МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп'ютерних наук та кібербезпеки

На правах рукопису

МАРКІТАН ВОЛОДМИР ОЛЕГОВИЧ

ПОТОКОВА ЦИФРОВА ОБРОБКА АУДІОСИГНАЛІВ ДЛЯ КОНФІГУРУВАННЯ МУЗИЧНИХ ІНСТРУМЕНТІВ

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
ГОЛОВІН МИКОЛА БОРИСОВИЧ,
кандитат фізико-математичних наук, доцент кафедри комп'ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №________
засідання кафедри_______________
від________________________2024 р.
Завідувач кафедри

(___________) ___________________
(підпис) ПІБ

ЛУЦЬК – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ ТА ТЕРМІНІВ 4
ВСТУП 5
РОЗДІЛ 1: ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО СПЕКТРАЛЬНИЙ АНАЛІЗ ТА ТЕХНОЛОГІЇ ДЛЯ РЕАЛІЗАЦІЇ ТЮНЕРА 7
1.1 Спектральний аналіз: теорія та застосування 7
1.1.1 Загальні відомості 7
1.1.2 Історія розвитку спектрального аналізу 8
1.1.3 Методи спектрального аналізу 9
1.1.4 Метод швидкого перетворення Фур'є (FFT) 10
1.1.5 Інші методи аналізу сигналів 12
1.2 Огляд переваг обраних інструментів та технологій................................ 13
1.2.1 Мова програмування Python 13
1.2.2 Огляд бібліотек мови програмування Python.................................... 16
1.3 Огляд та аналіз алогічних програмних застосунків................................ 19
РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ НАЛАШТУВАННЯ ІСТРУМЕНТІВ ЗА ДОПОМОГОЮ ПЕРЕТВОРЕННЯ ФУР'Є(МОВОЮ PYTHON) 24
2.1 Постановка задачі та позначення вимог для застосунку налаштування музичних інструментів «Chromatuna» 24
2.2 Аналіз та вибір моделі розробки програмного засобу............................... 25
2.3 Загальний опис проєкту 28
2.4 Обґрунтування вибору інструментальних засобів розробки 30
2.4.1 Вибір PyCharm для розробки програми налаштування музичних інструментів 30
2.4.2 Застосування бібліотеки CustomTkinter для розробки і створення інтерфейсу програмного засобу 32
2.5 Особливості програмної реалізації 34
2.5.1 Реалізація побудови інтерфейсу користувача та побудова функціоналу вибору інструменту та вибору варіацій налаштування................................ 34
2.5.2 Реалізація алгоритму визначення частот. 37
2.6 Організація тестування та налагодження програмного засобу.................. 41
ВИСНОВКИ.. 43

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 44
ДОДАТКИ 47
Додаток А 47
Додаток Б 50
Додаток В 51

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ ТА ТЕРМІНІВ

HPS	– Harmonic Product Spectrum (Гармонічний добуток спектрів): алгоритм для визначення висоти звуку, запропонований А. Майкл Ноллом у 1969 р.
DFT	– Дискретне перетворення Фур'є
FFT	– Швидке перетворення Фур'є
МП	– Мова Програмування
ПЗ	– Програмне Забезпечення
ТЗ	– Технічне Завдання
Гц	– Герц: одиниця вимірювання частоти періодичних коливань
PY	– Python: Інтерпретована МП
Tk	– Tkinter: Бібліотека для МП PY для побудови інтерфейсів
CTk	– CustomTkinter: PY бібліотека, що побудована на основі Tk, є його покращеною надбудовою
Git	– Система Управління Версіями, Віддалений Репозиторій

ВСТУП

Перетворення Фур'є відіграє важливу роль у спектральному аналізі, який знаходить широке застосування в науці та техніці. З його допомогою аналізуються сигнали у різних прикладних задачах, таких як обробка звуку, радіоастрономія, геофізика, медицина, а також в електронних та телекомунікаційних системах. Зокрема, у галузі акустики та звукових технологій перетворення Фур'є є основою для створення високоточних тюнерів та програм, які дозволяють точно налаштовувати музичні інструменти.

Актуальність теми. є дослідження спектрального аналізу акустичних сигналів, як в напрямку систем захисту та ідентифікації звуків, так і напрямку в розробки звукових тюнерів й інших застосунків для музикантів та звукоінженерів.

Крім того, реалізація тюнера на основі перетворення Фур’є може знайти застосування в різних сферах, таких як звукозапис, концертна діяльність та музична освіта.

Мета даної роботи – реалізація застосунку для налаштування струнних музичних інструментів за допомогою спектрального аналізу звуків перетворенням Фур’є мовою програмування Python.

Завдання. Для реалізації поставленої мети нам потрібно виконати наступне:

- Вивчити сучасні алгоритми спектрального аналізу, зокрема швидке перетворення Фур'є (FFT).
- Розробити алгоритм для реалізації тюнера, заснованого на методах спектрального аналізу.
- Реалізувати HPS(Harmomic Product Spectrum) алгоритм для точного визначення частотних характеристик звукового сигналу
- Ознайомитися з основними алгоритмами та бібліотеками мови PY, що використовуються для спектрального аналізу.

- У середовищі розробки IDE PyCharm реалізувати програму для тюнера, що використовує мову програмування PY.
- Оцінити ефективність та точність роботи розробленого тюнера та сформулювати висновки.

Об'єкт дослідження – методи спектрального аналізу та їх застосування для налаштування музичних інструментів.

Апробація результатів. Результати роботи були висвітлені в тезах I Міжнародної науково-практичної конференції «Проблеми комп'ютерних наук, програмного моделювання та безпеки цифрових систем» - Луцьк-Світязь. - 13-16 червня 2024 р.[1]

РОЗДІЛ 1: ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО СПЕКТРАЛЬНИЙ АНАЛІЗ ТА ТЕХНОЛОГІЇ ДЛЯ РЕАЛІЗАЦІЇ ТЮНЕРА

1.1 Спектральний аналіз: теорія та застосування

1.1.1 Загальні відомості

Спектральний аналіз — це потужний метод дослідження сигналів, який дозволяє представити складний сигнал у вигляді суми гармонічних компонент з різними частотами та амплітудами. Цей підхід надає глибоке розуміння структури та властивостей сигналу, що особливо важливо у звуковій сфері для аналізу звуків, музики та інших акустичних сигналів. Спектральний аналіз дає змогу визначити частотні складові сигналу, їхні амплітуди та фази — ключові параметри для аналізу та обробки звуків.

Спектральний аналіз ґрунтується на фундаментальній теоремі Фур'є, яка стверджує, що будь-який періодичний сигнал можна представити як суму гармонічних коливань з частотами, кратними основній частоті сигналу. Для неперіодичних сигналів використовується узагальнення — перетворення Фур'є, яке дозволяє розкласти сигнал на гармонічні компоненти з неперервним спектром частот. Таким чином, спектральний аналіз дозволяє дослідити частотний склад сигналу, виявити домінуючі частоти та їхні амплітуди.

Спектральний аналіз має широке застосування у різних галузях науки та техніки. У звуковій інженерії він використовується для аналізу та обробки музичних сигналів, розпізнавання мовлення, шумозаглушення, синтезу звуків тощо. В електроніці спектральний аналіз застосовується для аналізу спектрів радіосигналів, виявлення гармонік та інтермодуляційних спотворень. У механіці спектральний аналіз вібрацій дозволяє діагностувати стан обертових машин та механізмів. У медицині спектральний аналіз в ЕКГ використовується для діагностики серцево-судинних та неврологічних захворювань. Таким чином, спектральний аналіз є фундаментальним інструментом дослідження сигналів у багатьох галузях науки та техніки.

1.1.2 Історія розвитку спектрального аналізу

Витоки спектрального аналізу сягають ще XVIII століття, коли Жан-Батист Жозеф Фур'є сформулював свою знамениту теорему про розкладання періодичних функцій у тригонометричні ряди. У 1822 році Фур'є опублікував свою роботу «Аналітична теорія тепла», в якій довів, що будь-яку періодичну функцію можна представити як суму гармонічних коливань з частотами, кратними основній частоті. Ця фундаментальна теорема стала основою спектрального аналізу. [2]

Пізніше, у XIX столітті, Густав Кірхгоф та Роберт Бунзен застосували ідеї Фур'є для аналізу спектрів випромінювання хімічних елементів, що стало основою спектрального аналізу в хімії. У 1860 році Кірхгоф сформулював закони теплового випромінювання, які пов'язували спектр випромінювання з температурою та хімічним складом тіла. Бунзен та Кірхгоф винайшли спектроскоп — прилад для аналізу спектрів, що дозволило їм відкрити нові хімічні елементи цезій та рубідій. [3]

У 1940-х роках спектральний аналіз почав широко використовуватися у звуковій сфері для аналізу та обробки звуків. Розвиток електронної техніки дозволив реалізувати спектральний аналіз у реальному часі за допомогою аналогових фільтрів та осцилографів. Спектральний аналіз застосовувався для аналізу музичних інструментів, розпізнавання мовлення, шумозаглушення тощо.

З розвитком цифрової обробки сигналів у 1960-х роках, найпоширеніший алгоритм швидкого перетворення Фур'є (FFT), також званий як Алгоритм Кулі-Тюкі, запропонований Джеймсом Кулі та Джоном Тьюкі у 1965 році, значно підвищив ефективність обчислення спектрів дискретних сигналів. Це дозволило реалізувати спектральний аналіз у реальному часі на комп'ютерах та застосовувати його у все ширшому колі задач. [4]

Сучасні методи спектрального аналізу, такі як вейвлет-аналіз, кепстральний аналіз, аналіз Вігнера-Віля, дозволяють досліджувати сигнали з нестаціонарним спектром, виявляти тонкі ефекти та аналізувати сигнали у часово-частотній області. Розвиток комп'ютерної техніки та програмного

забезпечення зробив спектральний аналіз доступним та зручним інструментом для дослідників та інженерів у різних галузях.

Таким чином, спектральний аналіз пройшов тривалий шлях від теоретичних робіт Фур'є до сучасних цифрових методів обробки сигналів. Він став невід'ємною частиною багатьох галузей науки та техніки, зокрема звукової інженерії, радіотехніки, діагностики механічних систем тощо. Подальший розвиток методів спектрального аналізу та їхнє застосування у нових областях продовжує розширювати можливості дослідження складних сигналів.

1.1.3 Методи спектрального аналізу

Методи спектрального аналізу є різними способами, які дозволяють досліджувати спектри сигналів та їхні властивості. Звуковий спектральний аналіз повинен відображати наступні основні аспекти:

- Частотний склад звукового сигналу – визначення частотних компонентів, що входять до складу сигналу, та їх відносних амплітуд. Це дозволяє оцінити спектральний баланс звуку.

- Зміну спектру в часі – відображення динаміки спектру, тобто як змінюється частотний склад сигналу протягом часу. Це важливо для аналізу нестаціонарних звуків.

- Гармонічну структуру – виявлення основної частоти та її гармонік у періодичних сигналах, наприклад музичних інструментах. Гармонічний склад визначає тембр звуку.

- Наявність шумових компонентів – виявлення неперіодичних широкосмугових складових, що вказують на присутність шумів у сигналі. Шуми можуть маскувати корисний сигнал.

- Частотні межі сигналу – визначення діапазону частот, в якому зосереджена основна енергія звуку. Це важливо для оцінки смуги пропускання.

- Рівні звукового тиску на різних частотах – відображення амплітудно-частотної характеристики сигналу. Дозволяє оцінити баланс рівнів по спектру.

Методи спектрального аналізу можна класифікувати за кількома ознаками:

За типом сигналу, що аналізується:

- Аналіз неперервних сигналів (перетворення Фур'є)
- Аналіз дискретних (цифрових) сигналів (DFT, FFT)

За способом представлення спектра:

- Аналіз амплітудного спектра
- Аналіз фазового спектра
- Аналіз комплексного спектра

За часово-частотним представленням:

- Аналіз стаціонарних сигналів (класичний спектральний аналіз) [5]
- Аналіз нестаціонарних сигналів (короткочасне перетворення Фур'є, вейвлет-аналіз)

За типом спектрального вікна:

- Прямокутне вікно (класичне перетворення Фур'є)
- Вікна Хеммінга, Ханна чи Блекмана
- Адаптивні вікна (вікна змінної ширини)

Кожен з цих методів має свої особливості та області застосування. Наприклад, класичне перетворення Фур'є добре працює для аналізу стаціонарних сигналів, але має обмеження за роздільною здатністю у часі та частоті. Короткочасне перетворення Фур'є дозволяє аналізувати нестаціонарні сигнали, але вибір розміру вікна є компромісом між часовою та частотною роздільною здатністю. Вейвлет-аналіз забезпечує гнучке часово-частотне представлення сигналу, але вибір материнського вейвлета впливає на характер аналізу.

1.1.4 Метод швидкого перетворення Фур'є (FFT)

Метод швидкого перетворення Фур'є (FFT) — це ефективний алгоритм обчислення дискретного перетворення Фур'є (DFT). DFT дозволяє обчислити спектр дискретного сигналу, представивши його як суму гармонічних компонент з частотами, кратними основній частоті дискретизації. Однак пряме обчислення DFT має складність $O(N^2)$, де N — кількість відліків сигналу.

Алгоритм FFT, запропонований Джеймсом Кулі та Джоном Тьюкі у 1965 році та названий на їх честь, дозволив обчислювати DFT за складністю $O(N\,log\,N)$, що значно прискорило спектральний аналіз дискретних сигналів. FFT ґрунтується на розбитті вихідної послідовності на менші підпослідовності та рекурсивному обчисленні DFT для цих підпослідовностей.

