МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра информационных систем управления

РАДОВСКИЙ Александр Олегович

ТЕХНОЛОГИЯ УПРАВЛЕНИЯ ГРУППАМИ МОБИЛЬНЫХ АВТОНОМНЫХ ОБЪЕКТОВ В ТУРБУЛЕНТНОЙ СРЕДЕ

Дипломная работа

Научный руководитель:
кандидат технических наук,
доцент
А. Н. Вальвачев

Допущена к защите
«____»_____20___г.
Заведующий кафедрой информационных систем управления
Доктор технических наук, доцент А.М. Недзьведь

Минск, 2025

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

ЛА – летальный аппарат
ЦУ – центр управления
BDI – Belief Desire Intention
UML – Unified Modeling Language
TCP – Transmission Control Protocol
HTTP – Hypertext Transfer Protocol
MQTT – Message Queuing Telemetry Transport
DOS – Data Distribution Service
ROS – Robot Operating System
RRT – Rapidly-exploring Random Tree
IoT – Internet of Things
XSS – Cross-Site Scripting

РЕФЕРАТ

Структура и объём дипломной работы: 62 страницы, 25 рисунков, 1 таблица, 3 приложения, 14 источников

Ключевые слова: DRONE, SWARM, PYTHON, DJANGO, ГРУППЫ МОБИЛЬНЫХ ОБЪЕКТОВ, УПРАВЛЕНИЕ В ТУРБУЛЕНТНОЙ СРЕДЕ, UML, UAV

Объект исследования – управление группами автономных мобильных устройств в условиях турбулентной среды, включая координацию разнородных объектов, адаптацию к динамическим изменениям и минимизацию влияния неопределённостей.

Предмет исследования – методы, алгоритмы и программные решения для моделирования жизненного цикла групп автономных устройств, их взаимодействия в условиях внешних возмущений, а также оптимизации распределения задач и ресурсов.

Цель исследования – разработка комплексной технологии управления группами автономных мобильных устройств, обеспечивающей устойчивость, адаптивность и эффективность выполнения задач в динамической среде с неполной информацией и противодействием.

Методы исследования – а) теоретические: анализ научной литературы в области адаптивного управления, мультиагентных систем, имитационного моделирования и алгоритмов оптимизации; формализация моделей жизненного цикла группы, сцен операций и негативных воздействий; б) практические: проектирование архитектуры веб-платформы на базе Django и Python; реализация алгоритмов централизованного, децентрализованного и мультиагентного управления; тестирование системы на прикладных задачах с использованием инструментов OpenCV и IoT-протоколов (HTTP).

Полученные результаты – разработаны оригинальные модели управления группами устройств, интегрирующие методы прогнозирования, адаптации и распределённого принятия решений и создан программный комплекс, поддерживающий гибридные сценарии управления (централизованный, децентрализованный, мультиагентный) и обеспечивающий динамическую корректировку маршрутов в условиях помех.

Область возможного практического применения – имитационное моделирование в сфере логистики и транспорта, спасательных операций, умных городских инфраструктурах, военных операциях и сельском хозяйстве.

РЭФЕРАТ

Структура і аб’ём дыпломнай працы: 62 старонкі, 25 малюнкаў, 1 табліца, 3 дадатка, 14 крыніц

Ключавыя словы: DRONE, SWARM, PYTHON, DJANGO, ГРУПЫ МАБІЛЬНЫХ АБ'ЕКТАЎ, КІРАВАННЕ Ў ТУРБУЛЕНТНАЙ АСЯРОДДЗІ, UML, UAV

Аб’ект даследавання – кіраванне групамі аўтаномных мабільных прылад ва ўмовах турбулентнай асяроддзя, уключаючы каардынацыю разнастайных аб'ектаў, адаптацыю да дынамічным зменам і мінімізацыю ўплыву нявызначанасцяў.

Прадмет даследавання – метады, алгарытмы і праграмныя рашэнні для мадэлявання жыццёвага цыкла груп аўтаномных прылад, іх узаемадзеяння ва ўмовах знешніх абурэнняў, а таксама аптымізацыі размеркавання задач і рэсурсаў.

Мэта даследавання – распрацоўка комплекснай тэхналогіі кіравання групамі аўтаномных мабільных прылад, якая забяспечвае ўстойлівасць, адаптыўнасць і эфектыўнасць выканання задач у дынамічнай асяроддзі з няпоўнай інфармацыяй і процідзеяннем.

Метады даследавання – а) тэарэтычныя: аналіз навуковай літаратуры ў галіне адаптыўнага кіравання, мультиагентных сістэм, імітацыйнага мадэлявання і алгарытмаў аптымізацыі; фармалізацыя мадэляў жыццёвага цыкла групы, сцэн аперацый і негатыўных уздзеянняў; б) практычныя: праектаванне архітэктуры вэб-платформы на базе Django і Python; рэалізацыя алгарытмаў цэнтралізаванага, дэцэнтралізаванай і мультиагентного кіравання; тэставанне сістэмы на прыкладных задачах з выкарыстаннем інструментаў OpenCV і IoT-пратаколаў (HTTP).

Атрыманыя вынікі – распрацаваны арыгінальныя мадэлі кіравання групамі прылад, інтэгравальныя метады прагназавання, адаптацыі і размеркаванага прыняцця рашэнняў і створаны праграмны комплекс, які падтрымлівае гібрыдныя сцэнары кіравання (цэнтралізаваны, дэцэнтралізаванай, мультиагентный) і які забяспечвае дынамічную карэкціроўку маршрутаў ва ўмовах перашкод.

Вобласць магчымага практычнага прымянення – імітацыйнае мадэляванне ў сферы лагістыкі і транспарту, выратавальных аперацый, разумных гарадскіх інфраструктурах, ваенных аперацыях і сельскай гаспадарцы.

SUMMARY

Structure and scope of the diploma work: 62 pages, 25 figures, 1 table, 3 appendixes, 14 references

Keywords: DRONE, SWARM, PYTHON, DJANGO, GROUPS OF MOBILE OBJECTS, MANAGEMENT IN A TURBULENT ENVIRONMENT, UML, UAV

The object of the research – managing groups of autonomous mobile devices in a turbulent environment, including coordinating heterogeneous objects, adapting to dynamic changes, and minimizing the impact of uncertainties.

The subject of the research – methods, algorithms and software solutions for modeling the life cycle of groups of autonomous devices, their interaction in conditions of external disturbances, as well as optimizing the allocation of tasks and resources.

The aim of the research – development of an integrated technology for managing groups of autonomous mobile devices, ensuring stability, adaptability and efficiency of tasks in a dynamic environment with incomplete information and counteraction.

Research methods – a) theoretical: analysis of scientific literature in the field of adaptive management, multi-agent systems, simulation modeling and optimization algorithms; formalization of group lifecycle models, scenes of operations and negative impacts; b) practical: designing the architecture of a web platform based on Django and Python; implementation of algorithms for centralized, decentralized and multi-agent management; testing the system on application tasks using OpenCV tools and IoT protocols (HTTP).

The results of the work – original device group management models have been developed that integrate forecasting, adaptation, and distributed decision-making methods, and a software package has been created that supports hybrid control scenarios (centralized, decentralized, and multi-agent) and provides dynamic route correction in interference conditions.

Recommendations on the usage – simulation modeling in the field of logistics and transport, rescue operations, smart urban infrastructures, military operations and agriculture.

ВВЕДЕНИЕ

Технологии управления мобильными автономными объектами широко применяются в различных областях: от военных операций и исследования сложных природных условий до гражданских сфер, таких как мониторинг окружающей среды, логистика и спасательные миссии. Важнейшей особенностью таких систем является необходимость организации эффективного взаимодействия объектов, работающих в условиях ограниченных ресурсов и высокой неопределенности внешней среды.

Особое внимание уделяется управлению в турбулентных средах, где наблюдаются сложные динамические изменения, нелинейные взаимодействия и значительные возмущения. Это может быть как воздушная, так и подводная среда, где турбулентность затрудняет прогнозирование поведения объектов и координацию их действий.

Цель данной дипломной работы – разработка и совершенствование технологий управления группами мобильных автономных объектов, способных адаптироваться к турбулентным условиям. В рамках работы будут изучены алгоритмы координации и навигации, обеспечивающие устойчивость и надежность таких систем, а также методы оптимизации взаимодействия объектов для выполнения заданных задач.

Актуальность темы связана с растущим интересом к разработке автономных систем, которые могут эффективно работать в сложных и нестабильных условиях. Это открывает новые горизонты для развития робототехники, аэрокосмической и морской инженерии, а также других высокотехнологичных областей. Исследование направлено на решение важных задач управления группами в турбулентных средах и имеет как теоретическое, так и практическое значение.

ГЛАВА 1
АНАЛИЗ ПРОБЛЕМЫ И ПОСТАНОВКА ЗАДАЧИ

1.1 Анализ проблемы

В последние десятилетия как в нашей стране, так и за рубежом всё большее внимание уделяется созданию систем автоматизированного планирования применения летательных аппаратов (ЛА). Это связано с целым рядом факторов. Во-первых, с ростом количества и разнообразия типов ЛА, привлекаемых для решения различных задач. Во-вторых, с расширением круга самих задач, которые выполняют группы совместно действующих аппаратов. В-третьих, с многообразием оборудования, устанавливаемого на ЛА, и множеством способов их применения [1].

Ещё одним важным моментом является неравномерное распределение сил и средств по разным объектам и аэродромам, что требует их согласованного совместного использования для достижения целей. Кроме того, планирование серьёзно усложняется из-за неопределённостей – недостатка информации о целях, активного противодействия со стороны противника и постоянно меняющейся обстановки во время выполнения задания [2, 3].

Все эти факторы порождают огромное количество возможных сценариев применения авиационных групп. Провести их полноценный анализ вручную без внедрения средств автоматизации практически невозможно.

На сегодняшний день часть этих задач решается с помощью специализированных информационных систем, которые автоматизируют процесс планирования действий ЛА, в основном через подготовку полётных заданий. Такие системы значительно упрощают работу: избавляют оператора от сложных вычислений, предоставляют доступ к нормативно-справочным материалам и ускоряют рутинные операции. Однако по своей сути они остаются инструментами поддержки – ответственность за итоговый результат по-прежнему лежит на человеке, принимающем решение. Эффективность планирования при этом сильно зависит от его опыта и умения правильно оценить выбранные решения по заданным критериям.

В этой связи всё острее встаёт вопрос о разработке новых автоматизированных технологий управления группами мобильных автономных объектов, которые смогут эффективно взаимодействовать между собой в условиях турбулентной среды и соответствовать современным требованиям. При этом особое значение приобретают следующие аспекты:

1. Многообразие типов летательных аппаратов и задач: современные группы ЛА включают объекты с разными техническими характеристиками и оборудованием, что позволяет решать широкий спектр задач – от

разведывательных до спасательных операций. Однако интеграция разнородных объектов требует сложной координации, особенно в условиях нестабильной турбулентной среды. Разнообразие целевых задач, решаемых группами ЛА, накладывает дополнительные ограничения на их взаимодействие

2. Ограниченность ресурсов и неравномерное распределение сил Совместное использование разнородных ресурсов, особенно в условиях их дефицита, становится одной из ключевых задач. Неравномерное распределение сил по объектам и аэродромам осложняет их согласованное применение, особенно когда требуется быстрая адаптация к изменяющимся условиям [4].

3. Присутствие неопределенностей и противодействия Планирование действий ЛА осложняется недостаточной информированностью о целевых объектах, изменчивостью обстановки и активным противодействием со стороны противника. Турбулентные условия добавляют к этому внешние физические возмущения, требующие адаптивных решений.

4. Ограничения существующих информационных систем Современные автоматизированные системы планирования действий ЛА предоставляют специалистам мощные инструменты для работы с данными и выполнения расчетов, но остаются в значительной степени зависимыми от опыта пользователя.

1.2 Основные понятия и определения

Для однозначного понимания дипломной работы сформируем понятийный базис из основных терминов и определений.

Проект – задача, решение которой требует применения автономных мобильных устройств.

Автономное мобильное устройство – устройство, двигающееся по заданной траектории и формирующее траекторию самостоятельно.

Группа – комплекс автономных устройств, достаточных для выполнения проекта.

Траектория движения – координаты, по которым движется устройство в процессе выполнения проекта.

Жизненный цикл группы – этапы функционирования группы во время выполнения проекта.

Проблемы управления группой мобильных устройств включают несколько ключевых аспектов. Во-первых, существует сложность построения комплексной модели, которая формирует целостный взгляд на задачу, группу и среду, в которой будет реализован проект. Это особенно актуально в условиях турбулентной среды, где динамические изменения требуют учета множества факторов. Во-вторых, затруднено имитационное моделирование полного жизненного цикла группы для конкретного проекта перед его

началом, что препятствует оптимизации планирования. В-третьих, возникает сложность в прогнозировании возможного внешнего воздействия при планировании проекта, включая возмущения, вызванные турбулентностью и противодействием со стороны внешних факторов. Четвертым аспектом является трудность обхода препятствий, возникающих на траектории, что крайне важно для автономных устройств в условиях высокой неопределенности. Наконец, важным является вызов оперативной оценки состояния группы в процессе выполнения проекта и синтеза действий для минимизации потерь, особенно если группа состоит из разнородных объектов с различным уровнем функциональности и ресурсов.

Для решения перечисленных проблем требуется разработка подходов, которые учитывают:

1. Интеграцию методов адаптивного управления и прогнозирования поведения среды;
2. Моделирование взаимодействия автономных устройств в условиях внешних возмущений;
3. Разработку алгоритмов, обеспечивающих согласованное функционирование группы в рамках единой стратегии.

Данная дипломная посвящена комплексному решению перечисленных проблем, включая исследование методов и алгоритмов управления группами автономных мобильных устройств в условиях турбулентной среды, что обеспечит повышение надежности и эффективности выполнения поставленных задач.

1.3 Постановка задачи

Рассмотрим организационный центр, специализирующийся на прикладных проектах с использованием координируемых групп мобильных объектов. Организация задействует гетерогенные типы устройств для выполнения многофункциональных операций. Группы формируются динамически на основе требований задачи и параметров заранее заданной траектории полета.

Приведем более формальное определение системы:

Дано гетерогенная группа устройств (DG). Система оперирует полиморфным ансамблем устройств:

$$DG = \{DT_1, DT_2, \dots, DT_n\} \quad (1.1)$$

где DT_i - различные типы устройств (например, БПЛА, наземные роботы, аэросенсоры) с автономными функциональными возможностями.

Каждое устройство оснащено датчиком контроля состояния (ДКС) для оценки параметров в реальном времени:

$$\text{ДКС} = \{s_{DT_1}, s_{DT_2}, \dots, s_{DT_n}\} \quad (1.2)$$

где S_{DTi} - количественные параметры состояния (уровень заряда, позиционная точность, функциональная целостность).

Задачи для выполнения формализуются как кортежи с многомерными ограничениями:

$$\text{PSM} = (\text{PT}, \text{RDT}, \text{Qty}, \text{TP}, \text{PS}, \text{MO}, \text{Dur}, \text{EC}) \quad (1.3)$$

где

1. PT: Тип проекта (наблюдение, доставка, экологический мониторинг)
2. RDT: Требуемый тип устройства (совместимый с DTi)
3. Qty: Количество устройств
4. TP: Профиль траектории (3D-маршрут, ограничения скорости)
5. PS: Спецификации полезной нагрузки (масса, габариты, протоколы безопасности)
6. MO: Цели миссии (критерии успеха, KPI)
7. Dur: Продолжительность операции
8. EC: Экологические ограничения (воздушное пространство, погодные условия)

В процессе реализации проекта группа подвергается негативным воздействиям. Операционные риски моделируются как стохастические возмущения, влияющие на целостность группы:

$$\text{NIM} = \{\text{NIT}_1(\text{IL}_1, \text{ID}_1), \text{NIT}_2(\text{IL}_2, \text{ID}_2), \dots, \text{NIT}_\text{k}(\text{IL}_\text{k}, \text{ID}_\text{k})\} \quad (1.4)$$

где

NIT_k: Тип негативного воздействия (турбулентность, динамические препятствия, подавление сигнала)

IL_k: Уровень интенсивности (количественная величина возмущения)

ID_k: Длительность воздействия (временной интервал нарушения)

Требуется разработать методы и программное обеспечения для моделирования жизненного цикла группы в рамках имитационного подхода. Основное требование к результату – решение проблем, указанных в п.1.2.

1.4 Выводы

В первой главе проведен тщательный анализ научной литературы, посвященной управлению группами мобильных автономных объектов, что позволило выявить актуальную проблему – необходимость разработки эффективных методов координации таких групп в условиях неопределенности и динамических изменений.

На основе анализа была четко сформулирована формальная задача исследования: создание комплексной технологии, обеспечивающей управление группами автономных устройств в турбулентной среде.

ГЛАВА 2
МОДЕЛИ И АЛГОРИТМЫ

2.1 Модели

2.1.1 Жизненный цикл мобильной автономной группы

Жизненный цикл автономной мобильной группы можно представить как модель, включающую этапы, которые описывают полный процесс функционирования группы – от формирования до завершения проекта. Эта модель охватывает ключевые стадии подготовки, выполнения и завершения работы группы в условиях, заданных проектом. Комплекс моделей, описывающих группу мобильных объектов, должен отвечать требованиям задач, решаемых в процессе реализации жизненного цикла группы при решении практических задач в реальных условиях [4, 5].

При рассмотрении модели жизненного цикла мобильной автономной группы можно выделить ряд основных этапов:

1. Уяснение поставленной задачи, определение цели и траектории. Цель этапа – сформировать четкий план действий на основе требований миссии. Для этого анализируется тип задачи (доставка, мониторинг и т.д.), определяются конечные цели (координаты точки, сбор данных) и строится оптимальная траектория с учетом рельефа, зон ограничений и погодных условий, например, обход зон сильных ветров для дронов в горной местности.

2. Формирование группы, обеспечивающей решение задачи. Цель – подобрать устройства, соответствующие требованиям миссии. Выбираются типы техники (дроны, роботы-вездеходы), проверяются их характеристики (грузоподъемность, дальность), формируется группа с резервом на случай сбоев, как в случае использования дронов с тепловизорами и наземных роботов для мониторинга леса.

