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 настольного приложения для расчета маршрута сельскохозяйственного дрона по имеющимся характеристикам

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-348.ВКР

Научный руководитель,
профессор кафедры СП, д.ф.-м.н.,
доцент
_____ Т.А. Макаровских

Автор работы,
студент группы КЭ-403
_____ Е.В. Ращупкин

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-403
Ращупкину Евгению Владимировичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка настольного приложения для расчета маршрута сельскохозяйственного дрона по имеющимся характеристикам.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Makarovskikh T., Panyukov A., Abotaleb M., Maksimova V., Dernova O., Raschupkin E. Optimal Route for Drone for Monitoring of Crop Yields. // Olenev N., Evtushenko Y., Jaćimović M., Khachay M., Malkova V. (eds) Advances in Optimization and Applications. OPTIMA 2023. Communications in Computer and Information Science, Springer, Cham, 2023. – №1913. – 228–240 pp.

3.2. The Rust Programming Language. [Электронный ресурс] URL: https://doc.rust-lang.org/stable/book/ (дата обращения: 11.02.2024 г.).

3.3. Tauri guides. [Электронный ресурс] URL: https://tauri.app/v1/guides/ (дата обращения: 11.02.2024 г.).

3.4. «Геоскан» – беспилотные технологии. [Электронный ресурс] URL: https://www.geoscan.aero/ru/products/geoscan401 (дата обращения: 11.02.2024 г.).

4. Перечень подлежащих разработке вопросов

4.1. Изучить методы разделения сельскохозяйственного поля на сетку ячеек, расчета маршрута следования дрона по центрам этих ячеек. Методы учета характеристик дрона вносящих корректировки в маршрут.

4.2. Привести описание требований к разрабатываемому продукту на основе диаграмм вариантов использования UML.

4.3. Спроектировать структуру приложения и разработать необходимые модули для ее функционирования, связать модули для работы с графическим интерфейсом программы.

4.4. Протестировать возможности системы подав на вход реальные данные, координаты поля и характеристики дрона, сравнить результаты с ожидаемыми.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
профессор кафедры СП, д.ф.-м.н., доцент Т.А. Макаровских

Задание принял к исполнению Е.В. Ращупкин

ГЛОССАРИЙ

1. *UAV (Unmanned Aerial Vehicle)* – беспилотный летательный аппарат, управляемый дистанционно или автономно посредством встроенных систем управления [1].
2. *Валидация* – процесс проверки данных на соответствие заданным критериям. В контексте программирования, валидация обычно включает проверку корректности и полноты введенных пользователем данных [2].
3. *Фронтенд* – часть программной системы, которая взаимодействует с пользователем. Включает в себя интерфейс пользователя и логику, обеспечивающую его функционирование [3].
4. *Бэкенд* – часть программной системы, которая выполняет основную обработку данных и не взаимодействует напрямую с пользователем. Включает в себя серверные компоненты, базы данных и другие системы [4].
5. *Десктоп* – термин, обычно используемый для описания приложений, разработанных для работы на персональных компьютерах или рабочих станциях [5].
6. *Кроссплатформенность* – свойство программного продукта, которое позволяет ему работать на разных операционных системах или устройствах без необходимости внесения изменений в исходный код [6].
7. *Фреймворк* – набор библиотек и инструментов, которые упрощают разработку программного обеспечения, предоставляя структуру и набор стандартных функций [7].
8. *Методы* – в программировании, это определенные блоки кода, которые выполняют конкретные задачи. Методы используются для организации кода и повторного использования функций в различных частях программы [8].
9. *GeoJSON* – это формат для кодирования различных структур географических данных [9].
10. *WGS 84* – система координат для определения геопространственной информации [10].

11. *Задача коммивояжера* –задача комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные точки хотя бы по одному разу с последующим возвратом в исходный город [11].

12. *Алгоритм ближайшего соседа в задаче коммивояжера* – один из простейших эвристических алгоритмов решения задачи коммивояжера [12].

13. *Алгоритм полного перебора в задаче коммивояжера* – алгоритм решения задачи коммивояжера включающий в себя перебор всех возможных путей для нахождения оптимального пути [13].

14. *Евклидова метрика* – расстояние между двумя точками евклидова пространства, вычисляемое по теореме Пифагора [14].

15. *Остовное дерево* – это дерево, подграф данного графа, с тем же числом вершин, что и у исходного графа [15].

16. *Минимальное остовное дерево* – это остовное дерево графа, имеющее минимальный возможный вес, где под весом дерева понимается сумма весов входящих в него ребер [16].

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

Актуальность работы определяется растущим интересом к применению беспилотных летательных аппаратов (БПЛА) в сельском хозяйстве. В частности, дроны используются для мониторинга урожая, а также для создания точных карт полей.

Научно-технологический прорыв в сельскохозяйственном производстве невозможен без применения цифровых технологий точного земледелия (ТЗ), являющегося ключевым сегментом «умного сельского хозяйства». Направление точного земледелия – многопрофильное и характеризуется комплексностью и сложностью научных, инженерных, агрономических и организационных задач. Существующие методы и инструменты точного сельского хозяйства позволили передовым странам довольно быстро перевести сельское хозяйство на инновационный путь развития [1]. Одной их технологий, используемых для мониторинга состояния сельскохозяйственных угодий, является аэрофотосъемка при помощи БПЛА. Организация такой съемки зачастую осуществляется специализированными организациями и требует определенных вложений от заказчика. В частности, необходимо оплатить вызов бригады для осуществления съемки, а также амортизацию используемого оборудования (например, каждую перезарядку аккумуляторной батареи БПЛА).

С учетом этого, разработка десктопного приложения, способного оптимизировать маршруты полетов дронов на основе их характеристик, представляется актуальной и востребованной задачей. Данное приложение может помочь сельскохозяйственным предприятиям повысить эффективность использования БПЛА, уменьшить затраты на выезд специалистов за счет сокращения времени полета и, как следствие, оптимизации ресурса батареи дрона.

Постановка задачи

Целью выпускной квалификационной работы является разработка настольного приложения для расчета маршрута сельскохозяйственного дрона по имеющимся характеристикам.

Для достижения поставленной цели необходимо решить следующие задачи:

1) выполнить анализ предметной области и произвести обзор существующих решений;
2) разработать архитектуру приложения;
3) описать алгоритмы, использующиеся в системе;
4) выполнить реализацию приложения;
5) выполнить тестирование.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения и списка литературы. Объем работы составляет 76 страниц, объем списка литературы – 15 источников.

В первой главе осуществляется анализ предметной области и сопоставление существующих решений, что позволяет определить ключевые требования и задачи для разработки приложения. Рассматривается спектр инструментов, аналогичных проектов и технологий, что выявляет потенциальные возможности для улучшения функциональности и удобства использования. Также в этой главе представлены основные принципы и подходы, которые будут использоваться при создании нового приложения, включая открытый исходный код и возможность интеграции с другими системами.

Во второй главе проводится анализ требований к программной системе, включая как функциональные, так и нефункциональные аспекты. Определяются ключевые характеристики, которые должна обеспечивать система, а также устанавливаются ограничения для ее корректной работы. Формируется диаграмма вариантов использования и спецификация вариантов использования.

Третья глава посвящена описанию архитектуры разрабатываемой системы. В ней представлены основные компоненты и сервисы, их функциональность и взаимодействие. Описана структура базы данных для хранения данных. Также в главе приведена диаграмма деятельности, иллюстрирующая последовательность процесса работы с системой.

В четвертой главе обоснован выбор алгоритмов для построения маршрута, использующихся в системе, и приведено их пошаговое описание.

В пятой главе приведена реализация ключевых компонентов системы, пользовательского интерфейса, функций для взаимодействия с базой данных и логики алгоритмов, а также проводится тестирование системы.

В приложении А содержатся спецификации вариантов использования.

В приложении Б содержатся функции расчета сервиса Algorithms.

В приложении В содержатся рисунки интерфейса приложения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область проекта

Предметная область проекта связана с разработкой десктопного приложения, предназначенного для расчета маршрута полета сельскохозяйственного дрона. Основной задачей приложения является определение оптимального маршрута полета, учитывающего имеющиеся характеристики дрона и заданные параметры полета.

Анализ аналогичных проектов и существующих решений для реализации проекта

В рамках разработки десктопного приложения для расчета маршрута сельскохозяйственного дрона по имеющимся характеристикам, необходимо реализовать возможность планировки маршрута для фотографирования посевов полей. Эти данные, после планировки маршрута, уже передаются в другое приложение для склейки фотографий поверхности и, при необходимости, анализа. Важным аспектом при планировке маршрута будет учет размеров поля и наличие препятствий на его территории. Кроме того, приложение должно иметь возможность визуализировать маршрут на карте и предоставлять информацию о параметрах полета, таких как высота, длительность полета и т.д. Все эти функции должны быть реализованы в удобном и интуитивно понятном интерфейсе, который позволит пользователям быстро и эффективно планировать маршруты для фотографирования посевов полей.

Существует множество инструментов для планирования полетов дронов и обработки полученных данных, которые могут быть полезны при разработке десктопного приложения для расчета маршрута сельскохозяйственного дрона по имеющимся характеристикам. Рассмотрим некоторые из них.

DroneDeploy [2] – это один из наиболее распространенных инструментов для планирования полетов дронов и обработки данных. С помощью этого приложения пользователь может задавать параметры полета, такие

как высоту, скорость, угол наклона камеры и другие, а также задавать целевые точки на карте, которые дрон должен посетить. Приложение автоматически рассчитывает оптимальный маршрут полета и предоставляет возможность обработки полученных данных, например, создание карт и 3D-моделей местности.

Litchi [3] – еще один инструмент для планирования полетов дронов. Он позволяет пользователю задавать различные параметры полета, а также создавать специальные миссии, включающие несколько точек на карте, которые дрон должен посетить в определенном порядке. Litchi также предоставляет возможность записи видео и фото во время полета, а также позволяет управлять камерой дрона в режиме реального времени.

Pix4D Capture [4] – специализированный инструмент для создания карт и 3D-моделей с помощью дронов. Приложение позволяет задавать различные параметры полета, такие как высоту, скорость и угол наклона камеры, а также задавать целевые точки на карте. После полета Pix4D Capture обрабатывает полученные данные и создает точные 3D-модели местности.

UgCS [5] – это инструмент, который позволяет пользователю управлять не только дронами, но и другими беспилотными системами. Пользователь может задавать различные параметры полета, такие как скорость, высота полета и угол наклона камеры, а также задавать целевые точки на карте. Приложение автоматически рассчитывает оптимальный маршрут полета и позволяет обрабатывать полученные данные.

Для сравнительного анализа возможностей описанных продуктов можно привести таблицу, в которой будут указаны такие критерии сравнения как, планирование маршрута полета, управление полетом дрона, обработка полученных данных, визуализация карты, наличие ограниченного ряда поддерживаемых дронов, добавление собственных дронов, поддерживаемые ОС, лицензия, стоимость (таблица 1).

Таблица 1 – Сравнительный анализ приложений

Возможность	**DroneDeploy**	**Litchi**	**Pix4D Capture**	**UgCS**
Планирование маршрута полета	Да	Да	Да	Да
Управление полетом дрона	Да	Да	Да	Да
Обработка полученных данных	Да	Да	Да	Да
Визуализация карты	Да	Да	Да	Да
Ограниченный ряд поддерживаемых дронов	Да	Да	Да	Да
Добавление собственных дронов	Ограничение функционала	Ограничение функционала	Нет	Ограничение функционала
Поддержка ОС	iOS, Android + Windows, macOS, Linux	iOS, Android + Windows, macOS, Linux	iOS, Android	Windows, macOS, Linux, Android
Лицензия	Проприетарная	Проприетарная	Проприетарная	Проприетарная
Стоимость	$149+/месяц	$25	Бесплатно	€790+ или €149+/месяц

Рассмотрев аналогичные проекты и существующие решения для реализации проекта, можно заметить, что многие из них имеют проприетарную лицензию и не предоставляют возможность добавления собственных дронов. Кроме того, некоторые из них могут быть слишком дорогими для малого бизнеса, к тому же, стоит отметить, что все анализируемые аналоги приложения разработаны зарубежными компаниями. В свою очередь, наше приложение будет предоставлять открытый исходный код, что позволит пользователям настраивать и дополнять его функциональность. В приложении будет возможность добавления собственных дронов, что поможет людям с моделями дронов от малоизвестных компаний. Кроме того, распространение приложения будет бесплатным, что обеспечит его доступность и удобство использования для малых предприятий в сельском хозяйстве.

В целом, все эти решения могут быть полезны при разработке десктопного приложения для расчета маршрута сельскохозяйственного дрона по имеющимся характеристикам. Также необходимо обеспечить возможность

интеграции с другими системами, например, с системами отправки маршрута на дрон или экспорта в уже существующие форматы представления координат.

1.2. Особенности реализации

Для реализации десктопного приложения для расчета маршрута сельскохозяйственного дрона по имеющимся характеристикам, было принято решение использовать веб технологии. Такой подход позволяет обеспечить кроссплатформенность приложения и увеличить его доступность для пользователей.

Для реализации приложения будет использоваться Tauri [6]. Он выбран из-за своей высокой производительности и возможности создания кроссплатформенных десктопных приложений на основе веб-технологий. Tauri, обеспечивает быстрое и эффективное взаимодействие между ядром приложения и веб-интерфейсом.

Для обработки данных будет использоваться язык программирования Rust [7]. Rust позволяет обеспечить высокую производительность и безопасность при обработке данных, что критически важно для такого типа приложений. Более того, Rust имеет большую и быстрорастущую экосистему, что обеспечивает доступность и готовность библиотек и инструментов, необходимых для реализации конкретных функциональных требований приложения.

В качестве фреймворка для разработки пользовательского интерфейса будет использоваться SvelteKit [8]. Этот инструмент позволяет создавать эффективные и быстрые веб-приложения с помощью компиляции кода во время сборки и минимизации размера бандла приложения.

Для визуализации карты будет использована OpenLayers [9] – библиотека для работы с картами. Она предоставляет широкий спектр функциональности для работы с картами, таких как отображение маршрутов, меток,

векторных слоев и т.д. Это позволит создать интерактивную карту, на которой можно будет планировать маршруты для фотографирования посевов полей. Для ограничения поля можно использовать полигон, а для мест съемки точки.

Таким образом, использование Tauri, Rust, SvelteKit и OpenLayers обеспечивает высокую производительность, безопасность, готовность библиотек и инструментов, а также удобную и эффективную визуализацию карты. Все эти особенности сделают приложение более доступным для пользователей, а также обеспечат надежность в работе и обеспечивает простоту разработки.

Вывод по первой главе

Анализ предметной области и существующих работ по тематике выпускной квалификационной работы показал, что было бы полезно разработать приложение, предоставляющее возможность расчета маршрута сельскохозяйственного дрона по имеющимся характеристикам для мониторинга полей. Также был выявлен набор инструментальных средств для реализации поставленной задачи, наиболее полно удовлетворяющий требованиям к подобного рода системам. Было принято решение реализовать систему на основе на базе технологий Tauri, Rust, SvelteKit и OpenLayers.

2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОЙ СИСТЕМЕ

2.1. Основные требования

В результате анализа предметной области и обзора существующих аналогов были сформированы следующие два основных типа требований:

– функциональные требования – перечень сервисов, которые должна выполнять система;

– нефункциональные требования – описание характеристики системы и ее окружения, также содержит перечень ограничений.

Функциональные требования к проектируемой системе

1. Пользователь должен иметь возможность добавлять, просматривать, изменять и удалять записи о дронах и камерах в базе данных. Через пользовательский интерфейс системы.

2. Система должна иметь возможность строить маршрут на основе заданных характеристик дрона, камеры, координат старта съемки, координат границ, указываемых на карте и процента перекрытия полученных снимков.

3. Система должна иметь возможность отображать маршрут на карте.

Нефункциональные требования к проектируемой системе

1. Система должна использовать язык программирования Rust.

2. Система должна проверять корректность вводимых данных.

2.1. Величина полезной нагрузки дрона лежит в пределах: от 100 грамм до 10 кг.

2.2. Скорость полета дрона находится в пределах от 0.1 м/с до 50 м/с.

2.3. Продолжительность полета дрона находится в пределах от 1 минуты до 8 часов.

2.4. Минимальная и максимальная высота полета находятся в пределах от 2 метров до 5 км. При этом, максимальная высота полета больше минимальной.

2.5. Камера, установленная на дроне либо отсутствует, либо принадлежит списку доступных камер.

2.6. Масса камеры должна быть меньше 10 килограмм.

2.7. Углы обзора камеры по x и y должны быть меньше 180 градусов.

2.8. Разрешение съемки камеры по x и y – целочисленные.

3. Система должна использовать веб технологии такие как Tauri, Svelte, TypeScript для визуализации интерфейса и библиотеку OpenLayers для отображения карты и маршрута.

2.2. Диаграмма вариантов использования

На основе требований, предъявляемых к разрабатываемому приложению, были разработаны варианты его использования, которые представлены на рисунке 1.

