Федеральное государственное бюджетное учреждение науки
Институт проблем машиноведения Российской академии наук
(ИПМаш РАН)

На правах рукописи

Бабич Николай Александрович

УПРАВЛЕНИЕ ТРАНСПОРТНЫМ СРЕДСТВОМ НА ОСНОВЕ НЕЙРОИНТЕРФЕЙСА С ПРИМЕНЕНИЕМ МАШИННОГО ОБУЧЕНИЯ

2.3.1 – Системный анализ, управление и обработка информации, статистика

Диссертация
на соискание ученой степени
кандидата технических наук

Научный руководитель
доктор технических наук, профессор
Фрадков Александр Львович

Санкт-Петербург – 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современных биотехнологиях, нейротехнологиях и других человеко-машинных технологиях все чаще встречаются задачи, связанные с организацией эффективного взаимодействия нейронной активности человеческого мозга и технических систем. Для решения подобных задач разрабатываются нейроинтерфейсы – средства обмена сигналами и информацией между нейронами человеческого мозга и компьютерными устройствами. Нейроинтерфейсы бывают инвазивные, основанные на введении электродов непосредственно в мозг человека и неинвазивные, использующие сигналы измерений активности нейронов без введения электродов внутрь организма. Среди неинвазивных нейроинтерфейсов особый интерес представляют нейроинтерфейсы на основе электроэнцефалонрафии (ЭЭГ), являющиеся наиболее распространенными вследствие относительной дешевизны и простоты работы с ними. В любом случае нейроинтерфейс представляет собой систему, позволяющую управлять внешним электронным устройством при помощи сигналов, считываемых с мозга человека.

Одной из интереснейших и важнейших задач, для решения которых применяются нейроинтерфейсы, является управление оборудованием «силой мысли». Ее решение позволит расширить возможности реабилитации пациентов с нарушениями двигательного аппарата, улучшить качество робототехнических протезов, откроет новые возможности управления объектами, находящимися в труднодоступных и опасных областях и т.д.

Однако разработка подобных систем требует разработки средств измерения сигналов, алгоритмов их предварительной обработки, а также собственно алгоритмов управления, позволяющих управлять электронным оборудованием по сигналам полученным с человеческого мозга.

Эта система включает в себя несколько элементов – устройство считывания сигналов активности головного мозга, аппаратно-программный комплекс, выполняющий обработку и анализ этих сигналов, а также объект управления. Основную сложность представляет разработка методов и алгоритмов, способных

правильно распознавать и предсказывать намерения человека, который использует этот интерфейс, чтобы обеспечить решение задач управления.

Такие комплексы должны позволять производить предобработку сигналов ЭЭГ с целью снижений уровня шума и удаления артефактов, выделять информативные признаки, фиксировать и классифицировать намерения человека совершать то или иное движение. Результаты классификации передаются на исполнительные устройства, подключенные к оборудованию или транспортным средствам, что при правильной работе всего комплекса позволяет совершить задуманное движение.

Актуальность разработки подобных систем и технологий обуславливается тем, что они позволяют решать целый комплекс важных практических задач, связанных с применением систем управления с нейроинтерфейсами для облегчения условий жизни людям с ограниченными возможностями, для реабилитации после различных травм, инсультов, для бионического протезирования и других. Решение подобных задач осложняется тем, что в условиях задачи имеется значительная неопределённость, требующая разработки соответствующих методов обработки сигналов и управления. Широкие возможности для решения подобных задач предоставляют современные методы искусственного интеллекта: методы адаптивного управления и машинного обучения.

В России исследования в области систем с нейроинтерфейсами активно ведутся несколькими сильными коллективами, среди которых следует упомянуть группы А.Я. Каплана (МГУ), М.А. Лебедева (Сколтех, НИУ ВШЭ), А.Е. Храмова (БФУ им. И.Канта), В.Б. Казанцева (ННГУ, Нижний Новгород), А.Е. Осадчего (НИУ ВШЭ), С.Л. Шишкина, М.И. Бурцева (НИЦ "Курчатовский институт), Л.А. Станкевича (СПб ПУ Петра Великого), коллектив РНИМУ им. А.А. Пирогова и коллектив Института высшей нервной деятельности и нейрофизиологии РАН (А.А. Фролов, П.Д. Бобров и др.). Среди зарубежных ученых следует упомянуть К. Фристона (K. Friston, UK), Э. Шёлля (E. Schoell, TU Berlin), Tass, Jirsa. Есть и другие группы.

Тем не менее многие задачи еще предстоит решить. Важным классом задач управления на основе интерфейсов является нейроинтерфейсное управление подвижными объектами и транспортными средствами, где разнообразие видов поведения и управляющих действий относительно невелико, что упрощает распознавание и классификацию сигналов.

Исходя из сказанного, целью работы является разработка экспериментальной системы управления транспортными средствами на основе нейроинтерфейса с применением машинного обучения. Из всего множества транспортных средств для разработки системы управления в данной работе выбрана инвалидная коляска, являющаяся важным техническим средством взаимодействия с миром для людей с ограниченными возможностями.

Основными же задачами данной работы являются описание технических средств нейроинтерфейса, математическое описание задачи сбора данных и управления оборудованием на основе нейроинтерфейса, разработка и описание алгоритмов предобработки и обработки сигналов ЭЭГ, разработка программного обеспечения для реализации системы управления нейроинтерфейсом, разработка методики тестирования и проведение экспериментов для проверки работоспособности разработанной системы управления.

Научная новизна работы заключается в том, что ней предложен метод почти ближайших соседей, который является модификацией метода k-ближайших соседей и который добавляет радиальную зону нечёткости. Также предложен новый метод классификации, основанный на алгоритме «полоска», и учитывающий новый параметр – скорость дрейфа.

Научная и практическая значимость результатов заключается в том, что в данной работе получены новые методы, алгоритмы и программы, позволяющие выполнять обработку сигналов ЭЭГ для управления инвалидным креслом с помощью активности головного мозга человека. Разработанная система управления может служить основой для будущих исследований и разработок нейроинтерфейсов, а также послужить прототипом потребительского устройства. Также получено свидетельство о регистрации разработанного программного

обеспечения: «Программа трехпозиционного управления роботизированными механизмами с помощью сигналов ЭЭГ». Бабич Н.А., Фрадков А.Л. Свидетельство о регистрации программы для ЭВМ RU 2025618163, 02.04.2025. Заявка № 2025615087 от 11.03.2025. Часть результатов диссертации получена при поддержке гранта Министерства науки и высшего образования Российской Федерации № 075-15-2021-573 (мегагранта).

Методология и методы исследования: для достижения поставленной цели использовались методы искусственного интеллекта, технологии разработки программного обеспечения, а также собственные методики проведения испытаний.

Положения, выносимые на защиту:

- математическое описание процесса сбора и обработки сигналов ЭЭГ, а также структура нейроинтерфейса и его составные элементы;

- модифицированный алгоритм классификатора k-ближайших соседей – метод почти ближайших соседей (FANN), который позволяет снизить зависимость от значения числа k и повысить точность распознавания;

- алгоритм классификации сигналов ЭЭГ с учётом дрейфа параметров, а также метод дообучения классификатора, который позволяет повысить точность распознавания сигналов ЭЭГ;

- программная платформа на языке Python, которая позволяет реализовывать системы управления для нейроинтерфейсов, а также система управления инвалидным креслом;

- методика проведения экспериментов для тестирования разработанной системы управления, а также результаты тестирования.

Достоверность результатов подтверждается результатами численного моделирования, а также результатами экспериментальных проверок работоспособности разработанных и реализованных алгоритмов, методов и программ.

Апробация результатов заключается в том, что результаты диссертационных исследований докладывались на научных конференциях, форумах:

1. Балтийский форум нейронаук, искусственного интеллекта и сложных систем (BF-NAICS 2021), доклад, Калининград, Россия, 13-15 сентября 2021.

2. Балтийский форум нейронаук, искусственного интеллекта и сложных систем (BF-NAICS 2022), доклад, Калининград, Россия, 14-16 сентября, 2022.

3. Балтийский форум нейронаук, искусственного интеллекта и сложных систем (BF-NAICS 2024), постерная секция, Калининград, Россия, 19-21 сентября, 2024.

4. Международная научная конференция по механике «Десятые Поляховские чтения». Секционный доклад. Санкт-Петербург, Россия, 23-27 сентября, 2024.

5. Алгоритм классификации сигналов с учётом дрейфа параметров на основе метода рекуррентных целевых неравенств. XXVII конференция молодых ученых «Навигация и управление движением» с международным участием. Санкт-Петербург, Россия, 18-21 марта, 2025.

Результаты также рассматривались и обсуждались на многочисленных семинарах ИПМаш РАН.

Таким образом, в первой главе данной работы приводится анализ развития систем управления на основе нейроинтерфейсов.

Во второй главе описывается математическая постановка задачи управления оборудованием. Описываются методы предобработки сигналов ЭЭГ, их анализа, принятия решений о выдаче сигнала управления

В третьей главе описывается структура программной платформы и её компонентов. Алгоритмы обработки сигналов ЭЭГ и управления реализованы на языке программирования Python.

В четвёртой главе описывается процесс тестирования разработанной системы управления, приводится план проведения экспериментов и результаты экспериментальной проверки работоспособности системы.

1 АНАЛИЗ РАЗВИТИЯ СИСТЕМ УПРАВЛЕНИЯ НА ОСНОВЕ НЕЙРОИНТЕРФЕЙСОВ

Нейромашинный интерфейс (интерфейс мозг-компьютер, BCI — brain-computer interface) — физический интерфейс приёма или передачи сигналов между живыми нейронами биологического организма (например, мозгом животного) с одной стороны, и электронным устройством (например, компьютером) с другой стороны. В однонаправленных интерфейсах, устройства могут либо принимать сигналы от мозга, либо посылать ему сигналы (например, имитируя сетчатку глаза при восстановлении зрения электронным имплантатом). Двунаправленные интерфейсы позволяют мозгу и внешним устройствам обмениваться информацией в обоих направлениях [1].

НИ напрямую измеряет активность мозга, связанную с намерениями пользователя, и преобразует записанную активность мозга в соответствующие управляющие сигналы для программного обеспечения. Это преобразование включает в себя обработку сигналов и распознавание паттернов активности, которые обычно выполняются компьютером.

Способы сбора информации о мозговой активности можно разделить на инвазивные, то есть, с внедрением электродов непосредственно в кору головного мозга, полуинвазивные — электроды устанавливаются на открытой поверхности головного мозга, и неинвазивные — без непосредственного вмешательства в организм.

1.1 Развитие средств получения информации о мозговой активности

Развитие представлений о биоэлектрической активности нервной и мышечной ткани имеет длительную историю, начало которой связано с работами немецкого физиолога Э. Дюбуа-Реймона. В 1849 году он впервые экспериментально подтвердил способность живых тканей генерировать электрические сигналы, что заложило основы новой научной дисциплины — электрофизиологии.

Дальнейшие исследования продемонстрировали универсальность этого явления для различных биологических объектов. Так, в 1875 году британский исследователь Р. Катон сообщил о регистрации слабых электрических сигналов, возникающих в мозге кроликов и обезьян. Независимо от него, российский физиолог В. Я. Данилевский в докторской диссертации указал на наличие «самостоятельных токов мозга» у собак, не подвергавшихся внешнему воздействию [1].

Ключевой шаг в понимании ритмического характера биоэлектрической активности мозга был сделан И. М. Сеченовым, который в 1882 году опубликовал работу о гальванических явлениях в продолговатом мозге лягушки. Его ученик, Н. Е. Введенский, уже в 1884 году разработал метод регистрации мозговой активности животных с использованием телефона в качестве регистратора. В последующие десятилетия российские физиологи, включая Н. О. Цибульского и А. Бека, совершенствовали методы регистрации электрической активности центральной нервной системы животных.

Значительный прогресс был достигнут в 1913 году, когда В. В. Правдич-Неминский впервые опубликовал электроэнцефалограмму, полученную у собаки посредством струнного гальванометра. Примечательно, что регистрация велась без повреждения кожи головы, что сделало метод менее инвазивным. Серия его работ, опубликованных в 1913–1925 гг., впервые ввела в научный оборот понятие мозговых волн, впоследствии классифицированных как мозговые ритмы. Эти ритмы представляют собой электрические процессы, характеризуемые частотно-амплитудными параметрами. Частота колебаний варьирует от дельта-ритмов (1–2 Гц) до бета-ритмов (14 Гц и выше), что позволяет использовать их как индикатор функционального состояния мозга и уровня сознательной активности [2].

Вклад в становление современной электроэнцефалографии внес немецкий физиолог и психиатр Г. Бергер, который в 1929 году впервые представил регистрацию электрической активности головного мозга человека. Исследования проводились с использованием кожи головы его сына как объекта. Бергер детально изучил изменения биоэлектрической активности в условиях покоя,

когнитивной нагрузки и под действием наркоза. Его открытия, в частности альфа-ритм, способствовали признанию электроэнцефалографии в качестве клинического метода диагностики и исследования функций мозга. Уже в 1930-е годы в СССР начали формироваться специализированные лаборатории для изучения физиологии мозга и внедрения ЭЭГ в медицинскую практику.

К середине XX века электроэнцефалография зарекомендовала себя как метод, применяемый в нескольких направлениях: диагностика неврологических заболеваний, исследование функций мозга в нейрофизиологических лабораториях и терапевтическое использование механизмов биологической обратной связи.

Дополнительный импульс развитию функционального картирования мозга был дан трудами канадского нейрохирурга У. Пенфилда и электрофизиолога Г. Джаспера. В 1937 году ими была предложена методика интраоперационной электрической стимуляции, позволившая уточнить функциональную специализацию различных областей коры. Пациенты находились в сознании во время вмешательства и могли описывать возникающие ощущения, что дало уникальную возможность соотносить субъективные восприятия с зонами коры. На этой основе были уточнены карты сенсорных и моторных областей, включая речевые центры [3].

Таким образом, в течение почти столетия формировалась фундаментальная основа представлений о биоэлектрической активности мозга. От первых наблюдений локальных электрических потенциалов до клинически значимой регистрации электроэнцефалограмм был пройден путь, позволивший создать относительно целостное, хотя и во многом поверхностное, понимание принципов функционирования центральной нервной системы. Данный исторический этап заложил основу для последующих исследований по прямому взаимодействию с мозговыми структурами человека, а также для развития методов нейродиагностики и нейроинтерфейсов.

1.2 Предпосылки к появлению нейроинтерфейсов

История формирования концепции нейромашинных интерфейсов (НИ) уходит корнями в середину XX века, когда были предприняты первые попытки прямого взаимодействия между мозгом и техническими устройствами.

Первые эксперименты, заложившие основу будущих технологий НИ, принадлежат британскому нейрофизиологу Грею Уолтеру. В 1964 году он подключил электроды непосредственно к моторным зонам мозга пациента, перенесшего нейрохирургическую операцию по иным медицинским показаниям. Пациенту было предложено инициировать переключение слайдов проектора посредством нажатия кнопки, при этом фиксировалась активность моторной коры. На последующем этапе система была модифицирована: управление проектором осуществлялось уже напрямую на основе регистрации мозговой активности, предшествующей моторному акту. Примечательно, что Уолтер выявил необходимость введения временной задержки, так как электрическая активность мозга возникала раньше фактического движения. По существу, это стало первым демонстрационным образцом интерфейса, обеспечивающего контроль без двигательной активности. Несмотря на значимость данного открытия, работа Уолтера не была опубликована в научной печати, а представлена лишь в докладе лондонскому обществу Ostler Society [1].

В том же году профессор физиологии Йельского университета Хосе Дельгадо существенно усовершенствовал методику изучения подкорковых структур головного мозга с применением имплантированных электродов. Им были разработаны радиоуправляемые стимуляторы для раздражения нейронных структур как у животных, так и у человека. Используя электрод-стимосивер, Дельгадо продемонстрировал возможность воздействия на эмоциональные и поведенческие реакции, включая экспериментальную остановку агрессивного быка посредством радиосигнала. Эксперименты на кошках и обезьянах позволили выявить возможность модуляции внутригрупповых взаимодействий животных: от усиления агрессии до стимуляции заботливого поведения. В клинической практике методы электрической и радиостимуляции применялись им в терапии

психических заболеваний, включая эпилепсию и шизофрению. Исследователь также уделял внимание проблемам происхождения сознания, его развития у детей и обусловленности психических явлений сочетанием эндогенных и экзогенных факторов.

Важным этапом стало использование электроэнцефалографа (ЭЭГ) для активного управления внешними системами. В 1967 году Эдмонд Деван провел эксперименты, в ходе которых испытуемые обучались произвольно контролировать амплитуду альфа-ритмов мозга. Эти изменения преобразовывались компьютером в сигналы, аналогичные точкам и тире азбуки Морзе, что фактически представляло собой примитивную форму передачи информации из мозга в вычислительное устройство.

Концептуальное оформление направления произошло в 1973 году, когда Жак Видаль в своей статье «К прямой связи между мозгом и компьютером» впервые ввел термин «нейромашинный интерфейс». В работе были сформулированы два ключевых вопроса: возможность двусторонней передачи информации в системе «мозг–компьютер» и применимость данных об электрической активности мозга для управления внешними механизмами. Видаль рассматривал перспективы использования как спонтанной ЭЭГ, так и вызванных реакций, индуцированных сенсорной стимуляцией. Позднее в концепцию НИ были интегрированы и иные биосигналы: движения глаз, мышечные потенциалы, кожно-гальванические реакции, частота сердечных сокращений. Потенциально значимыми рассматривались акустические и соматосенсорные вызванные потенциалы, а также так называемая случайная отрицательная вариация, связанная с когнитивными процессами ожидания и возбуждения. На начальном этапе, однако, основное внимание уделялось исследованию параметров ЭЭГ в временной области и определенным вызванным реакциям, причем приоритет отдавался зрительным стимулам, как наиболее подходящим для построения невербальной символики в системах взаимодействия «человек–машина» [4].

Таким образом, уже во второй половине XX века были не только экспериментально продемонстрированы возможности считывания информации о

функциональной активности мозга, но и сформировано представление о принципиальных трудностях, ограничивающих создание полноценных НИ. Одновременно были очерчены потенциальные перспективы этих систем — от управления техническими устройствами до клинических применений в нейропсихиатрии. Данный период можно рассматривать как этап становления научно-технической базы, без которой дальнейшее развитие нейротехнологий было бы невозможным.

1.3 Основные проблемы разработки нейроинтерфейсов

Основным источником потенциалов является электрическая активность коры головного мозга, которая составляет внешнюю поверхность мозга под кожей головы. Кора головного мозга представляет собой тонкий слой серого вещества, содержащий нервные клетки (нейроны). Некоторые из них (пирамидные клетки) характеризуются так называемыми апикальными дендритами, которые представляют собой длинные мембранные трубки, идущие к поверхности, где они разветвляются в стороны на некоторое расстояние. В результате получается тонкая поверхность «белого вещества», на которой соединяется плотно сплетенная сетка мелких дендритных отростков, принадлежащих соседним апикальным дендритам. Дендриты - это электролитические соединители, которые передают электрические поля к телу нейрона, где они в конечном итоге запускают нервный импульс, «деполяризуя» нервную мембрану. Затем импульс распространяется по мембранному соединителю другого типа — аксону.

Обеспечение прямой связи между индуктивными мыслительными процессами, используемыми при решении проблем, и дедуктивными возможностями компьютера по манипулированию сигналами, в каком-то смысле является конечной целью человеко-машинного общения. Это действительно сделало бы компьютер настоящим протезом мозга. Достижение этой цели с достаточной общностью - сложная задача, которая потребует значительных успехов в нейрофизиологии (для выявления соответствующих коррелятов психических состояний и решений во внешних сигналах), в методах анализа

сигналов и в информатике. Выявив те состояния мозга, которые могут оптимизировать восприятие или обучение, мы можем значительно повысить эффективность программ обучения с помощью компьютера. Изучение восприятия, исследования дислексии и эпилепсии, изучение действия галлюциногенных препаратов и разработка методов ранней диагностики опухолей головного мозга, влияющих на восприятие, являются возможными клиническими применениями, как и использование таких систем для управления протезами [5].

Операция вживления электродов в головной мозг включает вскрытие черепа с помощью хирургической процедуры, называемой трепанацией черепа, и разрезание оболочек, покрывающих мозг. Когда электроды размещаются на поверхности коры головного мозга (полуинвазивный метод), сигнал, регистрируемый с этих электродов, называется электрокортикограммой (ЭКоГ). Сигнал, записанный с электродов, которые проникают в ткань мозга (инвазивный метод), называется интракортикальной записью. Инвазивный и полуинвазивный методы записи сочетают в себе отличное качество сигнала, очень хорошее пространственное разрешение и более высокий частотный диапазон. Артефакты менее проблемны с инвазивными записями. Интракортикальные электроды могут регистрировать нервную активность отдельной клетки головного мозга или небольших скоплений клеток головного мозга. ЭКоГ регистрирует интегрированную активность гораздо большего числа нейронов, находящихся в непосредственной близости от электродов ЭКоГ. Однако любой инвазивный метод имеет лучшее пространственное разрешение, чем ЭЭГ [6].

Несмотря на все достоинства инвазивного и полуинвазивного методов размещения электродов и чипов, есть у него и значительные недостатки. Самый очевидный недостаток в том, что при использовании инвазивного или полуинвазивного варианта существенна опасность инфекции.

Ещё одна проблема заключается в том, что электроды, внедрённые в мозг, повреждают ткани. Причём разрушение происходит не только в момент введения

электрода, но и при его длительном нахождении в мозге. Это мешает работе электрода: в месте его внедрения образуется рубцовая ткань, которая ухудшает контакт. Из-за этого, спустя какое-то время, получать данные о работе участка мозга становится сложнее, появляются дополнительные искажения сигналов. Существуют попытки решить эту проблему, например, с помощью биомиметической (то есть подражающей живой природе) концепции. Для электродов можно использовать особый материал, который в твёрдом состоянии будет размещаться в коре головного мозга, а затем размягчаться через некоторое время. Такой полимер в обычном состоянии по твёрдости напоминает пластик, из которого делают компакт-диски. В другом состоянии он сравним с мягкой резиной. Чтобы уменьшить твёрдость материала, нужно всего лишь заставить его контактировать с очищенной водой.

Другой минус замечен в процессе экспериментов. Имплантаты, в основном, требуют значительного времени настройки перед включением, а также имеют сложности в управлении. Наиболее явно недостатки проявляются, например, при манипуляциях с курсором на экране. Такое, казалось бы, несложное действие – переместить курсор и выбрать объект — реализуется не без труда. В одном из вариантов такой технологии для передвижения требуется 2,5 секунды (обычный пользователь делает аналогичное перемещение за одну), а попадание на нужный объект происходит только в 73-95% случаев (а в норме — практически 100%). Поэтому более перспективна идея получать сигналы вовсе не от нейронов, ответственных за движение, а из тех зон коры, что отвечают за намерение совершения действий. Это делает работу системы намного более быстрой. К примеру, чтобы сделать что-то с объектом на экране, совсем не нужно двигать к нему курсор – достаточно мысленно обозначить нужный объект (это сгенерирует необходимый паттерн активности нейронов), находящийся в поле зрения, и курсор сразу же окажется там, где нужно.

Волны мозга, регистрируемые на поверхности головы, сильно различаются, отражая огромное количество влияющих параметров. Общие характеристики

волновых цепей можно в некоторой степени предсказать в зависимости от места установки электрода, психического состояния пациента, а также наличия и типа сенсорной стимуляции. Некоторые из этих характеристик легко определить на глаз. Хорошо известными примерами являются распознавание альфа-активности и феномена альфа-блокирования, сна, а также спайк-волновой комплекс при малой эпилепсии. Однако более сложная информация в ЭЭГ требует компьютерного анализа.

Традиционные способы использования компьютера для анализа данных ЭЭГ отражают основное различие между непрерывной «продолжающейся» активностью и короткими, привязанными ко времени изменениями ЭЭГ, которые представляют собой вызванные реакции на кратковременные стимулы. К первым часто применялись рамки ритма мозга и такие понятия, как спектральная плотность. Последние, напротив, являются апериодическими событиями, и текущая практика обработки варьируется от простого усреднения форм сигналов до различных функциональных расширений (включая классическое преобразование Фурье) [4].

