Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра Информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**Получение данных по Bluetooth и их последующая обработка**

БГУИР КП 1-40 04 01 011 ПЗ

Выполнил: студент гр. 653502

Турцевич И. М.

Проверил: Шнейдер В.В.

Минск 2019

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc532435158)

[**1 ОБЗОР ИСТОЧНИКОВ** 5](#_Toc532435159)

[1.1 Сравнение с аналогами 5](#_Toc532435160)

[**2 ПОСТАНОВКА ЗАДАЧИ** 7](#_Toc532435161)

[**3 ПРОЕКТИРОВАНИЕ ПРОГРАМНОГО СРЕДСТВА** 8](#_Toc532435162)

[3.1 Фазы пульсации пульсатора доильной установки 8](#_Toc532435163)

[3.1.1 Пульсаторы попарного доения 8](#_Toc532435164)

[3.1.2 Фазы цикла пульсаций 10](#_Toc532435165)

[**4 РАЗРАБОТКА ПРОГРАМНОГО СРЕДСТВА** 12](#_Toc532435187)

[4.1 Описание механизма устройства и алгоритма вычисления фаз 12](#_Toc532435188)

[4.2 Подключение Bluetooth на Android 15](#_Toc532435189)

[4.3 Работа с архивами 15](#_Toc532435190)

[4.4 Описание классов 17](#_Toc532435191)

[**5 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА** 19](#_Toc532435192)

[5.1 Тестирование различных ситуаций программного взаимодействия 19](#_Toc532435193)

[**6 РУКОВОДСТВО ИСПОЛЬЗОВАНИЯ ПРИЛОЖЕНИЯ** 24](#_Toc532435194)

[6.1 Общие сведения об управлении прибором 24](#_Toc532435195)

[6.2 Описание меню программы ble\_std 25](#_Toc532435196)

[6.3 Описание окна «работа с устройством» 26](#_Toc532435197)

[6.4 Описание окна «архив измерений» 28](#_Toc532435198)

[6.5 Описание окна «параметры пульсации» 29](#_Toc532435199)

[6.6 Описание окна «графики» 30](#_Toc532435200)

[6.7 Описание окна «манометр» 31](#_Toc532435201)

[6.8 Описание диалогового окна «создание архива» окна «работа с устройством» 32](#_Toc532435202)

[**Заключение** 34](#_Toc532435203)

[**Список использованных источников** 35](#_Toc532435204)

[**Приложение А. Исходный код основного класса программы** 36](#_Toc532435205)

[**Приложение B. Исходный код класса для построения графиков** 44](#_Toc532435206)

# **ВВЕДЕНИЕ**

Целью разработки данного программного средства является создание приложения, которое будет анализировать колебание вакуума в системе доильной установки, полученное благодаря беспроводной связи с помощью технологии «Bluetooth LE[[1]](#_Список_использованных_источников)».

Данный курсовой проект представляет собой приложение, с помощью которого можно подключаться к bluetooth-модулю. Приложение получает данные из Bluetooth-модуля, строит специфичные графики и на их основе анализирует ранее полученные данные. Анализ в свою очередь будет собой представлять решение задачи, а именно диагностика доильной установки, которая в свою очередь будет описана графиком. График будет представлять оцифрованный сигнал с АЦП (аналоговый цифровой преобразователь) с датчиков давления.

Использование технологии «Bluetooth LE» и сама логическая архитектура приложения позволяет с лёгкостью переделать его под фитнес-браслеты, которые на данный момент очень популярны.

Особенностью данного приложения является его гибкость в создании и дополнении, а также особенностью является то, что оно способно принимать данные и отправлять данные, т. е. реализована двусторонняя связь между устройствами.

# **1 ОБЗОР ИСТОЧНИКОВ**

## 1.1 Сравнение с аналогами

На данный момент в мире существует мало аналогов разрабатываемого приложения на мобильных устройствах. Сейчас на фермах используются специальные оборудования (ППДУ[[2]](#_Список_использованных_источников) — прибор проверки доильных установок) (см. рис. 1) для измерения вакуума. Они в свою очередь довольно большие, сложны в использовании и у них нет связи с интернетом для последующей передачи измеренных показаний на сервера, а также нельзя измерять параметры в реальном времени, что также усложняет процесс проверки измерений.

Данный проект заключается в том, чтобы разработать приложение для мобильного устройства, которое будет включать в себя весь функционал специального оборудования, а также существенно дополнит его. Дополнение включает в себя такие функции, как измерение в реальном времени, индикация значений, передача измерений на удалённые сервера.

В заключении сравнения, разработка данного приложения нацелена на то, чтобы создать функциональный и в то же время простой в использовании продукт, который будет помогать отслеживать здоровье коров на фермах.



Рисунок 1 - ППДУ

# **2 ПОСТАНОВКА ЗАДАЧИ**

Необходимо разработать приложение для работы с Bluetooth-модулем, выполняющие следующие функции:

* Приложение должно иметь простой интерфейс и автоматизированность.
* При запуске приложения в стартовом окне должен появляться список с Bluetooth-устройствами, среди которых надо выбрать необходимое устройство.
* Приложение должно иметь две глобальные логики: сбор данных в реальном времени и их анализ.
* Сбор данных должны иллюстрировать графики, чтобы была возможность простого просмотра данных в реальном времени.
* Анализ должен представлять из себя просмотр тех же графиков, просмотр табличных значений с подсветкой (индексацией) в зависимости от значения измерения.

# **3 ПРОЕКТИРОВАНИЕ ПРОГРАМНОГО СРЕДСТВА**

## 3.1 Фазы пульсации пульсатора доильной установки

### 3.1.1 Пульсаторы попарного доения

Все доильные системы работают при постоянном вакууме, что и обеспечивает выдаивание молока из вымени коровы. Если соски животного подвергаются непрерывному воздействию вакуума, кровь перестает циркулировать, застаивается на конце соска, заставляя его набухать. В результате это приводит к тому, что сдавливаются молочные каналы соска и это препятствует свободному отделению молока из вымени.  Кроме того, неустойчивая пульсация создает дискомфорт для коровы, препятствует полному выдаиванию вымени, удлиняет сам процесс доения и не способствует здоровью вымени, часто приводя к маститу.

Правильно настроенная пульсация обеспечивает нормальный доильный процесс, имитирующий естественное сосание теленка, периодически прерывая подачу вакуума в доильные стаканы. Во время такой "фазы покоя” сосок отдыхает и готовится к следующей “фазе молока".

Скорость пульсации является важной характеристикой доильного пульсатора. Ее обеспечивает клапан, который открывается и закрывается, в соответствие с настроенной частотой пульсации. Современные пульсаторы попарного доения охватывают широкий диапазон скоростей между 45 PP/мин. до 75 PP\мин. Отношение пульсации определяется соотношение процента времени в фазе доения и в фазе покоя. Первое число означает доильный фазу, вторая - фазу отдыха. Например, в соотношении 60:40 - 60% в течение одной пульсации система находится в молочной фазе и 40% в фазе покоя. Скорость и соотношение установлены независимо друг от друга, но вместе играют важную роль в доильные системы.

Пневматические регулируемые пульсаторы имеют возможность регулировать величину пульсации (смотрите спецификацию для каждого из отдельных пульсаторов).

Оптимальная частота пульсации позволяет переключаться между "фазой молока" и "фазы покоя" быстро, плавно и, прежде всего, последовательно (постоянное точное повторение пульсации тренирует корову для быстрой молокоотдачи и максимизирует производство).

Ниже приведены различные варианты настройки пульсации.

Отношение скоростей и количество секунд отдыха за минуту:

70 PPM 70:30 18 сек/ мин

60 PPM 60:40 24 сек/ мин

50 PPM 50:50 30 сек/ мин

Опытным путем было установлено, что наилучший результат доения дает настройка пульсатора на частоту пульсации 60:40. По умолчанию все пульсаторы установлены на эту частоту.

### 3.1.2 Фазы цикла пульсаций

В таблице 3.1 представлены фазы цикла пульсаций.

Таблица 3.1 – Фазы цикла пульсаций

|  |  |  |
| --- | --- | --- |
| Символ | Обозначение | Вакуум в пространстве пульсации, кПа |
| A | Фаза эвакуации | От 4 кПа до макс.вакуум – 4 кПа |
| B | Фаза вакуума | Выше чем макс.вакуум – 4 кПа |
| C | Фаза обдува | От макс.вакуум – 4 кПа до 4 кПа |
| D | Фаза давления | Меньше чем 4 кПа |
| E | Фаза всасывания | A+B |
| F | Фаза расслабления | C+D |

На рисунке 3.2 представлены отдельные фазы пульсаций.



