МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет

імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри,
д-р техн. наук, проф.
_____ І. М. Журавська
« __ » _____ 2024 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Апаратно-програмний комплекс для моніторингу терористичної активності

Спеціальність 123 Комп'ютерна інженерія

123 – КБР.01 – 405.21910503

Студент _____ В. О. Шкромида
підпис
«__» _____ 202__ р.

Керівник ст. викладач _____ В. В. Старченко
підпис
«__» _____ 202__ р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Зав. кафедри _____ І. М. Журавська

«_____» _____ 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної бакалаврської роботи

Видано студенту групи 405 факультету комп'ютерних наук

Шкромиді Віталію Олексійовичу
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи
Апаратно-програмний комплекс для моніторингу терористичної активності

Затверджена наказом по ЧНУ ім. Петра Могили від 30.01.2024 № 17.

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Очікуваним результатом роботи є апаратне та програмне забезпечення комплекса для моніторингу терористичної активності. Вхідними даними роботи є специфікація вимог, що описує характеристики зазначеного апаратного та програмного забезпечення._____

4. Перелік питань, що підлягають розробці
1) аналітичний огляд систем моніторінгу терористичної активності;_____
2) аналіз переваг та недоліків існуючих систем моніторінгу терористичної активності;_____
3) розробка апаратної частини системи моніторінгу терористичної активності;_____
4) розробка програмної частини системи моніторінгу терористичної активності._____

5. Перелік графічних матеріалів

Блок-схема архітектури апаратно-програмного комплексу
Блок-схема структури сенсорного кластеру
Блок-схема Use Cases системи
Приклади розташування системи на мапі
Блок-схема мережевої архітектури проєкта
Дизайн друкованої плати
Блок-схема ієрархії обробки даних
Діаграма активності UML для проєкту
GUI для сервера та мобільного додатка
Формати повідомлень від мікроконтролера та постів в БД
План розташування вимірювачів для еталонного обрахунку коефіцієнтів
Приклад тріангуляції з відомими довжинами відрізків до точки

6. Завдання до спеціальної частини

– проаналізувати проблематику моніторінгу терористичної активності, включаючи основні причини, наслідки та статистичні дані про гучні події, пов'язані з цією проблемою;

– проаналізувати існуючі системи, які моніторять терористичну активність, та описати їх функції;

– розглянути, як такі системи можна впровадити в Україні, адаптувавши їх до місцевих умов та потреб служб;

– розробити ефективну систему моніторінга, яка буде доступною для широкого загалу;

– скласти рекомендації щодо ефективного використання таких систем, щоб підвищити безпеку у містах.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алексєєва А. О. канд. техн. наук, доцент	кафедра екології Медичного інституту ЧНУ імені Петра Могили	Спеціальна частина з охорони праці

Керівник роботи
ст. викладач, Старченко В’ячеслав Володимирович *(посада, прізвище, ім’я, по батькові)*

_____
(підпис)

Завдання прийнято до виконання

Шкромида Віталій Олексійович
(прізвище, ім’я, по батькові студента)

_____
(підпис)

Дата видачі завдання «____» _____ 20_____

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Апаратно-програмний комплекс для моніторингу терористичної активності

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КРБ	01.02.2024	03.02.2024	Виконав
2	Аналіз інформації стосовно загальної інформації про системи моніторинга терористичної активності	04.02.2024	18.02.2024	Виконав
3	Порівняльний аналіз існуючих систем моніторинга терористичної активності	19.02.2024	06.03.2024	Виконав
4	Формування ТЗ	19.02.2024	04.03.2024	Виконав
5	Актуалізація вимог дипломного проєкту	23.02.2024	09.03.2024	Виконав
6	Концептування математичної моделі дипломного проєкту	20.02.2024	27.02.2024	Виконав
7	Реалізація GUI для системи	01.03.2024	04.03.2024	Виконав
8	Написання DLL для проєкту	02.06.2024	07.06.2024	Виконав
9	Оформлення БКР та презентації	15.05.2024	13.06.2024	Виконав
10	Реалізація TCP сервера	28.05.2024	04.06.2024	Виконав
11	Реалізація прошивки для ESP32	04.06.2024	05.06.2024	Виконав
12	Рецензування	05.06.2024	13.06.2024	Виконав
13	Захист бакалаврської кваліфікаційної роботи	.06.2024	.06.2024	Виконав

Розробив здобувач ВО Шкромида Віталій Олексійович _____

(прізвище, ім'я, по батькові) *(підпис)*

«____» _____ 20__ р.

Керівник роботи

ст. викладач Старченко В'ячеслав Володимирович _____

(посада, прізвище, ім'я, по батькові) *(підпис)*

«____» _____ 20__ р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи
«Апаратно-програмний комплекс для моніторингу терористичної активності»
Студент 405 гр.: Шкромида Віталій Олексійович
Керівник: ст. викладач Старченко В'ячеслав Володимирович

Кваліфікаційна бакалаврська робота присвячена розробці системи апаратно-програмного комплексу для моніторингу терористичної активності. Головною метою проєкту є створення системи, яка допоможе своєчасно виявляти постріли, вибухи, інші гучні події і попереджати державні служби про це, що сприятиме підвищенню безпеки у місті та зниженню ризику терору.

Система складається з мікроконтролера ESP32, датчиків звуку KY-037 з напругою 5 В, мікрофона INMP441 з напругою 1,7 В, резисторів номіналом 35, 54, 100 Ом, конденсатора номіналом 500 мкФ, діодного міста у формі чипа, та однофазного трансформатора 230/24 В. Центр кластеру розміщується біля, або на опорах ЛЕП, де є WiFi сигнал і працює шляхом моніторингу гучних події і відправлення на сервер.

Програмний код написаний на мові програмування Arduino, яка базується на мові C++. Код забезпечує роботу з мікроконтролерами, датчиками та мікрофоном. Також він відправляє дані на сервер.

Результатом роботи є функціональний прототип системи апаратно-програмного комплексу для моніторингу терористичної активності, який може допомогти виявляти гучні події.

Пояснювальна записка кваліфікаційної бакалаврської роботи складається зі вступу, трьох розділів, висновків та трьох додатків. У вступі визначається актуальність теми, сформульовані мета, об'єкт, предмет та завдання дослідження та розроблення бакалаврської роботи. У першому розділі проведено аналітичний огляд проблеми та існуючі рішення. У другому розділі описано вибір елементів та вимоги до компонентів для системи апаратно-програмного комплексу для моніторингу терористичної активності. В третьому розділі етапи розробки системи та можливі покращення системи, показан головний код сервера, та проведено опис програмного забезпечення для системи апаратно-програмного комплексу для моніторингу терористичної активності. У висновках наведено аналіз виконаної роботи та отриманих результатів дослідження та розроблення. У додатку А наведено Довідку з перевірки на унікальність. У додатку Б – лістинг коду мікроконтролера. У додатку В – лістинг коду для віддаленого серверу.

В цілому, кваліфікаційна бакалаврська робота без додатків містить 72 с., 37 рис., 13 табл., 16 рівнянь, 23 джерела посилання.

Ключові слова: *Arduino, C++, ESP32, система апаратно-програмного комплексу для моніторингу терористичної активності*

ABSTRACT

of the Bachelor’s thesis

«Hardware and software complex for monitoring terrorist activity»

Student: Shkromyda Vitalii Oleksiiovych

Supervisor: Senior lecturer Starchenko Vyacheslav Vladimirovich

This work is devoted to the development of a system of hardware and software complex for monitoring terrorist activity. The main goal of the project is to create a system that will help timely detect shots, explosions, and other high-profile events and warn government services about them, which will help improve safety in the city and reduce the risk of terror.

The system consists of an ESP32 microcontroller, KY-037 sound sensors with a voltage of 5 V, a microphone INMP441 with a voltage of 1.7 V, resistors rated at 35, 54, 100 ohms, a capacitor rated at 500 μF, a diode city in the form of a chip, and a single-phase transformer 230/24 V..

The program code is written in the Arduino programming language, which is based on the C++ language. The code works with microcontrollers, sensors, and a microphone. It also sends data to the server.

The result of the work is a functional prototype of the system of hardware and software complex for monitoring terrorist activity, which can help to detect high-profile events.

The explanatory note of the bachelor's thesis consists of an introduction, three sections, conclusions and three annexes. The introduction defines the relevance of the topic, formulates the purpose, object, subject and objectives of the research and development of the bachelor's thesis. In the first section, an analytical review of the problem and existing solutions is carried out. The second section describes the selection of elements and component requirements for a hardware and software system for monitoring terrorist activity. In the third section, the stages of system development and possible improvements to the system are shown, the main code of the server is shown, and the description of the software for the system of hardware and software complex for monitoring terrorist activity is carried out. The conclusions provide an analysis of the work performed and the results of research and development. Appendix A contains the Certificate of Uniqueness Check. Appendix B contains a listing of the microcontroller code. In Appendix B – code listing for the remote server.

In general, a bachelor's thesis without appendices contains 72 p., 37 pic., 13 tables, 16 equations, 23 sources.

Keywords: *Arduino, C++, ESP32, hardware and software system for monitoring terrorist activity*

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ПЗ – програмне забезпечення

дБ – децибели

MP – Microphone

ID – Identificator

3D – Three Dimensions

ВСТУП

Останнім часом в Україні все активніше обговорюються питання легалізації стрілецької зброї. Світовий досвід показує, що зі зростанням кількості стрілецької зброї закономірно збільшується і кількість трагічних інцидентів з нею. Ось декілька прикладів:

– 14 січня 2023 р. У центрі Лондона, неподалік вокзалу Юстон, сталася стрілянина. Поранено трьох жінок у віці 54, 48 та 41 років, та семирічної дівчинки. Як повідомляє лондонська поліція, вогонь відкрили з автомобіля, що рухався поблизу церкви на прилеглій до вокзалу вулиці, де проходили похорони [1].

– 24 січня 2023 р. В американському штаті Каліфорнія сталося друге масове вбивство за три дні. Семеро людей убито, один – у критичному стані, вбивця здався поліції. Новий розстріл стався у прибережному місті Хаф-Мун-Бей за півсотні кілометрів на південь від Сан-Франциско [2].

Ще одним неприємним фактом є те, що у 80-95 відсотках випадків потенційні свідки до відповідних служб про постріли не повідомляють. Цивільні люди не роблять цього із різних причин: сподіваються, що це зроблять інші, не впевнені, що чули саме постріл, а хтось просто не полюбляє поліцію і не довіряє їй.

Тому все актуальнішою стає необхідність автоматизації процесу моніторингу терористичної активності. Особливо у великих містах.

Американський стартап ShotSpotter [3–6], створений у 1996 році, створив апаратуру, яка, встановлена на вулицях, фіксує постріли і негайно повідомляє про них поліцейські патрулі. Його мікрофони розташовані багатьох містах Америки. Мікрофони встановлюються на будь-якій споруді. Коли лунає гучний звук, комп'ютерна програма визначає, постріл це чи щось інше. Враховується те скільки сенсорів зафіксували постріл, і чи була звукова хвиля спрямованою, оскільки звук пострілу сильніше поширюється у бік, куди він був зроблений.

По всій Америці дані, отримані за допомогою цієї системи, використовуються в судах як доказ як звинуваченням, так і захистом.

Задоволені не всі. Критики стверджують, що прилади недосконалі, завалюють правоохоронців хибними викликами і завдають неприємностей випадковим перехожим. Крім того мікрофони надто дорогі, щоб повністю покрити ними усі міста, звук пострілу легко сплутати із вибухом петарди чи автомобільним вихлопом.

В принципі, надійність системи ShotSpotter сумнівів не викликає. Компанія заявляє, що її апаратура вірна у 97% випадків, і поліцейські, з'явившись за сигналом, можуть бути впевнені, що стрілянина справді мала місце. Але це твердження важко перевірити з тієї інформації, яку фірма надає громадськості. Тому остаточне рішення залишається за людьми.

Локалізація такої системи в України має очевидні складності. По перше це ціна обладнання та послуги. Вона є дуже високою для багатьох українських міст та громад. По друге – організація ефективної інформаційної взаємодії із відповідними службами. Адже система була розроблена для застосування у іншому законодавчому просторі. По третє – сам процес локалізації системи буде дуже затратним. Тому відчинянні розробки у цій галузі скоро будуть дуже необхідні.

Мета

Метою цього дослідження є реалізація TCP сервера з інтерфейсом користувача на мові Python збирающого з працюючих кластерів з мікроконтролерами ESP32 [7, 8] для покращення інфраструктури міських держслужб. Основні завдання включають в собі розробку програмного забезпечення для забезпечення взаємодії з мережею Інтернет, створення інтерфейсу користувача, обробку HTTP-запитів та автоматизації роботи з MongoDB [9].

Об’єкт

Методи підвищення ефективності спостереження та оперативності інформування спеціальних служб про випадки терористичної активності на контрольованої території.

Предмет

Апаратно-програмний комплекс для моніторингу терористичної активності на відкритої території з використанням акустичних сенсорів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– моделювання математичної моделі обрахунку координат;

– формування вимог для системи;

– вивчення аналогів, патентів та літератури;

– моделювання UML діаграм для проєкта;

– моделювання схеми живлення та схеми зв’язку мікроконтролера з датчиками та мікрофонов;

– формування кошториса;

– моделювання та розробка сервера та шаблону прошивки на мікроконтролер.

Практичне значення отриманих результатів:

– збір статистики;

– покращення інфраструктури для забезпечення міської безпеки.

Апробація результатів бакалаврської роботи

Результати роботи були представлені у півфіналі Міжнародного конкурсу студентських наукових робіт «Black Sea Science-2023» за напрямком «Інформаційні технології, автоматизація і робототехніка», що проводився на базі Одеського національного технологічного університету під егідою Black Sea Universities Network [10].

1 АНАЛІТИЧНА ЧАСТИНА. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ СИСТЕМИ, ЩО РОЗРОБЛЯЄТЬСЯ. ФОРМУВАННЯ ВИМОГ ДО АПАРАТНО-ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Патенти

При дослідженні патентів по темі дипломної роботи були знайдені патенти ShotSpotter, американської компанії, яка займається темою цієї роботи і є прямим аналогом. Нижче приведено три патента ShotSpotter.

1.1.1 Системи та методи зв'язку для систем виявлення зброї (US7719428B2)

Акустичні події, такі як постріли зі зброї, можуть бути важливими для безпеки та контролю. Для ефективного обробки таких даних розроблено системи та методи, які дозволяють ідентифікувати та локалізувати ці події. Один з прикладів реалізації цих методів включає:

– **Обробку повідомлень**;

Система отримує повідомлення про акустичні події, зокрема постріли зі зброї. Ці повідомлення можуть надходити від різних датчиків, розташованих на моніторинговій території. Датчики підключені через бездротову мережу, що дозволяє передавати сигнали від кожного датчика до центральної системи.

– **Локалізацію джерела події**;

Система визначає місцезнаходження джерела акустичної події. Це відбувається на основі часу прибуття інформації від різних датчиків та їх місцезнаходження. Алгоритми обчислюють координати джерела події, враховуючи розподіл часу прибуття сигналів.

– **Інші реалізації**.

Дані про час прибуття можуть бути синхронізовані для всіх датчиків, що спрощує обчислення. Система може використовувати інші параметри, такі як інтенсивність звуку, для точнішої локалізації. Ці системи та методи

допомагають забезпечити безпеку та ефективний контроль на моніторингових територіях.

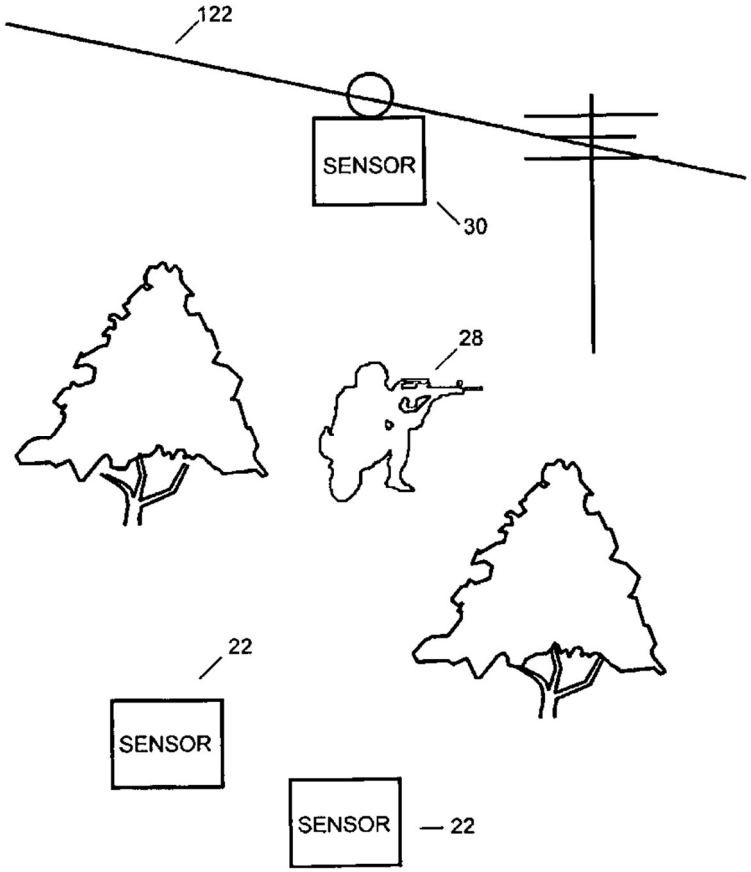


Рисунок 1.1 – Приклад роботи сенсорів

На цьому рисунку з патента показан приклад ситуації. Сенсори засікають гучний сигнал, він передається до адміністраторів, а адміністратори вже передають інформацію до державних служб.

1.1.2 Високопортативна система для виявлення акустичних подій(US7750814B2)

Акустичний датчик для носіння людиною – це інноваційний пристрій, який може виявити звукові сигнали, пов’язані з пострілами, та передати цю інформацію користувачеві. Ключові компоненти:

– **Корпус;**

Датчик має компактний корпус, який можна носити на одязі або прикріпити до обладнання. Це дозволяє використовувати його в різних ситуаціях.

– **Мікрофон;**

Вбудований мікрофон реєструє звукові хвилі. Коли відбувається постріл, мікрофон фіксує відповідний акустичний сигнал.

– **Процесор;**

Датчик обладнаний потужним процесором, який аналізує отримані звукові дані. Він визначає час надходження акустичної події, що дозволяє точно визначити момент пострілу.

– **GPS-приймач;**

Для визначення місцезнаходження датчика використовується GPS-приймач. Це дозволяє відстежувати місце, де сталася подія.

– **Дисплей.**

Інформація про акустичні події відображається на дисплеї. Користувач може бачити час, місцезнаходження та інші деталі.

Термін “натільний” вказує на те, що датчик може бути інтегрований в одяг або обладнання, яке зазвичай носить солдат. Це дозволяє забезпечити надійне виявлення пострілів та зберегти життя.

Цей акустичний датчик є важливим кроком у покращенні безпеки та оперативності військових та правоохоронних органів. Він може допомогти вчасно реагувати на загрози та зберегти життя людей.

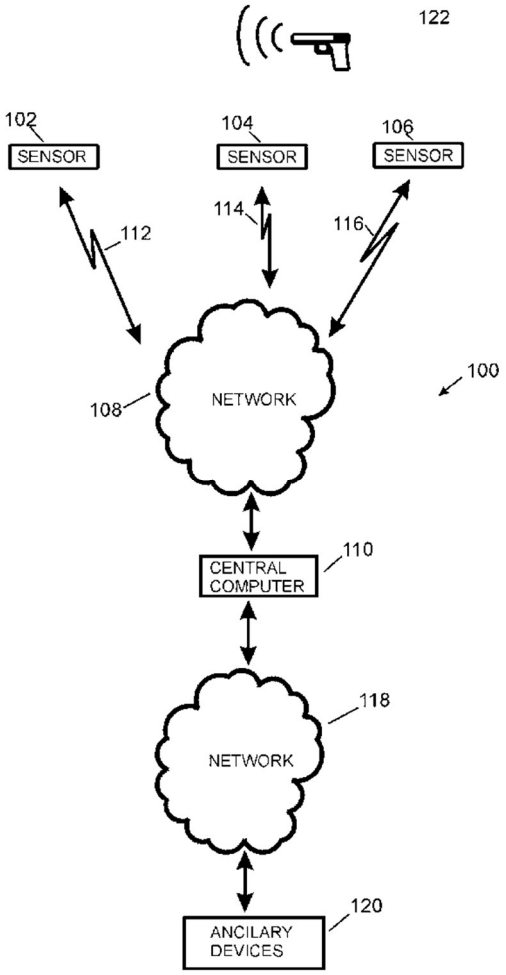


Рисунок 1.2 – Схема мережевого зв’язку у системі ShotSpotter

1.1.3 Системи та методи з удосконаленою обробкою місцезнаходження тривимірного джерела, включаючи обмеження локаційних рішень двовимірною площиною(US8369184B2)

Цей патент розкриває системи та методи обробки інформації про походження та місцезнаходження джерела або події.