3. Старт. Проводится диагностика устройств (заряд батарей, калибровка датчиков), загружается траектория в бортовые компьютеры, устанавливается связь между участниками группы, например, синхронизация GPS-координат дронов.

4. Движение по заданному маршруту. Группа продвигается к цели, поддерживая обмен данными о местоположении и состоянии устройств, корректируя скорость для сохранения целостности, как при движении роботов колонной с безопасной дистанцией.

5. Появление препятствия – облет и выход на курс. При обнаружении препятствий (скалы, деревья) датчики фиксируют помехи, маршрут автоматически пересчитывается для обхода, группа перестраивается и возобновляет движение, например, дрон меняет высоту полета при внезапном появлении дерева.

6. Продолжение движения. Контролируется восстановленный маршрут, проверяется состояние устройств, центр управления уведомляется об изменениях, как при возврате группы к исходной траектории после обхода препятствия.

7. Негативное воздействие. Пыльная буря. Оценка состояния устройств. Оценить последствия экстремальных условий. Определяется работоспособность устройств (целые, поврежденные), проверяется качество данных от уцелевшей техники, анализируется возможность достижения цели, например, при потере 2 дронов из 5 оставшиеся проверяются на функциональность.

8. Принятие решений для продолжения или возврата. На основе оставшихся ресурсов (заряд батарей, число рабочих устройств) сравниваются риски продолжения и возврата, принимается решение, как в случае прерывания миссии из-за потери 60% техники и возврата на базу

Применение этапов 1-8 представлено на рисунке 1.1.

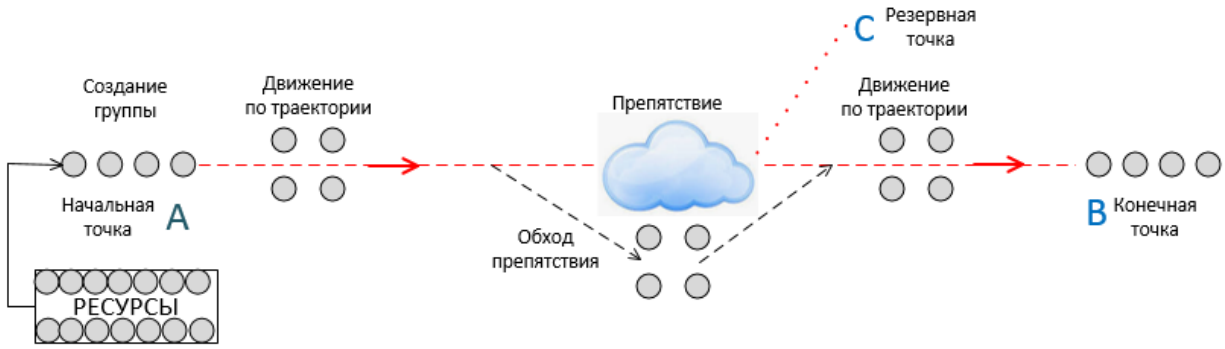

Рисунок 2.1 – Схема движения группы

На основании анализа литературы выделяют несколько основных подходов к пониманию модели. В контексте динамических сред и задач, требующих тесной координации множества сетевых устройств, ключевое значение приобретает выбор адекватных систем и архитектуры управления жизненным циклом конкретной группы.

Централизованный подход базируется на использовании единого центра управления (ЦУ), который осуществляет полный контроль над формированием рабочей группы, распределением целей, задач, назначением людей и коррекцией целевых и динамических траекторий [6]. Устройства в такой системе выполняют предзагруженные инструкции, а все данные с датчиков состояния (s_{DT_i}) передаются в ЦУ для анализа. Преимущество данного метода заключается в простоте разработки, реализации, оперативности и предсказуемости управленческих решений. Однако зависимость от стабильной связи с ЦУ делает информационную систему более уязвимой к сбоям коммуникации, что ограничивает её применение в условиях высокой коммерческой неопределённости. Типичными сценариями

использования являются телекоммуникационные и промышленные линии с гарантированной
инфраструктурой сотовой связи.

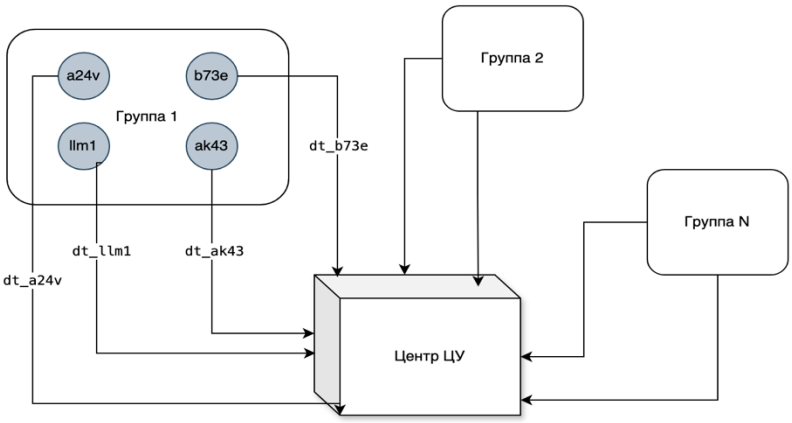

Рисунок 2.2 – Схема централизованного управления

Децентрализованный подход предполагает полную автономию электронных устройств, которые взаимодействуют через локальные протоколы, такие как алгоритмы роевого интеллекта (например, *flocking*). Решения принимаются на основе локальных данных, что обеспечивает высокую устойчивость к системным сбоям и быструю адаптацию к изменениям информационной среды [7].

Формирование группы происходит через самоорганизацию, где

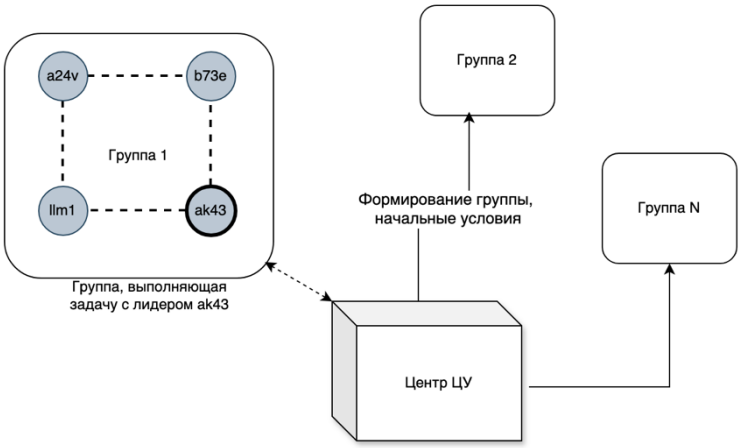

Рисунок 2.3 – Схема децентрализованного управления

устройства динамически выбирают лидера на основе критериев вроде уровня заряда или вычислительной мощности

$$L = \arg\max\left(s_{DT_i}^{\text{батарея}} \cdot s_{DT_i}^{\text{выч.мощность}}\right) \quad (2.1)$$

Недостатком является сложность достижения глобально оптимальных решений, что делает целенаправленный подход менее пригодным для задач с жёсткими ресурсными ограничениями. Децентрализованные поисковые системы находят широкое применение в поисково-спасательных операциях, военных маневрах и промысловом и сельском хозяйстве.

Гибридный подход комбинирует элементы централизованного и децентрализованного управления, создавая структурную и вертикальную иерархическую структуру, интегрируя иерархическое единство и управление. Глобальные цели задаются ЦУ, а локальная оптимизация траекторий, оптимизация заданий и решение задач осуществляются независимыми устройствами или сублидерами.

Например, итоговая траектория $TP_{\text{гибрид}}$ может быть выражена как взвешенная сумма централизованно заданного маршрута и локальных корректировок

$$\mathrm{T}P_{\text{гибрид}} = \alpha \cdot \sum \Delta TP_{\text{лок}_i} \qquad (2.2)$$

Такой подход балансирует между внутренней устойчивостью и предсказуемостью, что делает его более эффективным в умных городских и архитектурных системах, где требуется четкая координация между статичной городской инфраструктурой и мобильными объектами.

Мультиагентный подход рассматривает каждое электронное устройство как автономного агента, способного к постоянным переговорам, обучению, обучению и перераспределению ролей. Планирование задач осуществляется через механизмы аукционов или онтологий, где агенты формируют коалиции на основе оценки стоимости и рисков ($bid = 1 / (\text{стоимость} + \text{риск})$). Адаптация к возмущениям реализуется через модели $Belief - Desire - Intention$ (BDI), где намерения агентов ($Intention$) определяются их восприятием среды ($Belief$) и целями ($Desire$). Этот подход обеспечивает максимальную гибкость и масштабируемость, но требует значительных вычислительных ресурсов, навыков и аппаратных средств и сложной информационной и программной и вычислительной инфраструктуры. Применяется в гибких производственных системах, морских и океанических космических миссиях [8].

Итак, централизованные информационные системы доминируют в сложных и предсказуемых и стабилизированных средах, тогда как децентрализованные и мультиагентные архитектуры незаменимы в условиях высоких динамических и глобальных возмущений. Гибридные решения заполняют нишу между этими крайностями, предлагая высокую адаптивность и эффективность без полного отказа от внутреннего контроля. Выбор оптимальной экономической модели зависит от требований к финансовой

устойчивости, скорости принятия решений, условий функционирования организации и доступности финансовых ресурсов, что подчёркивает важность и необходимость стратегического и ситуационного анализа и прогнозирования и управленческого учета при проектировании систем коллективного и группового и централизованного управления.

2.1.2 Модель сцены

Модель сцены представляет собой структурированное описание среды, в которой группа мобильных устройств выполняет задачу. Она объединяет физические и логические элементы, необходимые для имитации жизненного цикла группы, и адаптируется под различные подходы управления: централизованный, децентрализованный, гибридный и мультиагентный.

$$\text{Scene} = \langle \text{ID}, \text{Center}, \text{Group}, \text{Trajectory}, \text{Hazards}, \text{Communications} \rangle \quad (2.3)$$

Идентификатор сцены задаёт уникальный контекст миссии, включая её тип (доставка, мониторинг, поиск), географические координаты и временные параметры. Например, сцена может моделировать доставку медицинских грузов в горный регион с учётом сезонных погодных условий. В зависимости от подхода к управлению, идентификатор также определяет, как будут распределяться роли между устройствами. В централизованных системах это может быть жёсткое назначение задач, тогда как в мультиагентных - динамическое формирование коалиций через аукционы.

$$\text{SceneID} = \langle \text{Type}, \text{Location}, \text{Timestamp}, \text{Approach} \rangle \quad (2.4)$$

где Type – тип миссии, Location – начальные координаты (положение на карте), Timestamp – время начала миссии, Approach – подход к организации жизненного цикла группы.

Центр управления выступает ключевым элементом в централизованных и гибридных системах. Он координирует действия группы, обрабатывает данные с датчиков и принимает стратегические решения. Например, в гибридной модели центр управляет сублидерами подгрупп, которые, в свою очередь, локально оптимизируют маршруты. В децентрализованных и мультиагентных системах центр может отсутствовать, а решения принимаются устройствами самостоятельно через алгоритмы консенсуса или переговоры.

$$\text{Center} = \langle \text{Mode}, \text{Protocol}, \text{SublLeaders} \rangle \quad (2.5)$$

где Mode – тип управления (в зависимости от выбранного подхода может быть с полным контролем, или через частичное лидерство объектов в группах), Protocol – протокол взаимодействия в группе и с центром (HTTP, MQTT, ROS и другие), SubLeaders – при необходимости выделенные лидеры в группах.

Группа устройств включает мобильные объекты, их технические характеристики и распределение ролей. В централизованной системе роли

(лидер, исполнитель, резерв) назначаются заранее, а в мультиагентной – определяются в реальном времени на основе компетенций и ресурсов. Например, дрон с самым высоким уровнем заряда может автоматически стать лидером при обходе препятствий. Состав группы также зависит от точек вылета и доставки: если миссия подразумевает доставку груза в удалённую зону, в группу включаются устройства с повышенной грузоподъёмностью и дальностью действия.

$$\text{DeviceGroup} = \{\, d_i \mid d_i = \langle \text{ID}, \text{Type}, Sensors, \text{Status}, \text{Battery}, \text{Role} \rangle \,\} \quad (2.6)$$

где ID – уникальный идентификатор устройства, Type – тип устройства, Sensors – сенсоры устройства, Status – текущий статус устройства (активный, поврежден, отключен и другие), Battery – батарея, Role – при необходимости роль устройства.

Точка вылета и точка доставки определяют начальные и конечные координаты маршрута. В централизованных системах траектория между ними строится заранее с учётом статических препятствий, тогда как в децентрализованных - корректируется динамически на основе данных сенсоров. Дополнительные точки посадки добавляют гибкости, позволяя группе перераспределять задачи в случае сбоев. Например, при повреждении одного из устройств груз может быть перенаправлен к ближайшей резервной точке.

$$\text{Trajectory} = \langle \text{Start}, \text{End}, \text{Waypoints}, \text{Path} \rangle \quad (2.7)$$

где $\text{Start} = (x_1, y_1, (z_1))$ – непосредственно точка вылета, $End = (x_2, y_2, (z_2))$ – точка доставки, $\text{Waypoints} = \{wp_1, wp_2, \dots, wp_n\}$ – дополнительные точки посадки, Path – путь, если он заранее просчитан (применительно для централизованных систем).

Негативные воздействия (пыльные бури, препятствия, помехи связи) моделируются как внешние факторы, влияющие на работоспособность группы. В централизованном подходе их обработка возлагается на центр управления, который пересчитывает маршруты и перераспределяет ресурсы. В мультиагентных системах каждое устройство самостоятельно оценивает риски и адаптирует поведение. Например, при обнаружении пыльной бури дроны могут снизить скорость или изменить высоту полёта, согласовав действия через аукционы.

$$\text{Hazards} = \left\{\, h_j \mid h_j = \langle \text{Type}, \text{Intensity}, \text{StartTime}, \text{Duration} \rangle \,\right\} \quad (2.8)$$

где Type – тип негативного воздействия (пыльная буря, препятствие и другие), Intensity – сила воздействия (для простоты считаем неотрицательным числом), StartTime – время возникновения, Duration – длительность негативного воздействия.

Коммуникации обеспечивают обмен данными между устройствами и центром. В централизованных системах связь строится по принципу звезда,

где все устройства подключены к единому узлу. В децентрализованных и мультиагентных подходах используется mesh-сеть, где устройства взаимодействуют напрямую, повышая устойчивость к обрывам связи. Например, при потере связи с центром гибридная система может временно переключиться на локальное управление через сублидеров.

$$\text{Communications} = \langle G, \text{Protocol}, \text{Bandwidth} \rangle \quad (2.9)$$

где $G = (V, E)$, $V = \text{DeviceGroup}$, $E \subseteq V \times V$ – граф связей, Protocol – протокол общения (TCP, UDP и другие), Bandwidth – пропускная способность (для простоты модели полагаем ее достаточной для наших целей).

Модель сцены интегрирует все эти элементы, обеспечивая гибкость под разные сценарии. В логистических задачах с чёткими маршрутами предпочтение отдаётся централизованному управлению, тогда как в поисково-спасательных операциях в условиях неопределённости эффективнее мультиагентные системы. Ключевая цель – создать среду, где группа устройств может адаптироваться к изменениям, минимизируя риски и сохраняя функциональность.

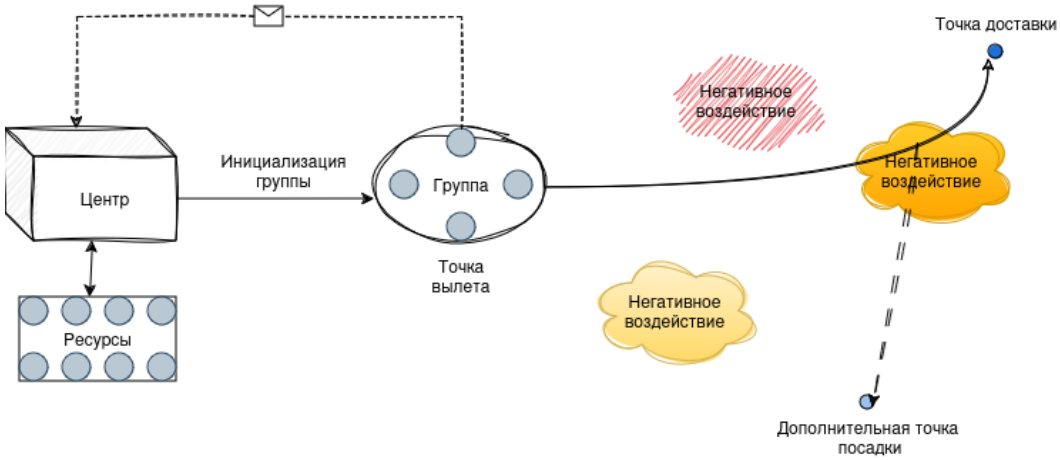

Рисунок 2.4 – Модель сцены

2.1.3 Автономное мобильное устройство

Модель автономного устройства в групповых системах объединяет аппаратные компоненты, алгоритмы управления и коммуникационные механизмы, обеспечивая выполнение задач в динамических условиях. Её ядром является ситуационная осведомленность: устройство анализирует данные сенсоров, оценивает состояние (заряд, целостность, позицию) и адаптирует поведение в реальном времени. Гибридная архитектура поддерживает как централизованное планирование, так и децентрализованные сценарии с роевым интеллектом, гарантируя устойчивость к сбоям. Такая формализация позволяет проектировать масштабируемые системы для задач любой сложности - от мониторинга до спасательных операций.

Рассмотрим по пунктам введенное выше определение-фрейм (2.6) модели автономного устройства в системе.

Каждое автономное устройство в группе обладает уникальным идентификатором (ID), который позволяет однозначно отличать его от других участников системы. Этот идентификатор используется для адресации в коммуникационных протоколах, логгирования действий и диагностики неисправностей. Классификация устройства (Type) определяет его функциональные возможности и роль в группе.