Метод FFT широко застосовується у цифровій обробці сигналів, зокрема у звуковій інженерії. Він дозволяє швидко обчислювати спектри звукових сигналів, реалізувати фільтрацію у частотній області, аналізувати спектрограми тощо. FFT є основою багатьох алгоритмів синтезу та обробки звуку, таких як цифрова реверберація, еквалайзери, аналіз спектру потужності. [6]

Ще одним з методів підвищення точності визначення основної частоти за допомогою FFT є використання гармонічного добутку спектрів (Harmonic Product Spectrum, HPS). Цей метод був запропонований А. М. Ноллом в 1969 році.

Ідея HPS полягає в наступному:

- Обчислюється FFT вхідного звукового сигналу для отримання його спектру.

- Спектр стискається по частоті в кілька разів (наприклад, в 3 рази) і множиться сам на себе.

- Отриманий добуток спектрів має виражений пік в точці, що відповідає основній частоті сигналу.

$$Y(f) = \prod_{r=1}^{R} |X(fr)|$$

Рис. 1.1 – Основна ідея HPS – множення спектрів R з різним частотним масштабуванням.

Це пояснюється тим, що в точці основної частоти всі гармоніки (стиснуті спектри) перетинаються, підсилюючи один одного. В інших точках гармоніки не збігаються і взаємно гасяться. [7]

Однак FFT має певні обмеження. Оскільки він базується на DFT, то припускає, що сигнал є періодичним з періодом, рівним довжині вікна аналізу. Це може призводити до ефекту витоку спектра (spectral leakage) та появи бокових пелюсток у спектрі. Для зменшення цих ефектів застосовуються віконні функції, такі як вікна Хеммінга, Ханна чи Блекмана.

Використання віконної функції Ханна у спектральному аналізі дозволяє покращити точність оцінювання кутових координат. Віконна функція Ханна є прикладом вікна з високою роздільною здатністю, але з низьким динамічним діапазоном. Це означає, що вона добре розрізняє компоненти подібної амплітуди, тоді коли частоти близькі, але погано розрізняє компоненти різної амплітуди, тоді коли частоти далеко один від одного.

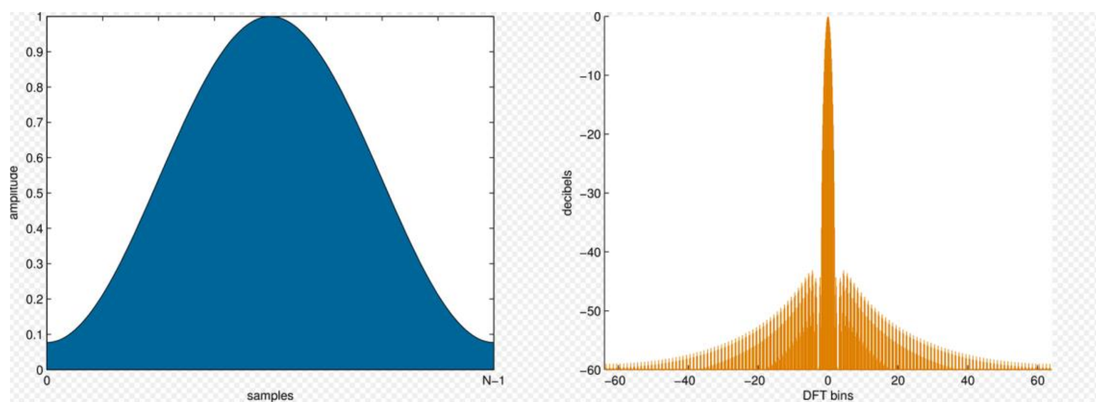

Рис. 1.2 – Результати використання віконної функції Ханна.

Незважаючи на ці обмеження, FFT залишається одним з найпотужніших та ефективних методів спектрального аналізу дискретних сигналів. Його реалізації доступні у багатьох бібліотеках цифрової обробки сигналів та математичного програмування.

1.1.5 Інші методи аналізу сигналів

Нижче наведені деякі з найпоширеніших методів спектрального аналізу:

- Дискретне перетворення Фур'є (DFT) – це метод, який дозволяє обчислити спектр дискретного звукового сигналу, представивши його як суму гармонічних коливань з частотами, кратними основній частоті дискретизації. DFT є основою цифрового спектрального аналізу звуку. Зауважте, що частотний діапазон є невід’ємною властивістю алгоритму DFT, але також існує тісний зв’язок із теоремою відліку Найквіста–Шеннона. Теорема стверджує, що неможливо отримати всю інформацію із сигналу, якщо найвищі зустрічаються частоти перевищують $f_s / 2$. Це означає, що DFT вже працює на теоретичній межі. [8]

- Кепстральний аналіз – метод, який дозволяє виявляти періодичні компоненти у спектрі звуку, такі як гармоніки. Кепстральний аналіз застосовується для аналізу мовних сигналів, виявлення збуджень у голосі, аналізу тембру музичних інструментів.

- Вейвлет-аналіз – метод, який забезпечує гнучке часово-частотне представлення звукового сигналу з адаптивною роздільною здатністю. Вейвлет-аналіз дозволяє виявляти тонкі ефекти, такі як частотна модуляція, та аналізувати нестаціонарні звуки. [9]

- Аналіз Вігнера-Віля – метод, який обчислює енергетичний спектр сигналу у кожній точці часово-частотної площини без використання вікна аналізу. Це дозволяє виявляти тонкі ефекти у звуках та аналізувати їх нестаціонарність.

Ці методи спектрального аналізу широко застосовуються в музиці та звукорежисурі для аналізу тембру музичних інструментів, розпізнавання нот та акордів, виявлення артефактів та шумів, синтезу звуків, реверберації, еквалізації тощо. Вони дозволяють отримати детальну інформацію про частотно-часові характеристики звукових сигналів та використовуються як в аналоговій, так і в цифровій обробці звуку.

1.2 Огляд переваг обраних інструментів та технологій

1.2.1 Мова програмування Python

Python — це МП високого рівня загального призначення, яка здобула популярність завдяки своїй простоті, читабельності та потужності. Вона була створена Гвідо ван Россумом і реліз її першої версії відбувся в 1991 році. Відтоді PY розвивався і став однією з найпопулярніших мов програмування у світі, що підтверджується її широким використанням у різних галузях, включаючи науку про дані та їх аналітику, веб-розробку, реалізація додатків, автоматизацію та штучний інтелект. [10]

Однією з основних переваг PY є його простота та зрозумілість. Синтаксис мови максимально наближений до природної англійської мови, що дозволяє легко читати та писати код. Це особливо важливо для новачків, які тільки починають вивчати програмування. PY дозволяє зосередитися на вирішенні проблеми, а не на синтаксичних деталях мови. Ця особливість робить МП популярним вибором для освітніх цілей та швидкого створення прототипів.

Ще однією важливою характеристикою PY є його велика базова бібліотека, яка забезпечує широкий спектр вбудованих модулів і функцій. Ця бібліотека дозволяє виконувати різноманітні завдання без необхідності встановлювати додаткові пакети. Для спектрального аналізу та обробки сигналів особливо корисними є бібліотеки для математичних розрахунків та візуалізації даних, які надають можливості для роботи з перетворенням Фур'є.

PY активно використовується в наукових дослідженнях і технічних обчисленнях завдяки наявності потужних бібліотек, що дозволяють проводити складні обчислення та аналіз даних. Це робить PY незамінним інструментом для реалізації програм, що потребують обробки сигналів та спектрального аналізу. В науковому середовищі PY отримав визнання завдяки своїй здатності ефективно обробляти великі обсяги даних та виконувати складні математичні розрахунки.

Кросплатформність даної МП є ще однією важливою перевагою. Програми, написані на PY, можуть виконуватися на різних операційних системах без необхідності внесення змін у код. Це дозволяє розробникам створювати універсальні рішення, які можуть працювати як на Windows, так і на macOS чи

Linux. Така гнучкість є важливою для розробників, які прагнуть забезпечити доступність своїх програм на різних платформах.

PY також має потужну і активну спільноту розробників, що сприяє швидкому розвитку мови та створенню нових бібліотек і інструментів. Спільнота забезпечує підтримку через форуми, документацію, навчальні посібники та відеоуроки, що робить процес навчання та розробки більш ефективним. Велика кількість відкритих проєктів і бібліотек доступних у репозиторіях, таких як GitHub, дозволяє розробникам використовувати готові рішення та адаптувати їх до своїх потреб.

PY також легко інтегрується з іншими мовами програмування та технологіями, такими як C/C++, Java, а також з різними платформами та інструментами. Це забезпечує гнучкість у розробці додатків та можливість використання найкращих інструментів для конкретних завдань. Наприклад, PY може бути використаний для побудови швидкого прототипу, а критично важливі компоненти можуть бути реалізовані на більш ефективних мовах, таких як C++ чи Rust.

Швидкий розвиток прототипів є однією з ключових переваг PY. Завдяки простоті і великій кількості бібліотек, МП дозволяє швидко створювати прототипи додатків і проводити експерименти. Це особливо важливо в контексті розробки програм для налаштування інструментів, де часто потрібно перевіряти різні алгоритми та методи аналізу сигналів.

МП також є основним інструментом у сфері машинного навчання та штучного інтелекту. Бібліотеки, що забезпечують потужний інструментарій для розробки та впровадження моделей машинного навчання, широко використовуються для покращення точності та ефективності налаштування інструментів. Це дозволяє створювати інтелектуальні системи, які можуть адаптуватися до змін умов і автоматично налаштовувати параметри інструментів для досягнення найкращих результатів.

PY є потужним і універсальним інструментом для розробки програмного забезпечення. Його простота, багатий набір бібліотек, кросплатформність та

активна спільнота роблять PY ідеальним вибором для виконання складних наукових та інженерних задач. Використання даної мови дозволяє швидко та ефективно розробляти додатки, проводити аналіз даних та впроваджувати сучасні технології в різних галузях.

1.2.2 Огляд бібліотек мови програмування Python

Однією із причин популярності МП PY є саме широкий спектр доступних бібліотек, які значно спрощують розробку програмного забезпечення. Ці бібліотеки надають готові рішення для різних завдань, дозволяючи розробникам швидко та ефективно створювати програми без необхідності повторного винаходження велосипеда. Давайте розглянемо найпопулярніші бібліотеки, які допомагатимуть у розробці ПЗ:

- **Tkinter** є базовою бібліотекою для створення графічного інтерфейсу користувача (GUI) у PY. Вона є частиною стандартної бібліотеки PY, тому не потребує додаткової установки. Tkinter надає простий і ефективний спосіб створення вікон, кнопок, текстових полів та інших елементів GUI. Завдяки інтеграції в стандартну бібліотеку, Tkinter є популярним вибором для створення кросплатформних графічних додатків на PY. Ця бібліотека дозволяє створювати форми для введення даних, кнопки для виконання дій та вікна для відображення результатів. [11]
- **NumPy** є бібліотекою для роботи з масивами та виконання числових обчислень. Це одна з основних бібліотек для наукових обчислень у PY, яка надає потужні інструменти для роботи з багатовимірними масивами та матрицями. NumPy включає велику кількість математичних функцій, що дозволяють виконувати операції над масивами швидко та ефективно. Ця бібліотека є незамінною для багатьох наукових і інженерних задач, де необхідні точні та швидкі обчислення. [12]
- **SoundDevice** є бібліотекою для роботи з аудіо у PY, що забезпечує простий інтерфейс для запису та відтворення звуку. Вона дозволяє працювати з аудіопристроями безпосередньо з PY, що робить її корисною для додатків, які

потребують обробки звуку в реальному часі. SoundDevice може використовуватися для запису аудіо сигналів, які потім можуть бути оброблені за допомогою інших бібліотек для аналізу та візуалізації. [13]

- **SciPy** є бібліотекою для наукових і технічних обчислень, що розширює можливості NumPy. Вона включає модулі для оптимізації, інтеграції, інтерполяції, спеціальних функцій, обробки сигналів та інших задач. SciPy забезпечує інструменти для виконання складних математичних операцій та аналізу даних, що робить її важливим інструментом для наукових досліджень та інженерних розробок. [14]

- **SoundFile** є бібліотекою для роботи з аудіофайлами у Python. Вона дозволяє легко зчитувати і записувати аудіофайли у різних форматах, таких як WAV, FLAC та інші. SoundFile забезпечує простий і ефективний спосіб роботи з аудіоданими, що робить її корисною для додатків, які потребують зберігання та обробки аудіоінформації. [15]

- **Pillow** є бібліотекою для обробки зображень у PY, яка є форком попередньої бібліотеки PIL (Python Imaging Library). Вона надає інструменти для роботи з різними форматами зображень, включаючи зчитування, запис, редагування та маніпуляцію зображеннями. Pillow підтримує широкий спектр форматів зображень, таких як JPEG, PNG, BMP, GIF та інші. Ця бібліотека дозволяє виконувати різноманітні операції з зображеннями, що робить її корисною для багатьох застосувань, включаючи візуалізацію результатів, створення графічних інтерфейсів та обробку зображень. [16]

- **Matplotlib** є бібліотекою для візуалізації даних у PY. Вона забезпечує інструменти для створення різноманітних графіків, діаграм і візуалізацій, що дозволяє ефективно представляти дані. Matplotlib підтримує створення статичних, анімованих та інтерактивних графіків, що робить її потужним інструментом для аналізу та візуалізації результатів. Ця бібліотека дозволяє створювати високоякісні графіки, які можуть бути інтегровані у звіти або наукові публікації. [17]

- **TensorFlow** та **PyTorch** є бібліотеками для машинного навчання, які забезпечують інструменти для створення, тренування та впровадження моделей машинного навчання. Ці бібліотеки широко використовуються для розробки алгоритмів штучного інтелекту(Artificial Intelligence), обробки природної мови, комп'ютерного зору та інших задач машинного навчання. Вони дозволяють створювати складні нейронні мережі та працювати з великими наборами даних, забезпечуючи високу продуктивність та ефективність. [18]
- **Flask** та **Django** є фреймворками для веб-розробки, які дозволяють створювати веб-додатки та API з використанням PY. Flask є мікрофреймворком, що забезпечує простий і гнучкий підхід до створення веб-додатків, тоді як Django є повноцінним фреймворком, що надає потужні інструменти для розробки складних веб-додатків з інтегрованою підтримкою баз даних, автентифікації користувачів та інших функцій. Обидва фреймворки є популярними серед розробників завдяки своїй зручності та широкому спектру можливостей. [19]
- **Requests** є бібліотекою для роботи з HTTP-запитами, яка спрощує взаємодію з веб-сервісами та API. Вона забезпечує зручний інтерфейс для відправки HTTP-запитів та отримання відповідей, підтримуючи всі основні методи HTTP, такі як GET, POST, PUT, DELETE та інші. Requests широко використовується для інтеграції з веб-сервісами, отримання даних з веб-ресурсів та автоматизації веб-запитів. [20]

PY також пропонує широкий спектр вбудованих бібліотек, які роблять його потужним інструментом для розробки програмного забезпечення у різних галузях. Ці бібліотеки охоплюють різні аспекти програмування, включаючи роботу з даними, наукові обчислення, машинне навчання, веб-розробку, обробку зображень та аудіо, візуалізацію даних та багато іншого. Завдяки багатству доступних бібліотек, PY є ідеальним вибором для реалізації різноманітних проектів.

