МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка приложения для системы отслеживания изменения состояния внешней среды

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-513.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ А.А. Шаблей

Автор работы,
студент группы КЭ-401
_____ А. Назарчук

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-401
Назарчук Альберто,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764 13/12)

Разработка приложения для системы отслеживания изменения состояния внешней среды.

2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.

3. **Исходные данные к работе**

3.1. Документация m5stack. [Электронный ресурс] URL: https://docs.m5stack.com (дата обращения: 01.02.2024 г.).

3.2. Документация Micropython. [Электронный ресурс] URL: https://docs.micropython.org/en/latest/ (дата обращения: 01.02.2024 г.).

3.3. Документация Python. [Электронный ресурс] URL: https://www.python.org/doc/ (дата обращения: 01.02.2024 г.).

3.4. Кокунин П.А. Введение в Интернет вещей. Учебное пособие / П.А. Кокунин, И.И. Латыпов, Л.С. Латыпова. Казань: Издательство Казанского университета, 2022. – 147 с.

4. Перечень подлежащих разработке вопросов

4.1. Изучить современные платформы и программные системы управления беспроводными устройствами интернет вещей.

4.2. Спроектировать архитектуру приложения.

4.3. Реализовать получение данных с датчиков.

4.4. Реализовать сборку и проезд по заданной траектории.

4.5. Реализовать и протестировать готовое приложение.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
ст. преподаватель кафедры СП А.А. Шаблей

Задание принял к исполнению А. Назарчук

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

Термин интернет вещей (Internet of Things, IoT) представляет собой концепцию физических устройств и объектов, подключенных к интернету, которые могут собирать, передавать и обмениваться данными [1].

Термин «Интернет вещей» официально ввели в обиход спустя девять лет, в 1999 году, благодаря Кевину Эштону. Кевин хотел развить технологию для автоматизации производства благодаря автоматическому сбору данную с помощью того же RFID [5].

Сегодня IoT охватывает широкий спектр приложений, от умных домов до промышленного оборудования. По данным Statista, в 2023 году количество IoT устройств в мире, достигло более 15 миллиардов, а к 2030 году ожидается рост до 30 миллиардов устройств [11]. Рост количества устройств в мире показан на рисунке 1. Российский рынок промышленного интернета вещей за 2023 год вырос на 5% в сравнении с 2022-м и достиг 144,4 млрд рублей [13].

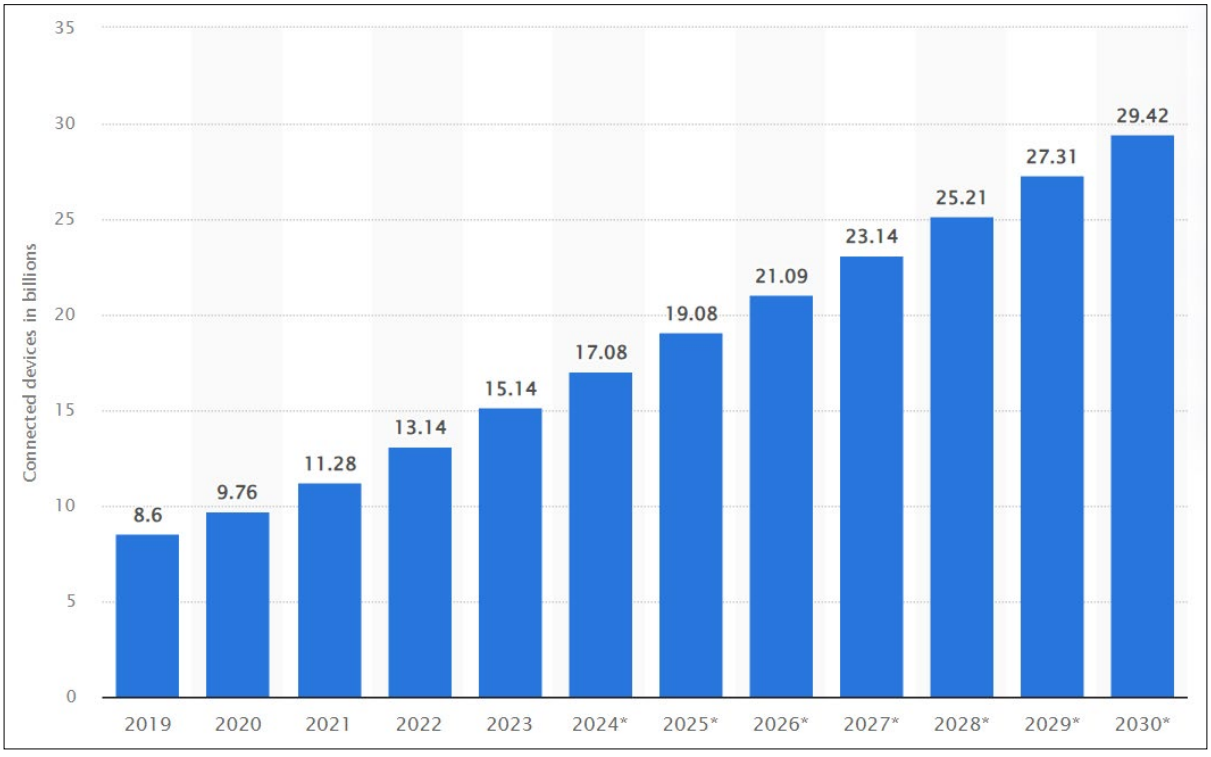

Рисунок 1 – Рост количества IoT устройств в мире

Одной из востребованных и перспективных областей применения IoT являются роботы. Оснащенные датчиками роботы, с прямым доступом в интернет, могут выполнять различные сложные задачи, получая данные и решения поставленных целей из различных источников. Это позволяет использовать роботов в различных сферах повседневной жизни, таких как производство, логистика, медицина, сельское хозяйство и во многих других отраслях.

В бизнесе, IoT имеет огромное значение. Внедрение интернета вещей, позволяет предприятиям оптимизировать производственные процессы, снижать затраты, рационально использовать ресурсы. Для примера, в сельском хозяйстве датчики могут контролировать влажность почвы и автоматизировать полив, это способствует повышению урожайности и снижению затрат на водные ресурсы. В сфере транспорта IoT позволяет отслеживать движение грузов и оптимизировать маршруты, что улучшает логистику и снижает расходы на топливо и время доставки товара до конечного пункта.

IoT делает повседневную жизнь удобнее и безопаснее. Умные дома, оснащенные умными устройствами, позволяют управлять освещением, температурой и другими устройствами с помощью любого устройства с доступом в интернет. В медицине устройства IoT, такие как фитнес-браслеты и умные часы, помогают следить за здоровьем и другими показателями организма.

В России активно внедряются IoT-решения в городской инфраструктуре. Например, умные системы управления уличным освещением и парковкой, умные домофоны и другие системы помогают улучшить качество жизни горожан и экономить драгоценное время [10].

Таким образом, IoT оказывает значительное влияние на различные сферы нашей жизни, от бизнеса до повседневных задач. Развитие технологий IoT открывает новые возможности для повышения эффективности и комфорта, и это направление будет продолжать активно развиваться в ближайшие годы.

Постановка задачи

Целью выпускной квалификационной работы является разработка приложения для системы отслеживания изменения состояния внешней среды. Для достижения поставленной цели необходимо решить следующие задачи:

1) изучить современные платформы и программные системы управления беспроводными устройствами интернета вещей;
2) спроектировать архитектуру приложения;
3) реализовать получение данных с датчиков;
4) реализовать сборку и проезд по заданной траектории;
5) реализовать и протестировать готовое приложение.

Структура и содержание работы

Выпускная квалификационная работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 56 страницы, объем списка литературы – 17 источников.

В первой главе «Анализ предметной области» приведены существующие аналоги 3 систем управления отслеживания изменения состояния внешней среды, выявлены их главные недостатки и преимущества.

Во второй главе «Проектирование системы» описывается диаграмма прецедентов и приведена архитектура приложения.

Третья глава «Реализация приложения» содержит описание реализации компонентов и взаимодействия приложения и датчиков.

В четвертой главе «Тестирование приложения» описывается тестирование приложения. Приведены результаты функционального и юзабилити тестирования, по результатам которых сделан вывод о достижении поставленных целей.

В главе «Заключение» сделаны выводы о проделанной работе и реализованному приложению.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область проекта

Интернет вещей представляет собой технологию, объединяющую физические устройства, оснащенные датчиками и программным обеспечением, которые могут взаимодействовать друг с другом через интернет. Данные технологии находят применение в самых разных сферах, от умных домов и городов до промышленности и сельского хозяйства, значительно повышая эффективность на предприятиях и удобство в повседневной жизни. Одним из ключевых преимуществ интернет вещей является его способность собирать и анализировать огромные объемы данных, что позволяет предприятиям принимать целенаправленные решения и повышать свою продуктивность. Важность таких программ и систем обусловлена их способностью автоматизировать повторяющиеся задачи, снижать затраты и улучшать качество продукции и услуг.

Робототехника играет ключевую роль в этой технологической экосистеме. Роботы, оснащенные сенсорами и подключенные к интернету, могут выполнять автоматизировано сложные задания с высокой степенью точности. В результате интеграции в различные сферы деятельности интернета вещей и робототехники не только повышается производительность, качество продукции и услуг, но и открываются новые горизонты для инноваций.

Темой выпускной квалификационной работы является разработка приложения для системы отслеживания изменения состояния внешней среды. Для реализации системы требуется передвижное устройство с датчиками. Управление данной сборкой должно осуществляться при помощи комплекта M5GO IoT Starter Kit с использованием BaseX и протокола MQTT [12]. В разрабатываемом приложении пользователи должны иметь возможность задавать маршрут, по которой сборка сможет перемещаться, что позволит использовать систему в различных условиях и на разных объектах. Во время передвижения или ожидания приложение собирает данные с установленных датчиков для дальнейшего анализа.

Проект рассчитан на применение на различных предприятиях, таких как склады, фермы и частные дома. Например, на складах робот может использоваться для мониторинга условий хранения и выявления отклонений от установленных норм, что поможет предотвратить порчу товара. В сельском хозяйстве система может отслеживать внутренние параметры окружающей среды теплиц и помогать в управлении сельскохозяйственными процессами. В частных домах такие системы могут использоваться для мониторинга условий окружающей среды и обеспечения безопасности.

Разработка данного приложения демонстрирует возможности подключенных IoT устройств в различных сферах, а также подчеркивает важность интеграции различных технологий для создания эффективных и удобных решений.

1.2. Сравнительный анализ аналогов

В настоящий момент существует различные системы управления для отслеживания изменения состояния внешней среды. Рассмотрим только те решения, которые предусматривают управление датчиками, которые находятся на передвижных основах.

T5 Экобот – компания, специализирующаяся на разработке роботов для экологического мониторинга и анализа окружающей среды. Их решения ориентированы на сбор данных о состоянии окружающей среды и предоставление аналитических отчетов [9]. Пример окна приложения представлен на рисунке 2.

Программное обеспечение:

– включает инструменты для управления роботом, сбора и анализа данных с датчиков, таких как датчики качества воздуха, температуры, влажности и других параметров окружающей среды;

– предоставляет возможность дистанционного управления роботом и мониторинга данных в реальном времени.

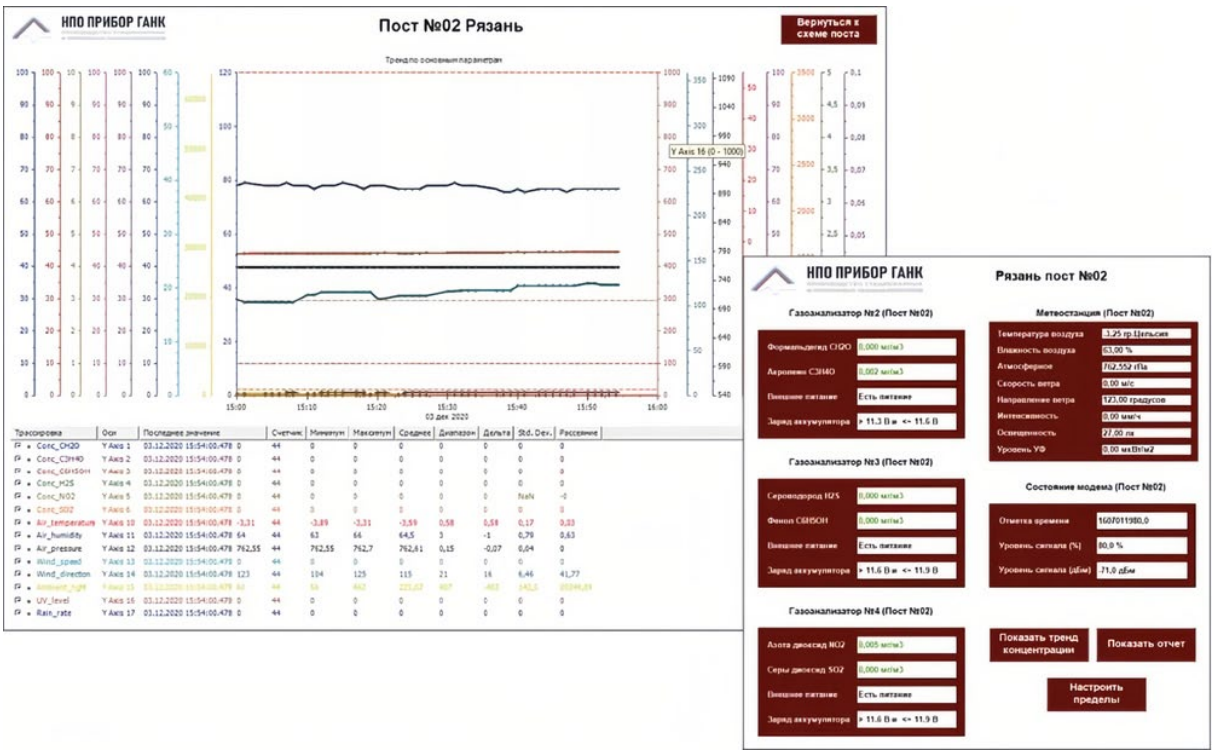

Рисунок 2 – Окно приложения T5 Экобот

Достоинства:

– подходит для задач по сбору экологических данных;

– интуитивно понятный интерфейс для управления и анализа данных;

– более доступная цена по сравнению с решениями от крупных международных компаний.