Динамика движения автономного устройства может описываться системой дифференциальных уравнений:

$$\begin{cases} \dot{x} = v\cos\theta\cos\psi \\ \dot{y} = v\cos\theta\sin\psi \\ \dot{z} = v\sin\theta \\ \dot{v} = \dfrac{T-D}{m} - g\sin\theta \end{cases} \tag{2.10}$$

где $T -$ тяга, $D = 0.5 \cdot C_d \cdot \rho \cdot S \cdot v^2 -$ сопротивление, $\psi -$ курс, $\theta -$ тангаж.

Сенсорная система (Sensors) устройства представляет собой набор датчиков, каждый из которых выполняет специфическую функцию. Лидары обеспечивают точное измерение расстояний и построение 3D-карт, GPS-модули определяют глобальные координаты, а инерциальные измерительные блоки (IMU) отслеживают ускорения и углы наклона. При моделировании данных сенсоров, в частном случае лидара, важным нюансом являются шумы и погрешность измерений. Существуют несколько альтернатив в зависимости от области применения. Для простоты реализации будем пользоваться формулой аддитивного гауссовского шума:

$$z_{\text{лидар}} = \sqrt{\left(x_{\text{цель}} - x\right)^2 + \left(y_{\text{цель}} - y\right)^2} + \mathcal{N}(0, \sigma^2) \tag{2.11}$$

Текущее состояние устройства (Status) – это динамическая совокупность параметров, отражающих его работоспособность и готовность к выполнению задач. Помимо базовых показателей (координаты, скорость, уровень заряда), система отслеживает целостность компонентов – например, степень повреждения двигателей или сенсоров. Устройство автоматически переключает статусы активности: как вариант при снижении заряда ниже 10% оно переходит в режим энергосбережения, а при механических повреждениях активирует аварийный протокол. В критических ситуациях, таких как полная потеря связи, устройство может временно брать на себя функции принятия решений, руководствуясь заложенными правилами безопасности.

Энергопотребление устройства – ключевой фактор, определяющий продолжительность его работы. Расход энергии зависит от трех основных компонентов: двигательной системы, сенсоров и коммуникационных модулей. В качестве базовой модели расхода энергии можно рассмотреть модель разряда по Пейкерту, тогда важным аспектом будет оптимизация скорости полета для минимизации энергопотребления:

$$\begin{cases} v_{\text{opt}} = \arg\min_{v} \left(\dfrac{P_{\text{move}}(v)}{v} \right) \\ P_{\text{move}} = k_1 v^3 + k_2 v + k_3 - \text{мощность движения} \end{cases} \tag{2.12}$$

Роль устройства определяется архитектурой управления всей системы. В централизованных системах роли жестко фиксированы: выделяется лидер, ответственный за планирование, и исполнители, следующие его указаниям. В мультиагентных системах роли динамически перераспределяются через алгоритмы коалиций – например, при обнаружении пожара одно устройство становится разведчиком, а другие формируют цепь для передачи данных. В гибридных моделях сублидеры управляют подгруппами, сохраняя связь с центральным узлом. Смена ролей происходит при изменении условий: выходе лидера из строя, появлении новых задач или изменении приоритетов миссии.

2.1.4 Негативные воздействия

Негативные воздействия (пыльные бури, препятствия, помехи связи) моделируются как внешние факторы, влияющие на работоспособность группы. Если в общем, то модель негативного воздействия задаёт совокупность внешних факторов, которые могут ухудшить работу автономных устройств – от погодных аномалий до физических препятствий и радиоэлектронных помех. Основная цель подраздела – дать чёткое описание структуры каждого опасного события, его динамики и способов учёта в алгоритмах управления группой.

Для этого рассмотрим введенный выше описание-фрейм (2.8), переходя к формальному математическому языку – каждый элемент множества негативных воздействий описывается кортежем

$$h_j = \left(\text{Type}_\text{j}, I_j, t_j^{\text{start}}, T_j \right) \tag{2.13}$$

для простоты и наглядности силу воздействия можно считать постоянной на всем интервале происходящего негативного воздействия, тогда получаем

$$I_j(t) = \begin{cases} I_j, & t_j^{\text{start}} \le t \le t_j^{\text{start}} + T_j \\ 0, & \text{иначе} \end{cases} \quad (2.14)$$

Понятно, что в более тонких моделях вводят зависимость от времени (линейное нарастание или затухание, гауссов профиль и т.п.), однако постоянная интенсивность упрощает расчеты и хорошо подходит для составления первичных оценок.

Так как указанные выше события непосредственно воздействуют на устройства, то необходимо рассмотреть математическую модель их влияния на надежность устройств. Для этого для каждого негативного воздействия h_j введем коэффициент деградации надежности

$$\lambda_j = \lambda_0 + kI_j \quad (2.15)$$

где λ_0 – базовая интенсивность отказов без внешних факторов, k – чувствительность к данному виду воздействия. Тогда, прибегая к теории вероятностей, вероятность безотказной работы устройства в период воздействия h_j можно оценить как

$$P_{\text{survive},j} = \exp(-\lambda_j T_j) \quad (2.16)$$

Если на устройство оказывают воздействия одновременно несколько факторов, то общую вероятность выживания на интервале можно аппроксимировать произведением соответствующих вкладов (для простоты считаем их статистически независимыми). Исходя из оценок безотказной работы можно производить, например, корректировку маршрутов и задач, а также перестроение группы или уход на дополнительные точки посадки.

2.1.5 Центр управления

Центр управления выступает ключевым элементом в централизованных и гибридных системах. По своей сути он является узлом глобального видения миссии, где обрабатываются телеметрические потоки, учитываются внешние воздействия и принимаются решения о корректировках маршрутов и ролей. При этом центр может как напрямую управлять каждым агентом, так и делегировать часть функций локальным сублидерам для оперативной адаптации в полевых условиях.

Исходя из определения (2.5), центр может работать в нескольких режимах контроля:

1. Централизованный (или полный) контроль. Центр держит руль всей группы, отправляя точные команды каждому агенту. Такой подход обеспечивает максимальную глобальную оптимизацию, но создаёт единый узел отказа: при проблемах связи или сбое центра система полностью теряет управление.

2. Частичный контроль. Центр задаёт лишь общие цели и границы задач, а локальные сублидеры внутри подгрупп самостоятельно выполняют тонкую настройку маршрутных решений и перераспределение задач на местах. Это повышает адаптивность и снижает нагрузку на центральный узел, сохраняя при этом стратегическую координацию.

3. Децентрализованный контроль. Центр предоставляет лишь сводную информацию задач, а все конкретные решения принимаются устройствами по алгоритмам консенсуса и аукционам. Такая схема максимально устойчива к отказам и полезна в условиях ненадёжной связи, но теряет преимущества глобального планирования.

Крайне важным в коммуникации мобильных автономных устройств и центра является выбор протокола взаимодействия. Для передачи команд и сбора телеметрии могут использоваться разные варианты, приведем сравнительную характеристику:

Таблица 2.1 – Протоколы взаимодействия

Протокол	Преимущества	Недостатки
HTTP/REST	Широко распространён, легко интегрируется с веб-сервисами и имеет богатую экосистему инструментов и библиотек Поддерживает сложные структуры данных и стандарты (JSON, XML), обширную аутентификацию и авторизацию (OAuth, JWT)	Высокие накладные расходы на заголовки и установку TCP-соединения, что увеличивает задержки и потребление энергии на "тонких" устройствах Подходит только для запросно-направленного обмена, плохо масштабируется в сценариях pub/sub.
MQTT	Очень лёгкий протокол с малым объемом служебных данных, оптимизирован для нестабильных сетей и устройств с ограниченными ресурсами Поддерживает QoS-уровни и, сохраняет сессии, что гарантирует доставку сообщений даже при обрывах связи	Зависимость от брокера, единая точка отказа и потенциальная узкая часть при масштабировании
ROS DDS	Децентрализованная архитектура без центрального брокера, высокая отказоустойчивость Гибкая система QoS-политик (задержка, надежность, приоритет) для реального времени	Сложность настройки и крупный размер библиотек Более высокая нагрузка на процессор и сеть по сравнению с MQTT для простых сценариев

В гибридных системах центр управления формирует небольшие тактические звенья, возглавляемые сублидерами, которым передаются права

на локальное принятие решений. Сублидеры оперативно адаптируют маршруты при обнаружении новых негативных воздействий, проводят локальные раунды аукционирования и собирают агрегированную статистику для последующей передачи в центр. Это устраняет узкие места в коммуникации, ускоряет реакцию на изменения и повышает отказоустойчивость.

2.1.6 Коммуникации

Коммуникации обеспечивают обмен данными между устройствами и центром. Рассмотрим подробную модель коммуникаций между автономными мобильными устройствами и центром управления, возможные топологии сетей, выбор протоколов и их параметры.

Так как, исходя из определения (2.9), коммуникационная сеть задается графом, то на основе анализа литературы, можно выделить основные варианты топологий сети: звездная и mesh топологии.

Звездная топология – простейший вариант топологии, характерный для централизованных систем. Все узлы напрямую подключены к центральному хабу, а между периферийными узлами прямых связей нет. Матрица смежности графа в этом случае имеет вид:

$$A_{i0} = A_{0i} = 1, A_{ij} = 0 (i, j \neq 0) \tag{2.17}$$

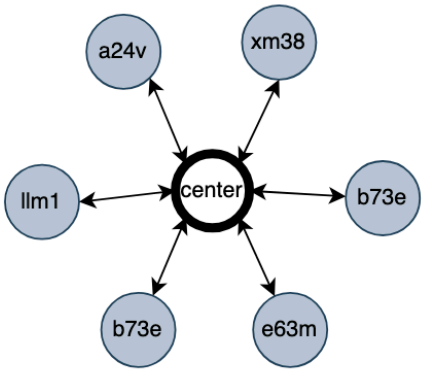

Рисунок 2.5 – Пример звездной топологии

Такой подход упрощает мониторинг и централизованное управление по определению. Однако, очевидно, что при выходе хаба из строя вся сеть перестает функционировать, поскольку центр становится единой точкой отказа.

В mesh-сети каждый узел может иметь несколько прямых соединений с другими, что обеспечивает избыточность путей и самовосстановление при локальных сбоях. Матрица смежности не имеет строгой схемы и определяется наличием физического или логического диапазона

$$A_{ij} = \begin{cases} 1, & \text{если i и j в радиусе действия друг друга,} \\ 0, & \text{иначе} \end{cases} \quad (2.18)$$

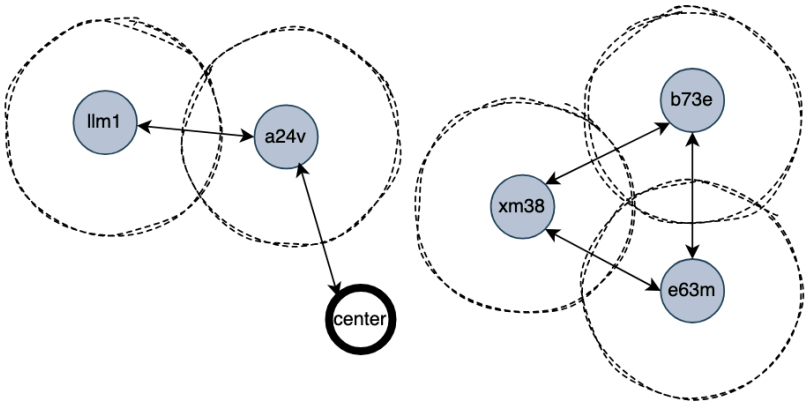

Рисунок 2.6 – Пример mesh топологии

Mesh-топология повышает устойчивость к разрывам связи и поддерживает маршрутизацию по принципу многопутевого обхода, но требует большего служебного трафика для поддержания таблиц маршрутов.

Существует еще гибридная схема, сочетающая в себе преимущества звёздной и mesh-топологий: центральный узел управляет основными потоками, но при потере связи устройства переключаются на локальную mesh-сеть внутри подгрупп, координируемых сублидерами. Это позволяет сохранить глобальное планирование и одновременно обеспечить отказоустойчивость на уровне локальных кластеров.

Протоколы взаимодействия в таких сетях аналогичны протоколам с которыми работает центр управления и представлены в таблице (2.1)

При составлении модели коммуникаций необходимо учесть возможность негативного воздействия на узлы сети (помехи). Пусть базовая (максимальная) пропускная способность канала – B_0. На каждое негативное воздействие $h_j \in H$ с условной силой I_j накладывается линейный штраф, тогда пропускная способность при одновременном воздействии факторов можно оценить как

$$B = \max\left(0, B_0 - \alpha \sum_{j=1}^{M} I_j\right) \quad (2.19)$$

где α – коэффициент деградации,

максимум – для неотрицательности

2.2 Централизованные алгоритмы

Централизованные алгоритмы предполагают наличие единого управляющего узла, который координирует все действия группы объектов. Центр принимает глобальные решения: распределяет задачи, рассчитывает маршруты, оптимизирует ресурсы и обрабатывает данные сенсоров. Такой подход обеспечивает высокую точность и предсказуемость, но требует надежной связи между центром и каждым дроном.

2.2.1 Глобальное планирование маршрутов

Планирование маршрутов – одна из ключевых задач централизованного управления, особенно в средах с препятствиями и ограничениями.

Алгоритм поиска пути A* является одним из самых популярных и эффективных алгоритмов для поиска кратчайшего пути в графах. Он сочетает в себе преимущества алгоритмов поиска в ширину и жадного поиска, что делает его мощным инструментом для решения задач поиска пути. Его ключевая особенность – использование эвристической функции $h(n)$, которая оценивает расстояние от текущего узла до цели. Формула оценки узла:

$$f(n) = g(n) + h(n) \tag{2.20}$$

где $g(n)$ – стоимость пути от старта до узла n,

$h(n)$ – эвристическая оценка

Функция $h(n)$ должна быть оптимистичной, то есть никогда не должна переоценивать реальную стоимость пути. Наиболее распространенной эвристикой является евклидово расстояние или манхэттенское расстояние.

RRT* (Rapidly-exploring Random Tree) строит дерево возможных путей, постепенно расширяя его в случайных направлениях, и оптимизирует траекторию по мере роста дерева. Он особенно полезен в многомерных пространствах, таких как лесные массивы, где требуется обходить стволы деревьев и неровности рельефа.

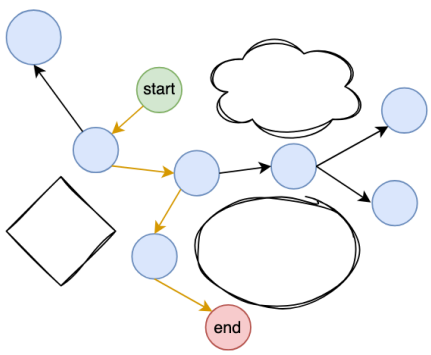

Рисунок 2.7 – Пример работы RRT*

2.2.2 Распределение задач

Распределение задач между группами мобильных объектов требует баланса между эффективностью и ресурсными затратами. Основными подходами к решению таких задач являются генетические алгоритмы и линейное программирование.

Генетические алгоритмы имитируют естественный отбор, генерируя популяцию решений и улучшая их через мутации и скрещивание. В контексте управления мобильными объектами применяются для распределения задач, таких как доставка грузов, где каждое решение (особь) представляет собой набор маршрутов. Тогда функция приспособленности оценивает эффективность по формуле:

$$F = \alpha * \text{Эффективность} - \beta * \text{Затраты} \tag{2.21}$$

где α и β – регулируют приоритеты (например, скорость vs энергопотребление).

По своей сути этапы работы алгоритма схожи с естественным отбором:

1. Генерация начальной популяции маршрутов
2. Оценка каждого маршрута по времени доставки и трудозатратам
3. Отбор лучших решений и создание новых через мутацию (изменение точек пути) и скрещивание (обмен участками маршрутов)

Линейное программирование решает задачи оптимизации с линейными зависимостями и ограничениями. Например, минимизация энергозатрат группы мобильных объектов при выполнении задачи:

$$\begin{cases} \sum_{i=1}^{n} E_i\, x_i \to min, \\ \sum_{i=1}^{n} x_i \geq Q_{\min}, \\ x_i \in \{0,1\}, \end{cases} \tag{2.22}$$

где E_i – энергопотребление объекта i,

x_i – назначение задачи

Подход эффективен для статических сценариев с фиксированным набором условий.

2.3 Децентрализованные алгоритмы

Децентрализованные алгоритмы обеспечивают полную автономию устройств, которые взаимодействуют через локальные протоколы без

централизованного управления. Такой подход основан на самоорганизации, где каждое устройство принимает решения на основе локальных данных и ограниченной информации от соседей. Это делает систему устойчивой к сбоям и способной быстро адаптироваться к изменениям среды, таким как динамические препятствия или потерю связи.

2.3.1 Роевой интеллект

Роевой интеллект (flocking) имитирует поведение природных стай, таких как птицы или рыбы, где каждое существо следует простым правилам, обеспечивающим согласованность группы. Обычно такой подход строится на трех базовых правилах роя:

1. Разделение – избегание столкновений с ближайшими соседями
2. Выравнивание – синхронизация направления и скорости движения
3. Сплочение – поддержание целостности группы (в центре массы группы)

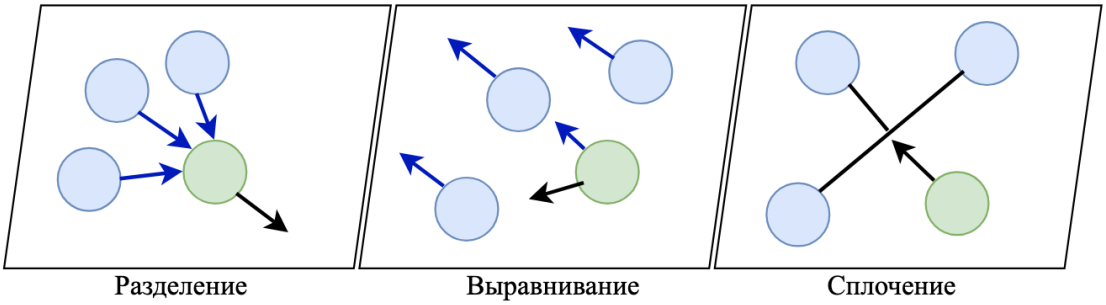

Рисунок 2.8 – Базовые правила роя

2.3.2 Динамический выбор лидера

Даже в децентрализованных системах иногда требуется временное лидерство для координации сложных задач. Динамический выбор лидера позволяет группе гибко реагировать на изменения, назначая наиболее подходящее устройство для управления без жёсткой иерархии. По своей сути лидер берет управление над организацией жизненного цикла группы: распределяет зоны ответственности, агрегирует и анализирует данные от группы. Выборы лидера в группе при необходимости можно проводить по формуле (2.1) в условиях коммуникации каждого члена группы.

