МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

В. о. завідувача кафедри,
д-р техн. наук, професор

_____ І. М. Журавська

« __ » _____ 2023р.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Апаратно-програмний модуль відстеження швидкості велосипеду на базі Raspberry Pi Zero

Спеціальність 123 «Комп'ютерна інженерія»
123 – КМР.1 – 605.21710522

Студент _____С. С. Рибченко
підпис

«__» _____ 2023 р.

Керівник канд. тех. наук, доцент _____ Я. М. Крайник
підпис

«__» _____ 2023р.

Миколаїв – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС	Операційна система
ШІ	Широтно-імпульсна модуляція
M	
DD	Double Data Rate
R	
FPS	Frames per second
GN	Global Navigation Satellite System
SS	
GPI	General Purpose Input Output
O	
HA	Hardware attached on top
T	
IoT	Internet of Things
I2C	Inter-Integrated Circuit
GPS	Global Positioning System
MQ	Message queuing telemetry transport
TT	
NM	National Marine Electronics Association
EA	
QoS	Quality of Service
SoC	System-on-a-Chip
SPI	Serial Peripheral Interface
UA	Universal asynchronous receiver/transmitter
RT	

ВСТУП

В житті сучасного суспільства роль велосипеду значно зросла завдяки тенденції на екотранспорт. Цей вид транспорту стає все більш популярним особливо у великих містах для швидкого переміщення, оминаючи затори ще й зберігаючи як довкілля так і власне здоров’я. Для більш комфортного та безпечного користування велосипедом необхідно контролювати швидкість. Особливо такий контроль необхідний комерційним велосипедам для, наприклад, оренди, що оснащені допоміжним електродвигуном, адже в такому випадку швидкість треба ще й обмежувати задля безпеки користувачів. Окремо виникає питання контролю та обмеження місцевості, де такі орендовані велосипеди можуть застосовуватись. Щоб забезпечити такий контроль необхідно знати не лише швидкість транспорту а і його місцезнаходження, а також мати інструменти для відслідковування, відображення поточної його локації та можливості обмеження пересування. Тож ця робота присвячення створенню апаратно-програмного комплексу, що міг би задовольнити потреби як приватних користувачів велотранспортом так і вирішити проблему комерційного застосування.

Мета: покращення віддаленого контролю за транспортом шляхом розробки апаратно-програмного модуля відстеження швидкості і географічного положення велосипеду на базі Raspberry Pi Zero.

Об’єкт: технології відстеження швидкості та географічного положення.

Предмет: апаратно-програмний комплекс відстеження та контролю швидкості та місцеположення транспорту на базі Raspberry Pi Zero.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

- розробити апаратно-програмний комплекс;
- реалізувати обробку даних GNSS;
- реалізувати обмін даних апаратного комплексу;
- реалізувати інтерфейс апаратного комплексу;

- реалізувати відображення даних в реальному часі на вебсайті;
- реалізувати зони обмеження в комплексі.

Практичне значення отриманих результатів: розроблений програмно-апаратний комплекс може ефективно вимірювати та надавати дані про швидкість і місцезнаходження механічного транспортного засобу а також виконувати диспетчеризацію абонентів. Використовуючи цей комплекс стає можливим контроль і обмеження транспортного засобу. Завдяки інформації про геолокацію та швидкість можуть бути створенні звіти для подальшого вивчення та покращення як технічних властивостей транспорту так і фізичних показників користувача. Комерційний користувач матиме змогу обмежувати використання транспорту у певних географічних межах та мати можливість віддалено впливати на обладнання транспорту на який встановлено такий комплекс.

Апробація результатів магістерської роботи відбулася в рамках Всеукраїнської науково-практичної конференції молодих вчених, аспірантів і студентів «Інформаційні технології та інженерія» (Миколаїв, ЧНУ ім. Петра Могили).

Публікації. За результатами кваліфікаційної роботи створено публікацію у збірнику матеріалів Всеукраїнської конференції [1].

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ КОМПЛЕКСУ ВІДСТЕЖЕННЯ ШВИДКОСТІ ВЕЛОСИПЕДУ

Мета роботи – розробити апаратно-програмний комплекс для відстеження швидкості велосипеду. Для досягнення поставленої мети слід розглянути технології визначення швидкості, знаходження геопозиції та передачі даних в реальному часі в умах нестабільного зв’язку. Усі описані технології необхідно оформити в автономний апаратний комплекс, що мав би відповідне програмне забезпечення для відображення інтерфейсу користувачу та окремий вебінтерфейс для відображення переданої інформації віддалено.

Для створення апаратного модуля необхідно обрати:

- модуль обчислення;
- модуль для визначення місцеположення;
- модуль GSM зв’язку;
- дисплей.
- Програмна частина модулю має складатися з таких складових:
- модуль обчислення, відображення та передачі даних на апаратній платформі;
- вебсайт, що слугуватиме панеллю відстеження та керування;
- сервер, що буде центром комунікації між панеллю адміністратора та окремими апаратних платформ.

Перед створенням власного комплексу варто дослідити існуючі рішення для поставленої мети.

1.1 Огляд сучасних рішень для відстеження та відображення швидкості велосипеду

Після пошуку в мережі, було знайдено низку готових рішень, що відповідають до поставленої мети. Найбільш поширеними є комплекси, що

виконують роль виключно автономного велосипедного комп’ютера, вони ж у свою чергу поділяються на простіші – що тільки вимірюють та відображають швидкість транспорту (зазвичай за допомогою датчика ефекту Холла), та складніші – що виконують також функцію навігації, якісніші дисплеї, внутрішню пам’ять, яка може зберігати інформацію про активність разом із даними маршруту.

Базові спідометри для велосипеду (рис. 1.1) достатньо точно вимірюють швидкість за рахунок вимірювань обертів колеса датчиком Холла. Вони відображають дані поточної швидкості на простому монохромному дисплеї, іноді також зберігають загальний пробіг (одометр), але не мають ніякої іншої функціональності і серед переваг мають тільки ціну і простоту використання.

Рисунок 1.1 – Найпростіший цифровий спідометр.

Також існують більш просунуті комплекси, особливості яких можна розглянути на прикладі одного з найпопулярніших – Elemnt Roam (рис. 2.1) від виробника Wahoo.

Wahoo Roam надає велосипедистам повний інструментарій для навігації, відстеження фізичної форми та аналізу даних, також спідометр і одометр, розглянуті в попередньому типі комплексів. Цей пристрій наповнений функціями та можливостями, які допоможуть водіям

оптимізувати свою продуктивність і покращити досвід їзди на велосипеді. Комплекс має наступні специфікації:

- 2,7 дюйма кольоровий дисплей з роздільною здатністю 240x400 пікселів і 64-ма кольорами;
- 32 ГБ внутрішньої пам'яті;
- заявлена автономність сягає 17 годин від одного заряду;
- підтримка супутників GPS, ГЛОНАСС, BEIDOU, Galileo та QZSS.

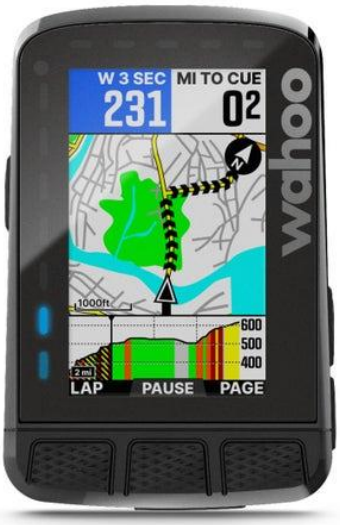


Рисунок 1.2 – Велокомп’ютер Wahoo Elemnt Roam.

Загалом Roam має такі переваги: міцний корпус захищений від води та вологи, гарну точність місцеположення за рахунок підтримки супутників різного типу. Але також має недоліки у вигляді невідповідної до заявленої автономності, відповідно до відгуків користувачів, комплекс має високу ціну (13-14 тис. грн), відносно своїх конкурентів. Також є великий недолік для поставленої мети, адже дані зберігаються виключно на самому комплексі, тож швидкість і пересування не можливо спостерігати віддалено, як зазначалось у вимогах до кінцевого пристрою.

Під час дослідження також було знайдено пристрої орієнтовані більше на спостереження за транспортом ззовні, ніж на контроль параметрів

користувачем транспорту. Серед таких слід відокремити трекер ST-901M від компанії SinoTrack (рис. 1.3).

Рисунок 1.3 – GPS трекер SinoTrack ST-901M.

Цей пристрій вже не має вбудованого дисплею, але він оснащений набором датчиків, включаючи G-сенсор для вимірювання прискорення та удару, а також електронний компас для визначення напрямку руху. Ці дані можна використовувати для різних цілей, наприклад для моніторингу поведінки водія, оптимізації маршрутів. ST-901M можна керувати та контролювати за допомогою веб-платформи або мобільного застосунку, який надає оновлення в режимі реального часу щодо розташування та стану відстежуваних транспортних засобів. Користувачі також можуть налаштувати спеціальні сповіщення та сповіщення, наприклад сповіщення про в’їзд або виїзд із визначеної зони або сповіщення про перевищення швидкості.

ST-901M має такі особливі характеристики:

- підтримка 4-ох частот GSM 850/900/1800/1900 МГц;
- підтримка супутників GPS, ГЛОНАСС, Galileo
- акумулятор 150 мА·год.
- Тож, загалом даний комплекс відповідає частині поставлених вимог, такі як передача даних географічного положення в реальному часі, а

також має інтерфейси віддаленого перегляду цих даних. Проте даний комплекс виконує роль лише трекеру положення й не відображає інформацію користувачу.

– Отже, провівши аналітичне дослідження існуючих рішень для відстеження швидкості велосипеду, було виявлено ряд комплексів, що виконують тільки частину поставлених задач, але жоден з них не відповідає повністю поставлені меті. Зазвичай, рішення вузько спеціалізовані під вирішення однієї задачі, наприклад, трекінгу положення, навігації, але жоден з них не поєднує ці можливості для отримання комплексного модулю.

1.2 Вибір апаратних рішень для створення комплексу

1.2.1 Вибір модуля обчислення

При виборі одноплатного комп’ютера для виконання усіх обчислень було відокремлено три представники: Raspberry Pi Zero, Banana Pi M2 Zero та Onion Omega2 Plus. Основним критерієм вибору була можливість повноцінного запуску операційної системи сімейства UNIX, але також критичним були розміри самого пристрою.

Найпростішим за можливостями але і найменшим за розміром є одноплатний комп’ютер Onion Omega2 Plus (рис. 1.4). За розміром у 42×25мм він співставний з простішими мікроконтролерами типу ESP32, але при цьому підтримуючи повноцінний запуск ОС Linux.

Рисунок 1.4 – Одноплатний комп'ютер Onion Omega2 Plus.

Omega2 Plus створено для взаємодії з іншим апаратним забезпеченням, тому він має широкий вибір апаратних інтерфейсів. Він має спеціальні контакти для USB2.0 і Ethernet, а також 12 контактів GPIO, якими може керувати користувач. Ці GPIO можуть підтримувати протоколи послідовного зв’язку I2C, UART і SPI, а також два ШІМ контролери. Але основними недоліками цієї плати є те що вона розроблена для малих IoT проєктів з малою необхідністю обчислення, а отже має малопотужний SoC MT7688 для виконання найпростіших задач, а також малу оперативну пам’ять розміром усього 128МБ стандарту DDR2.

Наступним до розгляду був одноплатний комп’ютер Raspberry Pi Zero (рис. 1.5). Це найбільш популярна платформа для розробки IoT систем серед тих що розглядаються, завдяки чому, саме цей комп’ютер має надзвичайно велику підтримку суспільства і багату документацію.

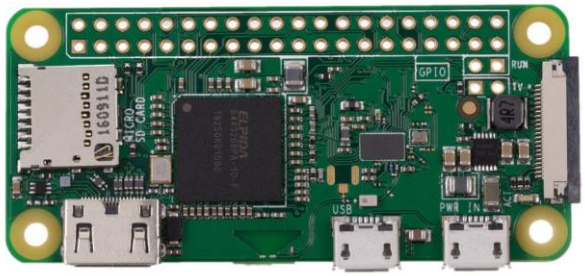


Рисунок 1.5 – Одноплатний комп’ютер Raspberry Pi Zero.

Raspberry Pi Zero працює на різних операційних системах, включаючи Raspbian(офіційна ОС Raspberry Pi), а також інші дистрибутиви Linux і навіть Windows 10 IoT Core. Ця універсальність означає, що Raspberry Pi Zero можна використовувати для широкого спектру додатків, від основних обчислювальних завдань, таких як перегляд веб-сторінок і обробка текстів, до більш складних проектів, таких як медіа-центри, системи домашньої автоматизації та навіть роботи.

Що стосується підключення, Raspberry Pi Zero має 2 порти micro-USB для живлення та даних, а також порт mini-HDMI для виведення відео. Він також має 40-контактний роз’єм GPIO, який дозволяє користувачам підключати різні датчики, модулі та велику кількість розширень із підключенням, стандартним для одноплатних комп’ютерів цього виробника.

Останнім розглядався Banana Pi M2 Zero, що являється значно менш популярним конкурентом Raspberry Pi. Основною його перевагою є те що він повністю повторює розташування елементів та інтерфейсів на ідентичній за розмірами платі, за рахунок чого, автоматично підтримує чисельні модулі розроблені для Raspberry. Але виробник не скопіював комп’ютер сліпо, але й додав власні особливості. Зокрема було додано дві фізичні кнопки: живлення та скидання, що відсутні у всіх конкурентів, але можуть стати в нагоді при

налагодженні. Ще одним поширеним доповненням до Banana Pi є окремий роз’єм налагоджувального UART для послідовного зв’язку, не займаючи місця GPIO. Ця плата також має антенний роз’єм U.FL, який може збільшити діапазон бездротового зв’язку цієї плати.

Рисунок 1.6 – Одноплатний комп’ютер Banana Pi M2 Zero.

Щодо характеристик, то дана плата має трохи потужніший за попередні чотирьох ядерний процесор Cortex-A7 з тактовою частотою 1,2 ГГц та ті ж самі 512 МБ оперативної пам’яті як і у Raspberry Pi. Завдяки Allwinner H2 SoC на цій платі, вона має можливість підключення Ethernet 100 МБ, хоча й самого роз’єму нема, проте є контактні площадки для розпаювання власноруч.

Отже розглянувши особливості кожного представника, необхідно порівняти технічні характеристики, особливості кожної платформи та ціну.

Таблиця 1.1 – Порівняння характеристик одноплатних комп’ютерів Raspberry Pi Zero, Banana Pi M2 Zero та Onion Omega2 Plus.

Характеристика	**Raspberry Pi Zero**	**Banana Pi M2 Zero**	**Onion Omega2 Plus**

Центральний процесор	Broadcom BCM2835	Allwinner Cortex-A7	MediaTek MT7688AN
Кількість ядер процесору	1 ядро	4 ядра	1 ядро
Тактова частота процесору	1 ГГц	1.2 ГГц	580 МГц
Оперативна пам’ять	512 МБ DDR2	512 МБ DDR3	128 МБ DDR2
Живлення	5В до 2А	5В до 2А	3.3 В до 800 мА
Ціна (заявлена виробником)	15$	20$	10$

Onion Omega2 Plus була одразу відхилена після першого погляду на порівняльну таблицю, адже має найменше оперативної пам’яті, не зручну напругу живлення, малопотужний процесор, та дуже не зручний метод під’єднання до інтерфейсів. Для поставлених задач дана плата має недостатню потужність обчислення, зважаючи на те, що більшість з заявлених виробником будуть вже зайняті ОС.

Плати що залишились, а саме Banana Pi M2 Zero так Raspberry pi Zero мають паритет по зручності монтування, завдяки популярності 40-ка контактному роз’єму Raspberry. Також вони не сильно відрізняються за потужністю обчислення, хоча перевага і за більш потужним Banana Pi. Оперативна пам’ять також має однаковий об’єм, хоча знову ж з перевагою на стороні Banana Pi, завдяки старшому поколінню і відповідно вищій швидкості. Перевагою ж Raspberry Pi стала значно більша підтримка заліза від спільноти, завдяки чому для неї існує більша кількість операційних систем, та велика кількість детальної документації, що значно спростить процес розробки.

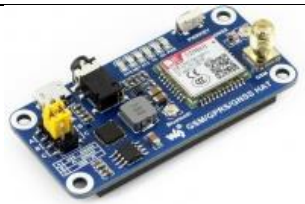
Отже, за результатом порівнянь було обрано одноплатний комп’ютер Raspberry Pi Zero. Вирішальним для вибору став фактор наявності, адже з причини дефіциту мікроелектроніки, викликану пандемією, кількість наявних виробів близиться до нулю. Отже обирати прийшлося з того що було

в наявності, де знову в нагоді стала більша популярність саме платформи Raspberry Pi.

1.2.2 Вибір модуля позиціювання та сигналу

Обравши модуль обчислення, було одночасно обрану і платформу, адже для цих пристроїв існує безліч готових рішень які вже адаптовані за кріпленням і з’єднанням до Raspberry Pi Zero. Тож усі подальші модулі будуть розглядатись лише за сумісності з обраним одноплатним комп’ютером.

Таблиця 1.2 – Порівняння характеристик модулів GSM та GNSS зв’язку.

Характеристика	**OzzMaker BerryGPS-GSM for Raspberry Pi**	**Waveshare SIM868 HAT**	**Waveshare SIM7600G-H HAT**
Зображення			
Модуль GSM	SARA-U201	SIM868	SIM7600G-H
Модуль GNSS	CAM-M8-FW3	SIM868	SIM7600G-H
Підключення	Micro USB	40pin GPIO	Pogo pin USB
Ціна (заявлена виробником)	100$	32$	75$

Усі три модулі мають різні методи підключення до плати комп’ютера, але Waveshare SIM868 HAT займає увесь роз’єм GPIO, що перешкоджає подальше підключення, тож цей модуль не підходить для поставлених задач, адже ще необхідно підключити дисплей. Модуль він OzzMaker має підключення за допомогою роз’єму microUSB, що займає один з роз’ємів на платі. В той же час Waveshare SIM7600G-H має підключення за допомогою pogo pin до USB інтерфейсу при цьому не займаючи сам роз’єм, адже на платі присутні спеціальні місця зовнішнього підключення до логічних контактів USB. Також цей модуль виконує роль USB хабу, а отже може

розширити кількість можливих підключень для подальшого розвитку проєкту.

