої мають обмежені ресурси для зберігання або обробки великих обсягів інформації, а також коли потрібен централізований аналіз даних із кількох платформ.

Завдяки хмарним рішенням автоматизовані системи здатні працювати за принципом «відкладеної обробки»: спочатку збираючи інформацію локально, а згодом передаючи її до хмари, коли з’являється доступ до інтернету. Такий механізм особливо важливий для мобільних платформ та автономних пристроїв, що виконують завдання в ізольованих районах — наприклад, у сільськогосподарських полях, гірських регіонах або зонах з нестабільним покриттям.

У таких системах локальні обчислювальні ресурси зазвичай обмежені: мікроконтролери (на зразок ESP32) мають невеликий обсяг пам’яті, обмежену обчислювальну потужність і не здатні зберігати або обробляти великі масиви історичних даних. Хмарна інфраструктура дозволяє перенести ці функції на потужні віддалені сервери. Замість зберігання даних локально, пристрої лише збирають та періодично передають дані на сервер, де вже виконується глибший аналіз, візуалізація або реагування на критичні значення.

Це відкриває можливості для реалізації складних сценаріїв, зокрема:

- Прогнозування на основі зібраних даних — наприклад, в агросекторі це аналіз вологості ґрунту та рекомендації щодо зрошення;
- Віддалений моніторинг — оператор може слідкувати за станом мобільної платформи або сенсорної мережі в режимі реального часу;
- Оновлення налаштувань у польових умовах — сервер надсилає нові координати або зміни маршруту, які автоматично застосовуються пристроєм після наступного сеансу зв’язку.

У сфері екологічного моніторингу, автономні платформи (наземні чи повітряні) збирають інформацію про рівень забруднення повітря, наявність шкідливих речовин, температуру, вологість або інші параметри. Хмара дозволяє зібрати ці дані в одну базу, побудувати карти розподілу показників, відстежити динаміку змін і, за необхідності, автоматично сповістити відповідальні органи.

Важливо зазначити, що хмара не лише виконує роль сховища, але й може виступати як аналітичний центр. Інтеграція зі службами машинного навчання (наприклад, Google Cloud AI або AWS SageMaker) дає змогу автоматично виявляти закономірності, аномалії або передбачати зміни на основі зібраної історичної інформації [15], [16].

Таким чином, хмарні технології формують інтелектуальний рівень системи автоматизації, де сенсори, платформи й оператори взаємодіють у єдиному цифровому середовищі, незалежно від географічного розташування. Це особливо важливо для проєктів, орієнтованих на моніторинг довкілля, управління

розподіленими ресурсами та автономне функціонування систем у складних середовищах.

З розвитком інтернету речей та мобільної робототехніки з'явився широкий спектр хмарних платформ, орієнтованих на обробку, зберігання та візуалізацію даних, що надходять від сенсорних пристроїв і автономних систем. Вибір тієї чи іншої платформи залежить від масштабів проєкту, технічних вимог, рівня безпеки, а також можливості інтеграції з мікроконтролерами або одноплатними комп'ютерами.

Одне з наймасштабніших рішень — AWS IoT Core, яке дозволяє безпечно підключати мільйони пристроїв, збирати їхні дані та обробляти їх у режимі реального часу. Платформа підтримує MQTT, HTTPS, WebSockets, надає засоби для створення правил, аналітики, візуалізації, а також можливість підключення до AWS Lambda для обробки подій. AWS добре підходить для великих промислових і комерційних проєктів, але має поріг входу внаслідок складності налаштування та ціноутворення [15].

Схожі можливості пропонує Microsoft Azure IoT Hub — платформа, що дозволяє двосторонню комунікацію між пристроями та хмарою, з можливістю використання Edge-модулів для локальної обробки даних. Azure особливо корисна в екосистемі Microsoft (наприклад, для візуалізації через Power BI або обробки даних у Azure Machine Learning).

Хоча Google Cloud IoT Core було офіційно припинено у 2023 році, вона мала активну спільноту і використовувалась для проєктів з високими вимогами до аналітики та масштабування. Google залишила інші сервіси, зокрема Firebase та BigQuery, які можуть бути використані у якості бекенду для мобільних платформ [16].

Для невеликих або середніх проєктів чудовим варіантом є Firebase від Google — це платформа, яка включає реальну базу даних у реальному часі (Realtime Database), автентифікацію, функції хмар (Cloud Functions) та інструменти для побудови веб- і мобільних додатків. Firebase підтримує передачу

даних через HTTPS-запити, що дозволяє легко інтегруватися з Raspberry Pi або ESP32 безпосередньо.

ThingsBoard — це open-source IoT-платформа, яка дозволяє візуалізувати телеметрію, керувати пристроями, будувати дашборди та сценарії автоматизації. Вона підтримує MQTT, CoAP, HTTP API, що робить її сумісною з більшістю IoT-пристроїв. Може бути розгорнута локально або в хмарі, що особливо зручно для освітніх або дослідницьких проєктів з обмеженим бюджетом.

Blynk — популярна платформа для швидкого створення інтерфейсів керування через мобільні додатки. Вона ідеально підходить для мікроконтролерних платформ, таких як ESP32, дозволяючи створити мобільний додаток для моніторингу та керування пристроями без написання коду на стороні клієнта. Платформа надає REST API та підтримує MQTT, зберігає історію даних і дозволяє налаштовувати автоматичні події.

Для систем, де критичною є мінімальна затримка обміну даними та стабільне з’єднання між пристроєм і сервером, ефективним рішенням виступає використання MQTT-брокерів. Протокол MQTT (Message Queuing Telemetry Transport) ідеально пристосований до потреб мобільних або енергообмежених платформ, оскільки забезпечує легкий, швидкий та надійний механізм обміну повідомленнями з мінімальним навантаженням на мережу. Такі програмні рішення, як Mosquitto або HiveMQ, можуть бути впроваджені як локально, так і в хмарному середовищі, що дає змогу адаптувати систему до конкретних умов експлуатації.

Окрему увагу в роботі з хмарними технологіями необхідно приділяти захисту даних, що набуває особливої ваги у випадках використання автономних мобільних систем. Такі системи часто працюють у відкритому середовищі або оперують чутливою інформацією, тож питання інформаційної безпеки стає принциповим. Незахищені канали зв’язку, слабкі механізми автентифікації чи відсутність шифрування можуть спричинити витік, спотворення або втрату даних, що є неприпустимим для критично важливих застосувань — зокрема у військовій, екологічній або промисловій сферах.

Початковий рівень безпеки досягається шляхом шифрування даних безпосередньо на пристрої до моменту їх збереження або передачі. Наприклад, в автономних мобільних платформах дані, зібрані з сенсорів і записані на SD-карту, можуть бути захищені за допомогою симетричних алгоритмів шифрування, таких як AES (Advanced Encryption Standard) чи ChaCha20. Це забезпечує конфіденційність інформації навіть у разі фізичної втрати носія. Додаткове зміцнення системи безпеки може включати використання одноразових криптографічних ключів (nonce) для кожного запису або впровадження електронного підпису, який підтверджує автентичність і цілісність даних.

На рівні передачі даних до хмари критично важливо використовувати протоколи, які забезпечують шифрування трафіку. Одним із найпоширеніших рішень є HTTPS — варіант протоколу HTTP із підтримкою SSL/TLS, що гарантує захищене з'єднання між клієнтом і сервером. У разі застосування MQTT-протоколу використовується його зашифрована версія — MQTT over TLS. Важливим кроком є також перевірка дійсності сертифіката сервера, щоб запобігти можливим атакам типу “людина посередині” (man-in-the-middle).

Ще одним критичним елементом є автентифікація пристроїв, які надсилають дані до хмари. Кожна платформа або сенсор повинні мати унікальний ключ доступу, що дозволяє серверу підтверджувати їхню справжність. У системах малого масштабу зазвичай використовуються API токени або JSON Web Tokens (JWT), тоді як у корпоративних рішеннях можуть застосовуватись клієнтські TLS-сертифікати або фільтрація за MAC-адресами чи IP. Це дозволяє формувати “білий список” дозволених пристроїв і запобігає несанкціонованому надсиланню даних.