Один з прикладів реалізації включає покращену обробку тривимірної локації джерела, де рішення локації обмежується двовимірною площиною. Цей спосіб також включає використання віртуальних чутливих елементів, розташованих на протилежних сторонах площини обмеження відносно реальних чутливих елементів. Інші можливі реалізації визначають місцезнаходження джерела на основі положення чутливих елементів та інформації про час прибуття та кут прибуття.

Цей патент може відкрити нові можливості для точної локації джерел та подій, що є важливим для багатьох сфер, включаючи навігацію, безпеку та наукові дослідження.

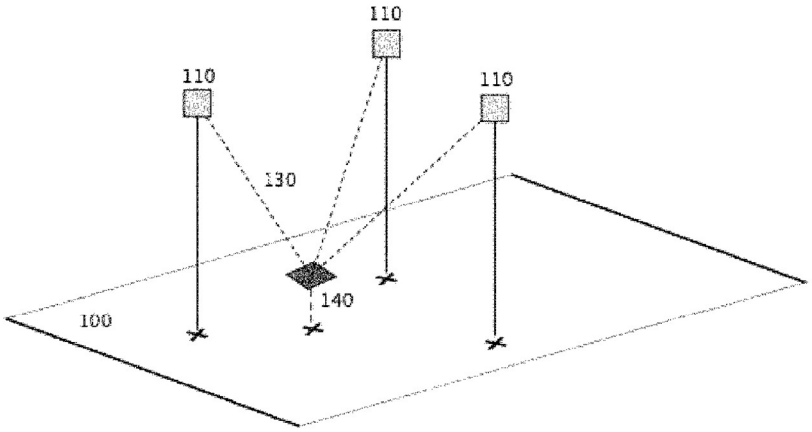


FIG. 1

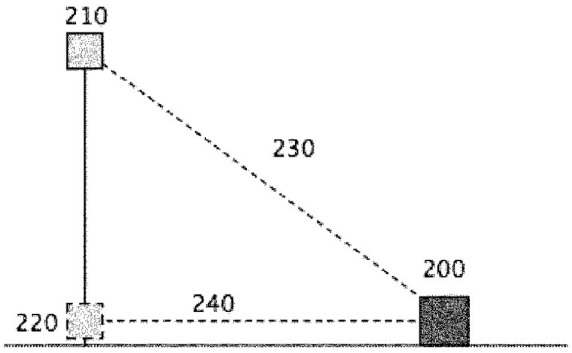


FIG. 2

Рисунок 1.3 – 3D обрахунок координат

1.2 Первинна обробка акустичних даних

Головною метою кластерного сервера є визначення та розташування джерел гучного імпульсного звуку у просторі. На рисунках показані результати реєстрації звукової хвилі від одного й того ж пострілу на відстанях 100 м та 500 м від місця події відповідно.

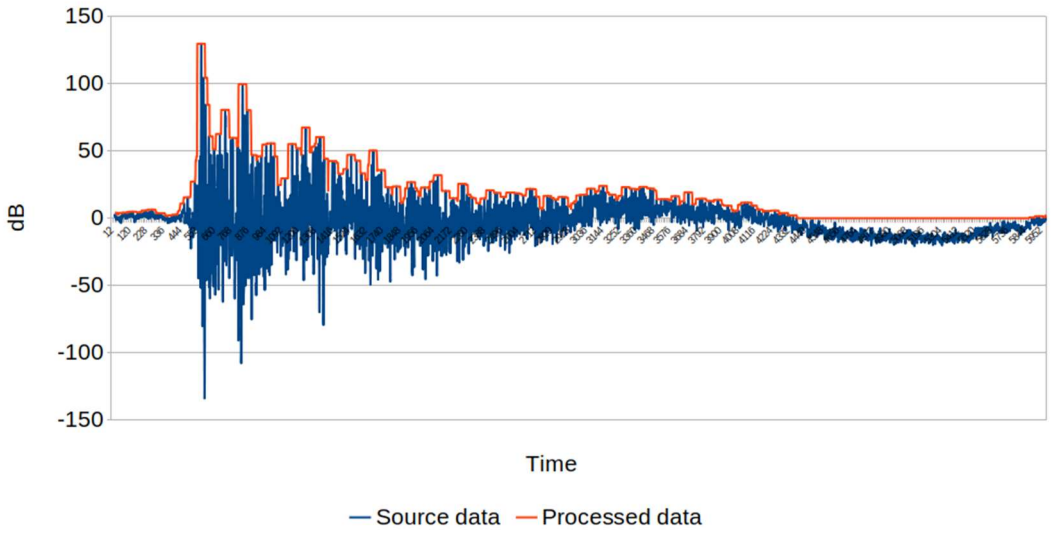


Рисунок 1.4 – Результат реєстрації та первинної обробки параметрів звукової хвилі на відстані 100 м від місця пострілу

Звуковий рівень в децибелах відображено синім кольором, і це значення реєструється сенсором. Амплітуда звукового сигналу, отримана після первинної обробки даних, показана червоним кольором.

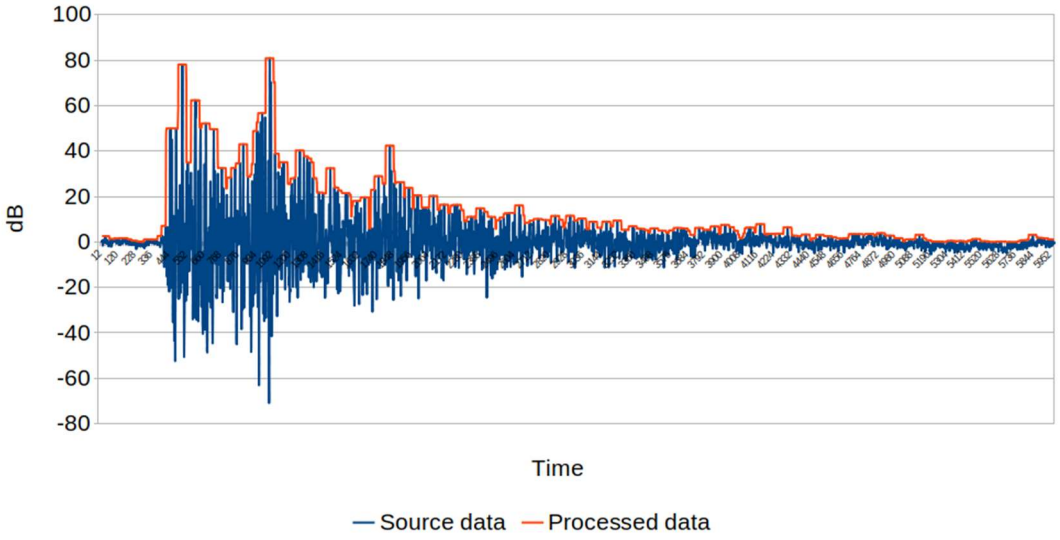


Рисунок 1.5 – Результат реєстрації та первинної обробки параметрів звукової хвилі на відстані 500 м від місця пострілу

При спостереженні за хвильовим фронтом виникає помітна деградація через дифракцію на рельєфних елементах місцевості та роздвоєння через відбиття від вертикальних перешкод. Після первинної обробки даних можна приблизно визначити відстань від місця пострілу до місця реєстрації звукової

хвилі сенсором. Проте точність цього обрахунку сильно залежить від погодних умов та особливостей місцевості.

Температура впливає на швидкість розповсюдження звукових коливань у повітрі, а широкий частотний діапазон звукових хвиль, що виникають при пострілі, призводить до природної дисперсії, рівень якої змінюється з відстанню. Крім того, рельєф поверхні впливає на можливі відбиття та розсіювання звукових хвиль

1.3 Концептування архітектури системи

Система моніторингу гучних акустичних сигналів має зіркову архітектуру. Давайте перепишемо це у власних словах. Ця система складається з декількох компонентів:

– **Сенсорні кластери;**

Це листя дерева графу. Вони розташовані на різних місцях і відповідають за збір акустичних сигналів.

– **Головний сервер.**

Це корінь дерева. Головний сервер відповідає за збір, обробку та накопичення інформації, яка надходить від сенсорних кластерів. Він також формує повідомлення про гучні акустичні сигнали та передає їх відповідним службам.

На рисунку, відображена загальна схема архітектури системи. Ця архітектура дозволяє ефективно відслідковувати гучні акустичні події та реагувати на них.

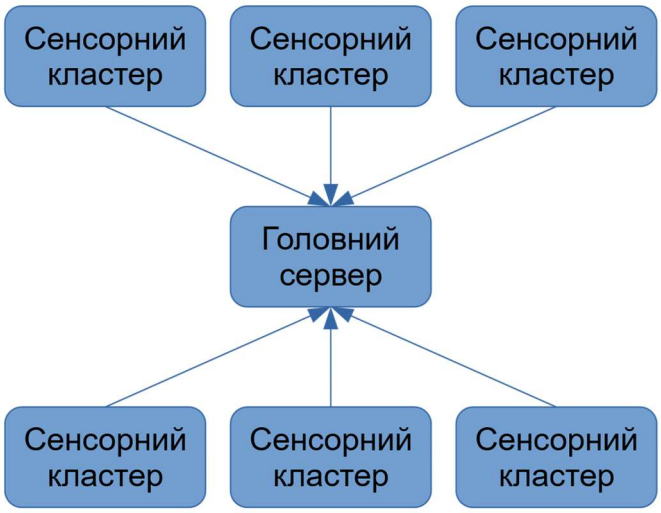


Рисунок 1.6 – Архітектура апаратно-програмного комплексу

Сенсорний кластер призначений для реєстрації часу та локалізації у просторі гучний імпульсних звуків. Він складається з трьох звукових сенсорів, що керуються за допомогою мікроконтролеру Arduino та локального серверу.

Структура сенсорного кластеру наведена на рисунку.

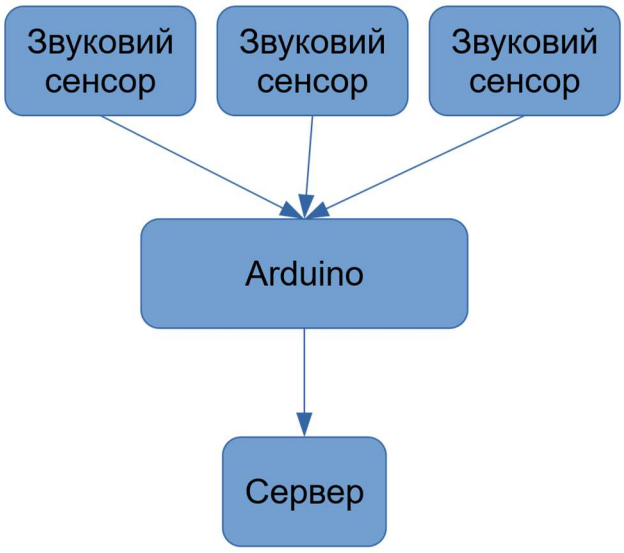


Рисунок 1.7 – Структура сенсорного кластеру

Також для проєкта смодельована діаграма Use Cases [11]. На діаграмі задіяно 5 головних акторів, тобто 3 користувача та адміністратор з монтажником. Адміністратор системи формує на сервері трикутники з даних від повідомлень мікроконтролерів. Сервер автономно реєструє гучні події та архивує в БД. Патрульна поліція може коментувати пости з БД, а адміністатор міняти статус постів та інші дані в БД. Швидка допомога, пожарні та

комунальні служби можуть також користуватися цим БД через мобільну програму. Гучна подія має три типа:

– постріл;
– вибух;
– гучний сигнал.

Гучну подію ідентифікує патрульний поліцейський. Пострілами та вибухами займається поліція та інші служби безпеки. Також вибухами займаються й пожарні з комунальними службами, а якщо гучним сигналом був крик людини у біді цим займається швидка допомога.

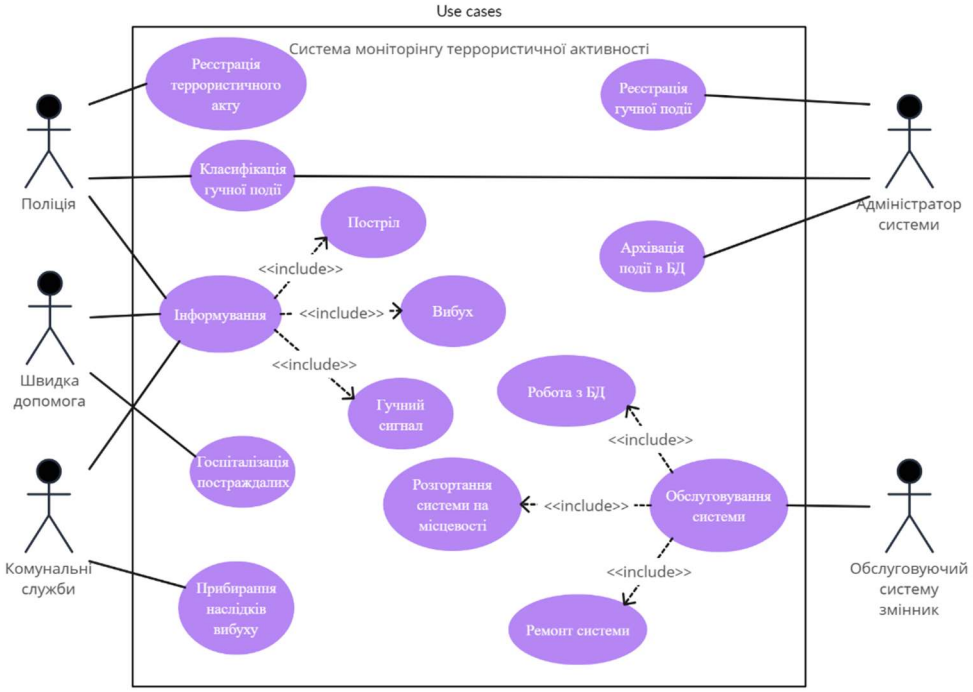


Рисунок 1.8 – Use Cases системи

Для мікроконтролерів та мікрофонів потрібні корпуса для захисту від дощу, вітру, землетрусу та інших природних та штучних перешкод роботи кластеру. Корпуса мають отвори для дротів та отвор для мікрофона. Мікроконтролер не видає велику потужність(ток споживаний одним мікрофоном чуть більше 1-го міліампера), тому вентиляція у корпусу мікроконтролера не потрібна. Кришки корпусів мають бути знімними.

1.4 Призначення та межі проєкту

Призначення системи, для якої розробляється апаратно-програмне забезпечення

Ця система призначена для виявлення вибухів, пострілів та визначення їх приблизної адреси.

Межі використання проєкту

Цей проєкт не має меж використання, але потребує опрацювання(наприклад додавання можливості оновлення прошивки на ESP32 у кластері). Також цей проєкт призначений для міської місцевості, де є столби від яких можна живити мікроконтролер, а також на які можна повісити мікрофони.

1.5 Загальний опис

Сфера застосування

Цей проєкт спроєктований для використання службами безпеки, зокрема для поліції, пожарників, швидкої допомоги та комунальних служб.

Характеристики користувачів

Користувач має бути службовцем, або обслуговуючим цієї системи.

Загальна структура і склад АПЗ. Системні вимоги

Загальна структура представляє собою групу кластерів зв’язаних з головним сервером. Мікроконтролер повинен мати WiFi-модуль та мінімум 3 цифрових GPIO піна. Сервер має мати підтримку версії Python 3.11.

Загальні обмеження

Цей проєкт намагається точно визначати місцезнаходження пострілa/вибуха але має похибку у 10 % при розрахуванні з-за того, що земля приплиснута. Також є фундаментальна проблема фальшивого спрацювання(ця проблема може бути частково вирішена використанням штучного інтелекта для перевірки звукової хвилі від гучного звука). Мікроконтролер має бути розташований у тому місці від якого його можна живити, а також там де є Wi-Fi.

1.6 Вимоги до апаратного забезпечення

Мікроконтролер

Необхідна підтримка бездротового WiFi зв‘язку або RJ45 вхід, підтримка I^2S інтерфейсу як мінімум на одному GPIO.

Датчик звука

Електретний мікрофон, наявність підсилювача та компаратора, підстроювання мінімального порога спрацьовування.

Мікрофон

Необхідна підтримка I^2S інтерфейсу.

Сервер

Дана конфігурація вибрана з урахуванням доступності комплектуючих, ПЗ та простоти використання.

Таблиця 1.1 – Конфігурація сервера

Операційна система	Windows XP SP3 або краще
Оперативна пам’ять	1 Гбайт або більше
Процесор	Pentium III або краще
Мережева карта	Від 100 Мбіт/с
Пам’ять	Від 100 Гбайт

Але рекомендованою конфігурацією сервера є LattePanda Sigma. Цей комп’ютер досить компактний, має різні графічні входи, слоти розширення, USB входи та WiFi-модуль.

Таблиця 1.2 – Рекомендована конфігурація сервера

Операційна система	Windows 10/11 або Ubuntu 22.04
Оперативна пам’ять	До 32 Гбайт
Процесор	Intel® Core™ i5-1340P 12-ядер, 16-потоків, 12 Мбайт кешу, до 4.60 GHz
Мережева карта	2 x 2.5GbE RJ45 Ports (Intel® i225-V) M.2 Wireless Module
Пам’ять	M.2 NVMe(500 Гбайт – 1 Тбайт)
Розмір та габаріти	146мм × 102мм

1.7 Вимоги до програмного забезпечення

Архітектура програмної системи

Програмна архітектура складається з кластерів що складаються з ESP32 та цифрових мікрофонів які також використовуються ESP32 з сусідніх кластерів та з сервера на які приходять дані з кластерів. Мікроконтролер повинен підтримувати роботу з *TCP* [13], *UDP* та WiFi, а сервер повинен підтримувати роботу з сокетами та *HTTP* запитами для роботи з *Google Maps API* [14].

Системне програмне забезпечення

ESP32 має мати в прошивці бібліотеки для роботи з апаратними можливостями мікроконтролера. Сервер має мати встановлений інтерпретатор мови Python та встановлений *pip3* для встановлення бібліотек.

Мережеве програмне забезпечення

ESP32 має мати в прошивці бібліотеки для роботи з сокетами. Python на сервері має мати бібліотеки *socket*, *mongodb*, *asyncio*, *requests* та *json* для зв’язка з кластерами, взаємодії з *Google Maps API* для зворотного геокодування та обробки результату у вигляді JSON [15].

Програмне забезпечення ведення інформаційної бази

Для ведення БД використовується MongoDB, а для зберігання файлів використовується додаток GridFS [16].

Мова і технологія розробки ПЗ

Для мікроконтролера використовується діалект C, а для сервера мова Python. Методологія розробки – Agile [17].

1.8 Вимоги до зовнішніх інтерфейсів

1.8.1 Інтерфейс користувача

На рисунках зображений інтерфейс [18] адміністратора системи, а на рисунку зображений приклад програми для смартфонів на базі Android. На

мобільному додатку при натисканні на кнопку з постом з БД вмикається 5-ти секундний запис прив’язаний до кнопки, якщо він є.