Отже для даного проєкту найбільше пасує саме модуль Waveshare SIM7600G-H, адже він не обмежує жоден з інтерфейсів одноплатного комп’ютера та навіть розширює кількість інтерфейсів підключення. Також при виборі компонента велику роль знову відіграли фактори наявності та ціни, адже обраний модуль був у наявності та має ціну меншу за модуль від OzzMaker, що також розглядався.

1.2.3 Вибір дисплею

Останнім компонентом готового апаратного комплексу має стати дисплей, як інтерфейс взаємодії з користувачем. Як і у випадку з модулем зв’язку, вибір дисплею буде робитися з оглядом не сумісність із обраним одноплатним комп’ютером Raspberry Pi Zero: Adafruit PiTFT Plus 2.8", HyperPixel 2.1, Waveshare 3.5". Усі дисплеї знаходяться в одній ціновій категорії, отже ціна не буде розглядатись як фактор що впливає на вибір.

Adafruit PiTFT Plus 2.8" (рис. 1.7) має прямокутний дисплей розміром 2.8 дюйма з роздільною здатністю 320×240 пікселів та 16-ти бітним кольорами. Також дисплей має ємкісний сенсор та 4 фізичні кнопки, що дозволяють взаємодіяти з пристроєм. Приєднується цей дисплей до 40-ка контактного стандартного роз’єму Raspberry. Одним з найбільших недоліків є технологія за якою виконано матрицю дисплею – TFT. Дисплеї цієї технології мають спотворення зображення через інверсію сірого під різними кутами, що призводить до нечитаємості під певними боковими кутами огляду. Також у цієї технології є недолік – під прямим сонячним світлом екран погано зчитуваний.

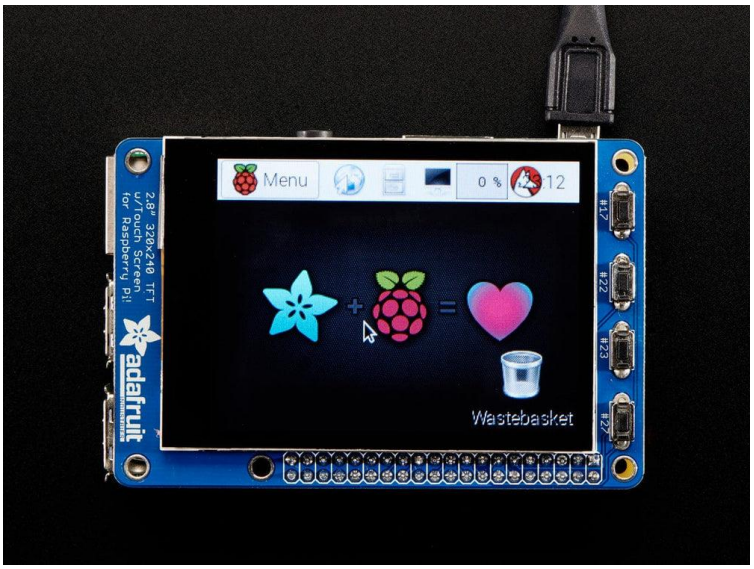


Рисунок 1.7 – Дисплей Adafruit PiTFT Plus 2.8".

HyperPixel 2.1 (рис. 1.8) – це круглий дисплей від компанії Pimoroni розміром 2.1 дюйм з роздільною здатністю 480×480 та 18-ти бітним кольором. Матриця дисплею виконана за технологією IPS, що має дуже широкий кут огляду без змін в зображенні (175°), та достатньо яскрава при прямо сонячному світла, завдяки чому буде краще видно зображення в сонячний день, ніж у попередньої моделі. Однією з найбільших переваг даного дисплею є частота оновлення зображення в 60 FPS, що надає плавності зображенню.

Рисунок 1.8 – Дисплей HyperPixel 2.1.

Waveshare 3.5" – останній дисплейний модуль до розгляду. Це 3,5 дюймовий прямокутний дисплей роздільною здатністю 640×480 пікселів та 18-ти бітним кольором з ємкісним сенсором. Підключається даний дисплей так само як і попередній до 40-ка контактного роз’єму, і так само має частоту оновлення зображення 60 FPS. Нажаль, за відгуками користувачів, модуль має не надійну підтримку системою, через що часто виникають проблеми з нестабільною роботи як екрану так і його сенсору.

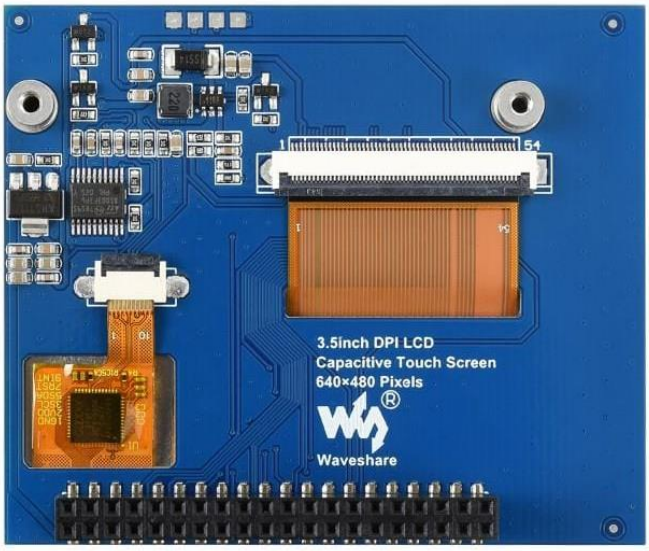


Рисунок 1.9 – Дисплей Waveshare 3.5".

Зрештою було обрано для даного проєкту дисплей HyperPixel 2.1, адже модуль має оптимальний розмір, порівняно з тими, що розглядалися. Модуль Adafruit PiTFT Plus 2.8 було одразу відхилено на фоні конкурентів через його технологією виконання матриці, адже вона відрізняється в значно гіршу сторону відносно інших дисплеїв за ту ж саму ціну. HyperPixel ідеально підходить до Raspberry Pi Zero як через метод відключення так і через метод його кріплення.

Висновки до розділу 1

Було виконано повний огляд існуючих рішень для відстеження та контролю швидкості транспорту. Проаналізувавши існуючі рішення, було знайдено комплекси що відповідають лише частково вимогам поставлених

задач в даній роботі, але жоден з них не відповідає критеріям для досягнення поставленої мети, а отже, аналіз остаточно підтвердив її актуальність.

Для виконання роботи необхідно розробити апаратне рішення і програмне забезпечення для нього. До апаратного комплексу було висунуто вимоги, аби воно мало: модуль обчислення, модуль GSM зв'язку так GNSS позиціювання. Після детального огляду існуючих рішень одноплатних комп’ютерів було виокремлено 3 конкуруючі моделі для порівняння: Raspberry Pi Zero, Banana Pi M2 Zero та Onion Omega2 Plus. В ході порівняння, обрано було саме Raspberry Pi Zero, за його надзвичайну популярність, яка призвела до великого різноманіття доступних модулів сумісних за наявними на платі роз’ємів та кріплень. Також дана платформа має чудову підтримку програмного забезпечення, що буде найкращим чином впливати на процес розробки програмної компоненти комплексу.

Наступним розглядались модулі зв’язку та геопозиціювання: OzzMaker BerryGPS-GSM, Waveshare SIM868 HAT та Waveshare SIM7600G-H HAT. Усі модулі розглядались як сумісні за кріпленням до вже обраного модуля обчислень, а також, критерієм було розташування обох необхідних сенсорів на одній платі для збереження компактності. Обрано було саме Waveshare SIM7600G-H адже він не обмежує жоден з інтерфейсів одноплатного комп’ютера, підключаючись до інтерфейсів pogo контактами, та навіть розширює кількість інтерфейсів підключення за допомогою USB хабу.

Останніми розглядались дисплеї, що мають відображати інтерфейс з необхідною інформацією, зокрема швидкістю транспорту. На огляді були: Adafruit PiTFT Plus 2.8", HyperPixel 2.1, Waveshare 3.5". Але зрештою було обрано саме HyperPixel 2.1, адже модуль має оптимальний розмір, порівняно з тими, що розглядалися. Технічні характеристики дисплею повністю задовольняють потреба, а метод кріплення до плати можна вважати найзручнішим з-поміж інших.

РОЗДІЛ 2
АЛГОРИТМИ, МЕТОДИ ТА ЗАСОБИ РОЗРОБКИ КОМПЛЕКСУ ВІДСТЕЖЕННЯ ШВИДКОСТІ

Для роботи усього комплексу необхідно щоб дані між модуля передавались надійно та швидко, для чого необхідно розглянути повнодуплексні протоколи передачі даних, а також протокол передачі в обмежених умовах зв’язку. Також в цьому проєкті необхідно отримувати дані про місцеположення, тож буде розглянуто яким чином це можливо реалізувати.

2.1 Алгоритм роботи протоколу WebSocket

WebSocket — це комп’ютерний протокол зв’язку, який забезпечує повнодуплексні канали зв’язку через одне з’єднання TCP. Протокол WebSocket був стандартизований IETF як RFC 6455 у 2011 році. WebSocket дає змогу клієнту робити запит даних на сервер, а потім отримувати керовані подіями відповіді від цього сервера в режимі реального часу. На відміну від багатьох вебтехнологій, WebSocket не використовує стратегію запит/відповідь, коли підключення відкривається під час виконання запиту, а потім закривається після його початкового виконання. У випадку WebSocket з’єднання залишається відкритим. Той факт, що клієнту не потрібно постійно опитувати сервер щодо оновлень, означає, що програма працює значно швидше та використовує ресурси ефективніше.

WebSocket складається з таких складових:

- клієнт: робить запит на необхідні дані на сервер;
- шлюз Websocket: забезпечує інтерфейс Websocket між клієнтом і сервером.
- сервер: надсилає оновлення клієнту через шлюз Websocket у реальному часі.

Метою WebSocket є надання альтернативи HTTP для зв’язку через TCP. Початковий запит клієнта може здійснюватися через HTTP за допомогою заголовка запиту на оновлення, після чого зв’язок відбувається за допомогою протоколу WebSocket. Іншими словами, замість того, щоб надсилати запит на *http://server.com*, ви надсилаєте його на *ws://server.com*. За допомогою цієї техніки досягаються такі цілі:

- зменшення накладних витрати на заголовок HTTP, передаючи лише важливу інформацію, що значно зменшує використання ресурсів і робить можливим спілкування в реальному часі;
- створення повнодуплексного середовища зв’язку, щоб позбутися потреби в опитуванні та архітектурі запит/відповідь. І клієнт, і сервер можуть передавати дані в потрібному напрямку, коли це необхідно. Зменшення мережевого трафіку також сприяє збільшенню швидкості додатків шляхом усунення затримки, яка зазвичай виникає під час вебзапиту;
- використання єдиного з’єднання TCP, щоб зменшити використання ресурсів;
- підтримка відкритого з’єднання через TCP, щоб уможливити потокове передавання даних;
- подолання обмежень за допомогою існуючих технологій, таких як:
  - опитування: періодичний, запланований, цикл запитів з боку клієнта для отримання інформації з сервера, навіть якщо така інформація не існує, витрачає величезну кількість ресурсів;
  - потокова передача HTTP: навіть якщо з’єднання залишається відкритим увесь час, використання стандартних заголовків HTTP збільшує розмір файлу та знижує ефективність;
  - Asynchronous JavaScript and XML (AJAX): використовує об’єкт JavaScript XmlHttpRequest для заміни лише частини сторінки за потреби для кожного оновлення. Використання заголовків HTTP збільшує розмір файлу, залежність від напівдуплексного зв’язку означає використання більшої

кількості каналів TCP, а потреба веб-сервера надсилати вміст окремим клієнтам збільшує використання ресурсів веб-сервера.

Встановлення з’єднання починається з рукостискання WebSocket (рис. 2.1), яке передбачає використання нової схеми WS або WSS. Використовуючи цю схему, очікується, що сервери та клієнти слідуватимуть стандартному протоколу підключення WebSocket. Встановлення з’єднання WebSocket починається з оновлення HTTP-запиту, який містить кілька заголовків, таких як «Connection: Upgrade», «Upgrade: WebSocket», «Sec-WebSocket- Key» тощо. Встановлення з’єднання ділиться на дві частини:

- запит – заголовок «Upgrade» позначає рукостискання WebSocket, тоді як «Sec-WebSocket-Key» містить випадкове значення в кодуванні Base64. Це значення довільно генерується під час кожного рукостискання WebSocket. Заголовок «Sec-WebSocket-Version» вказує для серверу версію протоколу, що використовується клієнтом. Поєднанні перелічені вище заголовки утворюють запит HTTP GET;
- відповідь – на відправлений вище запит приходить відповідь з HTTP статусом «101 Зміна протоколу» і заголовком «Sec-WebSocket-Accept» з хешованим ключем, надісланим в запиті, як повідомлення що сервер дозволяє ініціацію з’єдняння.

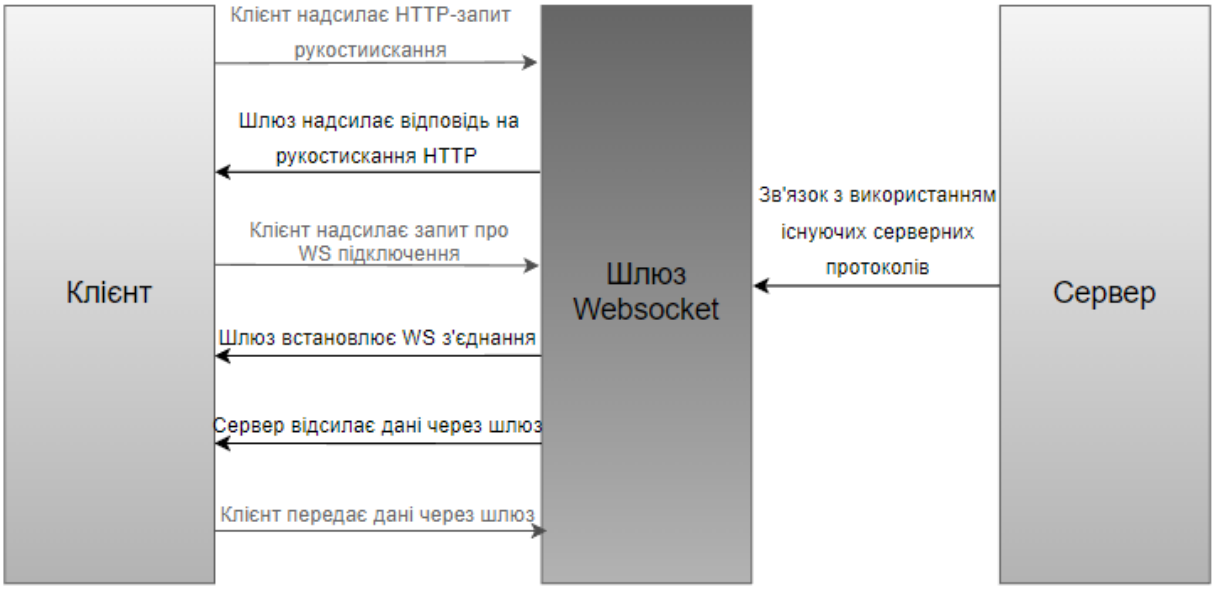


Рисунок 2.1 – Діаграма взаємодії в протоколі WebSocket.

Специфікація IETF RFC 6455 для протоколу WebSocket описує кілька способів використання кадрів даних, але, по суті, ці кадри (рис. 2.2) складаються з даних у бітовому форматі, які залежать від наступних полів:

- final (1 біт): встановлюється значення 1, щоб показати, що це останній кадр обміну даними;
- RSV1, RSV2, RSV3 (1 біт кожен): зазвичай встановлюється на 0, якщо обмін даними не визначає спеціальні значення для цих бітів;
- opcode (4 біти): визначає тип корисного навантаження, яке несе кадр даних, яке може мати будь-яке з наступних значень:
  - x0: кадр продовження;
  - x1: початок текстового кадру;
  - x2: початок бінарного кадру;
  - x3-x7: зарезервовано для подальших неконтрольних кадрів;
  - x8: Закрити з’єднання;
  - x9: Ping (початок перевірочного повідомлення);
  - xA: Pong (відповідь на перевірочне повідомлення);
  - xB-xF: зарезервовано для подальших контрольних кадрів;
- маска (1 біт): встановлюється значення 1, коли дані корисного навантаження маскуються, що означає включення ключа в поле «Ключ маскування»; клієнт має маскувати дані, які він надсилає на сервер, з міркувань безпеки;
- довжина корисного навантаження (варіюється від 7 бітів, 7+16 бітів або 7+64 біти): визначає довжину корисних даних. Дані корисного навантаження 125 біт або менше використовують лише 7 біт цього поля. Коли початкове 7-ми бітне значення дорівнює 126, тоді поле «Довжина корисного навантаження» має 7+16 біт. Якщо початкове 7-ми бітне значення дорівнює 127, тоді поле «Довжина корисного навантаження» має 7+64 біти;
- ключ маскування: містить ключ, який використовується для демаскування даних, коли біт поля маски встановлено на 1;

– дані корисного навантаження: дані корисного навантаження фактично розділені на дві частини:

○ дані розширення: описує, як надсилаються дані та як їх інтерпретувати. Наприклад, ви можете стиснути файл перед надсиланням. Дані розширення вкажуть, як інтерпретувати стислий файл, щоб повернути його до початкового стану;

○ дані застосунку: фактичні дані, що надсилаються між двома сторонами, займають решту кадру даних.

