

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**КАФЕДРА СИСТЕМОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ**

«На правах рукопису»
УДК _____

«До захисту допущено»
Завідувач кафедри СПіСКС

_____ В.П.Тарасенко
(підпис) (ініціали, прізвище)
“ ” _____ 2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія
(Системне програмування)

на тему: Локалізація об'єкта за акустичним сигналом в розподілених сенсорних мережах

Виконав (-ла): студент (-ка) II курсу, групи КВ-62м
(шифр групи)

Книш Петро Костянтинович
(прізвище, ім'я, по батькові)

_____ (підпис)

Науковий керівник Доцент кафедри СПіСКС, к.т.н., доц. Замятін Д.С.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) _____ (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) _____ (підпис)

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2018 року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність 123 Комп'ютерна інженерія

(Системне програмування)

ЗАТВЕРДЖУЮ

Завідувач кафедри СПСКС

В.П.Тарасенко

(підпис) (ініціали, прізвище)

«__» _____ 2018р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Книш Петро Костянтинович

(прізвище, ім'я, по батькові)

1. Тема дисертації Локалізація об'єкта за акустичним сигналом в розподілених сенсорних мережах

науковий керівник дисертації Замятін Денис Станіславович, к.т.н, доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «22» березня 2018 р. №986-с

2. Термін подання студентом дисертації 11 травня 2018 р.

3. Об'єкт дослідження реалізація моделі локалізації джерела акустичного сигналу в розподіленій сенсорній мережі за допомогою сервера, який отримує дані за протоколом UDP

4. Предмет дослідження обмеження локалізації сервера, що приймає повідомлення за протоколом UDP та використовує метод локалізації метод TDOA

5. Перелік завдань, які потрібно розробити розробка серверу, дослідження роботи моделі локалізації, аналіз отриманих результатів

6. Перелік ілюстративного матеріалу діаграма класів серверу, діаграма класів базової моделі, алгоритм роботи базової моделі, алгоритм роботи моделі з сервером, алгоритм роботи серверу, діаграма залежності модулів

7. Перелік публікацій ПМК 2018 «Локалізація об'єкта за акустичним сигналом в розподілених сенсорних мережах», САІТ 2018 «Визначення місцеположення джерела акустичного сигналу»

8. Дата видачі завдання 5 вересня 2016 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
	Вивчення літератури за тематикою проекту	6.09.2016	
	Розроблення та узгодження технічного завдання	12.12.2016	
	Аналіз існуючих рішень	1.04.2017	
	Підготовка матеріалів першого розділу магістерської дисертації	23.04.2017	
	Підготовка матеріалів другого розділу магістерської дисертації	20.08.2017	
	Підготовка матеріалів третього розділу магістерської дисертації	21.10.2017	
	Підготовка матеріалів четвертого розділу дипломного проекту	20.01.2018	
	Підготовка графічної частини дипломного магістерської дисертації	20.02.2018	
	Оформлення документації магістерської дисертації	23.03.2018	
	Попередній розгляд магістерської дисертації на кафедрі	26.04.2018	

Студент

_____ (підпис)

Книш П.К.

_____ (ініціали, прізвище)

Науковий керівник дисертації

_____ (підпис)

Зам'ятін Д.С.

_____ (ініціали, прізвище)

РЕФЕРАТ

Актуальність теми. Задача локалізації джерела сигналу є базовою в багатьох спеціалізованих системах. До неї зводиться чимало інших задач, наприклад, при побудові охоронних систем, при визначенні місцеположення абоненту мобільного зв'язку та у військових цілях. Серед відомих методів локалізації джерела акустичного сигналу було обрано TDOA (Time Difference of Arrival), який дозволяє створити модель з високою точністю визначення джерела звуку і створити ефективну реалізацію для застосування у сенсорних мережах. Актуальність дослідження розподілених сенсорних систем є надзвичайно високою, оскільки вони повсякчас використовуються в перерахованих системах, завдяки низькій вартості сенсорів.

Об'єктом дослідження є методи проектування розподілених сенсорних мереж для локалізації джерела акустичного сигналу.

Предметом дослідження є методи реалізацій розподілених сенсорних мереж для локалізації джерела акустичного сигналу.

Мета роботи: розробка моделі, головною частиною є сервер на який в реальному часі надсилаються дані з мікрофонів за допомогою протоколу UDP та відбувається локалізація за допомогою методу TDOA.

Наукова новизна:

1. Розроблено метод опрацювання даних з найбільшої кількості мікрофонів у системі локалізації.
2. Розроблено програмну реалізацію методу TDOA, що дозволяє ефективно опрацювати дані з великої кількості мікрофонів.

Практична цінність отриманих в роботі результатів полягає в тому, що розроблені способи можуть бути застосовані у нових або існуючих системах локалізації джерела акустичного сигналу. У свою чергу,

розроблена програмна реалізація запропонованих способів може бути використана для вирішення реальних задач локалізації джерела акустичного сигналу.

Апробація роботи. Основні положення і результати роботи будуть представлені та обговорюватимуться на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2018 (Київ, 21-23 березня 2018 р.) та САІТ-2018.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи, наведено відомості про апробацію результатів і їхнє впровадження.

У першому розділі розглядається опис предметної області досліджень та обґрунтування магістерської дисертації.

У другому розділі містить опис модифікації методу TDOA.

У третьому розділі розглядається реалізація розподіленого методу TDOA.

У четвертому розділі описані експерименти зі створеною системою.

У висновках представлені результати проведеної роботи.

Робота представлена на 80 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: розподілені сенсорні мережі, локалізація, Python, TDOA.

ABSTRACT

Topicality. The task of localizing the source of the signal is basic in many specialized systems. To it there are many other tasks to be built up, for example, in the construction of security systems, in determining the location of a subscriber for mobile communications and for military purposes. Among known methods of localization of the source of the acoustic signal was chosen TDOA (Time Difference of Arrival), which allows you to create a model with high accuracy to determine the source of sound and create an effective implementation for use in sensor networks. The relevance of the study of distributed sensor systems is extremely high, since they are always used in the listed systems, due to the low cost of sensors.

The object of study are methods of designing distributed sensor networks for locating the source of an acoustic signal.

The study examines methods of distributed sensor networks implementation for localization of the acoustic signal source.

Objective: to develop the model, the main part is the server on which the data from the microphones is sent in real time via the UDP protocol and the localization is carried out using the TDOA method.

Scientific innovation is as follows:

1. A method of processed data from the largest number of microphones in the localization system is developed.
2. Software implementation of the TDOA method is developed, which allows to efficiently process data from a large number of microphones.

Practical value obtained in the results is that developed techniques may be applied to new or already existing localization systems. At the same time, program implementation of those methods can be used for solving real localization tasks

Testing of work. The main provisions and results will be presented and discussed at a scientific conference undergraduates and graduate students “Applied mathematics and computing”, AMC-2017 (Kyiv, 21-23 March 2018).

The structure and scope of work. Master's thesis consists of an introduction, four chapters and conclusions.

The introduction presents the general characteristics of the work, done assessment of the current state of the problem, the urgency towards research, formulated the purpose and objectives of research, the scientific novelty of the results and practical value of work, provides information on testing results and their implementation.

The first section contains description of subject area of research and substantiation of master's dissertation.

The second section shows implementation of TDOA method.

The third section contains implementation of distributed TDOA method.

The fourth section describes experiments with the created system.

In conclusion, the findings of the work.

Work submitted 80 pages, containing a link to a list of used literature.

Keywords: distributed sensor networks, localization, Python, TDOA.

РЕФЕРАТ

Актуальность темы. Задача локализации источника сигнала является базовой во многих специализированных системах. К ней сводятся многие другие задачи, например, при построении охранных систем, при определении местоположения абонента мобильной связи и в военных целях. Среди известных методов локализации источника акустического сигнала был избран TDOA (Time Difference of Arrival), который позволяет создать модель с высокой точностью определения источника звука и создать эффективную реализацию для применения в сенсорных сетях. Актуальность исследования распределенных сенсорных систем чрезвычайно высока, поскольку они постоянно используются в перечисленных системах, благодаря низкой стоимости сенсоров.

Объектом методы проектирования распределенных сенсорных сетей для локализации источника акустического сигнала.

Предметом методы реализаций распределенных сенсорных сетей для локализации источника акустического сигнала.

Цель работы: разработка модели, главной частью является сервер на который в реальном времени направляются данные с микрофонов с помощью протокола UDP и происходит локализация с помощью метода TDOA.

Научная новизна заключается в следующем:

1. Разработан метод опрацювання данных с наибольшим количеством микрофонов в системе локализации.

2. Разработана программная реализация метода TDOA, что позволяет эффективно обрабатывать данные из большого количества микрофонов.

Практическая ценность заключается в том, что разработанные способы могут быть применены в новых или существующих системах локализации источника акустического сигнала. В свою очередь, разработана

программная реализация предложенных способов может быть использована для решения реальных задач локализации источника акустического сигнала.

Апробация работы. Основные положения и результаты работы будут представлены и будут обсуждаться на научной конференции магистрантов и аспирантов «Прикладная математика и компьютеринг» ПМК-2017 (Киев, 21-23 марта 2018) и САИТ-2018.

Структура и объем работы. Магистерская диссертация состоит из введения, четырех глав и выводов.

Во введении представлена общая характеристика работы, произведена оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулированы цели и задачи исследований, показано научную новизну полученных результатов и практическую ценность работы, приведены сведения об апробации результатов и их внедрение.

В первом разделе рассматривается описание предметной области исследований и обоснования магистерской диссертации.

Во втором разделе рассматривается реализация метода TDOA.

В третьем разделе описана реализация распределенного метода TDOA.

В четвертом разделе описаны эксперименты с созданной системой.

В выводах представлены результаты проведенной работы.

Работа представлена на 80 листах, содержит ссылки на список использованных литературных источников.

Ключевые слова: распределенные сенсорные сети, локализация, Python, TDOA.

Зміст

ПЕРЕЛІК ТЕРМІНІВ ТА УМОВНИХ ПОЗНАЧЕНЬ.....	3
ВСТУП.....	4
1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕНЬ ТА ОБГРУНТУВАННЯ МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ	5
1.1. Сенсорні мережі.....	5
1.2. Метод TOA	7
1.3. Метод TDOA	10
1.4. Порівняння методів TOA та TDOA	13
1.5. Висновок.....	14
2. МОДИФІКАЦІЯ МЕТОДУ ВИЗНАЧЕННЯ КООРДИНАТ ОБ'ЄКТУ ...	15
2.1 Реалізація методу TDOA.....	15
2.2 Реалізація розподіленого методу TDOA.....	22
2.3 Висновки.....	24
3. ОПИС РОЗРОБЛЕНОЇ МОДЕЛІ МЕРЕЖІ.....	26
3.1 Технологічні та програмні засоби розробки.....	26
3.2 Опис серверу.....	35
3.3 Опис клієнта.....	35
3.4 Опис взаємодії сервера та клієнта.....	38
3.5 Висновки.....	39
4. ЕКСПЕРИМЕНТИ ЗІ СТВОРЕНОЮ МОДЕЛЛЮ.....	40
4.1 Засоби для експериментів та тестування.....	40
4.2 Умови та результати.....	40
4.3 Аналіз отриманих результатів.....	65
4.4 Висновки.....	73
ВИСНОВКИ.....	75
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	78
ДОДАТКИ	

ДОДАТОК 1. Копії графічних матеріалів

ДОДАТОК 2. Лістинг програм

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

GPS – Global Positioning System – система глобального позиціонування

TDOA – Time Difference of Arrival – техніка визначення координат

TOA – Time of Arrival – техніка визначення координат

ВСТУП

У сучасному світі визначення географічних координат є повсякденною та водночас не тривіальною задачею. Під час одного дня відбуваються мільйони визначень географічних координат у майже всіх сферах життєдіяльності людини для сотень мільйонів людей. Така сильна необхідність почала виникати в останні десятиліття зі стрімким розвитком мобільних пристроїв з одного боку та росту кількості прикладних застосувань для них що потребують визначення місцезнаходження користувача з іншого.

В той же час зі зростанням кількості запитів на визначення координат сервісам що виконують дану процедуру постала необхідність постійно поліпшувати швидкодію та оптимізувати використання ресурсів для виконання даної операції, щоб надавати користувачам найкращу швидкість.

Зі зростанням потреби у визначенні місцезнаходження почали виникати нові алгоритми та покращені алгоритми для локалізації що дали змогу покращити швидкість знаходження об'єкту при використанні меншої кількості ресурсів. В той же час почався бурхливий розвиток веб технологій, що дозволив ще більше оптимізувати даний процес. Важливу роль у розвитку даної технології також відіграв неспинний розвиток апаратного забезпечення для даних систем.

Для виконання поставленої задачі була створена велика кількість алгоритмів, що базуються на різних методиках визначення географічних координат та мають різну складність і використовуються з врахуванням наявних апаратних можливостей.

Дана робота присвячена розробці розподіленої реалізації та дослідженню одного з існуючих методів обраної техніки визначення географічних координат та використанні його в сенсорних мережах.

1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕНЬ ТА ОБГРУНТУВАННЯ МАГІСТЕРСЬКОЇ РОБОТИ

1.1. Сенсорні мережі

Сенсорна мережа складається з крихітних пристроїв, що зазвичай живляться від батарей, та бездротової інфраструктури, яка відслідковує та записує умови в будь-якій кількості середовищ - від фабрики до центру обробки даних, лабораторії лікарні, полі бою та навіть у дикій природі. Мережа датчиків підключається до Інтернету, корпоративної глобальної мережі або локальної мережі або спеціалізованої промислової мережі, щоб зібрані дані могли передаватися до систем для аналізу та використовуватися в додатках. Після того як компетенція вчених та інших фахівців, мережа бездротових датчиків дозріла протягом останніх 10-ти років, до того часу, коли їх було порівняно легко встановити та використовувати в усьому світі. Датчики можуть контролювати температуру, тиск, світло та коливання, наприклад, годуючи компанії багатством оперативної розвідки, на якій вони можуть вживати заходів. У сценаріях, що використовуються, наведені такі приклади, як:

- постачання ланцюга постачання та логістика;
- промислове відстеження та видимість;
- розпізнавання місця та безпека;
- розпізнавання джерел акустичних сигналів;
- управління ресурсами центрів обробки даних.

З даними, зібраними з сенсорної мережі, підприємство може збільшити свою спритність, одночасно покращуючи операції та стаючи більш ефективними - все це за відносно низької вартості. Мережеві менеджери повинні бути готовими підтримувати сенсорні мережі та забезпечувати безперебійну зв'язок між ними, а також корпоративну мережеву інфраструктуру та архітектуру додатків. IT-фахівці також повинні вивчити, як сенсорна мережа може бути корисною для власних операцій,

наприклад, у центрі обробки даних для моніторингу споживання енергії або через корпоративну WAN для збору інформації про стан пристрою чи умови.

Розуміння сенсорних мереж починається з самих пристроїв, часто ідентифікаторів радіочастотної ідентифікації (RFID) та датчиків стану. Ці невеликі, малопотужні пристрої можуть бути розгорнуті практично в будь-якому місці, залишені для моніторингу та збору даних на певному просторі, об'єкті або взаємодії між об'єктами або між об'єктом та його середовищем. Пристрої з'єднуються через мережу бездротової мережі IEEE 802.15.4 з передачею даних безпосередньо на сервер шлюзу або від одного вузла до іншого, а потім до точки виходу[1].

Схеми мереж ранніх поколінь були зосереджені на ZigBee, специфічній технології бездротової мережної мережі на основі стандарту 802.15.4. ZigBee — бездротовий стандарт передачі даних. Підтримується і розвивається однойменним альянсом ZigBeeTM, який був створений в 2002 році з метою об'єднання зусиль з розроблення найефективніших протоколів і забезпечення сумісності пристроїв різних виробників. В міру удосконалення стандарту, альянс публікує на своєму сайті (www.zigbee.org) специфікації стандарту, опис профілів програмного забезпечення та інші нормативні документи. ZigBee — стандарт для набору високорівневих протоколів зв'язку, що використовують невеликі, малопотужні цифрові приймачі, заснований на стандарті IEEE 802.15.4-2006 для бездротових персональних мереж, таких як, наприклад, бездротові навушники, що з'єднані з мобільними телефонами за допомогою радіохвиль короткохвильового діапазону. Технологія визначається специфікацією ZigBee, яка розроблена з метою бути простішою та дешевшою, ніж інші персональні мережі, такі як Bluetooth. ZigBee призначений для мобільних пристроїв, де необхідна тривала робота від батарей і безпечність передачі даних у мережі.

Мережі ZigBee є мережами з самоорганізуванням та самовідновленням, оскільки ZigBee пристрої при вмиканні живлення,

завдяки вбудованому програмному забезпеченню, вміють самі знаходити один одного й формувати мережу, а у разі виходу з ладу котрогось із вузлів можуть встановлювати нові маршрути для передачі повідомлень.

У цьому сценарії шлюзовий пристрій полегшує передачу даних між мережею на базі ZigBee та IP-мережею[2].

З сервером IP-сервера вузли сенсора можуть взаємодіяти безпосередньо з іншими IP-пристроями, будь то в мережі бездротової мережі або іншої бездротової або дротової мережі, локальної або в будь-якій іншій частині Інтернету. Через мережу датчиків, фахівці з інформаційних технологій отримують прямий доступ в режимі реального часу до вузлів сенсорів і, імовірно, можливість керувати та захищати вузли, як і інші IP-пристрої[1].

1.1 Метод TOA

Time of arrival (TOA або ToA), іноді також називають Time of flight (TOF), час досягнення радіосигналу від одного передавача на віддалений приймач.

У порівнянні з технікою TDOA, ToA використовує абсолютний час прибуття на певній базовій станції, а не різницю виміряного часу між відступаючи від одного і які прибувають на іншій станції. Відстань може бути безпосередньо розрахована з часу моменту прибуття, так як сигнали передаються з відомою швидкістю. Час прибуття даних з двох базових станцій будуть звужувати положення до кола; Дані з третьої базової станції потрібно вирішити точне положення в одній точці. Багато системи радіолокації, в тому числі GPS, використовують ToA.

Як і в разі TDOA, синхронізація базової станції мережі з базовими станціями має важливе значення. Ця синхронізація може бути зроблено різними способами:

- При використанні точного синхроімпульсу з обох сторін. Неточність в синхронізації годинників перекладається безпосередньо в неточному місці.

- За двома сигналами, які мають різну частоту, отже, різну швидкість. Звук в діапазоні до удару блискавки працює таким чином (швидкість світла і швидкість звуку).
- За допомогою вимірювання до або спрацьовування від загальної опорної точки.
- Без прямої синхронізації, але з компенсацією різниць фаз.

Системи, які вимірюють час приходу сигналів можуть точно визначити місцезнаходження сигналів в 2- або 3 - мірному просторі і часі. Серед іншого, цей підхід є основою для системи глобального позиціонування (GPS), яка дає змогу створювати революційні технологічні, соціальні та наукові розробки. Час приходу сигналу (TOA), також використовуються в локалізації стільникових телефонів, для сейсмологічних досліджень.

Системи TOA в основному вирішують рівняння швидкість дорівнює часу відстань, $v \cdot t = d$, або більш конкретно, $v\delta t = \delta l$, де

$\delta t = (t_i - t)$ різниця між часом прибуття t_i в точці i джерелом за час t , і δl є відстанню між місцем розташування при вимірюванні x_i, y_i, z_i і місцем розташування джерела x, y, z . Таким чином, з теореми Піфагора, отримуємо, що

$$v(t_i - t) = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2}. \quad (1)$$

Вимірювання t_i в 4-х або більше місцях досить для визначення 4 невідомих x, y, z, t . Приклад вимірювання можна побачити на рисунку 1.1.

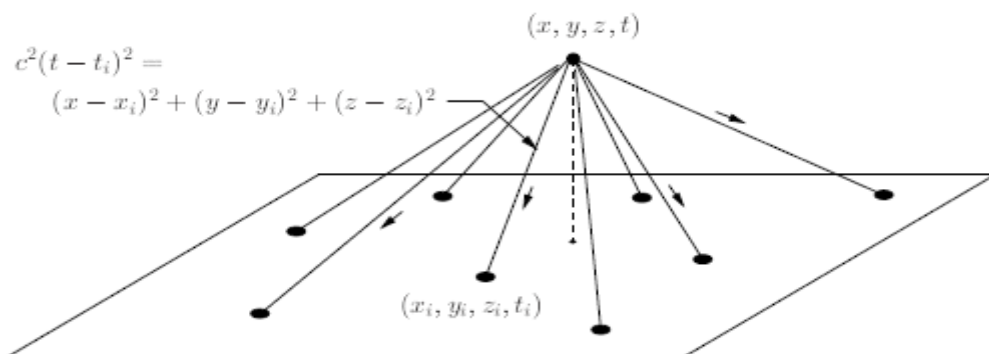


Рисунок 1.1 - Вимірювання в TOA

Для простоти ТОА буде розглядатися 2-вимірному випадку, в якому джерело і вимір місцеположення лежать в тій же площині z . Для цього випадку, $z_i = z$ та (1) стає

$$v(t_i - t) = \sqrt{(x_i - x)^2 + (y_i - y)^2} . \quad (2)$$

Три невідомих x , y , z можуть бути визначені з вимірів в 3-х різних локаціях. Порядок, в якому вимірювання часу прибуття може визначити джерело може бути визначено графічно з того, що різниця в часи прибуття на парі станцій i , j створюють обмеження для джерела, що воно повинно знаходитися на гіперболоїді, що визначає базову лінію між двома станціями. Це можна побачити, розглядаючи основні властивості еліпсів і гіпербол. Еліпс має властивість, що сума відстаней від двох фокусів еліпса є константою

$$d_1 + d_2 = 2a . \quad (3)$$

Рівняння еліпса показано та його використання показано на рисунку 1.2

$$\frac{y^2}{a^2} + \frac{x^2}{b^2} = 1 . \quad (4)$$

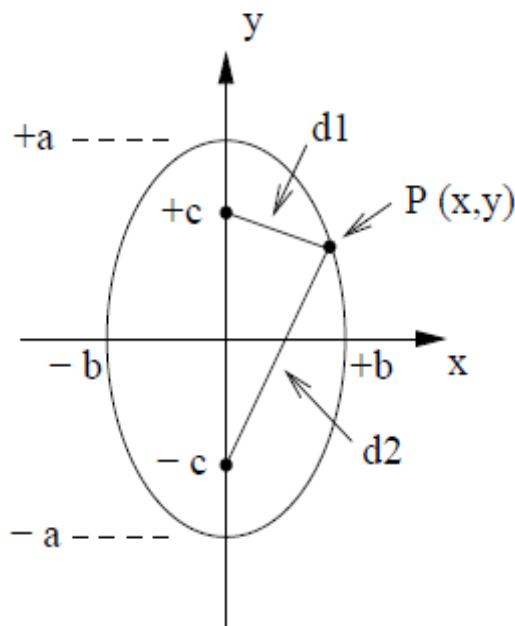


Рисунок 1.2 - Вимірювання за допомогою еліпса

Гіперболи, з іншого боку, мають властивість, що їх геометричне місце точок, має постійну різницю з двома фокусами.

$$|d_1 - d_2| = 2a . \quad (5)$$

Так як різниця може бути або додатною або від'ємною, ця властивість відноситься до величини різниці. Для даного набору фокусів є дві гіперболи, які задовольняють співвідношення (вгору і вниз), і що, на відміну як еліпсів, вони є відкритими кривими. Рівняння для висхідної і спадаючої гіпербол показано на рисунку 1.3, є аналогічним до рівняння еліпса за винятком знаку мінус у наступному рівнянні:

$$\frac{y^2}{a^2} - \frac{x^2}{b^2} = 1 . \quad (6)$$

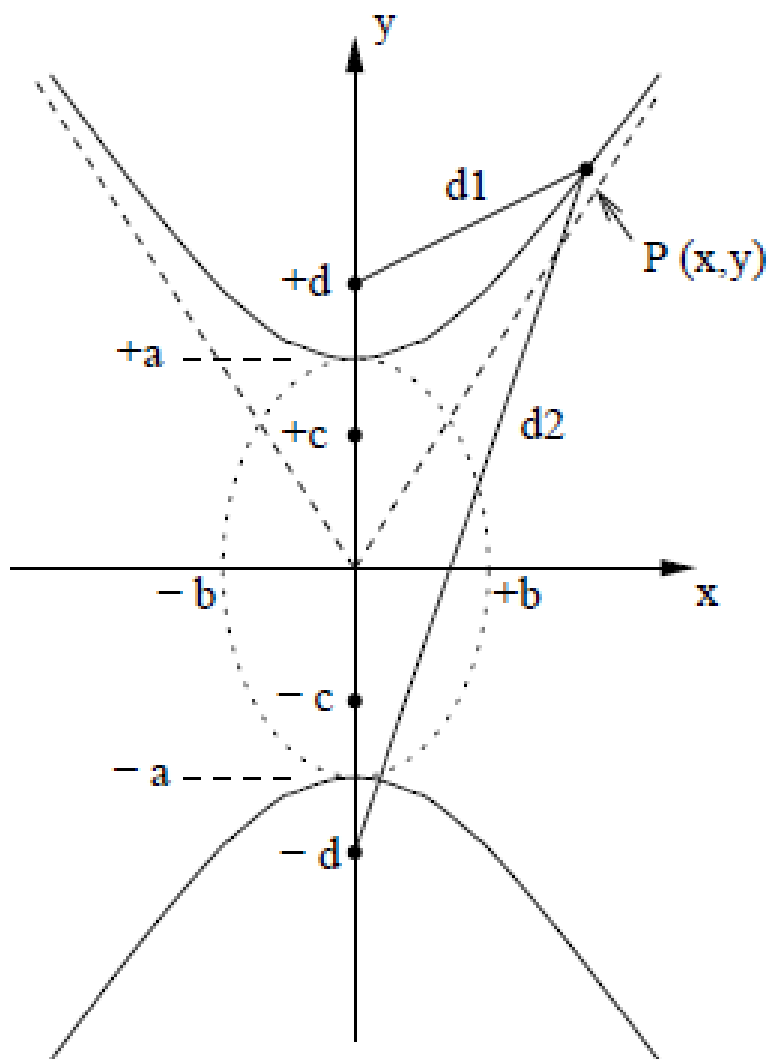


Рисунок 1.3 - Висхідна і спадаюча гіперболи

У той час як фокуси еквівалентного еліпса при $y = \pm c$, фокуси гіперболи знаходяться на $y = \pm d$. Аналогічне співвідношення існує між d і параметрами, a і b , а саме:

$$a^2 + b^2 = d^2 . \quad (6)$$

З урахуванням x , y координати двох фокусів гіперболи мають відповідати двом локаціям вимірювання ТОА, не важко бачити, що геометричне місце точок гіперболи, що відповідає постійно віддаленій різниці між відстанями d_1 і d_2 , також відповідає постійній різниці в часі прибуття (TDOA) в двох місцях. Ця різниця в часі позначається Δt [3].