Після отримання даних хмарною системою їх захист не повинен припинятись. Надійні платформи, як-от Firebase або Azure, забезпечують шифрування баз даних у стані спокою (at rest), наприклад, за допомогою AES-256. Для контролю доступу до інформації застосовується розмежування прав між користувачами та сервісами, що дозволяє обмежити перегляд, зміну чи видалення даних лише авторизованим суб’єктам. Додатково впроваджується журналювання

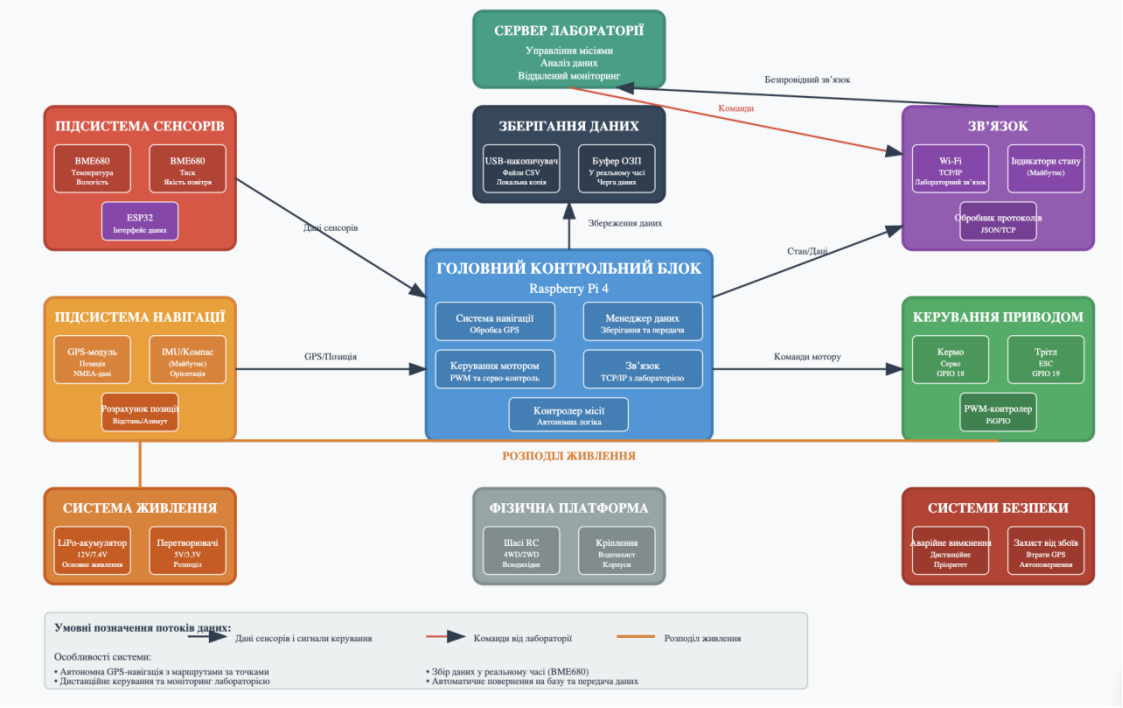
всіх дій із даними (audit trail), що дає змогу виявити підозрілу активність або помилки в роботі системи. Регулярне створення резервних копій дозволяє швидко відновити інформацію у разі збоїв або атак.

Окрім цифрових засобів захисту, важливо враховувати і фізичну безпеку пристроїв. Мікроконтролери, SD-картки, інтерфейси передачі даних мають бути фізично захищені від несанкціонованого доступу. У Raspberry Pi слід реалізувати налаштування міжмережевого екрану (брандмауера), обмежити порти, а також вмикати автентифікацію доступу по SSH. У системі варто реалізовувати механізми самовідновлення: наприклад, watchdog-таймери або автоматичне перезавантаження у разі втрати зв’язку чи збоїв у файловій системі.

РОЗДІЛ 3
РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ УПРАВЛІННЯ РУХОМОЮ ПЛАТФОРМОЮ

3.1. Структура та функції системи

В основі розробленої автоматизованої платформи лежить взаємодія двох ключових апаратних модулів: мікроконтролера ESP32-WROOM-32 та одноплатного комп'ютера Raspberry Pi 4B. Така архітектура забезпечує розподіл обов'язків: ESP32 відповідає за безпосереднє зчитування показників довкілля, а Raspberry Pi агрегує їх, зберігає та передає до наукової лабораторії (рис. 3.1.1).



Рисю 3.1.1 – Блок-схема автоматизованої платформи

ESP32 під'єднано до датчика BME680 (рис.3.1.1.2), який дозволяє вимірювати температуру, відносну вологість, атмосферний тиск та якість повітря (газовий опір та розрахований індекс IAQ). Інтерфейс зв'язку між ESP32 і

BME680 реалізовано через шину I²C на 3.3 В; прошивка на C/C++ передбачає обробку винятків у разі невдалого опитування датчика та логування аномальних значень. Для забезпечення стабільності даних ESP32 формує записи у вигляді raw-рядків і передає їх через USB-UART на Raspberry Pi (див. Додаток А).

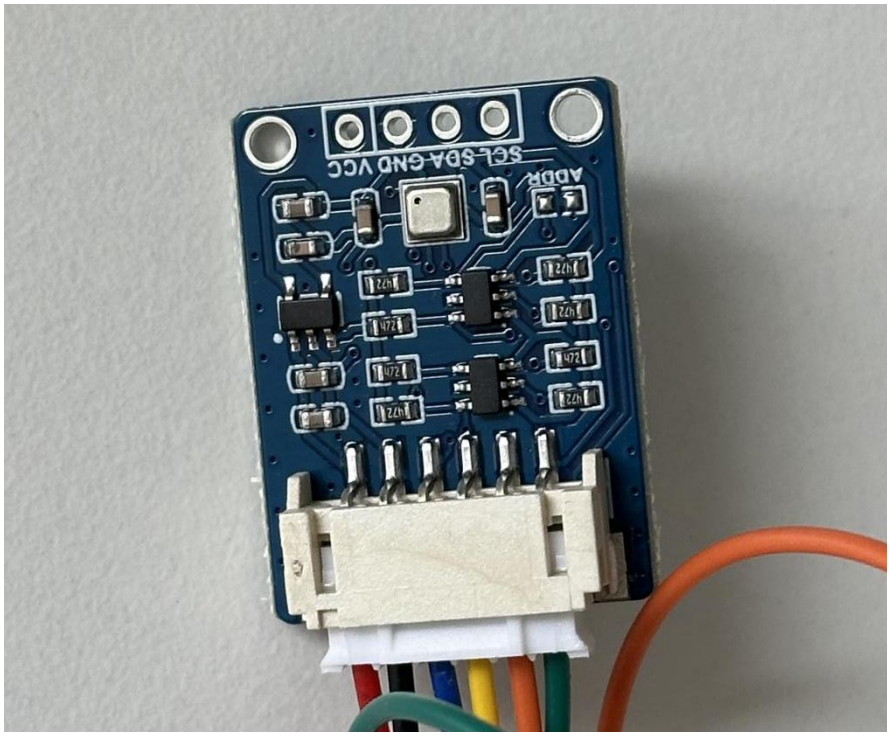


Рис.3.1.1.2 – Модуль навколишнього середовища BME680

Raspberry Pi 4B під’єднується до ESP32 через стандартний USB-кабель, що створює в системі віртуальний послідовний порт (наприклад, `/dev/ttyUSB0`). Після отримання raw-пакету Python-скрипт аналізує та перевіряє його структуру, зберігаючи в тимчасовий буферний файл на зовнішньому накопичувачі (USB-флешка або SD-картка через USB-адаптер). Після успішного завершення запису тимчасовий файл перейменовується і переміщується до каталогу архівів. Така двоетапна схема запису мінімізує ризик втрати даних у разі знеструмлення або відключення накопичувача.

Навігація платформи реалізована за допомогою GPS-модуля GT-U7 (рис. 3.1.1.3), підключеного безпосередньо до апаратного UART на GPIO14 (RXD) і GPIO15 (TXD) Raspberry Pi. Отримання координат підвищує автономність: платформа рухається за заздалегідь завантаженим маршрутом із кількома

контрольними точками й автоматично повертається на базу у разі виявлення аварійної ситуації (наприклад, втрати сигналу GPS понад заданий поріг) (див. Додаток Б).