Недостатки:

– специализированные датчики могут ограничивать использование робота для других задач;

– маленький срок существования на рынке может отражаться на надежности и количестве доступных обновлений и улучшений;

– габариты робота;

– высокая цена на робота и сопутствующее программное обеспечение.

Clearpath Robotics – канадская компания, специализирующаяся на производстве наземных беспилотных транспортных средств (UGV) и робототехнических решений для научных и промышленных приложений [7]. Пример окна приложения представлен на рисунке 3.

Программное обеспечение.

1. Husky UGV Platform – использует Robot Operating System (ROS) для управления роботом и интеграции различных сенсоров. ROS предоставляет множество инструментов для навигации, планирования маршрутов и анализа данных.

2. Gazebo – симулятор, работающий в связке с ROS, позволяющий тестировать алгоритмы и конфигурации робота в виртуальной среде перед развертыванием на реальном устройстве.

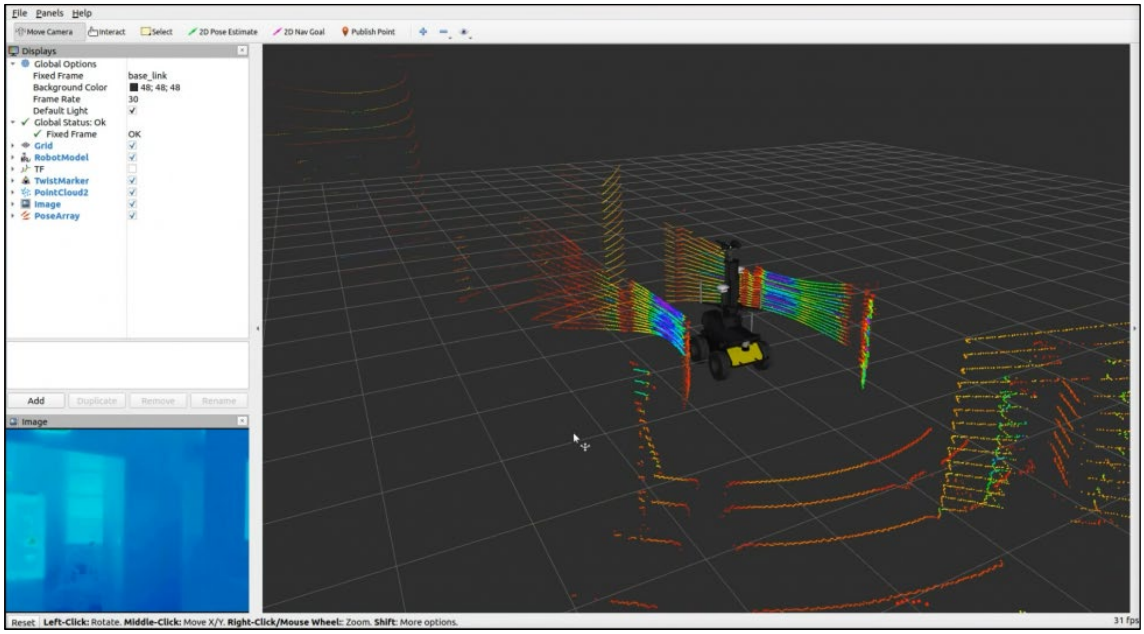

Рисунок 3 – Окно приложения Clearpath Robotics

Достоинства:

– мощная и универсальная платформа Husky UGV может быть оснащена различными датчиками и оборудованием для выполнения широкого спектра задач;

– широко используемая платформа ROS предоставляет доступ к обширной библиотеке инструментов и сообществу разработчиков;

– возможность интеграции новых модулей и датчиков по мере необходимости.

Недостатки:

– необходимы знания в области робототехники и программирования для эффективного использования всех возможностей платформы;

– могут возникнуть сложности при интеграции с другими системами, не поддерживающими ROS;

– высокая цена на робота и сопутствующее программное обеспечение.

Boston Dynamics – американская инженерная и робототехническая компания, известная своими инновационными роботами, такими как Spot, Atlas и Stretch [8].

Программное обеспечение.

1. Spot CORE – программная платформа, предоставляющая возможности для управления роботом Spot и интеграции пользовательских приложений. Включает средства для телеметрии, планирования миссий и анализа данных. Пример окна приложения представлен на рисунке 4.

2. Spot API – набор инструментов для разработки приложений, позволяющих взаимодействовать с различными компонентами робота.

Достоинства:

– Spot может работать в различных условиях, включая труднодоступные места;

– можно интегрировать различные датчики и модули для специфических задач;

– программа Spot CORE предоставляет удобный интерфейс для управления и настройки робота.

Недостатки:

– цена на робота и программное обеспечение высокая;

– требует специальных знаний для полной интеграции и настройки пользовательских приложений.

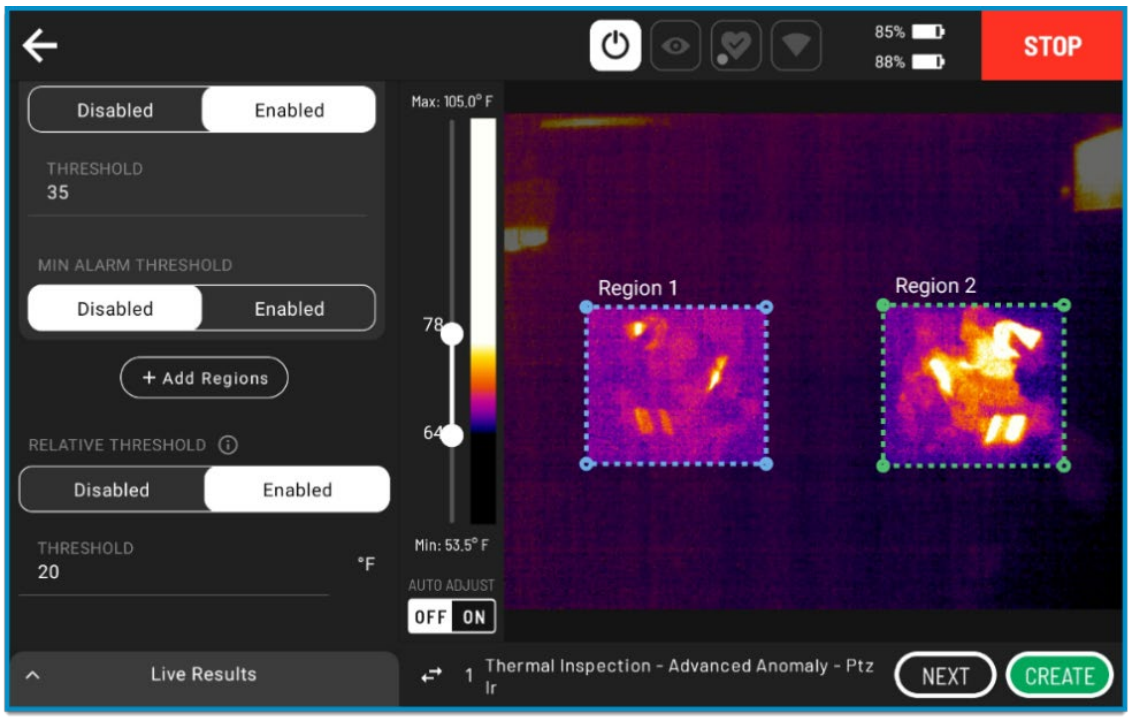

Рисунок 4 – Окно приложения Spot CORE

Каждый из представленных аналогов имеет свою робо-платформу, характеристики представлены в таблице 1.

Таблица 1 – Сравнительная характеристика робо-платформ

Компания	Boston Dynamics	Clearpath Robotics	T5 Экобот
Робот	Spot	Husky UGV	T5 Экобот
Назначение	Универсальные задачи.	Промышленные и исследовательские задачи	Экологический мониторинг
Размеры (Д х Ш х В)	1,1 м х 0,5 м х 0,84 м	0,99 м х 0,67 м х 0,39 м	1,2 м х 0,6 м х 0,8 м
Вес	32,5 кг	50 кг	40 кг
Максимальная скорость	1,6 м/с	1 м/с	0,5 м/с
Время работы от батареи	90 минут	До 8 часов	До 4 часов
Грузоподъемность	До 14 кг	До 75 кг	До 10 кг
Датчики	лидары; камеры; ИК-датчики.	лидары; камеры; ИК-датчики.	качества воздуха; температуры; влажности.
Особенности конструкции	4 ноги; высокая мобильность приспосабливаемость к различным поверхностям.	4 колеса; высокая проходимость; прочная конструкция.	4 колеса; защита от погодных условий.

1.3. Выбор средства реализации

Для сборки конструкции, будет использоваться набор Lego Mindstorms EV3 [4], представленный на рисунке 5. Данный набор удобен тем, что благодаря ему имеется возможность собрать в короткие сроки любую сборку, которая сможет передвигаться различными способами по разным поверхностям. Для универсального решения будет собираться модель на колесах.

Рисунок 5 – Набор Lego Mindstorms EV3

Главным модулем в реализации проекта является M5Go IoT Starter Kit [3], изображённый на рисунке 6. Он включает в себя:

- микроконтроллер ESP32-D0WDQ6-V3;
- Wi-Fi;
- встроенный динамик;
- микрофон;
- IPS - дисплей 320x240;
- 4 кнопки;
- разъемы TypeC, POGO PIN, I2C, GPIO, UART;
- встроенный акселерометр и гироскоп.

Дополнительные датчики:

- датчик ENV II – измеряет температуру, влажность и давление;

– датчик PIR – обнаруживает движение;
– датчик Light – освещенности;

Рисунок 6 – Набор M5Go IoT Starter Kit

Для соединения [3] модуля M5Go с конструкцией, используется модуль BaseX, представленный на рисунке 7. BaseX – это специальная база, совместимая с двигателем LEGO EV3. Модуль имеет 4-канальный порт, который позволяет одновременно подключить 4 двигателя через разъем RJ11, поддерживающий считывание и управление углом, скоростью и совместимому с оригинальными функциями двигателя. Для адаптации к различным сценариям использования предусмотрены интерфейс UART (16/17) и интерфейс GPIO (26/36), делающие доступ к различным датчикам более гибким. В базу встроен аккумулятор емкостью 950 мАч, который можно заряжать через интерфейс USB-C m5core.

В качестве средства реализации программы на M5Go будет использоваться язык программирования MicroPython [7] и среда разработки UiFlow 1.0. MicroPython – это интерпретируемый язык программирования, оптимизированный для работы на микроконтроллерах. UIFlow [10] – это среда разработки от M5Stack поддерживающая текстовое программирование на MicroPython позволяющая видеть интерфейс блока.

Рисунок 7 – Модуль BaseX

Для передачи данных между устройством M5Go и разработанным приложением будет использоваться протокол MQTT (Message Queuing Telemetry Transport) [6]. Этот протокол обеспечивает эффективную и надежную передачу сообщений в реальном времени между устройствами и брокерским сервером, который в свою очередь будет передавать данные в приложение. Использоваться будет коммерческий брокер от HiveMQ с авторизацией – HiveMQ Cloud.

Для написания приложения системы для отслеживания изменения состояния внешней среды будет использоваться язык программирования Python [9], вместе с библиотекой PyQt6 [8].

Вывод по первой главе

В рамках данной главы была сформулирована постановка задачи.

Были рассмотрены 3 системы управления для отслеживания изменения состояния внешней среды, выявлены их главные недостатки и преимущества.

Произведен выбор средств реализации. Главным модулем реализации проекта является платформа M5Go IoT Starter Kit в связке с модулем BaseX. Для написания программы выбран язык программирования – Python. Для передачи данных выбран протокол MQTT.

2. ПРОЕКТИРОВАНИЕ

2.1. Анализ требований

Функциональные требования – это требования, которые определяют действия, которые должна выполнять система, без учета ограничений, связанных с ее реализацией, то есть определяют поведение системы в процессе обработки информации.

Определены следующие функциональные требования к проектируемому приложению.

1. Наличие 2-х типов пользователей – авторизованный и неавторизованный.
2. Ознакомление с функционалом приложения.
3. Первичная настройка программы: выбор блока подключения и проверка раннее полученных данных.
4. Предоставление пользователю возможность выбрать, изменить и задать траекторию проезда робота.
5. Возможность приостановить и остановить проезд робота по заданной траектории.
6. Предоставление возможности пользователю управлять выбранным датчиком, а именно включить и выключить.
7. Отображение показаний датчиков в реальном времени, а также построение графика на их основе в реальном времени.
8. Управление данными, полученных с датчиков.

Нефункциональные требования – это требования, которые определяют свойства и ограничения которых должна придерживаться реализующая система.

Определены следующие нефункциональные требования к проектируемому приложению.

1. Приложение должно работать на операционной системе Windows 10 и новее.

2. Приложение должно быть реализовано на языке программирование Python.

На основе анализа требований была разработана диаграмма вариантов использования, которая представлена на рисунке 8.