1.2 Метод TDOA

Time difference of arrival (TDOA) являє собою техніку, яка використовується в пеленгації і навігації, в якому час приходу конкретного сигналу, при фізично окремих приймальних станцій з точно синхронізованими тимчасові посилення, обчислюються. У військовому контексті, вона є частиною як радіоелектронної боротьби і вимірювання і підписи інтелекту.

Одним комерційним застосування для TDOA є визначення координат мобільного телефону, що засноване на порівнянні різниці в надходженні сигналу на вежі оператора. Методика не вимагає додаткових схем, що входять в телефон, так як вона використовує стандартний сигнал. Він може бути доповнений інформацією кутом прибуття, якщо вежі мають спрямовану прийомну антену.

TDOA не слід плутати з часом прибуття (TOA). Навіть якщо потік TDOA виклик буде виглядати практично так само, як потік TOA виклику, є різниця в тому, як обчислюється розташування. TDOA і TOA схожі, але є різниця. TOA відрізняється тим, що він використовується абсолютний час прибуття на певну базову станцію, а не різницю між часом надходження на дві станції. Відстань може бути безпосередньо розрахована з моменту

прибуття, так як сигнали передаються з відомою швидкістю. Час прибуття даних з двох базових станцій будуть звужувати положення двох точок і даних третьої базової станції потрібно вирішити точне положення.

Він також використовується в пасивних радіолокаційних системах, які особливо привабливі для протидії новітнім військовим технологіям.

Затримка за часом прибуття (TDOA) може також застосовуватися для визначення місця розташування акустичного джерела (тобто звуку пострілу, вибуху і т.д.) поруч з масивом мікрофонів.

За рахунок використання різниці в часі приходу звуку до мікрофонів, TDOA дозволяє визначити місцеположення джерела звуку. В якості опції, вона приймає набір сигналів мікрофона в якості вхідних даних і повертає координати джерела по відношенню до мікрофона масив.

Нехай $\{(x_m, y_m, z_m)\}$, де $m=1\dots M$, буде координатами M мікрофонів. Нехай (x, y, z) будуть невідомими координатами джерела, що ми визначаємо. Нехай t_m час надходження звуку від джерела до мікрофона m . Нехай v буде швидкістю звуку (340.29 метрів в секунду в повітрі).

Нехай $R_m = vt_m$ буде відстанню між джерелом і мікрофоном m .

Нехай $\tau_m = T_m - t_1$ буде різницею часу в дорозі між мікрофоном m і мікрофоном 1.

Тепер необхідно вирішити замінити попередній результат в рівняння (z)

$$0 = v\tau_m - v\tau_2 + \frac{1}{v\tau_m}(x_1^2 + y_1^2 + z_1^2 - x_m^2 - y_m^2 - z_m^2 - 2x_1x - 2y_1y - 2z_1z + 2x_mx + 2y_my + 2z_mz) - \frac{1}{v\tau_m}(x_1^2 + y_1^2 + z_1^2 - x_2^2 - y_2^2 - z_2^2 - 2x_1x - 2y_1y - 2z_1z + 2x_2x + 2y_2y + 2z_2z) \quad (16)$$

Для випадків $m = 3, 4 \dots M$.

Можна переписати вищенаведене рівняння стисліше, як

$$0 = D_m + A_mx + B_my + C_mz \quad (17)$$

Де,

$$\begin{aligned}
 A_m &= \frac{1}{v\tau_m}(-2x_1 + 2x_m) - \frac{1}{v\tau_2}(2x_2 - 2x_1) \\
 B_m &= \frac{1}{v\tau_m}(-2y_1 + 2y_m) - \frac{1}{v\tau_2}(2y_2 - 2y_1) \\
 C_m &= \frac{1}{v\tau_m}(-2z_1 + 2z_m) - \frac{1}{v\tau_2}(2z_2 - 2z_1)
 \end{aligned} \tag{18}$$

та

$$\begin{aligned}
 D_m &= v\tau_m - v\tau_2 + \frac{1}{v\tau_m}(x_1^2 + y_1^2 + z_1^2 - x_m^2 - y_m^2 - z_m^2) \\
 &\quad - \frac{1}{v\tau_2}(x_1^2 + y_1^2 + z_1^2 - x_2^2 - y_2^2 - z_2^2)
 \end{aligned} \tag{19}$$

Для випадків $m = 3, 4 \dots M$.

Далі необхідно переписати вищенаведений перелік $M - 2$ рівнянь в матрицю, щоб отримати

$$\begin{bmatrix} A_3 & B_3 & C_3 \\ A_4 & B_4 & D_4 \\ \vdots & \vdots & \vdots \\ A_M & B_M & C_M \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -D_3 \\ -D_4 \\ \vdots \\ -D_M \end{bmatrix} \tag{20}$$

Далі необхідно псевдообернути матрицю з обох сторін для визначення x , y , z . Також, важливо звернути увагу, що вищенаведені дії дають рішення тільки тоді, коли $M \geq 5$. Іншими слова, для коректної реалізації необхідно 5 або більше мікрофонів[4].

1.3 Порівняння методів TOA та TDOA

Використовуючи в повній мірі TDOA набори вимірів, більше інформації буде обробляється для визначення місцеположення джерела звукового сигналу. Теоретичні нижні межі менше для коректної роботи, ніж

для локалізації TDOA з фіксованим контрольним датчиком, в порівнянні з реалізацією без контрольного датчика. У той час як теоретичні оцінки не показують ніякої різниці в TOA і локалізації TDOA з фіксованим еталонним датчиком, на практиці, така характеристика для методу та її вихідна вартість повинна бути прийняті до уваги. Дослідження по кутах між двома джерелами на одній лінії можуть вказувати на труднощі знаходження при максимальній витраті ресурсів при обробці TDOA даних та обчислень. Моделювання не показують очевидні переваги при використанні локалізації TOA, але показують майже рівні показники TOA в порівнянні до локалізації TDOA[5].

В залежності від складності існуючої задачі для визначення координат джерела та наявних технічних засобів необхідно обирати метод визначення враховуючи всі можливі ситуації. Необхідно одночасно враховувати бажану точність визначення місцеположення джерела, кількість самих джерел, що можуть створювати звук одночасно, кількість мікрофонів, що будуть оброблювати звук, синхронізацію між цими мікрофонами.

Для програми, що будуть займатися визначенням координат, необхідно також враховувати кількість даних що буде оброблятися, кількість запитів на визначення координат, що будуть знаходити від клієнтів та складність реалізованого алгоритму. Залежно від всіх перерахованих умов потрібно створювати реалізацію обраного методу, що буде задовольняти всі умови та поставлену задачу.

1.4 Висновки

Перед початком реалізації будь-якого методу чи алгоритму слід звертати увагу на вже існуючі вирішення поставленої задачі чи проблеми. Далі будуть розглянуті методи для визначення місцеположення об'єкту, а саме TOA та TDOA. Для додатків реального часу локалізації акустичного джерела, однією з головних проблем є значне зростання обчислювальної складності, пов'язані з появою все більш великих, активний чи пасивний,

розподілених сенсорних мереж.

Ці датчики в значній мірі залежать від компонентів системи на батарейках для досягнення високої функціональної автоматизації в передачі сигналів і обробки інформації. Для того щоб зберегти вимоги до зв'язку мінімальним, бажано виконувати якомога більше обробки на приймачі платформах, якщо це можливо. Проте, складність обчислень, необхідних для досягнення точної локалізації джерела різко зростає з розміром сенсорних масивів, що призводить до значного зростання обчислювальних вимог, які не можуть бути легко зустрічалися зі стандартним обладнанням. Для мережі або масиву датчиків чи мікрофонів, пасивна локалізація може бути здійснюється за рахунок використання прийнятих сигналів невідомих джерел звуку. В залежності від існуючої проблеми чи задачі можна обирати підходящий метод локалізації джерела звуку чи сигналу. У цьому розділі було розглянуто два методи для визначення місцеположення джерела сигналу чи звуку, а саме TOA та TDOA. Кожен з цих методів має свої переваги та недоліки, що найбільше проявляють себе при їх реалізації, а саме складність обчислень та обсяг даних для обробки, кількість датчиків чи мікрофонів необхідних для коректної та повноцінної роботи, та вимоги до обладнання на якому і будуть безпосередньо визначатися координати об'єкту.

При виборі одного з цих методів необхідно повністю враховувати вимоги до результату їх роботи та обирати в залежності від потреб у конкретній ситуації.

2. МОДИФІКАЦІЯ МЕТОДУ ВИЗНАЧЕННЯ КООРДИНАТ ОБ'ЄКТУ

2.1 Реалізація методу TDOA

В результаті аналізу існуючих алгоритмів для реалізації системи локалізації було обрано техніку TDOA. TDOA, на відміну від TOA, може забезпечити вищу точність при визначенні координаті об'єкту. В розробленому алгоритмі за обраною технікою є обмеження: кількість мікрофонів повинна бути не меншою за 5. Застосування було розроблено за допомогою мови Python версії 2.7.11, спеціалізованих бібліотек для математичних та інших наукових досліджень NumPy, SciPy та SciKits останніх версій.

NumPy є основним пакетом для наукових обчислень з Python. Він містить, серед іншого:

- потужний інструментарій для роботи з N-мірними масивами об'єктів;
- складні функції;
- інструменти для інтеграції з C / C ++ і Fortran кодом;
- корисні функції лінійної алгебри, перетворення Фур'є і інструментарій для роботи з випадковими числами.

Крім його очевидних наукових цілей, NumPy також може бути використаний в якості ефективного багатовимірного контейнеру для загальних даних. Також можуть бути визначені довільні типи даних для роботи. Це дозволяє NumPy плавно і швидко інтегруватися з широким спектром баз даних[6].

Matplotlib є бібліотекою для побудови 2D графіків у Python, яка виробляє якісні графіки з можливістю публікації в різних друкованих форматах і для інтерактивних середовищ на різних платформах. Matplotlib можна використовувати в скритах Python і IPython оболонки (на зразок MATLAB чи Mathematica), серверах веб-додатків, а також шість графічних наборів

інструментів засобів для користувацького інтерфейсу. На рисунку 2.1 можна побачити приклади побудованих графіків за допомогою Matplotlib.

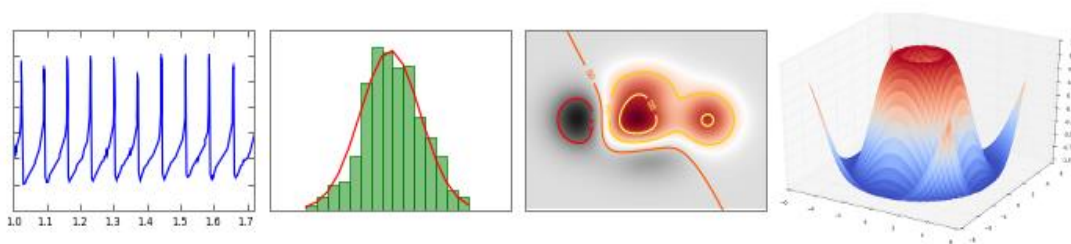


Рисунок 2.1 - Приклади побудованих графіків за допомогою Matplotlib

Matplotlib дозволяє вирішити легкі задачі речі легко і зробити вирішення складних задач можливим. За допомогою Matplotlib можна створювати графіки, гістограми, спектри потужності, діаграми розсіювання і т.д., за допомогою всього декількох рядків коду.

Для простого графіків інтерфейс pyplot забезпечує Matlab-подібний інтерфейс, особливо в поєднанні з IPython. Для просунутих користувачів, є можливість повного контролю над стилями ліній, властивостями шрифту, властивості осей і т.д., через об'єктно-орієнтований інтерфейс або через набір функцій, які доступні у MATLAB[7].

SciPy — відкрита бібліотека високоякісних наукових інструментів для мови програмування Python. SciPy містить модулі для оптимізації, інтегрування, спеціальних функцій, обробки сигналів, обробки зображень, генетичних алгоритмів, розв'язування звичайних диференціальних рівнянь та інших задач, які розв'язуються в науці і при інженерній розробці. Бібліотека розробляється для тієї ж аудиторії, що і MATLAB та SciLab. Для візуалізації при використанні SciPy часто застосовують бібліотеку Matplotlib, яка є аналогом засобів виводу графіків MATLAB[8].

SciKits (скорочення для SciPy Toolkits), є додатковими пакетами для SciPy і розробляється окремо від основного дистрибутиву SciPy[9].

Для створення серверу використовувався протокол UDP.

Мова Python була обрана через те, що вона дає можливість створити швидко

реалізацію майже будь якого алгоритму за допомогою вже існуючих основних та додаткових програмних засобів та бібліотек з підтримкою повноцінної візуалізації та подальшої роботи з отриманими даними при тестуванні розробленого додатку.

Під час розробки системи також були враховані гнучкість коду для внесення подальших змін та простої підтримки розробленого додатку.

Були створені декілька модулів, що забезпечують коректне функціонування розробленої системи:

Модуль `base_samples`:

Файл `core.py`:

`__init__()`: в якості параметрів приймає назву аудіо файлу, з яким буде відбуватися робота системи, кількість процесів за допомогою яких буде відбуватися локалізація об'єкту, кількість спроб при визначенні координат об'єкту за допомогою яких можна покращити точність при обчисленні координат та кількість мікрофонів за допомогою яких буде відбуватися симуляція отримання звукового сигналу від джерела

`generate_source_positions()`: реалізує створення джерела, пошуком якого буду займатися система

`generate_distances()`: реалізує створення структур даних для збереження визначених координат джерела, та генерацію структури даних та відповідних для наперед визчених координат об'єкту

`prepare()`: виконує створення затримок для мікрофонів та створення структури для подальшої можливості відображення отриманих даних за допомогою графіка

`generate_signals()`: відбувається створення сигналів та необхідних затримок на мікрофона, з подальшою обробкою та визначенням координат джерела, збереження отриманих даних для подальшої візуалізації отриманих результатів або для повернення результатів обчислень користувачу

`time_delay_func(x, y)`: в якості параметрів приймає оброблені звукові дані з двох мікрофонів, виконує кореляцію між ними, виконує пошук максимального значення та виконує завершальну обробку даних

`x, y` – вхідні масиви даних з мікрофонів

`draw_plot()`: реалізує можливість графічного відображення отриманих результатів обчислень та зображення різниці між отриманими та наперед визначеними координатами джерела

Файл `console_runner.py` містить консольну реалізацію програми для визначення координат, що приймає необхідні для роботи системи параметри та може бути використаний для автоматизованого тестування системи.

Модуль `client`:

Файл `microphone_proxu.py`:

Class `MicrophoneProxu`:

`__init__()`: в якості параметрів приймає адресу сервера і його порт, куди необхідно надіслати повідомлення, та саме повідомлення.

`run()`: сереалізує повідомлення, розділяє його на рівні частини, розмір яких обмежений максимальним розміром UDP повідомлення.

Модуль `logic`:

Файл `helpers.py`

`time_delay_func_parallel(start, end, outs, multi)`: розподілена реалізація методу `time_delay_func(x, y)`. В якості параметрів приймає межі проміжку для якого буде запущено виділений процес для пришвидшення часу виконання алгоритму, масив в який буде відбуватися запис оброки даних для пари даних з мікрофонів та набір даних з усіх мікрофонів.

`Start, end` – нижня та верхня проміжку

`Outs` – масив для запису результатів обробки даних

`Multi` – дані з мікрофонів

`perdelta(start, end, delta)`: в якості параметрів приймає початок, кінець проміжку та дельту що визначає на скільки проміжків необхідно розбити діапазон

`Start, end` – мінімальна та максимальна границя

`Delta` – кількість проміжків на який необхідно розділити діапазон

Файл `orchestrator.py`

`class Orchestrator:`

`__init__()`: в якості параметру приймає json об'єкт, в якому містяться дані для

```
{
    "server_address" : "localhost",
    "server_port" : 10000,
    "radius" : 50,
    "cores_amount" : 2,
    "trials": 1,
    "audio": "../samples/sample.wav"
    "microphone_amount": 5
}
```

моделювання. На рисунку 2.2 можна побачити приклад вмісту конфігураційного JSON файлу.

Рисунок 2.2 - Приклад вмісту конфігураційного JSON файлу

`retrieve_file_data()`: виконує опрацювання аудіо файлу та збереження його даних для подальшої обробки та моделювання.

`init_server()`: виконує ініціалізацію сервера котрий буде отримувати данні з проксі мікрофонів.

`send_data_to_server()`: для кожної з спроб локалізації джерела звуку відбувається генерації даних для кожного мікрофона та виклик приватного методу `__send_data_via_proxy`.

`__send_data_via_proxy()`: в якості параметру приймає повідомлення, що необхідно відправити. Виконує створення проксі для кожного мікрофону, з генерацією `uuid`. Викликає безпосередню відправку даних на проксі мікрофона.

locate(): викликає опрацювання отриманих даних на сервері, локалізацію джерела звуку, виведення отриманих результатів.

Файл parallel_process.py: містить реалізацію класу ProcessParallel, що відповідає за керування створеними для обробки даних з мікрофонів процесами.

Модуль server_dgram:

Файл server.py:

class Server:

__init__(): ініціалізує сервер за допомогою юданих отримах при ініфіалізації класу Orchestrator.

generate_data(): виконує всі підготовчі методи, а саме generate_source_positions, generate_distances та prepare.

run(): створює сокет за допомогою якого будуть отримуватися дані з мікрофонів. Містить структуру даних для збереження даних з кожного проксі мікрофону. За раз може отримати максимум 65507 байт. На початку кожного повідомлення знаходиться універсальний ідентифікатор кожного мікрофону, що дозволяє привильно обробляти дані та зручно ідентифікуту вхідні повідомлення серверу.

generate_source_positions(): реалізує створення джерела, пошуком якого буду займатися система.

generate_distances(): реалізує створення структур даних для збереження визначиних координат джерела, та генерацію структури даних та відповідних для наперед визчених координат об'єкту.

prepare(): виконує створення затримок для мікрофонів та створення структури для подальшої можливості відображення отриманих даних за допомогою графіка.

log_results(): виконує форматоване виведення результатів локалізації джерела акустичного сигналу. Приклад можна побачити на рисунку 2.3.

