МИНОБРНАУКИ РОССИИ
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

Факультет *компьютерных наук*

Кафедра *цифровых технологий*

Цифровая акустическая система оценки состояния лёгких

ВКР *Магистерская диссертация*

02.04.01 Математика и компьютерные науки

Компьютерное моделирование и искусственный интеллект

Допущено к защите в ГЭК

Зав. кафедрой _____ *С.Д. Кургалин, д. ф.-м. н., профессор* ___.___.2019

Обучающийся _____ *А.А. Родионов, 2 курс, д/о*

Руководитель _____ *Я.А. Туровский, к. мед. н., доцент*

Воронеж 2019

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет компьютерных наук

Кафедра цифровых технологий

УТВЕРЖДАЮ

заведующий кафедрой

_____ Кургалин С.Д.,

подпись *расшифровка подписи*

ЗАДАНИЕ

НА ВЫПОЛНЕНИЕ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

ОБУЧАЮЩЕГОСЯ РОДИОНОВА АЛЕКСАНДРА АЛЕКСАНДРОВИЧА

фамилия, имя, отчество

1. Тема работы «Цифровая акустическая система оценки состояния лёгких» , утверждена решением ученого совета факультета компьютерных наук от ___.___.20___
2. Направление подготовки 02.04.01 Математика и компьютерные науки.

шифр, наименование

3. Срок сдачи студентом законченной работы ___.___.20___
4. Календарный план:

№	Структура ВКР	Сроки выполнения	Примечание
1	Введение	03.09.2017	
2	Глава 1. Цель и задачи	10.11.2017	
3	Глава 2. Анализ предметной области	15.02.2018	
4	Глава 3. Модель распространения звука в легких	06.04.2018	
5	Глава 4. Аппаратная часть разработанной системы	20.11.2018	
6	Глава 5. Програмная часть разработанной системы	13.02.2019	
7	Заключение	11.05.2019	
8	Список использованных источников	03.06.2019	

Обучающийся _____ Родионов А.А.

подпись *расшифровка подписи*

Руководитель _____ Туровский Я.А.

подпись *расшифровка подписи*

РЕЦЕНЗИЯ

руководителя о ВКР магистерскую диссертацию Родионова Александра Алексанровиче, обучающегося по направлению подготовки 02.04.01 Математика и компьютерные науки, компьютерное моделирование и искусственный интеллект на факультете компьютерных наук Воронежского государственного университета на тему

«Цифровая акустическая система оценки состояния лёгких»

В работе описана разработка цифровой акустической системы оценки состояния лёгких.

Во второй главе представлен обзор существующих аналогов на рынке с разбором характеристик и цен, плюсов и минусов.

Во третьей главе рассмотрены теоретические аспекты: приводится описание математической модели распространения звука в легких на основе волнового уравнения. Математическая модель была реализована на компьютере для двухмерных и трехмерных случаев.

Четвертая глава описывает аппаратную часть разработанной системы, с использованием микроконтроллера Arduiono Due, датчиков давления BMP280, микрофона и динамиков. Усилитель для микрофона был создан самостаятельно в рамках данной работы на основе операционного усиителя MCP6022.

Пятая глава описывает програмное обеспечение. Были использованы современные технологии фреймворки: Numpy, Scipy, PyQt, Numba, Qt Signals. Приложение может быть запущено на современной операционной системе.

В магистерской диссертации продемонстрированы хорошие аналитические способности, навыки системного анализа проблемы, ссылки на современные научные источники, умение работать с современными программными фреймворками.

Работа удовлетворяет вем требованиям магистерской диссертации, проблема раскрыта всесторонне и заслуживает оценку ”отлично”.

Рецензент _____ Вахтин А.А.
подпись *расшифровка подписи*

кандидат физико-математических наук, доцент кафедры программирования и информационных технологий

___.___.20___

ОТЗЫВ

на ВКР магистерскую диссертацию Родионова Александра Александровиче, обучающегося по направлению подготовки 02.04.01 Математика и компьютерные науки, компьютерное моделирование и искусственный интеллект на факультете компьютерных наук Воронежского государственного университета на тему

«Цифровая акустическая система оценки состояния лёгких»

Магистерская диссертация посвящена решению актуальной проблемы: разработке цифровой акустической системы оценки состояния лёгких. Выбранная проблематика раскрыта в работе полно и всесторонне.

В ходе работы студент проявил высокую работоспособность, самостоятельное решение возникающих проблем и внимание к деталям. В магистерской диссертации продемонстрированы хорошие аналитические способности, умение систематизировать и анализировать собранную информацию, делать предложения, обобщения и выводы.

Студент проявил умения определять оптимальный для даной задачи набор технологий, быстро и качественно создавать компьютерные приложения используя современные програмные фреймворки. Было продемонстировано умение работать с математическими моделями, с литературой (в том числе англоязычной). Работа учавствовала в конкурсе Умник.

Магистерская диссертация удовлетворяет вем требованиям и заслуживает оценку ”отлично”.

Руководитель _____ Туровский Я.А. к. мед. н., доцент

подпись *расшифровка подписи*

___.___.20___

Реферат

Бакалаврская работа 50с., 32 рисунка, 25 источников

МЕДИЦИНСКАЯ ДИАГНОСТИКА, СТЕТОСКОП, АНАЛОГО ЦИФРОВЫЕ ПРЕОБРАЗОВАТЕЛИ, ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ, ВИЗУАЛИЗАЦИЯ ЗВУКОВОГО СИГНАЛА, БЫСТРОЕ ПРЕОБРАЗОВАНИЕ ФУРЬЕ

Целью данной работы является создание доступного и простого в производстве широкополосного цифрового стетоскопа. Стетоскоп должен быть способен регистрировать высокое качество звука. Он должен иметь высокую частоту дискретизации (больше 100кГц) и способен регистрировать сигнал в ультразвуковом диапозоне (больше 20кГц).

В процессе выполнения работы тестировались 3 различных аналого цифровых преобразователя. Были выявлены плюсы и минусы каждого из АЦП применительно к поставленной задаче. Также был создан усилитель сигнала для микрофона.

В результате работы был создан рабочий прототип цифрового широкополосного стетоскопа. Стетоскоп способен регистрировать сигнал с частотой дискретизации 660кГц и способен анализировать ультразвук до 100кГц. Также было написано програмное обеспечения для этого стетоскопа, позволяющего визуализировать сигнал, обрабатывать сигнал с помощью быстрого преобразования фурье (БПФ), а также визуализировать сигнал после обработки БПФ. В возможности програмного обеспечения входит обработка сигнала на удаленном высокопроизводительном сервере с помощью технологии Nvidia CUDA.

Содержание

Введение

Аускультация (выслушивание) звуков, исходящих от различных органов - одна из областей медицинской диагностики. Прибор для выслушивания звуков называется стетоскоп. Стетоскопы бывают акустические и электронные. Акустический стетоскоп передает звук от пациента непосредственно в ухо врачу. У электронных есть микрофон, который передает звук либо через динамики врачу, либо записывает для дальнейшего анализа.

Медицинская диагностика сердечно-сосудистых заболеваний является одной из важнейших отраслей современной медицины. По данным Всемирной Организации Здравоохранения (ВОЗ) сердечно-сосудистые заболевания - это основная причина смертей в мире. Максимальное количество смертей - это сердечно-сосудистые заболевания. Процентное сообношение этих смертей составляет примерно 31% или в абсолютных значениях 17.5 млн. человек. Сердечно-сосудистые заболевания нуждаются в раннем обнаружении, диагностики, консультировании и лечении.

Медицинская диагностика заболеваний легочной системы является не менее важной. Заболевания легочной системы, такие как респираторные инфекции дыхательных путей, хроническая обструктивная болезнь лёгких или ХОБЛ, рак легких и туберкулез входят в 10 болезней, от которых чаще всего умирают люди. (статистика ВОЗ на 2015 год).

Обычно люди, страдающие заболеваниями сердечно-сосудистой системы или легочной системы обращаются к участковому врачу в поликлинику (первичное звено). В первичном звене врач принимает решение о дальнейших действиях по лечению больного. Проблема заключается в том что в первичном звене РФ почти нет автоматизации. Из за этого врач первичного звена может принять неправильные решение из за несовершенства оборудования. Эту проблему можно решить, обеспечив врачей первичного звена оборудованием, позволяющем очень точно и автоматически определять состояние пациента и по-

могать принять однозначное решение о дальнейшем лечении пациента. Основным инструментом диагностики у врачей первичного звена РФ является стетоскоп, поэтому совершенствование стетоскопов должно быть основным трендом в развитии диагностики первичного звена. Совершенствование должно вестись не только в сторону улучшения обработки слышимого звука, но также и в сторону новых частотных диапазонов (ультразвука).

Проблемой существующих на рынке электронных стетоскопов является невысокое качество звука. Под невысоким качеством звука подразумевается низкая частота дискретизации получаемого на выходе сигнала и как следствие неспособность выдавать данные о высокочастотном диапазоне звука (в частности ультразвука). Эти стетоскопы имеют частоту, не охватывающую весь звуковой диапазон, слышимый человеком, следовательно они теряют массу информации, которая может быть полезна врачам для осуществления медицинской диагностики пациентов.

1 Цель и задачи

1.1 Цель

Целью данной работы является разработка информационной системы активной аускультации легких.

1.2 Задачи

1. создание аппаратной части устройства: микроконтроллер Arduino, микрофон. Обеспечить диапазон принятия сигналов в диапазоне до 40кГц
2. добавление динамиков к аппаратной части устройства
3. добавление датчиков давления к аппаратной части устройства
4. написание програмного обеспечения для микроконтроллера Arduino
5. програмное обеспечения для компьютера: синхронизация с аппаратным устройством по USB, прием данных
6. програмное обеспечения для компьютера: написание графического интерфейса
7. обеспечить обработку полученного сигнала с использованием спектральных методов оценивания
8. обеспечить возможность обработки сигнала при помощи технологии Nvidia CUDA
9. разработка компьютерной модели распространения звука в легких (двухмерный и трехмерный варианты)

2 Анализ предметной области

В настоящий момент на рынке есть много решений в области цифровых стетоскопов. Приведем несколько примеров и проанализируем плюсы и минусы.

2.1 3M™ Littmann® Electronic Stethoscope Model 3200

Рисунок 1 – 3M™ Littmann® Electronic Stethoscope Model 3200

Технические характеристики [1] стетоскопа представленного на рисунке 1:

— Диапазон принимаемого сигнала: 10 - 2000Гц

— Возможность записи 12ти 30-секундных звуковых дорожек

— Передача данных через Bluetooth

— Удаленная прослушка с помощью технологии 3M™ Littmann® TeleSteth™

— Удаление 85% (в среднем) окружающего шума

— 24х кратное усиление сигнала

— Цена - 22000 руб

— Вес - 185г

Это один из самых лучших стетоскопов, существующих на рынке. Стетоскоп обеспечивает высокое качества звука, но только в нижнем диапазоне (до 2кГц). Также к плюсам нужно отнести высокое качество подавления окружающего шума. Минусом является высокая цена.

2.2 CMS-VESD SPO2 PR

Рисунок 2 – CMS-VESD SPO2 PR

Технические характеристики [2] стетоскопа представленного на рисунке 2:

— Диапазон принимаемого сигнала для сердца: 20 - 230Гц

— Диапазон принимаемого сигнала для легких: 100 - 800Гц

— Диапазон принимаемого сигнала для сердца и легких: 20 - 800Гц

— Измерение пульса: 30 - 300 ударов в минуту

— Точность измерения пульса: +- 2 уд/мин

— Диапазон измерения насыщения кислородом (SpO2, процентное содержание оксигемоглобина в артериальной крови): 70% - 100%

— Дисплей, отображающий в реальном времени звуковую волну, пульс, SpO2

— Возможность загрузки данных на компьютер через USB

— ПО для анализа данных

— Цена - 5800руб

— Вес - 100г

Данный стетоскоп обладает невысокой разрешаюшей способностью (маленький диапазон принимаемого сигнала для сердца и легких) и невысокой ценой.

2.3 CMS-VE

Рисунок 3 – CMS-VE

Технические характеристики [3] стетоскопа представленного на рисунке 3:

— Диапазон принимаемого сигнала для сердца: 20 - 230Гц

— Диапазон принимаемого сигнала для легких: 100 - 800Гц

— Диапазон принимаемого сигнала для сердца и легких: 20 - 800Гц

— Измерение пульса: 30 - 300 ударов в минуту

— Точность измерения пульса: +- 2 уд/мин

— Диапазон измерения насыщения кислородом (SpO2, процентное содержание оксигемоглобина в артериальной крови): 70% - 100%

— Дисплей, отображающий в реальном времени звуковую волну, пульс, SpO2

— возможность загрузки данных на компьютер через USB

— ПО для анализа данных

— Цена - 5646руб

— вес - 100г

Преимуществом данного прибора является возможность записывать процентное содержание оксигемоглобина в артериальной крови (SpO2). Недостатками являются невысокая разрешающая способность 20 - 800Гц

2.4 CMS-M

Рисунок 4 – CMS-M

Технические характеристики [4] стетоскопа представленного на рисунке 4:

— Измерение пульса: 30 - 300 ударов в минуту

— Точность измерения пульса: +- 2 уд/мин

— Диапазон измерения насыщения кислородом (SpO2, процентное содержание оксигемоглобина в артериальной крови): 70% - 100%

— Дисплей, отображающий в реальном времени звуковую волну, пульс, SpO2

— возможность загрузки данных на компьютер через USB

— ПО для анализа данных

— Цена - 6500руб

Преимуществом данного прибора является возможность записывать процентное содержание оксигемоглобина в артериальной крови.) Недостатком является невысокая разрешающая способность 20 - 800Гц

2.5 Cardionics E-scope II

Рисунок 5 – Cardionics E-scope II

Технические характеристики [5] стетоскопа представленного на рисунке 5:

— Диапазон 45-900Hz для звуков сердца и 50-2000Hz

— 30-ти кратное усиление сигнала по сравнению с акустическим стетоскопом.

— вход для наушников

— возможность подключиться к компьютеру через USB-кабель для визуализации сигнала

— переключатель между звуками сердци и звуками легких

— Цена - 20900руб

Преимуществами данного прибора являются достаточно высокий, по сравнению с другими аналогами, диапазн принимаемого сигнала (20 - 2000Гц) и возможность подключится к к компьютеру для визуализации сигнала) Недостатком является высокая цена.