Чтобы выяснить способ представления или кодирования сенсорного ввода, или поведенческого вывода, нейронные события наблюдается, что они сосуществуют с этими стимулами или поведением. В более позднее время данные исследуется на корреляцию между произвольными атрибутами как события, так и стимулы. Существенная корреляция обычно считается допустимым результатом. Данные полученные из корреляционных исследований, дают очень мало надежды на то, что вариации обнаруженные в мозге сигналы могут когда-либо предоставить надежные индикаторы состояний мозга. Значительные, а иногда и кажущиеся безнадежными вариации - скорее правило, чем исключение. Причина этой изменчивости может быть связана с рядом причин [4].

Во-первых, входной стимул всегда не полностью охарактеризован. Его «актуальные» атрибуты неизвестны, как и множество возможных косвенных или сопутствующих влияний, вызванных номинальным «входом», способным влиять на нейронный «выход».

Во-вторых, состояние мозга неизвестно на момент возникновения стимула. (т.е. на языке системной инженерии система имеет неопределенные начальные условия), таким образом, любая доступная информация о состоянии (например, в потенциалах ЭЭГ) вообще игнорируется.

Как сама ЭЭГ, так и вызванный потенциал продемонстрировали значительную вариабельность в большинстве корреляционных исследований. Очевидно, что необходим совершенно другой подход, чтобы извлечь надежные признаки из полученной нейронной активности.

Первым шагом в подготовке сигналов ЭЭГ для последующего анализа является устранение возмущений ненейронного происхождения, таких как глазные артефакты. Проблема теперь в значительной степени решена. Напротив, более неуловимые артефакты «мускулов» остаются неизменными и по сей день. Эти артефакты являются основным источником посторонних помех, возникающих из-за электрической полярности глазного яблока. Эффект представляет собой отражение индуцированного электрического поля, которое движется, когда глаза двигаются по орбитам. Их можно удовлетворительно удалить путём вычитания соответствующей функции горизонтальной и вертикальной составляющих сигналов с каждого электрода. После удаления артефактов «сырые» образцы ЭЭГ переупорядочиваются для обеспечения «монополярных» каналов, которые можно отнести к одному электроду [4].

Второй шаг, который все ещё является частью фазы предварительной обработки, представляет собой новую попытку отфильтровать сигналы нейронного происхождения, которые присутствуют в ЭЭГ и особенно в записях скальпа, рассматриваемых в этом исследовании. Тканевая проводимость, а также функциональные связи между различными областями мозга вносят свой вклад в этот «шум». Функциональные взаимодействия - важные особенности, которые исследователи попытаются определить на основе данных. Тканевая проводимость через череп и скальп, напротив, является фактором размытия. Из-за этого обычно считается избыточным размещать электроды на расстоянии менее 3 миллиметров друг от друга на поверхности черепа. Чтобы облегчить эти проблемы и

изолировать источники помех, необработанные сигналы ЭЭГ подвергаются дополнительному преобразованию.

Описанные преобразования данных представляют собой этап предварительной обработки и подготовку к окончательному компьютерному анализу. Основная проблема в исследовании ЭЭГ - это огромное количество необработанных данных. Например, 14-канальная запись, дискретизируемая со скоростью 256 элементов данных в секунду, создаёт более 3500 элементов данных в секунду. Эксперименты по ЭЭГ обычно длятся от получаса до нескольких часов (как в исследованиях сна), и легко представить ошеломляющий объем данных, которые могут быть созданы в ходе таких экспериментов. Методы предварительной обработки, подобные только что описанной, не помогут облегчить ситуацию, поскольку предварительно обработанные данные будут занимать такой же или даже больший объем, чем исходные. Чтобы улучшить эту ситуацию, исследователи классически ограничили свой анализ короткими периодами продолжительностью 10 секунд или меньше, выбранными произвольно во время стационарного состояния эксперимента или, наоборот, привязанными к событию, которое генерирует ограниченный во времени переходный процесс.

В настоящее время мощности компьютерных систем, в том числе носимых устройств, вполне достаточно, чтобы производить всю обработку «на лету». Вычислительное устройство принимает на вход поток данных с устройства захвата мозговой активности, обрабатывает его и отправляет необходимые сигналы на объект управления.

1.4 Современное состояние разработки нейроинтерфейсов

В последние годы в инженерии, медицине и биологических исследованиях всё чаще используется понятие «интерфейс мозг-компьютер (ИМК)» или «нейроинтерфейс».

Функционирующий нейроинтерфейс предоставляет возможность создания различных устройств, управляемых посредством мыслительной активности, а

также разработки систем обучения компьютеров диагностике состояния пациента или животного. Кроме того, подобные интерфейсы значительно упрощают взаимодействие с внешним миром для лиц с ограниченными возможностями. В дальнейшем рассматриваются исключительно неинвазивные нейроинтерфейсы, не требующие хирургического вмешательства, что делает их предпочтительными с точки зрения удобства и безопасности эксплуатации.

Одним из наиболее значимых направлений применения нейроинтерфейсов является управление техническими системами с использованием мыслительных сигналов. Решение данной задачи способствует расширению возможностей реабилитации пациентов с нарушениями двигательных функций, повышению эффективности роботизированных протезов, а также открывает перспективы дистанционного управления объектами, находящимися в труднодоступных или опасных условиях. Современный подход к реализации подобных систем основан на создании программно-аппаратных комплексов, обеспечивающих регистрацию и обработку сигналов электроэнцефалограммы (ЭЭГ) в реальном времени [7–11]. Эти комплексы выполняют предварительную фильтрацию сигналов для уменьшения шумов и устранения артефактов, выделяют информативные признаки, классифицируют намерения пользователя и передают результаты классификации исполнительным механизмам, что позволяет осуществлять задуманное действие.

В научной литературе представлено множество разработок неинвазивных ИМК для управления мехатронными и робототехническими устройствами. Многолетние исследования в этом направлении ведутся в СПБ ФИЦ РАН (СПИИРАН) совместно с СПбПУ Петра Великого [9, 12, 13]. В [9] предложен метод классификации электроэнцефалографических паттернов воображаемых движений пальцев. Работа [12] описывает супервизорный подход к управлению роботом на основе неинвазивного ИМК, обеспечивающий декодирование мозговой активности при воображении управляющих команд. В [13] рассмотрено использование интерфейсов мозг-компьютер в ассистивных технологиях и

приведено сравнение классификаторов на основе метода опорных векторов, искусственных нейронных сетей и римановой геометрии.

В STEM-центре ТУСУРа получены результаты по управлению мобильным роботом ScEdBo и мехатронной рукой [14]. В том же источнике представлен обзор коммерчески доступных нейроинтерфейсов.

В работе [15] описано устройство на базе аналого-цифрового регистратора Arduino Mega 2560, способное распознавать ЭЭГ-сигналы и формировать команды управления роботизированными механизмами, включая бионические протезы, инвалидные коляски и экзоскелеты. Принципы построения нейроуправляемого транспортного средства для лиц с двигательными нарушениями изложены в [16].

Зарубежные публикации по тематике нейроинтерфейсов активно появляются с начала 2000-х годов, при этом существует ряд обзорных работ, посвящённых в том числе ЭЭГ-основным интерфейсам [17–19].

Значительная часть исследований связана с медицинскими применениями ИМК. Рассматриваются возможности их использования в реабилитации пациентов после инсульта [20], при параличе нижних конечностей [21], а также в психиатрической практике [22]. Большое количество экспериментальных исследований посвящено управлению инвалидными колясками при помощи нейроинтерфейсов [23, 24]. В ряде работ [25, 26] проведено сравнение различных алгоритмов машинного обучения для классификации ЭЭГ-сигналов; показано, что наилучшие результаты достигаются при применении метода k ближайших соседей (kNN) и метода опорных векторов (SVM). Однако в отечественной литературе практически отсутствуют описания экспериментов по управлению инвалидными колясками с использованием ИМК; исключение составляют разработки, совмещающие анализ мозговой активности с отслеживанием движений глаз — так называемые интерфейсы «мозг-глаз-компьютер» [27].

Работы по созданию программно-аппаратных комплексов на основе неинвазивных ИМК ведутся также в ИПМаш РАН [28–30]. Совместно с Институтом мозга человека РАН предложены подходы к применению ЭЭГ для

ранней диагностики шизофрении [29], а в сотрудничестве с кафедрой высшей нервной деятельности и психофизиологии СПбГУ — методы анализа самоинициированных движений на основе вызванных потенциалов [30].

Особый интерес представляет использование сигналов нейрообратной связи (НОС, или ЭЭГ-БОС) для управления техническими системами. Данный подход позволяет мозгу человека адаптивно регулировать собственную электрическую активность на основе сенсорной информации о результативности выполняемых действий. В контуре НОС используются данные ЭЭГ, характеризующие изменения потенциалов на поверхности головы. После их предобработки выполняется классификация сигналов методами машинного обучения, на основе чего формируется управляющее воздействие для мехатронных устройств. Обратная связь в виде визуальных стимулов (например, изменение яркости или высоты графического индикатора) позволяет испытуемому корректировать свою мозговую активность. Конкретные параметры ЭЭГ и расположение электродов определяются в зависимости от экспериментальной задачи. Подход к реализации НОС на основе адаптивного моделирования мозговой активности предложен в [31], а возможности моделей, основанных на сетях ФитцХью–Нагумо, исследованы в [32]. Следует отметить, что формирование сигнала нейрообратной связи остаётся сложной задачей, так как отсутствуют чёткие стандарты предъявления стимулов, обеспечивающих оптимальную обучаемость испытуемого.

1.5 Постановка задачи

В данной работе рассматривается построение нейроинтерфейса только на основе неинвазивных средств считывания сигналов мозга (электроэнцефалографов). В качестве решаемой задачи была выбрана задача распознавания воображаемых движений (motor imagery). В рамках этой задачи выполняется распознавание паттернов активности мозга при воображаемых движениях – движение левой рукой и движение правой рукой. В качестве объекта управления используется роботизированное инвалидное кресло. Управление

креслом в данном случае подразумевает его поворот в зависимости от текущей активности мозга пользователя нейроинтерфейса.

Основными задачами данной работы являются:

- описание технических средств нейроинтерфейса, а также общий алгоритм взаимодействия узлов системы управления;

- математическое описание задачи сбора данных и управления оборудованием на основе нейроинтерфейса;

- описание алгоритмов предобработки сигналов ЭЭГ, алгоритмов формирования признаков и классификации намерений испытуемого совершить движение на основе машинного обучения, описание алгоритмов адаптации;

- разработка программного комплекса для реализации системы управления нейроинтерфейсом с учётом разработанных алгоритмов;

- разработка методики проведения тестирования для проверки работоспособности разработанной системы управления и проведение экспериментов по управлению поворотом инвалидного кресла с помощью нейроинтерфейса.

2 ПОСТРОЕНИЕ СИСТЕМЫ УПРАВЛЕНИЯ

2.1 Структура системы управления нейроинтерфейса

Аппаратная часть используемого нейроинтерфейса основана на стандартном медицинском электроэнцефалографе, доступном в коммерческой продаже. Для взаимодействия с роботизированными устройствами применяются типовые каналы беспроводной связи — Wi-Fi или Bluetooth (например, те, что встроены в контроллеры конструктора ТРИК), а также специально разработанный интерфейсный модуль, обеспечивающий сопряжение с инвалидной коляской.

На рисунке 1 представлена структурная схема, иллюстрирующая процесс информационного обмена между основными элементами системы управления и сопряжёнными модулями.

Рисунок 1 – Схематическое изображение взаимодействия составляющих частей системы

Электроэнцефалограф осуществляет регистрацию мозговой активности испытуемого и передаёт данные ЭЭГ на управляющий модуль, где производится их последовательная обработка: фильтрация, выделение признаков, классификация и принятие решения о формировании управляющего воздействия на исполнительный блок инвалидного кресла. При необходимости осуществляется дополнительное обучение классификатора на вновь полученных данных.

Дальнейшие разделы содержат математические постановки ключевых задач, решаемых системой: получение исходных измерений, их предобработка, классификация намерений испытуемого и генерация управляющих сигналов.

2.2 Математическое описание задачи сбора данных и управления с помощью нейроинтерфейса

Приведём математическое описание задачи сбора данных и управления оборудованием на основе нейроинтерфейса. Пусть $X(t)$, $t > 0$ – векторный сигнал ЭЭГ, а $x_{i1}, x_{i2}, \ldots, x_{iN}$, где $i = 1, \ldots, M$ - последовательные результаты измерений сигнала с i-го канала (отведения) электроэнцефалографа (измеряются на самом деле разности потенциалов между текущим и некоторым референтным электродом). Величины $x_{i1}, x_{i2}, \ldots, x_{iN}$ относятся к моментам времени $t_{01}, t_{02}, \ldots, t_{0N}$, где $t_{0j} = t_0 + j\Delta t$, через t_0 обозначено время начала сбора данных, Δt – шаг сбора данных (sampling interval) $j = 1, \ldots, N$. Обычно $t_0 = 0$, $\Delta t = T/N$, где T – величина интервала сбора данных (кадра). Результаты измерений сигнала в следующем кадре обозначаются через $x_{i,N+1}, x_{i,N+2}, \ldots, x_{i,2N}$ и т.д. Время начала k-го кадра обозначается через t_k, так, что $t_{kj} = t_k + j\Delta t$ – моменты измерений в k-м кадре.

Размер кадра T определяется, исходя из того, что, с одной стороны, число отсчетов в кадре должно быть не менее 500-1000, чтобы уменьшить погрешность статистических оценок. С другой стороны, размер кадра T не должен быть большим, чтобы не вносить больших запаздываний в контур управления. В нашей работе было выбрано $T = 2c$, $N = 500$, $\Delta t = 4мс = 0.004\ c$. Близкая модель, основанная на анализе вызванных потенциалов P300 описана в [33].

2.3 Предобработка измерений

Расчет и выдача управляющих воздействий происходит для каждого кадра в моменты $t_k = kT$. Предобработка измерений внутри кадра выполняется по следующему алгоритму.

Для каждого кадра к сигналу $X(t)$ применяется некоторое преобразование G, выполняющее фильтрацию сигнала в заданном диапазоне частот: $g(t)= G(t_k,x(t), \omega_1, \omega_2)$, где $g(t)$ – сигнал после выполнения предобработки сигнала $x(t)$, $t_{k-1} < t < t_k$, ω_1, ω_2 – нижняя и верхняя границы диапазона фильтрации, соответственно. В нашей работе были выбраны значения $\omega_1= 8$ *Гц* и $\omega_2= 30$ *Гц*. Такой диапазон фильтрации выбран исходя из результатов работ на схожую тематику [34, 35], а также в следствие собственных экспериментов – такой диапазон позволял получить лучшие результаты.

К фильтру предъявляется условие гладкости амплитудно-частотной характеристики (АЧХ). В качестве фильтра был выбран фильтр Баттерворта, т.к. его АЧХ максимально гладкая на частотах полосы пропускания и снижается практически до нуля на частотах полосы подавления. В общем случае фильтр 2-го порядка описывается следующим уравнением [36, 37]:

$$Y(t_{kj}, j) = b_0X(t_{kj}) + b_1X(t_{kj-1}) + b_2X(t_{kj-2}) + a_1Y(t_{kj-1}) + a_2Y(t_{kj-2}), \qquad (1)$$

где $X(t_{kj})$ – входные данные, $Y(t_{kj})$ – выходные данные, a и b – весовые коэффициенты фильтра, $j = 1, 2, \ldots N$ – номер отсчета в кадре, $t_{kj} = t_k + j\Delta t$. В промежутках между отсчетами считаем выходной сигнал фильтра постоянным: $g(t) = g(t_{kj})$ при $t_{kj-1} < t < t_{kj}$. Начальные условия фильтра задаются нулевые.

2.4 Классификация намерений совершить воображаемое движение левой или правой рукой

После завершения этапа предобработки сигналов следует процедура анализа и распознавания (классификации) измеренных ЭЭГ-сигналов с последующей выдачей управляющих воздействий. Процесс начинается с подачи пользователю

акустического или визуального стимула, указывающего направление предполагаемого движения транспортного средства: влево или вправо. В экспериментальных исследованиях использовались звуковые сигналы — короткий импульс длительностью 0,5 с для команды «влево» и более длинный (2 с) — для команды «вправо».

После подачи стимула выполняется анализ предобработанного сигнала ЭЭГ $g(t)$, вычисляются значения вектора признаков, служащие основой для обучения и последующего распознавания намерений оператора.

Следующим этапом является обучение классификатора, которое направлено на определение принадлежности текущего вектора признаков к одному из двух классов — «влево» или «вправо». Обработка выполняется на наборе кадров (в эпохе содержится S кадров), включающих SN измерений, до момента поступления следующего стимула. Для реализации классификации применяются известные алгоритмы машинного обучения, включая метод опорных векторов (SVM), k-ближайших соседей (kNN) и случайный лес (RF) [38–40].

Во многих практических задачах классы признаков не оказываются линейно разделимыми, что усложняет построение разделяющей гиперплоскости и требует использования модифицированных методов. В данной работе в качестве базового классификатора применялся метод k-ближайших соседей, а также его усовершенствованная версия — метод нечетких почти ближайших соседей, который подробно рассмотрен в следующем разделе.

2.5 Метод ближайших соседей и его модификация

Напомним, что в стандартном методе ближайших соседей (kNN) из выборки S кадров формируется массив (выборка) из SN M-мерных векторов $x_i \in R^M$, $i = 1, \ldots, SN$, где N – число измерений в кадре (в нашем случае $N=500$)**.** Среди них могут быть векторы из класса А (поворот налево) и из класса В (поворот направо). При появлении нового (тестового) вектора y вычисляются расстояния от y до всех остальных векторов выборки и выбираются k ближайших к y векторов. Для удобства распознавания k берется нечетным. Считается, что вектор y

принадлежит классу A, если число векторов $\boldsymbol{x_i}$ из класса A среди ближайших больше, чем число векторов из класса B. В противном случае считается, что вектор y принадлежит классу B. Обычно рекомендуется брать $3 < k < 11$.

Достоинством метода kNN является его простота, а недостатками – большой объем вычислений, выполняемых нерекуррентно и зависимость результатов от выбора числа k. Метод может быть очень чувствителен к выбору k. Например, при выборе $k = 9$, если среди ближайших соседей есть пять векторов из класса A и четыре вектора из класса B, то тестовый вектор будет отнесен к классу A. Однако, если увеличить число k до $k = 11$ и оба из несколько более далеких дополнительных векторов окажутся принадлежащими классу B, то и тестовый вектор будет отнесен к классу B.

Также достоинством метода kNN является и то, что он не требует разделимости классов A и B гиперплоскостью или какой-то другой простой поверхностью. Важно, что в задачах распознавания сигналов ЭЭГ предположение о разделимости классов, по-видимому, несправедливо и поэтому в экспериментах по анализу ЭЭГ метод kNN зачастую дает более высокий процент верно распознанных тестовых векторов. Эта закономерность подтверждается и в проведённых экспериментах, что обуславливает выбор метода kNN для дальнейших исследований.

Предлагаемая модификация метода kNN основана на введении так называемых *радиально нечетких векторов*. Радиально нечетким (R-нечетким) вектором будем называть нечеткое множество $X \subset R^n$, имеющее функцию принадлежности $d_X(x) = R(||x-y||/f)$, где $R(a)$ – невозрастающая скалярная функция на $[0,\infty)$, такая, что $R(0) = 1$, $R(1) = 0$, $R(a) = 0$ при $a > 1$. Например, можно взять $R(a) = 1 - a$ при $0 \le a \le 1$, $R(a) = 0$ при $a > 1$. Вектор $y \in R^n$ называется центром радиально нечеткого вектора X, а число $f > 0$ - коэффициентом нечеткости. Очевидно, степень принадлежности радиально нечеткого вектора определяется расстоянием от него до заданного центра. Мы будем рассматривать формально более общий случай, когда ключевую роль играет расстояние не до центра, а до

некоторого шара, имеющего тот же центр, а именно, до шара, описанного вокруг множества ближайших соседей.

Модификация метода kNN состоит в том, что все векторы выборки x_i считаются R-нечеткими векторами с функциями принадлежности

$$d_i(x) = \mathrm{R}((\big|\|\mathrm{x} - \mathrm{x_i}\|\big| - \mathrm{R}_k)_+ / f)), \qquad (2)$$

где R_k, f – параметры алгоритма (положительные числа), $(.)_+$ - положительная часть числа, равная самому числу, если оно положительно и нулю в противном случае.

Алгоритм метода начинает работать так же, как и алгоритм обычного kNN. Задается целое число k, при поступлении нового тестового вектора y находится $N_k(y)$ - множество k ближайших кy векторов выборки x_i. Затем определяется величина R_k, равная максимуму $||y-x_i||$ по всем $x_i \in N_k(y)$. Очевидно, R_k – радиус шара, описанного вокруг множества ближайших соседей тестового вектора y. Далее вычисляются веса w_i векторов выборки, равные значениям их функций принадлежности $d_i(y)$, см. (2). Очевидно, веса ближайших соседей равны 1, а веса векторов, расположенных от тестового вектора на расстоянии больше, чем R_k+f равны нулю. Т.е. достаточно вычислять веса векторов, расположенных от тестового вектора на расстоянии от R_k до R_k+f. Наконец, сравниваются суммы весов векторов из класса A и из класса B. Тестовый вектор относится к тому классу, сумма весов которого больше.

Таким образом, предлагаемый вариант алгоритма представляет собой расширение метода kNN, при котором, кроме ближайших соседей, в расчёт принимаются так называемые «почти соседи» — элементы обучающей выборки, находящиеся от тестового вектора на расстоянии, не превышающем радиус шара ближайших соседей более чем на заданную величину f. Вклад таких элементов взвешивается обратно пропорционально расстоянию: чем дальше объект от границы шара, тем меньший вес он имеет при классификации.

Если параметр нечеткости *f* пренебрежимо мал и множество «почти соседей» отсутствует, алгоритм полностью совпадает с классическим kNN. Таким образом, полученный метод можно рассматривать как «метод нечетких почти ближайших соседей» (Fuzzy Almost Nearest Neighbors, FANN). Он естественным образом обобщается на многоклассовые задачи классификации.

Следует отметить, что в литературе имеется значительное количество исследований, посвящённых «нечетким» модификациям метода kNN [41, 42]. Однако большинство таких работ, начиная с классической публикации [41] (которая имеет более 3000 цитирований в Google Scholar), базируются на идее нечеткой классификации, когда каждой точке ставится в соответствие функция принадлежности к классу, определяемая эвристически как функция расстояния между векторами выборки и тестовым объектом.

В разработанном подходе принципиально сохраняется четкий характер классификации — итоговое решение совпадает с результатом метода kNN при малой величине параметра *f*.

Сравнение точности классификации для методов kNN и FANN для разных значений *k* представлено на рисунке 2.

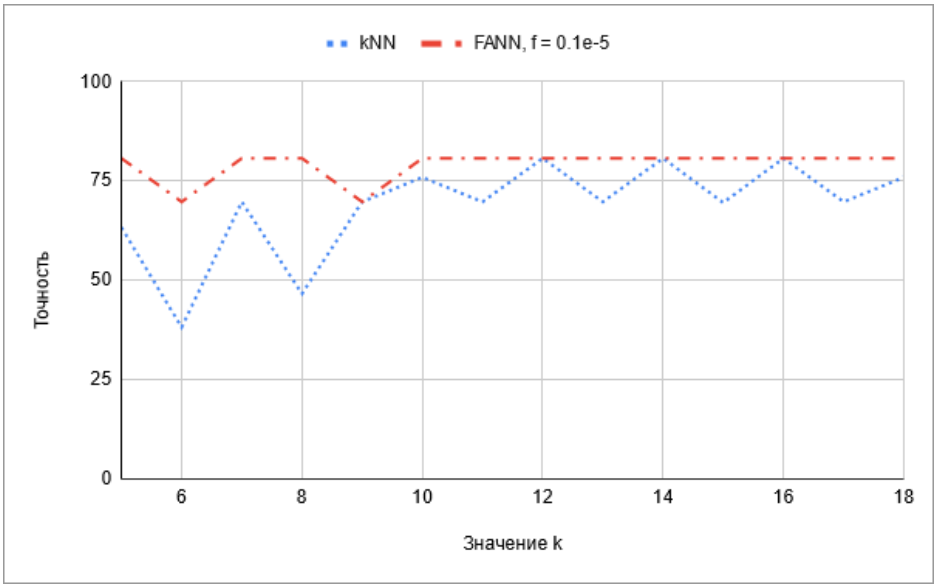


Рисунок 2 – Сравнение точности классификации для разных значений *k* для методов kNN и FANN

Ключевым преимуществом метода FANN по сравнению с классическим kNN является меньшая чувствительность к выбору параметра k и устойчивое достижение высокой точности классификации практически при любых его значениях.