`draw_plot()`: реалізує можливість графічного відображення отриманих результатів обчислень та зображення різниці між отриманими та наперед визначеними координатами джерела. Приклад можна побачити на рисунку 2.4.

```
2.345813989639282 passed for localization computation trial.  
Localized source.  
15.410365104675293 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -4.183125148716300, Estimated Y = -2.720778178984943, Estimated Z = 5.959746670997678  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 2.4 - Приклад результатів

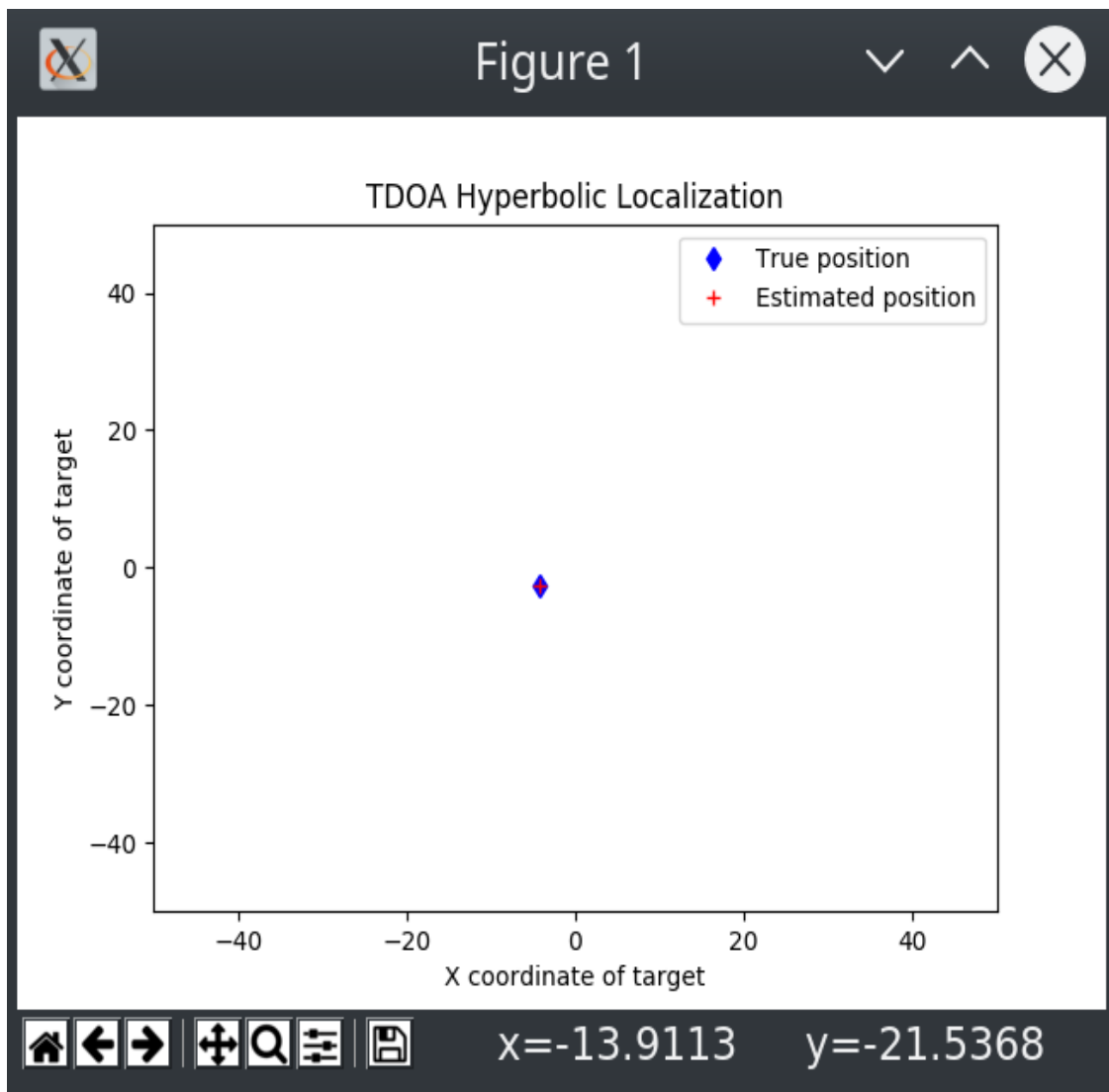


Рисунок 2.3 - Приклад графіку

`handle_retrieved_data()`: виноує десереалізацію отриманих даних з проксі мікрофонів та подальшу обробку для виконання локалізації.

`locate()`: виконує локалізації джерела звуку у заданому режимі, тобто може виноувати обчислення паралельно при необхідності .

Файл `server_runner.py` містить ініалізацію класу `Orchestrator` та всіх його підкомпонентів, що може бути використаним для подальшого та більш детального тестування.

2.2 Реалізація розподіленого методу TDOA

Під час реалізації алгоритму TDOA було виявлено, що найбільшу частину часу виконання займає визначення координат джерела, при готових проміжних даних. В конкретній реалізації алгоритму найдовше виконується почергова кореляція даних отриманих з мікрофонів, оскільки кореляції відбувається з двома великими масивами даних, розмір яких залежить від розміру аудіо файлу який буде аналізуватися під час роботи програми.

У статистиці, залежністю є будь-який статистичний зв'язок між двома випадковими змінними або двома наборами даних. Кореляція відноситься до будь-якого з широкого класу статистичних відносин, пов'язаних із залежністю, хоча в загальному користуванні вона найчастіше відноситься до ступеня, в якій дві змінні мають лінійну зв'язок один з одним. Знайомі приклади залежних явищ включають кореляцію між фізичними даними батьків і їх нащадків, а також співвідношення між попитом на вироби і його ціна.

Кореляція є корисною, тому що вона може вказувати на прогностичні відносини які можуть бути використані на практиці. Наприклад, електричний обігрівач може споживати менше енергії на день на основі співвідношення попиту на електроенергію і погодних умов. У цьому прикладі є причинно-наслідковий зв'язок, тому що екстремальні погодні умови змушують людей використовувати більше електроенергії для опалення або охолодження; однак статистична залежність не є достатньою,

щоб продемонструвати наявність такого причинно-наслідкового зв'язку (тобто, кореляція означає наявність причинно-наслідкового зв'язку).

Формально залежність відноситься до будь-якої ситуації, в якій випадкові величини не задовольняють математичній умові ймовірнісної незалежності.

У загальному використанні, кореляція може відноситися до будь-якого відхилення двох або більше випадкових величин, але технічно це відноситься до будь-якого з декількох більш спеціалізованих типів відносин між середніми значеннями. Є кілька коефіцієнтів кореляції, часто позначаються ρ або R , що вимірюють ступінь кореляції. Найбільш поширеним з них є коефіцієнт кореляції Пірсона, який чутливий тільки до лінійної залежності між двома змінними (які можуть існувати, навіть якщо один є нелінійною функцією іншого). Інші коефіцієнти кореляції були розроблені, щоб бути більш стійкими, ніж кореляції Пірсона, тобто, більш чутливі до нелінійних відносин. Взаємна інформація також може бути застосована для вимірювання залежності між двома змінними.

Найбільшу частину часу виконання в однопроцесерній реалізації алгоритму TDOA займає метод `time_delay_func(x, y)` який в якості параметрів приймає оброблені звукові дані з двох мікрофонів, виконує їх кореляцію та виконує пошук максимального значення та виконує завершальну обробку даних, де x, y – вхідні масиви даних з мікрофонів.

Для реалізації розподіленої версії алгоритму був проаналізований час виконання роботи програми з великою варіативністю вхідних параметрів, тобто з різними кількостями мікрофонів відносно яких відбувається робота алгоритму, різними розмірами вхідних файлів для моделювання та кількостями процесів на яких виконується найважча частина програми.

Спочатку була створена та протестована початкова версія паралельну програми.

З отриманих результатів тестування були зроблені наступні висновки:

- час виконання частини алгоритму залежить від підходу розбиття обробки звукових даних;
- час виконання залежить від кількості процесів на яких відбувається робота;
- метод синхронізації та структура даних для синхронізації кардинально впливають на швидкодію;
- вибір типу реалізації на потоках чи на процесах є дуже важливим.

Після врахування помилок, що були допущені при створенні першої версії паралельної програми, були внесені необхідні зміни, що дали змогу значно поліпшити час виконання роботи для визначення координат. Також були створені додаткові модулі для програми, що виконують сервісні функції для підвищення якості алгоритму та коду, а саме `parallel_process.py`, що містить реалізацію менеджера для синхронізації процесів на яких відбувається обчислення, `helpers.py`, що містить сервісну функцію `perdelta` та поліпшену паралельну версію `time_delay_func - time_delay_func_parallel`.

Основні ідеї для ефективної та швидкої реалізації розподіленого алгоритму полягають у наступному:

- обчислення повинні виконуватися на окремих процесах, з якомога зручнішим та швидким методом синхронізації між ними;
- кількість процесів на яких виконується алгоритм повинна бути такою, яку можна легко змінити та відповідати обсягу роботи;
- кореляція для різних даних повинна відбуватися паралельно.

2.4 Висновки

Під час планування створення програмного рішення важливим є вибір інструментів розробки, їх налаштування та обладнання на якому буде виконуватися розробка.

До засобів та інструментів таких можна віднести мову програмування з користувацькими бібліотеками та середовище розробки з доповненнями та плагінами. Звичайно, цієї пари достатньо для вирішення усіх поставлених

задач, але при збільшенні масштабу додатку з'являється необхідність у більш спеціалізованих програмах чи зміні обладнання для його роботи. Прикладами цих програм є засоби автоматичного збирання проектів, спеціальні відлагоджувачі, програми тестування та пошуку критичних місць чи помилок, програми аналізу та вдосконалення продуктивності додатків. Деякі середовища програмування за замовчуванням включають вищезгадані засоби або дозволяють розробнику опціонально приєднувати їх за допомогою плагінів.

Проектування є невід'ємною частиною циклу розробки програмного забезпечення. В залежності від методології роботи над проектом, проектування може бути одним з перших етапів на стадії реалізації проекту. Частіше за все, етап проектування йде після визначення мети та специфікації проекту, появи сформованого дизайну з вимогами та планом рішення.

При проектуванні програмного забезпечення розглядаються наступні елементи додатку: архітектуру, компоненти, що входять до додатку та користувацькі бібліотеки, необхідну гнучкість для можливості налаштування розробленого програмного забезпечення чи заміни його компонентів, що буде значною перевагою для користувача чи іншого розробника, що буде працювати з вже готовим рішенням.

Також, дуже важливим аспектом розробки є проектування таким чином, що створену систему можна було б гнучко конфігурувати та розширювати, саме через це було обрано парадигму програмування ООП та конфігурації з використанням JSON.

Застосування принципів SOLID дозволяє значно підвищити якість створеного коду, бути легко розширюваним та документованим.

Було створено та протестовано дві версії програми, а саме, консольний додаток та UDP сервер, що дозволяє проводити моделювання у зручний спосіб та при необхідності мати можливість порівняти результати двох версій, а саме моделювання локалізації та моделювання відправди даних на сервер з подальшою їх обробкою та локалізацією.

3. ОПИС РОЗРОБЛЕНОЇ МОДЕЛІ МЕРЕЖІ

3.1 Технологічні та програмні засоби розробки

Під час планування при створенні програмного рішення важливим є вибір інструментів розробки. До таких можна віднести мову програмування з бібліотеками та середовище розробки з доповненнями та плагінами. Звичайно, цієї пари достатньо для вирішення усіх поставлених задач, але при збільшенні масштабу додатку з'являється необхідність у більш спеціалізованих програмах. Прикладами цих програм є засоби автоматичного збирання проектів, спеціальні відлагоджувачі, програми тестування та пошуку критичних місць, програми аналізу та вдосконалення продуктивності додатків. Деякі середовища програмування за замовчуванням включають вищезгадані засоби або дозволяють програмісту опціонально приєднати їх за допомогою плагінів.

Мовою програмування для реалізації додатку було обрано Python.

Python — інтерпретована об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується декілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована[11].

Серед основних її переваг можна назвати такі:

- чистий синтаксис (для виділення блоків слід використовувати відступи);
- переносність програм (що властиве більшості інтерпретованих мов);
- стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);

- можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв'язання простих задач);
- стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написано на мові Python;
- зручний для розв'язання математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор).

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ.

Інтерпретатор мови Python може бути розширений функціями та типами даних, розробленими на C чи C++ (або на іншій мові, яку можна викликати із C). Python також зручна як мова розширення для прикладних програм, що потребують подальшого налагодження.

Для системи контролю версій було обрано Git. Git - це безкоштовна та відкрита, система контролю версій, призначена для обробки від малих до дуже великих проектів із швидкістю та ефективністю.

Git легко вчити і має крихітний відбиток з блискавичною швидкістю. Він перевершує інструменти SCM, такі як Subversion, CVS, Perforce і ClearCase, з такими функціями, як дешеві локальні розгалуження, зручні місця розташування та кілька робочих процесів[15].

В якості інтегрованого середовища розробки було обрано PyCharm.

Інтегроване середовище розробки (IDE) – це комп'ютерна програма, що допомагає програмісту розробляти нове програмне забезпечення чи модифікувати вже існуюче.

Зазвичай, IDE включає в себе наступні елементи:

- текстовий редактор;

- компілятор та/або інтерпретатор;
- засоби автоматизованого збирання;
- відлагоджувальник.

Додатково, середовище розробки може включати в себе засоби інтеграції з системою контролю версій і засоби створення графічного інтерфейсу користувача. Для об'єктно-орієнтованих мов IDE може включати браузер класів, інспектор об'єктів та діаграму ієрархії класів.

PyCharm — інтегроване середовище розробки для мови програмування Python. PyCharm надає засоби для аналізу коду, має власний графічний відлагоджувальник, інструменти для запуску юніт-тестів і підтримує веб-розробку на Django. PyCharm розроблена чеською компанією JetBrains на основі IntelliJ IDEA. PyCharm працює під операційними системами Windows, Mac OS X і Linux[7].

Можливості PyCharm:

- статичний аналіз коду, підсвічування синтаксису і помилок;
- навігація серед проектів і програмного коду: відображення файлової структури проекту, швидкий перехід між файлами, класами, методами і використаннями методів;
- рефакторинг : перейменування, витяг методу, введення змінної, введення константи, підняття і опускання методу тощо;
- інструменти для веб-розробки з використанням фреймворку Django;
- вбудований відлагоджувальник для Python;
- вбудовані інструменти для юніт-тестування;
- розробка з використанням Google App Engine;
- підтримка систем контролю версій: загальний користувацький інтерфейс для Mercurial, Git, Subversion, Perforce і CVS з підтримкою списків змін та злиття.

Зображення редактору PyCharm можна побачити на рисунку 3.1.

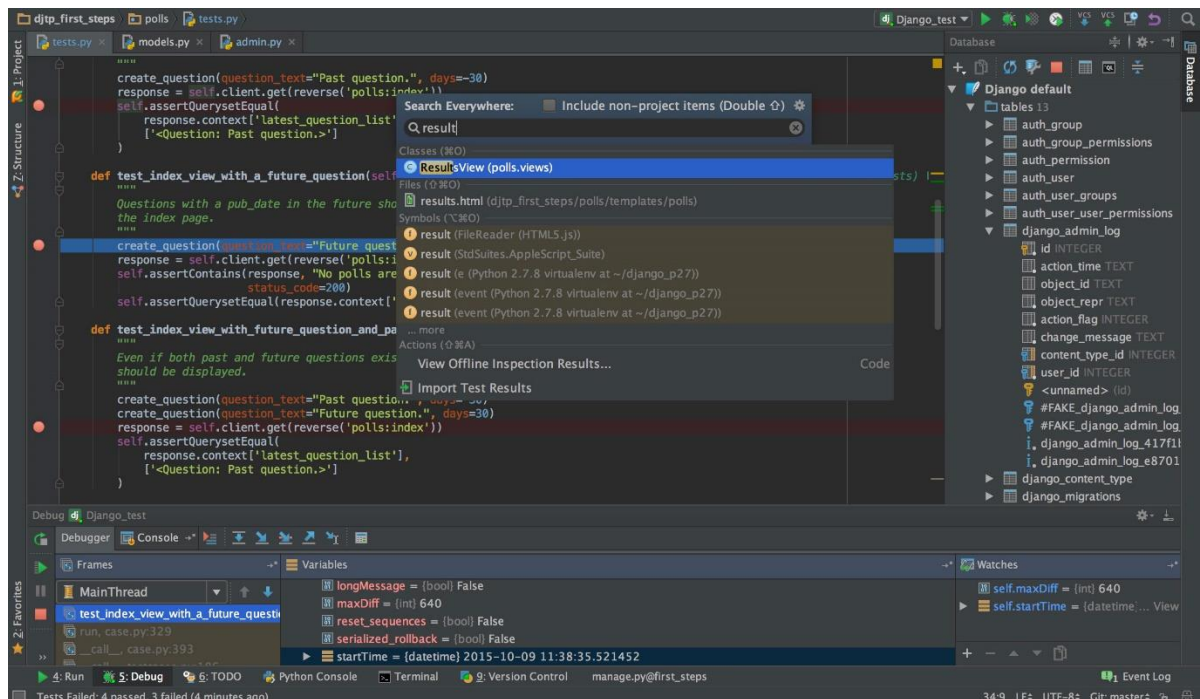


Рисунок 3.1 - Редактор PyCharm

В якості парадигми програмування для реалізації даної системи було обрано ООП. (ООП) — одна з парадигм програмування, яка розглядає програму як множину об'єктів, що взаємодіють між собою. Основу ООП складають три основні концепції: інкапсуляція, успадкування та поліморфізм. Одною з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-тих роках, вона не мала широкого застосування до 1990-тих, коли розвиток комп'ютерів та комп'ютерних мереж дозволив писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм[21].

Для передачі даних використовується протокол UDP. User Datagram Protocol, UDP — один із протоколів в стеку TCP/IP. Від протоколу TCP він відрізняється тим, що працює без встановлення з'єднання. UDP — це один з найпростіших протоколів транспортного рівня моделі OSI, котрий виконує обмін повідомленнями (датаграмами — англ. datagram) без підтвердження та гарантії доставки. При використанні протоколу UDP відповідальність за

обробку помилок і повторну передачу даних покладена на протокол рівнем вище. Але попри всі недоліки, протокол UDP є ефективним для серверів, що надсилають невеликі відповіді великій кількості клієнтів.

Протокол UDP використовують такі сервіси та протоколи вищого рівня:

- TFTP (Trivial File Transfer Protocol, найпростіший протокол передачі файлів);
- SNMP (Simple Network Management Protocol, простий протокол управління мережею);
- DHCP (Dynamic Host Configuration Protocol, протокол динамічної конфігурації вузла);
- DNS (Domain Name System, служба доменних імен).

Також цей протокол може використовуватися для різноманітних мережевих ігор реального часу, потокового відео та аудіо, інших типів даних.

UDP є одним з найпростіших протоколів транспортного рівня моделі OSI. Його детальний опис можна знайти в IETF RFC 768. UDP забезпечує дуже простий інтерфейс між мережним та програмним рівнями. UDP не гарантує доставку повідомлень, та відправник не запам'ятовує стан вже відісланих повідомлень. З цієї причини протокол UDP іноді розшифровують як «Unreliable Datagram Protocol» (протокол ненадійних датаграм). Якщо на базі UDP треба організувати надійну передачу даних, то для цього необхідно залучити протоколи більш високого рівня.

Заголовок UDP-конверту складається з 4 полів, з яких 2 є опціональними. «Порт відправника» та «контрольну суму» — це 16-бітні поля, котрі ідентифікують відправляючий та одержуючий процеси. «Порт відправника» є необов'язковим, оскільки UDP працює без встановлення з'єднання та відправник може не потребувати відповіді. В такій ситуації «порт відправника» повинен дорівнювати нулю. Поле «Розмір» є обов'язковим, воно визначає довжину усієї UDP-дейтаграми в байтах, з

полем «Дані» включно. Мінімальне значення цього поля дорівнює 8 байт. Останнє поле заголовка довжиною 16 біт містить у собі контрольну суму заголовка і поля даних. «Контрольна сума» теж є необов'язковим полем, але на практиці воно майже завжди використовується[17]. На рисунку 3.2 можна побачити UDP пакет.

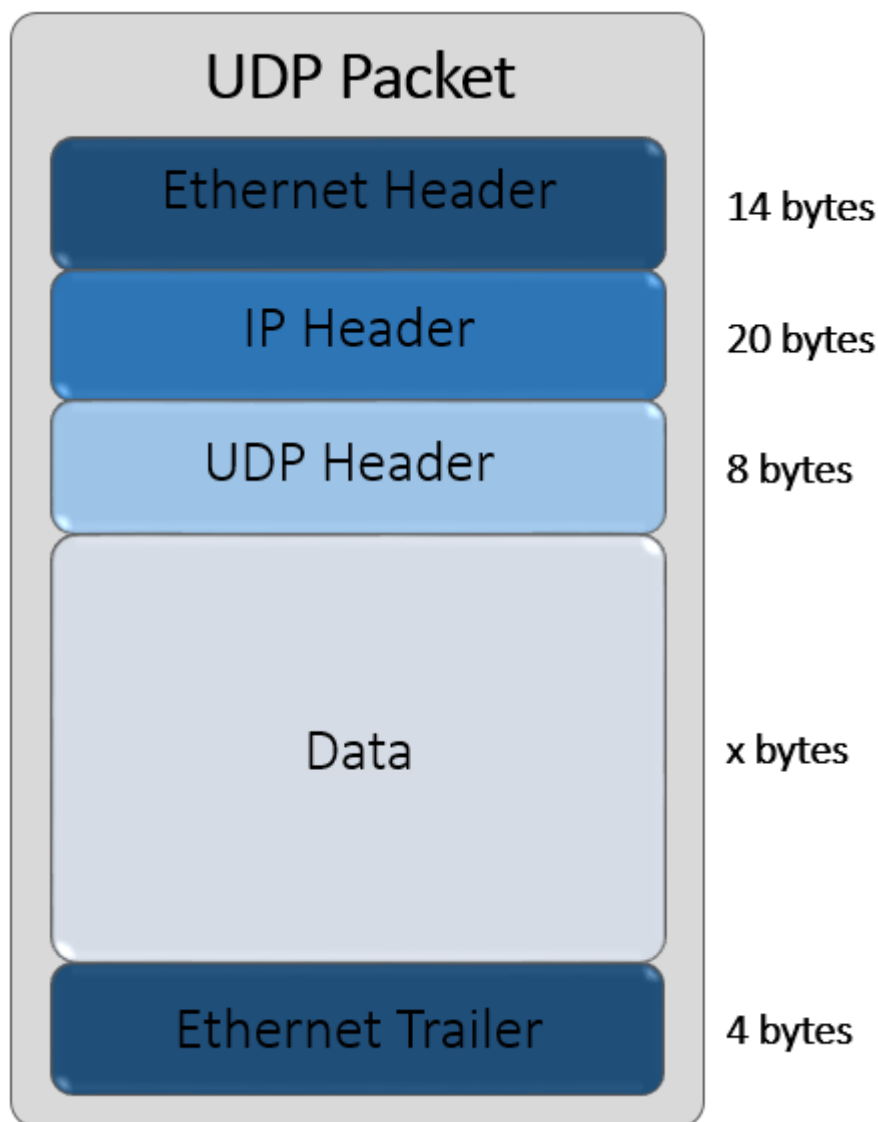


Рисунок 3.2 - UDP пакет

Програми, що використовують UDP як транспортний протокол, мають бути готові до помилок, втрати деяких конвертів та повторної передачі даних. Деякі програми, такі як TFTP, можуть використовувати додаткові програмні механізми для підвищення надійності передачі. Але у більшості випадків для таких програм надійність не є необхідною і може навіть

завадити уповільненням зв'язку. Потокове відео, ігри реального часу та VoIP (голос поверх IP) є прикладами програм, що дуже часто використовують UDP. Якщо ж програма потребує високого рівня надійності, то може використовуватися такий протокол як TCP або надлишковість коду, за допомогою якої можна знаходити помилки при передачі даних.

Оскільки у протоколі UDP відсутній будь-який контрольний механізм запобігання перевантаженням, мережні механізми повинні мати засоби для зменшення ефекту потенційних перевантажень від великого, неконтрольованого потоку UDP-трафіку. Кажучи інакше, оскільки UDP-відправники не спроможні виявляти перевантаженість, єдиним інструментом для призупинення надмірного UDP-трафіку залишаються мережні елементи, такі як роутери, що використовують «черги конвертів» та «відкидання конвертів». DCCP (англ. Datagram Congestion Control Protocol, протокол контролю навантаженості датаграм) був створений як часткове рішення цієї проблеми. Він контролює навантаження на кінцевих вузлах високошвидкісних потоків UDP-трафіку, наприклад, потокового відео.

Хоча кількість UDP-трафіку в типовій мережі сягає лишень кількох відсотків, проте багато важливих програм використовують UDP. Серед них: DNS (Domain Name System, служба доменних імен), SNMP (англ. Simple Network Management Protocol, простий протокол управління мережею), DHCP (англ. Dynamic Host Configuration Protocol, протокол динамічної конфігурації вузла), RIP (англ. Routing Information Protocol, протокол маршрутизації інформації) та багато інших[16].

Для збереження файлу конфігурації було обрано JSON (JavaScript Object Notation) - це легкий формат обміну даними. Людям легко читати і писати. Машини легко проаналізувати та генерувати. Вона заснована на підмножині мови програмування JavaScript, стандарт ECMA-262 3rd Edition - грудень 1999 р. JSON - це текстовий формат, який повністю не залежить від мови, але використовує конвенції, які знайомі програмістам C-сімейства мов, зокрема C, C++, C#, Java, JavaScript, Perl, Python та багато інших.

JSON побудований на двох структурах:

- колекція назв/значення пар. У різних мовах це реалізується як об'єкт, запис, структура, словник, хеш-таблиця, ключовий список або асоціативний масив;
- упорядкований список значень. У більшості мов це реалізується як масив, вектор, список або послідовність;

Це універсальні структури даних. Практично всі сучасні мови програмування підтримують їх у тій чи іншій формі. Має сенс, що формат даних, який взаємозамінний з мовами програмування, також повинен ґрунтуватися на цих структурах.

Ці властивості роблять JSON ідеальною мовою обмін даними[12].

Для серіалізації та десеріалізації використовується модуль `cPickle`. Модуль `cPickle` підтримує серіалізацію та десеріалізацію об'єктів Python, забезпечуючи інтерфейс і функціональність, майже ідентичні макетуючому модулю. Існує декілька відмінностей, найважливіша - продуктивність і підкласність.

`cPickle` може бути в 1000 разів швидше, ніж макіяж, оскільки перший реалізований у C. По-друге, в модулі `cPickle` розпізнані `Pickler()` і `Unpickler()` є функціями, а не класами. Це означає, що ви не можете використовувати їх для власних підкласів серіалізації та десеріалізації. Більшість програм не потребують цієї функції, і вони повинні мати переваги від значно покращеної роботи модуля `cPickle`[13].

Існують додаткові незначні відмінності в API між `cPickle` і іншими пакетами, однак для більшості програм вони взаємозамінні. Додаткова документація наведена в документації до макетированого модуля, яка включає перелік документованих відмінностей.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 2,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 16
}
```

Рисунок 4.1 - Скріншот параметрів запуску

Процесори цих комп'ютерів мають технологію Hyper-Threading (офіційно називається Hyper-Threading Technology або HT Technology і скорочено як НТТ або НТ) - це власна технологія багатопоточності (SMT) від Intel, що використовується для покращення розпаралелювання обчислень (виконання декількох завдань одночасно) на мікропроцесорів x86.

Для кожного ядра процесора, що фізично присутнє, операційна система адресує до двох віртуальних (логічних) ядер і ділиться навантаженням між ними, коли можливо. Основною функцією гіперпотоків є збільшення кількості незалежних інструкцій у конвеєрі; він використовує суперскалярну архітектуру, в якій декілька інструкцій працюють паралельно з окремими даними. З НТТ, одне фізичне ядро виступає двома процесорами операційної системи, що дозволяє одночасне планування двох процесів на ядро. Крім того, два або більше процесів можуть використовувати однакові ресурси: якщо ресурси для одного процесу недоступні, тоді, якщо доступні його ресурси, можна продовжити інший процес. На рисунку 3.3 можна побачити роботу технології НТ.

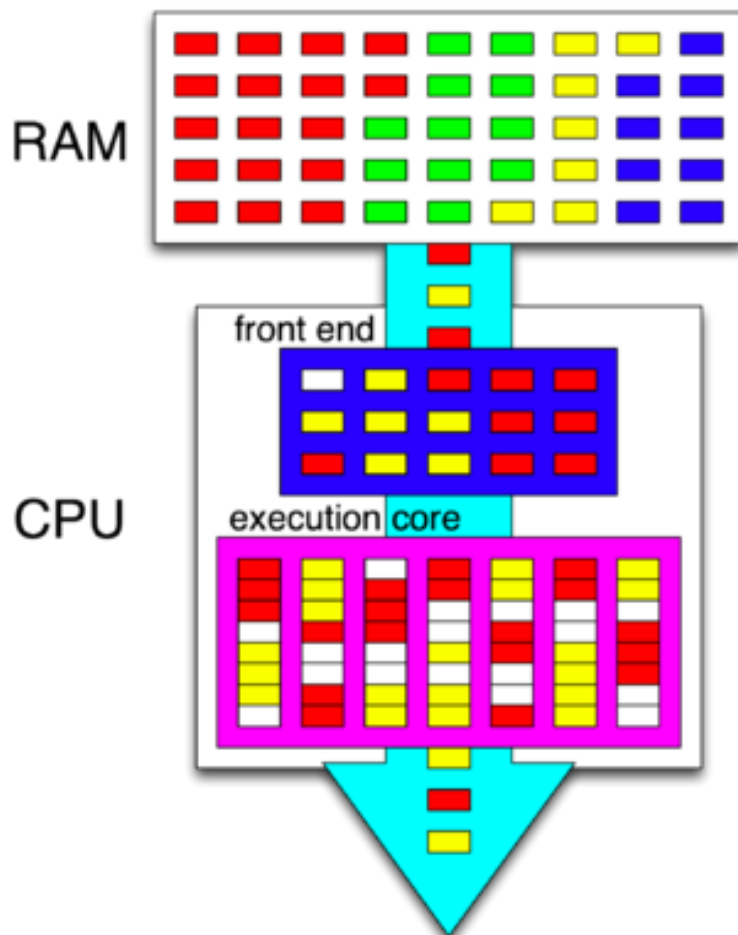


Рисунок 3.3 - Принцип роботи технології НТ

На додаток до необхідності підтримки одночасної багатопотокової (SMT) у операційній системі, гіперпоточність може бути належним чином використана лише з спеціально оптимізованою для неї операційною системою. Крім того, корпорація Intel рекомендує вимкнути НТТ при використанні операційних систем, які не знають про це апаратне забезпечення. Ця технологія значно покращить результати обчислень [14].

3.2 Опис серверу

Сервер отримує дані по протоколу UDP. Максимальний розмір повідомлення, що може отримати за один запит сервер складає 65507 байт, що є майже максимальним розміром повідомлення, відправку яких підтримує протокол UDP.

Кожне повідомлення містить на початку ідентифікатор мікрофону, подальша частина повідомлення містить частину сереалізованого на стороні проксі масив даних мікрофону. Отримані дані зберігаються в тимчасову структуру даних, після отримання повідомлення розміром 36 байт, в якому міститься ідентифікатор мікрофону, дані з тимчасової структури записуються у синхронізовану структуру даних та оновлюється рахунок мікрофонів, з котрих були отримані дані. Синхронізована структура даних дозволяє обмінюватися даними між інтерпретаторами мову Python, в яких відбувається виконання моделювання.

Після отримання даних зі всіх мікрофонів, сервер припиняє отримання даних та виконується підготовка отриманих даних для локалізації джерела звуку. Отримані дані десереалізуються у NumPy масив і відбувається безпосередня локалізація.

3.3 Опис клієнту

В якості клієнтів сервера виступаються проксі мікрофонів, що сереалізують дані кожного мікрофона, розділяють їх на рівні, крім двох останніх, повідомлення.

Серіалізація — процес перетворення будь-якої структури даних у послідовність бітів. Зворотною до операції серіалізації є операція десеріалізації — відновлення початкового стану структури даних із бітової послідовності.

Серіалізація використовується для передавання об'єктів мережею й для збереження їх у файлах. Наприклад, потрібно створити розподілений додаток, різні частини якого мають обмінюватися даними зі складною структурою. У такому випадку для типів даних, які передбачається передавати, пишеться код, який здійснює серіалізацію і десеріалізацію. Об'єкт заповнюється необхідними даними, потім викликається код серіалізації, в результаті виходить, наприклад, XML-документ. Результат серіалізації передається приймальній стороні, наприклад, електронною

поштою або через HTTP. Додаток-одержувач створює об'єкт того ж типу і викликає код десеріалізації, у результаті отримуючи об'єкт із тими ж даними, які були в об'єкті програми-відправника. За такою схемою працює, наприклад, серіалізація об'єктів через SOAP в Microsoft.NET[18].

Серіалізація надає декілька корисних можливостей:

- метод реалізації зберігання об'єктів, який зручніший, ніж запис їх властивостей в текстовий файл на диск і повторна збірка об'єктів читанням файлів;
- метод здійснення віддалених викликів процедур, як, наприклад, у SOAP;
- метод розповсюдження об'єктів, особливо в технологіях компонентно-орієнтованого програмування, таких як COM і CORBA;
- метод виявлення змін у даних, що змінюються з часом.