2.6 Thinklabs One

Рисунок 6 – Thinklabs One

Технические характеристики [6] стетоскопа представленного на рисунке 6:

— диапазон 10-1000Гц

— более чем 100 кратное усиление сигнала по сравнению с акустическим стетоскопом.

— система аудио-фильтров: несколько фильтров дают возможность контроллировать множество нюансов выходного сигнала

— шумоподавление

— возможность подключиться к iPhone, iPad, Android, планшету, компьютеру.

— управление устройством и запись сигнала через мобильное приложение

Преимуществом данного прибора является наличие качественных приложений под основные операционные системы (iPhone, iPad, Android, macOS, Windiws)

Недостатком является то что приложения с закрытым исходным кодом, что делает невозможным модификацию програмного обеспечения под свои нужды.

2.7 Eko Core

Рисунок 7 – Eko Core

Технические характеристики [7] стетоскопа представленного на рисунке 7:

— 40-кратное усиление сигнала

— Частота дискретезации 4000Гц

— Диапазон 20Гц – 2kГц

— Запись в .WAV формат

— Подключение через Bluetooth 4.0 low-energy

— Програмное обеспечение для iOS, Android, Windows

Преимуществом данного прибора является беспроводная передача данных через Bluetooth и наличие приложений под основные операционные системы (iPhone, iPad, Android, macOS, Windiws). Недостатком является то что приложения с закрытым исходным кодом, что делает невозможным модификацию програмного обеспечения под свои нужды.

2.8 Eko Duo

Рисунок 8 – Eko Core

Технические характеристики [8] стетоскопа представленного на рисунке 8:

— 60-кратное усиление сигнала

— подавление окружающего шума

— Частота дискретезации 4000Гц

— Диапазон 20Гц – 2kГц

— 4 аудиофильтра

— Запись в .WAV формат

— Подключение через Bluetooth 4.0 low-energy

— Програмное обеспечение для iOS, Android, Windows

— Возможность снимать ЭКГ

Преимуществом данного прибора является относительно высокая частота дискретизации (4000Гц), наличие аудиофильтров и качественное шумоподавление. Недостатком является програмное обеспечение с закрытым исходным кодом без возможности модифицировать его для своих задач.

Как видно из списка товаров, большинство стетоскопов не имеют возможность анализировать сигнал выше 1000 Гц. Те же которые имеют - дорого стоят. Многие дешевые стетоскопы вообще нацелены на измерение пульса, а не на подробный анализ звука.

3 Теоретическая часть исследования: компьютерная модель распространения звука в легких

3.1 Математическая модель

В рамках данной магистерской работы была разработана и протестирована компьютерная модель распространения звука в легких.

Модель строится на следующих предположениях

1. среда ведет себя как жидкость (80% тела является жидкостью)
2. среда неподвижна
3. полное отражение на границах тела

В качестве математической модели распространения звука было взято волновое уравнение для неоднородной среды, которое представлено на рисунке 9.

$$\frac{1}{c^2(\mathbf{x})}\frac{\partial^2 p(\mathbf{x},t)}{\partial t^2} = \rho(\mathbf{x})\nabla\cdot\left(\frac{1}{\rho(\mathbf{x})}\nabla p(\mathbf{x},t)\right),$$

Рисунок 9 – волновое уравнение для неоднородной среды

— px, t - давление воздуха (pressure)

— ρx - плотность ткани / мышц

— cx - скорость звука

— $x = x, y, z$ - точка пространства

Начальные условия для данной модели:

Предполагаем что в нулевой момент времени давление равно нулю во всех точках. В качестве альтернативы можно использовать атмосферное давление. Для всех x:

$$px, t = 0 = 0$$

производные давления для всех координат по времени в момент времени $t = 0$ также равны нулю:

$$\frac{\partial px, t}{\partial t}|t = 0 = 0$$

Также присутсвует условие затухания звука: (включая затухания на границах тела).

В качестве источника звука в течении первых нескольких секунд генерируется синусоидальная волна для давления:

$$pa, b, c, t = sin2\pi ft$$

где $x = a, b, c$ - положение источника звука. f - частота в Герцах.

Приведенное выше волновое уравнение для неоднородной среды решалось численным приближением при помощи метода конечных разностей. Производные заменялись на разность, давление в точке в последующий момент времени считалось как сумма давлений окружающих ее точек (с коэффициентами). В итоге формула для давления в точке i, j, k в момент времени $m + 1$ выглядит следующим образом:

$$\kappa_{ijk} = \frac{l}{h}c_{ijk}$$

$$l = \Delta t$$

$$h = \Delta x = \Delta y = \Delta z$$

$$P_{i,j,k}^{m+1} = (2 - 7.5\kappa_{i,j,k}^2)P_{i,j,k}^m - P_{i,j,k}^{m-1}$$
$$+ \frac{4\kappa_{i,j,k}^2}{3}\left[P_{i+1,j,k}^m + P_{i-1,j,k}^m + P_{i,j+1,k}^m + P_{i,j-1,k}^m + P_{i,j,k+1}^m + P_{i,j,k-1}^m\right]$$
$$- \frac{\kappa_{i,j,k}^2}{12}\left[P_{i+2,j,k}^m + P_{i-2,j,k}^m + P_{i,j+2,k}^m + P_{i,j-2,k}^m + P_{i,j,k+2}^m + P_{i,j,k-2}^m\right]$$
$$- \frac{\kappa_{i,j,k}^2}{3\rho_{i,j,k}}\left[(P_{i+1,j,k}^m - P_{i-1,j,k}^m) - (P_{i+2,j,k}^m + P_{i-2,j,k}^m)/8\right](\rho_{i+1,j,k} - \rho_{i-1,j,k})$$
$$- \frac{\kappa_{i,j,k}^2}{3\rho_{i,j,k}}\left[(P_{i,j+1,k}^m - P_{i,j-1,k}^m) - (P_{i,j+2,k}^m + P_{i,j-2,k}^m)/8\right](\rho_{i,j+1,k} - \rho_{i,j-1,k})$$
$$- \frac{\kappa_{i,j,k}^2}{3\rho_{i,j,k}}\left[(P_{i,j,k+1}^m - P_{i,j,k-1}^m) - (P_{i,j,k+2}^m + P_{i,j,k-2}^m)/8\right](\rho_{i,j,k+1} - \rho_{i,j,k-1})$$

Рисунок 10 – численное приближ. решение волнового уравнения для неоднородной среды

3.2 Human Visible Project Dataset

Особенностью данной модели является то, что она основана на реальных данных. В модели использовался датасет снимков компьютерной томографии Human Visible Project Dataset. Данный датасет представляет собой 1,871 изображений. Каждое изображение это снимок компьютерной томографии. Снимки делались с инервалом 1 милиметр друг от друга вдоль вертикали туловища. Примеры снимков представлены на рисунке 11.

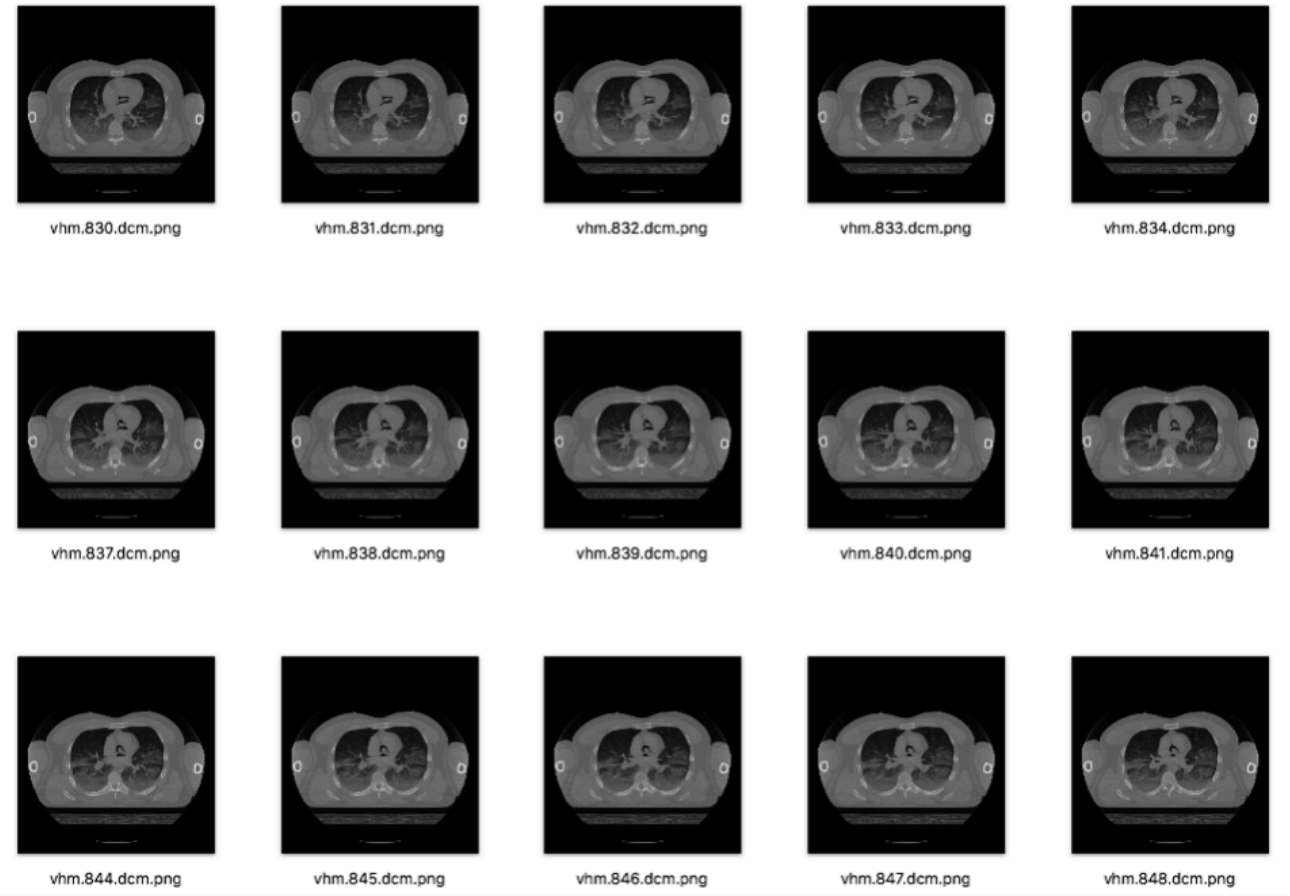

Рисунок 11 – датасет снимков компьютерной томографии Human Visible Project

Из данного датасета были выбраны только те снимки, которые отвечают за область легких.

Компьютерные снимки содержат информацию о яркости а для модели нужна плотность. Чтобы перевести значения яркости снимков в плотность использовалась таблица 1.

3.3 Компьютерная программа на основе математической модели

Компьютерная программа представляет собой кросплатформенное Desktop - приложение, написанное на языке python с использованием графического фреймворка PyQt. Для работы с многомерными массивами использовалась библиотека Numpy.

Компьютерная модель была реализована как для двухмерного так и для трехмерного случая. В двухмерном случае берется 1 снимок из датасета. В трехмерном случае все снимки легких были ”склеены”в один трехмерный массив, ко-

Таблица 1 – Перевод яркости снимка в плотность

Яркость CT	Плотность
0	0.000
20	0.001
254	0.292
444	0.438
944	0.895
957	0.945
986	0.980
1024	1.000
1139	1.116
1211	1.142
1504	1.285
1884	1.473
2307	1.707
4000	2.213
15160	4.510
43410	7.850

торый был сохранен в бинарный файл с расширением `.npy`. Это стандартный для numpy способ сериализации данных. Данный формат очень быстро загружается в память и превращается в массив `numpy.ndarray`.

С точки зрения программирования модель представляет собой 2 класса: один отвечает за графический интерфейс пользователя, другой - собственно за модель.

Класс модели хранит состояния следующих переменных в текущий момент времени а также 2 шага назад:

— состояние плотности для всех точек

— состояние давления для всех точек

— состояние скорости звука для всех точек

Также в данном классе можно задать частоту, местоположение и длительность начального звука который будет распространяться по легким. Инициа-

лизация переменных начальными значениями или параметрами по умолчанию происходит в методе `__init__`. Метод `step` расчитывает состояние системы на один временной шаг вперед. Метод `update_P` обновляет значение плотности на основе двух предыдущих значений плотности.

Графический интерфейс представляет собой окно с несколькоми графиками и элементами управления. Основные три графика, находящиеся по центру окна - это срезы легких в трех проекциях: X, Y, Z. На эти срезы накладывается график отвечающий за плотность звука.

Над графиками располагаются 3 слайдера, позволяющие перемещать срезы в проекциях. Слайдеры обозначены цветами, соответсвующие цвета присутсвуют на срезах и показывают их местоположение.

Ниже присутсвуют 3 графика, позволяющие посмотреть давление в трех произвольных точках за последние 100 временных шагов. Присутсвует кнопка, позволяющая сделать переход модели на 1 временной шаг вперед. Также есть возможность задать количество шагов и запустить модель.

Присутсвуют элементы управления основными параметрами модели: частотой генерируемой звуковой волны, положение источника звука, временной шаг в секундах. Есть кнопка которая сбрасывает параметры на значения по умолчанию. Трехмерный вариант модели можно увидеть на рисунке 12.