В конце каждой эпохи, завершающейся подачей нового звукового сигнала, проводится анализ обученной системы. А именно, по значению $H(t_k, g)$, вычисляется матрица ошибок $E_1(t_k) = ||e_{ab}(t_k)||$, a,b = 1,2 предсказания того, к какому классу относится сигнал. Вычисления проводятся следующим образом: элементы матрицы $E_1(t_k)$, расположенные на главной диагонали, равны числу правильно классифицированных значений сигнала $\boldsymbol{g(t)}$ за предыдущую эпоху, т.е. при $t_{k-S} < t < t_k$. Иначе говоря, $e_{1,11}(t_k)$ – число верно распознанных испытуемым поворотов налево, а $e_{1,22}(t_k)$ – число верно распознанных испытуемым поворотов направо. Соответственно, $e_{1,12}(t_k)$ и $e_{1,21}(t_k)$ – количества неверно распознанных поворотов.

2.6 Алгоритм классификации ЭЭГ с учетом дрейфа параметров

Опыт исследований по управлению движением инвалидных колясок через нейроинтерфейсы показывает [33], что в ходе тренировки происходит не только обучение системы управления, адаптирующейся к свойствам головного мозга и двигательной системы пациента, но и адаптация человека к особенностям работы при взаимодействии с компьютерной системой через нейроинтерфейс. В данной работе делается попытка учесть одновременно оба процесса адаптации в разрабатываемой системе управления.

Основой для изучения процесса адаптации человека является адаптивная модель принятия им решений в конкретных ситуациях. Поскольку в данной работе рассматривается задача управления поворотами транспортного средства, мы используем адаптацию решающего правила, моделирующего реакцию человека как вычисление линейной комбинации некоторых переменных (признаков) и действие на основе знака данной линейной комбинации. Таким образом, линейная комбинация параметров задает гиперплоскость в пространстве

переменных модели и решение о повороте налево или направо принимается в зависимости от того, с какой стороны от гиперплоскости находится текущий вектор переменных модели. Для реализации процесса взаимной адаптации для каждой эпохи на этапе обучения классификатора производится выполнение шага вычисления параметров гиперплоскости, разделяющей области принятия того или иного решения в пространстве признаков.

Обозначим через x_k набор значений признаков на k-м шаге (эпохе) процесса. Вектор коэффициентов разделяющей гиперплоскости на k-м шаге обозначим через Θ_k, $k = 1, 2, \ldots$ В качестве алгоритма машинного обучения классификации через адаптацию параметров Θ_k можно взять один из существующих алгоритмов: МНК, SVM и др. При этом задача обучения сводится к нахождению с некоторой точностью «идеального» вектора решения Θ_k, обеспечивающего хорошее качество классификации, например, ее оптимальность в том или ином смысле. В данной работе в качестве базового алгоритма берется один из наиболее простых и эффективных, а именно, алгоритм обучения по методу рекуррентных целевых неравенств (МРЦН) В.А. Якубовича [43].

В настоящей работе предлагается распространить алгоритмы решения РЦН на случай, когда «идеальный» набор параметров разделяющей гиперплоскости дрейфует во времени из-за адаптационных процессов в организме человека-пациента. Предполагается, что дрейф параметров происходит достаточно медленно и переход от каждой эпохи к следующей описывается линейным законом изменения «идеального» вектора Θ_{*k} параметров разделяющей гиперплоскости:

$$\Theta_{*k} = \Theta_{*0} + a_k, \qquad (3)$$

где Θ_{*0}– начальное значение вектора параметров, a – поправка, задающая направление и скорость изменения (дрейфа) параметров Θ_{*k}. За основу берется стандартный алгоритм «полоска» метода РЦН [44]:

$$\Theta_{k+1} = \Theta_k - \gamma \frac{(\gamma_k - \Theta_k^T x_k)}{\|x_k\|^2} x_k, \quad (4)$$

где Θ_k– вектор оценок параметров разделяющей гиперплоскости на k-м шаге процесса обучения, $\gamma > 0$ – коэффициент усиления (параметр алгоритма). Предполагается, что измеряемый выход (решение человека) на k-м шаге y_k связан с входами и параметрами линейной моделью с возмущением

$$y_k = \Theta_*^T x_k + \varphi_k, k = 1, 2, \ldots \quad (5)$$

Возмущение φ_k может порождаться неточностью модели, погрешностями измерения и другими причинами. Будем считать, что φ_k ограничено: $|\varphi_k| < \Delta\varphi$, где $\Delta\varphi > 0$. Ставится задача поиска решения при достаточно больших k системы целевых неравенств

$$|y_k - \Theta^T x_k| < \Delta, \quad (6)$$

где $\Delta > 0$ – параметр задачи, θ– искомый вектор решения. Очевидно, для разрешимости задачи необходимо выполнение неравенства $\Delta > \Delta\varphi$.

Формулу (4) применяют при нарушении неравенства

$$\left|y_k - \Theta_k^T x_k\right| < \Delta. \quad (7)$$

При выполнении (7), т.е. при выполнении ЦУ в текущий момент времени, коррекции оценок не производится, т.е. $\varphi_{k+1} = \varphi_k$. В.А. Якубович в работе [44] предложил алгоритм «полоска» (4) и установил условия, при которых алгоритм обеспечивает достижение цели (6).

В существующих работах считается, что вектор «идеального» решения постоянен: $\varphi_{*k} \equiv \varphi_*$. Однако в нашем случае вектор Θ_* меняется в соответствии с законом дрейфа (3), в котором вектор параметров дрейфа a неизвестен. Ниже

предлагается обобщение алгоритма «полоска» на случай переменного φ_* включающий блок оценки скорости дрейфа a.

Для описания предлагаемого алгоритма введем обозначения:

$$\overline{\Theta}_* = \begin{bmatrix} \Theta_{*0} \\ a \end{bmatrix}, \overline{x}_k = \begin{bmatrix} x_k \\ kx_k \end{bmatrix}, \overline{\Theta}_k = \begin{bmatrix} \Theta_k \\ a_k \end{bmatrix}, \tag{8}$$

где a_k– текущая оценка скорости дрейфа a. Модель реакции человека (5) для введенных обозначений принимает вид

$$y_k = \overline{\Theta}_*^T \overline{x}_k + \varphi_k, \tag{9}$$

а алгоритм оценивания параметров решения и скорости дрейфа решения имеет вид

$$\overline{\Theta}_{k+1} = \overline{\Theta}_k - \gamma \frac{\left(\gamma_k - \overline{\Theta}_k^T \overline{x}_k\right)}{\|\overline{x}_k\|^2} \overline{x}_k. \tag{10}$$

Алгоритм (10) можно записать в виде

$$\begin{cases} \Theta_{k+1} = \Theta_k - \gamma \frac{(\gamma_k - \Theta_k^T x_k - a_k^T x_k k)}{\|x_k\|^2 (1+k^2)} x_k, k = 1, 2, \dots \\ a_{k+1} = a_k - \gamma \frac{(\gamma_k - \Theta_k^T x_k - a_k^T x_k k)}{\|x_k\|^2 (1+k^2)} x_k k, k = 1, 2, \dots \end{cases} \tag{11}$$

Соотношения (11) применяются при нарушении неравенства

$$\left| y_k - \Theta_k^T x_k - a_k^T x_k k \right| < \Delta. \tag{12}$$

В противном случае, т.е. при выполнении (12) оценки не меняются:

$$\Theta_{k+1} = \Theta_k, a_{k+1} = a_k. \quad (13)$$

Величины $\gamma > 0$, $\Delta > 0$ являются параметрами алгоритма. Условия, гарантирующие сходимость алгоритма (11) и достижение цели (12) за конечное число шагов, выводятся из результатов [44]. Однако теоретические условия трудны для проверки и часто не выполняются на практике. Поэтому подбор параметров алгоритма выполняется экспериментально.

2.7 Определение наличия особых паттернов активности

Помимо поворота влево или вправо требуется также реализация движения транспортного средства вперёд. Это движение может быть достигнуто за счёт активности мозга, возникающей в результате возникновения воображаемых движений обеими руками.

Как и в случае с распознаванием воображаемых движений левой и правой рукой, распознавание воображаемых движений обеими руками строится на классификации сигналов ЭЭГ. В данном случае также выделяется два класса – первый соответствует сигналам ЭЭГ, возникающим при непосредственно воображаемых движениях обеими руками одновременно, а второй – сигналам, соответствующим фоновой активности мозга (воображаемые движения не совершаются).

На вход классификатора поступает предобработанный кадр сигнала ЭЭГ *g(t)*. Для каждого кадра определяются значения вектора признаков, используемых далее для обучения системы и распознавания наличия воображаемого движения обеими руками.

Разметка набора данных для обучения классификатора выполняется в автоматическом режиме во время записи ЭЭГ. С помощью звуковых сигналов испытуемому сообщается факт начала и конца периода времени, в течение которого он должен пытаться совершать воображаемые движения обеими руками. Этот период обычно длится примерно 15-20 секунд, прерывается на 15-20 секунд,

а затем начинается снова. Во время перерывов испытуемый не совершает воображаемых движений, записывается фоновая активность мозга.

Таким образом, в результате классификации сигналов ЭЭГ, как и в случае с классификацией воображаемых движений левой или правой рукой, можно получить матрицу ошибок – E_2. В этой матрице $e_{2,11}(t_k)$ – число верно распознанных воображаемых движений обеими руками, а $e_{2,22}(t_k)$ – число верно распознанных состояний отсутствия таких движений. Соответственно, $e_{2,12}(t_k)$ и $e_{2,21}(t_k)$ – количества неверно распознанных состояний.

2.8 Классификация состояний мозга с использованием ритмов головного мозга

Чтобы обеспечить полноценное управление транспортным средством также необходима реализация возможности его остановки с помощью активности мозга.

Дополнительным источником информации о состоянии головного мозга могут послужить ритмы головного мозга. Эти ритмы – набор особых колебаний электрической активности мозга, каждое из которых имеет свой диапазон частот и предназначение. Анализ совокупности определённых ритмов может помочь определить состояние человека. В контексте решаемой задачи управления транспортным средством особый интерес представляют альфа-, бета- и гамма-ритмы. Например, когда человек насторожен, альфа-волны замещаются низковольтными нерегулярными быстрыми колебаниями. Увеличение бета-активности при снижении активности альфа-ритма может свидетельствовать о росте психоэмоционального напряжения, появлении тревожных состояний. Повышенное сосредоточение внимания обычно сопровождается повышением активности гамма-ритма. Также следует учитывать, что каждый из ритмов локализуется в определённых областях мозга. Поэтому особое внимание необходимо уделить правильному выбору отведений электроэнцефалографа для анализа сигналов ЭЭГ. Альфа-ритм обычно регистрируется в затылочной и теменной областях, бета-ритм – в центральной и лобной, а гамма-ритм – во фронтальной и теменной областях.

Таким образом, значения активности этих ритмов в совокупности могут сигнализировать о том, что человек хочет остановить движение, например, если представит, что перед ним препятствие.

Для того, чтобы обеспечить определение этого состояния с помощью анализа значений активности ритмов мозга, необходим отдельный классификатор.

Для начала необходимо полученные после предобработки сигналы ЭЭГ (в рамках одного кадра) $g(t)$ перевести в частотную область с помощью быстрого преобразования Фурье. По полученным спектральным характеристикам $S_k(\omega)$ для каждого из выбранных отведений можно судить о выраженности того или иного ритма головного мозга. Пример графика спектральной плотности для альфа-, бета- и гамма ритмов представлен на рисунке 3.

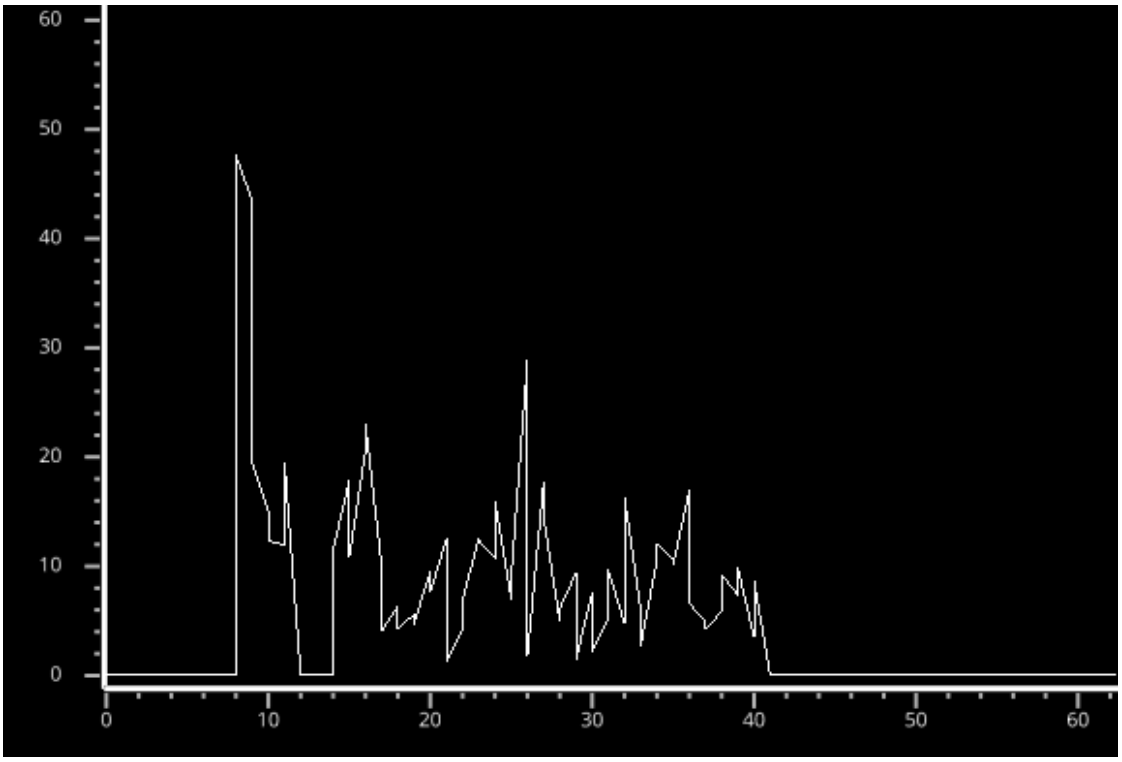


Рисунок 3 – Пример графика зависимости спектральной плотности (вертикальная ось) от частоты (горизонтальная ось)

В данном случае в качестве классификатора также может быть использован любой из известных алгоритмов машинного обучения, например, описанная ранее модификация метода ближайших соседей – FANN. На вход классификатору подаются значения спектральной плотности сигнала, соответствующие

выбранным частотным диапазонам: для альфа-ритма $8 \leq \omega_\alpha \leq 11$, для бета-ритма $14 \leq \omega_\beta \leq 35$, для гамма-ритма $30 \leq \omega_\gamma \leq 40$.

Обучающими данными для классификатора служат два типа состояний ритмов головного мозга. Первый тип – значения, характерные для состояния, когда испытуемый хочет остановить движение, представляя препятствие перед собой (класс A). Второй тип – когда испытуемый находится в относительно спокойном состоянии или управляет транспортным средством с помощью воображаемых движений руками (класс B).

Разметка набора данных для обучения классификатора в данном случае производится вручную, так как в экспериментах часто бывает сложно автоматизировать отслеживание достижения обозначенного момента, когда испытуемый должен представить препятствие.

Таким образом, в результате классификации состояния мозга по активности ритмов, как и в случае с классификацией воображаемых движений, можно получить матрицу ошибок – E_3. В этой матрице $e_{3,11}(t_k)$ – число верно распознанных намерений остановить транспортное средство, а $e_{3,22}(t_k)$ – число верно распознанных состояний отсутствия таких намерений. Соответственно, $e_{3,12}(t_k)$ и $e_{3,21}(t_k)$ – количества неверно распознанных состояний.

2.9 Принятие решения об отправке управляющего сигнала

Для принятия решения о том, какой управляющий сигнал следует отправить на объект управления, используются полученные матрицы ошибок E_1, E_2 и E_3.

Для принятия решения об отправке соответствующего управляющего сигнала необходимо вычислить долю верно распознанных участков сигнала в общем числе сигналов для каждой матрицы ошибок.

Для задачи поворота инвалидного кресла влево или вправо $P_L(t_k) = e_{1,11}(t_k) / S$, и $P_R(t_k) = e_{1,22}(t_k) / S$, где $0 \leq P_L \leq 1$ – вероятность наступления события, состоящего в том, что система верно распознала сигнал поворота налево, $0 \leq P_R \leq 1$ – частота наступления события, состоящего в том, что система верно распознала сигнал поворота направо, S – число элементов сигнала в кадре. Тогда $\Delta P = P_L - P_R$.

Если $\Delta P > P_1$, то сигнал управления y_i принимает значение 1, если $\Delta P < P_2$ то y_i принимает значение -1, что соответствует повороту влево и вправо соответственно, P_1 и P_2 – некоторые пороговые значения. Если ни одно из этих условий не выполняется (состояние неопределённости), то $y_i = 0$. В данной работе $P_1 = 0.2$ и $P_2 = -0.2$.

Для задачи движения инвалидного кресла вперёд $P_F(t_k) = e_{2,11}(t_k) / S$, где $0 \leq P_F \leq 1$ – вероятность наступления события, состоящего в том, что система верно распознала сигнал движения вперёд. Если $P_F > P_3$, то сигнал управления y_i принимает значение 2. В данной работе $P_3 = 0.85$.

Для задачи остановки инвалидного кресла $P_S(t_k) = e_{3,11}(t_k) / S$, где $0 \leq P_S \leq 1$ – вероятность наступления события, состоящего в том, что система верно распознала сигнал остановки движения. Если $P_S > P_3$, то сигнал управления y_i принимает значение 3. В данной работе $P_4 = 0.8$.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ УПРАВЛЕНИЯ

3.1 Программная платформа для проведения исследований

В прикладных задачах цифровой обработки эффективность непосредственного использования «сырых» данных ЭЭГ невысока. Для упрощения процесса исследований и проведения экспериментов была разработана специализированная программная платформа. Архитектура этой платформы представлена на рисунке 4.

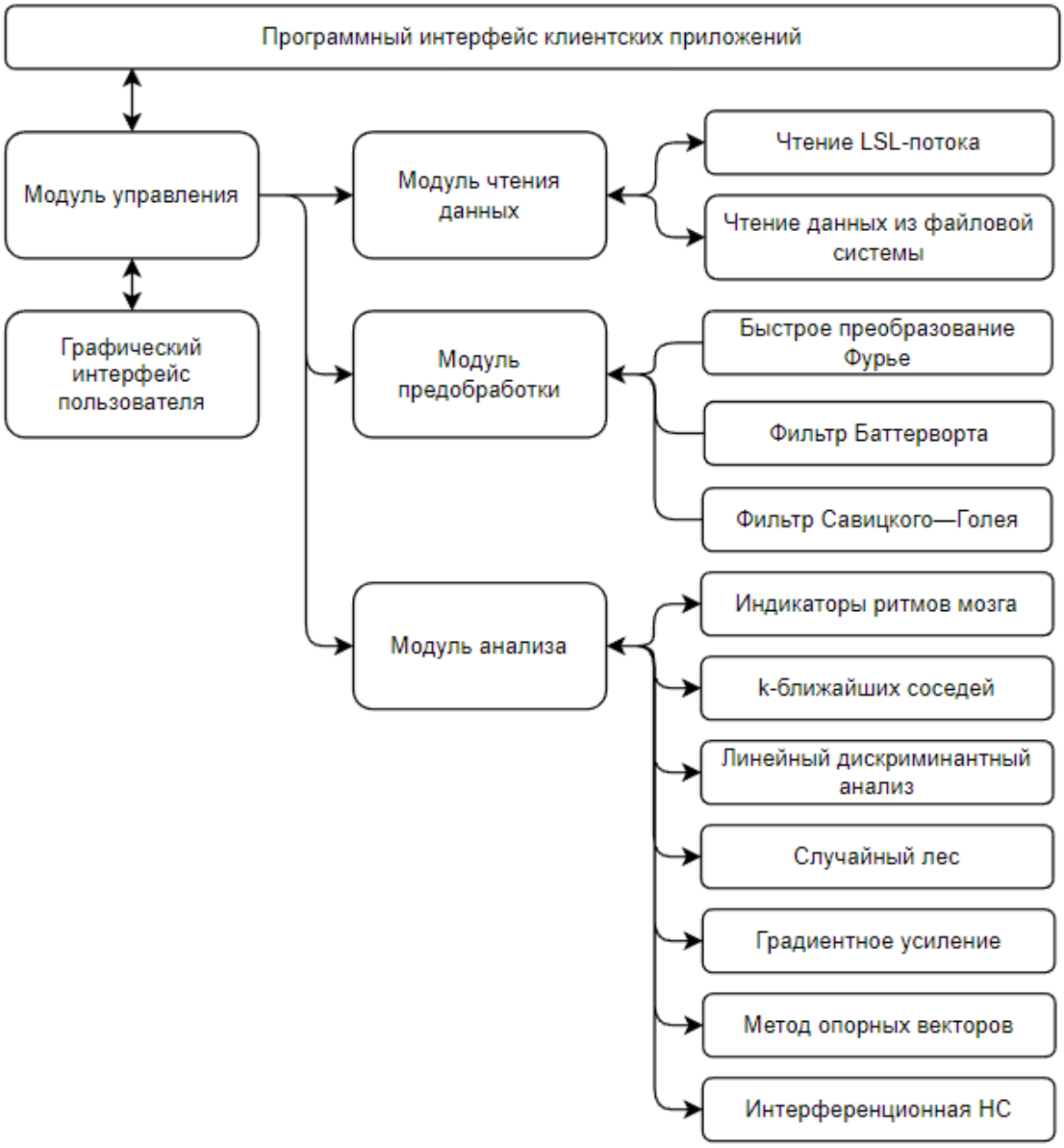


Рисунок 4 – Архитектура программной платформы

Для анализа данных ЭЭГ требуется широкий набор алгоритмов предобработки и анализа сигналов. Разработанная платформа включает инструменты автоматизированной обработки и анализа ЭЭГ-сигналов, включая методы машинного обучения [45].

На структурной схеме отображены модули управления, чтения, предобработки и анализа данных, а также вспомогательные компоненты. Модуль управления обеспечивает координацию взаимодействия всех элементов системы. Модуль чтения данных предназначен для работы с ЭЭГ-информацией в различных форматах, включая потоковое считывание напрямую с электроэнцефалографа посредством LSL-потока. Подробное описание остальных компонентов приведено в следующих разделах.

3.2 Алгоритмы предобработки ЭЭГ

Под предобработкой ЭЭГ понимают совокупность фильтров и алгоритмов, направленных на подготовку сигналов к дальнейшему анализу. Поскольку зарегистрированный сигнал ЭЭГ часто содержит артефакты, их устранение является ключевым этапом. Артефакт представляет собой любую компоненту сигнала, не связанную с мозговой активностью. Несмотря на существование множества алгоритмов удаления артефактов, вопрос их эффективного подавления остаётся актуальным.

Источниками артефактов могут выступать как технические причины — неисправные электроды, линейные наводки, повышенное сопротивление контактов, — так и физиологические процессы: моргание, движения глаз, активность мышц и сердца. Последние труднее поддаются устранению, так как могут имитировать когнитивную активность, искажая интерпретацию данных. Это особенно критично при диагностике неврологических и психофизиологических состояний, например болезни Альцгеймера или расстройств сна. Поэтому корректное удаление артефактов является необходимым условием для практического использования ЭЭГ.

Далее приведено описание методов предобработки, реализованных в программной платформе.

3.2.1 Фильтр Баттерворта

Фильтр Баттерворта относится к числу фильтров, обеспечивающих максимально ровную амплитудно-частотную характеристику в полосе пропускания [37]. Он лишён пульсаций как в полосе пропускания, так и в полосе подавления, что обусловливает его определение как «максимально плоского» фильтра. Достижение такой характеристики обеспечивается за счёт плавного перехода между полосами, при этом сохраняются средние переходные свойства.