Для найефективнішого використання даних можливостей необхідно підтримувати незалежність від архітектури. Наприклад, необхідно мати можливість надійно відтворювати серіалізований потік даних, незалежно від порядку байтів, що використовується в цій архітектурі. Це означає, що найбільш проста і швидка процедура прямого копіювання ділянки пам'яті, в якому розміщується структура даних, не може працювати надійно для всіх архітектур. Серіалізація структур даних в архітектурно-незалежний формат означає, що не повинно виникати проблем через різний порядок проходження байтів, механізмів розподілу пам'яті або відмінностей представлення структур даних в мовах програмування.

Будь-якій зі схем серіалізації властиво те, що кодування даних послідовно за визначенням, і вибірка будь-якої частини серіалізованої структури даних вимагає, щоб весь об'єкт був зчитаний від початку до кінця і був відновлений. У багатьох програмах така лінійність корисна, тому що дозволяє використовувати прості інтерфейси введення/виведення загального

призначення для збереження і передачі стану об'єкта. У додатках, де важлива висока продуктивність, можливо буде доречніше використовувати складнішу, нелінійну організацію зберігання даних.

Розмір всіх повідомлень, крім двох останніх, що відправляється за раз складає 65036 байт, в перші 36 байт записаний ідентифікатор мікрофону, що є UUID. UUID (Universally Unique Identifier) — це стандарт ідентифікації, який використовується при створенні програмного забезпечення, затверджений Open Software Foundation (OSF) як частина Розподіленого комп'ютерного середовища (DCE). Основне призначення UUID — дозволити розподіленим системам унікально ідентифікувати інформацію без центру координації. Таким чином, кожен може створити UUID і використовувати його для ідентифікації чого-небудь з достатнім рівнем впевненості, що даний ідентифікатор не буде ненавмисно використано для чогось іншого. UUID — це 16-байтний (128-бітний) номер. В шістнадцятковій системі числення UUID має вигляд рядка цифр, розділених дефісами на п'ять груп за схемою 8-4-4-4-12 — разом 36 символів (32 цифри і 4 дефіси). Наприклад: 550e8400-e29b-41d4-a716-446655440000. Загальна кількість унікальних ключів UUID становить $2^{128} = 25616$ або близько 3.4×10^{38} . Це означає, що генеруючи 1 трильйон ключів кожної наносекунди, перебрати всі можливі значення вдасться лише за 10 мільярдів років [19][20].

Розмір передостаннього повідомлення може варіюватися в залежності від розміру початкового NumPy масиву, що сереалізуються, а потім ділиться на частини. В останньому повідомленні відбувається надсилання ідентифікатору мікрофону.

3.4 Опис взаємодії сервера та клієнта

Обмін даними між клієтами та сервером відбувається за допомогою протоколу UDP. Взаємодію по протоколу UDP клієта та серверу можна побачити на рисунку 3.4. Розмір повідомлень з даними є майже максимально можливим при використанні протоколу UDP. Відправка даних з клієнтів на сервер відбувається послідовно. Виконання коду серверу та клієнтів

відбувається в різних інтерпретаторах мови Python.

UDP Client-Server

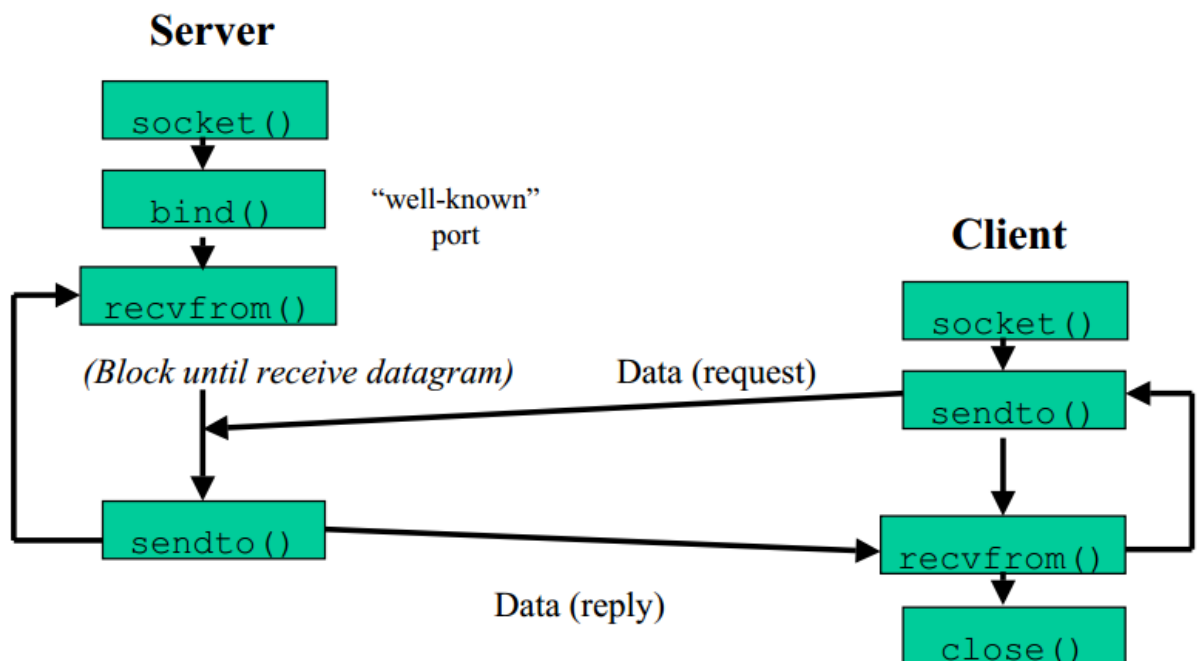


Рисунок 3.4 - Взаємодія по протоколу UDP

Клієнт надсилає частину серіалзованого масиву даних на сервер, коли всі дані з клієту надіслані, відбувається відправка ідентифікатору мікрофону. Коли сервер отримує ідентифікатор мікрофону, відбувається запам'ятовування цієї події сервером. Після отримання всіх повідомлень сервер розпочинає отримання даних від іншого мікрофону. Цей процес триває поки дані з усіх мікрофонів не надійдуть.

Після отримання всіх даних, сервер їх десеріалізує та розпочинається локалізація джерела акустичного сигналу, що може виконуватися розподілено, в залежності від параметрів запуску сервера.

3.5 Висновки

Модель мережі була розроблена з урахування усіх особливостей протоколу UDP, що дозволяє відправляти за раз достатньо велике за мірками протоколу повідомлення.

За допомогою конфігурації, що зберігається у JSON файлі можливо тестувати модель в автоматичному режимі, що дозволяє перевіряти випадки локалізації з використанням великої кількості мікрофонів, що може зайняти годину і більше.

Процес генерування даних для моделювання відбувається перед запуском серверу та мікрофонів, що дозволяє повторно використовувати сгенеровані дані для порівняння з отриманими від визначення місцеположення джерела звуку.

Процес відправки та прийняття повідомлень працює в двох інтерпретаторах Python, що дозволяє оптимально використовувати ресурси та після отримання усіх повідомлень розпочати розподілену локалізацію джерела акустичного сигналу, що в свою чергу буде проходити обраній кількості інтерпритаторів.

В моделі передбачена можливість роботи у випадку відсутності звуку з певних мікрофонів, що означає:

- для отримання точніших результатів необхідно використовувати меншу кількість мікрофонів (до 64);
- модель може бути використана для тестування процесу локалізації з кількістю мікрофонів, що обмежена лише об'ємом оперативної пам'яті комп'ютера.

4. ЕКСПЕРИМЕНТИ ЗІ СТВОРЕНОЮ СИСТЕМОЮ

4.1 Засоби для експериментів та тестування

Для експериментування з розробленою системою та її тестування як основний комп'ютер було використано два комп'ютери Dell Inspiron 3537 та Dell XPS 15.

Dell Inspiron 3537 комп'ютер має:

- 8 Гб оперативної пам'яті;
- процесор Intel Core i7-4500U (2/4);
- операційну систем Fedora 25 x64.

Dell Inspiron 3537 комп'ютер має:

- 16 Гб оперативної пам'яті;
- процесор Intel Core i7-7700HQ (4/8);
- операційну систем Fedora 27 x64.

Конфігурація даного комп'ютера дозволить перевірити можливості розробленої системи у повному обсязі та зробити повноцінні висновки з результатів.

4.2 Умови та результати

Програм без помилок не існує. Практика доводить, що винуватцями помилок у програмах найчастіше бувають самі розробники. Один із загальних законів практичного програмування полягає в тому, що жодна програма не дає бажаних результатів при першій спробі трансляції та виконання.

Розробник повинен не тільки створювати ефективні програми, але і знаходити в них усілякі помилки та досконально перевіряти на коректність роботи.

Існують два типи програмних помилок:

- синтаксичні помилки - виникають через порушення правил мови програмування. Такі помилки зазвичай виявляються під час компіляції. Можуть бути виключені порівняно легко. Навіть якщо не переглядати текст програми можна бути впевненим, що компілятор на стадії трансляції знайде помилки і видасть відповідні попередження. Фактично пошук помилок здійснює компілятор, а їхнє виправлення - розробник;
- семантичні (логічні) помилки - ті, що призводять до некоректних обчислень або помилок під час виконання (runtime error). Семантичні помилки усувають зазвичай за допомогою виконання програми з ретельно підібраними перевірочними даними, для яких відома правильна відповідь.
- Перевірка правильності роботи програмного забезпечення - це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог. До цього процесу входить виконання програми з метою знайдення помилок.

Якість не є абсолютною, це суб'єктивне поняття. Тому така перевірка не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення і забезпечення якості програмного забезпечення, до якого належать усі складові ділового процесу, а не тільки тестування.

Існує багато підходів до перевірки якості програмного забезпечення, але ефективне тестування складних продуктів - це по суті дослідницький процес, а не тільки створення і виконання рутинної процедури.

Перевірка пронизує весь життєвий цикл ПЗ, починаючи від проектування і закінчуючи невизначено довгим етапом експлуатації. Ці роботи

безпосередньо пов'язані із завданнями управління вимогами та змінами, адже метою контролю якості є якраз можливість переконатися у відповідності програм заявленим вимогам.

Перевірка якості та коректності - процес також ітераційний. Після виявлення та виправлення кожної помилки обов'язково слід повторити експерименти, щоб переконатися у працездатності програми. Більше того, для ідентифікації причини виявленої проблеми може знадобитися проведення спеціальної додаткової перевірки.

Існує безліч підходів до вирішення завдання підтвердження правильності роботи та верифікації ПЗ, але ефективна перевірка складних програмних продуктів - це процес у вищій мірі творчий, не зводиться до прямування строгими і чіткими процедурами або до створення таких.

Створення даних для перевірки, є важливою частиною тестування програмного забезпечення, це процес створення набору даних для перевірки адекватності нових або переглянутих програм. Це можуть бути фактичні дані, які були взяті з попередніх операцій або штучних даних, створених для цієї мети. Створення даних розглядається як складна проблема, і хоча багато рішень було створено, більшість з них обмежуються простими програмами. Використання динамічного розподілу пам'яті в більшій частині коду, є найбільш серйозною проблемою, що постає при створенні таких даних. Способи використання програмних продуктів стають дуже непередбачуваними, в зв'язку з цим стає все важче передбачити шляхи, що складна програма може прийняти, що робить його практично неможливим створення тестових даних для перевірки всіх варіантів її використання. Проте, в останнє десятиліття значний прогрес був досягнутий у вирішенні цієї проблеми з використанням генетичних алгоритмів та інших алгоритмів аналізу. Крім того, перевірка програмного забезпечення є важливою частиною циклу розробки програмного забезпечення життя і в основному є трудомісткою.

Для перевірки роботи створеної системи було обрані випадки для

тестування, що зможуть продемонструвати її можливості. Оскільки на комп'ютері, на якому відбувалося тестування вставлений двоядерний процесор, то для того, щоб розпаралелювання алгоритму давало значний ефект необхідно використовувати два окремих процеси для обчислень. При тестуванні на комп'ютерах з більшою кількістю ядер у процесорі, необхідно враховувати кількість ядер, що можуть виконувати задачі паралельно, в інакшому разі замість пришвидшення роботи можна отримати протилежний результат.

Для перевірки був створений аудіо файл розміром 124 КБ, що в повному обсязі дозволяє дослідити поведінку системи при різній кількості мікрофонів та процесів на яких буде виконуватися алгоритм. Нажаль використання аудіо файлів більших за 2000 Кб неможливе через обмежену кількість оперативної пам'яті, було прийняте рішення саме про використання аудіо файлу такого розміру, що не є граничним для даних системи та залишає певну кількість вільної оперативної пам'яті для роботи операційної системи та інших необхідних для коректної роботи алгоритму програм. Оскільки реалізація методу TDOA має обмеження на мінімальну допустиму кількість мікрофонів для роботи, а саме 5, з точки зору реалізації та необхідної точності для визначення координат джерела звуку, було прийнято рішення проводити досліди з 8, 16, 32, 64, 128, 256, 512, 1024 для Dell Inspiron 3537, Dell XPS 15 та з додатковим випадком для Dell XPS 15 - 2048 наявних мікрофонів та з використанням 1, 2, 4 для Dell Inspiron 3537 та 1, 2, 4, 8 для Dell XPS 15 для обчислень у найповільнішій частині програми. Далі будуть наведені рисунки, на котрих буде зображено параметри та результати.

Для випадку 8 мікрофонів та 1 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.1, 4.2 та 4.3 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 1,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 8
}
```

Рисунок 4.1 - Скріншот параметрів запуску

```
5.220946073532104 passed for localization computation trial.
Localized source.
18.901601791381836 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.183125148716300, Estimated Y = -2.720778178984943, Estimated Z = 5.959746670997678
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.2 - Скріншот результатів роботи на Dell Inspiron 3537

```
3.196430921554565 passed for localization computation trial.
Localized source.
16.423983812332153 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.183125148716343, Estimated Y = -2.720778178984958, Estimated Z = 5.959746670997683
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.3 - Скріншот результатів роботи на Dell XPS 15

Для випадку 8 мікрофонів та 2 процесів можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.4, 4.5 та 4.6 можна побачити параметри та результати роботи програми для цього випадку, на двох комп'ютерах.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 2,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 8
}
```

Рисунок 4.4 - Скріншот параметрів запуску

```
2.798053979873657 passed for localization computation trial.
Localized source.
15.892421007156372 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.183125148716300, Estimated Y = -2.720778178984943, Estimated Z = 5.959746670997678
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.5 - Скріншот результатів роботи на Dell Inspiron 3537

```
1.775823116302490 passed for localization computation trial.
Localized source.
14.988885879516602 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.183125148716343, Estimated Y = -2.720778178984958, Estimated Z = 5.959746670997683
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.6 - Скріншот результатів роботи на Dell XPS 15

Для випадку 8 мікрофонів та 4 процесів можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.7, 4.8 та 4.9 можна побачити параметри та результати роботи програми для цього випадку.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 4,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 8
}
```

Рисунок 4.7 - Скріншот параметрів запуску

```
2.345813989639282 passed for localization computation trial.
Localized source.
15.410365104675293 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.183125148716300, Estimated Y = -2.720778178984943, Estimated Z = 5.959746670997678
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.8 - Скріншот результатів роботи на Dell Inspiron 3537

```
0.893403053283691 passed for localization computation trial.
Localized source.
14.085331916809082 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.183125148716343, Estimated Y = -2.720778178984958, Estimated Z = 5.959746670997683
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.9 - Скріншот результатів роботи на Dell XPS 15

Для випадку 8 мікрофонів та 8 процесів можна побачити процес

формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.10, 4.11 можна побачити параметри та результати роботи програми для цього випадку.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 8,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 8
}
```

Рисунок 4.10 - Скріншот параметрів запуску

```
0.798724889755249 passed for localization computation trial.
Localized source.
14.006515979766846 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.183125148716343, Estimated Y = -2.720778178984958, Estimated Z = 5.959746670997683
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.11 - Скріншот результатів роботи на Dell XPS 15

Для випадку 16 мікрофонів та одного процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.12, 4.13 та 4.14 можна побачити параметри та результати роботи програми для цього випадку, на двох комп'ютерах.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 1,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 16
}
```

Рисунок 4.12 - Скріншот параметрів запуску

```
10.715283155441284 passed for localization computation trial.
Localized source.
37.007426977157593 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.539586973646927, Estimated Y = -2.945688797307469, Estimated Z = 6.037763312010591
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.13 - Скріншот результатів роботи на Dell Inspiron 3537

```
6.685275793075562 passed for localization computation trial.  
Localized source.  
33.225836992263794 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -4.539586973647015, Estimated Y = -2.945688797307522, Estimated Z = 6.037763312010613  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.14 - Скріншот результатів роботи на Dell XPS 15

Для випадку 16 мікрофонів та 2 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.15, 4.16 та 4.17 можна побачити параметри та результати роботи програми для цього випадку, на двох комп'ютерах.

```
{  
  "server_address" : "localhost",  
  "server_port" : 10000,  
  "radius" : 50,  
  "cores_amount" : 2,  
  "trials": 1,  
  "audio": "../samples/sample.wav",  
  "microphone_amount": 16  
}
```

Рисунок 4.15 - Скріншот параметрів запуску

```
5.667150974273682 passed for localization computation trial.  
Localized source.  
31.940102100372314 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -4.539586973646927, Estimated Y = -2.945688797307469, Estimated Z = 6.037763312010591  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.16 - Скріншот результатів роботи на Dell Inspiron 3537

```
3.453932046890259 passed for localization computation trial.  
Localized source.  
30.018072843551636 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -4.539586973647015, Estimated Y = -2.945688797307522, Estimated Z = 6.037763312010613  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.17 - Скріншот результатів роботи на Dell XPS 15

Для випадку 16 мікрофонів та 4 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні

та визначені координати джерела. На рисунках 4.18, 4.19 та 4.20 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```
{  
  "server_address" : "localhost",  
  "server_port" : 10000,  
  "radius" : 50,  
  "cores_amount" : 4,  
  "trials": 1,  
  "audio": "../samples/sample.wav",  
  "microphone_amount": 16  
}
```

Рисунок 4.18 - Скріншот параметрів запуску

```
4.830096006393433 passed for localization computation trial.  
Localized source.  
31.120068788528442 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -4.539586973646927, Estimated Y = -2.945688797307469, Estimated Z = 6.037763312010591  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.19 - Скріншот результатів роботи на Dell Inspiron 3537

```
1.815428972244263 passed for localization computation trial.  
Localized source.  
28.330079793930054 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -4.539586973647015, Estimated Y = -2.945688797307522, Estimated Z = 6.037763312010613  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.20 - Скріншот результатів роботи на Dell XPS 15

Для випадку 16 мікрофонів та 8 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.21, 4.22 можна побачити параметри та результати роботи програми для цього випадку.

```
{  
  "server_address" : "localhost",  
  "server_port" : 10000,  
  "radius" : 50,  
  "cores_amount" : 8,  
  "trials": 1,  
  "audio": "../samples/sample.wav",  
  "microphone_amount": 16  
}
```

Рисунок 4.21 - Скріншот параметрів запуску

```
1.554193019866943 passed for localization computation trial.
Localized source.
27.926271200180054 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.539586973647015, Estimated Y = -2.945688797307522, Estimated Z = 6.037763312010613
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.22 - Скріншот результатів роботи на Dell XPS 15

Для випадку 32 мікрофонів та 1 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.23, 4.24 та 4.25 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 1,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 32
}
```

Рисунок 4.23 - Скріншот параметрів запуску

```
21.392241001129150 passed for localization computation trial.
Localized source.
74.410810232162476 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.197438979636146, Estimated Y = -2.731280155952916, Estimated Z = 5.951381838966640
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.24 - Скріншот результатів роботи на Dell Inspiron 3537

```
13.329017162322998 passed for localization computation trial.
Localized source.
66.381081104278564 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.197438979636116, Estimated Y = -2.731280155952861, Estimated Z = 5.951381838966631
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.25 - Скріншот результатів роботи на Dell XPS 15

Для випадку 32 мікрофонів та 2 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.26, 4.27 та 4.28 можна побачити параметри та результати роботи програми для цього випадку на

двох комп'ютерах.

```
{  
  "server_address" : "localhost",  
  "server_port" : 10000,  
  "radius" : 50,  
  "cores_amount" : 2,  
  "trials": 1,  
  "audio": "../samples/sample.wav",  
  "microphone_amount": 32  
}
```

Рисунок 4.26 - Скріншот параметрів запуску

```
11.100197076797485 passed for localization computation trial.  
Localized source.  
64.017364025115967 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -4.197438979636146, Estimated Y = -2.731280155952916, Estimated Z = 5.951381838966640,  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.27 - Скріншот результатів роботи на Dell Inspiron 3537

```
7.016196012496948 passed for localization computation trial.  
Localized source.  
60.197953939437866 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -4.197438979636116, Estimated Y = -2.731280155952861, Estimated Z = 5.951381838966631  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.28 - Скріншот результатів роботи на Dell XPS 15

Для випадку 32 мікрофонів та 4 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.29, 4.30 та 4.31 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```
{  
  "server_address" : "localhost",  
  "server_port" : 10000,  
  "radius" : 50,  
  "cores_amount" : 4,  
  "trials": 1,  
  "audio": "../samples/sample.wav",  
  "microphone_amount": 32  
}
```

Рисунок 4.29 - Скріншот параметрів запуску

```
9.883577108383179 passed for localization computation trial.
Localized source.
62.537369012832642 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.197438979636146, Estimated Y = -2.731280155952916, Estimated Z = 5.951381838966640
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.30 - Скріншот результатів роботи на Dell Inspiron 3537

```
3.572170019149780 passed for localization computation trial.
Localized source.
56.719645023345947 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.197438979636116, Estimated Y = -2.731280155952861, Estimated Z = 5.951381838966631
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.31 - Скріншот результатів роботи на Dell XPS 15

Для випадку 32 мікрофонів та 8 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.32, 4.33 можна побачити параметри та результати роботи програми для цього випадку.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 8,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 32
}
```

Рисунок 4.32 - Скріншот параметрів запуску

```
3.572170019149780 passed for localization computation trial.
Localized source.
56.719645023345947 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -4.197438979636116, Estimated Y = -2.731280155952861, Estimated Z = 5.951381838966631
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.33 - Скріншот результатів роботи на Dell XPS 15

Для випадку 64 мікрофонів та 1 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.34, 4.35 та 4.36 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```

{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 1,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 64
}

```

Рисунок 4.34 - Скріншот параметрів запуску

```

42.535079002380371 passed for localization computation trial.
Localized source.
148.556290149688721 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.742313891921427, Estimated Y = -1.775181449961508, Estimated Z = 5.607154695896966
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.35 - Скріншот результатів роботи на Dell Inspiron 3537

```

26.600839853286743 passed for localization computation trial.
Localized source.
132.991595029830933 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.742313891921384, Estimated Y = -1.775181449961408, Estimated Z = 5.607154695896959
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.36 - Скріншот результатів роботи на Dell XPS 15

Для випадку 64 мікрофонів та 2 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.37, 4.38 та 4.39 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```

{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 2,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 64
}

```

Рисунок 4.37 - Скріншот параметрів запуску

```

22.617599010467529 passed for localization computation trial.
Localized source.
128.239194869995117 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.742313891921427, Estimated Y = -1.775181449961508, Estimated Z = 5.607154695896966
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.38 - Скріншот результатів роботи на Dell Inspiron 3537


```
14.262178182601929 passed for localization computation trial.
Localized source.
120.561423063278198 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.742313891921384, Estimated Y = -1.775181449961408, Estimated Z = 5.607154695896959
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.39 - Скріншот результатів роботи на Dell XPS 15

Для випадку 64 мікрофонів та 4 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.40, 4.41 та 4.42 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 4,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 64
}
```

Рисунок 4.40 - Скріншот параметрів запуску

```
21.562765121459961 passed for localization computation trial.
Localized source.
127.366177082061768 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.742313891921427, Estimated Y = -1.775181449961508, Estimated Z = 5.607154695896966
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.41 - Скріншот результатів роботи на Dell Inspiron 3537

```
7.136808156967163 passed for localization computation trial.
Localized source.
113.351583003997803 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.742313891921384, Estimated Y = -1.775181449961408, Estimated Z = 5.607154695896959
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.42 - Скріншот результатів роботи на Dell XPS 15

Для випадку 64 мікрофонів та 8 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.43, 4.44 можна побачити параметри та результати роботи програми для цього випадку.


```

{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 8,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 64
}

```

Рисунок 4.43 - Скріншот параметрів запуску

```

6.215605974197388 passed for localization computation trial.
Localized source.
112.456063032150269 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.742313891921384, Estimated Y = -1.775181449961408, Estimated Z = 5.607154695896959
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.44 - Скріншот результатів роботи на Dell XPS 15

Для випадку 128 мікрофонів та 1 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.45, 4.46 та 4.47 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```

{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 1,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 128
}

```

Рисунок 4.45 - Скріншот параметрів запуску

```

85.582956790924072 passed for localization computation trial.
Localized source.
296.937488794326782 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -3.303489956726679, Estimated Y = -2.145837278650664, Estimated Z = 5.741640554606016
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.46 - Скріншот результатів роботи на Dell Inspiron 3537

```

53.453463077545166 passed for localization computation trial.
Localized source.
265.953993082046509 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -3.303489956727057, Estimated Y = -2.145837278650790, Estimated Z = 5.741640554606063
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.47 - Скріншот результатів роботи на Dell XPS 15

Для випадку 128 мікрофонів та 2 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.48, 4.49 та 4.50 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```
{  
  "server_address" : "localhost",  
  "server_port" : 10000,  
  "radius" : 50,  
  "cores_amount" : 2,  
  "trials": 1,  
  "audio": "../samples/sample.wav",  
  "microphone_amount": 128  
}
```

Рисунок 4.48 - Скріншот параметрів запуску

```
53.101605892181396 passed for localization computation trial.  
Localized source.  
264.547935962677002 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -3.303489956726679, Estimated Y = -2.145837278650664, Estimated Z = 5.741640554606016  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.49 - Скріншот результатів роботи на Dell Inspiron 3537

```
28.435987949371338 passed for localization computation trial.  
Localized source.  
240.974565029144287 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -3.303489956727057, Estimated Y = -2.145837278650790, Estimated Z = 5.741640554606063  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.50 - Скріншот результатів роботи на Dell XPS 15

Для випадку 128 мікрофонів та 4 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.51, 4.52 та 4.53 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```

{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 4,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 128
}

```

Рисунок 4.51 - Скріншот параметрів запуску

```

46.751853942871094 passed for localization computation trial.
Localized source.
257.784079074859619 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -3.303489956726679, Estimated Y = -2.145837278650664, Estimated Z = 5.741640554606016
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.52 - Скріншот результатів роботи на Dell Inspiron 3537

```

14.321412086486816 passed for localization computation trial.
Localized source.
226.598459005355835 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -3.303489956727057, Estimated Y = -2.145837278650790, Estimated Z = 5.741640554606063
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.53 - Скріншот результатів роботи на Dell XPS 15

Для випадку 128 мікрофонів та 8 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.54, 4.55 можна побачити параметри та результати роботи програми для цього випадку.