Отож, Бібліотеки Python забезпечують широкий спектр можливостей для розробників, дозволяючи вирішувати різноманітні завдання та створювати ефективні рішення. Вибір конкретних бібліотек залежить від потреб проекту та

специфіки завдань, але незалежно від вибору, МП PY надає потужний інструментарій для досягнення поставлених цілей.

1.3 Огляд та аналіз алогічних програмних застосунків

Існує чимало програмних продуктів, призначених для налаштування музичних інструментів, і кожен з них має свої унікальні функції та особливості. Ніша музичних тюнерів є досить заповненою, що сприяє розвитку ринку та технологій в цілому завдяки спортивній конкуренції та суперництву. Розробники прагнуть зробити свої продукти більш зручними для користувачів (user-friendly), а також приділяють велику увагу якості інтерфейсу та алгоритмів. Вони намагаються залучити якомога більше клієнтів і працівників. Отже, ми можемо сміливо оцінити програмне забезпечення в цій сфері:

Однією з найбільш популярних і широко використовуваних програм є GuitarTuna. Цей додаток дозволяє музикантам швидко і точно налаштовувати свої інструменти, використовуючи сучасні технології для забезпечення високої точності налаштування.

GuitarTuna — це мобільний додаток для налаштування музичних інструментів, розроблений компанією Yousician. Він доступний для платформ iOS та Android і має широкий спектр функцій, які роблять його корисним для музикантів різних рівнів підготовки. [21]

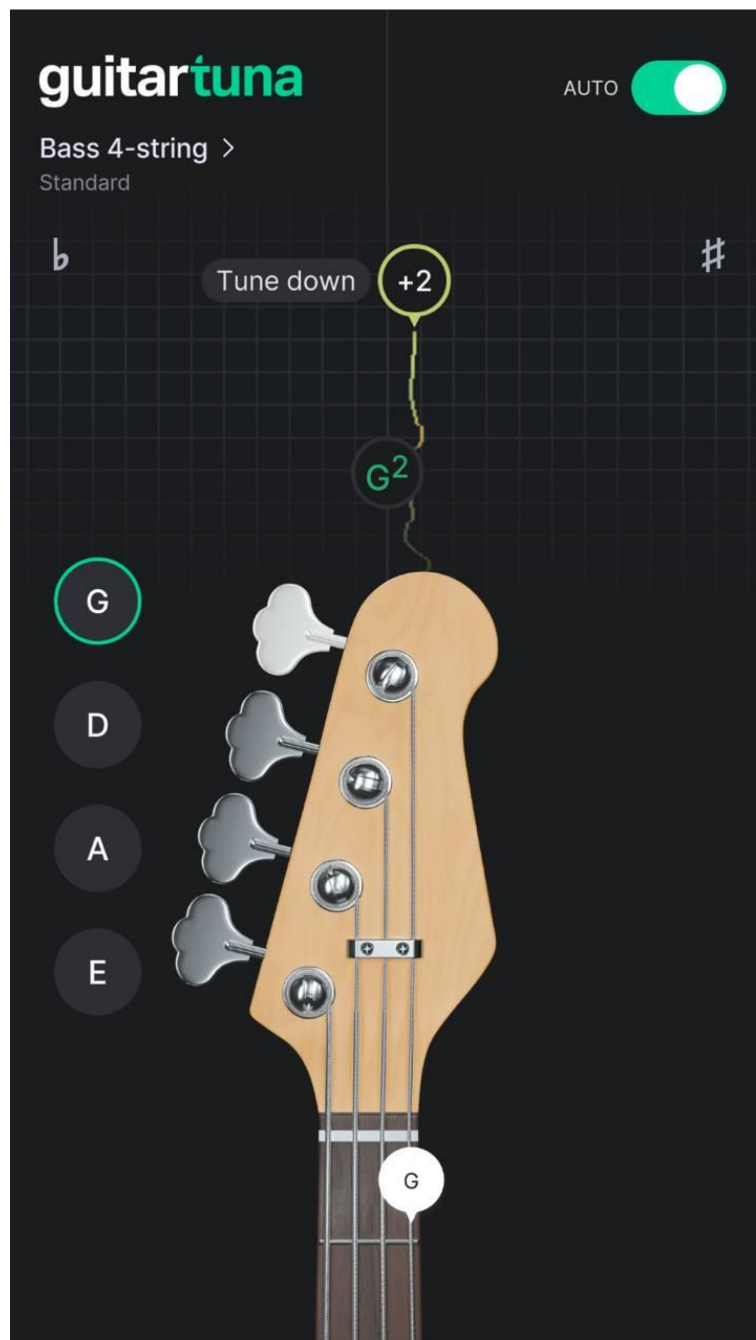

Рис. 1.3 – Інтерфейс застосунку GuitarTuna.

Розглянемо основні характеристики, переваги та недоліки:

Основні характеристики застосунку GuitarTuna:

- Використання передових алгоритмів для забезпечення високої точності налаштування інструментів. Він може налаштовувати різні типи струнних інструментів, включаючи гітари, бас-гітари, укулеле та інші.

- **О**крім налаштування, GuitarTuna включає в себе інтерактивні вправи, які допомагають музикантам покращити свої навички гри. Це включає тренування слуху, вправи на визначення акордів та інші музичні завдання.

- Наявність вбудованого метроному допомагає музикантам тримати ритм під час гри. Також важливим функціоналом є підтримка різних налаштувань для музичних інструментів що дозволяє користувачам експериментувати з різними звуками.

Головні переваги:

- Інтуїтивно зрозумілий інтерфейс дозволяє користувачам швидко розібратися в додатку та почати налаштовувати свої інструменти без необхідності тривалої підготовки чи адаптації.

- Застосунок поєднує в собі декілька корисних інструментів для музикантів, що робить його універсальним додатком для налаштування і тренування.

Серед недоліків можна зустріти наступне:

- Багато корисних функцій доступні тільки в платній версії додатку, що може бути недоліком для тих, хто не готовий витрачати допоміжні фінанси на застосунок.

- У безкоштовній версії додатку часто з'являються рекламні оголошення, які можуть відволікати користувача під час налаштування інструменту

- Для практикуючих музикантів відсутність хроматичного тюнеру призводить до того що вони не можуть повністю проекспериментувати із безлічами варіацій налаштування, так як застосунок обмежений у пресетах.

Наступним ми можемо розглянути інструмент Fender Tune – це ще один популярний додаток для налаштування музичних інструментів, розроблений компанією Fender Musical Instruments Corporation. Він доступний для платформ iOS та Android і забезпечує високий рівень точності та багатофункціональність. Нижче наведено огляд основних можливостей Fender Tune та аналіз його функціональності. [22]

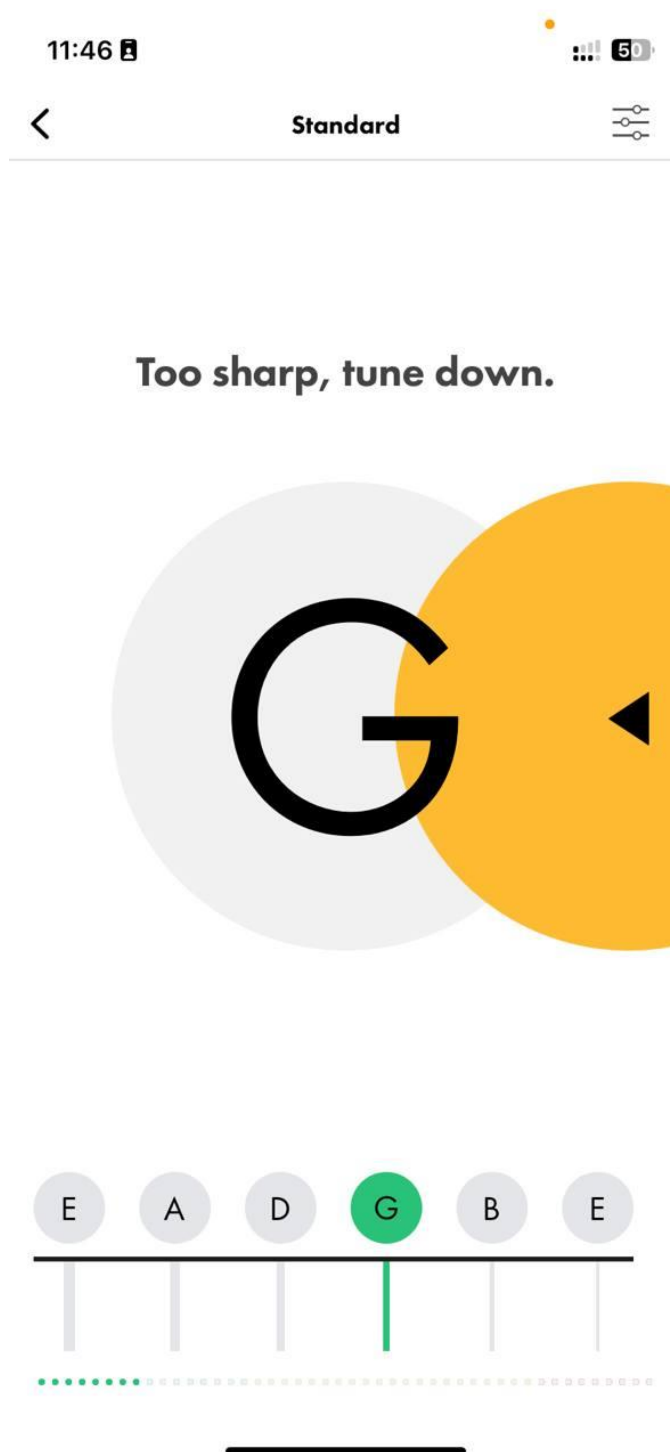

Рис. 1.4 – Інтерфейс застосунку Fender Tune.

Основні характеристики застосунку Fender Tune:

- Fender Tune підтримує налаштування для широкого спектру струнних інструментів, включаючи гітари, бас-гітари, укулеле та інші. Він забезпечує точне налаштування для різних музичних жанрів.

- Додаток дозволяє користувачам змінювати частоту калібрування, що робить його гнучким інструментом для музикантів, які грають в різних музичних традиціях.

- Fender Tune надає інтерактивні уроки та вправи, що допомагають музикантам покращити свої навички. Це включає тренування слуху, вправи на визначення акордів та ритм-ігри.

- Додаток включає широку бібліотеку акордів та схем налаштування для різних жанрів музики.

Головні переваги:

- Застосунок надає користувачам можливість налаштовувати інструменти для різних музичних жанрів та традицій, завдяки можливості зміни частоти калібрування.

- Уроки та вправи роблять процес навчання більш захоплюючим та ефективним, що є значною перевагою для музикантів-початківців.

Серед недоліків зазначмо наступне:

- Деякі функції, такі як розширені уроки та вправи, доступні тільки в платній версії додатку, що може бути недоліком для користувачів, які не готові платити за додатковий контент.

- Доступний лише на мобільних пристроях, що може бути незручним для тих, хто воліє використовувати програмне забезпечення на комп'ютері або спеціалізованих пристроях.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ НАЛАШТУВАННЯ ІСТРУМЕНТІВ ЗА ДОПОМОГОЮ ПЕРЕТВОРЕННЯ ФУР'Є(МОВОЮ PYTHON)

2.1 Постановка задачі та позначення вимог для застосунку налаштування музичних інструментів «Chromatuna»

Програма «Chromatuna» призначена для налаштування струнних музичних інструментів за допомогою спектрального аналізу звуків з використанням перетворення Фур'є мовою програмування Python. Після детального аналізу та оцінки різноманітних ПЗ та додатків було сформовано найважливіші вимоги проекту, давайте розберемо їх більш детально:

- програма повинна забезпечувати високу точність визначення частот для точного налаштування інструментів. Це досягається через використання алгоритму HPS та FFT для аналізу звукових сигналів;
- програма повинна бути інтуїтивно зрозумілою і легкою у використанні, надавати користувачам змогу швидко налаштовувати інструменти;
- ПЗ повинен підтримувати налаштування різних струнних інструментів, таких як гітара, скрипка, бас-гітара тощо. Користувачі повинні мати можливість вибору інструменту та варіацій налаштування;
- програма повинна містити функцію хроматичного тюнера, яка дозволяє налаштовувати інструменти в будь-якій тональності;
- користувачі повинні мати змогу змінювати налаштування програми відповідно до своїх потреб, включаючи зміну чутливості інструмента та інші параметри уже після того як зробили першочерговий вибір;
- застосунок повинен бути стабільним та продуктивним, забезпечуючи швидке оброблення звукових сигналів у режимі реального часу та надаючи точний результат.

