муниципальное бюджетное общеобразовательное учреждение

города Новосибирска «Гимназия №16 «Французская»

Предмет Информатика

**Исследование и создание ЭЭГ/ЭМГ тренингов**

Автор проекта:

Плющев Александр Алексеевич,

ученик 10 «В» класса

Руководитель проекта:

Кузнецова Галина Вячеславовна,

Преподаватель высшей квалификационной категории

Работа рекомендована к защите

Подпись руководителя проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\

Новосибирск 2023

Оглавление

[Введение 4](#_Toc134727097)

[Описание задач: 4](#_Toc134727098)

[Актуальность: 4](#_Toc134727099)

[Цель: 5](#_Toc134727100)

[Задачи: 5](#_Toc134727101)

[Исследовательская часть проекта 6](#_Toc134727102)

[Принципы и природа получаемых сигналов: 6](#_Toc134727103)

[Связь сигнала ЭЭГ/ЭМГ и состояния пациента 9](#_Toc134727104)

[Практическая часть проекта 13](#_Toc134727105)

[Получение и интерпретирование данных, отправляемых датчиком 13](#_Toc134727106)

[Обработка получаемых данных в реальном времени 17](#_Toc134727107)

[Создание графического интерфейса тренинга 19](#_Toc134727108)

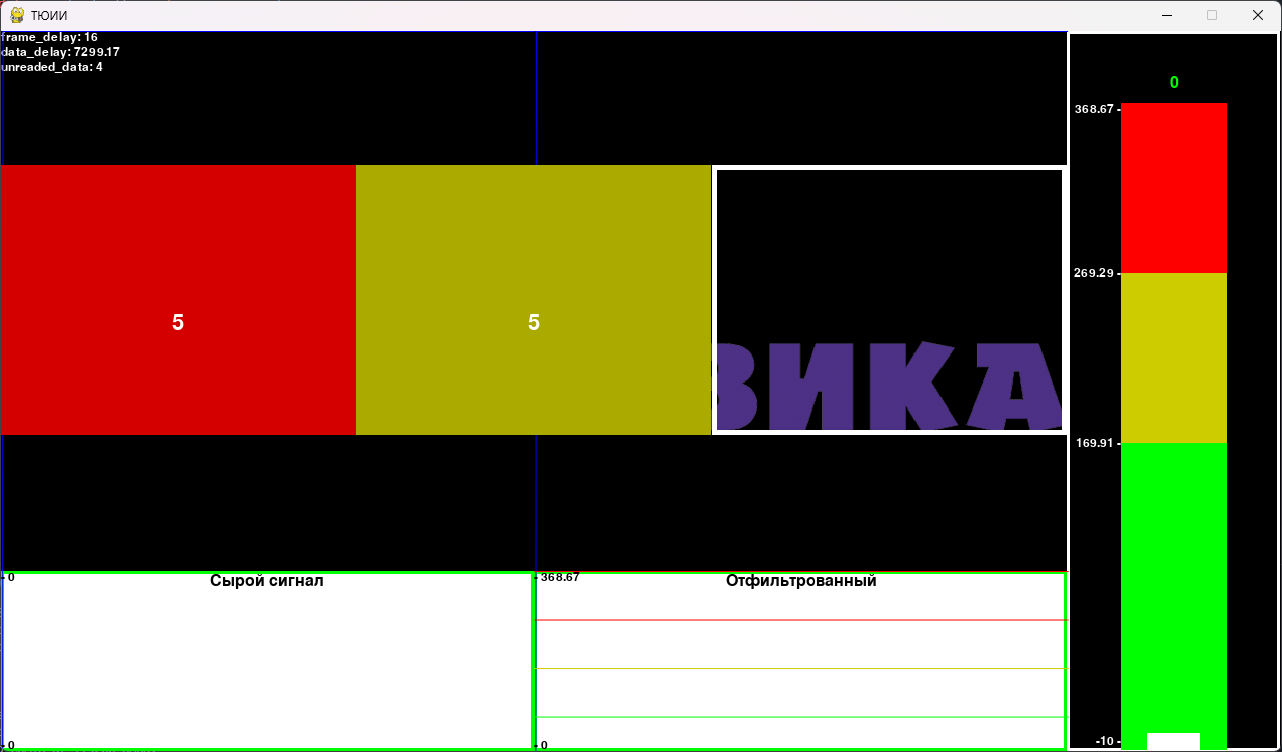
[Блок, содержащий информацию о работе программы 20](#_Toc134727109)

[Плоттер 21](#_Toc134727110)

[Вертикальная гистограмма 22](#_Toc134727111)

[Спрятанное изображение 23](#_Toc134727112)

[Итоговый вариант графического интерфейса 24](#_Toc134727113)

[ 24](#_Toc134727114)

[Вывод 25](#_Toc134727115)

[Список литературы 26](#_Toc134727116)

[Приложение 27](#_Toc134727117)

# Введение

## Описание тренингов:

Тренинг №1 «Радужное настроение» требует создать психологический тренинг, представляющий из себя программу демонстрирующую пациенту амплитуду ритма головного мозга в полосе частот от 8 до 14 Гц (α-ритм). В виде некоторой цветовой полоски, где пациенту необходимо повысить амплитуду α-ритма чтобы изменить цвет этой полосы от фиолетового (при минимальном значении α-ритм) до красного (при максимальном)

Тренинг №2 «Светофор» требует, используя сигнал интегральной электромиографии (далее ЭМГ). Откалибровать уровень сигнала от минимума до максимума, т.е. для каждого канала вычислить диапазон изменения амплитуды сигнала интегральной ЭМГ, разделить его на 3 равные области с цветовой кодировкой: зеленая (низкое напряжение), желтая (среднее), красная (высокое). Если сигнал ЭМГ, регистрируемый с мышцы испытуемого, в течение 30 секунд (суммарно) находится в соответствующей зоне, то открывается выделенный квадратик того же цвета, отображаемый на экране компьютера.

## Актуальность:

Тренинги ЭЭГ и ЭМГ становятся все более популярными в области физиотерапии и спортивной медицины. Было доказанно, что тренировка ЭЭГ помогает улучшить внимание и концентрацию, снизить уровень стресса и беспокойства и даже уменьшить симптомы СДВГ. Кроме того, ЭЭГ можно использовать для тренировки мозга, чтобы лучше модулировать эмоции и повышать общую ясность ума.

ЭМГ-тренинг — отличный способ улучшить мышечную силу и координацию, а также осанку и баланс. Его можно использовать для нацеливания на определенные мышцы или группы мышц для улучшения силы, гибкости и выносливости. ЭМГ также можно использовать для снижения риска травм за счет укрепления слабых мышц и суставов.

В целом, тренинги ЭЭГ и ЭМГ являются очень эффективными инструментами, которые можно использовать для улучшения физического и психического здоровья. Используя эти два метода вместе, практикующие врачи могут максимизировать эффективность своего лечения и добиться наилучших результатов для своих пациентов.

## Цель:

Исследовать принципы и природу сигналов ЭМГ/ЭЭГ, а также их связь с состоянием человека. А также создать два программных тренинга, использующих выбранные датчики, анализирующие сигналы ЭМГ/ЭЭГ и удовлетворяющие требованиям предъявленным в описании тренингов.

## Задачи:

1. Теоретическая часть
   1. Исследовать принципы и природу получаемых сигналов
   2. Понять связь показаний ЭЭГ/ЭМГ и состояния пациента
2. Практическая часть
   1. Получение и интерпретирование данных, отправляемых датчиком
   2. Обработка получаемых данных в реальном времени
   3. Создание графического интерфейса тренинга

# Исследовательская часть проекта

## Принципы и природа получаемых сигналов:

Сигналы ЭМГ (электромиография) и ЭЭГ (электроэнцефалография) возникают в результате электрических изменений в мышцах и мозге, соответственно. Эти сигналы представляют собой электрические потенциалы, которые возникают в результате действия миоэлектрических и нейрональных процессов и могут быть зарегистрированы с помощью соответствующего оборудования.

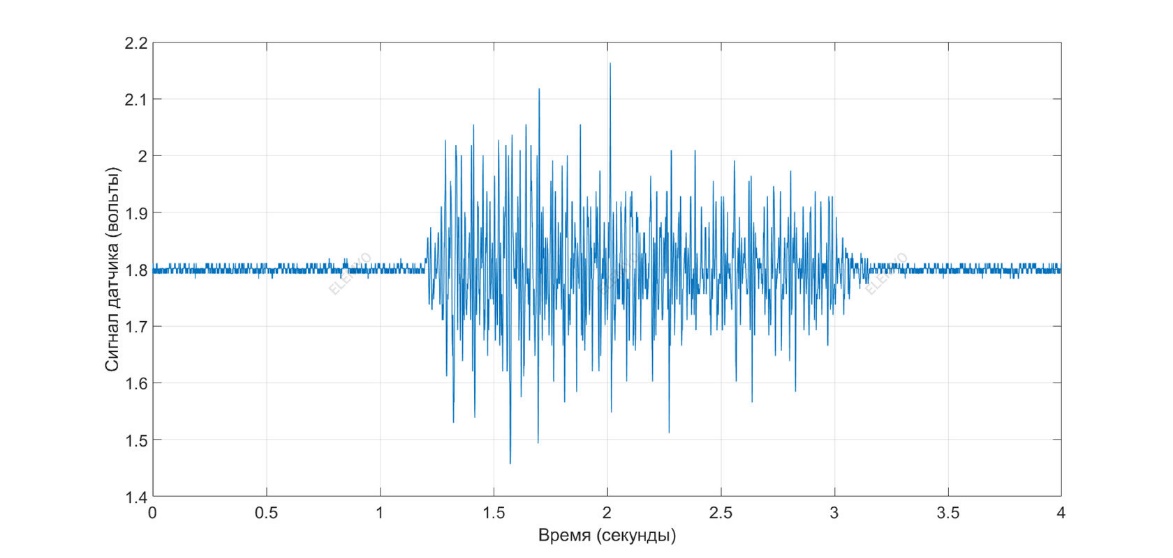


Рисунок 1 Оцифрованный сигнал ЭМГ

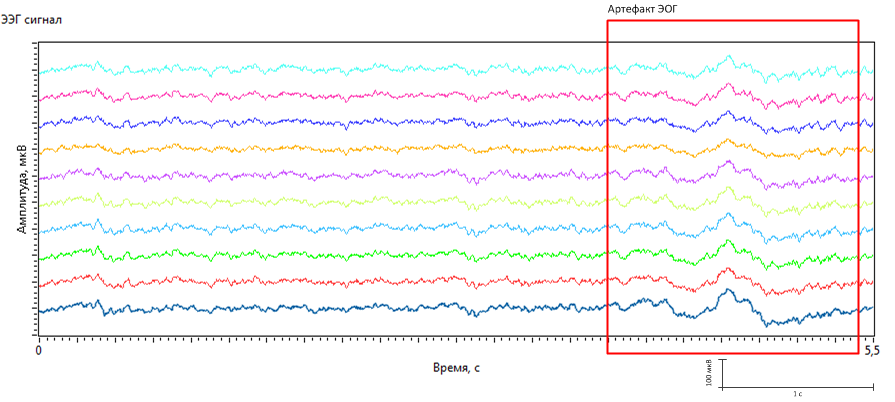
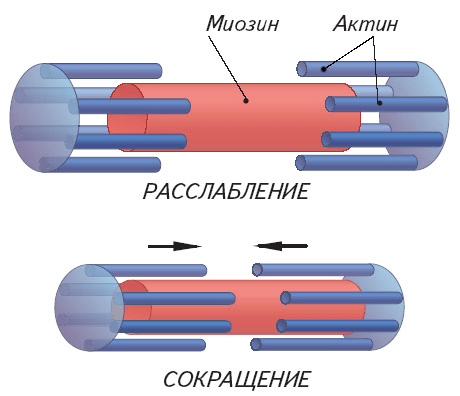


Рисунок 2 Оцифрованный сигнал ЭЭГ по 10 каналам

ЭМГ-сигналы возникают в мышцах в результате сокращения мышечных волокон. 

Процесс сокращения мышцы начинается с электрического импульса, который приходит от нервной системы и достигает конца нервного волокна, приводя к выделению нейромедиатора - ацетилхолина (вещество, осуществляющее передачу электрохимического импульса). Этот нейромедиатор связывается с рецепторами на поверхности мышечной клетки, что вызывает изменение ее внутреннего потенциала. Это вызывает открытие каналов для катионов, калия и кальция. Катионы в свою очередь вызывают деполяризацию мышечной клетки (изменение ее внутреннего потенциала с отрицательного на положительный). (Именно деполяризация, позволяет явно определить начало сокращения мышцы неинвазивным методом). После этого процесса в мышечной клетке появляются активные миофибриллы (специальные белки, которые взаимодействуют друг с другом и приводят к сокращению мышечных волокон). Внутри каждой миофибриллы находятся актиновые и миозиновые филаменты, которые являются основными компонентами мышечного сокращения.  
Когда мышечная клетка сокращается, актиновые и миозиновые филаменты сжимаются друг с другом при помощи быстрых процессов, которые обеспечивают быстрое и мощное сокращение мышцы. На этом этапе происходит уменьшение длины мышечной клетки, что приводит к сокращению самой мышцы.  
  