Рисунок 3.2 – Фазы пульсаций пульсатора доильной установки согласно ISO 3918 – 2007

На горизонтальной оси графика – время, на вертикальной – абсолютное значение вакуумметрического давления (кПа). Каждая из фаз цикла пульсации представляется либо в миллисекундах, либо в процентах относительно общего цикла пульсации:

А (%) = А/(А+B+C+D);

B (%) = B/(А+B+C+D);

C (%) = B/(А+B+C+D);

D (%) = D/(А+B+C+D);

E (%) = (A+B)/(А+B+C+D);

F (%) = (C+D)/(А+B+C+D).

Число пульсаций – расчетное число пульсаций N в минуту:

N = 60000/(A+B+C+D),

где A, B, C и D в миллисекундах.

3.1.3 Функции устройства:

Устройство предназначен для обеспечения определения технических параметров доильных установок и индикации параметров в реальном времени:

- вакуумметрического давления в контрольных узлах доильной системы.

- флуктуации давления на доильных стаканах

Основной функцией прибора является измерение вакуумметрического давления в диапазоне от 0 до 80 кПа. Преимущества измерения данных в реальном времени:

- скорость измерения

- индикация случайных выбросов вакуума

# **4 РАЗРАБОТКА ПРОГРАМНОГО СРЕДСТВА**

## 4.1 Описание механизма устройства и алгоритма вычисления фаз

Устройство всегда находится в режиме ожидания поступающих команд по Bluetooth. После получение команды «начать измерение» происходит включение датчиков и начало периодического измерения и оцифровки полученных данных датчиков, команда «закончить измерение» останавливает периодическое измерение и отключает датчики давления в цели минимизировать энергозатраты. За работу с Bluetooth отвечает класс BluetoothBase.

Оцифрованные данные передаются на телефон где они декодируются и записываются в один из буферных массивов. Каждые 100мс по данным массива строится график и производится расчёт параметров пульсации и давления. За декодирование данных и их запись в буферные массивы отвечает функция adcDataSpleat (String s) класса adcGraph. За построение графика и расчета времени дискретизации отвечает функция adcGraphUpdate() класса adcGraph.

При расчёте пульсации всегда обновляется минимальное и максимальное значение вакуума. Для определения параметров пульсации сначала ищется точка перехода (начала пульса) где вакуум начинает превышать 4 кПа рисунок 3.2, точка не будет принята пока 3 значения подряд не будут превышает 4 кПа, это необходимо для отсечки выбросов вакуума или ошибки измерения, что повлияет на последующие измерения фаз.

После того как мы нашли точку перехода мы начинаем расчёт фазы E. Длительность фазы E будет увеличиваться до тех пор, пока не будет найдена точка спада где значение вакуума меньше Pmax – 4 кПа. После этого рассчитываются фазы А и В обратным перебором буферного массив где ищутся точки согласно таблице 3.1. Далее находим фазу С, F и D. Обнаружение фазы F говорит о окончание анализируемого пульса начало следующего.

После того как найдена фаза F обновляются параметры пульсации на экране и заносятся в буфер до возможности последующего сохранения на телефон.

Анализ параметров приведен в функции find\_fluctuations (double ch, int s\_ID) класса adcGraph.

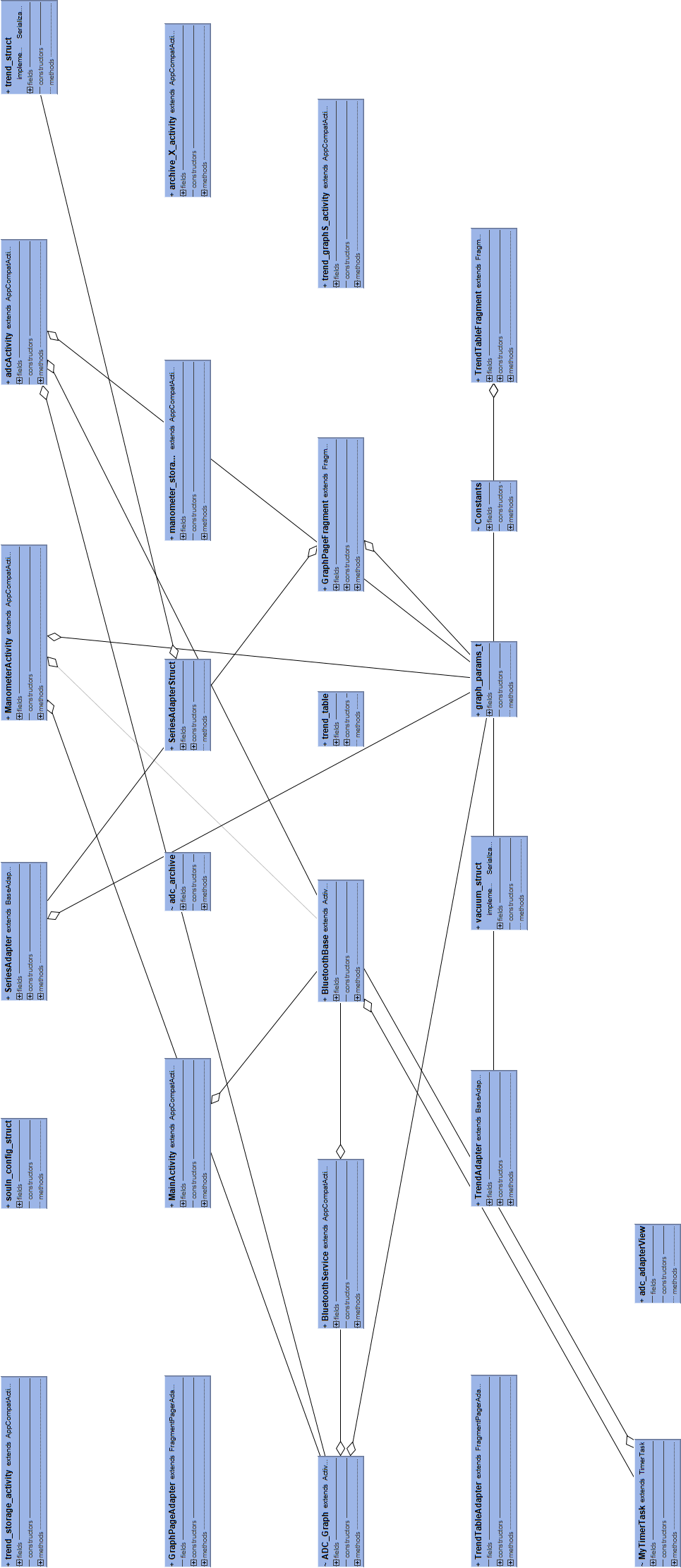


Рисунок 4.1 – UML диаграмма классов

## 4.2 Подключение Bluetooth на Android

Для разрешения использовать функции Bluetooth, в манифест файла добавлено разрешение Bluetooth BLUETOOTH. Это разрешение требуется для выполнения любой связи Bluetooth, такой как запрос на соединение, прием соединения и передача данных. Для разрешения обнаружения устройств и манипулировало настройками Bluetooth, в манифест файла добавлено разрешение BLUETOOTH\_ADMIN. Пример разрешений предоставлен ниже.

<uses-permission android:name="android.permission.BLUETOOTH" />  
<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN" />

Так же для того, чтобы приложение было доступно для устройств, Bluetooth в манифест файла добавлено разрешение см. ниже.

<uses-feature android:name="android.hardware.bluetooth\_le"  
android:required="false" />

Так как BLE связаны с местоположением, для использования BluetoothLeScanner без фильтра, в манифест файла добавлены разрешения ACCESS\_COARSE\_LOCATION и ACCESS\_FINE\_LOCATION без этих разрешений сканирование не вернет никаких результатов. Пример разрешений предоставлен ниже.

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />  
<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

Разрешение на использования место положения программы можно установить в настройках телефона: Настройки – Разрешения – Разрешения приложений – Местоположения – Название программы(ble\_std).

При отключенном Bluetooth адаптере программа выдаст запрос на включение адаптера в диалоговом окне. Код запроса виден ниже.

Intent enableBtIntent = new Intent(android.bluetooth.BluetoothAdapter.ACTION\_REQUEST\_ENABLE);  
startActivityForResult(enableBtIntent, REQUEST\_ENABLE\_BT);

## 4.3 Работа с архивами

Использование устройства предполагает 2 вида использование устройства:

- Личное использование

- Использование с поддержкой сервисной службы

При личном использовании сохранение будет настроено в папку документы archive\_trends в которой будут сохранены архивы с наименованием ферм с возможностью открытия архива на телефоне или компьютере, так же будет доступна настройка синхронизации папки телефона с облаком для удобства.