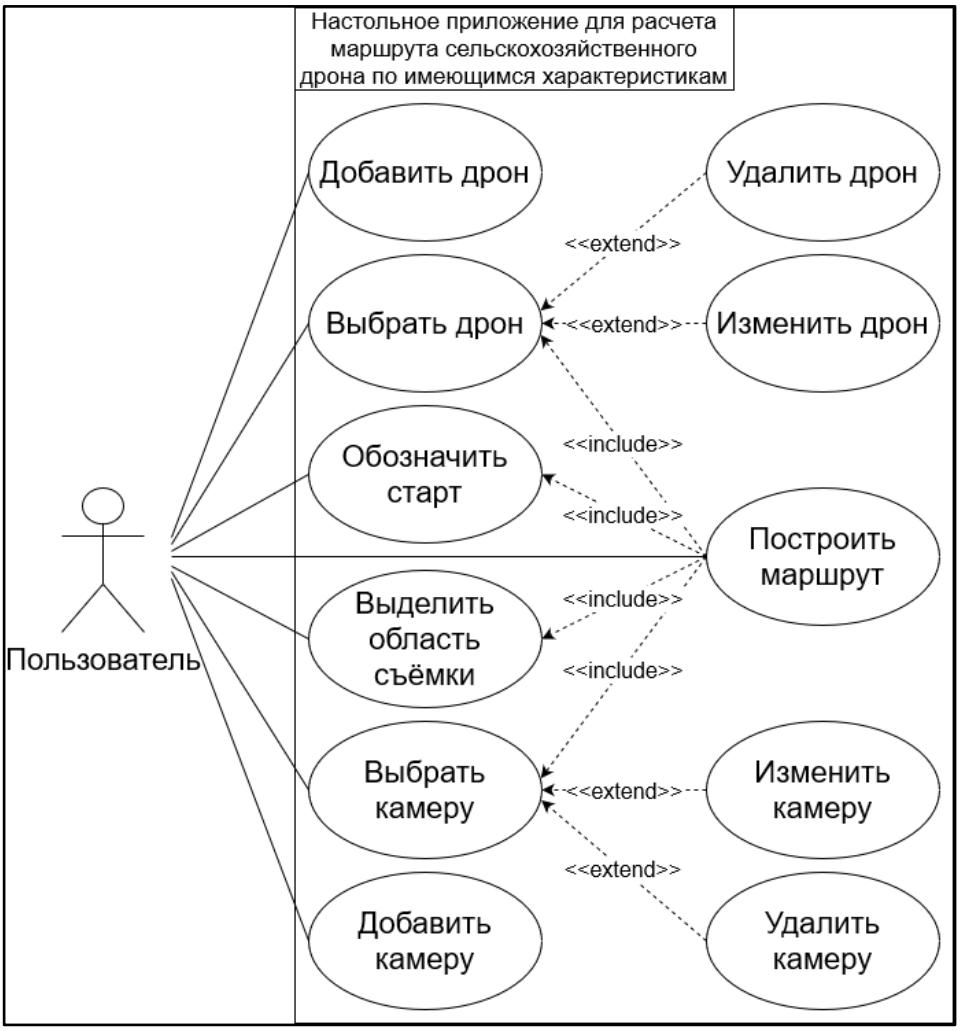

Рисунок 1 – Диаграмма вариантов использования

Актером является пользователь приложения, которому доступна возможность использовать весь функционал приложения.

Пользователь может совершать следующие действия:

1) добавить дрон – пользователь может добавить новый дрон с его характеристиками;

2) выбрать дрон – пользователь может выбрать один из добавленных дронов для расчета маршрута или редактирования;

3) удалить дрон – пользователь может удалить выбранный дрон;

4) изменить дрон – пользователь может изменить характеристики выбранного дрона;

5) обозначить старт – пользователь может задать стартовую точку маршрута;

6) выделить область съемки – пользователь может указать область, которую требуется снять с помощью дрона;

7) выбрать камеру – пользователь может выбрать камеру, которая будет установлена на дроне для расчетов или редактирования;

8) изменить камеру – пользователь может изменить характеристики выбранной камеры;

9) удалить камеру – пользователь может удалить выбранную камеру;

10) добавить камеру – пользователь может добавить новую камеру с ее характеристиками;

11) построить маршрут – пользователь может рассчитать маршрут дрона на основе введенных ранее данных.

Спецификации ВИ представлены в таблицах 1–11 приложения А.

Вывод по второй главе

В результате анализа предметной области и обзора существующих аналогов были сформированы функциональные и нефункциональные требования. Основываясь на требованиях, предъявляемых к разрабатываемому приложению, была разработана диаграмма вариантов использования приложения и их спецификации.

3. АРХИТЕКТУРА СИСТЕМЫ

3.1. Общее описание архитектуры системы

На рисунке 2 изображена диаграмма компонентов системы для планирования маршрутов сельскохозяйственных дронов. Система состоит из следующих сервисов и компонентов:

- сервис расчета маршрутов (Algorithms);
- сервис управления данными (Data Managment);
- компонент пользовательского интерфейса карты (Map);
- компонент пользовательского интерфейса левого меню (LMenu);
- компонент пользовательского интерфейса правого меню (RMenu), в состав которого входит сервис вызова функций расчета (Calculate);
- компонент базы данных (SQLite Database).

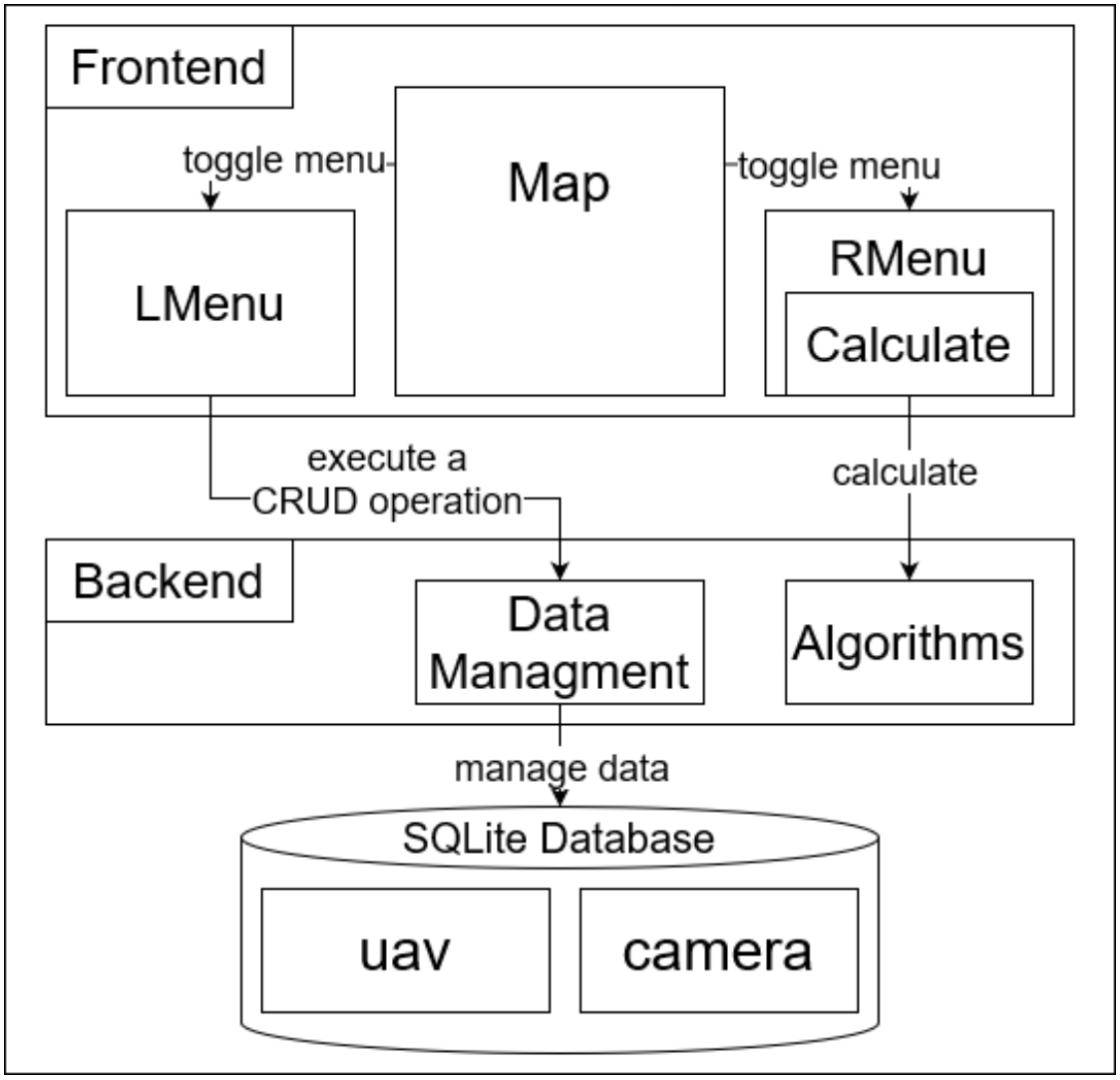

Рисунок 2 – Компоненты системы

Система включает в себя компонент базы данных, который содержит данные о дронах и камерах. Сервис расчета маршрутов использует данные, предоставленные пользователем через компоненты интерфейса. Сервис управления данными осуществляет взаимодействие с базой данных для создания, чтения, обновления и удаления данных о дронах и камерах. Компоненты пользовательского интерфейса взаимодействуют с сервисами для отправки данных, получения результатов расчетов и отображения информации пользователю.

3.2. Описание компонентов и сервисов, составляющих систему

Map – центральное меню с картой, компонент пользовательского интерфейса в его функционал входят:

- отображение карты и маршрута;
- переключение отображения правого и левого меню.

LMenu – этот компонент позволяет пользователю работать с дронами и камерами, он предлагает следующий набор функций:

- выбор дрона для построения маршрута;
- CRUD операции над дроном;
- выбор камеры для построения маршрута;
- CRUD операции над камерой.

RMenu – этот компонент позволяет пользователю управлять параметрами миссии, он предоставляет пользователю данный функционал:

- ввод высоты полета;
- выбор алгоритма расчета;
- панель инструментов для рисования маршрута;
- кнопка вызова Calculate;
- отображение текущих параметров миссии;
- экспорт маршрута в формат GeoJSON.

Calculate – часть сервиса Map он включает в себя:

– валидацию данных, он проверяет корректность ввода всех необходимых данных;

– отправку данных на бэкенд для расчетов;

– обновление отображаемого маршрута на карте.

Data Managment – этот сервис взаимодействует с базой данных SQLite, управляя данными о дронах и камерах. Он выполняет операции CRUD на основе команд с фронтенда.

Algorithms – это сервис расчета маршрута, он отвечает за расчет маршрута дрона. Детальное описание алгоритмов представлено в главе 4. Этот сервис обладает следующим набором возможностей:

– выполнение дискретизация области;

– выполнение алгоритма ближайшего соседа;

– выполнение алгоритма полного перебора;

– выполнение алгоритма прямоугольных областей;

– отправка результатов на фронтенд.

SQLite Database – база данных, является частью инфраструктуры системы, в которой находятся 2 таблицы `uav` и `camera` для хранения данных о дронах и камерах, хранится локально в файле mydatabase.db.

3.3. Модель базы данных

На рисунке 3 представлена модель базы данных приложения для планирования маршрутов сельскохозяйственных дронов. Она состоит из двух таблиц – `uav`, отвечающей за дроны и `camera` отвечающей за камеры. Эти таблицы содержат информацию о различных дронах и камерах, которые могут использоваться в миссиях. Связь между таблицами один к одному. Связь указывает, какая камера установлена на дроне. При этом, камера может быть не установлена на дрон.

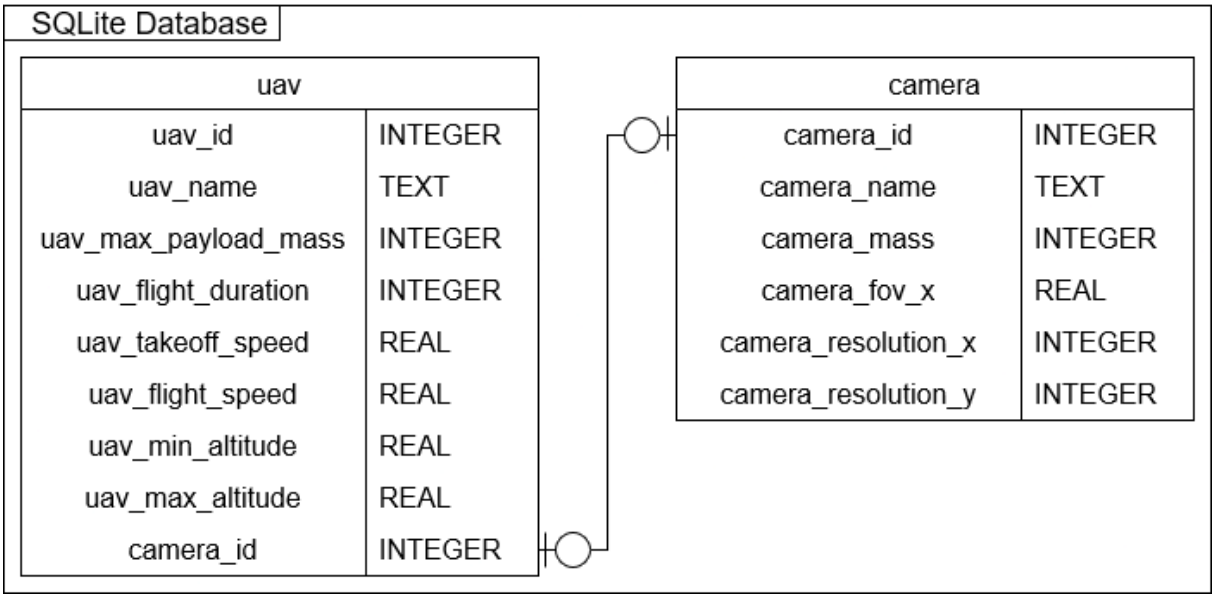

Рисунок 3 – Модель базы данных

Таблица `uav` – представляет собой экземпляр дрона и включает в себя следующие атрибуты:

– `uav_id` – идентификатор дрона;

– `uav_name` – название дрона;

– `uav_max_payload_mass` – максимальная полезная нагрузка дрона в граммах;

– `uav_flight_duration` – средняя продолжительность полета в секундах;

– `uav_takeoff_speed` – средняя скорость взлета в метрах в секунду;

– `uav_flight_speed` – средняя скорость полета в метрах в секунду;

– `uav_min_altitude` – минимальная безопасная высота полета в метрах;

– `uav_max_altitude` – максимальная безопасная высота полета в метрах;

– `camera_id` – идентификатор камеры, установленной на дрона, может быть не указан.

Таблица `camera` – представляет собой экземпляр камеры и включает в себя следующие атрибуты:

- `camera_id` – идентификатор камеры;
- `camera_name` – название камеры;
- `camera_mass` – масса камеры в граммах;
- `camera_fov_x` – угол обзора камеры по оси x в градусах;
- `camera_resolution_x` – разрешение камеры по оси x в пикселях;
- `camera_resolution_y` – разрешение камеры по оси x в пикселях.

3.4. Процесс работы с системой

На рисунке 4 приведена диаграмма деятельности, которая подробно описывает процесс работы с системой планирования маршрутов для сельскохозяйственных дронов.