2.3.3 Распределенные аукционы задач

Распределённые аукционы решают проблему назначения задач в условиях ограниченных ресурсов. Каждое устройство оценивает свою способность выполнить задачу и участвует в «торгах», что позволяет группе

находить локально оптимальные решения без централизованного планирования.

В базовом варианте устройства рассчитывают ставку по следующему принципу:

$$\mathrm{Bid}_i = \frac{E_{\mathrm{task}}}{B_i} + d_i \tag{2.23}$$

где E_{task} – энергозатраты на задачу,

B_i – текущий заряд батареи,

d_i – расстояние до цели

При этом задача назначается устройству с минимальной ставкой. Этот подход снижает нагрузку на коммуникационные каналы, так как решения принимаются локально.

2.3.4 Перераспределение задач

В условиях роя, отказ одного из устройств не должен приводить к сбою всей системы. Для обеспечения отказоустойчивости обычно вводят модель перераспределения задач: при выходе устройства из строя, его задачи динамически распределяются между остальными членами группы. В простейшем варианте это можно обеспечить с помощью жадного алгоритма:

$$\text{Исполнитель} = \arg\min_j \left(\frac{Q_j}{B_j}\right) \tag{2.24}$$

где Q_j – количество активных задач устройства j,

B_j – текущий заряд батареи

2.4 Мультиагентный подход

Мультиагентный подход рассматривает каждое устройство как автономного интеллектуального агента, способного к самостоятельному принятию решений, обучению и сложным переговорам. В отличие от децентрализованных систем, где устройства следуют жёстким правилам (например, алгоритмам роевого интеллекта), мультиагентные системы обеспечивают более гибкую координацию через формирование коалиций, оценку рисков и адаптацию на основе динамически меняющихся целей. Такой подход требует значительных вычислительных ресурсов, но открывает возможности для решения сверхсложных задач в условиях неопределённости, таких как управление гибкими производственными линиями или космическими миссиями.

Такие системы опираются на алгоритмы, которые позволяют агентам не только взаимодействовать, но и обучаться, прогнозировать последствия действий и формировать временные альянсы.

2.4.1 Механизмы аукционов и коалиций

Аукционы и формирование коалиций – ключевые инструменты распределения задач в мультиагентных системах. Эти механизмы позволяют агентам конкурировать или объединяться для достижения общих целей, учитывая индивидуальные ресурсы и риски.

По аналогии с распределенными аукционами задач, агенты участвуют и предлагают ставки для выполнения задач. Ставку можно рассчитать по формуле:

$$Bid = \frac{1}{\text{Стоимость} + \text{Риск}} \tag{2.25}$$

Тогда алгоритм работы поэтапно выглядит как:

1. Инициатор (агент или система) публикует задачу с заданными параметрами
2. Агенты оценивают свои возможности и риски (например, энергозатраты или вероятность сбоя), фактически происходит расчет ставок
3. Задача назначается агенту или группе с наивысшей ставкой

2.4.2 Модель BDI

Модель BDI формализует процесс принятия решений агентами, опираясь на их убеждения, желания и намерения. Это позволяет устройствам адаптироваться к динамическим изменениям среды, сохраняя фокус на стратегических целях. Такой подход выходит за рамки простой реактивности децентрализованных систем, обеспечивая долгосрочное планирование и анализ последствий действий.

В контексте рассматриваемой задачи можно формализовать BDI модель как: B – текущее состояние среды и собственные возможности, D – стратегические цели (например, завершить миссию с минимальными затратами), I – конкретные действия по достижению целей.

Переходя к математической формализации получаем:

$$\text{Intention} = \arg\max_{a \in A}\big(\text{Utility}(a) \cdot P(\text{ Success} \mid a, \text{Belief})\big) \tag{2.26}$$

где A – множество возможных действий,

$\text{Utility}(a)$ – полезность действия,

$P(\text{ Success} \mid a, \text{Belief})$ – вероятность успеха при текущих убеждениях.

Схема работает, например, так: при обнаружении агентом неисправности (B), он стремится сохранить функциональность (D) и выбирает действие перераспределить задачи на другие доступные узлы (I).

Ключевым достоинством модели BDI является ее способность обеспечивать целенаправленное поведение агентов в сложных, динамичных средах. В отличие от чисто реактивных систем BDI-агенты способны формировать и придерживаться долгосрочных планов (I), ориентированных на достижение стратегических целей (D), даже перед лицом изменяющихся обстоятельств (B). Это достигается за счет механизма намерений, который фиксирует выбранный курс действий, снижая склонность агента к постоянному пересмотру решений при каждом незначительном изменении среды.

Такой подход позволяет агентам эффективно управлять ресурсами (временем, вычислительной мощностью, энергией), фокусируясь на выполнении текущего намерения, а не на непрерывной переоценке всех возможных альтернатив.

Формула выбора намерения (2.26), основанная на максимизации ожидаемой полезности, предоставляет рациональную основу для принятия решений в условиях неопределенности, позволяя агенту взвешивать потенциальную выгоду действия против вероятности его успешного исхода.

Несмотря на свою выразительность, модель BDI сталкивается с рядом практических и концептуальных ограничений. Одной из основных трудностей является вычислительная сложность. Постоянный мониторинг среды и переоценка действий для выбора намерения может быть ресурсоемкой.

Другая проблема связана с конфликтами целей: желания (D) агента могут быть противоречивыми, а базовая модель не предоставляет явных механизмов для их разрешения. Также существует проблема задания функции полезности и точная оценка вероятности успеха в сложных средах, что часто является нетривиальной задачей.

2.5 Выводы

Вторая глава сосредоточена на разработке теоретических основ управления группами автономных устройств. Проведен обзор современных методов управления, включая централизованные, децентрализованные и мультиагентные подходы, а также изучены модели взаимодействия. На основе этого анализа разработаны ключевые модели: жизненного цикла системы, сцены миссии, автономного устройства, негативного воздействия и коммуникаций. Кроме того, предложены варианты реализации и критерии выбора оптимальных подходов к управлению.

ГЛАВА 3
РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1 Архитектура целевой системы

3.1.1 Подход к проектированию

В современном мире все большей популярностью обрастают веб-приложения, позволяющие удобно и всецело собрать весь функционал системы в браузере (а соответственно и в мобильном устройстве). Рассмотрим подробнее преимущества такого подхода:

1. Наличие веб-интерфейса позволяет администраторам системы осуществлять доступ и мониторинг через браузер любого устройства без установки специализированного ПО
2. Веб-приложение легко интегрируется с IoT-устройствами через стандартные протоколы (HTTP, MQTT и другие)
3. Веб-приложение легко можно развернуть в облачной инфраструктуре
4. Централизованные обновления появляются на сервере, пользователь автоматически получает актуальную версию
5. Возможность реализации механизмов защиты, таких как защита от CSRF, SQL-инъекций и XSS-атак
6. Простота горизонтальной масштабируемости и применения микросервисной архитектуры

Исходя из этого, целевая система управления группами мобильных объектов будет проектироваться как гибкое веб-приложение, способное базово интегрировать централизованные, децентрализованные и мультиагентные подходы.

3.1.2 Выбор стека технологий

Большинство современных веб-приложений написаны на языке Python, так как он довольно прост в изучении и при этом обладает огромным функционалом. В контексте управления группами мобильных объектов он еще хорош тем, что имеет широкую экосистему библиотек для этих целей, а также довольно дешево интегрируется с IoT-протоколами и поддерживает асинхронность. Альтернативами являются C++ или Go (особенно для критичных по производительности сервисов), но их интеграция усложняет общий стек и замедляет развитие приложения.

Для реализации функционала веб-приложения выбран фреймворк Django по следующим причинам:

1. Быстрая разработка за счет встроенных компонентов (есть ORM, аутентификация и админ-панель)

2. Система аутентификации и прав доступа работает из коробки (и есть ролевая модель)

3. Масштабируемость через микросервисную архитектуру (возможность разделения модулей)

4. Уже внедрена защита от CSRF, XSS и SQL-инъекций

5. Активное сообщество и зрелая экосистема плагинов

Для хранения данных стандартным подходом будет выбор Postgres. Дополнительно, при необходимости, можно добавить поддержку PostGIS для работы с географическими объектами (маршрутами), однако пока хватит и чистой базы.

3.1.3 Проект архитектуры

Архитектурно систему можно представить как многоуровневую со следующим разделением:

1. На клиентском уровне происходит визуализация компонентов системы и первоначальная обработка пользовательских запросов, реализацией в нашем случае будет веб-интерфейс

2. На серверном уровне происходит основная работа фреймворка Django: ядро системы реализует основную логику приложения из главы 2

3. Уровень данных (реализуемый базой данных) отвечает за хранение данных устройств, миссий и логов

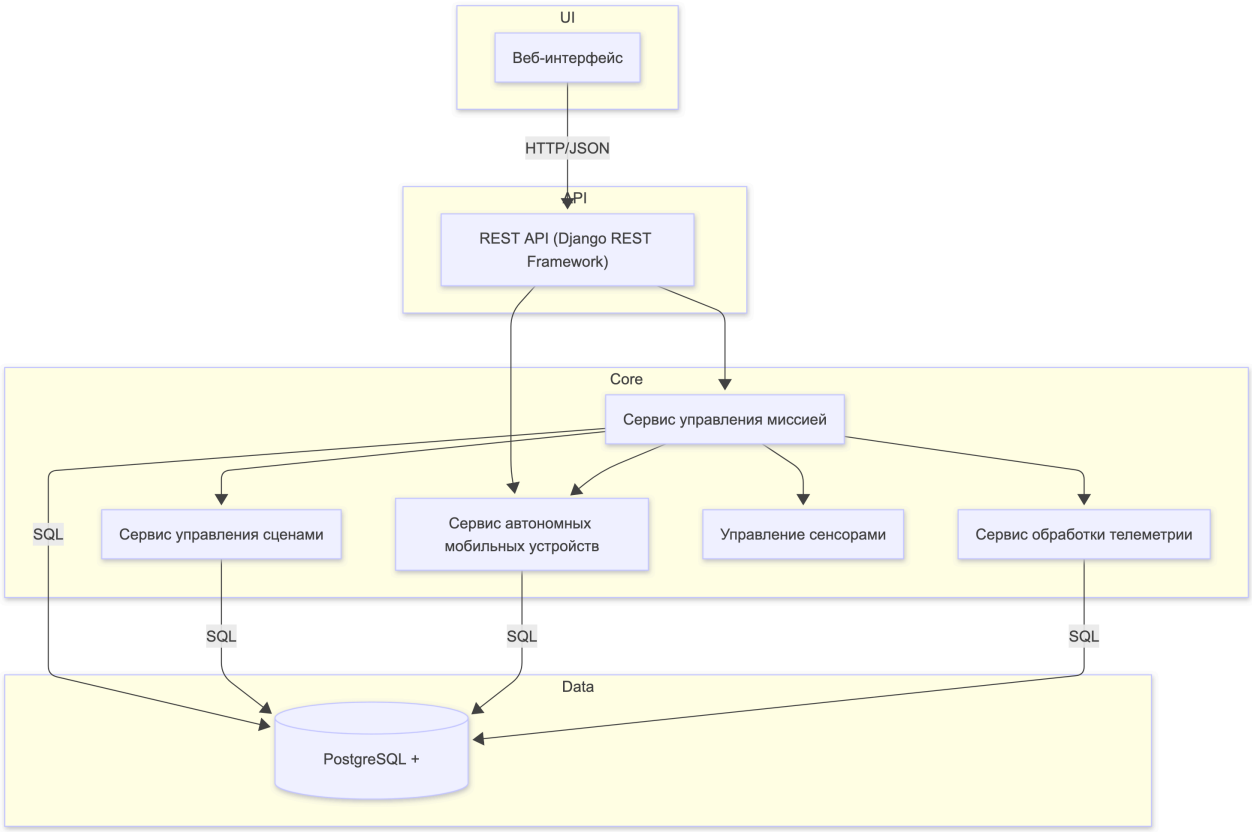

Рисунок 3.1 – Архитектура веб-приложения

3.2 Разработка системы имитационного моделирования

Система имитационного моделирования является ключевым инструментом для анализа и оптимизации процессов управления группами автономных мобильных устройств.

3.2.1 Функциональные требования

Опишем подробнее выполнение каких функций должна обеспечивать система имитационного моделирования:

1. Для управления устройствами система должна поддерживать регистрацию с уникальными идентификаторами и техническими характеристиками (тип, сенсоры, батарея), а также автоматическое обновление статуса устройств на основе данных телеметрии
2. Система должна уметь распределять задачи, назначать задачи устройствам в режиме реального времени с учётом их текущего состояния
3. Поддержка алгоритмов аукционов, роевого интеллекта (flocking) и BDI-модели для адаптивного перераспределения задач
4. Обеспечивать возможность создания и настройки миссий через модели сцен, моделировать жизненный цикл миссии
5. Поддерживать протоколы связи для обмена данных с устройствами через MQTT, HTTP
6. Агрегировать показатели сенсоров, сохранять данные в Postgres
7. Реализовывать ролевую модель доступа к ресурсам
8. Иметь веб-интерфейс для управления основными сущностями системы

Требования можно визуализировать с использованием диаграмм вариантов использования системы. Основные элементы такой диаграммы – участник и прецедент.

Участник – это множество логически связанных ролей, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема или класс).

Прецедент – описание множества последовательных событий (включая варианты), выполняемых системой, которые приводят к наблюдаемому участником результату. Прецедент представляет поведение сущности, описывая взаимодействие между участниками и системой. Прецедент не показывает, как достигается некоторый результат, а только что именно выполняется.

Прецеденты обозначаются очень простым образом – в виде эллипса, внутри которого указано его название.

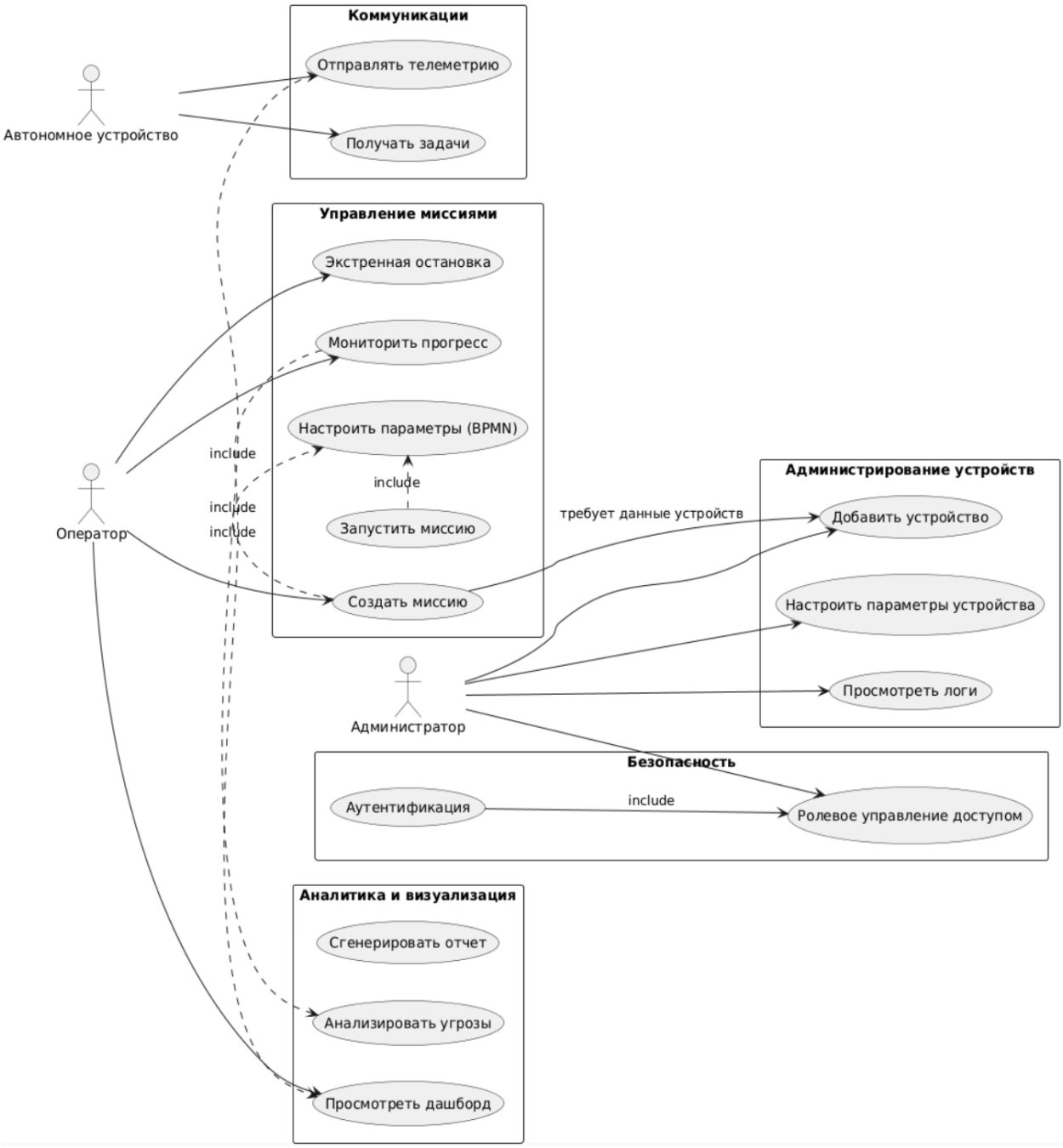

Рисунок 3.2 – Диаграмма вариантов использования

3.2.2 Диаграммы классов

В рамках проектирования сложных систем управления важным этапом является формализация структуры данных и взаимодействий между компонентами. Диаграммы классов, как элемент унифицированного языка моделирования (UML), предоставляют наглядный инструмент для описания сущностей системы, их атрибутов, методов и взаимосвязей. Воспользуемся диаграммами классов для стандартизации представления ключевых элементов системы.