При использовании с поддержкой сервисной службы сохранение будет настроено в папку документы archive\_x в которой будут сохранены подшефные хозяйства в которых будут храниться архивы с наименованием ферм с возможностью открытия архива на телефоне или компьютере, так же будет настроена синхронизация папки с облаком сервисной службы. Данные загружены в облако будут обрабатываться программой 2, анализировать данные и помечать неправильно работающие по данному подшефному хозяйству данной фермы данного рабочего места, для дальнейшей связи клиентом и устранение проблем, что будет минимизировать затраты и оптимизировать рабочее время.

Методы работы с архивами описаны в классе adc\_archive.

## 4.4 Описание классов

*BluetoothBase* основной класс работы с Bluetooth, отвечает за:

- поиск устройств и передачу списков устройств в активити - поиск устройств произведен с помощью BroadcastReceiver[[3]](#_Список_использованных_источников)

- работу с GATT - подключение, поиск сервисов, поиск характеристик сервисов, чтение атрибутов, установка нотификации характеристики и передачу принятых данных с устройства, адаптация данных отправки (20 байт слово), отправка данных на устройство)

- индикацию подключения

- установку таймера.

Взаимодействие с активностями происходит при помощи Handler message.

*ADC\_Graph* класс отвечающий за:

- декодирование

-запись данных в буферные массивы

- калибровка уровня

- расчёт времени дискретизации - время дискретизации рассчитывается по среднему с 10 значений принятых точек в 100мс

- расчёт параметров пульсации

- построение графиков при измерении.

*Constants* класс отвечающий за:

- хранит ID сообщений посылаемых Hendler

- хранит команды, допуска. И тд.

Graph\_params\_t класс ID параметров трендов.

Vacuum\_struct структура сохранения измереного вакуума в файл.

Trend\_struct структура сохранения параметров тренда в файл.

Souln\_config\_struct структура хранения калибровочных значений.

SeriesAdapter адаптер построения графиков архива.

TrendAdapter адаптер построения таблиц. Особенность значения вне допуска будут подсвечены желтым +- 10%, красным +-20%.

GraphView[[4]](#_Список_использованных_источников) библиотека для работы с графиками.

# **5 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА**

## 5.1 Тестирование различных ситуаций программного взаимодействия

Тест 1

**Тестовая ситуация:**

Пользователь открыл приложение и нажимает кнопку ПОИСК УСТРОЙСТВ для обнаружения устройств.

**Набор данных:**

Нажата кнопка Поиск устройств.

**Ожидаемый результат:**

Должен вывести список всех доступных устройств.

**Фактический результат:**

Выводит список доступных устройств.

Тест 2

**Тестовая ситуация:**

Попытка соединения с необходимым устройством.

**Набор данных:**

Нажато имя устройства на списке всех доступных устройств.

**Ожидаемый результат:**

Попытка соединения должна пройти успешна, должно открыться меню работы с устройствами

**Фактический результат:**

Попытка соединения успешна, открылось меню работы с устройствами.

Тест 3

**Тестовая ситуация:**

Меню работы с устройствами. Необходимо открыть график Анализ пульсации.

**Набор данных:**

Нажата кнопка Анализ пульсации.

**Ожидаемый результат:**

Должен открыться график анализа пульсации.

**Фактический результат:**

Открывается график пульсации.

Тест 4

**Тестовая ситуация:**

Меню работы с устройствами. Необходимо открыть график Манометр.

**Набор данных:**

Нажата кнопка Манометр.

**Ожидаемый результат:**

Должен открыться график манометра.

**Фактический результат:**

Открывается график манометра.

Тест 5

**Тестовая ситуация:**

Меню работы с устройствами. Необходимо открыть таблицу Тренд.

**Набор данных:**

Нажата кнопка Открыть тренд.

**Ожидаемый результат:**

Должна открыться таблица значений тренда.

**Фактический результат:**

Открывается таблица значений тренда.

Тест 6

**Тестовая ситуация:**

Меню работы с устройствами. Необходимо открыть График тренда.

**Набор данных:**

Нажата кнопка Открыть график тренда.

**Ожидаемый результат:**

Должен открыться график тренда.

**Фактический результат:**

Открывается график тренда.

Тест 7

**Тестовая ситуация:**

Меню работы с устройствами. Необходимо открыть таблицу Вакуум.

**Набор данных:**

Нажата кнопка Открыть вакуум.

**Ожидаемый результат:**

Должна открыться таблица значений вакуума.

**Фактический результат:**

Открывается таблица значений вакуума.

Тест 8

**Тестовая ситуация:**

Меню работы с устройствами. Необходимо открыть таблицу Вакуум.

**Набор данных:**

Нажата кнопка Открыть вакуум.

**Ожидаемый результат:**

Должна открыться таблица значений вакуума.

**Фактический результат:**

Открывается таблица значений вакуума.

Тест 9

**Тестовая ситуация:**

Меню работы с устройствами. Необходимо открыть другой архив.

**Набор данных:**

Нажата кнопка “<” или “>”.

**Ожидаемый результат:**

Должно сменится имя архива в поле состояния, если существуют другие архивы.

**Фактический результат:**

Имя архива в поле состояния изменено.

Тест 10

**Тестовая ситуация:**

Меню работы с устройствами. Необходимо открыть архив другого рабочего места.

**Набор данных:**

Нажата кнопка “<” или “>”.

**Ожидаемый результат:**

Должно сменится имя архива рабочего места в поле состояния, если существуют другие архивы рабочих мест.

**Фактический результат:**

Имя архива рабочего места в поле состояния изменено.

Тест 11

**Тестовая ситуация:**

Главное меню. Необходимо открыть меню Архива изменений.

**Набор данных:**

Нажата кнопка АРХИВ ИЗМЕНЕНИЙ.

**Ожидаемый результат:**

Должно перейти в меню Архив изменений.

**Фактический результат:**

Открылось меню Архив изменений.

Тест 12

**Тестовая ситуация:**

Меню работы с устройствами. Необходимо создать новый архив.

**Набор данных:**

Нажато поле состояния имени архива.

**Ожидаемый результат:**

Должно открыться диалоговое окно создания нового архива. После ввода имени и нажатия кнопки ОК, создаётся новый архив.

**Фактический результат:**

Открылось диалоговое окно создания нового архива. После ввода имени архива и нажатия кнопки ОК, создался новый архив.

#### 5.2 Заключение по тестированию программного средства

Исходя из полученных результатов можно сделать заключение, что приложение работает корректно, что позволяет его использовать в реальных условиях.

# **6 РУКОВОДСТВО ИСПОЛЬЗОВАНИЯ ПРИЛОЖЕНИЯ**

## 6.1 Общие сведения об управлении прибором

Управление работой прибора осуществляется с помощью программы ble\_std. Выбор конкретной функции прибора осуществляется с помощью системы меню и подменю. Пользователь выбирает, какие параметры доильной установки он собирается измерять, выходит на соответствующий экран, проводит измерения и сохраняет их в памяти телефона в папку документы/archive\_trends/название фермы/ под заранее выбранным номером. Для упрощения работой измерение некоторых функций сопровождается текстовыми подсказками, выводимыми на экран. При ошибках оператора прибор сообщает об этом путем вывода на экран соответствующих сообщений.

На рисунке 6.1 изображен прибор измерения вакуума. Прибор включается однократным нажатием на кнопку 1, при включении прибора кнопка подсветится жёлтым цветам. На лицевой панели также присутствуют светодиоды. Светодиод 2 показывает состояние подключения измерения вакуума, светодиод 3 индицирует передачу данных. 4 - Каналы подключения вакуума.