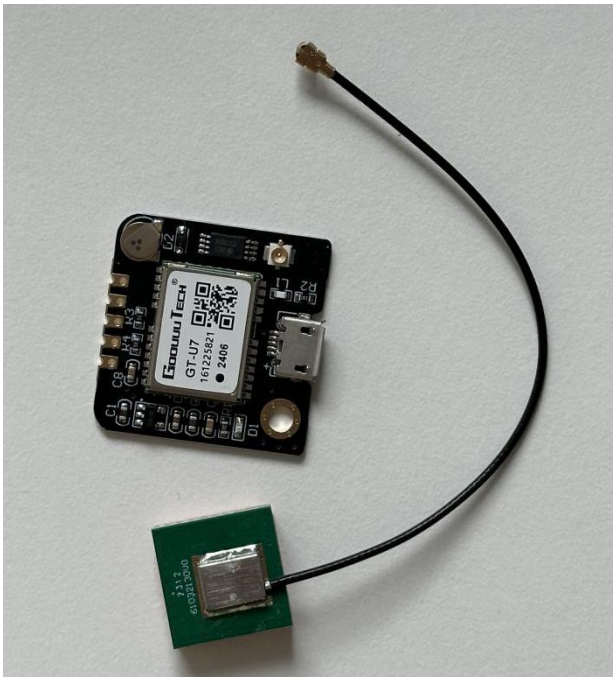


Рис. 3.1.1.3 – Модуль GPS GT-U7

Живлення системи поділено на два незалежні підсистеми. Логічна частина (Raspberry Pi та периферійні пристрої на 5 В) отримує живлення від окремого літій-полімерного акумулятора через стабілізатор 5 В, що запобігає заваджанням від двигунів. Двигуни ESC живляться від батареї 7.2 В. Однак потенціали «землі» (GND) обох джерел обов’язково з’єднані для запобігання зсувам рівня сигналів.

Рух платформи контролюється Python-скриптом на Raspberry Pi, який, окрім команд навігації через TCP/IP від серверу лабораторії, відповідає за плавне прискорення/уповільнення через ESC та моніторинг поточного стану (швидкість, кут повороту). За потреби скрипт надсилає до лабораторії повідомлення про координати, стан живлення та показники довкілля.

Сервер у кваліфікайній лабораторії реалізовано як Python-додаток, що прослуховує TCP/IP-з’єднання на заздалегідь заданому порту та приймає пакети в JSON-форматі. Після отримання кожного пакету дані перевіряються й зберігаються в реляційній базі даних для швидкого доступу до історії вимірювань

і аналітики. Тією ж ланкою оператор може створювати нові маршрути, коригувати координати або надсилати аварійні команди (див. Додаток В).

Отже, побудована система інтегрує кілька взаємопов’язаних підсистем: модуль збору даних на базі ESP32 із датчиком BME680, модуль обробки та зберігання на Raspberry Pi із зовнішнім USB-накопичувачем, навігаційний модуль на основі GPS GT-U7, поділену систему живлення з двома джерелами (5 В для логіки та 7.2 В для приводів) з об’єднаною “землею” та TCP/IP-канал із вбудованим механізмом автоматичного перепідключення для зв’язку з лабораторією.

3.2. Вибір апаратного забезпечення

При виборі компонентів для автоматизованої рухомої платформи головними критеріями стали надійність у польових умовах, енергетична автономність і достатня обчислювальна потужність для обробки як сенсорних даних, так і навігаційних алгоритмів. Серцем системи обрано одноплатний комп’ютер Raspberry Pi 4B, який поєднує в собі чотириядерний процесор із тактовою частотою 1,5 ГГц, 4 ГБ оперативної пам’яті та широкий набір інтерфейсів — USB 3.0 для зовнішніх накопичувачів, апаратний UART для GPS, Ethernet та вбудований Wi-Fi. Ця конфігурація гарантує достатню продуктивність для безперервного запису й обробки даних із показниками довкілля, реалізації TCP/IP-з’єднання з лабораторією та виконання алгоритмів керування моторами через ESC. Підтримка Linux (Raspbian) і багата екосистема програмних бібліотек спрощують розробку й відлагодження коду, а також забезпечують захищений доступ через SSH.

Для безпосереднього зчитування показників довкілля було обрано мікроконтролер ESP32-WROOM-32. Його двоядерний процесор (240 МГц), вбудовані інтерфейси I2C та SPI та можливість організації USB-UART під’єднання роблять ESP32 оптимальним модулем для опитування датчика BME680. BME680 об’єднує в одному корпусі чотири сенсори: температури,

відносної вологості, атмосферного тиску та газового опору з розрахунком індексу якості повітря [3]. Це дозволило мінімізувати кількість зовнішніх компонентів, знизити енергоспоживання та полегшити монтаж, водночас забезпечивши багатофакторний моніторинг екологічних параметрів.

Навігаційний модуль GT-U7 підключено до апаратного UART Raspberry Pi на GPIO14 (RXD) і GPIO15 (TXD). Цей GPS-приймач вирізняється високою чутливістю і стабільним прийомом сигналу в умовах відкритих просторів, необхідних для польових досліджень. Він підтримує поширений протокол NMEA та дозволяє одержувати координати з точністю до кількох метрів. Обраний саме GT-U7 через його низьку вартість, простоту конфігурації та наявність інтегрованої антени в компактному корпусі [4].

Запис даних організовано на зовнішню SD-картку, підключену до Raspberry Pi через USB-адаптер. Такий підхід дає змогу фізично відокремити сховище даних від системного розділу ОС, уникаючи конфліктів із завантажувальним microSD Raspbian. USB-накопичувач легко замінити в польових умовах і забезпечує широкі можливості для форматування у файлові системи ext4 або FAT32, залежно від вимог до швидкості та сумісності.

Живлення побудовано на двох незалежних джерелах: 5 В стабілізованого виходу літій-полімерного акумулятора для Raspberry Pi та ESP32, а також 7,2 В літієвого пакета для ESC і моторів. Розділення ліній живлення виключає вплив пікових струмових навантажень приводів на логіку, а об’єднана «земля» запобігає зривам рівня опорної напруги в цифрових інтерфейсах.

Силовими елементами платформи обрано безщіткові двигуни зі спеціалізованими ESC, сумісними з батареями 7,2 В. Цей вибір забезпечив високу ефективність і довговічність приводів, що особливо важливо для роботи в нерівному ландшафті. ESC під’єднано до Raspberry Pi через GPIO-виходи з функцією широтно-імпульсної модуляції, що дозволяє реалізувати плавне регулювання швидкості й гальмування.

В якості базового шасі для мобільної платформи було обрано радіокерований шорт-корс-трак ECX Torment масштабу 1/10 (рис. 3.2.1). Це шасі

виготовлене з високостійкого полімерного композиту і оснащене незалежною пружинно-амортизаційною підвіскою на кожному колесі, що забезпечує плавний рух та стійкість на нерівній поверхні. Заводська конструкція передбачає задній привід і штатний посадковий простір для установки електронного регулятора швидкості (ESC), серводвигуна та живильного акумулятора, що значно спрощує інтеграцію модулів ESP32, Raspberry Pi і GPS.