```

{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 8,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 128
}

```

Рисунок 4.54 - Скріншот параметрів запуску

```

12.414695024490356 passed for localization computation trial.
Localized source.
225.111600875854492 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -3.303489956727057, Estimated Y = -2.145837278650790, Estimated Z = 5.741640554606063
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.55 - Скріншот результатів роботи на Dell XPS 15

Для випадку 256 мікрофонів та 1 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.56, 4.57 та 4.58 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```
{  
  "server_address" : "localhost",  
  "server_port" : 10000,  
  "radius" : 50,  
  "cores_amount" : 1,  
  "trials": 1,  
  "audio": "../samples/sample.wav",  
  "microphone_amount": 256  
}
```

Рисунок 4.56 - Скріншот параметрів запуску

```
174.335762023925781 passed for localization computation trial.  
Localized source.  
597.266427993774414 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -0.842960873665933, Estimated Y = -0.551188487809368, Estimated Z = 5.211343246749102  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.57 - Скріншот результатів роботи на Dell Inspiron 3537

```
106.237064838409424 passed for localization computation trial.  
Localized source.  
531.317841053009033 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -0.842960873665981, Estimated Y = -0.551188487809398, Estimated Z = 5.211343246749111  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.58 - Скріншот результатів роботи на Dell XPS 15

Для випадку 256 мікрофонів та 2 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.59, 4.60 та 4.61 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```

{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 2,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 256
}

```

Рисунок 4.59 - Скріншот параметрів запуску

```

111.290722846984863 passed for localization computation trial.
Localized source.
534.237853050231934 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -0.842960873665933, Estimated Y = -0.551188487809368, Estimated Z = 5.211343246749102
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.60 - Скріншот результатів роботи на Dell Inspiron 3537

```

57.253488063812256 passed for localization computation trial.
Localized source.
482.360838174819946 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -0.842960873665981, Estimated Y = -0.551188487809398, Estimated Z = 5.211343246749111
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.61 - Скріншот результатів роботи на Dell XPS 15

Для випадку 256 мікрофонів та 4 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.62, 4.63 та 4.64 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```

{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 4,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 256
}

```

Рисунок 4.62 - Скріншот параметрів запуску

```

102.342792987823486 passed for localization computation trial.
Localized source.
525.220792055130005 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -0.842960873665933, Estimated Y = -0.551188487809368, Estimated Z = 5.211343246749102
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.63 - Скріншот результатів роботи на Dell Inspiron 3537

```
28.567512035369873 passed for localization computation trial.
Localized source.
453.597218036651611 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -0.842960873665981, Estimated Y = -0.551188487809398, Estimated Z = 5.21134324674911
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.64 - Скріншот результатів роботи на Dell XPS 15

Для випадку 256 мікрофонів та 8 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.65, 4.66 можна побачити параметри та результати роботи програми для цього випадку.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 8,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 256
}
```

Рисунок 4.65 - Скріншот параметрів запуску

```
27.016688108444214 passed for localization computation trial.
Localized source.
451.305165052413940 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -0.842960873665981, Estimated Y = -0.551188487809398, Estimated Z = 5.211343246749111
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.66 - Скріншот результатів роботи на Dell XPS 15

Для випадку 512 мікрофонів та 1 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.67, 4.68 та 4.69 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 1,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 512
}
```

Рисунок 4.67 - Скріншот параметрів запуску

```
349.592906951904297 passed for localization computation trial.
Localized source.
1195.250949144363403 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -1.474199022342221, Estimated Y = -0.960171905576400, Estimated Z = 5.355927474342941
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.68 - Скріншот результатів роботи на Dell Inspiron 3537

```
207.991411924362183 passed for localization computation trial.
Localized source.
1058.233165979385376 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -1.474199022342230, Estimated Y = -0.960171905576374, Estimated Z = 5.355927474342956
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.69 - Скріншот результатів роботи на Dell XPS 15

Для випадку 512 мікрофонів та 2 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.70, 4.71 та 4.72 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 2,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 512
}
```

Рисунок 4.70 - Скріншот параметрів запуску

```
228.803642034530640 passed for localization computation trial.
Localized source.
1074.792706966400146 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -1.474199022342221, Estimated Y = -0.960171905576400, Estimated Z = 5.355927474342941
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.71 - Скріншот результатів роботи на Dell Inspiron 3537

```
112.329607963562012 passed for localization computation trial.
Localized source.
962.319138050079346 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -1.474199022342230, Estimated Y = -0.960171905576374, Estimated Z = 5.355927474342956
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.72 - Скріншот результатів роботи на Dell XPS 15

Для випадку 512 мікрофонів та 4 процесу можна побачити процес

формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.73, 4.74 та 4.75 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```
{  
  "server_address" : "localhost",  
  "server_port" : 10000,  
  "radius" : 50,  
  "cores_amount" : 4,  
  "trials": 1,  
  "audio": "../samples/sample.wav",  
  "microphone_amount": 512  
}
```

Рисунок 4.73 - Скріншот параметрів запуску

```
206.816395998001099 passed for localization computation trial.  
Localized source.  
1052.396707057952881 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -1.474199022342221, Estimated Y = -0.960171905576400, Estimated Z = 5.355927474342941  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.74 - Скріншот результатів роботи на Dell Inspiron 3537

```
58.014392137527466 passed for localization computation trial. I  
Localized source.  
907.732805967330933 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -1.474199022342230, Estimated Y = -0.960171905576374, Estimated Z = 5.355927474342956  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.75 - Скріншот результатів роботи на Dell XPS 15

Для випадку 512 мікрофонів та 8 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.76, 4.77 можна побачити параметри та результати роботи програми для цього випадку.

```
{  
  "server_address" : "localhost",  
  "server_port" : 10000,  
  "radius" : 50,  
  "cores_amount" : 8,  
  "trials": 1,  
  "audio": "../samples/sample.wav",  
  "microphone_amount": 512  
}
```

Рисунок 4.76 - Скріншот параметрів запуску


```
51.616873979568481 passed for localization computation trial.
Localized source.
902.644188880920410 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -1.474199022342230, Estimated Y = -0.960171905576374, Estimated Z = 5.355927474342956
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.77 - Скріншот результатів роботи на Dell XPS 15

Для випадку 1024 мікрофонів та 1 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.78, 4.79 та 4.80 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 1,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 1024
}
```

Рисунок 4.78 - Скріншот параметрів запуску

```
699.888427972793579 passed for localization computation trial.
Localized source.
2391.163513898849487 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.006556927551303, Estimated Y = -1.306584572933023, Estimated Z = 5.473948780301037
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.79 - Скріншот результатів роботи на Dell Inspiron 3537

```
417.745423078536987 passed for localization computation trial.
Localized source.
2090.149323940277100 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.006556927551348, Estimated Y = -1.306584572933026, Estimated Z = 5.473948780301031
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.80 - Скріншот результатів роботи на Dell XPS 15

Для випадку 1024 мікрофонів та 2 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.81, 4.82 та 4.83 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```

{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 2,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 1024
}

```

Рисунок 4.81 - Скріншот параметрів запуску

```

455.682923078536987 passed for localization computation trial.
Localized source.
2147.428322076797485 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.006556927551303, Estimated Y = -1.306584572933023, Estimated Z = 5.473948780301031
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.82 - Скріншот результатів роботи на Dell XPS 15

```

230.259033918380737 passed for localization computation trial.
Localized source.
1901.863492965698242 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.006556927551348, Estimated Y = -1.306584572933026, Estimated Z = 5.473948780301031
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.83 - Скріншот результатів роботи на Dell XPS 15

Для випадку 1024 мікрофонів та 4 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.84, 4.85 та 4.86 можна побачити параметри та результати роботи програми для цього випадку на двох комп'ютерах.

```

{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 4,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 1024
}

```

Рисунок 4.84 - Скріншот параметрів запуску

```
416.479431152343750 passed for localization computation trial.
Localized source.
2107.808966159820557 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.006556927551303, Estimated Y = -1.306584572933023, Estimated Z = 5.473948780301037
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.85 - Скріншот результатів роботи на Dell Inspiron 3537

```
116.731297016143799 passed for localization computation trial.
Localized source.
1817.789361953735352 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -2.006556927551348, Estimated Y = -1.306584572933026, Estimated Z = 5.473948780301031
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.86 - Скріншот результатів роботи на Dell XPS 15

Для випадку 1024 мікрофонів та 8 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.87, 4.88 можна побачити параметри та результати роботи програми для цього випадку.

```
{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 8,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 1024
}
```

Рисунок 4.87 - Скріншот параметрів запуску

```
58.014392137527466 passed for localization computation trial.
Localized source.
907.732805967330933 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -1.474199022342230, Estimated Y = -0.960171905576374, Estimated Z = 5.355927474342956
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.88 - Скріншот результатів роботи на Dell XPS 15

Для випадку 2048 мікрофонів та 1 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.89, 4.90 можна побачити параметри та результати роботи програми для цього випадку.

```

{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 1,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 2048
}

```

Рисунок 4.89 - Скріншот параметрів запуску

```

850.097900152206421 passed for localization computation trial.
Localized source.
4250.403235197067261 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -1.781397791554035, Estimated Y = -1.159582133693149, Estimated Z = 5.420255897206768
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.90 - Скріншот результатів роботи на Dell XPS 15

Для випадку 2048 мікрофонів та 2 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.91, 4.92 можна побачити параметри та результати роботи програми для цього випадку.

```

{
  "server_address" : "localhost",
  "server_port" : 10000,
  "radius" : 50,
  "cores_amount" : 2,
  "trials": 1,
  "audio": "../samples/sample.wav",
  "microphone_amount": 2048
}

```

Рисунок 4.91 - Скріншот параметрів запуску

```

456.547212123870850 passed for localization computation trial.
Localized source.
3859.102652072906494 passed for SEND/RECEIVE/CALCULATE.
Trial number: 1
Estimated X = -1.781397791554035, Estimated Y = -1.159582133693149, Estimated Z = 5.420255897206768
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000

```

Рисунок 4.92 - Скріншот результатів роботи на Dell XPS 15

Для випадку 2048 мікрофонів та 4 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.93, 4.94 можна побачити

параметри та результати роботи програми для цього випадку.

```
{  
  "server_address" : "localhost",  
  "server_port" : 10000,  
  "radius" : 50,  
  "cores_amount" : 4,  
  "trials": 1,  
  "audio": "../samples/sample.wav",  
  "microphone_amount": 2048  
}
```

Рисунок 4.93 - Скріншот параметрів запуску

```
237.611618995666504 passed for localization computation trial.  
Localized source.  
3638.051261186599731 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -1.781397791554035, Estimated Y = -1.159582133693149, Estimated Z = 5.420255897206768  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.94 - Скріншот результатів роботи на Dell XPS 15

Для випадку 2048 мікрофонів та 8 процесу можна побачити процес формування запиту на сервер, результат роботи сервера та отримані реальні та визначені координати джерела. На рисунках 4.95, 4.96 можна побачити параметри та результати роботи програми для цього випадку.

```
{  
  "server_address" : "localhost",  
  "server_port" : 10000,  
  "radius" : 50,  
  "cores_amount" : 8,  
  "trials": 1,  
  "audio": "../samples/sample.wav",  
  "microphone_amount": 2048  
}
```

Рисунок 4.95 - Скріншот параметрів запуску

```
213.687644958496094 passed for localization computation trial.  
Localized source.  
3615.827843904495239 passed for SEND/RECEIVE/CALCULATE.  
Trial number: 1  
Estimated X = -1.781397791554035, Estimated Y = -1.159582133693149, Estimated Z = 5.420255897206768  
True X = -4.195357645382262, True Y = -2.720105554446849, True Z = 6.000000000000000
```

Рисунок 4.96 - Скріншот результатів роботи на Dell XPS 15

4.3 Аналіз отриманих результатів

Результати перевірки створених програми на двох комп'ютерах, дозволяють побачити суттєву різницю у часі виконання. Можна чітко

побачити, що зі збільшенням кількості мікрофонів програма, що використовує для обчислення найповільнішої частини від більше одного процесу, отримує значну перевагу у часі виконання. При використанні можливості обчислень за допомогою технології Intel Hyper-threading можна побачити незначне покращення результатів, а саме близько 7%, в порівнянні з використанням реальної кількості потоків виконання процесорів. Час виконання та порівняння можна побачити на рисунках 4.97, 4.98 та 4.100 – 4.103.

При цьому варто зазначити, що час попередньої обробки повідомлень, зменшився на 1% з використанням потужнішого процесору. Графіки залежності часу від кількості мікрофонів, з котрих оброблюються дані можна побачити на рисунках 4.98 та 4.49.

Окрім цього, також можна помітити різницю в точності визначення координат джерела, що зумовлена особливостями в створенні мікрофонів для аналізу звуку, а саме їх розташування відносно джерела звуку. Залежність неточності від кількості мікрофонів з котрих оброблюється сигнал можна побачити на рисунках 4.104 – 4.106.

Експериментальним шляхом була визначена максимально доцільна кількість мікрофонів, що можна використовувати для локалізації джерела акустичного сигналу, а саме – 1024 для Dell Inspiron 3537 (2/4) та 2048 для Dell XPS 15 (4/8).

Слід зауважити, що зі збільшенням розміру даних надісланих з мікрофонів для аналізу, збільшення кількості процесів для виконання обчислень, одзволяє значно поліпшити час локалізації на обох комп'ютерах, що зумовлене виконанням більшої кількості операції паралельно.

Отримані результати підтверджують правильність імплементації розподілення обчислень, що може дозволити покращити результати при використанні більшої кількості фізичних яду процесора.

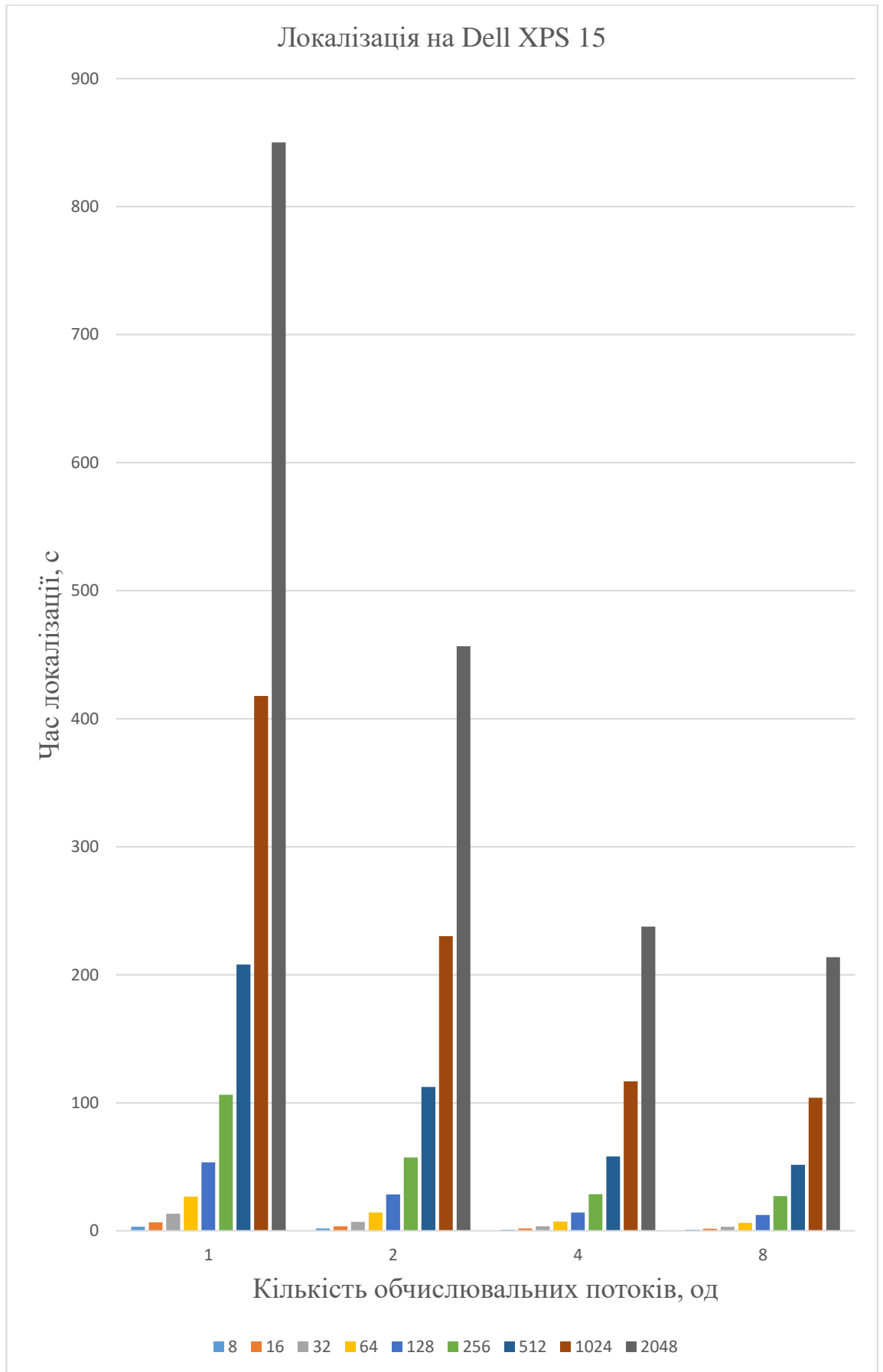


Рисунок 4.98 - Залежність часу локалізації від кількості мікрофонів на Dell XPS 15

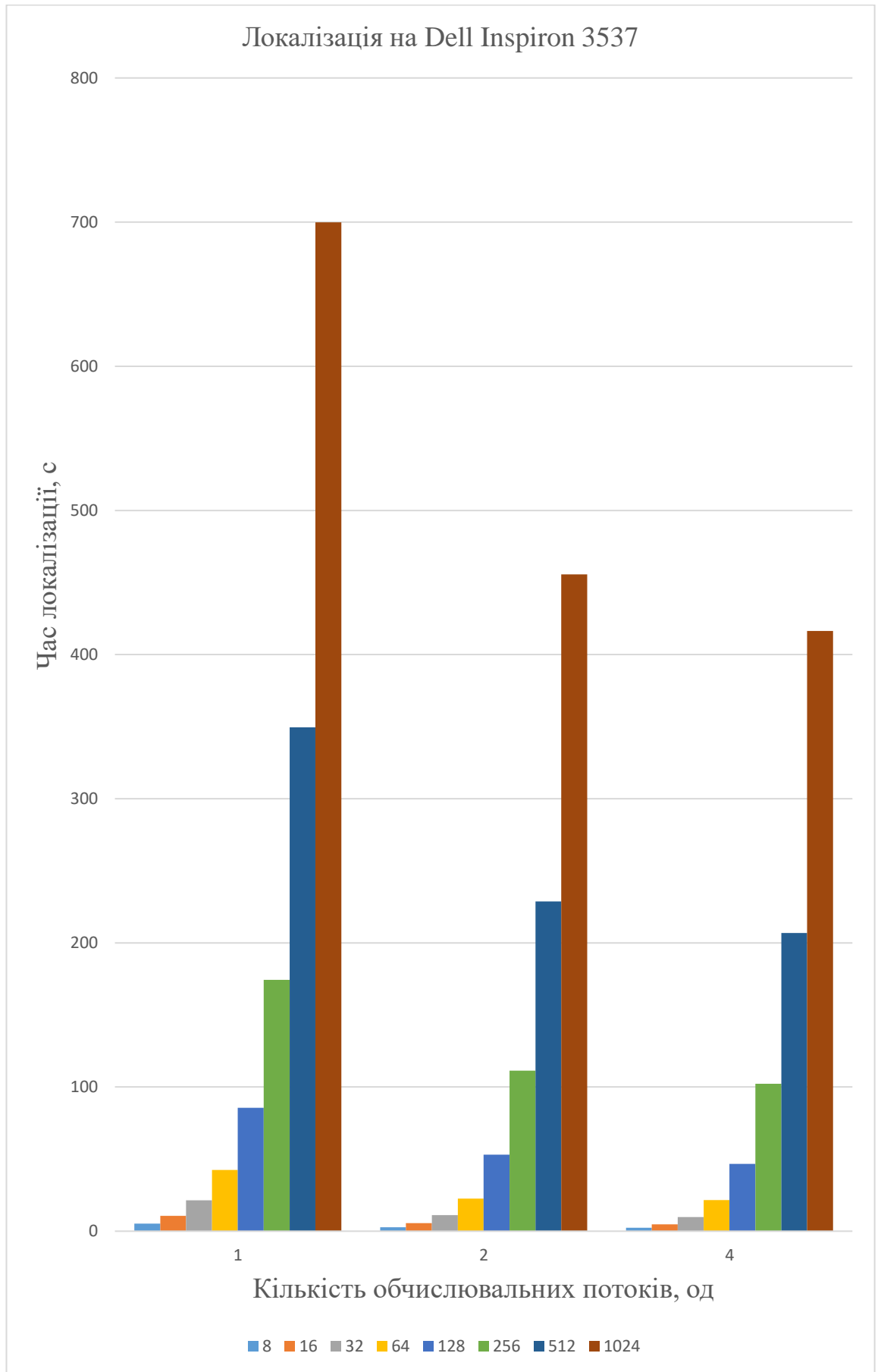


Рисунок 4.97 - Залежність часу локалізації від кількості мікрофонів на Dell Inspiron 3537

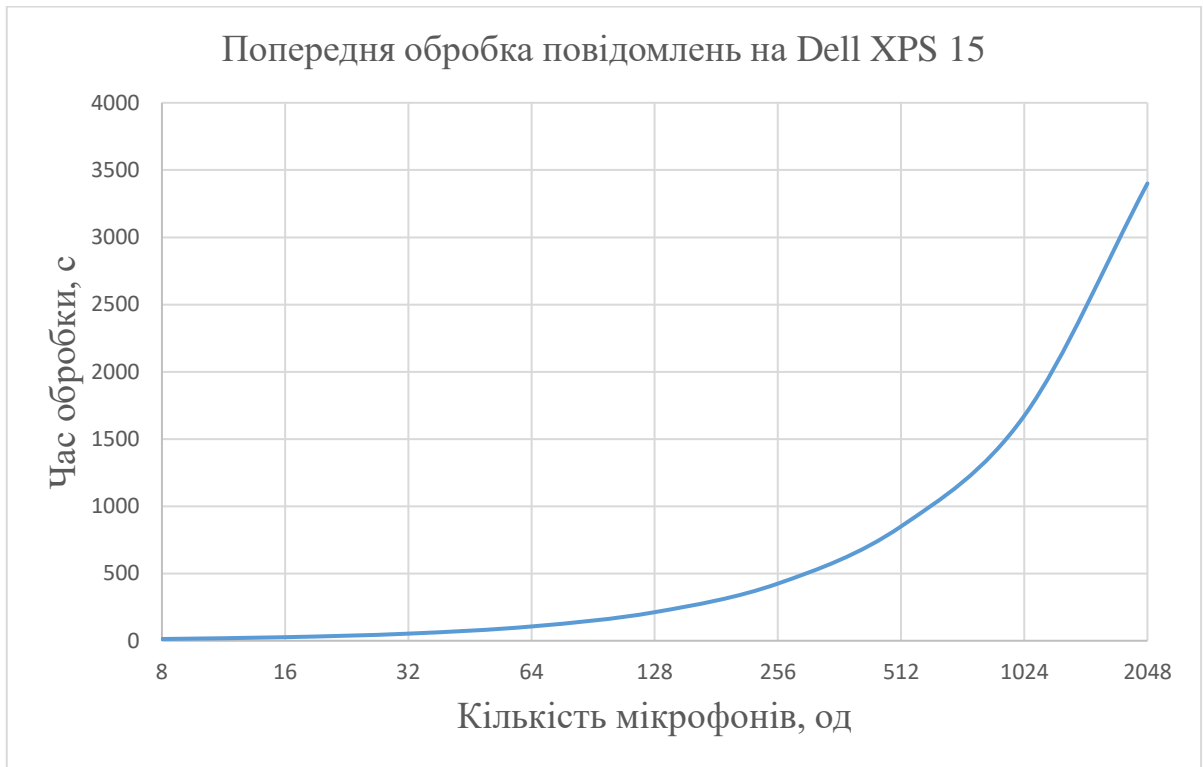


Рисунок 4.98 - Залежність часу обробки повідомлень від кількості мікрофонів на Dell Inspiron 3537