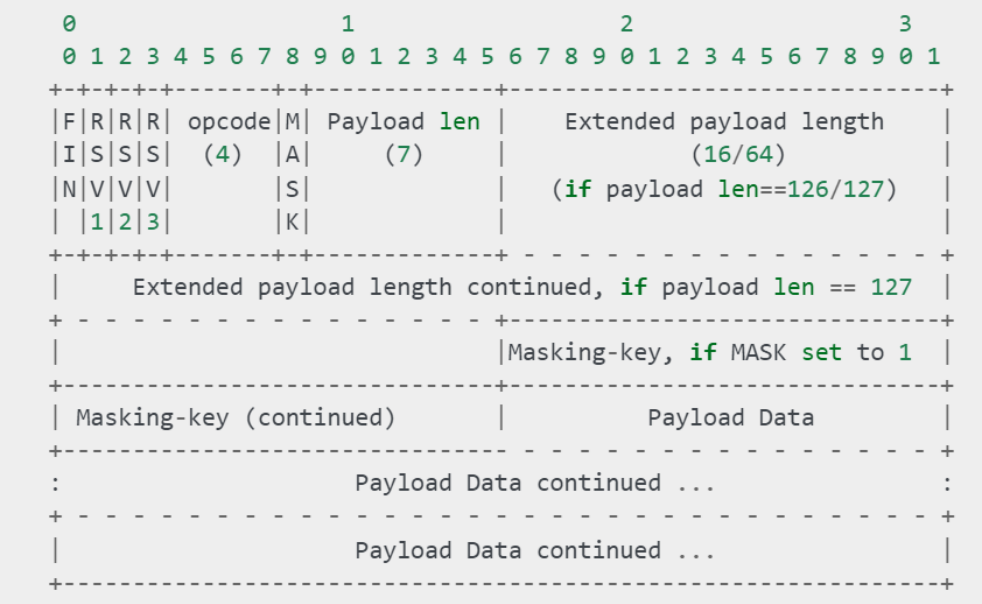


Рисунок 2.2 – Кадр даних протоколу WebSocket.

2.2 Алгоритм роботи протоколу MQTT

Сьогодні кожен день кількість підключених до Інтернету речей стрімко зростає, але не усім пристроям потрібне потужне стабільне підключення, до того в кожних умовах таке підключення можливе. Саме тому існує цілий кластер пристроїв Internet of Things (IoT) з обмеженою доступністю та малими обсягами даних для передачі. Для вирішення таких завдань на початку 2000-их років було створено протокол передачі даних Message Queuing Telemetry Transport (MQTT).

MQTT — це протокол обміну повідомленнями для обмежених мереж із низькою пропускною здатністю та пристроїв IoT із надзвичайно високою

затримкою. Оскільки протокол спеціалізується на середовищах із низькою пропускною здатністю та високою затримкою, це ідеальний протокол для міжмашинного зв’язку.

Комунікаційна стратегія публікації/підписки (pub/sub) MQTT (рис. 2.3), спрямована на максимізацію використання пропускної здатності, є заміною традиційної споживчої архітектури, яка безпосередньо взаємодіє з кінцевою точкою. Однак у парадигмі pub/sub клієнт, який передає дані (видавець), відокремлений від клієнтів, які отримують інформацію (або підписник). Оскільки ані автори, ані клієнти не спілкуються один з одним одразу, їхньою взаємодією в них займаються треті особи, які називаються брокерами.

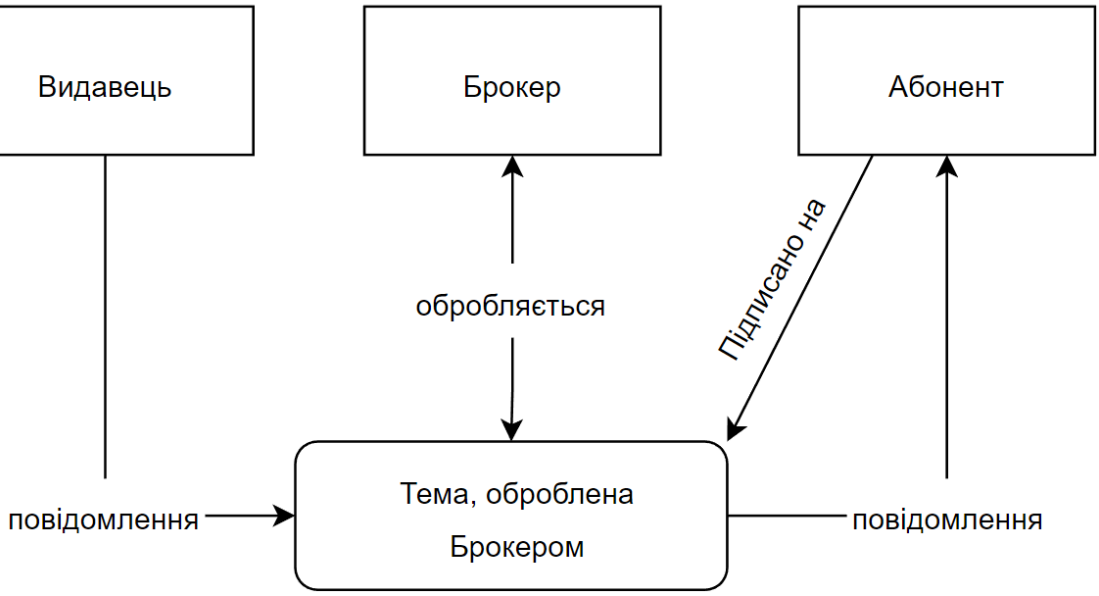


Рисунок 2.3 – Схема взаємодії протоколу MQTT.

Видавці та підписники — це два типи клієнтів MQTT, залежно від того, публікує клієнт чи отримує повідомлення. Можна поєднати дві функції в одному клієнті MQTT. Публікація — це коли пристрій (або клієнт) хоче надіслати інформацію на сервер (або посередника). Посередині є центральний сервер або брокер, який діє як посередник. Кожне вхідне повідомлення фільтрується брокером, який потім надсилає його відповідним клієнтам.

Клієнтами є як видавці, так і підписники. Підписники підписуються на актуальні для них повідомлення. Фільтрування за темами, за вмістом або за типом — усі можливі. Теми можна розташувати в логічному порядку. Видавцю та підписникам не потрібно ідентифікувати один одного під час використання структури публікації/підписки. Для налаштування та синхронізації вони не залежать один від одного. Розповсюдження повідомлень здійснюється за принципом «один до багатьох» і може масштабуватися з точки зору клієнтської бази, не створюючи надмірного навантаження на видавців.

Одним із способів, як MQTT зменшує комунікації, є використання чітко визначеної компактної структури повідомлення. Кожне повідомлення містить двобайтовий фіксований заголовок. Введення необов’язкового заголовка збільшує кількість повідомлення. Корисне навантаження повідомлення обмежене 256 МБ. Є можливість вибирати між обмеженням передачі даних і максимізацією надійності за допомогою трьох різних рівнів Quality of Service (QoS), які включають:

- QoS 0 : забезпечує мінімальний обсяг передачі даних. На цьому етапі кожне повідомлення передається користувачеві лише один раз без перевірки;
- QoS 1 : цей брокер намагається надіслати повідомлення, а потім перевіряє відповідь користувача на схвалення. Якщо протягом певного часу не отримано підтвердження, повідомлення надсилається повторно;
- QoS 2 : чотирьохетапне рукостискання використовується клієнтом і брокером, щоб забезпечити отримання інформації лише один раз.

QoS 0 (рис. 2.4) гарантує доставку за найменші зусиль, але гарантії доставки немає. Одержувач не підтверджує отримання повідомлення, і повідомлення не зберігається та не передається відправником. Рівень QoS 0 часто називають «запустив і забув» і забезпечує таку саму гарантію, як і базовий протокол TCP.

Рисунок 2.4 – Схема роботи QoS 0.

Рівень QoS 1 (рис. 2.5) гарантує, що повідомлення буде доставлено одержувачу принаймні один раз. Відправник зберігає повідомлення, доки не отримає пакет PUBACK від одержувача, який підтверджує отримання повідомлення. Повідомлення можна надіслати або доставити кілька разів.

Рисунок 2.5 – Схема роботи QoS 1.

Відправник використовує ідентифікатор пакета в кожному пакеті, щоб зіставити пакет PUBLISH з відповідним пакетом PUBACK. Якщо відправник не отримує пакет PUBACK протягом розумного періоду часу, відправник повторно надсилає пакет PUBLISH. Коли одержувач отримує повідомлення з QoS 1, він може негайно його обробити. Наприклад, якщо одержувачем є брокер, він надсилає повідомлення всім підписаним клієнтам, а потім відповідає пакетом PUBACK. Якщо клієнт публікації надсилає повідомлення знову, він встановлює прапор дублікату (DUP). У QoS 1 цей прапор DUP використовується лише для внутрішніх цілей і не обробляється брокером чи

клієнтом. Одержувач повідомлення надсилає PUBACK, незалежно від прапора DUP.

QoS 2 (рис. 2.6) — це найвищий рівень обслуговування в MQTT. Цей рівень гарантує, що кожне повідомлення буде отримано одержувачами лише один раз. QoS 2 є найбезпечнішим і найповільнішим рівнем якості обслуговування. Гарантія забезпечується щонайменше двома потоками запитів/відповідей (рукостискання з чотирьох частин) між відправником і одержувачем. Відправник і отримувач використовують ідентифікатор пакета вихідного повідомлення PUBLISH для координації доставки повідомлення.

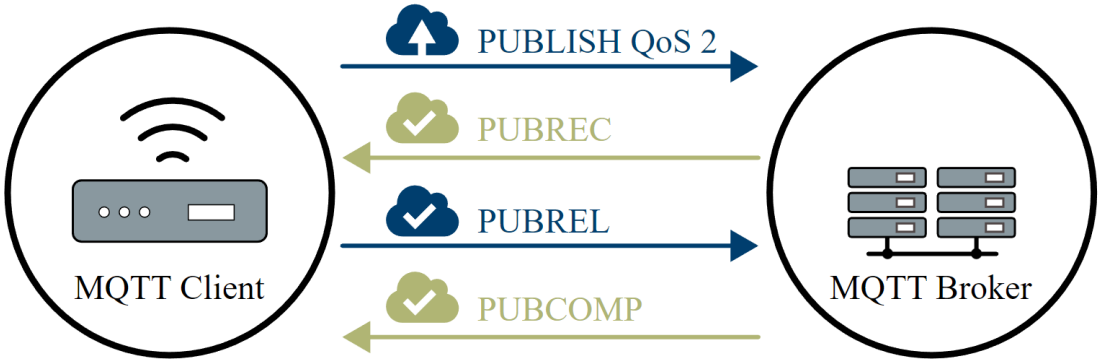


Рисунок 2.5 – Схема роботи QoS 2.

Коли отримувач отримує пакет QoS 2 PUBLISH від відправника, він відповідним чином обробляє повідомлення публікації та відповідає відправникові пакетом PUBREC , який підтверджує пакет PUBLISH. Якщо відправник не отримує пакет PUBREC від одержувача, він знову надсилає пакет PUBLISH із позначкою дублікату (DUP), доки не отримає підтвердження. Як тільки відправник отримає пакет PUBREC від одержувача, відправник може безпечно відкинути початковий пакет PUBLISH. Відправник зберігає пакет PUBREC від одержувача та відповідає пакетом PUBREL .

Після того, як отримувач отримає пакет PUBREL, він може відкинути всі збережені стани та відповісти пакетом PUBCOMP (те саме стосується, коли відправник отримує PUBCOMP). Поки отримувач не завершить обробку

та не надішле пакет PUBCOMP назад відправнику, отримувач зберігає посилання на ідентифікатор вихідного пакета PUBLISH. Цей крок важливий, щоб уникнути повторної обробки повідомлення. Після того, як відправник отримає пакет PUBCOMP, ідентифікатор пакета опублікованого повідомлення стає доступним для повторного використання. Коли потік QoS 2 завершено, обидві сторони впевнені, що повідомлення доставлено, і відправник має підтвердження доставки.

Якщо пакет буде втрачено в дорозі, відправник несе відповідальність за повторну передачу повідомлення протягом розумного періоду часу. Це однаково вірно, якщо відправником є клієнт MQTT або брокер MQTT . Одержувач несе відповідальність відповідним чином відповісти на кожне командне повідомлення.

2.3 Алгоритм взаємодії з GNSS

З кожним днем наше життя все більше покращується завдяки розвитку технологій. У нашому розпорядженні так багато технологій, які об’єднують усі частини нашого життя, від ваших фітнес-трекерів і голосового асистенту Siri до автоматизованих кавових машин, розумних будинків і розумних автомобілів. Світ наближається до ідеального середовища, де все пов’язано, а життя стає доступнішим. Одним із головних факторів, що сприяє цій зміні, є GNSS. Global Navigation Satellite System (GNSS) — це тип супутникової системи з широким глобальним покриттям. Він використовує супутники для забезпечення незалежного геопросторового позиціонування. Це дає змогу звичайним електронним пристроям визначати своє місцезнаходження, позиціонування та навігацію за допомогою датчиків, які приймають радіосигнали від супутників.

Система GNSS також дозволяє синхронізувати в реальному часі події, кінцеві пристрої тощо. Вона охоплює широку сферу супутникового відстеження та всі його застосування. Кожна система GNSS складається з групи супутників, які передають високочастотні сигнали з космосу на

приймачі по всьому світу. Глобальна навігаційна супутникова система використовується для позиціонування, навігації та визначення часу (PNT).

На даний момент у світі працює близько чотирьох повністю функціональних систем GNSS. Популярна Global Positioning System (GPS) Сполучених Штатів, вона ж NAVSTAR, була першою у своєму роді, за якою слідували супутникові угруповання з інших країн. Є китайська навігаційна супутникова система BeiDou, російська Глобальна навігаційна система (ГЛОНАСС) і Galileo Європейського Союзу. Інші регіональні системи доповнення включають супутникову систему Quasi-Zenith, що належить Японії.

Коли згадується глобальна навігаційна супутникова система, люди в першу чергу думають про GPS. Тому важливо виправити поширену помилкову думку про те, що GPS – це те саме, що GNSS. GPS або система глобального позиціонування – це тип GNSS, який належить США та інакше називається NAVSTAR. GNSS — це термін, який позначає кожну форму глобальної супутникової системи позиціонування, класичним прикладом якої є GPS. Таким чином, GPS є різновидом системи GNSS. Це перша система GNSS, яку було винайдено міністерством оборони Сполучених Штатів для використання як незалежна військова навігаційна система. Система навігації та позиціонування була випущена для загального використання роками пізніше. GPS є найточнішою системою позиціонування, доступною для сучасних користувачів. Однак обладнання, сумісне з GNSS, тобто все обладнання, яке використовує позиціонування, навігацію та визначення часу, може використовувати сигнали супутників у братніх мережах GNSS, але за межами GPS, і все одно отримувати подібну інформацію.По суті, GPS не є синонімом GNSS. Він відноситься до прикладів систем GNSS і може бути класифікований поряд з іншими типами GNSS.

Системи GNSS, що використовуються в цивільній навігації, включають GNSS 1 і GNSS 2. GNSS 1 — це глобальна супутникова система першого

покоління, яка поєднує існуючі супутникові навігаційні системи з іншими системами доповнення (наприклад, супутниковою системою доповнення або SBAS). Приклади включають GPS, доповнений системою розширення широкого діапазону (WAAS) у США. GNSS 2 — супутникова система другого покоління. Вони забезпечують точність і цілісність, що використовується для моніторингу цивільної навігації. Приклади включають Galileo, ГЛОНАСС тощо. Говорячи про продуктивність різних систем GNSS, використовуються чотири критерії, а саме:

- точність: вимірює, наскільки фактичне положення, швидкість або час відповідне до даних, виміряних за допомогою GNSS;
- цілісність: описує здатність системи надавати ступінь достовірності наданим нею даним, а також подавати сигнал тривоги, коли виявлено нестандартну інформацію;
- безперервність: це здатність GNSS надавати постійну інформацію без перебоїв;
- доступність: доступність системи GNSS вимірюється у відсотках; значення показує, як часто сигнали від цієї системи відповідають іншим трьом критеріям, згаданим вище.

Як згадувалося раніше, додавання регіональних систем доповнення може покращити якість вихідних даних цих глобальних навігаційних супутників.

Може здатись приголомшливим, як супутники, що обертаються навколо Землі, можуть повідомити людині, де вона знаходиться, і визначити напрямок до найближчого продуктового магазину. Тим не менш, ми можемо знати основні принципи, якими керуються системи GNSS, і як вони працюють. Кожна GNSS складається з трьох основних компонентів:

- супутники або космічний сегмент – це стосується фактичних супутників, що обертаються навколо Землі. Угруповання супутників GPS розташоване в шести рівновіддалених площинах. Кожен літак має принаймні

чотири супутники, і така домовленість гарантує, що принаймні чотири супутники завжди доступні для кожного приймача, будь то телефон, наручний годинник, розумний автомобіль тощо;

- контрольні сегменти – для кожної системи GNSS є побудовані станції, розташовані навколо екватора для контролю, моніторингу, відстеження та зв’язку з супутниками. Це гарантує, що супутникові годинники синхронізуються та контролюються для обслуговування, як будь-яка IT-інфраструктура;
- сегмент користувачів – це охоплює все, що має приймач, наприклад мобільні телефони, автомобілі, правоохоронні системи, літаки тощо. Приймачі інтерпретують сигнали та точно визначають місцезнаходження за допомогою процесу, відомого як трилатерація, який полягає в позиціонуванні об’єкта з трьох відстаней.

Нас цікавить саме остання ланка цього процесу, адже ми і є кінцеві користувачі. Приймач GNSS складається з двох елементів: процесора та антени. Антена вловлює сигнал, а процесор декодує необхідну інформацію. Деякі приймачі можуть мати дві антени: основну та додаткову. Коли супутник транслює, він передає час, коли надсилає цей сигнал, закодований в інформації про сигнал. Потім приймач використовує різницю в часі від моменту трансляції сигналу до моменту його отримання, враховуючи часову затримку, спричинену навколишніми шарами землі. Потім, використовуючи швидкість світла, він розраховує відстань, пройдену сигналами від трьох різних супутників.