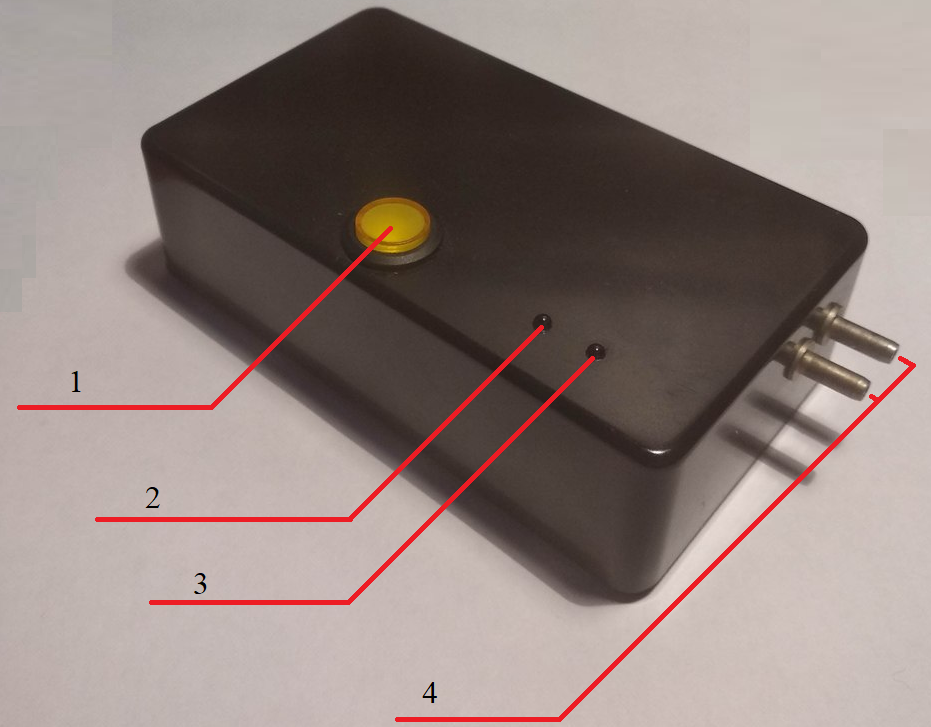


Рисунок 6.1 – Прибор измерения

## 6.2 Описание меню программы ble\_std

При загрузке программы поиск устройств начнется автоматически, при выключенном Bluetooth адаптере программа предоставит запрос включения Bluetooth.

На рисунке 6.2 изображено главное меню. Строка состояние показывает состояние подключения. Под строкой состояния находятся кнопки «поиск устройств» и «архив измерений».

Кнопка «поиск устройств» предназначена для очистки списка устройств и перезагрузки поиска устройств.

При нажатии кнопки «архив измерения» будет установлено окно «архив измерений», рисунок 6.4, для просмотра сохраненных данных.

Для подключения устройства необходимо нажать на нужное имя в списке найденных устройств. При нажатии программа попытается установить соединение с устройством. При успешном соединении будет установлено окно «работы с устройством» рисунок 6.3.

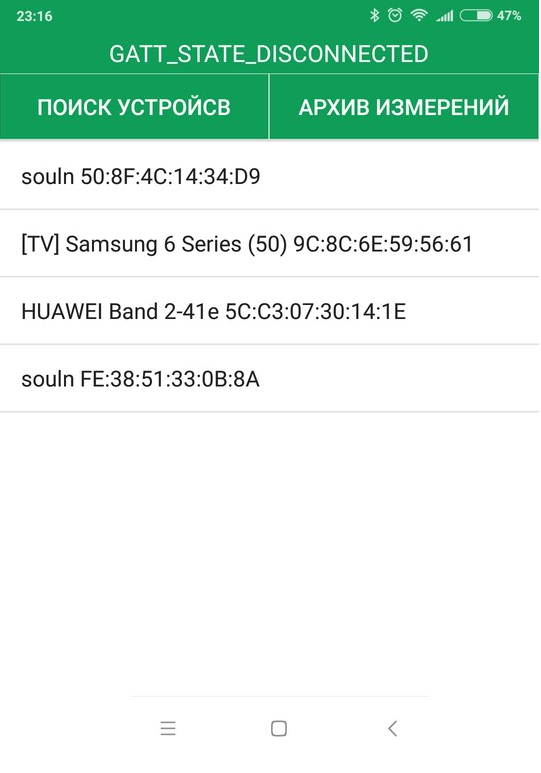


Рисунок 6.2 – главное меню

## 6.3 Описание окна «работа с устройством»

На рисунке 6.3 изображено окно «работа с устройством». Для создание нового архива нажать на поле «архив» находящееся под строкой состояния подключения, после чего подменю «создание архива» предложит создать новый архив, для выбора архива из списка уже имеющихся использовать кнопки « < » и « > » расположенной на поле «архив». Выбор рабочего место изменять нажатиями кнопок « < » и « > » находящееся на поле « рабочее место ». Для перехода в окно «анализ пульсации» и начала измерений параметров пульсации нажать «анализ пульсации». Для перехода в окно «манометр» и начала измерений давления нажать «манометр». Для перехода в окно «параметры пульсации» и просмотра параметров нажать «открыть тренд». Для перехода в окно «графики» и просмотра графиков нажать «открыть график тренда». Для перехода в окно «контрольные точки» и просмотра значений вакуума в контрольных точках «открыть вакуум».

При возможной потери связи с устройством программа попытается автоматически заново подключится к устройству 3 раза.

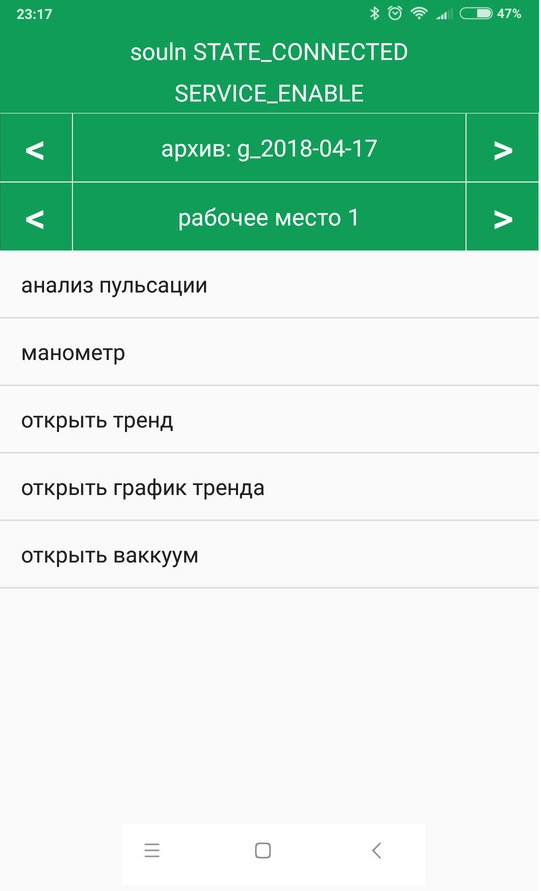


Рисунок 6.3 - окно «работа с устройством»

## 6.4 Описание окна «архив измерений»

На рисунке 6.4 изображено окно «работы с устройством». Работа с окном аналогична с работой окно «работа с устройством» описанным в пункте 6.3.

Особенность - установка папки синхронизации для личного пользования или с поддержкой сервисной службы. При нажатии 10 раз на заголовок окна будет предоставлено диалоговое окно для ввода имени организации и пароля на изменение рисунок 6.4

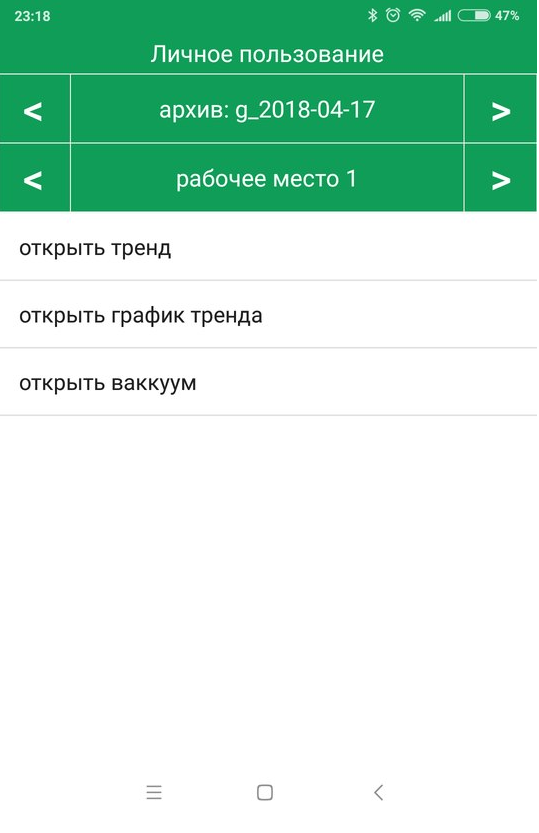


Рисунок 6.4 - окно «работы с устройством»

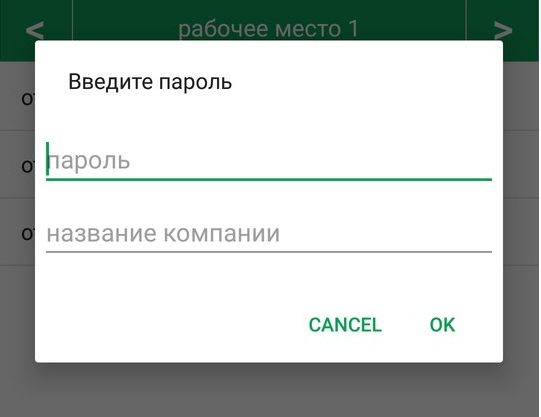


Рисунок 6.5 – диалоговое окно изменения параметров синхронизации

## 6.5 Описание окна «параметры пульсации»

На рисунке 5 изображено окно «параметры пульсации». В заголовке окна показано название выборного архива. В данном окне предоставлены параметры пульсации в табличном виде, 2 столбец канала 1, 3 столбец канала 2. Параметры вне допуска подсвечены зеленым (отклонение в приделах 10%), желтым зеленым (отклонение в приделах 20%) и красным (отклонение > 20%), для просмотра следующего рабочего места перелистнуть экран влево, обратно – в право.

