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 «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2025 г.

Разработка программного модуля отслеживания движения или дрейфа БПЛА относительно окружающей среды на основе последовательных изображений

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2025.406 ПЗ ВКР

Руководитель работы,
к.пед.н., доцент каф. ЭВМ
_____ Ю.Г. Плаксина
«___»_____2025 г.

Автор работы,
студентка группы КЭ-406
_____О.А. Губина
«___»_____2025 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2025 г.

Челябинск-2025

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 «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2025 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студентке группы КЭ-406
Губиной Олесе Анатольевне
обучающейся по направлению
09.03.01 «Информатика и вычислительная техника»

Тема работы: «Разработка программного модуля отслеживания движения или дрейфа БПЛА относительно окружающей среды на основе последовательных изображений» утверждена приказом по университету от «21» апреля 2025 г. № 648-13/12.

Срок сдачи студентом законченной работы: 01 июня 2025 г.

Исходные данные к работе:

Функциональные требования к программному модулю:

- возможность развертывание системы на одноплатном компьютере Orange Pi CM5 Базовая плата с вычислительной мощностью 16 Гб;
- возможность интеграции с модулем камеры QSZNTEC IMX415 с разрешением матрицы 8 мегапикселей.

Перечень подлежащих разработке вопросов:

1. Аналитический обзор научно-технической, нормативной и методической литературы по тематике работы.

2. Исследование алгоритмов для применения в системе оптического удержания позиции БПЛА.
3. Проектирование и реализация программного модуля отслеживания движения БПЛА относительно окружающей среды на основе последовательных изображений.
4. Тестирование программного модуля.

Дата выдачи задания: 02 декабря 2024 г.

Руководитель работы _____/ *Ю.Г. Плаксина* /

Студентка _____/ *О.А. Губина* /

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	03.03.2025	
Исследование алгоритмов для применения в системе оптического удержания позиции БПЛА	22.03.2025	
Проектирование и реализация программного модуля отслеживания движения БПЛА относительно окружающей среды на основе последовательных изображений	12.04.2025	
Тестирование программного модуля	26.04.2025	
Компоновка текста работы и сдача на нормоконтроль	22.05.2025	
Подготовка презентации и доклада	30.05.2025	

Руководитель работы _____/ *Ю.Г. Плаксина* /

Студентка _____/ *О.А. Губина* /

Аннотация

О.А. Губина. Разработка программного модуля отслеживания движения или дрейфа БПЛА относительно окружающей среды на основе последовательных изображений. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2025, 61 с., 16 ил., библиогр. список – 27 наим.

В рамках выпускной квалификационной работы производится анализ алгоритмов обработки изображений для позиционирования беспилотного летательного аппарата и существующих решений для применения в системе оптического удержания позиции.

В работе поднимается проблема зависимости беспилотных летательных аппаратов от данных спутниковых систем, таких как GPS. Решением поставленной проблемы является разработка автономных систем навигации с программным модулем, способным работать независимо от внешних источников позиционирования.

В выпускной квалификационной работе описан процесс проектирования и реализации программного модуля, предназначенного для оценки перемещения БПЛА относительно окружающей среды на основе последовательных изображений.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время с развитием беспилотных летательных аппаратов (БПЛА) или дронов появилась возможность решения множества задач: мониторинг пожаров с целью поиска людей [1], мониторинг объектов нефтегазовой отрасли для обеспечения безопасности [2], использование средств навигации в сложных условиях окружающей среды [3].

В современных беспилотных летательных аппаратах (БПЛА) навигация и отслеживание положения часто основываются на данных спутниковых систем, таких как GPS. Однако в некоторых ситуациях использование GPS оказывается ограниченным или невозможным: в условиях плотной городской застройки, внутри помещений, в зонах с глушением сигнала. Решением данной проблемы может являться разработка автономных систем навигации, способных работать независимо от внешних источников позиционирования, с определением трехмерных координат без применения часто используемых навигационных средств.

Одним из наиболее перспективных подходов к отслеживанию движения БПЛА является использование компьютерного зрения. Компьютерное зрение – это технология создания искусственных компьютерных систем, которые осуществляют обнаружение, классификацию и отслеживание объектов [4]. Алгоритмы анализа изображений, такие как оптический поток, извлечение ключевых признаков и визуальная одометрия позволяют оценивать перемещения БПЛА, основываясь на последовательных кадрах видео, и обеспечивают надежную локализацию в реальном времени.

1. АНАЛИТИЧЕСКИЙ ОБЗОР СОВРЕМЕННОЙ НАУЧНО-ТЕХНИЧЕСКОЙ, НОРМАТИВНОЙ, МЕТОДИЧЕСКОЙ ЛИТЕРАТУРЫ, ЗАТРАГИВАЮЩЕЙ ИССЛЕДУЕМУЮ НАУЧНО-ТЕХНИЧЕСКУЮ ПРОБЛЕМУ

1.1. МЕТОД ОПТИЧЕСКОГО ПОТОКА

Оптический поток (ОП) – это картина видимого движения объектов изображения между двумя последовательными кадрами, вызванная движением объекта или камеры. Это двумерное векторное поле, где каждый вектор – это вектор смещения, показывающий движение точек от первого кадра ко второму [5].

Пример отображения векторного поля представлен на рисунке 1.

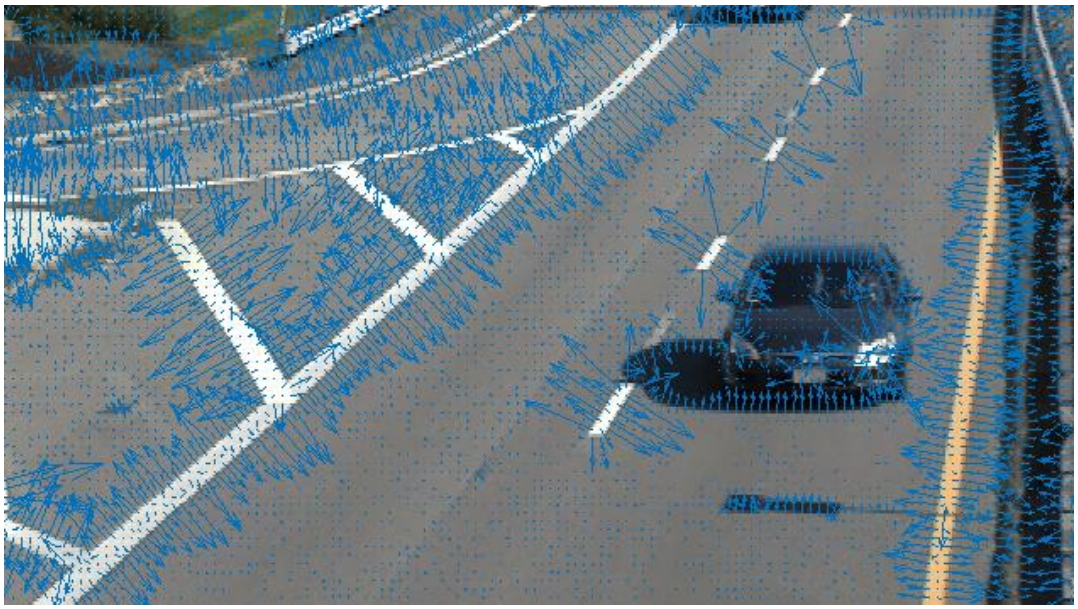

Рисунок 1 – Пример отображения векторного поля на снимке

Использование метода позволяет решать задачи посадки БПЛА [6], отслеживать объекты на местности, так как оптический поток содержит информацию о перемещении камеры.

Существует два вида расчета оптического потока: расчет сдвига всех точек изображения (плотный оптический поток) и расчет сдвига отдельных ключевых точек (выборочный оптический поток), что создает различие во времени вычисления сдвига.

К существующим методам вычисления оптического потока относят алгоритм Лукаса-Канаде, алгоритм Хорна-Шанка и алгоритм Фарнебака.

Алгоритм Лукаса-Канаде использует выборочный поток (точки, найденные алгоритмами извлечения ключевых признаков), оценивая изменение расположения точек на первом и следующих кадрах.

Вычисление оптического потока с помощью данного алгоритма основывается на трех гипотезах [7]:

1. Значения яркостей пикселей переходят между кадрами без изменений. То есть, пиксели смещаются без изменения значений яркости или цвета (для цветных изображений).
2. Область поиска смещения ограничивается окном небольшого размера, так как скорость перемещения пикселей от кадра к кадру невелика.
3. Соседние пиксели перемещаются на одинаковое расстояние.

Простейшее уравнение нахождения оптического потока можно записать так:

$$I_t + \nabla I(uv) = 0, \quad (1)$$

где I – точка пикселя определенного кадра;

t – текущий номер соответствующего кадра;

u, v – смещение по x и y соответственно.

В общем виде, алгоритм Лукаса-Канаде включает в себя разложение функции изображения в ряд Тейлора, представление уравнений для каждого пикселя с учетом гипотезы 2, получение матриц градиента пикселей, вектора смещения (искомого), вектора изменения цвета в окрестности и решение с помощью метода наименьших квадратов [8].

Алгоритм Лукаса-Канаде [9] основан на трех значительных допущениях, которые дают возможность вычисления оптического потока, но вносят погрешность. Метод не является эффективным, если существует значительное движение между кадрами, цвет или яркость не является постоянной

характеристикой точки, то есть, когда 3 вышеописанные предположения оказываются неверными.

С помощью метода Хорна-Шанка рассчитывается плотный оптический поток, что позволяет не использовать дополнительные алгоритмы для извлечения ключевых точек.

Этот метод основывается на предположении о плавности изменения оптического потока в пространстве, что позволяет сгладить локальные шумы и получить устойчивую оценку движения. Для корректной работы алгоритма необходимо выполнение следующих условий [10]:

1. Постоянство яркости. Предполагается, что яркость одной и той же точки сцены остается неизменной при движении.
2. Малость смещения. Движение между последовательными кадрами достаточно мало, чтобы линейная аппроксимация градиента была применима.
3. Векторы движения соседних точек сцены изменяются плавно.

Нахождение оптического потока методом Хорна-Шанка сводится к минимизации функционала (скалярной величины, выражающей совокупное отклонение решения от двух условий: закона сохранения яркости и гладкости потока). Такая минимизация позволяет определить значение параметров u, v, при которых отклонение минимально.

Функционал имеет вид:

$$E(u,v) = \iint \left[\left(I_x u + I_y v + I_t \right)^2 + \alpha^2 (\| \nabla u \|^2 + \| \nabla v \|^2)^2 \right] dx\, dy\,, \qquad (2)$$

где I_x, I_y, I_t – частные производные интенсивности изображения по координатам и времени;

u, v – компоненты оптического потока;

α – параметр, регулирующий баланс между точностью и гладкостью;

$\| \nabla u \|^2, \| \nabla v \|^2$ – градиенты, которые задают гладкость потока.

Алгоритм обеспечивает устойчивые результаты в условиях, когда сцена удовлетворяет вышеуказанным предположениям, однако его точность может

снижаться в областях с резкими изменениями яркости или при значительных нарушениях предположения о плавности.

Алгоритм Фарнебака так же относится к методу вычисления плотного оптического потока. Он предполагает постоянный градиент изображения и постоянный локальный оптический поток.

Основным условием использования этого метода является наличие малых смещений между последовательными кадрами и достаточно четкой текстуры изображения.

На начальном этапе работы алгоритма Фарнебака происходит создание пирамиды изображений различного разрешения. Затем на каждом уровне пирамиды выполняется разложение для нахождения изменений интенсивности между кадрами. На конечных этапах работы алгоритм ищет наилучшие смещения на каждом уровне пирамиды и объединяет оценки движения для получения карты оптического потока.

Рассмотренные алгоритмы для вычисления оптического потока можно сравнить по типу вычисляемого оптического потока, допущениях, при которых алгоритмы эффективны и чувствительности к шуму на изображениях. Приведенные в таблице 1 характеристики оказывают влияние на быстродействие алгоритма и его применимость для достижения поставленных задач.

Таблица 1 – Сравнение алгоритмов вычисления оптического потока

Критерий	Алгоритмы вычисления оптического потока		
	Лукас-Канаде	Хорн-Шанк	Фарнебак
Тип оптического потока	Выборочный	Плотный	Плотный
Условия использования	Постоянство яркости. Малое смещение. Одинаковое расстояние перемещения соседних пикселей	Гладкость потока на всем изображении. Малое смещение. Постоянство яркости	Малое смещение между кадрами. Достаточная текстура изображения Гладкость полей интенсивности

Продолжение таблицы 1

Критерий	Алгоритмы вычисления оптического потока		
	Лукас-Канаде	Хорн-Шанк	Фарнебак
Чувствительность к шуму	Высокая	Низкая	Может требоваться фильтрация
Вычислительная сложность	Низкая	Средняя	Средняя
Точность на сложных текстурах	Высокая (в текстурированных областях)	Средняя (в условиях сильного шума)	Высокая (при плавных текстурах)

Рассмотренный метод предоставляет информацию о движении между последовательными кадрами видеопотока. Оптический поток не всегда используется для распознавания движения как единственный метод, так как для более информативного отслеживания движения существуют алгоритмы извлечения ключевых признаков изображения.

1.2. АЛГОРИТМЫ ИЗВЛЕЧЕНИЯ КЛЮЧЕВЫХ ПРИЗНАКОВ

Алгоритмы выделения ключевых точек являются основополагающими в области компьютерного зрения. Ключевые точки – это наиболее информативные области изображения, которые обладают уникальными характеристиками. Такие точки располагаются, как правило, на углах и краях изображения или в областях с выраженной текстурой.

Для сопоставления ключевых точек между изображениями необходима дополнительная информация об областях вокруг этих точек. Для этого используются дескрипторы – наборы признаков, которые описывают окрестности ключевой точки. Дескрипторы позволяют сравнивать точки между кадрами в условиях изменения масштаба, угла обзора или освещения на изображениях.

Отображение ключевых точек и схематичное изображение дескриптора представлено на рисунке 2.

