

Санкт-Петербургский государственный университет

Карасева Елизавета Олеговна

Выпускная квалификационная работа

Уточнение GNSS позиционирования при
помощи визуально-инерциальной
одометрии

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование
информационных систем»*

Основная образовательная программа *СВ.5162.2021 «Технологии программирования»*

Научный руководитель:
старший преподаватель кафедры системного программирования, к. т. н. Литвинов Ю. В.

Рецензент:
директор ООО «Системы Компьютерного Зрения» Смирнов М. Н.

Санкт-Петербург
2025

Saint Petersburg State University

Elizaveta Karaseva

Bachelor's Thesis

Refinement of GNSS positioning using visual-inertial odometry

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5162.2021 "Programming Technologies"*

Scientific supervisor:
C.Sc, Y.V. Litvinov

Reviewer:
Director "Computer Vision Systems" M.N. Smirnov

Saint Petersburg
2025

Оглавление

Введение	4
1. Постановка задачи	6
2. Обзор	7
2.1. Задача SLAM и Visual SLAM	7
2.2. Классификация Visual SLAM по типу обработки изображений	8
2.3. Классификация Visual SLAM по способу оценки состояния	10
2.4. Visual-Inertial SLAM	11
2.5. Интеграция данных GNSS в VI-SLAM	13
2.6. Kimera	18
2.7. Диаграмма последовательностей системы	21
3. Используемые технологии	24
4. Описание решения	25
4.1. Формат и преобразование данных	25
4.2. Обработка параметров и чтение датасета	27
4.3. DataProvider module	28
4.4. Frontend module	29
4.5. Выбор типа связи и реализация backend module	31
4.6. Доработки	32
5. Эксперимент	34
5.1. Notebooks	34
5.2. Сравнение результатов	34
Заключение	37
Список литературы	38

Введение

Прогресс в области робототехники набирает темп, особенно это касается систем автономной навигации для автомобилей и дронов. Во время навигации такие системы определяют свое местоположение относительно других объектов, опираясь на данные с различных датчиков, установленных на устройстве: инерциальных модулей, камер, гироскопов, лидаров. Данный процесс называется **локализацией** [37].

В части случаев параллельно с локализацией происходит **картографирование** — распознанные объекты фиксируются на карте, которая постепенно корректируется и используется для дальнейшей навигации. Такие системы относятся к классу **SLAM** (Simultaneous Localization and Mapping). Алгоритмы SLAM являются важным элементом автономных устройств, так как связывают локализацию и картографирование: результаты одного процесса участвуют в вычислениях другого.

Визуально-инерциальная одометрия (**VIO**), основанная на обработке данных с камер и инерциальных измерительных модулей (**IMU**), является популярным подходом к решению задачи SLAM [29]. Одним из недостатков VIO является накопление ошибки, вызванное использованием относительных измерений. Оно приводит к искажениям траектории и постепенному расхождению оценок местоположения. Как следствие, глобальная карта не соответствует реальности, а робот сбивается с намеченного пути.

Данные глобальных навигационных спутниковых систем (**GNSS**) предоставляют абсолютные значения координат, поэтому добавление их в алгоритмы SLAM позволяет уменьшить накопление ошибки, а также уточнить общий масштаб карты. Существующие алгоритмы с открытым исходным кодом успешно реализуют VIO, но динамическая интеграция данных GNSS и VIO остается сложной задачей из-за разной частоты обновления данных и различий в точности измерений.

В рамках данной работы планируется интегрировать GNSS-данные в конвейер визуально-инерциальной одометрии. Для оценки точности алгоритма будет подготовлен расширенный датасет EuRoC, включаю-

щий GNSS-данные с разным уровнем искусственного шума, и проведена серия экспериментов с его использованием. Работа будет выполнена на базе библиотеки Kimera, а именно, её модуля Kimera-VIO [15]. Ожидается, что полученное решение повысит точность локализации и улучшит качество построенной карты, что сделает его полезным для задач автономной навигации.

1 Постановка задачи

Целью работы является разработка метода интеграции GNSS-данных в конвейер визуально-инерциальной одометрии. Для её выполнения были поставлены следующие задачи.

1. Провести обзор существующих решений, интегрирующих GNSS и VIO.
2. Провести обзор систем VIO-SLAM, потенциально подходящих для внедрения GNSS.
3. Изучить архитектуру и особенности работы библиотеки Kimera.
4. Обеспечить корректную загрузку, синхронизацию и передачу GNSS-данных в конвейер VIO.
5. Подготовить датасет EuRoC, добавив искусственный шум в GNSS-данные.
6. Провести серию экспериментов на модифицированном датасете, оценить влияние точности GNSS-данных на результат работы алгоритма.

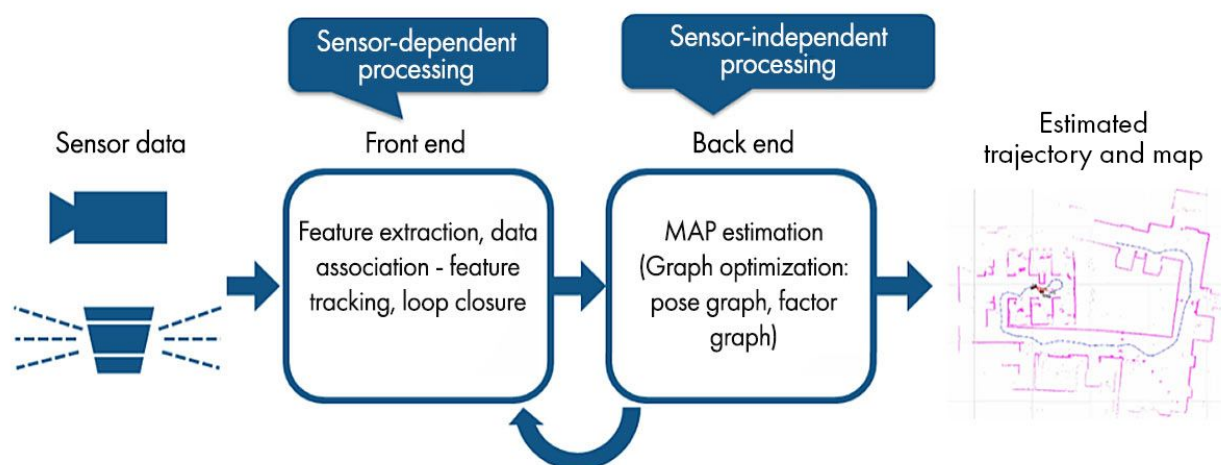


Рис. 1: Принцип работы алгоритмов SLAM, взято из <https://ch.mathworks.com/discovery/slam.html>

2 Обзор

В данном разделе рассматриваются принципы работы Visual SLAM и использование в нем VIO, приводятся примеры алгоритмов с поддержкой GNSS-данных, а также обосновывается выбор современного алгоритма VI-SLAM для интеграции GNSS. В заключение описывается архитектура библиотеки Kimera.

2.1 Задача SLAM и Visual SLAM

SLAM-алгоритм представляет собой конкретный метод, восстанавливающий траекторию и карту по входным данным. Его основными компонентами являются фронтенд, который обрабатывает данные и извлекает из них ориентиры, и бэкенд, который отвечает за построение и обновление карты, а также за определение текущего местоположения робота относительно распознанных ориентиров (рис. 1). Помимо самого алгоритма, полноценная SLAM-система включает в себя вспомогательные блоки, например, модуль визуализации траектории.

SLAM-системы делятся на группы в зависимости от используемых сенсоров. Подробный обзор различных видов SLAM представлен в статье [2]. В данной работе сделан акцент на Visual SLAM, который строит карту на основе данных с изображений.

Для оценки масштаба карты необходима информация о глубине, то есть о расстоянии между камерой и наблюдаемыми объектами [25]. Существует несколько типов визуальных сенсоров:

- **монокулярные камеры** дешевы и просты в использовании, но не способны напрямую измерять глубину;
- **стереокамеры** предоставляют два синхронизированных изображения и вычисляют глубину сцены на основе параллакса [26];
- **RGB-D камеры** одновременно формируют цветное изображение и измеряют глубину с помощью инфракрасных датчиков.

В рамках данной работы предполагается использование алгоритма Visual SLAM с использованием стереокамер. Рассмотрим основные компоненты таких алгоритмов подробнее.

2.2 Классификация Visual SLAM по типу обработки изображений

Рассматривая фронтенд-часть Stereo-SLAM, можно разделить все подобные алгоритмы по способу обработки изображений (рис. 2¹) на косвенные (feature-based) и прямые (direct) методы.

1. В **косвенных** методах на стереопаре — левом и правом изображениях — распознаются уникальные объекты (features), которые затем сопоставляются с элементами следующего снимка. Положение робота относительно этих объектов вычисляется исходя из их «движения» на последовательных кадрах.

Один из самых известных алгоритмов, работающих по такому принципу — ORB-SLAM2 [23] и его модификации [20]. Его преимущество в том, что распознавание ориентиров не зависит от точки обзора, однако недостаточная выраженность текстур окружающей среды негативно влияет на точность сопоставления.

¹ Авторы статьи [2] также выделяют мультисенсорные методы, которые интегрируют данные с камер с другими устройствами.

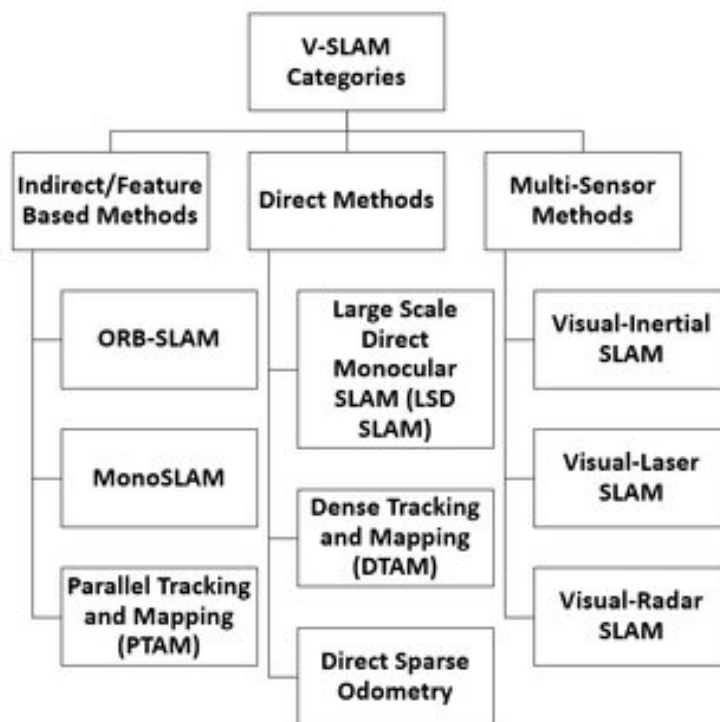


Рис. 2: Категории Visual-SLAM, взято из [2]

2. В основе **прямых** методов лежит следующий принцип: интенсивность пикселей должна быть одинаковой при наблюдении одной и той же точки на соседних кадрах [2]. Эта идея позволяет сократить время на извлечение и сопоставление ориентиров, при этом используя больше информации из исходного изображения. А наличие двух синхронизированных изображений позволяет повысить точность оценки глубины сразу при инициализации карты.