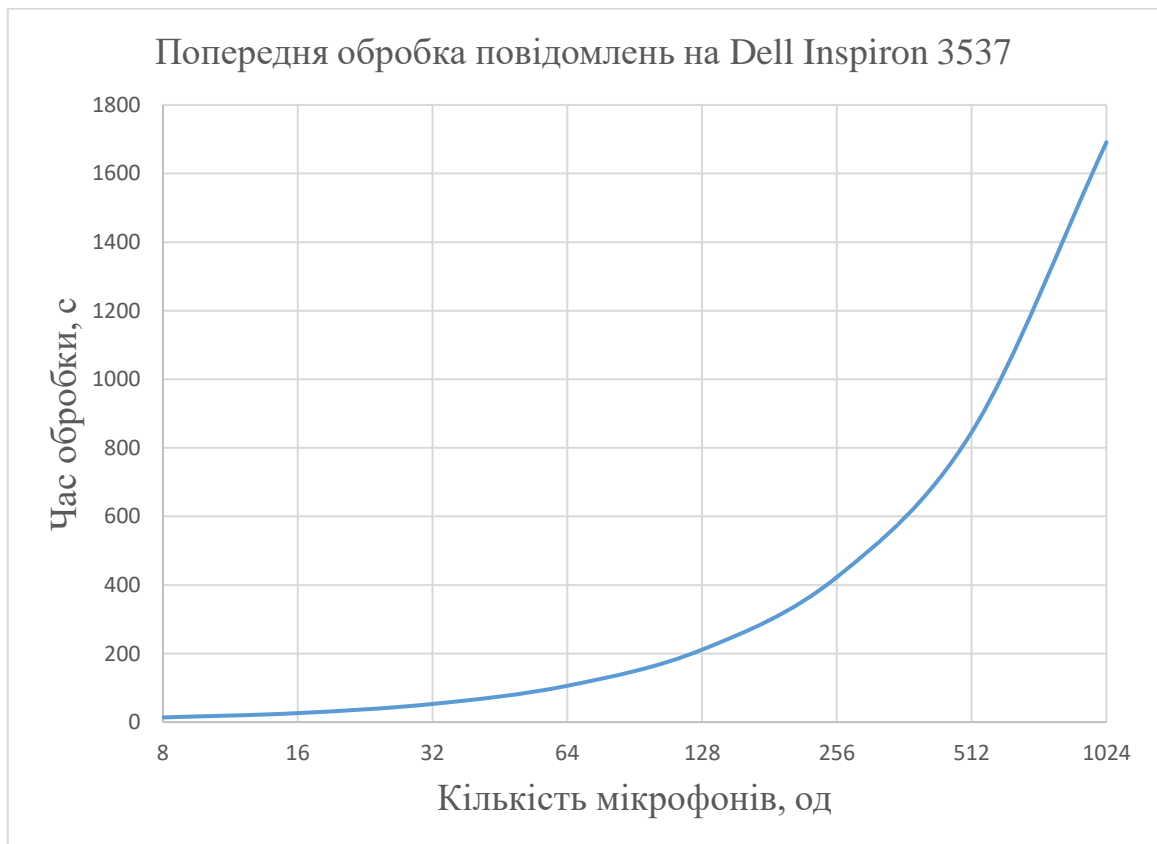


Рисунок 4.99 - Залежність часу обробки повідомлень від кількості мікрофонів на Dell XPS 15

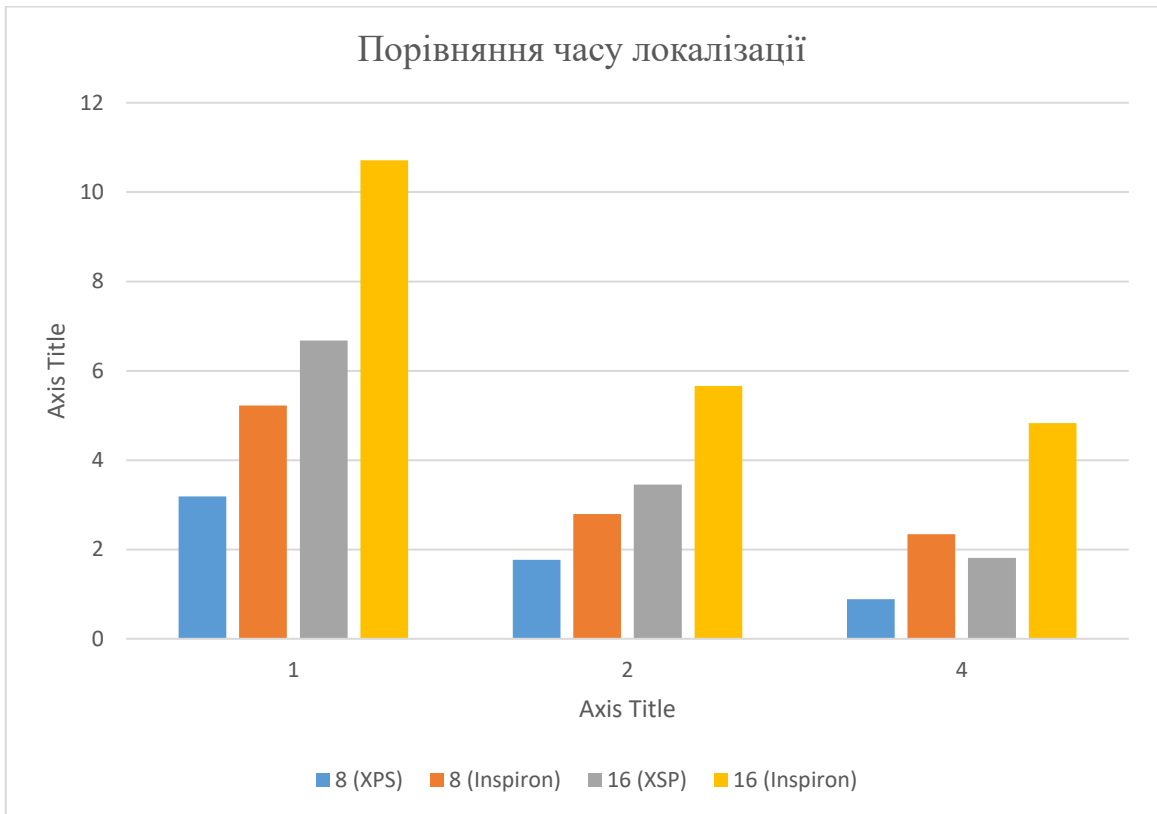


Рисунок 4.100 - Порівняння часу локалізації для відповідних випадків

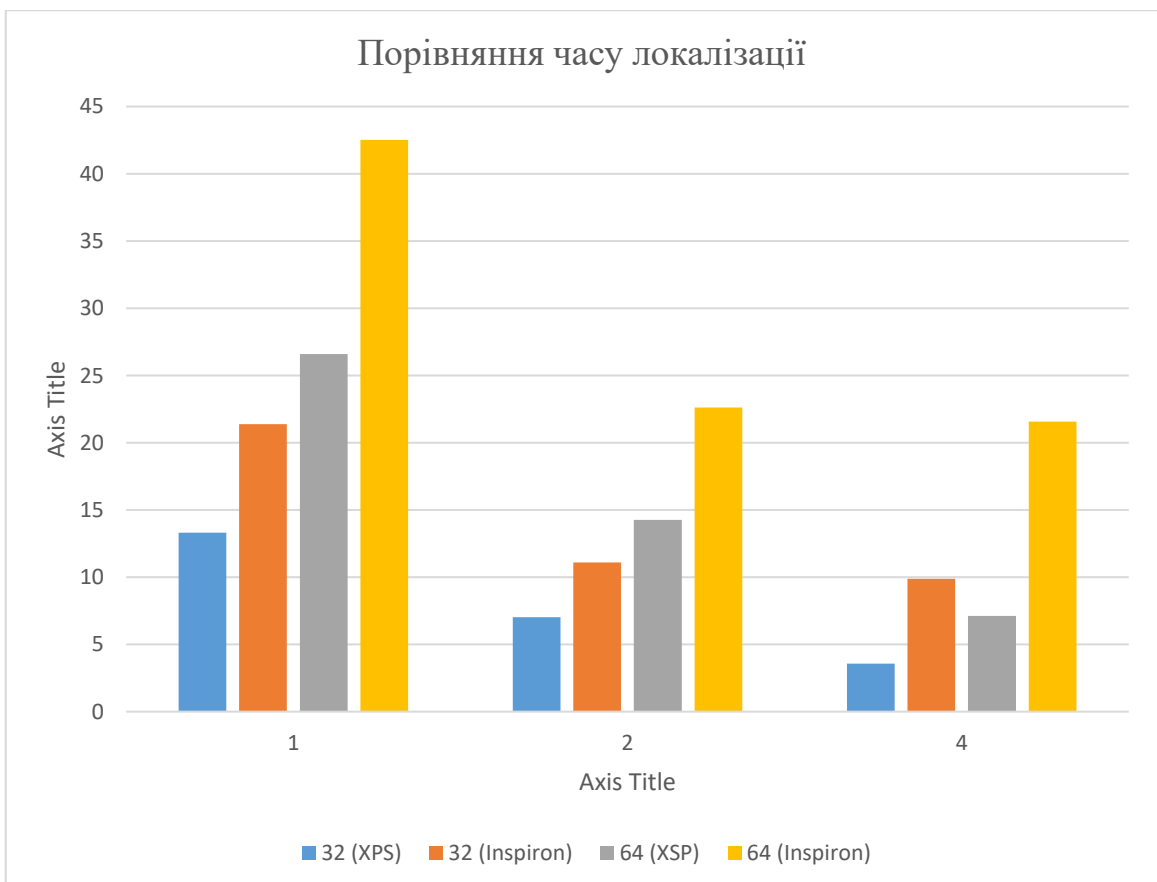


Рисунок 4.101 - Порівняння часу локалізації для відповідних випадків

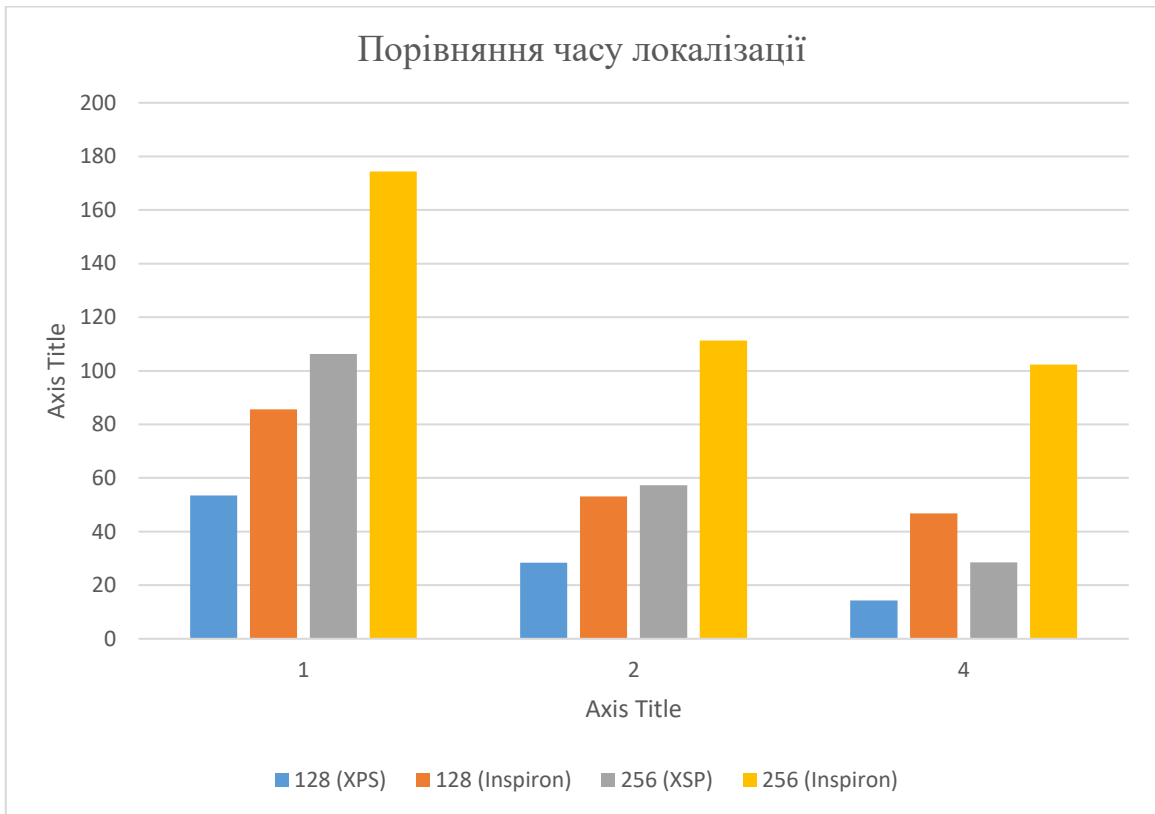


Рисунок 4.102 - Порівняння часу локалізації для відповідних випадків

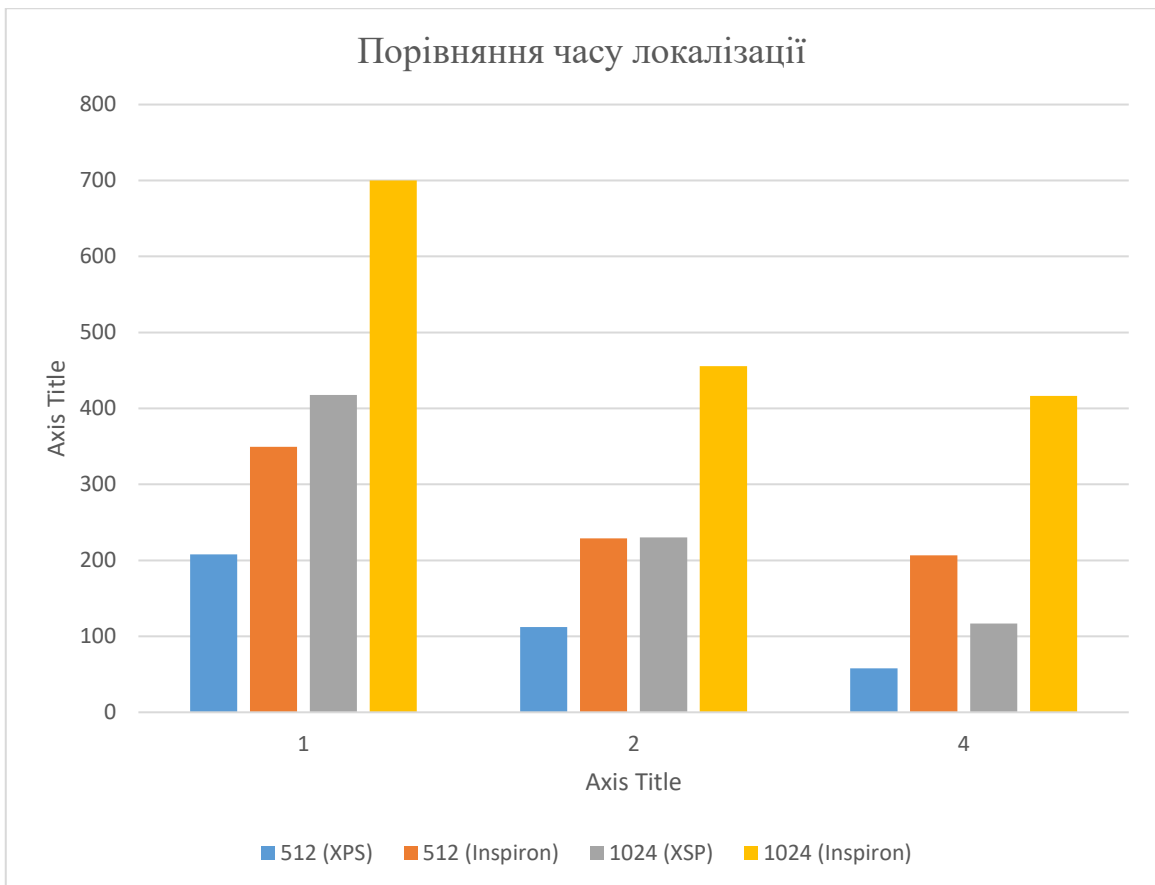


Рисунок 4.103 - Порівняння часу локалізації для відповідних випадків



Рисунок 4.104 - Неточність по X

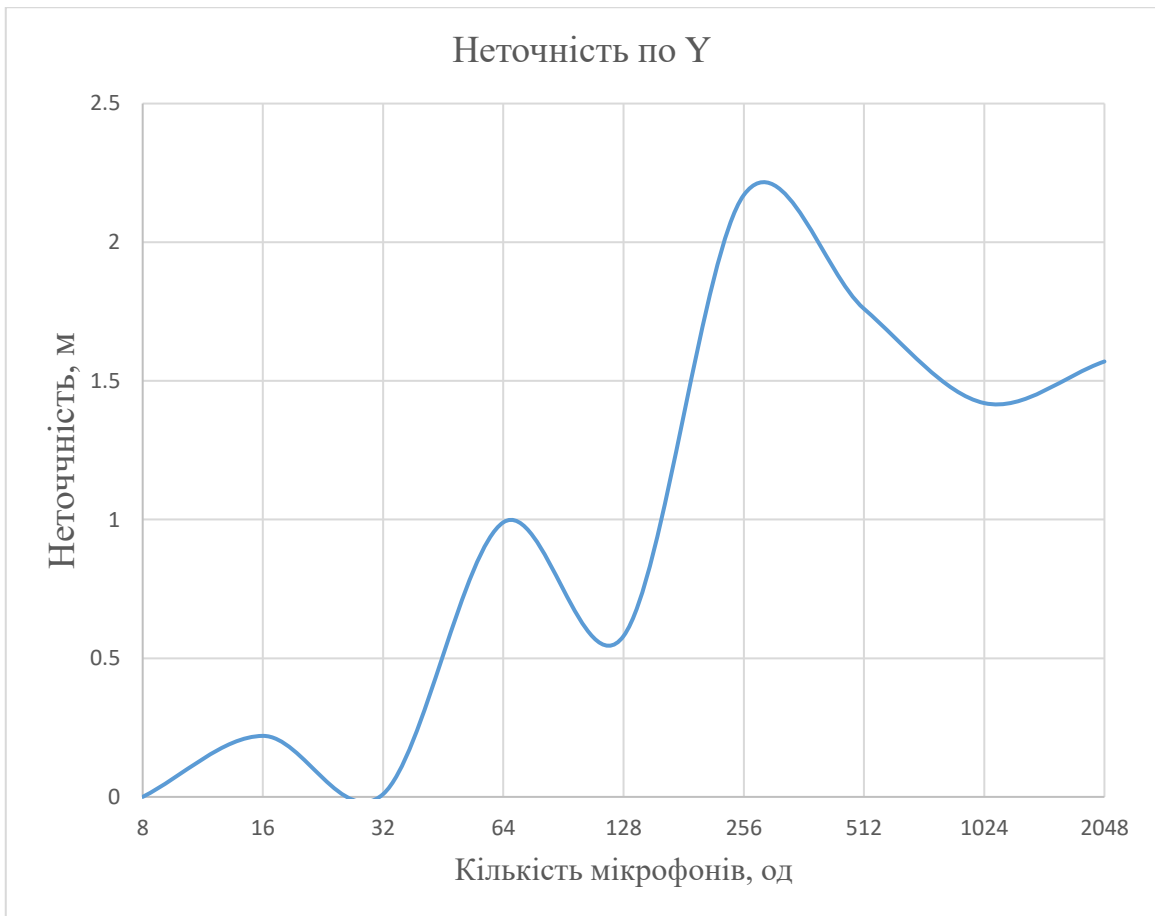


Рисунок 4.105 - Неточність по Y

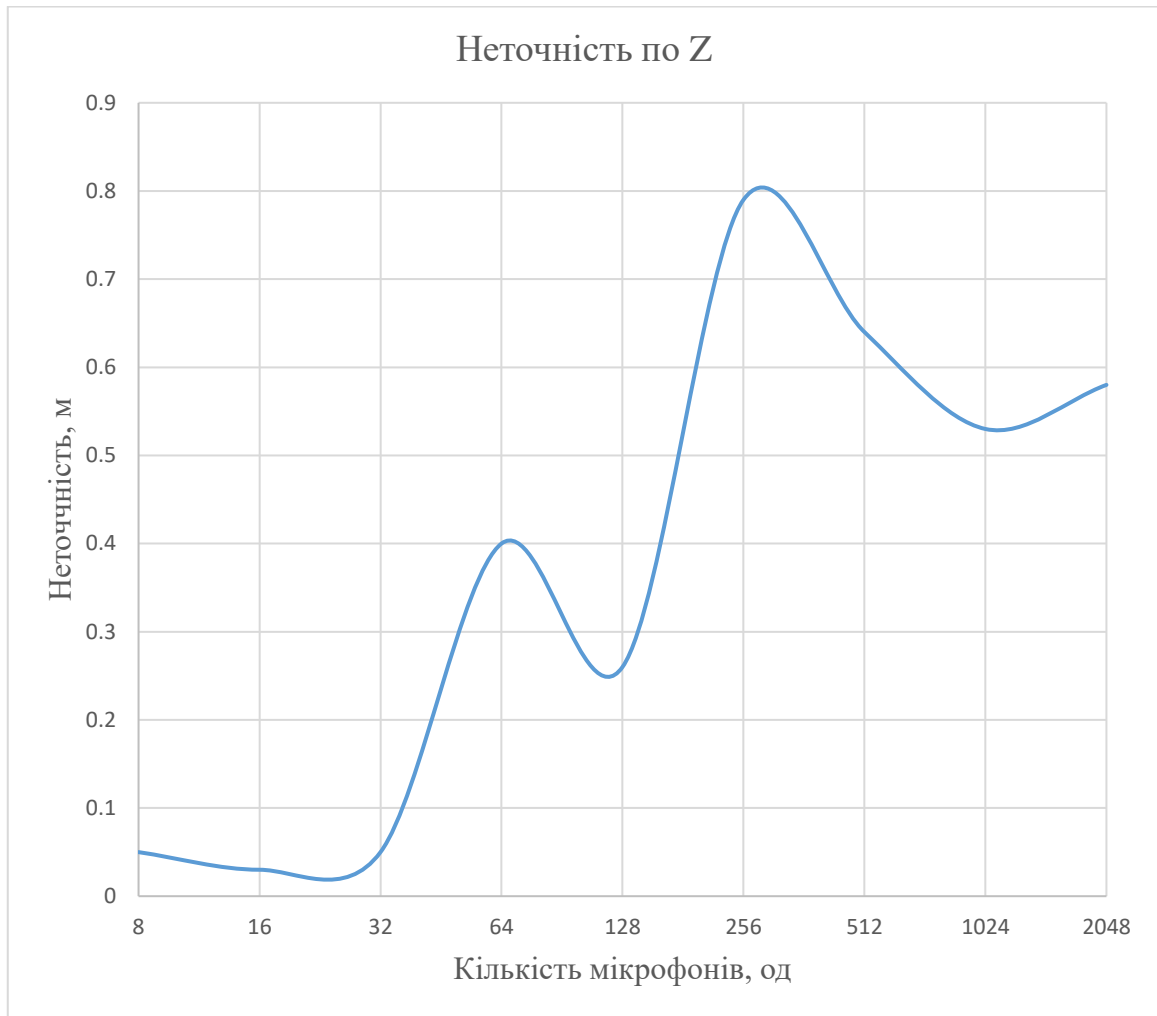


Рисунок 4.106 - Неточність по Z

4.4 Висновки

Для досліджень роботи моделі локалізації джерела акустичного сигналу за допомогою методу TDOA було розроблено сервер та необхідні клієнти що дозволяють проводити експерименти з бажаними аудіо файлами, кількістю спроб для визначення місцеположення джерела звуку, радіусом локалізації, адресу та порт серверу для дослідження, кількістю мікрофонів що будуть використовуватися та кількістю процесів на яких буде виконуватися частина алгоритму, паралелізація якої дозволяє значно покращити результати при використанні більшої кількості процесів та фізичних чи віртуальних ядер центрального процесору.

При порівнянні створених версій програми було виявлено, що в

залежності від вибраної кількості процесів, різниця в часі виконання складати до 50%. Для вищезгаданих комп'ютерів убли визначені максимальні кількості мікрофонів дані за яких можна обробити, а саме 1024 та 2048. Для більш простих досліджень має сенс використовувати моделювання на локальній машині, що може забезпечити необхідну простоту в роботі.

Також варто зауважити, що при роботі з розробленою моделлю необхідно враховувати можливості комп'ютера на якому відбуваються дослідження, оскільки максимальний розмір аудіо файлу для аналізу та кількість мікрофонів що використовуються залежить від обсягу оперативної пам'яті комп'ютера на якому відбуваються обчислення.

Було визначено, що кількість наявних обчислювальних ядр (фізичних та віртуальних) процесора комп'ютера на якому ведеться дослідження, може радикально вплинути на швидкодію в паралельній реалізації методу TDOA, тому для отримання якомога найкращих та якомога точніших результатів для виконання програми треба обирати їхню максимальну кількість – фізичних або при можливості віртуальних.

Було визначено, що кількість мікрофонів які можна моделювати та обчислювати дані з них одночасно прямо залежить від об'єму оперативної пам'яті яка доступна на комп'ютері або операційній системі, де відбувається моделювання.

Можна зробити висновок, що при використанні реальних мікрофонів залежності будуть аналогічні, також треба буде слідкувати за цілісністю даних з мікрофонів, що отримує сервер, оскільки протокол UDP не гарантує цілісність даних, що може вплинути на результати локалізації джерела акустичного сигналу, а саме точність.

Пакети з даними необхідно буде нумерувати та робити їх підтвердження, що може забезпечити надійність визначення місцеположення джерела звуку.

ВИСНОВКИ

Метою даної магістерської дисертації була розробка моделі локалізації джерела акустичного сигналу з використанням серверу що отримує дані за допомогою протоколу UDP та мікрофонів, з котрих відбувається надсилання даних, для визначення максимальної кількості мікрофонів, дані з яких може одночасно оброблювати сервер, з можливістю виконувати обчислення паралельно з використанням методу локалізації TDOA.

В якості варіанту методу TDOA був обраний варіант для локалізації джерела звуку за допомогою визначеної користувачем кількості мікрофонів та радіусу локалізації.

Аналіз засобів для розробки даного додатку показав доцільність використання для розробки мови програмування Python та додаткових бібліотек для математичних операцій. Для зручності роботи користувача та розширення його можливостей було розроблену гнучку систему конфігурування моделі.

Розроблена модель дозволяє:

- виконувати обчислення координат джерела звуку за допомогою обраного для аналізу аудіо файлу, визначеної кількості мікрофонів, заданої кількості спроб та обраної користувачем кількості процесів, на яких буде відбуватися робота частини алгоритму що можна паралелізувати;
- обирати варіанти конфігурації в залежності від його потреб та можливостей;
- отримати значне пришвидшення в швидкодії при визначенні місцеположення об'єкту при використанні більшої кількості наявних обчислювальних ядер (фізичних та віртуальних);

- досліджувати роботу методу TDOA з використанням клієнт-серверної архітектури в повному обсязі при наявності достатніх для цього ресурсів;
- демонструвати результати роботи у вигляді порівняння обчислених та заданих координат джерела звуку;
- обирати віддалений режим роботи для можливості більш широких досліджень.

Особливу увагу під час розроблення даного програмного продукту було приділено зручності роботи із системою та гнучкості її використання.

При порівнянні створених версій програми було виявлено, що в залежності від вибраної кількості процесів, різниця в часі виконання складає до 50% при використанні фізичних ядер процесора, що доводить результативність паралельної версії.