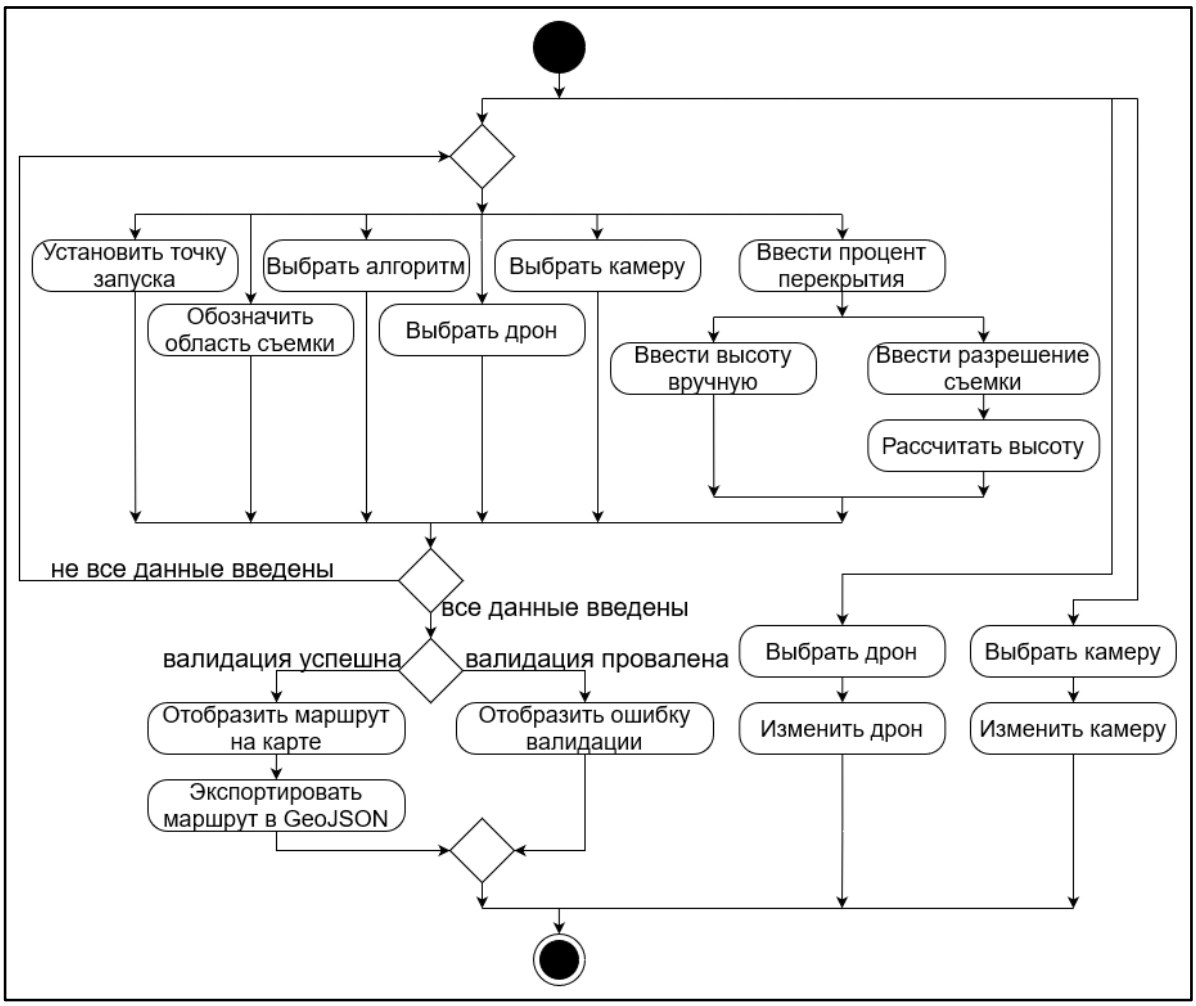

Рисунок 4 – Диаграмма деятельности

Процесс делится на 3 основных потока. Первый начинается одного из шести действий. Пользователь может установить начальную точку маршрута, обозначить область съемки, которую необходимо заснять, может выбрать алгоритм, модель дрона и камеры из списка доступных в системе (алгоритм ближайшего соседа, алгоритм полного перебора, алгоритм, предназначенный для прямоугольных областей [13]), установить процент перекрытия и определить высоту полета дрона. Здесь у пользователя есть два варианта: ввести высоту вручную или рассчитать ее. Во втором случае пользователю необходимо ввести разрешение съемки, после чего система сама рассчитывает высоту полета. После того как все параметры заданы, система производит расчет маршрута. Если все данные корректны, и валидация прошла успешно, система отображает маршрут на карте и предлагает пользователю экспортировать его в формате GeoJSON для дальнейшего использования. GeoJSON [10] является форматом данных, основанным на JSON, и используется для представления простых географических объектов. GeoJSON файл, который экспортирует данное приложение включает в себя последовательность точек с координатами в формате WGS84, которые должен посетить дрон. Если в процессе валидации обнаруживаются ошибки, система отображает сообщение об ошибке, давая пользователю возможность исправить введенные данные.

В то же время, независимо от основного процесса планирования маршрута, пользователь может выбрать и изменить характеристики модели дрона или камеры.

Вывод по третьей главе

В третьей главе была описана архитектура системы. Система включает в себя различные компоненты и сервисы, которые взаимодействуют для обеспечения планирования маршрутов. Была представлена модель базы данных, состоящая из двух таблиц, содержащих информацию о дронах и камерах. Также был описан процесс работы с системой, включающий в себя различные этапы от выбора параметров миссии до экспорта маршрута.

4. ТЕОРЕТИЧЕСКОЕ ОПИСАНИЕ АЛГОРИТМОВ

4.1. Описание задачи и алгоритмов

Не существует эффективных алгоритмов для решения задачи коммивояжера. Задача коммивояжера формулируется так, бродячий торговец (коммивояжер) должен посетить n пунктов. Известна стоимость проезда между любыми двумя пунктами. Требуется выбрать наиболее «дешевый» замкнутый путь, проходящий через все пункты. Эта задача является классической задачей дискретной оптимизации и относится к классу NP-сложных задач. [11]

Если рассматривать эту задачу в контексте данной работы, можно провести соответствие, пунктами являются точки съемки дрона, а стоимость выражена в расстоянии между пунктами. Данное приложение предназначено для применения некоторых алгоритмов, представляющих приближенные решения или оптимальное решение для построения маршрута сельскохозяйственного дрона.

Задачу построения маршрута дрона можно рассматривать, как задачу на плоскости, потому как дрон летает на высоте достаточной, чтобы ему не препятствовали деревья и потому как основной целью съемки являются поля, отсутствуют горы. Поэтому процесс расчета маршрута поделить на несколько основных этапов:

– дискретизация области съемки для получения точек, на карте, в которых будет произведена съемка;

– построение маршрута на точках, полученных в результате дискретизации.

В качестве алгоритмов построения маршрутов были выбраны алгоритмы ближайшего соседа (Nearest neighbour) [12], полного перебора (Brute force) и алгоритм, включающий обработку прямоугольных областей (Rectangular areas). Сравнение алгоритмов представлено в таблице 2.

Таблица 2 – Сравнительный анализ алгоритмов построения маршрута

Алгоритм	**Nearest neighbour**	**Brute force**	**Rectangular areas**
Входные данные	Одномерный массив точек	Одномерный массив точек	Трехмерный массив точек, стартовая точка, направление дискретизации
Выходные данные	Одномерный массив точек	Одномерный массив точек	Одномерный массив точек
Класс	Аппроксимационный алгоритм	Алгоритм оптимального решения	Аппроксимационный алгоритм
Ограничения	Нет	Количество точек съемки <13	Количество полей <10, прямоугольники, ограничивающие поля не пересекаются
Результат	Неоптимальный	Оптимальный	Неоптимальный, входят точки, не принадлежащие полю
Время работы алгоритма	$O(N^2)$	O(N!)	Объединение полей O(ElogV), построение пути внутри поля O(N)

Из таблицы 2 видно, что при различных входных данных лучше использовать тот или иной алгоритм, наиболее подходящий под нужды. Алгоритм ближайшего соседа может быть применен для большого количества точек, потому как его скорость выполнения очень высока, но его результат не является оптимальным. Алгоритм полного перебора применим лишь для случаев с минимальным количеством точек, потому как на его расчеты требуется несоизмеримо большое количество времени. Алгоритм для расчета прямоугольных областей может применен для большого количества точек и является достаточно точным для нужд построения маршрута для съемки сельскохозяйственным дроном, но его особенностью является то, что он применим лишь для непересекающихся прямоугольных областей. Для реализации данных алгоритмов, необходимо привести их высокоуровневое описание. Далее будут рассмотрены шаги каждого из алгоритмов подробнее.

4.2. Описание алгоритмов

Алгоритм ближайшего соседа

Входные данные:

– points: вектор пар чисел с плавающей точкой, представляющих координаты точек;

– start_point: пара чисел с плавающей точкой, представляющая координаты начальной точки.

Выходные данные: вектор пар чисел с плавающей точкой, представляющих координаты точек в порядке обхода ближайшего соседа.

Шаги алгоритма.

1. Проверить, не пуст ли вектор points. Если пуст, вернуть ошибку.
2. Инициализировать remaining_points как копию points, result как пустой вектор, и current_point как start_point.
3. Пока remaining_points не пуст, выполнить следующие шаги.

3.1. Найти индекс и значение ближайшей точки к current_point в remaining_points.

3.2. Удалить ближайшую точку из remaining_points.

3.3. Если result пуст, добавить start_point в result.

3.4. Добавить ближайшую точку в result.

3.5. Обновить current_point как ближайшую точку.

4. Добавить start_point в result.
5. Вернуть result как результат.

Алгоритм полного перебора

Входные данные:

– points: вектор пар чисел с плавающей точкой, представляющих координаты точек;

– start_point: пара чисел с плавающей точкой, представляющая координаты начальной точки.

Выходные данные: вектор пар чисел с плавающей точкой, представляющих координаты точек в порядке обхода ближайшего соседа.

Шаги алгоритма.

1. Создать пустой вектор best_path для хранения лучшего пути и переменную best_distance для хранения минимального расстояния. Инициализировать best_distance максимальным значением.

2. Вызвать рекурсивную функцию brute_force, передав в нее вектор точек, начальную точку, текущий путь (сначала только с начальной точкой), текущее расстояние (сначала 0) и ссылки на best_path и best_distance.

2.1. Если вектор точек пуст, вычислить общее расстояние, добавив расстояние от последней точки в текущем пути до начальной точки. Если общее расстояние меньше best_distance, обновить best_path и best_distance.

2.2. Если вектор точек не пуст, перебрать все точки. Для каждой точки создать новый путь, добавив эту точку к текущему пути, и вычислить новое расстояние. Если новое расстояние меньше best_distance, продолжить поиск с новым путем и новым расстоянием.

3. Вернуть best_path как результат функции.

Алгоритм дискретизации

Входные данные:

– photo_width: ширина фотографии;

– photo_height: высота фотографии;

– direction_degrees: направление в градусах;

– check_inside: проверка, находятся ли точки внутри полигона.

Выходные данные: трехмерный вектор кортежей, представляющих несколько дискретизированных областей.

Шаги алгоритма.

1. Преобразовать направление из градусов в радианы.

2. Инициализировать вектор для хранения результатов для каждого полигона.

3. Для каждого полигона выполнить.

3.1. Инициализировать минимальные и максимальные значения x и y до крайних противоположностей.

3.2. Применить преобразование к каждой точке.

3.3. Пройти через координаты полигона, чтобы найти минимальные и максимальные значения x и y.

3.4. Проверить, находится ли точка внутри полигона.

3.5. Инициализировать вектор результата.

3.6. Рассчитать половину ширины и высоты камеры.

3.7. Рассчитать количество фотографий по ширине и высоте.

3.8. Для каждой фотографии по ширине и высоте выполнить.

3.8.1. Если check_inside установлен на true, то рассчитать углы прямоугольника, проверить, находится ли какой-либо угол прямоугольника внутри полигона и, если какой-либо угол находится внутри, рассчитать центр прямоугольника и добавить его в результат.

3.8.2. В противном случае, рассчитать центр прямоугольника и добавить его в результат.

3.9. Добавить результат в результаты.

4. Вернуть результаты.

Алгоритм прямоугольных областей

Алгоритм прямоугольных областей состоит из трех основных этапов. В него входит алгоритм построения маршрута на прямоугольной области, алгоритм нахождение минимального остовного дерева и алгоритм связывания прямоугольных областей [13].

Алгоритм построения маршрута на прямоугольной области предназначен для построения эффективного маршрута внутри прямоугольника, представляющего поле. Он описывает нахождение маршрута с использованием прямоугольных областей и состоит из несколько этапов.

Нахождение минимального остовного дерева, используемый в алгоритме связывания областей, основан на использовании алгоритм Борувки.

Алгоритм связывания прямоугольных областей предназначен для объединения маршрутов, полученных на этапе расчета маршрутов в прямоугольных областях.

Алгоритм построения маршрута на прямоугольной области

Входные данные: region_points: двумерный вектор, представляющий прямоугольную, дискретизированную область.

Выходные данные: одномерный вектор, представляющий последовательность точек, описывающих оптимальный замкнутый маршрут на прямоугольной области.

Шаги алгоритма.

1. Проверить, является ли входной вектор прямоугольным (имеет ли он одинаковую длину всех своих строк). Если это не так, вернуть сообщение об ошибке.

2. Если ширина вектора меньше 2, то все точки вектора собираются в один регион, который добавляется в общий результат.

3. Для векторов с шириной больше или равной 2 происходит следующее.

3.1. Собрать точки с нижней строки вектора.

3.2. Собрать точки с правого столбца вектора.

3.3. Собрать точки, образующие рампы внутри региона, это делается путем прохода по внутренним столбцам с определенным шагом.

3.4. Если ширина вектора нечетная, обработать сегменты coils, собирая точки по сегментам.

3.5. В случае, если высота четная, добавить точки region_points[1][1] и region_points[0][1], чтобы завершить цепочку.

4. Сформировать результирующий регион, содержащий точки, собранные в предыдущих шагах.

5. Полученные регионы добавляются в общий результат.

Результат работы алгоритма в базовом случае представлена на рисунке 5, частный случай маршрута на прямоугольной области с четной высотой представлен на рисунке 6.

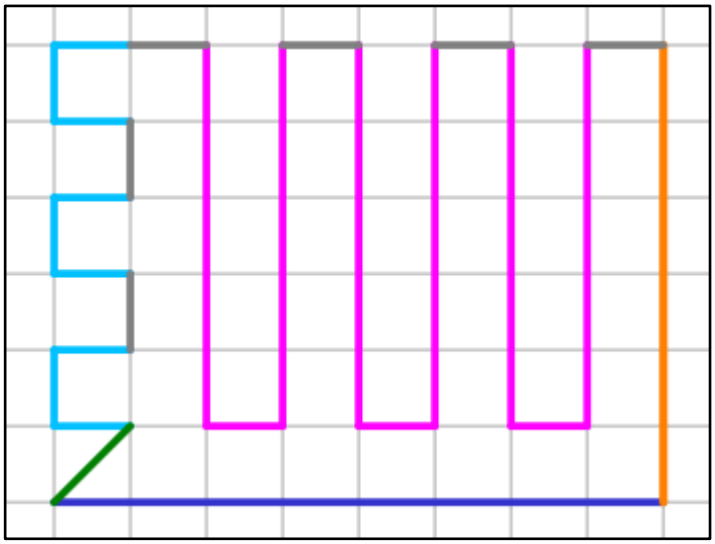

Рисунок 5 – Базовый случай маршрута на прямоугольной области

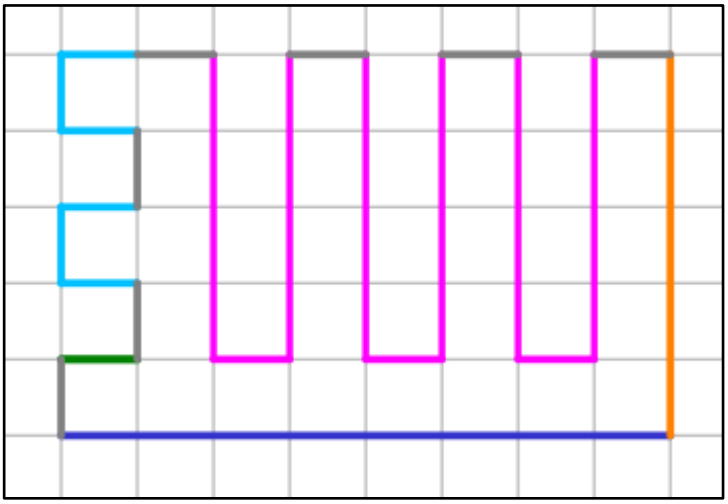

Рисунок 6 – Частный с случай маршрута на прямоугольной области с четной высотой

Алгоритм Борувки для построения минимального остовного дерева

Входные данные: weights: взвешенный связный граф G с вершинами V и ребрами E.

Выходные данные: одномерный вектор, представляющий последовательность точек, описывающих оптимальный замкнутый маршрут на прямоугольной области.

Шаги алгоритма.

1. Инициализировать T как пустое множество ребер.
2. Пока T не образует дерево (что эквивалентно условию: пока число ребер в T меньше, чем V − 1).

2.1. Для каждой компоненты связности (то есть, дерева в остовном лесе) в подграфе с ребрами T, найти самое дешевое ребро, связывающее эту компоненту с некоторой другой компонентой связности.

2.2. Добавить все найденные ребра в множество T.

3. Вернуть T как минимальное остовное дерево входного графа.

Алгоритм связывания прямоугольных областей

Входные данные:

– points: трехмерный вектор кортежей представляющий координаты нескольких дискретизированных областей;

– start_point: координата стартовой точки;

Выходные данные: вектор кортежей, представляющих последовательность координат для посещения, проходящих через несколько прямоугольных областей.

Шаги алгоритма.

1. Инициализировать weights, представляющей матрицу кортежей, которая указывает расстояния и направления между прямоугольными областями.

2. Между каждой областью A и областью B рассчитать расстояние и направление следующим образом.

2.1. Если правая сторона B левее левой стороны A.

2.1.1. Если верхняя сторона A ниже нижней стороны B, направление вверх-влево.

2.1.2. Иначе, если верхняя сторона B ниже нижней стороны A, направление вниз-влево.

2.1.3. В противном случае, направление влево.

2.2. Если правая сторона A левее левой стороны B.