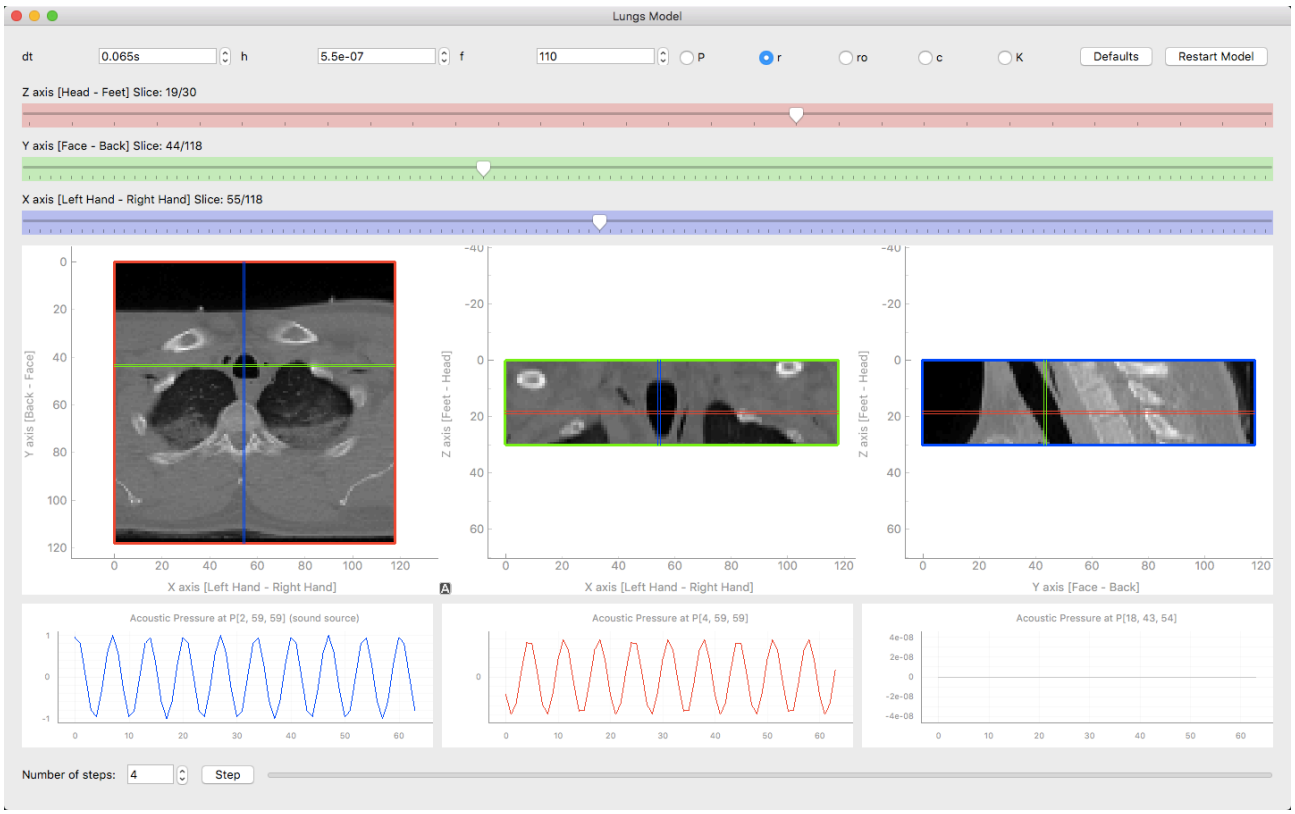

Рисунок 12 – Трехмерный вариант компьютерной модели распространения звука в легких

Также был реализован двухмерный вариант модели. Двухмерный вариант позволяет более наглядно наблюдать распространение звука. Двухмерный вариант модели можно увидеть на рисунке 13.

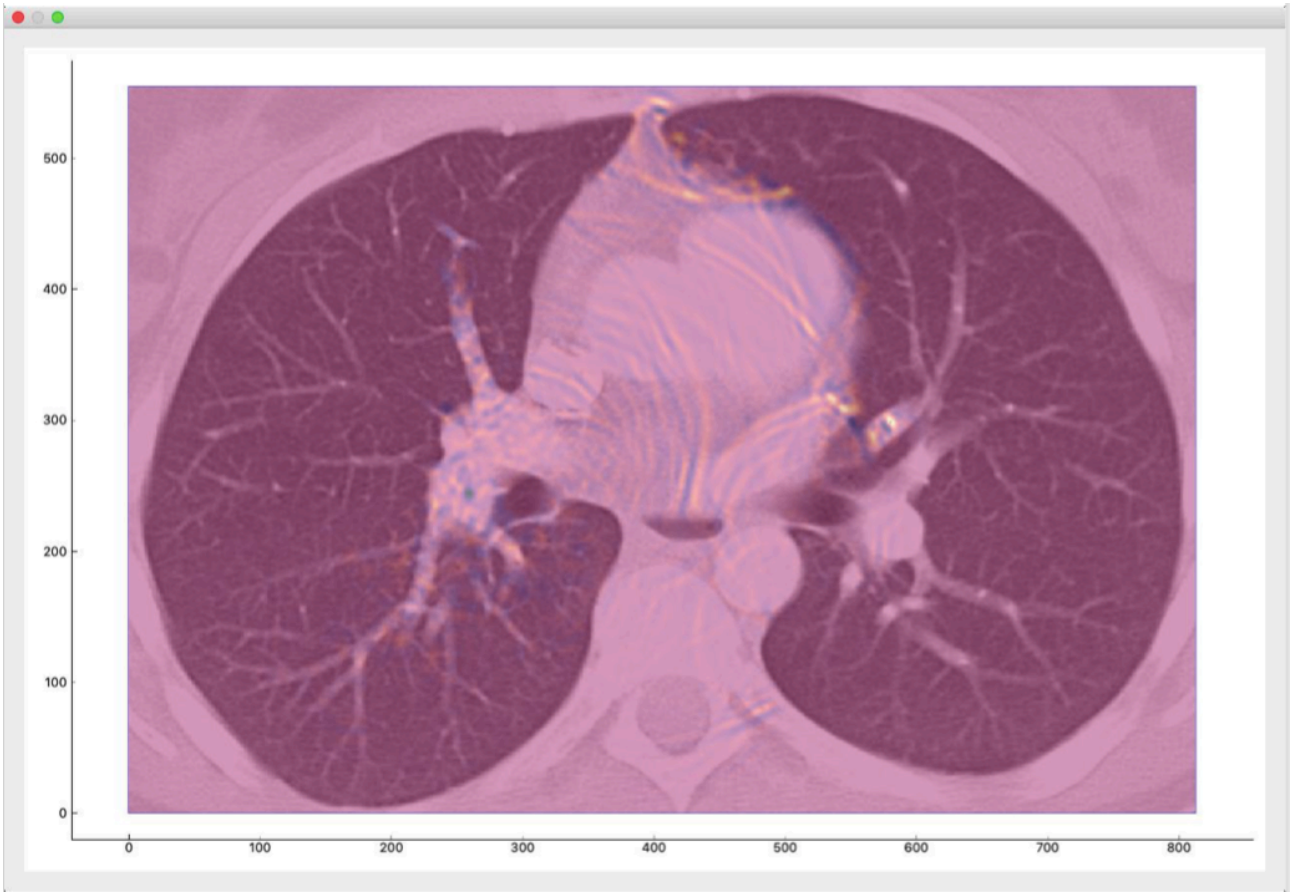

Рисунок 13 – Двухмерный вариант компьютерной модели распространения звука в легких

Более подробно результат работы модели можно посмотреть на видео [17]

Полный код компьютерной модели легких можно посмотреть в приложениях 1 и 2:

— Приложение 1 - класс LungsModel

— Приложение 1 - класс AppGUI

4 Описание аппаратной части разработанной системы

Разаботанная аппаратная часть должна отвечать следующим условиям. Прием сигнала стетоскопом осуществляется через микрофон. Далее, сигнал с микрофона должен подаваться на усилитель, усиливающий сигнал с микрофона до значений, в которых работает аналого цифровой преобразователь. Аналого цифровой преобразователь принимает сигнал и подключается к компьютеру через USB порт для передачи данных. Врач должен иметь возможность прослушивать пациента, видеть визуализацию сигнала на компьютере. Также врач должен иметь возможность увидеть спектр сигнала после преобразования Фурье.

С аппаратной точки зрения, устройство представляет собой микроконтроллер Arduino Due с подключенными к нему:

— микрофоном

— усилитель

— 2 датчика bmp280 (измерение давления чтобы детектировать вдохи и выдохи)

— USB динамики

4.1 Описание микрофона

В качестве микрофона был выбран SWEN MK-200. Микрофон был вынут из стандартного корпуса, чтобы лучше соединиться с трубкой, ведущей к мембране. Технические характеристики микрофона представлены в таблице 2.

Наилучшая чувствительность данного микрофона достигается в диапазоне частот от 50 до 16000 Гц. Тем не менее, микрофон с ослаблением принимает сигнал вплоть до 40кГц. Поэтому данный микрофон подходит для данного проекта.

Таблица 2 – Технические характеристики SWEN MK-200

Чувствительность, дБ	-60 ± 3
Диапазон частот, Гц	50 – 16 000
Размер микрофонного модуля, мм	9×7
Тип разъема	мини-джек Ø 3,5 мм (3 pin)
Длина кабеля, м	1,8
Вес, г	63

Для подавления шумов и лучшей передачи звука от сердца, легких и других органов к микрофону присоединяются мембрана и соединительная трубка от аналогового стетоскопа.

4.2 Описание усилителя

Усилитель для микрофона был создан самостоятельно в рамках данной работы на базе операционного усилителя **MCP6022** от производителя Microchip. Это усилитель типа Rail-to-Rail SO-8. Выбраная схема усилителя представлена на рисунке 14.

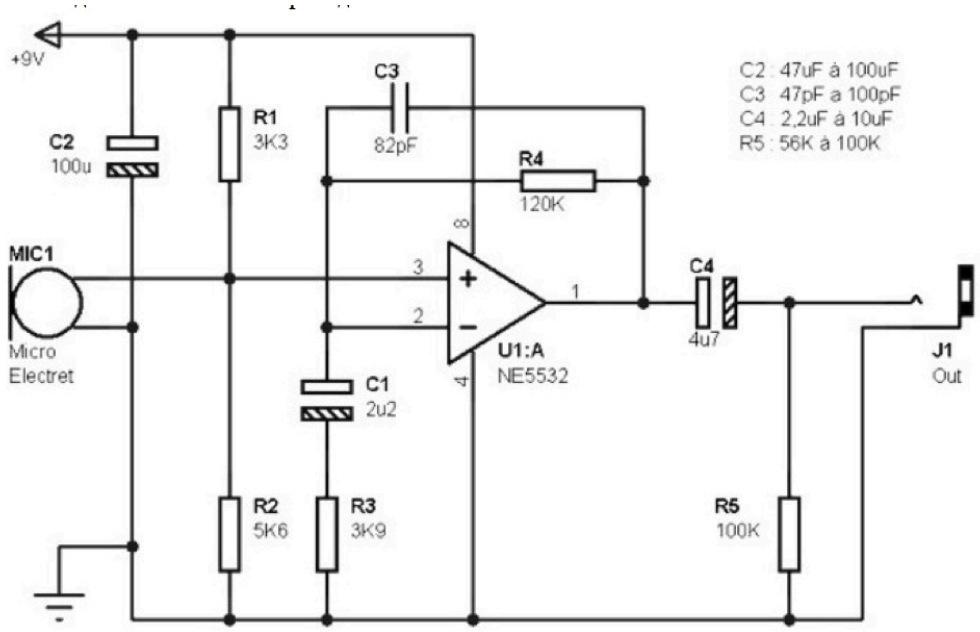

Рисунок 14 – Схема усилителя сигнала

4.3 Микроконтроллер Arduino Due

Самым оптимальным вариантом микроконтроллера для данного проекта оказался Arduino Due (изображение микроконтроллера представлено на рисунке 15). Полные технические характеристики микроконтроллера представлены в таблице 3. Микроконтроллер сочетает в себе как простоту в использовании так и возможность оцифровывать сигнал высокого качества. Максимальная частота дискретизации АЦП Arduino Due составляет 1МГЦ. В ходе данной работы удалось достичь максимума в 700кГц. Обычно среднее значение частоты дискретизации составляло 670кГц. Максимальное значение частоты дискретизации зависит также от производительности компьютера. Данное устройство тестировалось на MacBook Air 2014.

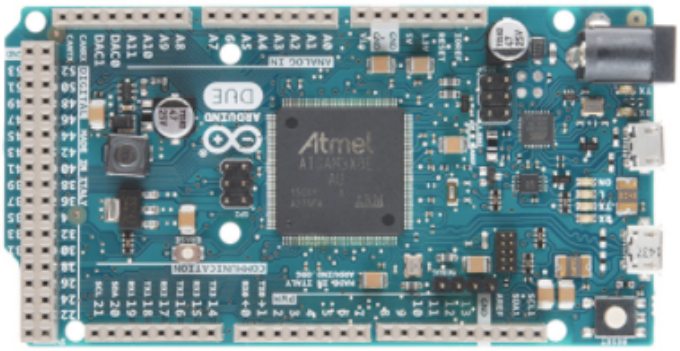

Рисунок 15 – Микроконтроллер Arduino Due

Таблица 3 – Технические характеристики Arduino Due

Число аналоговых входов	12
Максимальная частота дискретизации	1МГц
Объем буффера памяти	512 KB
Разрядность	12бит (4096 значений)
Рабочее напряжение	3.3V
Диапазоны входного напряжения	7-12V
Защита по входному напряжению	6-16V
Интерфейс	USB
Микроконтроллер	AT91SAM3X8E
Масса	36г

5 Описание программной части разработанной системы

В рамках данной работы было написано програмное обеспечение для работы с аппаратным обеспечением описаным выше. Программа обеспечение представляет собой кросплатформенное Desktop приложение, написанное на языке python с использованием графического фреймворка PyQt.

Чтобы запустить программу, предварительно нужно скачать и установить python для выбранной операционной системы. Установочные файлы и инструкции по установке есть на официальном сайте python.org. Также необходимо установить следующих набор библиотек с помощью комманд:

```
pip install PyQt5
pip install numpy
pip install pyqtgraph
pip install pyserial
```

Даллее в коммандной строке необходимо перейти в директорию, где находится файл main.py. Теперь нужно подключить устройство к компьютеру через USB порт и запустить программу с помощью команды:

```
python main.py
```

Программное обеспечение состоит из 2х основных компонент:

— программа для микроконтроллера Arduino Due

— программа для компьютера

Перед тем, как рассматривать эти модули отдельно, рассмотрим файл `main.py`. С начала в нем производится импорт следующих модулей:

создание тредов (модуль стандартной библиотеки)

```
import threading
```

общение между тредами с использованием сигналов (модуль стандартной библиотеки)

```
import signal
```

используется для парсинга аргументов коммандной строки

```
import sys
```

фреймворк для графического интерфейса

```
import PyQt5.QtWidgets
```

модуль отвечающий за графический интерфейс программы

```
import gui
```

сбор сигнала от устройства через usb(серийный порт)

```
import serial_port
```

чтобы запустить приложение с испольованием `PyQt5`, нужно создать объект `QApplication` а также объект `gui.GUI` - основной класс в котором описывается графический интерфейс:

```
app = PyQt5.QtWidgets.QApplication(sys.argv)
gui = gui.GUI()
```

Чтобы приложение можно было правильно закрыть по нажатию Ctrl-C, регистрируется функцию которая будет обрабатываеть сигнал `SIGINT` (Keyboard Interrupt, прерывание с клавиатуры):

```
def ctrl_c_handler(sig, frame):
    serial_port.stop_flag = True
    app.quit()

signal.signal(signal.SIGINT, ctrl_c_handler)
```

при прерывании с клавиатуры переменная `stop_flag` принимает значение `True` что говорит модулю `serial_port` что следует прекратить сбор данных с usb и закрыть соединение. Также производится выход из графического приложения.