Примеры: Stereo LSD-SLAM и системы типа Direct Sparse Odometry [7]. Работа с интенсивностями пикселей делает их устойчивыми к слаботекстурированным областям, но зависимыми от изменений освещения.

Выбор подхода зависит от условий среды и требований к системе. Когда в приоритете скорость обработки и устойчивость к изменениям среды, предпочтительнее использовать косвенные методы, но если местность содержит большое количество однородных поверхностей, прямые методы обеспечивают более точную локализацию [2].

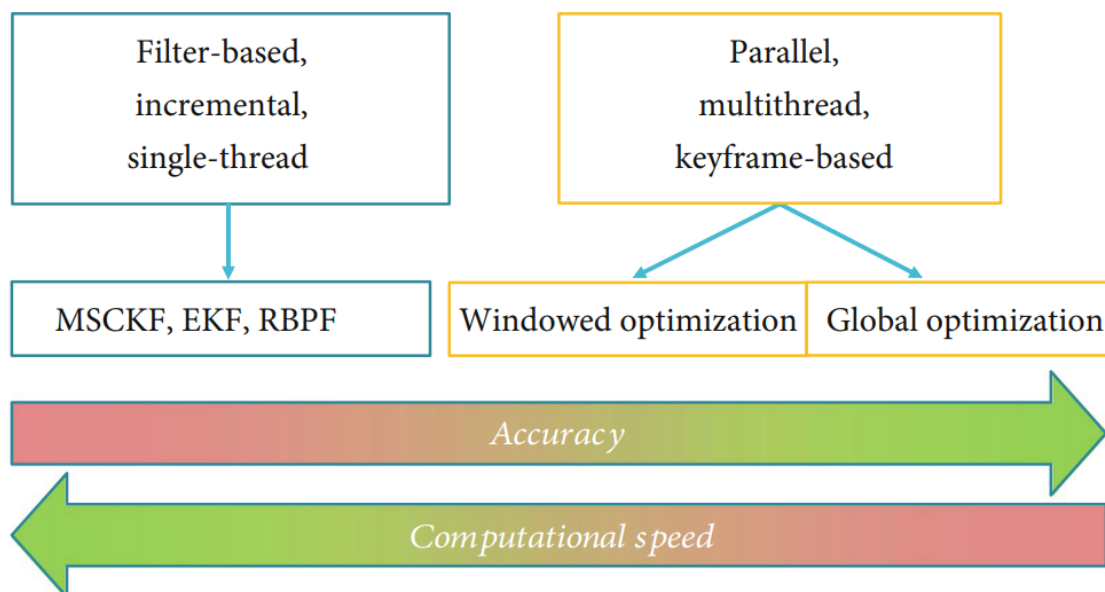


Рис. 3: Варианты проектирования алгоритмов V-SLAM, взято из [27]

2.3 Классификация Visual SLAM по способу оценки состояния

Алгоритмы V-SLAM можно разделить по способу оценки траектории и построения карты на два основных подхода: на основе фильтров и на основе ключевых кадров.

2.3.1 Вероятностный подход

В подходе **на основе фильтров** состояние устройства (его поза² и скорость) оценивается с помощью вероятностных моделей. Оценка происходит итеративно и состоит из двух фаз: предсказание новой позы на основе предыдущих измерений с учетом прогнозируемых ошибок и обновление оценки состояния с учетом новых измерений. Примеры этого подхода — фильтр частиц (Particle Filter), фильтр Калмана (KF) и его вариации: Extended Kalman Filter, Unscented Kalman Filter. Вероятностные методы предназначены для получения оценки в режиме реального времени, однако они плохо масштабируются, так как сложность возрастает квадратично по числу выделенных ориентиров, и со

²положение и ориентация

временем теряют точность из-за локального характера оценки.

2.3.2 Ключевые кадры и оптимизация

Алгоритмы этого класса создают граф на основе ориентиров с ключевых кадров: узлы представляют состояния системы, а рёбра — связи между ними (измерения). Оценка состояния выполняется путём минимизации общей ошибки, для этого используются оптимизационные алгоритмы (например, g2o или Ceres).

Тип графа зависит от уровня абстракции: например, граф поз фокусируется на позах между ключевыми кадрами (рёбра представляют относительные перемещения), а в фактор-графе узлы помимо поз учитывают параметры камеры, смещение IMU и другие переменные.

Графовые методы по сравнению с вероятностными лучше масштабируются и обеспечивают лучшую точность, особенно при использовании замыкания циклов, но и потребляют больше вычислительных ресурсов.

Современные алгоритмы часто комбинируют оба подхода: фильтры используются для быстрой оценки движения между кадрами, а оптимизация — для периодической коррекции ошибок и устранения неопределённостей. Это позволяет добиться баланса между скоростью и точностью (рис. 3).

2.4 Visual-Inertial SLAM

Визуально-инерциальный SLAM (VI-SLAM) — пример мультисенсорного подхода, в котором для оценки движения и построения карты используются данные с камер и IMU. Камеры дают информацию об объектах в окружающей среде, а измерения с IMU предоставляют данные о динамике движения устройства. Совместное использование этих сенсоров помогает компенсировать перепады освещённости и уменьшить накопление ошибки в оценке траектории.

Большинство алгоритмов VI-SLAM построены на работе с ключевыми кадрами. Основная сложность таких алгоритмов заключается в

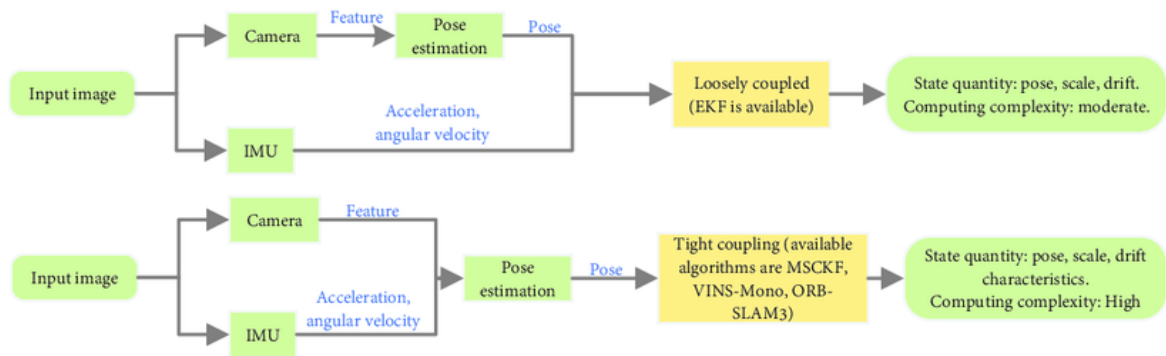


Рис. 4: Слабая и сильная связь в VI-SLAM, взято из [22]

объединении разнородных данных. Выделяют два уровня их взаимодействия [22] (рис. 4).

- Слабая связь (loose coupling): данные IMU и изображения обрабатываются независимо, а затем происходит объединение оценок от каждого сенсора. Такой подход проще в реализации, но в случае ухудшения качества данных одного из сенсоров точность оценок сильно снижается [19].
- Сильная связь (tight coupling): в уравнениях оценки и оптимизации используются все виды исходных данных. Это позволяет точнее оценивать траекторию и повышает надежность алгоритма: в случае сбоя одного сенсора данные компенсируются другими [6].

2.4.1 Сравнение VIO и SLAM

Визуально-инерциальная одометрия (VIO) и SLAM-алгоритмы решают схожую задачу — предоставлять оценку положения устройства и строить карту среды, — однако различаются по целям и областям применения.

VIO обеспечивает локальную оценку траектории в реальном времени, в вычислениях используя данные с камер и IMU. Она строит относительную траекторию, не сохраняя глобальную карту и не учитывая повторные посещения мест, что снижает вычислительную нагрузку, но

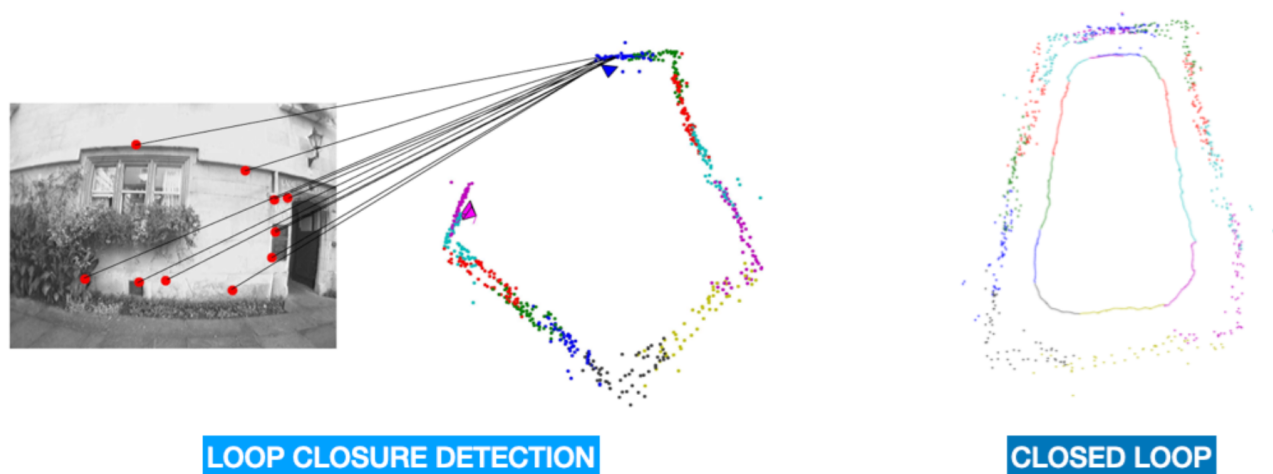


Рис. 5: Замыкание цикла и коррекция карты, взято из [30]

приводит к накоплению ошибок и отклонению построенной траектории от реальной.