2.2.1. Если верхняя сторона A ниже нижней стороны B, направление вверх-вправо.

2.2.2. Иначе, если верхняя сторона B ниже нижней стороны A, направление вниз-враво.

2.2.3. В противном случае, направление вправо.

2.3. В остальных случаях:

2.3.1. Если верхняя сторона A ниже нижней стороны B, направление вверх.

2.3.2. В противном случае, направление вниз.

2.4. В соответствии с направлением, высчитать расстояние между прямоугольными областями.

2.4.1. В случае, если направление соответствует значению вверх, вниз, влево, вправо, взять расстояние между ближайшими ребрами прямоугольников A и B.

2.4.2. В случае, если направление соответствует значению вверх-влево, вверх-влево, вниз-влево, вниз-вправо, взять расстояние между ближайшими вершинами.

3. Для каждой области рассчитать путь внутри, используя алгоритм построения маршрута на прямоугольной области.

4. Рассчитать минимальное остовное дерево используя алгоритм Борувки.

5. Для минимального остовного дерева выполнить связывание областей следующим образом.

5.1. Если направление между областями соответствует вверх, вниз, влево, вправо, то связать ближайшие точки ближайших ребер, вставив последовательность посещения внутри области, начиная с ближайших точек.

5.2. В ином случае связать области по диагоналям, вставив последовательность посещения внутри области, начиная с ближайшей вершины.

6. Добавить к результату стартовую точку, вставив ее перед ближайшей точке в последовательности посещения.

Визуализация шага 2 алгоритма связывания прямоугольных областей представлен на рисунке 7. Этот рисунок показывает, что в алгоритме является предпочтительным связывание областей по вертикали и горизонтали. Также стоит уточнить, что алгоритм не предназначен для связывания пересекающихся областей, для таких случаев необходим выбор других способов исследования.

На практике эти алгоритмы можно использовать даже если необходимо изменить направление дискретизации используя преобразование координат при повороте для всех полей разом, это будет учтено при разработке.

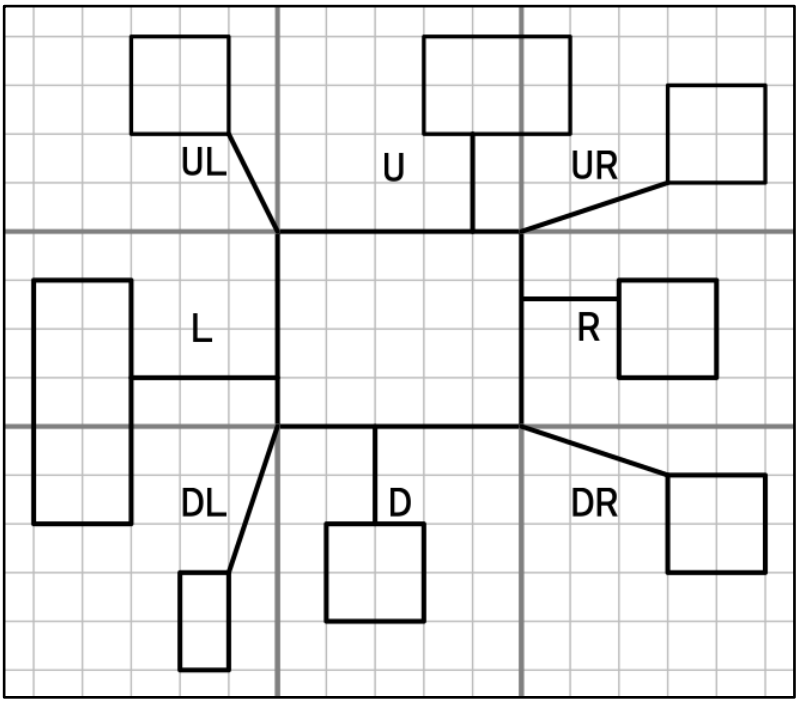

Рисунок 7 – Кратчайшие пути между парой регионов

Вывод по четвертой главе

В четвертой главе было приведено теоретическое описание алгоритмов, описана задача, которую необходимо решить, указаны подходящие алгоритмы. Описан алгоритм ближайшего соседа, алгоритм полного перебора, алгоритм дискретизации, алгоритм построения маршрута на прямоугольной области, алгоритм Борувки для построения минимального остовного дерева и алгоритм связывания прямоугольных областей.

5. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ СИСТЕМЫ

5.1. Реализация компонентов системы

В реализации компонентов системы имеется сервис Data Management, который предоставляет функциональность для работы с информацией о дронах и камерах. Этот сервис состоит из следующих файлов.

1. Основной файл «uav/mod.rs» определяет структуру `uav` и методы для создания экземпляров дронов с заданными параметрами. Пример структуры дрона представлен в листинге 1.

Листинг 1 – Структура дрона

```
#[derive(Debug, Deserialize, Serialize)]
pub struct Uav {
    id: u64,                    // uav id
    pub name: String,           // uav name
    pub max_payload_mass: u64, // maximum payload in grams
    pub flight_duration: u64,  // average flight duration in seconds
    pub takeoff_speed: f64,    // average takeoff speed in meters per
second
    pub flight_speed: f64,     // average flight speed in meters per second
    pub min_altitude: f64,     // minimum safe flight altitude in meters
    pub max_altitude: f64,     // maximum safe flight altitude in meters
}
```

2. Файл «uav/uav_handle.rs» содержит функции для работы с дронами, такие как создание, обновление и удаление дронов, а также получение списка всех дронов. Пример функции для создания нового дрона представлен в листинге 2.

Листинг 2 – Функции для создания нового дрона

```
#[tauri::command]
pub fn new_uav(uav: Uav) -> String {
    let conn = Connection::open("mydatabase.db").expect("Cant open base");
    println!("Received new UAV: {:?}", uav);
    match uav_sql::insert(&uav, &conn) {
        Ok(_) => "Ok».to_string(),
        Err(e) => e.to_string(),
    }
}
```

3. Файл «uav/uav_sql.rs» содержит функции для работы с базой данных, включая создание таблицы для хранения информации о дронах, а также вставка, обновление, удаление и получение списка всех дронов. Пример функции для создания таблицы в базе данных представлен в листинге 3.

Листинг 3 – Функции для создания таблицы uav

```
pub fn create_table(conn: &Connection) -> Result<usize> {
    let db_create = conn.execute(
        "CREATE TABLE IF NOT EXISTS uav (
                uav_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
                uav_name TEXT NOT NULL,
                uav_max_payload_mass INTEGER NOT NULL CHECK
(uav_max_payload_mass >= 0),
                uav_flight_duration INTEGER NOT NULL CHECK
(uav_flight_duration >= 0),
                uav_takeoff_speed REAL NOT NULL CHECK (uav_takeoff_speed >=
0),
                uav_flight_speed REAL NOT NULL CHECK (uav_flight_speed >=
0),
                uav_min_altitude REAL DEFAULT 0 NOT NULL CHECK
(uav_min_altitude >= 0),
                uav_max_altitude REAL DEFAULT 0 NOT NULL CHECK
(uav_max_altitude >= 0),
                camera_id INTEGER,
                    FOREIGN KEY (camera_id)
                    REFERENCES camera (camera_id)
                    ON DELETE SET NULL
                )",
        (),
    );
    let index_create = conn.execute(
        "CREATE UNIQUE INDEX IF NOT EXISTS camera_id_index
        ON uav(camera_id)",
        (),
    );
    match (db_create, index_create) {
        (Ok(val1), Ok(val2)) => Ok(val1 + val2),
        (Err(err), _) => Err(err),
        (_, Err(err)) => Err(err),
    }
}
```

4. Основной файл «camera/mod.rs» содержит структуру и методы для работы с камерой. Структура `camera` содержит различные свойства камеры, включая ее идентификатор, имя, массу, угол обзора по оси *X* и разрешение. Здесь также определены метод для создания новых камер. Пример структуры камеры представлен в листинге 4.

Листинг 4 – Структура камеры

```
#[derive(Debug, Deserialize, Serialize)]
pub struct Camera {
    id: u64,               // id
    pub name: String,      // name
    pub mass: u64,         // mass in grams
    pub fov_x: f64,        // x-axis viewing angle in degrees
    pub resolution_x: u16, // camera resolution x
    pub resolution_y: u16, // camera resolution y
}
```

5. Файл «camera/camera_handle.rs» предоставляет функции для обработки запросов к базе данных. Здесь определены функции для создания, обновления, удаления камеры и получения всех камер из базы данных. Каждая из этих функций открывает соединение с базой данных и вызывает соответствующую функцию из файла «camera/camera_sql.rs». Пример функции для обновления данных о камере представлен в листинге 5.

Листинг 5 – Функция для обновления данных о камере

```
#[tauri::command]
pub fn update_camera(camera: Camera) -> String {
 let conn = Connection::open("mydatabase.db").expect("Cant open base");
 println!("Received updated camera: {:?}", camera);
 match camera_sql::update(&camera, &conn) {
 Ok(_) =>"Ok».to_string(),
 Err(e) => e.to_string(),
 }
}
```

6. Файл «camera/camera_sql.rs» содержит функции для взаимодействия с базой данных. Здесь определены функции для создания таблицы в базе данных, вставки новой камеры в базу данных, обновления существующей камеры, удаления камеры из базы данных и получения всех камер из базы данных. Все эти функции работают с базой данных, выполняя соответствующие SQL-запросы. Пример функции обновления данных о камере представлен в листинге 6.

Листинг 6 – Функция для обновления данных о камере в базе

```
pub fn update(camera: &Camera, conn: &Connection) -> Result<usize> {
    conn.execute(
        "
                UPDATE camera SET
                    camera_name = ?1,
                    camera_mass = ?2,
                    camera_fov_x = ?3,
                    camera_resolution_x = ?4,
                    camera_resolution_y = ?5
                WHERE camera_id = ?6",
        (
            &camera.name,
            &camera.mass,
            &camera.fov_x,
            &camera.resolution_x,
            &camera.resolution_y,
            &camera.id,
        ),
    )
}
```

Сервис Algorithms, описанный файлом «algorithms.rs», представляет собой ключевую составляющую системы и выполняет алгоритмические функции. Он включает в себя основные функции для вызова из пользовательского интерфейса.

1. Функция `discretize_area` реализует алгоритм дискретизации области, описанный ранее, эта функция вызывается при нажатии кнопки «Discretize» пользовательского интерфейса, ее реализация представлена в листинге 1 приложения Б.

2. Функция `nearest_neighbor` реализует алгоритм ближайшего соседа для определения пути через все точки. Выполнить эту функции можно, если в блоке пользовательского интерфейса «Algorithm selection» выбрать пункт «Nearest Neighbor» и нажать на кнопку «Calculate». Реализация этой функции представлена в листинге 2 в приложения Б.

3. Вспомогательная функция `euclidean_distance` принимает две точки и возвращает расстояние между ними, используется евклидова метрика потому как в контексте данной работы задача построения маршрута рассматривается как задача на плоскости, поэтому расстояние вычисляется по теореме Пифагора. Реализация представлена в листинге 7.

Листинг 7 – Функция расчета расстояния между двумя точками

```
pub fn euclidean_distance(a: &(f64, f64), b: &(f64, f64)) -> f64 {
 let (x1, y1) = *a;
 let (x2, y2) = *b;
 ((x2 - x1).powi(2) + (y2 - y1).powi(2)).sqrt()
}
```

4. Функция `brute_force` использует алгоритм полного перебора для поиска оптимального пути через набор точек. Выполнить эту функции можно, если в блоке пользовательского интерфейса «Algorithm selection» выбрать пункт «Brute Force» и нажать на кнопку «Calculate». Использование данной функции очень затратно по времени, алгоритм представлен лишь в демонстративных целях. Функция начинает с создания отдельного потока для каждой точки, каждый из которых запускает функцию

`brute_force_helper`. Эта вспомогательная функция рекурсивно исследует все возможные пути через оставшиеся точки, для обнаружения более короткого пути. Реализация представлена в листинге 3 в приложения Б.

5. Функция `rectangular_areas` использует алгоритм, предназначенный для нахождения маршрута на нескольких прямоугольных полях. Выполнить эту функции можно, если в блоке пользовательского интерфейса «Algorithm selection» выбрать пункт «Rectangular Areas» и нажать на кнопку «Calculate». Реализация представлена в листинге 4 в приложения Б. Эта функция также использует другие служебные функции для совей работы. Функция `rectangles_shortest_path` используется для нахождения направления и расстояния между прямоугольниками, ее реализация представлена в листинге 5 приложения Б, она использует `find_direction` для нахождения направления, реализация представлена в листинге 6 приложения Б. Функция `coordinate_transformation` используется для замены системы координат на плоскости, в частности, ее поворот, реализация представлена в листинге 7 приложения Б. Функция `boruvka_mst` используется для нахождения минимального остовного дерева, ее реализация представлена в листинге 8 приложения Б.

Функция `calculate_distance` вычисляет общую длину пути, проходящего через заданный набор точек. Используется, для отображения длины пути на интерфейсе. Реализация представлена в листинге 8. Исходные коды остальных функций расположены в репозитории GitHub. [14]

Листинг 8 – Функция расчета общей длины пути

```
#[tauri::command]
pub fn calculate_distance(points: Vec<(f64, f64)>) -> f64 {
 points
 .iter()
 .zip(points.iter().cycle().skip(1))
 .map(|(a, b)| euclidean_distance(a, b))
 .sum()
}
```

5.2. Реализация интерфейса системы

Пользовательский интерфейс системы разделен на три основных блока: центральное меню, левое меню и правое меню.

Центральное меню является основной частью интерфейса и играет важную роль, так как оно содержит функции для работы с картой, показывает текущий маршрут и обеспечивает доступ к левому и правому меню, обеспечивает основную навигации по системе. Оно представлено на рисунке 8.

Рисунок 8 – Центральное меню

Центральное меню состоит из нескольких элементов.

1. Карта и маршрут: в центре экрана отображается карта (загружаемая с OpenStreetMap [15]), на которой отображается маршрут дрона. Маршрут представлен в виде линии, соединяющей точки съемки, по которым дрон должен пролететь. Это визуальное представление планируемого маршрута.

2. Кнопки переключения меню: в верхней части интерфейса расположены кнопки, которые позволяют переключать отображение левого и

правого меню. Пользователь может скрыть или показать эти меню по своему усмотрению для освобождения места на экране.

Левое меню включает два блока: блок UAV (Беспилотный Летательный Аппарат) и блок Camera (Камера). Левое меню представлено на рисунке 9.

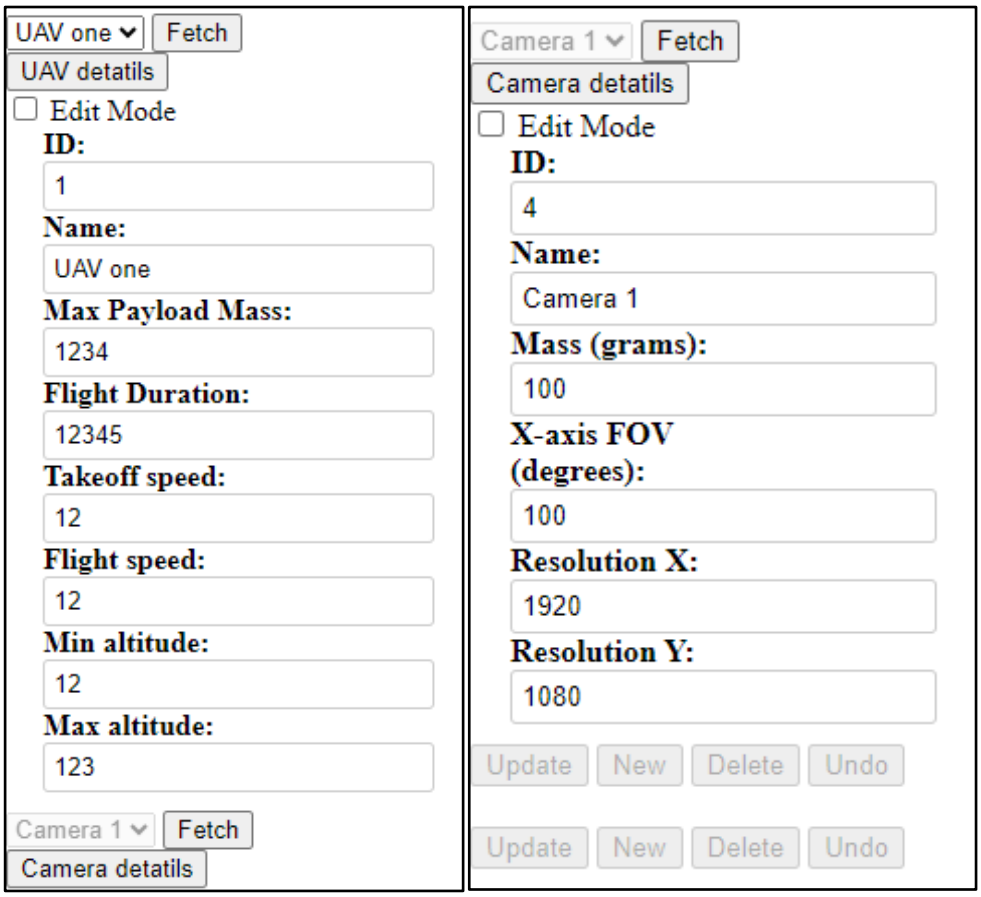

Рисунок 9 – Левое меню

Блок UAV включает в себя инструменты для выбора и редактирования UAV и состоит из нескольких блоков.