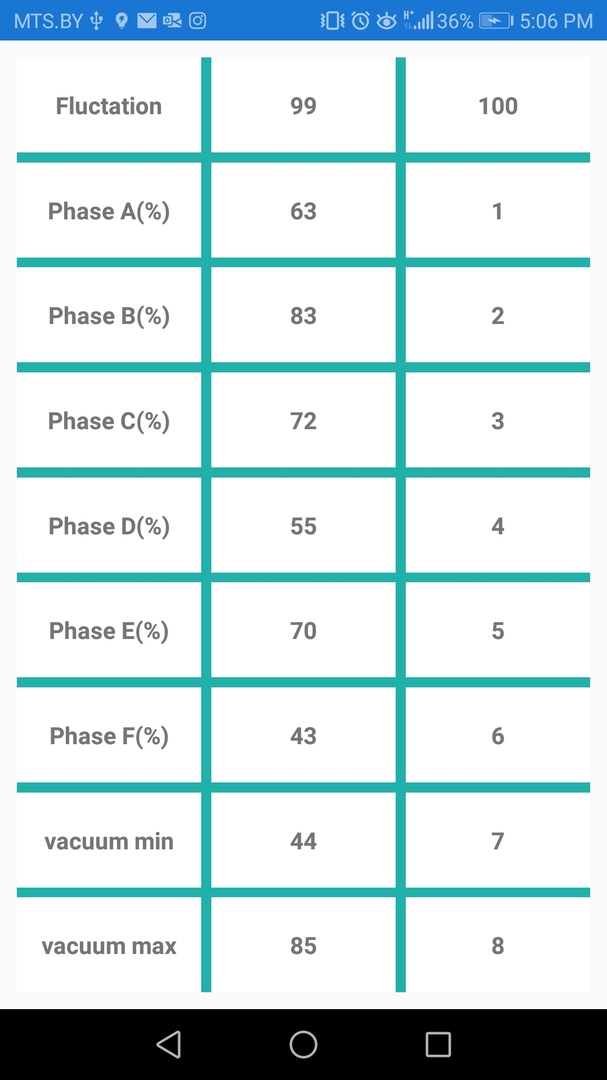


Рисунок 6.6 - окно «параметры пульсации».

## 6.6 Описание окна «графики»

На рисунке 6 изображено окно «графики». В заголовке окна показано название выборного архива. В данном окне предоставлены графики каналов 1 и 2 и рассчитанные параметры пульсации к ним. Зелеными треугольниками обозначены точки начала расчета фаз. для просмотра следующего рабочего места перелистнуть экран влево, обратно – в право.

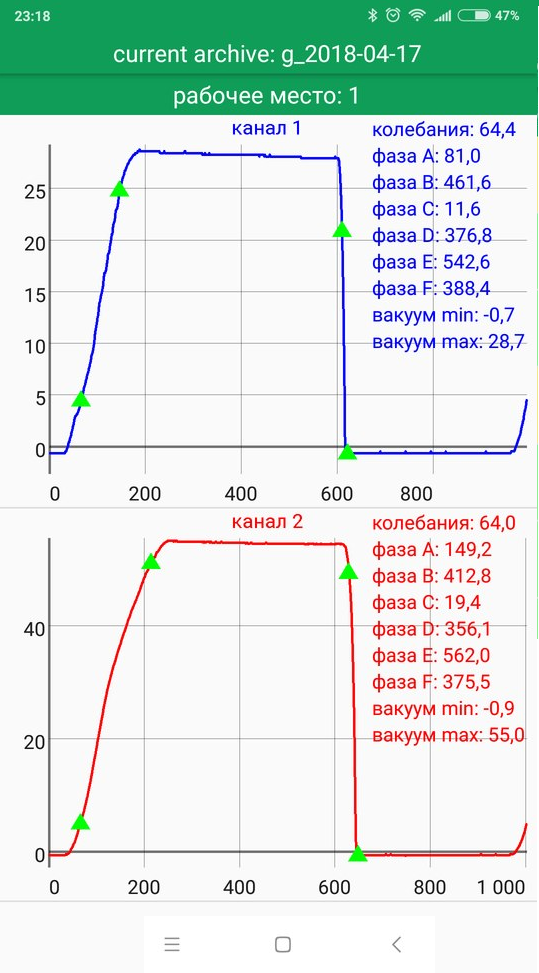


Рисунок 6.7 - окно «графики».

## 6.7 Описание окна «манометр»

На рисунке 6.7 изображено окно «манометр». В заголовке окна показано название выборного архива. В данном окне показаны значения вакуума в контрольных точках системы.

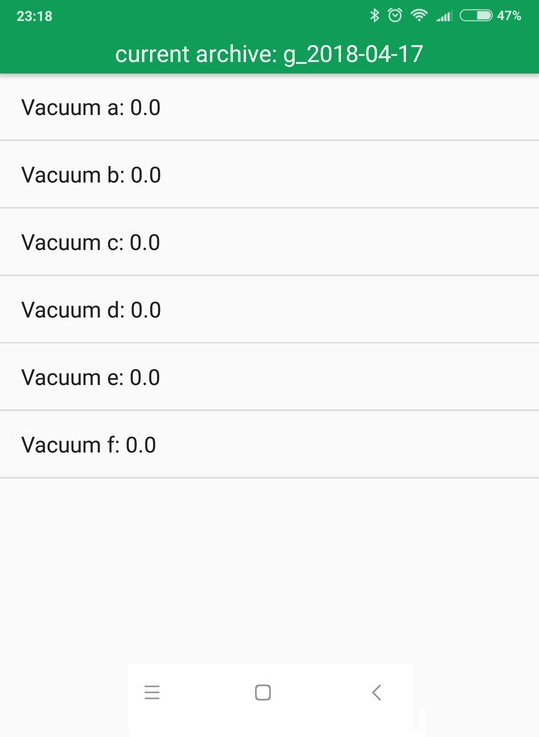


Рисунок 6.8 - окно «манометр»

## 6.8 Описание диалогового окна «создание архива» окна «работа с устройством»

На рисунке 6.9 изображено диалоговое окно «создание архива». В данном архиве программа предлагает создать архив для создания архива нужно ввести его имя и нажать кнопку «ok». Для удобства последующей работы с архивами в поле название архива добавлена текущая дата.

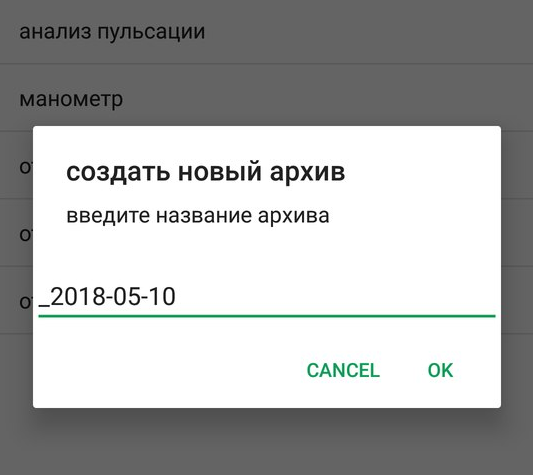


Рисунок 6.9 - диалоговое окно «создание архива»

# **Заключение**

В результате данного курсового проекта было разработано приложение, с помощью которого мы можем получать данные по беспроводной связи bluetooth от датчика, который снимает значения разности вакуума в доильных установках.