3.2.2 Фильтр Савицкого–Голея

Фильтр Савицкого–Голея представляет собой цифровой фильтр, предназначенный для обработки дискретных данных [46]. Его применение позволяет выделять полезные сигналы из зашумлённых данных без существенного искажения их формы при корректной настройке параметров. Это достигается с помощью полиномиальной аппроксимации отдельных участков входного сигнала по критерию минимизации среднеквадратичной ошибки. В отличие от фильтра скользящего среднего, в котором данные усредняются равномерно, метод Савицкого–Голея использует взвешенное усреднение, что позволяет точнее сохранять форму сигнала.

3.2.3 Быстрое преобразование Фурье

Быстрое преобразование Фурье (БПФ) является эффективным алгоритмом вычисления дискретного преобразования Фурье (ДПФ) [47]. Его основное преимущество заключается в существенном снижении вычислительных затрат по сравнению с классическим ДПФ. БПФ широко применяется в задачах цифровой обработки сигналов, спектрального анализа, сжатия данных, решения дифференциальных уравнений и численного моделирования. Спектральная

характеристика, получаемая с помощью БПФ, служит важным инструментом при исследовании частотных свойств сигналов и распределения мощности по спектру.

3.3 Алгоритмы анализа

После прохождения этапа предобработки ЭЭГ-сигналы становятся пригодными для анализа. Под анализом понимается применение набора методов, направленных на решение конкретной задачи: классификация, кластеризация, построение прогнозных моделей, выделение признаков и др.

3.3.1 Индикатор наличия ритмов головного мозга

Одной из важнейших задач анализа является определение мозговых ритмов — периодических колебаний электрической активности мозга, характеризующих различные функциональные состояния.

Альфа-ритм имеет диапазон частот 8–14 Гц и амплитуду 30–70 мкВ; он наиболее выражен при закрытых глазах в затылочной области. Бета-ритм (13–30 Гц, 5–30 мкВ) наблюдается при активной деятельности и преимущественно выражен в лобных областях. Гамма-ритм охватывает диапазон 30–120(180) Гц, с амплитудой 2–10 мкВ, и связан с процессами концентрации внимания; нарушения его активности характерны для больных шизофренией. Дельта-ритм (0,5–4 Гц, 50–500 мкВ) возникает при глубоком сне и коматозных состояниях, а тета-ритм (4–8 Гц, 10–30 мкВ) соответствует состоянию расслабления и медитации.

Индикатор ритма — это численная величина, позволяющая судить о присутствии того или иного ритма [48]. В рассматриваемом комплексе реализовано определение альфа-, тета- и дельта-ритмов. Для вычисления индикатора производится выделение требуемого диапазона частот с помощью фильтрации, анализ частотной характеристики и последующее сглаживание с использованием фильтра Савицкого–Голея.

Путём разбиения полученной кривой на три равные области и вычисления площадей под ними формируется индикаторное значение по формуле:

$$R = 1 - \frac{S_0 + S_2}{2S_1}, \quad (13)$$

где S_0, S_1 и S_2 – площади левой, средней и правой областей соответственно. Полученное значение индикатора $R \leq 1$ позволяет оценить наличие того или иного ритма. Чем значение ближе к единице, тем с большей вероятностью можно утверждать, что ритм присутствует. Значение в окрестности нуля и меньше свидетельствует о том, что ритма, скорее всего, нет.

Рисунки 5 и 6 демонстрируют сглаженные частотные характеристики сигналов ЭЭГ для состояний с закрытыми и открытыми глазами соответственно.

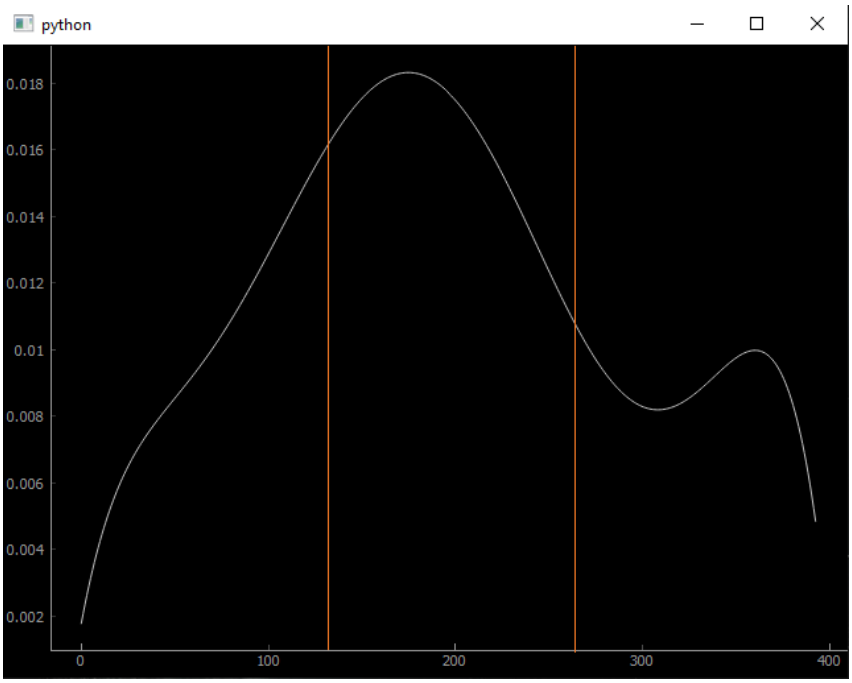


Рисунок 5 – Сглаженная частотная характеристика, разделённая на три зоны, глаза закрыты; горизонтальная ось – отсчёты сигнала, вертикальная – мощность сигнала

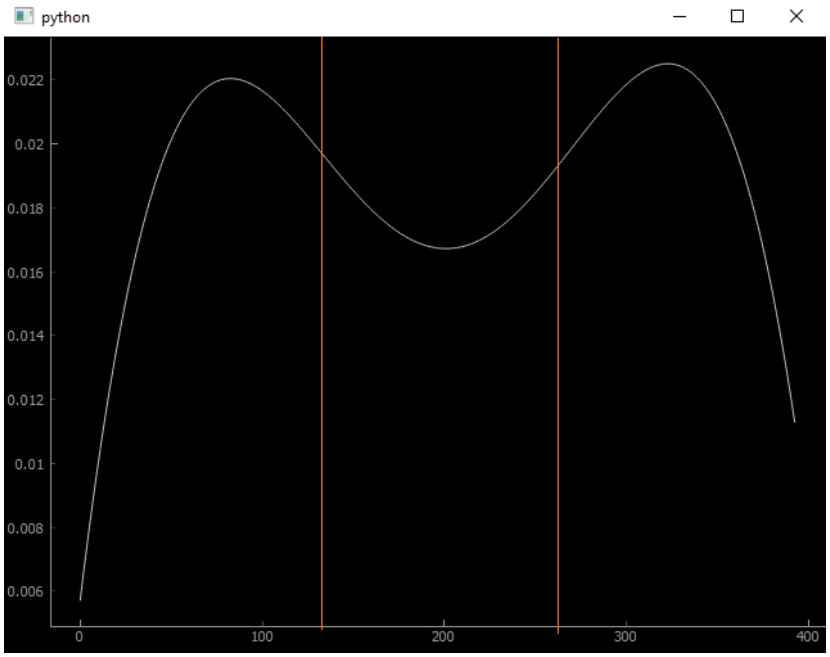


Рисунок 6 – Сглаженная частотная характеристика, разделённая на три зоны, глаза открыты; горизонтальная ось – отсчёты сигнала, вертикальная – мощность сигнала

3.3.2 Градиентное усиление (LGBM)

Градиентное усиление представляет собой метод машинного обучения, применяемый для задач классификации и регрессии [49]. Он строит итоговую модель в виде ансамбля слабых предсказателей (обычно деревьев решений), которые итеративно объединяются в одну сильную модель. Данный метод позволяет оптимизировать произвольную дифференцируемую функцию потерь и обеспечивает высокую точность прогнозирования.

3.3.3 Интерференционная нейросетевая модель

Интерференционная модель нейронной сети (НС) существенно отличается от традиционных архитектур искусственных НС [50]. Её структура приближена к биологическим сетям головного мозга: каждый нейрон является самоорганизующейся системой, обучение которой происходит за счёт смещения рецепторов под действием нейромедиаторов, выделяемых синапсами.

Обработка сигналов осуществляется последовательно во времени, что снижает объём данных, необходимых для обучения, и уменьшает требования к памяти — достаточно сохранять только конечные координаты рецепторов и длины их траекторий. Для корректной классификации достаточно одного нейрона на класс. Модель легко поддаётся дообучению без необходимости повторного обучения всей сети.

Эта архитектура показала высокую эффективность в задачах распознавания изображений и может быть успешно применена для классификации данных ЭЭГ [51–54].

Помимо описанных методов, платформа поддерживает и другие алгоритмы: линейный дискриминантный анализ, случайный лес, метод k ближайших соседей, FANN и метод опорных векторов (SVM).

3.4 Особенности программной реализации

Программный комплекс для чтения, обработки и анализа ЭЭГ-сигналов обладает рядом особенностей, связанных с работой в реальном времени. Во-первых, в нём реализована многопоточность, позволяющая выполнять параллельную обработку сигналов и существенно снижать временные задержки. Во-вторых, визуализация графиков осуществляется с использованием графического процессора, что обеспечивает высокую скорость отображения данных.

Платформа оснащена удобным графическим интерфейсом, позволяющим пользователю настраивать параметры эксперимента и выбирать алгоритмы анализа. Это делает систему доступной даже для специалистов, не имеющих опыта программирования. На рисунке 7 представлен внешний вид интерфейса.

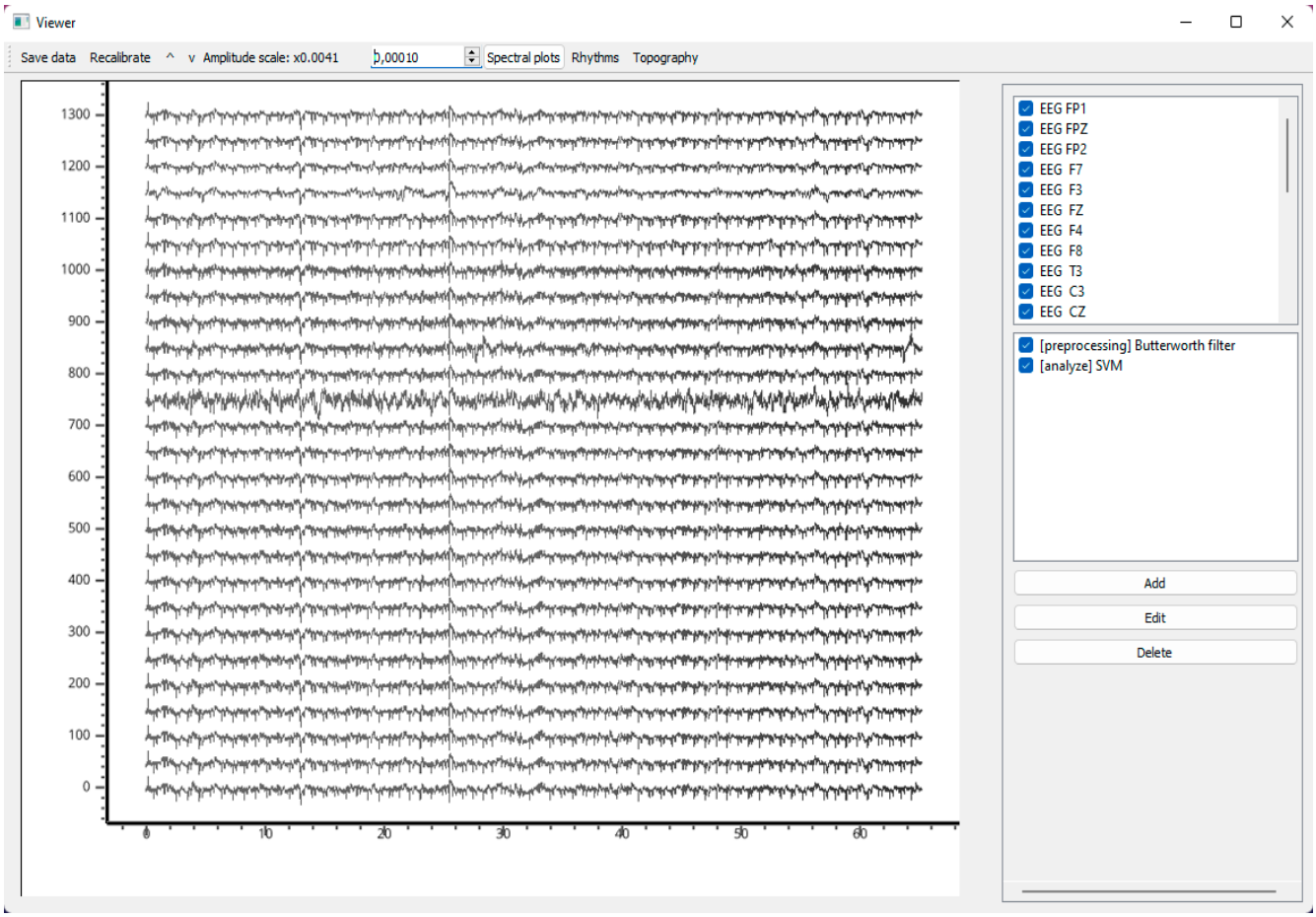


Рисунок 7 – Графический интерфейс программной платформы

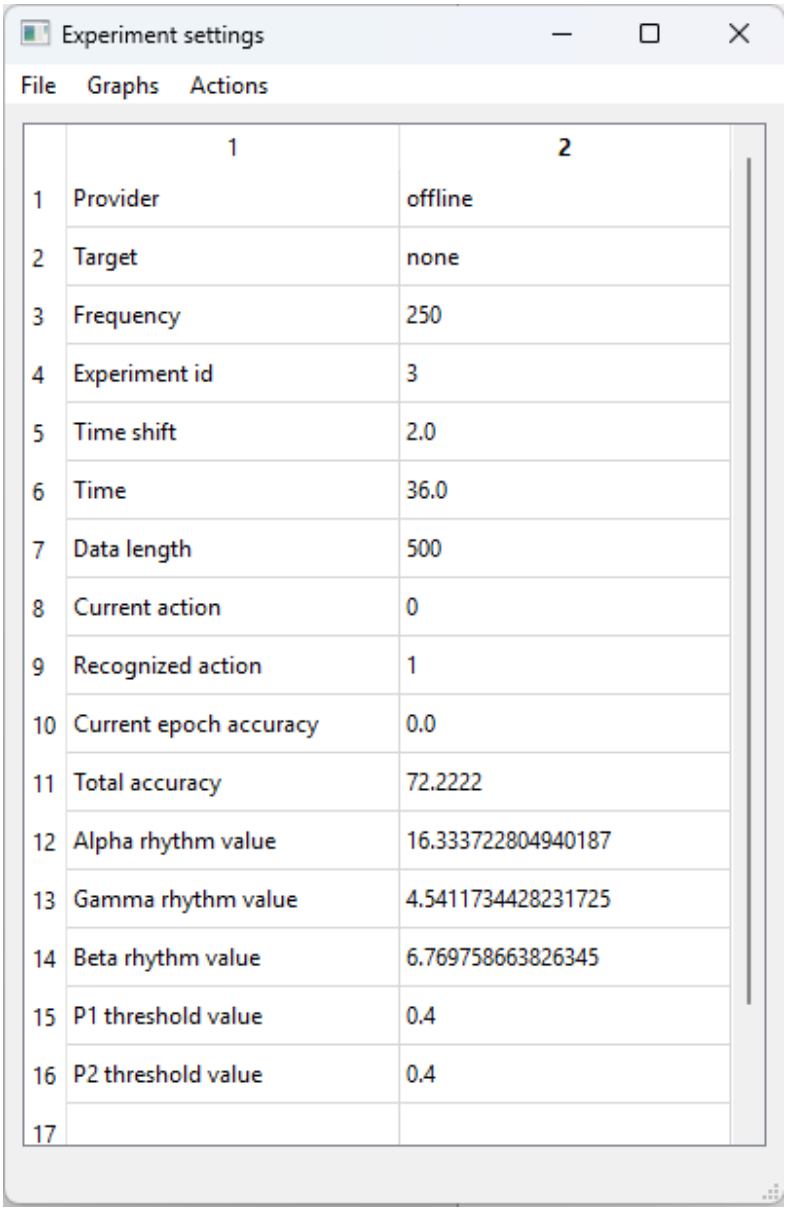
Кроме того, программная платформа предоставляет открытый API, что позволяет использовать её как библиотеку для разработки собственных приложений. Это особенно важно на этапе прототипирования новых устройств, в которых ЭЭГ-сигналы выступают источником управляющих воздействий. Разработчику необходимо лишь подключить модуль настройки окружения и реализовать взаимодействие с источником данных и объектом управления.

3.5 Особенности реализации программного обеспечения для проведения экспериментов

На базе описанной программной платформы также была реализована система управления, использованная при проведении экспериментов и разработан графический интерфейс специально для проведения экспериментов. Это имеет особенную актуальность, так как в задачах, связанных с обучением моделей

машинного обучения, необходимы специализированные подходы к разработке ПО [55, 56].

Графический интерфейс представлен на рисунке 8.



	1	2
1	Provider	offline
2	Target	none
3	Frequency	250
4	Experiment id	3
5	Time shift	2.0
6	Time	36.0
7	Data length	500
8	Current action	0
9	Recognized action	1
10	Current epoch accuracy	0.0
11	Total accuracy	72.2222
12	Alpha rhythm value	16.333722804940187
13	Gamma rhythm value	4.5411734428231725
14	Beta rhythm value	6.769758663826345
15	P1 threshold value	0.4
16	P2 threshold value	0.4
17		

Рисунок 8 – Главное окно графического интерфейса системы управления

Это ПО с множеством модулей, позволяющее эффективно и удобно проводить испытания, связанные с управлением инвалидным креслом при помощи активности мозга. Модульная архитектура позволяет достаточно быстро менять источники и способы обработки сигналов ЭЭГ и изменять их параметры.

Графический интерфейс позволяет контролировать различные параметры эксперимента, например, тип и время проведения эксперимента, параметры сигналов ЭЭГ, текущую точность распознавания паттернов активности мозга. Также интерфейс позволяет изменять некоторые параметры «на лету», прямо во время эксперимента.

Таким образом, в разработанной системе управления в общей сложности 15 модулей. На рисунке 9 изображена схема связей модулей разработанной программы.

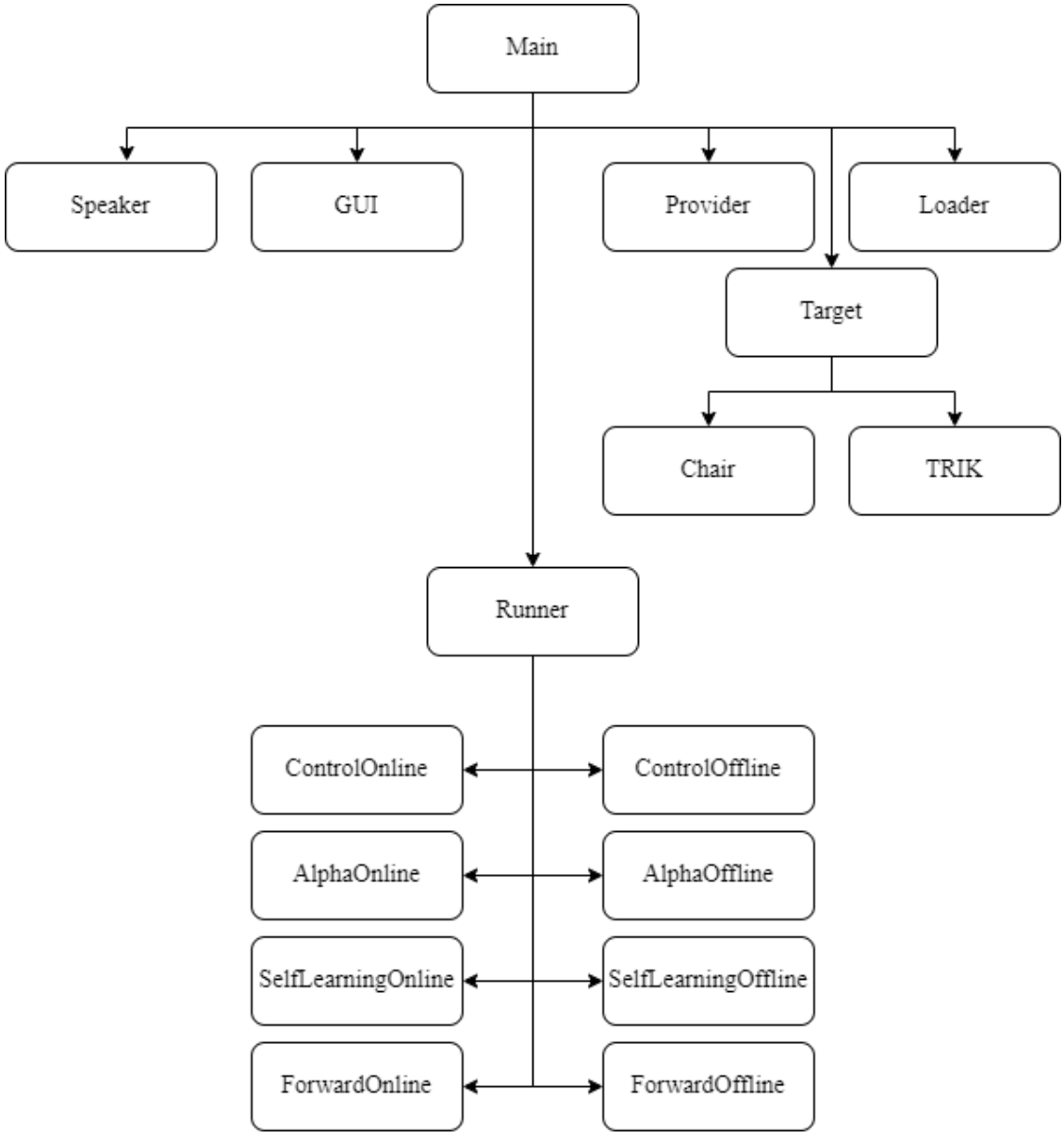


Рисунок 9 – Схема связей модулей системы управления

В таблице 1 приведены сведения об указанных модулях. В приложении А содержится исходный код основных из этих модулей.

Таблица 1 – Сведения о модулях системы управления

№	Название модуля	Назначение модуля
1	Main	Основной модуль программы, обеспечивающий выполнение программы
2	Loader	Модуль, обеспечивающий загрузку наборов данных из файлов в память
3	Provider	Модуль взаимодействия с электроэнцефалографом – установка связи и считывание данных
4	Target	Модуль, обеспечивающий связь с объектом управления. В данном случае поддерживается управление инвалидным креслом и роботом ТРИК
5	Speaker	Модуль вывода звуковых сигналов различной продолжительности и тональности
6	GUI	Модуль графического интерфейса

Продолжение таблицы 1

№	Название модуля	Назначение модуля
7	Runner	Модуль запуска эксперимента
8	AlphaOnline	Модуль эксперимента для задачи остановки инвалидного кресла с помощью ритмов мозга в режиме «онлайн»
9	AlphaOffline	Модуль тестирования классификаторов для задачи остановки инвалидного кресла с помощью ритмов мозга
10	ControlOnline	Модуль эксперимента для задачи поворота инвалидного кресла с помощью воображаемых движений в режиме «онлайн»
11	ControlOffline	Модуль тестирования классификаторов для задачи поворота инвалидного кресла с помощью воображаемых движений
12	SelfLearningOnline	Модуль эксперимента для задачи поворота инвалидного кресла с помощью воображаемых движений в режиме «онлайн» с самообучением

Продолжение таблицы 1

№	Название модуля	Назначение модуля
13	SelfLearningOffline	Модуль тестирования классификаторов для задачи поворота инвалидного кресла с помощью воображаемых движений с самообучением
14	ForwardOnline	Модуль эксперимента для задачи движения инвалидного кресла вперёд с помощью воображаемых движений
15	ForwardOffline	Модуль тестирования классификаторов для задачи движения инвалидного кресла вперёд с помощью воображаемых движений

4 ТЕСТИРОВАНИЕ СИСТЕМЫ УПРАВЛЕНИЯ

4.1 Общие сведения об экспериментах

В экспериментах испытуемый сидит в инвалидном кресле, снабженным устройством дистанционного управления, на него надет головной убор со встроенной электродной сеткой беспроводного электроэнцефалографа. На рисунке 10 показано инвалидное кресло, в котором сидит испытуемый [57-59].