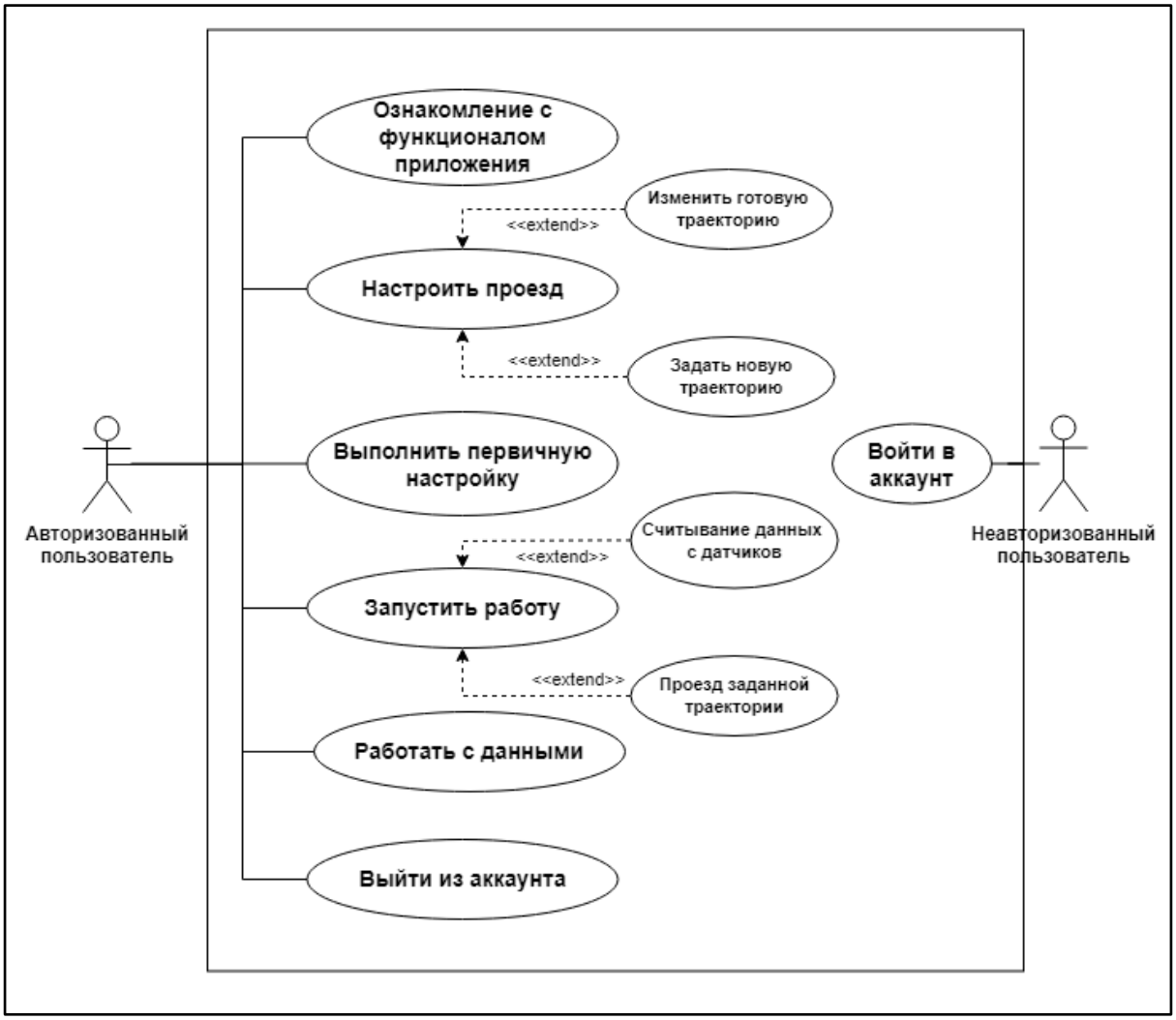

Рисунок 8 – Диаграмма вариантов использования

2.2. Архитектура приложения

Архитектура приложения представлена на рисунке 9 в виде диаграммы размещения. Диаграмма размещения показывает размещение программных элементов в физической архитектуре системы и взаимодействие между физическими элементами.

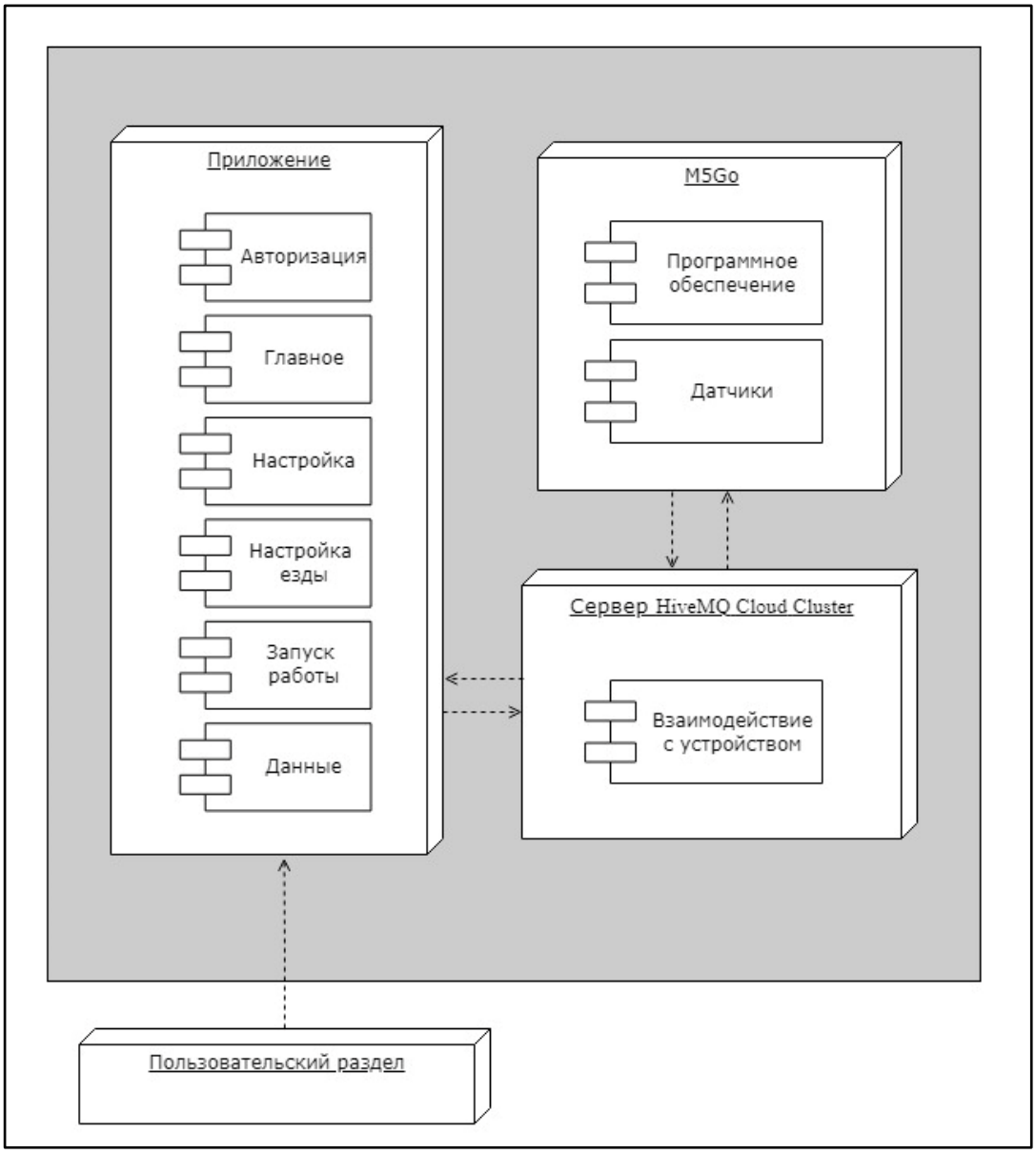

Рисунок 9 – Архитектура приложения

Узел «Приложение» представляет собой оконное приложение, в котором доступны следующие возможности для пользователя.

1. Авторизация – пользователь может войти под своим логином и паролем в личный профиль.

2. Главное – пользователь может ознакомиться с функционалом приложения.

3. Настройка – пользователь может просмотреть полученные данные с прошлого запуска программы. Имеется возможность подключить сборку.

4. Настройка движения – имеется возможность настроить движение сборки по определенной траектории. Есть возможность загрузить изображение-схему помещения, задать точное расстояние проезда.

5. Запуск работы – пользователь может запустить сборку по заданной траектории путем отправки команды на сервер. Вне зависимости от состояния сборки (в движении/ожидании), имеется возможность включить/выключить считывание показание с датчиков, путем получения команды с сервера.

6. Данные – возможность просмотреть и отфильтровать полученные данные по различным параметрам.

Данный узел взаимодействует с «Сервер HiveMQ Cloud Cluster».

Узел «Сервер HiveMQ Cloud Cluster» обеспечивает двустороннюю передачу данных, данные с датчиков к приложению, и команды от приложения к сборке.

Узел «M5Go» – это физическое устройство, содержащее датчики и исполнительные механизмы.

Вывод по второй главе

Были описаны функциональные и нефункциональные требования к проектируемому приложению.

Архитектура приложения представлена в виде диаграммы размещения, показывающей взаимодействие программных элементов с физическими компонентами системы.

Основные возможности приложения включают в себя авторизацию, настройку траектории движения робота, запуск робота и просмотр данных с датчика. Приложение взаимодействует с сервером HiveMQ Cloud Cluster, обеспечивающим передачу данных от датчиков к приложению и команд от приложения к роботу.

Приложение должно включать поддержку операционной системы Windows 10 и выше и использовать язык программирования Python.

3. РЕАЛИЗАЦИЯ

3.1. Реализация сборки

Для реализации конструкции был использован набор Lego Mindstorms EV3. Сборка представляет из себя своеобразную тележку, состоящую из балок, штифтов, осей, 2-х моторов и колес, шарикового подшипника. За движение отвечают 2 мотора в которых присутствует встроенный датчик вращения-энкодер с разрешением 1 градус для точного контроля. За передачу данных отвечают 4 датчика. Управление сборкой осуществляется благодаря модулю M5Go в связке с BaseX. Характеристика сборки представлена в таблице 2, демонстрация сборки представлена на рисунке 10.

Таблица 2 – Характеристика реализованной сборки

Робот	Тележка
Целевое назначение	Изменения состояния внешней среды.
Размеры (Д х Ш х В)	0,19 м х 0,14 м х 0,12 м
Вес	0,45 кг
Максимальная скорость	0,73 м/с
Время работы от батареи	60 минут
Грузоподъемность	До 2 кг
Датчики	датчик окружающей среды; датчик движения; датчик освещенности; акселерометр и гироскоп.
Особенности конструкции	Быстрая модернизация под различные задачи.

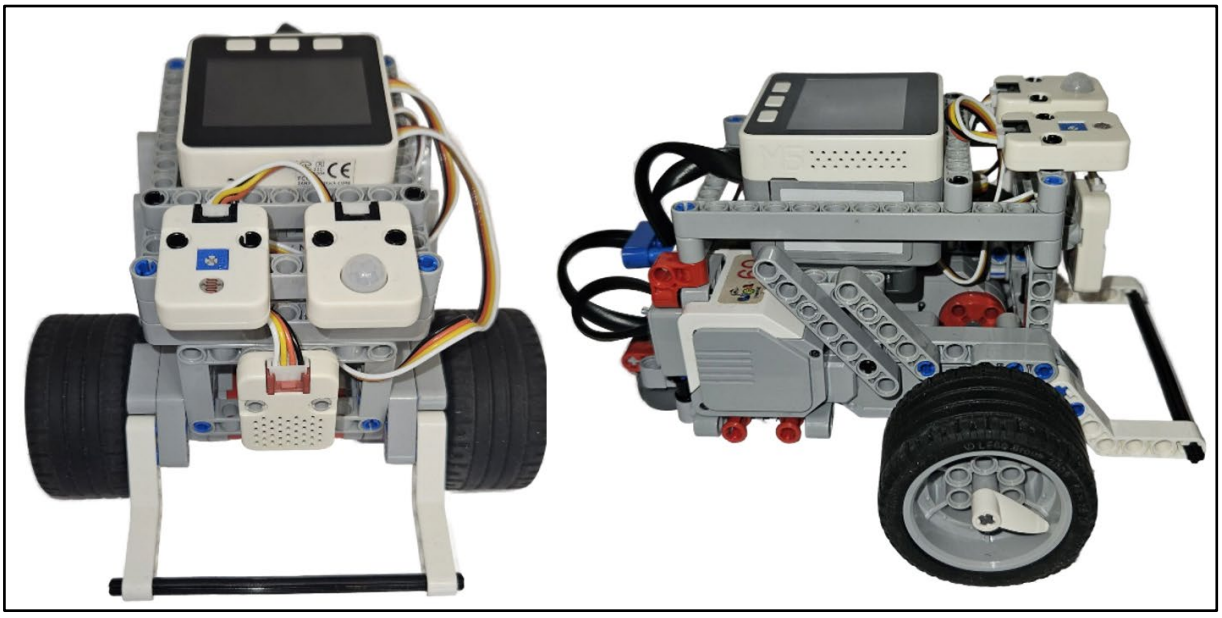

Рисунок 10 – Реализация сборки, вид спереди и сбоку

3.2. Реализация управления сборкой

В качестве средства реализации программы на M5Go [17] используется язык программирования MicroPython и среда разработки UiFlow 1.0.

Для корректной работы программы потребовалось подключить следующие библиотеки.

1. `M5stack` – базовая библиотека для работы с устройствами серии M5Stack.

2. `M5ui` и `uiflow` – библиотеки для настройки пользовательского интерфейса на экране устройства.

3. `Json` – стандартная библиотека Python для работы с JSON-данными, необходимая для обработки сообщений и данных.

4. `M5mqtt` – библиотека для работы с MQTT-протоколом, обеспечивающая возможность публикации и подписки на сообщения.

5. Дополнительные модули для работы с датчиками, управления временем, выполнения математических операций и управления потоками.

Для взаимодействия сборки и приложения используется MQTT брокер HiveMQ Cloud. Код представлен на рисунке 11.

```
mqtt = M5mqtt(
'm5stack',
'f5906e674a034090b87e779e1350a154.s1.eu.hivemq.cloud',
8883,
'AdminUser',
'BaWGs4@hb?D1%v,R35Ai',
300,
ssl=True,
ssl_params={'server_hostname':
'f5906e674a034090b87e779e1350a154.s1.eu.hivemq.cloud'}
)
mqtt.start()
```

Рисунок 11 – Код взаимодействия сборки и приложения

В данном коде мы указываем данные для подключения, а именно:

- имя клиента;
- URL сервера;
- порт (8883 для защищенного соединения);
- логин и пароль для аутентификации;

– таймаут (300 секунд);

– SSL для защиты соединения;

– SSL параметры для валидации сертификата сервера.

Для считывания команд из канала, используется команда `mqtt.subscribe`. Для отправки сообщений в канал используется команда `mqtt.publish`.

За инициализацию двигателей и датчиков отвечает следующий код, представленный на рисунке 12.

```
motorPort1 = 3
motorPort2 = 4
base_x = module.get(module.BASE_X)
imu_sensor = imu.IMU()
env_sensor = unit.get(unit.ENV2, unit.PORTA)
light_sensor = unit.get(unit.LIGHT, unit.PORTB)
pir_sensor = unit.get(unit.PIR, unit.PORTC)
```

Рисунок 12 – Код инициализации двигателей и датчиков

Код для авторизации пользователя, представлен на рисунке 13.

```
def check_login_and_password(topic_data):
    global authorized
    try:
        data = json.loads(topic_data)
        login = data.get('login')
        password = data.get('password')
        label0.setText("Login: " + login)
        label1.setText("Password: " + password)
        if login == admin_login и password == admin_password:
            mqtt.publish('/authorization', 'true')
            label2.setText("Status: true")
            authorized = True
        else:
            mqtt.publish('/authorization', 'false')
            label2.setText("Status: false")
    except Exception as e:
        label2.setText("Error: " + str(e))
```

Рисунок 13 – Код авторизации

Для авторизации используется следующий принцип работы.