1. Выбор и отображение списка UAV: в верхней части блока расположен выпадающий список (select), который позволяет пользователю выбрать один из доступных UAV. Каждый элемент списка содержит имя UAV. При выборе UAV из списка его параметры отображаются в полях ниже.

2. Кнопка «Fetch»: рядом с выпадающим списком находится кнопка «Fetch», которая позволяет пользователю обновить список UAV, загрузив его с базы.

3. Кнопка «UAV details»: эта кнопка позволяет пользователю переключать отображение блока с параметрами UAV. При нажатии на кнопку блок с параметрами UAV может быть скрыт или показан на экране.

4. Внутри блока UAV параметры UAV отображаются в виде полей для ввода которые позволяют пользователю просматривать и редактировать параметры UAV.

4.1. ID: идентификатор UAV (только для чтения).

4.2. Name: имя UAV.

4.3. Max Payload Mass: максимальная масса груза, которую может нести UAV.

4.4. Flight Duration: длительность полета UAV.

4.5. Takeoff speed: скорость взлета UAV.

4.6. Flight speed: скорость полета UAV.

4.7. Min altitude: минимальная высота полета UAV.

4.8. Max altitude: максимальная высота полета UAV.

5. Под блоком с параметрами UAV располагается панель инструментов, которая содержит следующие кнопки.

5.1. «Update»: кнопка, которая позволяет пользователю обновить параметры выбранного UAV.

5.2. «New»: кнопка, которая позволяет пользователю создать новый UAV с указанными в полях параметрами.

5.3. «Delete»: кнопка, которая позволяет пользователю удалить выбранный UAV.

5.4. «Undo»: кнопка, которая позволяет пользователю отменить последнее изменение параметров UAV.

Блок Camera аналогичен по функционалу блоку UAV и включает в себя элементы, представленные ниже.

1. Выпадающий список (select), который позволяет пользователю выбрать одну из доступных камер. Каждый элемент списка содержит имя

камеры. При выборе камеры из списка ее параметры отображаются в полях ниже.

2. Кнопка «Fetch»: эта кнопка позволяет пользователю обновить список камер, загрузив его с базы.

3. Кнопка «Camera details»: эта кнопка позволяет пользователю переключать отображение блока с параметрами камеры. При нажатии на кнопку блок с параметрами камеры может быть скрыт или показан на экране.

4. Внутри блока Camera параметры камеры отображаются в виде полей для ввода (input) и меток (label), которые позволяют пользователю просматривать и редактировать параметры камеры.

4.1. ID: идентификатор камеры (только для чтения).

4.2. Name: имя камеры.

4.3. Mass (grams): масса камеры в граммах.

4.4. X-axis FOV (degrees): угол обзора камеры по оси x в градусах.

4.5. Resolution X: разрешение камеры по оси x.

4.6. Resolution Y: разрешение камеры по оси y.

Правое меню состоит из трех блоков: «Altitude Menu» (меню высоты), «Algorithm Menu» (меню алгоритма) и «Mission Parameters» (параметры миссии). Правое меню представлено на рисунке 10.

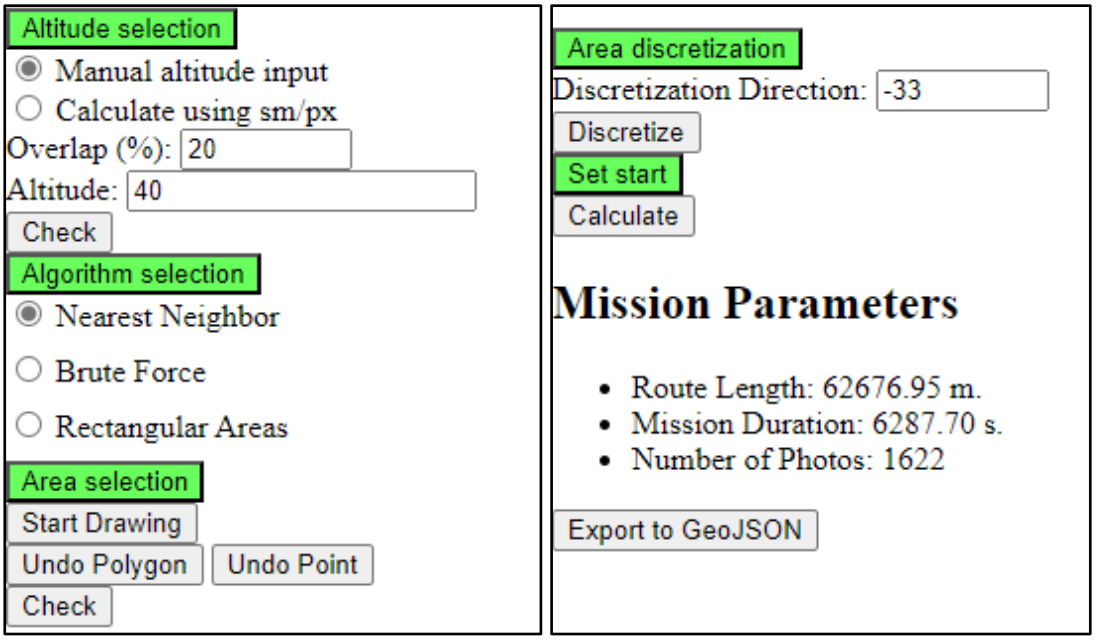

Рисунок 10 – Правое меню

Далее рассмотрен каждый из блоков подробнее.

1. Блок «Altitude selection», здесь можно выбрать режим ввода высоты. Есть два варианта: «Manual altitude input» (ручной ввод высоты) и «Calculate using sm/px» (расчет высоты на основе sm/px).

1.1. Если выбран режим ручного ввода, можно указать процент перекрытия и ввести высоту вручную.

1.2. Если выбран режим расчета на основе sm/px, нужно указать процент перекрытия и ввести значение sm/px. После ввода всех параметров можно нажать кнопку «Calculate Altitude» (рассчитать высоту) для получения результата.

2. Блок «Algorithm selection», здесь можно выбрать алгоритм для расчета маршрута. Есть три варианта: «Nearest Neighbor» (ближайший сосед), «Brute Force» (полный перебор), «Rectangular Areas» (прямоугольные области).

3. Блок «Area selection», предназначен для управления выбором поля. Он состоит из нескольких элементов.

3.1. Кнопка «Start Drawing», предназначена для перехода в режим выделения поля. Позволяет построить полигоны, ограничивающие поля, которые необходимо заснять.

3.2. Кнопка «Undo Polygon», предназначена для удаления последнего установленного полигона.

3.3. Кнопка «Undo Point», предназначена для удаления последней установленной точки полигона.

3.4. Кнопка «Check», предназначена для проверки установки полигона и выхода из режима выделения поля.

3.5. Установка стартовой точки (Set Starting Point): кнопка, позволяющая пользователю установить стартовую точку для маршрута дрона. После выбора этой опции пользователь может щелкнуть на карте, чтобы установить стартовую точку.

4. Блок «Area discretization», здесь устанавливаются параметры дискретизации области съемки. Он состоит из нескольких элементов.

4.1. Поле ввода «Discretization Direction», здесь устанавливается направление дискретизации области съемки в градусах.

4.2. Кнопка «Discretize», предназначена для выполнения алгоритма дискретизации области.

5. Кнопка «Calculate», выполняет валидацию введенных данных и отправляет их на обработку. Результаты расчета маршрута отображаются на карте.

6. Блок «Mission Parameters», здесь отображаются параметры миссии, такие как длина маршрута, продолжительность миссии и количество фотографий. Также есть кнопка «Export to GeoJSON» (экспорт в формате GeoJSON), которая позволяет экспортировать координаты миссии в этот формат для дальнейшего использования.

Дополнительные иллюстрации, демонстрирующие пользовательский интерфейс, представлены в приложении В.

5.3. Тестирование системы

В ходе разработки и последующего программного обеспечения были разработаны и проведены комплексные тесты для проверки функциональности и стабильности работы системы. Проведенные тесты относятся к виду функционального тестирования, а также тестирования пользовательского интерфейса.

Тестирование проводилось в соответствии с предложенной методологией, при которой каждый из тестовых сценариев включал в себя конкретные шаги для воспроизведения действий пользователя, а также определенный ожидаемый результат. В процессе тестирования внимание было уделено как отдельным функциям, так и взаимодействию между различными компонентами системы.

Процесс тестирования осуществлялся с использованием различных входных данных, включая граничные значения и невалидные данные, чтобы оценить устойчивость и надежность системы в различных условиях, а также чтобы выявить и устранить возможные недостатки. Протоколы тестирования приведены в таблице 3.

Таблица 3 – Протоколы тестирования системы

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
1	Загрузка дрона	1. В левом меню выбирать идентификатор дрона. 2. Нажать кнопку «Fetch».	В полях деталей дрона отображается информация о выбранном дроне.	Да
2	Редактирование дрона	1. Выбирать дрон. 2. Включить режим редактирования. 3. Изменить параметры дрона. 4. Нажать кнопку «Update».	Информация о дроне в базе данных обновляется.	Да
3	Расчет маршрута	1. Ввети необходимые параметры, высоту полета, алгоритм, обозначить точку старта, обозначить зону съемки, выбрать дрон, выбрать камеру. 2. Нажать кнопку. «Calculate»	На карте отображается рассчитанный маршрут.	Да
4	Некорректная попытка расчета маршрута	1. Не ввести необходимые параметры, высоту полета, алгоритм, точку старта, зону съемки, дрон, камеру. 2. Нажать кнопку «Calculate».	Отображается ошибка о том, что не все параметры выбраны	Да
5	Проверка параметров миссии	Просмотрите отображаемые параметры миссии в правом меню.	Параметры миссии (длина маршрута, продолжительность миссии, количество фотографий) отображаются корректно.	Да
6	Отображение маршрута	После расчета маршрута проверьте визуализацию маршрута на карте.	Маршрут корректно отображается на карте.	Да
7	Экспорт в GeoJSON	1. После расчета маршрута нажать кнопку «Export to GeoJSON». 2. Выбрать путь сохранения	Файл GeoJSON с информацией о маршруте сохраняется.	Да

Окончание таблицы 2

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
8	Загрузка камеры	1. В левом меню выберать идентификатор камеры. 2. Нажать кнопку «Fetch».	В полях деталей камеры отображается информация о выбранной камере.	Да
9	Редактирование камеры	1. Выбрать камеру. 2. Включить режим редактирования. 3. Изменить параметры камеры. 4. Нажать кнопку «Update».	Информация о камере в базе данных обновляется.	Да
10	Выбор алгоритма	В правом меню выберите желаемый алгоритм.	Выбранный алгоритм устанавливается как текущий.	Да
11	Ручной ввод высоты	1. В правом меню выберите режим ручного ввода высоты. 2. Введите желаемую высоту.	Введенная высота устанавливается как текущая.	Да
12	Расчет высоты	1. В правом меню выберите режим расчета высоты. 2. Введите значение sm/px. 3. Нажмите кнопку «Calculate Altitude».	Высота автоматически рассчитывается и устанавливается как текущая.	Да
13	Изменение параметра перекрытия	В правом меню ввести значение в поле «Overlap (%)».	Значение перекрытия изменяется в соответствии с введенным значением.	Да
14	Проверка работы с отсутствующими данными	Нажать кнопку «Calculate», когда база данных пустая.	Система сообщает о том, что дрон не выбран.	Да

Результат выполнения теста 5 «Проверка параметров миссии» представлен на рисунке 10. На рисунке отображены параметры миссии, длина маршрута, продолжительность миссии, количество фотографий.

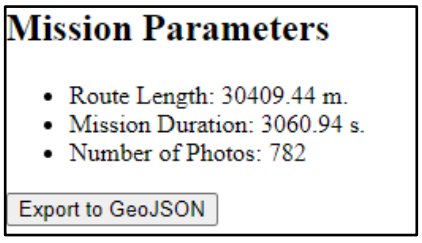

Рисунок 10 – Результат выполнения теста 5
«Проверка параметров миссии»

Результат выполнения теста 6 «Отображение маршрута» представлен на рисунке 11. Маршрут представлен в виде синей линии, соединяющей зеленые точки, представляющие точки съемки дрона.

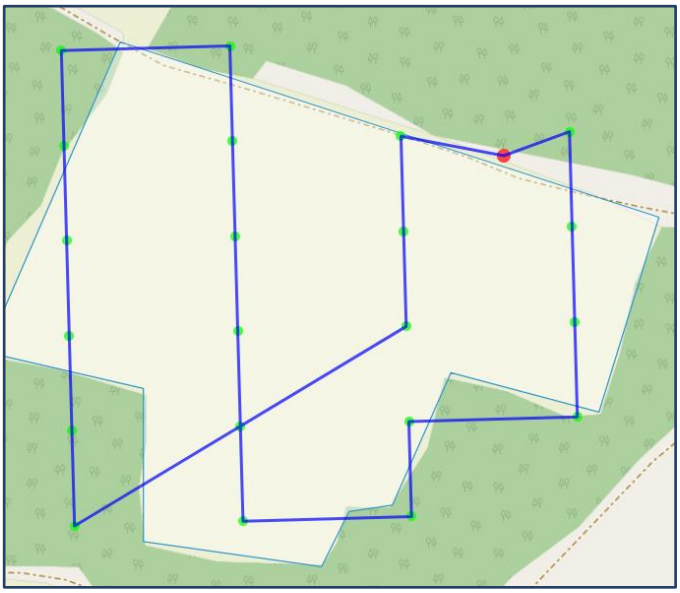

Рисунок 11 – Результат выполнения теста 6 «Отображение маршрута»

Вывод по пятой главе

В пятой главе была приведена реализация компонентов системы, включающая реализацию функций расчета и реализацию пользовательского интерфейса системы. Также проведено тестирование системы.

ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано десктопное приложение для расчета маршрута сельскохозяйственного дрона с учетом его характеристик. Были решены следующие задачи.

1. Был проведен анализ предметной области и изучены существующие решения, связанные с планированием маршрутов для дронов.
2. Была разработана архитектура приложения, обеспечивающая его масштабируемость и простоту внесения изменений в будущем.
3. Были описаны алгоритмы, использующиеся в системе.
4. Был выполнена реализация приложения. Разработаны компоненты бэкенда для расчета маршрута и взаимодействия с базой данных, а также компоненты пользовательского интерфейса.
5. Было проведено тестирование приложения, включая проверку корректности расчетов и работы пользовательского интерфейса.

В перспективе возможно дальнейшее развитие приложения включающее добавление новых и улучшение уже написанных алгоритмов планирования маршрута. Также возможна работа по интеграции приложения с другими системами для автоматического получения координат поля, и экспорта маршрута на дроны.

В рамках работы были опубликованы следующие статьи.

1. Makarovskikh T., Panyukov A., Abotaleb M., Maksimova V., Dernova O., Raschupkin E. Optimal Route for Drone for Monitoring of Crop Yields. // Olenev N., Evtushenko Y., Jaćimović M., Khachay M., Malkova V. (eds) Advances in Optimization and Applications. OPTIMA 2023. Communications in Computer and Information Science, Springer, Cham, 2023. – №1913. – 228–240 pp.

ЛИТЕРАТУРА

1. Якушев В.П., Якушев В.В., Матвеенко Д. А. Роль и задачи точного земледелия в реализации национальной технологической инициативы // Агрофизика. 2017. № 1. С. 51–65.

2. Официальный сайт проекта DroneDeploy. [Электронный ресурс] URL: https://www.dronedeploy.com/ (дата обращения: 11.02.2024 г.).

3. Официальный сайт проекта Litchi. [Электронный ресурс] URL: https://flylitchi.com/ (дата обращения: 11.02.2024 г.).

4. Официальный сайт проекта Pix4D. [Электронный ресурс] URL: https://www.pix4d.com/ (дата обращения: 11.02.2024 г.).

5. Официальный сайт проекта UgCS. [Электронный ресурс] URL: https://www.ugcs.com/ (дата обращения: 11.02.2024 г.).

6. Официальный сайт проекта Tauri. [Электронный ресурс] URL: https://tauri.app/ (дата обращения: 11.02.2024 г.).

7. Официальный сайт проекта Rust. [Электронный ресурс] URL: https://www.rust-lang.org/ (дата обращения: 11.02.2024 г.).

8. Официальный сайт проекта Svelte. [Электронный ресурс] URL: https://svelte.dev/ (дата обращения: 11.02.2024 г.).

9. Официальный сайт проекта OpenLayers. [Электронный ресурс] URL: https://openlayers.org/ (дата обращения: 11.02.2024 г.).