Рисунок 10 – Фотография испытуемого, сидящего в инвалидном кресле во время проведения испытаний

В экспериментах использовалось инвалидное кресло MET COMPACT 15. Кресло оборудовано пультом управления и блоком дистанционного управления, поддерживающим отправку команд по Wi-Fi согласно протоколу управления двигателем.

В экспериментах также использовался беспроводной электроэнцефалограф NeoRecCAP [60], который имеет 21 отведение, т.е. M = 21. На рисунке 11 схематично показано расположение электродов электроэнцефалографа на поверхности головы (стандартная система 10/20).

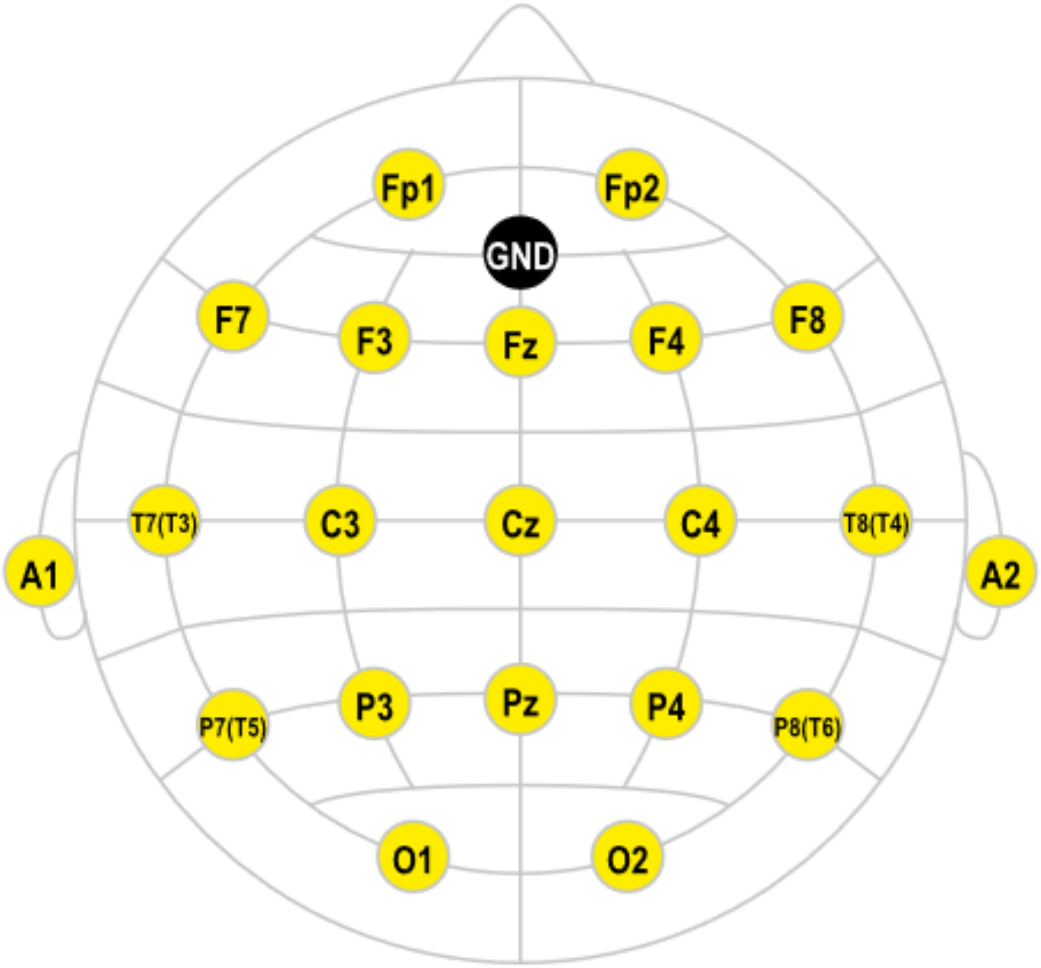


Рисунок 11 – Схематичное изображение положения электродов на поверхности головы испытуемого (система 10/20)

NeoRecCAP может записывать ЭЭГ, метки событий от ИК-триггеров и акселерометра в файлы различных форматов (EDF+ 16 bit, BDF+ 24 bit, GDF 32 bit

и т.д.) или передавать эти данные в LSL-поток (Lab Streaming Layer) для анализа сторонним программным обеспечением. NeoRecCAP может применяться для обучения, научных исследований и разработок в области ЭЭГ и нейромашинных интерфейсов.

Таким образом, схема подключения всех компонентов нейроинтерфейса представлена на рисунке 12.

Рисунок 12 – Схема подключения компонентов нейроинтерфейса, участвующего в экспериментах

Целью проведения экспериментов была оценка эффективности выбранных подходов. И инвалидное кресло, и электроэнцефалограф подключены к устройству управления – ноутбуку. Для экспериментальной проверки работоспособности разработанных моделей и алгоритмов всего было проведено более 40 экспериментов с пятью испытуемыми в общей сложности. От всех испытуемых было получено информированное согласие на участие в экспериментах.

4.2 Сравнение различных алгоритмов классификации ЭЭГ

Для сравнения и оценки точности различных алгоритмов классификации в задачах распознавания ЭЭГ были использованы данные, собранные при работе с испытуемыми. Им давались различные задания, которые они выполняли в течение отведённого времени, в то время как компьютер, подключённый к электроэнцефалографу, записывал данные ЭЭГ. Далее эти данные использовались для сравнения работы различных алгоритмов в режиме оффлайн.

4.2.1 Задача классификации ЭЭГ при воображаемых движениях левой и правой рукой

Для тестирования алгоритмов на данной задаче классификации было выполнено несколько сеансов записи ЭЭГ, в которых всё время записи поделено на несколько интервалов – эпох, длящихся по 6-10 секунд. Каждая эпоха начинается с подачи звукового сигнала, который говорит испытуемому, в какую сторону нужно повернуть кресло (влево или вправо). Задача испытуемого за каждую эпоху – попытаться представить себе движение левой или правой рукой (в зависимости от звукового сигнала).

Разметка набора данных происходит автоматически в зависимости от типа эпохи. Объём данных для тестирования алгоритмов – по 40 эпох на каждый класс (левая или правая рука), 4500 элементов. Размер данных обучения – 80% от общего набора данных, а размер тестовой выборки – 20%.

В таблице 2 приведены алгоритмы машинного обучения, которые были протестированы и оценены на данной задаче. В столбце «параметры» указаны выбранные параметры алгоритма, которые дают лучший результат для этого алгоритма. Алгоритмы отсортированы в порядке ухудшения точности классификации (от лучшего к худшему).

Таблица 2 – Сравнение точности различных алгоритмов в задаче классификации воображаемых движений левой и правой рукой

№	Название алгоритма	Параметры	Точность классификации
1	FANN	$f = 8 \times 10^{-7}$	0.94
2	kNN	k = 3	0.94
3	Random Forest	n = 300	0.87
4	LGBM	n = 100	0.75
5	SVM		0.74
6	LDA		0.60
7	Linear Regression (Ridge Classifier)	$\alpha = 5$	0.49

Из таблицы видно, что лучший результат показал алгоритм FANN, описанный в данной работе. Точность распознавания в задаче классификации воображаемых движений левой и правой рукой составила 94%. На втором месте с тем же результатом оказался метод kNN.

4.2.2 Задача классификации ЭЭГ при воображаемых движениях обеими руками одновременно

Для тестирования алгоритмов на данной задаче классификации было выполнено несколько сеансов записи ЭЭГ, в которых всё время записи поделено на отрезки двух типов. Первый тип соответствует сигналам ЭЭГ, характерным для состояния, когда испытуемый совершает воображаемые движения обеими руками, второй тип соответствует сигналам ЭЭГ при фоновой активности мозга (испытуемый не совершает воображаемых движений). Звуковой сигнал сообщает испытуемому, нужно ли ему совершать воображаемые движения или не делать ничего.

Разметка набора данных происходит автоматически в зависимости от типа записи. Объём данных для тестирования алгоритмов – по 100 кадров на каждый

тип. Размер данных обучения – 80% от общего набора данных, а размер тестовой выборки – 20%.

В таблице 3 приведено сравнение точности различных алгоритмов при решении задачи классификации воображаемых движений обеими руками. Элементы таблицы отсортированы от лучшего к худшему.

Таблица 3 – Сравнение точности различных алгоритмов в задаче классификации воображаемых движений обеими руками

№	Название алгоритма	Параметры	Точность классификации
1	FANN	$f = 2 \times 10^{-6}$	0.93
2	kNN	k = 5	0.93
3	Random Forest	n = 200	0.86
4	LGBM	n = 200	0.66
5	LDA		0.60
6	SVM		0.49
7	Linear Regression (Ridge Classifier)	$\alpha = 10$	0.49

Из таблицы видно, что при решении данной задачи наивысшая точность у FANN и kNN – 93%.

4.2.3 Задача классификации активности ритмов головного мозга

Для тестирования алгоритмов на данной задаче классификации было выполнено несколько сеансов записи ЭЭГ, в которых всё время записи поделено на отрезки двух типов. Первый тип соответствует активности ритмов головного мозга во время попыток испытуемого представить препятствие. Второй тип – активность ритмов в состоянии спокойствия (когда испытуемый просто сидит в инвалидном кресле и не представляет препятствие). Разметка набора данных

производилась вручную. Объём данных – 1900 элементов. Размер данных обучения – 80% от общего набора данных, а размер тестовой выборки – 20%.

В таблице 4 приведены результаты оценки точности различных алгоритмов машинного обучения при решении задачи классификации активности ритмов мозга. Они отсортированы от лучшего к худшему.

Таблица 4 – Сравнение точности различных алгоритмов в задаче классификации воображаемых движений левой и правой рукой

№	Название алгоритма	Параметры	Точность классификации
1	LDA		0.60
2	Linear Regression (Ridge Classifier)	$\alpha = 10$	0.60
3	FANN	$f = 8 \times 10^{-3}$	0.56
4	kNN	k = 11	0.56
5	LGBM	n = 150	0.55
6	SVM		0.54
7	Random Forest	n = 400	0.52

Таблица показывает, что наиболее точными методами оказались LDA и LR. Точность в данной задаче классификации составила 60%.

4.3 Эксперименты с полноценной системой управления

Данные эксперименты направлены на тестирование всей системы управления и должны происходить в режиме онлайн – сбор, предобработка, анализ сигналов, а также отправка управляющих сигналов на инвалидное кресло должны выполняться непосредственно в процессе эксперимента. Это позволяет оценить работу разработанной системы управления в условиях, приближенных к условиям реальной эксплуатации нейроинтерфейса.

Всего разработано три вида экспериментов. Информация о них содержится в таблице 5.

Таблица 5 – Список экспериментов для тестирования системы управления

№	Название эксперимента	Критерии оценки результатов
1	Эксперимент с поворотом инвалидного кресла	Количество успешных поворотов кресла во время контрольного тестирования
2	Эксперимент с движением инвалидного кресла вперёд	Количество успешных движений кресла вперёд
3	Эксперимент с остановкой инвалидного кресла	Количество успешных остановок кресла в заданной точке

Для каждого эксперимента должен быть составлен план и методика проведения испытаний, а также должны быть разработаны критерии оценки результатов испытаний.

Таким образом, в экспериментах участвует весь контур системы целиком – и электроэнцефалограф, и компьютер с программой приёма и анализа сигналов ЭЭГ и управления инвалидным креслом, и непосредственно инвалидное кресло.

4.3.1 Эксперимент с поворотом инвалидного кресла

Всё время эксперимента делится на несколько интервалов – эпох, длящихся по 6-10 секунд. Каждая эпоха начинается с подачи звукового сигнала, который говорит испытуемому, в какую сторону нужно повернуть кресло (влево или вправо). Задача испытуемого за каждую эпоху – попытаться представить себе движение левой или правой рукой (в зависимости от звукового сигнала) таким образом, чтобы инвалидное кресло повернулось влево или вправо соответственно.

Таким образом, эксперимент состоит из нескольких этапов: подача звукового сигнала, который говорит испытуемому, в какую сторону нужно повернуть;

реакция испытуемого; обработка и анализ сигнала на устройстве управления; принятие решения и выдача управляющего сигнала на объект управления (инвалидное кресло). Схематически этапы эксперимента показаны на рисунке 13.

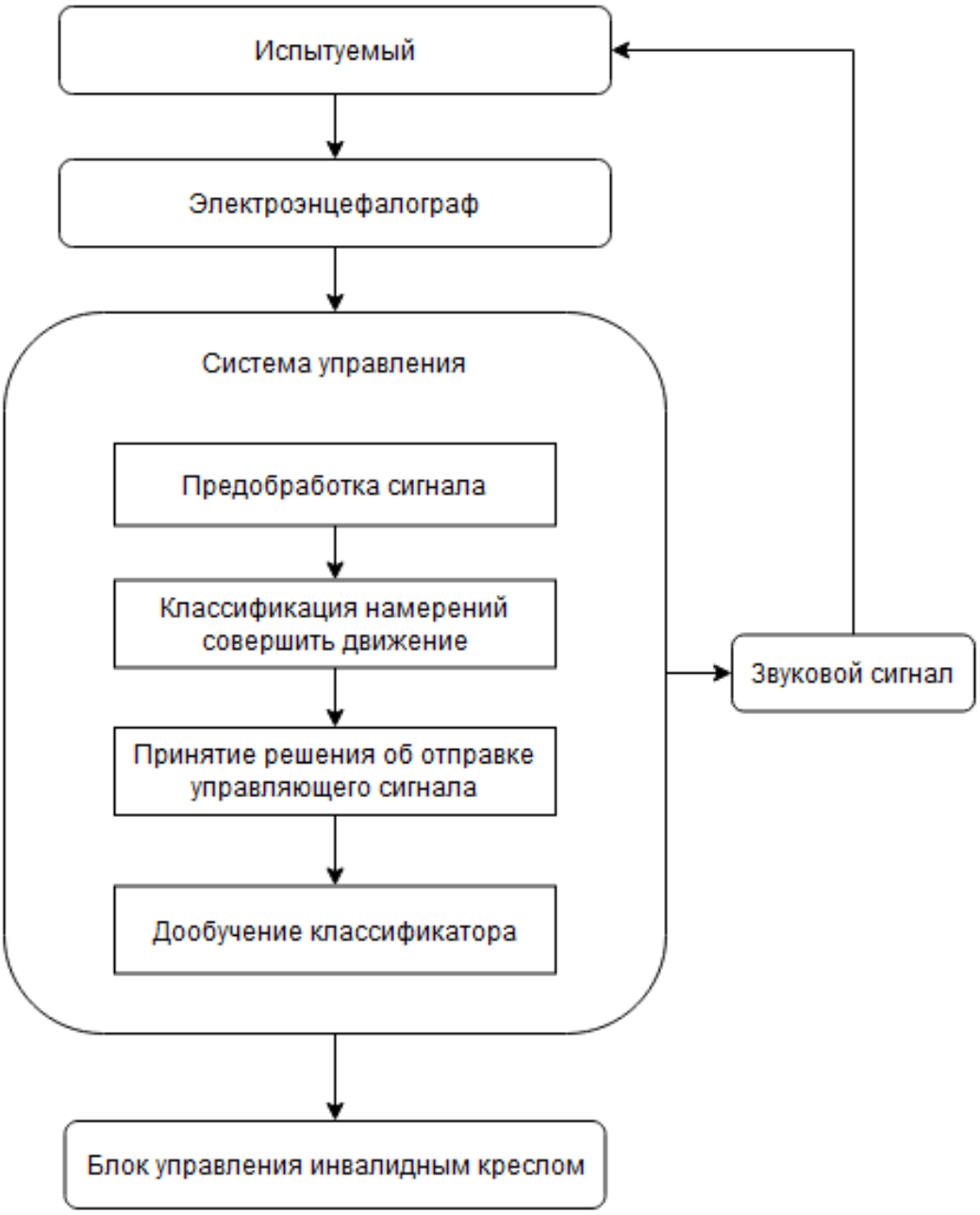


Рисунок 13 – Схематическое изображение взаимодействия составляющих частей системы в экспериментах

На этапе принятия решения о повороте происходит сохранение ЭЭГ сигнала при удачных попытках управления (когда тип поданного звукового сигнала

совпадает с классом распознанного воображаемого движения). Эти сохранённые отрезки сигнала ЭЭГ дополняют собой набор данных для обучения классификатора. Таким образом, возникает эффект адаптации алгоритма под конкретного испытуемого, позволяющий лучше распознавать его намерения.

Дообучение системы управления происходит через определённые промежутки времени, называемые сессиями. Один эксперимент длительностью 20-30 минут может состоять из 3-5 сессий и контрольного тестирования (валидации). Контрольное тестирование выполняется после окончания всех сессий сбора данных и дообучения системы.

Для оценки корректности решения задачи управления поворотом инвалидного кресла также было введено понятие успешности распознавания $P_{ER} = D_{ER} / K_{ER}$, где D_{ER} – количество успешных эпох, K_{ER} – общее количество эпох в сессии. Успешной эпоха считается в том случае, если инвалидное кресло двигалось в рамках этой эпохи в нужном направлении более чем 50% времени.

Таким образом, каждое испытание проводилось по плану, описанному ранее. Одно испытание продолжительностью 27.5 минуты состояло из 4 сессий: первая и вторая сессии длились по 2.5 минуты, остальные две по 10 минут каждая. После этого выполнялось контрольное тестирование, состоящее в том, что испытуемый должен за 2.5 минуты выполнить поворот инвалидного кресла не менее 10 раз.

На рисунке 14 показан график изменения успешности распознавания во время первой сессии, а на рисунке 15 – график изменения успешности распознавания во время контрольного тестирования.

Испытания показали, что разработанный подход позволяет существенно повысить успешность распознавания воображаемых движений левой и правой рукой. За время испытания происходит адаптация как системы управления (классификатора), так и испытуемого.

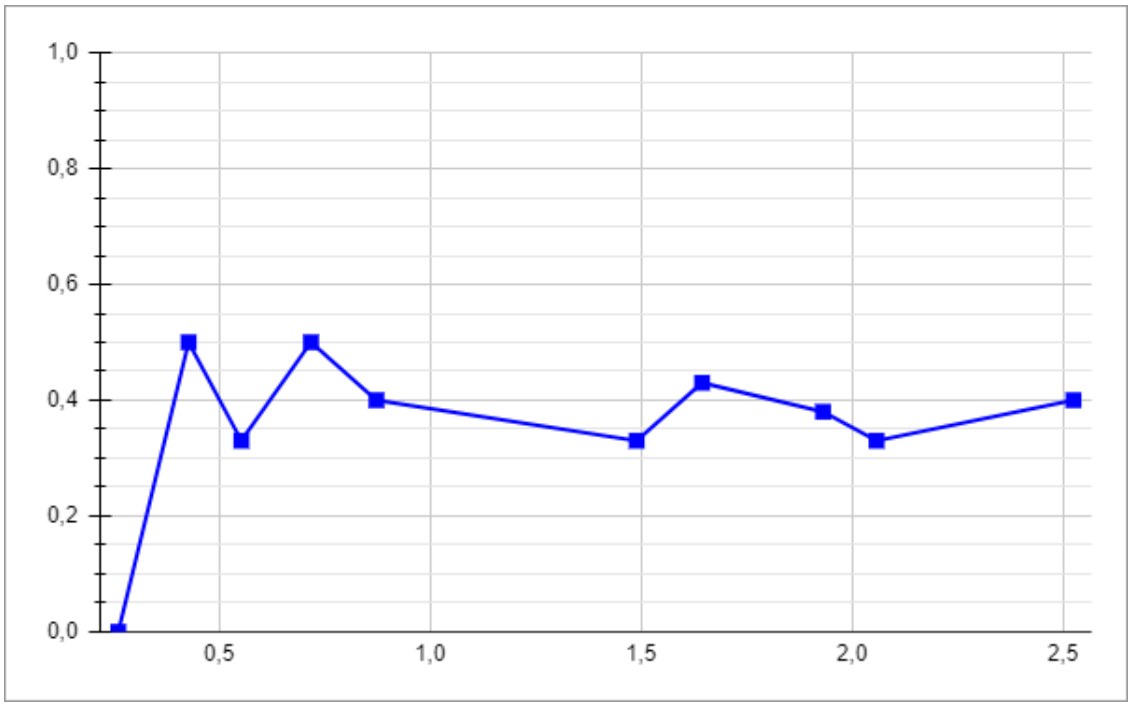


Рисунок 14 – График изменения успешности распознавания (вертикальная ось) во времени (горизонтальная ось, минуты) на первой сессии

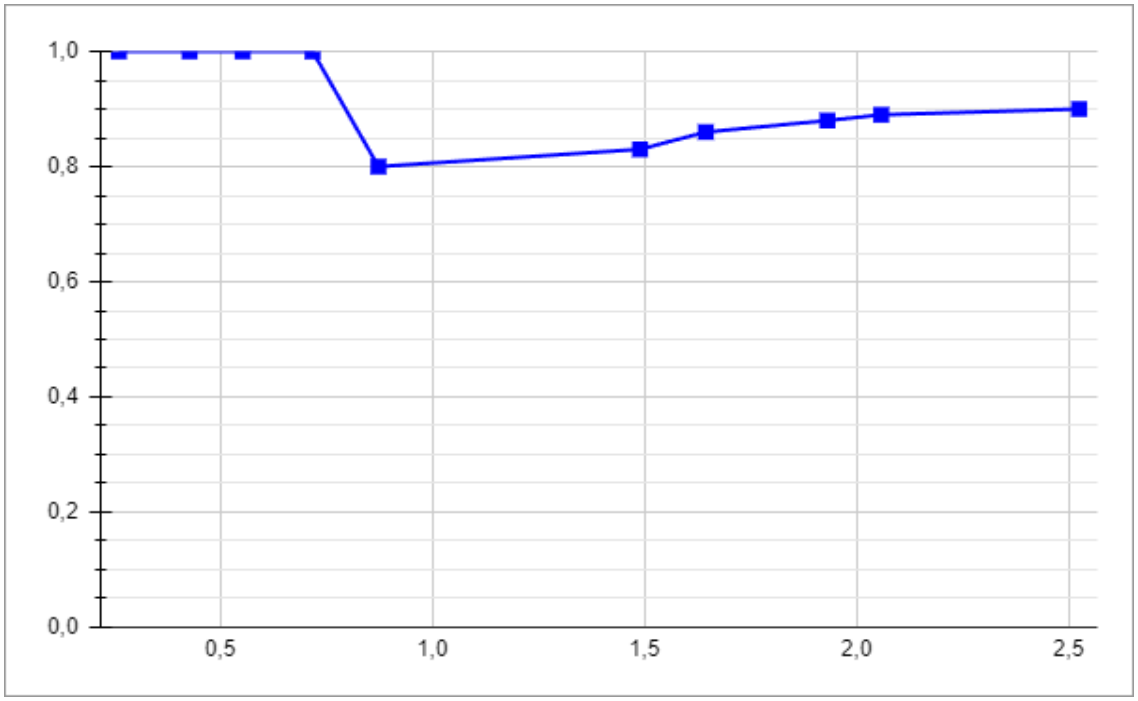


Рисунок 15 – График изменения успешности распознавания (вертикальная ось) во времени (горизонтальная ось, минуты) в процессе контрольного тестирования

Таким образом, сравнивая рисунки 10 и 11 можно отметить, что успешности распознавания в ходе испытания повысилась более, чем вдвое. На первой сессии

успешности распознавания $P_{ER} \approx 0.4$ (40%), а во время контрольного тестирования – $P_{ER} \approx 0.9$ (90%).

4.3.2 Эксперимент с движением инвалидного кресла вперёд

В данном эксперименте задачей испытуемого было заставить инвалидное кресло двигаться вперёд с помощью воображаемых движений обеими руками одновременно. Процесс движения в рамках эксперимента схематично показан на рисунке 16.