1. Получение данных – функция получает данные в формате JSON.
2. Анализ данных – извлекает логин и пароль.

3. Сравнение – проверяет соответствие логина и пароля с заданными значениями.

4. Публикация результата – результат проверки (true/false) отправляется через MQTT.

Для публикации данных датчиков используется функция `def publish_sensor_data`, представленная на рисунке 14. Происходит проверка флагов, то есть публикуются данные только тех датчиков, флаги которых установлены.

```
def publish_sensor_data():
    while True:
        if sensor_flags["temperature"]:
            temperature = str(env_sensor.temperature)
            mqtt.publish('/sensor/temperature', temperature, 0)
            label0.setText("Published temperature: " + tempera-
ture)
        if sensor_flags["pressure"]:
            pressure = str(env_sensor.pressure)
            mqtt.publish('/sensor/pressure', pressure, 0)
            label0.setText("Published pressure: " + pressure)
        if sensor_flags["humidity"]:
            humidity = str(env_sensor.humidity)
            mqtt.publish('/sensor/humidity', humidity, 0)
            label0.setText("Published humidity: " + humidity)
        if sensor_flags["movement"]:
            movement = str(pir_sensor.state)
            mqtt.publish('/sensor/movement', movement, 0)
            label0.setText("Published movement: " + movement)
        if sensor_flags["light"]:
            light = str(light_sensor.analogValue)
            mqtt.publish('/sensor/light', light, 0)
            label0.setText("Published light: " + light)
        if sensor_flags["aks"]:
            aks = str(imu_sensor.acceleration[1])
            mqtt.publish('/sensor/akselerometr', aks, 0)
            label0.setText("Published aks: " + aks)
        wait(2)
```

Рисунок 14 – Код публикации данных датчиков

Для движения сборки на прямые траектории используется функция `def move_up` и `def rotate_motor` – для перемещения на заданный угол (повороты).

Принцип работы кода следующий.

1. Рассчитывается количество градусов для вращения моторов на определенное расстояние.

2. Моторы переходят в режим позиционирования.
3. Задаются конечные углы поворота для моторов.
4. Моторы запускаются на заданной скорости.
5. Моторы ждут достижения заданного положения и останавливаются.

Формула 1 используется для расчета количества градусов [3] вращения мотора для движения на определенное расстояние:

$$change\_degrees = \left(\frac{distance_{cm}}{conversion_{factor}}\right) * 360, \quad (1)$$

где $distance\_cm$ – расстояние, на которое нужно двигаться (в сантиметрах);
$conversion\_factor$ – расстояние, которое проезжает робот за 1 оборот (17,6 см);
$change\_degrees$ – количество градусов, на которое нужно повернуть моторы.

Код функции проезда вперед представлен на рисунке 15.

```
def move_up(distance_cm):
    change_degrees = int(distance_cm / conversion_factor * 360)
    current_degrees1 = base_x.get_encoder(motorPort1)
    current_degrees2 = base_x.get_encoder(motorPort2)
    final_degrees1 = current_degrees1 - change_degrees
    final_degrees2 = current_degrees2 - change_degrees
    base_x.set_mode(motorPort1, base_x.POSITION_MODE)
    base_x.set_mode(motorPort2, base_x.POSITION_MODE)
    base_x.set_position_point(motorPort1, final_degrees1)
    base_x.set_position_point(motorPort2, final_degrees2)
    base_x.set_motor_speed(motorPort1, 50)
    base_x.set_motor_speed(motorPort2, 50)
    while (base_x.get_encoder(motorPort1) > final_degrees1) or
(base_x.get_encoder(motorPort2) > final_degrees2):
        wait_ms(100)
    base_x.set_motor_speed(motorPort1, 0)
    base_x.set_motor_speed(motorPort2, 0)
    base_x.set_mode(motorPort1, base_x.NORMAL_MODE)
    base_x.set_mode(motorPort2, base_x.NORMAL_MODE)
```

Рисунок 15 – Код реализации передвижения на прямые траектории

Формула 2 представлена для расчета длины (расстояния), для поворота на определенный угол:

$$arc\_length\_cm = \left(\frac{circumference_{cm} * rotation_{degrees}}{360}\right), \quad (2)$$

где $circumference\_cm$ – окружность колеса (в сантиметрах), вычисленная по формуле 3:

$$circumference\_cm = 2 * \pi * radius\_cm, \quad (3)$$

где $rotation\_degrees$ – угол поворота (в градусах);

arc_length_cm – длина дуги, на которую нужно повернуть (в сантиметрах).

Формула 4 для расчета количества градусов вращения мотора для движения на определенное расстояние (дуга):

$$change\_degrees = \left(\frac{arc_{length_{cm}}}{conversion_{factor}}\right) * 360. \quad (4)$$

Код функции проезда на заданный угол представлен на рисунке 16.

```
def rotate_motor(port, rotation_degrees):
    arc_length_cm = (circumference_cm * rotation_degrees) / 360
    change_degrees = round(arc_length_cm / conversion_factor * 360)
    current_degrees = base_x.get_encoder(port)
    final_degrees = current_degrees + change_degrees
    base_x.set_mode(port, base_x.POSITION_MODE)
    base_x.set_position_point(port, final_degrees)
    base_x.set_motor_speed(port, 50)
    while base_x.get_encoder(port) < final_degrees:
        wait_ms(100)
    base_x.set_motor_speed(port, 0)
    base_x.set_mode(port, base_x.NORMAL_MODE)
```

Рисунок 16 – Код реализация передвижения на заданный угол

Для обработки команд используется функция `def handle_command` код которой представлен на рисунке 17.

Принцип работы кода.

1. Функция получает команду в виде строки.
2. В зависимости от команды выполняется соответствующая функция:

- `move_up` – двигает устройство вверх;
- `move_down` – двигает устройство вниз;
- `pause` – ставит движение на паузу;

– `stop` – останавливает движение.

3. Обновление состояния – обновляет метку состояния и публикует статус команды через MQTT.

```
def handle_command(topic_data):
    global start_movement
    command = topic_data
    if command == 'move_up':
        label1.setText("Состояние: Двигается вверх")
        mqtt.publish('/ride/command', 'moveUpActive')
        move_up(10)
    elif command == 'move_down':
        label1.setText("Состояние: Двигается вниз")
        mqtt.publish('/ride/command', 'moveDownActive')
        move_down(10)
    elif command == 'pause':
        start_movement = False
        label1.setText("Состояние: Пауза")
        mqtt.publish('/ride/command', 'pauseActive')
    elif command == 'stop':
        start_movement = False
        label1.setText("Состояние: Остановлен")
        mqtt.publish('/ride/command', 'stopActive')
```

Рисунок 17 – Код функции обработки команд

3.3. Реализация приложения

Приложение для системы отслеживания изменения состояния внешней среды написано на языке программирования Python, с использованием библиотеки PyQt6. Скриншоты реализации представлены в приложении Б.

Для удобства пользователя было реализовано меню, для которого создается окно с боковой панелью которое позволяет переключаться между страницами и основной областью содержимого при помощи класса `QStackedWidget`.

Были реализованы следующие страницы в меню:

– авторизация;

– главное;

– настройки;

– настройка езды;

– данные;

– выход.

Ниже приводятся основные компоненты программы.

Модуль «Меню»

Модуль меню создает боковую панель с профилем, кнопками навигации и кнопкой выхода. Кнопки добавляются в вертикальный компоновщик, и каждому элементу назначаются соответствующие стили и обработчики событий. Код представлен в листинге 1 приложения А.

Для понимания, в каком разделе находится пользователь, подсвечивается активная кнопка при помощи функции из рисунка 18.

```
def highlight_active_button(self, index):
    all_buttons = [self.profile_button] + self.buttons +
[self.logout_button]
    for i, button in enumerate(all buttons):
```

Рисунок 18 – Код отображения активной кнопки

Модуль «Авторизация»

Код, отвечающий за авторизацию, отправку данных и получение ответа, включает несколько методов класса `AuthorizationWidget`. Основные компоненты данного модуля программы приведены в листинге 2 приложения А.

Отправка данных авторизации. Метод `on_login_clicked` собирает данные из полей ввода и отправляет их на сервер через MQTT.

Инициализация MQTT-клиента. Метод `init_mqtt_client` настраивает соединение с MQTT-брокером и подписывается на топики для получения сообщений.

Получение и обработка ответа от сервера. Методы `on_message`, `show_auth_success`, `show_auth_failure` и `show_invalid_response` обрабатывают ответы от сервера и обновляют интерфейс.

Модуль «Настройки»

Модуль настройки содержит в себе следующие функции.

1. Инициализация MQTT [12]. Создание MQTT клиента, настройка TLS, установка имени пользователя и пароля, подключение к MQTT серверу и запуск клиента в отдельном потоке для постоянного слушания.

2. Создание интерфейса. Интерфейс состоит из двух частей: левая часть для работы с файлами данных датчиков, правая часть для управления подключением к блоку M5Go.

3. Работа с файлами данных. Загрузка файлов данных из директории `sensor_data`.

4. Отображение содержимого выбранного файла в текстовом поле.

5. Очистка данных выбранного файла или всех файлов.

MQTT взаимодействие с блоком представлено на рисунке 19. Происходят следующие действия:

– отправка запроса на подключение к датчику;

– обработка данных о состоянии подключения, уровне заряда батареи, статусе зарядки и версии прошивки, поступающих по MQTT.

```
def on_sensor_data(self, client, userdata, msg):
    response = msg.payload.decode()
    if response == "true":
        self.status_label.setText("Состояние: Подключено")
        self.connected = True
        self.continue_button.setEnabled(True)
        self.connection_warning_label.hide()
        self.subscribe_to_data_channels()
    else:
        self.status_label.setText("Состояние: Отключено")
        self.continue_button.setEnabled(False)
        self.connection_warning_label.show()
def subscribe_to_data_channels(self):
    if self.connected:
        client.subscribe("/m5/battery")
        client.message_callback_add("/m5/battery", self.update_charge)
        client.subscribe("/m5/charging")
        client.message_callback_add("/m5/charging", self.update_charg-
ing)
        client.subscribe("/m5/version")
        client.message_callback_add("/m5/version", self.update_version)
def update_charge(self, client, userdata, msg):
    charge = msg.payload.decode()
    self.charge_label.setText(f"Заряд: {charge}")
def update_charging(self, client, userdata, msg):
    charging = msg.payload.decode()
    self.charging_label.setText(f"Зарядка: {charging}")
def update_version(self, client, userdata, msg):
    version = msg.payload.decode()
    self.version_label.setText(f"Версия: {version}")
self.continue_button.setEnabled(True)
self.continue_button.setEnabled(False)
```

Рисунок 19 – Код взаимодействия с блоком

Метод «Настройка езды»

Метод настройки езды, приведенный в листинге 3 приложения А, включает в себя следующие компоненты.

1. Создание пользовательского интерфейса (UI). Класс `RideSettings` наследуется от `QWidget` и создает окно с различными элементами управления и областями для рисования.

2. Основные компоненты UI:

– `QGraphicsView` – окно для рисования, отображающее `QGraphicsScene`;

– `QGroupBox` – группы параметров для управления направлением, изображением и сохранением/загрузкой направлений;

– `QSpinBox`, `QPushButton`, `QLabel` – элементы управления для ввода параметров и взаимодействия с пользователем.

3. Рисование направлений:

– `draw_square(self, x, y, size=10)` – рисует квадрат в заданных координатах;

– `draw_arrow(self, x1, y1, x2, y2)` – рисует стрелку между двумя точками;

– `draw_direction(self, direction,angle=None)` – определяет направление движения и рисует соответствующую стрелку.

4. За предпросмотр направления отвечает класс `DirectionPreview`. Является Внутренним классом, который используется для отображения предварительного предпросмотра направления с помощью пунктирной линии. Методы для управления предпросмотром:

– `show_preview(self, direction, angle=None)` – показывает предварительный просмотр направления;

– `remove_preview(self)` – удаляет предварительный просмотр.

5. Методы для загрузки и управления изображениями:

– `load_background_image(self, file_path)` – загружает изображение и добавляет его на сцену;

– `clear_background_image(self)` – очищает изображение;

– `rotate_image(self)` – поворачивает изображение на 90 градусов.

6. Методы для работы с направлениями:

– `save_directions(self)` – сохраняет текущие направления в файл;

– `load_directions(self)` – загружает направления из файла;

– `update_directions_text(self)` – обновляет текстовое поле с текущими направлениями;

– `draw_and_update_directions(self, direction)` – рисует направление и обновляет текстовое поле.

7. Методы для очистки и обновления сцены:

– `clear_drawing(self)` – очищает все нарисованные элементы, кроме изображения;

– `remove_last_arrow(self)` – удаляет последнюю нарисованную стрелку;

– `update_scene_rect(self)` – обновляет размеры сцены в соответствии с нарисованными элементами.

Модуль «Запуск M5Go»

Функции управления блоком включают:

– загрузку текстовых файлов из указанной директории с созданным направлением;

– отправку содержимого файла и команд на MQTT-сервер;

– управление состоянием работы через MQTT.

Код реализации представлен на рисунке 20.

Модуль также предназначен для мониторинга датчиков. Основные функции включают отображение данных от датчиков, управление состоянием датчиков (включение/выключение), и визуализацию данных с помощью графиков.

```
def on_mqtt_message(self, client, userdata, msg):
    response = msg.payload.decode()
    if response == "true":
        self.status_label.setText("Успешно отправлены")
        self.start_button.setDisabled(False)
        self.pause_button.setDisabled(False)
        self.stop_button.setDisabled(False)
    elif response == "pauseActive":
        self.status_label.setText("Пауза")
    elif response == "startActive":
        self.status_label.setText("Езда запущена")
    elif response == "stopActive":
        self.status_label.setText("Стоп")
```

Рисунок 20 – Основные функции управления блоком

Классы для взаимодействия с датчиками представлены ниже.

1. Класс `SensorWidget` – отвечает за создание виджетов для каждого датчика, отображение их данных и управление состоянием датчиков.

2. `set_default_style` – устанавливает стиль по умолчанию для виджета.

3. `set_active_style` – устанавливает стиль для активного состояния датчика.

4. `toggle_sensor` – переключает состояние датчика (включение/выключение) и отправляет соответствующее сообщение на MQTT-сервер. Также управляет подпиской на топики.