Благодаря способу разработки имеется возможность усовершенствовать приложение, например, что планируется в будущем, это отправка полученных данных на удалённый сервер, где далее будет происходить синхронизация со станционарными компьютерами.

В данном курсовом проекте были пройдены все этапы разработки программного средства, включая анализ поставленной задачи, проектирование приложения и его тестирование. В результате мы получили готовое работоспособное приложение для телефонов на базе Android.

Стоит заметить, что аналогов данного приложения в мире очень мало, а в Беларуси существует только отдельное устройство ППДУ, которое довольно габаритное и не легко в использовании. Данный курсовой проект решает проблему габаритов и упрощает работу с проверкой значений разности вакуума.

# **Список использованных источников**

1. Документация по работе с Bluetooth LE [Электронный ресурс]. – Режим доступа: https://developer.android.com/guide/topics/connectivity/bluetooth-le;

2. Документация по ППДУ [Электронный ресурс]. – Режим доступа:

http://luch.by/others/pribor-proverki-doilnykh-ustanovok-ppdu-01-.html;

3. Документация по работе с библиотекой GraphView по построению графиков [Электронный ресурс]. – Режим доступа: https://github.com/jjoe64/GraphView-Demos;

4. Документация по работе с перехватами событий телефона на базе Android [Электронный ресурс]. – Режим доступа: https://developer.android.com/reference/android/content/BroadcastReceiver

# **Приложение А. Исходный код основного класса программы**

public class BluetoothBase extends Activity {

public static ArrayList<BluetoothDevice> mBluetoothDevice = new ArrayList<BluetoothDevice>();

public static Handler mHandler;

public static BluetoothGatt mBluetoothGatt;

private static BluetoothGattService UART\_Service;

private static BluetoothGattCharacteristic UART\_Service\_RX\_CHARACTERISTYC;

private static BluetoothGattCharacteristic UART\_Service\_TX\_CHARACTERISTYC;

public static int mConnectionState = Constants.GATT\_STATE\_DISCONNECTED;

public static String CharacteristicValue;

private static List<BluetoothGattService> ServiceList;

public ArrayList<String> DiscoverBluetooth = new ArrayList<String>();

public BluetoothAdapter BluetoothAdapter = getDefaultAdapter();

public static Timer mTimer;

public static MyTimerTask mMyTimerTask;

static int timerTic = 0;

public boolean BluetoothAdapterInit()

{

if (BluetoothAdapter != null)

{

if (!BluetoothAdapter.isEnabled())

{

HandlerMessageSet(Constants.ADAPTER\_SUCCESSFUL\_INIT);

return true;

}

else

{

HandlerMessageSet(Constants.ADAPTER\_DISABLE);

return false;

}

}

else {

HandlerMessageSet(Constants.ADAPTER\_NOT\_FOUND);

return false;

}

}

public BroadcastReceiver mReceiver = new BroadcastReceiver() {

public void onReceive(Context context, Intent intent) {

String action = intent.getAction();

if (BluetoothDevice.ACTION\_FOUND.equals(action)) {

BluetoothDevice Device = intent.getParcelableExtra(BluetoothDevice.EXTRA\_DEVICE);

if (DeviceAdd(Device)) {

DiscoverBluetooth.add(Device.getName() + " " + Device.getAddress());

HandlerMessageSet(Constants.DEVICE\_FIND);

}

}

}

};

@Override

protected void onDestroy() {

super.onDestroy();

// Don't forget to unregister the ACTION\_FOUND receiver.

unregisterReceiver(mReceiver);

}

public boolean DeviceAdd(BluetoothDevice Device) {

if (mBluetoothDevice.size() > 0) {

for (int i = 0; i < mBluetoothDevice.size(); i++) {

if (Device.getAddress().equals(mBluetoothDevice.get(i).getAddress())) {

return false;

}

}

mBluetoothDevice.add(Device);

return true;

} else {

mBluetoothDevice.add(Device);

return true;

}

}

public void DeviceClear(){

mBluetoothDevice.clear();

DiscoverBluetooth.clear();

//Constants.ConnectionScrinClear();

ViewConnectionState();

HandlerMessageSet(Constants.DEVICE\_FIND);

}

public void ViewConnectionState(){

switch (mConnectionState){

case Constants.GATT\_STATE\_DISCONNECTED:

HandlerMessageSet(Constants.GATT\_STATE\_DISCONNECTED);

break;

case Constants.GATT\_STATE\_CONNECT:

HandlerMessageSet(Constants.GATT\_STATE\_CONNECT);

break;

case Constants.GATT\_STATE\_CONNECTED:

HandlerMessageSet(Constants.GATT\_STATE\_CONNECTED);

break;

}

}

public void GattConnect(int DeviceID){

if (mConnectionState != Constants.GATT\_STATE\_DISCONNECTED) mBluetoothGatt\_cancel();

if (mBluetoothDevice.get(DeviceID).getType() == Constants.DEVICE\_TYPE\_LE) {

BluetoothAdapter.cancelDiscovery();

Constants.LastDevice = mBluetoothDevice.get(DeviceID);

HandlerMessageSet(Constants.GATT\_STATE\_CONNECT);

mBluetoothGatt = mBluetoothDevice.get(DeviceID).connectGatt(this, false, mGattCallback);

} else HandlerMessageSet(Constants.DEVICE\_NOT\_SUPPORTED);

}

public static final BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {

@Override

public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {

if (newState == BluetoothProfile.STATE\_CONNECTED) {

mConnectionState = Constants.GATT\_STATE\_CONNECTED;

Constants.NumberTryConnectClear();

Constants.LastBluetoothGatt = mBluetoothGatt;

mBluetoothGatt.discoverServices();

} else if (newState == BluetoothProfile.STATE\_DISCONNECTED) {

mConnectionState = Constants.GATT\_STATE\_DISCONNECTED;

HandlerMessageSet(Constants.GATT\_STATE\_DISCONNECTED);

}

}

@Override

public void onServicesDiscovered(BluetoothGatt gatt, int status) {

if (status == BluetoothGatt.GATT\_SUCCESS) {

ServiceList = gatt.getServices();

Constants.NumberTryConnectClear();

if(VerifyService())HandlerMessageSet(Constants.SERVICES\_DISCOVERED);

else HandlerMessageSet(Constants.SERVICES\_NOT\_SUPPORTED);

} else {

HandlerMessageSet(Constants.GATT\_ERROR);

}

}

@Override

// Characteristic notification

public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic) {

try {

if (ADC\_Graph.adcStatus) ADC\_Graph.adcDataSpleat(new String(characteristic.getValue(), "UTF-8"));

else CharacteristicValue = new String(characteristic.getValue(), "UTF-8");

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

//HandlerMessageSet(Constants.RX\_DONE);

}

@Override

public void onCharacteristicWrite(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic, int status) {

super.onCharacteristicWrite(gatt, characteristic, status);

// Try to send some data to the device

}

};

public static boolean PacketSend(String data){

if (mConnectionState == Constants.GATT\_STATE\_CONNECTED) {

UART\_Service\_TX\_CHARACTERISTYC.setValue(data);

mBluetoothGatt.writeCharacteristic(UART\_Service\_TX\_CHARACTERISTYC);

return true;

}

else return false;

}

public String Get\_RX\_Packet()

{

return CharacteristicValue;

}

private static boolean VerifyService(){

for(int i = 0; i < ServiceList.size(); i++){

if (Constants.UART\_SERVICE.toString().equals(ServiceList.get(i).getUuid().toString())){

UART\_Service = ServiceList.get(i);

return true;

}

}

return false;

}

private boolean VerifyCharacteristic(BluetoothGattService Service)

{

/\*

boolean RX\_Characteristic = false;

boolean TX\_Characteristic = false;

for(int i = 0; i < Service.getCharacteristics().size(); i++)

{

if (Service.getCharacteristics().get(i).getUuid().toString().equals(RX\_DATA\_CHARACTERISTIC.toString()))

{

RX\_Characteristic = true;

}

else

{

if (Service.getCharacteristics().get(i).getUuid().toString().equals(TX\_DATA\_CHARACTERISTIC.toString()))

{

TX\_Characteristic = true;

}

}

}

if(RX\_Characteristic && TX\_Characteristic) return true;

else return false;

\*/

return true;

}

public void NotificationEnable(int ServiceID, int CharacteristicID, boolean enable){

BluetoothGattCharacteristic Characteristic = ServiceList.get(ServiceID).getCharacteristics().get(CharacteristicID);

for (int i = 0; i < Characteristic.getDescriptors().size(); i++){

if(setNotification(Characteristic)){

return;