Окрім вимог до проекту, однак, було також було проаналізовано і сформовано обмеження, без імплементації яких, користування застосунком стає неможливим.

Вимоги до користувачів:

- після входу в програму користувачі повинні вибрати інструмент, який вони бажають налаштувати;
- юзери повинні налаштувати параметри тюнера, такі як тип налаштування (наприклад, стандартне налаштування гітари або альтернативне налаштування) та інші;
- користувачі повинні слідувати інструкціям на екрані для налаштування інструменту, використовуючи візуальні підказки та індикатори точності;
- одразу після завершення налаштування користувачі повинні мати можливість закрити застосунок;

Ці вимоги та рекомендації забезпечують створення ефективного та зручного інструменту для налаштування струнних інструментів, що відповідає потребам сучасних музикантів.

2.2 Аналіз та вибір моделі розробки програмного засобу

При розробці програмного засобу було обрано ітераційну модель розробки. Ітераційна модель є гнучкою та ефективною методологією, яка дозволяє поступово вдосконалювати продукт через серію циклів розробки. Ця модель складається з кількох ітерацій, кожна з яких включає етапи планування, вимог, аналізу та проектування, реалізації, розгортання, тестування, управління конфігурацією та змінами, середовища та еволюції.

Загалом, ітераційна модель складається з конкретних етапів, які почергово виконуються при кожному новому циклі, що детальніше демонструє рис. 2.1.

Рис. 2.1 – Принципи та етапи ітераційної моделі розробки. [23]

Ітераційна модель розробки, як видно на зображенні, складається з циклів, які повторюються до досягнення бажаного результату. Кожна ітерація представляє собою завершений цикл, в кінці якого досягається певний функціональний продукт, готовий до використання. Далі розглянемо кожну ітерацію та її етапи детальніше:

На початку розробки необхідно зробити початкове планування що включає у себе визначення загальних цілей проекту, ресурсів та часових рамок. Наступним етапом визначення вимог. Тут збираються та аналізуються вимоги до продукту. Вимоги включають функціональні та нефункціональні аспекти, які продукт повинен виконувати. Згодом йде аналіз та проектування – на основі зібраних вимог проводиться аналіз та проектування системи. Визначаються архітектура, компоненти та інтерфейси. Слідом йде реалізація, тут розробник пише код, створюючи нові функції та інтегруючи їх у ПЗ. Одразу після імплементації нових функцій йде розгортання програмного забезпечення для тестування та відлагодження роботі а також цей етап включає перевірку функціональності, виявлення та виправлення помилок.

Після проходження такого спринту, можливе добавлення нового функціоналу – еволюція, що являє собою постійне вдосконалення продукту на основі зворотного зв'язку від користувачів та нових вимог. Зазвичай, після виконання етапу усі етапи, за виключенням початкового планування йдуть один за одним.

На першому етапі, створювалися вимоги до програмного продукту, після відповідного аналізу аналогічних сервісів. Було створено технічне завдання та визначено функціональні та нефункціональні вимоги. Згодом стратегія поділяє процес розробки на окремі ітерації, кожна з яких має чітко визначені цілі та завдання. Пріоритети додатково визначаються для кожної окремої ітерації. Наступним кроком для нас є проектування, ми розробляємо загальну архітектуру системи чітко визначаємо модулі та взаємодію між ними. Тут пояснюється внутрішня структура кожного модуля, вибираються технології та інструменти для реалізації.

Ітераційна модель має чимало переваг, серед яких зазначимо наступні:

- Ітераційна модель дозволяє швидко реагувати на зміни вимог та умов. Це особливо важливо в проектах, де потреби користувачів можуть змінюватися;
- Кожна ітерація дозволяє виявляти та виправляти помилки, що покращує якість кінцевого продукту;
- Завдяки розбиттю проекту на менші частини та постійному тестуванню, зменшується ризик виникнення критичних помилок на пізніх етапах розробки;
- Усі етапи розробки є прозорими для замовників та користувачів, що дозволяє їм бачити прогрес і вносити свої пропозиції.

До недоліків ітераційної моделі розробки можна віднести:

- Через непередбачуваність вимог та часті зміни, планування кожної ітерації може бути складним та вимагати значних ресурсів
- Постійне тестування та вдосконалення продукту можуть збільшувати загальні витрати на розробку;
- Завдяки розбиттю проекту на менші частини та постійному тестуванню, зменшується ризик виникнення критичних помилок на пізніх етапах розробки;

- Усі етапи розробки є прозорими для замовників та користувачів, що дозволяє їм бачити прогрес і вносити свої пропозиції.

Загалом, ітераційна модель є досить ефективним способом розробки ПЗ, так як забезпечує гнучкість, високу якість та відповідність продукту. Кожна ітерація дозволяє поступово вдосконалювати продукт, отримувати зворотний зв'язок та швидко реагувати на зміни. Такий підхід забезпечує створення надійного та зручного інструменту, який відповідає сучасним вимогам та очікуванням користувачів.

2.3 Загальний опис проєкту

Основний функціонал програми можна наочно проілюструвати за допомогою діаграми діяльності, що чітко відображає послідовність дій та логіку роботи програми:

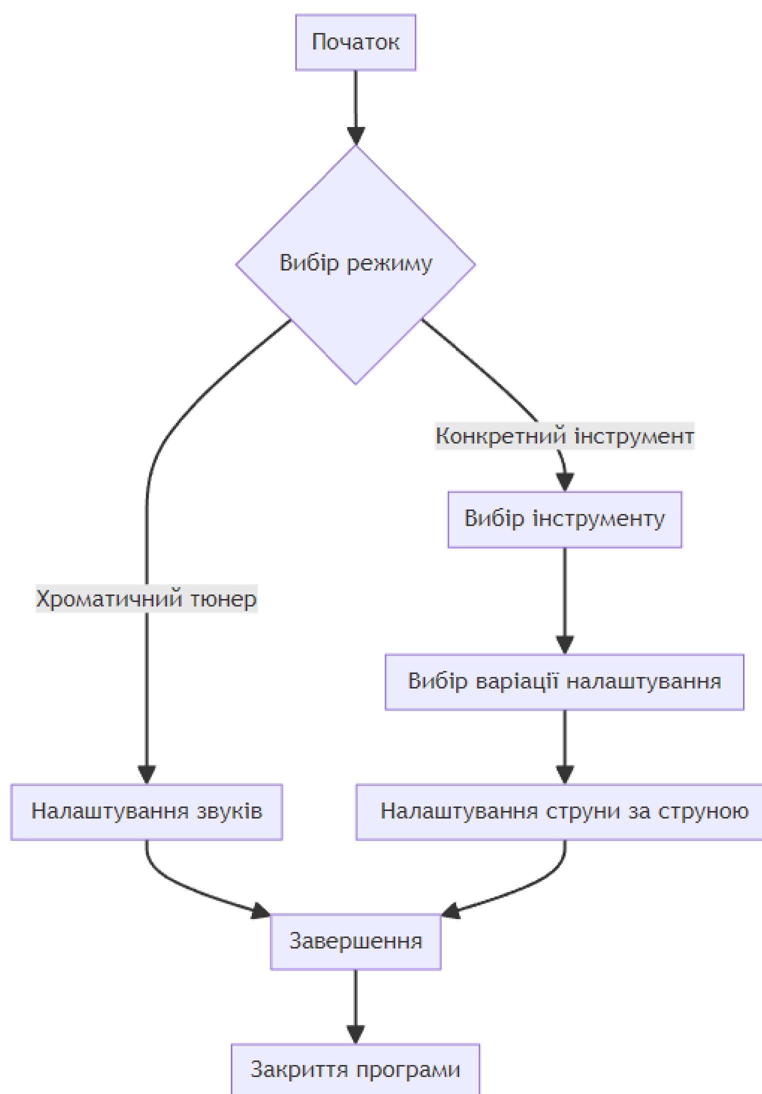

Рис. 2.2 – Діаграма діяльності програми «Chromatuna».

Програма розпочинається з вибору режиму роботи, де користувач має можливість обрати один із двох варіантів: «Chromatic Only» або вибравши конкретний інструмент. У режимі «Chromatic Only» програма здійснює налаштування звуків відповідно до стандартної хроматичної гами, забезпечуючи точність налаштування для широкого кола музичних інструментів.

Альтернативно, якщо було обрано певний музичний інструмент, для якого буде проводитися налаштування. Після цього користувач може вибрати одну з доступних варіацій налаштування, що може відрізнятися залежно від типу інструменту, його конфігурації та особливостей налаштування.

Наступним важливим кроком є налаштування струни за струною. Саме на цьому етапі програма застосовує потужний алгоритм визначення частоти на основі перетворення Фур'є, який можна візуалізувати за допомогою наступного зображення:

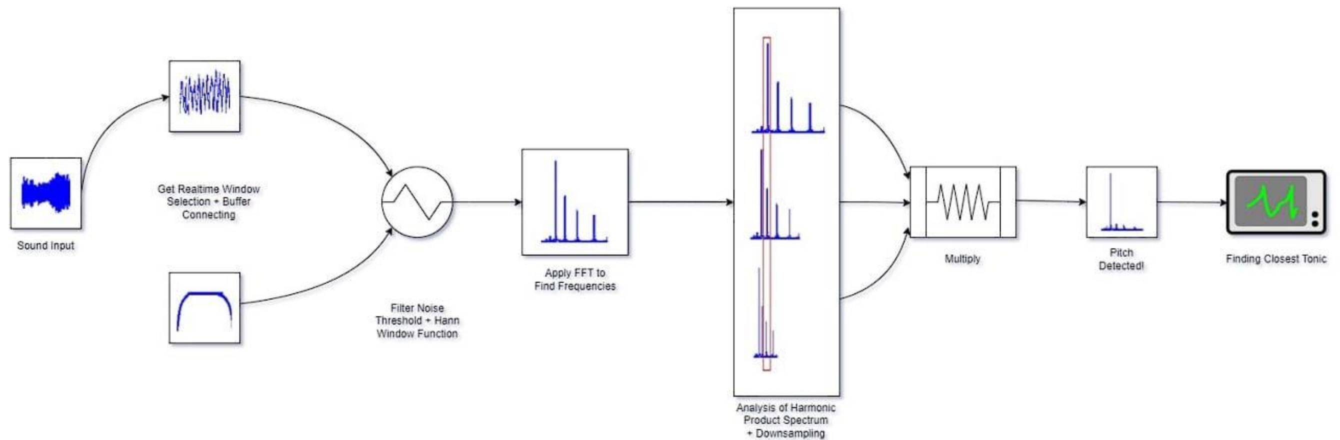

Рис. 2.3 – Алгоритмічна діаграма визначення частот «Chromatuna».

Таким чином, програма налаштування інструментів за допомогою перетворення Фур'є (мовою Python) забезпечує користувачів зручним та універсальним інструментом для налаштування широкого спектру музичних інструментів. Використовуючи передові методи спектрального аналізу на основі перетворення Фур'є для точного визначення частоти звуку, ця програма дозволяє досягти високої точності налаштування, економлячи час та зусилля користувачів. Гнучкість налаштування, підтримка різноманітних інструментів та інтуїтивно зрозумілий інтерфейс роблять цю програму корисним інструментом для музикантів та ентузіастів музики.

2.4 Обґрунтування вибору інструментальних засобів розробки

2.4.1 Вибір PyCharm для розробки програми налаштування музичних інструментів

У процесі розробки програми налаштування музичних інструментів за допомогою перетворення Фур'є було обрано PyCharm як основне інтегроване середовище розробки (IDE). Це рішення було прийнято завдяки численним

перевагам, які пропонує PyCharm для створення додатків на мові програмування Python. [24]

PyCharm є потужним та гнучким інструментом, який забезпечує зручний доступ до широкого спектру бібліотек та інструментів, необхідних для реалізації складних алгоритмів обробки звуку та спектрального аналізу. Серед ключових бібліотек, які легко інтегруються в PyCharm, варто згадати NumPy, SciPy та matplotlib, що є незамінними для реалізації методів перетворення Фур'є та візуалізації результатів.

Одним з головних плюсів PyCharm є вбудований інтерпретатор Python та зручна система керування пакетами, що дозволяє швидко встановлювати та використовувати потрібні бібліотеки та інструменти. Це значно економить час та зусилля розробників, дозволяючи їм зосередитися на безпосередній реалізації функціоналу програми.

Не менш важливою перевагою PyCharm є його потужні можливості для відлагодження коду. Інтегрована система відлагодження допомагає розробникам швидко виявляти та виправляти помилки, а також відстежувати поведінку програми на кожному етапі її виконання, що є критично важливим при роботі зі складними алгоритмами обробки звуку та спектрального аналізу.