Объяснение электрической активности мышцы в этот момент связано с деполяризацией клетки, когда мышечная клетка меняет свой внутренний потенциал и возникает электрический заряд, который передается через миофибриллы мышечной клетки и вызывает ее сокращение. Именно этот электрический заряд и называется электрической активностью мышцы во время ее сокращения. И именно это изменение потенциала мы можем измерить, и по нему определить момент и силу сокращения мышцы.

Рисунок 3 Процесс сокращения мышцы

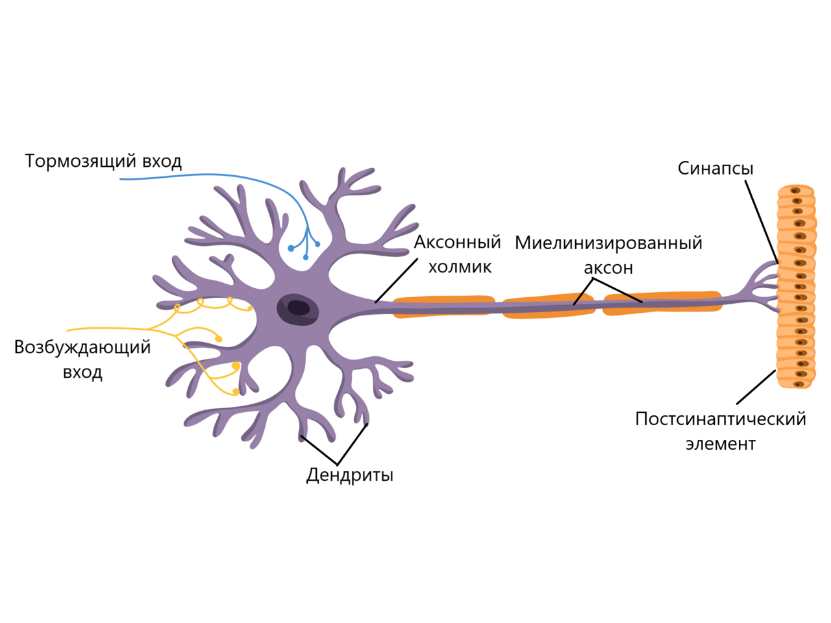
ЭЭГ - сигналы возникают в мозге благодаря электрической активности нейронов. При активности головного мозга, сенсорных восприятий и других процессах, наблюдается изменение электрического потенциала, возникающее в множестве нейронов мозга, которое и формирует электрическую активность. Электрическая активность головного мозга проявляется в виде электроэнцефалограммы (ЭЭГ), которая представляет собой график электрической активности мозга, регистрируемой на коже головы с помощью электродов.  
  
 Нейрон имеет тело клетки, дендриты, аксон и окончания синаптических волокон. При поступлении информации в нейрон, эта информация проходит через дендриты и тело клетки нейрона, вызывая изменение потенциала мембраны. Если этот потенциал достигает порога возбуждения, нейрон образует действительный потенциал действия, который передается вдоль аксона и заканчивается у пучка окончаний синаптических волокон.  
  
Синаптические волокна связывают нейроны между собой и передают информацию между ними. При передаче информации внутри синаптического пространства восстанавливается изменение потенциала мембраны, называемое постсинаптический потенциал. Если достаточное количество постсинаптических потенциалов возникает в клетке, то кроме нейронов возбудятся и окололежащие клетки. Если же все окончания синапсов не продолжат непрерывного возбуждения, мы получим период относительного покоя.  
  
Электрическая активность мозга может быть описана с помощью различных ритмов мозга. Ритмы мозга – это колебания потенциала мозга на разных частотах.   
В частности, существующие ритмы мозга подразделяются на четыре группы:

Рисунок 4 Строение нейрона

1. Дельта-ритмы (0,5-4 Гц). Они характеризуются низкой частотой, встречаются при глубоком сне, гипнозе и глубокой медитации, наблюдается в таламусе.
2. Тета-ритмы (4-8 Гц). Тета-ритмы встречаются в процессе сна, глубокой медитации и концентрации, наблюдается в гиппокампе.
3. Альфа-ритмы (8-12 Гц). Они особенно ярко проявляются в состояние покоя, когда человек не занят непосредственной деятельностью. Наблюдается во время медитации. Активнее всего в затылочной области.
4. Бета-ритмы (14-30 Гц). Эти ритмы возникают в состоянии бодрствования, концентрации, обучения и других активных процессах. Заметен в лобной доле.

## Связь сигнала ЭЭГ/ЭМГ и состояния пациента

Сигналы ЭМГ могут использоваться для оценки психологического, эмоционального и физического состояния человека. Эти оценки могут проводиться на основе анализа формы, частоты и амплитуды сигналов ЭМГ.  
  
Ниже представлены некоторые конкретные методики, которые используют электромиографию для оценки физиологического состояния человека:

1. **Критерий Лорда:** этот метод используется для оценки уровня тревожности пациента. Для этого исследователи вычисляют сумму квадратов амплитуд сигналов ЭМГ за определенный промежуток времени. Увеличение этой суммы показывает увеличение уровня тревоги.
2. **Тест Барского:** этот тест используется для оценки психического состояния человека, особенно его психической устойчивости. У пациента просят сделать многократные сжимания руки на максимальной амплитуде, в момент сжатия регистрируются сигналы ЭМГ. Если пациент успешно проходит тест, то сигналы ЭМГ будут короткими и малоамплитудными.
3. **Различные методы биофидбека:** биофидбек — это метод, с помощью которого пациент получает обратную связь о своих физиологических показателях, например, сигналах ЭМГ, что помогает ему научиться контролировать свои физические функции. Различные методы биофидбека могут быть использованы для улучшения показателей мышечного напряжения, амплитуды и частоты сигналов ЭМГ.
4. **Различные методы физической тренировки:** Эксперименты показали, что при выполнении физических упражнений амплитуда и длительность сигналов ЭМГ увеличиваются. Методы, которые используют эту зависимость, могут использоваться для оценки общей физической подготовленности спортсменов.
5. **Оценка утомления:** Утомление мышц можно оценить с помощью анализа сигналов ЭМГ. Если мышцы устали, то амплитуда ЭМГ-сигналов уменьшится, а частота и длительность увеличатся.

Сигналы ЭЭГ (электроэнцефалограмма) могут использоваться для оценки психологического, эмоционального и физического состояния человека. Для оценки используется амплитуда, частота и форма волн сигналов в различных местах наложения электродов.

Ниже представлены некоторые конкретные методики, которые используют электроэнцефалографию для оценки физиологического состояния человека:

1. **Режим сна:** ЭЭГ используется для мониторирования уровня пациента во время сна. Продолжительность этого режима может быть использована для диагностики нарушений сна, например, бессонницы.
2. **Событийно-связанный потенциал:** это метод, который позволяет исследовать взаимосвязь внешнего возбуждения и формы сигналов ЭЭГ. Например, это может быть использовано для исследования внимания пациента, в этом случае сигналы ЭЭГ должны быть несимметричны в задних частях мозга.
3. **Картирование пульсаций мозга**: этот метод основан на анализе изменений амплитуды зафиксированных волн на ЭЭГ, каждая из которых отвечает конкретным функциям мозга. Картирование позволяет определить положение и интенсивность активности отдельных участков коры головного мозга, что может помочь при диагностике различных психических нарушений (например, депрессия, синдром дефицита внимания с гиперактивностью и др.).
4. **Фрактальный анализ ЭЭГ:** данный метод основывается на оценке изменения показателей фрактальной размерности. Он позволяет не только выявить нарушения в состоянии мозга, но и оценить динамику изменений во времени. При этом фрактальный анализ позволяет оценить как режим покоя, так и функциональное состояние мозга во время выполнения конкретных задач.
5. **Топографический анализ ЭЭГ**: данный метод основан на анализе электрической активности мозга, записанной в определенный момент времени на нескольких точках головы. Такой анализ позволяет определить соответствующие структуры мозга, которые отвечают за модуляцию аффективных, мотивационных и психических функций.
6. **Анализ синхронизации мозговой активности:** с помощью этого метода можно найти зависимости между разными участками мозга, которые работают синхронно при выполнении определенных функций. Это позволяет выявлять изменения в физиологических взаимодействиях между различными частями мозга, что может предоставить информацию о нарушении эмоционального и психического состояний.
7. **Анализ мгновенных фазовых и амплитудных изменений**: данный метод помогает выявлять отдельные состояния мозга, например, усталость или тревогу. При этом проводится сравнительный анализ состояний мозга со временем, что позволяет выявить динамику изменений.
8. **Анализ полос частот:** данный метод используется для выявления особенностей функционирования различных структур мозга, так как каждая из полос частот отвечает за определенную функцию. Таким образом, анализ частот в ЭЭГ может дать информацию о соответствующей работе отдельных частей мозга и предоставить данные о психологическом, эмоциональном и физическом состоянии человека.

Так же достаточно показательным может оказаться амплитуда некоторых ритмов ЭЭГ, каждый из которых связан с определенного рода деятельностью например:

**Состояния, характерные для выраженного тета и дельта ритма**

* Дельта-ритм возникает как при глубоком естественном сне, так и при наркотическом, а также при коме.
* Дельта-ритм также наблюдается при регистрации ЭЭГ от участков коры, граничащих с областью травматического очага или опухоли.
* Низкоамплитудные (20—30 мкВ) колебания этого диапазона могут регистрироваться в ЭЭГ покоя при некоторых формах стресса и длительной умственной работе.
* Дельта-ритм имеет высокую амплитуду при выполнении задач непрерывной производительности.
* Увеличение дельта-активности в состоянии бодрствования у взрослого человека связана со многими неврологическими расстройствами и с некоторыми психическими заболеваниями.
* В состоянии алкогольного опьянения отмечается увеличение мощности дельта спектра.
* Региональная дельта-активность является признаком патологии. Она может регистрироваться в течение нескольких дней после приступа мигрени или фокального эпилептического приступа.
* Локально выраженная дельта-активность характерна для опухоли мозга, тогда как локальная медленная активность в целом не даёт специфичную диагностику (инфаркт мозга, опухоль, абсцесс и травма могут дать одинаковую картину) и свидетельствуют либо об очаговой нейрональной дисфункции мозга, либо о его очаговом повреждении.

**Состояния, характерные для выраженного альфа-ритма:**

* Наиболее выражен в затылочной доле, рядом с зрительной корой головного мозга.
* Наибольшую амплитуду α-ритм имеет в состоянии спокойного бодрствования, особенно при закрытых глазах в затемнённом помещении.
* Блокируется или ослабляется при повышении внимания (в особенности зрительного) или мыслительной активности.
* Существует предположение что альфа-ритм непосредственно связан с зрительной корой головного мозга и является её «холостым ходом».

**Состояния, характерные для выраженного бета-ритма:**

* Соответствует состоянию активного бодрствования
* Наиболее выражен в лобной доле, но при различных видах интенсивной деятельности резко усиливается и распространяется на другие области мозга
* Выраженность β-ритма возрастает при предъявлении нового неожиданного стимула, в ситуации внимания, при умственном напряжении, эмоциональном возбуждении.
* β-ритм характерен для стадии быстрого сна или при решении сложных вербальных задач.

# Практическая часть проекта

## Получение и интерпретирование данных, отправляемых датчиком

Использованный мной набор датчиков – это набор BOSLAB.

Выбор пал именно на этот набор по нескольким причинам:

1. Этот набор был в наличии в моей школе.
2. Беспроводная передача данных.
3. Неинвазивное наложение электродов.

Сам по себе набор представляет из себя несколько различных датчиков, каждый из которых предназначен для сбора разных биофизических сигналов. Все датчики подключаются по Bluetooth в режиме slave (сами не могут инициировать подключение). Так же в набор входил USB Bluetooth адаптер.

Документации по адаптеру и датчикам в интернете найти не удалось поэтому пришлось разбираться в принципе его работы методом обратного инжиниринга.