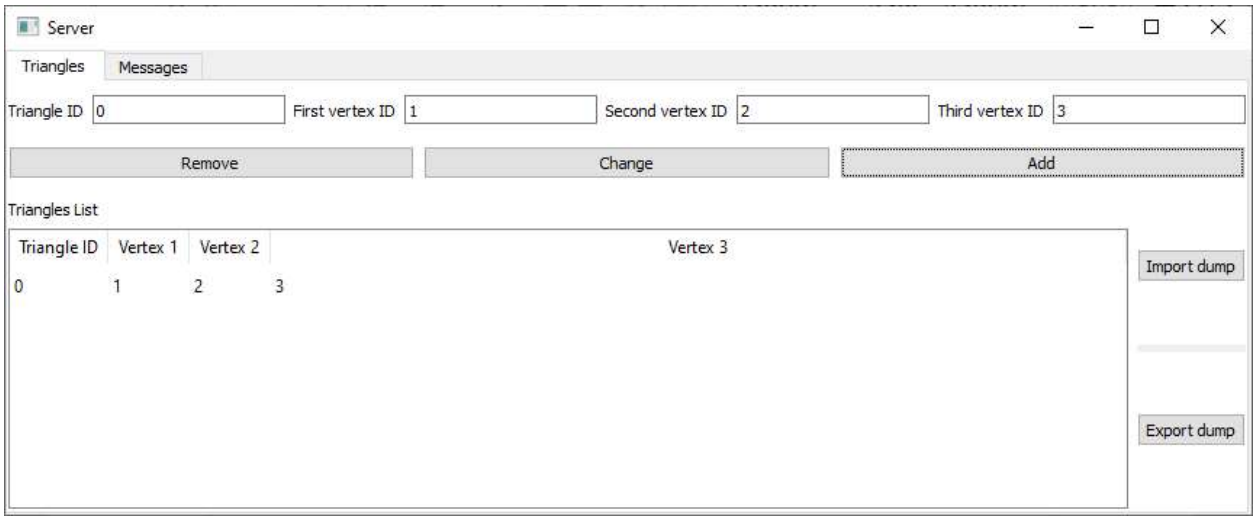


Рисунок 1.9 – Вкладка для роботи з трикутниками

На цієї вкладці створюються та редагуються трикутники. Також тут можно зробити експорт таблиці трикутників та її імпорт.

Рисунок 1.10 – Вкладка з обрахованими постами

На вкладці повідомлень з’являються обраховані пости, які паралельно завантажуються на БД-сервер MongoDB.

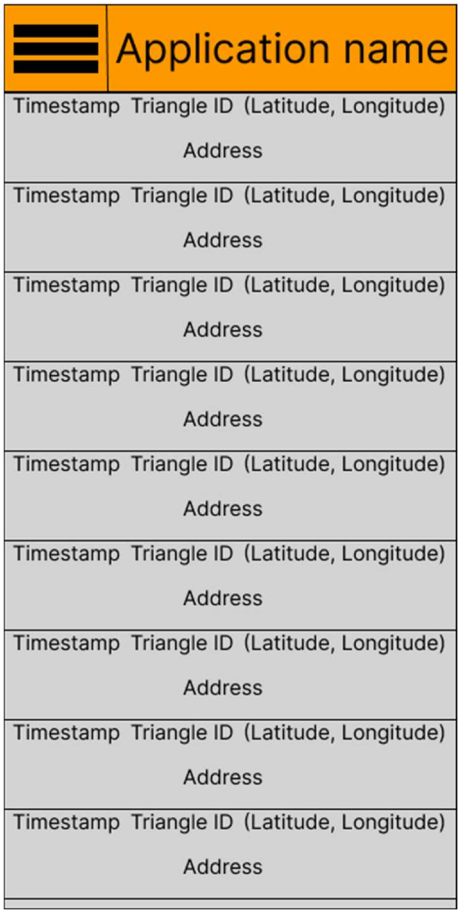


Рисунок 1.11 – Приклад GUI мобільної програми для користувачів

1.8.2 Апаратний інтерфейс

Формат першого повідомлення з кластера

Перше ядро мікроконтролера форматує перше повідомлення, яке передається кожні 5 с.

ESP32 ID | MP#1 ID | MP#1 Coords & Db | MP#2 ID | MP#2 Coords & Db | MP#3 ID | MP#3 Coords & Db

Рисунок 1.12 – Frame format першого повідомлення з мікроконтролера

Формат другого повідомлення з кластера

Друге ядро мікроконтролера форматує друге повідомлення, у якому знаходиться 5-ти секундне аудіо яке передається з затримкою 5 с після запису. Загальний час форматування та відправки другого повідомлення – 10 с.

ESP32 ID | MP#4 ID| MP#4 Coords & Db | MP#4 five second audio

Рисунок 1.13 – Frame format другого повідомлення з мікроконтролера

Структура трикутників для обрахування

Трикутники формуються адміністратором системи. Вони складаються з ID трьох мікрофонів. Один з мікрофонів може бути з підтримкою I2S [19] протокола передачі звука.

Triangle ID | First Vertex ID| Second Vertex ID | Third Vertex ID

Рисунок 1.14 – Формат трикутників для обрахунку

Формат поста у БД

База даних проєкта нереляційна, бо проєкт може розвиватись, але на даний момент є визначений формат, який вказаний на рисунку.

Timestamp | Triangle ID | Latitude | Longitude | Address | Five second audio

Рисунок 1.15 – Формат поста у БД

1.8.3 Програмний інтерфейс

ESP32 використовує Arduino API з бібліотеками для роботи з інтернетом. Сервер на мові Python не має якогось загального API. Замість цього він використовує бібліотеки зі стандартного комплекта бібліотек. Для зворотнього геокодування використовується Google Maps API. Для потенціальної мобільної програми буде використовуватись Android SDK [20].

1.8.4 Комунікаційний протокол

Мікрофони до мікроконтролера: фізичний протокол – I2S.

Мікроконтролери до сервера:

– Фізичний протокол – WiFi;

– Логічний протокол – TCP/IP.

Сервер до БД:

– Фізичний протокол – Ethernet;

– Логічний протокол – HTTP.

БД до мобільної програми:

– Фізичний протокол – 3G/LTE/4G;

– Логічний протокол – HTTP.

1.9 Інші вимоги

Супроводжуваність

Проєкт має бути таким, що легко розширюється та підлаштовується. Програмно-апаратна архітектура дозволяє це робити.

Мобільність

Для цього проєкта мобільність не потрібна. Усе стаціонарно.

Чутливість

Мікрофони обов’язково повинні бути цифрові та мати кілька виводів, бо можуть використовуватись ESP32 з сусідніх кластерів. Мінімальна чутливість мікрофонів має бути близько 70 дБ.

Надійність

У цьому проєкту дуже важлива надійність монтажа кластерів. Дроти потрібно прокладати під землею, WiFi для ESP32 має бути стабільним.

Безпека

Усі ESP32 повинні мати корпуса, бути вмонтованими на столби. Дроти повинні бути від 16-ти мм^2 в перерезі.

Висновки до розділа

У цьому розділі були розглянути патенти компанії ShotSpotter, сформовані вимоги до системи і кластерів, намальовані UML діаграми, формати повідомлень і постів у БД, GUI мобільного додатку та панелі адміністратора, обрані програмні та апаратні інтерфейси які будуть використовуватись у системі.

2 МАТЕМАТИЧНІ МЕТОДИ ТА МОДЕЛЮВАННЯ Ї ПРОЄКТУВАННЯ СИСТЕМИ, ЩО РОЗРОБЛЯЄТЬСЯ

2.1 Розрахунок координат методом тріангуляції

2.1.1 Методика вимірювання відстані до місця вибуху

Гучність звуку, вимірювана в децибелах (дБ), зменшується обернено пропорційно квадрату відстані від джерела звуку [21]. Це означає, що при подвоєнні відстані до джерела звуку його гучність зменшується на 6 дБ. Наприклад, якщо рівень звукового тиску на відстані 1 м від джерела становить 90 дБ, то на відстані 2-х м він буде 84 дБ, а на 4-х м – 78 дБ.

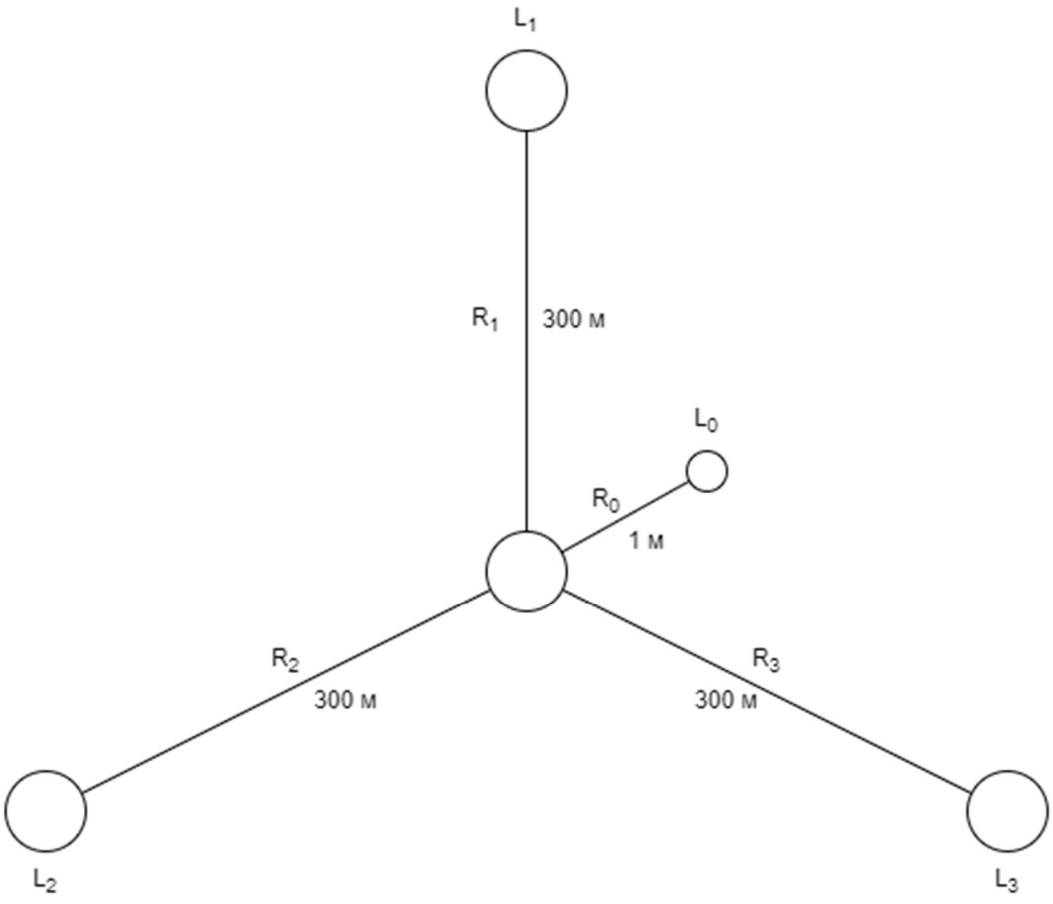


Рисунок 2.1 – План розташування вимірювачів для еталонного обрахунку коефіцієнтів

Цю залежність можна описати формулою:

$$L = L_0 - 20 * \log_{10}(\frac{R}{R_0}), \quad (2.1)$$

де L – рівень звукового тиску в децибелах на відстані R від джерела звуку;

L_0 – рівень звукового тиску в децибелах на еталонній відстані R_0 (зазвичай 1 м);

R – відстань від джерела звуку до точки вимірювання в метрах.

$$k_i = \frac{R_i}{L_i}, \quad (2.2)$$

$$R_i = k_i * L_i, \quad (2.3)$$

де R_i – відстань від датчика звуку до точки гучної події;

k_i – коефіцієнт для розрахунків який розраховується експериментально;

L_i – гучность яку вловив датчик звуку;

Важливо зазначити, що ця формула справедлива лише для точкових джерел звуку в ідеальних умовах (без відбиття та поглинання звуку). У реальних умовах поширення звуку може бути складнішим через вплив таких факторів, як відбиття від перешкод, поглинання звуку атмосферою та інші. Ось декілька додаткових факторів, які можуть впливати на гучність звуку:

– потужність джерела звуку: чим потужніше джерело звуку, тим воно буде гучнішим;

– частота звуку: люди більш чутливі до звуків середньої частоти, ніж до низьких або високих частот;

– напрямність джерела звуку: деякі джерела звуку, наприклад, динаміки, більш гучні в певному напрямку;

– акустика середовища: звук може поглинатися або відбиватися різними матеріалами, що впливає на його гучність;

– виконання калібрування системи: розміщуємо сенсори навколо точки вибуху у вигляді трикутника. Виконуємо тестовий вибух. На відстані 1 м замірюємо гучність L_0. Знаючи дистанцію R_i отримуємо k_i.

2.1.2 Тріангуляція з відомими довжинами відрізків до точки

Нехай є два мікрофона, розташовані на місцевості у точках a(x_a, y_a) та b(x_b, y_b), на відстані L одна від одної. Вони реєструють вибух з гучністю L_a і L_b відповідно. Відношення зареєстрованих гучностей дорівнює відношенню відповідних відстаней від місця вибуху M до точок розташування мікрофонів R_a та R_b. Тобто

$$\frac{L_a}{L_b} = \frac{R_a}{R_b} = const, \tag{2.4}$$

Таким чином, знаючи координати точок a і b, відстані R_a та R_b, треба знайти координати точки вибуху M.

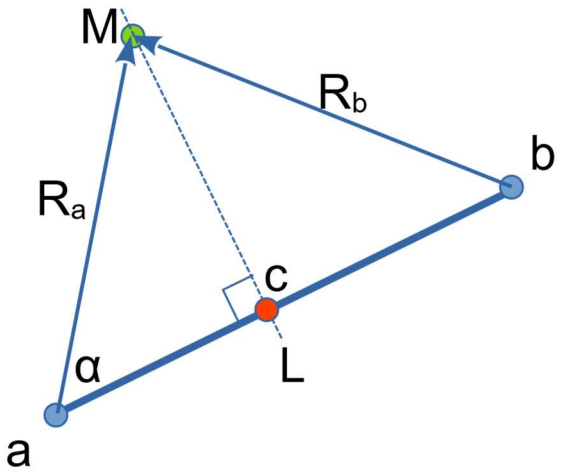


Рисунок 2.2 – Триангуляція завдяки двох точок

Знаючи координати точок a і b, можна визначити довжину та орієнтацію відрізка [a, b].

$$L = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}, \tag{2.5}$$

При незмінному відношенні $\frac{R_a}{R_b} = const$ усі можливі розташування точки M потраплять на пряму (MC), яка є перпендикулярною до прямої, який належить відрізок [a, b].

Рівняння прямої (AB) є лінійним. Тобто

$$y = k_{ab} \cdot x + b_{ab}, \tag{2.6}$$

де k_{ab} – коефіцієнт нахилу прямої до горизонтальної вісі координатя;

а b_{ab} – вертикальне зміщення прямої (AB).

Тобто

$$k_n = -\frac{x_a - x_b}{y_a - y_b}, \text{ тобто } k_n = -\frac{1}{k_{ab}}, \tag{2.7}$$

Для обрахунку b_n треба визначити координати точки перетину прямих (AB) і (MC), тобто точки $C(x_c, y_c)$. Тоді

$$b_n = y_c - k_n * x_c, \tag{2.8}$$

Для обрахунку координат точки C скористаємося залежністю

$$R_c = R_a * \frac{R_a^2 - R_b^2 + L^2}{2R_a L}, \tag{2.9}$$

де R_c – довжина проекції вектору R_a на пряму (AB).

Тепер, знаючи R_c можна обрахувати координати точки C

$$\begin{cases} x_c = R_c * x_b + (1 - R_c) * x_a \\ y_c = R_c * y_b + (1 - R_c) * y_a \end{cases}, \tag{2.10}$$

Тепер через точку *C* можна провести пряму можливих рішень, яка є перпендикулярною до прямої *(AB)*.

Перетин двох (або трьох) таких прямих дасть нам шукану точку *M*.

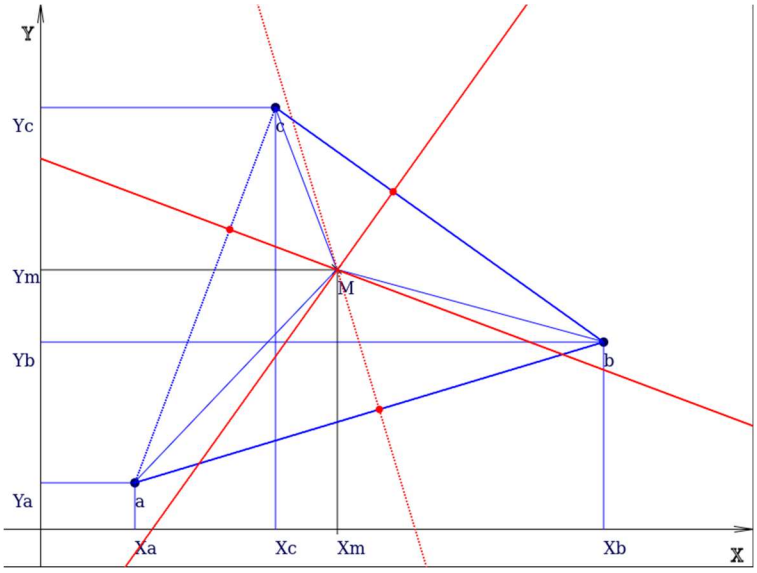


Рисунок 2.3 – Розрахунок нормалей

На малюнку такі прямі показані червоними лініями.

Таким чином задача пошуку координат точки $M(x_M, y_M)$ зводиться до рішення системи з двох лінійних рівнянь:

$$\begin{cases} y_M = k_{n1} \cdot x_M + b_{n1} \\ y_M = k_{n2} \cdot x_M + b_{n2} \end{cases} \tag{2.11}$$

Без квадратів, квадратних коренів, та без втрат знаків при від’ємних координатах.

За допомогою прототипу системи вдалося досягти точності локалізації місця пострілу на відкритій місцевості близько 10 м при розташуванні сенсорів на відстані понад 800 м.

2.2 Схема кластера

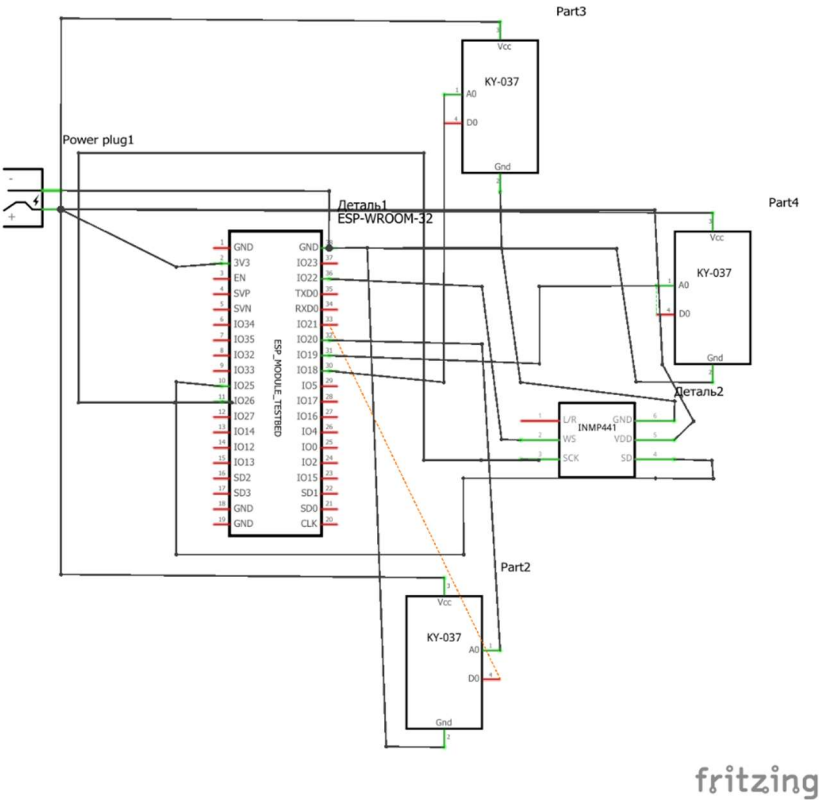


Рисунок 2.4 – Схема кластера

Схема живлення на виході має 24 В та 0,1067 А.