SLAM, в отличие от VIO, строит глобальную карту и может использовать другие сенсоры (лидары, GPS). С помощью замыкания циклов и глобальной оптимизации он сопоставляет текущее наблюдение с сохранёнными и определяет ситуации, когда устройство возвращается в ранее посещённую точку. Это позволяет скорректировать карту, существенно снижая накопленную ошибку траектории (рис. 5) [12].

Многие современные SLAM-алгоритмы используют VIO на фроненде. Локальная оценка передается на бэкенд, который затем проводит манипуляции по оптимизации глобальной карты.

2.5 Интеграция данных GNSS в VI-SLAM

Глобальная навигационная спутниковая система (GNSS) — общий термин для спутниковых систем навигации (GPS, ГЛОНАСС, BeiDou и другие). Устройство с приёмником вычисляет свое местоположение на Земле на основе анализа сигналов, принятых им от нескольких спутников одновременно.

GNSS-приемники отличаются невысокой стоимостью, а привязка данных к глобальным координатам позволяет уточнить масштаб при построении карты и уменьшать накопление ошибки. Однако точность

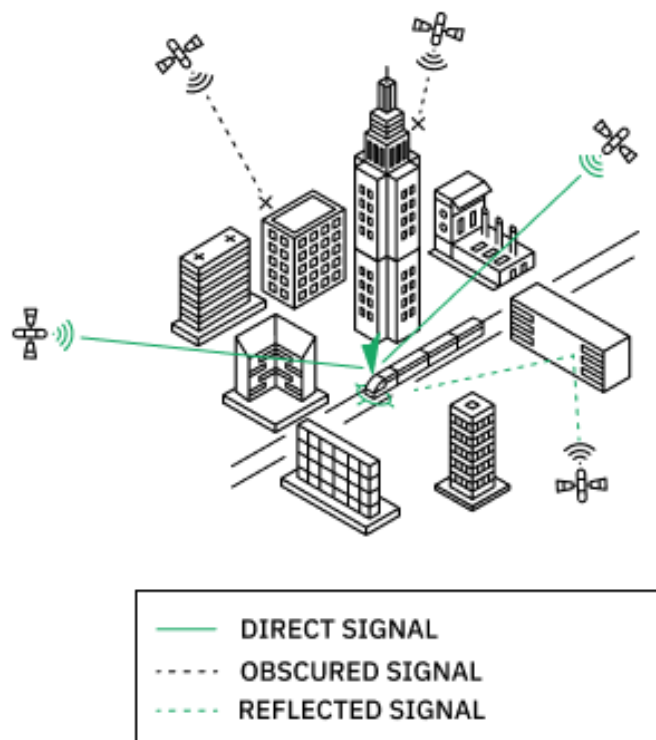


Рис. 6: Ошибки GNSS-сигналов, взято из [19]

получаемых данных зависит от условий приёма сигнала и резко ухудшается при наличии препятствий, например, в городской застройке или тоннелях (рис. 6). В таких случаях данные считаются шумными и требуют дополнительной фильтрации или обработки.

Без коррекции точность GNSS-приёмников составляет около пяти метров в горизонтальной плоскости. Для уточнения получаемой информации используются методы RTK (Real-Time Kinematic) и PPP (Precise Point Positioning) [5]. Для коррекции спутниковых сигналов RTK использует данные от наземных базовых станций, что позволяет достичь сантиметровой точности измерений в реальном времени. PPP обеспечивает точность около дециметра и требует время на расчеты, зато не привязан к станциям: ему достаточно данных коррекции со спутников.

Каждый производитель спутникового оборудования задает свой формат измерений (.jps, .tps, .cnb и другие), однако существует и общий текстовый формат RINEX. Хотя приёмники часто выполняют первичную обработку данных, для повышения точности может выполняться

их постобработка.

При интеграции в SLAM-алгоритм необходимо перевести GNSS-данные в локальную систему координат устройства, обеспечить синхронизацию GNSS с измерениями других сенсоров, а также предусмотреть своевременное «понижение доверия» к шумным данным, чтобы в этом случае точность работы алгоритма не снижалась [19].

2.5.1 Решения, интегрирующие GNSS, VIO и SLAM

Интеграция GNSS в VI-SLAM изучается, однако решений с открытым исходным кодом довольно мало. В большинстве случаев это модификации известных визуально-инерциальных алгоритмов, не обладающие достаточной документацией и не предоставляющие анализы точности построенных траекторий, что затрудняет их понимание и практическое применение. Кроме того, часть алгоритмов использует данные GPS. Это снижает надежность позиционирования, а для поддержки других спутниковых систем в алгоритмах требуются доработки.

Сравнительный анализ существующих решений приведен в таблице 1, при его проведении учитывались такие параметры, как поддерживаемая навигационная система, дата последнего обновления репозитория и используемая в проекте лицензия.

Алгоритм	Тип	Обновления	Лицензия
MINS [34]	GNSS	2024	GPLv3
GNSS-Stereo-Inertial Fusion [3]	GNSS	2024	GPLv3
RTK-Visual-Inertial-Navigation [36]	GNSS	2024	GPLv3
InGVIO [4]	GNSS	2023	GPLv3
IC-GVINS [31]	GNSS	2023	GPLv3
SVO-Pro-GPS [35]	GPS	2023	GPLv3
VINS-Fusion [11]	GPS	2021	GPLv3
GVINS [10]	GNSS	2021	GPLv3

Таблица 1: Visual-Inertial SLAM алгоритмы с поддержкой GNSS и открытым исходным кодом, данные актуальны на 14.02.2025

Все рассматриваемые алгоритмы распространяются под лицензией GNU GPL версии 3.0, которая требует открытого распространения исходного программного обеспечения. Это исключает их использование в коммерческих проектах, где исходный код закрыт. Также часть решений ориентирована на специфическое применение:

- GNSS-Stereo-Inertial Fusion разработан для применения в сельском хозяйстве, а не в городской местности;
- RTK-Visual-Inertial-Navigation акцентирует внимание на RTK, поэтому работает с ограниченным классом GNSS-данных, содержащим дополнительные измерения, не предоставляемые простыми приёмниками;
- IC-GVINS реализует инерциально-центричную архитектуру, то есть точность алгоритма зависит в первую очередь от инерциальной навигационной системы (INS), а не от визуальных данных.

Также стоит отметить, что VINS-Fusion и GVINS не обновлялись с 2021 года, поэтому используют в расчетах устаревшие версии библиотек.

Ни одно из рассмотренных решений не подходит для использования в коммерческих проектах, поэтому в рамках данной работы было принято решение провести внедрение GNSS в один из визуально-инерциальных SLAM-алгоритмов самостоятельно.

2.5.2 Обзор современных реализаций VI-SLAM

При выборе алгоритма для внедрения GNSS-данных было проведено сравнение известных реализаций VI-SLAM с открытым исходным кодом [33], [8], [6], [2] (рис. 7) Сравнение проводилось по следующим параметрам: тип поддерживаемых камер, метод обработки изображений (прямой или выделения особенностей), способ оценки состояния (вероятностный или на основе оптимизации), наличие механизма коррекции ошибок (замыкание цикла), тип лицензии (необходимая — разрешающая коммерческое использование), а также поддержка проекта — наличие обновлений в течение последних пяти лет.

Алгоритм	Тип камер	Фронтенд	Бэкенд	Loop closure	Лицензия	Обновления
ORB-SLAM3	Моно, Стерео, RGB-D	Особенности	Ключевые кадры	Да	GPLv3	2022
VINS-Fusion	Моно, Стерео	Особенности	Ключевые кадры	Да	GPLv3	2021
OKVIS	Стерео	Особенности	Ключевые кадры	Нет	BSD 3-Clause	2020
ROVIO	Моно	Прямой метод	Фильтр	Нет	BSD 3-Clause	2024
Kimera	Моно, Стерео, RGB-D	Особенности	Ключевые кадры	Да	BSD 2-Clause	2025

Таблица 2: Современные алгоритмы VI-SLAM, данные актуальны на 14.02.25

На основе проведенного сравнения систем VI-SLAM (таблица 2) можно сделать следующие выводы.

- ORB-SLAM3 и Kimera работают со всеми типами камер, в то время как OKVIS ориентирован в первую очередь на стереокамеры, ROVIO поддерживает только монокулярные, а VINS-Fusion — моно- и стереокамеры.
- ROVIO использует EKF вместо фактор-графовой оптимизации,

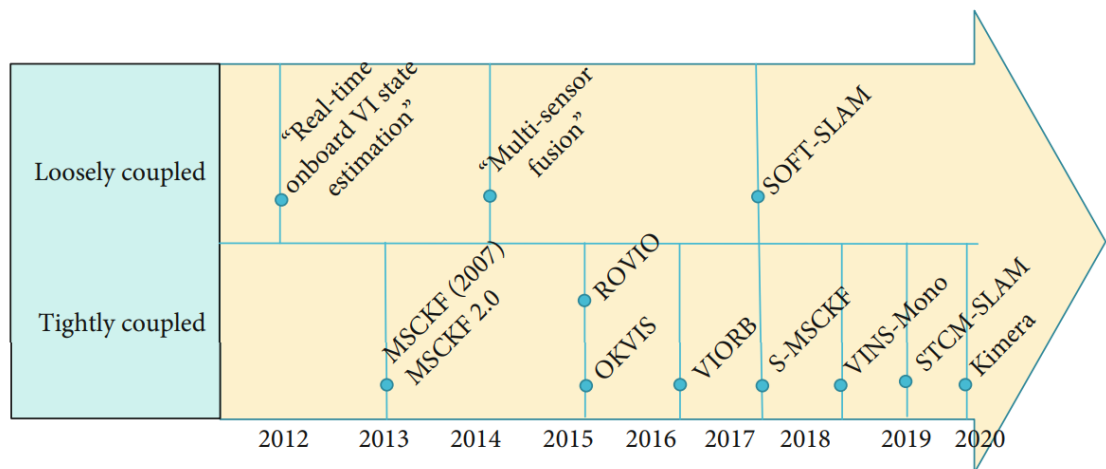


Рис. 7: Наиболее репрезентативные алгоритмы VI-SLAM, взято из [27]

поэтому сильно отличается от других рассматриваемых решений. Кроме того, для анализа изображений в нем применяется прямой метод, а не косвенный. В совокупности это делает алгоритм менее ресурсоемким и более быстрым, но потенциально менее точным и менее устойчивым к шумам, чем остальные.