модули `serial_port` и `gui` работают в разных потоках. Это нужно для того, чтобы сбор сигнала производился без пауз, а также чтобы графический интерфейс программы не зависал и быстро реагировал на действия пользователя.

Графический интерфейс будет испольняться в основном треде программы. Для моудуля `serial_port` создается дополнительные тред:

```
serial_reader = threading.Thread(
    target=serial_port.run,
    args=(gui.bmp_signal, gui.mic_signal)
)
```

`target` - функция, которую будет исполнять тред, `args` - аргументы функции - 2 Qt-сигнала, о которых будет сказано далее.

Рассмотрим программу для микроконтроллера Arduino а затем рассмотрим более детально, что происходит в модулях `serial_port` и `gui`

5.1 Програмное обеспечение для микроконтроллера Arduino Due

Микроконтроллер Arduino Due делает следующие операции:

— собирает данные с микрофона

— издает звуковые импульсы через динамик

— собирает данные с двух датчиков давления bmp280

— передает эти данные через usb на компьютер

5.1.1. Сбор данных с микрофона

Микрофон подсоединяется к пину `A0` микроконтроллера. Напряжение с этого пина считывается и передается в АЦП/ADC преобразуясь в числа. Далее микроконтроллер считывает данные с АЦП и передает их через USB на компютер.

В начале программы инициализируется буффер в который будет производится запись полученных данных с АЦП:

```
volatile int bufn, obufn;
uint16_t buf[4][256];   // 4 buffers of 256 readings
```

АЦП инициализируется следующим кодом:

```
void init_adc() {
    pmc_enable_periph_clk(ID_ADC);
    adc_init(ADC, SystemCoreClock, ADC_FREQ_MAX, ADC_STARTUP_FAST);
    ADC -> ADC_MR |= 0x80; // free running

    ADC -> ADC_CHER = 0x80;

    NVIC_EnableIRQ(ADC_IRQn);
    ADC -> ADC_IDR = ~(1 << 27);
    ADC -> ADC_IER = 1 << 27;
    ADC -> ADC_RPR = (uint32_t)buf[0]; // DMA buffer
    ADC -> ADC_RCR = 256;
```

```
    ADC -> ADC_RNPR = (uint32_t)buf[1]; // next DMA buffer
    ADC -> ADC_RNCR = 256;
    bufn = obufn = 1;
    ADC -> ADC_PTCR = 1;
    ADC -> ADC_CR = 2;
}
```

Код выше переводит АЦП в режим Free Running Mode. Это самый быстрый режим работы данного АЦП. Его особенность заключается в том, что нет необходимости вручную собирать данные через определенное количество времени. Программа пользователя запускает только первый сбор данных, а дальше АЦП сам автоматически начинает преобразование сразу как только закончилось предыдущее. Таким образом не возникает простоев АЦП и не теряются данные.

Стоит отметить включение режима прямого доступа к памяти микроконтроллера AT91S-AM3X8E (DMA: Direct Memory Access). Этот режим позволяет обращаться напрямую к памяти микроконтроллера, таким образом не отнимая процессорное время самого микроконтроллера. Процессор не прерывается на обработку запроса к памяти и постоянно занят сбором данных с АЦП. Это позволяет приблизится к максимальной частоте дискретизации.

АЦП записывает данные в буфферы DMA. Отуда они идут в переменную `buf`. Переменная `buf` это двухмерный массив размером (4, 256): запись производится циклически в эти 4 массива. Функция которая периодически сдвигает буфферы выглядит так:

```
void ADC_Handler() {    // move DMA pointers to next buffer
    int f = ADC -> ADC_ISR;
    if (f & (1 << 27)) {
        bufn = (bufn + 1) & 3;
        ADC -> ADC_RNPR = (uint32_t)buf[bufn];
        ADC -> ADC_RNCR = 256;
    }
}
```

Эта функция вызывается в стандартной для Arduino функции `setup()`:

```
void setup() {

    // ...

    init_adc();
}
```

В стандартной для Arduino функции `loop()` происходит ожидание пока буфер заполнится. Буффер содержит значения типа данных `uint16_t`. Чтобы передать данные через USB при помощи функции `SerialUSB.write` данные преобразуются в байты. Также производится увеличение переменной `obufn`, означающее что нужно записывать в следующий буффер.

```
while (obufn == bufn); // wait for buffer to be full
uint8_t* buffer_bytes = (uint8_t *) buf[obufn];
obufn = (obufn + 1) & 3; // 0 1 2 3 0 1 2 3 0 1 2 3 ..., like % 3
```

5.1.2. Генерация звука

К микроконтроллеру подсоединяются динамики через аудиовыход. Сами динамики питаются от USB. Аудиовыход подключается к ЦАП `DAC1`.

В начале программы устанавливаются следующие переменные и константы:

установка пина к которому подключены динамики

```
#define BEEP_PIN DAC1
```

частота издаваемых импульсов в герцах

```
float freq = 1498; // Hz
```

длина звукового импульса в микросекундах

```
const uint32_t tone_duration          =   1000000/16;
```

длина паузы (тишины) между импульсами в микросекундах

```
const uint32_t short_silence_duration =   1000000/16;
```

переменные для хранения текущих времени старта импульса и времени старта паузы

```
uint32_t short_silence_start_t = 0;
uint32_t tone_start_t          = 0;
```

переменная-флаг: звучит импульс или нет

```
uint8_t is_tone_playing = 1;
```

амплитуда звукового сигнала

```
const float A = 490;                              // amplitude of sine
    signal


const float pi = 3.14159265;
```

время семплирования (1/частота дискретизации)

```
float _T = 50 / 1000000.0;                        // set default
   sampling time in microseconds
```

массив для звуковой волны которая будет записываться в ЦАП т.к. эта переменная будет обновляться через прерывания то она должна быть volatile

```
volatile float a[3];                              // filter register
   for generating tone
```

угловая частота

```
float omega = 2.0 * pi * freq;                    // angular frequency
    in radians/second
```

константы, необходимые для генерации синусоиды

```
float wTsq = _T * _T * omega * omega;             // omega * sampling
   frequency squared
float c1  = (8.0 - 2.0 * wTsq) / (4.0 + wTsq); // c1 = first filter
    coefficient
```

В функции `setup.py` строчки отвечающие за генерацию звука делают следующее:

Устанавливается разрешение ЦАП

```
analogWriteResolution(10);
```

устанавливаем пин ЦАП в режим выхода

```
pinMode(BEEP_PIN, OUTPUT);
```

устанавливаем пин светодиода в режим выхода светодиод будет загораться одновременно со звуковыми импульсами

```
pinMode(LED_BUILTIN, OUTPUT);
```

функции beep_handler отвечающей за генерацию импульсов назначается вызов по прерыванию (каждые 50 микросекунд)

```
Timer1.attachInterrupt(beep_handler).start(50);
```

В функции `beep_handler` происходит следующее.

Если переменная `is_tone_playing` равна 1 то:

проверяется сколько прошло времени с момента запуска импульса. Если прошло меньше чем длительность импульса то продолжается генерация звуковой волны и запись ее в ЦАП. Также светодиод продолжает гореть. Если время превысило длительность импульса то `is_tone_playing` становится равна нулю. Также обнуляется фаза звуковой волны. Также фиксируется время начала паузы.

Если переменная `is_tone_playing` равна 0 то:

проверяется не прошло ли время паузы. Если нет то записываем 0 в ЦАП. И светодиод находится в выключенном положении. Если прошло то меняем `is_tone_playing` на 1 и фиксируем время старта звукового импульса.

```
void beep_handler() {
    if (is_tone_playing) {
        if (micros() – tone_start_t < tone_duration) {
            // play tone samples (src: https://github.com/cmasenas/
               SineWaveDue)
            digitalWrite(LED_BUILTIN, HIGH);
            a[2] = c1 * a[1] – a[0];       // compute the sample
```

```
            a[0] =      a[1]       ;       // shift the registers
               in preparation for the next cycle
            a[1] =      a[2]       ;
            analogWrite(BEEP_PIN, a[2] + 500); // write to DAC
        }
        else {
            is_tone_playing = 0;

            // reset sine phase (without reset, phases of sine-
               tones are constantly shifting)
            a[0] = 0.0;
            a[1] = A * sin(omega * _T);
            a[2] = 0.0;

            short_silence_start_t = micros();
        }
    }
    else {
        if (micros() - short_silence_start_t <
           short_silence_duration) {
            digitalWrite(LED_BUILTIN, LOW);
            analogWrite(BEEP_PIN, 0);
        }
        else {
            is_tone_playing = 1;
            tone_start_t = micros();
        }
    }
}
```

Код, отвечающий за генерацию звуковой волны взят из исходного кода библиотеки SineWaveDue (https://github.com/cmasenas/SineWaveDue) (из метода `SineWa`

5.1.3. Датчики давления BMP 280

Для работы с датчиками давления BMP 280 используется библиотека Adafruit BMP280 [10]. В начале программы библиотека импортируется вместе с зависимостями:

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_BMP280.h>
```

Затем необходимо объявить пины к которым физически подключены датчики:

```
#define BMP_CS 8
#define BMP_MOSI 11 // sdi
#define BMP_MISO 12 // sdo
#define BMP_SCK 13

#define BMP_2_CS 7
#define BMP_2_MOSI 5 // sdi
#define BMP_2_MISO 6 // sdo
#define BMP_2_SCK 4
```

Затем создаются объекты класса `Adafruit_BMP280`

```
Adafruit_BMP280 bmp0(BMP_CS, BMP_MOSI, BMP_MISO,  BMP_SCK);
Adafruit_BMP280 bmp1(BMP_2_CS, BMP_2_MOSI, BMP_2_MISO,  BMP_2_SCK);
```

В основном цикле программы `loop()` данные с датчиков считываются и помещаются в переменную-буффер:

```
bmp_buffer[0] = bmp0.readPressure();
bmp_buffer[1] = bmp1.readPressure();
```

5.1.4. Упаковка данных в пакет и передача через usb

Для того чтобы отправить данные с устройства по USB на компьютер, их необходимо конвертировать в массив байт (процесс сериализации). В начале файла объявляется переменная `packet`:

```
const int header_length = 4;
// 2 bmp sensors, is_tone_playing, adc mic
const int payload_length = 4 + 4 + 1 + 512;
const int packet_length = header_length + payload_length;
uint8_t packet[packet_length];
```

Пакет данных начинается со следующего заголовка:

```
uint8_t header[header_length] = { 0xd2, 0x2, 0x96, 0x49 };
for (int i = 0; i < header_length; i++)
    packet[i] = header[i];
```

Затем следуют даные с датчиков давления, они предварительно конвертируются в байты:

```
uint8_t* bmp_buffer_bytes = (uint8_t *) bmp_buffer;
for (int i = 0; i < 8; i++)
    packet[4 + i] = bmp_buffer_bytes[i];
```

Затем следует переменная “издается ли звук на динамиках или нет”:

```
packet[12] = is_tone_playing;
```

Затем следуют даные с АЦП, они предварительно конвертируются в байты:

```
uint8_t* buffer_bytes = (uint8_t *) buf[obufn];
for (int i = 0; i < 512; i++)
    packet[13 + i] = buffer_bytes[i];
```

Структура пакета данных изображена на рисунке 16.

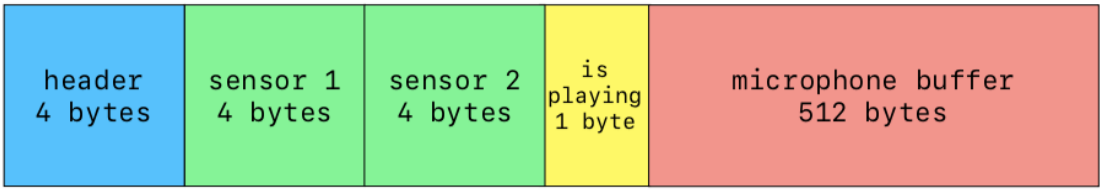

Рисунок 16 – структура пакета данных

В конце функции `loop()` пакет данных отправляется на компьютер по USB:

```
SerialUSB.write(packet, packet_length);
```

5.1.5. Установка биболиотек для Arduino

Также, данная программа для Arduino требует установки следующих библиотек.

Во первых, нужно установить `arduino-cli` - с помощью этого инструмента производится загрузка программ на Arduino. Установочные файлы под разные операционные системы можно найти по ссылке https://github.com/arduino/ardu cli во вкладке releases.

Установка драйверов для платы (arduino-cli/README.md)

С помощью `arduino-cli` нужно установить библиотеки:

```
arduino-cli lib install "DueTimer"
arduino-cli lib install "Adafruit BME280 Library"
```

Загрузка программы на ардуино производится с помощью скрипта `arduino.py`