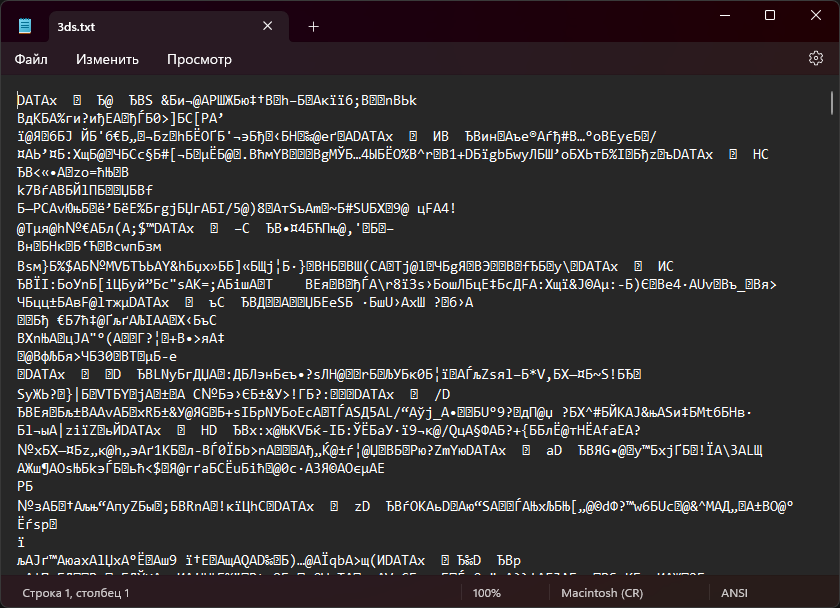
Как оказалось в дальнейшем адаптер предназначается специально для этих датчиков. Он предоставляет удобный способ получения данных через COM порт. А также автоматически подключается к датчикам, MAC адреса которых были прописаны в его конфигурации.

Попытка подключить датчики напрямую к пк – не увенчалась успехом. Поэтому использование адаптера оказалось необходимым.

Первым делом, подключив адаптер к пк в диспетчере устройств он отобразился как COM порт. Открыв монитор порта, методом перебора частоты передачи данных и отправки команды help на каждой, удалось обнаружить частоту, на которой этот адаптер принимает команды и отправляет данные, а именно 115200 бод.

На команду help был получен следующий ответ, содержащий все допустимые команды:

* *help* – выводит справку о допустимых командах
* *info* – выводит информацию о подключенных к адаптеру датчиках
* *start* – начинает передачу данных от датчика
* *stop* – прекращает передачу данных от датчика
* Так же был указан формат команды по добавлению нового датчика в конфигурацию



Отправив команду *start* в мониторе порта, не удалось увидеть понятных человеческому глазу данных.

Рисунок 5 Ответ на команду start

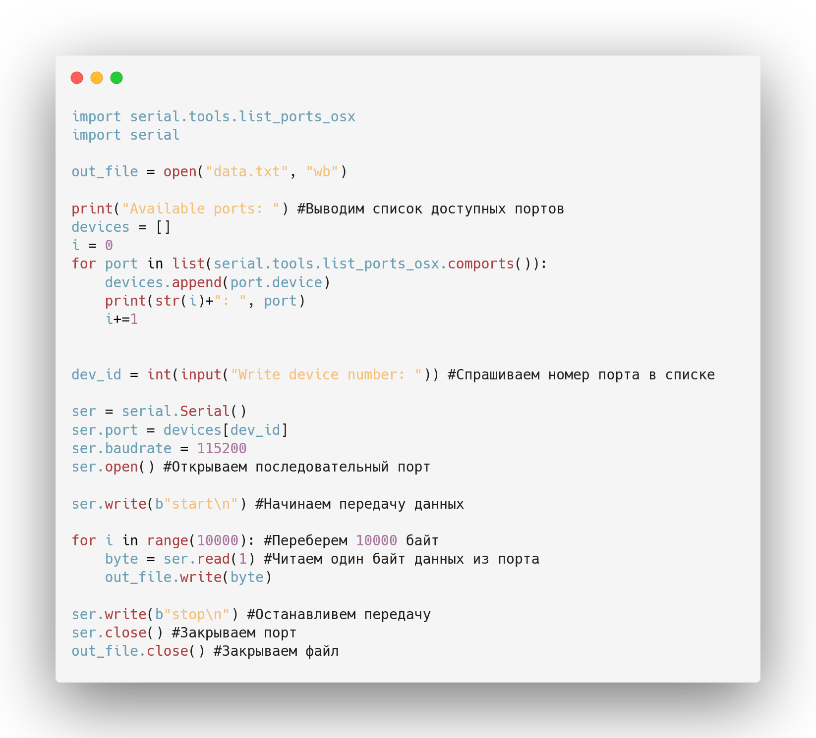
Взглянув на данные, было выдвинуто предположение, что данные отправляются в байтовом формате, это объясняло почему вместо чисел в монитор порта выводятся невнятные символы. Для подтверждения этой гипотезы, был написан скрипт на языке Python, цель которого записать полученные данные в байтовом представлении в отдельный файл с целью дальнейшего анализа этого файла.

Рисунок 6 Скрипт для записи приходящих данных

Открыв файл в байтовом редакторе, было обнаружено следующее:

* Каждые 11 байт, повторяются 4 байта которые содержащие слово «DATA»
* Каждый раз, после слова DATA в четвертом байте содержался номер датчика, с которого будет идти следующая “передача”.
* Каждый раз, после слова DATA в байтах с 9 по 12 содержалось значение   
  “00 00 80 42”, которое позже удалось интерпретировать как dB (Децибел).

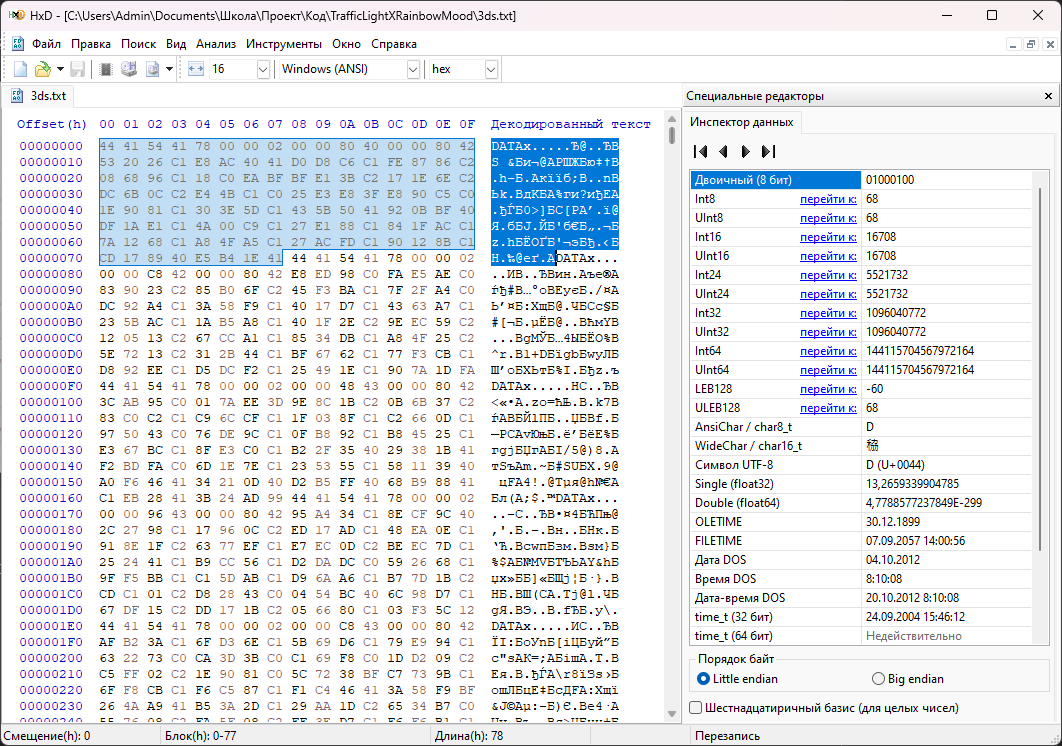


Рисунок 7 Открытый HEX редактор с записаным файлом и выделенным промежутком от DATA до DATA

В конечном итоге, структуру данных удалось выделить следующую:

→4 байта содержащие DATA(обозначает начало новой передачи) →

→3 незначимых байта →

→1 байт содержащий номер датчика, которому эта посылка принадлежит→

→ 4 байта в (float32) в которых численно выражен уровень связи в Децибелах →

→ 4 байта содержащие “dB” обозначающие конец “заголовка” →

→ 68 байт с данными →

  
  
  
  
  
  
Так как формат записи значений однозначно установить не удалось. Было принято решение написать скрипт, цель которого перевести записанные байты, в разные типы данных и в последствии построить график. Глядя на который, станет понятно в каком формате записаны значения.

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 8 Результат работы скрипта

Как видно из результата работы скрипта, достоверным является тип – ***float*** длинной 4 байта*.*

## Обработка получаемых данных в реальном времени

В качества фильтра был использован фильтр с плавающим окном.

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 9 Сырой сигнал

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 10 Модуль сигнала

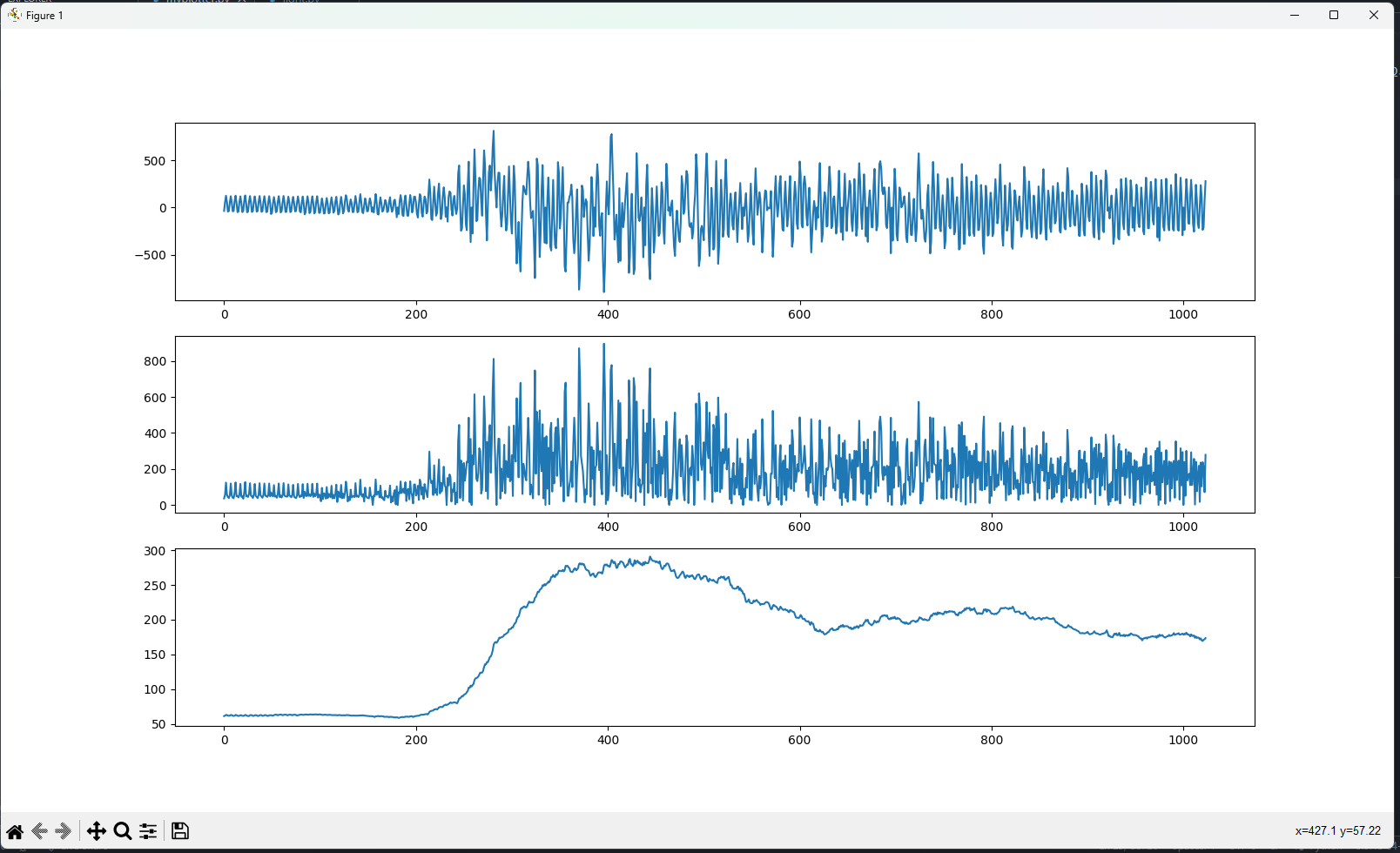


Рисунок 11 Абсолютный сигнал после фильтра с плавающим окном

Рисунок 12 Скрипт для фильтрации сигнала

Значением некоторой *n*-ой точки, после фильтра с плавающим окном длинны *L*,будет являться среднее арифметическое точек с *n-L* по *n* включительно.