При використанні можливості обчислень за допомогою технології Intel Hyper-threading можна побачити незначне покращення результатів, а саме близько 7%, в порівнянні з використанням реальної кількості потоків виконання процесорів.

Експериментальним шляхом була визначена максимальна кількість мікрофонів, що можна використовувати для локалізації джерела акустичного сигналу на комп'ютерах Dell Inspiron 3537 та Dell XPS 15, а саме – 1024 для комп'ютера з 8 Гб оперативної пам'яті та 2048 для випадку 16 Гб.

Використання розробленої системи дозволить спростити та зробити більш ефективними дослідження методу TDOA з використанням клієнт-серверної архітектури з використанням протоколу UDP для передачі даних для будь якого користувача та його потреб, що дозволить значно пришвидшити дослідження та отримати вичерпні результати. Точність визначення місцеположення джерела звуку підтверджує коректність та точність імплементації методу TDOA.

Завдяки гнучкості створених додатків та можливостей їх використання вони можуть бути використані при більш детальному

дослідженні процесу роботи методу TDOA з використанням клієнт-серверної архітектури, при наявності комп'ютеру для тестування.

З невеликими модифікаціями створену модель можна використовувати для локалізації джерела акустичного сигналу в розподіленій сенсорній мережі для локалізації джерела акустичного сигналу, що дозволить ефективно та дешево виконувати поставлену задачу за допомогою методу TDOA та наявного устаткування.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

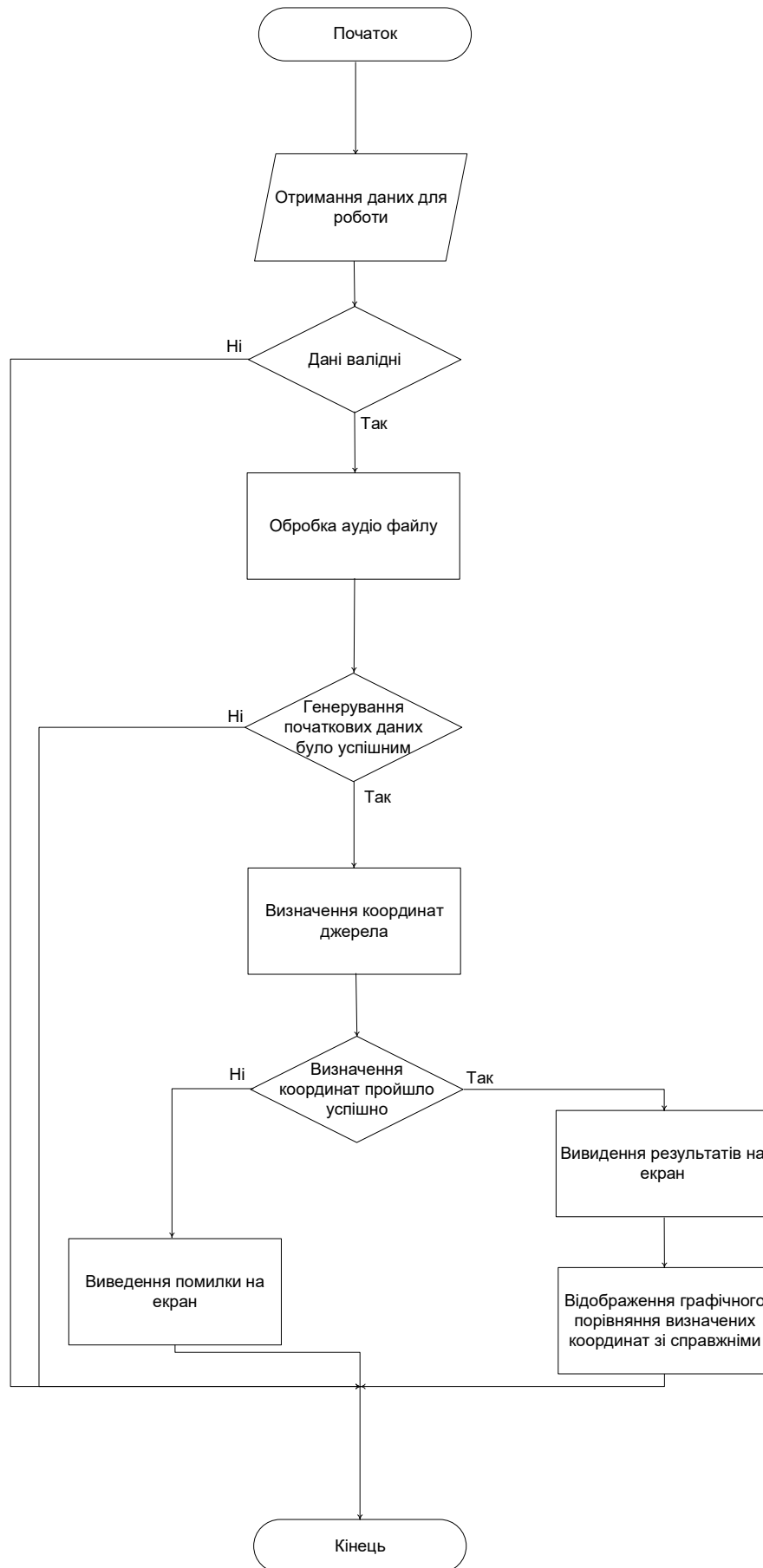
1. Sensor networks [Електронний ресурс]. Режим доступу: <https://bit.ly/2HOQc7L>. Дата доступу: квітень 2018.
2. Zigbee official website. [Електронний ресурс]. Режим доступу: <http://www.zigbee.org/>. Дата доступу: квітень 2018.
3. TIME OF ARRIVAL LOCATION TECHNIQUE. William Rison. 2008. [Електронний ресурс]. Режим доступу: http://www.ee.nmt.edu/~rison/ee389_spr08/toa.pdf.
4. TDOA Acoustic Localization. Steven Li. July 5, 2011. [Електронний ресурс]. Режим доступу: https://s3-us-west-1.amazonaws.com/stevenjl-bucket/tdoa_localization.pdf. Дата доступу: квітень 2018.
5. Performance Comparison of TOA and TDOA Based Location Estimation Algorithms in LOS Environment. Guowei Shen, Rudolf Zetik, and Reiner S. Thomä. PROCEEDINGS OF THE 5th WORKSHOP ON POSITIONING, NAVIGATION AND COMMUNICATION 2008 (WPNC'08). [Електронний ресурс]. Режим доступу: <http://www-emt.tu-ilmeneau.de/EMTPub/uploads/pdf/04510359.pdf>. Дата доступу: квітень 2018.
6. NumPy official website. [Електронний ресурс]. Режим доступу: <http://www.numpy.org/>. Дата доступу: квітень 2018.
7. JetBrains Strikes Python Developers with PyCharm IDE. [Електронний ресурс]. Режим доступу: <http://www.webcitation.org/6GfdVEK2i>. Дата доступу: квітень 2018
8. Matplotlib official website. [Електронний ресурс]. Режим доступу: <http://matplotlib.org/>. Дата доступу: квітень 2018.
9. SciPy official website. [Електронний ресурс]. Режим доступу: <https://www.scipy.org/about.html>. Дата доступу: квітень 2018.

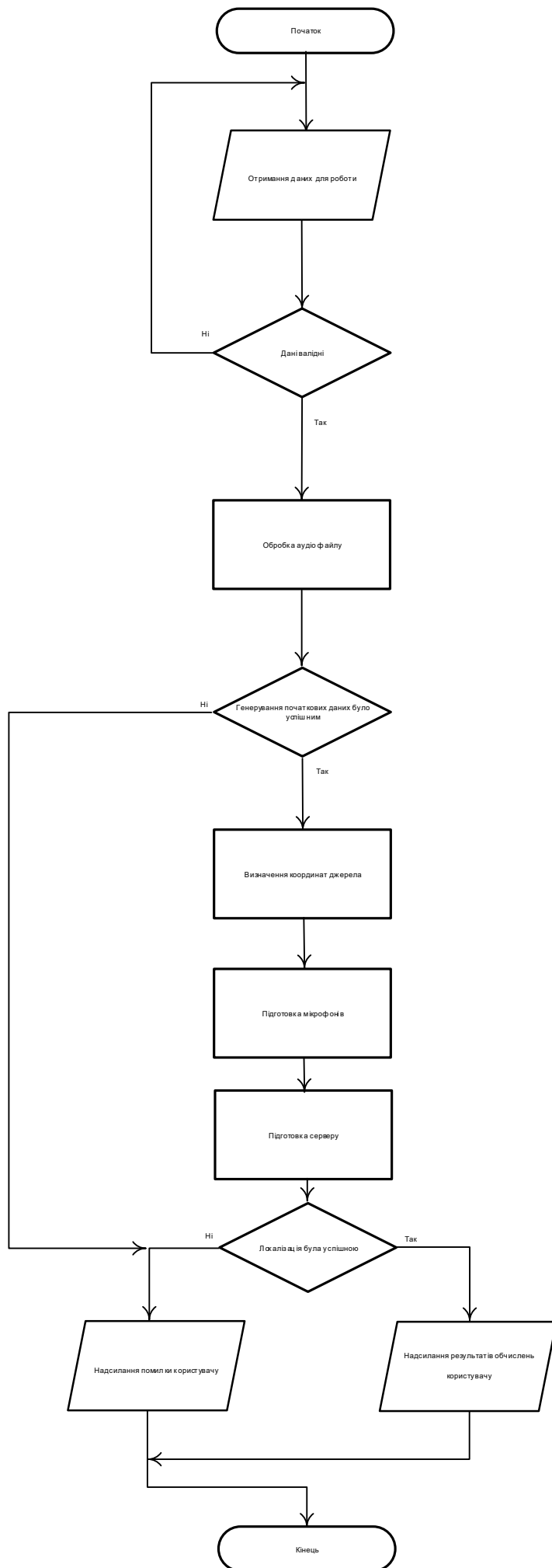
10. SciKits official website. [Електронний ресурс]. Режим доступу: <https://scikits.appspot.com/about>. Дата доступу: квітень 2018.
11. Python official documentation. [Електронний ресурс]. Режим доступу: <https://www.python.org/doc/>. Дата доступу: квітень 2018.
12. Introducing JSON. [Електронний ресурс]. Режим доступу: <https://www.json.org/>. Дата доступу: квітень 2018.
13. cPickle - A faster pickle. [Електронний ресурс]. Режим доступу: <https://docs.python.org/2.2/lib/module-cPickle.html>. Дата доступу: квітень 2018.
14. Intel Technology Journal. February 14, 2002. [Електронний ресурс]. Режим доступу: https://web.archive.org/web/20080224023922/http://download.intel.com/technology/itj/2002/volume06issue01/vol6iss1_hyper_threading_technology.pdf. Дата доступу: квітень 2018.
15. Git official website. [Електронний ресурс]. Режим доступу: <https://git-scm.com/>. Дата доступу: травень 2016.
16. UDP on MSDN Magazine Sockets and WCF [Електронний ресурс]. Режим доступу: <http://msdn.microsoft.com/en-us/magazine/cc163648.aspx> Дата доступу: квітень 2018.
17. UDP, User Datagram Protocol [Електронний ресурс]. <http://www.networksorcery.com/enp/protocol/udp.htm>. Дата доступу: квітень 2018.
18. Marshall Cline. "C++ FAQ: "What's this "serialization" thing all about?"". [Електронний ресурс]. Режим доступу: <https://web.archive.org/web/20150405013606/http://isocpp.org/wiki/faq/serialization>. Дата доступу: квітень 2018.
19. RFC 4122 — A Universally Unique Identifier (UUID) URN Namespace [Електронний ресурс]. Режим доступу: <https://tools.ietf.org/html/rfc4122>. Дата доступу: квітень 2018.

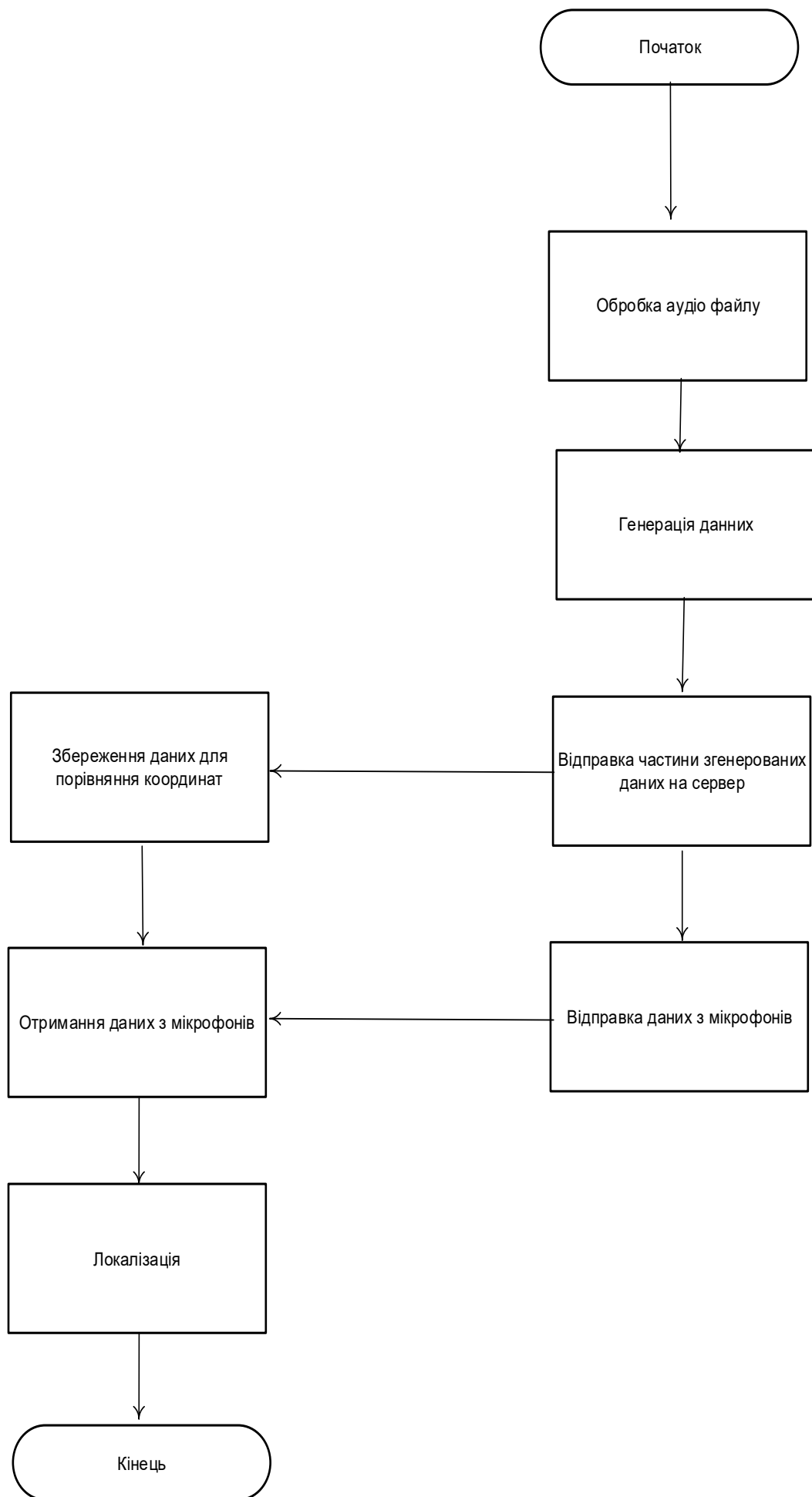
20. Global UUID registration function at ITU-T [Електронний ресурс].
Режим доступу: <http://www.itu.int/ITU-T/asn1/uuid.html> Дата доступу:
квітень 2018.
21. ООР [Електронний ресурс]. Режим доступу: <https://bit.ly/20Rx76M>
Дата доступу: квітень 2018.