- ORB-SLAM3, VINS-Fusion и Kimera реализуют механизм замыкания циклов, что позволяет им компенсировать ошибку при возвращении в ранее посещенные области. OKVIS и ROVIO такой функциональностью не обладают.
- Все системы, кроме ORB-SLAM3 и VINS-Fusion (которые разрабатывается под GPLv3), имеют лицензию BSD, что позволяет применять их для коммерческого использования.
- Наибольшую поддержку со стороны разработчиков на момент анализа демонстрируют Kimera, ORB-SLAM3 и ROVIO, тогда как OKVIS и VINS-Fusion не получали значительных обновлений в последние годы и могли потерять актуальность.

Учитывая эти факторы, в качестве системы для интеграции GNSS-данных была выбрана библиотека **Kimera**. Она обладает подходящей лицензией, активно развивается, поддерживает разные типы сенсоров, а также использует фактор-графовую оптимизацию и механизм замыкания циклов, что существенно упрощает добавление нового типа данных и увеличивает точность оценки траектории.

2.6 Kimera

Kimera — это библиотека на C++ с открытым исходным кодом. Она предназначена для метрико-семантического SLAM, то есть не только строит карту окружающей среды, но и добавляет к объектам на ней семантические метки: «стена», «стол». Kimera работает на CPU, что позволяет использовать её на компьютерах, не обладающих большими вычислительными ресурсами (которые часто требуются для работы SLAM-алгоритмов). Библиотека состоит из четырех модулей (рис. 8).

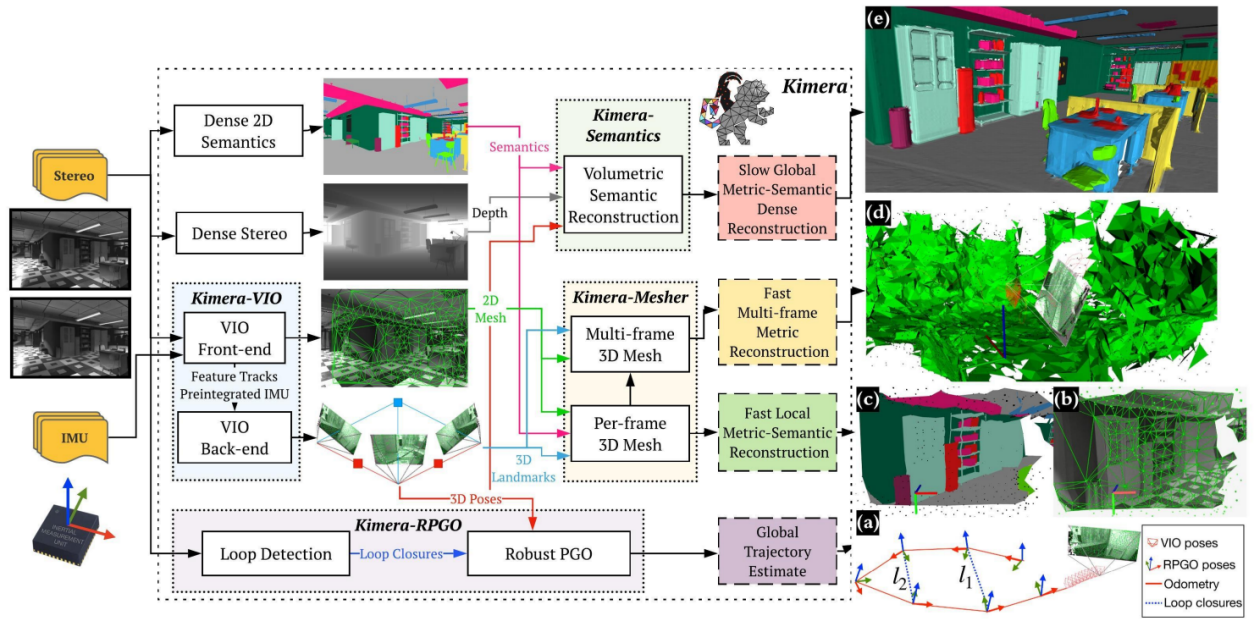


Рис. 8: Архитектура библиотеки Kimera

- **Kimera-VIO:** конвейер визуально-инерциальной одометрии, здесь формируется локальная траектория.
- **Kimera-RPGO:** модуль оптимизации графа поз, он отвечает за построение глобальной карты с учетом замыкания циклов.
- **Kimera-Mesher:** модуль для создания трехмерных геометрических сеток (мешей) по данным VIO. Строит как покадровые, так и многокадровые 3D-сетки.
- **Kimera-Semantics:** генератор глобальной 3D-карты с семантической аннотацией, то есть он объединяет на одной карте информацию от VIO и результаты распознавания отдельных объектов.

2.6.1 Kimera-VIO

Kimera-VIO представляет из себя модуль визуально-инерциальной одометрии, предназначенный для оценки позы устройства на основе данных от стереокамеры и IMU (рис. 13).

Для оценки позы на бэкенде системы используется факторграф, реализованный с помощью библиотеки GTSAM (тип

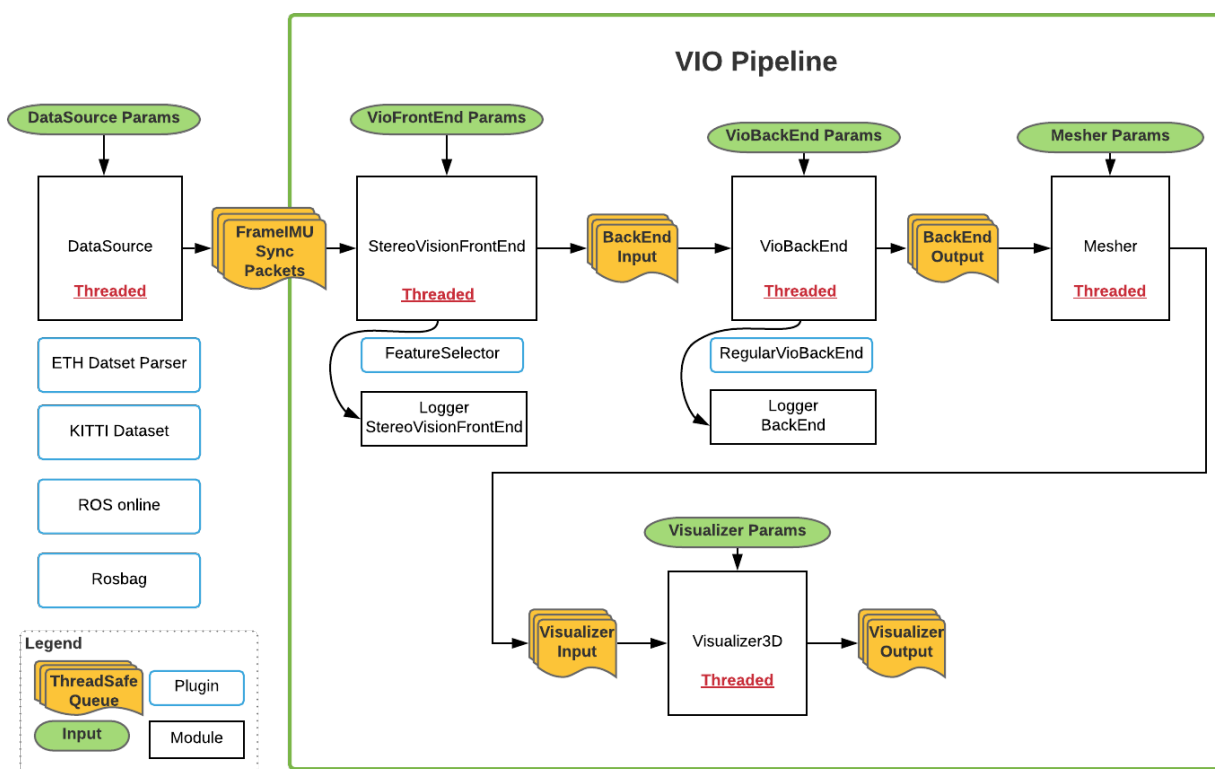


Рис. 9: Архитектура конвейера VIO в системе Kimera-VIO

NonlinearFactorGraph) и содержащий переменные (состояния, которые оценивает система) — поза, скорость, смещение IMU и распознанные ориентиры (landmarks) в конкретный момент времени t — и факторы, то есть наблюдения, накладывающие ограничения на переменные.

- При инициализации графа используются **PriorFactors**, накладывающие ограничения на начальную позу, скорость и смещение. Они позволяют задать «точку отсчета» для траектории.
- **NoMotionFactor** и **ConstantVelocityFactor** — пользовательские, накладывают ограничения на скорость между двумя состояниями в случае, когда данные IMU ненадежны.
- **ImuFactor** связывает соседние ключевые кадры, учитывая изменение скорости и смещения устройства между ними.
- **BetweenFactor** — универсальный фактор для задания относитель-

ных поз между кадрами.

- **SmartStereoProjectionPoseFactor** объединяет множество наблюдений одного и того же ориентира из разных кадров и позволяет более надёжно и компактно учитывать визуальные измерения при оптимизации.

После добавления факторов в граф с помощью метода `predict` библиотеки GTSAM [9] формируется начальное предположение о новом состоянии системы. Оно предсказывается на основе данных IMU и помогает ускорить процесс оптимизации. Граф оптимизируется алгоритмом ISAM2 (incremental smoother), который выполняет обновление наблюдаемых переменных и маргинализацию (удаление) из графа состояний, которые уже перестали отслеживаться. После оптимизации происходит обновление текущей позы устройства. Она передается по конвейеру в другие модули, где происходит распознавание объектов и визуализация траектории. Количество итераций оптимизации, как и многие другие настройки бэкенда, задаются в файле `BackendParams.yaml` перед запуском алгоритма.

Для учета данных GNSS при построении траектории было решено добавлять их в качестве фактора на конкретную позу устройства.

2.7 Диаграмма последовательностей системы

На рисунке 10 представлена диаграмма последовательностей, описывающая процесс работы алгоритма. Библиотека реализует конвейерную параллельность: обработка каждого кадра разбита на этапы, а данные передаются между модулями асинхронно, через потокобезопасные очереди и буферы.

Пользователь инициирует запуск скрипта, при этом в главном потоке запускается функция `main`. Она инициирует чтение датасета через модуль `DataProviderInterface` и запускает `PipelineModule`, который инициализирует остальные компоненты системы: регистрирует обратные вызовы (callbacks), запускает потоки обработки и ожидает завершения работы всего конвейера.

Метод `DataProviderInterface::spin()` читает датасет, передавая данные в буферы `DataProviderModule`. Данные с IMU отправляются сразу полностью, а стереоизображения — по мере обработки датасета. Остальные модули системы обрабатывают пакеты данных одновременно в своих потоках.