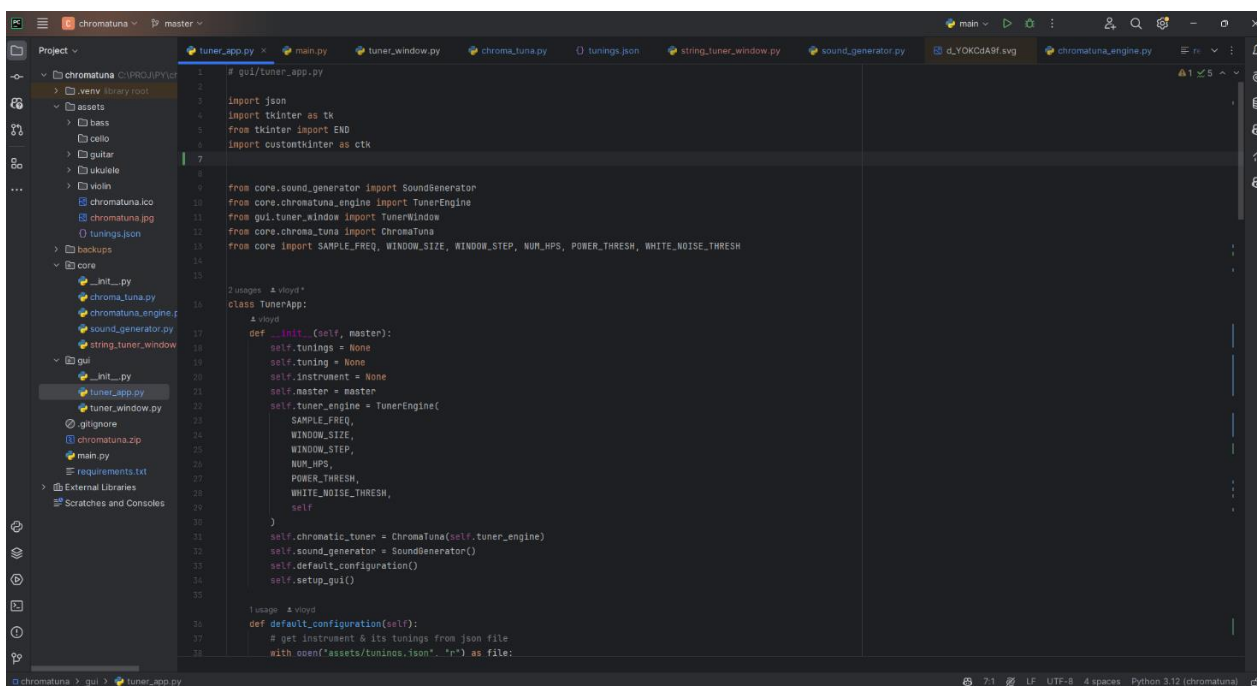

Рис. 2.4 – Інтерфейс програми PyCharm.

Крім того, PyCharm пропонує дружній інтерфейс користувача та широкі можливості налаштування, що дозволяє розробникам адаптувати середовище розробки під свої індивідуальні потреби та вподобання. Це сприяє підвищенню продуктивності та зручності роботи, оскільки розробники можуть налаштувати PyCharm таким чином, щоб він максимально відповідав їхньому стилю роботи та вимогам проекту.

Загалом, PyCharm є потужним та зручним інструментом для розробки додатків на Python, який забезпечує розробників необхідними функціями та можливостями для ефективної реалізації складних алгоритмів та досягнення високої якості коду. Саме тому PyCharm був обраний як оптимальне середовище розробки для створення програми налаштування музичних інструментів за допомогою перетворення Фур'є.

2.4.2 Застосування бібліотеки CustomTkinter для розробки і створення інтерфейсу програмного засобу

Під час розробки програми налаштування музичних інструментів за допомогою перетворення Фур'є, була обрана бібліотека CustomTkinter для створення інтуїтивного та зручного інтерфейсу користувача. CTk є сучасною та потужною бібліотекою Python, яка базується на популярному фреймворку Tkinter і пропонує додаткові можливості для стилізації та кастомізації графічного інтерфейсу.

Одною з ключових причин вибору CTk є її здатність створювати привабливі та сучасні інтерфейси, які відповідають найновішим тенденціям в галузі дизайну користувацького досвіду (UX). Бібліотека пропонує широкий вибір готових компонентів інтерфейсу, таких як кнопки, поля введення, вікна діалогу та багато іншого, що дозволяє швидко та ефективно створювати функціональні і приємні для ока інтерфейси.

Крім того, CTk надає розробникам гнучкі можливості для налаштування зовнішнього вигляду та поведінки кожного елемента інтерфейсу. Завдяки цьому, програма налаштування музичних інструментів може бути адаптована під конкретні вимоги користувачів та забезпечити максимально зручний та інтуїтивно зрозумілий досвід взаємодії.

Окрім візуальної привабливості, CTk також пропонує широкий набір функцій для обробки подій та взаємодії з користувачем. Це дозволяє реалізувати складну логіку програми, такі як обробка введених даних, відображення результатів та інтеграція з іншими компонентами програми.

Важливо відзначити, що CTk є кросплатформною бібліотекою, що забезпечує однакову поведінку та зовнішній вигляд інтерфейсу на різних операційних системах, таких як Windows, macOS та Linux. Це гарантує, що програма налаштування музичних інструментів буде працювати однаково на різних платформах, що є важливою вимогою для забезпечення широкої доступності та зручності використання програми.

```
2 usages
class StringTunerWindow(ctk.CTkToplevel):
    def __init__(self, master, app, instrument, tuning, string_order, target_freq, tuner_engine, chromatic_tuner):
        super().__init__(master)
        self.sound_generator = SoundGenerator()
        self.title(f"Tuning {instrument.capitalize()} - {tuning.capitalize()}")
        self.geometry("600x1200")
        self.app = app
        self.instrument = instrument
        self.tuning = tuning
        self.string_order = string_order
        self.current_string_index = 0
        self.target_freq = target_freq
        self.tuner_engine = tuner_engine
        self.chromatic_tuner = chromatic_tuner
```

Рис. 2.5 – Імплементація вікна типу TopLevel у бібліотеці CTk.

Загалом, застосування бібліотеки CTk для розробки інтерфейсу програмного засобу налаштування музичних інструментів за допомогою перетворення Фур'є дозволило створити привабливий, зручний та функціональний інтерфейс користувача, що сприяє покращенню загального досвіду взаємодії з програмою та підвищенню її практичної цінності для користувачів.

2.5 Особливості програмної реалізації

2.5.1 Реалізація побудови інтерфейсу користувача та побудова функціоналу вибору інструменту та вибору варіацій налаштування

Реалізація побудови інтерфейсу користувача виконується у файлі tuner_app.py. Використовується бібліотека Tkinter разом із її розширенням CTk, що дозволяє створювати сучасний вигляд для додатку. Головний клас TunerApp відповідає за ініціалізацію всіх необхідних компонентів та налаштування графічного інтерфейсу.

```
class TunerApp:
    vloyd
    def __init__(self, master):
        self.tunings = None
        self.tuning = None
        self.instrument = None
        self.master = master
        self.tuner_engine = TunerEngine(
            SAMPLE_FREQ,
            WINDOW_SIZE,
            WINDOW_STEP,
            NUM_HPS,
            POWER_THRESH,
            WHITE_NOISE_THRESH,
            self
        )
        self.chromatic_tuner = ChromaTuna(self.tuner_engine)
        self.sound_generator = SoundGenerator()
        self.default_configuration()
        self.setup_gui()
```

Рис. 2.6 – Побудова інтерфейсу та класу TunerApp.

У наведеному коді відбувається ініціалізація основних компонентів: двигуна для налаштувань, генератора звуків та хроматичного тюнера. Після цього викликаються методи для завантаження конфігурації за замовчуванням та налаштування графічного інтерфейсу.

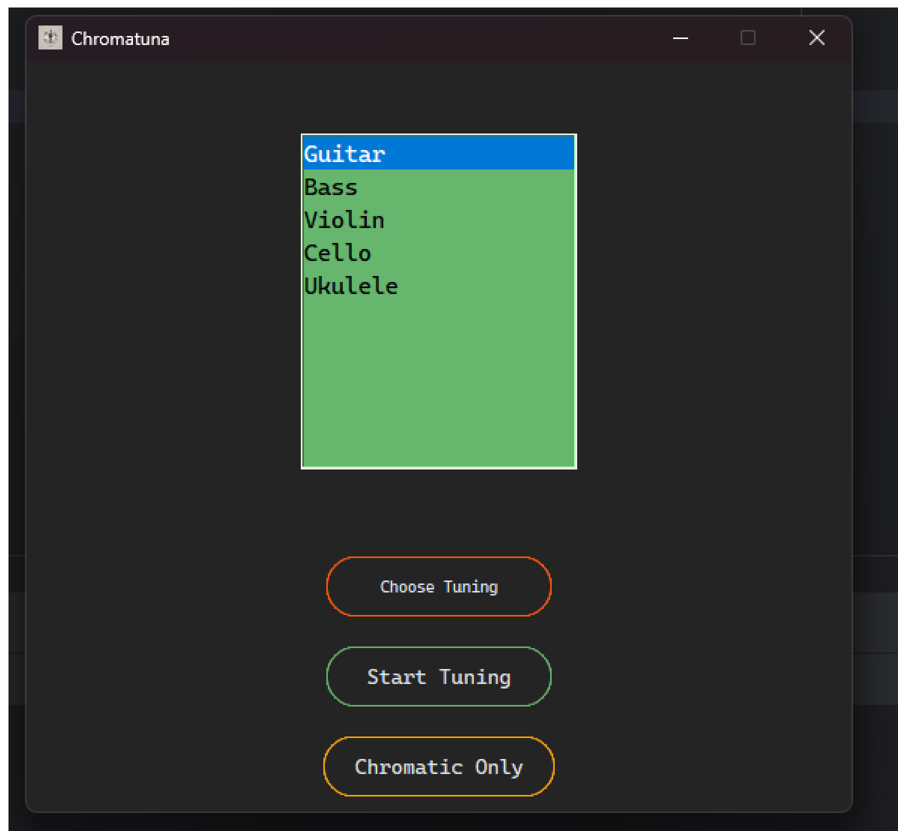

Рис. 2.7 – Головне вікно програми.

Налаштування конфігурацій відбувається у методі default_configuration що завантажує дані про інструменти та їхні налаштування з JSON-файлу tunings.json. Це дозволяє програмі підтримувати кілька інструментів та різні типи альтернативних налаштувань для кожного з них. До того ж, формат JSON є дуже зручним для зберігання динамічних даних, які можна буде оновлювати.

```
{
  "guitar": {
    "standard": [{
      "name": "Standard Tuning: EADGBE",
      "tuning": {
        "E2": 82.4,
        "A2": 110.0,
        "D3": 146.8,
        "G3": 196.0,
        "B3": 246.9,
        "E4": 329.6
      }
    }],
    "drop_d": [{
      "name": "Drop D: DADGBE",
      "tuning": {
        "D2": 73.4,
        "A2": 110.0,
        "D3": 146.8,
```

Рис. 2.8 – Зберігання частот та варіацій налаштувань для інструменту гітари у форматі JSON.

Метод setup_gui відповідає за створення головного вікна програми та додавання всіх необхідних елементів інтерфейсу, таких як списки для вибору інструментів та налаштувань, кнопки для взаємодії з користувачем та області для відображення результатів.

```
def setup_gui(self):
    self.master.title("Chromatuna")
    self.master.geometry("550x500")
    self.master.resizable(False, False)
    self.master.iconbitmap("assets/chromatuna.ico")
    # self.master.wm_attributes('-toolwindow', 'True')

    instrument_list = tk.Listbox(
        self.master,
        font=("Cascadia Mono", 12),
        bg="#66B66E",
        selectmode=tk.SINGLE
    )

    instrument_list.pack(expand=False, padx=5, pady=5)
    instrument_list.bind('<<ListboxSelect>>', self.instrument_selected)

    for name in self.tunings:
        instrument_list.insert(END, *elements: name.capitalize())
    instrument_list.selection_set(list(self.tunings).index(self.instrument))
```

Рис. 2.9 – Побудова інтерфейсу – методу setup_gui.

2.5.2 Реалізація алгоритму визначення частот.

Основна частина алгоритму визначення частот реалізована у файлі chromatuna_engine.py в класі TunerEngine. Цей клас використовує методи обробки сигналів для аналізу звукового вхідного сигналу та визначення основної частоти.

Метод audio_callback класу TunerEngine відповідає за обробку вхідного звукового сигналу та визначення частоти. Використовуються методи обчислення перетворення Фур'є(FFT) та аналіз гармонічного продукту спектра (HPS) та застосування віконної функції Ханна для уникнення проблем із витоком потоків.

```
def audio_callback(self, indata, frames, time_info, status):
    if status:
        print(status)
        return
    if any(indata):
        self.window_samples = np.concatenate((self.window_samples, indata[:, 0]))
        self.window_samples = self.window_samples[len(indata[:, 0]):]

        signal_power = (np.linalg.norm(self.window_samples, ord=2) ** 2) / len(self.window_samples)
        if signal_power < self.power_thresh:
            print(f"Signal is too weak, check your connection: {signal_power} . Need at least {self.power_thresh}")
            return
```

Рис. 2.10 – audio_callback: відсіювання шумів та лишніх частот.

На цьому етапі отримання вхідного звукового сигналу програма отримує вхідний звуковий сигнал від мікрофону або іншого джерела звуку. Це відбувається за допомогою бібліотеки sounddevice, яка забезпечує доступ до аудіо-пристроїв на різних платформах. Функція audio_callback викликається кожен раз, коли нові аудіо-дані стають доступними. Ці дані зберігаються у буфері window_samples для подальшої обробки.

Згодом відбувається застосування віконної функції та обчислення FFT.