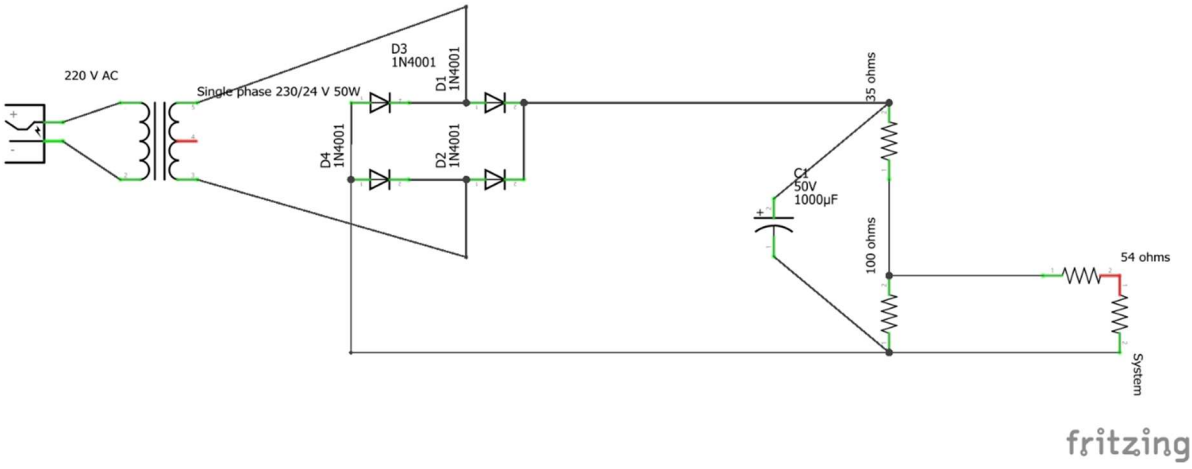


Рисунок 2.5 – Схема живлення

На рисунках зображені приклади розташування мікрофонів та мікроконтролерів. Шестикутниками зображені мікроконтролери, а прямокутниками мікрофони. Ліній зображені для наочності.

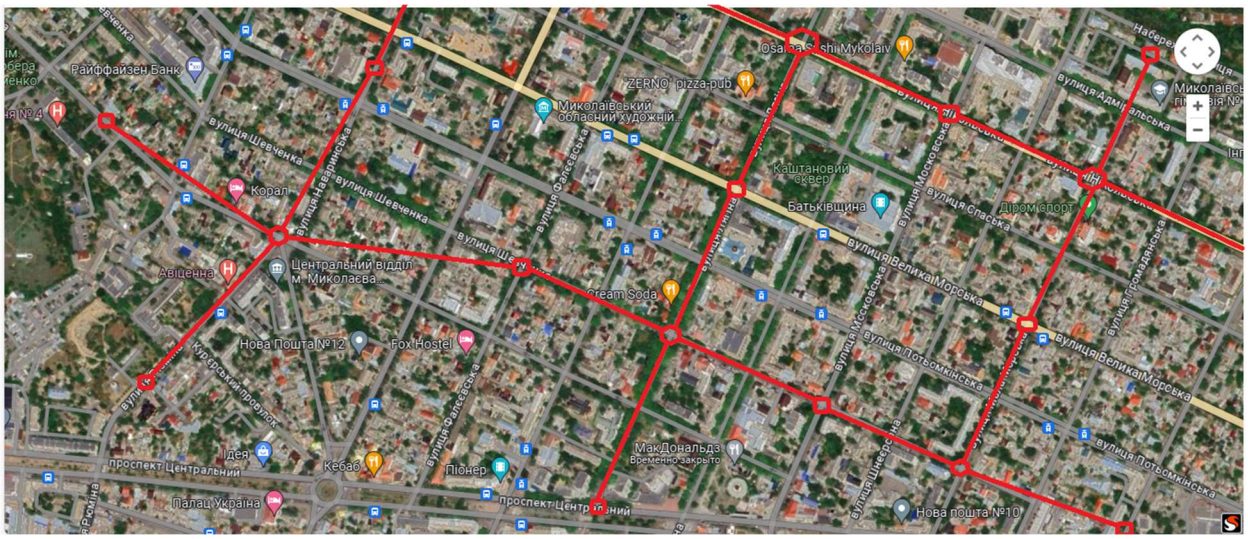


Рисунок 2.6 – Приклад розташування системи у Центральному районі Миколаєва

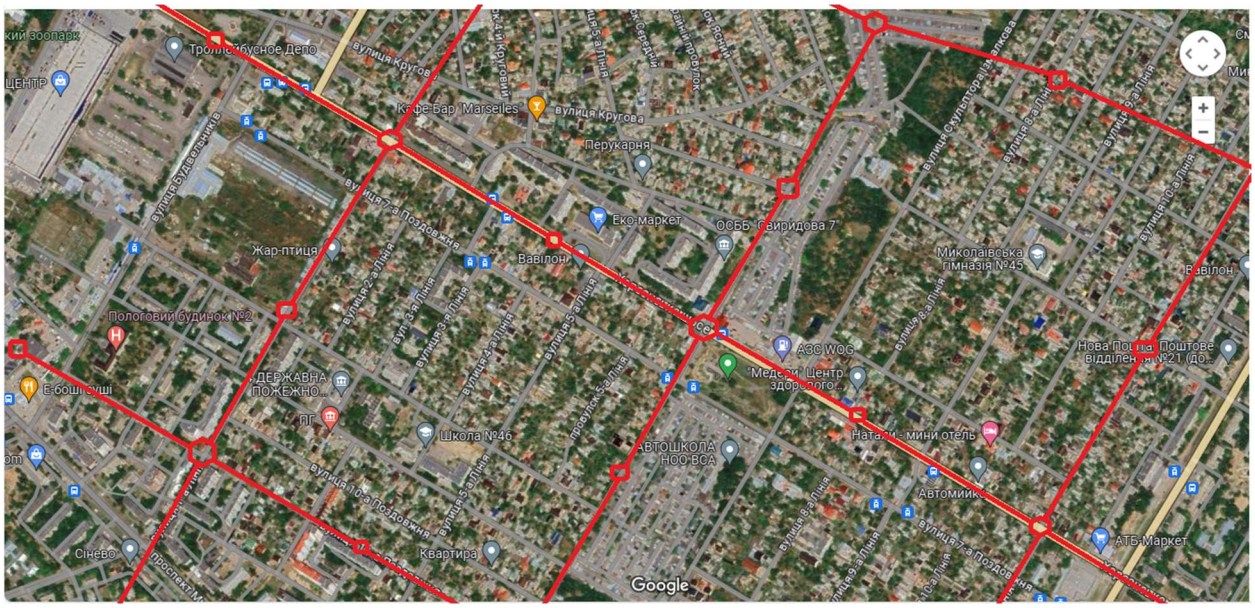


Рисунок 2.7 – Приклад розташування системи в Інгульскому районі Миколаєва

Розмітка у Корабельному районі виявилась найпроблемнішою з-за неквадратної застройці. Потребуються нестандартні коефіцієнти з-за нестандартної відстані між датчиками.

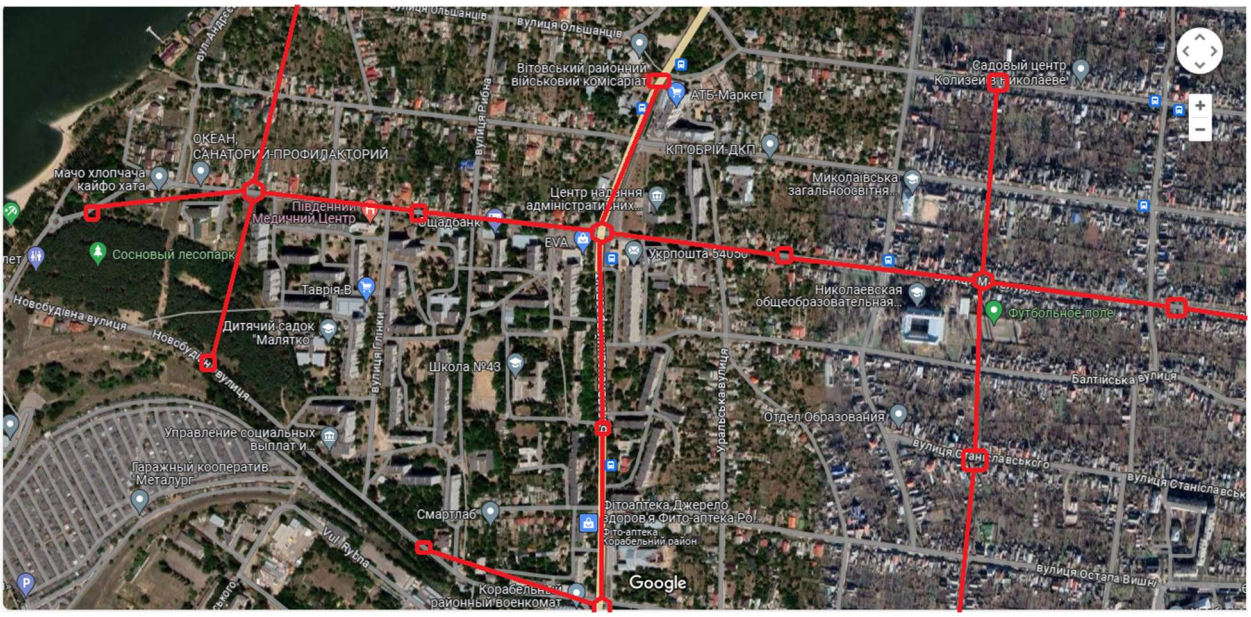


Рисунок 2.8 – Приклад розташування системі у Вітовському районі Миколаєва

2.3 Компоненти кластера

2.3.1 ESP32

ESP32 – це захоплюючий мікроконтролер, який об’єднує в собі низку потужних можливостей для розробки IoT-проєктів та сенсорних систем.

ESP32 використовує 32-бітний процесор Xtensa LX7 з двома ядрами, який працює на частоті до 240 МГц. Це дозволяє виконувати завдання швидко та ефективно.

Модуль має 512 Кбайт оперативної пам’яті (SRAM) та 384 Кбайт постійної пам’яті (ROM) на чіпі. Також є можливість підключення зовнішньої флеш-пам’яті через інтерфейси SPI, Dual SPI, Quad SPI, Octal SPI, QPI та OPI.

ESP32 має 45 програмованих GPIO, а також інтерфейси SPI, I2S, I2C, PWM, RMT, ADC та UART.

Це дозволяє підключати різноманітні пристрої та сенсори. Мікроконтролер забезпечений захистом від несанкціонованого доступу, включаючи RSA-базовий безпечний запуск, AES-XTS-шифрування флеш-пам’яті та інші механізми.

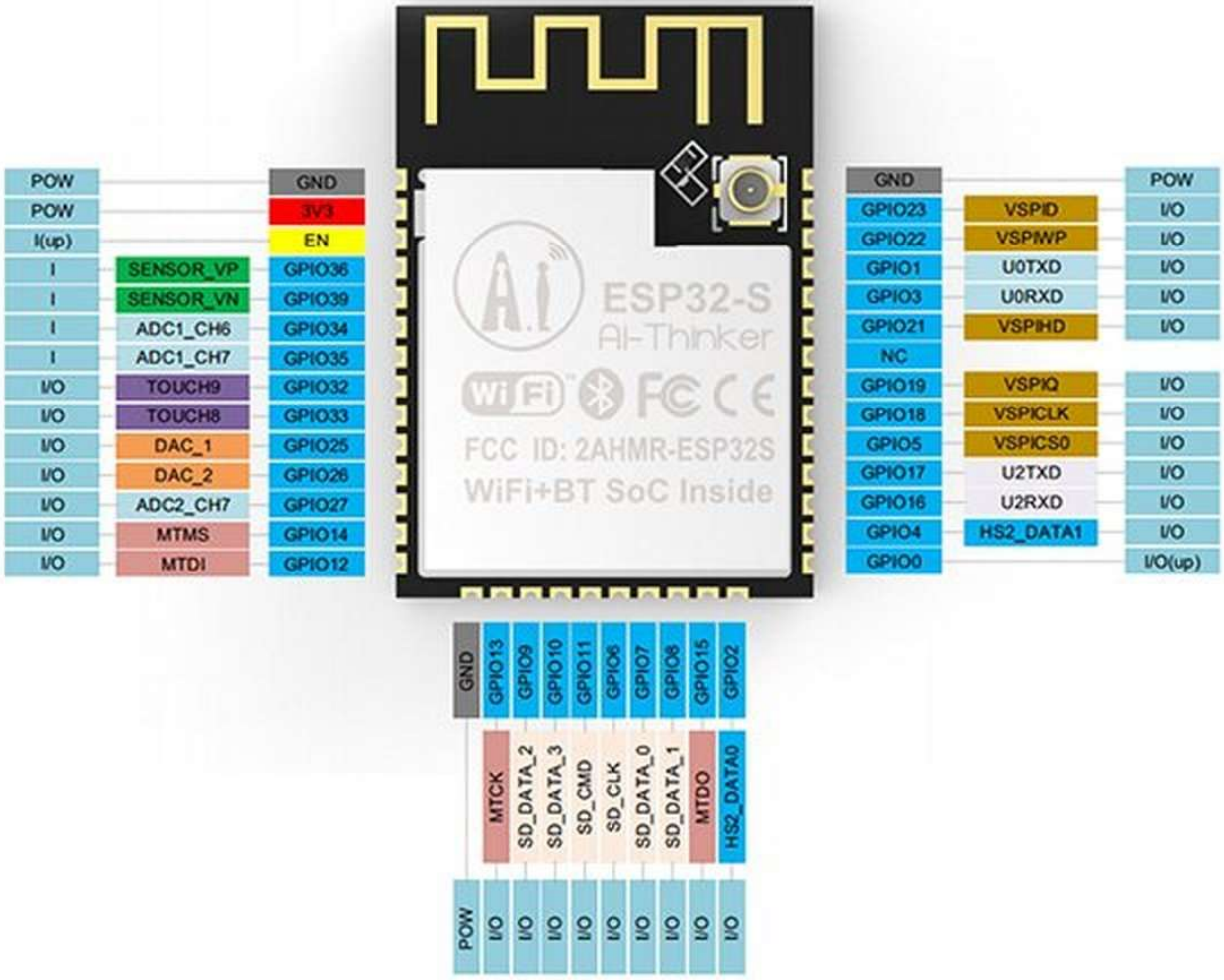


Рисунок 2.9 – ESP32

Для розробки на ESP32 доступні різні плати, такі як ESP323-DevKitC-1, яка має роз’єми для підключення периферійних пристроїв та може бути використана для розробки прототипів.

Цей мікроконтролер є потужним інструментом для розробки IoT-проєктів, сенсорних систем та інших застосувань. Його можливості та надійність роблять його популярним серед розробників.

Таблиця 2.1 – Характеристики ESP32

Процесор, мГц	Tensilica Xtensa LX6 microprocessor @ 160-240
Пам’ять, Кбайт SRAM	520
Живлення, $В_{пост}$	2.2-3.6
Бездротовий зв’язок	WiFi: 802.11 b / g / N Bluetooth: v4.2 BR/EDR and BLE
Дротовий зв’язок	Ethernet MAC interface with dedicated DMA and planned IEEE 1588 Precision Time
Максимальна швидкість передачі даних, Мегабіт/с	150 при 11n HT40, 72 при 11n HT20, 54 @ 11g, и 11 при 11b
Робоча температура, °C	від −40 до +125

2.3.2 KY-037

KY-037 – це цікавий мікрофонний сенсорний модуль з високою чутливістю. Він має ядро у вигляді чутливого конденсаторного мікрофона, який може відчувати звукові коливання в навколишньому середовищі та перетворювати їх на електричний сигнал. Потім цей сигнал підсилюється в підсилювачі для подальшої обробки.

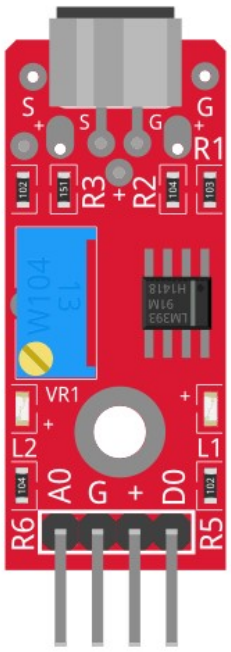


Рисунок 2.10 – KY-037

Сенсор надає як аналоговий, так і цифровий вихід. Аналоговий вихід змінюється відповідно до інтенсивності звуку, який отримує мікрофон, а цифровий вихід може використовуватися як ключ, що активується, коли інтенсивність звуку досягає певного порогу. KY-037 можна використовувати для вимірювання інтенсивності звуку.

Таблиця 2.2 – Характеристики KY-037

Вхідна напруга, $В_{пост}$	3.5 – 5.5
Вихідний сигнал	Цифровий та аналоговий
Робоча частота, Гц	20 – 20000
Відношення сигнал/шум, дБ	58
Розміри, мм	40×17

У схемі використовується саме аналоговий пін датчика.

2.3.3 INMP441

INMP441 – це цифровий MEMS мікрофон з нижнім портом. Він має високу чутливість та велике співвідношення сигнал/шум (SNR) – 61 дБА. Його електричний вихід підключається за допомогою I²S інтерфейсу, що дозволяє безпосередньо з’єднувати INMP441 з цифровими процесорами, такими як DSP та мікроконтролери, без потреби в аудіокодеку в системі. Цей мікрофон має плоску широкосмугову частотну характеристику, що дозволяє отримувати природний звук з високою зрозумілістю.

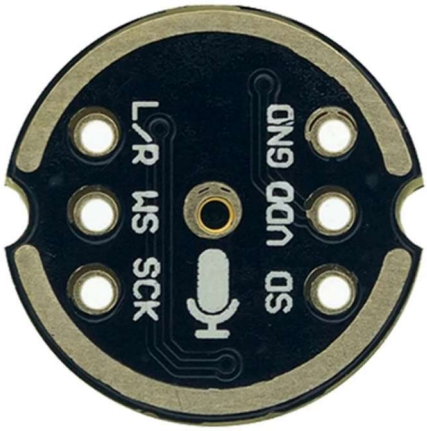


Рисунок 2.11 – INMP441

INMP441 можна використовувати для:

– телеконференцій;
– ігрових консолей;
– мобільних пристроїв;
– систем безпеки.

Таблиця 2.3 – Характеристики INMP441

Інтерфейс, біт	I2S – 24
Живлення, $В_{пост}$	1.62 – 3.63
Струм споживання, мА	1.4
Чутливість, dBFS	-26
Робоча частота, Гц	60 – 15000
Відношення сигнал/шум, dBA	61(SNR)
Робоча температура, °С	-40 – + 85
Розміри, мм	15x10x3

Цей мікрофон має плоску широкосмугову частотну характеристику, що дозволяє отримувати природний звук з високою зрозумілістю.

2.3.4 GBJ2510 GBJ

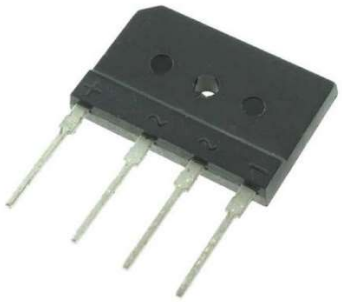


Рисунок 2.12 – Діодний міст GBJ2510 GBJ

Цей чіп розрахован на однофазне живлення з частотою 60 Гц.

Таблиця 2.4 – Основні характеристики GBJ2510 GBJ

Максимальне постійне напруження, В	1000
Максимальне змінне напруження, В	700
Максимальний струм, А	25
Робоча температура, °С	від -65 до +150

2.3.5 Керамічні трубчаті резистори С5-35В

Рисунок 2.13 – С5-35В

Ці резистори використовуються на дільнику напруги, тому потрібно, щоб вони мали високу потужність. Так як дільник напруги 1 до 0,35, то резистори мають мати номінали 35 та 100 Ом.

Таблиця 2.5 – Характеристики С5-35В

Ліміт напруги, В	1400
Відхил супротиву	±5%, ±10%
Номінали, Ом	35, 100
Габариты D×L×H×d, мм	30×90×43×20
Маса, г	90
Номінальна потужність, Вт	50

2.3.6 SQP резистори

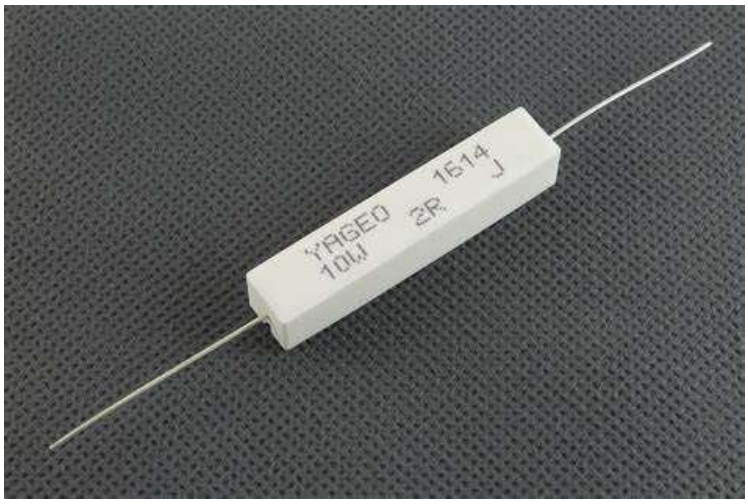


Рисунок 2.14 – SQP резистор

Цей резистор використовується тільки для того, щоб зменшити вихідний струм до 0,1067 А.