Справді, GNSS виявився критично важливим у теперішньому та майбутньому міському середовищі, особливо для Інтернету речей (IoT) . У міру того як технологія поступово розвивається в інтелектуальні пристрої та прилади, роль GNSS стає все більш очевидною. GNSS забезпечує відстеження в реальному часі, час, навігацію та інші сфери міжмашинного зв’язку, що є основою керування пристроями IoT. Дійсно, у міру зростання

додатків Інтернету речей зростає попит на супутникову систему, і ці вимоги поступово задовольняються, відкриваючи простір для нових відкриттів. Важливість GNSS в Інтернеті речей полягає просто в здатності пристроїв знати своє місцезнаходження, розташування інших машин навколо них, а також у здатності зіставляти дані в корисну інформацію. Наприклад, здатність автомобіля відчувати інші транспортні засоби на своєму шляху та уникати зіткнення або навіть відчувати об’їзд попереду та йти коротшим маршрутом, потенційні застосування нескінченні.

National Marine Electronics Association (NMEA) існувала задовго до того, як було винайдено GPS. Згідно з веб-сайтом NMEA, асоціація була створена в 1957 році групою дилерів електроніки для покращення зв’язку з виробниками. Сьогодні у світі GPS NMEA є стандартним форматом даних, який підтримується всіма виробниками GPS, подібно до того, як ASCII є стандартом для цифрових комп’ютерних символів у комп’ютерному світі. Мета NMEA — надати користувачам обладнання можливість поєднувати апаратне та програмне забезпечення. Дані GPS у форматі NMEA також полегшують розробникам програмного забезпечення написання програмного забезпечення для різноманітних GPS-приймачів замість того, щоб писати спеціальний інтерфейс для кожного GPS-приймача.

Для пояснення структури повідомлення NMEA скористуємось прикладом:
«*$GPGGA,181908.00,3404.7041778,N,07044.3966270,W,4,13,1.00,495.144,M,29.200,M,0.10,0000*40»*, де:

- **GP –** визначає позицію GPS (GL позначає ГЛОНАСС);
- **181908.00 –** це позначка часу (UTC у годинах, хвилинах і секундах);
- **3404,7041778** – це широта у форматі DDMM.MMMMM;
- **N –** позначає північну широту;
- **07044,3966270** – це довгота у форматі DDMM.MMMMM;
- **W** позначає західну довготу;

- **4** – показник якості (точність);
- **13 –** кількість супутників , що використовуються для визначення координати;
- **1,00 –** горизонтальне зниження точності (HDOP);
- **495.144 –** це висота GPS-антени;
- **M –** одиниця висоти (метри або фути);
- **29,200 –** це геоідальне розділення;
- **M –** одиниця геоїдального поділу (метри або фути);
- **1,0 –** вік корекції (якщо є);
- **0000 –** показує ідентифікатор станції корекції (якщо є);
- ***40 –** це контрольна сума.

2.4 Топологія мережі кінцевого комплексу

Перш за все необхідно визначитись яким чином будуть комунікувати між собою внутрішні програмні частини апаратного комплексу. З огляду на те що інтерфейс та обчислювальні компоненти будуть працювати на різних технологіях і інтерфейс буде виконано як вебзастосунок було обрано протокол обміну даними WebSocket (рис. 2.6). Завдяки такому вибору майже нівелюється затримка, яку можна було б отримати користуючись до прикладу протоколом HTPP. Також завдяки тому що протокол повнодуплексний, можлива реалізація двостороннього спілкування.

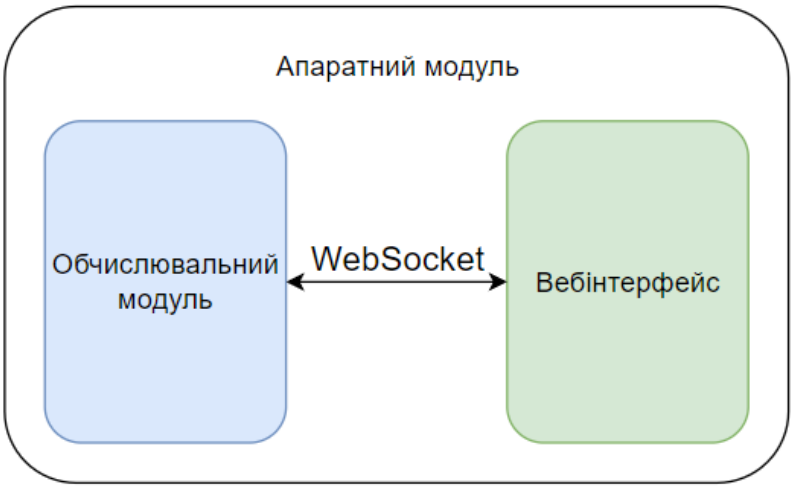


Рисунок 2.6 – Комунікація програмних модулів апаратного комплексу.

Для комунікації апаратного модуля з вебінтерфейсом диспетчеризації вирішено використовувати протокол MQTT, що найбільше підходить для даних обмежених умов зв’язку. Так як в якості підключення до мережі буде використовуватись моільний зв’язок, то з’єднання може бути обірване в будь-який момент і для того щоб дані не зникли в момент погано зв’язку було обрано цей протокол. Тож в центрі усього готового комплексу має стояти сервер з двома шлюзами (рис. 2.7): MQTT брокер та шлюз WebSocket. Далі до MQTT брокера під’єднуються апаратні комплекси за принципом «один до багатьох». І для отримання актуальних даних від апаратних модулів у вебінтерфейсі використовується WebSocket, також за принципом «один до багатьох».

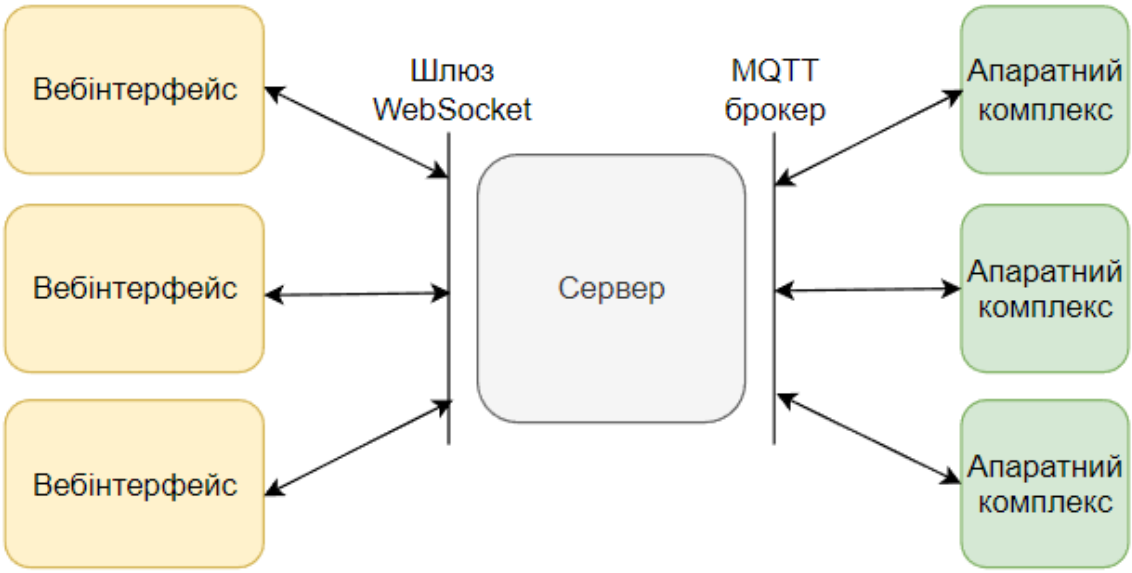


Рисунок 2.7 – Топологія мережі кінцевого комплексу.

Висновки до розділу 2

В цьому розділі було розглянуто алгоритми та технології необхідні для досягнення поставленої мети, а саме: протоколи MQTT та WebSocket, взаємодію з даними GNSS, а також розглянули яким чином усі компоненти комплекси будуть взаємодіяти між собою.

Для початку було розглянуто протокол MQTT, що часто використовується саме для IoT пристроїв з обмеженою доступністю до

мережі. Даний протокол має посередника між відправником та отримувачем у вигляді брокера. Брокер забезпечує ведення сесій підключень та усі повідомлення що надходять за темами надсилає підписаним на них користувачам. Також брокер відіграє важливу роль у стабільності отримання даних. Для цього в протоколі існує QoS трьох рівнів, де QoS0 – найлегший по навантаженню мережі рівень, але не може стверджувати що повідомлення дійшло, а QoS2 – рівень де отримувач гарантовано отримує дані, але натомість використовується більше навантаження на мережу.

Також було розглянуто інший повнодуплексний протокол передачі даних WebSocket. WebSocket не використовує стратегію запит/відповідь, коли підключення відкривається під час виконання запиту, а потім закривається після його початкового виконання. У випадку WebSocket з’єднання залишається відкритим. Той факт, що клієнту не потрібно постійно опитувати сервер щодо оновлень, означає, що програма працює значно швидше та використовує ресурси ефективніше.

Було також розглянуто систему GNSS, завдяки якій можливе геопозиціонування. GNSS складається з великої кількості супутників які транслюють час, коли надісланий цей сигнал, закодований в інформації про сигнал. Потім приймач використовує різницю в часі від моменту трансляції сигналу до моменту його отримання, враховуючи часову затримку, спричинену навколишніми шарами землі і отримує місцезнаходження. Також було розглянуто формат NMEA, який є найпоширенішим видом даних з приймачів сигналу GNSS.

РОЗДІЛ 3
РОЗРОБКА АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ ВІДСТЕЖЕННЯ ШВИДКОСТІ ВЕЛОСИПЕДУ НА БАЗІ RASPBERRY PI ZERO

Для досягнення поставленої в цій роботі мети необхідно розробити апаратно-програмний комплекс, що складається з апаратної частини, програмної частини для неї та програмної частини у вигляді вебзастосунку для адміністрування мережі апаратних модулів. Апаратний комплекс в свою чергу ділиться на три апаратні компоненти і два програмних модулі. Програмні модулі складаються з модуля зв’язку та позиціонування і модуля інтерфейсу.

3.1 Налаштування апаратної платформи

З обраних в першому розділі апаратник компонентів необхідно зібрати повноцінний комплекс, а також забезпечити його програмним забезпеченням.

3.1.1 Збірка та підключення апаратної частини

Зазвичай Raspberry Pi Zero постачається з не розпаяною гребінкою з 40-ка з’єднань, тож перед початком підключення інших модулів необхідно цю гребінку припаяти. Також необхідно закріпити стійки кріплення в спеціально відведені для того місця (рис. 3.1).

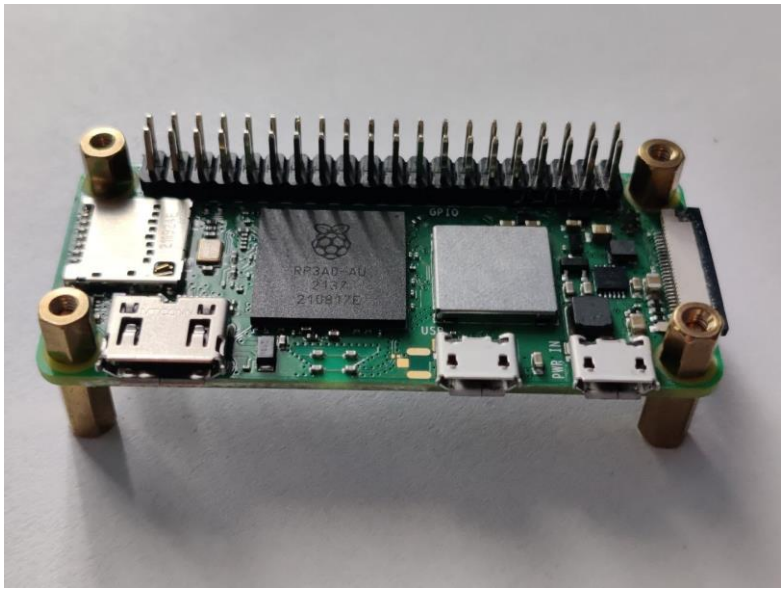


Рисунок 3.1 – Виконання підготовчих робіт з Raspberry Pi.

Після виконання приготувань, слід встановити перший модуль, а саме модуль GNSS та GSM сигналу. Як описувалось в першому розділі, даний модуль під’єднується спеціальними pogo з’єднаннями, що є просто підпружиненими пінами. Під час закріплення плати до стійок важливо звернути увагу аби піни дотикались виключно до необхідних контактів, а це +5B, GND, USB+ та USB- (рис. 3.2). Так як існує великий ризик що під час з'єднання деякі піни таки доторкнуться до інших до моменту закріплення – то весь процес з'єднання має проходити на відключених від живлення компонентах.

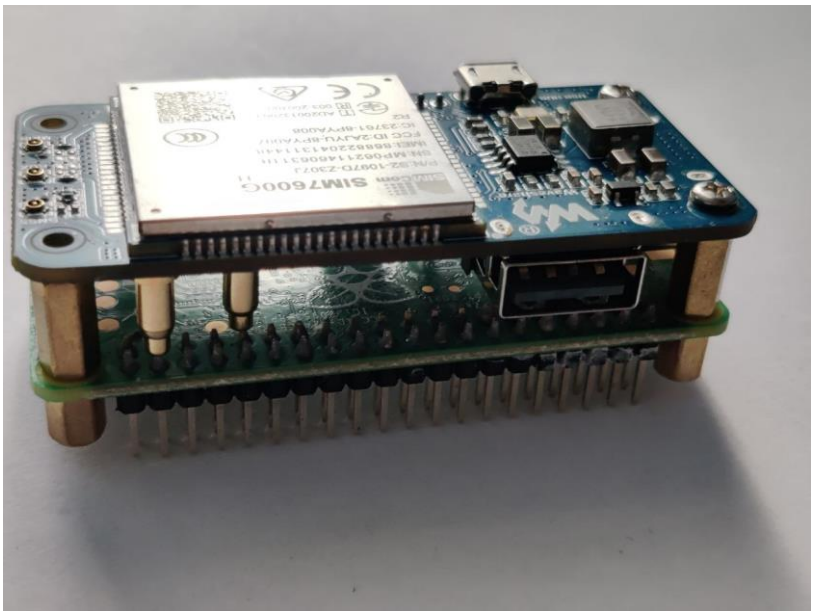


Рисунок 3.2 – Закріплення WaveShare модуля зв’язку на плат обчислення.

Наступним кроком необхідно під’єднати до роз’ємів IPEX/U.FL на платі перехідники з U.FL на SMA. На платі присутні три таких роз’єми, необхідні нам позначені маркуванням «GNSS» та «MAIN» (рис. 3.3). Ці роз’єми відповідають за підключення антен мобільного зв’язку так супутникових даних GNSS. Цей тип з’єднання надзвичайно популярний в техніці, де дуже мало місця адже він дуже малий за розмірами, тож його можна зустріти, до прикладу, в мобільній техніці, де його застосовують для подовження з’єднання антени зв’язку. В цьому проєкті використовуються

масивні антени лише для проєктування, але в подальшому в ці роз’єми можна становити компактну гнучку антену.

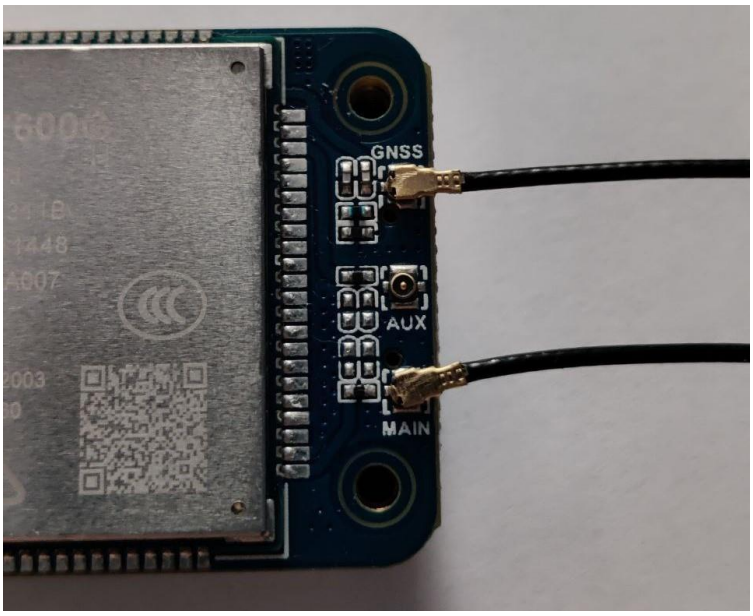


Рисунок 3.3 – З’єднання GNSS та GSM антен.

Наступним кроком треба під’єднати SD картку пам'яті та nano SIM картку, адже в подальшому це буде зробити важче (рис. 3.4).

Рисунок 3.4 – Під’єднані до роз’ємів картки.

Останнім кроком необхідно під’єднати дисплей до 40-ка пінового роз’єму (рис. 3.5). Так велика кількість пінів одночасно під’єднується, то виникає значний опір, що потребує зусиль, але важливо зважати на скляну

поверхню лицевої частини дисплею й не перенавантажувати одну точку, адже це може призвести до його пошкодження.

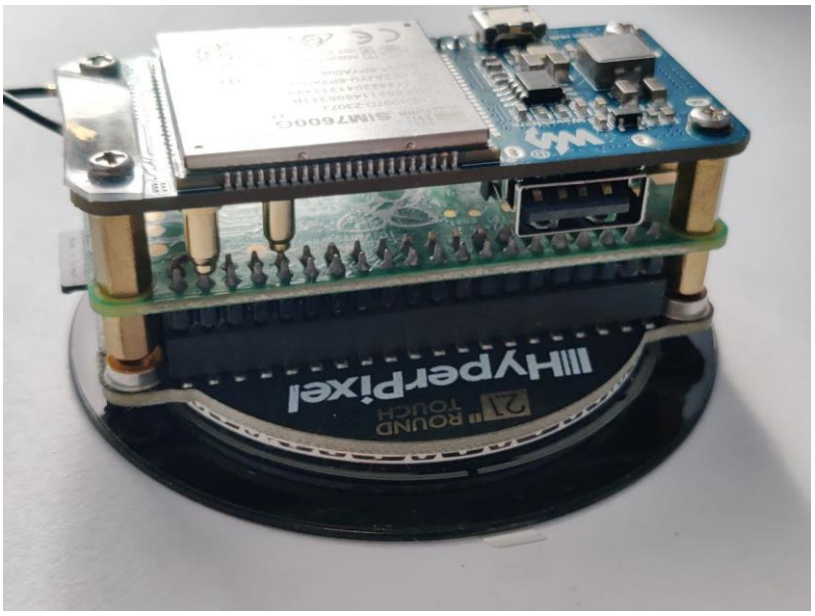


Рисунок 3.5 – Під’єднаний дисплей до комплексу.