Диаграмма классов модели автономного устройства визуализирует, описанную в предыдущих главах, иерархию сущностей: от базовых

параметров (идентификатор, тип) до динамических состояний (режим работы, батарея), а также предоставляет базовое понимание о взаимодействии компонентов (например, как данные датчиков влияют на изменение статуса устройства и т.д.)

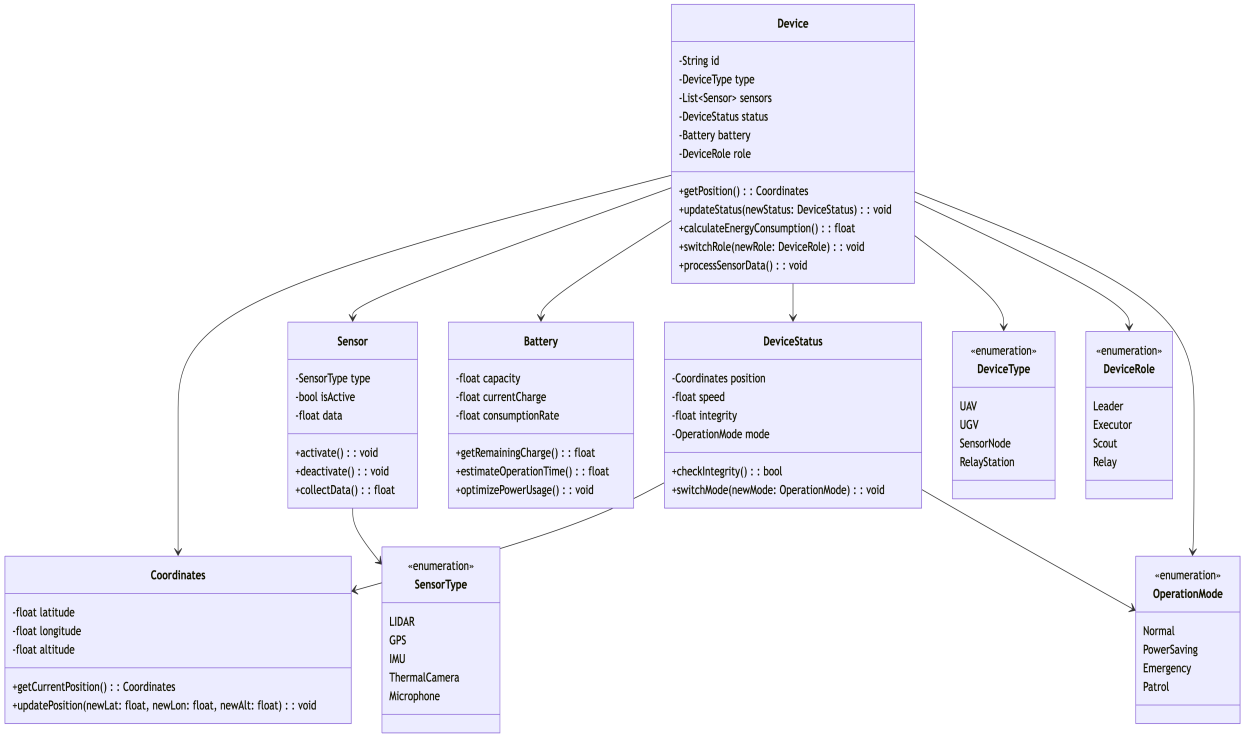

Рисунок 3.3 – Диаграмма классов модели автономного устройства

Содержательная часть кода реализации представлена в приложении А.

Диаграмма классов модели негативного воздействия позволяет стандартизировать описание угроз, а также унифицировать параметры воздействий (тип, интенсивность, длительность) для всех компонентов системы. Также визуализирована связь с математическими формулами. Дополнительно модель Hazard инкапсулирует логику расчета текущего уровня угрозы в зависимости от времени, обеспечивая совместимость как с простыми (постоянная интенсивность), так и с продвинутыми (нарастающие/затухающие) сценариями.

Содержательная часть реализация представлена в приложении Б.

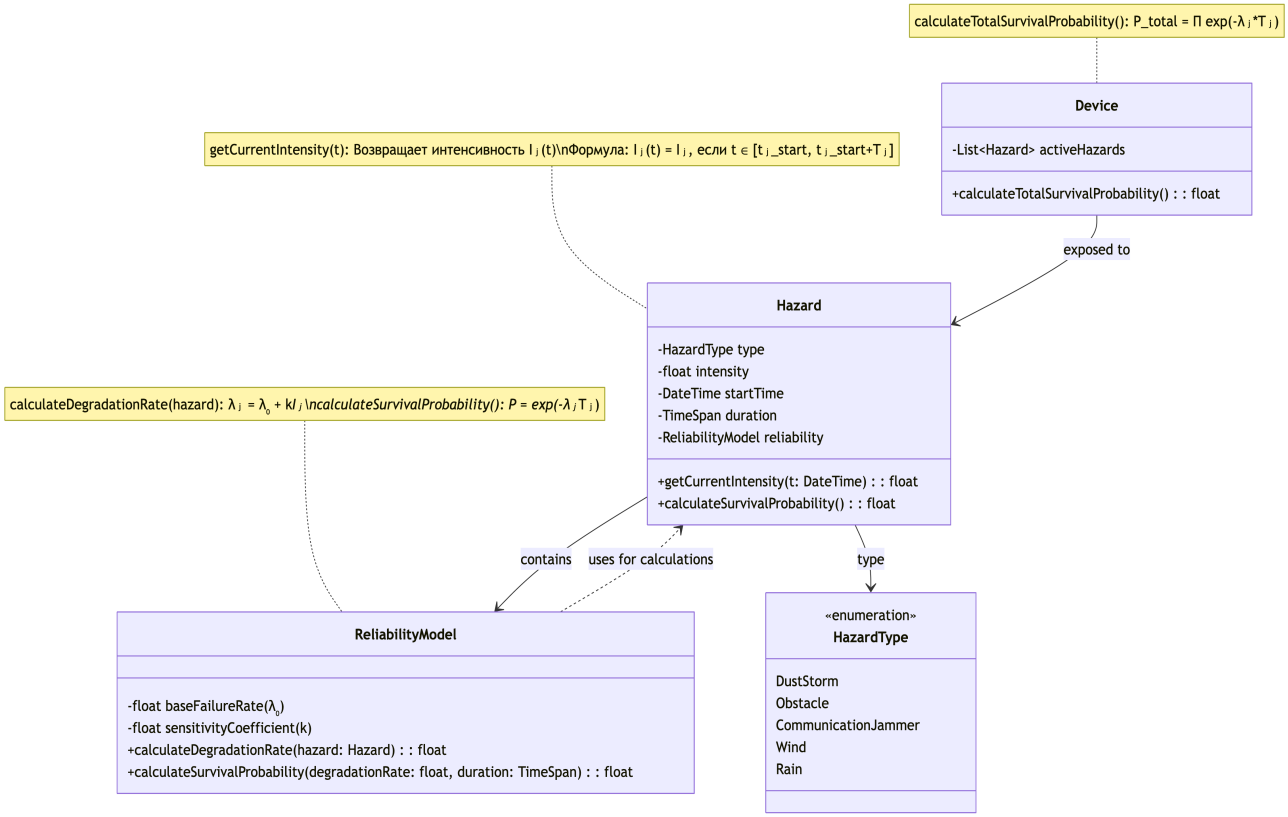

Рисунок 3.4 – Диаграмма классов модели негативного воздействия

Аналогично, по главе 2 строится диаграмма модели коммуникаций

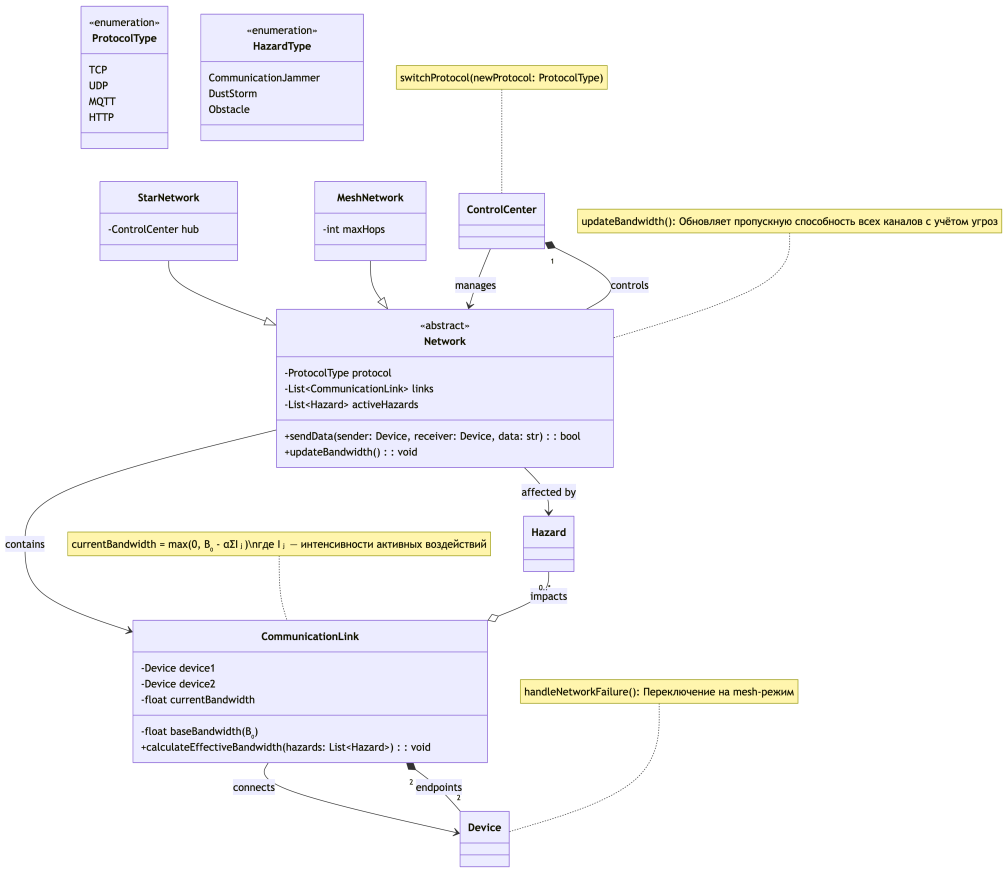

Рисунок 3.5 – Диаграмма классов модели коммуникаций

Диаграмма центра управления завершает архитектурную триаду системы, связывая ранее описанные сущности — автономные устройства, негативные воздействия и коммуникационные сети — в единый адаптивный контур управления. Она демонстрирует, как центр: координирует устройства через гибкую коммуникационную модель (звезда/mesh), выбирая оптимальные протоколы (MQTT, HTTP) и топологии; интегрирует данные о негативных воздействиях для динамического расчета пропускной способности каналов и адаптирует стратегии в зависимости от режима контроля, делегируя задачи сублидерам или устройствам. Содержательная часть реализации модели центра управления представлена в приложении В.

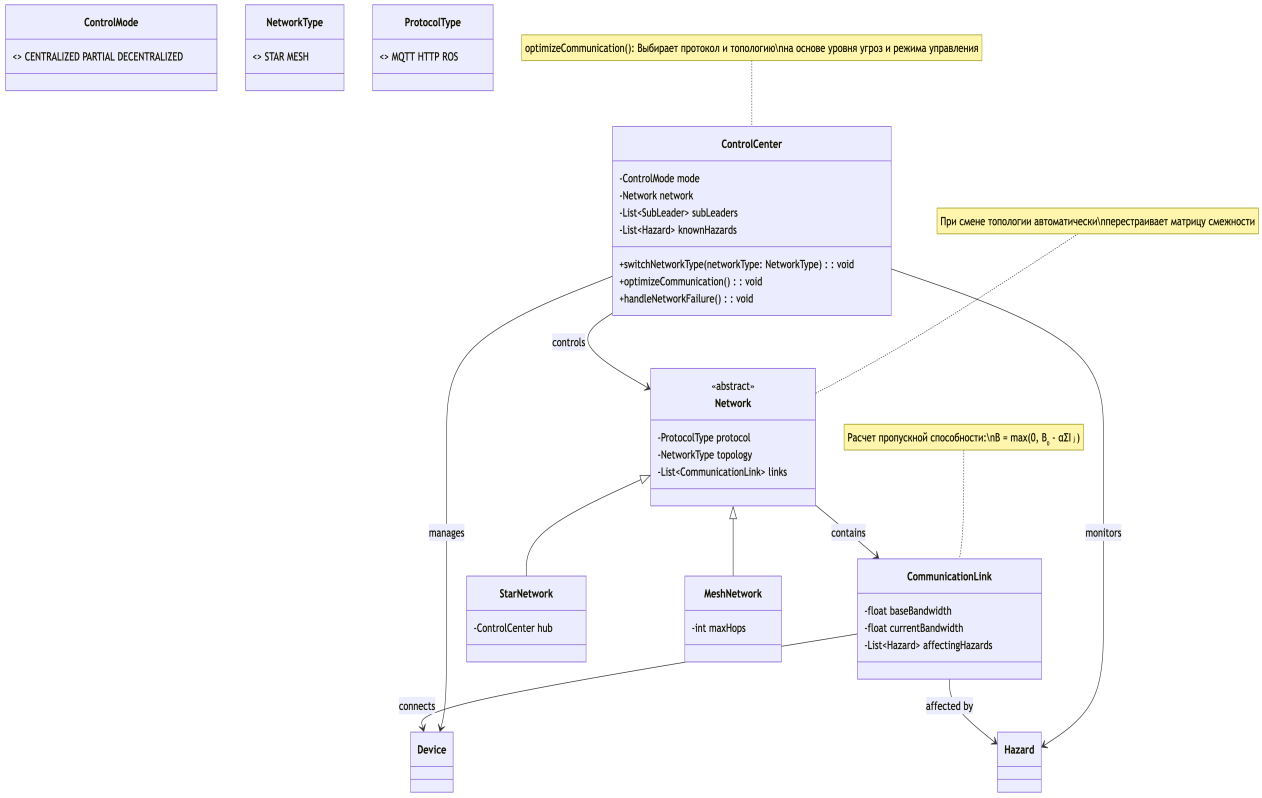

Рисунок 3.6 – Диаграмма классов модели центра управления

3.2.3 Интерфейс администрирования

Встроенная в Django административная панель служит центральным инструментом для управления ключевыми сущностями системы.

Администратор через удобный веб-интерфейс может создавать, редактировать и удалять записи о негативных воздействиях, задавая их типы, интенсивности и временные параметры.

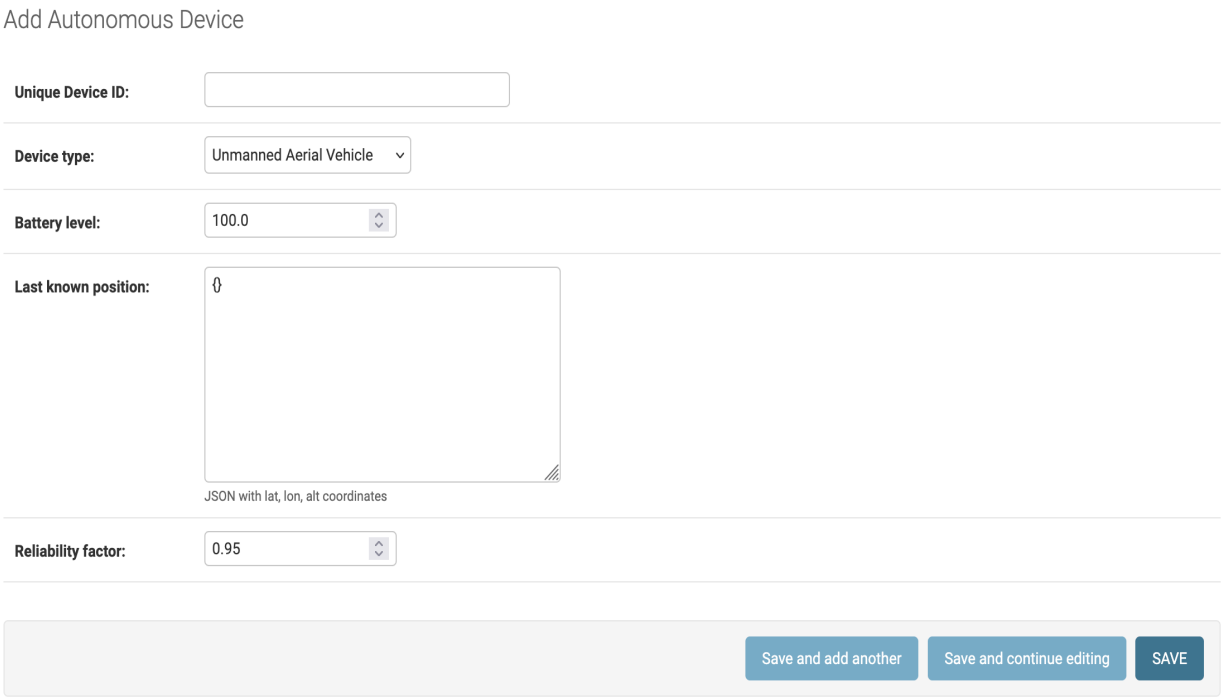

Рисунок 3.7 – Форма добавления нового автономного устройства

Аналогичным образом осуществляется управление списком устройств: добавление новых девайсов, настройка их характеристик и мониторинг текущего статуса. Для миссий предусмотрен отдельный раздел, где задаются сценарии, маршруты и подходы к координации группы агентов.