```
python3 arduino.py _arduino
```

Схема основных элементов программного обеспечения для микроконтроллера Arduino представлена на рисунке 17.

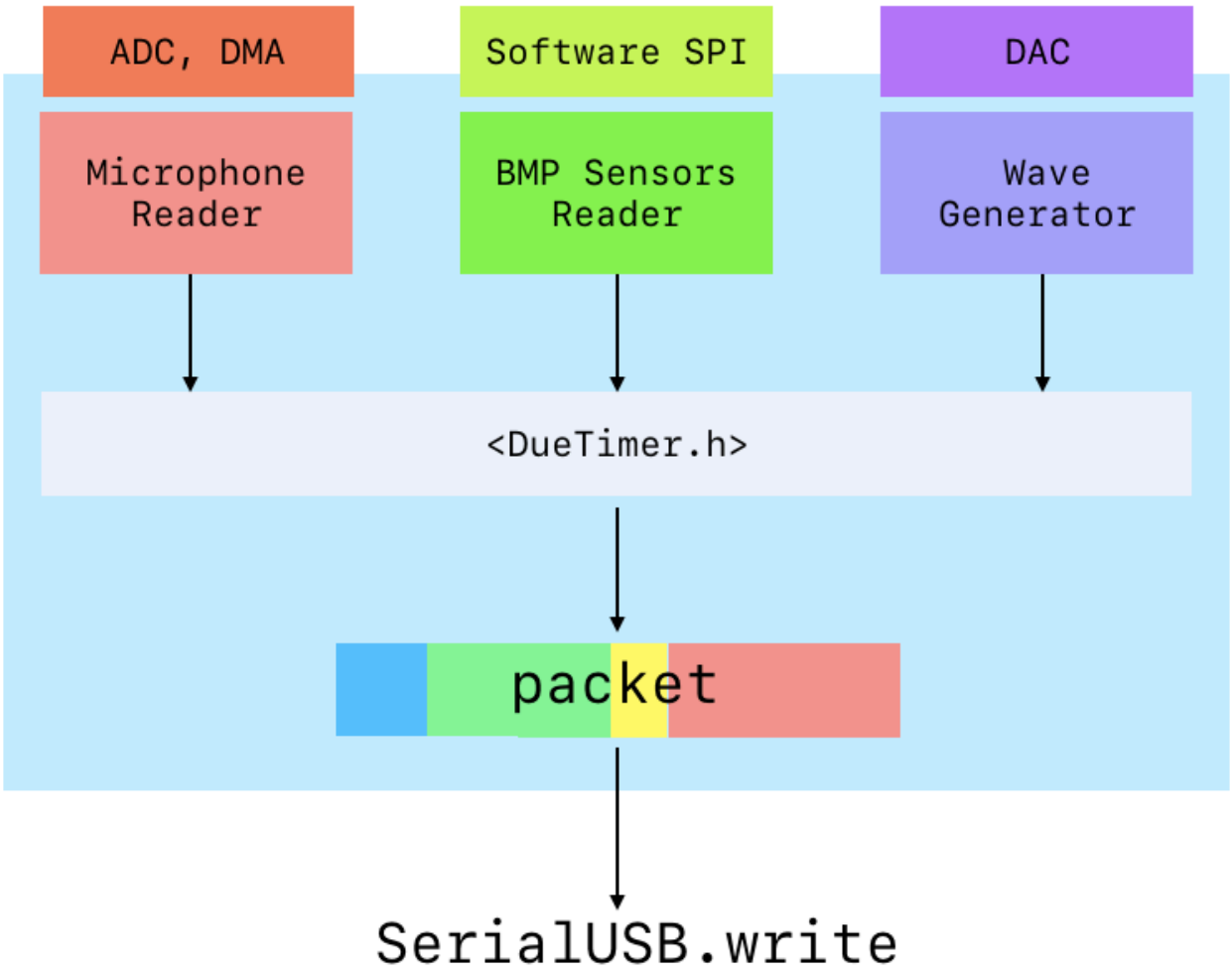

Рисунок 17 – Диаграмма классов ПО для Arduino Due

5.2 Програмное обеспечение для компьютера: сбор данных по USB

Подключение к ардуино по usb происходит с использованием файла `arduino.p`
Этот файл используется для многих проектов поэтому он лежит за пределами папки данного проекта. В python чтобы импортировать модуль, который лежит по произвольному пути, этот путь нужно добавтить в `sys.path`:

```
import sys; sys.path.append('/Users/tandav/Documents/spaces/arduino
   '); import arduino
port = arduino.find_device()
```

Функция `find_device` пытается 60 раз просканировать доступные usb-устройства с помощью функции `serial.tools.list_ports.comports()` если в описании какого либо из устройств содержаться следующие ключевые слова:

```
device_keywords = (
    'Arduino',
    'USB Serial Device',Устройство
    ' споследовательныминтерфейсом   USB',
)
```

то устройство опознается как Arduino и возвращается.

Если через 60 попыток Arduino не найдено, то выбрасывается исключение: `serial.SerialException('device not found, check the connect`

Далее в модуле `serial_port` происходит объявление следующих переменных и констант:

заголовок пакета с данными от Arduino должен быть такой же как в коде программы для Arduino

```
header = b'\xd2\x02\x96I'
```

длины различных частей пакета в байтах:

длина блока данных, отвечающего за один сенсор давления

```
bmp_pressure_length    =   4 # float32 number
```

длина блока данных, отвечающего за данные с микрофона

```
mic_length              = 512 # 256 uint16
mic_chunk_size          = mic_length // 2 # uint16 takes 2 bytes
```

длина блока данных, отвечающего за информацию о том, издают ли звук динамики

```
is_tone_playing_length =   1 # uint8 (used like bool)
```

сумарная длина пакета без заголовка

```
payload_length          =   2 * bmp_pressure_length +
   is_tone_playing_length + mic_length
```

сумарная длина пакета с заголовком

```
packet_length           = len(header) + payload_length
```

переменная-флаг, отвечающая за то, следует ли продолжать сбор данных (используется в момент закрытия приложения)

```
stop_flag = 0
```

число пакетов которые прошли без потери данных (используется для дебага)

```
n_good_packets = 0
```

Далее в модуле `serial_port` объявляются функции `read_packet`, `read_pa` `wait_header`. Они непосредственно отвечают за сбор данных через серийный порт.

5.2.1. чтение пакета в байтах

Функция `read_packet_bytes` читает из серийного порта количество байт равное длине пакета. Далее происходит проверка: полученные байты должны начинаться с заголовка. Если условие выполняется, то возвращаются полезные нагрузочные байты (payload) пакета (без заголовка). Если условие не выполняется то вызывается вспомогательная функция `wait_header` которая читает данные пока не дождется заголовка. После того как заголовок был считан, читается отавшаяся часть пакета и возвращается.

```
def read_packet_bytes():
    '''
    packet_length: packet length in bytes
    returns packet (as bytes) without header
    '''

    global n_good_packets

    packet = port.read(packet_length)

    if packet.startswith(header):
        n_good_packets += 1
        if n_good_packets % 5000 == 0:
            print(f'n_good_packets = {n_good_packets}')
        return packet[len(header):]
    else:
        print(f'wrong header {packet[:len(header)]} before:
           n_good_packets = {n_good_packets}')
        n_good_packets = 0
        # time.sleep(1)
        wait_header()
        return port.read(payload_length) # rest of packet
```

Ожидание заголовка в функции `wait_header` реализовано с помощью использования структуры данных `deque` из модуля стандартной библиотеки питона `collections`. `deque` - двухстороння очередь. Эту структуру можно использовать и как для очередей с порядком LIFO (Stack, используется только одна сторона `deque`), так и для очереди с порядком FIFO (используется 2 стороны `deque`). В данной функции `deque` используется как очередь с порядком `FIFO` с максимальным количеством элементов равным 4 (количество байт в заголовке).

Данные читаются с серийного порта по одному байту и помещаются в очередь справа (`append`). Старые байты автоматически вытесняются с другого

конца (слева). Данные читаются до тех пор, пока 4 байта, которые хранятся в очереди не будут равняться четырем байтам заголовка (с учетом порядка). Как только это условие выполняется, происходит выход из функции.

5.2.2. ожидание заголовка в случае потери пакета

```
def wait_header():
    deque = collections.deque(maxlen=len(header))

    while b''.join(deque) != header:
        deque.append(port.read())

    print('wait done', b''.join(deque), '==', header)
```

Функция `read_packet` использует функцию `read_packet_bytes`, чтобы получить байты пакета без заголовка. Затем происходит разделение байт на различные переменные с использованием функции библиотеки `numpy` `np.frombu` с кастингом байт в необходимые типы данных:

5.2.3. разбиение байтов пакета на части

```
def read_packet():
    packet = read_packet_bytes()

    # данные 1 датчикадавления
    bmp0 = np.frombuffer(packet[ :4], dtype=np.float32)[0]

    # данные 2 датчикадавления
    bmp1 = np.frombuffer(packet[4:8], dtype=np.float32)[0]

    # переменная, хранящаяинформациюотом   , издаютлизвукдинамики
    is_tone_playing = np.frombuffer(packet[8:9], dtype=np.uint8)[0]

    # данныесмикрофона
    mic = np.frombuffer(packet[9:521], dtype=np.uint16)
```

```
    # возможноепрореженииесигнала   /
       понижениечастотыдискретизации
    # используется( наслабыхкомпьютерах  )
    # downsampling = 4
    # downsampling = 8
    downsampling = 1

    mic =  (
        mic
        .reshape(len(mic) // downsampling, downsampling)
        .mean(axis=1)
    )

    return bmp0, bmp1, is_tone_playing, mic
```

далее в модуле `serial_port` объявляется замок, который будет в дальнейшем использоваться в тех местах где потенциально 2 процесса могут обращаться к одной переменной:

```
lock = threading.Lock()
```

Поток `serial_port` собирает данные в буффер-очередь. Этот буффер нужен является временным хранилищем прочитанных данных до тех пор, пока поток графического интерфейса не прочитает их.

5.2.4. основной цикл чтения данных

В основной функции `run`, которую будет исполнять тред присутсвует бесконечный цикл.

В начале цикла проверяется не включен ли флаг `stop_flag` (который может быть включен если пользователь закроет окно (метод `closeEvent` в классе `gui`) или через прерывание с клавиатуры в коммандной строке (CTRL-C). Если флаг включен то соединение с устройством через серийный порт закрывается:

```
while True:

    if stop_flag:
        port.close()
        return
```

Далее по циклу происходит вызов функции `read_packet`:

```
bmp0, bmp1, is_tone_playing, mic = read_packet()
```

Графический интерфейс будет читать данные не каждый раз когда они появились а реже (слишком частое чтение тормозит графический интерфейс и делает его неотзывчивым). Более точно, поток `serial_port` посылает потоку графического интерфейса сигнал о том, что нужно получить новую порцию данных всякий раз когда: - считалось новое значение на сенсорах давления, отличающееся от предыдущего значения (скорость обновления давления на сенсорах значительно медленнее чем скорость обновления данных с микрофона, поэтому перерисовка происходит сразу как только данные пришли) - так как данные микрофона обновляются очень часто, то сигнал о том что нужно обноивить графики в графическом интерфейсе посылается через каждые 128 чтений.

Эта логика реализована в оставшейся части функции `run`:

```
if bmp0 != bmp0_prev or bmp1 != bmp1_prev:

    bmp0_prev = bmp0
    bmp1_prev = bmp1

    bmp_signal.emit()


mic_buffer.extend(mic)
mic_i += len(mic)

if mic_i == mic_un:
```

```
    mic_i = 0

    t1 = time.time()
    dt = t1 - t0
    rate = mic_un / dt
    _rate_arr[_rate_i] = rate
    _rate_i += 1
    if _rate_i == len(_rate_arr):
        rate_mean = _rate_arr.mean()
        _rate_i = 0
    t0 = t1
    mic_signal.emit()
```

Также здесь замеряется средняя частота дискретизации `rate_mean` - сколько в среднем приходило значений в секунду с микрофона.

Когда потоку графического интерфейса посылаются сигналы `bmp_signal.em` и `mic_signal.emit()` чтобы он запросил новые данные, то поток ‘gui‘ вызывает функции `get_bmp` и `get_mic`. В `get_bmp` просто возвращаются последние значения с сенсоров. В `get_mic` берутся последние несколько значений из `mic_buffer` и также делается преобразование фурье:

5.2.5. пребразование Фурье сигнала

получаются последние nfft значений

```
mic_for_fft = mic_buffer.most_recent(nfft)  # with overlap (running
    window for STFT)
```

формируем массив частот для графика спектра

```
f = np.fft.rfftfreq(nfft, d=1/rate_mean/2)
```

преобразуем сигнал в коэффициенты фурье

```
a = np.fft.rfft(mic_for_fft)
```

берем модуль комплексного числа

```
a = np.abs(a)  # magnitude
```

преобразуем магнитуду в децибеллы

```
a = 10 * np.log10(a)
```

ограничение возвращаемых частот спектра (интересующий нас диапазон)

```
hz_limit  = (f > 40) & (f < 40_000)
fft_f = f[hz_limit]
fft_a = a[hz_limit]

return mic, fft_f, fft_a, rate_mean
```

5.3 Програмное обеспечение для компьютера: графический интерфейс

Графический интерфейс программы представляет собой виджет PyQt5, он наследуется от класса `PyQt5.QtWidgets.QWidget`. Класс содержит следующие методы:

- `__init__`
- `init_ui`
- `autorange`
- `pressure_diff`
- `init_mic`
- `init_bmp`
- `bmp_update`
- `mic_update`
- `closeEvent`

В начале класса объявляются статические поля - стандартный для PyQt5 способ обзявления сигналов:

```
bmp_signal = PyQt5.QtCore.pyqtSignal()
mic_signal = PyQt5.QtCore.pyqtSignal()
```

5.3.1. подключение сигналов к потоку сбора данных с USB

В инициализаторе класса `__init__` эти сигналы подключаются к Qt-слотам: при срабатывании первого сигнала будет вызываться метод обновления графика датчкиков давления. А при срабатывании второго синнала будет вызываться метод обновления графика с микрофона а также графика FFT-спектра.