10. Официальный сайт GeoJSON. [Электронный ресурс] URL: https://geojson.org/ (дата обращения: 11.02.2024 г.).

11. Макаровских Т.А. Комбинаторика и теория графов // Стереотип, 2022. – С. 117–122.

12. Gutin G., Yeo A., Zverovich A. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP // Discrete Applied Mathematics, 2002. – №117. – 81–86 pp.

13. Makarovskikh T., Panyukov A., Abotaleb M., Maksimova V., Dernova O., Raschupkin E. Optimal Route for Drone for Monitoring of Crop Yields. // Olenev N., Evtushenko Y., Jaćimović M., Khachay M., Malkova V.

(eds) Advances in Optimization and Applications. OPTIMA 2023. Communications in Computer and Information Science, Springer, Cham, 2023. – №1913. – 228–240 pp.

14. Исходный код программы. [Электронный ресурс] URL: https://github.com/evgenkot/uav-route-calculation (дата обращения: 11.02.2024 г.).

15. Официальный сайт проекта OpenStreetMap. [Электронный ресурс] URL: https://www.openstreetmap.org (дата обращения: 11.02.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

Спецификация вариантов использования (ВИ) системы приведена в таблицах 1–11.

Таблица 1 – Спецификация ВИ «Добавить дрон»

Прецедент: Добавить дрон
ID: 1
Краткое описание: Пользователь добавляет новый дрон с его характеристиками в систему.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь открыл меню с изменением дрона
Основной поток. 1. Пользователь вводит характеристики дрона. 2. Пользователь нажимает кнопку добавления дрона. 3. Система валидирует введенные данные. 4. Система сохраняет дрон в базе данных.
Постусловия. Новый дрон добавлен в систему и доступен для выбора.
Альтернативные потоки. 4А. Ошибка валидации данных: 4А.1. Система отображает сообщение об ошибке и предлагает пользователю исправить введенные данные. 4А.2. Возврат к шагу 3 основного потока. 5А. Ошибка сохранения в базе данных: 5А.1. Система отображает сообщение об ошибке сохранения в базе 5А.2. Возврат к шагу 1 основного потока.

Таблица 2 – Спецификация ВИ «Выбрать дрон»

Прецедент: Выбрать дрон
ID: 2
Краткое описание: Пользователь выбирает дрон для расчета маршрута.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: В системе есть хотя бы один сохраненный дрон

Основной поток. 1. Пользователь нажимает на listbox с доступными дронами. 2. Система отображает список доступных дронов. 3. Пользователь выбирает дрон из списка. 4. Система устанавливает выбранный дрон как текущий.
Постусловия. Дрон выбран и установлен как текущий для дальнейших операций.
Альтернативные потоки. 3А. Пользователь не выбрал дрон: 3А.1. Возврат к шагу 1 основного потока.

Таблица 3 – Спецификация ВИ «Удалить дрон»

Прецедент: Удалить дрон
ID: 2.1
Краткое описание: Пользователь удаляет выбранный дрон из системы.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь выбрал дрон (ID2).
Основной поток: 1. Пользователь выбирает опцию «Удалить дрон». 2. Система удаляет дрон из базы данных.
Постусловия. Дрон удален из системы.
Альтернативные потоки: 2А. Ошибка удаления из базы данных: 2А.1. Система отображает сообщение об ошибке удаления. 2А.2. Возврат к шагу 1 основного потока.

Таблица 4 – Спецификация ВИ «Изменить дрон»

Прецедент: Изменить дрон
ID: 2.2
Краткое описание: Пользователь изменяет характеристики выбранного дрона.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь выбрал дрон (ID2).

Окончание таблицы 4 приложения А

Основной поток: 1. Пользователь изменяет характеристики выбранного дрона 2. Пользователь выбирает опцию по изменению дрона. 3. Система валидирует измененные данные. 4. Система сохраняет изменения в базе данных.
Постусловия. Характеристики дрона успешно изменены.
Альтернативные потоки: 2А. Пользователь отменяет изменения: 2А.1. Пользователь нажимает на кнопку отмены. 2А.2. Возврат к шагу 1 основного потока. 4А. Ошибка изменения в базе данных: 4А.1. Система отображает сообщение об ошибке. 4А.2. Возврат к шагу 1 основного потока.

Таблица 5 – Спецификация ВИ «Обозначить старт»

Прецедент: Обозначить старт
ID: 3
Краткое описание: Пользователь задает стартовую точку маршрута дрона.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь находится в меню карты.
Основной поток: 1. Пользователь выбирает опцию «Обозначить старт». 2. Пользователь устанавливает маркер стартовой точки на карте. 3. Система сохраняет координаты стартовой точки.
Постусловия. Стартовая точка маршрута успешно задана и сохранена в системе.
Альтернативные потоки: 2А. Пользователь изменяет инструмент на карте: 2А.2. Возврат к шагу 1 основного потока.

Таблица 6 – Спецификация ВИ «Выделить область съемки»

Прецедент: Выделить область съемки
ID: 4
Краткое описание: Пользователь указывает область, которую требуется снять с помощью дрона.
Главные актеры: Пользователь
Второстепенные актеры: Нет

Предусловия: Пользователь находится в меню карты.
Основной поток: 1. Пользователь выбирает опцию «Выделить область съемки». 2. Пользователь выделяет область на карте, которую требуется снять с помощью дрона. 3. Система сохраняет координаты стартовой точки.
Постусловия. Область съемки успешно задана и сохранена в системе.
Альтернативные потоки: 2А. Пользователь изменяет инструмент на карте: 2А.2. Возврат к шагу 1 основного потока.

Таблица 7 – Спецификация ВИ «Выбрать камеру»

Прецедент: Выбрать камеру
ID: 5
Краткое описание: Пользователь выбирает камеру для использования вместе с дроном.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: В системе есть хотя бы одна сохраненная камера.
Основной поток: 1. Пользователь нажимает на listbox с доступными камерами. 2. Система отображает список доступных камер. 3. Пользователь выбирает дрон из списка. 4. Система устанавливает выбранную камеру, как текущую.
Постусловия. Камера выбрана и установлена как текущая для дальнейших операций.
Альтернативные потоки: 3А. Пользователь не выбрал камеру: 3А.1. Возврат к шагу 1 основного потока.

Таблица 8 – Спецификация ВИ «Удалить камеру»

Прецедент: Удалить камеру
ID: 5.1
Краткое описание: Пользователь удаляет выбранную камеру из системы.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь выбрал камеру (ID5).

Основной поток: 1. Пользователь выбирает опцию «Удалить камеру». 2. Система удаляет камеру из базы данных.
Постусловия. Камера удалена из системы.
Альтернативные потоки: 2А. Ошибка удаления из базы данных: 2А.1. Система отображает сообщение об ошибке удаления. 2А.2. Возврат к шагу 1 основного потока.

Таблица 9 – Спецификация ВИ «Изменить камеру»

Прецедент: Изменить камеру
ID: 5.2
Краткое описание: Пользователь изменяет характеристики выбранной камеры.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь выбрал камеру (ID5).
Основной поток: 1. Пользователь изменяет характеристики выбранной камеры. 2. Пользователь выбирает опцию по изменению камеры. 3. Система валидирует измененные данные. 4. Система сохраняет изменения в базе данных.
Постусловия. Характеристики камеры успешно изменены.
Альтернативные потоки: 2А. Пользователь отменяет изменения: 2А.1. Пользователь нажимает на кнопку отмены. 2А.2. Возврат к шагу 1 основного потока. 4А. Ошибка изменения в базе данных: 4А.1. Система отображает сообщение об ошибке. 4А.2. Возврат к шагу 1 основного потока.

Таблица 10 – Спецификация ВИ «Добавить камеру»

Прецедент: Добавить камеру
ID: 6
Краткое описание: Пользователь добавляет новую камеру с ее характеристиками в систему.
Главные актеры: Пользователь
Второстепенные актеры: Нет

Предусловия: Пользователь открыл меню с изменением камеры.
Основной поток: 1. Пользователь вводит характеристики камеры 2. Пользователь нажимает кнопку добавления камеры. 3. Система валидирует введенные данные. 4. Система сохраняет камеру в базе данных.
Постусловия. Новая камера добавлена в систему и доступна для выбора.
Альтернативные потоки: 4А. Ошибка валидации данных: 4А.1. Система отображает сообщение об ошибке и предлагает пользователю исправить введенные данные. 4А.2. Возврат к шагу 1 основного потока. 5А. Ошибка сохранения в базе данных: 5А.1. Система отображает сообщение об ошибке. 5А.2. Возврат к шагу 2 основного потока.

Таблица 11 – Спецификация ВИ «Построить маршрут»

Прецедент: Построить маршрут
ID: 7
Краткое описание: Пользователь строит маршрут по заданным параметрам.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: 1. Пользователь выбрал дрон (ID2). 2. Обозначил старт (ID3). 3. Выделил область съемки (ID4). 4. Выбрал камеру (ID5).
Основной поток: 1. Пользователь нажимает кнопку calculate 2. Система проверяет корректность введенных данных. 3. Система отображает маршрут.
Постусловия. Маршрут отображается на карте.
Альтернативные потоки: 2А. Ошибка валидации данных: 2А.1. Система отображает сообщение об ошибке и предлагает пользователю исправить введенные данные. 2А.2. Возврат к шагу 1 основного потока.

Приложение Б. Функции расчета сервиса Algorithms

Листинг 1 – Функция дискретизации области съемки

```
#[tauri::command]
pub fn discretize_area(
    // Vector of tuples representing x and y coordinates of the polygon.
    polygons: Vec<Vec<(f64, f64)>>,
    // Width of the photo.
    photo_width: f64,
    // Height of the photo.
    photo_height: f64,
    // Direction
    direction_degrees: f64,
    // Verification whether the points are inside the polygon
    check_inside: bool,
) -> Result<Vec<Vec<Vec<(f64, f64)>>>, String> {
    // Returns a Result containing either a vector of tuples representing
the discretized area or a String error.
    println!("Received polygon coordinates: {:?}", polygons);
    let direction_radians = direction_degrees * PI / 180.0;
    // Initialize a vector to store the results for each polygon.
    let mut results = Vec::new();

    for polygon in polygons {
        // Initialize min and max x and y values to extreme opposites.
        let (mut min_x, mut max_x, mut min_y, mut max_y) =
            (INFINITY, NEG_INFINITY, INFINITY, NEG_INFINITY);

        // Apply transformation to each point
        let polygon_transformed: Vec<(f64, f64)> = polygon
            .iter()
            .map(|&(x, y)| coordinate_transformation(x, y,
direction_radians))
            .collect();

        // Loop through polygon coordinates to find min and max x and y
values.
        for (x, y) in &polygon_transformed {
            min_x = min_x.min(*x);
            max_x = max_x.max(*x);
            min_y = min_y.min(*y);
            max_y = max_y.max(*y);
        }
        // Inner function to check if a point is inside the polygon.
        fn is_inside_polygon(point: (f64, f64), polygon: &Vec<(f64, f64)>)
-> bool {
            // Initialize inside flag to false.
            let mut inside = false;
            let len = polygon.len();
            let mut j = len - 1;

            // Loop through the polygon's vertices, checking if the point
intersects.
            for i in 0..len {
                let (x_i, y_i) = polygon[i];
                let (x_j, y_j) = polygon[j];
```

```
                // Determine if the point is intersecting with the edge
from vertex i to vertex j.
                let intersect = (y_i > point.1) != (y_j > point.1)
                    && point.0 < (x_j - x_i) * (point.1 - y_i) / (y_j -
y_i) + x_i;
                if intersect {
                    inside = !inside;
                }
                j = i;
            }
            inside
        }

        // Initialize the result vector.
        let mut result = Vec::new();

        // Calculate half the camera width and height.
        let (half_camera_width, half_camera_height) = (photo_width / 2.0,
photo_height / 2.0);

        let polygon_width = (max_x - min_x).abs();
        let polygon_height = (max_y - min_y).abs();

        let photo_count_width = (polygon_width / photo_width) as u64 + 1;
        let photo_count_height = (polygon_height / photo_height) as u64 +
1;

        for i in 0..photo_count_width {
            let x = min_x + (i as f64) * photo_width;
            let mut line = Vec::new();
            for j in 0..photo_count_height {
                let y = min_y + (j as f64) * photo_height;

                if check_inside {
                    // Calculate the corners of the rectangle at (x, y).
                    let corners = vec![
                        (x, y),
                        (x + photo_width, y),
                        (x, y + photo_height),
                        (x + photo_width, y + photo_height),
                    ];

                    // Check if any corner of the rectangle is inside the
polygon.
                    let is_any_corner_inside = corners
                        .iter()
                        .any(|&corner| is_inside_polygon(corner,
&polygon_transformed));
                    // If any corner is inside, calculate the center of the
rectangle and add it to the result.
                    if is_any_corner_inside {
                        let center_x = x + half_camera_width;
                        let center_y = y + half_camera_height;
                        line.push(coordinate_restore(center_x, center_y,
direction_radians));
                    }
                } else {
                    let center_x = x + half_camera_width;
                    let center_y = y + half_camera_height;
                    line.push(coordinate_restore(center_x, center_y,
direction_radians));
```

```
                }
            }
            result.push(line);
        }
        // Push the result vector containing the centers of the rectangles
that intersect with the polygon.
        results.push(result);
    }
    // Return the vector of results containing the centers of the
rectangles that intersect with each polygon.
    Ok(results)
}
```

Листинг 2 – Функция расчета «Ближайший сосед»

```
#[tauri::command]
pub fn nearest_neighbor(
    points: Vec<(f64, f64)>,
    start_point: (f64, f64),
) -> Result<Vec<(f64, f64)>, String> {
    if points.is_empty() {
        return Err("The input points must not be empty".to_string());
    }

    let mut remaining_points: Vec<(f64, f64)> = points;
    let mut result: Vec<(f64, f64)> = Vec::new();
    let mut current_point = start_point;

    while !remaining_points.is_empty() {
        let (nearest_index, nearest_point) = remaining_points
            .iter()
            .enumerate()
            .min_by(|(_, a), (_, b)| {
                euclidean_distance(&current_point, a)
                    .partial_cmp(&euclidean_distance(&current_point, b))
                    .unwrap_or(std::cmp::Ordering::Equal)
            })
            .ok_or("Failed to find the nearest point".to_string())?;
        let nearest_point = *nearest_point;
        remaining_points.remove(nearest_index);

        if result.is_empty() {
            result.push(start_point);
        }
        result.push(nearest_point);
        current_point = nearest_point;
    }

    result.push(start_point);
    Ok(result)
}
```

Листинг 3 – Функции расчета «Полный перебор»

```
// Main function to find the shortest path using the brute force approach.
#[tauri::command]
pub fn brute_force(points: Vec<(f64, f64)>, start_point: (f64, f64)) ->
Vec<(f64, f64)> {
    // Wrap the points and best_path in Arc for shared ownership across
threads.
    let points = Arc::new(points);
    let best_path = Arc::new(Mutex::new((Vec::new(), f64::MAX)));

    let mut threads = Vec::new();

    // Iterate through each point, creating a separate thread for each.
    for (i, point) in points.iter().enumerate() {
        // Clone Arc references to share ownership with the spawned
threads.
        let points = Arc::clone(&points);
        let best_path = Arc::clone(&best_path);
        let start_point = start_point.clone();
        let removed_point = point.clone();

        // Spawn a thread for each point, removing it from the remaining
points.
        let thread = thread::spawn(move || {
            let mut remaining_points = points.to_vec();
            remaining_points.remove(i);

            let current_path = vec![start_point, removed_point];
            let current_distance = euclidean_distance(&start_point,
&removed_point);

            // Call the helper function in each thread.
            brute_force_helper(
                remaining_points,
                current_path,
                start_point,
                current_distance,
                &best_path,
            );
        });

        threads.push(thread);
    }

    // Wait for all threads to complete.
    for thread in threads {
        thread.join().unwrap();
    }

    // Extract the best path from the Arc<Mutex<_>>.
    let (best_path, _) =
Arc::try_unwrap(best_path).unwrap().into_inner().unwrap();
    best_path
}
```

```
// Recursive helper function to find the shortest path using the brute
force approach.
fn brute_force_helper(
    points: Vec<(f64, f64)>,
    current_path: Vec<(f64, f64)>,
    start_point: (f64, f64),
    current_distance: f64,
    best_path: &Arc<Mutex<(Vec<(f64, f64)>, f64)>>,
) {
    // Base case: if there are no points left, update the best path if the
current path is shorter.
    if points.is_empty() {
        let total_distance =
            current_distance + euclidean_distance(&start_point,
current_path.last().unwrap());

        let mut best_path = best_path.lock().unwrap();
        if total_distance < best_path.1 {
            best_path.0 = current_path.clone();
            best_path.0.push(start_point);
            best_path.1 = total_distance;
        }
    } else {
        // Iterate through each remaining point, removing it from the list
and updating the path.
        for (i, point) in points.iter().enumerate() {
            let mut remaining_points = points.clone();

            let removed_point = remaining_points.remove(i);

            let last_point = current_path.last().unwrap();

            let new_distance = current_distance +
euclidean_distance(last_point, &removed_point);

            // If the updated path is shorter than the current best path,
continue the search.
            if new_distance < best_path.lock().unwrap().1 {
                let mut new_path = current_path.clone();
                new_path.push(*point);

                brute_force_helper(
                    remaining_points,
                    new_path,
                    start_point,
                    new_distance,
                    best_path,
                );
            }
        }
    }
}
```