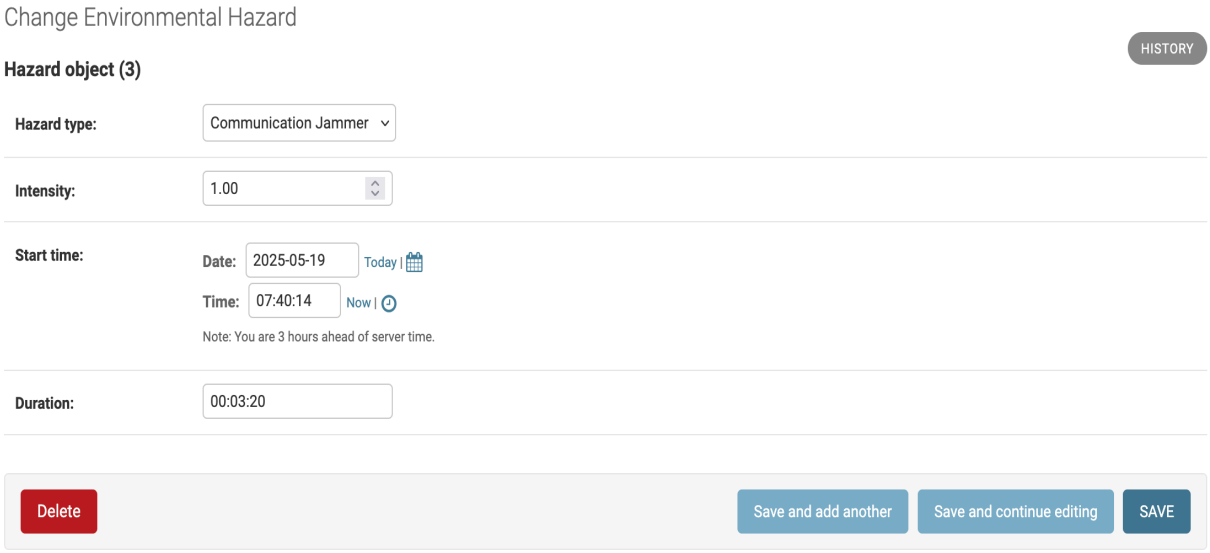

Рисунок 3.8 – Форма редактирования информации о негативных воздействиях

Также доступен просмотр и фильтрация системных логов: админка поддерживает удобный поиск по событиям и уровням логирования, что облегчает диагностику и аудит.

Благодаря гибкой системе прав доступа, разные роли пользователей (операторы, разработчики) могут работать с теми разделами, которые соответствуют их задачам. Такая организованная структура административного интерфейса позволяет быстро реагировать на изменения условий миссии и поддерживать консистентность данных во всей системе.

3.2.4 Управление группой автономных мобильных устройств

Система в автоматическом режиме в зависимости от запущенного варианта управления осуществляет контроль за группой. В централизованном режиме все решения о перераспределении маршрутов и ролей принимаются головным узлом, который непрерывно отслеживает статус каждого устройства.

В децентрализованной архитектуре каждый агент самостоятельно корректирует своё поведение на основе локальной телеметрии и соседских сообщений, обеспечивая высокую устойчивость к отказам. Наконец, в мультиагентной модели управленческие функции распределяются через аукционные и консенсусные протоколы, что позволяет группе адаптироваться к изменяющимся условиям среды без единой точки отказа.

Визуализацию централизованного процесса управления проще всего сделать в BPMN нотации:

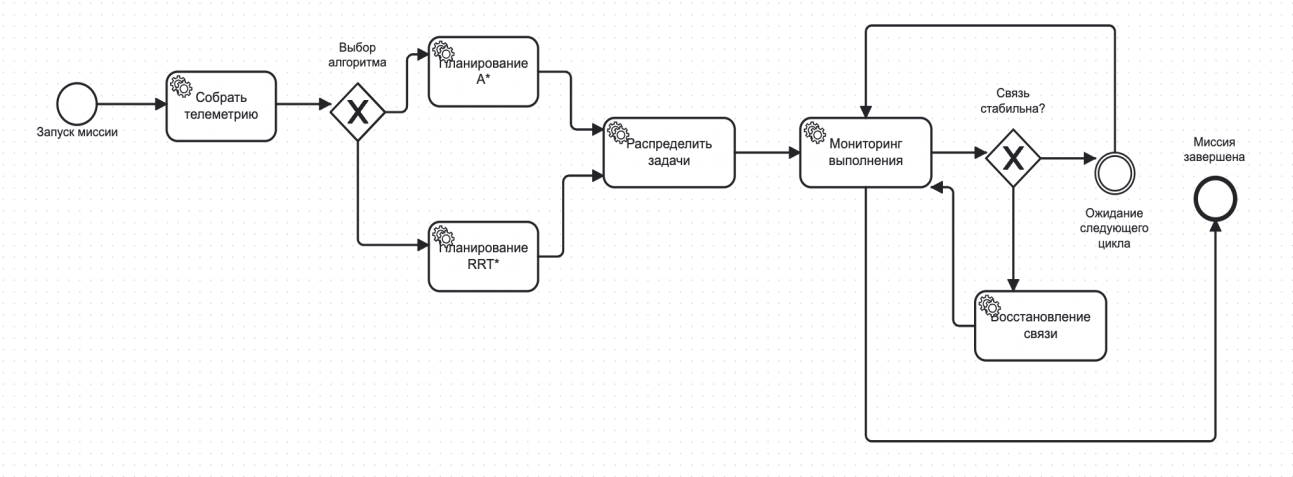

Рисунок 3.9 – Централизованный процесс управления

В простейшем варианте работу системы можно увидеть на обходе препятствий группой из двух мобильных объектов на пути к финальной точке. На примере экрана представлено прогнозирование маршрутов группы из двух объектов при обходе группового препятствия. При поиске маршрута применяется алгоритм A*

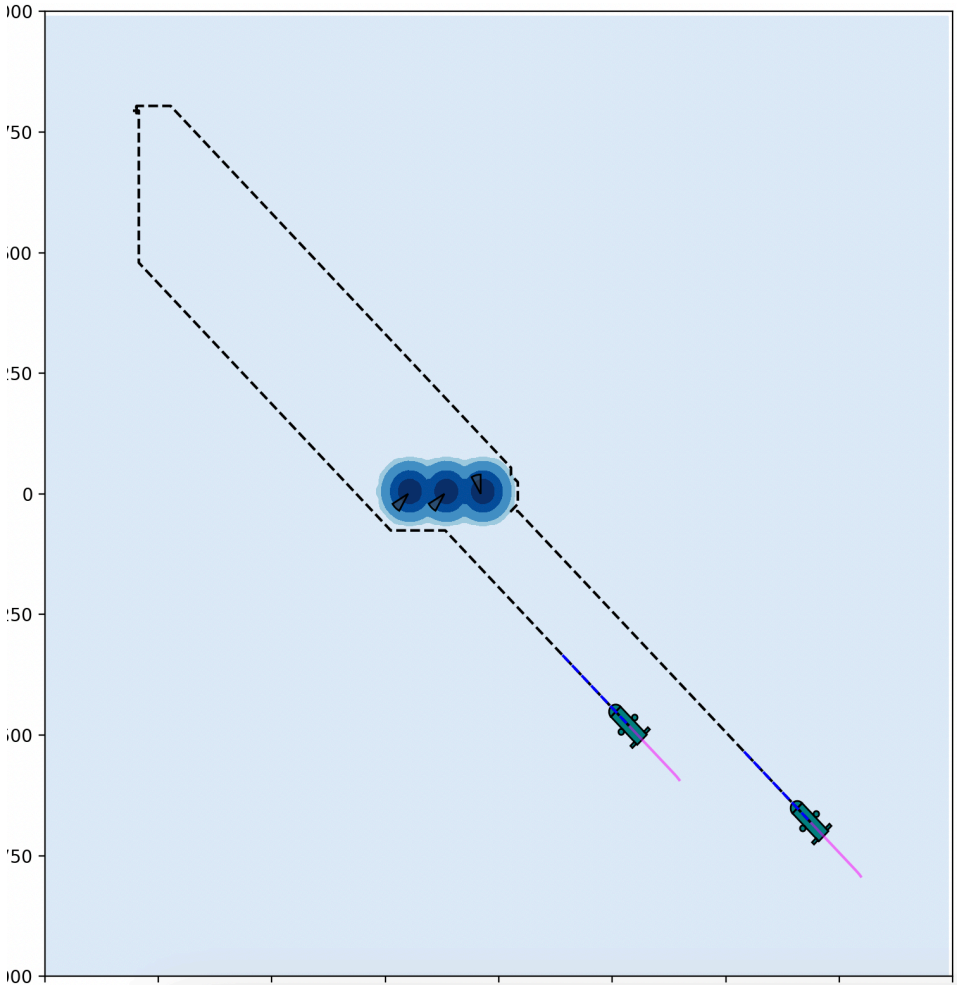

Рисунок 3.10 – Простейший обход негативного воздействия

В работе с картами местности на основе изображений удобен модуль OpenCV (cv2), позволяющий быстро загружать растровые данные вызовом cv2.imread() и переводить их в нужное цветовое пространство с помощью cv2.cvtColor().

Для выделения препятствий конкретного цвета чаще всего применяют цветовую сегментацию через cv2.inRange(), задавая верхнюю и нижнюю границы BGR/HSV-диапазона интересующего оттенка. Полученную маску можно преобразовать в чёрно-белое изображение, где белым отмечены потенциальные зоны препятствий, используя бинаризацию cv2.threshold() DataFlair.

Итоговая бинарная карта пригодна для дальнейших шагов — поиска контуров (cv2.findContours()) или построения графа проходимости learnopencv.com. Такой подход позволяет легко адаптировать обработку к различным типам карт (лес, город) и динамически менять цветовые пороги под изменяющееся освещение.

Для демонстрации пример экрана системы с переходом к такой карте и выполнением задачи группой из трех мобильных автономных устройств:

Рисунок 3.11 – cv2 карта с выделенными препятствиями

Кроме этого, в системе можно эмулировать работу в условиях ограниченной дальности сенсоров: на карте заранее расположены все препятствия, но мобильное устройство видит их только в тот момент, когда они попадают в радиус его датчиков. При этом планировщик строит траекторию сразу от старта до финиша по кратчайшему пути, не учитывая скрытые до момента обнаружения помехи объекты. В процессе движения устройство движется по этой оптимальной линии, но при появлении новой информации о препятствии в зоне досягаемости он мгновенно пересчитывает локальный участок маршрута, обходя непроходимые зоны. Такой подход позволяет оценить, как система реагирует на внезапно возникающие угрозы и корректирует свой курс на лету, сохраняя приоритет минимальной длины пути.

Благодаря этому модель учитывает реальное ограничение сенсорного обзора и демонстрирует, насколько эффективно алгоритм удерживает баланс между прямолинейностью движения и безопасным обходом препятствий.

Далее будет представлен экранный пример, на котором видно исходную кратчайшую траекторию, зону обнаружения первого препятствия и результат её динамической корректировки.

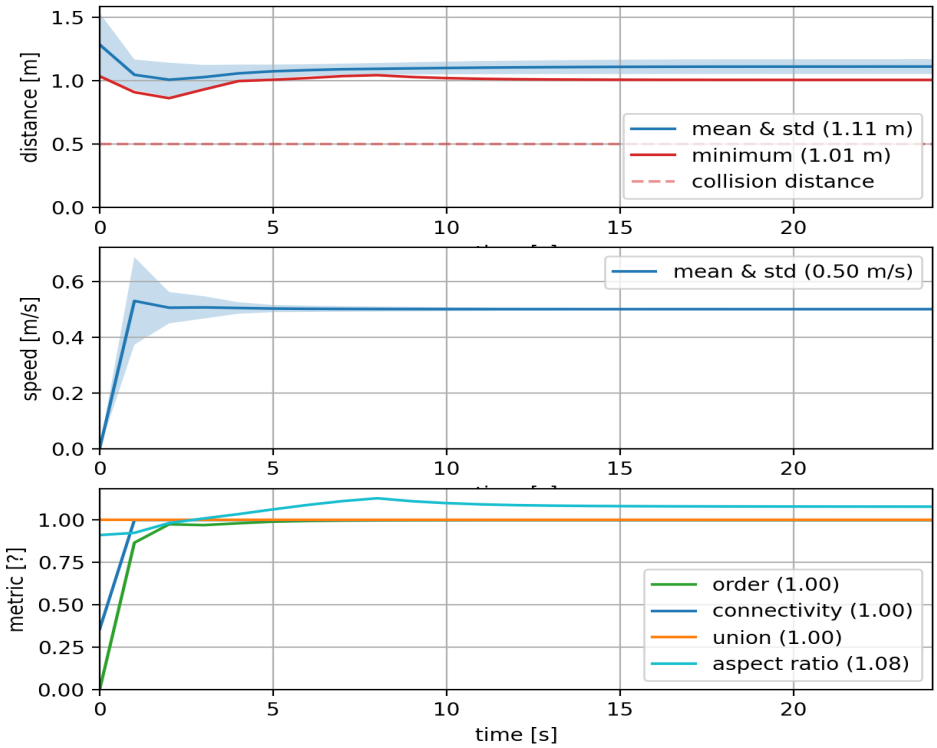

Рисунок 3.12 – Данные с датчиков устройств

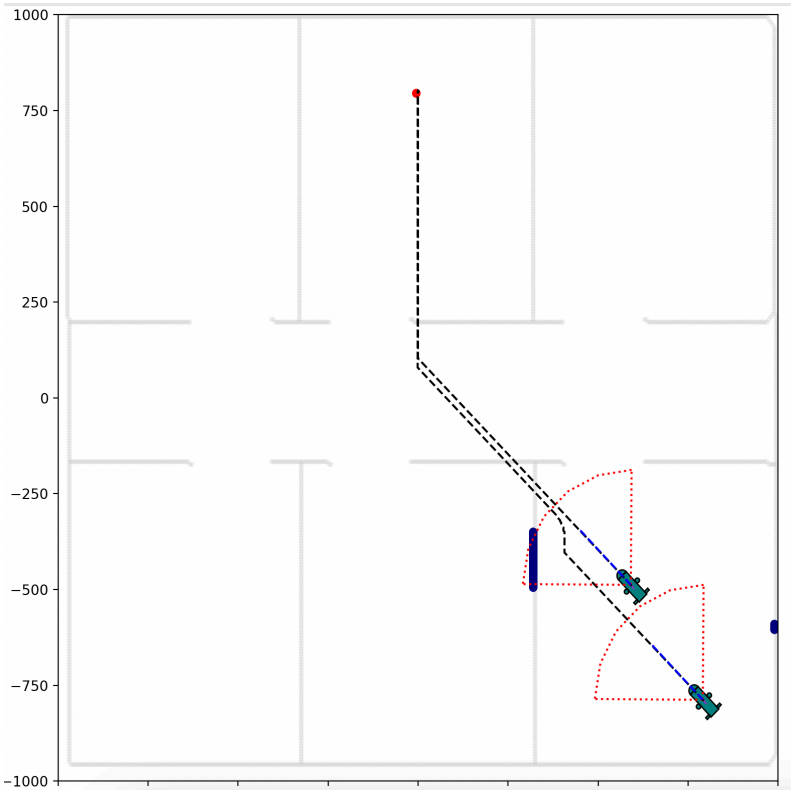

Рисунок 3.13 – Работа в условиях ограниченности сенсоров

3.3 Методика применения комплекса

Использование созданного комплекса начинается с четкой формулировки входных данных: описания ситуации миссии, свойств устройств и характеристик потенциальных угроз.

Вначале в административном интерфейсе конфигурируются сущности: задаются пространственные и временные рамки ситуации, регистрируются имеющиеся устройства с их датчиками и энергетическими ресурсами, а также классифицируются типы и модели потенциальных негативных воздействий. Это обеспечивает централизованный и единоразовый ввод всей исходной информации, что гарантирует её согласованность при последующих расчетах.

Система самостоятельно создает группу агентов, подбирая оптимальное сочетание устройств, исходя из типа выполняемой миссии и текущей ситуации с ресурсами. В централизованном режиме вся информация передается в центр управления, где алгоритмы планирования маршрутов (A*, RRT*) и распределения задач (такие как генетический алгоритм или линейное программирование) совместно разрабатывают единый глобальный план. В децентрализованной системе каждый участник запускает собственные локальные механизмы согласования или аукционы для взаимодействия, что особенно актуально в условиях слабой связи. Мультиагентный подход объединяет эти два метода, предоставляя возможность формировать временные союзы и осуществлять BDI-циклы на уровне отдельных автономных агентов.

В процессе выполнения миссии комплекс постоянно отслеживает телеметрию, обрабатывает данные сенсоров и оценивает состояние связи. При возникновении новых угроз или изменениях условий (например, ограничение обзора или динамика помех), система автоматически корректирует траектории. Централизованный режим предполагает пересчет плана в центральном узле, децентрализованный – локальную адаптацию маршрута каждым агентом, а мультиагентный режим активирует процедуры аукциона и пересмотра ставок для оптимизации действий всех участников. Этот подход обеспечивает гармоничное сочетание оптимальности и отказоустойчивости, что позволяет группе эффективно преодолевать препятствия и не снижать темп выполнения задания.

После завершения миссии оператор может воспользоваться Django интерфейсом для анализа логирования системы, оценки эффективности использованных стратегий и, при необходимости, корректировки параметров комплекта для будущих запусков.

BPMN-нотация и визуализация маршрутов на черно-белых картах OpenCV упрощают анализ прохождения сценариев и выявление проблемных

зон, что ускоряет процесс совершенствования алгоритмов и повышает надежность всей системы в целом.

3.4 Выводы

Во третьей главе получены следующие основные результаты:

1. проанализирована литература по методам управления группами автономных устройств, включая централизованные, децентрализованные и мультиагентные подходы, а также современные модели взаимодействия
2. обоснована необходимость использования веб-приложения в качестве архитектурной основы системы управления
3. выбраны технологии реализации: язык Python и фреймворк Django, обоснован выбор СУБД PostgreSQL
4. разработана многоуровневая архитектура системы, включающая клиентский, серверный и уровень хранения данных
5. сформулированы функциональные требования к системе имитационного моделирования
6. построены UML-диаграммы классов, описывающие структуру системы и взаимодействие между компонентами
7. реализован административный интерфейс для управления ключевыми сущностями системы
8. описаны механизмы централизованного, децентрализованного и мультиагентного управления группой автономных устройств;
9. реализована визуализация миссий и препятствий с применением инструментов OpenCV;
10. предложена методика применения комплекса, включающая этапы конфигурации, выполнения миссии и последующего анализа

ГЛАВА 4
РЕШЕНИЕ ПРИКЛАДНОЙ ЗАДАЧИ

4.1 Постановка задачи

Лесные пожары представляют серьезную угрозу для экосистем, имущества и человеческих жизней, особенно в труднодоступных районах. Традиционные методы тушения пожаров часто ограничены из-за сложности координации и скорости реагирования. Группы мобильных автономных объектов предлагают решение, обеспечивая раннее обнаружение пожаров, мониторинг в реальном времени и возможность доставки огнетушащих веществ [9].