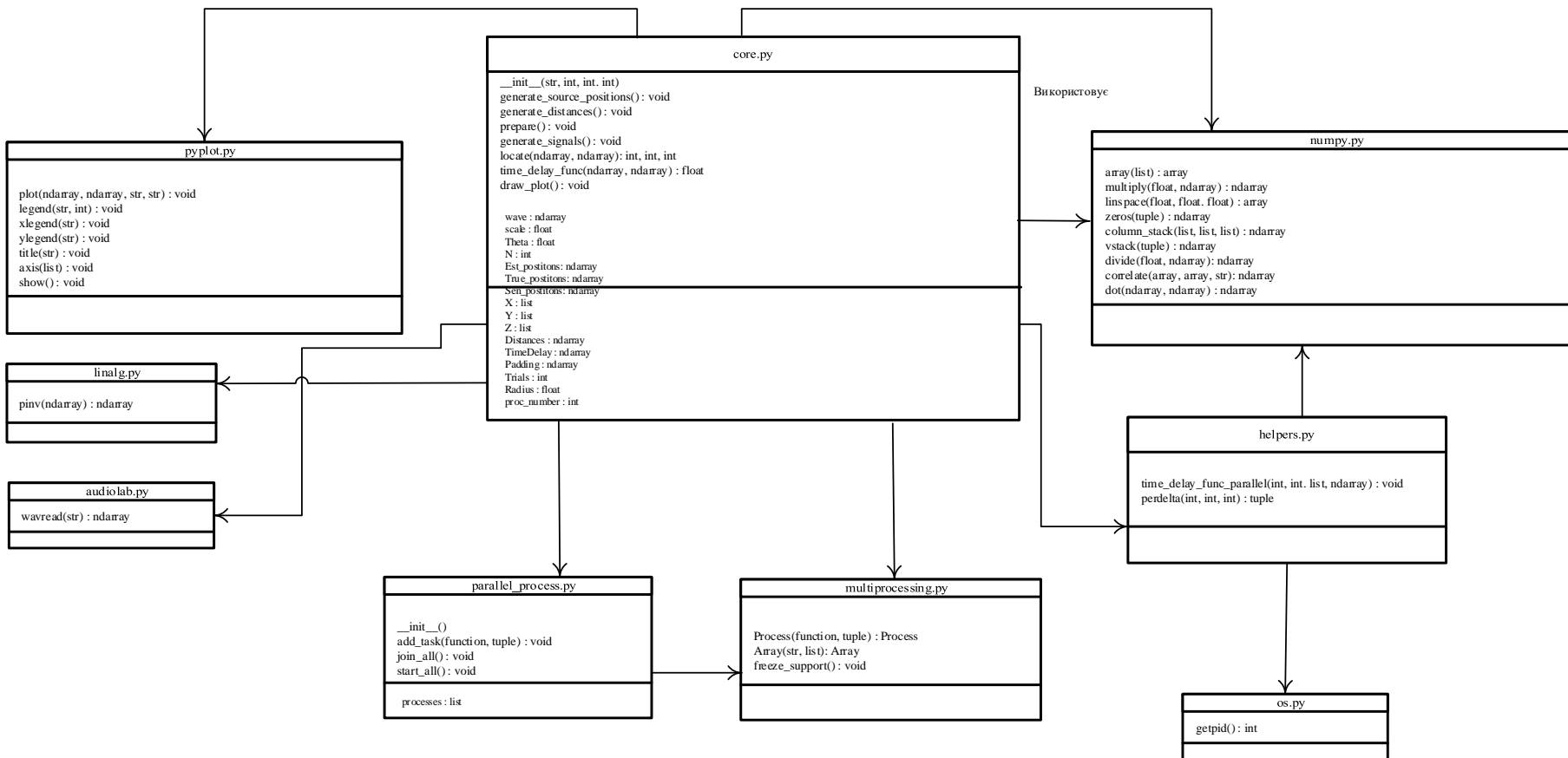
Додаток 1

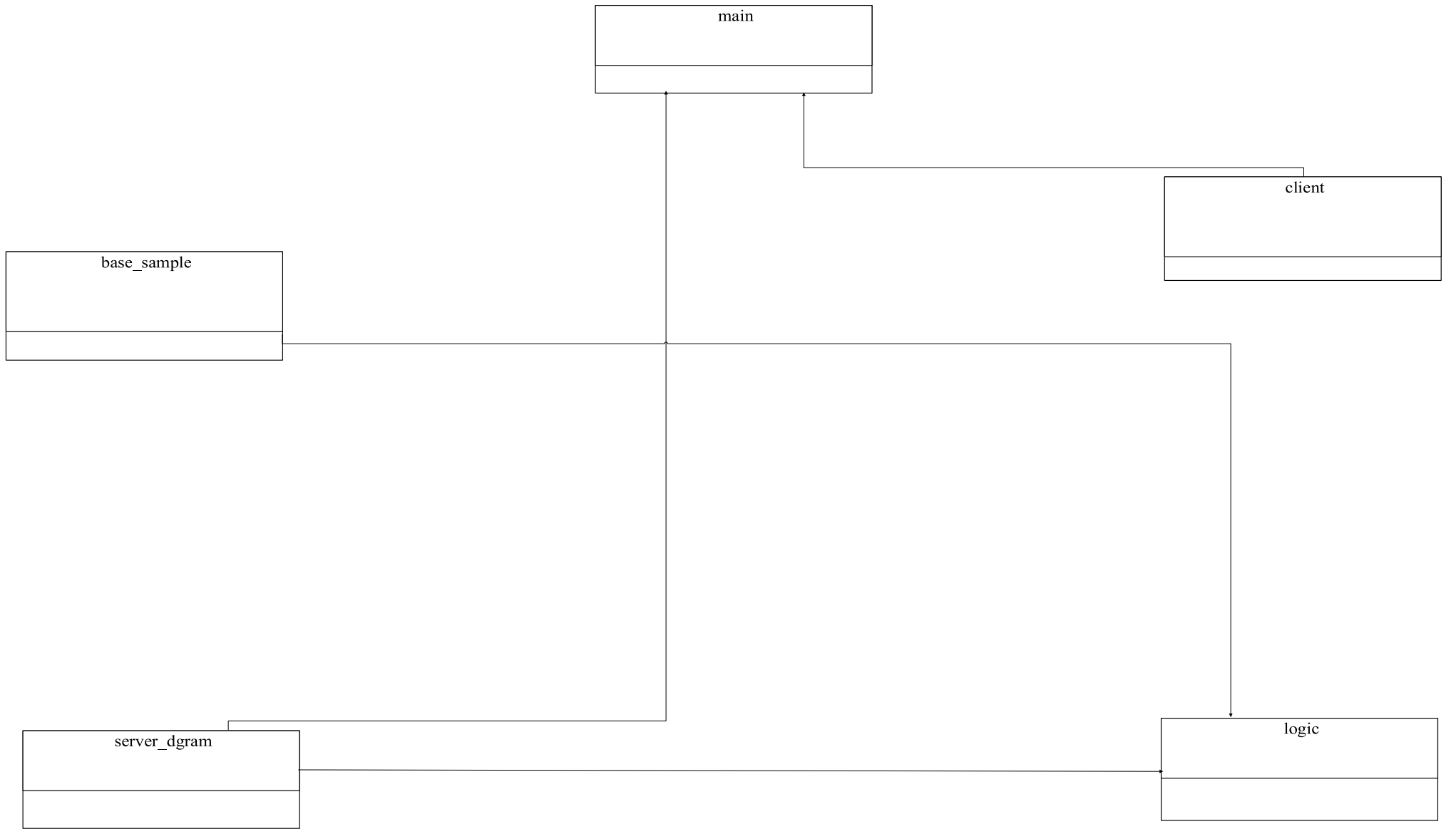
Копії графічних матеріалів

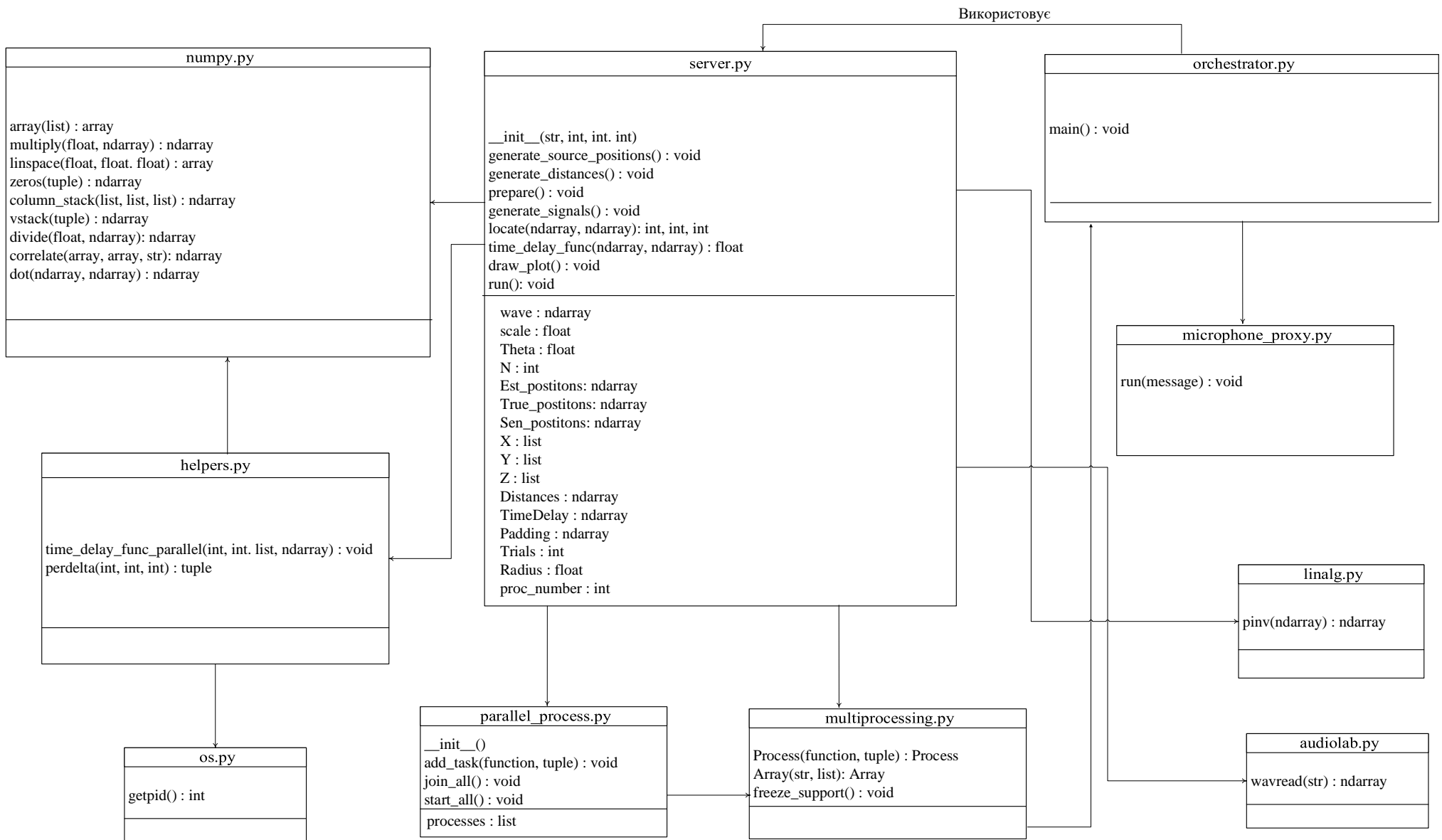












Додаток 2

Лістинг програми

base_sample/core.py

```
from scikits.audiolab import wavread
import numpy
import math
from matplotlib import pyplot
from scipy import linalg
import time
from src.logic.parallel_process import ProcessParallel
from multiprocessing import Array
from src.logic import helpers
import logging

class Core:
    def __init__(self, audio, mic_amount, trials, proc_number):
        logging.info('Starting core init.')

        self.proc_numer = proc_number

        # the magic of preparing audio data; from numpy arrays to flatten
        list with removed duplicated elements
        self.wave = wavread(audio)[0] # removing wav technical data; only
        audio data stays
        self.wave = [list(pair) for pair in self.wave]
        audio_data = numpy.array(self.wave)
        self.wave = list(audio_data.flatten())
        self.wave = self.wave[:,2]
        self.wave = numpy.array(self.wave).reshape(-1, 1)

        self.scale = 0.8 / max(self.wave)
        self.wave = numpy.multiply(self.scale, self.wave)

        self.trials = trials
        self.__radius = 50
        self.__microphone_amount = mic_amount
        self.Theta = numpy.linspace(0, 2 * math.pi,
self.__microphone_amount + 1)

        self.X = [self.__radius * math.cos(x) for x in self.Theta[0: -1]]
        self.Y = [self.__radius * math.sin(x) for x in self.Theta[0: -1]]

        self.Z = [-1 if z % 2 == 0 else 1 for z in
range(self.__microphone_amount)]
        self.Z = [5 * z + 5 for z in self.Z]

        self.sensor_positions = numpy.column_stack((self.X, self.Y,
self.Z))
        self.true_positions = numpy.zeros((self.trials, 3))
        self.estimated_positions = numpy.zeros((self.trials, 3))

        self.distances = []
        self.time_delays = []
        self.padding = []

        logging.info('Inited core.')

    def generate_source_positions(self):
```

```

logging.info('Generating sources positions.')

for i in range(self.trials):
    r = numpy.random.rand(1) * 50
    t = numpy.random.rand(1) * 2 * math.pi
    #r = 0.1 * 50
    #t = 0.2 * 50
    #z = 0.3 * 20
    x = r * math.cos(t)
    y = r * math.sin(t)
    z = numpy.random.rand(1) * 20
    self.true_positions[i, 0] = x
    self.true_positions[i, 1] = y
    self.true_positions[i, 2] = z

logging.info('Generated sources positions.')

def generate_distances(self):
    logging.info('Generating distances.')

    self.distances = numpy.zeros((self.trials,
self.__microphone_amount))
    for i in range(self.trials):
        for j in range(self.__microphone_amount):
            x1 = self.true_positions[i, 0]
            y1 = self.true_positions[i, 1]
            z1 = self.true_positions[i, 2]
            x2 = self.sensor_positions[j, 0]
            y2 = self.sensor_positions[j, 1]
            z2 = self.sensor_positions[j, 2]
            self.distances[i, j] = math.sqrt((x1 - x2) ** 2 + (y1 - y2)
** 2 + (z1 - z2) ** 2)

logging.info('Generated distances.')

def prepare(self):
    logging.info('Preparing stage started.')

    self.time_delays = numpy.divide(self.distances, 340.29)
    self.padding = numpy.multiply(self.time_delays, 44100)

logging.info('Preparing stage ended.')

def generate_signals(self):
    for i in range(self.trials):
        x = self.true_positions[i, 0]
        y = self.true_positions[i, 1]
        z = self.true_positions[i, 2]

        mic_data =
[ numpy.vstack((numpy.zeros((int(round(self.padding[i, j])), 1)),
self.wave)) for j in
            range(self.__microphone_amount)]
        lenvec = numpy.array([len(mic) for mic in mic_data])
        m = max(lenvec)
        c = numpy.array([m - mic_len for mic_len in lenvec])
        mic_data = [numpy.vstack((current_mic, numpy.zeros((c[idx],
1)))) for idx, current_mic in
            enumerate(mic_data)]
        mic_data = [numpy.divide(current_mic, self.distances[i, idx])
for idx, current_mic in enumerate(mic_data)]
        multitrack = numpy.array(mic_data)

logging.info('Prepared all data.')

```

```

        logging.info('Started source localization.')

        x, y, z = self.locate(self.sensor_positions, multitrack)

        logging.info('Localized source.')

        self.estimated_positions[i, 0] = x
        self.estimated_positions[i, 1] = y
        self.estimated_positions[i, 2] = z

def locate(self, sensor_positions, multitrack):
    s = sensor_positions.shape
    len = s[0]

    time_delays = numpy.zeros((len, 1))

    starts = time.time()

    if self.proc_numer == 1:
        for p in range(len):
            time_delays[p] =
helpers.time_delay_function(multitrack[0,], multitrack[p,])
    else:
        pp = ProcessParallel()

        outs = Array('d', range(len))

        ranges = []

        for result in helpers.per_delta(0, len, len / self.proc_numer):
            ranges.append(result)

        for start, end in ranges:
            pp.add_task(helpers.time_delay_function_optimized, (start,
end, outs, multitrack))

        pp.start_all()
        pp.join_all()

        for idx, res in enumerate(outs):
            time_delays[idx] = res

    ends = time.time()

    logging.info('%.15f passed for trial.', ends - starts)

    Amat = numpy.zeros((len, 1))
    Bmat = numpy.zeros((len, 1))
    Cmat = numpy.zeros((len, 1))
    Dmat = numpy.zeros((len, 1))

    for i in range(2, len):
        x1 = sensor_positions[0, 0]
        y1 = sensor_positions[0, 1]
        z1 = sensor_positions[0, 2]
        x2 = sensor_positions[1, 0]
        y2 = sensor_positions[1, 1]
        z2 = sensor_positions[1, 2]
        xi = sensor_positions[i, 0]
        yi = sensor_positions[i, 1]
        zi = sensor_positions[i, 2]
        Amat[i] = (1 / (340.29 * time_delays[i])) * (-2 * x1 + 2 * xi)
- (1 / (340.29 * time_delays[1])) * (
            -2 * x1 + 2 * x2)

```

```

        Bmat[i] = (1 / (340.29 * time_delays[i])) * (-2 * y1 + 2 * yi)
- (1 / (340.29 * time_delays[1])) * (
            -2 * y1 + 2 * y2)
        Cmat[i] = (1 / (340.29 * time_delays[i])) * (-2 * z1 + 2 * zi)
- (1 / (340.29 * time_delays[1])) * (
            -2 * z1 + 2 * z2)
        Sum1 = (x1 ** 2) + (y1 ** 2) + (z1 ** 2) - (xi ** 2) - (yi **
2) - (zi ** 2)
        Sum2 = (x1 ** 2) + (y1 ** 2) + (z1 ** 2) - (x2 ** 2) - (y2 **
2) - (z2 ** 2)
        Dmat[i] = 340.29 * (time_delays[i] - time_delays[1]) + (1 /
(340.29 * time_delays[i])) * Sum1 - (1 / (
            340.29 * time_delays[1])) * Sum2

    M = numpy.zeros((len + 1, 3))
    D = numpy.zeros((len + 1, 1))
    for i in range(len):
        M[i, 0] = Amat[i]
        M[i, 1] = Bmat[i]
        M[i, 2] = Cmat[i]
        D[i] = Dmat[i]

    M = numpy.array(M[2:len, :])
    D = numpy.array(D[2:len])

    D = numpy.multiply(-1, D)

    Minv = linalg.pinv(M)

    T = numpy.dot(Minv, D)
    x = T[0]
    y = T[1]
    z = T[2]

    return x, y, z

    def draw_plot(self):
        pyplot.plot(self.true_positions[:, 0], self.true_positions[:, 1],
'bd', label='True position')
        pyplot.plot(self.estimated_positions[:, 0],
self.estimated_positions[:, 1], 'r+', label='Estimated position')
        pyplot.legend(loc='upper right', numpoints=1)
        pyplot.xlabel('X coordinate of target')
        pyplot.ylabel('Y coordinate of target')
        pyplot.title('TDOA Hyperbolic Localization')
        pyplot.axis([-50, 50, -50, 50])
        pyplot.show()

```

base_sample/console_runner.py

```

from src.base_sample.core import Core
from multiprocessing import freeze_support, cpu_count
import argparse
import logging

if __name__ == '__main__':
    freeze_support()

    parser = argparse.ArgumentParser()

    parser.add_argument("-m", "--mic_amount", type=int,
                        help="microphone amount")
    parser.add_argument("-p", "--proc_number", type=int,

```

```

        help="process number")
parser.add_argument("-t", "--trials", type=int,
                    help="trials number")
parser.add_argument("-f", "--file", type=str,
                    help="file name")
parser.add_argument("-l", "--log_file", type=str,
                    help="log file")

args = parser.parse_args()

if args.log_file:
    logging.basicConfig(format='%(levelname)s, PID: %(process)d,
%(asctime)s:\t%(message)s', level=logging.INFO)
else:
    logging.basicConfig(format='%(levelname)s, PID: %(process)d,
%(asctime)s:\t%(message)s', filename=args.log_file,
                        level=logging.INFO)

if args.proc_number:
    if args.proc_number <= 0:
        raise ValueError('proc_number can't be less than zero.')
    cores_to_use = args.proc_number
else:
    cores_to_use = cpu_count()

core = Core(args.file,
            mic_amount=args.mic_amount,
            trials=args.trials,
            proc_number=cores_to_use)

core.generate_source_positions()
core.generate_distances()
core.prepare()
core.generate_signals()

for trial_number in range(core.trials):
    logging.info('Trial number: %d', trial_number + 1)

    logging.info('Estimated X = %.15f, Estimated Y = %.15f, Estimated Z
= %.15f', float(core.estimated_positions[trial_number][0]),
            float(core.estimated_positions[trial_number][1]),
            float(core.estimated_positions[trial_number][2]))

    logging.info('True X = %.15f, True Y = %.15f, True Z = %.15f',
float(core.true_positions[trial_number][0]),
            float(core.true_positions[trial_number][1]),
            float(core.true_positions[trial_number][2]))

    core.draw_plot()

```

client/microphone_proxy.py

```

import logging
import socket
import time
from cPickle import dumps

class MicrophoneProxy:
    def __init__(self, server_address, server_port, id):
        self.__server_address = server_address
        self.__server_port = server_port
        self.id = id

```

```

self.__message_check_len = 65535 - 28 - 36
self.__message_len = 65000
self.__send_delay = 0.1

def run(self, message):
    # Create a UDP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    server_address = (self.__server_address, self.__server_port)

    try:
        # Send data
        logging.info("Sending data from microphone with id %s.",
self.id)
        serialized = dumps(message)
        data_len = len(serialized)

        logging.info("Data length is %s id %s", data_len, str(self.id))
        if data_len > self.__message_check_len:

            data = [serialized[i:i+self.__message_len] for i in
range(0, data_len, self.__message_len)]

            for i in range(len(data)):
                logging.info("Sending %s chunk from mic with id %s and
len %s", i, self.id, len(data[i]) + 36)

                sock.sendto(str(self.id) + data[i], server_address)
                time.sleep(self.__send_delay )
            else:
                sock.sendto(str(self.id) + serialized, server_address)

            time.sleep(self.__send_delay )
            sock.sendto(str(self.id), server_address)
            logging.info("Sending info chunk from mic with id %s and len
%s", self.id, len(str(self.id)))
            logging.info("Data sent.")
        finally:
            logging.info("Closing microphone's socket with id %s.",
self.id)

            sock.close()

            logging.info("Closed microphone's socket with id %s.", self.id)

```

logic/helpers.py

```

import numpy
import logging

def time_delay_function_optimized(start, end, outs, multi):
    for idx in range(start, end):
        logging.info('Locating...')

        c = numpy.correlate(multi[0,][:, 0], multi[idx,][:, 0], "full")
        C, I = c.max(0), c.argmax(0)
        outs[idx] = ((float(len(c)) + 1.0) / 2.0 - I) / 44100.0

def per_delta(start, end, delta):
    curr = start
    while curr < end and curr + delta < end:

```



```

        yield (curr, curr + delta)
        curr += delta
    yield (curr, end)

```

```

def time_delay_function(x, y):
    c = numpy.correlate(x[:, 0], y[:, 0], "full")
    C, I = c.max(0), c.argmax(0)
    out = ((float(len(c)) + 1.0) / 2.0 - I) / 44100.0
    return out

```

logic/orchestrator.py

```

import numpy
import math
import uuid
from scikits.audiolab import wavread
from src.client.microphone_proxy import MicrophoneProxy
from src.server_dgram.server import Server

class Orchestrator:
    def __init__(self, config):
        # File name
        self.__audio = config["audio"]

        # Audio data
        self.__wave = []

        self.__proxies = []

        # Server data
        self.__trials = int(config["trials"])
        self.__cores_amount = int(config["cores_amount"])
        self.server = None
        self.__server_address = config["server_address"]
        self.__server_port = int(config["server_port"])
        self.__microphone_amount = int(config["microphone_amount"])
        self.__radius = int(config["radius"])
        self.__true_positions = None
        self.__estimated_positions = None
        self.__sensor_positions = None

    def retrieve_file_data(self):
        # removing header and second channel data
        wave = wavread(self.__audio)[0]
        wave = [list(pair) for pair in wave]
        audio_data = numpy.array(wave)
        wave = list(audio_data.flatten())
        wave = wave[::2]
        wave = numpy.array(wave).reshape(-1, 1)

        scale = 0.8 / max(wave)
        self.__wave = numpy.multiply(scale, wave)

    def init_server(self):
        theta = numpy.linspace(0, 2 * math.pi, self.__microphone_amount +
1)
        X = [self.__radius * math.cos(x) for x in theta[0: -1]]
        Y = [self.__radius * math.sin(x) for x in theta[0: -1]]
        Z = [-1 if z % 2 == 0 else 1 for z in
range(self.__microphone_amount)]

```

```

Z = [5 * z + 5 for z in Z]

self.__sensor_positions = numpy.column_stack((X, Y, Z))
self.__true_positions = numpy.zeros((self.__trials, 3))
self.__estimated_positions = numpy.zeros((self.__trials, 3))

self.server = Server(self.__server_address,
                    self.__server_port,
                    self.__true_positions,
                    self.__estimated_positions,
                    self.__sensor_positions,
                    self.__microphone_amount,
                    self.__trials,
                    (X, Y, Z),
                    self.__cores_amount)

def send_data_to_server(self):
    for i in range(self.__trials):
        microphone_data =
[ numpy.vstack((numpy.zeros((int(round(self.server.padding[i, j])), 1)),
self.__wave)) for
                                j in
                                range(self.__microphone_amount)]
        lenvec = numpy.array([len(mic) for mic in microphone_data])
        m = max(lenvec)
        c = numpy.array([m - mic_len for mic_len in lenvec])
        microphone_data = [numpy.vstack((current_mic,
numpy.zeros((c[idx], 1)))) for idx, current_mic in
                                enumerate(microphone_data)]
        microphone_data = [numpy.divide(current_mic,
self.server.distances[i, idx]) for idx, current_mic in
                                enumerate(microphone_data)]

        for j in range(self.__microphone_amount):
            self.__send_data_via_proxy(microphone_data[j])

def __send_data_via_proxy(self, message):
    proxy = MicrophoneProxy(self.__server_address, self.__server_port,
uuid.uuid4())
    self.__proxies.append(proxy)
    proxy.run(message)

def locate(self, s):
    self.server.handle_retrieved_data(s)

def get_results(self):
    self.server.log_results()
    self.server.draw_plot()

```

logic/parallel_process.py

```

from multiprocessing import Process

class ProcessParallel(object):
    """
    To Process the jobs in separate interpreters
    """
    def __init__(self):
        self.processes = []

    def add_task(self, job, arg):
        self.processes.append(Process(target=job, args=arg))

```

```

def start_all(self):
    """
    Starts the functions process all together.
    """
    [process.start() for process in self.processes]

def join_all(self):
    """
    Waits until all the processes executed.
    """
    [process.join() for process in self.processes]

```

server_dgram/server.py

```

import logging
import socket
import numpy
import time
from cPickle import loads
from scipy import linalg
from matplotlib import pyplot
from multiprocessing import Array
from src.logic import helpers
from src.logic.parallel_process import ProcessParallel
from scipy import *
from numpy import *

class Server:
    def __init__(self,
                 server_address,
                 server_port,
                 true_positions,
                 estimated_positions,
                 sensor_positions,
                 microphone_amount,
                 trials,
                 coordinates,
                 cores_amount):
        self.__x, self.__y, self.__z = coordinates
        self.__server_address = server_address
        self.__microphone_amount = microphone_amount
        self.__server_port = server_port
        self.__true_positions = true_positions
        self.__estimated_positions = estimated_positions
        self.__trials = trials
        self.__sensor_positions = sensor_positions
        self.__distances = []
        self.__time_delays = []
        self.__padding = []
        self.__cores_amount = cores_amount
        self.__microphone_data = None
        self.__raw_microphone_data = []

    def generate_data(self):
        self.generate_source_positions()
        self.generate_distances()
        self.prepare()

    def run(self, received_data):
        # Create a TCP/IP socket

```

```

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to the port
server_address = (self.__server_address, self.__server_port)
logging.info('Starting up on %s port %s', self.__server_address,
self.__server_port)

sock.bind(server_address)

microphones_data = {}

received_data_count = 0
while received_data_count < self.__microphone_amount:
    logging.info('Waiting to receive message...')

    data, address = sock.recvfrom(65535 - 28)
    logging.info("Received %s", len(data))
    if len(data) == 36:
        received_data[received_data_count] = microphones_data[data]
        received_data_count += 1
        logging.info("Received data from %s microphones",
received_data_count)
    else:
        microphone_id = data[0:36]

        if not microphone_id in microphones_data:
            microphones_data[microphone_id] = data[36:]
        else:
            microphones_data[microphone_id] += data[36:]

    logging.info("Received data from all microphones")

def generate_source_positions(self):
    logging.info('Generating sources positions.')

    for i in range(self.__trials):
        #r = numpy.random.rand(1) * 50
        #t = numpy.random.rand(1) * 2 * math.pi
        r = 0.1 * 50
        t = 0.2 * 50
        z = 0.3 * 20
        x = r * math.cos(t)
        y = r * math.sin(t)
        #z = numpy.random.rand(1) * 20
        self.__true_positions[i, 0] = x
        self.__true_positions[i, 1] = y
        self.__true_positions[i, 2] = z

    logging.info('Generated sources positions.')

def generate_distances(self):
    logging.info('Generating distances.')

    self.__distances = numpy.zeros((self.__trials,
self.__microphone_amount))
    for i in range(self.__trials):
        for j in range(self.__microphone_amount):
            x1 = self.__true_positions[i, 0]
            y1 = self.__true_positions[i, 1]
            z1 = self.__true_positions[i, 2]
            x2 = self.__sensor_positions[j, 0]
            y2 = self.__sensor_positions[j, 1]
            z2 = self.__sensor_positions[j, 2]

```

```

        self.__distances[i, j] = math.sqrt((x1 - x2) ** 2 + (y1 -
y2) ** 2 + (z1 - z2) ** 2)

        logging.info('Generated distances.')

    def log_results(self):
        for trial_number in range(self.__trials):
            logging.info('Trial number: %d', trial_number + 1)

            logging.info('Estimated X = %.15f, Estimated Y = %.15f,
Estimated Z = %.15f',
float(self.__estimated_positions[trial_number][0]),
float(self.__estimated_positions[trial_number][1]),
float(self.__estimated_positions[trial_number][2]))

            logging.info('True X = %.15f, True Y = %.15f, True Z = %.15f',
float(self.__true_positions[trial_number][0]),
float(self.__true_positions[trial_number][1]),
float(self.__true_positions[trial_number][2]))

    def draw_plot(self):
        pyplot.plot(self.__true_positions[:, 0], self.__true_positions[:,
1], 'bd', label='True position')
        pyplot.plot(self.__estimated_positions[:, 0],
self.__estimated_positions[:, 1], 'r+',
label='Estimated position')
        pyplot.legend(loc='upper right', numpoints=1)
        pyplot.xlabel('X coordinate of target')
        pyplot.ylabel('Y coordinate of target')
        pyplot.title('TDOA Hyperbolic Localization')
        pyplot.axis([-50, 50, -50, 50])
        pyplot.show()

    def prepare(self):
        logging.info('Preparing stage started.')

        self.__time_delays = numpy.divide(self.__distances, 340.29)
        self.__padding = numpy.multiply(self.__time_delays, 44100)

        logging.info('Preparing stage ended.')

    def handle_retrieved_data(self, received_data):
        for i in range(self.__trials):
            x = self.__true_positions[i, 0]
            y = self.__true_positions[i, 1]
            z = self.__true_positions[i, 2]

            data = []
            for j in range(self.__microphone_amount):
                data.append(received_data[j])

            multi_track = numpy.array([loads(raw) for raw in data])
            logging.info('Prepared all data.')
            logging.info('Started source localization.')

            x, y, z = self.locate(self.__sensor_positions, multi_track)

            logging.info('Localized source.')

            self.__estimated_positions[i, 0] = x
            self.__estimated_positions[i, 1] = y

```

```

        self.__estimated_positions[i, 2] = z

def locate(self, sensor_positions, multi_track):
    s = sensor_positions.shape
    len = s[0]

    time_delays = numpy.zeros((len, 1))

    starts = time.time()

    if self.__cores_amount == 1:
        for p in range(len):
            time_delays[p] =
helpers.time_delay_function(multi_track[0,], multi_track[p,])
    else:
        pp = ProcessParallel()

        outs = Array('d', range(len))

        ranges = []

        for result in helpers.per_delta(0, len, len /
self.__cores_amount):
            ranges.append(result)

        for start, end in ranges:
            pp.add_task(helpers.time_delay_function_optimized, (start,
end, outs, multi_track))

        pp.start_all()
        pp.join_all()

        for idx, res in enumerate(outs):
            time_delays[idx] = res

    ends = time.time()

    logging.info('%0.15f passed for localization computation trial.',
ends - starts)

    Amat = numpy.zeros((len, 1))
    Bmat = numpy.zeros((len, 1))
    Cmat = numpy.zeros((len, 1))
    Dmat = numpy.zeros((len, 1))

    for i in range(2, len):
        x1 = sensor_positions[0, 0]
        y1 = sensor_positions[0, 1]
        z1 = sensor_positions[0, 2]
        x2 = sensor_positions[1, 0]
        y2 = sensor_positions[1, 1]
        z2 = sensor_positions[1, 2]
        xi = sensor_positions[i, 0]
        yi = sensor_positions[i, 1]
        zi = sensor_positions[i, 2]
        if time_delays[i] == 0 and time_delays[1] == 0:
            Amat[i] = 0
            Bmat[i] = 0
            Cmat[i] = 0
            Dmat[i] = 0
            continue

        if time_delays[i] == 0:
            ti_value = 0

```

```

else:
    ti_value = 1 / (340.29 * time_delays[i])

if time_delays[1] == 0:
    t1_value = 0
else:
    t1_value = 1 / (340.29 * time_delays[1])

Amat[i] = ti_value * (-2 * x1 + 2 * xi) - t1_value * (
    -2 * x1 + 2 * x2)
Bmat[i] = ti_value * (-2 * y1 + 2 * yi) - t1_value * (
    -2 * y1 + 2 * y2)
Cmat[i] = ti_value * (-2 * z1 + 2 * zi) - t1_value * (
    -2 * z1 + 2 * z2)
Sum1 = (x1 ** 2) + (y1 ** 2) + (z1 ** 2) - (xi ** 2) - (yi **
2) - (zi ** 2)
Sum2 = (x1 ** 2) + (y1 ** 2) + (z1 ** 2) - (x2 ** 2) - (y2 **
2) - (z2 ** 2)
Dmat[i] = 340.29 * (time_delays[i] - time_delays[1]) + ti_value
* Sum1 - t1_value * Sum2

M = numpy.zeros((len + 1, 3))
D = numpy.zeros((len + 1, 1))
for i in range(len):
    M[i, 0] = Amat[i]
    M[i, 1] = Bmat[i]
    M[i, 2] = Cmat[i]
    D[i] = Dmat[i]

M = numpy.array(M[2:len, :])
D = numpy.array(D[2:len])

D = numpy.multiply(-1, D)

Minv = linalg.pinv(M)

T = numpy.dot(Minv, D)
x = T[0]
y = T[1]
z = T[2]

return x, y, z

@property
def padding(self):
    return self.__padding

@property
def distances(self):
    return self.__distances

```

src/server_runner.py

```

import json
import logging
from multiprocessing import freeze_support
import multiprocessing
import time
from src.logic.orchestrator import Orchestrator
from src.logic.parallel_process import ProcessParallel

def main():

```

```

freeze_support()

logging.basicConfig(format='%(levelname)s, PID: %(process)d,
%(asctime)s:\t%(message)s', level=logging.INFO)

config = json.load(open('server_config.json'))

orchestrator = Orchestrator(config)
orchestrator.retrieve_file_data()
orchestrator.init_server()
orchestrator.server.generate_data()

manager = multiprocessing.Manager()
received_data = manager.dict()

pp = ProcessParallel()

pp.add_task(orchestrator.server.run, (received_data,))
pp.add_task(orchestrator.send_data_to_server, ())

pp.start_all()
starts = time.time()

pp.join_all()

orchestrator.locate(received_data)

ends = time.time()

logging.info('%.15f passed for SEND/RECEIVE/CALCULATE.', ends - starts)

orchestrator.get_results()

if __name__ == '__main__':
    main()

```