*Где: x – входной сигнал; y – выходной сигнал; L – длинна окна*

Так как код на Python работает в одном потоке, который блокируется прерываниями графического интерфейса программы или прочих прерываний. Для решения этой проблемы было принято решение, создать отдельный класс, который будет собирать и обрабатывать данные из последовательного порта. И запускать этот класс в отдельном потоке. Это позволяет избегать задержек в получении данных, и переполнения буфера последовательного порта. См. приложение 1.

Изображение выглядит как текст

Автоматически созданное описание

Благодаря этому классу мы можем вычислить частоту дискретизации сигнала. Для этого соберем данные за промежуток в 10 секунду, а потом количество значений разделим на время в секундах.

Полученная частота дискретизации – **256 Гц**

Помимо всего прочего этот класс позволяет реализовать автоматическую настройку пороговых значений, которые требуются в описании тренинга.



Рисунок 13 Функция измерения пороговых значений

Пороговое значение измеряется как среднее арифметическое среднего значения между максимумом и минимумом за каждую секунду некоторого временного отрезка, 5 секунд в данном случае. Так же для того, чтобы избежать излишних повторных вычислений для одного пациента, была реализована возможность сохранения порогов в файл *porogs.txt*.  
Это позволяет избежать повторения достаточно продолжительного процесса настройки, при повторном запуске программы.

## Создание графического интерфейса тренинга

Графический пользовательский интерфейс – GUI в дальнейшем (Graphical User Interface)

Первая потребность в GUI, появилась в следствии необходимости визуально наблюдать за приходящими данными, а также для отладки некоторых алгоритмов (например алгоритм расчета порогов). Поэтому для создания «отладочного» GUI было принято решение идти по пути наименьшего сопротивления, с целью экономии времени. Для этих целей отлично подошла библиотека *matplotlib*. Эта библиотека широко известна, имеет объемную документацию и проста в использовании (базовый график можно вывести буквально за 4 строки кода). Основным преимуществом данной библиотеки является большое количество готовых объектов, которые можно настраивать. Не нужно задумываться о расчете масштаба графика или диаграммы, библиотека делает это автоматически, объекты автоматически подстраиваются под входные данные любой амплитуды и длинны. Так же эта библиотека поддерживает анимации, благодаря которым можно отображать входящие данные практически в реальном времени. Главным минусом же в свою очередь является тот факт, что при помощи этой библиотеки не получится создать свой объект, что необходимо для реализации тренингов (закрытая картинка для тренинга 2 и разноцветная полоска для тренинга 1).

Некоторые примеры интерфейсов, созданных с использованием этой библиотеки, вы уже могли наблюдать на рисунках с 8 по 11 включительно.

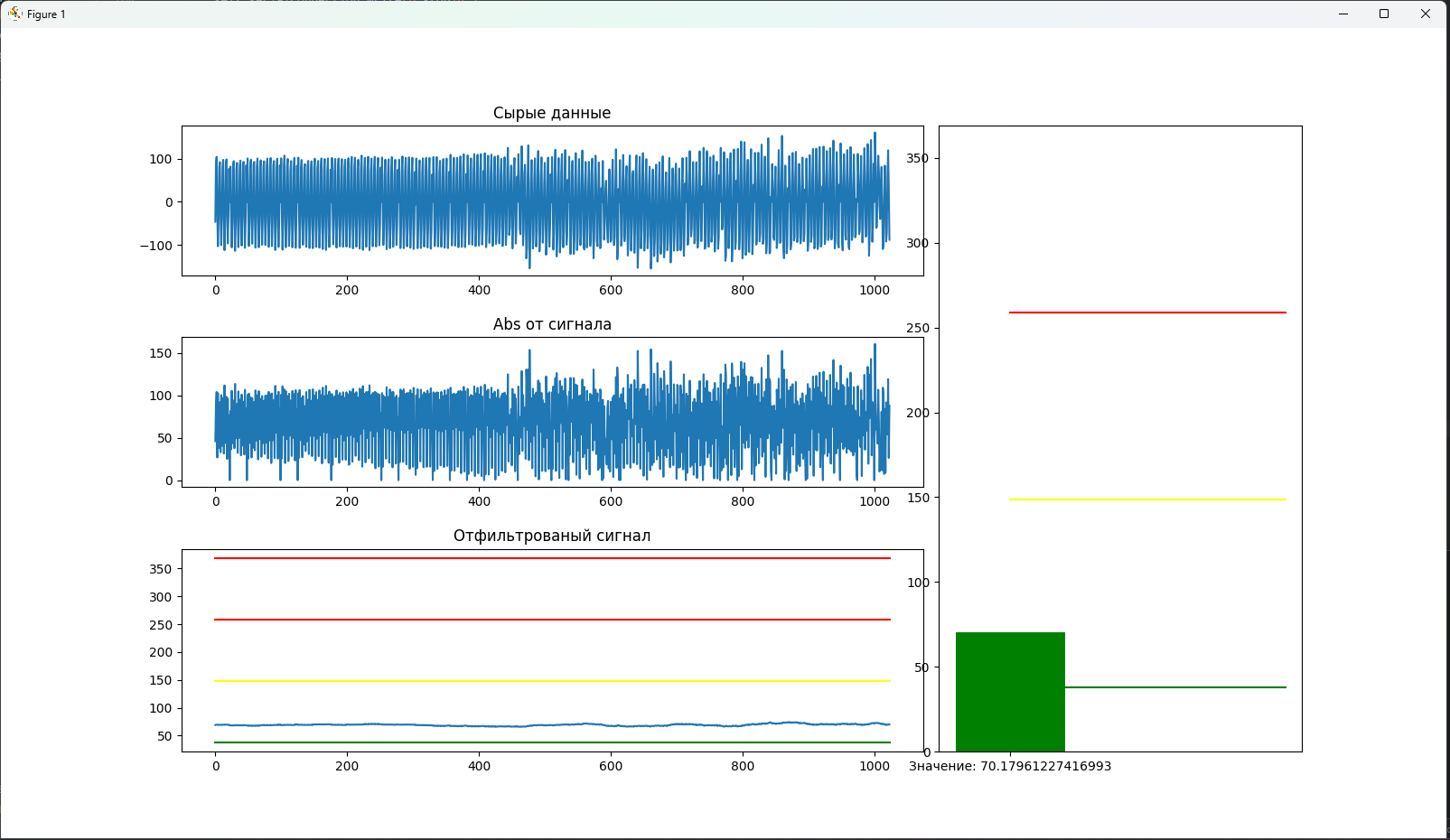


Рисунок 14 Итоговая версия "отладочного" GUI

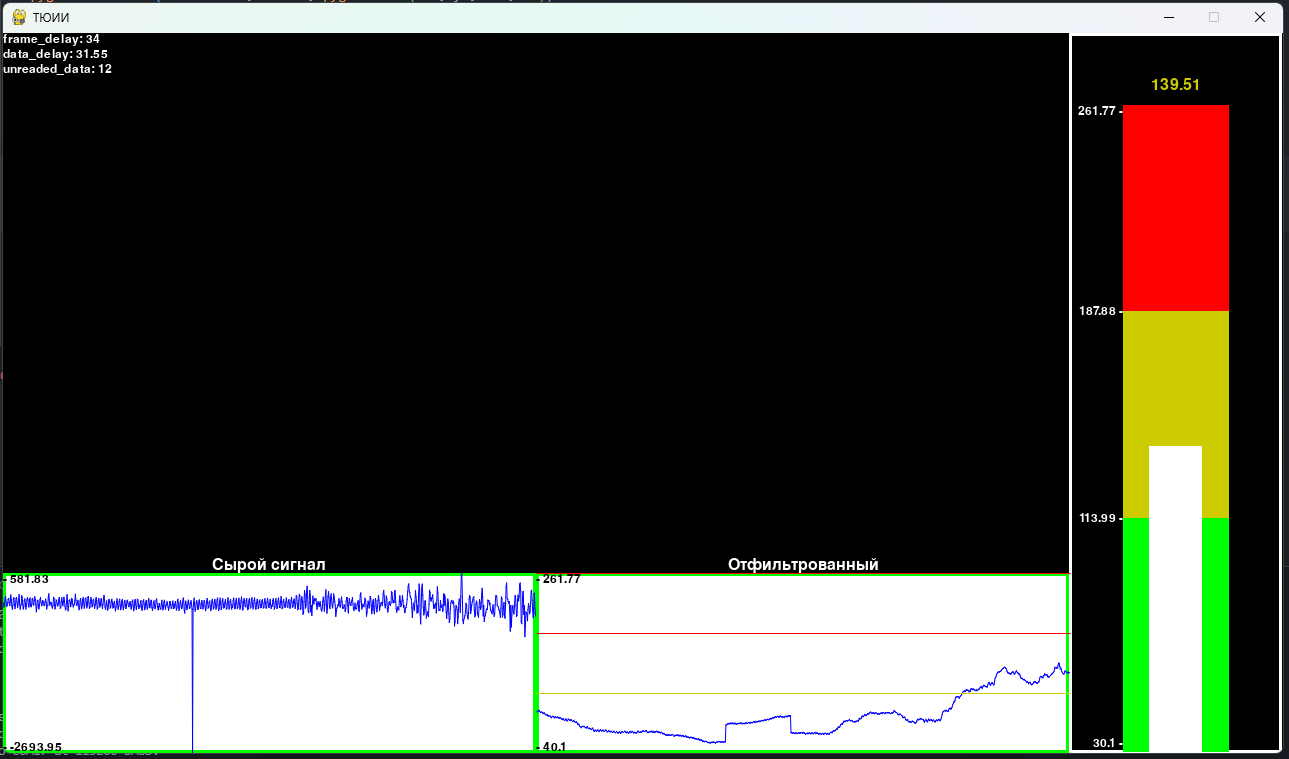
На изображении 14, можно наблюдать итоговую версию «отладочного» GUI. В окне находится 4 объекта, 3 графика по которым можно отследить процесс фильтрации сигнала и наблюдать вычисленные пороговые значения. А также вертикальную гистограмму, которая визуализирует текущее напряжение мышцы. На гистограмме и отфильтрованном сигнале можно наблюдать разноцветные линии позволяющие оценить текущее напряжение относительно вычисленных пороговых значений.

Как уже было упомянуто выше, библиотека matplotlib не подходит для реализации итогового решения. Соответственно была найдена более подходящая альтернатива - библиотека ***PyGame***.

Эта библиотека является мультиплатформенной (работает без изменения кода на всех современных платформах: Windows, Linux, MacOS). Не требовательной к ресурсам. Главным преимуществом и недостатком данной библиотеки является тот факт, что она не имеет готовых объектов, и требует создания собственных полностью с нуля, оперируя лишь простейшими геометрическими примитивами такими как: точка, прямая, прямоугольник, окружность, текст.