Однак FFT має певні обмеження. Оскільки він базується на дискретному перетворенні Фур'є (DFT), то припускає, що сигнал є періодичним з періодом, який рівний довжині вікна аналізу. Це також пов'язано з теоремою відліку Найквіста-Шеннона. Зокрема, це може призводити до витоку спектру та появи

бокових пелюсток у спектрі. Для зменшення цих ефектів використовуються віконні функції, такі як вікно Ханна.

Обчислення FFT та накладання вікна виконується за допомогою наступного коду:

```
self.window_size = 441000
self.hann_window = np.hanning(self.window_size)
hann_samples = self.window_samples * self.hann_window
# save fft data for charts visualisation.
self.fft_data = np.abs(np.fft.rfft(hann_samples))
self.fft_freqs = np.fft.rfftfreq(len(hann_samples), 1 / self.sample_freq)
```

Рисунок 2.11 – Використання FFT та віконної функції Ханна.

Наступним кроком ми робимо фільтрація шумів та нормалізацію спектру. Для зменшення впливу шуму та покращення точності визначення частоти, програма виконує фільтрацію спектру та його нормалізацію.

```
magnitude_spec = abs(scipy.fftpack.fft(hann_samples)[:len(hann_samples) // 2])

for i in range(int(16 / DELTA_FREQ)):
    magnitude_spec[i] = 0

for j in range(len(OCTAVE_BANDS) - 1):
    ind_start = int(OCTAVE_BANDS[j] / DELTA_FREQ)
    ind_end = int(OCTAVE_BANDS[j + 1] / DELTA_FREQ)
    ind_end = ind_end if len(magnitude_spec) > ind_end else len(magnitude_spec)
    avg_energy_per_freq = (np.linalg.norm(magnitude_spec[ind_start:ind_end], ord=2) ** 2) / (
            ind_end - ind_start)
    avg_energy_per_freq = avg_energy_per_freq ** 0.5
    for i in range(ind_start, ind_end):
        magnitude_spec[i] = magnitude_spec[i] if magnitude_spec[
                                                 i] > WHITE_NOISE_THRESH * avg_energy_per_freq else 0

mag_spec_ipol = np.interp(np.arange(0, len(magnitude_spec), 1 / NUM_HPS),
                          np.arange(0, len(magnitude_spec)),
                          magnitude_spec)
mag_spec_ipol = mag_spec_ipol / np.linalg.norm(mag_spec_ipol, ord=2)
```

Рис. 2.12 – Реалізація приглушення шумів а також подальша нормалізація спектру.

Також, для підвищення точності визначення основної частоти за ми імплементовуємо HPS, запропонований А. М. Ноллом у 1969 році. Його ціль

можна пояснити декількома словами: спектр у результаті FFT стискається по визначеним гармонікам в кілька разів та множиться сам на себе; отриманий добуток спектрів має виражений пік у точці, що відповідає основній частоті сигналу.

Це пояснюється тим, що в точці основної частоти всі гармоніки перетинаються, підсилюючи один одного. В інших точках гармоніки не збігаються і взаємно гасяться.

```
hps_spec = copy.deepcopy(mag_spec_ipol)

for i in range(NUM_HPS):
    tmp_hps_spec = np.multiply(hps_spec[:int(np.ceil(len(mag_spec_ipol) / (i + 1)))],
                               mag_spec_ipol[::(i + 1)])
    if not any(tmp_hps_spec):
        break
    hps_spec = tmp_hps_spec
max_ind = np.argmax(hps_spec)
max_freq = max_ind * (SAMPLE_FREQ / WINDOW_SIZE) / NUM_HPS

closest_note, closest_pitch = self.find_closest_note(max_freq)
max_freq = round(max_freq, 1)
closest_pitch = round(closest_pitch, 1)
```

Рис. 2.13 – Застосування HPS добутку для визначення основної частоти.

Визначивши основну частоту звуку та застосувавши HPS, ми можемо знайти найближчу ноту, яка відповідає цій частоті. Для порівняння, ми беремо концертну частоту – нота A4(440Hz).

```
def find_closest_note(self, pitch):
    i = int(np.round(np.log2(pitch / CONCERT_PITCH) * 12))
    closest_note = ALL_NOTES[i % 12] + str(4 + (i + 9) // 12)
    closest_pitch = CONCERT_PITCH * 2 ** (i / 12)
    return closest_note, closest_pitch
```

Рисунок 2.14 – find_closest_note визначає найближчу ноту то основної частоти звуку.

Отримавши найближчу частоту, ми прораховуємо різницю та кидаємо на інтерфейс користувача дані, на скільки Гц йому необхідно дотягнути струну щоб

закінчити її налаштування а також будуємо графіки на яких відображаємо дані – результат FFT та амплітуду звуку.

```
def update_gui(self, note, freq, pitch, diff=0):
    if self.running is False:
        return
    if self.closest_note_label:
        self.closest_note_label.configure(text=f"Closest Note: {note}")
    if self.freq_label:
        self.freq_label.configure(text=f"Freq: {freq}")
    if self.pitch_label:
        self.pitch_label.configure(text=f"Target Pitch: {pitch}")
    if self.diff_label:
        if diff != 0:
            diff = int(diff * 10)
            if diff > 0:
                diff = f"+{diff}"
            self.diff_label.configure(text=f"Difference: {diff}")
        else:
            self.diff_label.configure(text='')
```

Рис. 2.14 – find_closest_note визначає найближчу ноту то основної частоти звуку.

У результаті, GUI користувача при налаштуванні інструменту має наступний вигляд:

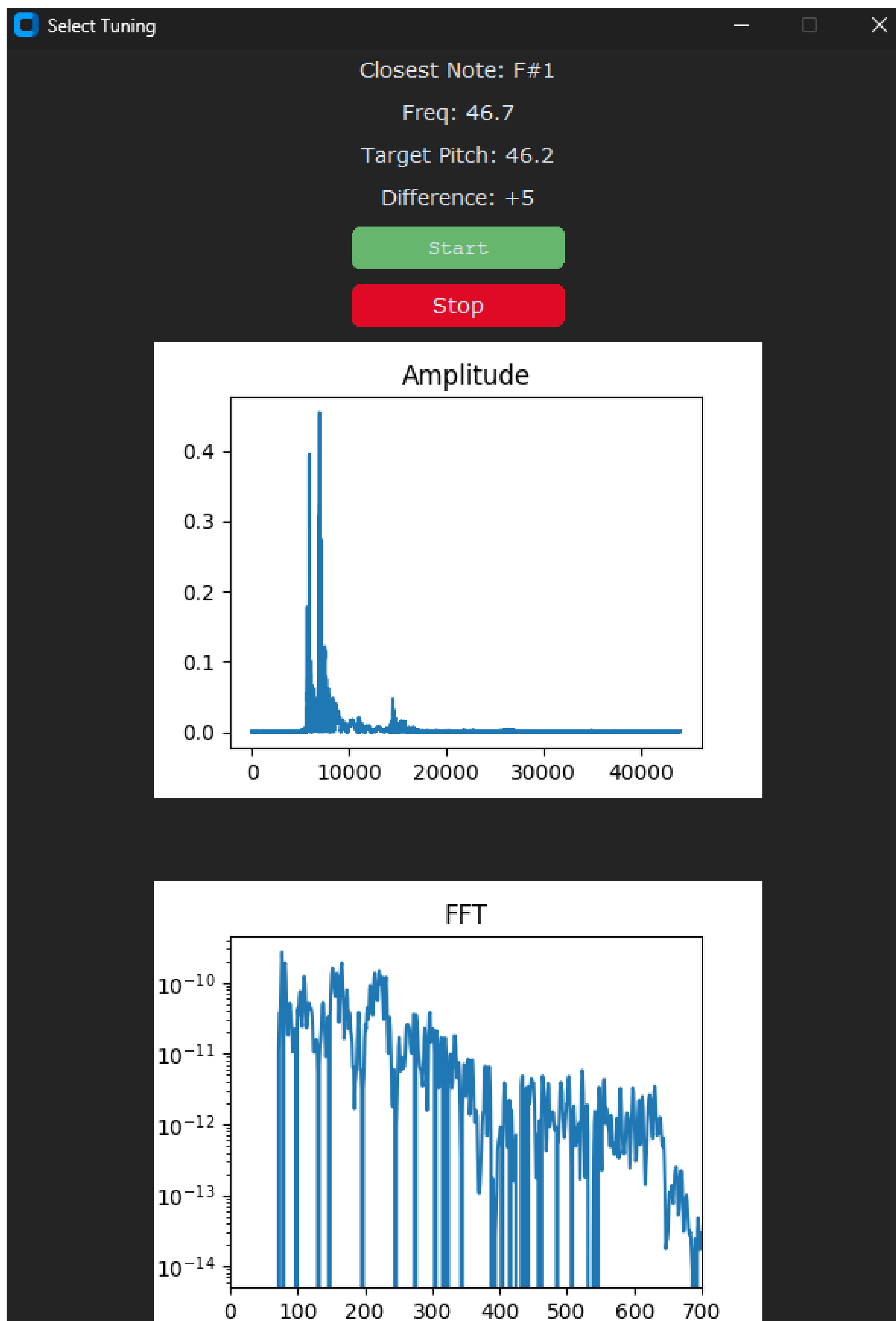

Рис. 2.15 – GUI при налаштуванні звучання струни.

2.6 Організація тестування та налагодження програмного засобу

Тестування продукту переважно виконувалося вручну, слідуючи етапам розробки. Відомо, що PyCharm добре взаємодіє з відлагоджувачами, тому для

тестування використовували PyDev. Він відстежував код, переходи між класами та повідомляв про будь-які неполадки. Також проводилося тестування сторонніми особами. Помилки часто виправляли, особливо ті, що стосувалися роботи з частотами, включаючи їх отримання та управління вікном. При виникненні помилок, вони логувалися у консоль проєкту. Якщо виявлялася критична помилка, програма призупиняла свою роботу.

Підрахування результатів. Також для перевірки працездатності та точності роботи тюнера було проведено серію тестів з різними музичними інструментами, включаючи гітару та бас-гітару. Програма показала високу точність визначення частот та відповідність нот, що підтвердило ефективність використаних алгоритмів спектрального аналізу. Програма була протестована в режимах реального часу та аналізу записаних звукових сигналів. В обох випадках тюнер продемонстрував стабільну роботу та високу точність результатів. Було також перевірено роботу програми в умовах різного рівня шуму – алгоритм працює за наявності шуму, але потребує вдосконалень.

Цей проект розміщений на сервісі контролю версій у віддаленому репозиторії GitHub. У разі додавання функціоналу або виправлення багів, виконується коміт та робиться новий реліз.

ВИСНОВКИ

У роботі було успішно досліджено методи спектрального аналізу на предмет їх застосування для розробки тюнера:

- Вивчено сучасні алгоритми спектрального аналізу, зокрема швидке перетворення Фур'є (FFT).
- Розроблено алгоритм для реалізації тюнера, заснованого на методах спектрального аналізу з використання FFT.
- Реалізовано HPS алгоритм для точного визначення частотних характеристик звукового сигналу
- Ознайомлено з основними алгоритмами та бібліотеками мови PY, що використовуються для спектрального аналізу.
- У середовищі розробки IDE PyCharm створено програмний продукт на основі мови програмування Python, який дозволяє налаштовувати музичні інструменти з високою точністю.
- Оцінено ефективність та точність роботи розробленого тюнера.

Цей проект також ілюструє глибоке розуміння принципів цифрової обробки сигналів і показує, як ефективно поєднувати теоретичні знання з практичною розробкою програмного забезпечення. «Chromatuna» є прикладом успішного поєднання технічних навичок, програмування і музичної теорії для створення корисного інструменту, який може бути використаний як професійними музикантами, так і аматорами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Маркітан В. Використання спектрального аналізу для побудови функціонального тюнера | міжнародна науково-практична конференція "Проблеми комп’ютерних наук, програмного моделювання та безпеки цифрових систем". Прикладні проблеми комп’ютерних наук, безпеки та математики. URL: https://apcssm.vnu.edu.ua/index.php/conf/article/view/91 (дата звернення: 19.06.2024).

2. Fourier J. Théorie analytique de la chaleur. Google Books. URL: https://books.google.com.ua/books?id=TDQJAAAAIAAJ (date of access: 19.06.2024).

3. Spectrum_analysis. chemeurope.com - The chemistry information portal from laboratory to process. URL: https://www.chemeurope.com/en/encyclopedia/Spectrum_analysis.html (date of access: 19.06.2024).

4. Adams V. H. Understanding the Cooley-Tukey FFT. V. Hunter Adams. URL: https://vanhunteradams.com/FFT/FFT.html (date of access: 19.06.2024).

5. Linear time-invariant system - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Linear_time-invariant_system (date of access: 19.06.2024).

6. Fast Fourier transform – FFT – Librow – Digital LCD dashboards for cars and boats. Digital instrument clusters and body control systems – Librow – Digital LCD dashboards for cars and boats. URL: http://www.librow.com/articles/article-10 (date of access: 19.06.2024).

7. Harmonic product spectrum (HPS). Department of Music. URL: http://musicweb.ucsd.edu/~trsmyth/analysis/Harmonic_Product_Spectrum.html (date of access: 19.06.2024).

8. Nyquist H. Certain topics in telegraph transmission theory. Probability Theory As Extended Logic. URL:

https://bayes.wustl.edu/Manual/CertainTopicsInTelegraphTransmissionTheory.pdf (date of access: 19.06.2024).

9. Polikar R. The wavelet tutorial. Wayback Machine. URL: https://web.archive.org/web/20040210231301/http://users.rowan.edu/~polikar/WAVELETS/WTtutorial.html (date of access: 19.06.2024).