5. `update_data` – обновляет отображаемые данные датчика и отправляет сигнал для обновления графика. Также сохраняет данные.

6. `show_graph` – отображает окно с графиком для данного датчика.

7. `on_response` – обрабатывает сообщения от сервера для управления состоянием датчика.

В листинге 4 приложения А приведен фрагмент кода «Взаимодействие с датчиками».

Модуль «Данные»

Пользователь может выбирать файл, где сохранены данные датчиков, просматривать их содержимое, применять фильтры к данным и сохранять

отфильтрованные данные в новые файлы. Интерфейс включает в себя графическое отображение данных с использованием библиотеки pyqtgraph.

Функция `apply_filter` отвечает за применение фильтров к отображаемым данным. Реализация функции представлена на рисунке 21.

```
def apply_filter(self):
    current_item = self.sensor_list.currentItem()
    if current_item:
        filename = current_item.text()
        start = self.start_value_spinbox.value()
        end = self.end_value_spinbox.value()
        start_date = self.start_date_edit.date().toPyDate()
        start_time = self.start_time_edit.time().toPyTime()
        end_date = self.end_date_edit.date().toPyDate()
        end_time = self.end_time_edit.time().toPyTime()
        content = self.file_preview.toPlainText()
        self.update_graph(content, start, end, start_date, start_time,
end_date, end_time)
    else:
        QMessageBox.warning(self, "Ошибка", "Пожалуйста, выберите файл
датчика.")
```

Рисунок 21 – Функция применения фильтра

3.4. Реализация передачи данных

Для приёма показаний с датчиков, а также отправку команд на выполнение был использован брокер HiveMQ Cloud.

HiveMQ Cloud – это мощный и гибкий MQTT-брокер, который обеспечивает надежную связь для интернета вещей (IoT) и других приложений, использующих протокол MQTT (Message Queuing Telemetry Transport).

Для передачи данных требуется получить информацию о кластере, для дальнейшего использования. Данные для подключения представлены на рисунке 22.

После создания пользователя, который в дальнейшем используется в программе, создаются каналы для получения данных, которые приведены на рисунке 23.

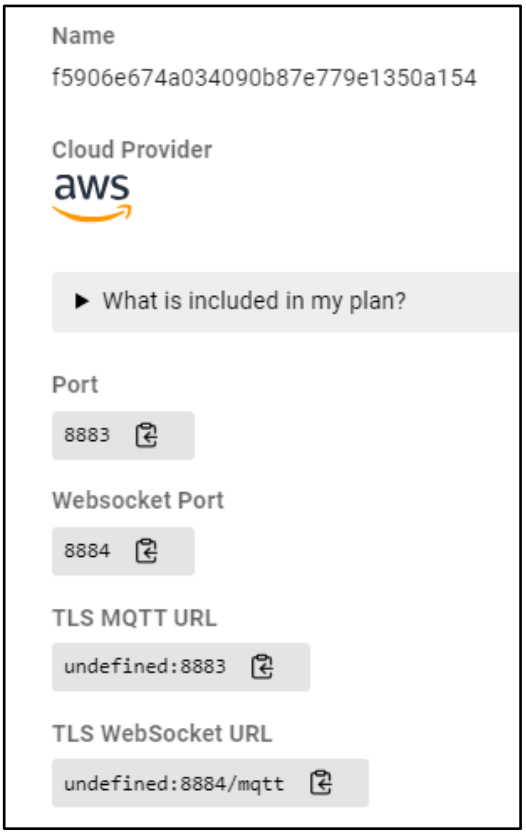

Рисунок 22 – Данные подключения

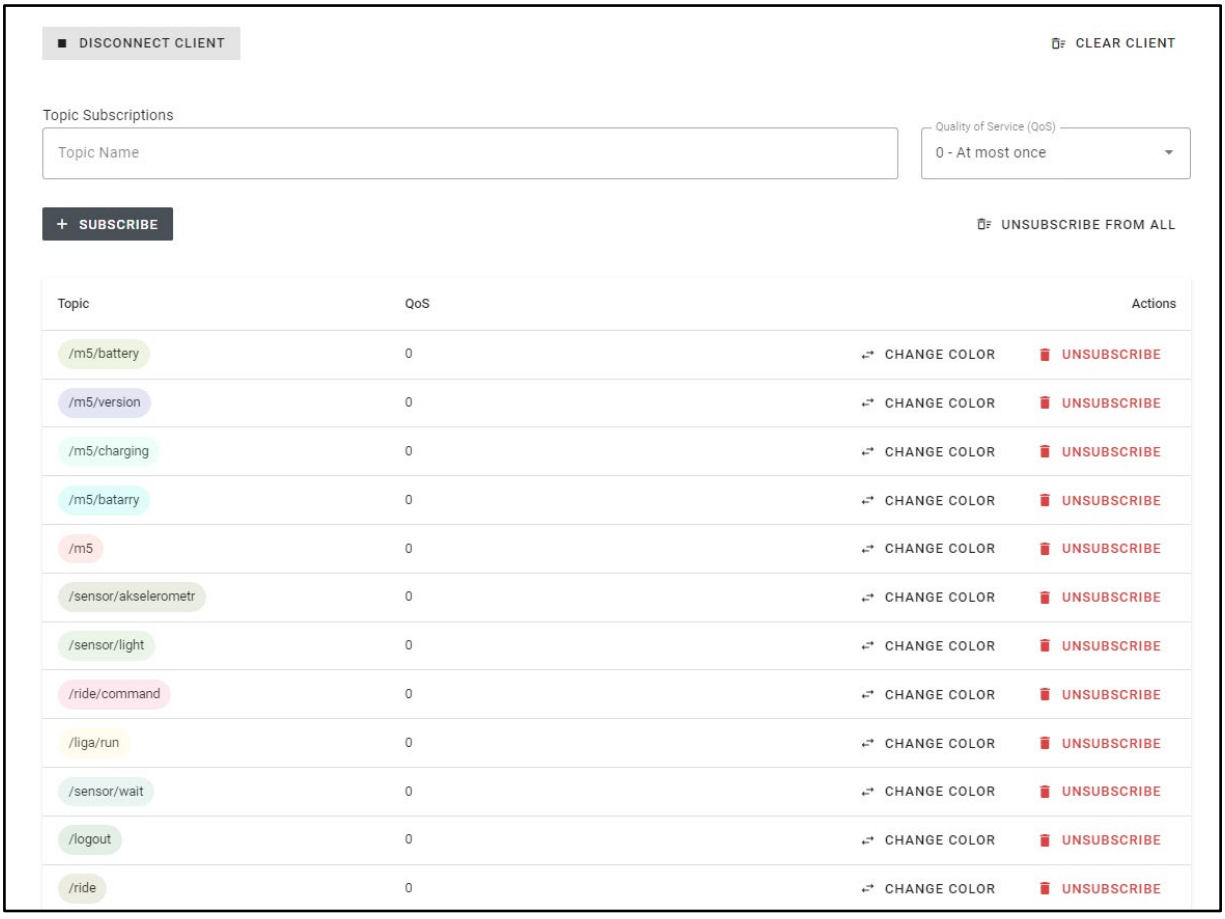

Рисунок 23 – Созданные каналы

Вывод по третьей главе

Глава посвящена реализации приложения для системы отслеживания состояния внешней среды на базе блока M5Go.

В разделе 3.1 описана конструкция сборки, которая представляет собой тележку с двумя моторами и набором датчиков для получения данных. Управление осуществляется с помощью M5Go и BaseX, а точное движение обеспечивается встроенными энкодерами моторов. Приложение разработано с использованием MicroPython и среды UiFlow 1.0.

В разделе 3.2 рассмотрена реализация управления сборкой через MQTT брокер HiveMQ Cloud. Основные используемые библиотеки включают M5stack, M5ui, uiflow, JSON, M5mqtt и другие для работы с датчиками, управления временем и выполнения математических операций. Приведен код для подключения к MQTT серверу, обработки команд и публикации данных с датчиков. Также описан процесс авторизации пользователя и функции для управления движением тележки.

Приложение для системы написано на Python с использованием библиотеки PyQt6 для создания пользовательского интерфейса. Реализованы различные модули, такие как меню, авторизация, главное, настройки, настройка езды и данные. В каждом модуле предусмотрены функции для взаимодействия с пользователем, управления подключением к M5Go, обработки и отображения данных с датчиков, а также отправки команд на выполнение через MQTT.

В разделе 3.3 описана реализация передачи данных с использованием брокера HiveMQ Cloud. Приведен пример подключения к кластеру и создания каналов для получения и отправки данных. Обсуждаются особенности работы с брокером для обеспечения надежной связи в приложении IoT.

Глава демонстрирует комплексный подход к созданию системы с использованием современных технологий и протоколов, обеспечивая надежное управление и сбор данных для отслеживания изменений внешней среды.

4. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

4.1. Функциональное и юзабилити тестирования

В данной работе реализована система сбора данных от датчиков с последующей передачей и получением этих данных через протокол MQTT. Эта система является важной частью системы, поэтому необходимо убедиться в ее надежной и корректной работе. Для этого применяется тестирование, которое помогает выявить и устранить возможные ошибки и дефекты, а также гарантировать соответствие требованиям к функциональности.

Для тестирования работоспособности системы были проведены следующие тесты, представленные в таблице 3.

Таблица 3 – Набор тестов

№	Название теста	Ожидаемый результат	Полученный результат	Тест пройден?
1	Авторизация пользователя с помощью логина и пароля	Пользователь вошел в аккаунт, Пользователь вошел в аккаунт	Пользователь вошел в аккаунт, Пользователь вошел в аккаунт	Да
2	Просмотр данных полученных раннее в программе.	Пользователь просмотрел ранее полученные данные	Программа отобразила все данные.	Да
3	Пользователь может удалить ранее полученные данные датчиков.	Данные очищаются из файлов.	Данные очищены.	Да
4	Пользователь перед запуском конструкции должен подключить блок.	Блок успешно подключился.	Блок успешно подключен и вывел информацию о статусе подключения.	Да
5	Настройка нового проезда по заданной траектории.	Пользователь начертил траекторию проезда сборки.	Новая траектория отображается на экране.	Да
6	Загрузка раннее сохраненной траектории.	Пользователь загрузил раннее начерченную траекторию.	Загрузилась старая траектория.	Да
7	Отображение предпросмотра траекторий при наведении мыши на кнопку.	Отображается красная пунктирная линия, которая указывает направление.	Отобразилась красная пунктирная линия, которая указывает направление.	Да
8	Самостоятельный выбор датчика, показания которого требуется отобразить.	Отображаются показания только выбранных датчиков.	Отображаются показания только выбранных датчиков.	Да

Окончание таблицы 3

№	Название теста	Ожидаемый результат	Полученный результат	Тест пройден?
9	Во время запуска конструкции, мы можем приостановить и остановить движение.	Сборка приостанавливает и продолжит движение.	Сборка приостанавливает и продолжит движение.	Да
10	Отображение показаний датчиков и передвижение по заданной траектории работают не зависимо друг от друга.	Мы можем не передвигаться и принимать значения датчиков.	Показания датчиков и езда работает независимо друг от друга.	Да
11	Построение графиков данных происходит в реальном времени.	При выборе датчика, и включения отображения графика, график строится параллельно получению новых данных.	График строится параллельно получению новых данных.	Да
12	Пользователь может отфильтровать полученные данные по различным критериям.	Пользователь выбирает показания и настраивает показания, которые хочет увидеть.	Программа выводит заданные показания.	Да

Юзабилити тестирование проводилось среди коллег по работе. В ходе тестирования были обнаружены определенные проблемы, которые не были обнаружены в ходе функционального тестирования. Данные ошибки были устранены. При повторном тестировании ошибок не было обнаружено.

После проведения тестирования подтвердилась работоспособность системы сбора и передачи данных от датчиков через протокол MQTT. Все функциональные возможности работают корректно, а система демонстрирует стабильную и надежную работу. Тестирование позволило выявить и устранить потенциальные проблемы.

Вывод по четвертой главе

В данной главе производится описание тестирований реализации приложения. Приведены результаты 12 функциональных тестирований, по результатам которых можно сделать вывод, что система работает стабильно и выполняет все поставленные цели.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано приложение для системы отслеживания изменения состояния внешней среды. Поэтапно были выполнены поставленные задачи.

1. Изучены современные платформы и программные системы управления беспроводными устройствами интернет вещей.
2. Спроектирована архитектура приложения.
3. Реализовано получение данных с датчиков.
4. Реализована сборку и проезд по заданной траектории.
5. Реализовано и протестировано готовое приложение

Приложение было введено в опытную эксплуатацию с подписанием акта о внедрении.

В дальнейшем планируется:

– разработка приложения для смартфонов под управлением Android;

– внедрение в систему искусственного интеллекта для улучшения процесса мониторинга;

– расширение ассортимент подключаемых датчиков;

– внедрение системы в повседневную жизнь для каждого желающего пользователя;

– сокращение расходов на сборку конструкции.

ЛИТЕРАТУРА

1. A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications // IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347-2376, Fourthquarter 2015, doi: 10.1109/COMST.2015.2444095.

2. M5Go. [Электронный ресурс] URL: https://github.com/m5stack/M5GO/blob/master/README.md (дата обращения: 11.02.2024 г.).

3. M5Stack (Micro Python). [Электронный ресурс] URL: https://developer.alexanderklimov.ru/python/m5stack/ (дата обращения: 11.02.2024 г.).

4. Внутренняя библиотека международной школы робототехники и программирования «Лига Роботов». (дата обращения: 01.02.2024 г.).

5. Глушак Е.В. Введение в Интернет вещей: учебное пособие / Е.В. Глушак, А.В. Куприянов. – Самара: Издательство Самарского университета, 2023. – 104 с.

6. Документация m5stack. [Электронный ресурс] URL: https://docs.m5stack.com (дата обращения: 01.02.2024 г.).

7. Документация Micropython. [Электронный ресурс] URL: https://docs.m5stack.com (дата обращения: 01.02.2024 г.).

8. Документация PyQt6. [Электронный ресурс] URL: https://doc.qt.io/qtforpython-6/index.html (дата обращения: 01.02.2024 г.).

9. Документация Python. [Электронный ресурс] URL: https://www.python.org/doc/ (дата обращения: 01.02.2024 г.).

10. Кокунин П.А. Введение в Интернет вещей. Учебное пособие / П.А. Кокунин, И.И. Латыпов, Л.С. Латыпова. Казань: Издательство Казанского университета, 2022. – 147 с.