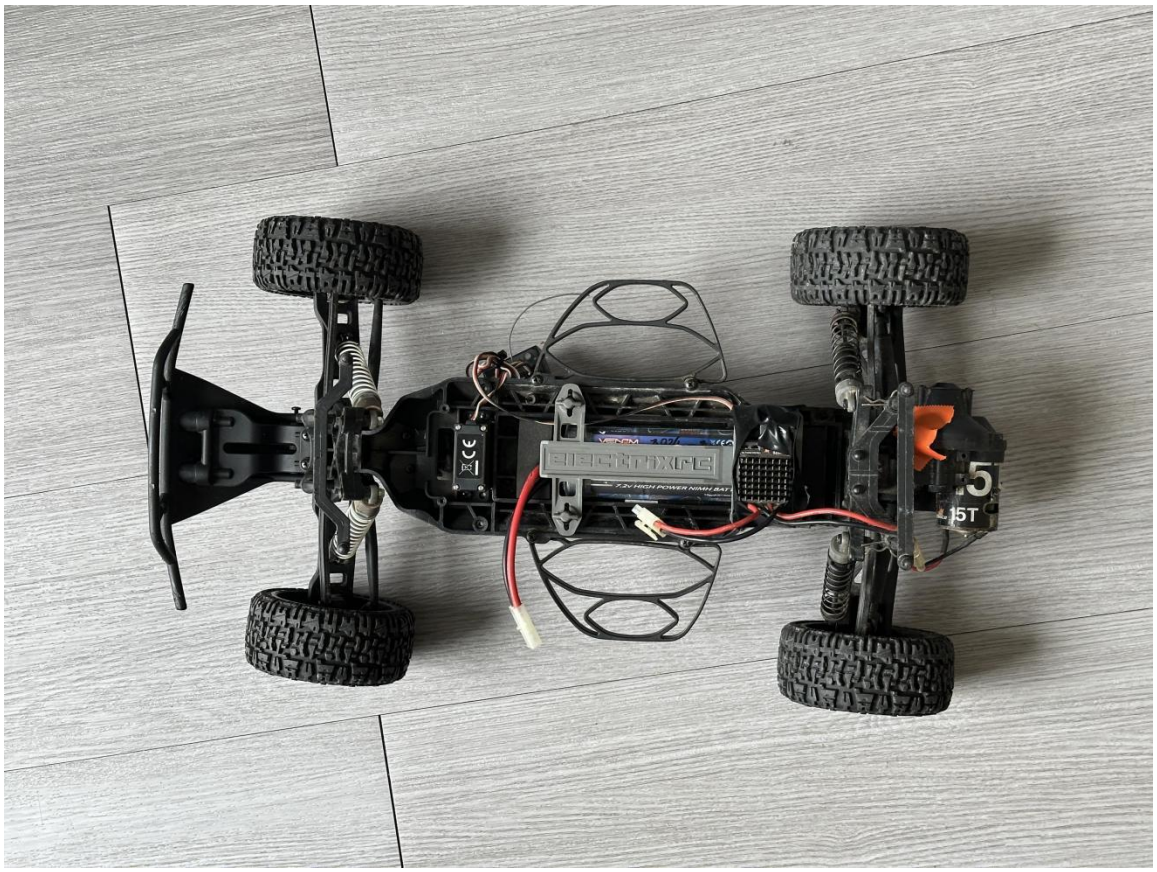


Рис. 3.2.1 – Базове RC-шасі з заднім приводом

У результаті поєднання обраних компонентів створена гнучка, модульна архітектура, яка забезпечує надійну роботу в польових умовах, достатню обчислювальну потужність для реалізації складних алгоритмів та високу ефективність збирання, зберігання й передачі екологічних даних. Якщо в майбутньому знадобиться заміна сенсорів або розширення можливостей комунікації, достатньо буде оновити відповідний модуль без глобальних змін у всій системі.

Фото готового прототипу з усіма змонтованими модулями зображенно на рис. 3.2.2.

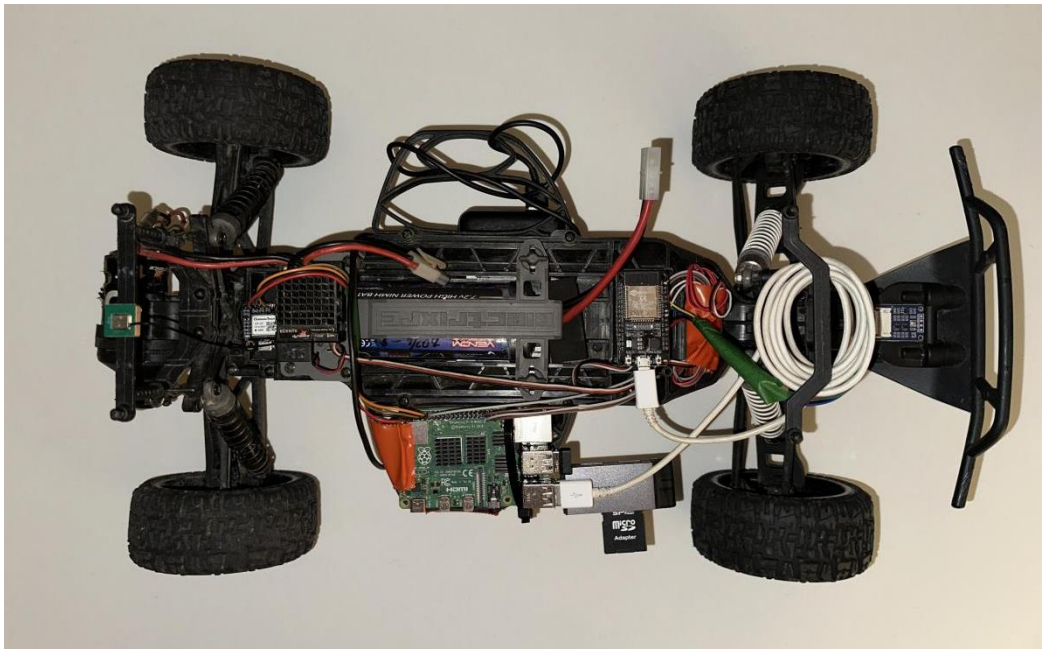


Рис. 3.2.2 – Готовий прототип розробленої автоматизованої рухомої платформи

3.3. Вибір середовищ розробки та інструментів прототипування

Для кожного з ключових компонентів системи було обрано специфічний інструментарій для розробки програмного та апаратного забезпечення для інтергованої мобільної платформи. Даний підхід дозволив ефективно реалізувати функціонал як на низькому рівні мікроконтролера, так і на рівні операційної системи одноплатного комп'ютера.

Для програмування мікроконтролера ESP32 було обрано мову C++. Основним середовищем розробки слугувала Arduino IDE, яка була налаштована для роботи з архітектурою ESP32 шляхом встановлення відповідного пакету підтримки плат (board support package). Цей вибір дозволив використовувати численні готові бібліотеки та спростити процес компіляції коду і завантаження прошивки на пристрій. Зокрема, для взаємодії з датчиком BME680 була інтегрована спеціалізована бібліотека Adafruit BME680 Library, що значно прискорило розробку модуля збору даних [6].

Програмне забезпечення для Raspberry Pi було розроблено мовою Python безпосередньо в операційній системі Raspberry Pi OS [5]. Як середовище розробки використовувався Thonny IDE, що є стандартним для цієї платформи. Його переваги — легкість, простота та наявність вбудованих інструментів для налагодження коду, що ідеально підходить для швидкого прототипування та тестування скриптів, відповідальних за логіку, навігацію та зв'язок із сервером.

Процес тестування системи базувався на принципах модульного тестування. Для цього було створено окремі програмні скрипти, за допомогою яких кожна функція та кожен апаратний елемент (GPS-модуль, датчики, сервоприводи) перевірялись ізольовано. Такий підхід дозволив послідовно переконатися у працездатності всіх компонентів системи перед їхньою фінальною інтеграцією, що мінімізувало кількість помилок на етапі комплексних випробувань платформи.

3.4. Розробка електричної схеми

У середовищі KiCad була виконана принципова електрична схема мобільної платформи, що реалізує всі живильні й сигнальні ланцюги в єдиному кресленні [7]. Система живлення поділена на дві незалежні шини: перша, стабілізована на 5 В від USB-A портативного акумулятора (Power Bank), живить одноплатний комп’ютер Raspberry Pi та підключені до нього малопотужний GPS-модуль GT-U7, а також через USB живить мікроконтролер ESP32. Друга шина формується з Li-Po акумулятора номіналом 7,2 В і подається на ESC, який керує безщітковим двигуном і серводвигуном. На вході силової лінії встановлено запобіжник на 10 А і TVS-діод для захисту від імпульсних перенапруг. Усі «земельні» виводи (GND) об’єднані в єдину точку-зірку, що гарантує відсутність плаваючих потенціалів і коректну передачу сигналів між компонентами (див табл 3.4.1).

Таблиця 3.4.1

Узагальнена таблиця всіх основних з’єднань

Компонент	Вивід	Підключення до	Примітка
BME680	SDA	ESP32 GPIO21	Шина I2C
	SCL	ESP32 GPIO22	

	VCC (3,3 В)	ESP32 3V3	
	GND	GND (спільна)	
ESP32	USB micro-USB	Raspberry Pi USB-A	Живлення + UART-канал
	GPIO2 (LED)	—	Індикатор стану
GPS GT-U7	VCC (3,3 В)	Raspberry Pi 3V3	UART-канал
	GND	GND (спільна)	
	TXD	Raspberry Pi GPIO15 (RXD)	
	RXD	Raspberry Pi GPIO14 (TXD)	
Raspberry Pi 4B	USB-C (5 В)	Power Bank	ttyUSB0
	GPIO14/15	GPS TXD/RXD	
	USB-A	ESP32 USB	
	GPIO18 (PWM1)	Servo-двигун PWM	ШІМ-управління
	GPIO19 (PWM2)	ESC PWM	ШІМ-керування швидкістю
ESC	VCC (7,2 В)	Li-Po 7,2 В	Запобіжник 10 А, TVS-діод
	GND	GND (спільна)	
	PWM	Raspberry Pi GPIO19	ШІМ-управління двигуном

Датчик BME680 підключено до мікроконтролера ESP32 по шині I2C: контакт SDA сенсора спаяний із GPIO21, контакт SCL — із GPIO22, живлення 3,3 В узято зі стабілізатора ESP32, а земля сенсора підключена до головної шини GND. Через USB-кабель ESP32 передає сформовані JSON-пакети віртуальному послідовному порту Raspberry Pi (/dev/ttyUSB0), одночасно отримуючи живлення.