Дано: на базе находится 120 автономных устройств, оснащенных баками для воды. Спутник обнаружил два пожара на расстоянии 60 км от базы, и внутренняя платформа ИИ оценила потребность в огнетушащих веществах: 600 литров для первого пожара и 700 литров для второго. При полете к пожарам возможны негативные воздействия, такие как песчаные бури, вызывающие сбои устройств. Также необходимо учитывать заранее известные препятствия, например, горные объекты.

Требуется разработать программу имитации тушения пожаров, демонстрирующую возможности разработанной технологии управления группами автономных устройств в условиях неопределенности, включая координацию гетерогенных устройств и минимизацию влияния внешних факторов.

Цели задачи:

1. Координировать доставку воды к двум пожарам для их тушения.
2. Учитывать время на перемещение устройств (60 км, предполагаемое время 12 временных шагов в одну сторону).
3. Моделировать случайные сбои устройств из-за песчаных бурь
4. Обеспечить устойчивость системы к динамическим изменениям и препятствиям.

С помощью данной задачи можно продемонстрировать возможности разработанной технологии управления группами автономных устройств в условиях неопределенности, включая координацию гетерогенных устройств и минимизацию влияния внешних факторов.

4.2 Решение задачи

Для решения задачи используется разработанная в дипломе система моделирования, адаптированная к сценарию управления лесными пожарами.

Этап 1 – Настройка сцены

На начальном этапе определяется природный массив с учетом характеристик растительности, погодных условий, таких как температура, влажность и скорость ветра, а также зон потенциальных песчаных бурь и горных препятствий. Моделируются два пожара с заданными потребностями в воде, а также динамические изменения, такие как смена направления ветра или возникновение новых очагов.

Этап 2 – Конфигурация группы мобильных автономных объектов

На втором этапе формируется рой из 120 гетерогенных дронов, оснащенных тепловыми камерами, GPS, датчиками окружающей среды и баками для воды емкостью 10 литров. Дроны распределяются поровну между двумя пожарами, при этом учитываются ограничения по емкости батарей и грузоподъемности, что требует оптимизации распределения задач.

Важным признаком являются характеристики дронов, так как ограниченная емкость батареи и грузоподъемность может негативно повлиять на исход выполнения задачи.

Этап 3 – Распределение задач и выполнение

Алгоритмы многоагентного управления распределяют дроны на группы: одни патрулируют для обнаружения пожаров, другие доставляют воду. При обнаружении пожара дроны с тепловыми камерами создают карту его периметра, определяя приоритетные зоны.

Дроны с водой направляются к этим зонам, используя алгоритмы оптимизации маршрутов, учитывающие препятствия. Песчаные бури моделируются как случайные сбои с некоторой вероятностью за временной шаг и заменой сбойных устройств (или их восстановлением)

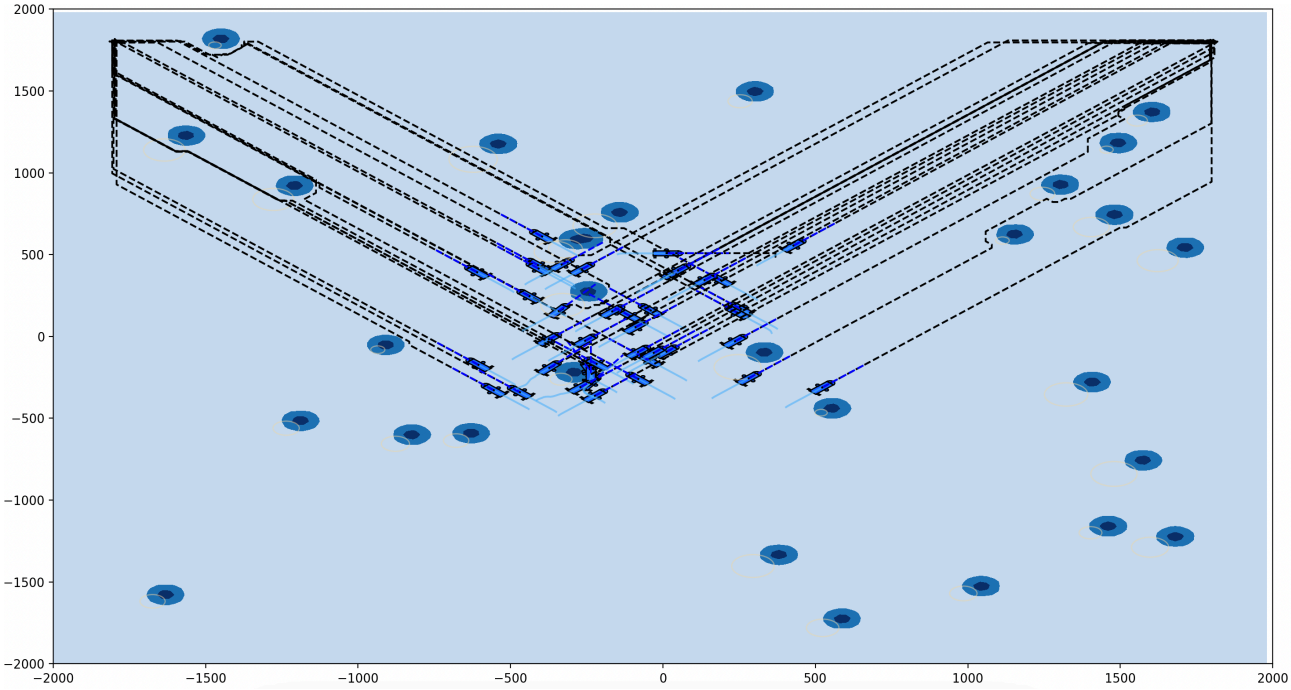

Рисунок 4.1 – Траектории движения к пожарам с корректировкой курса

Этап 4 – Адаптация и устойчивость

Программный комплекс непрерывно отслеживает состояние дронов и окружающей среды, перераспределяя задачи при сбоях. Динамическая корректировка маршрутов и задач позволяет рою адаптироваться к изменениям.

По логам системы с использованием специализированных библиотек на Python, например Seaborn, можно построить графики для аналитики процесса имитационного моделирования тушения пожаров в условиях негативных воздействий.

Библиотека предоставляет высокоуровневый интерфейс для рисования привлекательных и информативных статистических графиков. Seaborn позволяет легко создавать сложные визуализации с минимальными усилиями, что делает его отличным инструментом для анализа данных.

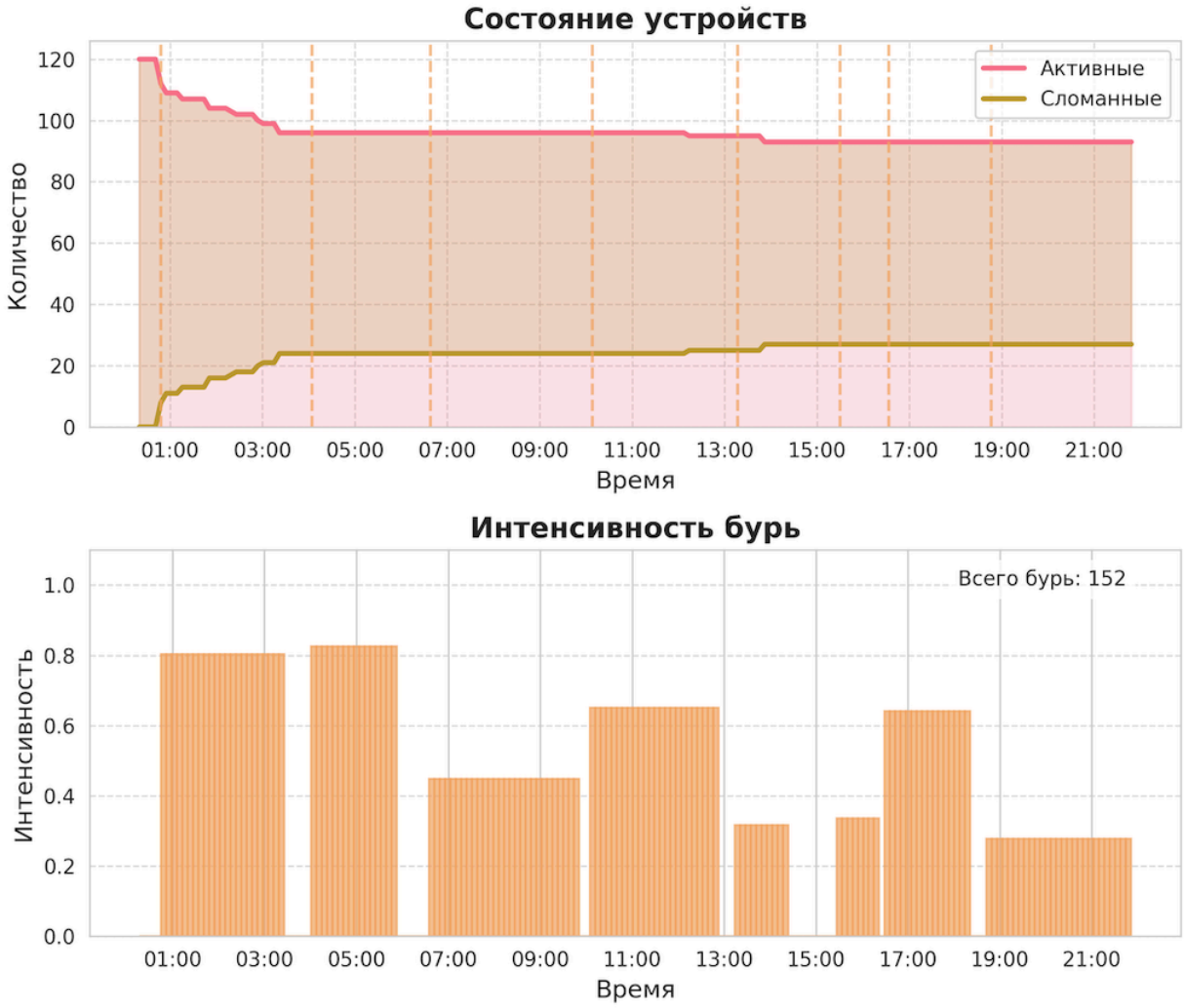

Рисунок 4.2 – График состояния устройств и интенсивности бурь

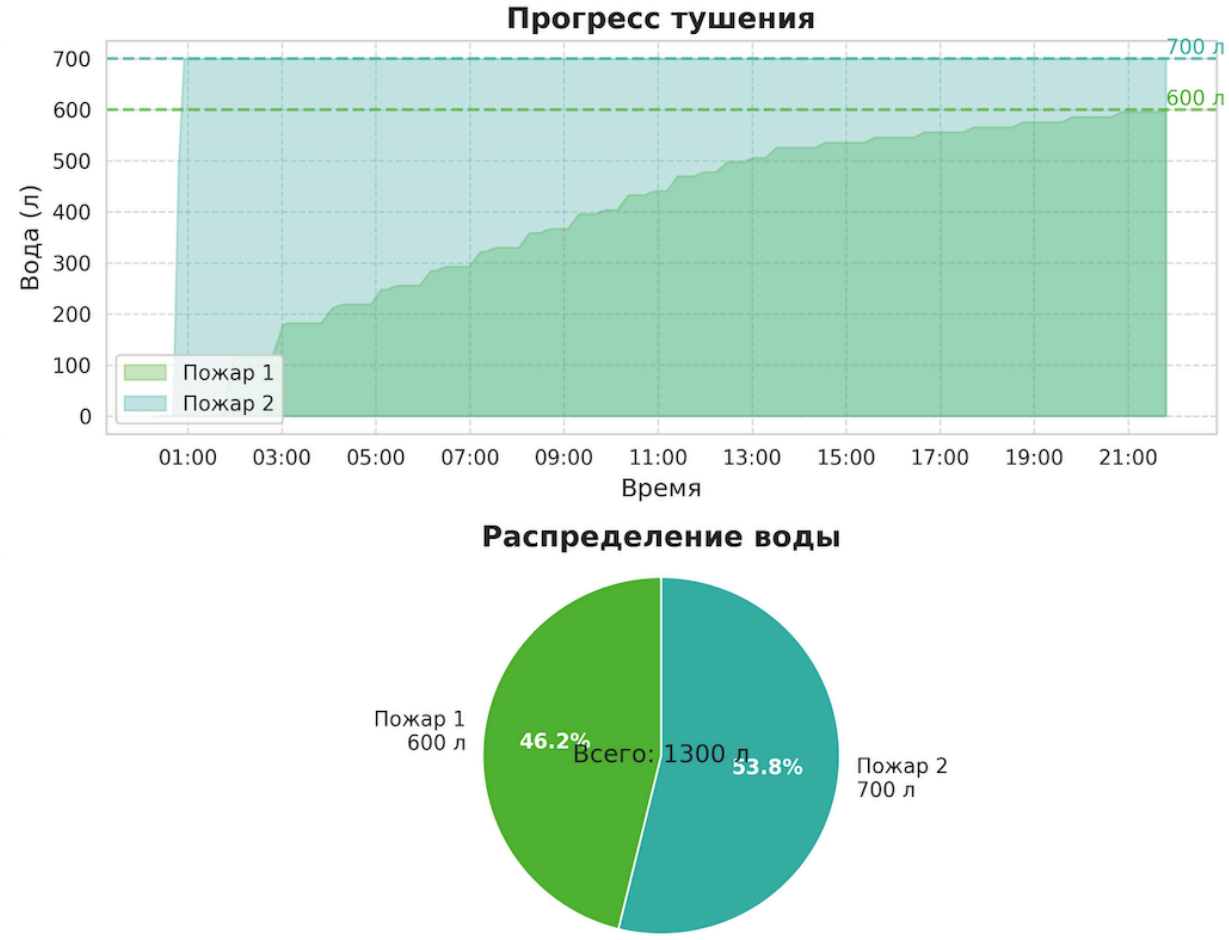

Рисунок 4.3 – Прогресс тушения и распределение воды

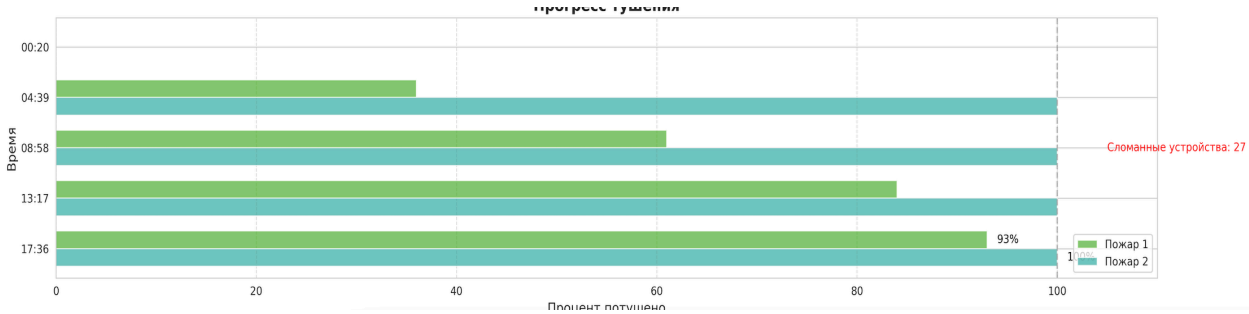

Рисунок 4.4 – Процент тушения

4.3 Выводы

Имитация показывает, что группа мобильных автономных объектов успешно доставляет воду к обоим пожарам, несмотря на сбои. Количество работающих устройств колеблется из-за песчаных бурь, но система сохраняет устойчивость благодаря замене.

Пожары тушатся с учетом количества потерянных во время негативных воздействий устройств. График визуализирует эти динамики, подтверждая способность технологии координировать действия и адаптироваться.

Моделирование демонстрирует следующие результаты:

1. Сокращение времени реагирования: группы мобильных автономных объектов позволяют обнаружить пожар в течение нескольких минут, что значительно быстрее традиционных методов.
2. Эффективность тушения: оптимизированное распределение ресурсов позволяет минимизировать ущерб от пожара.
3. Устойчивость: система сохраняет работоспособность даже при выходе из строя отдельных устройств или изменении условий.

Таким образом, прикладная задача демонстрирует эффективность разработанной технологии в реальном сценарии управления тушением пожаров. Группа мобильных автономных объектов обеспечивает быстрое обнаружение, точный мониторинг и эффективное тушение пожаров, что может значительно снизить ущерб и повысить безопасность операций. Технология подтверждает свою универсальность и применимость в сложных условиях, открывая перспективы для использования в других областях, таких как спасательные операции и сельское хозяйство.

Разработанная система имитационного подтверждает высокую практическую применимость для задач мониторинга и тушения лесных пожаров в оперативной деятельности МЧС, обеспечивая критическое сокращение времени реагирования, точное картирование очагов возгорания и оптимизированное распределение ресурсов при минимальном риске для персонала.

Технология демонстрирует устойчивость к экстремальным условиям, что особенно актуально для труднодоступных регионов. Внедрение системы позволит МЧС создать интегрированный инструмент для круглосуточного мониторинга, раннего обнаружения и оперативного подавления пожаров, значительно снижая экономический и экологический ущерб от стихийных бедствий при масштабируемости решения на другие задачи гражданской обороны.

Дополнительно система имитационного моделирования демонстрирует потенциал для применения в различных сферах: от мониторинга сельскохозяйственных угодий до поисково-спасательных операций в труднодоступных регионах. Она обеспечивает эффективное управление гетерогенными устройствами в условиях неопределенности, минимизируя влияние внешних факторов и максимизируя результативность операции.