11. Количество подключений к IoT по всему миру. [Электронный ресурс] URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ (дата обращения: 11.05.2024 г.).

12. Обзор публичных брокеров MQTT. [Электронный ресурс] URL: https://dzen.ru/a/YzR7GRxirQbqC2Mr (дата обращения: 19.02.2024 г.).

13. Объем российского рынка IoT вещей. [Электронный ресурс] URL: https://telecomdaily.ru/news/2024/01/18/obem-rossiyskogo-telekom-rynka-po-itogam-2023-goda-vpervye-prevysit-2-trln-rubley (дата обращения: 10.05.2024 г.).

14. Официальный сайт Boston Dynamics. [Электронный ресурс] URL: https://bostondynamics.com/ (дата обращения: 01.05.2024 г.).

15. Официальный сайт Clearpath Robotics. [Электронный ресурс] URL: https://clearpathrobotics.com/ (дата обращения: 01.05.2024 г.).

16. Официальный сайт T5 Экобот. [Электронный ресурс] URL: https://www.smprobotics.ru/ (дата обращения: 01.05.2024 г.).

17. Среда разработки UiFlow. [Электронный ресурс] URL: https://flow.m5stack.com/ (дата обращения: 19.02.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Фрагменты кода

Листинг 1 – Функция создания боковых кнопок

```
def create_sidebar(self):
    sidebar = QWidget()
    sidebar.setStyleSheet("background-color: #222;")
    sidebar_layout = QVBoxLayout(sidebar)
    sidebar_layout.setAlignment(Qt.AlignmentFlag.AlignTop)
    sidebar_layout.setContentsMargins(20, 20, 20, 20)
    self.profile_button = SidebarButton("", "images/profile.png", par-
ent=sidebar)
    self.profile_button.clicked.connect(self.show_profile_blocked_message)
    sidebar_layout.addWidget(self.profile_button)
    self.greeting_label.setStyleSheet("QLabel { color: white; font-size:
12px; }")
    self.greeting_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
    self.buttons = [
        SidebarButton("Главная", "images/main.png", parent=sidebar),
        SidebarButton("Настройки", "images/settings.png", parent=sidebar),
        SidebarButton("Настройка езды", "images/set_tr.png", parent=side-
bar),
        SidebarButton("Запуск", "images/run.png", parent=sidebar),
        SidebarButton("Данные", "images/data.png", parent=sidebar)
    ]
    self.start_button = self.buttons[3]
    for i, button in enumerate(self.buttons):
        button.clicked.connect(lambda _, index=i: self.on_button_click(in-
dex + 1))
        sidebar_layout.addWidget(button)
    spacer = QSpacerItem(20, 40, QSizePolicy.Policy.Minimum, QSizePol-
icy.Policy.Expanding)
    sidebar_layout.addItem(spacer)
    self.error_label = QLabel("")
    self.error_label.setStyleSheet("QLabel { color: red; font-size: 12px;
}")
    self.error_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
    sidebar_layout.addWidget(self.error_label)
    self.logout_button = SidebarButton("Выход", "images/exit.png", par-
ent=sidebar)
    self.logout_button.clicked.connect(self.logout)
    sidebar_layout.addWidget(self.logout_button)
```

Листинг 2 – Фрагмент кода авторизации пользователя

```
class AuthorizationWidget(QWidget):

    def init_mqtt_client(self):
        self.client = mqtt.Client()
        self.client.username_pw_set(MQTT_USER, MQTT_PASS)
        self.client.tls_set()  # Используем TLS
        self.client.connect(MQTT_HOST, MQTT_PORT)
        self.client.on_connect = self.on_connect
        self.client.on_message = self.on_message
        self.client.loop_start()
    def on_connect(self, client, userdata, flags, rc):
        self.client.subscribe(MQTT_TOPICS[0])

    def on_message(self, client, userdata, msg):
        response = msg.payload.decode()
        if response.lower() == "true":
```

```
            self.show_auth_success()
        elif response.lower() == "false":
            self.show_auth_failure()
        else:
            self.show_invalid_response()

    def on_login_clicked(self):
        login = self.login_edit.text()
        password = self.password_edit.text()

        if not login or not password:
            self.info_label.setText("Заполните все поля")
            self.info_label.setStyleSheet("QLabel { color: red; font-size:
12px; }")
            return

        if re.search('[а-яА-Я]', login) or re.search('[а-яА-Я]', password):
            self.info_label.setText("Используйте только латинские символы")
            self.info_label.setStyleSheet("QLabel { color: red; font-size:
12px; }")
            return

        payload = json.dumps({"login": login, "password": password})
        self.client.publish(MQTT_TOPICS[0], payload)

    def show_auth_success(self):
        self.info_label.setText("Авторизация прошла успешно")
        self.info_label.setStyleSheet("QLabel { color: green; font-size:
12px; }")
        self.auth_success.emit()
        self.authenticated = True

    def show_auth_failure(self):
        self.info_label.setText("Неверный логин или пароль")
        self.info_label.setStyleSheet("QLabel { color: red; font-size:
12px; }")

    def show_invalid_response(self):
        self.info_label.setText("Ожидаем ответ от сервера")
        self.info_label.setStyleSheet("QLabel { color: red; font-size:
12px; }")
```

Листинг 3 – Фрагмент кода модуля настройки езды.

```
class RideSettings(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(300, 300, 800, 600)
        self.step = 20
        self.angle = 0
        self.x = 400
        self.y = 300
        self.start = True
        self.first_line_drawn = False
        self.directions = []
        self.preview_line = None
        self.working_area = QRectF(0, 0, 800, 600)
        self.has_image = False
        self.drawn_items = []
```

```
        self.image_frame = None
        main_layout = QVBoxLayout()
        self.setLayout(main_layout)
        self.drawing_window = QGraphicsView(self)
        self.scene = QGraphicsScene()
        self.drawing_window.setScene(self.scene)
        self.image_item = None
        main_layout.addWidget(self.drawing_window)
        control_panel = QWidget()
        control_panel_layout = QHBoxLayout()
        control_panel.setLayout(control_panel_layout)
        main_layout.addWidget(control_panel)
        line_group = QGroupBox("Параметры движения")
        line_group_layout = QVBoxLayout()
        line_group.setLayout(line_group_layout)
        control_panel_layout.addWidget(line_group)
        line_length_layout = QHBoxLayout()
        line_group_layout.addLayout(line_length_layout)
        self.line_length_label = QLabel("Длина(см):")
        self.line_length_input = QSpinBox()
        self.line_length_input.setRange(1, 1000)
        self.line_length_input.setValue(20)
        line_length_layout.addWidget(self.line_length_label)
        line_length_layout.addWidget(self.line_length_input)
        angle_layout = QHBoxLayout()
        line_group_layout.addLayout(angle_layout)
        self.angle_label = QLabel("Угол:")
        self.angle_input = QSpinBox()
        self.angle_input.setRange(0, 360)
        self.angle_input.setValue(0)
        angle_layout.addWidget(self.angle_label)
        angle_layout.addWidget(self.angle_input)
        draw_group = QGroupBox("Траектория")
        draw_group_layout = QVBoxLayout()
        draw_group.setLayout(draw_group_layout)
        control_panel_layout.addWidget(draw_group)
        self.button_up = QPushButton("Вперед")
        self.button_down = QPushButton("Назад")
        self.button_left = QPushButton("Влево")
        self.button_right = QPushButton("Вправо")
        self.button_custom = QPushButton("Под углом")
        self.button_clear_drawing = QPushButton("Очистить траектории")
        self.button_remove_last_arrow = QPushButton("Удалить последнюю тра-
екторию")
        draw_group_layout.addWidget(self.button_up)
        draw_group_layout.addWidget(self.button_down)
        draw_group_layout.addWidget(self.button_left)
        draw_group_layout.addWidget(self.button_right)
        draw_group_layout.addWidget(self.button_custom)
        draw_group_layout.addWidget(self.button_clear_drawing)
        draw_group_layout.addWidget(self.button_remove_last_arrow)
        image_group = QGroupBox("Изображение")
        image_group_layout = QVBoxLayout()
        image_group.setLayout(image_group_layout)
        control_panel_layout.addWidget(image_group)
        self.button_load_image = QPushButton("Загрузить")
        self.button_clear_image = QPushButton("Очистить")
        self.button_rotate_image = QPushButton("Повернуть на 90°")
        image_group_layout.addWidget(self.button_load_image)
        image_group_layout.addWidget(self.button_clear_image)
        image_group_layout.addWidget(self.button_rotate_image)
        save_load_group = QGroupBox("Направления")
        save_load_group_layout = QVBoxLayout()
```

```
        save_load_group.setLayout(save_load_group_layout)
        control_panel_layout.addWidget(save_load_group)
        self.button_save_directions = QPushButton("Сохранить направления")
        self.button_load_directions = QPushButton("Загрузить направления")
        save_load_group_layout.addWidget(self.button_save_directions)
        save_load_group_layout.addWidget(self.button_load_directions)
        self.directions_text = QTextEdit()
        self.directions_text.setReadOnly(True)
        self.directions_text.setMaximumHeight(50)
        main_layout.addWidget(self.directions_text)
        self.button_up.clicked.connect(lambda: self.draw_and_update_direc-
tions('up'))
        self.button_down.clicked.connect(lambda: self.draw_and_update_di-
rections('down'))
        self.button_left.clicked.connect(lambda: self.draw_and_update_di-
rections('left'))
        self.button_right.clicked.connect(lambda: self.draw_and_update_di-
rections('right'))
        self.button_custom.clicked.connect(self.draw_line_with_angle)
        self.button_load_image.clicked.connect(self.load_image)
        self.button_clear_image.clicked.connect(self.clear_image)
        self.button_clear_drawing.clicked.connect(self.clear_drawing)
        self.button_save_directions.clicked.connect(self.save_directions)
        self.button_load_directions.clicked.connect(self.load_directions)
      self.button_remove_last_arrow.clicked.connect(self.remove_last_arrow)
        self.button_rotate_image.clicked.connect(self.rotate_image)
        self.button_up.enterEvent = lambda event: self.show_preview('up')
        self.button_up.leaveEvent = lambda event: self.remove_preview()
        self.button_down.enterEvent = lambda event: self.show_pre-
view('down')
        self.button_down.leaveEvent = lambda event: self.remove_preview()
        self.button_left.enterEvent = lambda event: self.show_pre-
view('left')
        self.button_left.leaveEvent = lambda event: self.remove_preview()
        self.button_right.enterEvent = lambda event: self.show_pre-
view('right')
        self.button_right.leaveEvent = lambda event: self.remove_preview()
        self.button_custom.enterEvent = lambda event: self.show_pre-
view('custom', self.angle_input.value())
        self.button_custom.leaveEvent = lambda event: self.remove_preview()
       self.line_length_input.valueChanged.connect(self.update_line_length)
        self.angle_input.valueChanged.connect(self.update_angle)
        class DirectionPreview(QGraphicsLineItem):
        def __init__(self, x1, y1, x2, y2, parent=None):
            super(RideSettings.DirectionPreview, self).__init__(x1, y1, x2,
y2, parent)
            pen = QPen(Qt.GlobalColor.red, 1, Qt.PenStyle.DashLine)
            self.setPen(pen)
    def draw_square(self, x, y, size=10):
        square = self.scene.addRect(x - size / 2, y - size / 2, size, size,
QPen(Qt.GlobalColor.black),
                                    QBrush(Qt.GlobalColor.black))
        self.drawn_items.append(square)
        self.update_scene_rect()
    def draw_arrow(self, x1, y1, x2, y2):
        line = QLineF(x1, y1, x2, y2)
        arrow = QGraphicsLineItem(line)
        angle = math.atan2(-line.dy(), line.dx())
        arrow_length = 10
        arrow_head = QPolygonF([
            line.p2(),
```

```
            line.p2() - QPointF(math.sin(angle + math.pi / 3) * ar-
row_length,
                                math.cos(angle + math.pi / 3) * ar-
row_length),
            line.p2() - QPointF(math.sin(angle + math.pi - math.pi / 3) *
arrow_length,
                                math.cos(angle + math.pi - math.pi / 3) *
arrow_length)
        ])
        arrow_head_item = self.scene.addPolygon(arrow_head, QPen(Qt.Glob-
alColor.black),                                          QBrush(Qt.Glob-
alColor.black))
        self.scene.addItem(arrow)
        self.drawn_items.append(arrow)
        self.drawn_items.append(arrow_head_item)
        self.first_line_drawn = True
        self.update_scene_rect()
    def draw_direction(self, direction, angle=None):
        if self.start:
            self.draw_square(self.x, self.y)
            self.start = False
        drew = False
        if not self.has_image:
            if direction == 'up':
                self.draw_arrow(self.x, self.y, self.x, self.y - self.step)
                self.y -= self.step
                drew = True
            elif direction == 'down':
                self.draw_arrow(self.x, self.y, self.x, self.y + self.step)
                self.y += self.step
                drew = True
            elif direction == 'left':
                self.draw_arrow(self.x, self.y, self.x - self.step, self.y)
                self.x -= self.step
                drew = True
            elif direction == 'right':
                self.draw_arrow(self.x, self.y, self.x + self.step, self.y)
                self.x += self.step
                drew = True
            elif direction == 'custom' and angle is not None:
                angle_rad = math.radians(-angle + 90)
                x_end = self.x + self.step * math.cos(angle_rad)
                y_end = self.y - self.step * math.sin(angle_rad)
                self.draw_arrow(self.x, self.y, x_end, y_end)
                self.x = x_end
                self.y = y_end
                drew = True
        else:
            if direction == 'up':
                new_y = self.y - self.step
                if self.working_area.contains(self.x, new_y):
                    self.draw_arrow(self.x, self.y, self.x, new_y)
                    self.y = new_y
                    drew = True
            elif direction == 'down':
                new_y = self.y + self.step
                if self.working_area.contains(self.x, new_y):
                    self.draw_arrow(self.x, self.y, self.x, new_y)
                    self.y = new_y
                    drew = True
            elif direction == 'left':
                new_x = self.x - self.step
```

```
                if self.working_area.contains(new_x, self.y):
                    self.draw_arrow(self.x, self.y, new_x, self.y)
                    self.x = new_x
                    drew = True
            elif direction == 'right':
                new_x = self.x + self.step
                if self.working_area.contains(new_x, self.y):
                    self.draw_arrow(self.x, self.y, new_x, self.y)
                    self.x = new_x
                    drew = True
            elif direction == 'custom' and angle is not None:
                angle_rad = math.radians(-angle + 90)
                x_end = self.x + self.step * math.cos(angle_rad)
                y_end = self.y - self.step * math.sin(angle_rad)
                if self.working_area.contains(x_end, y_end):
                    self.draw_arrow(self.x, self.y, x_end, y_end)
                    self.x = x_end
                    self.y = y_end
                    drew = True
        if drew:
            self.directions.append(direction)
            self.update_scene_rect()
    def show_preview(self, direction, angle=None):
        self.remove_preview()
        if direction == 'up':
            new_y = self.y - self.step
            if self.working_area.contains(self.x, new_y) or not
self.has_image:
                self.preview_line = self.DirectionPreview(self.x, self.y,
self.x, new_y)
        elif direction == 'down':
            new_y = self.y + self.step
            if self.working_area.contains(self.x, new_y) or not
self.has_image:
                self.preview_line = self.DirectionPreview(self.x, self.y,
self.x, new_y)
        elif direction == 'left':
            new_x = self.x - self.step
            if self.working_area.contains(new_x, self.y) or not
self.has_image:
                self.preview_line = self.DirectionPreview(self.x, self.y,
new_x, self.y)
        elif direction == 'right':
            new_x = self.x + self.step
            if self.working_area.contains(new_x, self.y) or not
self.has_image:
                self.preview_line = self.DirectionPreview(self.x, self.y,
new_x, self.y)
        elif direction == 'custom' and angle is not None:
            angle_rad = math.radians(-angle + 90)
            x_end = self.x + self.step * math.cos(angle_rad)
            y_end = self.y - self.step * math.sin(angle_rad)
            if self.working_area.contains(x_end, y_end) or not self.has_im-
age:
                self.preview_line = self.DirectionPreview(self.x, self.y,
x_end, y_end)
        if self.preview_line:
            self.scene.addItem(self.preview_line)
    def remove_preview(self):
        if self.preview_line:
            self.scene.removeItem(self.preview_line)
```

```
    def load_background_image(self, file_path):
        if self.has_image:
            QMessageBox.warning(self, "Ошибка", "Изображение уже загружено.
Очистите текущее изображение перед загрузкой нового.")
            return
        pixmap = QPixmap(file_path)
        self.image_item = self.scene.addPixmap(pixmap)
        self.image_item.setZValue(-1)
self.image_item.setTransformOriginPoint(self.image_item.boundingRect().cen-
ter())
        self.working_area = self.image_item.boundingRect()
        self.has_image = True
        self.draw_image_frame()
        image_center = self.working_area.center()
        self.image_item.setPos(image_center - self.image_item.bound-
ingRect().center())
        self.update_scene_rect()
    def clear_background_image(self):
        if self.image_item:
            self.scene.removeItem(self.image_item)
            self.image_item = None
            self.has_image = False
            self.working_area = QRectF(0, 0, self.width(), self.height())
            self.remove_image_frame()
            self.update_scene_rect()
    def draw_image_frame(self):
        if self.image_frame:
            self.scene.removeItem(self.image_frame)
        pen = QPen(Qt.GlobalColor.blue, 2)
        self.image_frame = self.scene.addRect(self.working_area, pen)
        self.update_scene_rect()
    def remove_image_frame(self):
        if self.image_frame:
            self.scene.removeItem(self.image_frame)
            self.image_frame = None
            self.update_scene_rect()
    def rotate_image(self):
        if self.image_item:
            self.image_item.setRotation(self.image_item.rotation() + 90)
            self.working_area = self.image_item.mapRectToScene(self.im-
age_item.boundingRect())
            self.draw_image_frame()
            self.update_scene_rect()
    def mousePressEvent(self, event):
        if event.button() == Qt.MouseButton.LeftButton:
            scene_pos = self.drawing_window.mapToScene(event.pos())
            if not self.first_line_drawn:
                # Remove old points
                self.clear_drawing()
                self.x = scene_pos.x()
                self.y = scene_pos.y()
                self.directions = []
                self.draw_square(self.x, self.y)
                self.start = False
                self.update_scene_rect()
    def clear_drawing(self):
        for item in self.scene.items():
            if item != self.image_item and item != self.image_frame:
                self.scene.removeItem(item)
        self.drawn_items = []
        self.start = True
        self.first_line_drawn = False
```

```
        self.x = 400
        self.y = 300
        self.directions = []
        self.update_directions_text()
        self.update_scene_rect()
    def remove_last_arrow(self):
        if len(self.drawn_items) >= 2:
            arrow_head = self.drawn_items.pop()
            self.scene.removeItem(arrow_head)
            arrow_line = self.drawn_items.pop()
            self.scene.removeItem(arrow_line)
            if self.directions:
                last_direction = self.directions.pop()
                if last_direction == 'up':
                    self.y += self.step
                elif last_direction == 'down':
                    self.y -= self.step
                elif last_direction == 'left':
                    self.x += self.step
                elif last_direction == 'right':
                    self.x -= self.step
                elif last_direction == 'custom':
                    angle_rad = math.radians(-self.angle + 90)
                    self.x -= self.step * math.cos(angle_rad)
                    self.y += self.step * math.sin(angle_rad)
        self.update_directions_text()
        self.update_scene_rect()
    def draw_line_with_angle(self):
        angle = self.angle_input.value()
        self.draw_direction('custom', angle)
        self.update_directions_text()
    def load_image(self):
        file_dialog = QFileDialog()
        file_path, _ = file_dialog.getOpenFileName(self, "Выбрать изображе-
ние", "", "Images (*.png *.jpg *.jpeg)")
        if file_path:
            self.load_background_image(file_path)
    def clear_image(self):
        self.clear_background_image()
    def update_line_length(self):
        length = self.line_length_input.value()
        self.step = length
    def update_angle(self):
        angle = self.angle_input.value()
        self.angle = angle
    def draw_and_update_directions(self, direction):
        self.draw_direction(direction)
        self.update_directions_text()
    def update_directions_text(self):
        directions_str = ""
        for direction in self.directions:
            if direction == "custom":
                directions_str += f"custom_{self.line_length_in-
put.value()}_{self.angle_input.value()};"
            else:
                directions_str += f"{direction}_{self.line_length_in-
put.value()}_0;"
        self.directions_text.setPlainText(directions_str)
    def save_directions(self):
        file_dialog = QFileDialog()
        file_path, _ = file_dialog.getSaveFileName(self, "Сохранить направ-
ления", "", "Text files (*.txt)")
```

```
        if file_path:
            try:
                with open(file_path, 'w') as f:
                    for direction in self.directions:
                        if direction == "custom":
f.write(f"custom_{self.line_length_input.value()}_{self.angle_in-
put.value()};\n")
                        else:                           f.write(f"{direc-
tion}_{self.line_length_input.value()}_0;\n")
            except Exception as e:
                QMessageBox.warning(self, "Ошибка", f"Не удалось сохранить
направления: {e}")

    def load_directions(self):
        file_dialog = QFileDialog()
        file_path, _ = file_dialog.getOpenFileName(self, "Загрузить направ-
ления", "", "Text files (*.txt)")
        if file_path:
            try:
                with open(file_path, 'r') as f:
                    loaded_directions = f.read().splitlines()
                self.clear_drawing()
                for direction_str in loaded_directions:
                    if ";" in direction_str:
                        direction, length, angle = direction_str[:-
1].split("_")
                        self.line_length_input.setValue(int(length))
                        self.update_line_length()
                        if direction == 'custom':
                            self.angle_input.setValue(int(angle))
                            self.update_angle()
                            self.draw_direction(direction, int(angle))
                        else:
                            self.draw_direction(direction)
                self.update_directions_text()
            except Exception as e:
                QMessageBox.warning(self, "Ошибка", f"Не удалось загрузить
направления: {e}")
    def update_scene_rect(self):
        self.scene.setSceneRect(self.scene.itemsBoundingRect())
```