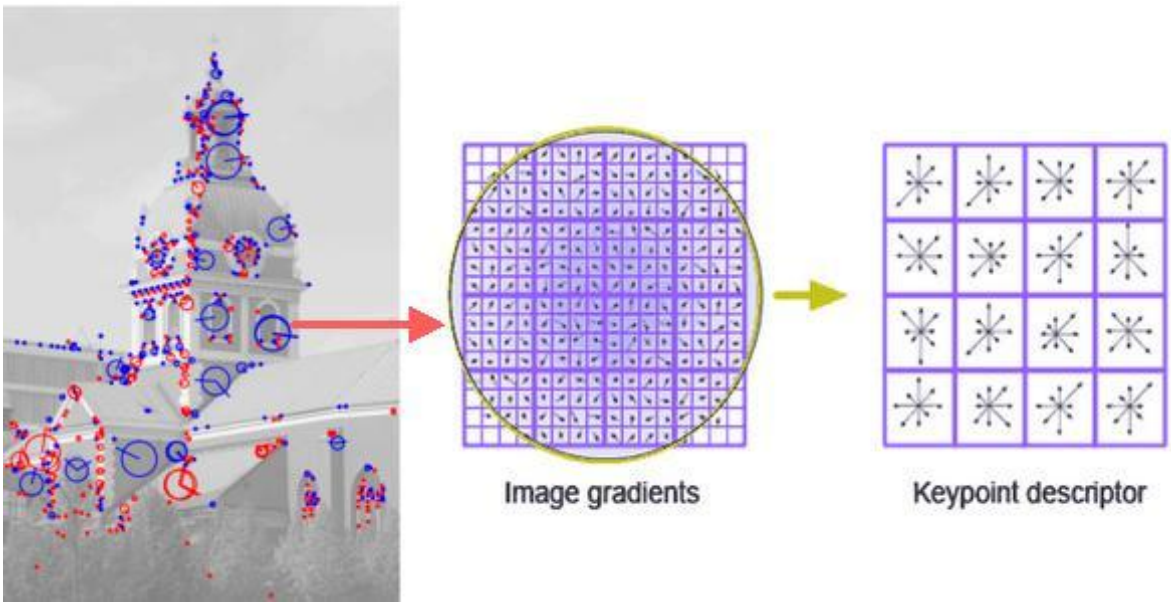

Рисунок 2 – Отображение ключевых точек на снимке

Использование ключевых точек и соответствующих дескрипторов важно в условиях, где другие алгоритмы обработки изображений (например, вычисление оптического потока) не являются достаточными. К таким условиям относятся резкое изменение кадра, вращение камеры или слабое освещение. Кроме того, алгоритмы извлечения ключевых признаков часто являются частью других алгоритмов для обработки изображений, что делает их незаменимыми для задач отслеживания траектории.

К свойствам ключевых точек [11] относятся следующие характеристики:

- постоянство (ключевые точки обладают неизменным местоположением при изменении точки обзора и условий освещенности);
- различимость (окрестности ключевых точек должны обладать уникальными характеристиками, позволяющими отличать их от других точек);
- локальность (ключевая точка должна быть связана с небольшой областью изображения);
- производительность (процесс обнаружения ключевых точек должен быть достаточно быстрым, чтобы соответствовать требованиям времени обработки в приложениях, где критически важна высокая скорость анализа изображения);

- достаточность (число выделяемых ключевых точек должно быть достаточным для распознавания объектов);
- стабильность (положение ключевых точек должно оставаться неизменным при изменении масштаба изображения).

В общем виде алгоритмы поиска ключевых точек состоят из следующих этапов:

1. Подготовка снимков. Для предварительной подготовки применяют перевод изображения объекта из полноцветных в содержащие полутона, повышение резкости снимка, равномерное распределение интенсивности, устранение шума и размытия [12]. В некоторых случаях этот этап является необязательным, так как может быть деструктивным для снимка [13].
2. Обнаружение ключевых точек и вычисление соответствующих дескрипторов. Дескриптор содержит градиенты яркости, записанные в виде векторов, которые суммируются по крупным участкам для определения выраженного направления.
3. Разделение снимков на фрагменты, которое гарантирует нахождение сопоставления.
4. Сопоставление ключевых точек. Пример изображения с сопоставленными ключевыми точками представлен на рисунке 3.

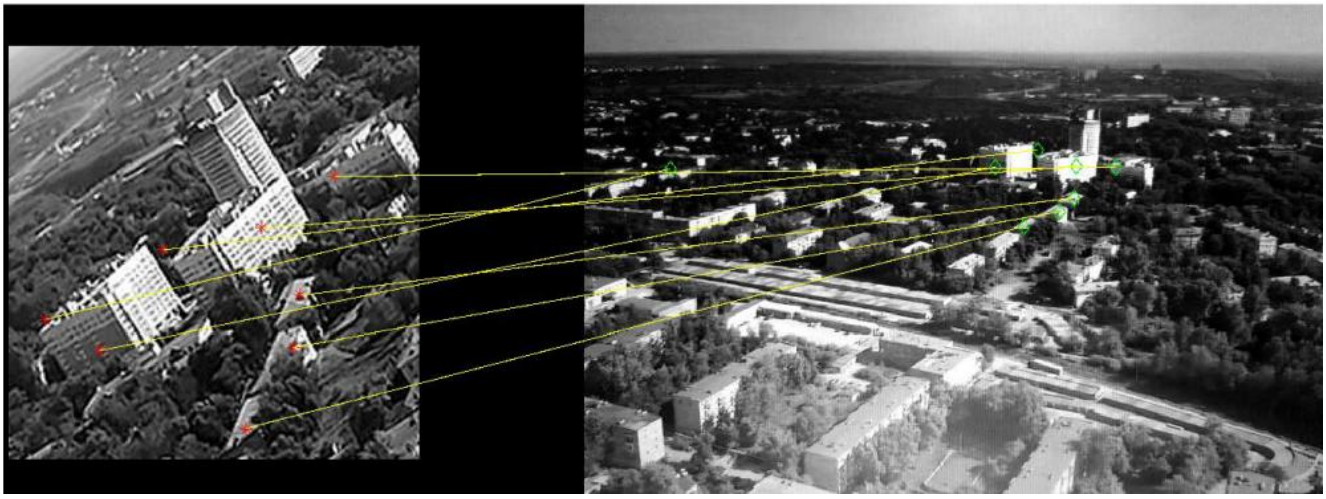

Рисунок 3 – Пример изображения с сопоставленными ключевыми точками

Алгоритмы извлечения ключевых признаков, такие как SIFT, SURF и ORB позволяют не только выделять ключевые точки, но и создавать их дескрипторы, предоставляя инструмент для анализа последовательностей изображений и выполнения задач автономной навигации.

В основе алгоритма SIFT лежит обнаружение экстремумов на изображении в различных масштабах. Под такими изображениями понимают множество вариантов исходного кадра, к которому применяются сглаживающие фильтры. Этот процесс реализуется с помощью пирамиды Гаусса и нахождения разностей гауссианов (изображений, многократно размытых с помощью фильтра Гаусса с различными значениями стандартного отклонения) [14].

Из множества выделенных ключевых точек выбираются точки, устойчивые к изменениям масштаба, ориентации и шуму. Ключевые точки, обладающие низкой контрастностью и находящиеся на границах изображения, отбрасываются.

Для каждой оставшейся ключевой точки рассчитывается дескриптор, который представляет собой вектор признаков, описывающих локальные характеристики изображения в окрестности ключевой точки. Для этого окрестность точки делится на небольшие блоки, и для каждого блока вычисляются градиенты яркости. Эти градиенты используются для формирования гистограмм, которые затем объединяются в дескриптор.

На последнем этапе алгоритма SIFT используется сопоставление дескрипторов ключевых точек между изображениями для нахождения соответствующих точек. Для этого обычно применяется метод ближайшего соседа, где ключевые точки из одного изображения сравниваются с ключевыми точками другого изображения на основе расстояния между их дескрипторами.

Алгоритм SURF был разработан как более быстрый и эффективный аналог методу SIFT, но, в отличие от вышеназванного метода, SURF основывается на приближении гауссовых фильтров с помощью детерминант

матриц Гессе [15]. Гессиан устойчив к ориентации изображения, но неустойчив к масштабированию. Эта проблема решается с помощью перебора масштабов и фильтров и применению их к каждому пикселю.

Алгоритм ищет точки, которые характеризуются высокой изменчивостью интенсивности, что обычно происходит на краях объектов или угловых точках.

В SURF используется более эффективное и компактное представление дескрипторов, что позволяет ускорить вычисления по сравнению с SIFT, так как дескриптор представляет собой массив чисел.

Сопоставление ключевых признаков происходит путем вычисления расстояния между дескрипторами разных точек и поиском наиболее схожих точек.

Еще одним алгоритмом, созданным в целях уменьшения вычислительной сложности, является ORB. На начальном этапе работы ORB формируется масштабная гауссова пирамида изображения. Затем, на каждом уровне пирамиды, определяются экстремумы функции яркости с использованием алгоритма FAST [16]. Согласно этому методу, для каждой точки изображения строится окружность определённого радиуса, и подсчитывается количество точек на окружности, яркость которых больше или меньше яркости центральной точки. Если таких точек оказывается достаточно от общего числа точек окружности, центральная точка считается кандидатом на роль особой точки. Кандидаты, расположенные на границах объектов, исключаются из дальнейшего рассмотрения [17].

Для расчёта дескриптора в окрестности особой точки выделяется область текущего масштаба, где определяется центр масс. Вектор от точки к центру масс задаёт её ориентацию, после чего формируется квадратное окно, выровненное с этой ориентацией. В окне сравниваются яркости пар точек: если яркость первой выше второй, записывается "1", иначе – "0". Таким образом, дескриптор представлен бинарным вектором.

Увеличение скорости извлечения ключевых точек с помощью алгоритма ORB достигается более простым методом вычисления дескрипторов, который представляет собой бинарный вектор, а не массив чисел или вектор множества признаков.

В таблице 2 представлено сравнение рассмотренных алгоритмов для поиска ключевых точек по таким критериям, как инвариантность по отношению к условиям съемки, применяемым вычислениям, содержанию дескриптора и его размеру. Данные критерии влияют на точность работы алгоритма и его быстродействие.

Таблица 2 – Сравнение алгоритмов извлечения ключевых признаков

Характеристика	Алгоритмы извлечения ключевых признаков		
	SIFT	SURF	ORB
Инвариантность	К масштабу, повороту, частично к освещению	К повороту, освещению, частично к масштабу	К масштабу, повороту
Применяемые вычисления	Разности гауссиан	Приближённая гауссова пирамида и детектор детерминанта Гессиана	Алгоритм FAST
Дескриптор	Гистограмма направлений градиентов	Массив чисел	Бинарный дескриптор, основанный на сравнении яркостей пар точек
Размер дескрипторов	128 элементов	64 элемента	Бинарный

Сравнение алгоритмов SIFT, SURF и ORB показывает их различную применимость в зависимости от требований к точности, устойчивости и производительности. Алгоритм SIFT инвариантен к изменению внешних факторов, но из-за высокой вычислительной сложности становится неприменимым в задачах, требующих быстрого распознавания и сопоставления ключевых точек.

Алгоритм SURF обладает большим быстродействием, но обладает не самыми эффективными характеристиками.

Алгоритм ORB является самым эффективным по скорости работы из рассмотренных благодаря методу детектирования ключевых признаков и размеру дескрипторов.

1.3. ВИЗУАЛЬНАЯ ОДОМЕТРИЯ

Алгоритмы извлечения ключевых признаков обеспечивают распознавание главных компонентов для работы систем, основанных на визуальной одометрии. Такие системы способны строить карты окружающего БПЛА пространства и восстанавливать положение камеры.

Визуальная одометрия – это метод оценки траектории движения устройства в трёхмерном пространстве на основе анализа последовательности изображений, полученных с одной или нескольких камер, находящихся на борту БПЛА.

Основной задачей визуальной одометрии является определение относительного перемещения устройства путём выявления изменений в положении и ориентации между кадрами. Для этого используются ключевые точки на изображениях, которые сопоставляются между собой, а затем на основе их смещения оцениваются параметры трансформации, такие как сдвиг, вращение и масштаб.

Системы визуальной одометрии могут использовать разные типы входных данных.

Монокулярная одометрия основана на анализе последовательности изображений, полученных с одной камеры. Для такой системы может быть достаточным использование камеры мобильного устройства при условии высокого разрешения и вычислительной мощности, что значительно упростит процесс калибровки.

Стереоскопическая одометрия использует данные двух камер для получения информации о глубине сцены, что позволяет напрямую

восстанавливать абсолютные параметры движения. Такой подход требует предварительной обработки изображения и синхронизированных значениях IMU [18].

Одометрия на основе глубины (RGB-D) сочетает изображение и карту глубины, предоставляемую датчиками, такими как камеры Kinect [19].

Алгоритм визуальной одометрии состоит из следующих этапов [20]:

1. Получение и выравнивание изображений со стереопар.

В стереозрении информацию о расположении объектов относительно друг друга возможно получить за счет явления параллакса – перспективного смещения объекта, зависящее от точки наблюдения.

2. Обнаружение и сопоставление ключевых точек.

Этот этап обеспечивает идентификацию особенностей, которые могут быть отслежены между кадрами. Ключевые точки могут быть найдены с помощью алгоритмов извлечения ключевых признаков, описанных ранее. Для устранения ложных сопоставлений применяется алгоритм RANSAC. С помощью данного алгоритма из множества соответствий извлекается такое подмножество, которое максимально соответствует геометрическим ограничениям [21].

3. Оценка параметров движения.

На основании сопоставленных точек вычисляются параметры движения устройства (сдвиг, вращение). Для оценки движения при расчетах используются данные о калибровке камеры и строится фундаментальная матрица [22]. Фундаментальная матрица описывает геометрическое соотношение между двумя изображениями одной и той же сцены. Фундаментальная матрица зависит от внешних параметров камеры (положения и ориентации), а также от внутренней калибровки камеры, если она известна. Если параметры камеры известны, то фундаментальная матрица преобразовывается в основную, откуда извлекаются параметры, описывающие изменение положения устройства.