Готовий модуль потребує живлення плати Raspberry Pi Zero 5B. До готового апаратного комплексу живлення можна подати напругу через три доступних роз’єми micro USB (рис. 3.6): два на платі Raspberry Pi і один на платі зв’язку, адже їх поєднано на лінії 5B і нулю підпружиненими пінами. Не дивлячись на те що до плати розширення подавати напругу можна, але вважається не правильним, адже немає достатньо міцного з’єднання.

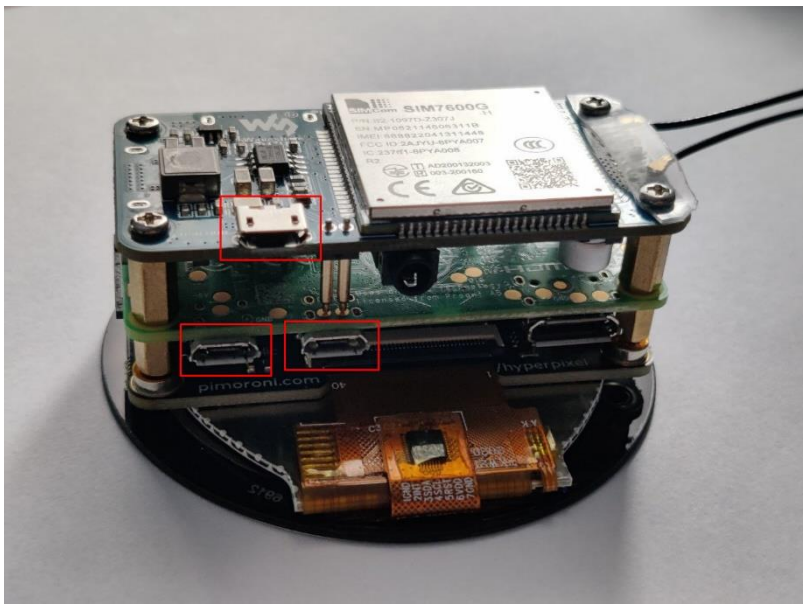


Рисунок 3.6 – Доступні роз’єми живлення.

3.1.2 Налаштування операційної системи та збірка образу

Так як апаратний модуль потенційно має бути швидко повторюваний було вирішено одноразово встановити і налаштувати ОС і стиснути до готового для інсталяції файлу. Для встановлення початкової операційної системи на обчислювальним модуль необхідно встановити під’єднати до комп’ютера SD картку та за допомогою офіційної програми від компанії Raspberry Pi – Raspberry Pi Imager. В якості базової ОС було обрано офіційну облегшену Raspberry Pi OS Lite розрядністю 32 біт, це видозмінені Linux Debian 11 з кодовим ім’ям «bullseye» з ядром версії 5.15.

Після того як дані на SD картку записані необхідно зробити маніпуляції до її повернення в Raspberry pi Zero, а саме:

- в директорії */boot* створити порожній файл з назвою «ssh», що дасть зрозуміти ОС що нам потрібно увімкнути цей протокол автоматично;
- в директорії */boot* створити файл з назвою «wpa_supplicant.conf» та додати наступний текст в нього:

```
country=UA
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
network={
    ssid="<Назву WIFI>"
    psk="<Пароль від WIFI>"
    key_mgmt=WPA-PSK
}
```

Після цих підготувань вставивши картку в молуль обчислення і зажививши модуль, після завантаження буде можливим під ’єднатися до нього по SSH протоколу в локальній мережі, параметри якої було вказано в файлі вище.

Отримавши доступ до терміналу, щоб налаштувати пріоритет підключення до мережі вводимо команду «sudo nano /etc/dhcpcd.conf», та у відкритому вікні вставляємо наступний код і перезавантажуємо модуль:

```
interface wlan0
```

```
metric 200

interface usb0

metric 300
```

Завдяки цій операції модуль буде під’єднуватись спочатку до WiFi, що було вказано, а вже потім до модулю зв'язку по інтерфейсу USB.

Наступним кроком є встановлення програмного забезпечення для підтримки дисплею, для цього необхідно встановити інструмент Git командою «*sudo apt install --no-install-recommends git -y*». Далі вде за допомогою встановленого інструменту завантажуємо надані виробником файли: «*git clone https://github.com/pimoroni/hyperpixel2r*», переходимо в директорію що завантажилась і запускаємо bash файл для встановлення і обов’язково перезавантажуємо пристрій.

Наступним кроком є встановлення мінімального середовища робочого столу та браузера. Перед тим необхідно виконати оновлення даних про пакети командою «*sudo apt update && sudo apt upgrade -y*». Наступник кроком вже встановлюмо самі пакети «*sudo apt install --no-install-recommends xserver-xorg x11-xserver-utils xinit openbox chromium-browser -y*», важливо зазначити, що завантаження і встановлення пакетів може зайняти певний час. Після встановлення відкриваємо файл автозапуску середовища робочого столу і вставляємо наступний код

```
# Disable any form of screen saver / screen blanking / power management

xset s off

xset s noblank

xset -dpms

# Allow quitting the X server with CTRL-ATL-Backspace

setxkbmap -option terminate:ctrl_alt_bksp

# Start Chromium in kiosk mode

sed -i 's/"exited_cleanly":false/"exited_cleanly":true/' ~/.config/chromium/'Local
State'

sed -i 's/"exited_cleanly":false/"exited_cleanly":true/;
s/"exit_type":"[^"]\+"/"exit_type":"Normal"/' ~/.config/chromium/Default/Preferences

chromium-browser \
```

```
  --start-fullscreen \
  --kiosk \
  --incognito \
  --noerrdialogs \
  --no-first-run \
  --fast \
  --fast-start \
  --disable-infobars \
  --disable-features=TranslateUI \
  --disk-cache-dir=/dev/null \
  --overscroll-history-navigation=0 \
  --disable-pinch \
  'http://localhost'
```

Наступним кроком вимикаємо авторизацію при кожному запуску ОС командою «*sudo raspi-config nonint do_boot_behaviour B2*».

Далі додаємо скрипт автозапуску, відкриваємо файл командою «*sudo nano ~/.profile*» і додаємо туди наступний код:

```
if [ -f /boot/asset-hard.tar ]; then
    sudo rm -rf /var/www/html/*
    sudo tar xvf /boot/asset-hard.tar -C /var/www/html
    sudo rm -rf /boot/asset-hard.tar
    sudo chmod -R 755 /var/www/html
fi
if [ -f /boot/py-app.tar ]; then
    if [ ! -d /home/pi/app ]; then
        sudo mkdir /home/pi/app
    fi
    sudo rm -rf /home/pi/app/*
    sudo tar xvf /boot/py-app.tar -C /home/pi/app/
    sudo rm -rf /boot/py-app.tar
    sudo chmod -R 755 /home/pi/app/
    sudo systemctl restart asset.service
fi
[[ -z $DISPLAY && $XDG_VTNR -eq 1 ]] && startx -- -nocursor
```

Далі необхідно встановити вебсервер nginx, що міг би запускати вебзастосунок інтерфейсу, робимо це командою «*sudo apt install --no-install-recommends nginx -y*». Відкриваємо файл налаштування вебсерверу командою «*sudo nano /etc/nginx/sites-enabled/default*» і додаємо туди наступну конфігурацію серверу:

```
server {
        listen 127.0.0.1:80;
        root /var/www/html;
        index index.html;

        location / {
                try_files $uri $uri/ /index.html;
        }
}
```

Далі встановлюємо менеджер пакетів pip командою «*sudo apt install python3-pip -y*» і вже за допомогою цього менеджера встановлюємо потрібні модулі «*sudo pip3 install pyserial websocket-server paho-mqtt*». Необхідно створити deamon для автоматичного запуску програми при запуску системи, для цього створюємо та відкриваємо файл командою «*sudo nano /etc/systemd/system/asset.service*», до файлу додаємо наступний код:

```
[Unit]
Description=Asset device python script service (gps, ws, mqtt)
After=multi-user.target
[Service]
Type=simple
Restart=always
ExecStart=/usr/bin/python3 /home/pi/app/app.py
[Install]
WantedBy=multi-user.target
```

І після збереження файлу виконуємо операцію, а саме перезавантаження системи deamon і включення створеного вище «*sudo systemctl daemon-reload && sudo systemctl enable asset.service»*.

В результаті виконаних налаштувань маємо повністю готову до використання ОС де налагоджено роботу з дисплеєм, встановлені усі необхідні модулі для роботи програмної частини, налаштовано автоматичний запуск браузера після запуску системи, що слугуватиме вебінтерфейсом для апаратного модуля. Маючи повністю підготовлену ОС, необхідно її скопіювати, стиснути і підготувати до наступних встановлень, аби маючи цей образ можна було одразу встановити повністю підготовлену ОС і не проходити довгий процес налаштування повторно. Підготовку системи до створення образу необхідно виконати в два кроки: зчитати ОС в файл з розширенням «.img» та видалити усі порожні блоки образу. Для того щоб зчитати ОС в файл скористаємося застосунком Win32 Disk Imager для ОС Windows. Під’єднавши картку пам’яті до комп’ютера обираємо її як джерело і обираємо шлях і назву файлу куди зчитані дані будуть записані. Після успішного копіювання ОС використовуємо утиліту pishrink для зменшення її розміру в декілька кроків:

1) виконати завантаження утиліти командою «*wget https://raw.githubusercontent.com/Drewsif/PiShrink/master/pishrink.sh*»;

2) надати файлу права на виконання командою «*sudo chmod +x pishrink.sh*»;

3) додати виконавчий файл в загальнодоступну директорію «*sudo mv pishrink.sh /usr/local/bin*»;

4) перейди в директорію де знаходиться файл попередньо скопійованої ОС;

5) виконати команду стиснення образу «*sudo pishrink.sh -z myimg.img*», де myimg.img - це назва файлу образу.

Отже, після виконання всіх описаних вище процесів ми отримуємо готовий файл, розміром трохи більше за початковий, але вже з усіма встановленими і налаштованими компонентами. Тепер його можна легко встановлювати використовуючи той самий застосунок від Raspberry Pi,

зберігаючи кожен раз близько години часу, необхідного для виконання первинного налаштування ОС з нуля.

3.2 Розробка програмної частини апаратного комплексу

Як вже розглядалося в другому розділі даної роботи, програмна частина апаратного модуля буде складатися з двох складових: інтерфейс та модуль обробки даних. Виконання цих задач різними технологіями було вирішено після проведення тестування. Під час тесту було створено найпростіший макет інтерфейсу засобами пакету pygame, але було виявлено, що малювання кожного кадру займає дуже багато ресурсів процесора і не досягає мінімальної необхідної частоти – частоти оновлення дисплею в 60 разів на секунду. Також інтерфейс виглядав дуже погано, адже рекомендованими виробником засобами малювання не передбачено ефекту аліасингу, а отже усі не прямі фігури мали коструватий вигляд. Саме тому було обрано шлях розділення технологій для роботи з даним та для їх відображення.

3.2.1 Розробка модулів зв’язку та обчислення даних

Для розробки цієї частини було обрано мову програмування Python, через її надзвичайну простоту і велику кількість модулів. Для зручності подальшої розробки та внесення змін в проєкт було вирішено розділити усі частини роботи на окремі класи, що відповідатимуть за окремі складові усієї програми. Отже завдання було поділено на:

- модуль, що відповідає за зв’язок з вебінтерфейсом протоколом WebSocket;
- модуль, що відповідає за зв’язок з сервером протоколом MQTT;
- модуль, що зчитує данні GNSS з приймача і приводить їх до необхідного серверу вигляду;
- модуль, що відповідає за роботу мобільного зв’язку.

Усі описані модулі імпортуються в основний виконавчий файл (додаток А). Спочатку програма перевіяє чи правильно налаштований GSM модуль і якщо ні – змінює його налаштування на необхідні і перезавантажує модуль зв’язку. Після перевірки відбувається ініціалізація усіх модулів описаних вище. Виконується запуск процесу опитування GNSS приймача і після кожного отримання оновлених даних, швидкість надсилається модулем WS на інтерфейс. Запускається процес відправки даних MQTT за встановленим інтервалом, наприклад, в даному випадку інтервал 1 секунда, але його можна змінити в будь-яке інше значення. Після кожної відповіді по MQTT протоколу, дані що надіслані з серверу надсилаються по WS на інтерфейс.

Весь цей застосунок запускається системним сервісом deamon під час запуску системи, створення якого описувалось при налаштуванні ОС. В конфігурації цього сервісу встановлено, щоб він перезапускався у випадку виникнення помилки.

3.2.2 Розробка інтерфейсу апаратного модулю

Як було вже раніше зазначено, інтерфейс апаратного комплексу було вирішено робити у вигляді вебзастосунку. Для цього попередньо, під час підготовки ОС, було налаштовано автоматичний запуск браузеру Chromium в режимі кіоск, коли браузер відображається на весь доступний екран і його неможливо згорнути чи закрити. А також було встановлено вебсервер nginx, що буде запускати файли вебзастосунку в локальній мережі.

Для виконання вебзастосунку було обрано фреймворк Angular. Angular — це платформа та фреймворк для створення односторінкових клієнтських програм за допомогою HTML і TypeScript. Angular написаний на TypeScript. Він реалізує основні та додаткові функції як набір бібліотек TypeScript, які використовуються в проєкті.

Знаючи, що ми маємо дисплей розподільною здатністю 480×480 пікселів можемо знехтувати адаптивністю готового компоненту і розробляти

дизайн відповідно до цього розміру. Готовий інтерфейс має в основному графічне відображення а також числове відображення швидкості по центру інтерфейсу в контрастуючих кольорах (рис. 3.7).

Рисунок 3.7 – Приклад інтерфейсу апаратного модуля.

Для відображення графіки використовуються дані, що надходять по протоколу WebSocket від модуля обчислення в реальному часі (додаток Б).

Завдяки використанню саме вебзастосунку як засобу відображення інтерфейсу, маємо дуже гнучку систему. Графіка даних відокремлена від їх обробки і може бути видозмінена в будь-який момент іншими фахівцями, адже мова програмування TypeScript та мова розмітки HTML надзвичайно популярні і не складні в освоєнні. Також завдяки саме цим технологіям не складно реалізовувати плавні анімації, що могли б викликати трудності за реалізації іншими засобами.

3.3 Розробка вебзастосунку диспетчеризації

Для того щоб спостерігати за візуалізованими даними, які передають серверу апаратні модулі, необхідно створити вебзастосунок що надав би таку можливість. Реалізацію було вирішено робити тією ж технологією що і інтерфейс апаратної частини – фреймворком Angular. Під розробки також

було підключено бібліотеку готових модулів інтерфейсу Angular Material, що дозволять пришвидшити та полегшити процес реалізації.

3.3.1 Розробка панелі контролю усіх активних апаратних модулів

Для відображення геопозиційних даних було вирішено використовувати інтерактивну мапу світу. Для реалізації мапи було орано бібліотеку з відкритим сирцевим кодом leaflet, а за провайдерів картографічних даних – OpenStreetMap та MapBox. Тож використовуючи компоненти інтерфейсу та leaflet було створено сторінку (рис. 3.8), що відображає позначкою на карті апаратний комплекс, за даними що він надсилає.

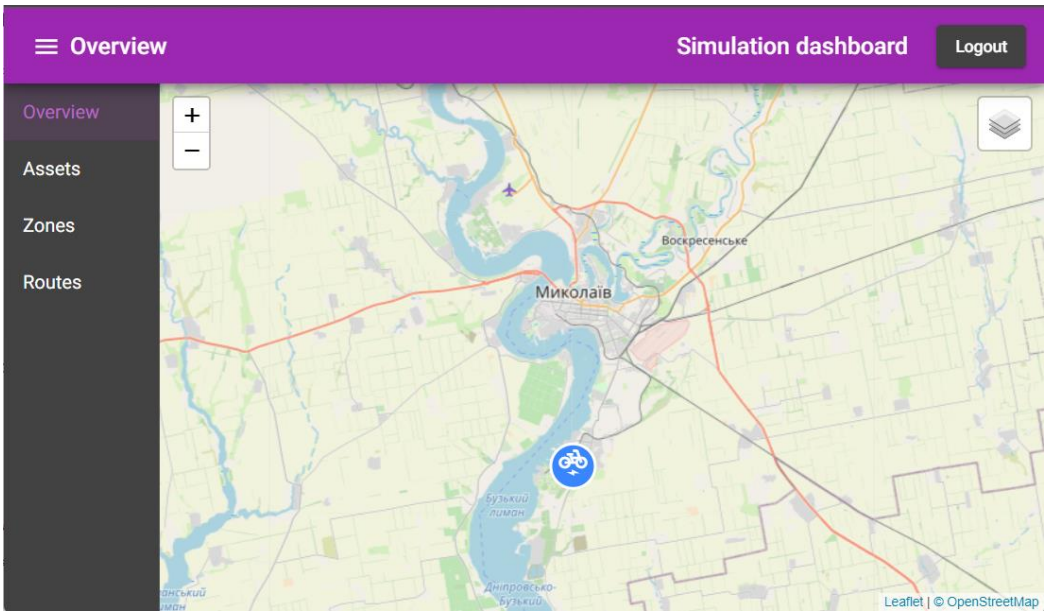


Рисунок 3.8 – Приклад візуалізації географічних даних на мапі.

Дані, що вебзастосунок використовує для відображення, надходять до сервісу по протоколу WebSocket. Сервіс одразу після отримання нових даних оновлює вже існуючу мітку і якщо позиція змінилась – то відбувається плавна анімація пересування мітки від минулого положення в актуальне. Сервіс може отримувати дані не лише географічні або дані про швидкість, а можуть бути й інші, адже платформа не обмежена лише цим набором. Для відображення додаткової інформації було використано модуль Popup з тієї ж бібліотеки, що використана для реалізації інтерфейсу мапи (рис. 3.9).