Навігаційний модуль GT-U7 отримує живлення від виводу 3,3 В Raspberry Pi, а передача координат здійснюється по апаратному UART: лінія TXD модуля під’єднана до GPIO15 (RXD Pi), а RXD — до GPIO14 (TXD Pi). Роз’єми та кабелі для GPS винесено як окремі конектори з маркуванням у схемі для зручності монтажу антени.

У проєкті керування рухом реалізовано за допомогою одного серводвигуна, який відповідає за зміну кута повороту коліс, і одного безщіткового двигуна, який забезпечує тягу платформи через ESC. Серводвигун під’єднаний до GPIO 18 Raspberry Pi з функцією ШІМ, завдяки чому змінюється його кут повороту відповідно до команд навігаційного алгоритму. Безщітковий двигун живиться від

лінії 7,2 В і підключений до ESC, керування яким здійснюється з GPIO 19 Raspberry Pi також через ШІМ-сигнал. Така конфігурація дозволяє розділити функції напрямку й руху: перший канал відповідає за керування «кермом», другий – за регулювання швидкості рушія.

У результаті отримано комплексну електричну схему без апаратних конфліктів між живильними контурами та інтерфейсами, чітко промарковану шинами VCC_5V, VCC_7.2V і GND, готову до монтажу з використанням уже придбаних модулів і компонентів. Принципова електрична схема представлена в Додатку Г.

3.5. Розробка програмного забезпечення

Було розроблено програмне забезпечення системи з трикомпонентною архітектурою, де кожен елемент виконує чітко визначену роль: прошивка для мікроконтролера ESP32 відповідає за збір сирих даних, основний скрипт на Raspberry Pi керує автономною логікою та навігацією, а серверне програмне забезпечення забезпечує віддалене управління та централізоване зберігання даних.

3.5.1. Прошивка для мікроконтролера ESP32

Прошивка для ESP32, написана на C++ у середовищі Arduino IDE, є першою ланкою в ланцюгу збору даних. Її головна мета — періодично опитувати датчик BME680 та передавати результати вимірювань на Raspberry Pi у простому та надійному форматі. Для цього використовується апаратний інтерфейс Serial2 на виводах GPIO 16 і GPIO 17, що дозволяє ізолювати потік даних від основного порту Serial і уникнути захаращення інформацією з налагодження. Отримані показники не пакуються в JSON, а формуються у вигляді рядка, розділеного комами, що значно спрощує як формування повідомлення на ESP32, так і його розбір на Raspberry Pi. Для підвищення надійності при запуску в функції setup() реалізовано механізм повторних спроб ініціалізації сенсора BME680, що

забезпечує стійку роботу системи навіть при тимчасових неполадках. Приклад фрагмента коду, що демонструє формування та відправку даних наведено у Додатку А.

3.5.2. Програмне забезпечення платформи

Скрипт наведений у Додатку Б є «мозком» платформи та реалізований як об’єктно-орієнтований додаток із багатопотоковим виконанням завдань навігації, зв’язку і збору даних. У окремому потоці відбувається управління та навігація: скрипт зчитує координати з GPS-модуля, обчислює азимут і відстань до поточних цілей, керує сервоприводами для руху й повороту, реалізує послідовне виконання місій із декількома точками маршруту (MISSION_PATH) та автоматичне повернення на базу (return_home). Паралельно в іншому потоці підтримується TCP-з’єднання з лабораторним сервером: по цьому сокету платформа отримує команди типу “goto”, “mission”, “stop”, “status” тощо й надсилає у відповідь актуальний стан та зібрані дані, що забезпечує інтерактивне керування в реальному часі. Третій потік відповідає за обробку даних із ESP32: рядки з показниками сенсора надходять через /dev/ttyUSB0, об’єднуються з GPS-координатами та часовими мітками і записуються у локальний CSV-файл на USB-накопичувачі, що гарантує збереження інформації навіть за відсутності зв’язку з лабораторією.

3.5.3. Сервер лабораторії

Скрипт серверу реалізує багатопотоковий TCP-сервер, що виступає командним центром для однієї або кількох автономних платформ і надає оператору зручний інтерфейс командного рядка для управління місіями. Після запуску сервер відкриває прослуховування на заданому порту і, у разі отримання вхідного з’єднання від платформи, створює для кожного клієнта окремий потік

обробки. Це забезпечує можливість одночасної роботи з кількома платформами без взаємного блокування.

Через CLI оператор може контролювати стан підключених платформ і відправляти їм команди. Команда list показує перелік активних клієнтів із їхніми ідентифікаторами, goto <id> <lat> <lon> ініціює рух до вказаної точки, mission <id> <lat1:lon1> <lat2:lon2 запускає послідовність Waypoint, status <id> повертає поточний стан платформи, а stop <id> негайно зупиняє виконання місії. Усі повідомлення CLI серіалізуються в JSON і надсилаються через відповідний сокет клієнту.

Водночас сервер централізовано зберігає дані від платформ у CSV-файл (наприклад, lab_data_20250608.csv), що спрощує їхній подальший аналіз. Після завершення місії платформа може також передати свій локальний лог-файл із показниками — для цього передбачено окремий механізм прийому файлів з розпізнаванням кінця передачі та записом у відведений каталог. Фрагмент методу, який відповідає за відправку команди конкретному клієнту наведено у Додатку В.

Таким чином, розроблене програмне забезпечення функціонує як єдина розподілена система: багатопотоковий сервер обробляє підключення платформ, CLI-інтерфейс надає гнучкий контроль за їхніми діями, а централізоване логування і можливість прийому файлів гарантують надійний збір та управління екологічними даними.

3.6. Аналіз потенційних помилок та заходи безпеки

У ході реалізації системи автономного моніторингу та управління рухомою платформою були виявлені низка потенційних помилилок та збоїв, пов’язаних з апаратною конфігурацією. По-перше, з’єднання між модулем ESP32 та одноплатним комп’ютером Raspberry Pi 4B організовано через USB-інтерфейс. Незважаючи на надійність USB, можливі сценарії некоректного розпізнавання пристрою (відсутність прав доступу в ОС Raspbian, невірно змонтований серійний порт) або фізичного обриву кабеля, що призводить до втрати даних. Щоб

мінімізувати ці ризики, програмна частина реалізує періодичну перевірку доступності /dev/ttyUSB0 , а в разі відсутності з'єднання здійснює автоматичні спроби повторного підключення з експоненційним інтервалом затримки. До того ж для запобігання електричним перешкодам рекомендується використовувати екрановані USB-кабелі та USB-фільтри, а також стежити за якістю контактів роз'ємів під час монтажу.

По-друге, датчик BME680, який застосовується для вимірювання температури, вологості, атмосферного тиску та якості повітря, підключається до ESP32 по інтерфейсу I2C. Типовими помилками на цьому етапі є протікання шини (bus contention), некоректна адресація та збої через перешкоди живлення. Щоб уникнути складнощів, було обрано рівень живлення 3,3 В із стабілізованого виходу ESP32, рекомендується додада розподільні конденсатори ближче до контактів датчикаю, також таймаути на читання даних із сенсора з обробкою виключень у коді C++. У разі перевищення лімітів пауз між опитуваннями або отримання несподіваних значень (наприклад, різкий стрибок вологості) система фіксує подію в локальний журнал і переходить у безпечний режим очікування повторної ініціалізації шини.

По-третє, збереження інформації відбувається на SD-карті, вставленій в Raspberry Pi через USB-адаптер. Основними ризиками є некоректне монтування файлової системи, пошкодження картки або несподіване відключення адаптера під час запису. Для зниження ймовірності втрачених даних програмно рекомендовано впроватити перевірку цілісності файлової системи перед початком сеансу запису, а також ведення двохетапного протоколу: спочатку дані зберігаються у тимчасовому буферному файлі, а тільки після успішного запису та закриття дескрипторів перенаправляються в основний лог. У випадку помилки монтування SD-картки система автоматично переключається на буферизацію даних у оперативній пам'яті з обмеженим часовим лімітом, доки носій не стане доступним.