`DataProviderModule` следит за состоянием буферов, формирует синхронизированные пакеты данных и отправляет их во `FrontendModule`. Этот модуль выполняет распознавание ориентиров и предсказание позы на основе данных IMU, передавая результат в `BackendModule`, где данные преобразуются в факторы для построения фактор-графа, и выполняется оптимизация позы. `MeshesModule` строит 3D-сетку на основе данных с фронтенда и бэкенда, а `VisualizerModule` подготавливает данные для визуализации. Наконец, `DisplayModule` формирует итоговое изображение с точками и проекциями и выводит его на экран.

По окончании чтения датасета `DataProviderInterface` подает сигнал о завершении. Подача данных в очереди модулей прекращается, `PipelineModule` завершает работу каждого компонента и возвращает управление в `main()`, где результаты фиксируются, а программа завершается.

При работе с GNSS-данными было решено вносить изменения в модули `DataProviderInterface`, `DataProviderModule`, `PipelineModule`, `FrontendModule` и `BackendModule`, так как остальные модули взаимодействуют с построенной траекторией, а не с измерениями конкретных сенсоров.

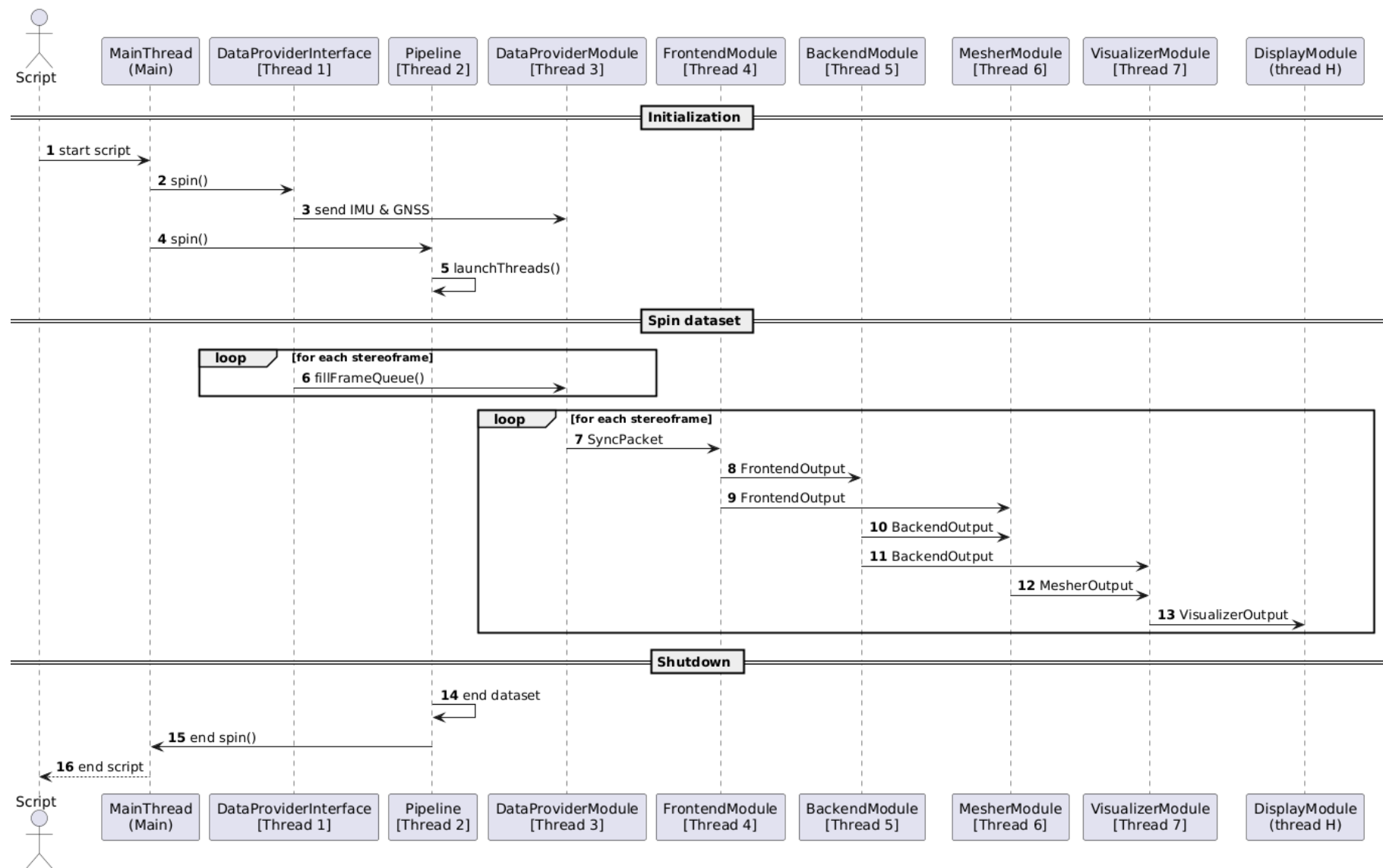


Рис. 10: Диаграмма последовательностей системы Kimera-VIO

3 Используемые технологии

Разработка велась на языке C++, так как на нем написана исходная библиотека.

1. Для сборки и запуска проекта использовался Docker Desktop [1]. Этот фреймворк позволяет легко создавать изолированные контейнеры, содержащие приложения с необходимыми зависимостями, и управлять ими. Так как разработка велась на операционной системе Windows, для работы Docker использовался WSL (Windows Subsystem for Linux). Использование docker-файлов из репозитория библиотеки [16] позволило собрать ее под Windows, избежав конфликтов версий и проблем с совместимостью используемых пакетов.
2. Для написания кода использовался редактор Visual Studio Code [28]. Это бесплатная кроссплатформенная среда разработки, поддерживающая различные языки программирования в одном проекте при помощи подключаемых расширений. Решено было выбрать эту среду разработки, так как в Kimera-VIO используются несколько языков: логика приложения написана на C++, параметры сенсоров задаются в yaml-документах, а для рисования графиков используются библиотеки языка Python. Было решено не использовать Visual Studio или Pycharm, так как они предназначены для работы с конкретными языками.
3. Для визуализации результатов работы Kimera-VIO был использован Jupyter Notebook [13], который позволяет интерактивно работать с данными и строить необходимые графики. В проекте представлены инструменты для отладки записей из журналов фронта, бэкенда и lcd частей системы [14], предполагающие использование именно этого инструмента.

4 Описание решения

Основным требованием к решению является наличие поддержки GNSS-данных в алгоритме и повышение точности построенной траектории при запуске алгоритма на датасете EuRoC. Для выполнения задачи необходимо было разобраться с форматом данных, понять устройство конвейера и модифицировать часть модулей Kimera-VIO.

4.1 Формат и преобразование данных

Запуск Kimera-VIO предполагался на датасете EuRoC, который предоставляет данные, собранные с помощью микролетательного аппарата (MAV), оснащённого стереокамерами и IMU. Для каждой локации также присутствуют данные ground-truth, предоставляемые в качестве эталонных — это измерения, полученные от системы оптического трекинга Vicon [24], выдающий данные в виде полного 6-мерного вектора состояния: координаты (x, y, z) и углы поворота вокруг осей ($roll, pitch, yaw$) при частоте обновлений в 100-500Гц и точности порядка нескольких миллиметров. Помимо данных от Vicon датасет содержит измерения с лазерного трекера Leica MS50 [38], которые задаются в виде трех координат (заявленная точность — сотые части миллиметра).

На основе формата эталонных измерений в качестве формата для GNSS было решено также взять три координаты (x, y, z) и временную метку, к которой они относятся. В качестве самих данных брались эталонные измерения (в экспериментах — с добавлением к ним шума). В папку с датасетом, с которой работает алгоритм, была добавлена папка `gnss0`, в которой симулированные данные GNSS хранятся в `.csv`-файле в формате `#timestamp;x;y;z`.

Каждый сенсор задает координаты в своей локальной системе, поэтому для использования их в едином конвейере необходимо преобразовать их в общие координаты. Рядом с данными каждого типа представлен файл с конфигурацией используемого сенсора, где задается матрица трансформации между системами координат сенсора и телом системы (в EuRoC в качестве общей системы выбрано положение IMU-

датчика).

Для использования GNSS-данных в алгоритме необходимо также преобразовать их в координаты системы с помощью матрицы трансформации:

$$p_s^{(h)} = \begin{pmatrix} x_s \\ y_s \\ z_s \\ 1 \end{pmatrix}, T_{BS} = \begin{pmatrix} \mathbf{R}_{3 \times 3} & \mathbf{t}_{3 \times 1} \\ \mathbf{0}_{1 \times 3} & 1 \end{pmatrix} \in SE(3),$$

где T_{BS} — матрица трансформации, задана в однородных координатах, $\mathbf{R}_{3 \times 3}$ — матрица вращения $\in SO(3)$ ³, а $\mathbf{t}_{3 \times 1}$ — вектор сдвига (трансляции); $p_s^{(h)}$ — координаты, полученные от сенсора (переводим в однородные добавлением $w = 1$).

Полное преобразование выглядит следующим образом (переводим координаты в однородные, умножаем на матрицу, переводим обратно в аффинные):

$$p_b^{(h)} = T_{BS} \cdot p_s = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \cdot \begin{pmatrix} x_s \\ y_s \\ z_s \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} \cdot \begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix} + \mathbf{t} \\ 1 \end{pmatrix},$$

$$p_b = \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot p_b^{(h)} = \mathbf{R} \cdot \begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix} + \mathbf{t}.$$

Получаем вектор p_b — GNSS-координаты в общей системе.

³ $SE(3)$ — евклидова группа вращений, $SO(3)$ — специальная ортогональная группа, задает 3×3 ортогональные матрицы с определителем, равным 1. Однородные координаты используются в Кинематическом ВХО, чтобы не накапливать погрешности при вычислениях.