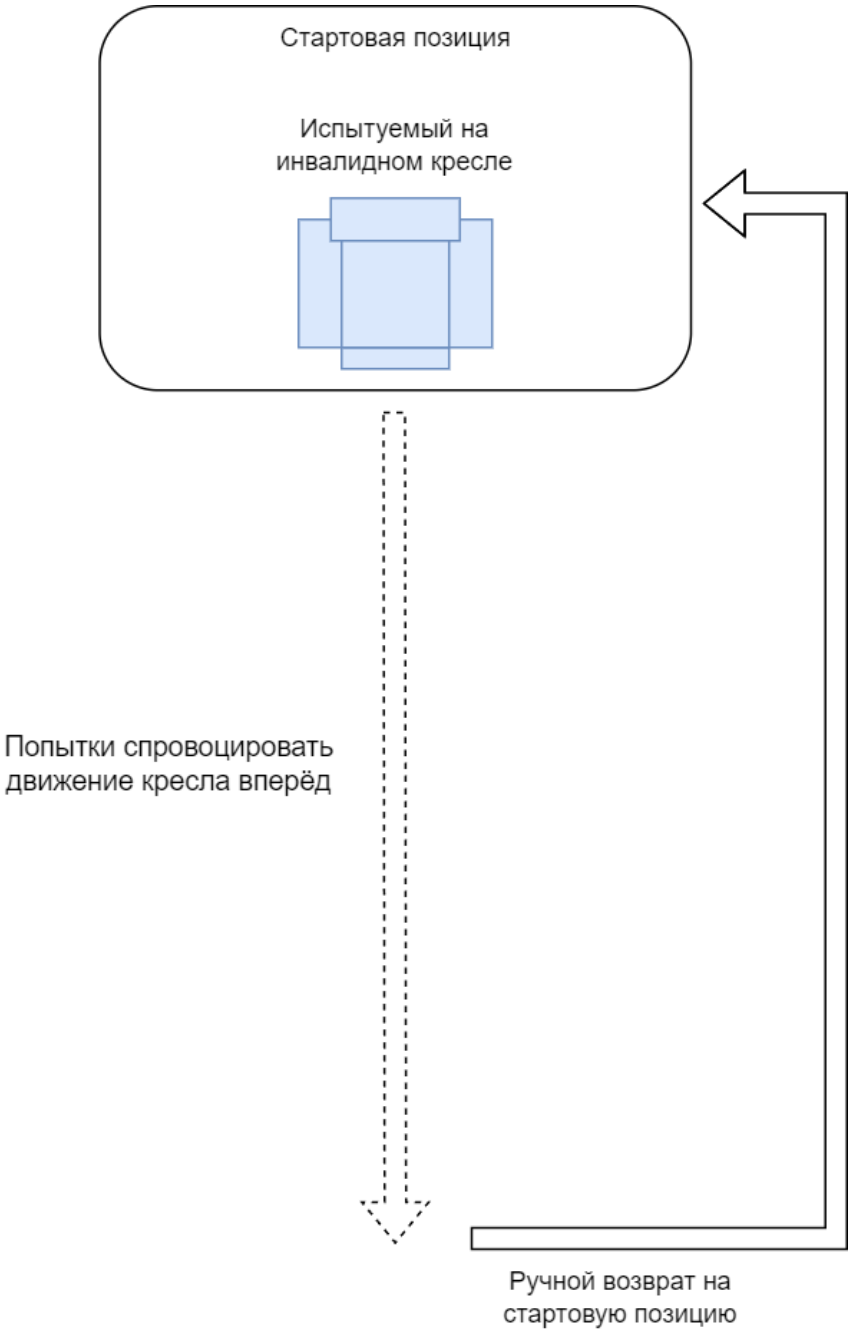


Рисунок 16 – Схема движения испытуемого в рамках эксперимента

Была выбрана определённая стартовая позиция инвалидного кресла. После длинного звукового сигнала (продолжительность 1 с.) задача испытуемого в течение определённого времени спровоцировать движение кресла вперёд. После короткого звукового сигнала (продолжительность 0.5 с.) испытуемый останавливается и самостоятельно возвращается на стартовую позицию с помощью элементов управления, расположенных на инвалидном кресле, после чего дожидается очередного продолжительного звукового сигнала. Диапазон времени между длинным и коротким сигналом равен 20 секундам (эпоха управления), а между коротким и длинным (время отдыха) – 10 секундам.

Весь эксперимент длится 7.5 минут – 15 эпох управления. Для оценки корректности решения задачи управления движением инвалидного кресла вперёд также было введено понятие успешности распознавания $P_{EF} = D_{EF} / K_{EF}$, где D_{EF} – количество успешных эпох, K_{EF} – общее количество эпох в испытании. Успешной эпоха считается в том случае, если инвалидное кресло двигалось вперёд в рамках этой эпохи более чем 50% времени. На рисунке 17 показан график изменения успешности распознавания.

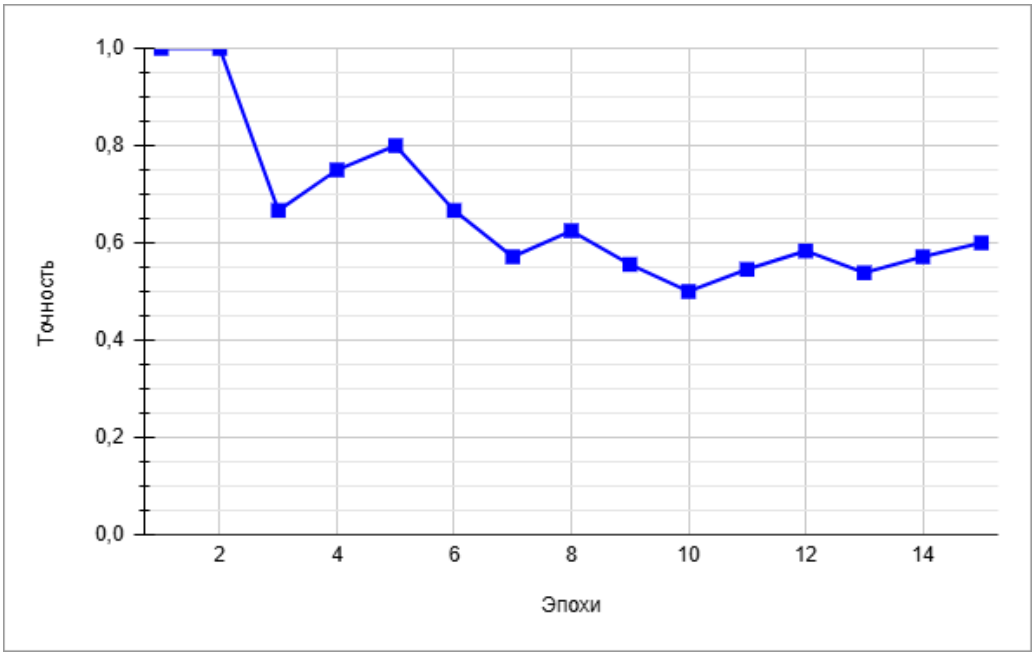


Рисунок 17 – График изменения успешности распознавания (вертикальная ось) в ходе испытания (горизонтальная ось, эпохи)

Таким образом, испытания показали, что разработанный метод обеспечивает неплохую успешность распознавания воображаемых движений левой и правой рукой одновременно. Из графика следует, что средняя успешность распознавания $P_{EF} \approx 0.66$ (66%).

4.3.3 Эксперимент с остановкой инвалидного кресла

В данном эксперименте задачей испытуемого была остановка инвалидного кресла с помощью воображения препятствия перед собой. Процесс движения в рамках эксперимента схематично показан на рисунке 18.

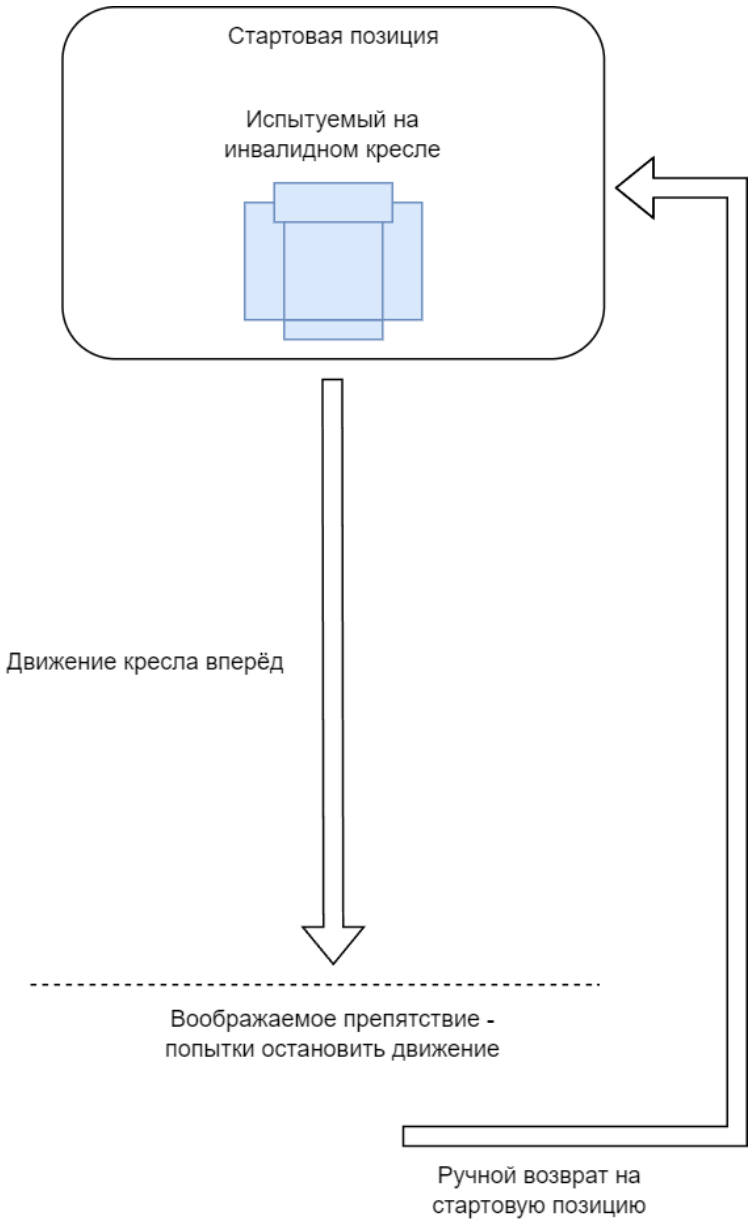


Рисунок 18 – Схема движения испытуемого в рамках эксперимента

По умолчанию кресло двигалось вперёд по прямой, а испытуемый в определённый момент должен был представить препятствие и тем самым спровоцировать остановку кресла. Этот момент был заранее обозначен. Затем испытуемый должен был вручную вернуться на стартовую позицию с помощью элементов управления, расположенных на инвалидном кресле, где процесс начинался заново.

Испытание начинается со стартовой позиции. После длинного звукового сигнала (продолжительность 1 с.) начинается автоматическое движение инвалидного кресла вперёд. Задача испытуемого в течение определённого времени спровоцировать его остановку. После короткого звукового сигнала (продолжительность 0.5 с.) инвалидное кресло останавливается, а испытуемый самостоятельно возвращается на стартовую позицию с помощью элементов управления, расположенных на инвалидном кресле, после чего дожидается очередного продолжительного звукового сигнала. Процесс повторяется. Диапазон времени между длинным и коротким сигналом равен 8 секундам (эпоха управления), а между коротким и длинным (время отдыха) – 10 секундам.

Весь эксперимент длится 6 минут – 20 эпох управления. Для оценки корректности решения задачи остановки инвалидного кресла также было введено понятие успешности распознавания $P_{ES} = D_{ES} / K_{ES}$, где D_{ES} – количество успешных эпох, K_{ES} – общее количество эпох в испытании. Успешной эпоха считается в том случае, если инвалидное кресло было остановлено в рамках этой эпохи более чем 50% времени. На рисунке 19 показан график изменения успешности распознавания.

Таким образом, испытания показали, что разработанный метод обеспечивает хорошую успешность распознавания намерений остановить инвалидное кресло. Из графика следует, что средняя успешность распознавания $P_{EF} \approx 0.8$ (80%).

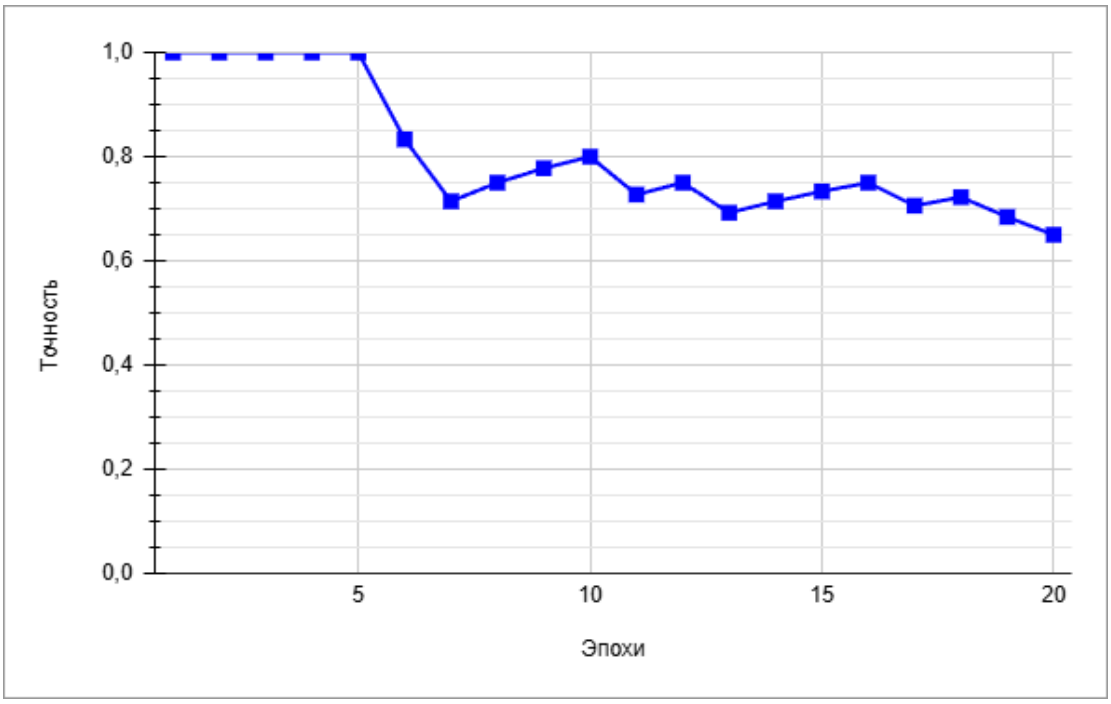


Рисунок 19 – График изменения успешности распознавания (вертикальная ось) в ходе испытания (горизонтальная ось, эпохи)

4.4 Адаптация испытуемого под задачи управления

Проведенные эксперименты показали, что помимо адаптации (обучения) системы управления инвалидным креслом в процессе эксперимента также происходит и адаптация (тренировка) испытуемого. Это означает, что чем дольше испытуемый пытается выполнять задачи управления с помощью активности своего головного мозга, тем чаще у него должно это получаться.

И действительно, было замечено, что в течение одной сессии (т.е. в течение времени, когда параметры обучения неизменны) способность испытуемого совершать воображаемые движения так, чтобы получить правильную реакцию системы управления, растёт. Это выражается в том, что значение P для соответствующего эксперимента возрастает в течение некоторого времени.

Однако следует заметить, что также имеет место быть накопление усталости испытуемого. После некоторого пика концентрации (и точности распознавания) в каждом эксперименте наблюдался её спад, означающий, что испытуемый устаёт выполнять воображаемые движения.

ЗАКЛЮЧЕНИЕ

В работе получены следующие основные результаты.

Во-первых, дано формальное математическое описание процесса сбора и обработки сигналов ЭЭГ. Разработана структура нейроинтерфейса, выбраны его составные элементы.

Во-вторых, выбраны алгоритмы предобработки, анализа и классификации сигналов ЭЭГ, разработан алгоритм принятия решений и управления.

В-третьих, разработана модификация классификатора k-ближайших соседей – метод почти ближайших соседей (FANN), что позволило снизить зависимость от значения числа k и повысить точность распознавания.

В-четвёртых, разработан алгоритм классификации сигналов ЭЭГ с учётом дрейфа параметров, а также дообучения классификатора, что представляет собой метод адаптации системы управления, позволяющий повысить точность распознавания сигналов ЭЭГ.

В-пятых, разработана программная платформа на языке Python, позволяющая реализовывать системы управления для нейроинтерфейсов. На её основе разработана система управления, описанная в данной работе.

В-шестых, разработан план эксперимента для тестирования разработанной системы управления. Проведены эксперименты с реальными испытуемыми, выполнен расчёт успешности работы системы.

Разработанный и описанный в данной работе подход и его программная реализация в ходе испытаний продемонстрировали эффективность в задачах управления движением инвалидного кресла. В процессе контрольного тестирования люди, участвовавшие в экспериментах, демонстрировали способность выполнять поворот инвалидного кресла вправо и влево с точностью ~90%, выполнять движение вперёд – с точностью ~66%, а останавливать движение инвалидного кресла – с точностью ~80%. Предложенная в данной работе модификация классического метода k ближайших соседей, как показано, упрощает подбор параметров алгоритма классификации.

Реализованный программный комплекс позволяет автоматизировать процесс обучения системы управления, что упрощает практическое применение подхода в системах управления движением транспортного средства. Этот процесс не требует привлечения специалистов и может быть выполнен самим испытуемым. При этом ноутбук, на котором реализована система, для удобства испытуемого может быть расположен на самом транспортном средстве.

Также было уделено особое внимание ресурсоёмкости программной реализации. Методы и алгоритмы были реализованы с учётом требований, возникающих при их выполнении на низкопроизводительных устройствах с ограниченным объемом памяти. Программная реализация может быть адаптирована для выполнения на, например, микропроцессорных устройствах Orange Pi 3B [61] без существенной потери скорости обработки сигналов ЭЭГ. Этот микрокомпьютер может иметь до 8 гигабайт оперативной памяти, позволяет выполнять многопоточную обработку данных на 4 процессорных ядрах, при этом имеет умеренное энергопотребление (порядка 6 Вт при максимальной нагрузке), а также небольшие габариты.

Таким образом, предложенный подход к управлению поворотом транспортного средства на основе нейроинтерфейса показал свою перспективность.

Основные результаты диссертации содержатся в следующих публикациях:

1. Babbysh N. Computing brain rhythm indicators of EEG signal // 2021 5th Scientific School Dynamics of Complex Networks and their Applications (DCNA). 2021. Калининград, С. 32-35.

2. Babbysh N. Data classification using interferential neural network model // BF-NAICS 2022, 2022 6th Scientific School Dynamics of Complex Networks and their Applications (DCNA). 2022. Калининград. С. 31-33.

3. Бабич Н.А. Программная платформа для чтения, обработки и анализа данных ЭЭГ // Программная инженерия. 2023. №5. С. 254–260.

4. Babich N. Fradkov A. Brain controlled wheelchair: system description and fuzzy almost nearest neighbors classification // 2024 Sixth International Conference Neurotechnologies and Neurointerfaces (CNN). 2024. Калининград. С. 31-33.

5. Бабич Н.А., Фрадков А.Л. Управление поворотом транспортного средства на основе нейроинтерфейса. Международная научная конференция по механике «Десятые Поляховские чтения», тезисы. 2024. С. 594-597.

6. Бабич Н.А., Фрадков А.Л. Трехпозиционное управление транспортным средством на основе нейроинтерфейса с применением машинного обучения. Информатика и автоматизация. 2025. №1. С. 5-29.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

НИ – нейромашинный интерфейс, нейроинтерфейс;

ИМК – интерфейс «мозг-компьютер»;

BCI – brain-computer interface;

ПО – программное обеспечение;

LSL – поток Lab Streaming Layer;

ЭЭГ – электроэнцефалограмма;

НОС – нейрообратная связь;

ОУ – объект управления;

САУ – система автоматического управления.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лунев Д.В., Полетыкин С.К., Кудрявцев Д.О. Нейроинтерфейсы: обзор технологий и современные решения // Современные инновации, системы и технологии. – 2022. – Т. 2. – № 3. – С. 117–126.

2. Андреев А.М., Гусев А.П. Разработка прецизионного измерителя потенциалов ЭЭГ для комплекса нейроинтерфейса // Инженерный вестник – 2014. – №12.

3. Wilder P. Mystery of the Mind: A Critical Study of Consciousness and the Human Brain, – 1975. – 164 p.

4. Vidal J. Toward Direct Brain-Computer Communication // Annual Review of Biophysics and Bioengineering. – 1973. – pp. 157–180.

5. Левицкая О.С., Лебедев М.А. Интерфейс мозг–компьютер: будущее в настоящем // Вестник РГМУ. – 2016. – №2б. – С. 4–16.

6. Каплан А.Я., Кочетова А.Г., Шишкин С.Л., Басюл И.А., Ганин И.П., Васильев А.Н., Либуркина С.П. Экспериментально-теоретические основания и практические реализации технологии «интерфейс мозг-компьютер» // Бюллетень сибирской медицины. – 2013. – №12(2). – С. 21–29.

7. Лунев Д.В., Полетыкин С.К., Кудрявцев Д.О. Нейроинтерфейсы: обзор технологий и современные решения // Современные инновации, системы и технологии. – 2022. – №2(3). – С. 117–126.

8. Станкевич Л.А., Сонькин К.М., Нагорнова Ж.В., Хоменко Ю.Г., Шемякина Н.В. Классификация электроэнцефалографических паттернов воображаемых движений пальцами руки для разработки интерфейса мозг-компьютер // Труды СПИИРАН. – 2015. – Том 15. – №6. – С. 163–182.

9. Hramov A.E., Maksimenko V.A., Pisarchik A.N. Physical principles of brain-computer interfaces and their applications for rehabilitation, robotics and control of human brain states // Physics Reports. – Vol. 918. – 2021. – pp. 1-133. DOI: 10.1016/j.physrep.2021.03.002.

10. Grubov V., Kiselev A., Badarin A., Schukovsky N. Brain-computer interface for post-stroke rehabilitation // CYBERNETICS AND PHYSICS. – Vol. 8. – Is. 4. – 2019. – pp. 251–256.

11. Maksimenko V. BCI for a brain state control in a dual-task paradigm // CYBERNETICS AND PHYSICS. – Vol. 8. – Is. 4. – 2019. – pp. 262–266.

12. Станкевич Л.А., Гунделах Ф.В. Управление роботом с использованием интерфейса «мозг-компьютер» // Робототехника и техническая кибернетика. – 2017. – № 2. – С. 52–56.

13. Гунделах Ф.В., Станкевич Л.А., Сонькин К.М., Нагорнова Ж.В., Шемякина Н.В. Применение интерфейсов "мозг-компьютер" в ассистивных технологиях // Труды СПИИРАН. 2020. Том 19. №2. С. 277-301.

14. Лобода Ю.О., Функ А.В., Гасымов З.А., Рачкован О.А. Управление мехатронными системами нейроинтерфейсом // XIII Международная научно-практическая конференция, посвященная 55-летию ТУСУРа (г. Томск, 10-12 мая 2017 г.). 2017. С. 143-146.

15. Бодин О.Н., Солодимова Г.А., Спиркин А.Н. Нейроинтерфейс для управления роботизированными устройствами // Измерение. Мониторинг. Управление. Контроль. 2019. № 4 (30). С. 70–76.

16. Миронов В.И., Лобов С.А., Крылова Н.П., Гордлеева С.Ю., Каплан А.Я., Буйлова Т.В., Казанцев В.Б. Разработка нейроуправляемого автомобиля для мобилизации людей с двигательным дефицитом — нейромобиля // Современные технологии в медицине (СТМ). 2018. Том 10, №4. С.49-59.

17. Rashid Mamunur, Sulaiman Norizam, P. P. Abdul Majeed Anwar, Musa Rabiu Muazu, Ab. Nasir Ahmad Fakhri, Bari Bifta Sama, Khatun Sabira. Current Status, Challenges, and Possible Solutions of EEG-Based Brain-Computer Interface: A Comprehensive Review // Frontiers in Neurorobotics. 2020. Vol 14. No. 25.

18. Värbu K., Muhammad N., Muhammad Y. Past, Present, and Future of EEG-Based BCI Applications // Sensors. 2022. vol. 22. no. 9.

19. Yadav H., Maini S. Electroencephalogram based brain-computer interface: Applications, challenges, and opportunities // *Multimedia Tools and Applications*. 2023. Vol. 82.