Живлення платформи розділено на дві незалежні лінії: логічна частина (Raspberry Pi та периферія GPIO) отримує 5 В від окремого літій-полімерного

акумулятора через стабілізований модуль, тоді як виконавчі елементи руху (ESC, двигуни, серводвигуни) живляться від батареї 7,2 В. Це рішення знижує взаємний вплив стрибків струму високонавантажених приводів на живлення обчислювальної частини, але водночас вимагає обов’язкового об’єднання «земель» (GND) обох джерел живлення для коректного рівня опорної напруги. Для безпеки до ліній живлення можна додати захист у вигляді запобіжників та TVS-діодів від перенапруги, а у програмному забезпеченні впровадити контроль напруги вхідних джерел із виводом попередження та переводом системи в захищений стан при виході за межі допустимого діапазону.

Нарешті, загальні заходи безпеки включають використання Watchdog-таймера на рівні Raspberry Pi (апаратного або програмного), що перезавантажує систему у разі зависання, та захист паролем SSH-доступу для віддаленого обслуговування. Ретельне тестування всіх функцій у лабораторних умовах із емуляцією втрати інтернет-з’єднання, відключень USB-кабеля та перебоїв живлення дозволяє підтвердити надійність алгоритмів відновлення.

У результаті проведеного аналізу потенційних помилок було виявлено ключові ризики: від нестабільного живлення та шумів у лініях даних до втрати сигналу GPS і некоректної обробки телеметрії. Запропоновані заходи безпеки — встановлення запобіжників і фільтрів на живильних шинах, реалізація періодичних самодіагностичних перевірок, обробка винятків у прошивці ESP32 та резервне збереження даних на SD-карті — мінімізують ці ризики. Завдяки цьому забезпечується надійність роботи мобільної платформи в польових умовах і запобігання втраті критично важливої інформації. Це дозволяє лабораторії оперативно відстежувати стан системи, швидко реагувати на неполадки та гарантувати безпеку проведення випробувань.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було створено інтегровану систему управління мобільною платформою з використанням хмарних технологій, яка забезпечує автономний збір, локальне збереження та захищену передачу даних у середовищах із нестабільним або відсутнім інтернет-доступом. Розроблена архітектура спирається на поєднання мікроконтролера ESP32 (для зчитування сенсорних даних і буферизації на SD-карті) та одноплатного комп'ютера Raspberry Pi 4B (для агрегації даних, контролю зв'язку та передачі в хмару), що дозволяє досягти високої гнучкості, модульності та енергоефективності.

У першому розділі проведено аналітичний огляд сучасного стану робототехніки та автоматизованих систем управління рухомими платформами. Було досліджено історію розвитку галузі, класифікацію колісних платформ та світові тенденції впровадження локальних і хмарних рішень. На основі порівняльного аналізу існуючих підходів обґрунтовано необхідність створення системи з локальною буферизацією й адаптивним механізмом передачі даних.

Другий розділ присвячено методам і засобам комп'ютерного моделювання та огляду апаратно-програмних платформ. Розглянуто особливості мікроконтролерів ESP32, Arduino, плат Raspberry Pi і STM32, а також засоби передачі даних. Виконано порівняльний аналіз, за результатами якого обрано оптимальні компоненти проєкту. Окрім того, обґрунтовано вибір інструментів для моделювання.

Третій розділ містив опис структури та функціональних можливостей розробленої системи, обґрунтування вибору апаратного забезпечення і створення електричної схеми підключення сенсорів, мікроконтролера, одноплатного комп'ютера та джерел живлення. Було реалізовано прошивку ESP32 на C++, що відповідає за зчитування даних з сенсора BME680, запис на SD-карту, програму на Python для Raspberry Pi з підтримкою перевірки доступності мережі й

надсилання даних до сервера, а також і сам локальний сервер, який відправля команди та отримував дані.

Наукова новизна роботи полягає в поєднанні автономної навігації за GPS-координатами з інтелектуальним керуванням місіями, що реалізовано в системі «Autonomous Environmental Monitoring Platform». Запропоновано механізм «розумного» прийому команд лише в безпечній зоні базової станції та автоматичного повернення додому за втратою GPS-сигналу або в разі помилок виконання завдання. Розроблено адаптивний протокол зв’язку з лабораторією, який здійснює спроби повторного підключення кожні 10 с, і гібридний алгоритм буферизації та пакетної передачі даних із просторово-часовою прив’язкою показників (температура, вологість, тиск, якість повітря) та GPS-координатою кожного зчитування. Запропоновано інтелектуальну логіку управління енергоспоживанням: платформа вмикає режими глибокого сну під час простоїв, а база забезпечує сонячне підживлення акумулятора й автоматичний обмін даними при поверненні в зону дії супутникового каналу Starlink.

Загальний висновок полягає в тому, що створена автономна роботизована платформа ефективно поєднує навігацію, безперервний моніторинг довкілля та інтеграцію з віддаленою лабораторією. Вона здатна самостійно виконувати набір запланованих точок маршруту, збирати та локально зберігати якісні просторово-часові дані, а за наявності зв’язку — автоматично передавати їх на сервер дослідницької лабораторії. Використання «розумного» керування місіями й адаптивного зв’язку підвищує надійність роботи в складних та віддалених умовах, а інтелектуальні енергозберігаючі режими й сонячна підзарядка забезпечують тривалу автономність. Запропонована система є перспективним інструментом для наукових досліджень у галузі клімату, екології та надзвичайних ситуацій, оскільки дозволяє збирати високоточні дані без прямої участі людини та адаптуватися до різних сценаріїв застосування.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ESP32-WROOM-32 Datasheet. Version 3.3 [Електронний ресурс] / Espressif Systems. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf (дата звернення: 07.06.2025).
2. Raspberry Pi 4 Model B Datasheet [Електронний ресурс] / Raspberry Pi Foundation. 2024. URL: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf (дата звернення: 07.06.2025).
3. BME680 Datasheet [Електронний ресурс] / Bosch Sensortec. 2020. URL: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme680-ds000.pdf (дата звернення: 07.06.2025).
4. GT-U7 GPS Module User Manual [Електронний ресурс] / U-blox. 2019. URL: https://www.u-blox.com/sites/default/files/products/documents/GPS-Modules-FGP-Hardware-User-Manual-V1.0.pdf (дата звернення: 07.06.2025).
5. Python 3.12.4 documentation [Електронний ресурс] / Python Software Foundation. 2024. URL: https://docs.python.org/3/ (дата звернення: 07.06.2025).
6. Arduino IDE [Електронний ресурс] / Arduino. 2024. URL: https://www.arduino.cc/en/software (дата звернення: 07.06.2025).
7. KiCad EDA [Електронний ресурс] / KiCad. 2024. URL: https://www.kicad.org/ (дата звернення: 07.06.2025).
8. NMEA 0183 standard [Електронний ресурс] / National Marine Electronics Association. 2023. URL: https://www.nmea.org/nmea-0183.html (дата звернення: 07.06.2025).
9. RFC 2822: Internet Message Format [Електронний ресурс] / Internet Engineering Task Force (IETF). 2001. URL: https://www.ietf.org/rfc/rfc2822.txt (дата звернення: 07.06.2025).
10. IEEE 802.11-2020 - IEEE Standard for Information Technology [Електронний ресурс] / Institute of Electrical and Electronics Engineers. 2020. URL: https://standards.ieee.org/standard/802_11-2020.html (дата звернення: 07.06.2025).
11. LoRaWAN Specification [Електронний ресурс] / LoRa Alliance. 2023. URL: https://lora-alliance.org/resource-hub/lorawan-specification/ (дата звернення: 07.06.2025).
12. Serial Peripheral Interface (SPI) [Електронний ресурс] / SparkFun Electronics. 2023. URL: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi (дата звернення: 07.06.2025).
13. I2C-Bus Specification and User Manual. Rev. 6 [Електронний ресурс] / NXP Semiconductors. 2014. URL: https://www.nxp.com/docs/en/user-guide/UM10204.pdf (дата звернення: 07.06.2025).