4. Реконструкция 3D-сцены.

На основе координат ключевых точек и параметров движения восстанавливается трёхмерное положение объектов сцены [23]. Оптимальным методом для отображения положения является триангуляция. Триангуляция в контексте визуальной одометрии – процесс восстановления трёхмерных координат точки на основе её проекций на два или более изображений, сделанных с разных позиций камеры.

Для выполнения триангуляции используется информация о параметрах камер, включая их положение, ориентацию и внутреннюю калибровку (например, фокусное расстояние и положение оптического центра). На основании пиксельных координат точки на каждом изображении строятся проекционные линии, которые пересекаются в пространстве, определяя восстановленную трёхмерную точку. Если параметры камер известны, это сводится к решению системы уравнений, связывающих пиксельные координаты с пространственными координатами.

5. Построение локальной карты и обновление траектории.

Обнаруженные точки добавляются на локальную трехмерную карту сцены, которая обновляется с каждым новым кадром. Пример отображения ключевых точек на трехмерной сцене представлен на рисунке 4.

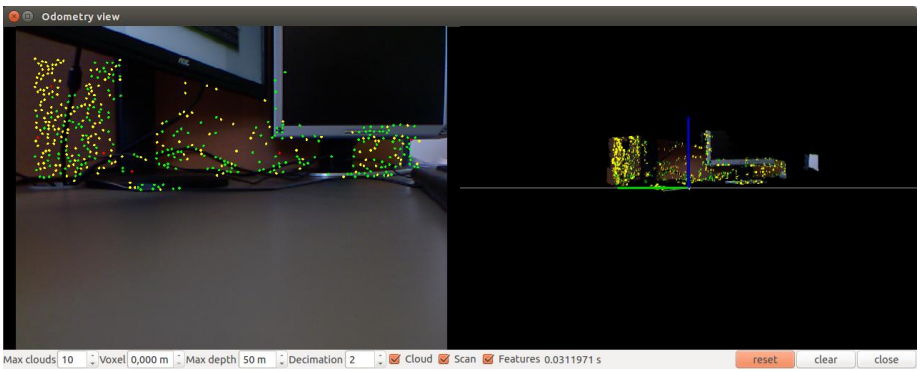

Рисунок 4 – Пример трехмерного отображения ключевых точек

К преимуществам систем, основанных на визуальной одометрии, можно отнести работу в режиме реального времени и получение большого количества информации об окружающей среде. Современные методы визуальной одометрии также интегрируют данные с инерциальных датчиков,

что позволяет повысить точность и устойчивость к временным потерям ключевых точек или движению камеры в текстурно-пустых областях.

Существенным недостатком является накопление ошибок (дрейф) при длительной работе. Малые ошибки в оценке движения на каждом шаге накапливаются, что приводит к значительным отклонениям от реальной траектории.

1.4. ОБЗОР АНАЛОГОВ

В настоящее время на рынке представлены системы, позволяющие отслеживать положение беспилотных летательных средств, осуществлять планирование маршрута и анализ изображений. Разрабатываемая в данной работе система должна обладать набором обозначенных признаков: автономная работа без GPS сигнала, возможность построения трехмерной карты местности по ключевым точкам, интеграция в существующую модель БПЛА. Поэтому аналоги будут рассматриваться по данным критериям.

1. Полетный контроллер ArduPilot. Система ArduPilot может автоматически управлять полетом, имеет несколько интегрированных инерциальных датчиков.

При выборе режима автономной навигации без использования спутниковых данных система позиционирования чувствительна к дрейфу инерциальных датчиков и требует большого объема памяти устройства.

К преимуществам данной системы можно отнести наличие открытого исходного кода, что обеспечивает интеграцию в существующую систему.

2. DJI Terra.

Программное обеспечение DJI Terra предназначено для реконструкции трехмерных моделей и выполнения задач картографирования. Система преобразует данные с бортовой камеры БПЛА в цифровые карты и облака точек для обработки в области геодезии, строительства и других отраслях.

DJI Terra поддерживает использование визуальной одометрии и других систем для позиционирования в условиях слабого GPS-сигнала. Однако её функциональность наиболее эффективна при наличии GPS.

Интеграция программного обеспечения в существующую систему ограничена, так как DJI Terra полностью зависима от оборудования компании.

3. UgCS (Universal Ground Control Software).

UgCS – программное обеспечение для управления полётами дронов, поддерживающее множество платформ и производителей. Оно используется

для сложного планирования маршрутов, мониторинга в реальном времени и работы с различными сенсорами.

UgCS осуществляет сбор данных с помощью лидаров, камер и инерциальных датчиков, осуществляет процессы сложных калибровок. Прямой поддержки методов извлечения ключевых точек в данной системе нет, как и метода оптического потока. Их использование возможно при подключении дополнительных модулей.

Характеристики основных параметров систем представлены в таблице 3.

Таблица 3 – Сравнение характеристик систем навигации БПЛА

Параметр	Системы управления и отслеживания движения БПЛА		
	ArduPilot	DJI Terra	UgCS
Возможность работы без GPS	Возможна (при использовании алгоритмов обработки изображений и интеграции инерциальных сенсоров и дополнительных датчиков)	Ограниченная поддержка. Используется для кратковременной стабилизации при слабом GPS	Поддерживается через интеграцию сторонних камер и сенсоров
Зависимость от оборудования	Независимая	Полностью аппаратно зависима от оборудования DJI	Независимая
Сложность настройки	Высокая, требует настройки оборудования	Готовое решение только для платформ DJI	Высокая, нет готовых решений для необходимых методов
Интеграция в российские системы	Открытый исходный код	Ограниченная, так как система закрытая и полностью зависима от оборудования DJI	Возможна через API
Стоимость	Открытый исходный код	Лицензионная	Лицензионная

Продолжение таблицы 3

Параметр	Системы управления и отслеживания движения БПЛА		
	ArduPilot	DJI Terra	UgCS
Зависимость от оборудования	Независимая	Полностью аппаратно зависима от оборудования DJI	Независимая
Сложность настройки	Высокая, требует настройки оборудования	Готовое решение только для платформ DJI	Высокая, нет готовых решений для необходимых методов
Интеграция в российские системы	Открытый исходный код	Ограниченная, так как система закрытая и зависима от оборудования DJI	Возможна через API
Стоимость	Бесплатная	Лицензионная	Лицензионая
Преимущества	Бесплатная и открытая система	Готовое решение с высокой надёжностью	Универсальность, поддержка множества платформ и устройств
Недостатки	Требует значительной настройки для интеграции	Зависимость от оборудования DJI, закрытый код	Высокая стоимость лицензии, требует ресурсов для интеграции

Сравнение систем ArduPilot, DJI Terra и UgCS показывает, что каждая из них имеет свои преимущества и ограничения. ArduPilot выделяется гибкостью и открытым кодом, но требует значительной настройки. DJI Terra предлагает готовое и надёжное решение, но ограничено аппаратной зависимостью и закрытым кодом. UgCS обладает широкой универсальностью и поддержкой различных платформ, но её высокая стоимость и сложность интеграции могут ограничивать применение. Таким образом, ни одна из описанных систем не обладает всеми характеристиками, необходимыми для достижения поставленной цели.

2. ИССЛЕДОВАНИЕ АЛГОРИТМОВ ДЛЯ ПРИМЕНЕНИЯ В СИСТЕМЕ ОПТИЧЕСКОГО УДЕРЖАНИЯ ПОЗИЦИИ БПЛА

Автономный полет беспилотных летательных аппаратов требует решения двух ключевых задач: построения карты окружающей среды и ее уточнения и определения собственного положения в пространстве с последующим его пересчетом. В большинстве случаев точная карта местности отсутствует, поэтому летательному аппарату необходимо рассчитывать ее самостоятельно на основе данных с бортовых камер и сенсоров. В условиях неэффективности распространенных методов навигации (например, GPS) поставленные задачи решаются с помощью алгоритмов одновременной локализации и построения карты – SLAM (Simultaneous Localization And Mapping) алгоритмов, которые позволяют дрону ориентироваться, используя только визуальные и сенсорные данные.

SLAM – алгоритмы обеспечивают возможность построения карты местности в режиме реального времени и определения местоположения БПЛА относительно этой карты. Такой результат достигается за счет обработки последовательностей изображений и дополнительных измерений, таких как данные инерционных сенсоров. В настоящее время используются два различных метода – визуальный SLAM и Lidar SLAM.

В данной главе рассматриваются ключевые алгоритмы визуального SLAM, применяемые для оптического удержания и навигации БПЛА, такие как FastSLAM, ViSLAM и ORB-SLAM.

2.1. ОБЩИЕ ПРИНЦИПЫ РАБОТЫ SLAM-АЛГОРИТМОВ

Алгоритмы SLAM решают задачу одновременного построения карты окружающего пространства и определения местоположения беспилотного летательного аппарата на этой карте. Несмотря на большое количество различных подходов к реализации SLAM, их работу можно свести к трем основным процессам, каждый из которых включает в себя ряд подзадач (рисунок 5).

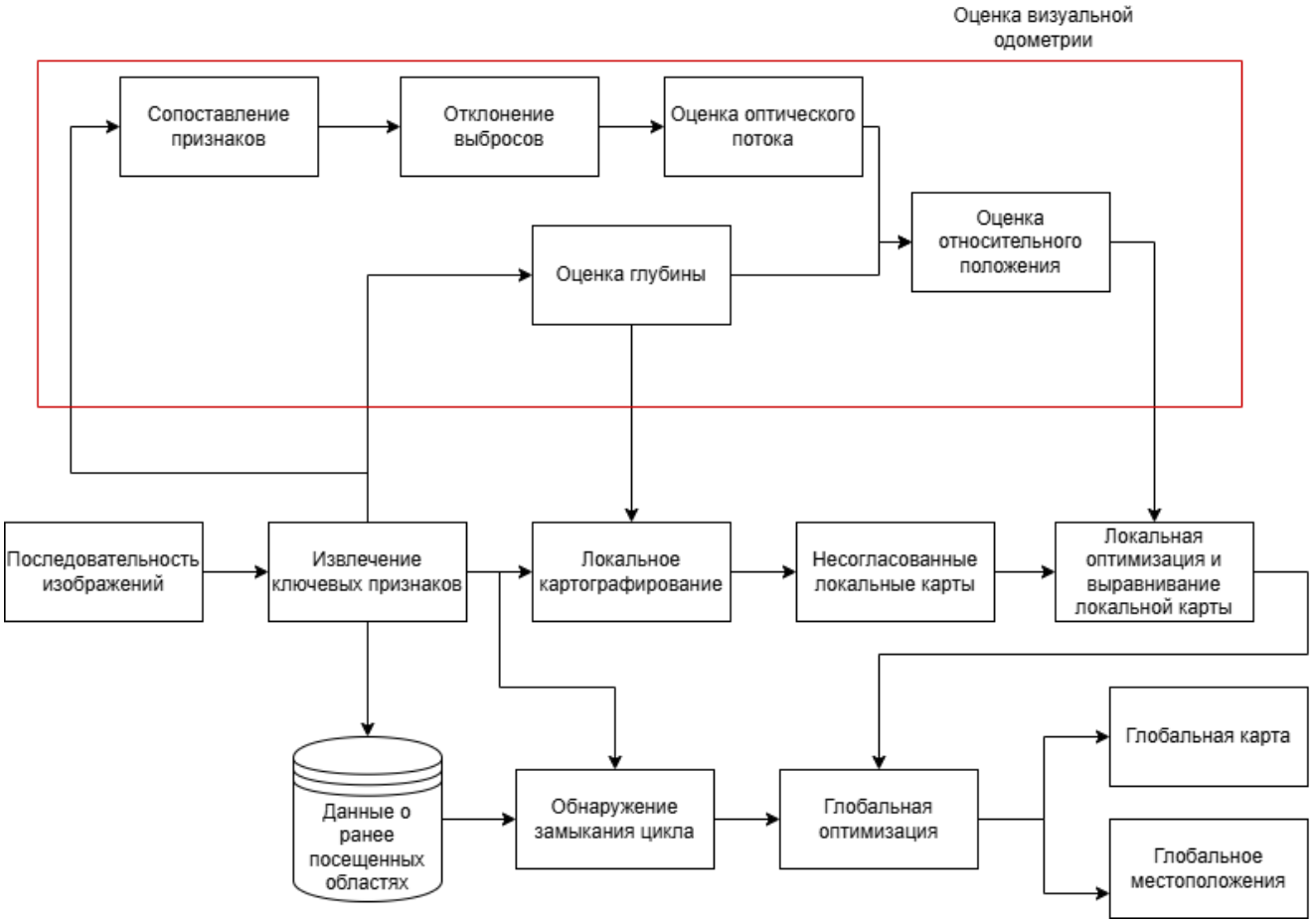

Рисунок 5 – Схема работы SLAM-систем

К первому этапу работы алгоритма относится извлечение ключевых признаков и визуальная одометрия. Алгоритмы визуального SLAM могут работать только с видеопотоком, что исключает возможность использования датчиков для извлечения облака точек. Поэтому ключевые точки детектируются непосредственно на наборе последовательных кадров. Для этого используются алгоритмы вычисления ключевых признаков (SIFT, SURF, ORB и другие), которые включают в себя детекторы углов, краев, контрастных областей и построение дескриптора каждой такой точки. Визуальная одометрия оценивает перемещение камеры, анализируя изменение видимого движения объектов на серии изображений.

В свою очередь, алгоритм визуальной одометрии включает в себя сопоставление ключевых точек от кадра к кадру, оценку оптического потока (расчет смещения с течением времени) и отклонение выбросов. Таким образом, результатом первого процесса SLAM-алгоритмов являются

извлечение признаков и оценка их движения, которые используются для последующих шагов.