Были реализованы следующие графические элементы:

### Блок, содержащий информацию о работе программы

*frame\_delay* (задержка кадра) – промежуток времени между двумя кадрами в мс

*data\_delay* (задержка данных) – промежуток времени между двумя показаниями в мс

Рисунок 15 Информационный блок

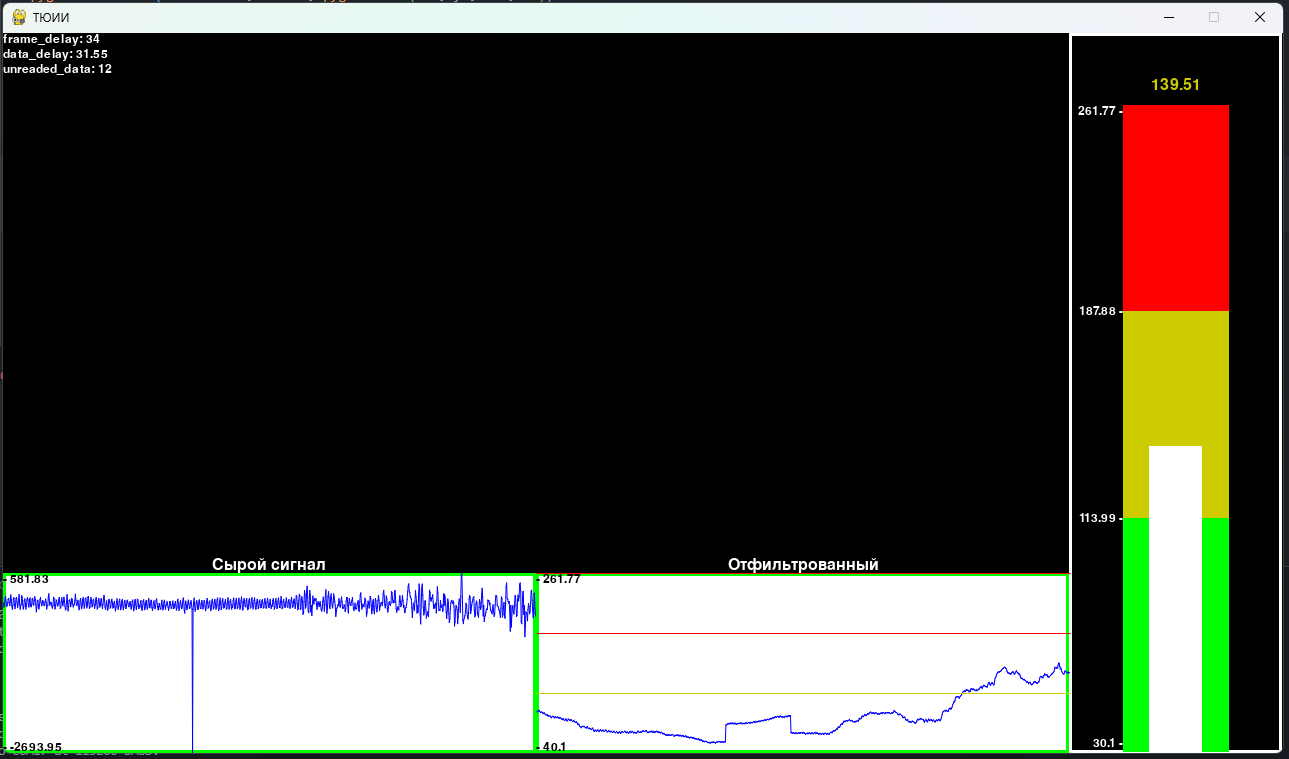
*unreaded\_data* (непрочитанные данные) – количество байт данных которые не были прочитанны и обработаны

Полный код элемента, см. приложение 2 функция *drawinfo*

### Плоттер

Рисунок 16 Плоттер с сигналом высокой амплитуды

Рисунок 17 Плоттер с сигналом низкой амплитуды



Плоттер – это блок, отображающий график содержащий данные, обновляющийся в реальном времени, способный изменять масштаб в зависимости от получаемых данных.

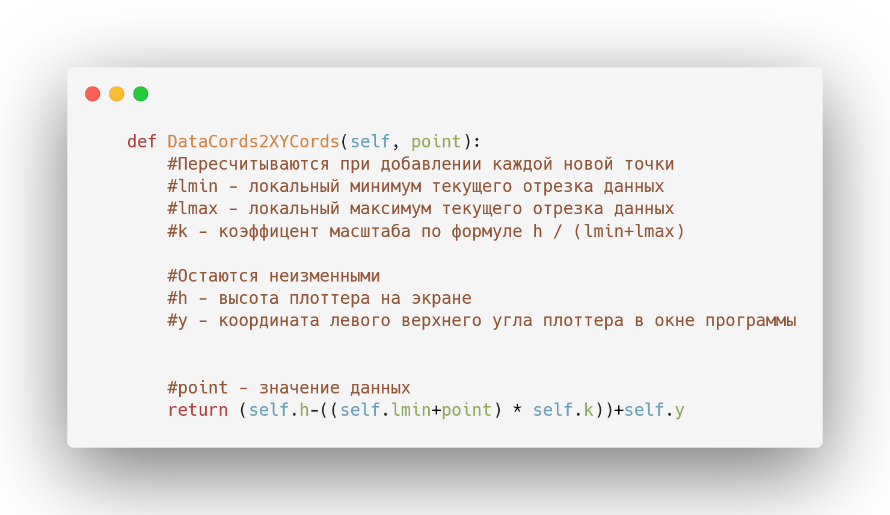
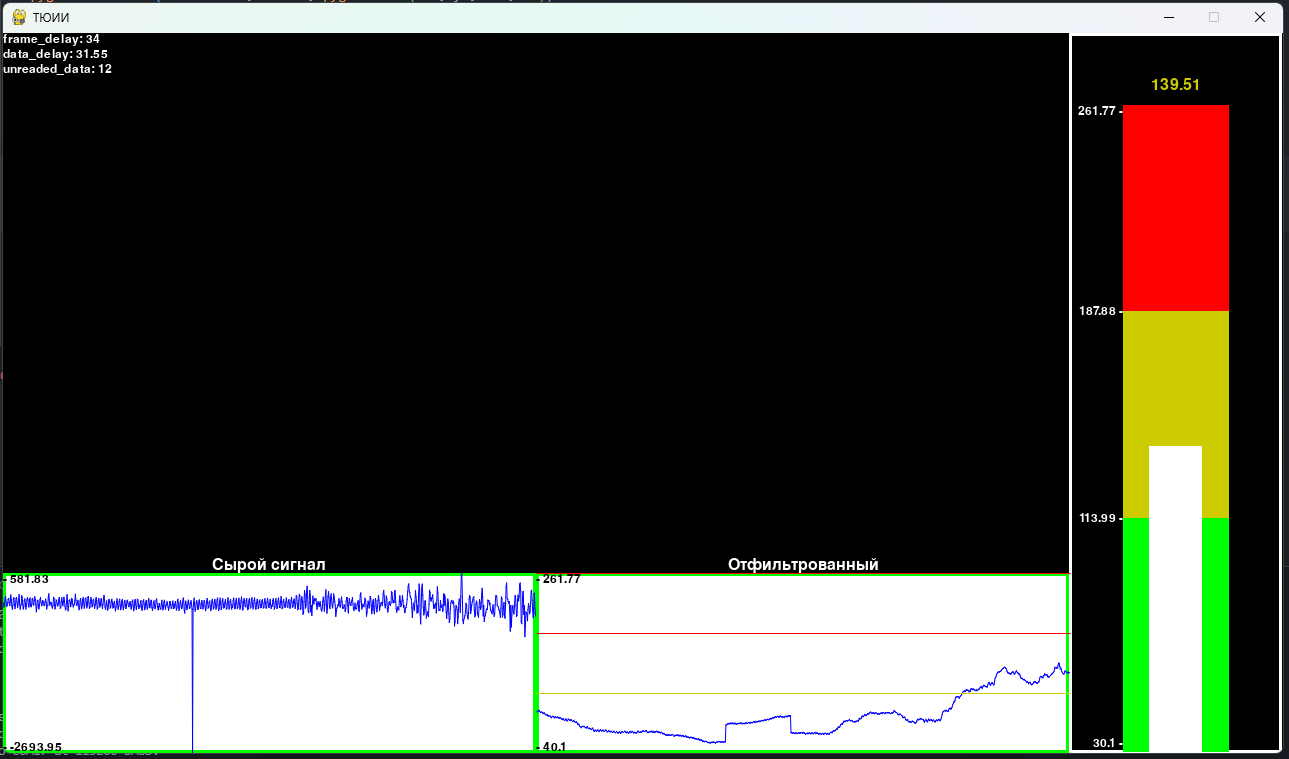
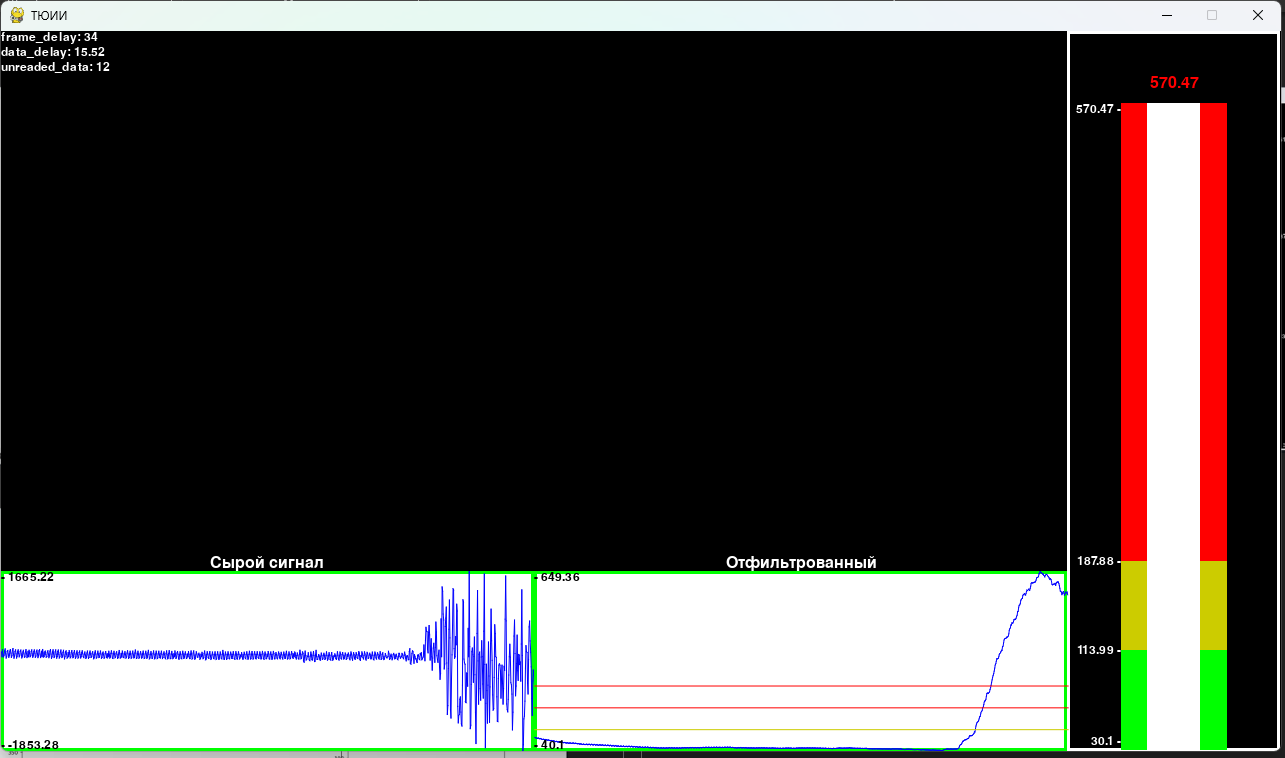
Фактически плоттер представляет из себя набор прямых каждая из которых соединяет последовательно две точки из массива данных.

Рисунок 18 Функция приведения координат данных к координатам на экране

Так же существует реализация плоттера, отображающая пороговые линии, и старающаяся держать такой масштаб, при котором все эти линии будут видны.

Рисунок 19 Плоттер с сигналом низкой амплитуды

Рисунок 20 Плоттер с сигналом высокой амплитуды



Вместе с масштабом самого графика, пороговые линии так же перемещаются в соответствующее им место на оси Y.