Листинг 4 – Функция расчета пути для прямоугольных областей

```
pub fn rectangular_areas(
    points: Vec<Vec<Vec<(f64, f64)>>>,
    start_point: (f64, f64),
    // Direction for quick calculation of polygon distance
    direction_degrees: f64,
) -> Result<Vec<(f64, f64)>, String> {
    if points.is_empty() {
        return Err("The input points must not be empty".to_string());
    }
    let direction_radians = direction_degrees * PI / 180.0;
    let mut calculation_result: Vec<(f64, f64)> = Vec::new();
    let mut multiple_region_result: Vec<Vec<(f64, f64)>> = Vec::new();

    let region_count = points.len();
    let mut weights: Vec<Vec<Option<(f64, Direction)>>> =
        vec![vec![None; region_count]; region_count];
for i in 0..region_count - 1 {
        let i_height = points[i][0].len();
        let i_width = points[i].len();
        for j in i + 1..region_count {
            let j_height = points[j][0].len();
            let j_width = points[j].len();

            let (a, b) = (
                (
                    coordinate_transformation(
                        points[i][0][i_height - 1].0,
                        points[i][0][i_height - 1].1,
                        direction_radians,
                    ),
                    coordinate_transformation(
                        points[i][i_width - 1][0].0,
                        points[i][i_width - 1][0].1,
                        direction_radians,
                    ),
                ),
                (
                    coordinate_transformation(
                        points[j][0][j_height - 1].0,
                        points[j][0][j_height - 1].1,
                        direction_radians,
                    ),
                    coordinate_transformation(
                        points[j][j_width - 1][0].0,
                        points[j][j_width - 1][0].1,
                        direction_radians,
                    ),
                ),
            );

            let ((a_left, a_top), (a_right, a_bottom)) = a;
            let ((b_left, b_top), (b_right, b_bottom)) = b;
```

```
            for inner_point in [(b_left, b_top), (b_right, b_bottom),
(b_left, b_bottom), (b_right, b_top)].into_iter() {
                if inner_point.0 >= a_left
                    && inner_point.0 <= a_right
                    && inner_point.1 >= a_bottom
                    && inner_point.1 <= a_top
                {
                    return Err(String::from("The rectangles are intersect-
ing."));
                }
            }

            let (weight, direction) = rectangles_shortest_path(a, b);

            weights[i][j] = Some((weight, direction.clone()));
            weights[j][i] = Some((weight.abs(), direction.oppo-
site().clone()));
        }
    }
    // Route inside rectangle
    for region_points in points.clone() {
        // Check if the input vector is rectangular
        let height = region_points[0].len();
        let width = region_points.len();

        if region_points.iter().any(|row| row.len() != height) {
            return Err(String::from("Input vector is not rectangular"));
        }

        if width < 2 {

multiple_region_result.push(region_points.into_iter().flatten().collect());
            break;
        }

        let mut region_result: Vec<(f64, f64)> = Vec::new();

        for i in 0..width {
            region_result.push(region_points[i][0]);
        }

        for j in 1..height {
            region_result.push(region_points[width - 1][j]);
        }

        let ramps = (width - 2) / 2;

        for n in 0..ramps {
            for j in (1..height).rev() {
                region_result.push(region_points[width - 2 - 2 * n][j]);
            }

            for j in 1..height {
                region_result.push(region_points[width - 3 - 2 * n][j]);
            }
        }

        if width % 2 == 1 {
            let coils = (height - 1) / 2;

            for n in 0..coils {
```

```
                region_result.push(region_points[1][height - 1 - n * 2]);
                region_result.push(region_points[0][height - 1 - n * 2]);
                region_result.push(region_points[0][height - 2 - n * 2]);
                region_result.push(region_points[1][height - 2 - n * 2]);
            }

            if height % 2 == 0 {
                region_result.push(region_points[1][1]);
                region_result.push(region_points[0][1]);
            }
        } else {
            for j in (1..height).rev() {
                region_result.push(region_points[0][j]);
            }
        }

        multiple_region_result.push(region_result)
    }

    let mst = boruvka_mst(weights.clone());
    println!("points {:?}", points);
    println!("start_point {:?}", start_point);
    println!("weights {:?}", weights);
    println!("mst {:?}", mst);
    println!("multiple_region_result {:?}", multiple_region_result);

    let mut result_vec: Vec<(f64, f64)> =
multiple_region_result[0].clone();
    let mut done: Vec<usize> = vec![];

    let start_node = 0;
    let mut visited = vec![false; mst.len()];
    let mut stack = vec![start_node];
    let mut done = vec![start_node];
    while let Some(i) = stack.pop() {
        if !visited[i] {
            println!("Visiting node: {}", i);
            for j in mst[i].clone() {
                if done.iter().position(|&x| x == j).is_none() {
                    println!("connecting {} to {}", i, j);
                    match weights[i][j].clone() {
                        Some(edge) => match edge.1 {
                            Direction::U => {
                                println!("Going up");
                                let points2: Vec<(f64, f64)> =
                                    points[j].iter().map(|row|
row[0]).collect();

                                let points1: Vec<(f64, f64)> = points[i]
                                    .iter()
                                    .filter_map(|row| row.last().copied())
                                    .collect();

                                let (p1, p2) = find_minimal_pair(&points1,
&points2).unwrap();
                                result_vec.insert_tuple_after_element(
                                    multiple_region_result[j].clone(),
                                    p2,
                                    p1,
                                )
                            }
```

```
                            Direction::D => {
                                println!("Going down");
                                let points2: Vec<(f64, f64)> = points[j]
                                    .iter()
                                    .filter_map(|row| row.last().copied())
                                    .collect();
                                let points1: Vec<(f64, f64)> =
                                    points[i].iter().map(|row|
row[0]).collect();
                                let (p1, p2) = find_minimal_pair(&points1,
&points2).unwrap();
                                result_vec.insert_tuple_after_element(
                                    multiple_region_result[j].clone(),
                                    p2,
                                    p1,
                                )
                            }
                            Direction::L => {
                                println!("Going Left");
                                let points2: Vec<(f64, f64)> =
points[j].last().unwrap().clone();
                                let points1: Vec<(f64, f64)> =
points[i][0].clone();
                                let (p1, p2) = find_minimal_pair(&points1,
&points2).unwrap();
                                result_vec.insert_tuple_after_element(
                                    multiple_region_result[j].clone(),
                                    p2,
                                    p1,
                                )
                            }
                            Direction::R => {
                                println!("Going Right");
                                let points2: Vec<(f64, f64)> =
points[j][0].clone();
                                let points1: Vec<(f64, f64)> =
points[i].last().unwrap().clone();
                                let (p1, p2) = find_minimal_pair(&points1,
&points2).unwrap();
                                result_vec.insert_tuple_after_element(
                                    multiple_region_result[j].clone(),
                                    p2,
                                    p1,
                                )
                            }
                            Direction::UL => {
                                println!("Going UL");
                                result_vec.insert_tuple_after_element(
                                    multiple_region_result[j].clone(),
                                    points[j][points[j].len() - 1][0],
                                    points[i][0][points[i][0].len() - 1],
                                )
                            }
```

```
                            Direction::UR => {
                                println!("Going UR");
                                result_vec.insert_tuple_after_element(
                                    multiple_region_result[j].clone(),
                                    points[j][0][0],
                                    points[i][points[i].len() -
1][points[i][0].len() - 1],
                                )
                            }
                            Direction::DL => {
                                println!("Going DL");
                                result_vec.insert_tuple_after_element(
                                    multiple_region_result[j].clone(),
                                    points[j][points[j].len() -
1][points[j][0].len() - 1],
                                    points[i][0][0],
                                )
                            }
                            Direction::DR => {
                                println!("Going DR");
                                result_vec.insert_tuple_after_element(
                                    multiple_region_result[j].clone(),
                                    points[j][0][points[j][0].len() - 1],
                                    points[i][points[i].len() - 1][0],
                                )
                            }
                        },
                        None => (),
                    }
                    done.push(j);
                }
            }

            // Mark the current node as visited
            visited[i] = true;

            // Push unvisited neighbors onto the stack
            for &neighbor in &mst[i] {
                if !visited[neighbor] {
                    stack.push(neighbor);
                }
            }
        }
    }
    result_vec.push(result_vec[0].clone());

    let mut closest = result_vec[0];
    let mut min_distance = euclidean_distance(&start_point, &closest);

    for &point in result_vec.iter().skip(1) {
        let distance = euclidean_distance(&start_point, &point);
        if distance < min_distance {
            min_distance = distance;
            closest = point;
        }
    }

    result_vec.insert(
        result_vec.iter().position(|&x| x == closest).unwrap(),
        start_point,
    );
```

```
    println!("res: {:?}", result_vec);

    trait InsertTupleAfterElement {
        fn insert_tuple_after_element(
            &mut self,
            vec2: Vec<(f64, f64)>,
            element: (f64, f64),
            at_element: (f64, f64),
        );
    }

    impl InsertTupleAfterElement for Vec<(f64, f64)> {
        fn insert_tuple_after_element(
            &mut self,
            vec2: Vec<(f64, f64)>,
            element: (f64, f64),
            at_element: (f64, f64),
        ) {
            let index = vec2.iter().position(|&x| x == element).unwrap();
            let index_insert = self.iter().position(|&x| x ==
at_element).unwrap() + 1;
            let (left, right) = vec2.split_at(index);
            self.splice(index_insert..index_insert, right.to_vec());
            self.splice(
                index_insert + right.len()..index_insert + right.len(),
                left.to_vec(),
            );
        }
    }

    Ok(result_vec)
}
```

Листинг 5 – Функция расчета «Ближайший путь между прямоугольниками»

```
// Method to calculate the shortest path between two rectangles
fn rectangles_shortest_path(a: ((f64, f64), (f64, f64)), b: ((f64, f64),
(f64, f64))) -> (f64, Direction) {
    let direction = find_direction(a, b);
    let ((a_left, a_top), (a_right, a_bottom)) = a;
    let ((b_left, b_top), (b_right, b_bottom)) = b;

    (
        match direction {
            Direction::U => b_bottom - a_top,
            Direction::D => a_bottom - b_top,
            Direction::L => a_left - b_right,
            Direction::R => b_left - a_right,
            Direction::UL => euclidean_distance(&(a_top, a_left),
&(b_bottom, b_right)),
            Direction::UR => euclidean_distance(&(a_top, a_right),
&(b_bottom, b_left)),
            Direction::DL => euclidean_distance(&(a_bottom, a_left),
&(b_top, b_right)),
            Direction::DR => euclidean_distance(&(a_bottom, a_right),
&(b_top, b_left)),
        },
        direction,
    )
}
```

Листинг 6 – Функция расчета «Направление пути между прямоугольниками»

```
fn find_direction(a: ((f64, f64), (f64, f64)), b: ((f64, f64), (f64, f64)))
-> Direction {
    let ((a_left, a_top), (a_right, a_bottom)) = a;
    let ((b_left, b_top), (b_right, b_bottom)) = b;
    let direction = if b_right < a_left {
        if a_top < b_bottom {
            Direction::UL
        } else if b_top < a_bottom {
            Direction::DL
        } else {
            Direction::L
        }
    } else if a_right < b_left {
        if a_top < b_bottom {
            Direction::UR
        } else if b_top < a_bottom {
            Direction::DR
        } else {
            Direction::R
        }
    } else {
        if a_top < b_bottom {
            Direction::U
        } else {
            Direction::D
        }
    };
    direction
}
```

Листинг 7 – Функция расчета «Трансформация координат»

```
// Coordinate transformation at rotation, they express old coordinates
through new coordinates
fn coordinate_restore(x: f64, y: f64, direction_radians: f64) -> (f64, f64)
{
    let cosinus = direction_radians.cos();
    let sinus = direction_radians.sin();
    let xd = x * cosinus - y * sinus;
    let yd = x * sinus + y * cosinus;
    (xd, yd)
}
```

Листинг 8 – Функция расчета «Алгоритм Борувки»

```
fn find(parent: &mut [usize], i: usize) -> usize {
    if parent[i] == i {
        i
    } else {
        parent[i] = find(parent, parent[i]);
        parent[i]
    }
}
```

```
fn union_set(parent: &mut [usize], rank: &mut [usize], x: usize, y: usize)
{
    let xroot = find(parent, x);
    let yroot = find(parent, y);

    if rank[xroot] < rank[yroot] {
        parent[xroot] = yroot;
    } else if rank[xroot] > rank[yroot] {
        parent[yroot] = xroot;
    } else {
        parent[yroot] = xroot;
        rank[xroot] += 1;
    }
}
fn boruvka_mst(weights: Vec<Vec<Option<(f64, Direction)>>>) ->
Vec<Vec<usize>> {
    let mut parent: Vec<usize> = (0..weights.len()).collect();
    let mut rank: Vec<usize> = vec![0; weights.len()];
    let mut cheapest: Vec<Option<(usize, usize, f64)>> = vec![None;
weights.len()];
    let mut num_trees = weights.len();
    let mut mst_weight = 0.0;
    let mut result = vec![vec![]; weights.len()];

    while num_trees > 1 {
        for u in 0..weights.len() {
            for v in 0..weights[u].len() {
                if let Some((w, _)) = weights[u][v] {
                    let set1 = find(&mut parent, u);
                    let set2 = find(&mut parent, v);
                    if set1 != set2 {
                        if cheapest[set1].is_none() ||
cheapest[set1].unwrap().2 > w {
                            cheapest[set1] = Some((u, v, w));
                        }
                        if cheapest[set2].is_none() ||
cheapest[set2].unwrap().2 > w {
                            cheapest[set2] = Some((u, v, w));
                        }
                    }
                }
            }
        }
        for node in 0..weights.len() {
            if let Some((u, v, w)) = cheapest[node] {
                let set1 = find(&mut parent, u);
                let set2 = find(&mut parent, v);
                if set1 != set2 {
                    mst_weight += w;
                    union_set(&mut parent, &mut rank, set1, set2);
                    result[u].push(v);
                    result[v].push(u); // For undirected graph
                    num_trees -= 1;
                }
            }
        }
        cheapest.iter_mut().for_each(|x| *x = None);
    }
    println!("Weight of MST is {}", mst_weight);
    result
}
```

Приложение В. Скриншоты приложения

Скриншоты разработанного приложения приведены на рисунках 1–7.