10. Contributors to Wikimedia projects. Python (programming language) - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Python_(programming_language) (date of access: 19.06.2024).

11. Tkinter - Python interface to Tcl/Tk. Python documentation. URL: https://docs.python.org/uk/3/library/tkinter.html (date of access: 19.06.2024).

12. NumPy tutorials – NumPy Tutorials. NumPy. URL: https://numpy.org/numpy-tutorials/ (date of access: 19.06.2024).

13. Usage – python-sounddevice, version 0.4.7. Play and Record Sound with Python – python-sounddevice, version 0.4.7. URL: https://python-sounddevice.readthedocs.io/en/0.4.7/usage.html (date of access: 19.06.2024).

14. SciPy documentation – scipy v1.13.1 manual. Numpy and Scipy Documentation – Numpy and Scipy documentation. URL: https://docs.scipy.org/doc/scipy/ (date of access: 19.06.2024).

15. Python-soundfile – python-soundfile 0.11.0 documentation. python-soundfile – documentation. URL: https://python-soundfile.readthedocs.io/en/0.11.0/ (date of access: 19.06.2024).

16. Pillow documentation. Pillow (PIL Fork). URL: https://pillow.readthedocs.io/en/stable/ (date of access: 19.06.2024).

17. Matplotlib – visualization with python. Matplotlib – Visualization with Python. URL: https://matplotlib.org/ (date of access: 19.06.2024).

18. Порівняння найкращих інструментів машинного навчання: tensorflow, keras, scikit-learn і pytorch. Zfort Group. URL: https://www.zfort.com.ua/blog/porivnyannya-naikrashikh-instrumentiv-mashinnogo-navchannya-tensorflow-keras (дата звернення: 19.06.2024).

19. Differences between django vs flask. GeeksforGeeks. URL: https://www.geeksforgeeks.org/differences-between-django-vs-flask/#:~:text=Django%20includes%20a%20built-in,libraries%20for%20more%20comprehensive%20testing. (date of access: 19.06.2024).

20. Requests - documentation. PyPI. URL: https://pypi.org/project/requests/ (date of access: 19.06.2024).

21. GuitarTuna | Online Guitar Tuner for Acoustic, Electric and Bass. Yousician | Learn Guitar, Piano, Ukulele With The Songs you Love. URL: https://yousician.com/guitartuna (date of access: 19.06.2024).

22. Guitar tuner | fender. Fender Guitars | Electric, Acoustic & Bass Guitars, Amps, Pro Audio. URL: https://www.fender.com/online-guitar-tuner (date of access: 19.06.2024).

23. Ітеративна модель (iterative model) - QALight. QALight. URL: https://qalight.ua/baza-znaniy/iterativna-model-iterative-model/ (дата звернення: 19.06.2024).

24. JetBrains. PyCharm: the Python IDE for Professional Developers by JetBrains. JetBrains. URL: https://www.jetbrains.com/pycharm.

ДОДАТКИ

Додаток А

ТЕХНІЧНЕ ЗАВДАННЯ

ВСТУП

Найменування програмного продукту – застосунок для налаштування музичних інструментів «Chromatuna».

Призначення застосування – застосування програми призначене для музикантів і любителів музики, які прагнуть точно налаштувати свої музичні інструменти завдяки музичному тюнеру.

ПРИЗНАЧЕННЯ І ПІДСТАВИ ДЛЯ РОЗРОБКИ

Призначення цієї роботи полягає у створенні програмного забезпечення для тюнера музичних інструментів, яке буде ґрунтуватися на методах спектрального аналізу. Точне налаштування музичних інструментів є критично важливим для досягнення високої якості музичного виконання та запису. Відсутність правильно налаштованого інструмента може призвести до негармонійного звучання та зниження якості музичного продукту. Розробка такого тюнера дозволить музикантам, звукоінженерам та викладачам музичних шкіл ефективно налаштовувати інструменти, що підвищить якість музичних виступів та навчальних занять.

ВИМОГИ ДО ПРОГРАМИ

Вимоги до функціональних характеристик:

- програма повинна забезпечувати налаштування музичних інструментів з високою точністю;
- програма повинна підтримувати різні типи інструментів (гітара, бас-гітара, скрипка тощо);
- надання користувачеві візуальних і звукових підказок для точного налаштування;

Вимоги до надійності:

- програма повинна забезпечувати стабільну роботу без збоїв;
- час відновлення після збою не повинен перевищувати 5 хвилин;
- програма повинна мати функцію автоматичного збереження налаштувань користувача;

Умови експлуатації:

- програма повинна працювати на операційних системах Windows, а також може бути портовано на платформи Linux, macOS;
- програма повинна бути легкою у використанні, з інтуїтивно зрозумілим інтерфейсом;

Вимоги до складу і параметрів технічних засобів:

- мінімальні вимоги до процесора: Intel Core i3 або еквівалентний;
- мінімальні вимоги до оперативної пам'яті: 4 ГБ;
- мінімальні вимоги до вільного місця на диску: 50 МБ;

СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

№	Етап	Опис
1	Аудит функціоналу, вимог	Визначення вимог до роботи тюнера. Складання списку необхідних модулів та бібліотек. Аналіз функціоналу, складання списку поліпшень.
2	Проектування	Розробка алгоритмів спектрального аналізу для тюнера. Проектування архітектури програмного забезпечення.
3	Розробка	Написання програмного коду на мові Python. Реалізація основних функцій тюнера, включаючи обробку звукових сигналів, застосування FFT та HPS.
4	Тестування	Перевірка роботи тюнера на різних музичних інструментах. Створення сценаріїв тестування для оцінки точності та ефективності програмного забезпечення.

5	Оптимізація	Поліпшення алгоритмів для забезпечення швидкого та точного аналізу в реальному часі. Оптимізація коду та виправлення помилок.

Додаток Б

Інструкція користувача

Для початку роботи з програмою «Chromatuna» необхідно отримати її вихідний код або встановлений дистрибутив. Це можна отримати, перейшовши на сторінку репозиторію на GitHub. Необхідно зробити його клонування, чи, якщо доступний реліз, викачати його у секції Releases, вибравши вашу необхідну операцій систему.

Налаштування мікрофону:

Для коректної роботи програми «Chromatuna» необхідно, щоб мікрофон був увімкнений та налаштований належним чином. Виконайте наступні дії:

- переконайтеся, що мікрофон підключений до вашого комп'ютера та увімкнений;
- у налаштуваннях операційної системи перевірте, чи програма «Chromatuna» дійсно має доступ до використання мікрофону;
- у разі необхідності надайте програмі відповідні дозволи на доступ до мікрофону.

Запуск програми:

Після отримання вихідного коду або дистрибутиву програми та налаштування мікрофону, ви можете запустити програму «Chromatuna». У вас з'явиться вікно з інтерфейсом користувача. Дотримуйтесь інструкцій на екрані та використовуйте різні функції програми для налаштування ваших музичних інструментів за допомогою перетворення Фур'є.

Пам'ятайте, що для коректної роботи програми «Chromatuna» необхідно, щоб мікрофон був увімкнений та мав доступ до нього. Якщо виникнуть будь-які проблеми або питання під час використання програми, зверніться до документації або зверніться за підтримкою до розробників, контакти вказані у репозиторії продукту.