```
def __init__(self):
    super().__init__()

    self.init_ui()

    self.bmp_signal.connect(self.bmp_update)
    self.mic_signal.connect(self.mic_update)
```

Также в инициализаторе класса при помощи ключевого слова super вызывается инициализатор родительского класса виджета. Далее вызывается метод `init_ui` который создаст необходимые элементы интерфейса.

5.3.2. инициализация графического интерфейса

В данном методе иницализируются элементы графического интерфейса. Создается элемент `layout = PyQt5.QtWidgets.QVBoxLayout()` - этот элемент позволяет добавлять в него другие элементы и распоолагать их вертикально. В этот `layout` будут помещаться графики с датчиков давления, с микрофона и также график FFT для сигнала с микрофона. Также добавляется другой layout для кнопок управления.

С помощью вызова метода `self.setGeometry(100, 100, 1200, 800)` задаются размеры окна-виджета приложения. Затем вызываются методы `init_mic` и `init_bmp`.

5.3.3. инициализация графиков сигнала с микрофона

В данном методе иницализируются параметры связанные с графиками сигнала с микрофона и графиком FFT. Объявляется буффер, в котором будут храниться значения с микрофона:

```
self.mic = np.full(self.mic_n, np.nan)
```

Указатель на текущее положение в буффере задается переменной `mic_cursor`.

Создается объект fft графика:

```
self.fft_plot = pyqtgraph.PlotWidget(disableAutoRange=True)
```

Графику добавляется сетка и включается логарифмический режим для осей X (частота звука) и Y (громкость звука в децибеллах):

```
self.fft_plot.showGrid(x=True, y=True, alpha=0.15)
self.fft_plot.setLogMode(x=True, y=True)
```

Оба графика для сигнала и для фурье-образа добавляются на layout созданный в методе `init_ui`:

```
self.layout.addWidget(self.fft_plot)
self.layout.addWidget(self.mic_plot)
```

5.3.4. инициализация графиков сигнала с датчиков давленя

В данном методе инициализируются 2 буффера для хранения последних `self.bmp_n = 100` показаний с датчиков давления:

```
self.bmp0 = np.full(self.bmp_n, np.nan)
self.bmp1 = np.full(self.bmp_n, np.nan)
```

Изначально буфферы инициализируются значениями `np.nan`. Это позволит не отображать их на графике и не сбивать масштаб по вертикали. `self.state = 'норм'`: инициализируется состояние (вдох/выдох/норм).

Определение вдоха или выдоха будет определяться на основе разнице с датчиков давления. Изначально в нормальном состоянии между датчиками тоже может присутсвовать разница. Поэтому нужно будет делать поправку на разность давления в нормальном состоянии (не вдох и не выдох). Это значение будет храниться в переменной `self.normal_dp`.

5.3.5. обновление графиков датчиков давления

Данный метод вызывается по сигналу из потока `serial_port` когда приходят новые данные с датчиков давления. В данном методе читаются данные

из буфферов потока `serial_port`. Полученные данные сохраняются в собственные буфферы в классе `GUI`: `self.bmp0`, `self.bmp0`.

С помощью новых данных обновляются кривие на графике:

```
self.bmp0_curve.setData(self.bmp0)
self.bmp1_curve.setData(self.bmp1)
```

Если буфферы заполнились то они заполняются значениями `np.nan`:

```
if self.bmp_cursor == self.bmp_n:
    self.bmp0[:] = np.nan
    self.bmp1[:] = np.nan
    self.bmp_cursor = 0
```

Также в заголовок графика выводится состояние (вдох/выдох/норм):

```
self.bmp_plot.setTitle(f'<h1>{state}  bmp0 – bmp1 = {bmp0 – bmp1
   :>+3.2f}</h1>')
```

5.3.6. обновление графиков микрофона

Данный метод обновляет график сигнала с микрофона и график FFT. Этот метод также вызывается по сигналу из процесса отвечающего за сбор данных с USB-устройства. После того как новые данные получены. То обновляются кривые на графиках:

```
self.mic_curve.setData(self.mic)
self.fft_curve.setData(fft_f, fft_a)
```

В заголовок графика выводится частота дискретизации с которой был принят сигнал. (она может меняться со временем)

```
self.mic_plot.setTitle(f'<h1>Sample Rate: {rate/1000:0.2f} kHz</h1
   >')
```

5.3.7. обработка безопасного закрытия приложения

Данный метод является стандартным для Qt методом который вызывается при закрытии приложения. (Например через значек (x) в полоске меню). При

закрытии приложения устанавливается флаг для потока `serial_port`. Когда эта переменная примет значение False, то поток прекратит прием данных с устройства и безопасно закроет соединение через последовательный порт.

```
def closeEvent(self, event):
    serial_port.stop_flag = True
```

Схема основных элементов программного обеспечения для компьютера Arduino представлена на рисунке 18.

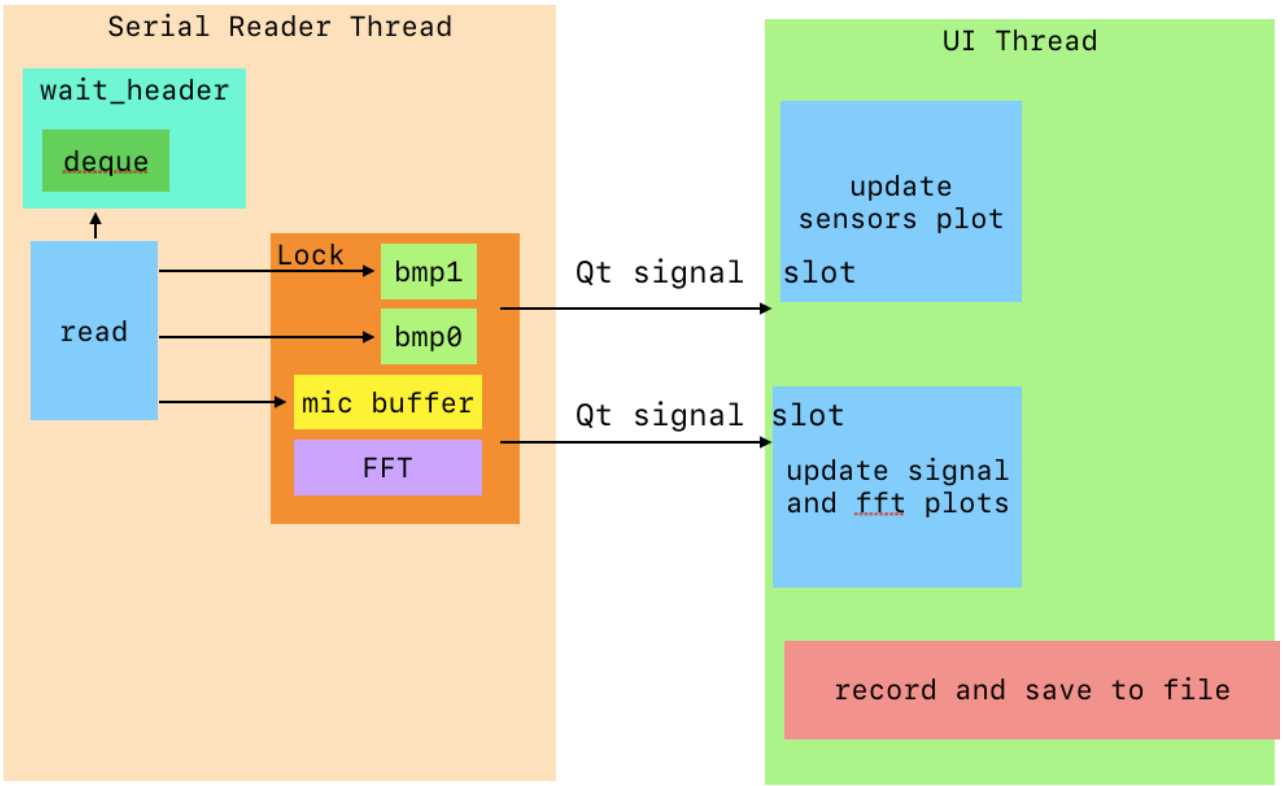

Рисунок 18 – Диаграмма классов ПО для компьютера

Выводы

1. Создана аппаратная часть устройства состоящая из микроконтроллера, микрофона, динамиков, датчиков давления и компьютера.
2. Обеспечен диапазон принятия звуковых сигналов в диапазоне до 40кГц
3. Написано програмное обеспечение для микроконтроллера Arduino
4. Обеспечена обработка полученного сигнала с использованием спектральных методов оценивания
5. Написано програмное обеспечения для компьютера
6. Обеспечена возможность обработки сигнала при помощи технологии Nvidia CUDA
7. Разработана компьютерная моделб распространения звука в легких (двухмерный и трехмерный варианты)

Заключение

В результате данной работы был создан рабочий прототип устройства, позволяющего анализировать в реальном времени звуковой сигнал высокого качества Также было написано програмное обеспечение к этому прибору. Программное обеспечение позволяет позволяет записывать и анализировать звуковые сигналы в реальном времени. Есть возможность рассмотреть различные характеристики сигнала, такие как спектр Фурье, скользящее среднее. Также есть возможность записывать аудиосигнал на жесткий диск для его последующей обработки. Сигнал записывается с частотой дискретизации 660кГц. Данная частота позволяет анализировать ультразвуковой сигнал до 330кГц.

Данное устройство предназначено для получения дополнительной информации из звука поступающего от сердца, легких и других внутренних органов, которую нельзя услышать на обычном стетоскопе. Данная дополнительная информация может быть использована врачом для осуществления более качественной медицинской диагностики.

Развитие проекта можно продолжить в направлении улучшения качества звука. Для этого нужно использовать более качественный микрофон, который будудет обладать более высокой границей воспринемаемых частот и более качественный усилитель. Также нужно произвести опрос врачей о том, какие именно характеристики звука со стетоскопа важны.

Улучшений в програмном обеспечении можно достигнуть путем оптимизации алгоритмов распаралеливания на нескольких ядрах процессора или на видеокарте. Также, на основе информации от докторов, можно сделать систему распознавания различных забовалеваний легких и сердца.

Разработка данного проекта велась с помощью системы контроля версий git. Исходный код програмного обеспечения, этапы создания и документация доступны по адресу: [18]

Список использованных источников

1 Стетоскоп 3M™ Littmann® Electronic Stethoscope Model 3200
http://www.littmann.com/3M/en_US/littmann-stethoscopes/products/~/3M-Littmann-Electronic-Stethoscope-Model-3200?N=5932256+8711017+3293188392&rt=rud

2 Стетоскоп CMS-VESD SPO2 PR
http://www.contecmed.com/index.php?page=shop.product_details&flypage=flypage.tpl&product_id=26&category_id=13&option=com_virtuemart&Itemid=600

3 Стетоскоп CMS-VE
http://www.contecmed.com/index.php?page=shop.product_details&flypage=flypage.tpl&product_id=21&category_id=13&option=com_virtuemart&Itemid=600

4 Стетоскоп CMS-M
http://www.contecmed.com/index.php?page=shop.product_details&flypage=flypage.tpl&product_id=27&category_id=13&option=com_virtuemart&Itemid=600

5 Стетоскоп Cardionics E-scope II
http://www.cardionics.com/product/clinical-systems/hearing-impaired-e-scope

6 Стетоскоп Thinklabs One
http://www.thinklabs.com/one-digital-stethoscope

7 Стетоскоп Eko Core
https://ekodevices.com/core/

8 Стетоскоп Eko Duo

https://ekodevices.com/duo/

9 АЦП Arduino Due

https://www.arduino.cc/en/Main/ArduinoBoardDue

10 Библиотека Adafruit BMP280

https://github.com/adafruit/Adafruit_BMP280_Library

11 Схема усилителя для микрофона

http://full-chip.net/analogovaya-elektronika/70-usilitel-dlya-elektretnogo-mikrofona-s-nizkim-urovnem-shuma-shema.html

12 Руководство пользователя и технические характеристики операционного усилителя MCP6022

https://lib.chipdip.ru/291/DOC000291231.pdf

13 Язык программирования Python

https://www.python.org/downloads/

14 Программа для запуска на компьютере для АЦП Arduino Due

https://github.com/tandav/ultrasonic-stethoscope/blob/master/arduino/app.py

15 Технология Nvidia CUDA

http://www.geforce.com/hardware/technology/cuda

16 Программа Arduino IDE

https://www.arduino.cc/en/Main/Software

17 Видео модели распространения звука в легких

https://www.youtube.com/watch?v=E6hVebjHmME

18 Исходный код, документация и этапы создания проекта ультразвукового стетоскопа
https://github.com/tandav/ultrasonic-stethoscope

19 Горшков, Ю. Г. Обработка речевых и акустических биомедецинских сигналов на основе вейвлетов - Идательство Радиотехника, 2017. - 240 с.

20 Дьяконов, В. П. Вейвлеты. От теории к практике. М.: СОЛОН-Р, 2002. - 400 с.

21 Чистович Л. А. Физиология речи. Восприятие речи человеком. АН СССР. Л.: Наука, 1976. - 388 с.

22 Дворянкин С.В. Взаимосвязь цифры и графики, звука и изображения // Открытые системы, 2000. - 93 с.

23 Блаттер К. Вейвлет-анализ. Основы теории. М.: Техносфера, 2006. - 280 с.

24 Галяшина Е.И. Речь под микроскопом // Компьютерра, 1999. - 10 с.

6 Приложение 1

Класс LungsModel для модели распространения звука в легких

```
from pyqtgraph.Qt import QtCore, QtGui
from PyQt5.QtGui import QApplication
import pyqtgraph as pg
import numpy as np
import sys
import signal


class LungsModel():
    l_default = 0.065
    h_default = 0.55e-6
    f_default = 110
    # l_default = 0.1
    # h_default = 1
    # f_default = 440

    def __init__(self, L=l_default, H=h_default, F=f_default):
        # self.r = np.load('../cube-full-460-512-512.npy')
        # self.r = np.load('../cube-full-460-512-512.npy')[::1,
           ::1, ::1]
        # self.r = np.load('../cube-full-460-512-512.npy')[::2,
           ::2, ::2]
        # self.r = np.load('../cube-full-460-512-512.npy')[::4,
           ::4, ::4]
        # k = 11
        # self.r = np.load('../cube-full-460-512-512.npy')[::k, ::k
           , ::k]
        y_start = 160
        x_start = 145
        sqr = 235
        k = 3
```

```
self.r = np.load('../cube-full-460-512-512.npy')[
    20:80:k,
    y_start : y_start + sqr : k,
    x_start : x_start + sqr : k,
]
# self.r = np.load('../cube-full-460-512-512.npy')[::2,
   ::2, ::2]
# self.r = np.random.random((2,3,4))
self.r[0,0,0] = 3
self.ro  = 1e-5 + 1.24e-3 * self.r - 2.83e-7 * self.r *
   self.r + 2.79e-11 * self.r * self.r * self.r
self.c = (self.ro + 0.112) * 1.38e-6

self.t = 0
self.l = L # dt, time step
self.h = H # dx = dy = dz = 1mm
self.K = self.l / self.h * self.c
self.K2 = self.K**2
self.K_2_by_3 = self.K**2 / 3


# initial conditions
self.P_pp = np.zeros_like(self.ro) # previous previous t -
   2
self.P_p  = np.zeros_like(self.ro) # previous          t -
   1
self.P    = np.zeros_like(self.ro) # current           t

N = self.P.shape[1]
self.A, self.B, self.C = 2, N//2, N//2 # sound source
   location
# self.A, self.B, self.C = 2, N//2, N//2 # sound source
   location
# self.oA, self.oB, self.oC = 6, N//2, N//2 # sound source
   location
```

```
        self.oA, self.oB, self.oC = 4, N//2, N//2 # sound source
            location

        self.f = F

        self.signal_window = 64
        self.source_signal = np.zeros(self.signal_window)
        self.observ_signal = np.zeros(self.signal_window)

        print(f'init model l={self.l} h={self.h} f={self.f}')

    def update_P(self):
        '''
        mb work with flat and then reshape in return
        norm by now, mb add some more optimisations in future, also
            cuda
        '''