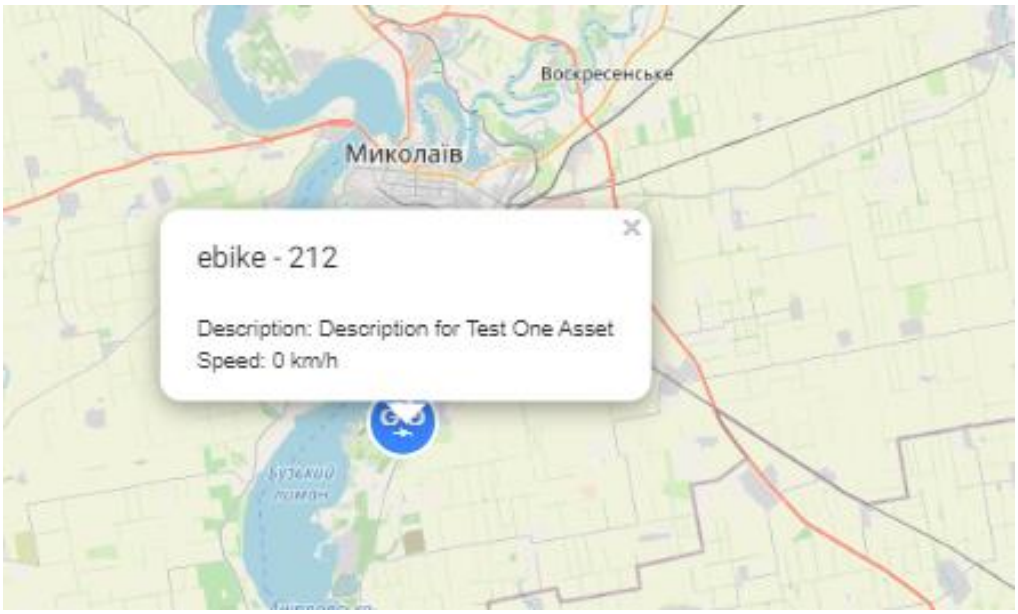


Рисунок 3.9 – Візуалізація суміжних даних апаратного комплексу.

Отже, було розроблено сторінку (додаток В), що надає можливості спостереження за переміщенням транспортного засобу на який встановлено апаратний комплекс. Окрім переміщення міток, що відображають місцеположення, також було розроблено механізм перегляду усіх даних, що надходять.

3.3.2 Розробка сторінки створення заборонених зон

Для встановлення двостороннього зв’язку між диспетчером, що користується цією панеллю керування та користувачем апаратного модуля, було вирішено створити сторінку керування зонами, потрапляючи в які, користувач апаратного комплексу отримував би сповіщення в його інтерфейсі.

Зовнішній вигляд сторінки має схожий вигляд до попередньо реалізованої, адже за основу було взято інтерфейс мапи. Дана сторінка надає користувачу можливість творити зони двох типів: радіус і полігон (рис. 3.10). Ці зони візуально відображаються на мапі і надають диспетчеру можливість створити керовані зони впливу на апаратний модуль.

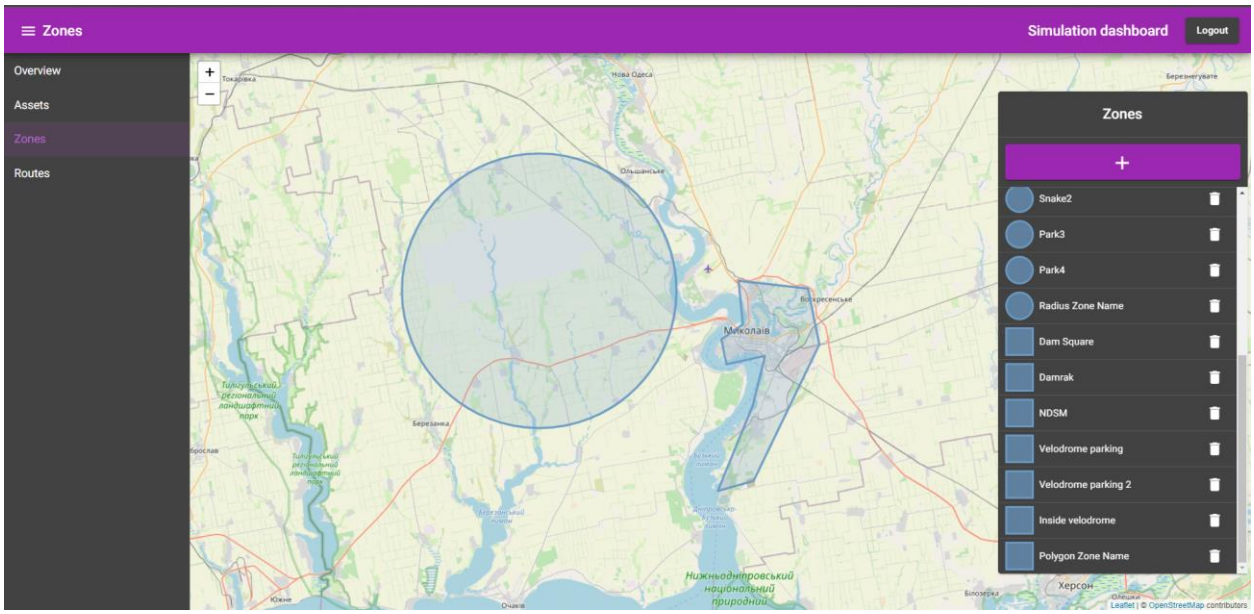


Рисунок 3.10 – Приклад сторінки керування зон.

Так само, як і для міток місцеположення, для зон реалізовано Popup віконце (рис. 3.11), що відображає інформацію про обрану зону обмеження.

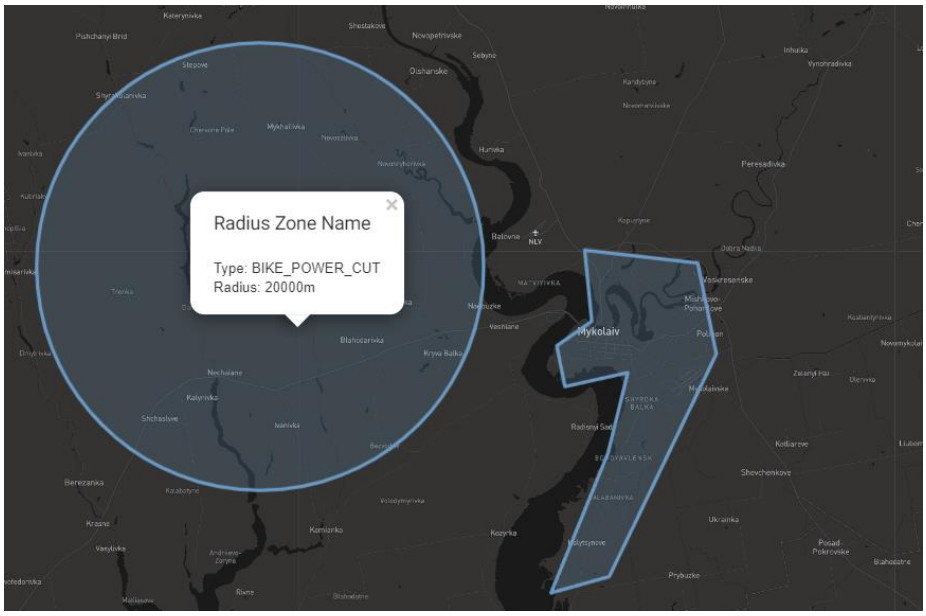


Рисунок 3.11 – Приклад інформаційного віконця зони.

Висновки до розділу 3

В цьому розділі було описано процес розробки апаратно-програмного комплексу відстеження швидкості та положення механічного транспортного засобу на базі Raspberry Pi Zero. Було зібрано апаратну частину, яка складається з: обчислювального одноплатного комп’ютера Raspberry Pi Zero,

модуля зв’язку та дисплею. Описано повний процес та підключення усіх типів з’єднання. Після збірки було виконано і описано повний процес встановлення і налаштування ОС Raspbian. Також описано процес підготування повністю підготовленої ОС до встановлення, шляхом її копіювання і стиснення в образ. В процесі налаштування системи було сконфігуровано автоматичний запуск сервісу що виконує обробку даних, браузеру, шо слугуватиме інтерфейсом, та вебсерверу, що буде запускати вебзастосунок інтерфейсу.

Наступними були описані внутрішні процеси апаратного модуля, а саме: розробка інтерфейсу та обробки даних. Модуль обробки даних було написано мовою програмування Python і розділено на окремі класи, що відповідають за вузьке призначення: MQTT, WS, GNSS, GSM/LTE. Усі ці окремі модулі були імпортовані в один виконавчий файл, де запускались усі процеси обміну даними. Для зв’язку обчислювальної ланки і інтерфейсу було використано протокол WS. Інтерфейс було виконано у вигляді вебзастосунку, обґрунтовуючи це тим, що це значно спростить процес розробки та підтримки графічної складової комплексу. Отже розроблено інтерфейс, використовуючи фреймворк Angular, що відображає користувачу поточну швидкість і, у разі необхідності, сповіщення від системи диспетчеризації.

Останньою розробленою складовою є панель управління усіма підключеними апаратними модулями. Ця панель виконана також на фреймворці Angular із застосування бібліотек leaflet для мапи і Angular Material для модулів інтерфейсу. Було реалізовано дві сторінки: сторінку з оглядом на мапі місцеположення апаратного комплексу, за переданими ним даними, та сторінку створення зон обмежень.

РОЗДІЛ 4
ЕКСПЕРИМЕНТАЛЬНІ ДОСТЛІЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОЗРОБКИ АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ

В даному розділі буде розглянуто результати розробки апаратно-програмного комплексу, та детальніше описано деякі аспекти його використання.

4.1 Огляд процесу створення зон обмежень

Як вже було описано в третьому розділі, всього було розроблено два типи зон: радіус і полігон, що мають повністю покрити усі види задач що можуть постати перед користувачами. Інтерфейс сторінки створення зон передбачає панель, що містить кнопку створення нових зон та список усіх вже створених (рис. 4.1).

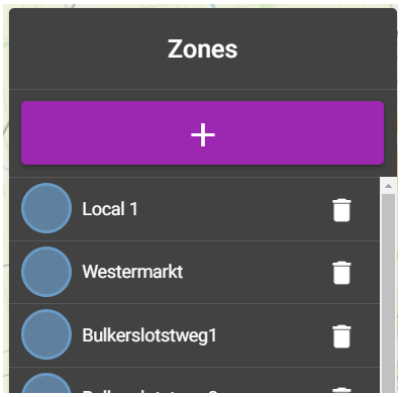


Рисунок 4.1 – Панель відображення зон списком.

Коли користувач хоче створити нову зону, він має натиснути на велику фіолетову кнопку з позначкою плюса і йому відобразиться нове вікно з вибором типу нової зони та можливістю відхилити операцію (рис. 4.2).

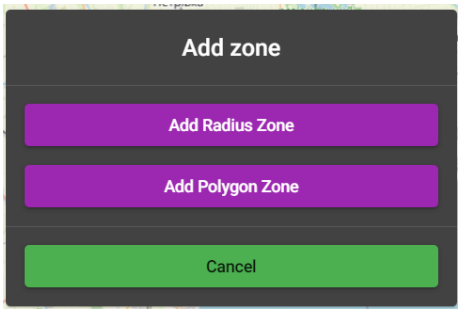


Рисунок 4.2 – Панель вибору типу створюваної зони.

Спершу розглянемо створення зони типу радіус. Після того як ми обрали тип зони радіус, ми побачимо що попереднє віконце змінилось (рис. 4.3) і нам тепер доступні три поля введення: назва зони, тип зони (різні типи для різного транспорту і різних умов) та поле для зміни радіусу кола в метрах. Також нам доступні дві кнопки: для того щоб зберегти зміни, та для того щоб відмінити будь-які зміни. Також нам доступна можливість змінити центр даного кола на мапі, шляхом перетягування його по ній. Колір рамок і колір заповнення кола змінюється відповідно до вибраного з випадаючого списку типу зони.

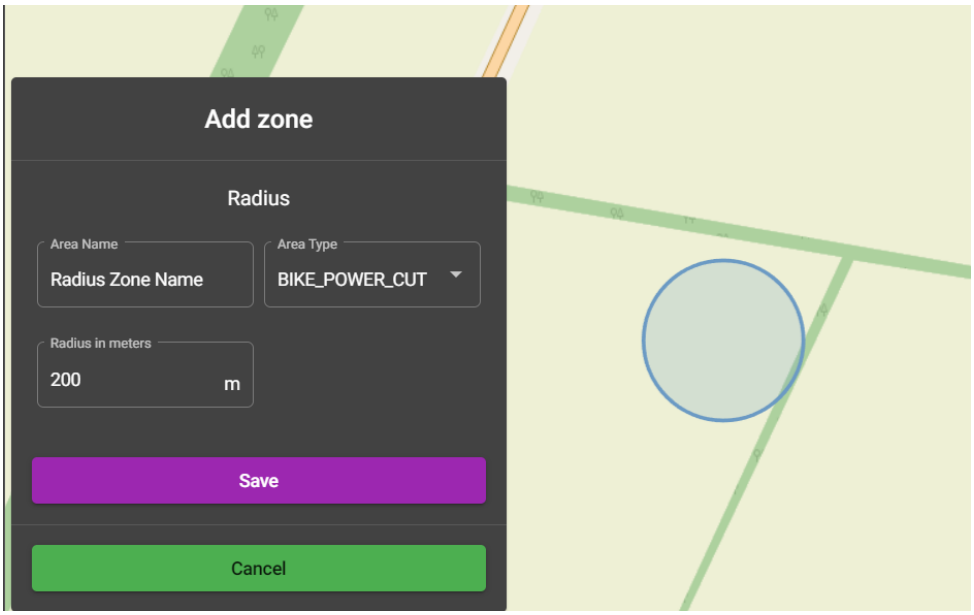


Рисунок 4.3 – Панель створення зони типу радіус.

Наступним типом зони до розгляду є полігон. В панелі налаштувань цього типу є вже два поля введення: текстове поле назви зони та поле з випадаючим списком типу зони до вибору, що так само як і у типу радіус змінює колір зони для візуальної відмінності. Так само як і у радіуса панель має кнопку збереження змін та кнопку відміни операції. Процес створення зони відбувається шляхом послідовного додавання точок на карті, що виконується простим кліком по карті. В наслідок додавання точок утворюється замкнений полігон довільної форми що створює необхідну зону.

Важливо якомога сильніше збільшити масштаб мапи для отримання точного результату.

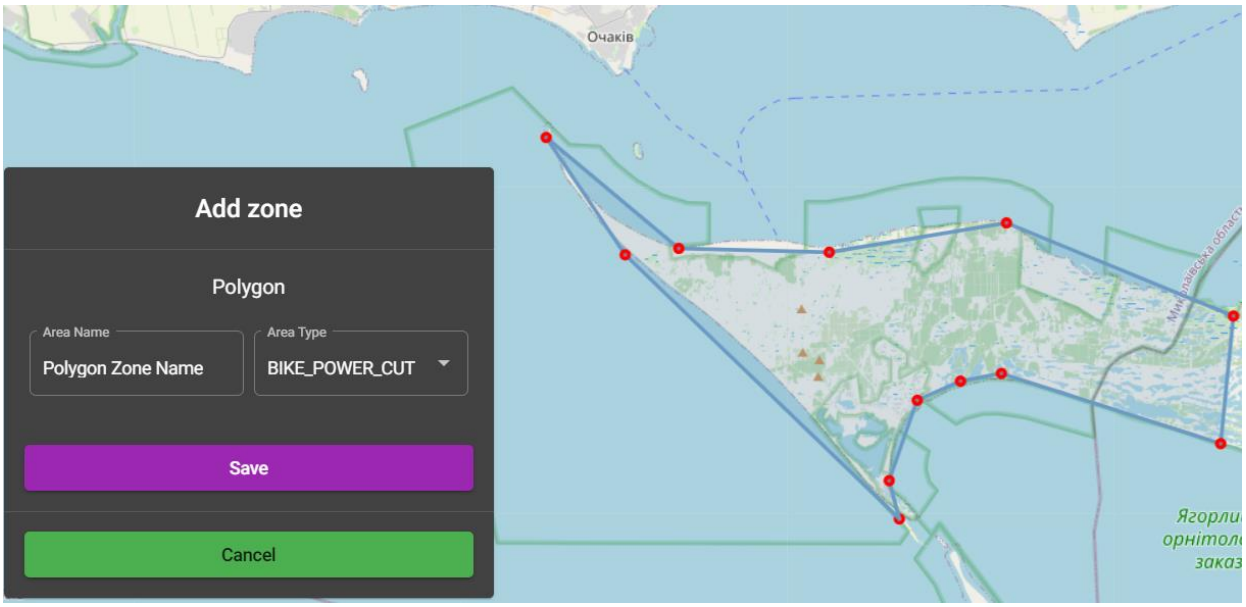


Рисунок 4.4 – Панель створення зони типу полігон.

Після того як зони було створено, вони додаються до списку описаного на початку розділу. Зони з цього списку можна видалити за допомогою кнопки в кінці рядку з зоною, що має піктограму смітника. Також, якщо натиснути на рядок з назвою зони, то карту буде переміщено так, щоб зона знаходилась по її центру і була припасована до її країв. Також якщо навести курсор миші на назву зони в списку – то відповідна зона буде виділена посеред інших.