Таблиця 2.6 – Характеристики SQP резистора

Ліміт напруги, $В_{пост}$	350
Відхил супротиву	±5%
Номінал, Ом	54
Габариты D×L×H, мм	10×10×48
Маса, г	10
Номінальна потужність, Вт	10

Цей тип резисторів має відносно середню номінальну потужність.

2.3.7 Силовий кабель для живлення КГ 2х1,5

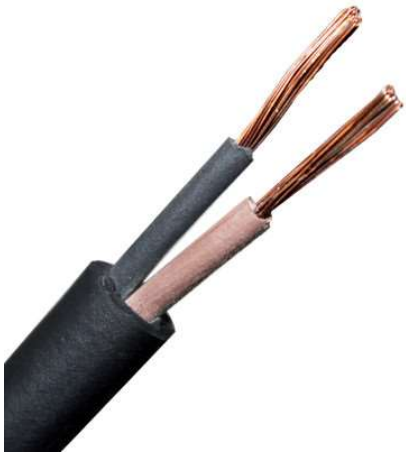


Рисунок 2.15 – КГ 2х1,5

Цей кабель є силовим та використовується у схемі живлення. Достатньо 2-х м, так як мікроконтролер знаходиться біля верхівки опори ЛЕП. Також так як кластер живиться від одної фази 380 В, то жили мають мати наконечники, для того, щоб уся жила проводила живлення.

Таблиця 2.7 – Характеристики силового кабелю

Матеріал жили	Мідь
Кількість жил, шт	2
Оболонка	Шлангова гума
Перетин жили, $мм^2$	1,5
Тип кабелю	КГ
Потужність в 1 фазной мережі (U=220V), кВт	6,6

2.3.8 Кабель для аналогового сигналу

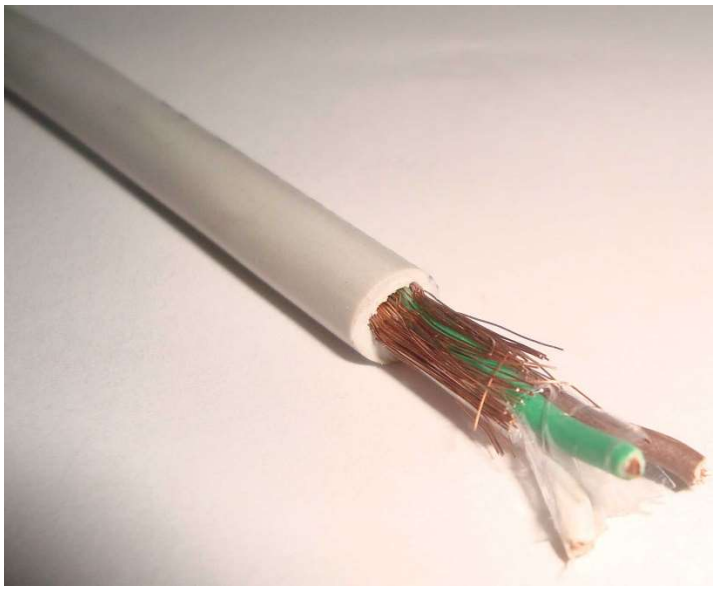


Рисунок 2.16 – Кабель управління екранований 3х1

Цей кабель використовується для зв’язку датчиків та мікрофону з АЦП пінами мікроконтролера. Так як датчики за реальним розміром розташовані на 300 м від мікроконтролера, то максимальна втрата напруги на 300-т метровому кабелі є 0,01 В. Мікрофон розташован поряд з мікроконтролером, тому для нього можна використати кабель з набагато меньшим перетином жили. Загальна потрібна кількість цього кабеля 901 м.

Таблиця 2.8 – Характеристики кабелю управління

Матеріал жили	Мідь
Кількість жил, шт	3
Ізоляція жил	ПВХ пластикат ТІ2. Не підтримує горіння
Перетин жили, мм^2	1
Екран	Мідне плетіння
Зовнішня ізоляція	ПВХ пластикат ТМ2. Не підтримує горіння
Колір зовнішньої ізоляції	Сірий
Номінальна напруга, $В_{пост/перем}$	450/750
Категорія гнучкості	5
Робоча температура, °С	До +70

2.3.9 Однофазний трансформатор NDK-150VA 230/24 IEC CHINT

Рисунок 2.17 – Однофазний трансформатор

Цей трансформатор використовується для зниження напруги до 24 В змінного струму. За формулою при зниженні напруги трансформатором зворотньо-пропорційно збільшується струм. При потужності 50 Вт на другії обмотці буде сила струма 2,08 А.

Таблиця 2.9 – Характеристики трансформатора

Номінальна потужність, Вт	150
Напруга первинна U_1, В	230
Вторинна напруга U_2, В	24
Частота, Гц	50/60
Температура навколишнього середовища, °С	від -25 до +40
Висота над рівнем моря, м	не більше 2000
Відносна вологість повітря, %	не більше 95
Категорія розміщення	3
Стандарт відповідност	МЕК 61558-2-6

2.3.10 Згладжуючий конденсатор

Рисунок 2.18 – Згладжуючий конденсатор

Цей конденсатор використовується для згладження хвилі напруги.

Таблиця 2.10 – Характеристики конденсатора

Номінал, мкФ	500
Номінальна напруга, Вт	450
Робоча температура, °С	до +105

2.4 Кошторис

Як видно з кошториса, значно дешевше біля кожного датчика примонтувати мікроконтролер, але у місцях де нема зв’язку треба використовувати кабелі. Якщо на місцевості є 3G сигнал, то можна приєднати GSM модуль для мікроконтролерів з сімкою з пакетом безлімітного інтернета(за місяць цей прототип відправляє 106,272 Гбайт на сервер).

Таблиця 2.11 – Кошторис системи з трьома датчиками

Найменування	К-ть	Од. Вим	Ціна, грн	Вартість, грн
ESP32	1	шт.	391,00	391,00
KY-037	3	шт.	24,00	72,00
INMP441	1	шт.	121,60	121,60
GBJ2510 GBJ	1	шт.	94,00	94,00
С5-35В 100 Ом 50 Вт	1	шт.	99,31	99,31
С5-35В 35 Ом 50 Вт	1	шт.	126,39	126,39
SQP резистор 54 Ом 10 Вт	1	шт.	11,80	11,80
Пусковий конденсатор Eazil 500 мкф 450 В	1	шт.	314,00	314,00
КГ 2х1,5	2	м	23,30	46,60
Кабель управління екранований 3х1 мідний гнучкий трижильний	901	м	58.90	53068,90
			Разом	54345,60

Висновки до розділа

В другому розділі була розглянута математична модель обрахунку координат та спроєктована система загалом. З обрахунків видно, що варіант з передаванням сигнала по кабелю є дуже коштовним з-за ціни кабелю, тому цей варіант раціонально використовувати для малоурбанізованої місцевості. Також була представлена схема живлення та схема з’єднання контролера та датчиків і мікрофона. Були намальовані приблизні схеми розташування мікроконтролера та датчиків у районах міста Миколаєва.

3 АПАРАТНО-ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Розробка апаратного забезпечення

Понижаючий трансформатор

Трансформатор використовується для зниження напруги. При роботі у трансформаторі зникає 10-25 % потужності. Вихідний струм на трансформаторі 2,08 А, а напруга 24 В змінного струму.

Діодний міст

Діодний міст використовується конвертації змінного струму на постійний струм. При цьому на виході напруга множиться на $\sqrt{2}$(або на 1,41).

Згладжуючий конденсатор

Так як на виході міста хвиля залишається хвилястою, потрібно згладити хвилю як мінімум на 95 %.

Дільник напруги

У схемі використовується дільник напруги з пропорцією 1:0,35.

3.2 Розробка програмного забезпечення для сервера

3.2.1 Архітектура програмного забезпечення системи

На рисунку зображена діаграма мережевої архітектури між різними пристроями та програмами. На зв’язуючих гілках мережевої схеми вказані апаратні та програмні інтерфейси зв’язку компонентів системи. Також показаний зв’язок з сервером *Google Maps API*. Для прикладу у схемі вказаний додаток на Java для Android.

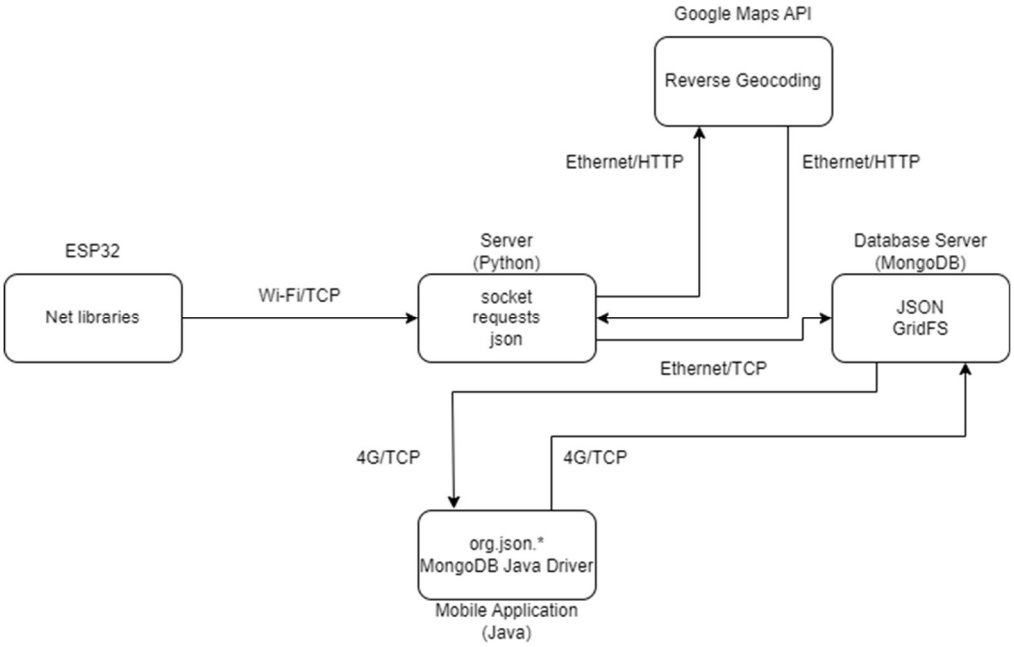


Рисунок 3.1 – Мережева архітектура проєкта

На рисунку зображена ієрархія обробки даних у мережі. На кожному етапі дані перетворюються у інші типи даних та структури. У кінці ланцюга дані оброблюються у програмі для кінцевого користувача на телефоні(також можливо написати *вебзастосунок* [12] для кросплатформої роботи з *БД* та адміністратором системи).

Рисунок 3.2 – Ієрархія обробки даних

Знизу приведена діаграма активності системи.

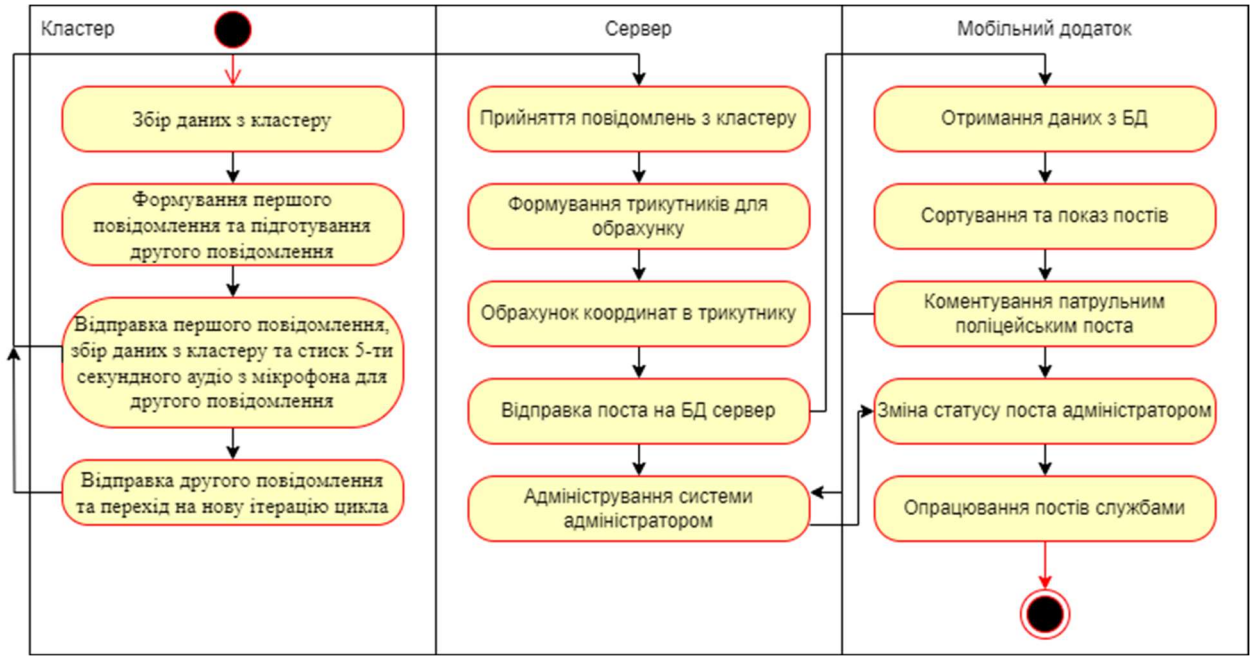


Рисунок 3.3 – Діаграма активності UML для проєкту

Ця діаграма описує роботу всієї системи. Також показаний зв’язок між кластерами, сервером та додатками для використання державними службами.

3.2.2 Мова Python

Python – це високорівнева інтерпретована мова програмування з простим синтаксисом, що робить її легкочитабельною та дуже зручною для початківців. Перша версія Python була випущена у 1991 році, і вона продовжує зростати в популярності. Python є крос-платформеною мовою, що означає, що ви можете виконувати один і той же код на будь-якій операційній системі з інтерпретатором Python.

Код Python можна писати на інших мовах (наприклад, C++), і користувачі можуть додавати низькорівневі модулі до інтерпретатора Python для налаштування та оптимізації своїх інструментів. Ця бібліотека доступна для всіх і дозволяє користувачам не писати код для кожної окремої функції. Ви можете використовувати вбудовані модулі, які допомагають в повсякденному програмуванні та більш складних завданнях.

Python є популярним серед розробників, дослідників та даних вчених. Його простота, гнучкість та широкий спектр застосувань роблять його

потужним інструментом для різних завдань. Наприклад, Python активно використовується в області наукових досліджень, веб-розробки, аналізу даних, машинного навчання та штучного інтелекту

3.2.3 Мова C++

C++ – це потужна мова програмування, яка поєднує можливості мови C з додатковими функціями для об’єктно-орієнтованого програмування. Ось деякі ключові аспекти мови C++:

– **Об’єктно-орієнтований підхід;**

C++ дозволяє створювати класи та об’єкти, що сприяє більш зручному та логічному програмуванню.

– **Шаблони;**

Мова підтримує шаблони, що дозволяють створювати загальні алгоритми та контейнери.

– **Низькорівневий доступ до пам’яті;**

C++ дозволяє прямий доступ до пам’яті, що корисно для оптимізації та роботи з апаратними ресурсами.

– **Багатопотоковість;**

Мова має вбудовану підтримку багатопотокового програмування.

– **Стандартна бібліотека.**

C++ має багатий набір функцій та класів у своїй стандартній бібліотеці.

У наукових статтях та дослідницьких роботах, які використовують мову C++, автори часто детально описують алгоритми, реалізації та результати своїх досліджень. Це дозволяє іншим вченим розуміти та використовувати ці знання для подальших досліджень.

Таким чином, мова C++ є потужним інструментом для розробки програмного забезпечення, а її використання в наукових дослідженнях сприяє розвитку інформаційних технологій та наукового прогресу.

3.2.4 Інтеграція C++ у Python

Модуль *ctypes* дозволить нам викликати функції з бібліотеки C++ безпосередньо з Python. В Windows можна завантажити бібліотеки, які експортують функції зі стандартною угодою про виклик *cdecl*, використовуючи об’єкт *cdll*. Якщо бібліотека використовує угоду про виклик *stdcall*, то використовуйте об’єкт *windll*. Наприклад, для завантаження бібліотеки *kernel32.dll* в Windows.

```
from ctypes import cdll, windll
kernel32 = windll.kernel32
```

На Linux потрібно вказати ім’я файлу бібліотеки разом з розширенням .so. Можна використовувати конструктор *CDLL*.

```
from ctypes import CDLL
libc = CDLL("libc.so.6")
```

Доступ до функцій здійснюється як атрибути об’єктів *DLL*.

```
from ctypes import libc
printf_func = libc.printf
```

Для створення більш зручного інтерфейса для використання функцій з бібліотеки C++, можна розглянути створення Python-обгортки за допомогою інструментів, таких як *SIP* або *pybind11*. Це дозволить викликати функції з бібліотеки C++ безпосередньо з Python, використовуючи більш Python-подібний синтаксис.

3.2.5 Знаходження максимальної гучності у записаної аудіо доріжки

З мікрофона збирається 8-бітний PCM звук з частотою дискретизації 8 кГц [22]. PCM звук не має стиснення, але з-за маленької частоти дискретизації якість звуку є маленькою. Також з-за цієї частоти семпли не досягають точок екстремума хвилі, тому є деякий відхил у гучності.

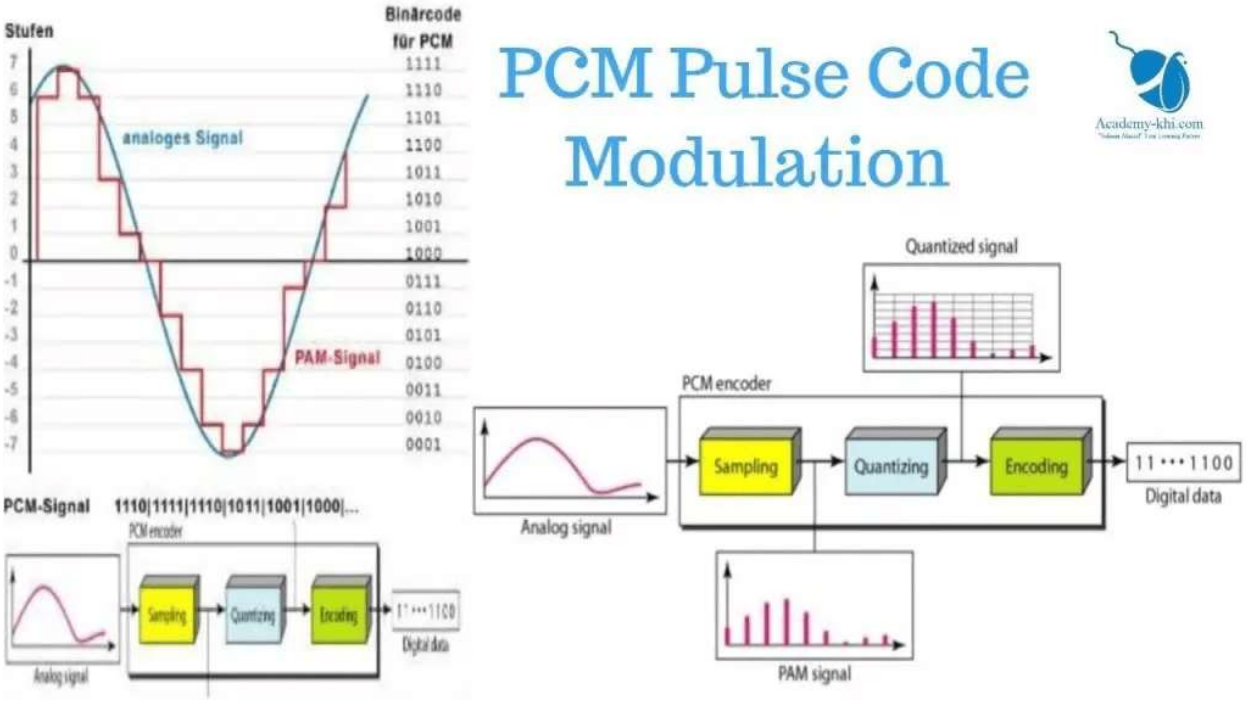


Рисунок 3.4– Опис PCM

Рекомендовано використовувати частоту дискретизації 44100 Гц, так як максимальна частота звука 22000 Гц і подвійна цієї частоті частота дискретизації ідеально описує хвилю звука, плюс 100 Гц додається з-за затухання хвилі при дискретизації.