20. Сазонова Н.Н., Дегтярев С.В., Сазонова Е.С. Аппаратно-программный комплекс на основе нейроинтерфейса и VR-технологии в системе реабилитации пациентов с поражением головного мозга после инсульта // Информационные технологии в управлении, автоматизации и мехатронике: Сборник научных статей 4-й Международной научно-технической конференции (г. Курск, 7 апреля 2022 г.). 2022. С. 180–183.

21. Боброва Е.В., Решетникова В.В., Вершинина Е.А., Гришин А.А., Исаев М.Р., Бобров П.Д., Герасименко Ю.П. Оценка эффективности управления мозг-компьютерным интерфейсом при обучении воображению движений верхних и нижних конечностей // Журнал высшей нервной деятельности. 2023. Том 73. № 1. С. 52–61.

22. Каплан А.Я. Некоторые теоретические и практические основания к реализации нейроинтерфейсных технологий в психиатрии // Психическое здоровье человека и общества. Актуальные междисциплинарные проблемы. Научно-практическая конференция (г. Москва, 30 октября 2017 г.). Москва: МГУ, 2018. С. 366-372.

23. Hongtao Wang, Fan Yan, Tao Xu, Haojun Yin, Peng Chen, Hongwei Yue, Chuangquan Chen, Hongfei Zhang, Linfeng Xu, Yuebang He, Anastasios Bezerianos. Brain-Controlled Wheelchair Review: From Wet Electrode to Dry Electrode, From Single Modal to Hybrid Modal, From Synchronous to Asynchronous. *IEEE Access*. 2021. vol. 9.

24. Fernández-Rodríguez Á., Velasco-Álvarez F.,Ron-Angevin R. Review of real brain-controlled wheelchairs // Neural Eng. 2016. Vol. 13. No. 6.

25. Sha'abani, M.N.A.H., Fuad, N., Jamal, N., Ismail, M.F. (2020). kNN and SVM Classification for EEG: A Review. In: Kasruddin Nasir, A.N., et al. Lecture Notes in Electrical Engineering, vol 632. Springer, Singapore.

26. Palumbo, A., Gramigna, V., Calabrese, B., Ielpo, N. Motor-Imagery EEG-Based BCIs in Wheelchair Movement and Control: A Systematic Literature Review. Sensors 2021, 21, 6285.

27. Maksimenko, V.A., Kurkin, S.A., Pitsik, E.N. et al. Artificial neural network classification of motor-related EEG: An increase in classification accuracy by reducing signal complexity. Complexity. 2018.

28. Яшин А. С., Васильев А. Н., Шишкин С. Л. Чем квазидвижения полезны для изучения произвольных движений? Взгляд со стороны нейронаук, психологии и философии. Гены и Клетки. 2023. Т. 18. № 4. С. 649-652.

29. Shanarova N., Pronina M., Lipkovich M., Ponomarev V., Müller A., Kropotov J. Application of Machine Learning to Diagnostics of Schizophrenia Patients Based on Event-Related Potentials. *Diagnostics.* 2023. vol. 13. no. 3.

30. Lipkovich M., Knyazeva V., Aleksandrov A., Shanarova N., Sagatdinov A., Fradkov A. Evoked Potentials Detection During Self-Initiated Movements Using Machine Learning Approach. *2023 Fifth International Conference on Neurotechnologies and Neurointerfaces (CNN)*. Kaliningrad, RussianFederation.2023, pp. 47-50.

31. Овод И.В., Осадчий А.Е., Пупышев А.А., Фрадков А.Л. Формирование нейрообратной связи на основе адаптивной модели активности головного мозга // *Нейрокомпьютеры*. 2012. №2. С. 36-41.

32. Rybalko A., Fradkov A. Identification of Two-Neuron FitzHugh-Nagumo Model Based on the Speed-Gradient and Filtering. *Chaos*. 2023. vol. 33. no. 8.

33. Обухов С.А., Степанов В.П., Рудаков И.В. Математическая модель нейрокомпьютерного интерфейса на основе анализа вызванных потенциалов P300 // Вестник РГГУ. Серия: Информатика. Информационная безопасность. Математика. 2021. № 2. С. 48-67.

34. Chen Z, Wang Y, Song Z. Classification of Motor Imagery Electroencephalography Signals Based on Image Processing Method. Sensors (Basel). 2021. doi: 10.3390/s21144646

35. Lun Xiangmin, Liu Jianwei, Zhang Yifei, Hao Ziqian, Hou Yimin, A Motor Imagery Signals Classification Method via the Difference of EEG Signals Between Left and Right Hemispheric Electrodes // Frontiers in Neuroscience. 2022. doi: 10.3389/fnins.2022.865594

36. Васильев В.П., Муро Э.Л., Смольский С.М. Основы теории и расчета цифровых фильтров: учебное пособие, 2-е изд., стер. // М. 2024. 272 с.

37. Siti Farah H., Zunainah H., Gauri B. Design of Butterworth Band-Pass Filter//Politeknik & Kolej Komuniti Journal of Engineering and Technology.2016. vol. 1. no. 1.

38. Haykin S. Neural Networks: A Comprehensive Foundation Second Edition. 2019. 1104 p.

39. Mohri M., Rostamizadeh A., Talwalkar A. Foundations of Machine Learning. *The MIT Press*. 2012.

40. Капралов Н.В., Нагорнова Ж.В., Шемякина Н.В. Методы классификации ЭЭГ-паттернов воображаемых движений // Информатика и автоматизация. 2021. Т.2. №1. С. 94-132.

41. Keller J.M., Gray M.R., Givens J.A. A fuzzy K-nearest neighbor algorithm // IEEE Transactions on Systems, Man, and Cybernetics. 1985. vol. 15, no. 4, pp. 580-585.

42. Kumbure M.M., Luukka P. A generalized fuzzy k-nearest neighbor regression model based on Minkowski distance // GranularComputing. 2022. vol. 7. pp. 657–671.

43. Якубович В.А. Рекуррентные конечно-сходящиеся алгоритмы решения систем неравенств // ДАН СССР. 1966. Т. 166.№ 6. С. 1308–1312.

44. Фомин В.Н., Фрадков А.Л., Якубович В.А. Адаптивное управление динамическими объектами. М.: Наука, 1981.

45. Бабич Н. А. Программная платформа для чтения, обработки и анализа данных ЭЭГ // Программная инженерия. 2023. №5. С. 254–260.

46. Luo J., Ying K., Bai J. Savitzky–Golay smoothing and differentiation filter for even number data // Signal Processing. 2005. Vol. 85, Issue 7, P. 1429-1434. DOI: 10.1016/j.sigpro.2005.02.002.

47. Duhamel P., Vetterli M. Fast fourier transforms: A tutorial review and a state of the art // Signal Processing. 1990. Vol. 19, Issue 4. P. 259-299. DOI: 10.1016/0165-1684(90)90158-U.

48. Babbysh N. Computing brain rhythm indicators of EEG signal // 2021 5th Scientific School Dynamics of Complex Networks and their Applications (DCNA). 2021. Калининград, С. 32-35. DOI: 10.1109/DCNA53427.2021.9587284.

49. Jerome H. Greedy function approximation: A gradient boosting machine // The Annals of Statistics, Institute of Mathematical Statistics. 2001. P. 1189-1232. DOI: 10.1214/aos/1013203451.

50. Бабич Н. А. Параметрический синтез интерференционной модели нейронной сети // Вестник современных исследований. 2019. № 1-13 (28). С. 52-56.

51. Бабич Н. А., Останин М. Л. Распознавание объектов на изображениях высокого разрешения с помощью интерференционной нейронной сети // Молодёжь. Техника. Космос: труды XI Общероссийской молодёжной науч.техн. Конф. Т1 / БГТУ “Военмех”. СПб, 2019. С. 229-231.

52. Бабич Н. А. О применении интерференционной нейронной сети для динамического анализа данных в реальном времени // Автоматизация в промышленности. 2020. №4. С. 19-21.

53. Interference C++ library. Открытый репозиторий GitHub. URL: https://github.com/nickware44/interference (дата обращения 12.08.2024).

54. Babbysh N. Data classification using interferential neural network model // BF-NAICS 2022, 2022 6th Scientific School Dynamics of Complex Networks and their Applications (DCNA). 2022. Калининград. С. 31-33.

55. Kabir M.S., Kurkin S., Portnova G., Martynova O., Wang Zh., Hramov A. Contrastive machine learning reveals in EGG resting-state network salient features

specific to autism spectrum disorder. Chaos, Solitons & Fractals. 2024. Vol. 185, pp. 115-123.

56. Антипов В.М., Смирнов Н.М., Бадарин А.А., Киселев А.Р., Андреев А.В., Куркин С.А., Храмов А.Е., Драпкина О.М. Использование интерфейсов мозг-компьютер для персонализированной нейрореабилитации: роль субъективного восприятия и нейрофизиологических показателей // Врач и информационные технологии. 2025. № 2. С. 84-97.

57. Babich N. Fradkov A. Brain controlled wheelchair: system description and fuzzy almost nearest neighbors classification // 2024 Sixth International Conference Neurotechnologies and Neurointerfaces (CNN). 2024. Калининград. С. 31-33.

58. Бабич Н.А., Фрадков А.Л. Управление поворотом транспортного средства на основе нейроинтерфейса. Международная научная конференция по механике «Десятые Поляховские чтения», тезисы. 2024.

59. Бабич Н.А., Фрадков А.Л. Трехпозиционное управление транспортным средством на основе нейроинтерфейса с применением машинного обучения // Информатика и автоматизация. 2025. №1. С. 5-29.

60. Официальная страница электроэнцефалографа NeoRecCAP. – Режим доступа: https://mks.ru/product/neoreccap/ (дата обращения: 10.04.2024).

61. Официальная страница процессорного модуля Orange-Pi-3B. – Режим доступа: http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-3B.html (дата обращения: 03.05.2024).

ПРИЛОЖЕНИЕ А

main.py

```
import sys
from PyQt5 import QtWidgets
from gui.conrol import ControlWindow

def main():
    app = QtWidgets.QApplication(sys.argv)
    control_window = ControlWindow(app)
    sys.exit(app.exec_())



if __name__ == '__main__':
    main()
```

provider.py

```
# -*- coding: utf-8 -*-
import os
from datetime import datetime

import mne
import numpy as np
from pylsl import StreamInlet, resolve_byprop, resolve_bypred
from pylsl.pylsl import lib, StreamInfo, FOREVER, c_int, c_double, byref, handle_error
import utils.edf_saver as edf_saver
import xml.etree.ElementTree as ET
import time
import socket
fmt2string = ['undefined', 'float32', 'float64', 'str', 'int32', 'int16',
              'int8', 'int64']
LSL_STREAM_NAMES = ['AudioCaptureWin', 'NVX136_Data', 'example']
LSL_RESOLVE_TIMEOUT = 10



def save_eeg(data, name, group, channel_labels, fs):
    info = mne.create_info(ch_names=channel_labels,
                           ch_types=['misc'] * len(channel_labels),
                           sfreq=fs)
```

```
    print(len(channel_labels), data.shape)
    raw_buffer_t = np.zeros((len(channel_labels), data.shape[0]))
    for i in range(0, data.shape[1]):
        raw_buffer_t[i, :] = data[:, i]

    # for it2 in range(0, np.shape(chunk)[0]):
    #     data.append([])
    #     for it1 in test_learn_loader.channels:
    #         data[it2].append(chunk[it2][it1])
    raw = mne.io.RawArray(raw_buffer_t, info, verbose=0)
    now = datetime.now()
    dt_string =
str(now.date())+"_"+str(now.time().hour)+"_"+str(now.time().minute)+"_"+str(now.time().second)
    full_path =
os.path.realpath(os.path.dirname(os.path.realpath(__file__)))+"/saved_eeg_new/"+group+"/"+name+"/"+dt_stri
ng+".edf"
    f = open(full_path, "x")
    f.close()
    # print("Saved with name: ", full_path)
    edf_saver.write_mne_edf(raw, full_path, overwrite=True)


class FixedStreamInfo(StreamInfo):
    def as_xml(self):
        return lib.lsl_get_xml(self.obj).decode('utf-8', 'ignore')  # add ignore

class FixedStreamInlet(StreamInlet):
    def info(self, timeout=FOREVER):
        errcode = c_int()
        result = lib.lsl_get_fullinfo(self.obj, c_double(timeout),
                                      byref(errcode))
        handle_error(errcode)
        return FixedStreamInfo(handle=result) # StreamInfo(handle=result)


class Provider:
    def __init__(self, name=LSL_STREAM_NAMES[2], only_this_host=False):
        if not only_this_host:
```

```
            streams = resolve_byprop('name', name, timeout=LSL_RESOLVE_TIMEOUT)
        else:
            streams = resolve_bypred("name='{}' and hostname='{}'".format(name, socket.gethostname()))

        self.inlet = None
        self.dtype = 'float64'
        if len(streams) > 0:
            self.inlet = FixedStreamInlet(streams[0], max_buflen=2)
            # self.dtype = fmt2string[self.inlet.info().channel_format()]
            print('Connected to {} LSL stream successfully'.format(name))
            self.n_channels = self.inlet.info().channel_count()
            # self.channels_labels = self.inlet.info().channel_format()
            self.fs = 250
            print("Channels:", self.n_channels)
        else:
            raise ConnectionError('Cannot connect to "{}" LSL stream'.format(name))

    def get_next_chunk(self, mode=0):
        # get next chunk
        chunk, timestamp = self.inlet.pull_chunk() if mode == 0 else self.inlet.pull_chunk(timeout=3,
max_samples=900000)
        # convert to numpy array
        chunk = np.array(chunk, dtype=self.dtype)
        # return first n_channels channels or None if empty chunk
        return (chunk, timestamp) if chunk.shape[0] > 0 else (None, None)

    def update_action(self):
        pass

    def save_info(self, file):
        with open(file, 'w', encoding="utf-8") as f:
            f.write(self.info_as_xml())

    def info_as_xml(self):
        xml = self.inlet.info().as_xml()
        return xml

    def get_frequency(self):
```

```
        return self.inlet.info().nominal_srate()

    def get_n_channels(self):
        return self.inlet.info().channel_count()

    def get_channels_labels(self):
        for t in range(3):
            time.sleep(0.5*(t+1))
            try:
                rt = ET.fromstring(self.info_as_xml())
                channels_tree = rt.find('desc').findall("channel") or rt.find('desc').find("channels").findall(
                    "channel")
                labels = [(ch.find('label') if ch.find('label') is not None else ch.find('name')).text
                          for ch in channels_tree]
                return labels
            except OSError:
                print('OSError during reading channels names', t+1)
        return [str(n+1) for n in range(self.get_n_channels())]

    def disconnect(self):
        del self.inlet
        self.inlet = None

    def save_data(self, data, name, group):
        save_eeg(data, name, group, self.get_channels_labels(), self.fs)
```

target.py

```
from time import sleep
from targets.trik import Trik
from targets.chair import Chair



class Target:
    def __init__(self, type):
        self.type = type
        if type == "trik":
            self.mailbox = Trik('192.168.77.1', 8889, 2)  # подключение к ТРИКу
```

```
    elif type == "chair_sim":
        self.chair = Chair("127.0.0.1", "40840", "127.0.0.1", "40841")
        self.chair.do_connect()
    elif type == "chair":
        self.chair = Chair("192.200.1.1", "40840", "192.200.1.2", "40841")
        self.chair.do_connect()
        pass
    pass

def do_self_test(self):
    # self.do_action("left", 10)
    # sleep(1)
    # self.do_action("right", 10)
    # sleep(1)

    self.do_action("forward", 1)
    sleep(3)
    self.do_action("stop")

def do_action(self, action, param=0.0):
    if action == "left":
        if self.type == "trik":
            self.mailbox.send_message('R'+str(param))
            self.mailbox.send_message('L-'+str(param))
        elif self.type == "chair":
            self.chair.send_message(-param, 0)
            pass

    elif action == "right":
        if self.type == "trik":
            self.mailbox.send_message('R-'+str(param))
            self.mailbox.send_message('L'+str(param))
        elif self.type == "chair":
            self.chair.send_message(param, 0)
            pass

    elif action == "stop":
        if self.type == "trik":
```

```
            self.mailbox.send_message('S')
        elif self.type == "chair":
            self.chair.send_message(0, 0)
            pass

    elif action == "forward":
        if self.type == "trik":
            pass
        elif self.type == "chair":
            self.chair.send_message(0, param)
            pass

def do_disconnect(self):
    if self.type == "trik":
        pass
    elif self.type == "chair":
        self.chair.do_disconnect()
        pass

def get_type(self):
    return self.type
```

speaker.py

```
import numpy
import pygame


class Speaker:
    def __init__(self):
        pass
    def play_long(self):
        sampleRate = 44100
        freq = 440
        arr = numpy.array([4096 * numpy.sin(2.0 * numpy.pi * freq * x / sampleRate) for x in range(0, sampleRate)]).astype(numpy.int16)
        arr2 = numpy.c_[arr, arr]
        pygame.mixer.init(44100,-16,2,512)
```

```
        sound = pygame.sndarray.make_sound(arr2)
        sound.play(-1)
        pygame.time.delay(1000)
        sound.stop()

    def play_small(self):
        sampleRate = 14100
        freq = 200
        arr = numpy.array([4096 * numpy.sin(2.0 * numpy.pi * freq * x / sampleRate) for x in range(0, sampleRate)]).astype(numpy.int16)
        arr2 = numpy.c_[arr, arr]
        pygame.mixer.init(44100,-16,2,512)
        sound = pygame.sndarray.make_sound(arr2)
        sound.play(-1)
        pygame.time.delay(300)
        sound.stop()
```

targets/trik.py

```
import sys
from threading import Thread

import serial
import serial.tools.list_ports
import socket
import ctypes
# import debugpy

from PyQt5.QtCore import QThread, pyqtSignal, QTimer

from .messages.MessageUtility import msg_from_packet, packet_from_msg, check_crc, isknown_packet
from .messages.JoyMessages import Control, Telemetry
import time

FREQUENCY = 2  # Hz



def bytes_to_hex_str(data: bytes):
```

```
    return data.hex(' ').upper()


class SerialSenderListener(QThread):
    """
    Поток для отправки и чтения данных с серийного порта

    Параметры:
    ----------
    port : str
        Название порта
    baudrate : int
        Скорость передачи
    parity :
        Контроль четности
    stopbits : int
        Стоп-бит
    startbit_listen :
        Стартовый бит прослушиваемого пакета
    num_bits_packet : int
        Количество бит в пакете
    """

    end_signal = pyqtSignal(object)
    data_received = pyqtSignal(bytes)
    warnings = pyqtSignal(str)

    def __init__(self, port: str, baudrate: int, parity: str, stopbits: int, startbit_listen, num_bits_packet: int):
        """Конструктор класса"""
        super(SerialSenderListener, self).__init__()
        self.stopCalled = False
        self.port = port
        self.baudrate = baudrate
        self.parity = parity
        self.stopbits = stopbits
        self.startbit = startbit_listen
        self.num_bits = num_bits_packet
        self.buffer = bytes()
```

```
def stop(self):
    self.stopCalled = True
    if self.ser is not None:
        self.ser.cancel_read()
        self.ser.cancel_write()

def configure(self):
    try:
        self.ser = serial.Serial(port=self.port, baudrate=self.baudrate, parity=self.parity, stopbits=self.stopbits)
        return True, 'OK'
    except Exception as e:
        return False, str(e)

def send_data(self, data):
    try:
        self.ser.write(data)
    except Exception as e:
        self.warnings.emit(str(e))
        return

def run(self):
    try:
        self.listen_port()
        self.end_signal.emit((True, 'Ok'))
    except Exception as e:
        self.end_signal.emit((False, str(e)))

def listen_port(self):
    # debugpy.debug_this_thread()
    while not self.stopCalled:
        data = self.ser.read(1)

        if (data == self.startbit) and len(self.buffer) == 0:
            self.buffer += data
        elif len(self.buffer) == ctypes.sizeof(Telemetry) + 2:
            self.data_received.emit(self.buffer)
            self.buffer = bytes()
```

```
            elif len(self.buffer) != 0:
                self.buffer += data


class UdpSenderListener(QThread):
    """
    Поток для отправки и чтения UDP пакетов

    Параметры:
    ----------
    bu : tuple
        Адресс БУ
    pc : tuple
        Адресс ПК
    timeout : float
        Таймаут на чтение пакета
    """

    end_signal = pyqtSignal(object)
    warnings = pyqtSignal(str)
    data_received = pyqtSignal(bytes)

    def __init__(self, bu: tuple, pc: tuple, timeout):
        """Конструктор класса"""

        super(UdpSenderListener, self).__init__()
        self.bu_address = bu
        self.pc_address = pc
        self.stopCalled = False
        self.timeout = timeout
        self.socket_receive = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket_send = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    def configure(self):
        try:
            self.socket_receive.bind(self.pc_address)
            self.socket_receive.settimeout(self.timeout)
            return (True, 'OK')
```

```
            except Exception as e:
                return (False, str(e))

        def run(self):
            while not self.stopCalled:
                try:
                    data, _ = self.socket_receive.recvfrom(32)
                except Exception as e:
                    if self.stopCalled:
                        self.warnings.emit(f"Не удалось отправить сообщение: {str(e)}")
                    else:
                        self.warnings.emit(f"Таймаут при чтении сообщения: {str(e)}")
                    continue

                if len(data) >= 0:
                    self.data_received.emit(data)

            self.end_signal.emit((True, 'OK'))

        def send_data(self, data):
            try:
                self.socket_send.sendto(data, self.bu_address)
            except Exception as e:
                self.warnings.emit(str(e))

        def stop(self):
            self.stopCalled = True
            self.socket_send.close()
            self.socket_receive.close()


    class Chair():
        def __init__(self, address_bu, port_bu, address_pc, port_pc):
            # super().__init__()
            self.pos_x = 0
            self.pos_y = 0
            self.address_bu = address_bu
            self.port_bu = port_bu
```

```
    self.address_pc = address_pc
    self.port_pc = port_pc

    self._idx: int = 0
    self.__run = True
    self.th_sender = None
    self.__send_thread = Thread(target=self.__timerToSend)
    self.__send_thread.start()
    # self.timer_sender = QTimer()
    # self.timer_sender.timeout.connect(self.__timerToSend)
    # self.timer_sender.start(1000)

@property
def idx(self):
    if self._idx >= 255:
        self._idx = 0
    else:
        self._idx += 1
    return self._idx

def __onReceivedData(self, data):
    try:
        msg = msg_from_packet(data)
    except Exception as e:
        return

    # self.update_packet_ui.emit(msg)

def __onSignalStop(self, result):
    self.th_sender = None

def send_message(self, pos_x, pos_y):
    print("command "+str(pos_x)+" "+str(pos_y))
    self.pos_x = pos_x
    self.pos_y = pos_y

def do_disconnect(self):
    if self.th_sender is None:
```

```
            return

        self.__run = False
        self.th_sender.stop()

    def do_connect(self):
        if self.th_sender is not None:
            return
        else:
            self.th_sender = UdpSenderListener(bu=(self.address_bu, int(self.port_bu)),
                                               pc=(self.address_pc, int(self.port_pc)),
                                               timeout=3)

        result, str_result = self.th_sender.configure()
        if not result:
            self.th_sender = None
            return
        self.th_sender.start()
        print("Connected")

    def __timerToSend(self):
        while self.__run:
            if self.th_sender:
                msg = packet_from_msg(Control(idx=self.idx,
                                              joy_x=int(self.pos_x),
                                              joy_y=int(self.pos_y)))
                self.th_sender.send_data(msg)
                # print(msg)
            time.sleep(int(1000/FREQUENCY)/1000)
```