Полный код элемента, см. приложение 2 класс *Plotter*

### Вертикальная гистограмма

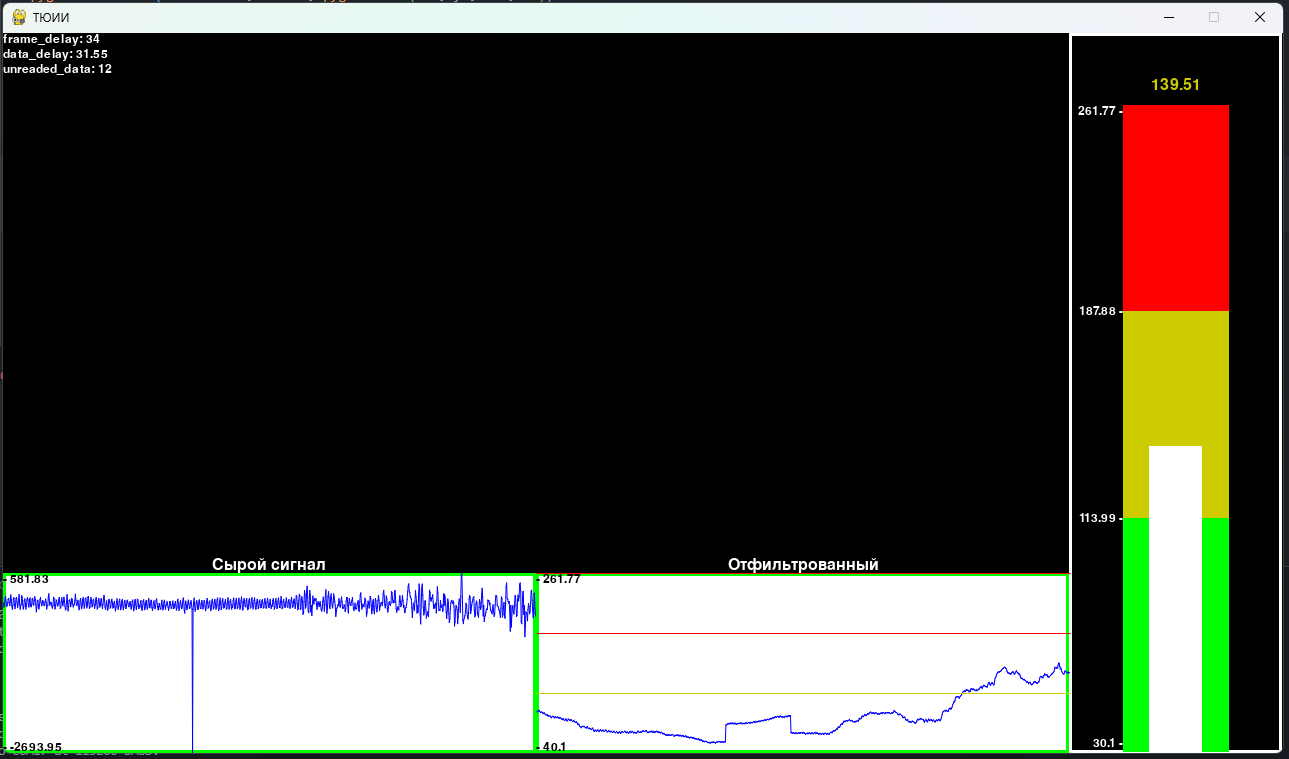


Рисунок 21 Гистограмма с низким значением

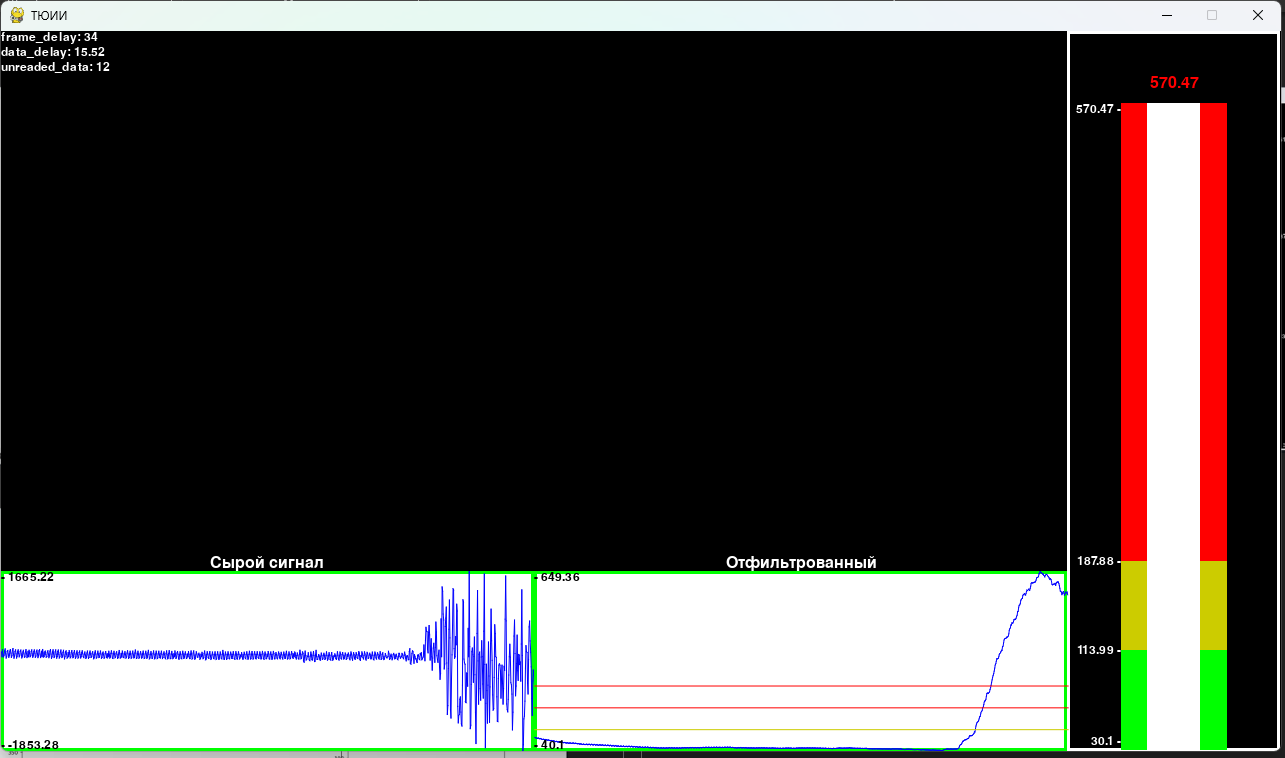
Вертикальная гистограмма позволяет пользователю тренинга наиболее наглядно отследить уровень напряжение мышцы, а так же понять в какой зоне в настоящий момент пользователь находится.

Рисунок 22 Гистограмма с высоким значением

Гистограмма разделена на три зоны (по требованиям тренинга), каждая из которых изменяет свой размер в зависимости от текущего напряжения (в случае если пользователь перешел максимальный/минимальный порог).

Значение сверху отображает текущий уровень напряжения и изменяет свой цвет в зависимости от зоны в которой находится пользователь.



Рисунок 23 Расчёт координат и высот зон в гистограмме

Код представленный выше, рассчитывает координату левого верхнего угла зоны а так же ее высоту. Это позволяет гистограмме растягивать зоны, в зависимости от напряжения.

Полный код элемента, см. приложение 2 класс *BarPlot*

### Спрятанное изображение

Это объект, который необходим для создания тренинга №2. Его суть состоит в том, чтобы отрисовать некоторое произвольное изображение, закрыть его тремя равными прямоугольниками, закрашенных цветами соответствующих им зон. Части изображения должны открываться по прошествии определенного времени нахождения напряженности мышцы пользователя в соответствующей зоне. Время, которое осталось до открытия зоны, отображается в середине прямоугольника. Таймер сбрасывается каждый раз, когда напряженность покидает соответствующую зону. Текущая зона выделяется белым контуром.

Я старался сделать так, чтобы любое изображение, любого размера, занимало наибольшую площадь на экране. А также чтобы горизонтальные изображения делились по горизонтали, а вертикальные по вертикали.

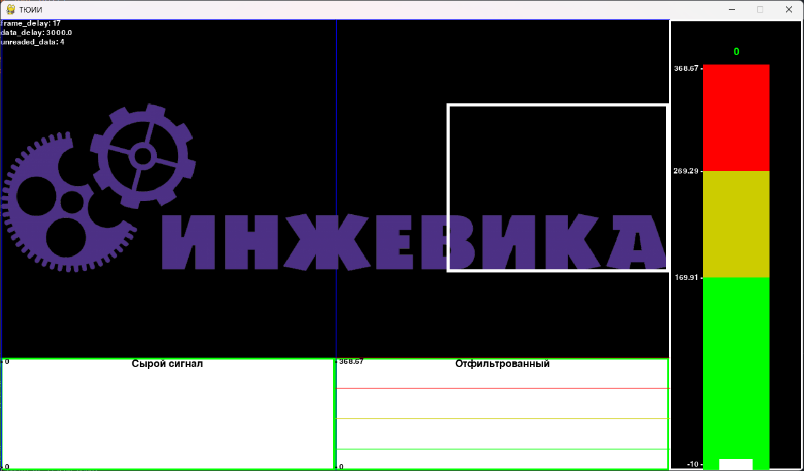
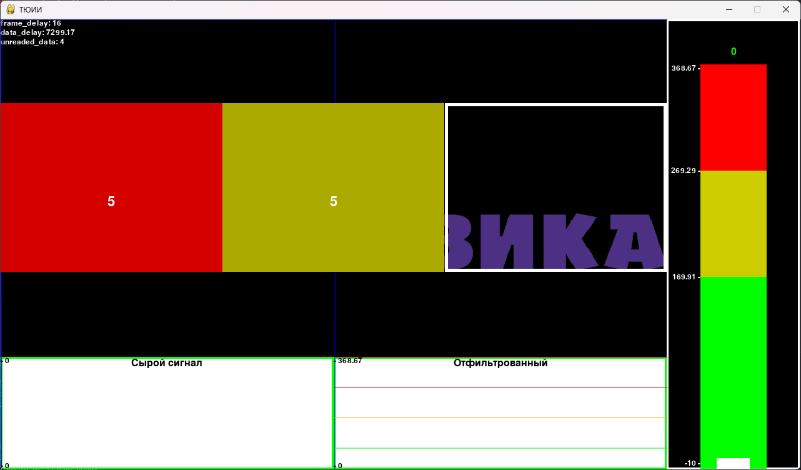


Рисунок 25 Горизонтальное закрытое изображение

Рисунок 25 Горизонтальное открытое изображение

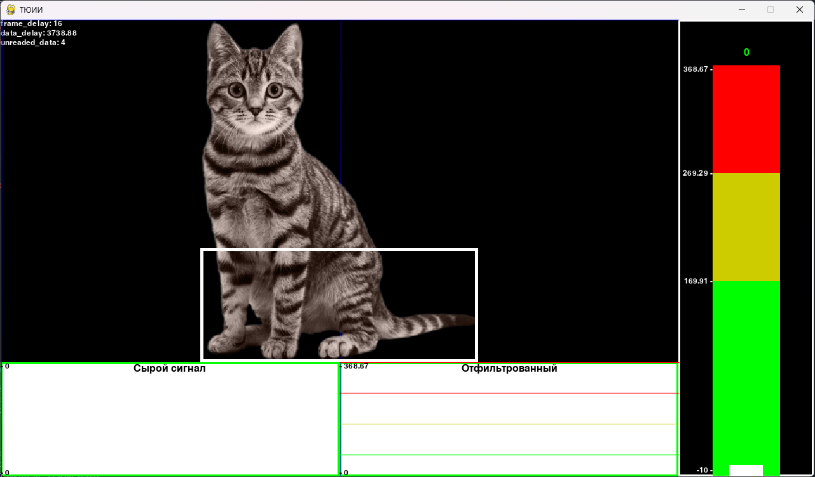
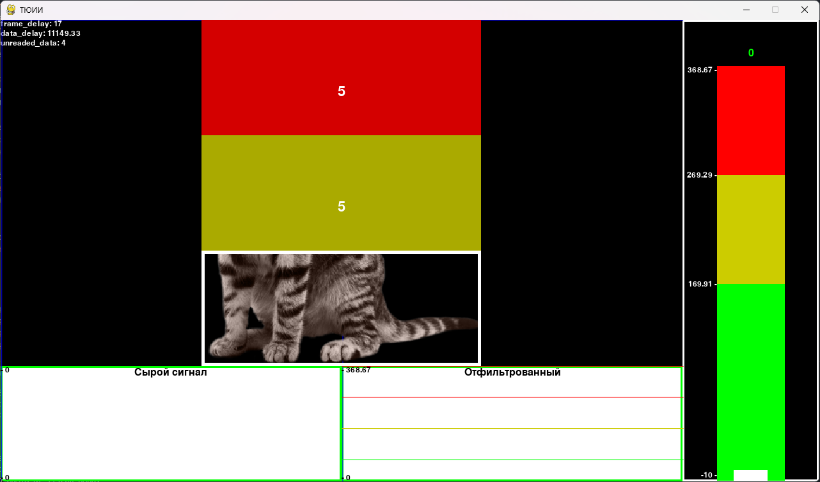


Рисунок 27 Вертикальное закрытое изображение

Рисунок 27 Вертикальное открытое изображение

Тренинг считается завершенным в момент, когда изображение полностью открыто.