3.2.6 Зворотнє геокодування

Термін геокодування зазвичай означає перетворення зручної для читання людини адреси в місце на карті. Процес перетворення місця на карті в зрозумілу людині адресу відомий як зворотне геокодування [23].

Обов’язкові параметри

latlng – координати широти та довготи, які визначають місце, для якого потрібна найближча адреса, зручна для читання людиною.

key – ключ *API* виданий для сервера цієї системи. Цей ключ ідентифікує сервер для роботи з *Google Maps API*.

Приклад HTTP запиту GET типу

```
https://maps.googleapis.com/maps/api/geocode/json?latlng=40.714224,-73.961452&key=YOUR_API_KEY
```

3.2.7 Python скрипти

Для цієї дипломної роботи написано три головних скрипта:

– **ServerGUI.py**;

Це *front-end* у цієї дипломної роботі. Цей скрипт зв’язується з *БД*. З колекцій *firstmessage* та *secondmessage* беруться повідомлення які згенеровані мікроконтролерами та оброблені та закинуті у *БД* скриптом *TCPServer*. Адміністратором системи формуються трикутники по розмітці на мапі. Відносно цих трикутників та повідомлень з мікроконтролерів обраховуються координати та формуються пости які завантажуються у *БД*.

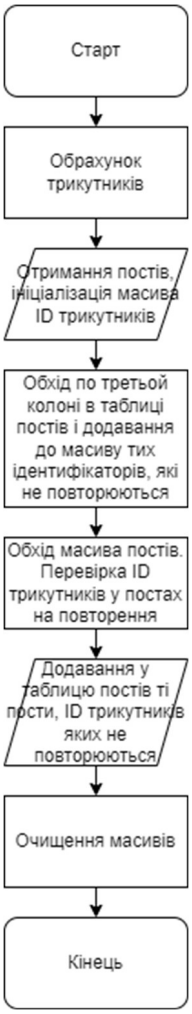


Рисунок 3.5 – Алгоритм функції оновлення таблиці постів

– **TCPServer.py**;

Це *back-end* цього проєкта. У скрипту реалізован асінхроний *TCP* сервер з використанням *asyncio*. У функції-хендлері для кожного сокета

оброблюються повідомлення з використанням самописного *DLL cppForServer* який використовується для парсинга та десеріалізації структури з даними від мікроконтролерів.

```
async def handle_client(reader, writer):
    logging.info("Client connected")
    addr = writer.get_extra_info('peername')
    msgParser = MsgParser()
    dataBuffer = bytearray()
    logging.info("Start reading")
    data = await reader.read(16384)
    dataBuffer.extend(data)
    cUByteArray = (c_ubyte * len(dataBuffer)).from_buffer_copy(dataBuffer)
    msgParser.setMessage(cUByteArray)
    testIter = 0
    for i in cUByteArray:
        logging.info(str(testIter) + " " + str(i))
        testIter += 1
    logging.info("Data parsing")
    msgParser.parsing()
    logging.info("Converting data...")
    if msgParser.getSize() < 80020:
        logging.info("Creating timestamp")
        timestamp = strftime("%Y-%m-%d %H:%M:%S", gmtime())
        logging.info("Parsing JSON")
        tmpdict = json.loads(msgParser.getJSONString())
        tmpdict['timestamp'] = timestamp
        logging.info("Data formated")
        tmp = coll1.insert_one(tmpdict)
        logging.info("Data added to DB")
    else:
        logging.info("Creating timestamp")
        timeInSec = time() - 5
        timestamp = strftime("%Y-%m-%d %H:%M:%S", gmtime(timeInSec))
        audio = []
        logging.info("Extracting audio")
        msgParser.getAudio(audio)
        logging.info("Uploading audio")
        DBFunctions.uploadDataAsFile(audio, fs, timestamp + ".mp3")
        logging.info("Finding audio ID")
        fileID = DBFunctions.findFileIdInDB(mydb, fs, timestamp + ".mp3")
        logging.info("Parsing JSON")
        tmpdict = json.loads(msgParser.getJSONString())
        tmpdict['timestamp'] = timestamp
        tmpdict['audioID'] = fileID
        logging.info("Data formated")
        tmp = coll2.insert_one(tmpdict)
        logging.info("Data added to DB")
    writer.close()
```

Лістинг 3.1 – Асинхрона функція-хендлера для сокета, яка використовує програмний інтерфейс *asyncio* та *ctypes*

– **testMsg.py**.

Цей скрипт тестує *TCP* сервер підключаясь до нього та відправляя два типа повідомлень. Для цього використовується зазначена самописна *DLL*.

```
hashTag = c_ubyte(35) # знак '#' у ASCII

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    logging.info("Socket created")
    s.connect((HOST, PORT))
    logging.info("Socket connected")
    for i in range(1):
        logging.info("Creating first message")
        firstMessage = libcpp.getFirstMessageInstance()
        firstMsgBA = (c_ubyte * (firstMessageSize + 8))()
        firstMsgPointer = pointer(firstMessage)
        libcpp.FirstMessageToByteArray(firstMsgPointer,       cast(firstMsgBA,
c_void_p))
        for j in range(firstMessageSize, firstMessageSize + 8):
            firstMsgBA[j] = hashTag
        logging.info("First message was formated")
        s.send(firstMsgBA)
        logging.info("First message was sent")

        time.sleep(5)

        logging.info("Creating second message")
        secondMessage = libcpp.getSecondMessageInstance()
        secondMsgBA = (c_ubyte * (secondMessageSize + 8))()
        secondMsgPointer = pointer(secondMessage)
        libcpp.SecondMessageToByteArray(secondMsgPointer,    cast(secondMsgBA,
c_void_p))
        for j in range(secondMessageSize, secondMessageSize + 8):
            secondMsgBA[j] = hashTag
        logging.info("Second message was formated")
        s.send(secondMsgBA)
        logging.info("Second message was sent")
```

Лістинг 3.2 – Код для тестування сервера

3.2.8 Python модулі

– **MsgParser.py**;

Цей модуль використовує самописну *DLL* та займається обробкою, десеріалізацією повідомлень та експортом аудіо з повідомлення другого типу.

```
def parsing(self):
        specialSymNum = 0
        endFlag = False
        logging.info("Parsing started")
        iter = 0
        k = 0
```

Парсинг повідомлення з мікроконтролера.

```
for i in self._msg:
    logging.info("Iteration: " + str(iter))
    self._tmp.append(i)
    if len(self._tmp) > 8:
        for j in range(len(self._tmp)-8, len(self._tmp)):
            if self._tmp[j] == 35: # знак '#' у ASCII
                specialSymNum += 1
        if specialSymNum != 8:
            logging.info("Special symbols: " + str(specialSymNum))
            specialSymNum = 0
    if specialSymNum == 8:
        logging.info("Special symbols: " + str(specialSymNum))
        endFlag = True
        for k in range(8):
            logging.info("Popping array")
            self._tmp.pop()
            logging.info(str(k))
        if k == 7:
            iter -= 8
    logging.info(str(self._tmp[iter]))
    iter += 1
    if endFlag == True:
        break
```

Ініціалізація змінних для конвертації повідомлень.

```
iter = 0
logging.info(str(len(self._tmp)))
logging.info("Creating array")
cTypeArray = (c_ubyte * len(self._tmp))()
```

Визначення типу повідомлення, конвертація повідомлення, та експорт аудіо з повідомлення другого типу.

```
logging.info("Copying array")
for m in range(0, len(self._tmp)):
    logging.info("Iteration: " + str(m))
    cTypeArray[m] = self._tmp[m]
    logging.info(str(cTypeArray[m]))
logging.info("Array copied")
if len(cTypeArray) < 80020:
    logging.info("Creating first message JSON")
    firstMsg = self._libCPP.ByteArrayToFirstMessage(cTypeArray)
    tmpChar = self._libCPP.FirstMessageCString(pointer(firstMsg))
    self._res = tmpChar.decode()
    logging.info("JSON created")
    self._size = 52
else:
    logging.info("Creating second message JSON")
    secondMsg = self._libCPP.ByteArrayToSecondMessage(cTypeArray)
    tmpChar = self._libCPP.SecondMessageCString(pointer(secondMsg))
    self._res = tmpChar.decode()
    logging.info("JSON created")
    self._libCPP.ExtractAudioFromSecondMessage(pointer(secondMsg),
self._audio)
    logging.info("Audio extracted")
    self._size = 80020
```

Лістинг 3.3 – Алгоритм функції парсингу повідомлень з мікроконтролера

– **DBFunctions.py**;

У цьому модулі знаходяться самописні функції для роботи з MongoDB.

```
import pymongo
import gridfs

def uploadDataAsFile(data, fs, fileName):
    fs.put(bytes(data), filename=fileName)

def findFileIdInDB(db, fs, fileName):
    data = db.files.files.find_one({"filename": fileName})
    return data['_id']
```

Лістинг 3.4 – Функції для роботи з файлами у БД

– **cCoordsAndDb.py**;

Це *ctypes* реалізація структури, яка використовується у повідомленнях з мікроконтролера.

– **cFirstMessage.py**;

Це *ctypes* реалізація структури повідомлення першого типу. Ця структура використовується функціями DLL.

– **cSecondMessage.py**;

Це *ctypes* реалізація структури повідомлення другого типу. Ця структура використовується функціями DLL.

– **Triangle.py**;

У цьому модулі реалізований клас трикутника, вертекси якого є датчики звуку та/або мікрофони.

– **Post.py**;

У цьому модулі реалізований клас поста, який вже буде зберігатись у БД як фінальний об’єкт системи.

– **TriangleController.py**;

Цей модуль реалізує клас-предок для класа *Server* з *Server.py*. Клас має функції для роботи з трикутниками.

– **PostController.py**.

Цей модуль реалізує клас-предок для класу *Server* з *Server.py*. Клас має функції для конвертації повідомлень та трикутників у пости. При генерації постів проходить обрахунок координат та зворотний геокодінг для того, щоб дізнатись адрес.

Функція конвертації повідомлень у *dictionary* для мікрофона завдяки його ідентифікатору. Ця функція робить зручнішим і стандартізованішим роботу з даними.

```
def findVertexByID(self, id):
        tmpMsg1 = self.coll1.find_one({"mp1ID": id}, sort=[("_id", -1)])
        tmpMsg2 = self.coll1.find_one({"mp2ID": id}, sort=[("_id", -1)])
        tmpMsg3 = self.coll1.find_one({"mp3ID": id}, sort=[("_id", -1)])
        tmpMsg4 = self.coll2.find_one({"mp4ID": id}, sort=[("_id", -1)])
        if tmpMsg1 != None:
            return {
            "mpString": "mp1ID",
            "timestamp": tmpMsg1["timestamp"],
            "mpX": tmpMsg1["mp1CAndDb"]["x"],
            "mpY": tmpMsg1["mp1CAndDb"]["y"],
            "mpDb": tmpMsg1["mp1CAndDb"]["db"]
            }
        elif tmpMsg2 != None:
            return {
            "mpString": "mp2ID",
            "timestamp": tmpMsg2["timestamp"],
            "mpX": tmpMsg2["mp2CAndDb"]["x"],
            "mpY": tmpMsg2["mp2CAndDb"]["y"],
            "mpDb": tmpMsg2["mp2CAndDb"]["db"]
            }
        elif tmpMsg3 != None:
            return {
            "mpString": "mp3ID",
            "timestamp": tmpMsg3["timestamp"],
            "mpX": tmpMsg3["mp3CAndDb"]["x"],
            "mpY": tmpMsg3["mp3CAndDb"]["y"],
            "mpDb": tmpMsg3["mp3CAndDb"]["db"]
            }
        elif tmpMsg4 != None:
            return {
            "mpString": "mp4ID",
            "timestamp": tmpMsg4["timestamp"],
            "mpX": tmpMsg4["mp4CAndDb"]["x"],
            "mpY": tmpMsg4["mp4CAndDb"]["y"],
            "mpDb": tmpMsg4["mp4CAndDb"]["db"],
            "audioID": tmpMsg4["audioID"]
            }
```

Лістинг 3.5 – Функція пошуку даних від датчиків за ідентифікатором

Функція конвертації трикутника у пост.

```
def triangleToPost(self, triangle):
        firstVertex = self.findVertexByID(int(triangle._firstVertex))
        secondVertex = self.findVertexByID(int(triangle._secondVertex))
        thirdVertex = self.findVertexByID(int(triangle._thirdVertex))
        tmpPost = self.coll3.find_one({}, sort=[('_id', -1)])
        postID = None
        if tmpPost:
            postID = tmpPost["_postID"] + 1
        else:
            postID = 1
        timestamp = firstVertex["timestamp"]
        triangleID = int(triangle._triangleID)
        lal = self.calculateLatitudeAndLongitude()
        latitude = lal["latitude"]
        longitude = lal["longitude"]
        address = self.coordsToAddress(lal)
        audioID = None
        if firstVertex["mpString"] == "mp4ID":
            audioID = firstVertex["audioID"]
        elif secondVertex["mpString"] == "mp4ID":
            audioID = secondVertex["audioID"]
        elif thirdVertex["mpString"] == "mp4ID":
            audioID = thirdVertex["audioID"]
        tmpRetPost = Post.Post(postID, timestamp, triangleID, latitude,
longitude, address, audioID)
        tmp                                                                    =
self.coll3.insert_one(json.loads(json_util.dumps(vars(tmpRetPost))))
        return tmpRetPost
```

Лістинг 3.6 – Функція конвертації трикутника у пост

3.2.9 Динамічна бібліотека на C++

Ця бібліотека написана для роботи з C++ класами. Вона має функції серіалізації та десеріалізації класів. Також вона робить з класів JSON, генерує тестові об’єкти та експортує аудіо у вигляді масива байт. Бібліотека використовується для тестування та роботи з даними з мікроконтролера. Прошивка мікроконтролера має схожі функції.

Функція конвертує повідомлення першого типу у масив *unsigned char*. Потрібно для відправки даних

Рисунок 3.6 – Алгоритм конвертації повідомлення першого типу у масив *unsigned char*

Функція конвертує масив *unsigned char* у повідомлення першого типу. Потрібно для роботи з даними.

Рисунок 3.7 – Алгоритм конвертації масива *unsigned char* у повідомлення першого типу

Функція конвертує повідомлення першого типу у JSON в масив *char*(*C string*). Потрібна для зручного зберігання даних.

```
extern "C" CPPFORSERVER_API const char* FirstMessageCString(FirstMessage* fm)
{
	json tmp = {
	  {"esp32ID", fm->esp32ID},
	  {"mp1ID", fm->mp1ID},
	  {"mp1CAndDb", {
	       {"x", fm->mp1CAndDb.x},
	       {"y", fm->mp1CAndDb.y},
	       {"db", fm->mp1CAndDb.db}
	  }},
	  {"mp2ID", fm->mp2ID},
	  {"mp2CAndDb", {
	       {"x", fm->mp2CAndDb.x},
	       {"y", fm->mp2CAndDb.y},
	       {"db", fm->mp2CAndDb.db}
	  }},
	  {"mp3ID", fm->mp3ID},
	  {"mp3CAndDb", {
	       {"x", fm->mp3CAndDb.x},
	       {"y", fm->mp3CAndDb.y},
	       {"db", fm->mp3CAndDb.db}
	  }}
	};
	return tmp.dump().c_str();
}
```

Лістинг 3.7 – Функція конвертації повідомлення першого типу у формат JSON в масив *char*(*C string*)

Функція конвертує повідомлення другого типу у масив *unsigned char*. Потрібно для відправки даних.

Рисунок 3.8 – Алгоритм функції конвертації масива *unsigned char* у повідомлення першого типу

Функція конвертує масив *unsigned char* у повідомлення другого типу. Потрібно для роботи з даними.

Рисунок 3.9 – Алгоритм функції конвертації масива *unsigned char* у повідомлення другого типу

Функція конвертує повідомлення другого типу у JSON в масив *char*(*C string*). Аудіо експортується іншої функцією.

```
extern "C" CPPFORSERVER_API const char* SecondMessageCString(SecondMessage* sm)
{
	json tmp = {
	  {"esp32ID", sm->esp32ID},
	  {"mp4ID", sm->mp4ID},
	  {"mp4CAndDb", {
	       {"x", sm->mp4CAndDb.x},
	       {"y", sm->mp4CAndDb.y},
	       {"db", sm->mp4CAndDb.db}
	  }}
	};
	return tmp.dump().c_str();
}
```

Лістинг 3.8 – Функція конвертації повідомлення другого типу у формат JSON в масив *char*(*C string*)

Функція експортує аудіо з повідомлення другого типу в масив *unsigned char* розміром 80 кілобайт.

```
	extern "C" CPPFORSERVER_API void ExtractAudioFromSecondMessage(SecondMessage* sm, unsigned char* byteArray)
	{
		memcpy(byteArray, sm->audio, 80000);
	}
```

Лістинг 3.9 – Функція експорту аудіохвилі з повідомлення

Функція генерує тестове повідомлення першого типу.

```
	extern "C" CPPFORSERVER_API FirstMessage getFirstMessageInstance()
	{
		FirstMessage fm;
		fm.esp32ID = 1;
		fm.mp1ID = 1;
		fm.mp1CAndDb.x = 1;
		fm.mp1CAndDb.y = 5;
		fm.mp1CAndDb.db = 90;
		fm.mp2ID = 2;
		fm.mp2CAndDb.x = 5;
		fm.mp2CAndDb.y = 10;
		fm.mp2CAndDb.db = 90;
		fm.mp3ID = 3;
		fm.mp3CAndDb.x = 10;
		fm.mp3CAndDb.y = 5;
		fm.mp3CAndDb.db = 90;
		return fm;
	}
```

Лістинг 3.10 – Функція генерації тестового повідомлення першого типу

Функція генерує тестове повідомлення другого типу.

```
	extern "C" CPPFORSERVER_API SecondMessage getSecondMessageInstance()
	{
		SecondMessage sm;
		sm.esp32ID = 1;
		sm.mp4ID = 4;
		sm.mp4CAndDb.x = 5;
		sm.mp4CAndDb.y = 5;
		sm.mp4CAndDb.db = 90;
		for(int i = 0; i < 80000; i++)
		{
			sm.audio[i] = 1;
		}
		return sm;
	}
```

Лістинг 3.11 – Функція генерації тестового повідомлення другого типу

3.2.10 Розробка програмного забезпечення для мікроконтролера

Це прошивка для мікроконтролера. Цей варіант прошивки написаний для варіанта кластера з трьома датчиками звуку, та одним мікрофоном.

ESP32.ino

На блок-схемі зображен алгоритм роботи мікроконтролера.