}

}

HandlerMessageSet(Constants.NOTIFICATION\_SERVICE\_NOT\_SUPPORTED);

}

private boolean setNotification(BluetoothGattCharacteristic Characteristic){

for (int i = 0; i < Characteristic.getDescriptors().size(); i++){

if (Characteristic.getDescriptors().get(i).getUuid().toString().equals(Constants.CHARACTERISTIC\_UPDATE\_NOTIFICATION\_DESCRIPTOR\_UUID.toString()))

{

mBluetoothGatt.setCharacteristicNotification(Characteristic, true);

BluetoothGattDescriptor descriptor = Characteristic.getDescriptor(Constants.CHARACTERISTIC\_UPDATE\_NOTIFICATION\_DESCRIPTOR\_UUID);

descriptor.setValue(BluetoothGattDescriptor.ENABLE\_NOTIFICATION\_VALUE);

mBluetoothGatt.writeDescriptor(descriptor);

HandlerMessageSet(Constants.NOTIFICATION\_SERVICE\_ENABLE);

return true;

}

}

return false;

}

public void UART\_ServiceInit() {

if(mConnectionState == Constants.GATT\_STATE\_CONNECTED) {

UART\_Service\_RX\_CHARACTERISTYC = UART\_Service.getCharacteristic(Constants.RX\_DATA\_CHARACTERISTIC);

UART\_Service\_TX\_CHARACTERISTYC = UART\_Service.getCharacteristic(Constants.TX\_DATA\_CHARACTERISTIC);

setNotification(UART\_Service\_RX\_CHARACTERISTYC);

}

}

public static void HandlerMessageSet(int MessageID){

Message m = new Message();

m.what = MessageID;

mHandler.sendMessage(m);

}

public ArrayList<String> GetServices(){

if (mConnectionState == Constants.GATT\_STATE\_CONNECTED) {

ArrayList<String> Service = new ArrayList<String>();

for (int i = 0; i < ServiceList.size(); i++) {

Service.add(ServiceList.get(i).getUuid().toString());

}

return Service;

}

else return null;

}

public ArrayList<String> GetCharacteristic(int ServiceID){

List<BluetoothGattCharacteristic> CharacteristicsList = ServiceList.get(ServiceID).getCharacteristics();

ArrayList<String> Characteristic = new ArrayList<String>();

for (int i = 0; i < CharacteristicsList.size(); i++){

Characteristic.add(CharacteristicsList.get(i).getUuid().toString());

}

return Characteristic;

}

public void testX(TextView t){

t.setText("qwerty");

}

static boolean ADC\_start(){

if (!ADC\_Graph.adcStatus) ADC\_Graph.adcStatus = true;

return PacketSend("start\_adc");

}

static boolean ADC\_stop(){

if (ADC\_Graph.adcStatus) ADC\_Graph.adcStatus = false;

return PacketSend("stop\_adc");

}

public boolean commandSend(int commandID){

return PacketSend(Constants.ble\_command[commandID]);

}

public void Timer1\_set(int period){

if(MyTimerTask.TIMER1\_ENABLE) mTimer.cancel();

mTimer = new Timer();

mMyTimerTask = new MyTimerTask();

MyTimerTask.TIMER1\_ENABLE = true;

mTimer.schedule(mMyTimerTask, 100, period);

}

public void Timer1\_stop(){

if(MyTimerTask.TIMER1\_ENABLE) {

MyTimerTask.TIMER1\_ENABLE = false;

mTimer.cancel();

}

}

public void mBluetoothGatt\_cancel(){

if (mConnectionState != Constants.GATT\_STATE\_DISCONNECTED){

mBluetoothGatt.disconnect();

}

if (mBluetoothGatt != null) {

mBluetoothGatt.close();

mBluetoothGatt = null;

mConnectionState = Constants.GATT\_STATE\_DISCONNECTED;

}

}

public void service\_re\_connect(){

}

}

# **Приложение B. Исходный код класса для построения графиков**

class ADC\_Graph extends Activity {

private BluetoothBase mBluetoothBase = new BluetoothBase();

static boolean adcStatus = false;

static boolean fluctuation\_analys\_flag = false;

static int[][] ADC\_data\_mass = new int[2][Constants.max\_buff\_size];

static int[] ADC\_data\_mass\_size = new int[2];

static boolean adc\_set\_null\_flag = false;

static boolean last\_adc\_set\_null\_flag = false;

static boolean[] adc\_null\_flag = new boolean[2];

static int[] adc\_calib\_volum = {289, 289};

static int[] adc\_calib\_volumX = {289, 289, 289, 289};

static int pac\_count = 1;

static int[] pac\_count\_mas = new int[12];

static int pac\_average\_value = 0;

static int adcDataSpleat\_n = 1;

static int adcDataSpleat\_n\_j = 1;

static int AsisX = 0;

static int number\_buff = 0;

static int x\_buff;

static double sampling\_Time;

static boolean receptionDataFlag = false;

static boolean adcGraphUpdateFlag = false;

static boolean[] manometerUpdateFlag = new boolean[2];

static boolean[] manometerUpdateFlag\_t = new boolean[2];

static boolean[] series\_fluctuations\_flag = {true,true};

static boolean[] series\_phase\_e\_flag = {false,false};

static boolean[] series\_phase\_f\_flag = {false,false};

static boolean[] series\_phase\_b\_flag = {false,false};

static boolean[] graph\_array\_flag = {true,true};

static double[][] graphX\_params = new double[2][9];

static double[][] graphX\_params\_scrin = new double[2][9];

static double[] fluctuations\_time = new double[2];

static double[] series\_max\_vacuum\_buf = new double[2];

static double[] total\_volume = new double[2];

static int[][] series\_x\_s = new int[2][4];

static int[][] series\_x\_s\_scrin = new int[2][4];

static ArrayList<Double>[] graph\_array\_preBuf = new ArrayList[]{new ArrayList<Double>(), new ArrayList<Double>()};

static ArrayList<Double>[] graph\_array\_buf\_t = new ArrayList[]{new ArrayList<Double>(), new ArrayList<Double>()};

static final int array\_preBuf\_max\_length = 50;

static ArrayList<Double>[] graph\_array = new ArrayList[]{new ArrayList<Double>(), new ArrayList<Double>()};

static ArrayList<Double>[] graph\_array\_scrin = new ArrayList[]{new ArrayList<Double>(), new ArrayList<Double>()};

static int[] series\_filter = new int[2];

static final int num\_filter = 3;

graph\_params\_t g\_P = new graph\_params\_t();

static LineGraphSeries<DataPoint> series;

static LineGraphSeries<DataPoint> series1;

public void ADC\_Graph\_Init(){

for(int i = 0; i < 2; i++) {

graph\_array\_buf\_t[i].clear();

graph\_array\_preBuf[i].clear();

graph\_array[i].clear();

graph\_array\_scrin[i].clear();

graphX\_params[i][g\_P.min\_vacuum] = 50.0;

graphX\_params[i][g\_P.max\_vacuum] = -10.0;

}

ADC\_data\_mass\_size = new int[]{0,0};

series\_fluctuations\_flag = new boolean[]{true, true};

series\_phase\_e\_flag = new boolean[]{false, false};

series\_phase\_f\_flag = new boolean[]{false, false};

graph\_array\_flag = new boolean[]{true, true};

series = new LineGraphSeries<>();

series1 = new LineGraphSeries<>();

AsisX = 0;

adcDataSpleat\_n = 1;

adcDataSpleat\_n\_j = 1;

pac\_average\_value = 0;

}

private boolean series\_filter(int s\_ID){

series\_filter[s\_ID]++;

if(series\_filter[s\_ID] > num\_filter) {

series\_filter[s\_ID] = 0;

return true;

}

else return false;

}

private void find\_fluctuations(double ch, int s\_ID){

if (ch < graphX\_params[s\_ID][g\_P.min\_vacuum]) {

graphX\_params[s\_ID][g\_P.min\_vacuum] = ch;

}

else if (ch > series\_max\_vacuum\_buf[s\_ID]) series\_max\_vacuum\_buf[s\_ID] = ch;

if (fluctuation\_analys\_flag) {

fluctuations\_time[s\_ID] += sampling\_Time;

if (graph\_array\_flag[s\_ID]) {

if (graph\_array\_preBuf[s\_ID].size() > array\_preBuf\_max\_length) {

graph\_array\_preBuf[s\_ID].remove(0);

graph\_array\_preBuf[s\_ID].add(ch);

} else graph\_array\_preBuf[s\_ID].add(ch);