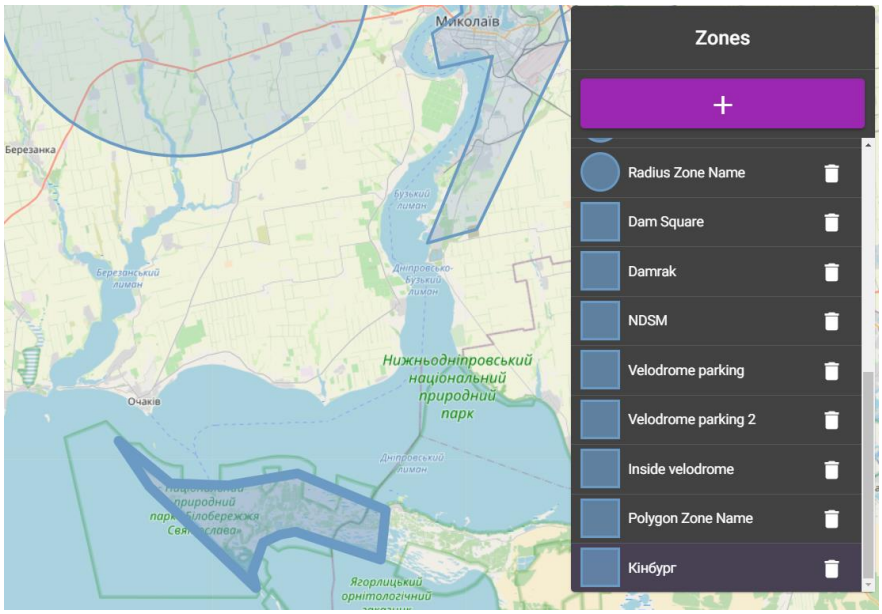


Рисунок 4.5 – Виділення кольором зони при наведенні на неї в списку.

У випадку, якщо апаратний модуль, попадає в зону обмеження, сервер надсилає йому назву типу обмеження. Інтерфейс апаратного комплексу маючи таку інформацію відповідно обробляє її і відображає користувачу піктограму обмеження руху (рис. 4.6).

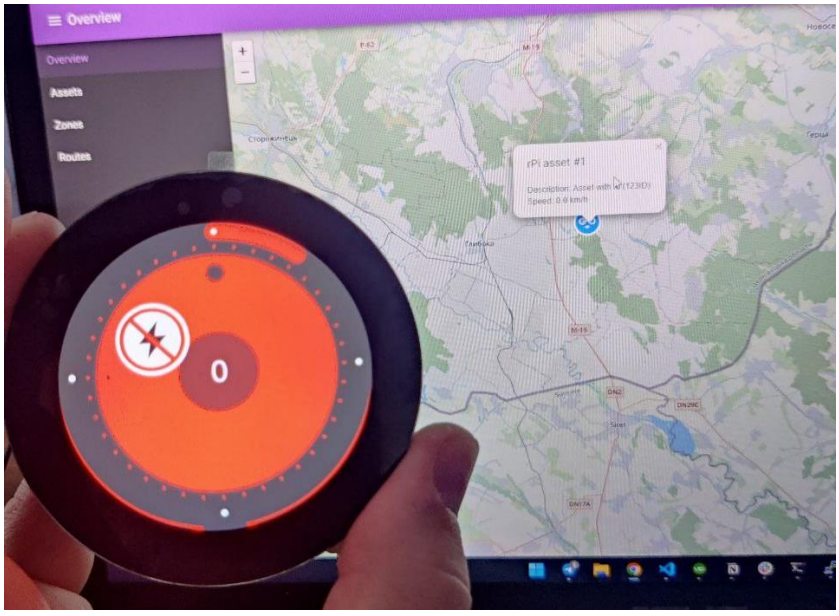


Рисунок 4.6 – Демонстрація зміни інтерфейсу апаратного модуля при потраплянні в зону обмеження.

4.2 Процес оновлення інтерфейсу та модулю обробки даних апаратного комплексу

Розробляючи будь-який комплекс, важливо постійно вносити зміни та правки, відповідно до того як змінюються цілі та задачі. Виконавши один раз налаштування усього апаратного комплексу, він більше не буде потребувати змін, а от процеси які відбуваються всередині, такі як вигляд інтерфейсу чи окремі програмні модулі, потребують постійних змін. В процесі налаштування ОС, було закладено можливість оновлювати файли як інтерфейсу так і модулю обробки даних, додавши відповідні рядки до bash скрипту у файлі «*~/.profile*». В скрипті налаштована перевірка на наявність файлів «asset-hard.tar» та «py-app.tar» в директорії «*/boot*» під час кожного завантаження системи. У випадку якщо файл в цій директорії є – то він буде розпакований і файли замінять ті що були в необхідних директорія, а сервіси, що відповідають за виконання цих файлів буде перезапущено. Для того щоб почати процес оновлення, треба встановити SD картку пам’яті в комп’ютер та скопіювати файли оновлення в єдину доступну директорію «*/boot*». Після чого підключаємо картку пам’яті до апаратного модуля і процес оновлення буде виконано автоматично під час завантаження ОС.

Для того щоб сформувати файли оновлення, було створено скрипти на виконання їх збірки для кожної складової відповідно. У випадку з інтерфейсом, щоб його оновити, необхідно виконати збірку проєкту Angular а готові файли архівувати в файл під назвою «asset-hard.tar». Щоб створити файл оновлення програмного модуля обробки даних, необхідно виконати команду «*python3 createTar.py*» в директорії з сирцевим кодом.

Висновки до розділу 4

В даному розділі було розглянуто процес створення зон обмежень в вебзастосунку диспетчеризації. Було розглянуло обидва типи цих зон: радіус та полігон. Окремо для кожного з типів зон було розглянуто панелі їх налаштування та особливості їх створення. Також описано структуру та

процес використання панелі зі списком усіх створених зон. Розглянуто доступний його функціонал: підкреслення кольором обраної зони при наведенні, переміщення мапи до обраної зони при натисненні на рядок з її назвою та можливість видалити зони натиснувши на піктограму смітника і кінці кожного рядку цього списку. Було розглянуто можливість зміни типу обмеження зони та підкреслено зміну кольору відповідно. На кінець, було розглянуто реакцію інтерфейсу апаратного модулю в момент потрапляння в одну з таких зон.

Також було розглянуто налаштований в ОС механізм оновлення складових апаратного комплексу шляхом додавання архівів з оновленням в спеціальну директорію. ОС під час кожного запуску перевіряє наявність цих файлів і у випадку успіху виконує їх заміну, перезавантажуючи усі відповідні сервіси, для вступу змін в силу.

ВИСНОВКИ

Метою даної роботи було розробити апаратно-програмний комплекс контролю механічного транспортного засобу на базі Raspberry Pi Zero. Для досягнення мети і виконання поставлених задач було виконано обширний обсяг роботи з аналізу , дослідження та розробки.

В ході виконання роботи було виконано огляд існуючих рішень для вирішення поставленої мети, проаналізувавши які, було виявлено, що жодне з існуючих рішень не виконує поставлені задачі комплексно, а лише частково. А отже результатом аналізу було остаточно підтвердження актуальності роботи. Для досягнення мети необхідно було обрати компоненти апаратного комплексу. Для задоволення вимог комплексу необхідні три модулі: обчислення, зв’язку та дисплей. З поміж конкуруючих моделей було обрано одноплатний комп’ютер Raspberry Pi Zero як основний модуль обчислення. Обрано його було за надзвичайну популярність, завдяки якій доступна велика кількість сумісних модулів розширення можливостей. На роль модуля зв’язку було обрано плату Waveshare SIM7600G-H HAT, за те що маючи приймачі GNSS та GSM сигналів, вона також є суміжною за кріпленням до обраного комп’ютера та не блокує 40-ка контактний роз’єм GPIO. Дисплей було обрано HyperPixel 2.1, що є компактним рішенням з відповідним до обраних комплектуючих кріпленням і конкуруючими показниками якості зображення.

Далі були розглянуті технології необхідні для виконання поставлених завдань. Протокол MQTT було обрано, як той, що ідеально підходить для IoT пристроїв з обмеженою доступністю до мережі. Розглянуто схему його роботи та взаємодії між відправником, отримувачем та брокером. Розглянуто важливість ролі брокера в даному протоколі. Також досліджено три рівні забезпечення доставки повідомлення, або ж QoS. Для встановлення зв’язку в системою диспетчеризації було розглянуло протокол WebSocket, що є повнодуплексним TCP протоколом передачі даних. Його найбільша перевага

для виконання заданих задач полягає в тому, що після встановлення з’єднання, воно не закривається після однієї передачі даних, а залишається відкритим, а отже забезпечує обмін даних з найменшими затримками.

В ході виконання роботи було зібрано апаратний комплекс, що складається з обраних раніше модулів, а процес збірки був ретельно описаний. Було детально описано процес встановлення, налаштування та підготовки до копіювання ОС. Завдяки такій підготовці ОС було реалізовано автоматичний запуск усіх процесів разом з системою, а також механізм оновлення. Було описано структуру програмної частини апаратного комплексу: модуля обробки даних та інтерфейсу. Також було виконано розробку вебзастосунку диспетчеризації. Виконуючи одну з поставлених задач було розроблено сторінку огляду даних, наданих з апаратного комплексу, на мапі в наближеному до реального часі. Вирішуючи ще одну задачу було реалізовано механізм зон обмежень, завдяки якому в панелі адміністрування можна встановлювати на мапі зони, потрапляючи в які апаратний модуль отримує і відображає в інтерфейсі відповідні сповіщення.

Загалом, було виконано усі поставлені задачі, а отже і досягнуто мети, поставленої на початку виконання роботи. Оскільки в результаті було отримано інтегроване IoT рішення для механічних транспортних засобів з можливістю диспетчеризації, то для подальшого розвитку проєкту є безліч шляхів, зокрема: розробити стандартизоване або унікальне кріплення для кожного виду транспорту, додати джерела інформації для їх передачі на панель управління, наприклад стан двигуна чи температуру, для панелі управління можна додати методи управління апаратним модулем тощо.

Результатом цієї роботи є комплекс готовий до інтегрування в системи різних комерційних проєктів. Наприклад, це може бути оренда електровелосипедів, або ж система моніторингу автомобільного транспорту з візуальною системою оповіщення, система відображення руху маршрутного громадського транспорту тощо.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. С. С. Рибченко, Я. М. Крайник, «Застосування MQTT в IoT системах,» Інформаційні технології та інженерія : тези доп. Всеукр. наук.-практ. конф. Миколаїв, 07–10 лют. 2023 р. Миколаїв : Чорном. нац. ун-т ім. Петра Могили, 2023. Миколаїв, 2023, с. 79-81.

2. Y. Krainyk, “Information technology of university class internet-of-things-module,” in CEUR Workshop Proceedings, 2019, vol. 2516, с. 58–68.

3. Graceful degradation of websocket connections to nonpersistent HTTP-based communications : Сполучені Штати Америки (США) : US9143550B2. Заявл. 01.12.2012 ; опубл. 22.09.2015.

4. Communication method and apparatus : Сполучені Штати Америки (США) : US20230017306A1. Заявл. 16.09.2022 ; опубл. 19.01.2023.

5. Information technology – Message Queuing Telemetry Transport (MQTT) (ISO/IEC 20922:2016). ISO. URL: https://www.iso.org/standard/69466.html (Last accessed: 1.02.2023).

6. Raspberry Pi wins MacRobert Award from the Royal Academy of Engineering: University of Cambridge news. URL: https://www.staff.admin.cam.ac.uk/awards/raspberry-pi-wins-macrobert-award-from-the-royal-academy-of-engineering/ (Last accessed: 1.02.2023).

7. Kondratenko, Y., Kondratenko, G., Sidenko, I.: Multi-criteria selection of the wireless communication technology for specialized IoT network. In: 14th Interna-tional Conference on ICT in Education, Research and Industrial Applications. Inte-gration, Harmonization and Knowledge Transfer, Workshops, ICTERI, Vol. 2104, pp. 501-516. Kyiv, Ukraine (2018).

8. HyperPixel 2.1 Round - Hi-Res Display for Raspberry Pi. Pimoroni: Raspberry Pi. URL: https://shop.pimoroni.com/products/hyperpixel-round (Last accessed: 01.02.2023).

9. Raspberry Pi Documentation - Configuration. Raspberry Pi. URL: https://www.raspberrypi.com/documentation/computers/configuration.html (Last accessed: 01.02.2023).

10. SIM7600G-H 4G HAT (B) - Waveshare Wiki. Waveshare Electronics. URL: https://www.waveshare.com/wiki/SIM7600G-H_4G_HAT_(B) (Last accessed: 01.02.2023).

11. Raspberry Pi Kiosk using Chromium. Pi My Life Up. URL: https://pimylifeup.com/raspberry-pi-kiosk/ (Last accessed: 01.02.2023).

12. Documentation - Leaflet - a JavaScript library for interactive maps. URL: https://leafletjs.com/reference.html#popupevent (Last accessed: 01.02.2023).

13. National Marine Electronics Association. Pratiques et Techniques de la Plaisance. URL: https://www.plaisance-pratique.com/IMG/pdf/NMEA0183-2.pdf (Last accessed: 01.02.2023).

14. Що таке MQTT і для чого потрібний протокол. Highload.today. URL: https://highload.today/uk/shho-take-mqtt-i-dlya-chogo-vin-potribnij/ (дата звернення: 01.02.2023).

15. FEATURES & BENEFITS: Angular official website. URL: https://angular.io/features (Last accessed: 01.02.2023).

16. Sonnenberg G. J. Radar and Electronic Navigation. Elsevier Science & Technology Books, 2013.

17. Brynjolfsson E., Mcafee A. Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies. Norton & Company, Incorporated, W. W., 2014. 304 p.

18. Kerrisk M. The Linux Programming Interface: A Linux and Unix System Programming Handbook. San Francisco : No Starch Press, 2010. 1506 p.

19. Angular. API reference for Angular Material toolbar. Angular Material. URL: https://material.angular.io/components/toolbar/overview (date of access: 01.02.2023).

20. Web sockets vs Ajax . EDUCBA. URL: https://www.educba.com/web-sockets-vs-ajax/ (date of access: 01.02.2023).

21. ELEMNT ROAM v1 (2019) and BOLT v2 (2021) Information and Setup. Wahoo. URL: https://support.wahoofitness.com/hc/en-us/articles/115000485450-ELEMNT-ROAM-BOLT-Product-Instructions (date of access: 01.02.2023).

22. Omega2 – Onion. Onion – Compute Platform for IoT. URL: https://onion.io/omega2/ (date of access: 01.02.2023).

23. Banana Pi BPI-M2 ZERO - Banana Pi Wiki. Banana Pi Wiki. URL: https://wiki.banana-pi.org/Banana_Pi_BPI-M2_ZERO (date of access: 01.02.2023).

24. A brief overview of the websocket protocol – noio_ws 0.0.5 documentation. noio_ws. URL: https://noio-ws.readthedocs.io/en/latest/overview_of_websockets.html (date of access: 01.02.2023).

25. BerryGPS-GSM Overview - ozzmaker.com. ozzmaker.com. URL: https://ozzmaker.com/berrygps-gsm-overview/ (date of access: 01.02.2023).

26. GSM/GPRS/GNSS HAT - Waveshare Wiki. Waveshare Electronics. URL: https://www.waveshare.com/wiki/GSM/GPRS/GNSS_HAT (date of access: 01.02.2023).

27. MQTT (Quality of Service) QoS levels explained | Cedalo. Cedalo. URL: https://cedalo.com/blog/understanding-mqtt-qos/ (date of access: 01.02.2023).

28. Raspberry Pi WiFi Setup: Here are 5 Methods for Raspbian. Raspberry Expert. URL: https://raspberryexpert.com/raspberry-pi-wifi-setup/ (date of access: 01.02.2023).

29. What are Global Navigation Satellite Systems?. NovAtel. URL: https://novatel.com/tech-talk/an-introduction-to-gnss/what-are-global-navigation-satellite-systems-gnss (date of access: 01.02.2023).

30. What is GNSS?. EU Agency for the Space Programme. URL: https://www.euspa.europa.eu/european-space/eu-space-programme/what-gnss (date of access: 01.02.2023).