Вторым ключевым этапом в работе алгоритма являются локальная оптимизация и картографирование. Визуальные особенности на последовательных изображениях – это двухмерные точки и их дескрипторы. Основной целью картографирования является сопоставление двухмерных точек с их соответствующими местоположениями на трехмерной сцене, что представляет из себя цель SLAM-алгоритмов. Результат достигается за счет получения точных параметров камеры, определения соответствия между кадрами и триангуляции.

После построения локальных карт местности возникает проблема накопления ошибок, так как точки проецируются на трехмерную сцену с каждого полученного кадра. Этап локальной оптимизации направлен на повторное выравнивание значений последовательных карт. Способ локальной оптимизации зависит от выбранного SLAM- алгоритма: это может быть оптимизация графа, фильтр частиц и другие.

Результатом картографирования и локальной оптимизации является наличие множества локальных синхронизированных карт, поэтому заключительным этапом работы алгоритма будет глобальная оптимизация и замыкание цикла.

Замыкание цикла – процесс, который заключается в выравнивании облака точек, обнаруженных ранее. При повторном распознавании посещенной области корректируются любые отклонения карты или траектории с течением времени, а также обновляется глобальная карта и положение камеры на ней.

Далее происходит выравнивание глобальной карты, что приводит к получению финального результата работы алгоритма локализации и картографирования: карты местности и местоположения БПЛА относительно этой карты.

2.2. АЛГОРИТМ FastSLAM

FastSLAM – это алгоритм одновременной локализации и построения карты, который использует фильтры частиц для оценки траектории робота и представление карты с помощью набора гауссовых распределений [24]. Алгоритм FastSLAM основан на фильтре частиц (гипотез о положении робота и соответствующие карты) и применении байесовской сети [25]. Все наблюдения производятся независимо друг от друга при условии безошибочного определения положения робота.

При работе алгоритма рассчитывается вероятность для каждого состояния робота, представленным набором частиц. Алгоритм FastSLAM обновляет частицы по мере того, как робот движется и наблюдает за своим окружением. Данные наблюдения используются для расчета вероятности каждой частицы, и они повторно выбираются в соответствии с их вероятностями. Этот процесс приводит к набору частиц, которые представляют текущую позу робота и оценку карты.

Так как алгоритм основывается на вероятностях, его основным недостатком является вырождение частиц. В таком случае все частицы имеют нулевой вес и эффективность алгоритма снижается.

2.3. АЛГОРИТМ ViSLAM

ViSLAM – это алгоритм локализации и картографирования, который обрабатывает визуальные и инерциальные данные. Такой подход позволяет работать системам отслеживания движения более устойчиво, так как алгоритм использует различные источники данных.

Основной принцип ViSLAM заключается в одновременной обработке двух источников информации: данных с камеры, которые позволяют определять ключевые точки, строить карту и определять относительное перемещение и данные IMU (акселерометр, гироскоп), которые предоставляют данные о линейных ускорениях и угловых скоростях.

Данные IMU используются для интеграции ориентации и положения между последовательными изображениями, обратной коррекции (drift correction) – снижения накопленной ошибки, обнаружения и компенсации потерь визуального трекинга в моменты резких движений или при нехватке визуальных ориентиров.

К достоинствам таких систем относится устойчивость к резким движениям, вибрациям и кратковременной потере визуальных ориентиров, возможность работы в условиях слабой текстурированности, уменьшение накопленной ошибки за счёт IMU-коррекции.

Существуют ограничения для работы методов, основанных на визуально-инерциальных данных, поскольку требуется точная калибровка не только камеры, но и IMU для предотвращения накопления ошибки.

К недостаткам ViSLAM можно отнести высокую вычислительную сложность по сравнению с визуальным SLAM, поскольку данные с камеры и датчиков должны быть синхронизированы, и необходимость внешней коррекции для устранения дрейфа.

2.4. АЛГОРИТМ ORB-SLAM3

ORB-SLAM3 – это полноценная система, способная работать в визуальном или визуально-инерциальном режимах с монокулярными, стерео или RGB-D камерами [26]. Она способна вычислять траекторию камеры в реальном времени и строить разреженную трехмерную реконструкцию сцены различных размеров.

Название ORB-SLAM обусловлено тем, что этот алгоритм использует признаки Oriented FAST и Rotated BRIEF (ORB) для обнаружения и сопоставления ключевых точек на последовательных изображениях. Еще одной особенностью алгоритма является наличие дополнительных методов для повышения точности работы, таких как замыкание петель (закрытие цикла) и оптимизация позы.

Закрытие цикла – это процесс выявления случаев, когда алгоритм распознает ранее посещенные области. ORB-SLAM3 сопоставляет полученные изображения с сохраненными ключевыми кадрами. Если алгоритм обнаруживает, что текущий кадр соответствует кадру из прошлого, он вносит соответствующие корректировки в построенную карту. Это позволяет уменьшить накопленные ошибки и устранить дрейф, который возникает при последовательном обновлении позиции без глобального пересчета.

Оптимизация позы – это метод, используемый для повышения точности локализации путем минимизации ошибок в оценке положения камеры с помощью вычисления поворота и смещения камеры относительно предыдущего кадра [27].

Благодаря сочетанию ORB-признаков, механизма закрытия цикла и оптимизации позы ORB-SLAM достигает высокой точности и стабильности при решении задач одновременной локализации и построения карты.

На основе анализа SLAM алгоритмов можно сделать вывод, что ORB-SLAM3 является наиболее подходящим для решения задачи автономной навигации, так как обладает меньшими вычислительными затратами за счет способа детектирования ключевых признаков и предлагает более точную локализацию и построение карты благодаря механизмам закрытия цикла и оптимизации позы.

3. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ПРОГРАММНОГО МОДУЛЯ ОТСЛЕЖИВАНИЯ ДВИЖЕНИЯ БПЛА ОТНОСИТЕЛЬНО ОКРУЖАЮЩЕЙ СРЕДЫ НА ОСНОВЕ ПОСЛЕДОВАТЕЛЬНЫХ ИЗОБРАЖЕНИЙ

В данной главе рассматривается процесс проектирования и реализации программного модуля, предназначенного для оценки перемещения БПЛА относительно окружающей среды на основе последовательных изображений. Основной задачей модуля является локализация и построение карты с использованием алгоритма ORB-SLAM3. Для передачи и обработки данных применяется фреймворк ROS2.

3.1. АЛГОРИТМ ЛОКАЛИЗАЦИИ И КАРТОГРАФИРОВАНИЯ И ОПИСАНИЕ ЕГО ВХОДНЫХ КОМПОНЕНТОВ

В качестве алгоритма локализации и картографирования был выбран алгоритм ORB-SLAM3. К его преимуществам по сравнению с другими алгоритмами, рассмотренными ранее, относится:

1. Высокая скорость сопоставления ключевых точек благодаря методу их извлечения и описания (используются бинарные дескрипторы).
2. Поддержка режима работы с монокулярной камерой. Данный фактор является ключевым при выборе алгоритма локализации и картографирования, так как для получения изображений используется монокулярная бортовая камера. Помимо монокулярного режима, алгоритм способен работать со стереокамерой, камерой глубины и дополнительными датчиками.
3. Наличие методов закрытия цикла. Данный метод предотвращает случаи ошибочного определения траектории движения или дрейфа БПЛА из-за нахождения летательного аппарата на ранее посещенной местности. Отсутствие методов распознавания таких областей приводит к значительному накоплению ошибки.

4. Возможность интеграции с другими компонентами модуля.

На рисунке 6 представлена схема работы алгоритма ORB-SLAM3.

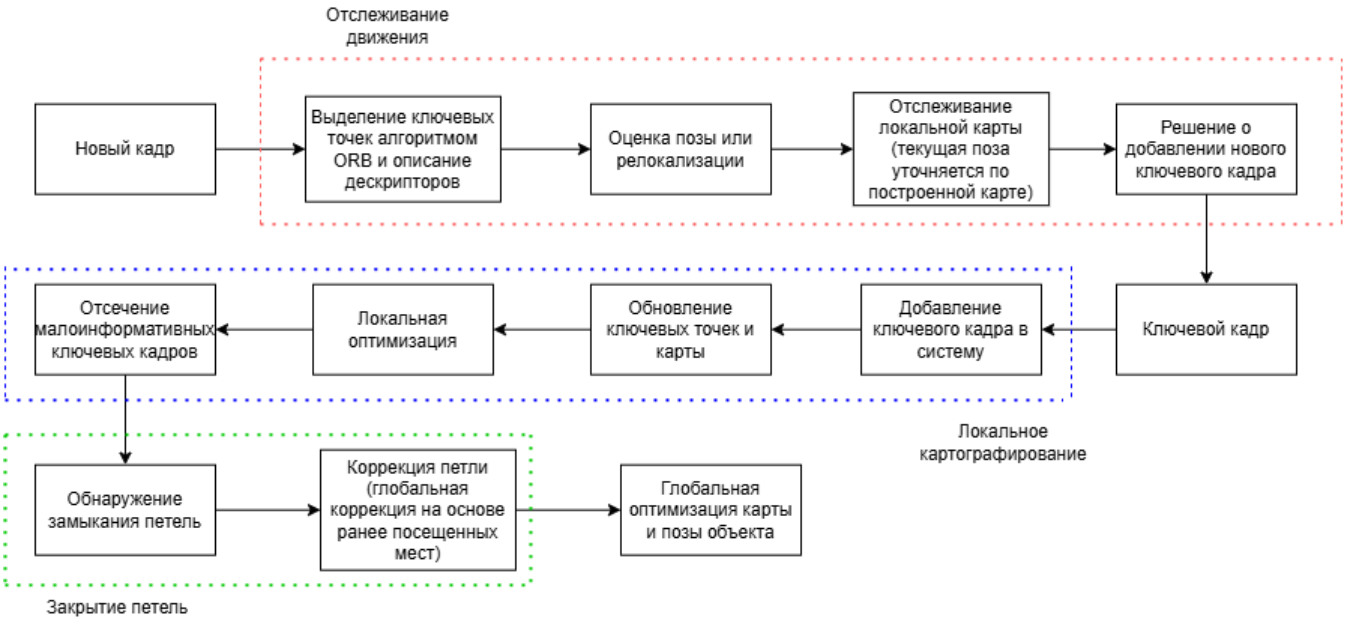

Рисунок 6 – Схема работы алгоритма ORB-SLAM3

Алгоритм ORB-SLAM3 обеспечивает отслеживание движения при наличии входных параметров. К ним относится файл конфигурации камеры и словарь признаков.

Конфигурационный файл камеры содержит информацию о внутренней калибровке камеры, которая необходима для корректной обработки изображений. К конфигурационным параметрам камеры относится:

- фокусное расстояние fx, fy. Фокусное расстояние – это оптическое расстояние от точки схождения лучей света внутри объектива до матрицы камеры. Значение этого параметра позволяет учесть масштабные искажения из-за особенностей оптической системы камеры;
- оптический центр cx, cy Это координаты центра изображения, относительно которых производится расчет;
- дисторсия – это параметры искажения изображения, которые необходимо учесть для корректной обработки изображения, вызванные

несоответствием между фактической геометрией объектов на сцене и их представлением из-за использования оптических систем, где линзы и зеркала могут вносить аномалии в прохождение света;

- размеры изображения.

Другим входным компонентом алгоритма является словарь ORB-признаков.

Словарь ORB-признаков – это предобученный набор визуальных признаков, который используется в SLAM-алгоритме для представления изображений, позволяющий сопоставлять кадры между собой, выполнять распознавание уже посещённых мест и повышать устойчивость локализации в условиях изменений окружения. Словарь не требует дополнительной настройки, поскольку набор признаков основывается на обучающих данных.

Таким образом, алгоритм локализации и картографирования ORB-SLAM3 представляет собой подходящее решение для построения карты окружающей среды и отслеживания положения БПЛА на основе последовательных изображений. Для эффективной работы алгоритма в программном модуле требуется обеспечить корректную передачу входных данных и видеопотока с помощью коммуникационной среды. В данной работе для этих целей был выбран фреймворк ROS2 (Robot Operating System 2), который предоставляет необходимые инструменты для организации потоков данных и интеграции алгоритма локализации и картографирования.

3.2. ОРГАНИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ КОМПОНЕНТОВ ПРОГРАММНОГО МОДУЛЯ НА БАЗЕ ФРЕЙМВОРКА ROS2

ROS 2 (Robot Operating System 2) – это коммуникационный фреймворк и программная платформа с открытым исходным кодом, предназначенная для разработки робототехнических систем, позволяющая различным частям систем робота обнаруживать, отправлять и получать данные.

Фреймворк ROS 2 реализует архитектурную модель, основанную на обмене сообщениями между независимыми программными компонентами. В данной работе осуществлялось взаимодействие с такими элементами фреймворка, как нода, топик, публикация, подписчик и сообщения.

1. Нода или базовая единица вычислений в ROS2. Это независимая исполняемая программа. Ноды запускаются как процессы публикации или подписки, обработки данных.
2. Топик – это именованный канал. С помощью топиков происходит обмен сообщений между единицами вычислений.
3. За отправку данных по именованным каналам в ROS2 отвечают публикации. Они создают объекты и отправляют сообщения.
4. Подписчик – это процесс, который получает сообщения из топика.
5. Между нодами передаются данные, называемые сообщениями. В ROS2 типы сообщений стандартизированы, но есть возможность создания собственных типов.

3.3. АРХИТЕКТУРА ПРОГРАММНОГО МОДУЛЯ ДЛЯ ОТСЛЕЖИВАНИЯ ДВИЖЕНИЯ БПЛА

Архитектура программного модуля разработана с учётом функционала, связанного с получением входных данных, необходимых для работы SLAM алгоритма, получением и обработкой изображений, выполнением алгоритма SLAM, а также с демонстрацией результатов в виде трехмерной сцены и записи траектории. Архитектура программного модуля представлена на рисунке 7.