14. Universal Asynchronous Receiver-Transmitter (UART) [Електронний ресурс] / SparkFun Electronics. 2023. URL: https://learn.sparkfun.com/tutorials/serial-communication (дата звернення: 07.06.2025).
15. What is Cloud Computing? [Електронний ресурс] / Amazon Web Services. 2024. URL: https://aws.amazon.com/what-is-cloud-computing/ (дата звернення: 07.06.2025).
16. What is Firebase? [Електронний ресурс] / Google Cloud. 2024. URL: https://firebase.google.com/docs/guides (дата звернення: 07.06.2025).
17. JSON Web Token (JWT) [Електронний ресурс] / jwt.io. 2024. URL: https://jwt.io/introduction/ (дата звернення: 07.06.2025).
18. STM32F103C8T6 Datasheet [Електронний ресурс] / STMicroelectronics. 2017. URL: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf (дата звернення: 07.06.2025).
19. Sophia the Robot [Електронний ресурс] / Hanson Robotics. 2024. URL: https://www.hansonrobotics.com/sophia/ (дата звернення: 05.06.2025).
20. Pepper and NAO Robots [Електронний ресурс] / SoftBank Robotics. 2024. URL: https://www.softbankrobotics.com/emea/en/pepper (дата звернення: 05.06.2025).
21. Roomba Series [Електронний ресурс] / iRobot Corporation. 2024. URL: https://www.irobot.com/roomba (дата звернення: 05.06.2025).
22. Robotics in Manufacturing [Електронний ресурс] / General Motors. 2023. URL: https://www.gm.com/robots (дата звернення: 05.06.2025).
23. Industrial Robots [Електронний ресурс] / KUKA AG. 2024. URL: https://www.kuka.com/en-de/products/robot-systems/industrial-robots (дата звернення: 05.06.2025).

Додаток А

Лістинг функцій ініціалізації ESP32 та потоку збору даних із BME680

```
/ Підключення бібліотек та глобальні оголошення
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME680.h>
// Об'єкт датчика BME680
Adafruit_BME680 bme;
// Налаштування апаратного Serial2 (чистий, без повідомлень завантаження)
#define SERIAL2_BAUD_RATE 9600
#define RXD2 16  // GPIO16 - підключити до TX Raspberry Pi
#define TXD2 17  // GPIO17 - підключити до RX Raspberry Pi
// Змінні стану
bool sensorInitialized = false;
unsigned long lastReading = 0;
const unsigned long READING_INTERVAL = 10000; // 10 секунд
// Функція ініціалізації BME680
bool initializeBME680() {
  // Ініціалізація I2C для BME680
  Wire.begin();
  // Ініціалізація датчика BME680 з логікою повторних спроб
  int attempts = 0;
  while (!bme.begin() && attempts < 5) {
    Serial2.println("BME680 sensor not found, retrying...");
    Serial.println("BME680 sensor not found, retrying...");
    delay(1000);
    attempts++;
  }
  if (attempts >= 5) {
    Serial2.println("ERROR: Could not initialize BME680 sensor after 5 attempts!");
    Serial.println("ERROR: Could not initialize BME680 sensor!");
    sensorInitialized = false;
    return false;
  } else {
    // Налаштування надвибірки та ініціалізація фільтра
    bme.setTemperatureOversampling(BME680_OS_8X);
    bme.setHumidityOversampling(BME680_OS_2X);
    bme.setPressureOversampling(BME680_OS_4X);
    bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
    bme.setGasHeater(320, 150); // 320°C протягом 150 мс
    Serial2.println("BME680 sensor initialized successfully!");
    Serial.println("BME680 sensor initialized successfully!");
    sensorInitialized = true;
    // Увімкнути світлодіод для індикації успішної ініціалізації
    digitalWrite(2, HIGH);
    // Відправити сигнал готовності
    Serial2.println("SYSTEM_READY");
```

```
    return true;
  }
 }
 // Функція setup
 void setup() {
  // Ініціалізація вбудованого світлодіода для індикації стану (GPIO2 на більшості плат ESP32)
  pinMode(2, OUTPUT);
  digitalWrite(2, LOW);
  // Ініціалізація апаратного Serial2 для чистого зв'язку
  Serial2.begin(SERIAL2_BAUD_RATE, SERIAL_8N1, RXD2, TXD2);
  // Ініціалізація звичайного Serial для відладки (додатково)
  Serial.begin(115200);
  // Коротка затримка для ініціалізації serial
  delay(1000);
  Serial2.println("=== ESP32 BME680 STARTING ===");
  Serial.println("ESP32 BME680 Data Transmitter Starting...");
  // Ініціалізація датчика BME680
  initializeBME680();
  delay(1000);
 }
 // Головний цикл опитування
 void loop() {
  unsigned long currentTime = millis();
  // Перевірити, чи настав час нового зчитування
  if (currentTime - lastReading >= READING_INTERVAL) {
   if (sensorInitialized) {
    readAndSendData();
   } else {
    // Спроба повторної ініціалізації датчика
    if (bme.begin()) {
     Serial2.println("BME680 sensor reconnected!");
     Serial.println("BME680 sensor reconnected!");
     sensorInitialized = true;
     digitalWrite(2, HIGH);
    } else {
     Serial2.println("ERROR: BME680 sensor still not available");
    }
   }
   lastReading = currentTime;
  }
  // Моргання світлодіодом для індикації роботи системи
  if (currentTime % 2000 < 100) {
   digitalWrite(2, !digitalRead(2));
  }
 }
 // Функція зчитування даних та передачі
 void readAndSendData() {
  // Зчитати дані з датчика
  if (!bme.performReading()) {
   Serial2.println("ERROR: Failed to perform BME680 reading");
   Serial.println("ERROR: Failed to perform BME680 reading");
```

```
    return;
  }
  // Отримати поточний часовий штамп (мс з моменту запуску)
  unsigned long timestamp = millis();

  // Перевірити коректність показань датчика
  if (isnan(bme.temperature) || isnan(bme.humidity) ||
      isnan(bme.pressure) || isnan(bme.gas_resistance)) {
    Serial2.println("ERROR: Invalid sensor readings (NaN detected)");
    Serial.println("ERROR: Invalid sensor readings");
    return;
  }
  // Підготувати рядок даних у форматі CSV
  String dataString = "DATA," + String(timestamp) + "," +
                      String(bme.temperature, 2) + "," +
                      String(bme.humidity, 2) + "," +
                      String(bme.pressure / 100.0, 2) + "," +
                      String(bme.gas_resistance / 1000.0, 2);
  // Відправити дані через апаратний Serial2 (чистий канал)
  Serial2.println(dataString);
  // Також відправити на звичайний серіал для відладки
  Serial.println(dataString);
  // Відправляти «heartbeat» кожні 5 зчитувань
  static int readingCount = 0;
  readingCount++;
  if (readingCount >= 5) {
    Serial2.println("HEARTBEAT," + String(timestamp));
    readingCount = 0;
  }
}
```