ДОДАТОК А

Код програмних модулів апаратної частини

classes/gps.py

```
from datetime import datetime
import serial
import time


class GPS:
    lat = 0
    long = 0
    alt = 0
    speed = 0
    timestamp = 0

    def __init__(self):
        self.ser = serial.Serial('/dev/ttyUSB2', 115200)

    def send_AT(self, command, back, timeout):
        rec_buff = ''
        self.ser.write((command+'\r\n').encode())
        time.sleep(timeout)
        if self.ser.inWaiting():
            rec_buff = self.ser.read(self.ser.inWaiting())
        if rec_buff != '':
            if back in rec_buff.decode():
                return str(rec_buff.decode())
        else:
            print('GPS is not ready')
            return 0

    def startGpsGetting(self, func, *args):
        self.ser.flushInput()
        self.send_AT('AT+CGPS=1,1', 'OK', 1)
        while True:
            res = self.send_AT('AT+CGPSINFO', '+CGPSINFO: ', 0.1)
            if 0 != res and ',,,,' not in res:
                self._setData(res[25:-8])
                func(*args)

    def _setData(self, input):
        splited = input.split(',')

        latDD = int(float(splited[0])/100)
        latMM = float(splited[0]) - latDD * 100
        self.lat = (latDD + latMM/60) * (1 if splited[1] == 'N' else -1)

        longDD = int(float(splited[2])/100)
        longMM = float(splited[2]) - longDD * 100
        self.long = (longDD + longMM/60) * (1 if splited[3] == 'E' else -1)

        self.timestamp = str(int(round(datetime.strptime(
            splited[4] + splited[5][:-2], '%d%m%y%H%M%S').timestamp())))

        self.alt = splited[6]

        self.speed = str(float(splited[7]) * 1.852)
```

classes/mqtt.py

```
import paho.mqtt.client as mqtt
import json
```

```
class MQTT_Client:
    assistLevel = 0
    restricted = False
    restrictZones = []

    def __init__(self, host, port, username, password, assetID, ws_callback):
        self.assetID = assetID
        self.ws_callback = ws_callback

        self.client = mqtt.Client(
            "testclient", protocol=mqtt.MQTTv31, clean_session=True)
        self.client.on_message = self.on_message
        self.client.on_connect = self.on_connect

        self.client.tls_set()
        self.client.username_pw_set(username=username, password=password)
        self.client.connect(host, port, 60)

        topics = ["assets/" + self.assetID + "/restrict",
                  "assets/" + self.assetID + "/unrestrict"]

        for tpc in topics:
            self.client.subscribe(tpc, 0)
        self.client.loop_start()

    def on_message(self, client, userdata, msg):
        if "restrict" in msg.topic:
            payload = json.loads(msg.payload.decode('utf-8'))
            self.assistLevel = payload["level"]
            self.restrictZones = payload["zoneTypes"]
            self.restricted = "unrestrict" not in msg.topic

            self.ws_callback()
        return True

    def on_connect(self, client, userdata, flags, rc):
        print("Connected with result code", rc)

    def sendAssetData(self, assetData):
        stringifiedData = json.dumps(assetData, separators=(',', ':'))
        self.client.publish("bikemovedevent-topic", stringifiedData)
```

classes/websocket.py

```
from websocket_server import WebsocketServer
import json


class WS:

    def __init__(self, host='127.0.0.1', port=3333):
        self.server = WebsocketServer(host=host, port=port)
        self.server.set_fn_new_client(self._new_client)
        self.server.run_forever(True)

    def _new_client(self, client, server):
        res = {"sts": "OK"}
        server.send_message(client, self.stringify(res))

    def sendSpeed(self, speed):
        type = "speed"
```

```
        res = {
            "type": type,
            "speed": speed
        }
        self.server.send_message_to_all(self.stringify(res))

    def sendMqttData(self, assistLevel, zones, restricted):
        type = "mqtt"
        res = {
            "type": type,
            "assistLevel": assistLevel,
            "zones": zones,
            "restricted": restricted
        }
        self.server.send_message_to_all(self.stringify(res))

    def stringify(self, msg):
        return json.dumps(msg, separators=(',', ':'))
```

app.py

```
from ast import Pass
from classes.gps import GPS
from classes.websocket import WS
from classes.mqtt import MQTT_Client
from classes.checkHat import Check_HAT

import threading
from time import sleep


"""
Asset GPS LTE Device

v0.0.1

"""


class AssetDevice:
    gps = None
    ws = None
    mqtt = None

    assetID = "123ID"

    secondsPerMQTTSend = 1  # seconds

    def __init__(self):
        # check if dial-up mode is setted on HAT, if now will turn and reboot
        Check_HAT()

        #If dial-up (LTE Internet) is already setted -> then run app

        self.gps = GPS()
        self.ws = WS()
        self.mqtt = MQTT_Client(
            host="friendly-florist.cloudmqtt.com",
            port=8883,
            username="rqviljoj",
            password="rhqNjh9Qsw_X",
```

```
            assetID=self.assetID,
            ws_callback=self._sendMqttDataToDisplay
        )

        self._startGpsThread()
        self._startMQTTThread()

    def _sendGpsSpeedToDisplay(self):
        self.ws.sendSpeed(int(float(self.gps.speed)))

    def _sendMqttDataToDisplay(self):
        self.ws.sendMqttData(
            self.mqtt.assistLevel,
            self.mqtt.restrictZones,
            self.mqtt.restricted
        )

    def _startGpsThread(self):
        thread = threading.Thread(
            target=self.gps.startGpsGetting,
            args=(self._sendGpsSpeedToDisplay,)
        )
        thread.daemon = True
        thread.start()

    def _startMQTTThread(self):
        thread = threading.Thread(
            target=self._startMqttSending
        )
        thread.daemon = True
        thread.start()

    def _startMqttSending(self):
        while True:
            if self.gps.timestamp != 0:
                assetData = {
                    "assetId": self.assetID,
                    "name": 'rPi asset #1',
                    "description": 'Asset with id (' + self.assetID + ')',
                    "speed": self.gps.speed,
                    "lat": self.gps.lat,
                    "lng": self.gps.long,
                    "timestamp": self.gps.timestamp,
                    "assetType": "EBIKE"
                }
                self.mqtt.sendAssetData(assetData)
            sleep(self.secondsPerMQTTSend)


asset = AssetDevice()


while True:
    sleep(10)
```

ДОДАТОК Б

Код інтерфейсу апаратного комплексу

home.page.html

```
<ion-content [fullscreen]="true">
  <div class="wrapper">
    <asset-device-circle-speedometer
      [top]="3"
      [left]="-10"
      [speed]="speed"
      [restricted]="restricted"
      [assistLevel]="assistLevel"
      [zoneTypes]="zones"
    >
    </asset-device-circle-speedometer>
  </div>
</ion-content>
```

home.page.ts

```
import { Component } from '@angular/core';
import { webSocket } from 'rxjs/webSocket';
import { retry } from 'rxjs/operators';
import { environment } from '../../environments/environment';

@Component({
  selector: 'asset-device-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
})
export class HomePage {
  speed = 0;
  assistLevel = 0;
  zones = [];
  restricted = false;

  constructor() {
    webSocket(environment.ws)
      .pipe(retry({ delay: 3000 }))
      .subscribe((msg: any) => this.receiveWS(msg));
  }

  ionViewDidEnter() {}

  receiveWS(msg) {
    switch (msg.type) {
      case 'speed':
        this.speed = msg.speed;
        break;

      case 'mqtt':
        this.zones = msg.zones;
        this.restricted = msg.restricted;
        this.assistLevel = msg.assistLevel;
        break;

      default:
        break;
    }
  }
}
```

circle-speedometer.component.html

```
<div class="circle-speedometer" [style.top.px]="top" [style.left.px]="left">
  <div class="circle-speedometer__data"
       [style.background]="restricted ? restrictedColor : (restricted === false ?
unrestrictedColor : backgroundColor)">
    <svg class="icon" viewBox="0 0 40 40">

      <defs>
        <linearGradient id="comet" x1="0%" y1="0%" x2="0%" y2="100%">
          <stop offset="0%" stop-opacity="1"
                [attr.stop-color]="restricted === null ? unrestrictedColor :
backgroundColor"/>
          <stop offset="75%" stop-opacity="0.75"
                [attr.stop-color]="restricted === null ? unrestrictedColor :
backgroundColor"/>
          <stop offset="100%" stop-opacity="0"
                [attr.stop-color]="restricted === null ? unrestrictedColor :
backgroundColor"/>
        </linearGradient>
      </defs>

      <g class="comet">
        <title>Comet</title>
        <circle cx="20" cy="3" r="1.5"
                [style.fill]="restricted === null ? unrestrictedColor :
backgroundColor"></circle>
        <polygon #pol points="" fill="url(#comet)">
          <animate #animatePol attributeName="points" dur="200ms"></animate>
        </polygon>
      </g>

    </svg>
    <div class="circle-speedometer__data__restrictions">
      <div *ngFor="let type of zoneTypes | slice:0:4"
           class="circle-speedometer__data__restriction">
        <ng-container [ngSwitch]="type">
          <asset-device-badge-no-access *ngSwitchCase="'NO_ACCESS'"></asset-device-
badge-no-access>
          <asset-device-badge-no-heavy-load *ngSwitchCase="'NO_HEAVY_LOAD'"></asset-
device-badge-no-heavy-load>
          <asset-device-badge-no-access-time *ngSwitchCase="'NO_ACCESS_9_17'"></asset-
device-badge-no-access-time>
          <asset-device-badge-bike-power-cut *ngSwitchCase="'BIKE_POWER_CUT'"></asset-
device-badge-bike-power-cut>
          <ng-container *ngSwitchDefault>not impl yet</ng-container>
        </ng-container>
      </div>
    </div>
    <div class="circle-speedometer__data__speed">
      {{ speed }}
    </div>
  </div>
  <div class="circle-speedometer__speed">
    <svg class="icon" viewBox="0 0 40 40">

      <circle class="circle-border"
              [style.stroke]="restricted ? restrictedColor : unrestrictedColor"
              r="19" cy="20" cx="20"/>
      <!--      stroke-dashoffset: range between  0 (100%) and 108.38 (0%), where 0%
is the perimeter of circle -->
      <circle class="circle-speed"
              [style.stroke]="restricted ? restrictedColor : unrestrictedColor"
              [attr.stroke-dashoffset]="calculateAssistPercentage"
              r="17.25" cy="20" cx="20"
              transform="rotate(-90,20,20)"/>

      <circle class="circle-round-point" r="0.4" cy="2.75" cx="20"></circle>
      <circle class="circle-round-point" r="0.4" cy="20" cx="37.25"></circle>
```

```
      <circle class="circle-round-point" r="0.4" cy="20" cx="2.75"></circle>
      <circle class="circle-round-point" r="0.4" cy="37.25" cx="20"></circle>

      <line *ngFor="let _ of lineCounter(40);let i = index"
            class="circle-dashes"
            [style.stroke]="restricted ? restrictedColor : unrestrictedColor"
            x1="20" y1="4.5" x2="20" y2="4.7"
            [attr.transform]="'rotate(' + (i * 360)/40 +' 20 20)'"></line>

    </svg>
  </div>
</div>
```

circle-speedometer.component.ts

```
import { Component, ElementRef, Input, OnChanges, SimpleChanges, ViewChild } from
"@angular/core";

@Component({
  selector: 'asset-device-circle-speedometer',
  templateUrl: './circle-speedometer.component.html',
  styleUrls: ['./circle-speedometer.component.scss'],
})
export class CircleSpeedometerComponent implements OnChanges {
  @Input() speed = 0;
  @Input() maxSpeed = 80;

  @Input() assistLevel = 0;
  @Input() maxAssistLevel = 10;

  @Input() zoneTypes: string[] = [];

  @Input() restricted: boolean = null;

  @Input() restrictedColor = '#ee2123';
  @Input() unrestrictedColor = '#99ca3b';
  @Input() backgroundColor = '#313232';

  @Input() top = 0;
  @Input() left = 0;

  @ViewChild('pol', {static: false}) pol: ElementRef;
  @ViewChild('animatePol', {static: false}) animatePol: ElementRef;

  private speedCircleRadius = 108.35;
  private speedCometMax = 37;

  constructor() {}

  get calculateAssistPercentage(): number {
    return (100 - (this.assistLevel * 100 / this.maxAssistLevel)) *
this.speedCircleRadius / 100;
  }

  calculateSpeedPercentage(value): number {
    // 3 in the end is a margin from the top
    return value / this.maxSpeed * (this.speedCometMax - 3) + 3;
  }

  ngOnChanges(changes: SimpleChanges) {
    if (!changes.speed?.isFirstChange() && changes.speed?.currentValue !==
changes.speed?.previousValue) {
      this.animatePol.nativeElement.setAttribute('from',
        '20.5,' + this.calculateSpeedPercentage(changes.speed.previousValue) +
        ',  19.5,' + this.calculateSpeedPercentage(changes.speed.previousValue) + ',
18.5, 3,  21.5, 3');

      this.animatePol.nativeElement.setAttribute('to',
        '20.5,' + this.calculateSpeedPercentage(changes.speed.currentValue) +
```

```
        ',  19.5,' + this.calculateSpeedPercentage(changes.speed.currentValue) + ',
18.5, 3,  21.5, 3');

      this.pol.nativeElement.setAttribute('points', '20.5,' +
this.calculateSpeedPercentage(changes.speed.currentValue) +
        ',  19.5,' + this.calculateSpeedPercentage(changes.speed.currentValue) + ',
18.5, 3,  21.5, 3');

      this.animatePol.nativeElement.beginElement();

    }
  }

  lineCounter(i: number) {
    return new Array(i);
  }

}
```

ДОДАТОК В

Код інтерфейсу диспетчеризації

overview-page.component.html

```
<div class="w-100 h-100" leaflet [leafletOptions]="setupOptions">
  <div *ngFor="let l of markersLayers" [leafletLayer]="l"></div>
  <div *ngFor="let l of zonesLayers" [leafletLayer]="l"></div>
</div>
```

overview-page.component.ts

```
import {
  AfterViewInit,
  Component,
  OnDestroy,
  OnInit,
  ViewChild,
} from '@angular/core';
import {
  latLng,
  Layer,
  MapOptions,
  tileLayer,
  circle,
  divIcon,
  marker,
  layerGroup,
  control,
  polygon,
} from 'leaflet';
import { RxStompService } from '@stomp/ng2-stompjs';
import { ZonesService } from '../../services/zones.service';
import { Subscription } from 'rxjs';
import { MapZoneTypes } from '../../../models/zone.type';
import { MapAssetTypes } from '../../../models/asset.type';
import { environment } from '../../../../environments/environment';
import { LeafletDirective } from '@asymmetrik/ngx-leaflet';
import layers = control.layers;
import { ZoneLayer } from '../../../models/zone-layer.model';

const MapBoxLayer = tileLayer(

'https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token={accessToken}',
  {
    maxZoom: 18,
    attribution:
      '<a href="https://www.mapbox.com/about/maps/">&copy; MapBox</a>',
    id: 'mapbox/dark-v10',
    accessToken: environment.mapboxToken,
  }
);

const CartoDBDarkLayer = tileLayer(
  'https://{s}.basemaps.cartocdn.com/rastertiles/dark_all/{z}/{x}/{y}.png',
  {
    maxZoom: 18,
    attribution: '<a href="http://cartodb.com/attributions">&copy; CartoDB</a>',
  }
);

@Component({
  selector: 'app-overview-page',
  templateUrl: './overview-page.component.html',
  styleUrls: ['./overview-page.component.scss'],
})
export class OverviewPageComponent implements OnInit, OnDestroy, AfterViewInit {
```

```
@ViewChild(LeafletDirective, { static: true }) leafletDirective:
  | LeafletDirective
  | undefined;

setupOptions: MapOptions = {
  layers: [MapBoxLayer],
  zoom: 14,
  center: latLng(52.367175, 4.889559),
};

setupToggle = true;

zonesLayers: ZoneLayer[] = [];
markersLayers: Layer[] = [];

zoneTypesObj: any = {};
assetTypesObj: any = {};

subscriptions: Subscription[] = [];

constructor(
  private wsService: RxStompService,
  private zonesService: ZonesService
) {
  for (const [key, value] of Object.entries(MapZoneTypes)) {
    this.zoneTypesObj[key] = value;
  }
  for (const [key, value] of Object.entries(MapAssetTypes)) {
    this.assetTypesObj[key] = value;
  }
}

ngOnInit(): void {
  this.setupMapZones();
  this.startMarkerSubscription();
}

ngAfterViewInit() {
  const interval = setInterval(() => {
    const map = this.leafletDirective?.getMap();
    if (map) {
      layers({
        'Dark  | MapBox': MapBoxLayer,
        'Light | OpenStreetMap': OpenStreetMapLayer,
        'Light | CartoDB': CartoDBLightLayer,
        'Dark  | CartoDB': CartoDBDarkLayer,
      }).addTo(map);
      clearInterval(interval);
    }
  }, 200);
}

setupMapZones() {
  this.subscriptions.push(
    this.zonesService.zones$.subscribe(([radiuses, polygons]) => {
      const zoneLayers = [];
      if (radiuses?.length > 0) {
        for (const radius of radiuses) {
          const rad = this.zonesLayers.find(
            (item) => item.options?.id === radius.id
          );
          if (rad) {
            zoneLayers.push(rad);
            continue;
          }

          const _circle = circle(
            latLng(radius.point.lat, radius.point.lng),
            radius.radiusMeters,
            // @ts-ignore
```

```
            { ...radius }
          )
            .bindPopup(() => {
              return `
                  <h3>${radius.name}</h3>
                  <p class="m-0">Type: ${radius.zoneType}</p>
                  <p class="m-0">Radius: ${radius.radiusMeters}m</p>
            `;
            })
            .setStyle({
              fillColor: this.zoneTypesObj[radius.zoneType],
              color: this.zoneTypesObj[radius.zoneType],
              fillOpacity: 0.2,
              weight: 3,
            });
          zoneLayers.push(_circle);
        }
      }

      if (polygons?.length > 0) {
        for (const polyg of polygons) {
          const pol = this.zonesLayers.find(
            (item) => item.options?.id === polyg.id
          );
          if (pol) {
            zoneLayers.push(pol);
            continue;
          }

          console.log(polyg);
          // @ts-ignore
          const _polygon = polygon(polyg.points, { ...polyg })
            .bindPopup(() => {
              return `
                  <h3>${polyg.name}</h3>
                  <p class="m-0">Type: ${polyg.zoneType}</p>
            `;
            })
            .setStyle({
              fillColor: this.zoneTypesObj[polyg.zoneType],
              color: this.zoneTypesObj[polyg.zoneType],
              fillOpacity: 0.2,
              weight: 3,
            });
          zoneLayers.push(_polygon);
        }
      }

      // @ts-ignore
      this.zonesLayers = zoneLayers;
    })
  );
}

startMarkerSubscription() {
  this.subscriptions.push(
    this.wsService.watch('/topic/messages').subscribe((data) => {
      // decode binary message body
      const body = JSON.parse(new TextDecoder().decode(data.binaryBody));
      this.updateMarkers(body);
    })
  );
}

updateMarkers(data: any) {
  const index = this.markersLayers.findIndex(
    // @ts-ignore
    (layer) => layer.options.assetId === data.assetId
  );
  if (index >= 0) {
```

```
      // @ts-ignore
      this.markersLayers[index].setLatLng(latLng(data.lat, data.lng));
      this.markersLayers[index].bindPopup(`
            <h3>${data.name}</h3>
            <p class="m-0">Description: ${data.description}</p>
            <p class="m-0">Speed: ${data.speed} km/h</p>
        `);
    } else {
      // @ts-ignore
      const style = `
        width: 40px;
        height: 40px;
        border-radius: 50%;
        background: ${this.assetTypesObj[data.assetType].color};
        border: solid 3px white;
        display: flex;
        justify-content: center;
        align-items: center;
        transform: translate(-43%, -43%)
      `;
      const icon = divIcon({
        html: `<div style="${style}">
                    <span asset-icon="${
                      this.assetTypesObj[data.assetType].icon
                    }"></span>
                </div>`,
      });
      const _marker = marker(latLng(data.lat, data.lng), {
        ...data,
        icon: icon,
      }).bindPopup(`
            <h3>${data.name}</h3>
            <p class="m-0">Description: ${data.description}</p>
            <p class="m-0">Speed: ${data.speed} km/h</p>
        `);
      this.markersLayers.push(_marker);
    }
  }

  ngOnDestroy() {
    for (const subscription of this.subscriptions) {
      subscription && subscription.unsubscribe();
    }
  }
}
```