Проектируемый программный модуль состоит из 4 ключевых компонентов:

1. Обработки и публикации изображений в соответствующий топик на базе фреймворка ROS2.
2. Запуска алгоритма ORB-SLAM3 как отдельной ноды ROS2 с входными данными: калибровочным файлом камеры и словарем признаков.
3. Локализации и картографирования.
4. Представления результата в виде воспроизведения видеопотока с нанесенными маркерами ключевых точек каждого кадра, реконструкции траектории движения на трехмерной сцене и записи вычисленной траектории в текстовый файл.

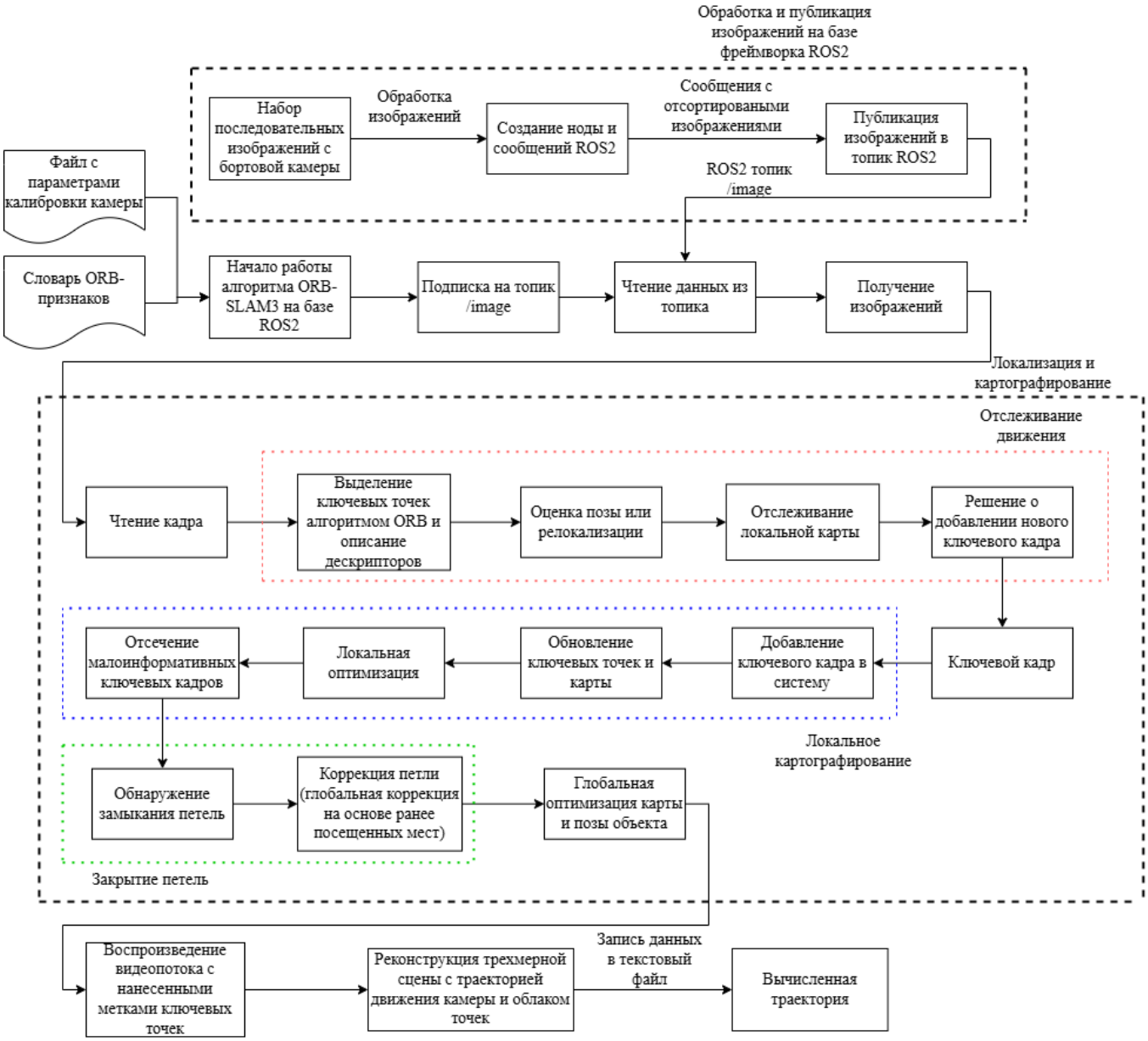

Рисунок 7 – Архитектура программного модуля

3.4. РЕАЛИЗАЦИЯ ПРОГРАММНОГО МОДУЛЯ

В рамках реализации программного модуля была использована версия ROS2 Foxy, так как она является одной из наиболее стабильных версий, рекомендованных для долгосрочного использования. Выбор ROS2 Foxy обусловлен её совместимостью с алгоритмом ORB-SLAM3.

Фреймворк ROS2 ориентирован на использование в операционной системе Linux, и на сегодняшний день наилучшая производительность и совместимость обеспечиваются именно в этой среде. В связи с этим

разработка и тестирование программного модуля осуществлялись в операционной системе Ubuntu 20.04 LTS, которая является официально поддерживаемой платформой для версии ROS2 Foxy.

На начальном этапе работы программного модуля формируется последовательность изображений с камеры. Так как камера является монокулярной, алгоритм ORB-SLAM3 запускается в соответствующем режиме. Перед началом работы системы необходимо предоставить два конфигурационных файла: файл параметров калибровки камеры и словарь ORB-признаков.

Калибровочный файл содержит параметры внутренней матрицы камеры, коэффициенты дисторсии и размеры изображений, полученные в результате калибровки с использованием утилиты, предоставляемой в рамках ROS2 – пакета «camera_calibration». Процесс калибровки включает в себя демонстрацию шахматного узора с указанием количества перекрестий этого узора и размера клетки. После получения необходимого количества снимков шахматного узора в различных ракурсах вычисляются внутренние характеристики камеры.

В качестве пробного запуска системы была откалибрована монокулярная веб-камера марки Logitech модели C270.

Для передачи изображения в топик «/image_raw» фреймворка ROS2 была использована утилита «Opencv_cam». Для получения изображений с нужного устройства и возможностью передачи в ROS2 необходимо запустить утилиту со следующими аргументами: ros2 run opencv_cam opencv_cam_main --ros-args -p device:="/dev/video1", где ros2 run – запуск отдельной ноды, opencv_cam – имя пакета, opencv_cam_main – имя исполняемого файла, который публикует изображения в топик, device:="/dev/video1" – путь к видеоустройству, являющийся ROS2 параметром. Информация о топиках, которые передаются в среде ROS2, представлена на рисунке 8. На рисунке видно, что в данный момент в топике «/image_raw» содержится как публикация, так и подписка.

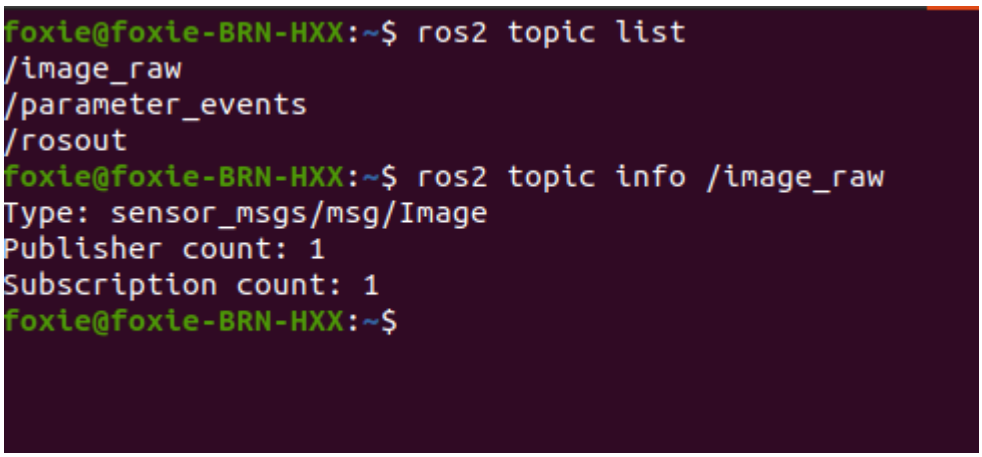

Рисунок 8 – Информация о топике передачи изображений

После запуска алгоритма ORB-SLAM3 с входными параметрами и настройки входного видеопотока можно наблюдать открытие двух окон, представленных на рисунке 9 и 10: с демонстрацией видеопотока с нанесенными метками ключевых точек и с демонстрацией трехмерной сцены с положением камеры на ней.

Окно с демонстрацией кадра содержит информацию о количестве ключевых точек, найденных на изображении. Максимальное количество таких точек можно задать в конфигурационном файле камеры, но этот параметр значительно влияет на скорость работы алгоритма. Для тестового запуска было выбрано 1000 ключевых точек.

Трехмерная сцена демонстрирует положение камеры и ее перемещение в реальном времени, а также проекцию облака точек, найденных на каждом кадре видеопотока.

Рисунок 9 – Просмотр кадра с метками ключевых точек

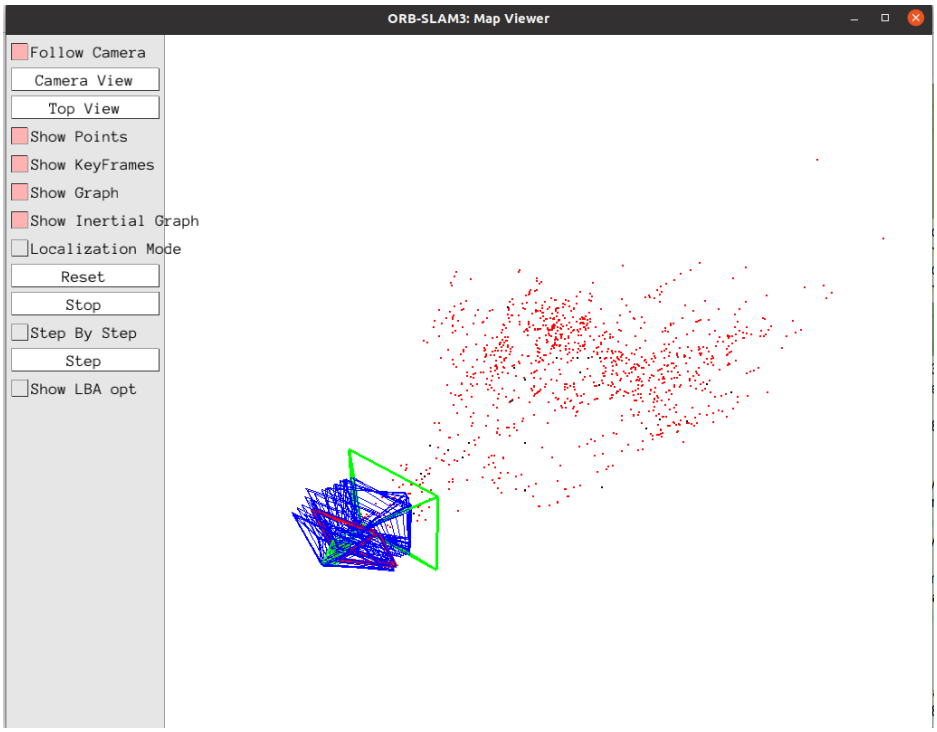

Рисунок 10 – Просмотр трехмерной сцены

После выполнения тестового запуска алгоритма локализации и картографирования на видеопотоке с веб-камеры была подтверждена его работоспособность. Следующим этапом стало использование реальных

изображений, полученных с бортовой камеры беспилотного летательного аппарата. Эти данные представляли собой последовательность кадров в формате PNG, сохранённых на диске.

В отличие от потока с реального устройства, такие изображения не могут быть напрямую переданы в SLAM-алгоритм через ROS2, так как отсутствует непрерывный видеопоток и публикация в соответствующий топик. Для обеспечения интеграции этих изображений с ROS2 было необходимо реализовать отдельную программу, выполняющую чтение изображений из каталога, их преобразование в формат ROS-сообщений и публикацию в заданный топик.

Программа была написана на языке Python, так как ROS 2 предоставляет полную поддержку разработки как на C++, так и на Python. Последний был выбран как более удобный для задач чтения, обработки и публикации изображений.

В качестве основы для реализации была выбрана библиотека rclpy – официальная клиентская библиотека ROS2 для языка Python. Она обеспечивает все необходимые функции для создания нод, объявления публикаций и подписок, управления таймерами, логирования и взаимодействия с ядром ROS. Разработанная нода получила название «image_publisher» и публиковала изображения в топик «/image_raw». Частота задаётся таймером. Таким образом удаётся добиться поведения, приближенного к реальному видеопотоку с камеры.

Все изображения предварительно сортируются с использованием регулярного выражения, чтобы сохранить хронологический порядок кадров, что необходимо для корректной работы алгоритма, правильного воспроизведения видеопотока и возможности восстановления траектории движения.

Исходный код для публикации изображений представлен в листинге 1.