Листинг 4 - Взаимодействие с датчиками

```
class SensorWidget(QWidget):
    data_signal = pyqtSignal(str)
    def __init__(self, sensor_name, sensor_icon, mqtt_topic, logout_topic,
logout_message, response_topic):
        super().__init__()
        self.sensor_name = sensor_name
        self.mqtt_topic = mqtt_topic
        self.logout_topic = logout_topic
        self.logout_message = logout_message
        self.response_topic = response_topic
        self.is_active = False
        layout = QVBoxLayout()
        self.label = QLabel(sensor_name)
        self.label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.label.setFont(QFont("Arial", 16, QFont.Weight.Bold))
        self.label.setStyleSheet("QLabel { color: #333; }")
        self.label.mousePressEvent = self.toggle_sensor
        self.icon = QLabel()
```

```
        icon_pixmap = QPixmap(sensor_icon).scaled(100, 100, Qt.AspectRati-
oMode.KeepAspectRatio, Qt.TransformationMode.SmoothTransformation)
        self.icon.setPixmap(icon_pixmap)
        self.icon.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.data = QLabel("Данные: ---")
self.data = QLabel("Данные: ---")
        self.data.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.data.setStyleSheet("QLabel { color: #666; }")
        self.data.mousePressEvent = self.show_graph
        layout.addWidget(self.label)
        layout.addWidget(self.icon)
        layout.addWidget(self.data)
        self.setLayout(layout)
        self.set_default_style()
        self.graph_window = SensorGraphWindow(self.sensor_name)
        self.data_signal.connect(self.graph_window.update_graph)
        client.subscribe(self.response_topic)
        client.message_callback_add(self.response_topic, self.on_response)
    def set_default_style(self):
        self.setStyleSheet(
            """
            QWidget {
                border: 1px solid #d3d3d3;
                border-radius: 10px;
                background-color: #fff;
            }
    def toggle_sensor(self, event):
        if self.is_active:
            client.publish(self.logout_topic, f"off_{self.logout_message}")
            self.set_default_style()
            client.unsubscribe(self.mqtt_topic)
            self.is_active = False
        else:
            client.publish(self.logout_topic, f"on_{self.logout_message}")
            self.set_active_style()
            client.subscribe(self.mqtt_topic)
            client.message_callback_add(self.mqtt_topic, self.update_data)
            self.is_active = True
    def update_data(self, client, userdata, msg):
        data = msg.payload.decode()
        self.data.setText(f"Данные: {data}")
        self.data_signal.emit(data)
        save_sensor_data(self.sensor_name, data)
    def show_graph(self, event):
        self.graph_window.show()
    def on_response(self, client, userdata, msg):
        response = msg.payload.decode()
        if response == "On":
            self.set_active_style()
            client.subscribe(self.mqtt_topic)
            client.message_callback_add(self.mqtt_topic, self.update_data)
            self.is_active = True
        elif response == "Off":
            self.set_default_style()
            cl
```