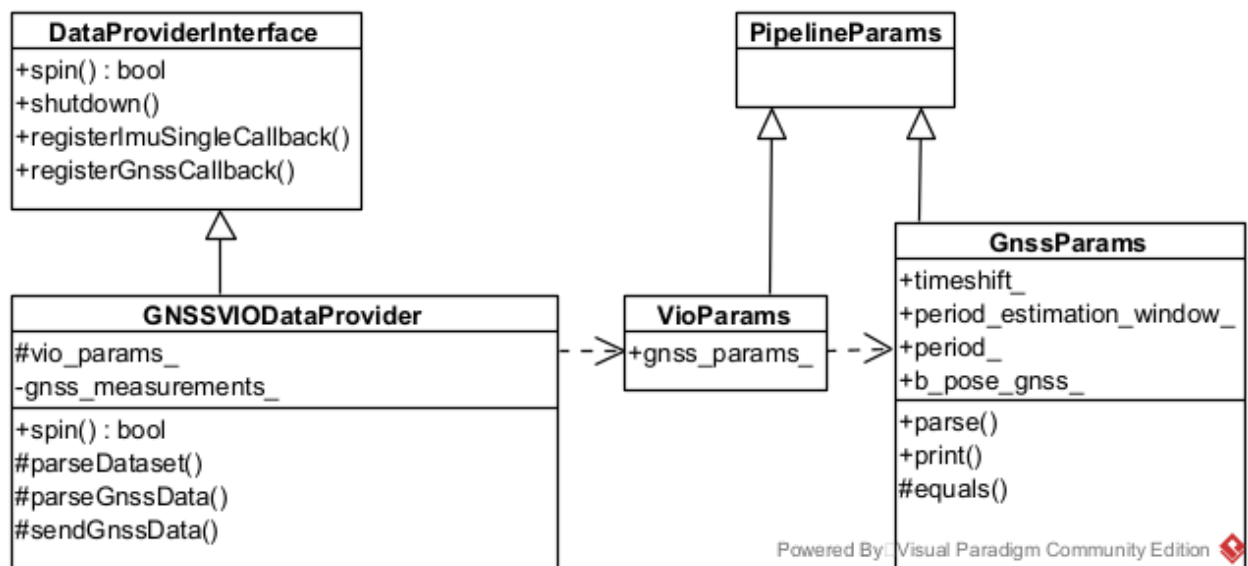


Рис. 11: Обработка датасета и параметры запуска системы

4.2 Обработка параметров и чтение датасета

Данное преобразование происходит на самом первом этапе работы алгоритма — при чтении параметров запуска и данных датасета.

Для обработки GNSS-измерений в библиотеку был добавлен модуль **GNSSVIODataProvider**, который наследуется от базового класса — **DataProviderInterface**. Он отвечает за чтение файлов датасета и заполнение контейнеров данными.

Параметры, необходимые для работы с GNSS-данными, задаются в структуре **GnssParams**. В неё входят:

- Матрица T_{BS} для координат GNSS.
- Сдвиг данных по времени (в секундах), задающий расхождение временных меток GNSS-сенсора с глобальным временем системы.
- Период поступления GNSS-измерений, если он известен заранее. Если же период неизвестен, он вычисляется на основе среднего значения интервалов между n первыми прочитанными измерениями, число которых можно задать параметром `gnss_period_estimation_window_`.

Структура **GnssParams** включается в общую структуру параметров **VioParams**. GNSS-измерения хранятся в векторе `gnss_measurements_` и

передаются в следующий модуль с помощью callback-функций, которые регистрируются при инициализации модулей.

4.3 DataProvider module

По мере чтения изображений модуль `DataProviderInterface` отправляет данные в `DataProviderModule`, который отвечает за синхронизацию информации от различных сенсоров по временным меткам. Измерения каждого типа помещаются в соответствующие контейнеры. В случае GNSS данных это реализуется следующим образом: callback-функция вызывает метод `fillGnssQueue()`, который помещает очередное измерение в потокобезопасный буфер, содержащийся в поле `gnss_data_`.

Когда в буферах накапливается «достаточное количество» данных от камер, IMU и GNSS, вызывается метод `getInputPacket()`, формирующий синхронизированный пакет типа `GnssStereoImuSyncPacket`. Этот пакет передаётся в очередь фронтенда.

Функция, отвечающая за синхронизацию измерений по времени называется `getInputPacket()` и работает следующим образом:

1. Из очереди с изображениями от левой камеры извлекается самый старый кадр.
2. К нему добавляются данные о скорости и ориентации от IMU, полученные методом линейной интерполяции между соседними измерениями.
3. Затем из правой очереди подбирается изображение, максимально близкое по времени к левому.

Если GNSS-данных в очереди недостаточно (например, из-за низкой частоты обновлений), и при этом все вышеперечисленные шаги завершились успехом, то собранный пакет отправляется в следующий модуль.

Если же GNSS-измерения присутствуют, выполняется линейная интерполяция по ближайшим временным меткам t_0 и t_1 , для которых из-

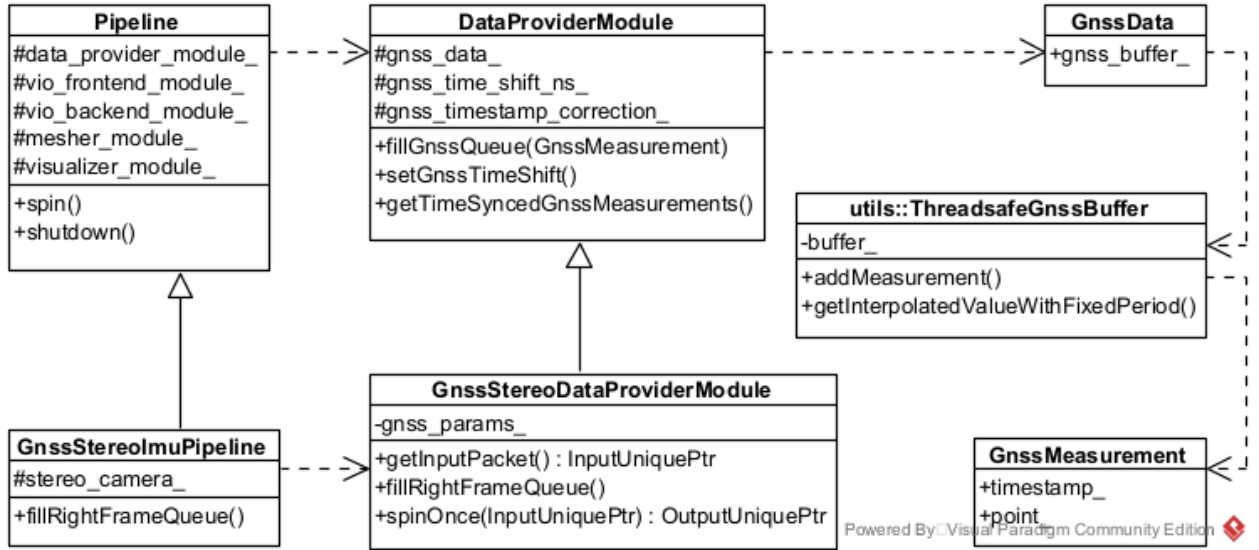


Рис. 12: Обработка и синхронизация данных

вестны координаты y_0 и y_1 , по формуле:

$$y(t) = \begin{cases} y_0, & t_0 = t_1, \\ y_0 + \frac{t - t_0}{t_1 - t_0} (y_1 - y_0), & t_0 < t_1. \end{cases}$$

Результат интерполяции — координата $y(t)$ в момент времени пакета t — добавляется к сформированному пакету, который также передается на фронтенд.

Добавленный в рамках данной работы класс, участвующий в общем конвейере и учитывающий при синхронизации GNSS-данные, представлен на рис. 12 как `GnssStereoDataProviderModule`.

4.4 Frontend module

Модуль фронтенда привязан к конкретному `Pipeline`. Он содержит экземпляр фронтенда, функциональность которого определяется типом `FrontendType`, заданным при запуске алгоритма. Для поддержки GNSS был добавлен новый тип модуля — `GnssVisionImuFrontendModule`, а также соответствующий фронтенд (`GnssStereoVisionImuFrontend`) с новыми типами входных и выходных данных.

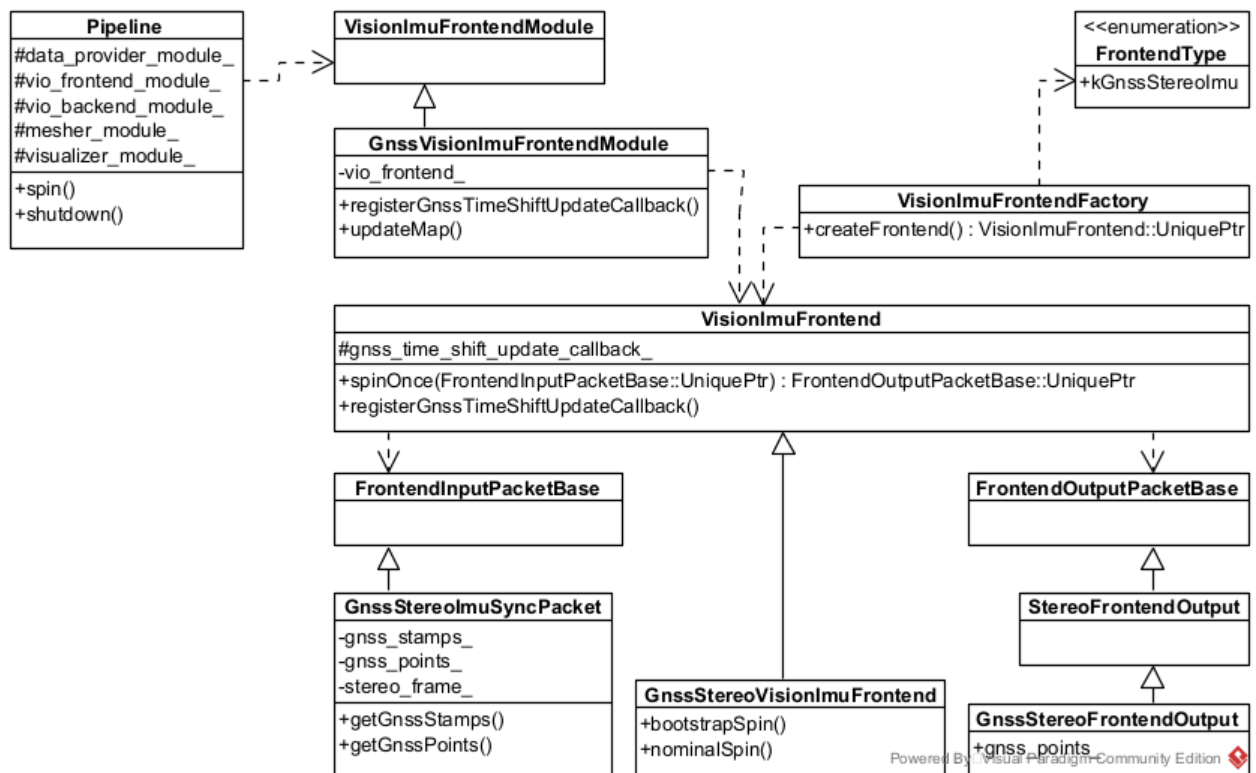


Рис. 13: Архитектура фронтенд-модуля системы

При первом поступлении данных от `DataProviderModule`, когда фронтенд ещё не инициализирован, вызывается функция `bootstrapSpin()`, которая задаёт начальное состояние системы: определяются общие точки на изображениях, инициализируются поза, скорость и смещение. Далее пакет с начальными данными передаётся в следующий модуль.