Листинг 1 – Исходный код для публикации изображений

```
import rclpy
```

Продолжение листинга 1

```
    from rclpy.node import Node
    from sensor_msgs.msg import Image
    from cv_bridge import CvBridge
    import cv2
    import re
    import time
    #Функция сортировки изображений по имени
    def sort_key(filename):
        return [int(text) if text.isdigit() else text for text in
re.split(r'(\d+)', filename)]
    #Класс узла ROS 2, публикующего изображения
    class ImagePublisher(Node):
        def __init__(self, image_folder):
            super().__init__('image_publisher') #Инициализация узла с именем
«image_publisher»
            self.publisher = self.create_publisher(Image, '/image_raw', 10)
#Создание публикации для топика «/image_raw»
            self.bridge = CvBridge()#Объект CvBridge для преобразования
изображений OpenCV в ROS-сообщения
            self.image_folder = image_folder
            self.image_list =
sorted(os.listdir(self.image_folder),key=sort_key)
            self.timer = self.create_timer(0.1, self.publish_image) #Таймер,
вызывающий публикацию изображений каждые 0.1 сек

            self.index = 0

       #Функция публикации изображений и логирования отправки
        def publish_image(self):
            if self.index >= len(self.image_list):
                self.get_logger().info("Все изображения отправлены")
                return

            img_path = os.path.join(self.image_folder,
self.image_list[self.index])
            frame = cv2.imread(img_path)

            if frame is None:
                self.get_logger().warn(f"Не удалось загрузить {img_path}")
                return

            msg = self.bridge.cv2_to_imgmsg(frame, encoding="bgr8")
#Преобразование изображения в ROS-сообщение
            self.publisher.publish(msg)

            self.get_logger().info(f"Опубликовано: {img_path}")
            self.index += 1

    def main(args=None):
        rclpy.init(args=args) #Инициализация ROS2
        #image_publisher =
ImagePublisher('olesya_repo/dji_data_1/videos/DJI_20250122153829_0147_D.mp4')
        #image_publisher =
ImagePublisher('olesya_repo/dji_data_1/videos/DJI_20250204164411_0155_D.mp4')
        image_publisher =
ImagePublisher('/home/foxie/Desktop/ilya_repo/dji_data_1/videos/DJI_202502201
35015_0169_D.mp4')
        rclpy.spin(image_publisher) #Запуск цикла событий ROS2
        image_publisher.destroy_node()#Уничтожение узла
        rclpy.shutdown()#Завершение работы ROS 2
```

Окончание листинга 1

```
#Точка входа в программу
if __name__ == '__main__':
    main()
```

Реализация публикации изображений в ROS 2 была выполнена в виде программы, основанной на объектно-ориентированном подходе с использованием класса ImagePublisher, который наследуется от базового класса Node, предоставляемого библиотекой rclpy.

В конструкторе класса ImagePublisher выполняется несколько ключевых инициализаций. Во-первых, создаётся публикация с помощью метода self.create_publisher, который инициализирует публикацию сообщений типа sensor_msgs.msg.Image в топик «/image_raw». Во-вторых, инициализируется объект CvBridge, необходимый для преобразования изображений OpenCV в ROS-сообщения. Далее программа считывает все изображения из указанной директории и сортирует их в правильном порядке с помощью функции sort_key, которая использует регулярные выражения для корректной сортировки файлов с числовыми индексами в названии.

Также внутри конструктора создаётся таймер через функцию self.create_timer, вызывающую метод publish_image через заданный интервал (0.1 секунды, что соответствует 10 кадрам в секунду). Этот метод является основным обработчиком публикации изображений. Внутри него происходит поочерёдная загрузка изображений из списка файлов, их преобразование в сообщение ROS формата Image и публикация в заданный топик.

Перед публикацией каждое изображение загружается с диска функцией cv2.imread. В программе предусмотрено логирование чтения файлов. Если изображение успешно загружено, оно преобразуется в ROS-сообщение с помощью метода cv2_to_imgmsg(), указывается цветовое пространство bgr8 и публикуется в топик. Также осуществляется логирование успешной публикации изображения с указанием его пути. Индекс изображения увеличивается, и следующий кадр будет обработан при следующем вызове таймера.

Функция main() отвечает за инициализацию ROS-среды вызовом rclpy.init(), создание экземпляра класса ImagePublisher с передачей пути к директории изображений и запуск бесконечного цикла обработки событий через rclpy.spin(). После завершения публикации вызывается метод destroy_node() и происходит завершение работы ROS-ноды.

Для корректной работы SLAM алгоритма были получены данные о бортовой камере и записаны в конфигурационный файл, представленный в листинге 2.

Листинг 2 – Файл внутренней конфигурации камеры

```
%YAML:1.0

#--------------------------------------------------------------------------
--------------------
# Camera Parameters. Adjust them!
#--------------------------------------------------------------------------
--------------------
Camera.type: "PinHole"
Image_topic: "/image_raw"
# Camera calibration and distortion parameters (OpenCV)
Camera.fx: 1120.00
Camera.fy: 1120.00
Camera.cx: 640.00
Camera.cy: 360.00
Camera.k1: 0.0
Camera.k2: 0.0
Camera.p1: 0.0
Camera.p2: 0.0
Camera.k3: 0.000000

# Camera frames per second
Camera.fps: 30.0
# Color order of the images (0: BGR, 1: RGB. It is ignored if images are
grayscale)
Camera.RGB: 1

# Camera resolution
Camera.width: 1280
Camera.height: 720

#--------------------------------------------------------------------------
--------------------
# ORB Parameters
#--------------------------------------------------------------------------
--------------------

# ORB Extractor: Number of features per image
ORBextractor.nFeatures: 1500

# ORB Extractor: Scale factor between levels in the scale pyramid
ORBextractor.scaleFactor: 1.2

# ORB Extractor: Number of levels in the scale pyramid
ORBextractor.nLevels: 8
```

Окончание листинга 2

```
    # ORB Extractor: Fast threshold

    # Image is divided in a grid. At each cell FAST are extracted imposing a
minimum response.
    # Firstly we impose iniThFAST. If no corners are detected we impose a lower
value minThFAST
    # You can lower these values if your images have low contrast
    ORBextractor.iniThFAST: 10
    ORBextractor.minThFAST: 7
```

Блок «Camera Parameters» конфигурационного файла является основным и определяет внутренние параметры камеры, которые необходимы для правильного проецирования изображений и построения карты окружающей среды. В него входят следующие параметры:

1. Camera.type. В данном случае используется модель с центральной проекцией ("PinHole"), которая подходит для монокулярных камер.
2. Image_topic – имя топика ROS 2, из которого SLAM будет получать изображения.
3. Camera.fx, Camera.fy, Camera.cx, Camera.cy: параметры внутренней калибровки камеры – фокусные расстояния по осям x и y, координаты оптического центра.
4. Camera.k1, k2, p1, p2, k3 – коэффициенты дисторсии объектива. В данном случае все значения равны нулю, так как отсутствуют искажения или они предварительно устранены.
5. Camera.fps – частота кадров, поступающих с камеры.
6. Camera.width и Camera.height – разрешение изображений, подаваемых на вход системы.
7. ORBextractor.nFeatures – максимальное количество ключевых точек. Для работы с реальными данными параметру было задано значение 1500.

Далее был запущен программный модуль, включающий как ROS2 узел для публикации изображений, так и SLAM-алгоритм, функционирующий в монокулярном режиме. Изображения, полученные с бортовой камеры, последовательно передавались в топик «/image_raw», откуда они принимались

алгоритмом ORB-SLAM3 и использовались для локализации и построения карты.

В ходе работы модуля была протестирована последовательность кадров, снятая при прямолинейном движении без поворотов. Это позволило проанализировать стабильность траектории, получаемой в результате работы алгоритма. На видеопотоке на рисунке 11 можно наблюдать ключевые точки, обнаруженные на каждом кадре, а в отдельном окне трехмерной сцены – реконструкцию траектории движения БПЛА.

Рисунок 11 – Демонстрация текущего кадра с ключевыми точками

Следует отметить, что, несмотря на возможность камеры снимать цветные изображения, ключевые признаки кадров определяются на черно-белых снимках, так как извлечение ключевых точек является оптимальным при наивысшей контрастности. На рисунках 12 и 13 видно, что траектория движения является прямой и устойчивой, что подтверждает корректную работу программного модуля в условиях отсутствия резких изменений траектории движения.

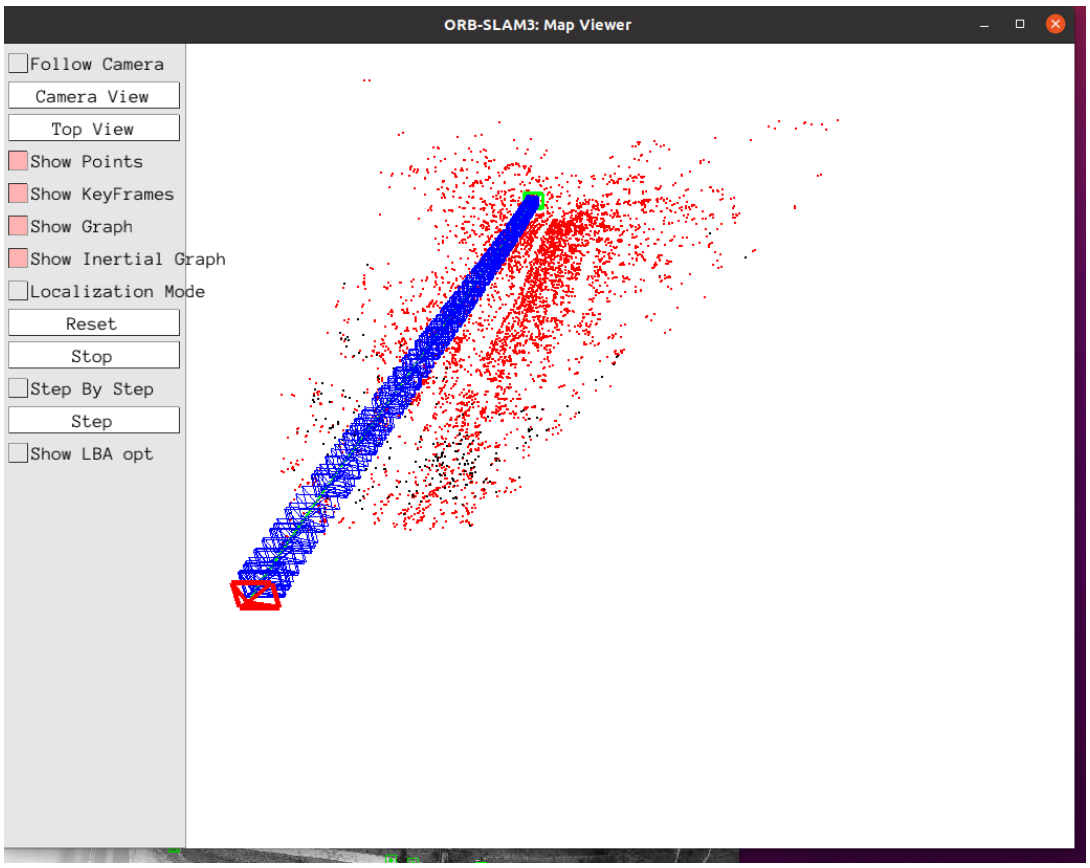

Рисунок 12 – Отслеживание прямолинейного движения

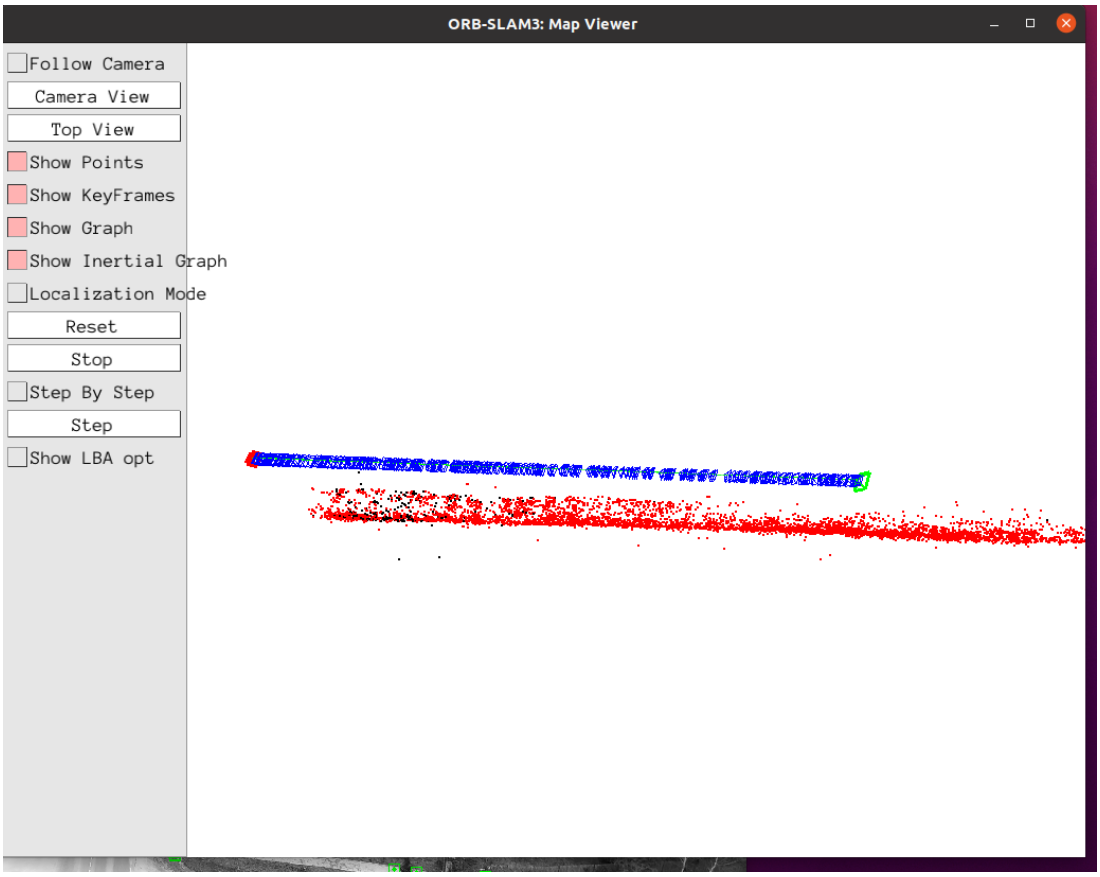

Рисунок 13 – Отслеживание прямолинейного движения

После завершения работы алгоритма ORB-SLAM3 траектория движения записывается в текстовый файл, что позволяет впоследствии проанализировать траекторию. Такой файл представляет собой последовательность записей, каждая из которых соответствует одному временному моменту кадра, обработанному алгоритмом ORB-SLAM3.

Каждая строка в этом файле содержит следующие данные:

$< timestamp >< tx >< ty >< tz >< qx >< qy >< qz >< qw >$,

где $< timestamp >$ – временная метка в миллисекундах, которая соответствует времени захвата кадра. Это позволяет синхронизировать траекторию с другими источниками данных, такими как спутниковые системы или инерциальные данные. Так как целью работы является отслеживания движения только на последовательных кадрах, временная метка не является информативным параметром, так как алгоритм не способен самостоятельно определить разницу между кадрами;

$< tx >< ty >< tz >$ – координаты позиции камеры в локальной системе координат по осям x, y и z соответственно. Эти значения отражают путь, пройденный камерой в процессе обработки изображения;

$< qx >< qy >< qz >< qw >$ – кватернион, описывающий ориентацию камеры в пространстве.