Додаток Б

Лістинг функції скрипта Raspberry Pi

```
#Налаштування та імпорти
import serial, json, socket, threading, time, csv, os, math
from datetime import datetime
import RPi.GPIO as GPIO
from gpiozero import Servo
CONFIG = {
   "bme680_device": "/dev/ttyUSB0", "bme680_baud": 9600,
   "gps_device": "/dev/ttyAMA0", "gps_baud": 9600,
   "lab_server_ip": "192.168.1.100", "lab_server_port": 8080,
   "home_tolerance": 2.0, "max_speed": 0.3}
#Обробка даних UART
def bme680_thread(self):
   """Обробляє дані BME680 від ESP32"""
   while True:
      if self.bme680_serial and self.bme680_serial.in_waiting > 0:
         data_line = self.bme680_serial.readline().decode('utf-8').strip()
         if data_line and ',' in data_line:
            parts = data_line.split(',')
            if len(parts) == 5:  # DATA,мітка_часу,температура,вологість,тиск,газ
               sensor_data = {
                  'timestamp': datetime.now().isoformat(),
                  'gps_lat': self.current_lat, 'gps_lon': self.current_lon,
                  'esp32_timestamp': int(parts[0]),
                  'temperature': float(parts[1]), 'humidity': float(parts[2]),
                  'pressure': float(parts[3]), 'gas_resistance': float(parts[4])    }
               self.process_sensor_data(sensor_data)
      time.sleep(1)
def parse_nmea_gps(self, nmea_sentence):
   """Розбирає NMEA-рядки GPS"""
   if nmea_sentence.startswith('$GPGGA') or nmea_sentence.startswith('$GNGGA'):
      parts = nmea_sentence.split(',')
      if len(parts) >= 15 and parts[6] != '0' and parts[2] and parts[4]:
         # Розбір координат
         lat_raw, lon_raw = float(parts[2]), float(parts[4])
         lat = int(lat_raw/100) + (lat_raw % 100)/60.0
         lon = int(lon_raw/100) + (lon_raw % 100)/60.0
         if parts[3] == 'S': lat = -lat
         if parts[5] == 'W': lon = -lon
         self.current_lat, self.current_lon = lat, lon
         self.gps_fix = True
         return True
   return False
#Буферизація та збереження даних
def process_sensor_data(self, sensor_data):
   """Буферизує та зберігає дані датчиків"""
```

```
        self.data_buffer.append(sensor_data)
        if len(self.data_buffer) > 100:
            self.data_buffer.pop(0)
        self.save_sensor_data(sensor_data)
        self.send_data_to_lab(sensor_data)
    def save_sensor_data(self, data):
        """Зберігає дані у CSV-файл"""
        if not self.log_file_path:
            self.log_file_path = f"BME680_data_{datetime.now().strftime('%Y%m%d')}.csv"
            with open(self.log_file_path, 'w', newline='') as f:
                csv.writer(f).writerow([
                    'timestamp', 'gps_lat', 'gps_lon', 'esp32_timestamp',
                    'temperature', 'humidity', 'pressure', 'gas_resistance'])
        with open(self.log_file_path, 'a', newline='') as f:
            csv.writer(f).writerow([
                data['timestamp'], data['gps_lat'], data['gps_lon'],
                data['esp32_timestamp'], data['temperature'],
                data['humidity'], data['pressure'], data['gas_resistance']  ])
    #Зв’язок з лабораторією
    def setup_lab_communication(self):
        """Налаштувати TCP-з’єднання"""
        try:
            self.lab_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.lab_socket.connect((self.config['lab_server_ip'], self.config['lab_server_port']))
            self.lab_connection_active = True
            self.send_data_to_lab({'type': 'PLATFORM_CONNECTED', 'platform_id': 'env_platform_01'})
            return True
        except:
            return False
    def send_data_to_lab(self, data):
        """Відправляє JSON-повідомлення до лабораторії"""
        try:
            if self.lab_socket and self.lab_connection_active:
                self.lab_socket.send((json.dumps(data) + '\n').encode())
                return True
            elif time.time() - self.last_lab_connection_attempt >= 10:
                self.last_lab_connection_attempt = time.time()
                self.setup_lab_communication()
        except:
            self.lab_connection_active = False
        return False
    #Обробка команд та навігація
    def handle_lab_command(self, command):
        """Обробляє команди з лабораторії"""
        cmd_type = command.get('type')
        if cmd_type == 'GOTO':
            if self.at_home:
                self.target_lat, self.target_lon = command['lat'], command['lon']
                self.mission_active = True
                self.send_data_to_lab({'type': 'MISSION_ACCEPTED'})
            else:
                self.send_data_to_lab({'type': 'MISSION_REJECTED', 'reason': 'Платформа не вдома'})
```

```
        elif cmd_type == 'EMERGENCY_STOP':
            self.stop_motors()
            self.mission_active = False
        elif cmd_type == 'RETURN_HOME':
            self.return_home()
    def navigate_to_target(self):
        """Проста навігація"""
        if not self.gps_fix or not self.mission_active:
            self.stop_motors()
            return
        distance = self.calculate_distance(
            self.current_lat, self.current_lon,
            self.target_lat, self.target_lon)
        if distance < 5.0:  # Досягнуто ціль
            self.stop_motors()
            self.mission_active = False
            self.return_home()
        else:
            throttle = min(0.3, distance / 100.0)
            if self.throttle_servo:
                self.throttle_servo.value = throttle
    def check_at_home(self):
        """Перевіряє, чи на місці «додому»"""
        if not self.gps_fix:
            return False
        home_lat = self.config['home_coordinates']['lat']
        home_lon = self.config['home_coordinates']['lon']
        if home_lat == 0.0:
            return False
        distance = self.calculate_distance(
            self.current_lat, self.current_lon, home_lat, home_lon
        )
        self.at_home = distance <= self.config['home_tolerance']
        return self.at_home
    #Головне виконання
    def run(self):
        """Головний цикл роботи платформи"""
        self.setup_bme680_serial()
        self.setup_gps_serial()
        self.setup_lab_communication()
        threading.Thread(target=self.bme680_thread, daemon=True).start()
        threading.Thread(target=self.gps_thread, daemon=True).start()
        threading.Thread(target=self.navigation_thread, daemon=True).start()
        try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            self.cleanup()
```

Додаток В

Лістинг функцій TCP-сервера лабораторії

```
# Message Processing
def handle_platform_message(self, address, data, client_socket):
    """Обробка повідомлень від платформи"""
    msg_type = data.get('type', 'SENSOR_DATA')
    client_info = self.clients[address]
    if msg_type == 'PLATFORM_CONNECTED':
        client_info['platform_id'] = data.get('platform_id')
        client_info['status'] = 'online'
    elif msg_type == 'HEARTBEAT':
        client_info['last_heartbeat'] = time.time()
    elif msg_type == 'MISSION_ACCEPTED':
        self.logger.info(f"Місію ПРИЙНЯТО
            платформою {client_info.get('platform_id')}")
    elif msg_type == 'MISSION_REJECTED':
        self.logger.warning(f"✗ Місію ВІДХИЛЕНО: {data.get('reason')}")
    else:
        # Дані з сенсорів — запис у CSV
        self.log_sensor_data(data)
# Command Transmission
def send_command(self, client_socket, command):
    """Надіслати JSON-команду платформі"""
    try:
        message = json.dumps(command) + '\n'
        client_socket.send(message.encode())
        return True
    except:
        return False
def send_command_to_all(self, command):
    """Розіслати команду всім онлайн-платформам"""
    count = 0
    for addr, info in self.clients.items():
        if info.get('status') == 'online':
            if self.send_command(info['socket'], command):
                count += 1
    return count
# CLI Interface
def command_interface(self):
    """Інтерактивний інтерфейс командного рядка"""
    print("LAB COMMAND CENTER")
    print("Commands: goto <lat> <lon>, mission <lat1,lon1> <lat2,lon2>, home, stop, list, quit")
    while self.running:
        try:
            parts = input("\nLab> ").strip().split()
            if not parts: continue
            cmd = parts[0].lower()
            if cmd == 'quit':
                self.running = False
```

```
            elif cmd == 'list':
                self.list_platforms()
            elif cmd == 'stop':
                if input("Аварійна зупинка? (yes/no): ") == 'yes':
                    self.send_command_to_all({'type': 'EMERGENCY_STOP'})
            elif cmd == 'home':
                self.send_command_to_all({'type': 'RETURN_HOME'})
            elif cmd == 'goto' and len(parts) >= 3:
                lat, lon = float(parts[1]), float(parts[2])
                self.send_command_to_all({'type': 'GOTO', 'lat': lat, 'lon': lon})
            elif cmd == 'mission':
                waypoints = []
                for part in parts[1:]:
                    if ',' in part:
                        lat, lon = part.split(',')
                        waypoints.append({'lat': float(lat), 'lon': float(lon)})
                if waypoints:
                    self.send_command_to_all({'type':'MISSION_PATH','waypoints': waypoints})
        except KeyboardInterrupt:
            self.running = False
# Platform Status
def list_platforms(self):
    """Показати підключені платформи"""
    print(f"\nПлатформи ({len(self.clients)}):")
    for i, (addr, info) in enumerate(self.clients.items()):
        status = info.get('status', 'unknown')
        platform_id = info.get('platform_id', 'Unknown')
        icon = "✓" if status == 'online' else "✗"
        print(f"[{i}] {icon} {platform_id} ({addr}) - {status}")
# Main Server Loop
def start_server(self):
    """Запустити TCP-сервер"""
    self.server_socket.bind((self.host, self.port))
    self.server_socket.listen(10)
    print(f"Сервер слухає на {self.host}:{self.port}")
    # Запустити потік інтерфейсу CLI
    threading.Thread(target=self.command_interface, daemon=True).start()
    while self.running:
        try:
            self.server_socket.settimeout(1.0)
            client_socket, address = self.server_socket.accept()
            threading.Thread(
                target=self.handle_client,
                args=(client_socket, address),
                daemon=True
            ).start()
        except socket.timeout:
            continue
// Key Commands
// Server → Platform
{"type": "GOTO", "lat": 45.123, "lon": -63.456}
{"type": "MISSION_PATH", "waypoints": [{"lat": 45.123, "lon": -63.456}]}
```

```
{"type": "RETURN_HOME"}
{"type": "EMERGENCY_STOP"}

// Platform → Server
{"type": "PLATFORM_CONNECTED", "platform_id": "env_01"}
{"type": "MISSION_ACCEPTED", "mission_type": "GOTO"}
{"type": "MISSION_REJECTED", "reason": "Not at home"}
{"type": "HEARTBEAT"}
```

Додаток Г

Принципова електрична схема