Если фронтенд уже запущен, то срабатывает `nominalSpin()`, где осуществляется отслеживание ориентиров (feature tracking), триангуляция и предсказание позы по данным IMU. Полученные данные формируются в пакет и передаются далее по конвейеру.

Поскольку фронтенд отвечает за визуальную обработку, работает с измерениями между двумя последовательными кадрами и не взаимодействует с глобальной позицией системы, было решено не вносить в него существенных изменений, а «провести» через него GNSS-данные для обработки их на бэкенде.

4.5 Выбор типа связи и реализация backend module

Как говорилось в обзоре, в SLAM-алгоритмах есть два способа взаимодействия с данными: *tightly-coupled* и *loosely-coupled*. Сильная связь требует явной модели измерения псевдодальности, для GNSS это уравнение вида:

$$\rho_i = \|\mathbf{x} - \mathbf{s}_i\| + cb + \epsilon_i,$$

где ρ_i — измеренная псевдодальность к i -му спутнику, $\mathbf{x}$ — неизвестная текущая позиция приёмника, $\mathbf{s}_i$ — позиция спутника, c — скорость света, b — clock bias в секундах, вместе — поправка на ошибку часов приёмника, ϵ_i — шум измерения.

Такой подход обеспечивает более точную оптимизацию, но требует доступа к сырым GNSS-измерениям. В датасете EuRoC доступа к таким данным нет, поэтому был выбран подход со слабой связью — он проще в реализации и позволяет использовать готовые координаты GNSS в качестве дополнительных факторов.

Модуль бэкенда также привязан к конкретному конвейеру и содержит свой экземпляр бэкенда в зависимости от `BackendType` (рис. 14). Его инициализация происходит по данным от `bootstrapSpin()` из фронтенда: создается граф факторов, добавляются `PriorFactors` на первую позу, скорость и смещение IMU, а также создается первый узел в оптимизаторе (`smoother`).

В рабочем режиме бэкенда выполняется `nominalSpin()`, где происходит добавление в граф новых факторов, удаление старых состояний и факторов за пределами «скользящего окна» и вызов алгоритма оптимизации. Скорректированная поза, состояние графа и карта триангулированных точек добавляются в очереди для `MeshesModule` и `VisualizerModule`, которые используют эти данные для построения и визуализации траектории.

Для интеграции GNSS был реализован `GnssFactor`, основанный на типе `NoiseModelFactor1<Pose3>` из библиотеки GTSAM. Он выделяет из поз графа координаты, не задействуя ориентацию при расчете ошибки по формуле: $r = (\text{pose.translation}()) - p_{GNSS}$.

Для избежания дублирования кода было решено не создавать отдельный `BackendModule`, а отнаследоваться от `RegularVioBackend`, который по умолчанию работает со стерео IMU данными, и с помощью функции `BeforeOptimizeHook()` добавлять GNSS-факторы перед вызовом оптимизации, когда необходимая поза гарантированно находится в графе.

Параметры модели шума задаются в `backend_params_` и влияют на характер ошибки GNSS в уравнениях оптимизации:

- **L2 (гауссовская):** квадратичная ошибка. Чувствительна к выбросам, учитывается как $\|\mathbf{x} - \mathbf{p}_{GNSS}\|_{\Sigma}^2$.
- **Huber:** для ошибок меньше определенного порога ведет себя как L2, если ошибка становится больше — как L1 (линейная).
- **Tukey:** на маленьких ошибках ведет себя как L2, на больших — перестает учитывать измерения в текущей оптимизации. Подходит для использования в зоне плохого сигнала или при поступлении зашумленных данных.

После добавления в граф всех типов факторов происходит оптимизация через вызов функции `updateSmoother()`. Алгоритм iSAM2 минимизирует общую функцию ошибки, включая в неё и ошибку GNSS. Таким образом данные GNSS учитываются при построении траектории.

Также рассматривался вариант добавления GNSS-данных в виде `PriorFactor` при инициализации системы. Однако такое решение привело к усложнению архитектуры и не дало прироста в точности траектории. В связи с этим было решено учитывать факторы GNSS только в рабочем режиме бэкенда.

4.6 Доработки

Помимо основной интеграции GNSS, в библиотеку были внесены дополнительные изменения.

- Разработан скрипт для запуска конвейера с GNSS-данными.

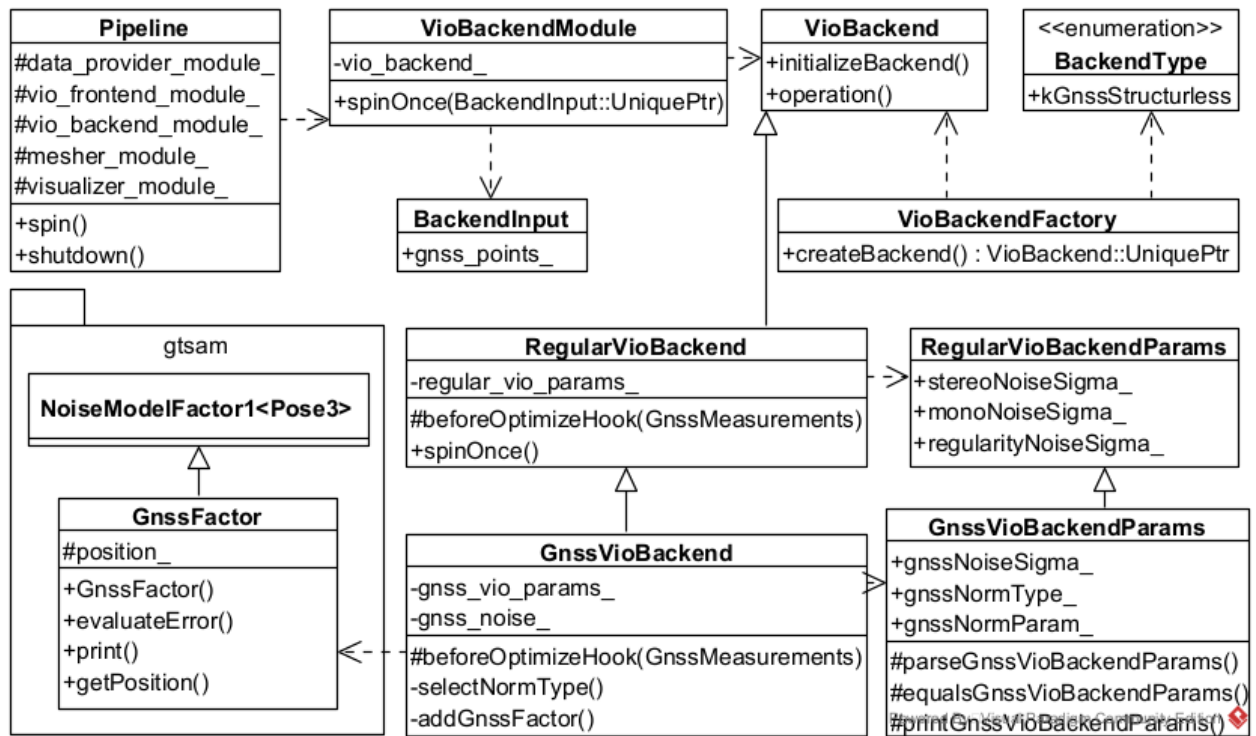


Рис. 14: Архитектура бэкенд-модуля системы

- Обновлен механизм измерения времени работы модулей: ранее использовался тип `high_resolution_clock`, обеспечивающий высокоточные измерения, но не гарантирующий корректную работу в многопоточной среде. Он был заменён на `steady_clock`, что обеспечило стабильность временных оценок при параллельной работе модулей.
- Для проверки соответствия кода стандарту оформления был использован линтер [18], поддерживающий проверку стиля как для C++, так и для Python. Линтер был интегрирован в проект с помощью Git-хуков.
- Были актуализированы оригинальные бенчмарки [14] для анализов траектории Kimera-VIO: обновлены версии библиотек и переписаны некоторые функции оценки.
- Разработаны модульные тесты для новых компонентов системы.

5 Эксперимент

Для оценки точности алгоритма использовался модифицированный набор данных EuRoC с добавлением симулированных GNSS-данных.

5.1 Notebooks

Для отладки и оценки качества работы алгоритма в Kimera-VIO предусмотрены инструменты [14] в формате блокнотов Jupyter Notebook. Каждый блокнот предназначен для анализа одного из компонентов системы — фронтенда, бэкенда и замыкания циклов. Блокноты независимы и обрабатывают файлы журналов, сформированные во время выполнения конвейера.

Блокнот фронтенда показал идентичные результаты при работе конвейера с GNSS-данными и без них, что подтверждает корректность решения: измерения GNSS не должны оказывать влияния на визуальную обработку. Механизм замыкания циклов и связанный с ним блокнот в данной работе не использовались.

Основной анализ производился в блокноте бэкенда, где вычисленная траектория сравнивается с эталонной (ground-truth). Помимо визуального сопоставления траекторий, блокнот содержит расчет абсолютной ошибки позы (Absolute Pose Error, APE), отражающей отклонение текущей оценки положения от эталонного значения в каждый момент времени.

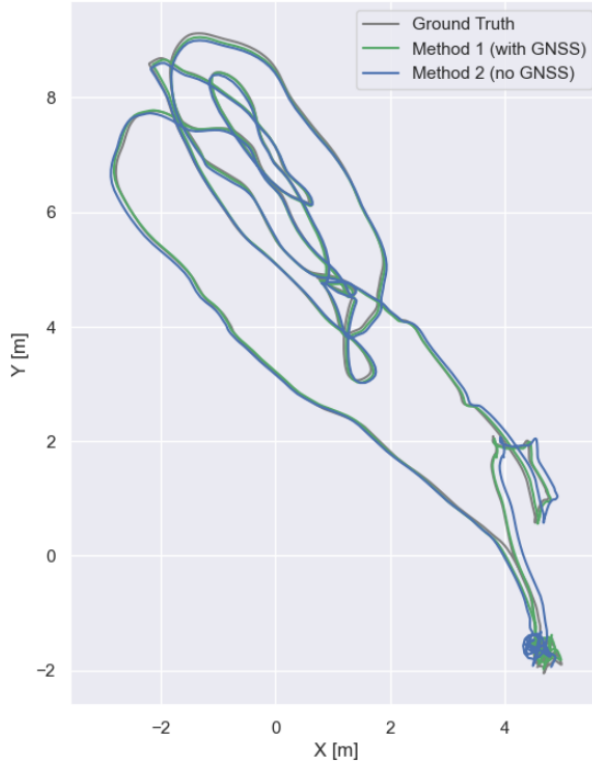
5.2 Сравнение результатов

Согласно оригинальной публикации по Kimera [17], для локации `Machine_Hall_01_easy` из EuRoC наилучшее достигнутое значение среднеквадратичной ошибки (RMSE APE) составляет 0.11 метра. В рамках данной работы, после подбора параметров и проведении серии запусков на реализации с GNSS, удалось снизить ошибку до 0.07 метра.