Ориентация камеры или летательного аппарата обычно записывается в углах Эйлера, представляющих собой способ описания поворота тела в трёхмерном пространстве с помощью вращений вокруг координатных осей: roll (крен, вращение вокруг продольной оси x), pitch (тангаж, вращение вокруг поперечной оси y) и yaw (рыскание, вращение вокруг вертикальной оси z).

Кватернионы являются численно устойчивыми представлениями поворота, которые можно преобразовать в углы Эйлера при необходимости.

Таким образом, каждая запись полностью описывает положение и ориентацию камеры в определённый момент времени. В рамках выпускной

квалификационной работы файл с траекторией движения БПЛА служит выходным результатом программного модуля.

Благодаря описанию траектории движения возможно отслеживать дрейф беспилотного летательного аппарата. Дрейф возникает под воздействием внешних факторов, таких как ветер, и приводит к отклонению от заданного маршрута. В процессе локализации и картографирования программный модуль фиксирует даже незначительные смещения БПЛА, что дает возможность отслеживать дрейф при наличии заданной траектории движения. Описание положения и ориентации камеры может использоваться для последующей корректировки траектории.

В данной главе были рассмотрены проектирование и реализация программного модуля для задачи отслеживания движения БПЛА на основе последовательных изображений. Основой для построения траектории движения послужил алгоритм локализации и картографирования ORB-SLAM3, функционирующий в монокулярном режиме.

Была описана архитектура программного модуля, включающая обработку изображений с помощью программы на языке Python, публикацию кадров в топик ROS2, приём изображений алгоритмом ORB-SLAM3, визуализацию ключевых точек и построение трехмерной сцены в реальном времени. Также внимание было уделено использованию фреймворка ROS2 версии Foxy, как инструмента для взаимодействия компонентов модуля.

В результате проведённой работы был реализован программный модуль, способный обрабатывать видеопоток с бортовой камеры БПЛА, восстанавливать и визуализировать траекторию движения и сохранять результаты в виде текстовых данный в файле с возможностью дальнейшей обработки. Реализованная архитектура является модульной и может быть адаптирована под другие режимы работы ORB-SLAM3 или интеграцию с сенсорами.

ТЕСТИРОВАНИЕ ПРОГРАММНОГО МОДУЛЯ

Тестирование программного модуля проводилось на трёх наборах данных, каждый из которых демонстрирует различные типы движения БПЛА.

Для оценки корректности работы программного модуля было проведено функциональное тестирование на основе разнообразных сценариев полета, представленных тремя наборами данных.

Первый набор последовательных изображений представляет прямолинейное движение без изменения ориентации.

Второй набор содержит движение с поворотами, характеризуемыми изменением направления движения, но без вращения вокруг вертикальной оси.

Третий набор включает движение, сопровождаемое вращением вокруг собственной оси (вертикальной оси), что имитирует резкие повороты дрона при относительно постоянном положении в пространстве.

Для корректного сравнения траекторий, полученных с помощью алгоритма ORB-SLAM3, с реальными данными, GPS координаты были переведены из глобальной географической системы координат в локальную систему координат, измеряемую в метрах. Такое решение обусловлено тем, что алгоритм ORB-SLAM3 формирует относительную траекторию движения в собственной локальной системе координат, которая отсчитывается от начальной позиции в момент начала работы. Перевод в локальную систему также упрощает численное сравнение траекторий движения.

Перевод координат из глобальной системы в локальную осуществлялся с помощью программы, написанной на языке Python. Исходный код программы представлен в листинге 3.

Листинг 3 – Исходный код для перевода координат из глобальной системы в локальную

```
import re
from datetime import datetime
from pyproj import CRS, Transformer
#Функция преобразования даты во временную метку
```

Продолжение листинга 3

```
#Функция преобразования временных меток
def convert_timestamp(srt_time):
    if not isinstance(srt_time, str):
        print(f"Ошибка: Ожидалась строка, а получено {type(srt_time)} ->
{srt_time}")
        return None
    dt = datetime.strptime(srt_time, "%Y-%m-%d %H:%M:%S.%f")
    return int(dt.timestamp() * 1e9)
#Функция преобразования глобальных координат в локальные
def gps_to_local(lat, lon, alt, ref_lat, ref_lon, ref_alt, transformer):
    local_x, local_y = transformer.transform(lon, lat) #Преобразование широты
и долготы
    local_z = alt - ref_alt #Высота относительно первой точки
    return local_x, local_y, local_z

#Функция чтения файла с глобальными координатами и логирование получения
данных
def read_real_gps(file_path):
    real_gps = []
    timestamp = None
    with open(file_path, 'r', encoding='utf-8') as file:
        for line in file:
            line = line.strip()
            if re.match(r'^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}\.\d{3}$',
line):
                timestamp = line
                continue
            match = re.search(
                r'\[latitude:\s*([-\d.]+)\]\s*'
                r'\[longitude:\s*([-\d.]+)\]\s*'
                r'\[rel_alt:\s*([-\d.]+)\s*abs_alt:\s*([-\d.]+)\]', line)
            if match:
                if timestamp is None:
                    print(f"Не найдено время для строки: {line}")
                    continue
                try:
                    lat = float(match.group(1))
                    lon = float(match.group(2))
                    rel_alt = float(match.group(3))
                    abs_alt = float(match.group(4))

                    real_gps.append((lat, lon, rel_alt, abs_alt, timestamp))
                except ValueError as e:
                    print(f"Ошибка при разборе данных: {line} | Ошибка: {e}")
            else:

                print(f"Пропущена строка: {line}")

    return real_gps

#Выбор UTM-зоны для Челябинска (UTM зона 41U)
wgs84 = CRS("EPSG:4326") #Глобальная система координат
utm = CRS("EPSG:32641") #Локальная система координат
transformer = Transformer.from_crs(wgs84, utm, always_xy=True)

real_gps_data = read_real_gps(srt_file)

if not real_gps_data:
```

Окончание листинга 3

```
    print("Не было найдено данных для записи.")
    exit()


trajectory = []
first_lat, first_lon, first_alt = None, None, None

for lat, lon, rel_alt, abs_alt, timestamp_str in real_gps_data:
    if first_lat is None:
        first_lat, first_lon, first_alt = lat, lon, abs_alt
    timestamp_ns = convert_timestamp(timestamp_str)

    if timestamp_ns is None:
        print(f"Ошибка конвертации времени для {timestamp_str}")
        continue
    local_x, local_y, local_z = gps_to_local(lat, lon, abs_alt, first_lat,
first_lon, first_alt, transformer) #Преобразование координат в локальные
    trajectory.append(f"{timestamp_ns} {local_x:.6f} {local_y:.6f}
{local_z:.6f} 0.0 0.0 0.0 1.0\n") #Формирование строки траектории
with open(output_file, "w", encoding="utf-8") as f:
    f.writelines(trajectory) #Запись траектории в выходной файл

print(f"saved {output_file}")
```

Программа обрабатывает данные GPS, полученные из SRT-файла (вспомогательного текстового файла, который содержит координаты и временные метки).

На первом этапе выполняется импорт необходимых библиотек. Используется библиотека «re» для обработки строк регулярными выражениями, библиотека «datetime» для работы со временем, библиотека «pyproj», которая предназначена для трансформации координат между различными географическими системами. В частности, из глобальной системы координат WGS84, используемой в GPS, данные преобразуются в локальную проекцию UTM.

Основной частью программы является функция «gps_to_local», в которой происходит трансформация географических координат в локальные. Здесь используется объект «transformer», созданный из систем координат EPSG:4326 (широта/долгота) и EPSG:32641 (UTM зона 41N, подходящая для Челябинской области). Преобразование осуществляется с помощью метода «.transform». Выходными данными являются координаты X и Y в метрах. Координата Z (высота) вычисляется как разность между текущей абсолютной

высотой и высотой первой точки, принимаемой за начало отсчета. Вся траектория представляется относительно первой GPS-точки.

Функция «read_real_gps» открывает файл с GPS-данными и построчно извлекает временные метки и значения координат, применяя регулярные выражения.

После чтения координат из файла начинается основной цикл по преобразованию данных. Первая координата используется в качестве опорной точки, относительно которой все остальные координаты переводятся в локальные. Также происходит преобразование временной метки в наносекунды. К каждой точке добавляются кватернионы поворота, чтобы формат строки соответствовал формату, который используется алгоритмом локализации и картографирования.

После завершения работы программы формируется файл, в котором каждая строка содержит временную метку, три координаты в метрах и параметры, описывающие ориентацию БПЛА в виде кватерниона.

На основе полученных данных были построены графики, визуализирующие реальные GPS-траектории и траектории, вычисленные ORB-SLAM3.

На рисунке 14 представлен график сравнения истинной и вычисленной траекторий при прямолинейном движении без поворотов.

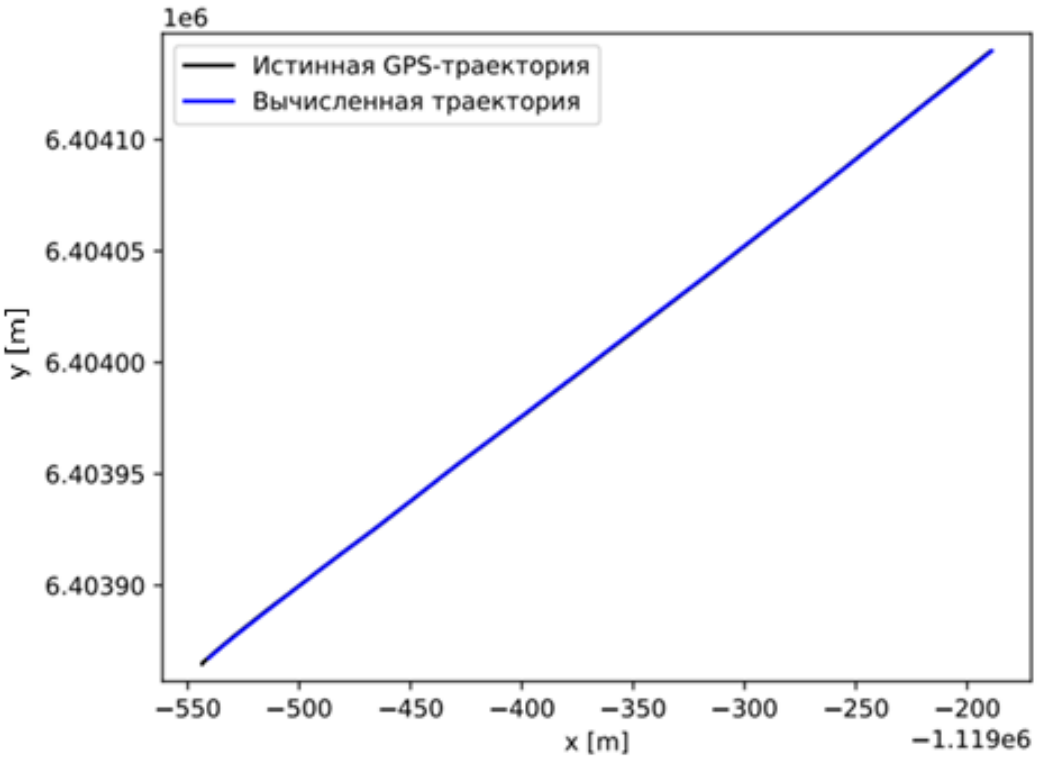

Рисунок 14 – График сравнения прямолинейной траектории

На рисунке 15 представлен график сравнения истинной и вычисленной траекторией при изменении направления движения, но без вращения вокруг вертикальной оси.

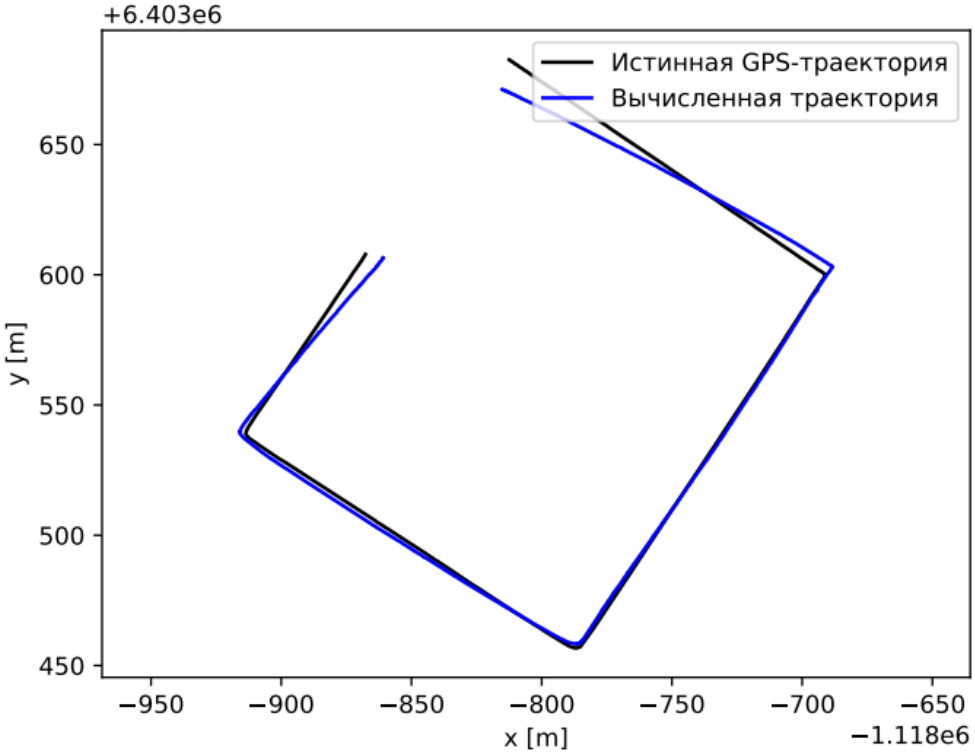

Рисунок 15 – График сравнения траектории с изменением направления движения

На рисунке 16 представлен график сравнения истинной и вычисленной траекторией при движении, сопровождаемым активным вращением вокруг вертикальной оси.