Рисунок 3.10 – Алгоритм роботи прошивки

Рисунок 3.11 – Алгоритм функції зчитування аналогового сигнала з датчиків звука

Функція зчитує звук з мікрофона 5 секунд і записує результат у *unsigned char* масив.

```
      void readMicrophone(SecondMessage* smP, unsigned char* audio)
      {
            size_t bytes_read = 0;
            i2s_read(I2S_PORT,  raw_samples,  sizeof(int8_t)  *  SAMPLE_BUFFER_SIZE,
&bytes_read, 5000U); // 5 секунд
            int samples_read = bytes_read / sizeof(int8_t); // int8_t = char
            for (int i = 0;i < samples_read; i++)
            {
          Serial.printf("%ld\n", raw_samples[i]);
            smP->audio[i] = static_cast<unsigned char>(raw_samples[i]);
            }
            //smP->mp4CAndDb.db = checkMaxWaveDb(raw_samples); // Не реалізовано
      }
```

Лістинг 3.12 – Функція зчитування звуку з мікрофона у форматі 5-ти секундної аудіохвилі у *unsigned char* масив

Підготування та відправка першого повідомлення на сервер.

```
void sendRequestOne() {
	readSoundSensors(fmP);
	FirstMessageToByteArray(fmP, fmBA);

	for(int i = 0;i < 8;i++)
	{
		fmBA[sizeof(FirstMessage)+i] = static_cast<unsigned char>(35); // 35 це # у таблиці ASCII
	}

	if (localClient.connect(ip, port))
	{
		if (localClient.connected())
		{
	  localClient.write(fmBA, sizeof(FirstMessage) + 8); // У хейдері написано typedef unsigned char uint8_t;
		}
	}
}
```

Лістинг 3.13 – Функція підготування та відправки першого повідомлення на сервер

Підготування та відправка другого повідомлення на сервер.

```
void sendRequestTwo() {
	SecondMessageToByteArray(smP, smBA);

	for(int i = 0;i < 8;i++)
	{
		smBA[sizeof(SecondMessage)+i] = static_cast<unsigned char>(35); // 35 це # у таблиці ASCII
	}

	if (localClient.connect(ip, port))
	{
		if (localClient.connected())
		{
	  localClient.write(smBA, sizeof(SecondMessage) + 8); // У хейдері написано typedef unsigned char uint8_t;
		}
	}
}
```

Лістинг 3.14 – Функція підготування та відправки другого повідомлення на сервер

3.3 Пропозиції для поліпшення системи

3.3.1 Варіант кластера без кабелів

На місцевості з великою урбанізацією раціонально не використовувати кабелі для передачі аналогового сигналу, а біля кожного датчика розташовувати свій ESP32 у місцях де є WiFi сигнал.

3.3.2 Варіант кластера з GSM модулем

Якщо місцевість має середню урбанізацію(головне наявність 3G), то має сенс використати варіант кластера без кабелів з використанням GSM модулів. За місяць весь кластер відправляє на сервер 106,272 гігабайт(99,75% трафіку займає аудіо з мікрофона).

3.3.3 Використання нейромережи

На майбутнє є можливість натренувати нейромережу на типізацію аудіо з мікрофона. Ось приклад типів гучного сигналу:

- крик;
- легковий транспорт;
- мототранспорт;
- грузовий транспорт;
- звуковий сигнал від транспорта(при натисканні на руль);
- постріл;
- грім;
- вітер;
- пуск гранати/ракети;
- вибух;
- свист ракети.

Висновки до розділа

В останньому розділі була розглянута реалізація прототипу кластера та сервера. Не було реалізовано мобільний додаток та аналіз гучності аудіозапису з-за браку часу, але ці пункти були частково описані теоретично. Був реалізован варіант кластеру з трьома датчиками, мікрофоном та кабелями для аналогового сигналу. Були представлені ідеї для поліпшення системи та розвитку варіативності кластерів. Загалом на програмну частину було витрачено найбільша кількість часу.

ВИСНОВКИ

Розробка відносно дешевого аналога апаратно-програмного комплексу для моніторінгу терористичної активності виявилась цілком реалізованою. Розробленої системі потрібні поліпшення, але вона вже виконує свою головну роботу.

Актуальність реалізації такої системи доводиться світовою статистикою та військовою ситуацією в Україні, тому що у купи українських громадян з’явилася зброя, а також лунають вибухи у всіх українських містах.

Крім того, ця система може бути модіфікована для використання для комунальних служб, швидкої допомоги, та пожежної служби.

Успішне впровадження цієї системи послужить стимулом у поліпшенні інфраструктури державних служб, а також збільшить загальну безпеку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. BBC: Euston shooting: Girl, 7, and five others injured near church. URL: https://www.bbc.com/news/uk-england-london-64277185 (Last accessed: 12.05.2024).

2. BBC News: Друга масова стрілянина у Каліфорнії за кілька днів – семеро загиблих. URL: https://www.bbc.com/ukrainian/news-64383486 (Last accessed: 12.05.2024).

3. SoundThinking URL: https://www.soundthinking.com/ (Last accessed: 12.05.2024).

4. System and method for archiving data from a sensor array US7411865B2 United States Applied on 23.12.2005; Published on 12.08.2008 URL:https://patents.google.com/patent/US7411865B2/en?assignee=Shotspotter%2c+Inc.

5. Systems and methods of identifying/locating weapon fire including return fire, targeting, laser sighting, and/or guided weapon features US7586812B2 United States Applied on 30.10.2007; Published on 08.09.2009 URL: https://patents.google.com/patent/US7586812B2/en?assignee=Shotspotter%2c+Inc
.

6. Systems and methods of automated correlation of weapon fire data with monitored persons-of-interest/location data US8351297B2 United States Applied on 08.04.2010; Published on 08.01.2013 URL: https://patents.google.com/patent/US8351297B2/en?assignee=Shotspotter%2c+Inc
.

7. Design of Internet Electrical Power Usage Metering System Based on ESP32 Mcu and Mysql Database with User Interface Application / A. Suryowinoto et al. International Conference on Advanced Engineering and Technology, Surabaya, Indonesia, 18 February 2023. 2023. URL: https://doi.org/10.5220/0012108700003680 (Last accessed: 12.05.2024).

8. Design and Implementation of ESP32-Based IoT Devices URL: https://www.researchgate.net/publication/372742618_Design_and_Implementation_of_ESP32-Based_IoT_Devices (Last accessed: 12.05.2024).

9. O'Higgins N. MongoDB and Python. Sebastopol, CA : O'Reilly Media, 2011.

10. International Competition of Student Scientific Works “Black Sea Science 2023”. Results for the field of “Information Technologies, Automation and Robotics”. URL: http://isc.ontu.edu.ua/wp-content/uploads/sites/50/2023/04/2023-1-IT.pdf (Last accessed: 07.06.2024).

11. UML 2002 (2002 Dresden, Germany). UML 2002-- the Unified Modeling Language: Model engineering, concepts, and tools : 5th International Conference, Dresden, Germany, September 30-October 4, 2002 : proceedings. Berlin : Springer, 2002. 447 p.

12. Що таке веб-додаток? | Блог WEBCASE URL: https://webcase.com.ua/uk/blog/cho-takoe-web-prilozhenie-vse-vidy/ (Last accessed: 12.05.2024).

13. Bonde P., Jharia B., K. Shrivastav A. Wi-TCP: A TCP in Wireless Environment. International Journal of Computer Applications. 2013. Vol. 73, no. 8. P. 30–34. URL: https://doi.org/10.5120/12763-9731 (Last accessed: 12.05.2024).

14. Get Started | Geocoding API | Google for Developers URL: https://developers.google.com/maps/documentation/geocoding/start#reverse (Last accessed 12.05.2024)

15. Smith B. Serving JSON. Beginning JSON. Berkeley, CA, 2015. P. 159–189. URL: https://doi.org/10.1007/978-1-4842-0202-9_10 (date of access: 12.05.2024). Electrical Circuits. *Students quarterly journal*. 1967. Vol. 37, no. 148. P. 242. URL: https://doi.org/10.1049/sqj.1967.0054 (Last accessed: 06.05.2024).

16. GridFS / D. Hows et al. The Definitive Guide to MongoDB. Berkeley, CA, 2015. P. 91–101. URL: https://doi.org/10.1007/978-1-4842-1182-3_5 (Last accessed: 12.05.2024).

17. What is Agile? | Atlassian URL: https://www.atlassian.com/agile (Last accessed: 12.05.2024).

18. Pyforms GUI documentation! – Pyforms GUI 4.9.2 documentation URL: https://pyforms.readthedocs.io/projects/Pyforms-GUI/en/v4/ (Last accessed: 12.05.2024).

19. Cameron N. I2S Audio. ESP32 Formats and Communication. Berkeley, CA, 2023. P. 55–122. URL: https://doi.org/10.1007/978-1-4842-9376-8_2 (Last accessed: 12.05.2024).

20. Android SDK. Wearable Android™. Hoboken, NJ, 2015. P. 87–109. URL: https://doi.org/10.1002/9781119051091.ch4 (Last accessed: 12.05.2024).

21. Інтенсивність і рівень звуку URL: https://ukrayinska.libretexts.org/%D1%84%D1%96%D0%B7%D0%B8%D0%BA%D0%B8/%D0%A3%D0%BD%D1%96%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%82%D0%B5%D1%82%D1%81%D1%8C%D0%BA%D0%B0_%D1%84%D1%96%D0%B7%D0%B8%D0%BA%D0%B0/%D0%9A%D0%BD%D0%B8%D0%B3%D0%B0%3A_%D0%A4%D1%96%D0%B7%D0%B8%D0%BA%D0%B0_(Boundless)/16%3A_%D0%97%D0%B2%D1%83%D0%BA/16.2%3A_%D0%86%D0%BD%D1%82%D0%B5%D0%BD%D1%81%D0%B8%D0%B2%D0%BD%D1%96%D1%81%D1%82%D1%8C_%D1%96_%D1%80%D1%96%D0%B2%D0%B5%D0%BD%D1%8C_%D0%B7%D0%B2%D1%83%D0%BA%D1%83 (Last accessed: 12.05.2024).

22. Pulse Code Modulation (PCM) NETWORK ENCYCLOPEDIA URL: https://networkencyclopedia.com/pulse-code-modulation-pcm/ (Last accessed: 14.06.2024)

23. Reverse Geocoding | Maps JavaScript API | Google for Developers URL:https://developers.google.com/maps/documentation/javascript/examples/geocoding-reverse (Last accessed: 14.06.2024)

ДОДАТОК А

Довідка

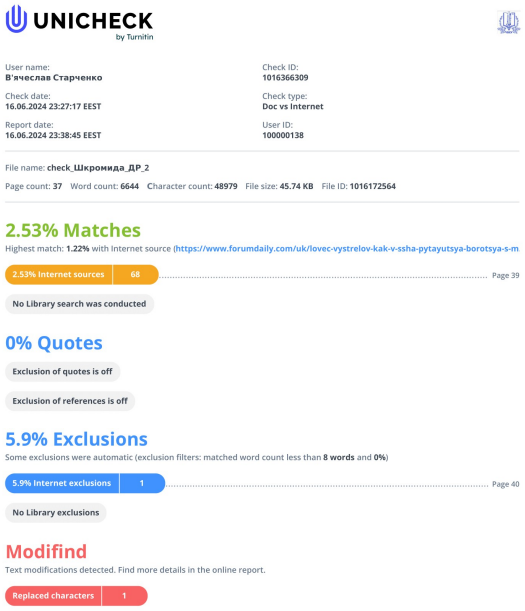
про перевірку на унікальність пояснювальної записки

бакалаврської кваліфікаційної роботи на тему:
«Апаратно-програмний комплекс
для моніторингу терористичної активності»

студента спеціальності 123 «Комп’ютерна інженерія», 405 групи
Шкромида Віталій Олексійович
прізвище, ім’я, по-батькові

Перевірку тексту здійснено сервісом: онлайн-сервіс Unicheck

Результат перевірки тексту бакалаврської кваліфікаційної роботи: схожість складає 2,53%.



Здобувач:

_____В. О. Шкромида
підпис ініціали, прізвище

Керівник:

ст. викладач

_____В. В. Старченко
підпис ініціали, прізвище

Дата: «___» _____ 2024 р.

ДОДАТОК Б

Код для мікроконтролера

```
#include <Arduino.h>
#include <WiFi.h>
#include <driver/i2s.h>

#define I2S_MIC_CHANNEL I2S_CHANNEL_FMT_RIGHT_LEFT // Стерео звук
#define I2S_MIC_SERIAL_CLOCK GPIO_26
#define I2S_MIC_LEFT_RIGHT_CLOCK GPIO_22
#define I2S_MIC_SERIAL_DATA GPIO_21
#define SAMPLE_BUFFER_SIZE 80000
#define SAMPLE_RATE 8000 // За стандартом GSM

WiFiClient localClient; // Це TCP сокет для зв’язку з сервером

const char* ssid = "..."; // Назва точки доступу
const char* password = "..."; // Пароль до точки доступу

const uint port = 9876; // Порт TCP-сервера
const char* ip = "45.79.112.203"; // IP-адреса сервера

struct CAD
{
	float x;
	float y;
	float db;
};

struct FirstMessage
{
	int esp32ID;
	int mp1ID;
	CAD mp1CAndDb;
	int mp2ID;
	CAD mp2CAndDb;
	int mp3ID;
	CAD mp3CAndDb;
};

struct SecondMessage
{
	int esp32ID;
	int mp4ID;
	CAD mp4CAndDb;
	unsigned char audio[80000];
};

FirstMessage fm;
fm.esp32ID = 1;
fm.mp1ID = 1;
fm.mp1CAndDb.x = 1;
fm.mp1CAndDb.y = 5;
fm.mp1CAndDb.db = 1;
fm.mp2ID = 2;
fm.mp2CAndDb.x = 5;
fm.mp2CAndDb.y = 10;
fm.mp2CAndDb.db = 1;
fm.mp3ID = 3;
fm.mp3CAndDb.x = 10;
```

```
fm.mp3CAndDb.y = 5;
fm.mp3CAndDb.db = 1;

SecondMessage sm;
sm.esp32ID = 1;
sm.mp4ID = 4;
sm.mp4CAndDb.x = 5;
sm.mp4CAndDb.y = 5;
sm.mp4CAndDb.db = 1;.

unsigned char fmBA[sizeof(FirstMessage)+8];
unsigned char smBA[sizeof(SecondMessage)+8];
FirstMessage* fmP = &fm;
SecondMessage* smP = &sm;


int FirstSS = A0;
int SecondSS = A1;
int ThirdSS = A2;

const i2s_port_t I2S_PORT = I2S_NUM_0;

i2s_config_t i2s_config =
      {
    .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX),
    .sample_rate = SAMPLE_RATE,
    .bits_per_sample = I2S_BITS_PER_SAMPLE_8BIT, // 8 біт бітової глибини
    .channel_format = I2S_MIC_CHANNEL,
    .communication_format = I2S_COMM_FORMAT_I2S,
    .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
    .dma_buf_count = 4,
    .dma_buf_len = SAMPLE_BUFFER_SIZE,
    .use_apll = false,
    .tx_desc_auto_clear = false,
    .fixed_mclk = 0
      };

      i2s_pin_config_t i2s_mic_pins =
      {
    .bck_io_num = I2S_MIC_SERIAL_CLOCK,
    .ws_io_num = I2S_MIC_LEFT_RIGHT_CLOCK,
    .data_out_num = I2S_PIN_NO_CHANGE,
    .data_in_num = I2S_MIC_SERIAL_DATA
      };

void setup() {
      pinMode(FirstSS, INPUT);
      pinMode(SecondSS, INPUT);
      pinMode(ThirdSS, INPUT);

      Serial.begin(9600);
      i2s_driver_install(I2S_PORT, &i2s_config, 0, NULL);
      i2s_set_pin(I2S_PORT, &i2s_mic_pins);


      Serial.println("Connect Wlan");
      WiFi.begin(ssid, password);

      while (WiFi.status() != WL_CONNECTED) {
            delay(500);
            Serial.print(".");
      }
```

```
	Serial.println(WiFi.localIP());
}

int8_t raw_samples[SAMPLE_BUFFER_SIZE];

void loop() {
	sendRequestOne();
	readMicrophone(smP, audioArray);
	sendRequestTwo();
}

void readSoundSensors(FirstMessage* fmP)
{
	fmP->mp1CAndDb.db = analogRead(FirstSS);
	fmP->mp2CAndDb.db = analogRead(SecondSS);
	fmP->mp3CAndDb.db = analogRead(ThirdSS);
}

//float checkMaxWaveDb(int8_t* r_s); // Не реалізовано

void readMicrophone(SecondMessage* smP, unsigned char* audio)
{
	size_t bytes_read = 0;
	i2s_read(I2S_PORT,  raw_samples,  sizeof(int8_t)  *  SAMPLE_BUFFER_SIZE,
&bytes_read, 5000U); // 5 секунд
	int samples_read = bytes_read / sizeof(int8_t);
	for (int i = 0;i < samples_read; i++)
	{
    Serial.printf("%ld\n", raw_samples[i]);
	smP->audio[i] = static_cast<unsigned char>(raw_samples[i]);
	}
	//smP->mp4CAndDb.db = checkMaxWaveDb(raw_samples); // Не реалізовано
}

int FirstMessageToByteArray(FirstMessage* fm, unsigned char* byteArray)
{
	memcpy(byteArray,	reinterpret_cast<unsigned	char*>(fm),
sizeof(FirstMessage));
	return sizeof(FirstMessage);
}

int SecondMessageToByteArray(SecondMessage* sm, unsigned char* byteArray)
{
	memcpy(byteArray,	reinterpret_cast<unsigned	char*>(sm),
sizeof(SecondMessage));
	return sizeof(SecondMessage);
}

void sendRequestOne() {
	readSoundSensors(fmP);
	FirstMessageToByteArray(fmP, fmBA);

	for(int i = 0;i < 8;i++)
	{
		fmBA[sizeof(FirstMessage)+i] = static_cast<unsigned char>(35); //
35 це # у таблиці ASCII
	}

	if (localClient.connect(ip, port))
	{
		if (localClient.connected())
```

```
                {
              localClient.write(fmBA, sizeof(FirstMessage) + 8); // У хейдері написано
typedef unsigned char uint8_t;
                }
            }
      }

      void sendRequestTwo() {
            SecondMessageToByteArray(smP, smBA);

            for(int i = 0;i < 8;i++)
            {
                  smBA[sizeof(SecondMessage)+i] = static_cast<unsigned char>(35); //
35 це # у таблиці ASCII
            }

            if (localClient.connect(ip, port))
            {
                  if (localClient.connected())
                  {
              localClient.write(smBA,  sizeof(SecondMessage)  +  8);  //  У  хейдері
написано typedef unsigned char uint8_t;
                  }
            }
      }
```

ДОДАТОК В

Код для desktop серверу

TCPServer.py

```
import functools
import socket
import sys
from pymongo import *
import gridfs
from time import gmtime, strftime, time
from ctypes import *
from MsgParser import MsgParser
import json
import DBFunctions
import asyncio
import logging
import signal

def ask_exit(signame, loop):
     print("got signal %s: exit" % signame)
     loop.stop()

async def handle_client(reader, writer):
    logging.info("Client connected")
    addr = writer.get_extra_info('peername')
    msgParser = MsgParser()
    dataBuffer = bytearray()
    logging.info("Start reading")
    data = await reader.read(16384)
    dataBuffer.extend(data)
    cUByteArray = (c_ubyte * len(dataBuffer)).from_buffer_copy(dataBuffer)
    msgParser.setMessage(cUByteArray)
    testIter = 0
    for i in cUByteArray:
        logging.info(str(testIter) + " " + str(i))
        testIter += 1
    logging.info("Data parsing")
    msgParser.parsing()
    logging.info("Converting data...")
    if msgParser.getSize() < 80020:
        logging.info("Creating timestamp")
```

```
            timestamp = strftime("%Y-%m-%d %H:%M:%S", gmtime())
            logging.info("Parsing JSON")
            tmpdict = json.loads(msgParser.getJSONString())
            tmpdict['timestamp'] = timestamp
            logging.info("Data formated")
            tmp = coll1.insert_one(tmpdict)
            logging.info("Data added to DB")
        else:
            logging.info("Creating timestamp")
            timeInSec = time() - 5
            timestamp = strftime("%Y-%m-%d %H:%M:%S", gmtime(timeInSec))
            audio = []
            logging.info("Extracting audio")
            msgParser.getAudio(audio)
            logging.info("Uploading audio")
            DBFunctions.uploadDataAsFile(audio, fs, timestamp + ".mp3")
            logging.info("Finding audio ID")
            fileID = DBFunctions.findFileIdInDB(mydb, fs, timestamp + ".mp3")
            logging.info("Parsing JSON")
            tmpdict = json.loads(msgParser.getJSONString())
            tmpdict['timestamp'] = timestamp
            tmpdict['audioID'] = fileID
            logging.info("Data formated")
            tmp = coll2.insert_one(tmpdict)
            logging.info("Data added to DB")
        writer.close()

async def runServer():
    logging.info("Entered in runServer()")
    server = await asyncio.start_server(handle_client, host, port)
    logging.info("Server created")
    await server.serve_forever()



if __name__ == '__main__':
    global host, port, myclient, mydb, fs, coll1, coll2
    host = "127.0.0.1"
    port = 9876
    myclient = MongoClient(
```

```
"mongodb+srv://vitaliyskromyda:test@clusterdb.4wu0t0a.mongodb.net/?retryWrites=true&w
=majority&appName=clusterdb",
                port=27017)
            mydb = myclient["clusterdb"]
            fs = gridfs.GridFS(mydb, collection="files")
            coll1 = mydb["firstmsg"]
            coll2 = mydb["secondmsg"]
            if sys.platform == 'win32':

asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())
            logging.basicConfig(level=logging.DEBUG,          filename="TCPServer.log",
filemode="w"
                                , format="%(asctime)s – %(levelname)s – %(message)s")
            logging.info("Starting TCPServer")
            asyncio.run(runServer(), debug = True)
            logging.info("TCPServer started")
```