targets/chair.py

```
import sys
from threading import Thread

import serial
import serial.tools.list_ports
import socket
```

```
import ctypes
# import debugpy

from PyQt5.QtCore import QThread, pyqtSignal, QTimer

from .messages.MessageUtility import msg_from_packet, packet_from_msg, check_crc, isknown_packet
from .messages.JoyMessages import Control, Telemetry
import time

FREQUENCY = 2  # Hz


def bytes_to_hex_str(data: bytes):
    return data.hex(' ').upper()


class SerialSenderListener(QThread):
    """
    Поток для отправки и чтения данных с серийного порта

    Параметры:
    ----------
    port : str
        Название порта
    baudrate : int
        Скорость передачи
    parity :
        Контроль четности
    stopbits : int
        Стоп-бит
    startbit_listen :
        Стартовый бит прослушиваемого пакета
    num_bits_packet : int
        Количество бит в пакете
    """

    end_signal = pyqtSignal(object)
    data_received = pyqtSignal(bytes)
```

```
warnings = pyqtSignal(str)

def __init__(self, port: str, baudrate: int, parity: str, stopbits: int, startbit_listen, num_bits_packet: int):
    """Конструктор класса"""
    super(SerialSenderListener, self).__init__()
    self.stopCalled = False
    self.port = port
    self.baudrate = baudrate
    self.parity = parity
    self.stopbits = stopbits
    self.startbit = startbit_listen
    self.num_bits = num_bits_packet
    self.buffer = bytes()

def stop(self):
    self.stopCalled = True
    if self.ser is not None:
        self.ser.cancel_read()
        self.ser.cancel_write()

def configure(self):
    try:
        self.ser = serial.Serial(port=self.port, baudrate=self.baudrate, parity=self.parity, stopbits=self.stopbits)
        return True, 'OK'
    except Exception as e:
        return False, str(e)

def send_data(self, data):
    try:
        self.ser.write(data)
    except Exception as e:
        self.warnings.emit(str(e))
        return

def run(self):
    try:
        self.listen_port()
        self.end_signal.emit((True, 'Ok'))
```

```
            except Exception as e:
                self.end_signal.emit((False, str(e)))

        def listen_port(self):
            # debugpy.debug_this_thread()
            while not self.stopCalled:
                data = self.ser.read(1)

                if (data == self.startbit) and len(self.buffer) == 0:
                    self.buffer += data
                elif len(self.buffer) == ctypes.sizeof(Telemetry) + 2:
                    self.data_received.emit(self.buffer)
                    self.buffer = bytes()
                elif len(self.buffer) != 0:
                    self.buffer += data


    class UdpSenderListener(QThread):
        """
        Поток для отправки и чтения UDP пакетов

        Параметры:
        ----------
        bu : tuple
            Адресс БУ
        pc : tuple
            Адресс ПК
        timeout : float
            Таймаут на чтение пакета
        """

        end_signal = pyqtSignal(object)
        warnings = pyqtSignal(str)
        data_received = pyqtSignal(bytes)

        def __init__(self, bu: tuple, pc: tuple, timeout):
            """Конструктор класса"""
```

```
    super(UdpSenderListener, self).__init__()
    self.bu_address = bu
    self.pc_address = pc
    self.stopCalled = False
    self.timeout = timeout
    self.socket_receive = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.socket_send = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

def configure(self):
    try:
        self.socket_receive.bind(self.pc_address)
        self.socket_receive.settimeout(self.timeout)
        return (True, 'OK')
    except Exception as e:
        return (False, str(e))

def run(self):
    while not self.stopCalled:
        try:
            data, _ = self.socket_receive.recvfrom(32)
        except Exception as e:
            if self.stopCalled:
                self.warnings.emit(f"Не удалось отправить сообщение: {str(e)}")
            else:
                self.warnings.emit(f"Таймаут при чтении сообщения: {str(e)}")
            continue

        if len(data) >= 0:
            self.data_received.emit(data)

    self.end_signal.emit((True, 'OK'))

def send_data(self, data):
    try:
        self.socket_send.sendto(data, self.bu_address)
    except Exception as e:
        self.warnings.emit(str(e))
```

```
    def stop(self):
        self.stopCalled = True
        self.socket_send.close()
        self.socket_receive.close()


class Chair():
    def __init__(self, address_bu, port_bu, address_pc, port_pc):
        # super().__init__()
        self.pos_x = 0
        self.pos_y = 0
        self.address_bu = address_bu
        self.port_bu = port_bu
        self.address_pc = address_pc
        self.port_pc = port_pc

        self._idx: int = 0
        self.__run = True
        self.th_sender = None
        self.__send_thread = Thread(target=self.__timerToSend)
        self.__send_thread.start()
        # self.timer_sender = QTimer()
        # self.timer_sender.timeout.connect(self.__timerToSend)
        # self.timer_sender.start(1000)

    @property
    def idx(self):
        if self._idx >= 255:
            self._idx = 0
        else:
            self._idx += 1
        return self._idx

    def __onReceivedData(self, data):
        try:
            msg = msg_from_packet(data)
        except Exception as e:
            return
```

```
    # self.update_packet_ui.emit(msg)

def __onSignalStop(self, result):
    self.th_sender = None

def send_message(self, pos_x, pos_y):
    print("command "+str(pos_x)+" "+str(pos_y))
    self.pos_x = pos_x
    self.pos_y = pos_y

def do_disconnect(self):
    if self.th_sender is None:
        return

    self.__run = False
    self.th_sender.stop()

def do_connect(self):
    if self.th_sender is not None:
        return
    else:
        self.th_sender = UdpSenderListener(bu=(self.address_bu, int(self.port_bu)),
                                           pc=(self.address_pc, int(self.port_pc)),
                                           timeout=3)

    result, str_result = self.th_sender.configure()
    if not result:
        self.th_sender = None
        return
    self.th_sender.start()
    print("Connected")

def __timerToSend(self):
    while self.__run:
        if self.th_sender:
            msg = packet_from_msg(Control(idx=self.idx,
                                          joy_x=int(self.pos_x),
```

```
                                joy_y=int(self.pos_y)))
            self.th_sender.send_data(msg)
            # print(msg)
        time.sleep(int(1000/FREQUENCY)/1000)
```

loader.py

```
import glob
import os
from pathlib import Path

import h5py as h5py
import mne as mne
import numpy
import numpy as np



def map_channel_name(old_name):
    """Clean a channel name."""
    new_name = old_name.replace('EEG ', '').replace('-Ref', '')
    return new_name

def sortName(val):
    return int(Path(val).stem)

class Loader:
    def __init__(self, params=None, method="default"):
        if params is None:
            params = {}
        self.params = params
        self.dataset = []
        self.dataset.append([])
        self.dataset.append([])
        self.exclude = []
        self.last_dataset_id = 0
        self.freq = 0
        self.channels = range(0, 21)
        self.info = None
```

```
    self.method = method
    pass

def get_channels_and_fs(self, filepath):
    # save channels names and sampling frequency
    with h5py.File(filepath, 'r') as f:
        return [s.decode('utf-8') for s in f['channels'][:]], int(f['fs'].value)

def dataset_reset(self):
    self.dataset = []
    self.dataset.append([])
    self.dataset.append([])

def load_dataset(self, filepath, dataset_id=None, count_last=-1, sort=False):
    print(filepath)
    if dataset_id is None:
        self.last_dataset_id += 1
    else:
        self.last_dataset_id = dataset_id

    _, _, files = next(os.walk(filepath))
    if sort:
        files.sort(key=sortName)
    file_count = len(files)
    # dir_list = glob.iglob(filepath+'**/*.edf', recursive=True)
    for i in range(0, file_count):
        if count_last == -1 or i >= file_count-count_last:
            # print(i, filepath+"/"+files[i])
            self.load(filepath+"/"+files[i], self.last_dataset_id)

def load(self, filepath, dataset_id=0, markers=None, raw_flag=False):
    source_buffer = []
    # print(filepath)
    self.last_dataset_id = dataset_id
    file_name, file_extension = os.path.splitext(filepath)

    if file_extension == '.fif':
        raw = mne.io.read_raw_fif(filepath, verbose='ERROR', preload=True)
```

```
    source_buffer = raw.get_data()
elif file_extension == '.edf':
    raw = mne.io.read_raw_edf(filepath, verbose='ERROR', preload=True)
    # start, stop = raw.time_as_index([0, 600])  # read the first 15s of data

    # raw.filter(2.0, 30.0, fir_design="firwin", skip_by_annotation="edge", verbose=False)

    source_buffer = raw.get_data()

    montage = mne.channels.make_standard_montage('standard_1020')

    #   FIXME: info
    # r = raw.pick_channels(raw.info["ch_names"][:self.channels.stop])
    # r.rename_channels(mapping=map_channel_name)
    # r.set_montage(montage)
    # self.info = r.info

    # self.info = raw.pick_channels(raw.info["ch_names"][:30]).info

    # print(raw.info)
try:
    if file_extension in ('.fif', '.vhdr', '.edf'):
        labels = raw.info['ch_names']
        self.freq = raw.info['sfreq']
        # print(raw.info)
        # print(labels)
    else:
        labels, self.freq = self.get_channels_and_fs(filepath)

    # exclude = [ex.upper() for ex in ChannelsSelector.parse_channels_string(reference)]
    # labels = [label for label in labels if label.upper() not in exclude]
    # print('Using {} channels and fs={}.\n[{}]'.format(len(labels), fs, labels))
except FileNotFoundError:
    print('Channels labels and fs not found. Using default 32 channels and fs=500Hz.')
    labels = None
    fs = None

# print("buffer size: ", source_buffer.shape[1])
```

```
# print(source_buffer.shape)
# print(source_buffer.shape[1]/self.freq)

# print("Using channels:", self.channels)

max_a = source_buffer[self.channels[0]][0]
min_a = source_buffer[self.channels[0]][0]

if raw_flag:
    self.dataset[0] = source_buffer.copy()
    self.dataset[1].append(dataset_id)
    return

if markers is None:
    event = []

    # range1 = range(0, source_buffer.shape[1])
    range1 = range(0, 250)
    range2 = self.channels
    if self.method == "event":
        range1 = self.channels
        # range2 = range(0, source_buffer.shape[1])
        range2 = range(0, 250)

    for i in range1:
        channels = []
        for c in range2:
            if self.method == "event":
                channels.append(source_buffer[i][c])
            else:
                channels.append(source_buffer[c][i])

        if max_a < channels[0]:
            max_a = channels[0]
        if min_a > channels[0]:
            min_a = channels[0]
        # if source_buffer[33][i] != 0.0:
        #     print(source_buffer[33][i])
```

```
            if self.method == "event":
                event.append(channels)
            else:  # default
                self.dataset[0].append(channels)
                self.dataset[1].append(dataset_id)
        pass
        if self.method == "event":
            self.dataset[0].append(event)
            self.dataset[1].append(dataset_id)
        # print("min", min_a, "max", max_a)
    else:
        for m in markers:
            event = []

            range1 = range(self.time_to_index(m[0]), self.time_to_index(m[1]))
            range2 = self.channels
            if self.method == "event":
                range1 = self.channels
                range2 = range(self.time_to_index(m[0]), self.time_to_index(m[1]))

            for i in range1:
                channels = []
                for c in range2:
                    if self.method == "event":
                        channels.append(source_buffer[i][c])
                    else:
                        channels.append(source_buffer[c][i])

                if max_a < channels[0]:
                    max_a = channels[0]
                if min_a > channels[0]:
                    min_a = channels[0]

                if self.method == "event":
                    event.append(channels)
                else:  # default
                    self.dataset[0].append(channels)
                    self.dataset[1].append(dataset_id)
```

```
            pass
            if self.method == "event":
                self.dataset[0].append(event)
                self.dataset[1].append(dataset_id)
        # print("min", min_a, "max", max_a)
        pass

def get_chunk(self, t_from=0, t_to=-1, scale=1):
    if t_to == -1:
        t_to = len(self.dataset[0][0])

    chunk = []
    max_a = self.dataset[0][0][0]
    min_a = self.dataset[0][0][0]
    for i in range(self.time_to_index(t_from), self.time_to_index(t_to)):
        if scale != 1:
            for n in range(0, len(self.dataset[0][i])):
                self.dataset[0][i][n] = self.dataset[0][i][n]*scale
        chunk.append(self.dataset[0][i])
        if max_a < self.dataset[0][i][0]:
            max_a = self.dataset[0][i][0]
        if min_a > self.dataset[0][i][0]:
            min_a = self.dataset[0][i][0]
        # print(self.dataset[0][i])
        # print("min", min_a, "max", max_a)
    return chunk

def get_chunk_raw(self, t_from=0, t_to=-1):
    chunk = []
    for n in self.channels:
        channel = []
        for i in range(self.time_to_index(t_from), self.time_to_index(t_to)):
            channel.append(self.dataset[0][n][i])
        chunk.append(channel)
    return chunk

def get_total_time(self):
    return int(len(self.dataset[0])/self.freq)
```

```
    def time_to_index(self, time):
        return int(self.freq*time)

    def load_csv(self):
        data = []
        data.append([])
        data.append([])
        counter_0 = 0
        counter_1 = 0
        with open("rhythm_data.csv", "r") as file:
            for line in file:
                if line != "":
                    splited = line.split(";")
                    if int(splited[3]) == 1:
                        counter_1 += 1



        with open("rhythm_data.csv", "r") as file:
            for line in file:
                if line != "":
                    splited = line.split(";")
                    # print([float(splited[0]), float(splited[1]), float(splited[2])])
                    if int(splited[3]) == 0:
                        if counter_0 < counter_1:
                            counter_0 += 1
                            data[0].append([float(splited[0]), float(splited[1]), float(splited[2])])
                            data[1].append(int(splited[3]))
                    else:
                        data[0].append([float(splited[0]), float(splited[1]), float(splited[2])])
                        data[1].append(int(splited[3]))

        print(counter_0, counter_1)
        return data
```

gui/control.py

```
import time
```

```
import numpy as np
from PyQt5 import QtWidgets, QtGui
from PyQt5.QtCore import QThread
from PyQt5.QtWidgets import QTableWidget, QTableWidgetItem, QHeaderView, QVBoxLayout, QHBoxLayout, QWidget

from runner import RunnerObject
from gui.spectral import SpectralWindow
from gui.stat import StatWindow


class ControlWindow(QtWidgets.QMainWindow):
    def __init__(self, app):
        super(ControlWindow, self).__init__()
        # self.setWindowIcon(QtGui.QIcon(STATIC_PATH + '/imag/settings.png'))
        self.freq = 0
        self.layout = None

        self.spectral = SpectralWindow("OZ")
        self.stats = StatWindow()
        self.app = app



        self.runner_thread = QThread()
        self.runner_object = RunnerObject()
        self.runner_object.moveToThread(self.runner_thread)
        self.runner_object.set_static_parameters.connect(self.set_static_parameters)
        self.runner_object.set_dynamic_parameters.connect(self.set_dynamic_parameters)
        self.runner_object.set_graph_data.connect(self.set_graph_data)
        self.runner_thread.started.connect(self.runner_object.start)
        # objThread.finished.connect(app.exit)

        self.init_ui()
        self.runner_thread.start()

    def init_ui(self):
        central_widget = QWidget(self)              # Create a central widget
```

```
self.setCentralWidget(central_widget)       # Install the central widget

# actions
exit_action = QtWidgets.QAction('&Exit', self)
exit_action.setShortcut('Ctrl+Q')
exit_action.setStatusTip('Exit application')
exit_action.triggered.connect(QtWidgets.QApplication.quit)

open_spectral = QtWidgets.QAction(QtGui.QIcon(''), 'Spectral', self)
open_spectral.setShortcut('Ctrl+S')
open_spectral.setStatusTip('Open spectral window')
open_spectral.triggered.connect(self.spectral.show)

open_stats = QtWidgets.QAction(QtGui.QIcon(''), 'Statistics', self)
open_stats.setShortcut('Ctrl+t')
open_stats.setStatusTip('Open statistics window')
open_stats.triggered.connect(self.stats.show)

start_action = QtWidgets.QAction(QtGui.QIcon(''), 'Start action event', self)
start_action.setShortcut('F3')
start_action.setStatusTip('Send action event to experiment')
start_action.triggered.connect(self.start_experiment_action)

stop_action = QtWidgets.QAction(QtGui.QIcon(''), 'Stop action event', self)
stop_action.setShortcut('F4')
stop_action.setStatusTip('Stop action event')
stop_action.triggered.connect(self.stop_experiment_action)

# status bar init
self.statusBar()
menubar = self.menuBar()

file_menu = menubar.addMenu('&File')
file_menu.addAction(exit_action)

graphs_menu = menubar.addMenu('&Graphs')
graphs_menu.addAction(open_spectral)
graphs_menu.addAction(open_stats)
```

```
actions_menu = menubar.addMenu('&Actions')
actions_menu.addAction(start_action)
actions_menu.addAction(stop_action)

# parameter tree
# self.widget = SettingsWidget(self.app)
# self.setCentralWidget(self.widget)

self.table_widget = QTableWidget(self)

# Row count
self.table_widget.setRowCount(18)

# Column count
self.table_widget.setColumnCount(2)

self.table_widget.setItem(0, 0, QTableWidgetItem("Provider"))
self.table_widget.setItem(1, 0, QTableWidgetItem("Target"))
self.table_widget.setItem(2, 0, QTableWidgetItem("Frequency"))
self.table_widget.setItem(3, 0, QTableWidgetItem("Experiment id"))
self.table_widget.setItem(4, 0, QTableWidgetItem("Time shift"))
self.table_widget.setItem(5, 0, QTableWidgetItem("Time"))
self.table_widget.setItem(6, 0, QTableWidgetItem("Data length"))
self.table_widget.setItem(7, 0, QTableWidgetItem("Current action"))
self.table_widget.setItem(8, 0, QTableWidgetItem("Recognized action"))
self.table_widget.setItem(9, 0, QTableWidgetItem("Current epoch accuracy"))
self.table_widget.setItem(10, 0, QTableWidgetItem("Total accuracy"))
self.table_widget.setItem(11, 0, QTableWidgetItem("Alpha rhythm value"))
self.table_widget.setItem(12, 0, QTableWidgetItem("Gamma rhythm value"))
self.table_widget.setItem(13, 0, QTableWidgetItem("Beta rhythm value"))
self.table_widget.setItem(14, 0, QTableWidgetItem("P1 threshold value"))
self.table_widget.setItem(15, 0, QTableWidgetItem("P2 threshold value"))

self.table_widget.itemChanged.connect(self.set_values_from_table)
self.table_widget.horizontalHeader().setStretchLastSection(True)
self.table_widget.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
```

```
        # window settings
        self.setGeometry(200, 200, 400, 600)
        self.setWindowTitle('Experiment settings')

        self.layout = QVBoxLayout()
        self.layout.addWidget(self.table_widget)
        central_widget.setLayout(self.layout)

        self.show()

    def start_experiment_action(self):
        self.runner_object.start_action()

    def stop_experiment_action(self):
        self.runner_object.stop_action()

    def set_static_parameters(self, provider_name, target_name, freq, exp_id, time_shift):
        self.freq = freq
        self.table_widget.setItem(0, 1, QTableWidgetItem(provider_name))
        self.table_widget.setItem(1, 1, QTableWidgetItem(target_name))
        self.table_widget.setItem(2, 1, QTableWidgetItem(str(freq)))
        self.table_widget.setItem(3, 1, QTableWidgetItem(str(exp_id)))
        self.table_widget.setItem(4, 1, QTableWidgetItem(str(time_shift)))

    def set_dynamic_parameters(self, t, data_len, current_action, rec_action, epoch_accuracy, total_accuracy, alpha_value, gamma_value, beta_value):
        self.table_widget.setItem(5, 1, QTableWidgetItem(str(t)))
        self.table_widget.setItem(6, 1, QTableWidgetItem(str(data_len)))
        self.table_widget.setItem(7, 1, QTableWidgetItem(str(current_action)))
        self.table_widget.setItem(8, 1, QTableWidgetItem(str(rec_action)))
        self.table_widget.setItem(9, 1, QTableWidgetItem(str(epoch_accuracy)))
        self.table_widget.setItem(10, 1, QTableWidgetItem(str(total_accuracy)))
        self.table_widget.setItem(11, 1, QTableWidgetItem(str(alpha_value)))
        self.table_widget.setItem(12, 1, QTableWidgetItem(str(gamma_value)))
        self.table_widget.setItem(13, 1, QTableWidgetItem(str(beta_value)))

        self.stats.set_data(alpha_value, gamma_value, beta_value)
```

```
        # self.spectral.set_size(data_len, self.freq)

    def set_graph_data(self, data_spectral):
        self.spectral.set_data(data_spectral)

    def set_values_from_table(self, item):
        if item.column() == 1:
            if item.row() == 14:
                self.runner_object.set_value("p1", item.text())
            elif item.row() == 15:
                self.runner_object.set_value("p2", item.text())
```

runner.py

```
import os
from dataclasses import dataclass
from time import sleep

import numpy as np
from PyQt5.QtCore import QObject, pyqtSignal

from classifier import Classifier

from experiments.control_offline import test_run_control_offline
from experiments.control_online import test_run_control_online
from experiments.self_learning_online import ExperimentSelfLearningOnline
from experiments.self_learning_forward_online import ExperimentSelfLearningForwardOnline
from experiments.self_learning_offline import ExperimentSelfLearningOffline
from experiments.alpha_online import ExperimentAlphaOnline

from loader import Loader
from markers_psytask import MarkersPsytask
from provider import Provider, save_eeg
from rhm_analyze import get_indicator
from speaker import Speaker
from target import Target
from utils.common import get_markers_psy
```

```
def new_experiment():
    # mode = 0 - forward offline
    # mode = 1 - forward online
    # mode = 2 - self-learning online
    # mode = 3 - self-learning imitation offline
    # mode = 4 - alpha rhythm online
    # mode = 5 - alpha rhythm offline
    mode = 0

    speaker = Speaker()
    # speaker.play_long()
    # speaker.play_small()

    # target = Target("trik")
    target = Target("chair")

    # test_learn_loader = Loader(method="event")
    test_learn_loader = Loader()
    markers_psy = MarkersPsytask(path+"Marks/Imaginary_Movement.PRO")
    print(markers_psy.marks)

    test_markers, test_markers_l, test_markers_r, test_markers_n = get_markers_psy(markers_psy.marks)
    if mode == 0:
        forward_learn_loader = Loader()
        forward_learn_loader.load_dataset(os.path.realpath(os.path.dirname(os.path.realpath(__file__))) +
"/saved_eeg_new/positive/forward", 0, 45)
        forward_learn_loader.load_dataset(os.path.realpath(os.path.dirname(os.path.realpath(__file__))) +
"/saved_eeg_new/positive/stop", 1, 45)
        forward_classifier = Classifier()
        forward_classifier.learn('lr', forward_learn_loader.dataset, forward_learn_loader.info, True)
    if mode == 1:
        forward_learn_loader = Loader()
forward_learn_loader.load_dataset(os.path.realpath(os.path.dirname(os.path.realpath(__file__)))+"/saved_eeg_n
ew/positive/forward", 0, 45)
forward_learn_loader.load_dataset(os.path.realpath(os.path.dirname(os.path.realpath(__file__)))+"/saved_eeg_n
ew/positive/stop", 1, 45)
        forward_classifier = Classifier()
        forward_classifier.learn('knn', forward_learn_loader.dataset, forward_learn_loader.info, True)
```

```
        return ExperimentSelfLearningForwardOnline(forward_learn_loader, forward_classifier, test_markers, speaker, target)
    elif mode == 2:
        classifier = Classifier()
        classifier.learn('knn', test_learn_loader.dataset, test_learn_loader.info, True)
        # sleep(1)
        # target.do_self_test()
        return ExperimentSelfLearningOnline(test_learn_loader, classifier, test_markers, speaker, target)
    elif mode == 3:
        classifier = Classifier()
        classifier.learn('fann', test_learn_loader.dataset, test_learn_loader.info, True)
        return ExperimentSelfLearningOffline(classifier, test_markers)
    elif mode == 4:
        loader = Loader()
        data = loader.load_csv()
        classifier = Classifier()
        classifier.learn('svm', data, None, True)
        return ExperimentAlphaOnline(classifier, target)
    elif mode == 5:
        loader = Loader()
        data = loader.load_csv()
        classifier = Classifier()
        classifier.learn('fann', data, None, True)
        return
    pass

class RunnerObject(QObject):
    set_static_parameters = pyqtSignal(str, str, int, int, float)
    set_dynamic_parameters = pyqtSignal(float, int, str, str, float, str, float, float, float)
    set_graph_data = pyqtSignal(np.ndarray)
    experiment = None

    @dataclass
    class LocalSignals:
        set_static_parameters: pyqtSignal
        set_dynamic_parameters: pyqtSignal
        set_graph_data: pyqtSignal
```

```
    def start(self):
        local_signals = self.LocalSignals(self.set_static_parameters, self.set_dynamic_parameters,
self.set_graph_data)
        self.experiment = new_experiment()
        self.experiment.run(local_signals)


    def set_value(self, name, value):
        self.experiment.set_value(name, value)

    def start_action(self):
        self.experiment.start_action()

    def stop_action(self):
        self.experiment.stop_action()
```