Додаток В

```
# General settings
SAMPLE_FREQ = 48000  # sample frequency in Hz
WINDOW_SIZE = 44000  # window size of the DFT in samples
WINDOW_STEP = 12000  # step size of window
NUM_HPS = 5  # max number of harmonic product spectrums
POWER_THRESH = 1e-5  # tuning is activated if the signal power exceeds this threshold
CONCERT_PITCH = 440  # base frequency of the a4 note - 440Hz
WHITE_NOISE_THRESH = 0.2  # everything under WHITE_NOISE_THRESH*avg_energy_per_freq is cut off
WINDOW_T_LEN = WINDOW_SIZE / SAMPLE_FREQ  # length of the window in seconds
SAMPLE_T_LENGTH = 1 / SAMPLE_FREQ  # length between two samples in seconds
DELTA_FREQ = SAMPLE_FREQ / WINDOW_SIZE  # frequency step width of the interpolated DFT
OCTAVE_BANDS = [50, 100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600] # octave bands for the frequency calculation
ALL_NOTES = ["A", "A#", "B", "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#"]  # there are 12 notes in an octave

# core/chromatuna_engine.py

import numpy as np
import sounddevice as sd
import scipy.fftpack
import time
import copy
```

```
# importing variables from __init__.py
from core import SAMPLE_FREQ, WINDOW_SIZE, NUM_HPS, CONCERT_PITCH, WHITE_NOISE_THRESH, \
    DELTA_FREQ, OCTAVE_BANDS, ALL_NOTES, POWER_THRESH


class TunerEngine:

    def __init__(self, sample_freq, window_size, window_step, num_hps, power_thresh, white_noise_thresh, app):
        self.sample_freq = sample_freq
        self.window_size = window_size
        self.window_step = window_step
        self.num_hps = num_hps
        self.power_thresh = power_thresh
        self.white_noise_thresh = white_noise_thresh
        self.window_samples = [0 for _ in range(window_size)]
        self.note_buffer = ["1", "2"]
        self.running = False
        self.hann_window = np.hanning(self.window_size)
        self.app = app

    def find_closest_note(self, pitch):
        i = int(np.round(np.log2(pitch / CONCERT_PITCH) * 12))
        closest_note = ALL_NOTES[i % 12] + str(4 + (i + 9) // 12)
        closest_pitch = CONCERT_PITCH * 2 ** (i / 12)
        return closest_note, closest_pitch

    def audio_callback(self, indata, frames, time_info, status):
        if status:
```

```
            print(status)
            return
        if any(indata):
            self.window_samples = np.concatenate((self.window_samples, indata[:,
0]))
            self.window_samples = self.window_samples[len(indata[:, 0]):]

            signal_power = (np.linalg.norm(self.window_samples, ord=2) ** 2) /
len(self.window_samples)
            if signal_power < self.power_thresh:
                print(f"Signal is too weak, check your connection: {signal_power} .
Need at least {self.power_thresh}")
                return

            hann_samples = self.window_samples * self.hann_window
            # save fft data for charts visualisation.
            self.fft_data = np.abs(np.fft.rfft(hann_samples))
            self.fft_freqs = np.fft.rfftfreq(len(hann_samples), 1 / self.sample_freq)

            magnitude_spec =
abs(scipy.fftpack.fft(hann_samples)[:len(hann_samples) // 2])

            for i in range(int(16 / DELTA_FREQ)):
                magnitude_spec[i] = 0

            for j in range(len(OCTAVE_BANDS) - 1):
                ind_start = int(OCTAVE_BANDS[j] / DELTA_FREQ)
                ind_end = int(OCTAVE_BANDS[j + 1] / DELTA_FREQ)
                ind_end = ind_end if len(magnitude_spec) > ind_end else
len(magnitude_spec)
```

```
avg_energy_per_freq =
(np.linalg.norm(magnitude_spec[ind_start:ind_end], ord=2) ** 2) / (
    ind_end - ind_start)
avg_energy_per_freq = avg_energy_per_freq ** 0.5
for i in range(ind_start, ind_end):
    magnitude_spec[i] = magnitude_spec[i] if magnitude_spec[
        i] > WHITE_NOISE_THRESH *
avg_energy_per_freq else 0

mag_spec_ipol = np.interp(np.arange(0, len(magnitude_spec), 1 /
NUM_HPS),
                          np.arange(0, len(magnitude_spec)),
                          magnitude_spec)
mag_spec_ipol = mag_spec_ipol / np.linalg.norm(mag_spec_ipol,
ord=2)

hps_spec = copy.deepcopy(mag_spec_ipol)

for i in range(NUM_HPS):
    tmp_hps_spec =
np.multiply(hps_spec[:int(np.ceil(len(mag_spec_ipol) / (i + 1)))],
                                mag_spec_ipol[::(i + 1)])
    if not any(tmp_hps_spec):
        break
    hps_spec = tmp_hps_spec
self.fft_data = hps_spec
max_ind = np.argmax(hps_spec)
max_freq = max_ind * (SAMPLE_FREQ / WINDOW_SIZE) /
NUM_HPS
```

```
            closest_note, closest_pitch = self.find_closest_note(max_freq)
            max_freq = round(max_freq, 1)
            closest_pitch = round(closest_pitch, 1)

            self.note_buffer.insert(0, closest_note)
            self.note_buffer.pop()

            if self.note_buffer.count(self.note_buffer[0]) == len(self.note_buffer):
                print(f"Closest note: {closest_note} {max_freq}/{closest_pitch}:
{signal_power}")
            self.update_gui(closest_note, max_freq, closest_pitch, (max_freq -
closest_pitch))

        def update_gui(self, note, freq, pitch, diff):
            pass

        def start_stream(self):
            self.running = True
            print("Starting tuner...")
            try:
                with sd.InputStream(channels=1, callback=self.audio_callback,
blocksize=self.window_step,
                                    samplerate=self.sample_freq):
                    while self.running:
                        time.sleep(2)
            except Exception as exc:
                print(str(exc))

        def stop_stream(self):
            self.running = False
```

```
        self.update_gui('-', '0.00', '0.00', 0)
        print("Stopping tuner...")

# core/chroma_tuna.py

import threading
import customtkinter as ctk
import numpy as np
import matplotlib.pyplot as plt

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from core.chromatuna_engine import TunerEngine
from core.string_tuner_window import StringTunerWindow
from gui.tuner_window import TunerWindow



class ChromaTuna(TunerEngine):
    def __init__(self, tuner_engine):
        super().__init__(
            tuner_engine.sample_freq,
            tuner_engine.window_size,
            tuner_engine.window_step,
            tuner_engine.num_hps,
            tuner_engine.power_thresh,
            tuner_engine.white_noise_thresh,
            tuner_engine.app
        )
        self.top = None
        self.stream_thread = None
        self.closest_note_var = None
```

```
    self.freq_var = None
    self.pitch_var = None
    self.diff_var = None
    self.closest_note_label = None
    self.freq_label = None
    self.pitch_label = None
    self.diff_label = None
    self.fig_signal, self.ax_signal = plt.subplots(figsize=(4, 3))
    self.fig_fft, self.ax_fft = plt.subplots(figsize=(4, 3))
    self.signal_canvas = None
    self.fft_canvas = None
    self.fft_data = None
    self.fft_freqs = None

def start_tuning(self):
    if self.stream_thread is None or not self.stream_thread.is_alive():
        self.stream_thread = threading.Thread(target=self.start_stream)
        self.stream_thread.start()

def stop_tuning(self):
    self.stop_stream()

def update_gui(self, note, freq, pitch, diff=0):
    if self.running is False:
        return
    if self.closest_note_label:
        self.closest_note_label.configure(text=f"Closest Note: {note}")
    if self.freq_label:
        self.freq_label.configure(text=f"Freq: {freq}")
    if self.pitch_label:
```

```
    self.pitch_label.configure(text=f"Target Pitch: {pitch}")
if self.diff_label:
    if diff != 0:
        diff = int(diff * 10)
        if diff > 0:
            diff = f"+{diff}"
        self.diff_label.configure(text=f"Difference: {diff}")
    else:
        self.diff_label.configure(text='')

if self.signal_canvas is None:
    self.signal_canvas = FigureCanvasTkAgg(self.fig_signal, self.top)
    self.signal_canvas.draw()
    self.signal_canvas.get_tk_widget().pack(padx=5, pady=5)
self.ax_signal.clear()
self.ax_signal.plot(np.abs(self.window_samples))
self.ax_signal.set_title("Amplitude")
self.signal_canvas.draw()

if self.fft_canvas is None:
    self.fft_canvas = FigureCanvasTkAgg(self.fig_fft, self.top)
    self.fft_canvas.draw()
    self.fft_canvas.get_tk_widget().pack(padx=50, pady=50)

self.ax_fft.clear()
nyquist_freq = self.sample_freq / 2
max_freq = 700  # set 700 as the max frequency - crop rest of the data.

# crop the FFT data to the max frequency.
start_idx = 0
```

```
        end_idx = int(max_freq / (nyquist_freq / len(self.fft_freqs)))

        self.ax_fft.semilogy(self.fft_freqs[start_idx:end_idx],
self.fft_data[start_idx:end_idx])
        self.ax_fft.set_xlim(0, max_freq)
        self.ax_fft.set_title("FFT")
        self.ax_fft.set_xlabel("Frequency (Hz)")
        self.ax_fft.set_ylabel("Amplitude")
        self.fft_canvas.draw()

    def chromatic_tuning(self):
        print("Chromatic Tuner...")
        x, y = self.app.master.winfo_x(), self.app.master.winfo_y()
        self.top = TunerWindow(self.app.master, "Select Tuning", "+%d+%d" %
(x + 50, y + 50))
        # self.top.geometry("400x300")
        self.top.geometry("600x1400")
        self.closest_note_label = ctk.CTkLabel(self.top, text="Closest Note: -",
font=("Verdana", 14))
        self.freq_label = ctk.CTkLabel(self.top, text="Freq: -", font=("Verdana",
14))
        self.pitch_label = ctk.CTkLabel(self.top, text="Target Pitch: -",
font=("Verdana", 14))
        self.diff_label = ctk.CTkLabel(self.top, text="Difference: -",
font=("Verdana", 14))

        self.closest_note_label.pack()
        self.freq_label.pack()
        self.pitch_label.pack()
        self.diff_label.pack()
```

```
    start_button = ctk.CTkButton(
        self.top,
        text="Start",
        command=self.start_tuning,
        fg_color="#66B66E",
        hover_color="#6FC276",
        font=("Courier New", 14)
    )

    def stop_chromatic():
        self.stop_tuning()
        # self.top.destroy()

    stop_button = ctk.CTkButton(
        self.top,
        text="Stop",
        command=stop_chromatic,
        fg_color="#DE0A26",
        hover_color="#DC143C",
        font=("Verdana", 14)
    )
    start_button.pack(padx=5, pady=5)
    stop_button.pack(padx=5, pady=5)

def full_tuning(self):
    tuning_data = self.app.tunings[self.app.instrument][self.app.tuning][0]
    string_order = list(tuning_data['tuning'].keys())
    target_freq = list(tuning_data['tuning'].values())[0]
```

```
        self.top = StringTunerWindow(
            self.app.master,
            self.app,
            self.app.instrument,
            self.app.tuning,
            string_order,
            target_freq,
            self,
            self.app.tuner_engine
        )
        self.top.grab_set()

# core/sound_generator.py

import numpy as np



class SoundGenerator:
    def generate_sound(self, instrument='guitar', frequency=55, duration=2.5,
sample_rate=44100):
        t = np.linspace(0, duration, int(duration * sample_rate), False)
        num_samples = len(t)

        # define the amplitudes for the fundamental and harmonics
        fundamental_amp = 0.8

        if instrument == 'violin':
            harmonic_amps = [0.6, 0.4, 0.3, 0.2, 0.1, 0.1, 0.1, 0.05, 0.05]
            attack_duration = 1  # attack -  1 sec
            decay_duration = 0.2  # decay(falling) - 200 ms
```

```
    sustain_level = 0.8
    release_duration = 1  # release - 100ms
elif instrument == 'guitar':
    harmonic_amps = [0.8, 0.4, 0.2, 0.1, 0.05]
    attack_duration = 0.1
    decay_duration = 0.3
    sustain_level = 0.5
    release_duration = 0.4
elif instrument == 'ukulele':
    harmonic_amps = [0.7, 0.3, 0.1, 0.05]
    attack_duration = 0.05
    decay_duration = 0.15
    sustain_level = 0.6
    release_duration = 0.2
else:
    harmonic_amps = [0.6, 0.4, 0.3, 0.2, 0.1]
    attack_duration = 0.02  # 20 ms
    decay_duration = 1.2  # decay(falling) - 1200 ms
    sustain_level = 0.6
    release_duration = 0.7  # release - 700ms


# generate the fundamental sine wave
fundamental = fundamental_amp * np.sin(2 * np.pi * frequency * t)

# generate the harmonics
harmonics = np.zeros_like(fundamental)
for i, amp in enumerate(harmonic_amps, start=2):
    harmonic_freq = frequency * i
    harmonics += amp * np.sin(2 * np.pi * harmonic_freq * t)
```

```
        # combine the fundamental and harmonics
        sound_tone = fundamental + harmonics

        # apply a container to shape the tone
        attack_env = np.linspace(0, 1, int(attack_duration * sample_rate))
        decay_env = np.linspace(1, sustain_level, int(decay_duration * sample_rate))
        sustain_env = np.full(num_samples - len(attack_env) - len(decay_env) - int(release_duration * sample_rate),
                              sustain_level)
        release_env = np.linspace(sustain_level, 0, int(release_duration * sample_rate))

        envelope = np.concatenate((attack_env, decay_env, sustain_env, release_env))

        # ensure the envelope length matches the sound tone length
        envelope = np.resize(envelope, num_samples)
        sound_tone *= envelope
        return sound_tone
# gui/tuner_app.py

import json
import tkinter as tk
from tkinter import END
import customtkinter as ctk
import sounddevice as sd
import soundfile as sf
```

```
from core.sound_generator import SoundGenerator
from core.chromatuna_engine import TunerEngine
from gui.tuner_window import TunerWindow
from core.chroma_tuna import ChromaTuna
from core import SAMPLE_FREQ, WINDOW_SIZE, WINDOW_STEP, NUM_HPS, POWER_THRESH, WHITE_NOISE_THRESH


class TunerApp:
    def __init__(self, master):
        self.tunings = None
        self.tuning = None
        self.instrument = None
        self.master = master
        self.tuner_engine = TunerEngine(
            SAMPLE_FREQ,
            WINDOW_SIZE,
            WINDOW_STEP,
            NUM_HPS,
            POWER_THRESH,
            WHITE_NOISE_THRESH,
            self
        )
        self.chromatic_tuner = ChromaTuna(self.tuner_engine)
        self.sound_generator = SoundGenerator()
        self.default_configuration()
        self.setup_gui()

    def default_configuration(self):
        # get instrument & its tunings from json file
```

```
        with open("assets/tunings.json", "r") as file:
            self.tunings = json.load(file)
        self.instrument = 'guitar'
        self.tuning = 'standard'

    def setup_gui(self):
        self.master.title("Chromatuna")
        self.master.geometry("550x500")
        self.master.resizable(False, False)
        self.master.iconbitmap("assets/chromatuna.ico")
        # self.master.wm_attributes('-toolwindow', 'True')

        instrument_list = tk.Listbox(
            self.master,
            font=("Cascadia Mono", 12),
            bg="#66B66E",
            selectmode=tk.SINGLE
        )

        instrument_list.pack(expand=False, padx=5, pady=5)
        instrument_list.bind('<<ListboxSelect>>', self.instrument_selected)

        for name in self.tunings:
            instrument_list.insert(END, name.capitalize())
        instrument_list.selection_set(list(self.tunings).index(self.instrument))

        tuning_button = ctk.CTkButton(
            self.master,
            text="Choose Tuning",
            command=self.select_tuning,
```

```
    width=150,
    height=40,
    fg_color="transparent",
    hover_color="#FF5B00",
    border_width=1,
    border_color="#FF5B00",
    corner_radius=20,
    font=("Cascadia Mono", 11),
)
start_tuning = ctk.CTkButton(
    self.master,
    text="Start Tuning",
    command=self.chromatic_tuner.full_tuning,
    width=150,
    height=40,
    fg_color="transparent",
    hover_color="#66B66E",
    border_width=1,
    border_color="#66B66E",
    corner_radius=50,
    font=("Cascadia Mono", 14),
)
chromatic_button = ctk.CTkButton(
    self.master,
    text="Chromatic Only",
    command=self.chromatic_tuner.chromatic_tuning,
    width=150,
    height=40,
    fg_color="transparent",
    hover_color="#FFA500",
```

```
            border_width=1,
            corner_radius=50,
            border_color="#FFA500",
            font=("Cascadia Mono", 14),
        )

        instrument_list.pack(expand=True, padx=5, pady=5)
        tuning_button.pack(padx=10, pady=10)
        start_tuning.pack(padx=10, pady=10)
        chromatic_button.pack(padx=10, pady=10)

    def select_tuning(self):
        print("Selecting tuning...")
        x, y = self.master.winfo_x(), self.master.winfo_y()
        top = TunerWindow(self.master, "Selecting tuning", "+%d+%d" % (x +
135, y + 75))
        top.geometry("350x300")
        top.overrideredirect(True)
        top.config(bg="#242424", bd=5, relief=tk.RAISED, border=5)
        tunings = self.tunings[self.instrument]
        listbox = tk.Listbox(top, bg="#66B", font=("Cascadia Mono", 12),
selectmode=tk.SINGLE)
        if tunings:
            for tuning in tunings.keys():
                # insert tuning names into the listbox
                listbox.insert(END, tunings[tuning][0]['name'])
        listbox.selection_set(list(self.tunings[self.instrument]).index(self.tuning))

        def on_selection(event):
            # wrapper function to handle select and window close
```

```
            self.get_tuning_selection(event)
            top.destroy()

        listbox.bind("<<ListboxSelect>>", on_selection)
        listbox.pack(padx=5, pady=5)

    def get_tuning_selection(self, event):
        widget = event.widget
        index = int(widget.curselection()[0])
        self.tuning = list(self.tunings[self.instrument].keys())[index]  # update the
current tuning preset
        print(f"Tuning selected: {self.tuning}")

    def instrument_selected(self, event):
        widget = event.widget
        selections = widget.curselection()
        selected_items = [widget.get(i) for i in selections]
        # check if instrument is selected
        if selected_items is not None and len(selected_items) > 0:
            selected_item = selected_items.pop()
            self.instrument = selected_item.lower()
            self.tuning = 'standard'
            print("Selected instrument: " + selected_item)

    def get_tuning_frequencies(self):
        return list(self.tunings[self.instrument][self.tuning][0]['tuning'].values())

    def get_tuning_keys(self):
        list(self.tunings[self.instrument][self.tuning][0]['tuning'].keys())
```

АНОТАЦІЯ

Маркітан В.О – **Потокова цифрова обробка аудіосигналів для конфігурування музичних інструментів** – Рукопис.

Кваліфікаційна робота за спеціальністю 122 Комп'ютерні науки. - Волинський національний університет імені Лесі Українки, Луцьк. - 2024р.

Робота присвячена розробці програмного забезпечення для налаштування струнних музичних інструментів шляхом спектрального аналізу звуків з використанням перетворення Фур'є. Розглянуто актуальність створення таких програм для музикантів, звукоінженерів та в музичній освіті. Мета роботи: реалізація застосунку-тюнера для налаштування музичних інструментів на основі алгоритмів швидкого перетворення Фур'є та гармонічного спектрального аналізу мовою Python. Завдання включають огляд методів спектрального аналізу, розробку алгоритмів, реалізацію в IDE PyCharm, оцінку точності та ефективності програми.

Ключові слова: тюнер, перетворення Фур'є, FFT, HPS, спектральний аналіз, Python, PyCharm, музичні інструменти, налаштування.