MsgParser.py

```
        import sys
        from ctypes import *
        import cCoordsAndDb
        import cFirstMessage
        import cSecondMessage
        import logging

        class MsgParser():
            def __init__(self):
                self._msg = None
                self._tmp = []
                self._res = None
                self._audio = (c_ubyte * 80000)()
                cdll.LoadLibrary("./cppForServer.dll")
                self._libCPP = CDLL("./cppForServer.dll")
                logging.info("Library loaded")
                self._libCPP.ByteArrayToFirstMessage.restype                          =
cFirstMessage.cFirstMessage
                self._libCPP.FirstMessageCString.restype = c_char_p
                self._libCPP.ByteArrayToSecondMessage.restype                         =
cSecondMessage.cSecondMessage
                self._libCPP.SecondMessageCString.restype = c_char_p
                self._libCPP.ExtractAudioFromSecondMessage.restype = None
```

```
            logging.info("Results type set")
            self._libCPP.ByteArrayToFirstMessage.argtypes = [c_void_p]
            self._libCPP.FirstMessageCString.argtypes =
[POINTER(cFirstMessage.cFirstMessage)]
            self._libCPP.ByteArrayToSecondMessage.argtypes = [c_void_p]
            self._libCPP.SecondMessageCString.argtypes =
[POINTER(cSecondMessage.cSecondMessage)]
            self._libCPP.ExtractAudioFromSecondMessage.argtypes =
[POINTER(cSecondMessage.cSecondMessage), c_void_p]
            logging.info("Library set")

        def setMessage(self, msg):
            self._msg = msg

        def parsing(self):
            specialSymNum = 0
            endFlag = False
            logging.info("Parsing started")
            iter = 0
            k = 0

            for i in self._msg:
                logging.info("Iteration: " + str(iter))
                self._tmp.append(i)
                if len(self._tmp) > 8:
                    for j in range(len(self._tmp)-8, len(self._tmp)):
                        if self._tmp[j] == 35:
                            specialSymNum += 1
                    if specialSymNum != 8:
                        logging.info("Special symbols: " + str(specialSymNum))
                        specialSymNum = 0
                if specialSymNum == 8:
                    logging.info("Special symbols: " + str(specialSymNum))
                    endFlag = True
                    for k in range(8):
                        logging.info("Popping array")
                        self._tmp.pop()
                        logging.info(str(k))
                    if k == 7:
                        iter -= 8
                logging.info(str(self._tmp[iter]))
```

```
                iter += 1
                if endFlag == True:
                    break

            iter = 0
            logging.info(str(len(self._tmp)))
            logging.info("Creating array")
            cTypeArray = (c_ubyte * len(self._tmp))()

            logging.info("Copying array")
            for m in range(0, len(self._tmp)):
                logging.info("Iteration: " + str(m))
                cTypeArray[m] = self._tmp[m]
                logging.info(str(cTypeArray[m]))
            logging.info("Array copied")
            if len(cTypeArray) < 80020:
                logging.info("Creating first message JSON")
                firstMsg = self._libCPP.ByteArrayToFirstMessage(cTypeArray)
                tmpChar = self._libCPP.FirstMessageCString(pointer(firstMsg))
                self._res = tmpChar.decode()
                logging.info("JSON created")
                self._size = 52
            else:
                logging.info("Creating second message JSON")
                secondMsg = self._libCPP.ByteArrayToSecondMessage(cTypeArray)
                tmpChar = self._libCPP.SecondMessageCString(pointer(secondMsg))
                self._res = tmpChar.decode()
                logging.info("JSON created")
                self._libCPP.ExtractAudioFromSecondMessage(pointer(secondMsg),
self._audio)
                logging.info("Audio extracted")
                self._size = 80020

        def getSize(self):
            return self._size

        def getTempMessage(self):
            return self._tmp

        def getJSONString(self):
            return self._res
```

```
        def getAudio(self, audioArray):
            for l in range(0, 80000):
                audioArray.append(self._audio[l])
```

TriangleController.py

```
    import pickle

    class TriangleController(object):

        def __init__(self):
            self._triangles = []

        def addTriangle(self, triangle):
            self._triangles.append(triangle)

        def removeTriangle(self, index):
            return self._triangles.pop(index)

        def save(self, filename):
            output = open(filename, 'wb')
            pickle.dump(self._triangles, output)

        def load(self, filename):
            pkl_file = open(filename, 'rb')
            self._triangles = pickle.load(pkl_file)

        def getTriangles(self):
            return self._triangles
```

Server.py

```
import pyforms
from pyforms.basewidget import BaseWidget
from pyforms.controls   import ControlText
from pyforms.controls   import ControlButton
from pyforms.controls   import ControlList
from pyforms import start_app
from Triangle           import Triangle
from TriangleController import TriangleController
from PostController     import PostController
import asyncio
import logging

class Server(BaseWidget, TriangleController, PostController):
    def __init__(self, *args, **kwargs):
        logging.info("Initializing")
        BaseWidget.__init__(self, 'Server')
        TriangleController.__init__(self)
```

```
        PostController.__init__(self)
        logging.info("Class Server initialized")

        self._tmpTriangle     = Triangle('-1', '-1', '-1', '-1')
        self._triangleID    = ControlText('Triangle ID')
        self._firstVertex    = ControlText('First vertex ID')
        self._secondVertex    = ControlText('Second vertex ID')
        self._thirdVertex    = ControlText('Third vertex ID')
        self._list        = ControlList('Triangles List')
        self._removeButton     = ControlButton('Remove')
        self._addButton     = ControlButton('Add')
        self._changeButton     = ControlButton('Change')
        self._importButton     = ControlButton('Import dump')
        self._exportButton     = ControlButton('Export dump')

        self._postList        = ControlList('Post List')
        self._refresh = ControlButton('Refresh')

        self._list.horizontal_headers = ['Triangle ID', 'Vertex 1', 'Vertex 2', 'Vertex
3']
        self._postList.horizontal_headers = ['Post ID', 'Timestamp', 'Triangle ID',
'Latitude', 'Longitude', 'Address', 'Audio ID']
        self._postList.readonly = True
        self._removeButton.value       = self.__removeEvent
        self._changeButton.value       = self.__changeEvent
        self._addButton.value       = self.__addEvent
        self._refresh.value = self.__refreshMessages

        self._formset = [{
            'a:Triangles':[('_triangleID',      '_firstVertex',      '_secondVertex',
'_thirdVertex'),
            ('_removeButton', '_changeButton', '_addButton'),
            ('_list', ('_importButton', '=', '_exportButton'))],
            'b:Messages':['_postList', '_refresh']
        }]

        #eventServerInitialized.set()

    def __refreshMessages(self):
        logging.info("Start refreshing messages")
        super(Server, self).calculateTriangles(super(Server, self).getTriangles())
        posts = super(Server, self).getPosts()
        trianglesIDs = []
        for i in range(0, self._postList.rows_count-1):
            trianglesIDs.append(self._postList.get_value(3, i))
        for post in posts:
            if post._triangleID not in trianglesIDs:
                self._postList += [post._postID, post._timestamp, post._triangleID,
post._latitude, post._longitude, post._address, post._audioID]
        trianglesIDs.clear()
        posts.clear()
        logging.info("End refreshing messages")

    def addTriangleToList(self, triangle):
        super(Server, self).addTriangle(triangle)
        self._list      +=      [triangle._triangleID,      triangle._firstVertex,
triangle._secondVertex, triangle._thirdVertex]

    def removeTriangleFromList(self, index):
        super(Server, self).removeTriangle(index)
        self._list -= index
```

```
    def __changeEvent(self):
        self.removeTriangleFromList( self._list.selected_row_index )

    def __addEvent(self):
        self.addTriangleToList(Triangle(
            self._triangleID.value,
            self._firstVertex.value,
            self._secondVertex.value,
            self._thirdVertex.value
        ))

    def __removeEvent(self):
        self.removeTriangleFromList( self._list.selected_row_index )

#async def waitInit():
#    await eventServerInitialized.wait()
#    await asyncio.sleep(3)

async def main():
    logging.info("Entered in main()")
    blockingTask = asyncio.to_thread(start_app, Server)
    task1 = asyncio.to_thread(waitInit)
    logging.info("Started ServerGUI thread")
    await task1
    task2 = asyncio.create_task(Server.__refreshMessages())
    await task2
    await blockingTask

if __name__ == '__main__':
    global eventServerInitialized
    eventServerInitialized = asyncio.Event()
    logging.basicConfig(level=logging.DEBUG, filename="serverGUI.log", filemode="w",
                        format="%(asctime)s – %(levelname)s – %(message)s")
    logging.info("Starting ServerGUI")
    start_app(Server)
    #asyncio.run(main(), debug = True)
```

PostController.py

```
import pickle
import Post
from pymongo import *
import json
from bson import json_util
#import pymongo

class PostController(object):

    def __init__(self):
        self._posts = []
        self._coefficientOne = 1
        self._coefficientTwo = 1
        self._coefficientThree = 1
        self.myclient                                                    =
MongoClient("mongodb+srv://vitaliyskromyda:test@clusterdb.4wu0t0a.mongodb.net/?retryW
rites=true&w=majority&appName=clusterdb", port=27017)
        self.mydb = self.myclient["clusterdb"]
        self.coll1 = self.mydb["firstmsg"]
        self.coll2 = self.mydb["secondmsg"]
        self.coll3 = self.mydb["post"]

    def addPost(self, post):
        self._posts.append(post)
```

```
    def removePost(self, index):
        return self._posts.pop(index)

    def save(self, filename):
        output = open(filename, 'wb')
        pickle.dump(self._posts, output)

    def load(self, filename):
        pkl_file = open(filename, 'rb')
        self._posts = pickle.load(pkl_file)

    def findVertexByID(self, id):
        tmpMsg1 = self.coll1.find_one({"mp1ID": id}, sort=[("_id", -1)])
        tmpMsg2 = self.coll1.find_one({"mp2ID": id}, sort=[("_id", -1)])
        tmpMsg3 = self.coll1.find_one({"mp3ID": id}, sort=[("_id", -1)])
        tmpMsg4 = self.coll2.find_one({"mp4ID": id}, sort=[("_id", -1)])
        if tmpMsg1 != None:
            return {
            "mpString": "mp1ID",
            "timestamp": tmpMsg1["timestamp"],
            "mpX": tmpMsg1["mp1CAndDb"]["x"],
            "mpY": tmpMsg1["mp1CAndDb"]["y"],
            "mpDb": tmpMsg1["mp1CAndDb"]["db"]
            }
        elif tmpMsg2 != None:
            return {
            "mpString": "mp2ID",
            "timestamp": tmpMsg2["timestamp"],
            "mpX": tmpMsg2["mp2CAndDb"]["x"],
            "mpY": tmpMsg2["mp2CAndDb"]["y"],
            "mpDb": tmpMsg2["mp2CAndDb"]["db"]
            }
        elif tmpMsg3 != None:
            return {
            "mpString": "mp3ID",
            "timestamp": tmpMsg3["timestamp"],
            "mpX": tmpMsg3["mp3CAndDb"]["x"],
            "mpY": tmpMsg3["mp3CAndDb"]["y"],
            "mpDb": tmpMsg3["mp3CAndDb"]["db"]
            }
        elif tmpMsg4 != None:
            return {
            "mpString": "mp4ID",
            "timestamp": tmpMsg4["timestamp"],
            "mpX": tmpMsg4["mp4CAndDb"]["x"],
            "mpY": tmpMsg4["mp4CAndDb"]["y"],
            "mpDb": tmpMsg4["mp4CAndDb"]["db"],
            "audioID": tmpMsg4["audioID"]
            }

    def calculateLatitudeAndLongitude(self, firstVertex, secondVertex, thirdVertex):
        ra = self._coefficientOne * firstVertex["mpDb"]
        rb = self._coefficientTwo * secondVertex["mpDb"]
        rc = self._coefficientThree * thirdVertex["mpDb"]

        k_ab = (firstVertex["mpY"] - secondVertex["mpY"]) / (firstVertex["mpX"] -
secondVertex["mpX"])
        b_ab = (firstVertex["mpX"] * secondVertex["mpY"] - secondVertex["mpX"] *
firstVertex["mpY"]) / (firstVertex["mpX"] - secondVertex["mpX"])
        k_bc = (secondVertex["mpY"] - thirdVertex["mpY"]) / (secondVertex["mpX"] -
thirdVertex["mpX"])
```

```
        b_bc = (secondVertex["mpX"] * thirdVertex["mpY"] - thirdVertex["mpX"] *
secondVertex["mpY"]) / (secondVertex["mpX"] - thirdVertex["mpX"])

        l_ab = math.hypot(firstVertex["mpX"] - secondVertex["mpX"],
firstVertex["mpY"] - secondVertex["mpY"])
        r_ab = ra * (ra * ra - rb * rb + l_ab * l_ab) / (2 * ra * l_ab) / l_ab
        l_bc = math.hypot(secondVertex["mpX"] - thirdVertex["mpX"],
secondVertex["mpY"] - thirdVertex["mpY"])
        r_bc = rb * (rb * rb - rc * rc + l_bc * l_bc) / (2 * rb * l_bc) / l_bc

        xc_ab = r_ab * secondVertex["mpX"] + (1 - r_ab) * firstVertex["mpX"]
        yc_ab = r_ab * secondVertex["mpY"] + (1 - r_ab) * firstVertex["mpY"]
        xc_bc = r_bc * thirdVertex["mpX"] + (1 - r_bc) * secondVertex["mpX"]
        yc_bc = r_bc * thirdVertex["mpY"] + (1 - r_bc) * secondVertex["mpY"]

        k_ab_n = -(firstVertex["mpX"] - secondVertex["mpX"]) / (firstVertex["mpY"] -
secondVertex["mpY"])
        b_ab_n = yc_ab - k_ab_n * xc_ab
        k_bc_n = -(secondVertex["mpX"] - thirdVertex["mpX"]) / (secondVertex["mpY"] -
thirdVertex["mpY"])
        b_bc_n = yc_bc - k_bc_n * xc_bc

        ym = ((b_bc_n * k_ab_n) - (b_ab_n * k_bc_n)) / (k_ab_n - k_bc_n)
        xm = (ym - b_ab_n) / k_ab_n

        return {
            "latitude": xm,
            "longitude": ym
            }

    def triangleToPost(self, triangle):
        firstVertex = self.findVertexByID(int(triangle._firstVertex))
        secondVertex = self.findVertexByID(int(triangle._secondVertex))
        thirdVertex = self.findVertexByID(int(triangle._thirdVertex))
        tmpPost = self.coll3.find_one({}, sort=[('_id', -1)])
        postID = None
        if tmpPost:
            postID = tmpPost["_postID"] + 1
        else:
            postID = 1
        timestamp = firstVertex["timestamp"]
        triangleID = int(triangle._triangleID)
        lal = calculateLatitudeAndLongitude()
        latitude = lal[“latitude”]
        longitude = lal[“longitude”]
        address = "test"
        audioID = None
        if firstVertex["mpString"] == "mp4ID":
            audioID = firstVertex["audioID"]
        elif secondVertex["mpString"] == "mp4ID":
            audioID = secondVertex["audioID"]
        elif thirdVertex["mpString"] == "mp4ID":
            audioID = thirdVertex["audioID"]
        tmpRetPost = Post.Post(postID, timestamp, triangleID, latitude, longitude,
address, audioID)
        tmp = self.coll3.insert_one(json.loads(json_util.dumps(vars(tmpRetPost))))
        return tmpRetPost

    def calculateTriangles(self, triangles):
        for triangle in triangles:
            self._posts.append(self.triangleToPost(triangle))
```

```
    def getPosts(self):
        tmp = self._posts
        return tmp
```

cppForServer.dll

```
#pragma once
#ifdef CPPFORSERVER_EXPORTS
#define CPPFORSERVER_API __declspec(dllexport)
#else
#define CPPFORSERVER_API __declspec(dllimport)
#endif
#include "pch.h"
#include "winfunctions.h"
using json = nlohmann::json;

extern "C" CPPFORSERVER_API int FirstMessageToByteArray(FirstMessage * fm, unsigned
char* byteArray)
{
	memcpy(byteArray, reinterpret_cast<unsigned char*>(fm), sizeof(FirstMessage));
	return sizeof(FirstMessage);
}

extern "C" CPPFORSERVER_API FirstMessage ByteArrayToFirstMessage(unsigned char*
byteArray)
{
	FirstMessage firstMessage = *(reinterpret_cast<FirstMessage*>(byteArray));
	return firstMessage;
}

extern "C" CPPFORSERVER_API const char* FirstMessageCString(FirstMessage* fm)
{
	json tmp = {
	  {"esp32ID", fm->esp32ID},
	  {"mp1ID", fm->mp1ID},
	  {"mp1CAndDb", {
	      {"x", fm->mp1CAndDb.x},
	      {"y", fm->mp1CAndDb.y},
	      {"db", fm->mp1CAndDb.db}
	  }},
	  {"mp2ID", fm->mp2ID},
	  {"mp2CAndDb", {
	      {"x", fm->mp2CAndDb.x},
	      {"y", fm->mp2CAndDb.y},
	      {"db", fm->mp2CAndDb.db}
	  }},
	  {"mp3ID", fm->mp3ID},
	  {"mp3CAndDb", {
	      {"x", fm->mp3CAndDb.x},
	      {"y", fm->mp3CAndDb.y},
	      {"db", fm->mp3CAndDb.db}
	  }}
	};
	return tmp.dump().c_str();
}

extern "C" CPPFORSERVER_API int SecondMessageToByteArray(SecondMessage* sm, unsigned
char* byteArray)
{
	memcpy(byteArray, reinterpret_cast<unsigned char*>(sm), sizeof(SecondMessage));
	return sizeof(SecondMessage);
}
```

```
extern "C" CPPFORSERVER_API SecondMessage ByteArrayToSecondMessage(unsigned char*
byteArray)
{
	SecondMessage secondMessage = *(reinterpret_cast<SecondMessage*>(byteArray));
	return secondMessage;
}

extern "C" CPPFORSERVER_API const char* SecondMessageCString(SecondMessage* sm)
{
	json tmp = {
	  {"esp32ID", sm->esp32ID},
	  {"mp4ID", sm->mp4ID},
	  {"mp4CAndDb", {
	      {"x", sm->mp4CAndDb.x},
	      {"y", sm->mp4CAndDb.y},
	      {"db", sm->mp4CAndDb.db}
	  }}
	};
	return tmp.dump().c_str();
}

extern "C" CPPFORSERVER_API void ExtractAudioFromSecondMessage(SecondMessage* sm,
unsigned char* byteArray)
{
	memcpy(byteArray, sm->audio, 80000);
}

extern "C" CPPFORSERVER_API FirstMessage getFirstMessageInstance()
{
	FirstMessage fm;
	fm.esp32ID = 1;
	fm.mp1ID = 1;
	fm.mp1CAndDb.x = 1;
	fm.mp1CAndDb.y = 5;
	fm.mp1CAndDb.db = 90;
	fm.mp2ID = 2;
	fm.mp2CAndDb.x = 5;
	fm.mp2CAndDb.y = 10;
	fm.mp2CAndDb.db = 90;
	fm.mp3ID = 3;
	fm.mp3CAndDb.x = 10;
	fm.mp3CAndDb.y = 5;
	fm.mp3CAndDb.db = 90;
	return fm;
}

extern "C" CPPFORSERVER_API SecondMessage getSecondMessageInstance()
{
	SecondMessage sm;
	sm.esp32ID = 1;
	sm.mp4ID = 4;
	sm.mp4CAndDb.x = 5;
	sm.mp4CAndDb.y = 5;
	sm.mp4CAndDb.db = 90;
	for(int i = 0; i < 80000; i++)
	{
		sm.audio[i] = 1;
	}
	return sm;
}
```