Приложение Б. Скриншоты приложения

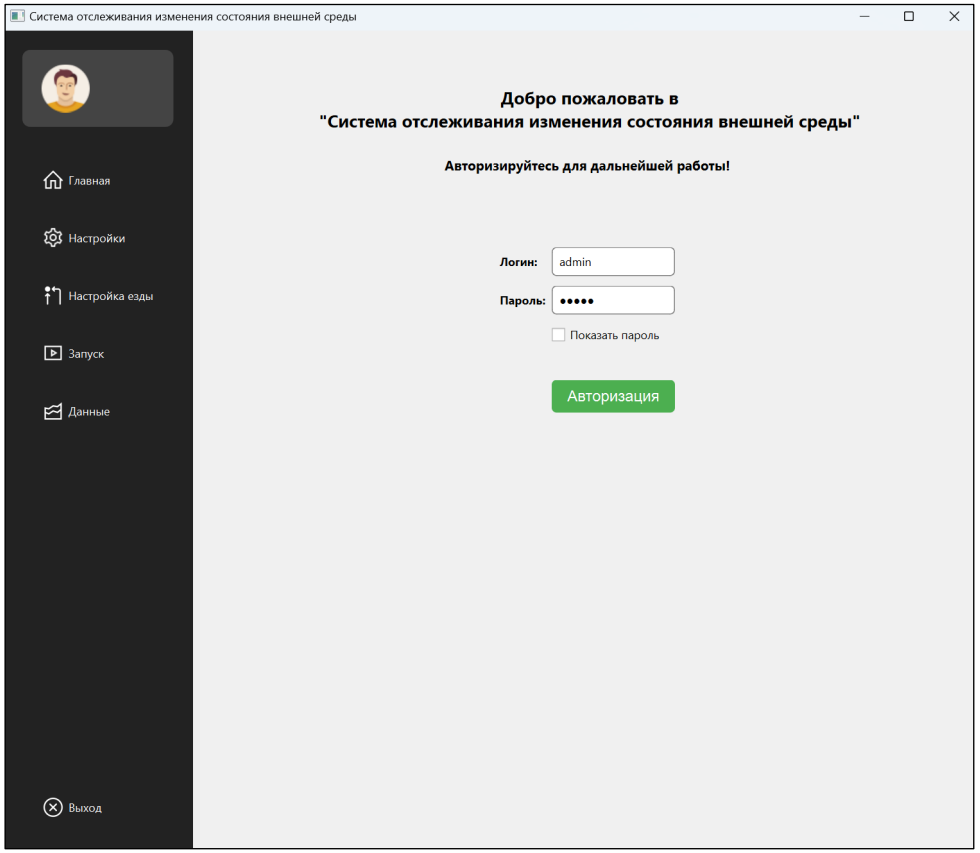

Рисунок 1 – Окно авторизации пользователя

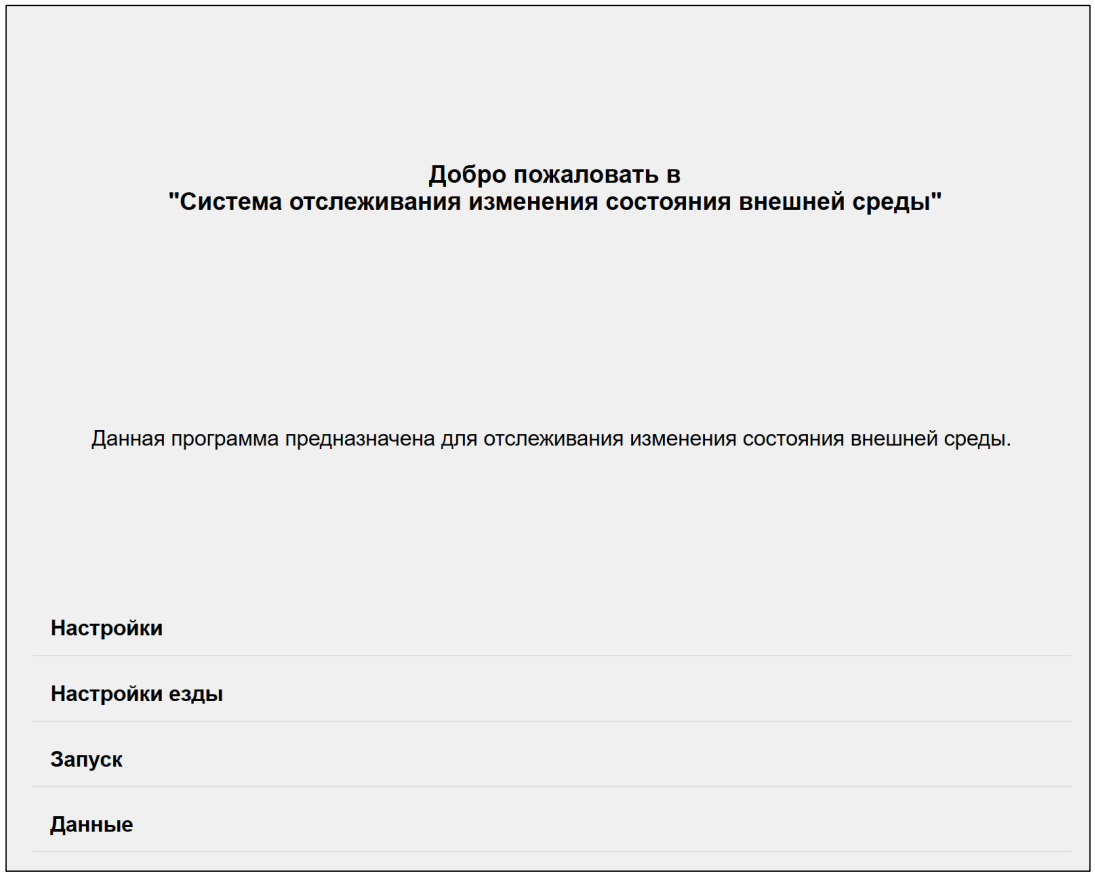

Рисунок 2 – Главное окно

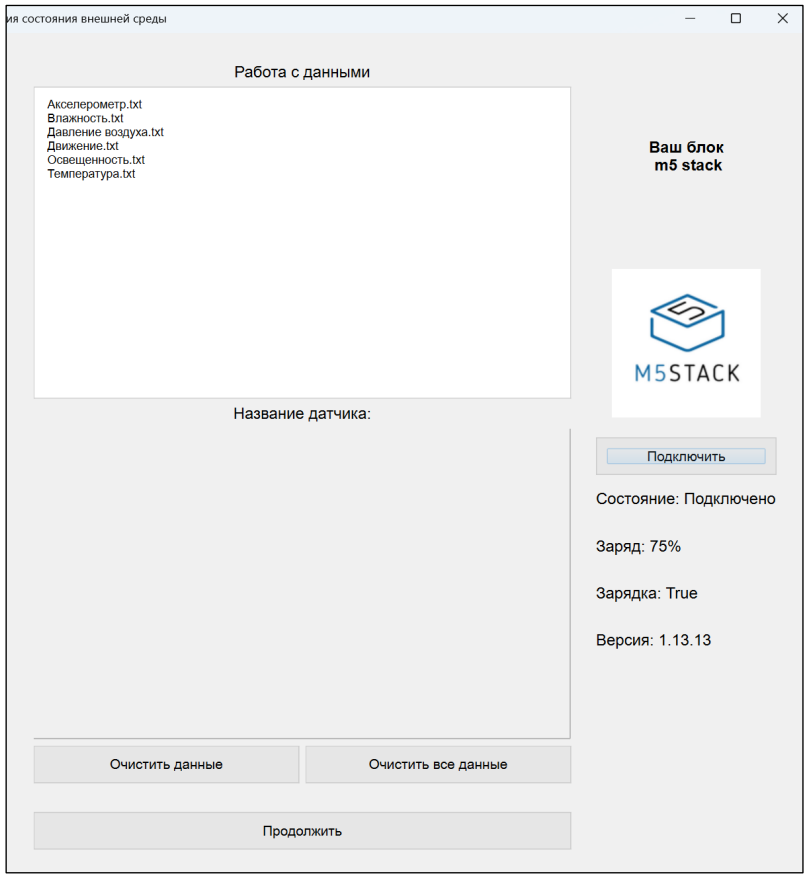

Рисунок 3 – Окно настройки

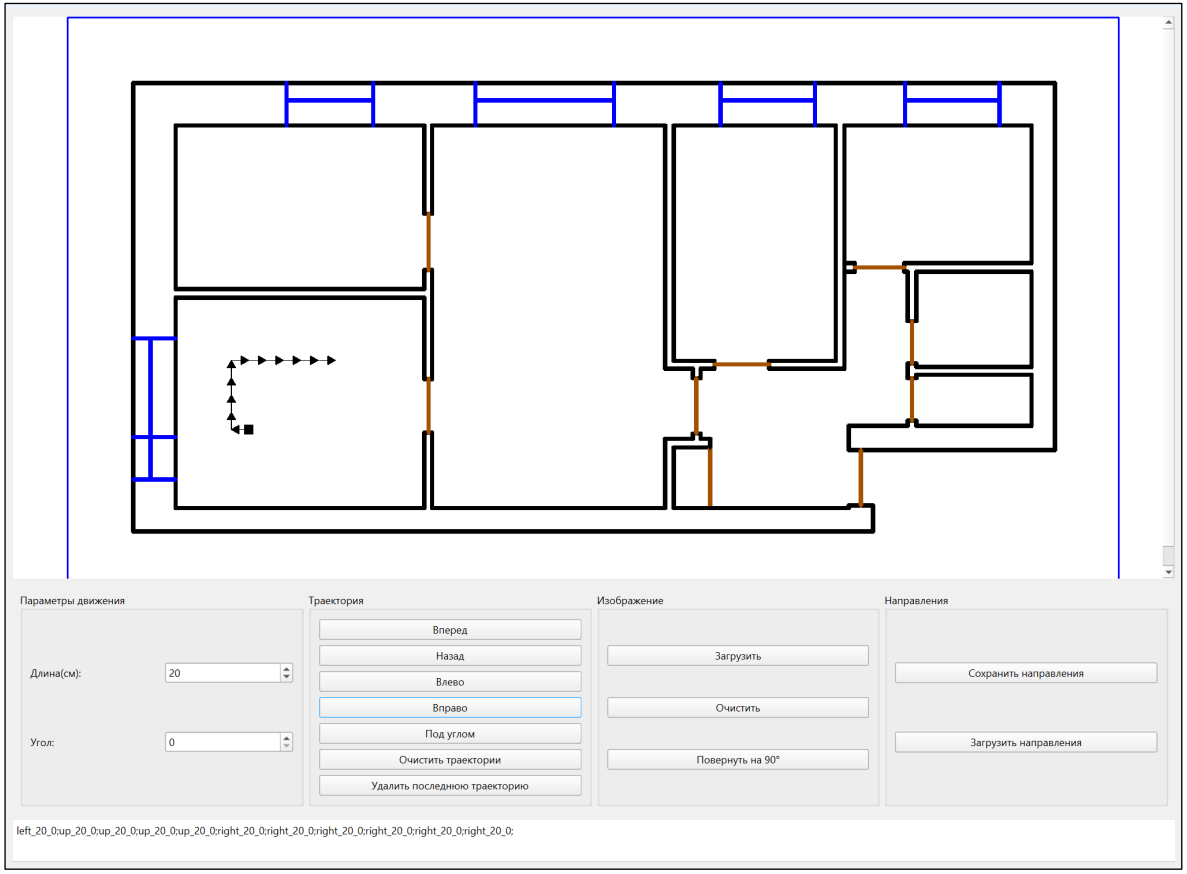

Рисунок 4 – Окно настройки движения

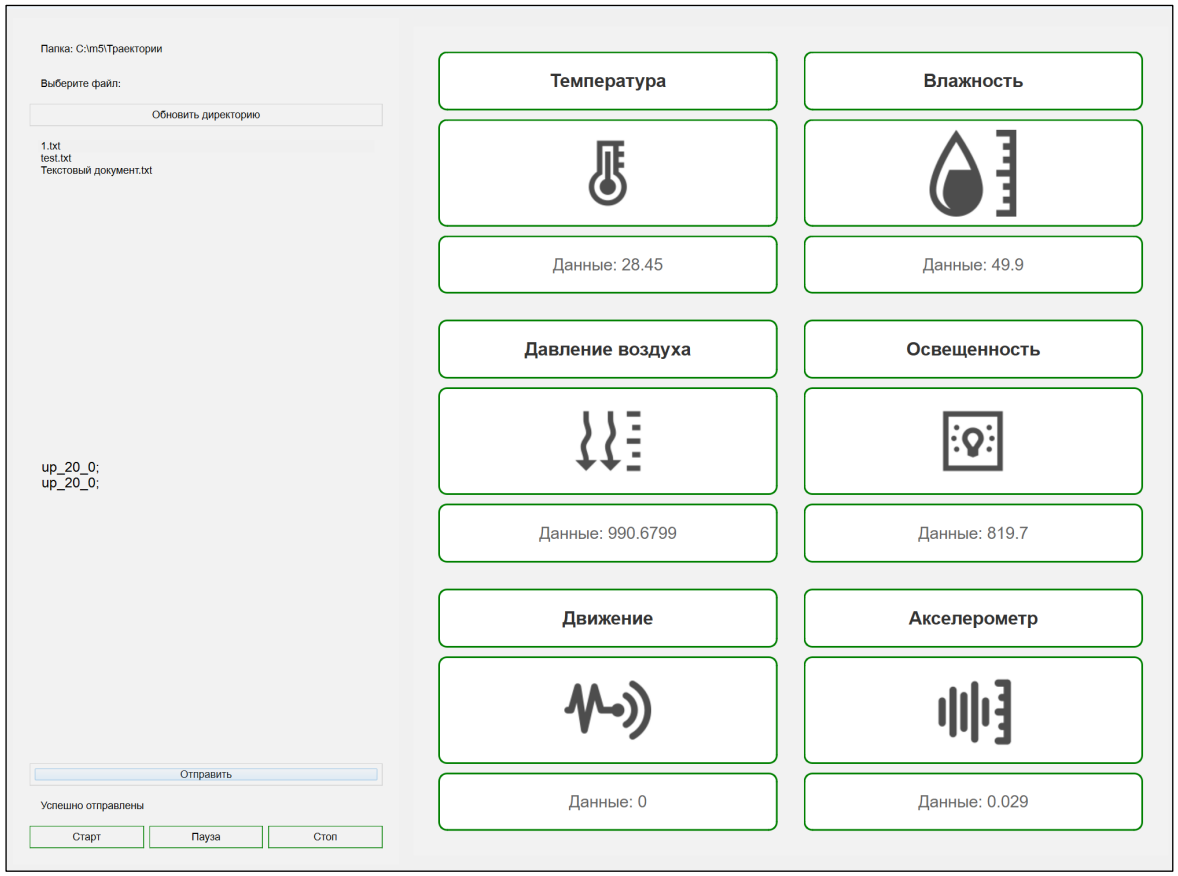

Рисунок 5 – Окно запуска работы

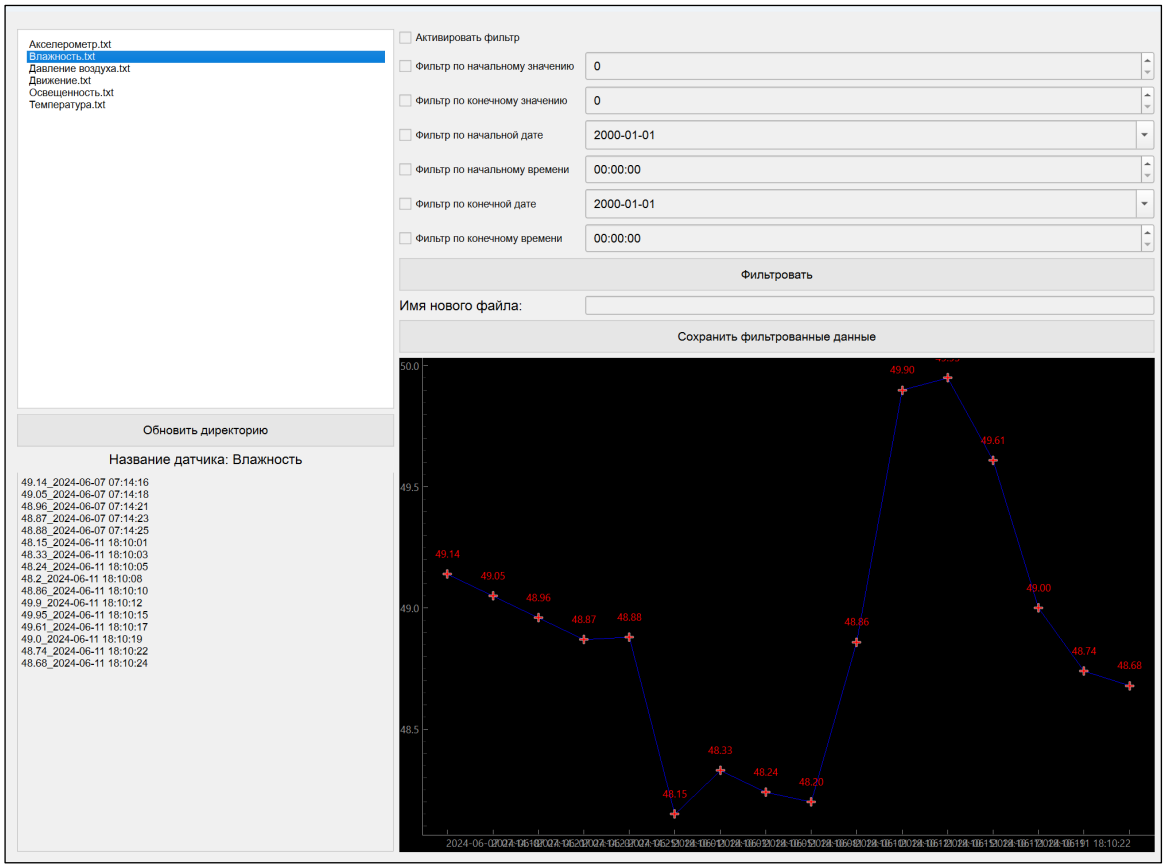

Рисунок 6 – Окно работы с данными