Полный код элемента, см. приложение 2 классы *HidingSquare* и *ImageHidden*

### Итоговый вариант графического интерфейса

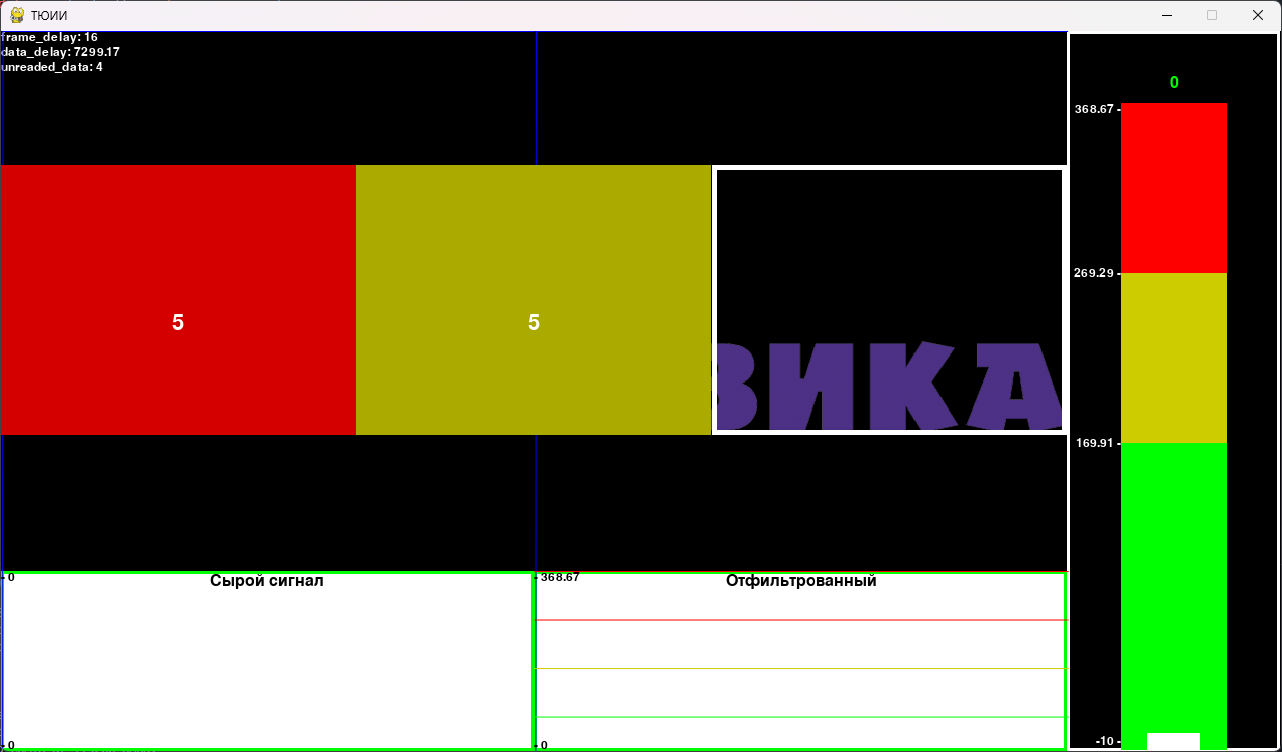


Рисунок 28 Итоговый графический интерфейс

Удалось создать наглядный графический интерфейс, содержащий всю необходимую для проведения тренинга информацию, демонстрирующий данные в реальном времени.

Получилось с нуля реализовать все необходимые графические элементы, сделать их инкапсулированными (что дает возможность переиспользования), а также адаптивными к размеру и положению. Каждый элемент можно разместить в любом месте на экране, задать любой размер, и они продолжат выполнять свою функцию в полном объеме.

Рисунок 29 Алгоритм отрисовки и деления изображения

Програмный код всех элементов, см. приложение 2

Интеграция этих элементов в одно окно, см. приложение 3

# Вывод

В процессе реализации этого проекта, мне удалось изучить принципы работы датчиков ЭМГ и ЭЭГ. Виды, эффективность и пользу от ЭМГ/ЭЭГ тренингов. И зависимость психофизиологических показателей от состояния пациента.

А также разобраться с использованием датчиков BOSLAB, протоколом их передачи данных. И разработать свой собственный ЭМГ тренинг с нуля, который включает в себя модуль получения и обработки данных, а также графический интерфейс.

В дальнейшем планируется разработать ЭЭГ тренинг, который будет использовать графические элементы, алгоритмы и функции из этого тренинга. И реализовать возможность записи показаний во время проведения тренинга, и подведения результатов прохождения, численно выраженная успешность прохождения.

# Список литературы

1. Высоцкая Л. В. Биология 10 класс [Текст] под редакцией В. К. Шумного и Г. М. Дымшица // Высоцкая Л. В., Дымшиц Г. М., Рувинский А. О., Саблина О. В., Кузнецова Л. Н. // М. – 2021. – С. 21-30.
2. Еремин В. В. Химия 10 класс [Текст] под редакцией В. В. Лунина // Еремин В. В., Кузьменко Н. Е., Теренин В. И., Дроздов А. А., Лунин В. В. // М. – 2021. – С. 372-377.
3. Липкин В. М. Белки [Электронный ресурс] // Липкин В. М., Шуваева Т. М. – Режим доступа: <https://bigenc.ru/biology/text/1853941> – статья в интернете.
4. Студопедия. Классификация белков [Электронный ресурс]. – Режим доступа: <https://studopedia.ru/17_98987_klassifikatsiya-belkov.html> , свободный. (Дата обращения: 14.10.2015 г.).
5. Bstudy – статьи для высших учебных заведений. Молочные продукты [Электронный ресурс]. – Режим доступа: <https://bstudy.net/741105/meditsina/molochnye_produkty> – статья в интернете.
6. Научно-технический энциклопедический словарь. Что такое Этанол [Электронный ресурс]. – Режим доступа: [https://dic.academic.ru/dic.nsf/ntes/5814/ЭТАНОЛ](https://dic.academic.ru/dic.nsf/ntes/5814/%D0%AD%D0%A2%D0%90%D0%9D%D0%9E%D0%9B) , свободный

# Приложение

## Приложение 1 *KomsibUNIORReader.py*

1. from threading import Thread
2. import serial
3. import time
4. import matplotlib.pyplot as plt
5. import matplotlib.animation as animation
6. # import serial.tools.list\_ports\_osx
7. import serial.tools.list\_ports\_windows
8. import struct
9. import numpy as np
10. class SerialReader:
11. #Инициализатор класса
12. def \_\_init\_\_(self) -> None:
13. baudrate = 115200
14. port = self.askport()
15. self.Thread = None
16. self.data = [0,0,0]
17. self.abs\_data = [0,0,0]
18. self.filt\_data = [0,0,0]
19. self.isRuining = True
20. self.WINDOWLENGTH = 125
21. self.last\_data = time.time()\*1000
22. print('Trying to connect to: ' + str(port) + ' at ' + str(baudrate) + ' BAUD.')
23. try:
24. self.serialConnection = serial.Serial(port, baudrate, timeout=0.02)
25. print('Connected to ' + str(port) + ' at ' + str(baudrate) + ' BAUD.')
26. except:
27. print("Failed to connect with " + str(port) + ' at ' + str(baudrate) + ' BAUD.')
29. #Функция для запроса порта к которому необходимо подключится
30. def askport(self) -> int:
31. print("Available ports: ")
32. devices = []
33. i = 0
34. for port in list(serial.tools.list\_ports\_windows.comports()):
35. devices.append(port.device)
36. print(str(i)+": ", port)
37. i+=1
38. return devices[int(input("Write device number: "))]
40. #Начало передачи данных и создание потока
41. def startThread(self):
42. self.serialConnection.write(b"stop\n")
43. time.sleep(0.1)
44. self.serialConnection.write(b"start\n")
45. if(self.Thread == None):
46. self.Thread = Thread(target=self.GetData)
47. self.Thread.start()
49. def GetData(self):
50. while(self.isRuining):
51. if(self.serialConnection.in\_waiting >= 16):
52. readed = self.serialConnection.read(4)
53. if(readed == b"DATA"): #Пропускаем часть не показательных данных
54. self.serialConnection.read(12)
55. else:
56. unpacked = struct.unpack("f", readed)[0] #Переводим в float
57. if(np.isnan(unpacked)):
58. continue #Проверяем на число
60. if(unpacked > 3000 or unpacked < -3000):
61. continue #Фильтруем выбросы
62. current\_point = float(unpacked)
63. self.data.append(current\_point)
64. self.abs\_data.append(abs(current\_point))
65. #Применяем фильтр с плаваюшим окном
66. self.filt\_data.append(sum(self.abs\_data[-self.WINDOWLENGTH:]) / self.WINDOWLENGTH)
67. #Обрезаем данные до необходимого окна во избежании утечек памяти
68. self.data = self.data[-1024:]
69. self.abs\_data = self.abs\_data[-1024:]
70. self.filt\_data = self.filt\_data[-1024:]
71. self.last\_data = time.time()\*1000
72. #Закрытие порта, потока, остановка передачи данных
73. def close(self):
74. self.isRuining = False
75. self.Thread.join()
76. self.serialConnection.write(b"stop\n")
77. self.serialConnection.close()
78. print('Disconnected...')
79. #Функция для очистки данных
80. def clear(self):
81. self.data = []
82. self.abs\_data = []
83. self.filt\_data = []
84. def getPorogs(reader):
85. def getPorog(text1, text2, text3):
86. print("Переходим к настройке "+text1+" порога!")
87. print("В течении следующих 5 секунд "+text2+" руку.")
88. print("Напишите что-ниубдь когда будете готовы")
89. input()
90. vals = []
91. for i in range(5):
92. reader.clear()
93. time.sleep(1)
94. print(i)
95. vals.append( (max(reader.filt\_data)+min(reader.filt\_data)) / 2)
96. value = sum(vals) / len(vals)
97. print("Готово!")
98. print(value, " - "+text3+" значение")
99. return value
101. ans = input("Провести повторную калибровку пороговых значений или загрузить значения из файла?\n1 - Загрузить\n2 - Провести повторно\nОтвет:")
102. if(ans == "2"):
103. maximum = getPorog("максимального", "максимально напрягите", "Максимальное")
104. minimum = getPorog("минимального", "максимально расслабьте", "Минимальное")
105. open("porogs.txt", "w").write(str(maximum)+";"+str(minimum))
106. else:
107. maximum, minimum = list(map(float, open("porogs.txt", "r").readline().split(";")))
108. porog1 = minimum + ((maximum-minimum) / 3)
109. porog2 = minimum + (((maximum-minimum) / 3)\*2)
110. return (maximum, minimum, porog1, porog2)

## Приложение 2 *elements.py*

1. import pygame
2. import time
3. WHITE = (255, 255, 255)
4. BLACK = (0, 0, 0)
5. RED = (255, 0, 0)
6. GREEN = (0, 255, 0)
7. BLUE = (0, 0, 255)
8. YELLOW = (204,204,0)
9. pygame.font.init()
10. font = pygame.font.Font(None, 24)
11. font2 = pygame.font.Font(None, 18)
12. font3 = pygame.font.Font(None, 32)
13. class HidingSquare():
14. def \_\_init\_\_(self, w,h,x,y,screen, color):
15. self.w = w
16. self.h = h
17. self.x = x
18. self.y = y
19. self.screen = screen
20. self.color = color
21. self.selected = False
23. self.start\_timer = time.time()
24. self.completed = True
25. def update(self):
26. if(not self.selected):
27. self.start\_timer = time.time()
29. if(int(round(time.time() - self.start\_timer)) >= 5):
30. self.completed = True

33. if(not self.completed):
34. pygame.draw.rect(self.screen, self.getLighterColor(), pygame.Rect(self.x, self.y, self.w, self.h))
35. if(self.selected):
36. title = font3.render(str(round(5 - (time.time() - self.start\_timer))), True, WHITE)
37. else:
38. title = font3.render(str(5), True, WHITE)
39. t\_rect = title.get\_rect()
40. self.screen.blit(title, (self.x+(self.w/2)-t\_rect.centerx, self.y+(self.h/2)+t\_rect.centery))
41. if(self.selected):
42. pygame.draw.rect(self.screen, WHITE, pygame.Rect(self.x, self.y, self.w, self.h),5)


46. def getLighterColor(self):
47. return [self.color[0] - self.color[0]/6, self.color[1] - self.color[1]/6, self.color[2] - self.color[2]/6]
48. class ImageHidden():
49. def \_\_init\_\_(self, w, h, x,y, screen):
51. self.W = w
52. self.H = h
53. self.X = x
54. self.Y = y
55. self.screen = screen
57. self.image = pygame.image.load("./image.png")
58. self.im\_W = self.image.get\_rect().w
59. self.im\_H = self.image.get\_rect().h
60. self.mini = min(self.H/self.im\_H, self.W/self.im\_W)
61. self.new\_W = self.im\_W \* self.mini
62. self.new\_H = self.im\_H \* self.mini
63. self.image = pygame.transform.scale(self.image, (int(self.new\_W), int(self.new\_H)))
65. self.new\_X = self.X + ((self.W - self.new\_W)/2)
66. self.new\_Y = self.Y + ((self.H - self.new\_H)/2)
68. self.squares = []
69. colors = [RED, YELLOW, GREEN]
70. for i in range(1,4):
71. if(self.H/self.im\_H < self.W/self.im\_W):
72. self.squares.append(HidingSquare(self.new\_W, self.new\_H/3, self.new\_X, self.new\_Y+((self.new\_H/3)\*(i-1)), self.screen,colors[i-1]))
73. else:
74. self.squares.append(HidingSquare(self.new\_W/3, self.new\_H, self.new\_X+((self.new\_W/3)\*(i-1)), self.new\_Y, self.screen,colors[i-1]))
76. self.current\_state = 0
78. def diselectAll(self):
79. for sq in self.squares:
80. sq.selected = False
81. def upadte(self):
82. self.screen.blit(self.image, pygame.Rect(self.new\_X, self.new\_Y, self.new\_W, self.new\_H))
83. for square in self.squares:
84. square.update()
86. self.squares[self.current\_state].update()