ЗАКЛЮЧЕНИЕ

Данная дипломная работа посвящена решению актуальных проблем управления группами автономных мобильных устройств в условиях динамической и неопределённой среды. Исследование охватывает теоретический анализ, разработку моделей, создание программного комплекса и его апробацию на практических задачах, демонстрируя системный подход к повышению эффективности координации разнородных систем.

В рамках теоретического анализа выявлены ключевые вызовы, включая многообразие типов летательных аппаратов, ограниченность ресурсов, неопределённость внешней среды и противодействие. Установлено, что существующие системы остаются зависимыми от человеческого фактора, что требует перехода к автономным адаптивным технологиям. На основе этого разработаны оригинальные модели жизненного цикла группы, сцены операций и автономных устройств, интегрирующие физические и логические аспекты среды. Алгоритмы управления, такие как гибридные методы, роевой интеллект и мультиагентные стратегии, позволили объединить глобальную оптимизацию с локальной адаптацией, обеспечив устойчивость к динамическим изменениям.

Практическая реализация представлена веб-платформой на базе Django и Python, поддерживающей централизованное, децентрализованное и мультиагентное управление. Система интегрирована с IoT-устройствами через протоколы MQTT и HTTP, а её функциональность включает визуализацию данных, администрирование миссий и обработку телеметрии в реальном времени.

Значимость работы заключается в создании универсального инструментария для координации автономных систем, применимого в логистике, спасательных операциях и умных городских инфраструктурах. Разработанные методы демонстрируют потенциал для масштабирования и интеграции с технологиями машинного обучения, что открывает перспективы для прогнозирования возмущений и автономного стратегического планирования.

Дальнейшее развитие проекта может стать основой для формирования стандартов в области управления автономными системами, отвечающих вызовам цифровой трансформации и растущей автоматизации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Будаев Д.С., Вощук Г.Ю., Гусев Н.А., Майоров И.В., Мочалкин А.Н. Разработка интеллектуальной системы адаптивного планирования действий групп летательных аппаратов для согласованного выполнения задач Материалы 3-й Всероссийской научно-технической конференции «РТИ Системы ВКО - 2015», 28 мая 2015г., г. Москва.
2. Рушкевич А., Осадчий В. Мониториг подвижных объектов // Беспроводные технологии. 2010. № 3. С. 56–60.
3. P.Skobelev. Multi-Agent Systems for Real Time Adaptive Resource Management. In Industrial Agents: Emerging Applications of Software Agents in Industry. Paulo Leitão, Stamatis Karnouskos (Ed.). Elsevier, 2015. P. 207–230.
4. Austin, Reg. Unmanned aircraft systems UAVs design, development and deployment. 1st ed. Wiley Aerospace Series, United Kingdom, 2010, pp. 221–226.
5. Методика разработки имитационных моделей [Электрон. ресурс]. Режим доступа: https://www.researchgate.net/publication/228409450 – Дата доступа: 03.03.2025.
6. Горяинов А. Н., Литовчинко Е. С. Диагностика транспортного потенциала систем перевозки грузов // Восточно-европейский журнал передовых технологий. 2010. Вып. 2/9 (44). С. 25–28
7. Holovaty A., Kaplan-Moss J. The Django Book, 2nd. читабельное онлайн-руководство, [Электрон. ресурс]. – Режим доступа: https://djangobook.com – Дата доступа: 24.04.2025.
8. Evolutionary and Swarm Intelligence Algorithms, 2019, Jagdish Chand Bansal, Pramod Kumar Singh, Nikhil R. Pal
9. Drone Swarms in Fire Suppression Activities [Электрон. ресурс]. Режим доступа: https://www.mdpi.com/2504-446X/5/1/17 – Дата доступа: 16.04.2025.
10. Swarm Troopers, David Hambling, 2015, pp. 1-100
11. Skobelev P., Budaev D., Brankovsky A., Voshuk G. Multi-agent tasks scheduling for coordinated actions of unmanned aerial vehicles acting in group// International Journal of Design & Nature and Ecodynamics 13(1), 2018. – pp. 39-45.
12. Koo T.J., Shahruz S.M. Formation of a group of unmanned aerial vehicles (UAVs). American Control Conference (ACC), 2001, pp. 69–74
13. C. Di Franco, G. Buttazzo. Energy-Aware Coverage Path Planning of UAVs. Autonomous Robot Systems and Competitions (ICARSC), 2015 IEEE International Conference, pp. 111-117
14. Baxter J.W., Horn G.S., Leivers D.P. Fly-by-Agent: Controlling a Pool of UAVs via a Multi-Agent System. The 27th SGAI International Conference on Artificial Intelligence, 2008, vol. 21, no.3, pp. 232-237

ПРИЛОЖЕНИЕ А

```
from __future__ import annotations
from dataclasses import dataclass, field
from enum import Enum
from typing import List, Protocol
from datetime import datetime

from services.geo import GeoLocator
from services.power import PowerManager
from services.logging import LoggerFactory
from services.communication import CommunicatorFactory

class DeviceType(Enum):
    UAV = "UAV"
    UGV = "UGV"
    SensorNode = "SensorNode"
    RelayStation = "RelayStation"

class SensorType(Enum):
    LIDAR = "LIDAR"
    GPS = "GPS"
    IMU = "IMU"
    ThermalCamera = "ThermalCamera"
    Microphone = "Microphone"

class DeviceRole(Enum):
    Leader = "Leader"
    Executor = "Executor"
    Scout = "Scout"
    Relay = "Relay"

class OperationMode(Enum):
    Normal = "Normal"
    PowerSaving = "PowerSaving"
    Emergency = "Emergency"
    Patrol = "Patrol"

@dataclass
class Coordinates:
    latitude: float = 0.0
    longitude: float = 0.0
    altitude: float = 0.0

    _geo: GeoLocator = field(default_factory=GeoLocator, init=False)
```

```
    def update(self) -> None:
        """Запрашивает актуальные координаты у сервиса геолокации."""
        lat, lon, alt = self._geo.fetch(self)
        self.latitude, self.longitude, self.altitude = lat, lon, alt

    def as_tuple(self) -> tuple[float, float, float]:
        return self.latitude, self.longitude, self.altitude

@dataclass
class Sensor:
    type: SensorType
    is_active: bool = False
    raw_value: float = 0.0

    def activate(self) -> None:
        self.is_active = True

    def deactivate(self) -> None:
        self.is_active = False

    def collect(self) -> float:
        if not self.is_active:
            return 0.0
        # Эмуляция вызова драйвера
        return self._read_driver()

    def _read_driver(self) -> float:
        # Заглушка для реального API
        return self.raw_value * 1.01  # калибровка

@dataclass
class Battery:
    capacity_wh: float
    charge_wh: float
    base_draw_w: float

    _power_mgr: PowerManager = field(default_factory=PowerManager,
init=False)

    def remaining_pct(self) -> float:
        return max(0.0, self.charge_wh / self.capacity_wh * 100.0)

    def hours_left(self) -> float:
        draw = self.base_draw_w + self._power_mgr.dynamic_load()
```

```
        return self.charge_wh / draw if draw > 0 else float('inf')

    def adjust_for_mode(self, mode: OperationMode) -> None:
        self.charge_wh = self._power_mgr.optimize(self.charge_wh, mode)

@dataclass
class DeviceStatus:
    position: Coordinates = field(default_factory=Coordinates)
    speed_mps: float = 0.0
    integrity: float = 1.0
    mode: OperationMode = OperationMode.Normal

    def degrade(self, amount: float) -> None:
        self.integrity = max(0.0, self.integrity - amount)

    def switch_mode(self, mode: OperationMode) -> None:
        self.mode = mode

@dataclass
class Device:
    id: str
    type: DeviceType
    sensors: List[Sensor] = field(default_factory=list)
    status: DeviceStatus = field(default_factory=DeviceStatus)
    battery: Battery = field(default_factory=lambda: Battery(200.0, 200.0, 20.0))
    role: DeviceRole = DeviceRole.Executor

    _logger = LoggerFactory.get_logger("Device")
    _comm = CommunicatorFactory.get("mqtt")

    def update_position(self) -> None:
        self.status.position.update()
        self._logger.info(f"{self.id} position updated to
{self.status.position.as_tuple()}")

    def sample_sensors(self) -> dict[str, float]:
        data = {}
        for sensor in self.sensors:
            if sensor.is_active:
                data[sensor.type.name] = sensor.collect()
        return data

    def report_telemetry(self) -> None:
        coords = self.status.position.as_tuple()
        battery = self.battery.remaining_pct()
```

```
    sensor_data = self.sample_sensors()
    payload = {
        "id": self.id,
        "timestamp": datetime.utcnow().isoformat(),
        "position": coords,
        "battery": battery,
        "sensors": sensor_data,
        "mode": self.status.mode.value
    }
    self._comm.publish(topic=f"devices/{self.id}/telemetry", message=payload)

def compute_energy_draw(self) -> float:
    draw = self.battery.base_draw_w + sum(
        2.0 for s in self.sensors if s.is_active
    )
    return self._power_adjust(draw)

def _power_adjust(self, draw: float) -> float:
    # Эмуляция динамической регулировки потребления
    factor = 0.8 if self.status.mode == OperationMode.PowerSaving else 1.0
    return draw * factor

def switch_role(self, new_role: DeviceRole) -> None:
    self.role = new_role
    self._logger.debug(f"{self.id} role switched to {self.role.value}")

def __repr__(self) -> str:
    return (
        f"<Device {self.id} [{self.type.value}] "
        f"Role={self.role.value} Integrity={self.status.integrity:.2f}>"
    )
```

ПРИЛОЖЕНИЕ Б

```
from __future__ import annotations
from enum import Enum
from datetime import datetime, timedelta
import math
from typing import List, Protocol
from geo import GeospatialService  # Проектный модуль для геоданных
from comms import ProtocolFactory   # Фабрика протоколов связи
from metrics import TelemetryStore  # Сервис хранения телеметрии

class HazardType(Enum):
    DustStorm = "DustStorm"
    Obstacle = "Obstacle"
    CommunicationJammer = "CommunicationJammer"
    Wind = "Wind"
    Rain = "Rain"

class ReliabilityModel:
    def __init__(self, base_rate: float, sensitivity: float):
        self.base_rate = base_rate
        self.sensitivity = sensitivity
        self.logger = TelemetryStore.get_logger("ReliabilityModel")

    def degradation_rate(self, hazard: Hazard) -> float:
        rate = self.base_rate + self.sensitivity * hazard.intensity
        self.logger.debug(f"Degradation rate for {hazard.type}: {rate:.4f}")
        return rate

    def survival_prob(self, rate: float, duration: timedelta) -> float:
        secs = duration.total_seconds()
        return math.exp(-rate * secs)


class Hazard:
    def __init__(self, htype: HazardType, intensity: float,
                 start: datetime, duration: timedelta,
                 model: ReliabilityModel):
        self.type = htype
        self.intensity = intensity
        self.start = start
        self.duration = duration
        self.model = model

    def current_intensity(self, now: datetime) -> float:
```

```
        end = self.start + self.duration
        return self.intensity if self.start <= now <= end else 0.0

    def survival(self) -> float:
        λ = self.model.degradation_rate(self)
        return self.model.survival_prob(λ, self.duration)

class Device:
    def __init__(self, device_id: str, hazards: List[Hazard]):
        self.id = device_id
        self.hazards = hazards
        self.geo = GeospatialService(self.id)
        self.telemetry = TelemetryStore(self.id)

    def overall_survival(self) -> float:
        prob = 1.0
        for h in self.hazards:
            prob *= h.survival()
        return prob

    def report(self) -> None:
        coords = self.geo.get_coords()
        batt = self.telemetry.fetch_battery()
        self.telemetry.save({
            "time": datetime.utcnow(),
            "coords": coords,
            "battery": batt,
            "survival": self.overall_survival()
        })

class ControlCenter:
    def __init__(self, protocol: ProtocolType):
        self.protocol = ProtocolFactory.create(protocol)
        self.devices: List[Device] = []
        self.reports = TelemetryStore("ControlCenter")

    def register_device(self, device: Device):
        self.devices.append(device)

    def broadcast(self, cmd: Command):
        for d in self.devices:
            self.protocol.send(d.id, cmd.payload)

    def collect(self):
        for d in self.devices:
```

```
            d.report()
            data = d.telemetry.latest()
            self.reports.save(data)


# Псевдосущности для контекста
class CommandType(Enum):
    ROUTE_UPDATE = "ROUTE_UPDATE"
    ROLE_CHANGE = "ROLE_CHANGE"
    EMERGENCY_STOP = "EMERGENCY_STOP"

@dataclass
class Command:
    type: CommandType
    payload: dict
```

ПРИЛОЖЕНИЕ В

```
from __future__ import annotations
from dataclasses import dataclass, field
from enum import Enum
from typing import List, Protocol
from datetime import datetime

# Внутренние сервисы проекта
from services.comm import CommunicatorFactory
from services.logging import LoggerFactory
from services.routing import RoutePlanner
from services.metrics import TelemetryRepository

class ControlMode(Enum):
    CENTRALIZED   = "centralized"
    PARTIAL       = "partial"
    DECENTRALIZED = "decentralized"

class ProtocolType(Enum):
    HTTP      = "http"
    MQTT      = "mqtt"
    ROS       = "ros"
    WebSocket = "websocket"

class CommandType(Enum):
    ROUTE_UPDATE   = "route_update"
    ROLE_CHANGE    = "role_change"
    EMERGENCY_STOP = "emergency_stop"

@dataclass
class Command:
    type: CommandType
    payload: dict[str, any]

@dataclass
class Telemetry:
    timestamp: datetime
    device_id: str
    data: dict[str, any]

class Communicator(Protocol):
    def publish(self, topic: str, message: dict) -> None:
        ...
```

```
@dataclass
class Device:
    id: str
    center: ControlCenter
    sub_leader: SubLeader | None = None

    def send_telemetry(self) -> None:
        metrics = TelemetryRepository.for_device(self.id)
        payload  = {
            "position":   self.get_position(),
            "battery":    self.get_battery_status(),
            "status":     self.get_status_snapshot()
        }
        telemetry = Telemetry(timestamp=datetime.utcnow(), device_id=self.id,
data=payload)
        target = self.sub_leader or self.center
        target.process_telemetry(telemetry)
        metrics.record(telemetry)

    def receive_command(self, cmd: Command) -> None:
        handler = self._get_command_handler(cmd.type)
        handler(cmd.payload)

    def _get_command_handler(self, cmd_type: CommandType) -> Callable[[dict],
None]:
        return {
            CommandType.ROUTE_UPDATE:   self._apply_new_route,
            CommandType.ROLE_CHANGE:    self._apply_role_change,
            CommandType.EMERGENCY_STOP: self._halt_operations
        }[cmd_type]

    # stub methods—предполагают реализацию в mixin'ах или сервисах
    def get_position(self) -> tuple[float, float, float]: ...
    def get_battery_status(self) -> float: ...
    def get_status_snapshot(self) -> dict[str, any]: ...
    def _apply_new_route(self, payload: dict) -> None: ...
    def _apply_role_change(self, payload: dict) -> None: ...
    def _halt_operations(self, payload: dict) -> None: ...


@dataclass
class SubLeader:
    center: ControlCenter
    group: List[Device] = field(default_factory=list)
```

```
    _planner: RoutePlanner = field(default_factory=RoutePlanner)
    _logger  = LoggerFactory.get("SubLeader")

    def optimize_routes(self) -> None:
        routes = {d.id: d.get_position() for d in self.group}
        updated = self._planner.optimize(routes)
        for device_id, path in updated.items():
            cmd = Command(type=CommandType.ROUTE_UPDATE,
payload={"path": path})
            self.center.send_global_command(cmd, via_subleader=True)

    def report_summary(self) -> None:
        summary = {"active": len(self.group), "mode": self.center.mode.value}
        cmd     = Command(type=CommandType.ROLE_CHANGE,
payload=summary)
        self.center.send_global_command(cmd, via_subleader=True)
        self._logger.info(f"Reported summary: {summary}")

@dataclass
class ControlCenter:
    mode: ControlMode
    protocol: ProtocolType

    devices:    List[Device]    = field(default_factory=list)
    sub_leaders:List[SubLeader] = field(default_factory=list)
    _comm:      Communicator    = field(init=False)
    _logger     = LoggerFactory.get("ControlCenter")

    def __post_init__(self):
        self._comm = CommunicatorFactory.build(self.protocol)

    def register_device(self, device: Device) -> None:
        self.devices.append(device)
        self._logger.debug(f"Device registered: {device.id}")

    def register_subleader(self, sl: SubLeader) -> None:
        self.sub_leaders.append(sl)
        self._logger.debug("SubLeader assigned")

    def send_global_command(self, cmd: Command, via_subleader: bool = False) -
> None:
        targets = self._get_targets(via_subleader)
        for dev in targets:
            topic = f"devices/{dev.id}/commands"
```

```
            self._comm.publish(topic, {"type": cmd.type.value, "payload":
cmd.payload})
        self._logger.info(f"Broadcasted {cmd.type.value} to {len(targets)} devices")

    def process_telemetry(self, telem: Telemetry) -> None:
        # Сбор и анализ телеметрии
        TelemetryRepository.global_store().record(telem)
        # Реакция на аварийные ситуации
        if telem.data.get("status", {}).get("integrity", 1.0) < 0.5:
            self._handle_emergency(telem.device_id)

    def switch_mode(self, new_mode: ControlMode) -> None:
        self.mode = new_mode
        self._logger.warning(f"Control mode switched to {new_mode.value}")

    def assign_subleader(self, sl: SubLeader) -> None:
        self.register_subleader(sl)

    def _get_targets(self, via_subleader: bool) -> list[Device]:
        if self.mode == ControlMode.CENTRALIZED and not via_subleader:
            return self.devices
        return [d for sl in self.sub_leaders for d in sl.group]

    def _handle_emergency(self, device_id: str) -> None:
        cmd = Command(type=CommandType.EMERGENCY_STOP,
payload={"device": device_id})
        self.send_global_command(cmd)
        self._logger.error(f"Emergency stop issued for {device_id}")
```