        S = self.P_p.shape[0]
        N = self.P_p.shape[1]

        self.P[2:-2, 2:-2, 2:-2] = 2 * self.P_p[2:-2, 2:-2, 2:-2] -
            self.P_pp[2:-2, 2:-2, 2:-2]

        Z = np.zeros_like(self.P_p)
        Z[2:-2, 2:-2, 2:-2] = 22.5 * self.P_p[2:-2, 2:-2, 2:-2]

        cell_indeces_flat = np.arange(S * N * N).reshape(S, N, N)
            [2:-2, 2:-2, 2:-2].ravel().reshape(-1, 1) # vertical
            vector

        s1_indexes_flat = cell_indeces_flat + np.array([-1, 1, -N,
            N, -N**2, N**2])      # i±1 j±1 k±1
        s2_indexes_flat = cell_indeces_flat + np.array([-1, 1, -N,
            N, -N**2, N**2]) * 2  # i±2 j±2 k±2
```

```
s1_values = self.P_p.ravel()[s1_indexes_flat] # each row
   contains 6 neighbors of cell
s2_values = self.P_p.ravel()[s2_indexes_flat] # each row
   contains 6 neighbors of cell
s1 = np.sum(s1_values, axis=1) # sum by axis=1 is faster
   for default order
s2 = np.sum(s2_values, axis=1)

Z[2:-2, 2:-2, 2:-2] -=   4 * s1.reshape(S-4, N-4, N-4)
Z[2:-2, 2:-2, 2:-2] += 1/4 * s2.reshape(S-4, N-4, N-4)

m1 = np.array([1, -1, -1/8, -1/8])
m2 = np.array([1, -1])

s3_V_indexes = cell_indeces_flat + np.array([N**2, -N**2,
   2*N**2, -2*N**2])
s3_V_values = self.P_p.ravel()[s3_V_indexes] * m1 # po idee
    mozhno za skobki kak to vinesti m1 i m2
s3_V_sum = np.sum(s3_V_values, axis=1)
s3_N_indexes = cell_indeces_flat + np.array([N**2, -N**2])
s3_N_values = self.ro.ravel()[s3_N_indexes] * m2
s3_N_sum = np.sum(s3_N_values, axis=1)
s3 = (s3_V_sum * s3_N_sum).reshape(S-4, N-4, N-4)

s4_V_indexes = cell_indeces_flat + np.array([N, -N, 2*N,
   -2*N])
s4_V_values = self.P_p.ravel()[s4_V_indexes] * m1
s4_V_sum = np.sum(s4_V_values, axis=1)
s4_N_indexes = cell_indeces_flat + np.array([N, -N])
s4_N_values = self.ro.ravel()[s4_N_indexes] * m2
s4_N_sum = np.sum(s4_N_values, axis=1)
s4 = (s4_V_sum * s4_N_sum).reshape(S-4, N-4, N-4)

s5_V_indexes = cell_indeces_flat + np.array([1, -1, 2, -2])
s5_V_values = self.P_p.ravel()[s5_V_indexes] * m1
```

```
        s5_V_sum = np.sum(s5_V_values, axis=1)
        s5_N_indexes = cell_indeces_flat + np.array([1, -1])
        s5_N_values = self.ro.ravel()[s5_N_indexes] * m2
        s5_N_sum = np.sum(s5_N_values, axis=1)
        s5 = (s5_V_sum * s5_N_sum).reshape(S-4, N-4, N-4)

        Z[2:-2, 2:-2, 2:-2] += (s3 + s4 + s5) * self.ro[2:-2, 2:-2,
            2:-2]
        self.P[2:-2, 2:-2, 2:-2] -= Z[2:-2, 2:-2, 2:-2] * self.
           K_2_by_3[2:-2, 2:-2, 2:-2]
        # self.P[self.ro < 0.1] = 0

    def step(self):
        self.P_old = self.P
        self.update_P()
        self.P[self.A, self.B, self.C] = np.sin(2 * np.pi * self.f
           * self.t) # sound source
        self.P_pp  = self.P_p
        self.P_p   = self.P_old


        self.source_signal = np.roll(self.source_signal, -1)
        self.observ_signal = np.roll(self.observ_signal, -1)
        self.source_signal[-1] = self.P[self.A, self.B, self.C]
        self.observ_signal[-1] = self.P[self.oA, self.oB, self.oC]

        self.t += self.l
```

7 Приложение 2

Класс AppGUI для модели распространения звука в легких

```
class AppGUI(QtGui.QWidget):
    steps_state = QtCore.pyqtSignal([int])

    def __init__(self):
        super().__init__()

        self.model = LungsModel()

        self.data = self.model.P
        self.observ_slice = np.zeros(self.model.signal_window)
        self.z_slice = self.model.A
        self.y_slice = self.model.B
        self.x_slice = self.model.C

        self.init_ui()
        self.qt_connections()

    def init_ui(self):
        pg.setConfigOption('background', 'w')
        pg.setConfigOption('imageAxisOrder', 'row-major')

        self.z_axis_name = ('Head', 'Feet')
        self.y_axis_name = ('Face', 'Back')
        self.x_axis_name = ('Left Hand', 'Right Hand')



        self.layout = QtGui.QVBoxLayout()

        # self.setGeometry(50, 50, 700, 700)
        self.setWindowTitle('Lungs Model')
        self.l_label = QtGui.QLabel('dt')
```

```
self.h_label = QtGui.QLabel('h')
self.f_label = QtGui.QLabel('f')

self.l_spin = pg.SpinBox(value=self.model.l, step=0.01,
   siPrefix=False, suffix='s')
self.h_spin = pg.SpinBox(value=self.model.h, step=0.01,
   siPrefix=False)
self.f_spin = pg.SpinBox(value=self.model.f, step=1,
   siPrefix=False)
self.l_spin.setMaximumWidth(150)
self.h_spin.setMaximumWidth(150)
self.f_spin.setMaximumWidth(150)

self.reset_params_button = QtGui.QPushButton('Defaults')
self.reinit_button = QtGui.QPushButton('Restart Model')

self.model_params_layout = QtGui.QHBoxLayout()
self.model_params_layout.addWidget(self.l_label)
self.model_params_layout.addWidget(self.l_spin)
self.model_params_layout.addWidget(self.h_label)
self.model_params_layout.addWidget(self.h_spin)
self.model_params_layout.addWidget(self.f_label)
self.model_params_layout.addWidget(self.f_spin)

# radio buttons
   ____________________________________________________________

self.arrays_to_vis = [QtGui.QRadioButton('P'), QtGui.
   QRadioButton('r'), QtGui.QRadioButton('ro'), QtGui.
   QRadioButton('c'), QtGui.QRadioButton('K')]
self.arrays_to_vis[0].setChecked(True)
# self.radio_layout = QtGui.QHBoxLayout()

for rad in self.arrays_to_vis:
    self.model_params_layout.addWidget(rad)
```

```
    rad.clicked.connect(self.array_to_vis_changed)

self.model_params_layout.addWidget(self.reset_params_button
    )
self.model_params_layout.addWidget(self.reinit_button)

self.z_slice_label = QtGui.QLabel(f'Z axis [{self.
    z_axis_name[0]} - {self.z_axis_name[1]}] Slice: {self.
    z_slice + 1}/{self.data.shape[0]}')
self.y_slice_label = QtGui.QLabel(f'Y axis [{self.
    y_axis_name[0]} - {self.y_axis_name[1]}] Slice: {self.
    y_slice + 1}/{self.data.shape[1]}')
self.x_slice_label = QtGui.QLabel(f'X axis [{self.
    x_axis_name[0]} - {self.x_axis_name[1]}] Slice: {self.
    x_slice + 1}/{self.data.shape[2]}')


# slices plots
    ------------------------------------------------------------


self.autolevels = True
self.levels = (0, 100)
self.glayout = pg.GraphicsLayoutWidget()
self.glayout.ci.layout.setContentsMargins(0, 0, 0, 0)
self.glayout.ci.layout.setSpacing(0)
self.z_slice_img = pg.ImageItem(self.data[self.z_slice, :,
    :], autoLevels=self.autolevels, levels=self.levels,
    border=pg.mkPen(color='r', width=3))
self.y_slice_img = pg.ImageItem(self.data[:, self.y_slice,
    :], autoLevels=self.autolevels, levels=self.levels,
    border=pg.mkPen(color='g', width=3))
self.x_slice_img = pg.ImageItem(self.data[:, :, self.
    x_slice], autoLevels=self.autolevels, levels=self.levels
    , border=pg.mkPen(color='b', width=3))
```

```
self.z_slice_plot = self.glayout.addPlot()
self.y_slice_plot = self.glayout.addPlot()
self.x_slice_plot = self.glayout.addPlot()
# self.z_slice_plot.setTitle(f'Z axis [{self.z_axis_name
  [0]} - {self.z_axis_name[1]}]')
# self.y_slice_plot.setTitle(f'Y axis [{self.y_axis_name
  [0]} - {self.y_axis_name[1]}]')
# self.x_slice_plot.setTitle(f'X axis [{self.x_axis_name
  [0]} - {self.x_axis_name[1]}]')
self.z_slice_plot.setAspectLocked()
self.y_slice_plot.setAspectLocked()
self.x_slice_plot.setAspectLocked()
self.z_slice_plot.setMouseEnabled(x=False, y=False)
self.y_slice_plot.setMouseEnabled(x=False, y=False)
self.x_slice_plot.setMouseEnabled(x=False, y=False)
self.z_slice_plot_y_helper1 = self.z_slice_plot.plot([0
                , self.data.shape[2] ], [self.y_slice    ,
    self.y_slice      ], pen='g')
self.z_slice_plot_y_helper2 = self.z_slice_plot.plot([0
                , self.data.shape[2] ], [self.y_slice + 1,
    self.y_slice + 1  ], pen='g')
self.z_slice_plot_x_helper1 = self.z_slice_plot.plot([self.
  x_slice    , self.x_slice       ], [0              ,
  self.data.shape[1]], pen='b')
self.z_slice_plot_x_helper2 = self.z_slice_plot.plot([self.
  x_slice + 1, self.x_slice + 1   ], [0              ,
  self.data.shape[1]], pen='b')
self.y_slice_plot_z_helper1 = self.y_slice_plot.plot([0
                , self.data.shape[2] ], [self.z_slice    ,
    self.z_slice      ], pen='r')
self.y_slice_plot_z_helper2 = self.y_slice_plot.plot([0
                , self.data.shape[2] ], [self.z_slice + 1,
    self.z_slice + 1  ], pen='r')
self.y_slice_plot_x_helper1 = self.y_slice_plot.plot([self.
  x_slice    , self.x_slice       ], [0              ,
```

```
        self.data.shape[0]], pen='b')
    self.y_slice_plot_x_helper2 = self.y_slice_plot.plot([self.
        x_slice + 1, self.x_slice + 1   ], [0               ,
        self.data.shape[0]], pen='b')
    self.x_slice_plot_z_helper1 = self.x_slice_plot.plot([0
                    , self.data.shape[1] ], [self.z_slice    ,
         self.z_slice      ], pen='r')
    self.x_slice_plot_z_helper2 = self.x_slice_plot.plot([0
                    , self.data.shape[1] ], [self.z_slice + 1,
         self.z_slice + 1  ], pen='r')
    self.x_slice_plot_y_helper1 = self.x_slice_plot.plot([self.
        y_slice    , self.y_slice       ], [0               ,
        self.data.shape[0]], pen='g')
    self.x_slice_plot_y_helper2 = self.x_slice_plot.plot([self.
        y_slice + 1, self.y_slice + 1   ], [0               ,
        self.data.shape[0]], pen='g')
    self.z_slice_plot.invertY(True)
    self.y_slice_plot.invertY(True)
    self.x_slice_plot.invertY(True)
    self.z_slice_plot.setLabel('bottom', f'X axis [{self.
        x_axis_name[0]} - {self.x_axis_name[1]}]')
    self.z_slice_plot.setLabel('left'  , f'Y axis [{self.
        y_axis_name[1]} - {self.y_axis_name[0]}]')
    self.y_slice_plot.setLabel('bottom', f'X axis [{self.
        x_axis_name[0]} - {self.x_axis_name[1]}]')
    self.y_slice_plot.setLabel('left'  , f'Z axis [{self.
        z_axis_name[1]} - {self.z_axis_name[0]}]')
    self.x_slice_plot.setLabel('bottom', f'Y axis [{self.
        y_axis_name[0]} - {self.y_axis_name[1]}]')
    self.x_slice_plot.setLabel('left'  , f'Z axis [{self.
        z_axis_name[1]} - {self.z_axis_name[0]}]')
    self.z_slice_plot.addItem(self.z_slice_img)
    self.y_slice_plot.addItem(self.y_slice_img)
    self.x_slice_plot.addItem(self.x_slice_img)
```

```
self.z_slice_img.setRect(pg.QtCore.QRectF(0, 0, self.data.
   shape[2], self.data.shape[1]))
self.y_slice_img.setRect(pg.QtCore.QRectF(0, 0, self.data.
   shape[2], self.data.shape[0]))
self.x_slice_img.setRect(pg.QtCore.QRectF(0, 0, self.data.
   shape[1], self.data.shape[0]))
self.z_slice_img.setZValue(-1)
self.y_slice_img.setZValue(-1)
self.x_slice_img.setZValue(-1)

#------------------------------ signal plots
   ------------------------------
plots_font = QtGui.QFont()
fontsize = 9
plots_font.setPixelSize(fontsize)
plots_height = 250

self.source_plot = pg.PlotWidget(title=f'Acoustic Pressure
   at P[{self.model.A}, {self.model.B}, {self.model.C}] (
   sound source)')
self.source_plot.showGrid(x=True, y=True, alpha=0.1)
self.source_plot.setYRange(-1, 1) # w\o np.log(a)
# self.fft_widget.setYRange(-15, 0) # w/ np.log(a)
self.source_plot.getAxis('bottom').setStyle(tickTextOffset
   = fontsize)
self.source_plot.getAxis('left').setStyle(tickTextOffset =
   fontsize)
self.source_plot.getAxis('bottom').tickFont = plots_font
self.source_plot.getAxis('left').tickFont = plots_font
self.source_plot.setMaximumHeight(plots_height)
self.source_curve = self.source_plot.plot(pen='b')



self.observ_plot = pg.PlotWidget(title=f'Acoustic Pressure
   at P[{self.model.oA}, {self.model.oB}, {self.model.oC
```