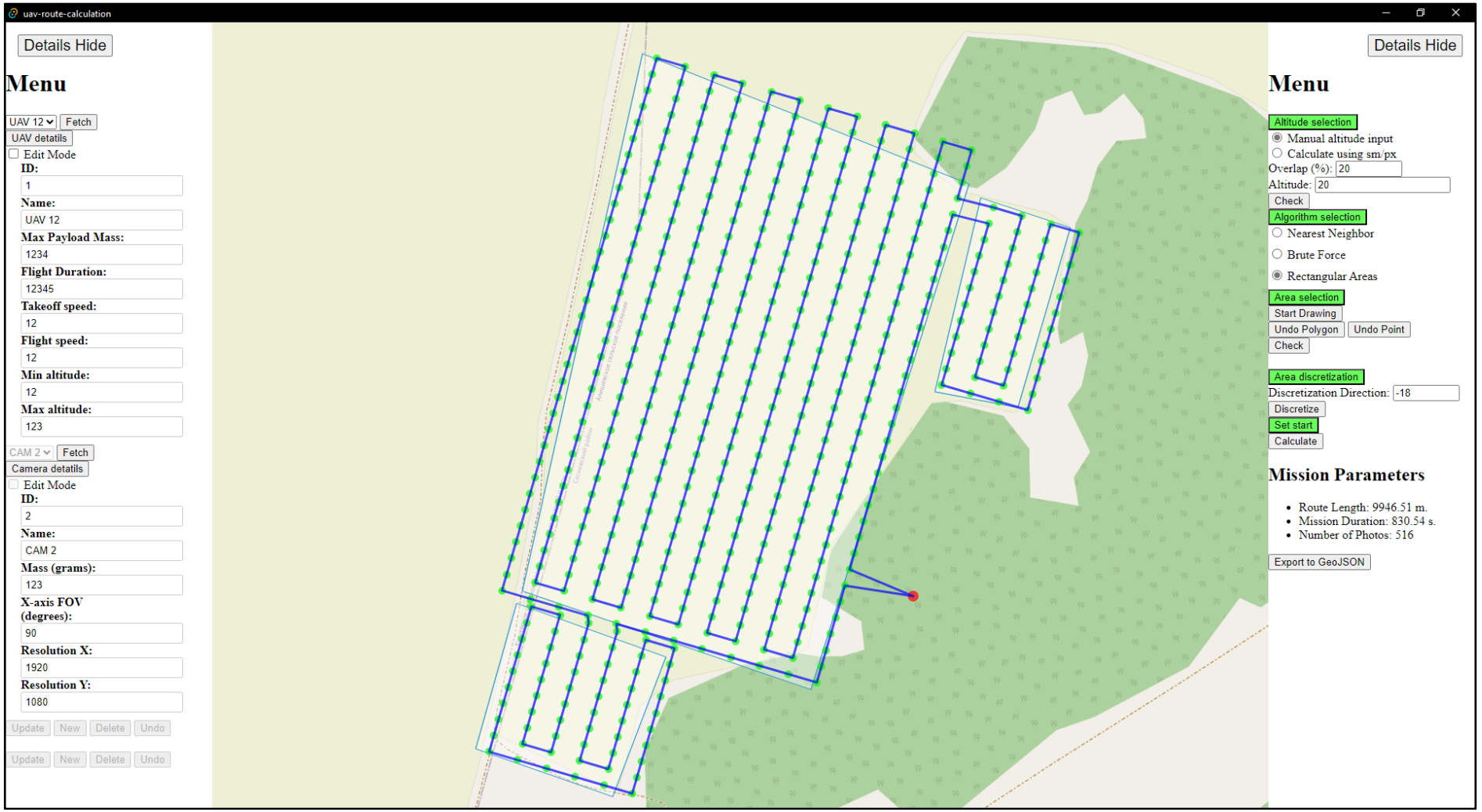

Рисунок 1 – Отображение всех меню

Рисунок 2 – Расчет исходя из разрешения съемки

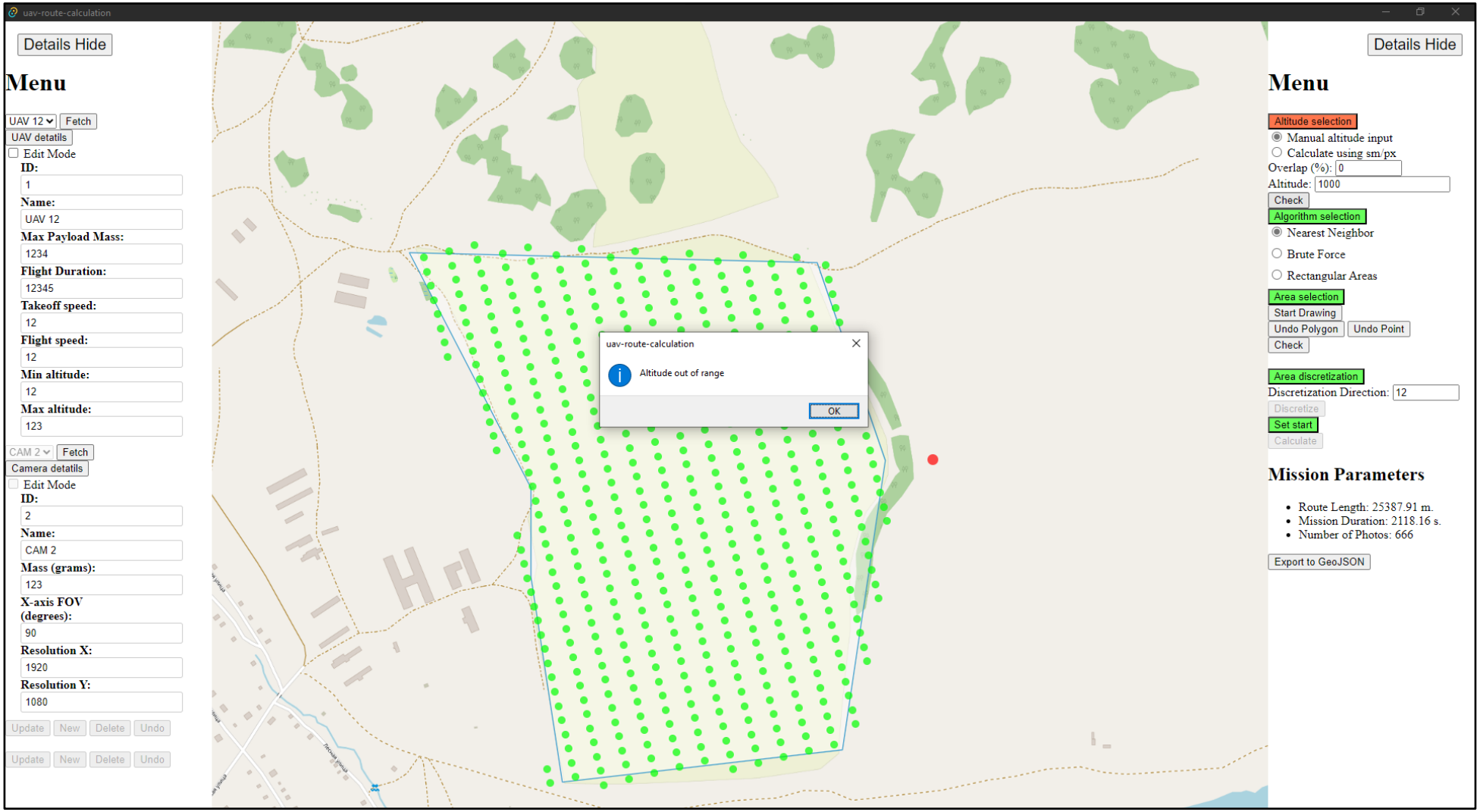

Рисунок 3 – Сообщение об ошибке

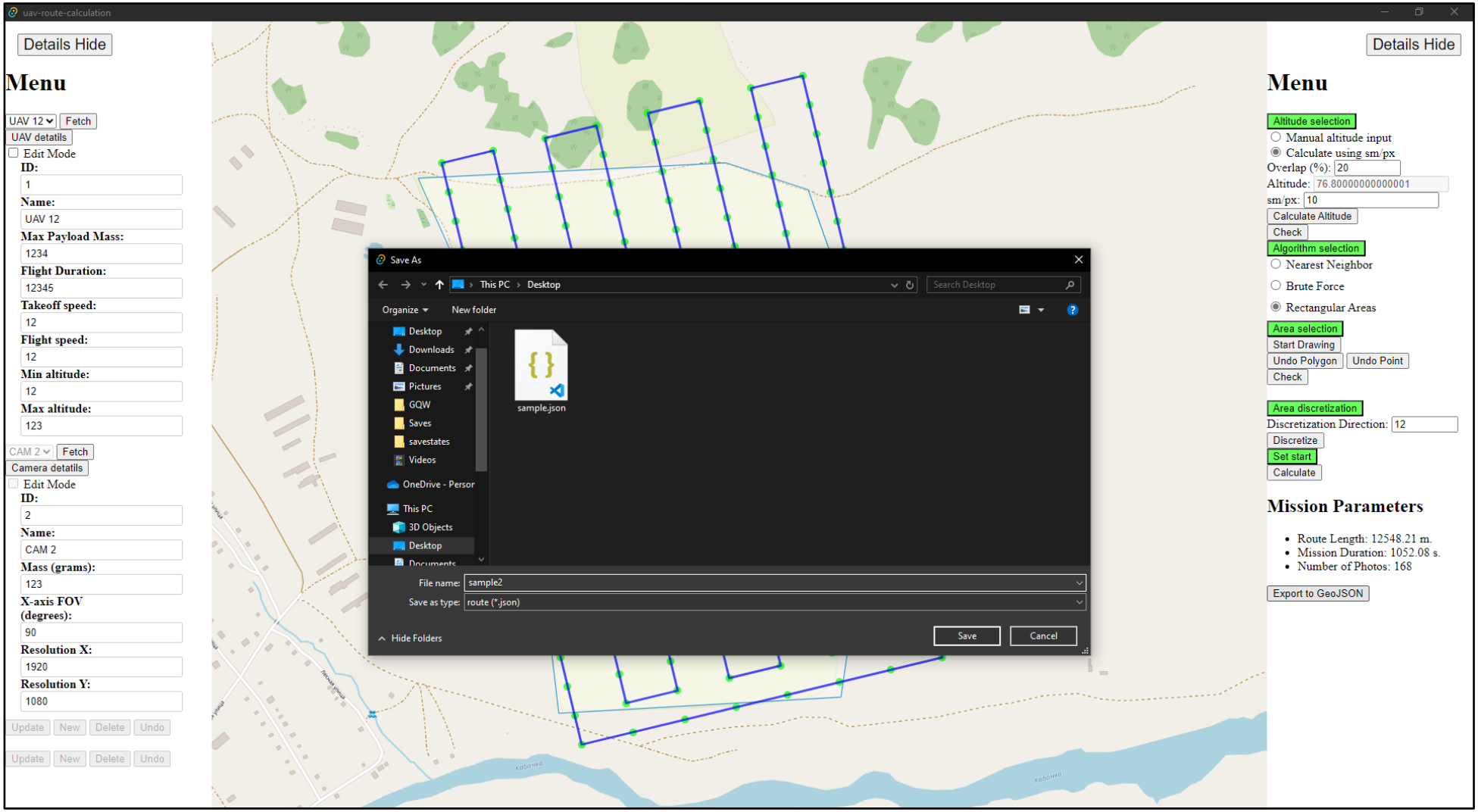

Рисунок 4 – Окно экспорта в GeoJSON

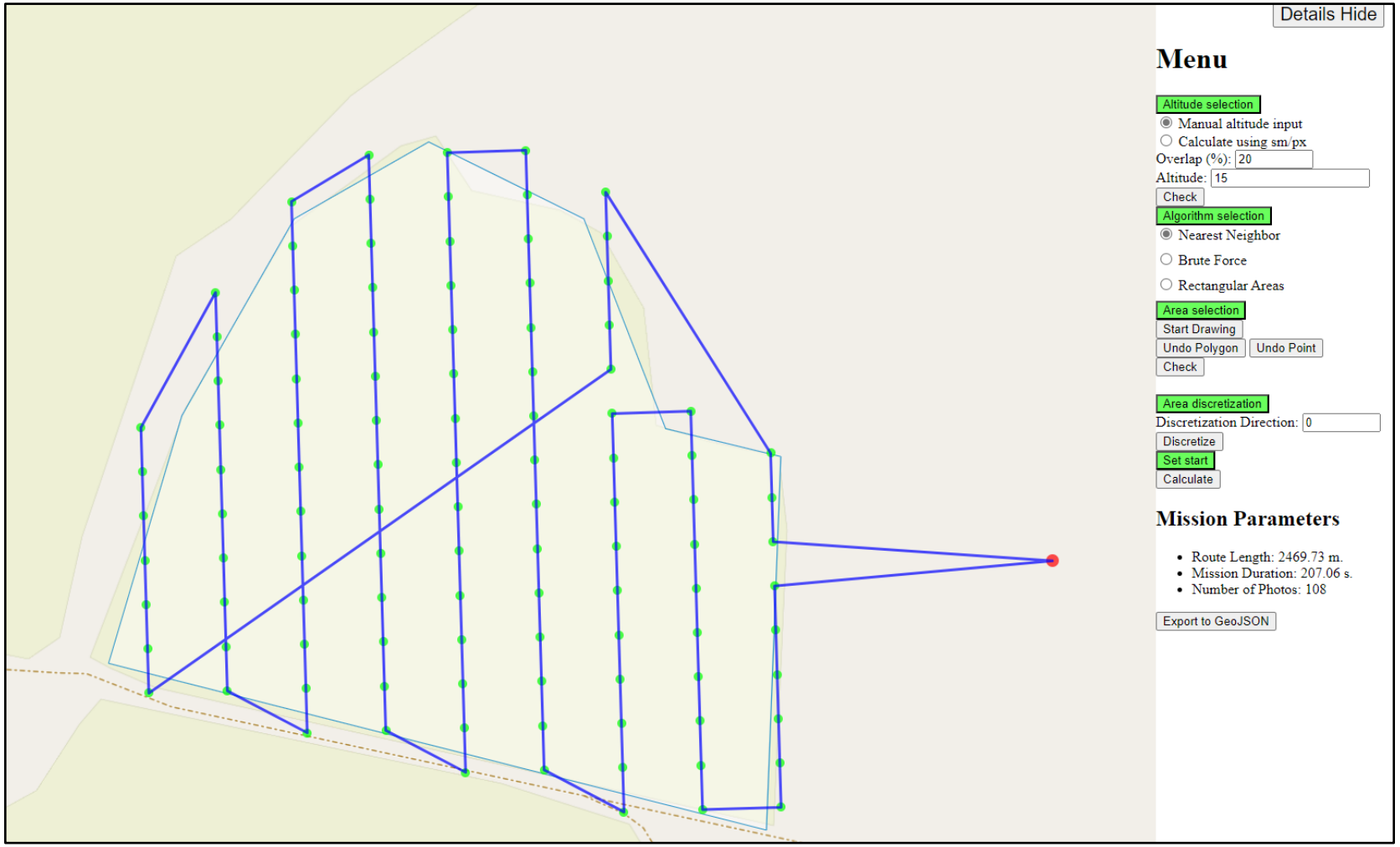

Рисунок 5 – Работа алгоритма ближайшего соседа

Рисунок 6 – Работа алгоритма полного перебора

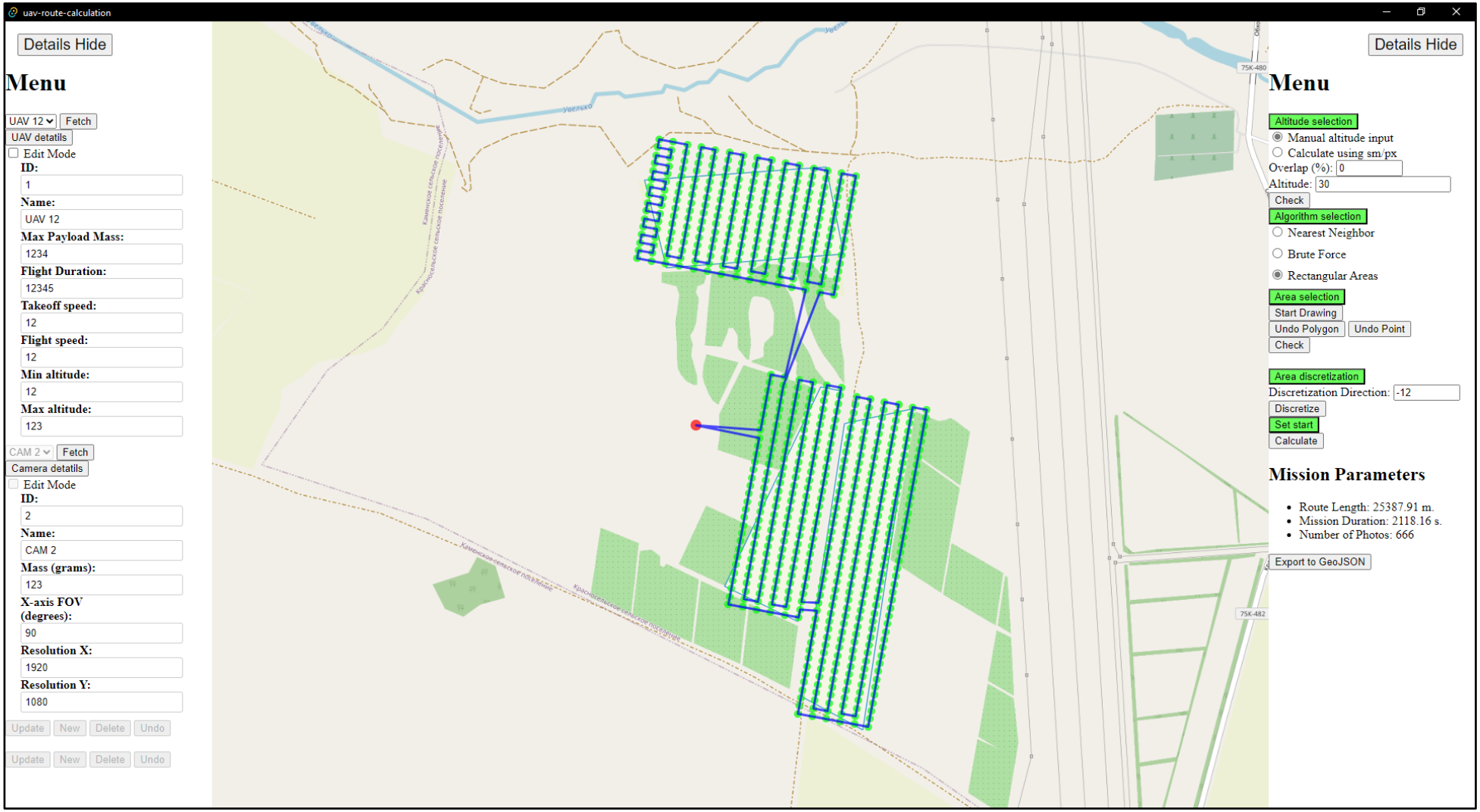

Рисунок 7 – Работа алгоритма для прямоугольных областей