89. class BarPlot():
90. def \_\_init\_\_(self, w, h, x,y, screen, minimum=0, maximum=0, porog1=0, porog2=0):
91. self.w = w
92. self.h = h
93. self.x = x
94. self.y = y
95. self.screen = screen
96. self.current\_val = 50
97. self.maximum = maximum
98. self.minimum = minimum
99. self.porog1 = porog1
100. self.porog2 = porog2
101. self.lmax = 0
102. self.lmin = 0
103. self.h2 = self.h-(self.h/10)
104. self.max\_window\_y = self.y+(self.h/10)
106. self.current\_color = WHITE
107. def update(self):
108. # Title
109. title = font.render(str(round(self.current\_val, 2)), True, self.current\_color)
110. t\_rect = title.get\_rect()
111. self.screen.blit(title, (self.x+(self.w/2)-t\_rect.centerx, self.y+(self.h/20)+t\_rect.centery))
112. # MainRect
113. pygame.draw.rect(self.screen, WHITE, pygame.Rect(self.x, self.y, self.w, self.h), 3)
114. # Groups Rects
115. srx = self.x+self.w/4
116. srw = self.w/2
117. y1 = self.DataCords2XYCords(self.lmax)
118. h1 = self.h-y1
119. y2 = self.DataCords2XYCords(self.porog2)
120. h2 = self.h-y2
121. y3 = self.DataCords2XYCords(self.porog1)
122. h3 = self.h-y3
123. pygame.draw.rect(self.screen, RED, pygame.Rect(srx, y1, srw, h1))
124. pygame.draw.rect(self.screen, YELLOW, pygame.Rect(srx, y2, srw, h2))
125. pygame.draw.rect(self.screen, GREEN, pygame.Rect(srx, y3, srw, h3))
126. # Current Value Bar rect
127. curr\_y = self.DataCords2XYCords(self.current\_val)
128. pygame.draw.rect(self.screen, WHITE, pygame.Rect(srx+srw/4, curr\_y, srw/2, self.h-curr\_y))
129. # Values Markers
130. lmintext = font2.render(str(round(self.lmin,2))+" -", True, WHITE)
131. lmaxtext = font2.render(str(round(self.lmax,2))+" -", True, WHITE)
132. porog1text = font2.render(str(round(self.porog1,2))+" -", True, WHITE)
133. porog2text = font2.render(str(round(self.porog2,2))+" -", True, WHITE)
134. self.screen.blit(lmaxtext, (srx-lmaxtext.get\_rect().w, self.max\_window\_y))
135. self.screen.blit(lmintext, (srx-lmintext.get\_rect().w, (self.y+self.h)-lmintext.get\_rect().h-4 ))
136. self.screen.blit(porog1text, (srx-porog1text.get\_rect().w, y3-lmintext.get\_rect().centery ))
137. self.screen.blit(porog2text, (srx-porog2text.get\_rect().w, y2-lmintext.get\_rect().centery ))
138. def UpdateValue(self, value):
139. self.current\_val = value
140. self.lmax = max(self.maximum, value)
141. self.lmin = min(self.minimum,value)-10
142. d = self.lmax-self.lmin+0.00000001
143. self.k = self.h2 / d
144. if(value > self.porog2):
145. self.current\_color = RED
146. elif(value > self.porog1 and value < self.porog2):
147. self.current\_color = YELLOW
148. else:
149. self.current\_color = GREEN
150. def DataCords2XYCords(self, point):
151. return (self.h2-((point-self.lmin) \* self.k))+self.max\_window\_y
152. class Plotter():
153. def \_\_init\_\_(self, w, h, x,y, res,title, screen, minimum=0, maximum=0, porog1=0, porog2=0, drawPorog=False ):

156. self.w = w
157. self.h = h
158. self.x = x
159. self.y = y
160. self.res = res
161. self.title = title
162. self.screen = screen
163. self.drawPorog = drawPorog
164. self.k = 0
165. self.data = [] #Массив для хранения данных в том виде в котором они приходят
166. self.nicedata = [] #Массив для хранения данных приведеных к координатам на экране
167. self.xs = [] #Массив для хранения координат по х для каждой точки
169. self.lmin = 0 #Локальный минимум на графике
170. self.lmax = 0 #Локальный максимум на графике
171. self.maximum = maximum
172. self.minimum = minimum
173. self.porog1 = porog1
174. self.porog2 = porog2
175. for i in range(res):
176. self.nicedata.append(0)
177. self.xs.append(x+((w/res)\*i))
178. def addPoint(self,value):
179. self.data.append(value)
180. self.data = self.data[-self.res:]
181. self.lmax = max(self.maximum, max(self.data))
182. self.lmin = -min(self.minimum,min(self.data))
183. d = self.lmin+self.lmax+0.00000001
184. self.k = self.h / d
185. for i in range(len(self.data)):
186. self.nicedata[i] = self.DataCords2XYCords(self.data[i])
188. def updateAllData(self, data):
189. self.data = data
190. self.data = self.data[-self.res:]
191. self.lmax = max(self.maximum, max(self.data))
192. self.lmin = -min(self.minimum,min(self.data))
193. d = self.lmin+self.lmax+0.00000001
194. self.k = self.h / d
195. for i in range(len(self.data)):
196. self.nicedata[i] = self.DataCords2XYCords(self.data[i])
197. def DataCords2XYCords(self, point):
198. #Пересчитываются при добавлении каждой новой точки
199. #lmin - локальный минимум текущего отрезка данных
200. #lmax - локальный максимум текущего отрезка данных
201. #k - коэффицент масштаба по формуле h / (lmin+lmax)
203. #Остаются неизменными
204. #h - высота плоттера на экране
205. #y - координата левого верхнего угла плоттера в окне программы

208. #point - значение данных
209. return (self.h-((self.lmin+point) \* self.k))+self.y
210. def update(self):
211. # Нарисовать очертания плоттера
212. pygame.draw.rect(self.screen, WHITE, pygame.Rect(self.x,self.y,self.w,self.h))
213. pygame.draw.rect(self.screen, GREEN, pygame.Rect(self.x,self.y,self.w,self.h), 3)
214. # Нарисовать сам график
215. lines = []
216. for i in range(len(self.nicedata)):
217. lines.append([self.xs[i], self.nicedata[i]])
218. pygame.draw.aalines(self.screen, BLUE, False, lines)
219. if(self.drawPorog):
220. pygame.draw.aaline(self.screen, RED, (self.x, self.DataCords2XYCords(self.maximum)), (self.x+self.w, self.DataCords2XYCords(self.maximum)),4)
221. pygame.draw.aaline(self.screen, RED, (self.x, self.DataCords2XYCords(self.porog2)), (self.x+self.w, self.DataCords2XYCords(self.porog2)),4)
222. pygame.draw.aaline(self.screen, YELLOW, (self.x, self.DataCords2XYCords(self.porog1)), (self.x+self.w, self.DataCords2XYCords(self.porog1)),4)
223. pygame.draw.aaline(self.screen, GREEN, (self.x, self.DataCords2XYCords(self.minimum)), (self.x+self.w, self.DataCords2XYCords(self.minimum)),4)
224. #Нарисовать максимальное и минимальное значение на графике
225. lmintext = font2.render("- "+str(round(-self.lmin,2)), True, BLACK)
226. lmaxtext = font2.render("- "+str(round(self.lmax,2)), True, BLACK)
227. self.screen.blit(lmaxtext, (self.x, self.y))
228. self.screen.blit(lmintext, (self.x, (self.y+self.h)-lmintext.get\_rect().h ))
229. # Нарисовать заголовок
230. title = font.render(self.title, True, BLACK)
231. t\_rect = title.get\_rect()
232. self.screen.blit(title, (self.x+(self.w/2)-t\_rect.centerx,self.y+2))
234. class Point(pygame.sprite.Sprite):
235. def \_\_init\_\_(self, x,y, radius):
236. self.x = x
237. self.y = y
238. self.radius = radius
239. pygame.sprite.Sprite.\_\_init\_\_(self)
240. self.image = pygame.Surface((2\*radius, 2\*radius))
241. self.image.fill(RED)
242. pygame.draw.circle(self.image, RED, (radius,radius), radius)
243. self.rect = self.image.get\_rect()
244. self.rect.center = (self.x+radius,self.y+radius)

247. def update(self):
248. pass
249. self.rect.center = (self.x+self.radius,self.y+self.radius)
250. def drawinfo(framedelay, datadelay,unreaded\_data, screen):
251. framedelay\_text = font2.render("frame\_delay: "+str(framedelay), True, WHITE)
252. datadelay\_text = font2.render("data\_delay: "+str(datadelay), True, WHITE)
253. unreaded\_data\_text = font2.render("unreaded\_data: "+str(unreaded\_data), True, WHITE)
254. bottom = framedelay\_text.get\_rect().bottom+2
255. screen.blit(framedelay\_text, (0,0))
256. screen.blit(datadelay\_text, (0,bottom))
257. screen.blit(unreaded\_data\_text, (0,bottom\*2))

## Приложение 3 *PygameGUI.py*

1. import pygame
2. import math as m
3. import struct
4. from elements import Plotter, drawinfo, BLACK, BarPlot, ImageHidden
5. import time
6. from KomsibUNIORReader import SerialReader, getPorogs
7. WIDTH = 1280
8. HEIGHT = 720
9. FPS = 60
10. Timer = 5
11. #Инициализация модуля для получения данных
12. reader = SerialReader()
13. reader.startThread()
14. #Получение пороговых значений
15. maximum, minimum, porog1, porog2 = getPorogs(reader)
16. print(maximum, minimum, porog1, porog2)
17. # Создаем игру и окно
18. pygame.init()
19. pygame.mixer.init()
20. screen = pygame.display.set\_mode((WIDTH, HEIGHT))
21. pygame.display.set\_caption("ТЮИИ")
22. clock = pygame.time.Clock()
23. all\_sprites = pygame.sprite.Group()
24. #Инициализация всех граф элементов
25. Plotter1 = Plotter(WIDTH/2.4, HEIGHT/4,   0   ,HEIGHT-HEIGHT/4, 1024, "Сырой сигнал", screen)
26. Plotter3 = Plotter(WIDTH/2.4, HEIGHT/4,WIDTH/2.4,HEIGHT-HEIGHT/4, 1024, "Отфильтрованный",screen, minimum, maximum, porog1, porog2, drawPorog=True)
27. Bar1 = BarPlot(WIDTH/6,HEIGHT,(WIDTH/2.4)\*2,0, screen, minimum, maximum, porog1, porog2)
28. Image1 = ImageHidden(WIDTH-(WIDTH/6), HEIGHT-(HEIGHT/4),  0, 0, screen)
29. # Цикл игры
30. running = True
31. while running:
32. # Держим цикл на правильной скорости
33. delay = clock.tick(FPS)
34. for event in pygame.event.get():
35. if event.type == pygame.QUIT:
36. running = False
37. reader.close()
38. # Обновление
39. Plotter1.updateAllData(reader.data)
40. Plotter3.updateAllData(reader.filt\_data)
41. Bar1.UpdateValue(reader.filt\_data[-1])
42. if(reader.filt\_data[-1] > porog2):
43. Image1.diselectAll()
44. Image1.squares[0].selected = True
45. # self.current\_color = RED
46. elif(reader.filt\_data[-1] > porog1 and reader.filt\_data[-1] < porog2):
47. # self.current\_color = YELLOW
48. Image1.diselectAll()
49. Image1.squares[1].selected = True
50. else:
51. # self.current\_color = GREEN
52. Image1.diselectAll()
53. Image1.squares[2].selected = True
54. all\_sprites.update()
55. # for count in RESOLUTION:
56. # Рендеринг
57. screen.fill(BLACK)
58. drawinfo(delay, round((time.time()\*1000)-reader.last\_data,2), reader.serialConnection.in\_waiting ,screen)
59. Bar1.update()
60. Plotter1.update()
61. Plotter3.update()
62. Image1.upadte()
63. all\_sprites.draw(screen)
64. # После отрисовки всего, переворачиваем экран
65. pygame.display.flip()
66. pygame.quit()