```
    }]')
self.observ_plot.showGrid(x=True, y=True, alpha=0.1)
self.observ_plot.getAxis('bottom').setStyle(tickTextOffset
    = fontsize)
self.observ_plot.getAxis('left').setStyle(tickTextOffset =
    fontsize)
self.observ_plot.getAxis('bottom').tickFont = plots_font
self.observ_plot.getAxis('left').tickFont = plots_font
self.observ_plot.setMaximumHeight(plots_height)
self.observ_curve = self.observ_plot.plot(pen='r')


self.observ_slice_plot = pg.PlotWidget(title=f'Acoustic
    Pressure at P[{self.z_slice}, {self.y_slice}, {self.
    x_slice}]')
self.observ_slice_plot.showGrid(x=True, y=True, alpha=0.1)
self.observ_slice_plot.getAxis('bottom').setStyle(
    tickTextOffset = fontsize)
self.observ_slice_plot.getAxis('left').setStyle(
    tickTextOffset = fontsize)
self.observ_slice_plot.getAxis('bottom').tickFont =
    plots_font
self.observ_slice_plot.getAxis('left').tickFont =
    plots_font
self.observ_slice_plot.setMaximumHeight(plots_height)
self.observ_slice_curve = self.observ_slice_plot.plot(pen
    =0.8)



self.plots_layout = QtGui.QHBoxLayout()
# self.plots_layout.addStretch()
# self.plots_layout.setMinimumWidth(200)
self.plots_layout.addWidget(self.source_plot)
self.plots_layout.addWidget(self.observ_plot)
```

```
self.plots_layout.addWidget(self.observ_slice_plot)

#-------------------------------------------------------------------------------


self.step_layout = QtGui.QHBoxLayout()
self.steps_label = QtGui.QLabel('Number of steps: ')
self.steps_spin = QtGui.QSpinBox()
self.steps_spin.setRange(1, 10000)
self.steps_spin.setValue(50)
self.steps_spin.setMaximumWidth(100)
# self.steps_spin.setMaximumSize(100, 50)
# self.steps_spin.setGeometry(QtCore.QRect(10, 10, 50, 21))
self.step_button = QtGui.QPushButton('Step')
# self.step_button.setMaximumSize(100, 50)
self.step_button.setMaximumWidth(100)
self.steps_progress_bar = QtGui.QProgressBar()
self.step_layout.addWidget(self.steps_label)
self.step_layout.addWidget(self.steps_spin)
self.step_layout.addWidget(self.step_button)
self.step_layout.addWidget(self.steps_progress_bar)



self.z_slice_slider = QtGui.QSlider()
self.z_slice_slider.setStyleSheet('background-color: rgba
   (255, 0, 0, 0.2)')
self.z_slice_slider.setOrientation(QtCore.Qt.Horizontal)
self.z_slice_slider.setRange(0, self.data.shape[0] - 1)
self.z_slice_slider.setValue(self.z_slice)
self.z_slice_slider.setTickPosition(QtGui.QSlider.
   TicksBelow)
self.z_slice_slider.setTickInterval(1)
```

```
self.y_slice_slider = QtGui.QSlider()
self.y_slice_slider.setStyleSheet('background-color: rgba
   (0, 255, 0, 0.2)')
self.y_slice_slider.setOrientation(QtCore.Qt.Horizontal)
self.y_slice_slider.setRange(0, self.data.shape[1] - 1)
self.y_slice_slider.setValue(self.y_slice)
self.y_slice_slider.setTickPosition(QtGui.QSlider.
   TicksBelow)
self.y_slice_slider.setTickInterval(1)

self.x_slice_slider = QtGui.QSlider()
self.x_slice_slider.setStyleSheet('background-color: rgba
   (0, 0, 255, 0.2)')
self.x_slice_slider.setOrientation(QtCore.Qt.Horizontal)
self.x_slice_slider.setRange(0, self.data.shape[2] - 1)
self.x_slice_slider.setValue(self.x_slice)
self.x_slice_slider.setTickPosition(QtGui.QSlider.
   TicksBelow)
self.x_slice_slider.setTickInterval(1)

self.layout.addLayout(self.model_params_layout)
# self.layout.addLayout(self.radio_layout)
self.layout.addWidget(self.z_slice_label)
self.layout.addWidget(self.z_slice_slider)
self.layout.addWidget(self.y_slice_label)
self.layout.addWidget(self.y_slice_slider)
self.layout.addWidget(self.x_slice_label)
self.layout.addWidget(self.x_slice_slider)
self.layout.addWidget(self.glayout)
self.layout.addLayout(self.plots_layout)
# self.layout.addWidget(self.source_plot)
# self.layout.addWidget(self.observ_plot)
self.layout.addLayout(self.step_layout)

self.setLayout(self.layout)
```

```
        self.setGeometry(0, 0, 1440, 900)
        # self.setGeometry(0, 0, 1200, 900)
        self.show()

    def qt_connections(self):
        self.step_button.clicked.connect(self.do_steps)
        self.l_spin.valueChanged.connect(self.l_spin_value_changed)
        self.h_spin.valueChanged.connect(self.h_spin_value_changed)
        self.f_spin.valueChanged.connect(self.f_spin_value_changed)
        self.reset_params_button.clicked.connect(self.reset_params)
        self.reinit_button.clicked.connect(self.reinit_model)
        self.z_slice_slider.valueChanged.connect(self.
           z_slice_slider_changed)
        self.y_slice_slider.valueChanged.connect(self.
           y_slice_slider_changed)
        self.x_slice_slider.valueChanged.connect(self.
           x_slice_slider_changed)
        self.steps_state.connect(self.update_steps_progress_bar)

    # def mouseMoved(self, event):
        # print('mouseMoved event')

    def mouseMoveEvent(self, ev):
        print('pp')

    def l_spin_value_changed(self):
        self.model.l = self.l_spin.value()

    def h_spin_value_changed(self):
        self.model.h = self.h_spin.value()

    def f_spin_value_changed(self):
        self.model.f = self.f_spin.value()
```

```
@QtCore.pyqtSlot(int)
def update_steps_progress_bar(self, current_step):
    self.steps_progress_bar.setValue(current_step / self.
       steps_spin.value() * 100)
    QApplication.processEvents()

def reset_params(self):
    self.l_spin.setValue(LungsModel.l_default)
    self.h_spin.setValue(LungsModel.h_default)
    self.f_spin.setValue(LungsModel.f_default)

    self.z_slice = self.model.A
    self.y_slice = self.model.B
    self.x_slice = self.model.C
    self.z_slice_slider.setValue(self.z_slice)
    self.y_slice_slider.setValue(self.y_slice)
    self.x_slice_slider.setValue(self.x_slice)

    self.arrays_to_vis[0].setChecked(True)
    self.array_to_vis_changed()

def reinit_model(self):
    self.model = LungsModel(self.l_spin.value(), self.h_spin.
       value(), self.f_spin.value())
    self.array_to_vis_changed()

def array_to_vis_changed(self):

    mapping = {
        'P' : self.model.P,
        'r' : self.model.r,
        'ro': self.model.ro,
        'c' : self.model.c,
        'K' : self.model.K,
    }
```

```
    for r in self.arrays_to_vis:
        if r.isChecked():
            self.data = mapping[r.text()]
            self.z_slice_img.setImage(self.data[self.z_slice
                    ])
            self.y_slice_img.setImage(self.data[:, self.y_slice
                , :])
            self.x_slice_img.setImage(self.data[:, :, self.
                x_slice])

def print_mean(self):
    # pass
    print(f'slices mean Z, Y, X   {np.mean(self.data[self.
        z_slice]):8.3e}    {np.mean(self.data[:, self.y_slice,
        :]):8.3e}    {np.mean(self.data[:, :, self.x_slice]):8.3
        e}    cube mean: {np.mean(self.data):8.3e}')

def do_steps(self):
    for i in range(self.steps_spin.value()):
        self.model.step()

        self.z_slice_img.setImage(self.data[self.z_slice
            ])
        self.y_slice_img.setImage(self.data[:, self.y_slice,
            :])
        self.x_slice_img.setImage(self.data[:, :, self.x_slice
            ])

        self.source_curve.setData(self.model.source_signal)
        self.observ_curve.setData(self.model.observ_signal)

        self.observ_slice = np.roll(self.observ_slice, -1)
        self.observ_slice[-1] = self.model.P[self.z_slice, self
            .y_slice, self.x_slice]
```

```
            self.observ_slice_curve.setData(self.observ_slice)

            self.steps_state.emit(i + 1)
            self.print_mean()
        self.steps_state.emit(0)

    def wheelEvent(self, event):
        if self.z_slice_img.sceneBoundingRect().contains(self.
           glayout.mapFromParent(event.pos())):
            self.z_slice = np.clip(self.z_slice + np.sign(event.
               angleDelta().y()), 0, self.data.shape[0] - 1) #
               change bounds 0..N-1 => 1..N
            self.z_slice_slider.setValue(self.z_slice)
        elif self.y_slice_img.sceneBoundingRect().contains(self.
           glayout.mapFromParent(event.pos())):
            self.y_slice = np.clip(self.y_slice + np.sign(event.
               angleDelta().y()), 0, self.data.shape[1] - 1) #
               change bounds 0..N-1 => 1..N
            self.y_slice_slider.setValue(self.y_slice)
        elif self.x_slice_img.sceneBoundingRect().contains(self.
           glayout.mapFromParent(event.pos())):
            self.x_slice = np.clip(self.x_slice + np.sign(event.
               angleDelta().y()), 0, self.data.shape[2] - 1) #
               change bounds 0..N-1 => 1..N
            self.x_slice_slider.setValue(self.x_slice)

    def keyPressEvent(self, event):
        if type(event) == QtGui.QKeyEvent and event.key() == QtCore
           .Qt.Key_Up:
            self.do_steps()
            #here accept the event and do something
            # self.record_values_button_clicked()
            event.accept()
        else:
            event.ignore()
```

```
def update_observ_slice_plot(self):
    self.observ_slice_plot.getPlotItem().setTitle(f'Acoustic
       Pressure at P[{self.z_slice}, {self.y_slice}, {self.
       x_slice}]')
    self.observ_slice = np.zeros(self.model.signal_window)
    self.observ_slice_curve.setData(self.observ_slice)

def update_slice_helpers_lines(self):
    # self.z_slice_plot_y_helper.setData([0           , self.
       data.shape[2] ], [self.y_slice, self.y_slice      ])
    # self.z_slice_plot_x_helper.setData([self.x_slice, self.
       x_slice       ], [0           , self.data.shape[1]])
    # self.y_slice_plot_z_helper.setData([0           , self.
       data.shape[2] ], [self.z_slice, self.z_slice      ])
    # self.y_slice_plot_x_helper.setData([self.x_slice, self.
       x_slice       ], [0           , self.data.shape[0]])
    # self.x_slice_plot_z_helper.setData([0           , self.
       data.shape[1] ], [self.z_slice, self.z_slice      ])
    # self.x_slice_plot_y_helper.setData([self.y_slice, self.
       y_slice       ], [0           , self.data.shape[0]])
    self.z_slice_plot_y_helper1.setData([0               , self
       .data.shape[2] ], [self.y_slice    , self.y_slice
       ])
    self.z_slice_plot_y_helper2.setData([0               , self
       .data.shape[2] ], [self.y_slice + 1, self.y_slice + 1
       ])
    self.z_slice_plot_x_helper1.setData([self.x_slice    , self
       .x_slice       ], [0               , self.data.shape
       [1]])
    self.z_slice_plot_x_helper2.setData([self.x_slice + 1, self
       .x_slice + 1   ], [0               , self.data.shape
       [1]])
    self.y_slice_plot_z_helper1.setData([0               , self
       .data.shape[2] ], [self.z_slice    , self.z_slice
```

```
            ])
        self.y_slice_plot_z_helper2.setData([0              , self
            .data.shape[2] ], [self.z_slice + 1, self.z_slice + 1
            ])
        self.y_slice_plot_x_helper1.setData([self.x_slice    , self
            .x_slice       ], [0              , self.data.shape
            [0]])
        self.y_slice_plot_x_helper2.setData([self.x_slice + 1, self
            .x_slice + 1   ], [0              , self.data.shape
            [0]])
        self.x_slice_plot_z_helper1.setData([0              , self
            .data.shape[1] ], [self.z_slice    , self.z_slice
            ])
        self.x_slice_plot_z_helper2.setData([0              , self
            .data.shape[1] ], [self.z_slice + 1, self.z_slice + 1
            ])
        self.x_slice_plot_y_helper1.setData([self.y_slice    , self
            .y_slice       ], [0              , self.data.shape
            [0]])
        self.x_slice_plot_y_helper2.setData([self.y_slice + 1, self
            .y_slice + 1   ], [0              , self.data.shape
            [0]])

    def z_slice_slider_changed(self):
        self.z_slice = self.z_slice_slider.value()
        self.z_slice_label.setText(f'Z axis [{self.z_axis_name[0]}
            - {self.z_axis_name[1]}] Slice: {self.z_slice + 1}/{self
            .data.shape[0]}')
        self.z_slice_img.setImage(self.data[self.z_slice])
        self.print_mean()
        self.update_observ_slice_plot()
        self.update_slice_helpers_lines()

    def y_slice_slider_changed(self):
        self.y_slice = self.y_slice_slider.value()
```

```
            self.y_slice_label.setText(f'Y axis [{self.y_axis_name[0]}
               - {self.y_axis_name[1]}] Slice: {self.y_slice + 1}/{self
               .data.shape[1]}')
            self.y_slice_img.setImage(self.data[:, self.y_slice, :])
            self.print_mean()
            self.update_observ_slice_plot()
            self.update_slice_helpers_lines()

        def x_slice_slider_changed(self):
            self.x_slice = self.x_slice_slider.value()
            self.x_slice_label.setText(f'X axis [{self.x_axis_name[0]}
               - {self.x_axis_name[1]}] Slice: {self.x_slice + 1}/{self
               .data.shape[2]}')
            self.x_slice_img.setImage(self.data[:, :, self.x_slice])
            self.print_mean()
            self.update_observ_slice_plot()
            self.update_slice_helpers_lines()

    app = QtGui.QApplication(sys.argv)
    # print(sys.argv[1])
    gui = AppGUI()
    signal.signal(signal.SIGINT, signal.SIG_DFL)
    sys.exit(app.exec())
```