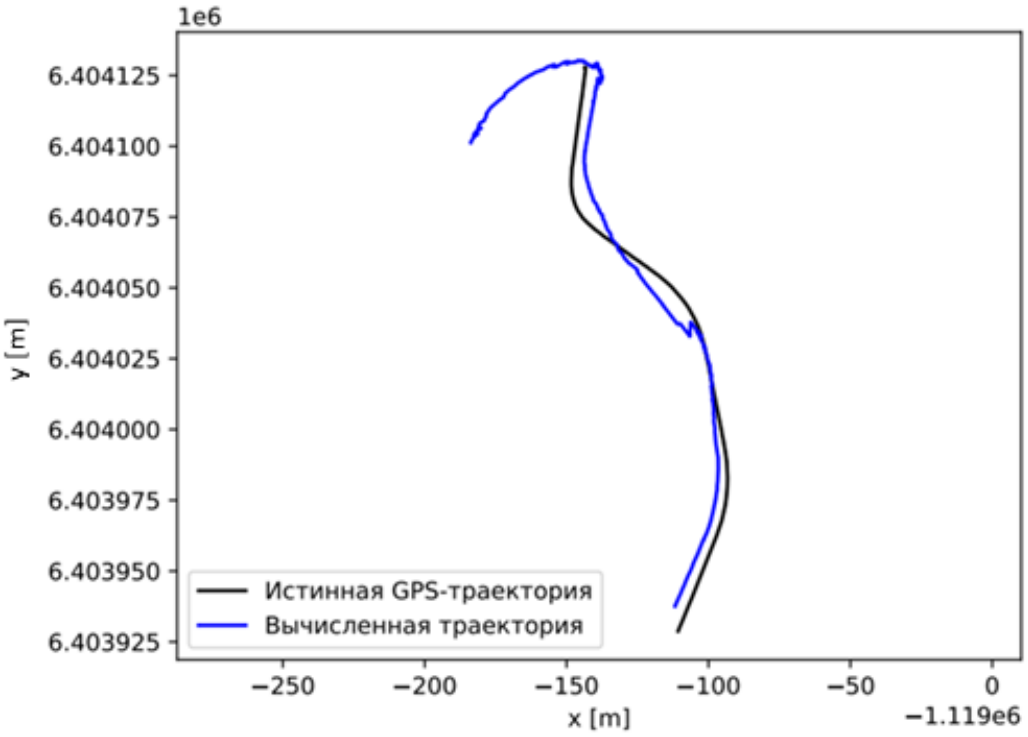

Рисунок 16 – График сравнения траектории с вращением вокруг вертикальной оси

Тестирование программного модуля показало, что наибольшее расхождение между траекториями наблюдается в случае движения с поворотами вокруг вертикальной оси. Это обусловлено тем, что при резких вращениях камера фиксирует меньшее количество смещения между кадрами, поэтому алгоритм испытывает трудности с точной оценкой относительной позиции в пространстве. Для повышения точности при подобных условиях возможно использование данных с инерциальных датчиков, что позволит компенсировать получение меньшего количества информации путём учета угловых скоростей и ускорений.

Следует отметить, что в случае использования монокулярной камеры SLAM-алгоритм не способен определить абсолютный масштаб траектории. Это связано с тем, что при отсутствии стереозрения или инерциальных данных алгоритм воспринимает только самостоятельно вычисленное относительное положение ключевых точек, то есть ORB-SLAM3 не имеет информации о реальном масштабе объектов и масштабе перемещения. В рамках работы

масштаб вычисленной траектории был подобран программным путем с использованием GPS-данных, что позволило сделать обе траектории сопоставимыми.

Эта проблема может быть устранена путем использования дополнительных сенсоров – стереопары или комбинации монокулярного зрения с инерциальным измерительным модулем. Благодаря использованию фреймворка ROS2 и возможностям работы алгоритма ORB-SLAM3 изменение типа камеры или интеграция инерциальных датчиков не представляет значительных трудностей. Необходимые изменения сводятся к корректировке конфигурационных файлов и перенастройке источника изображения.

В ходе тестирования была выполнена оценка точности вычисленной траектории движения БПЛА. Сравнение осуществлялось на основе трёх различных сценариев движения: прямолинейного без поворотов, с поворотами по горизонтали и с активными вращениями вокруг вертикальной оси. Для обеспечения корректности сравнения глобальные GPS-координаты были переведены в локальную систему координат.

Результаты тестирования показали высокую степень соответствия вычисленных траекторий реальным GPS-данным, особенно в сценариях без резких поворотов. В случае активного вращения вокруг вертикальной оси наблюдались наибольшие отклонения от действительной траектории. Эти расхождения объясняются использованием монокулярной камеры, так как алгоритм не способен различать вращение камеры без дополнительных источников информации.

Для повышения точности и устранения выявленных отклонений возможно использование инерциальных датчиков, а также замена монокулярной камеры на стереопару. Благодаря модульной реализации с помощью фреймворка ROS2, переключение между типами камер и добавление дополнительных датчиков не требует изменений архитектуры программного модуля.

ЗАКЛЮЧЕНИЕ

В рамках работы были рассмотрены алгоритмы анализа изображений, такие как оптический поток, извлечение ключевых признаков и визуальная одометрия, которые позволяют оценивать перемещения БПЛА на основе визуальных данных. Сравнение систем ArduPilot, DJI Terra и UgCS показало, что каждая из них имеет свои преимущества и ограничения, но и одна из них не обладает всеми характеристиками, необходимыми для достижения поставленной цели.

На основе анализа SLAM алгоритмов можно сделать вывод, что ORB-SLAM3 является наиболее подходящим для решения задачи автономной навигации, так как обладает меньшими вычислительными затратами за счет способа детектирования ключевых признаков и предлагает более точную локализацию и построение карты благодаря механизмам закрытия цикла и оптимизации позы.

В результате проведённой работы был реализован программный модуль, способный обрабатывать видеопоток с бортовой камеры БПЛА, восстанавливать и визуализировать траекторию движения и сохранять результаты в виде текстовых данный в файле с возможностью дальнейшей обработки. Реализованная архитектура является модульной и может быть адаптирована под другие режимы работы ORB-SLAM3 или интеграцию с сенсорами.

В ходе тестирования была выполнена оценка точности вычисленной траектории движения БПЛА. Сравнение осуществлялось на основе трёх различных сценариев движения: прямолинейного без поворотов, с поворотами по горизонтали и с активными вращениями вокруг вертикальной оси.

Результаты тестирования показали высокую степень соответствия вычисленных траекторий реальным GPS-данным, особенно в сценариях без резких поворотов. В случае наличия активных вращений вокруг вертикальной оси наблюдались наибольшие отклонения от действительной траектории. Эти

расхождения объясняются использованием монокулярной камеры, так как алгоритм не способен безошибочно различать вращение камеры без дополнительных источников информации.

Для повышения точности и устранения выявленных отклонений возможно использование инерциальных датчиков, а также замена монокулярной камеры на стереопару. Благодаря модульной реализации с помощью фреймворка ROS2, переключение между типами камер и добавление дополнительных датчиков не требует изменений архитектуры программного модуля.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Мокрова, М.И. Исследование влияния сложных условий пожарной обстановки на качество наблюдения и безопасность полёта БЛА. // Известия ЮФУ. Технические науки. – 2021. – № 1. – С. 112-124.
2. Зоев, И.В. Интеллектуальная система компьютерного зрения беспилотных летательных аппаратов для мониторинга технологических объектов предприятий нефтегазовой отрасли / И.В. Зоев, Н.Г. Марков, С.Е. Рыжова // Известия Томского политехнического университета. Инжиниринг георесурсов. – 2019. – Т. 330, № 11. – С. 34-49.
3. Шмидт, Дж.Т. Эксплуатация навигационных систем на основе GPS в сложных условиях окружающей среды // Гироскопия и навигация. – 2019. – Т. 27, № 1 (104). – С. 3-21.
4. Манюкова, Н.В. Компьютерное зрение как средство извлечения информации из видеоряда // Математические структуры и моделирование. – 2015. – № 4 (36). – С. 123-128.
5. Optical Flow. [Электронный ресурс] // – URL: https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html (дата обращения: 11.12.2024).
6. Serra, P. Nonlinear IBVS Controller for the Flare Maneuver of Fixed-Wing Aircraft using Optical Flow / P. Serra, L. Bras, T. Hamel // 49th IEEE Conf. on Decision and Control (CDC). – 2010. – P. 1656-1661.
7. Девайкин, П.А. Вычисление оптического потока для выделения транспортных средств на видеоизображении / П.А. Девайкин, А.В. Шикуть // Инженерный журнал: наука и инновации. – 2013. – № 2 (14). – 9 с.
8. Сакович, И.О. Применение метода Лукаса–Канаде для вычисления оптического потока / И.О. Сакович, Ю.С. Белов // Инженерный журнал: наука и инновации. – 2014. – Вып. 7. – 9 с.
9. Лоу, Ц. Отслеживание траектории движения объектов с использованием алгоритмов Лукаса–Канаде и Фарнебека / Ц. Лоу, С. Вэнь, Ц. Ли //

Международный журнал информационных технологий и энергоэффективности. – 2023. – С. 36-43.

10. Optical Flow: Horn-Schunck. [Электронный ресурс] // URL: https://www.cs.cmu.edu/~16385/s17/Slides/14.3_OF__HornSchunck.pdf (дата обращения 11.12.2024).
11. Саблина, В.А. Обзор методов нахождения ключевых точек для сопоставления изображений / В.А. Саблина, А.А. Терехина // Методы и средства обработки и хранения информации: межвуз. сб. науч. тр. / под ред. Б. В. Кострова. – Рязань: Ред.-изд. центр РГРТУ, 2018. – С. 113-117.
12. Пименова, М.Б. Разработка системы навигации учебного мобильного робота // Политехнический молодежный журнал. – 2019. – № 10(39). – 13 с.
13. Райченко, Б.В. Практическое применение методов ключевых точек на примере сопоставления снимков со спутника «Канопус-В» / Б.В. Райченко, В.В. Некрасов // Геоматика. – 2013. – № 2 (19). – С. 58-63.
14. Супрун, Д.Е. Алгоритм сопоставления изображений по ключевым точкам при масштабируемости и вращении объектов // Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. – 2016. – № 5. – С. 86-98.
15. Ali, B.A. A Review of Navigation Algorithms for Unmanned Aerial Vehicles Based on Computer Vision Systems / B.A. Ali, V.V. Tsodokova // Giroskopiya i Navigatsiya. – 2022. – V. 30. – № 4 (119). – P. 87-105.
16. Rosten, E. Machine learning for highspeed corner detection / E. Rosten, T. Drummond // European Conference on Computer Vision. – 2006. – V. 4. – P. 430-443.
17. Алпатов, Б.А. Исследование эффективности применения алгоритмов анализа изображений в задаче навигации беспилотных летательных аппаратов / Б.А. Алпатов, В.С. Муравьев, В.В. Стротов, А.Б. Фельдман // Цифровая обработка сигналов. – 2012. – № 3. – С. 29-34.
18. Андреев, А.М. Прямой стереоподход визуальной одометрии, основанный на линиях для решения задачи SLAM / А.М. Андреев, А.Н. Галичин // Научно-образовательный журнал для студентов и преподавателей «StudNet». – 2021. – № 5. – 6 с.

19. Обзор методов визуальной одометрии в ROS: использование камер глубины. [Электронный ресурс] // – URL: https://habr.com/ru/articles/404757/ (дата обращения: 12.12.2024).
20. Данилова, С.Д. Разработка модели визуальной одометрии на основе сенсоров и анализа видеопотока // Comp. nanotechnol. – 2024.– С. 36-47.
21. Гошин, Е.В. Метод согласованной идентификации в задаче определения соответственных точек па изображениях / Е.В. Гошин, В.А. Фурсов // Компьютерная оптика. – 2012. – Т. 36. – №1. – С. 131-135.
22. Шепилова, К.М. МЕТОД ТРЕХМЕРНОЙ РЕКОНСТРУКЦИИ СЦЕНЫ В ОТНОСИТЕЛЬНЫХ КООРДИНАТАХ ПО ДВУМ ИЗОБРАЖЕНИЯМ С НЕОТКАЛИБРОВАННЫХ ВИДЕОКАМЕР / К.М. Шепилова, А.В. Сотников, А.В. Шипатов, Ю.В. Савченко // Известия вузов. Электроника. – 2020. – № 3. – С. 265-276.
23. Терехов, М.А. Обзор современных методов визуальной одометрии // Компьютерные инструменты в образовании. – 2019. – № 3. – С. 4-11.
24. Евдокимова, Т.С. Система позиционирования и идентификации мобильной робототехнической платформы в ограниченном и открытом пространстве / Т.С. Евдокимова, А.А. Синодкин, Л.О. Федосова, М.И. Тюриков // Труды НГТУ им. Р. Е. Алексеева. – 2018. – № 2(121). – С. 16-25.
25. Michael, M. FastSLAM: a factored solution to the simultaneous localization and mapping problem / M. Michael, T. Sebastian, K. Daphne // Eighteenth national conference on Artificial intelligence. American Association for Artificial Intelligence. – 2002. – P. 593-598.
26. Introduction and application of ORB-SLAM3. [Электронный ресурс] // – URL: https://matom.ai/insights/slam/ (дата обращения: 19.03.2025).
27. Yang, G. Real-time Visual-Inertial Odometry based on Point-Line Feature Fusion / G. Yang, W. Meng, G. Hou, N. Feng // Xi’an University of Posts and Telecommunications. – 2023. – V. 31. – no. 4(123). – P. 96-117.