}

if (series\_fluctuations\_flag[s\_ID] && ch > 3.5) {

if (series\_filter(s\_ID)) {

graphX\_params[s\_ID][g\_P.phase\_f] = fluctuations\_time[s\_ID];

graphX\_params[s\_ID][g\_P.fluctuation] = graphX\_params[s\_ID][g\_P.phase\_f] + graphX\_params[s\_ID][g\_P.phase\_e];

fluctuations\_time[s\_ID] = 0;

graphX\_params[s\_ID][g\_P.max\_vacuum] = series\_max\_vacuum\_buf[s\_ID];

series\_max\_vacuum\_buf[s\_ID] = 0;

save\_trend\_volume(s\_ID);

for (int i = 0; i < graph\_array\_preBuf[s\_ID].size(); i++){

graph\_array\_buf\_t[s\_ID].add(graph\_array\_preBuf[s\_ID].get(i));

}

series\_x\_s[s\_ID][g\_P.fluctuation] = graph\_array\_buf\_t[s\_ID].size();

series\_fluctuations\_flag[s\_ID] = false;

graph\_array\_flag[s\_ID] = false;

series\_phase\_e\_flag[s\_ID] = true;

series\_phase\_b\_flag[s\_ID] = true;

BluetoothBase.HandlerMessageSet(Constants.SERIES0\_EVT[s\_ID]);

}

}

if(graph\_array\_buf\_t[s\_ID] != null){

if (graph\_array\_buf\_t[s\_ID].size() > 3000){

graph\_array\_buf\_t[s\_ID].remove(0);

graph\_array\_buf\_t[s\_ID].add(ch);

}

else graph\_array\_buf\_t[s\_ID].add(ch);

}

if (series\_phase\_e\_flag[s\_ID] && ch < (series\_max\_vacuum\_buf[s\_ID] - 4)){

graphX\_params[s\_ID][g\_P.phase\_e] = fluctuations\_time[s\_ID];

series\_x\_s[s\_ID][g\_P.phase\_b] = graph\_array\_buf\_t[s\_ID].size();

find\_phase\_a(s\_ID);

fluctuations\_time[s\_ID] = 0;

series\_phase\_e\_flag[s\_ID] = false;

series\_phase\_f\_flag[s\_ID] = true;

}

if (series\_phase\_f\_flag[s\_ID] && ch < 3.5) {

if (series\_filter(s\_ID)) {

graphX\_params[s\_ID][g\_P.phase\_c] = fluctuations\_time[s\_ID];

series\_x\_s[s\_ID][g\_P.phase\_c] = graph\_array\_buf\_t[s\_ID].size();

series\_fluctuations\_flag[s\_ID] = true;

series\_phase\_f\_flag[s\_ID] = false;

graph\_array\_flag[s\_ID] = true;

graph\_array\_preBuf[s\_ID].clear();

graphX\_params[s\_ID][g\_P.min\_vacuum] = 50.0;

}

}

}

else {

if (!manometerUpdateFlag[s\_ID]) manometerUpdateFlag[s\_ID] = true;

total\_volume[s\_ID] = ch;

if (manometerUpdateFlag\_t[s\_ID]) {

manometerUpdateFlag\_t[s\_ID] = false;

BluetoothBase.HandlerMessageSet(Constants.SERIES0\_EVT[s\_ID]);

}

}

}

private void find\_phase\_a(int ID){

int max\_size = graph\_array\_buf\_t[ID].size();

int null\_point = graph\_array\_preBuf[ID].size();

int phase\_e\_length = max\_size - null\_point;

double step\_time = graphX\_params[ID][g\_P.phase\_e]/phase\_e\_length;

double phase\_a\_time = 0.0;

for(int i = null\_point; i < max\_size; i++){

phase\_a\_time += step\_time;

if (graph\_array\_buf\_t[ID].get(i) > (series\_max\_vacuum\_buf[ID] - 4)) {

graphX\_params[ID][g\_P.phase\_a] = phase\_a\_time;

series\_x\_s[ID][g\_P.phase\_a] = i;

return;

}

}

}

private void save\_trend\_volume(int s\_ID){

graph\_array[s\_ID].clear();

for (int i = 0; i < graph\_array\_buf\_t[s\_ID].size();i++){

graph\_array[s\_ID].add(graph\_array\_buf\_t[s\_ID].get(i));

}

graph\_array\_buf\_t[s\_ID].clear();

}

public void series\_vacuum\_clear(int s\_ID){

graphX\_params[s\_ID][g\_P.min\_vacuum] = 50.0;

series\_max\_vacuum\_buf[s\_ID] = -10.0;

manometerUpdateFlag[s\_ID] = false;

}

public void adcGraphUpdate(){

x\_buff = number\_buff;

number\_buff = Math.abs(number\_buff - 1);

if (pac\_count > 0) {

if (adcDataSpleat\_n < 11) {

pac\_count\_mas[adcDataSpleat\_n] = pac\_count;

pac\_average\_value += pac\_count;

adcDataSpleat\_n++;

} else {

if (adcDataSpleat\_n\_j > 10) adcDataSpleat\_n\_j = 1;

pac\_average\_value -= pac\_count\_mas[adcDataSpleat\_n\_j];

pac\_count\_mas[adcDataSpleat\_n\_j] = pac\_count;

pac\_average\_value += pac\_count;

adcDataSpleat\_n\_j++;

}

sampling\_Time = 200 / ((double)pac\_average\_value \* (12 - adcDataSpleat\_n));

pac\_count = 1;

}

for (int i = 0; i < ADC\_data\_mass\_size[x\_buff]; i++) {

if (i%2 == 0) {

double ch = (double) (ADC\_data\_mass[x\_buff][i] - adc\_calib\_volum[0])/5.85;

find\_fluctuations(ch, 0);

series.appendData(new DataPoint(AsisX, ch), true, 1250);

}

else {

double ch = (double) (ADC\_data\_mass[x\_buff][i] - adc\_calib\_volum[1])/5.85;

find\_fluctuations(ch, 1);

series1.appendData(new DataPoint(AsisX, ch), true, 1250);

AsisX++;

}

}

ADC\_data\_mass\_size[x\_buff] = 0;

}

private static void adc\_cheek\_null(int ch, int s\_ID){

if (!adc\_null\_flag[s\_ID]) {

while (ch > 0) {

ch = ch - 1;

adc\_calib\_volum[s\_ID] += 1;

}

adc\_null\_flag[s\_ID] = true;

}

}

private static void adc\_set\_null(){

x\_buff = number\_buff;

int ch1 = ADC\_data\_mass[x\_buff][0];

int ch2 = ADC\_data\_mass[x\_buff][1];

adc\_cheek\_null(ch1, 0);

adc\_cheek\_null(ch2, 1);

if (!last\_adc\_set\_null\_flag) {

number\_buff = Math.abs(number\_buff - 1);

ADC\_data\_mass\_size[x\_buff] = 0;

}

BluetoothBase.HandlerMessageSet(Constants.ADC\_NULL\_SETT);

}

public void ADC\_set\_null(){

adc\_set\_null\_flag = true;

if (adcStatus) last\_adc\_set\_null\_flag = true;

else {

last\_adc\_set\_null\_flag = false;

BluetoothBase.ADC\_start();

}

}

static void adcDataSpleat(String s){

// Переносим сюда старый код

receptionDataFlag = true;

pac\_count++;

int packet\_length = s.length();

int count\_ = 0;

for (int i = 0; i < packet\_length ; i++){

if (count\_ == 1){

int exp\_size = 0;

ADC\_data\_mass[number\_buff][ADC\_data\_mass\_size[number\_buff]] = 0;

for (int j = i - count\_; j <= i; j++) {

if (s.charAt(j) != ' '){

ADC\_data\_mass[number\_buff][ADC\_data\_mass\_size[number\_buff]] += ((int) s.charAt(j) - 40) \* Math.pow(10, count\_ - exp\_size);

}

exp\_size++;

}

count\_ = 0;

ADC\_data\_mass\_size[number\_buff]++;

if (ADC\_data\_mass\_size[number\_buff] > Constants.max\_buff\_size) ADC\_data\_mass\_size[number\_buff] = 0;

}

else count\_++;

}

receptionDataFlag = false;

if (adcGraphUpdateFlag){

adcGraphUpdateFlag = false;

BluetoothBase.HandlerMessageSet(Constants.TIMER1\_TIC);

}

if (adc\_set\_null\_flag){

if (!last\_adc\_set\_null\_flag) BluetoothBase.ADC\_stop();

adc\_set\_null\_flag = false;

adc\_set\_null();

}

}

}