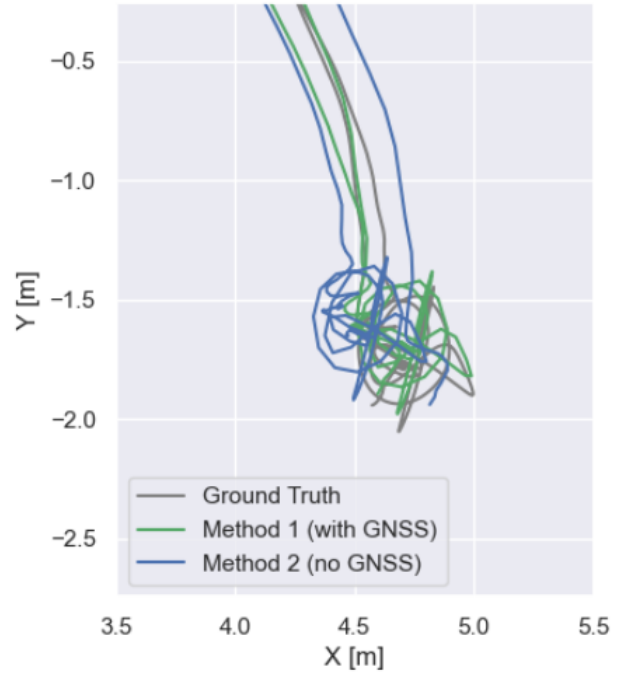
На рисунке 15 представлены траектории: зеленая — результат работы с GNSS, синяя — стандартный VIO-IMU конвейер без GNSS, серая

пунктирная линия — эталонная траектория.

Рисунок 16 демонстрирует сравнение среднеквадратичных ошибок. Полученные результаты не только превосходят результаты стандартной реализации без GNSS, но и оказываются лучше значений, представленных в оригинальной работе. Это подтверждает эффективность интеграции GNSS и достижение основной цели работы — повышения точности оценки траектории.



(a) Полная траектория в XY-плоскости



(b) Фрагмент траектории (возврат в начальную точку)

Рис. 15: Сравнение траекторий: ground-truth, с GNSS и без GNSS

Дополнительно были проведены эксперименты с GNSS-данными, содержащими различный уровень искусственно добавленного шума. Шум равномерно применялся ко всем координатам и моделировался как гауссовский (нормальный) по формуле:

$$x = x_{\text{true}} + \mathcal{N}(0, \sigma^2).$$

где x — значение с шумом, передающееся в систему, x_{true} — истинное значение координаты, $\mathcal{N}(0, \sigma^2)$ — нормально распределённый шум со средним 0 и дисперсией σ^2 .

Несмотря на наличие отклонений в измерениях, точность построенной траектории практически не ухудшилась благодаря использованию моделей ошибок, устойчивых к выбросам: в параметрах системы можно выбрать тип нормы (например, Huber или Tukey), которая подавляет влияние зашумленных измерений при оптимизации графа. Таким образом, даже при работе с неточными или редкими GNSS-данными алгоритм остаётся стабильным и обеспечивает построение довольно точной траектории.

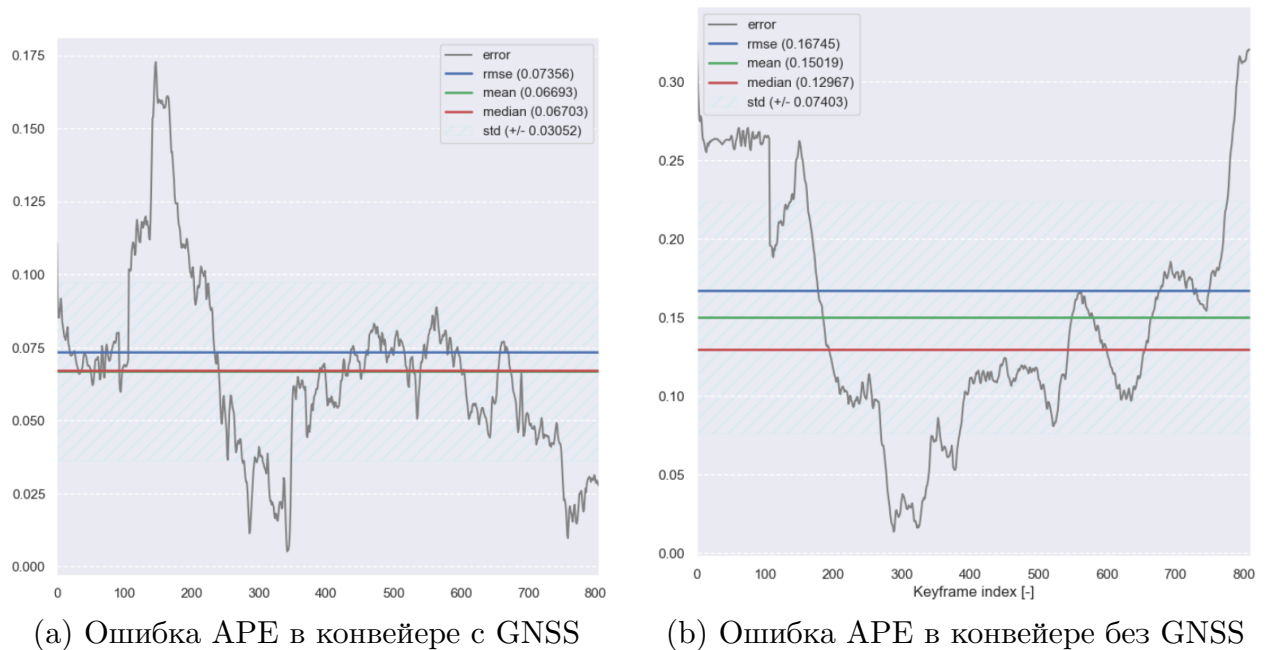


Рис. 16: Сравнение абсолютной ошибки позы (APE) по всей траектории

Заключение

В результате работы были выполнены следующие задачи.

1. Проведен обзор алгоритмов SLAM, интегрирующих GNSS и VIO.
2. Проведен обзор систем VIO-SLAM, потенциально подходящих для внедрения GNSS.
3. Изучена архитектура и особенности работы библиотеки Kimera.
4. Обеспечена загрузка, синхронизация и передача GNSS-данных в конвейер VIO.
5. В датасет EuRoC добавлены GNSS-данные с искусственным шумом.
6. Проведена серия экспериментов с использованием GNSS-данных с разным уровнем шума, сравнение результатов алгоритма показало, что точность построенной траектории увеличилась.

Особенностью реализации, выполненной по итогам данной работы, является возможность продолжения работы системы не только при шумных GNSS-данных, но и при полном их отсутствии. Это позволит использовать алгоритм в местности, где GNSS-данные временно недоступны, при этом построенная траектория не будет терять точности.

Код проекта доступен в github-репозитории [32], разработка ведется под именем `kleo-53`. Реализация была оформлена в виде pull request [21] в официальный репозиторий Kimera-VIO для возможного включения в основную ветку проекта.

Список литературы

- [1] The 1 containerization software for developers and teams.— URL: <https://www.docker.com/products/docker-desktop/> (дата обращения: 20 мая 2025 г.).
- [2] Bala Jibril, Adeshina Steve, Aibinu Abiodun. Advances in Visual Simultaneous Localisation and Mapping Techniques for Autonomous Vehicles: A Review // [Sensors](#). — 2022. — 11. — Vol. 22. — P. 8943.
- [3] CIFASIS. GNSS-Stereo-Inertial Fusion. — URL: <https://github.com/CIFASIS/gnss-stereo-inertial-fusion> (дата обращения: 30 апреля 2025 г.).
- [4] ChangwuLiu. InGVIO. — URL: <https://github.com/ChangwuLiu/InGVIO> (дата обращения: 30 апреля 2025 г.).
- [5] Comprehensive Comparison of GNSS RTK and PPP Techniques.— URL: <https://www.cdebyte.com/news/788> (дата обращения: 2025-05-05).
- [6] A Comprehensive Survey of Visual SLAM Algorithms / Andréa Macario Barros, Yoann Moline, Frédérick Carrel et al. // [Robotics](#). — 2022. — 02. — Vol. 11.
- [7] Engel Jakob, Koltun Vladlen, Cremers Daniel. Direct Sparse Odometry. — 2016. — [1607.02565](#).
- [8] [Experimental Comparison of Open Source Visual-Inertial-Based State Estimation Algorithms in the Underwater Domain](#) / Bharat Joshi, Brennan Cain, James Johnson et al. // IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). — 2019.
- [9] Factor Graphs and GTSAM. — URL: <https://gtsam.org/tutorials/intro.html> (дата обращения: 20 мая 2025 г.).

- [10] HKUST-Aerial-Robotics. GVINS. — URL: <https://github.com/HKUST-Aerial-Robotics/GVINS> (дата обращения: 30 апреля 2025 г.).
- [11] HKUST-Aerial-Robotics. VINS-Fusion. — URL: <https://github.com/HKUST-Aerial-Robotics/VINS-Fusion> (дата обращения: 30 апреля 2025 г.).
- [12] Introduction to Loop Closure Detection in SLAM. — URL: <https://www.thinkautonomous.ai/blog/loop-closure/> (дата обращения: 22 февраля 2025 г.).
- [13] Jupyter Notebook. — URL: <https://jupyter.org/> (дата обращения: 20 мая 2025 г.).
- [14] Kimera-VIO Debugging and Evaluation. — URL: https://github.com/MIT-SPARK/Kimera-VIO/blob/ce8c59b7b273ab5ac29db7e5572e1623760e19c7/docs/kimera_vio_debug_evaluation.md (дата обращения: 12 марта 2025 г.).
- [15] Kimera-VIO: Open-Source Visual Inertial Odometry. — URL: <https://github.com/MIT-SPARK/Kimera-VIO> (дата обращения: 20 ноября 2024 г.).
- [16] Kimera-VIO docker-files. — URL: <https://github.com/MIT-SPARK/Kimera-VIO/blob/master/scripts/docker/Dockerfile> (дата обращения: 20 мая 2025 г.).
- [17] Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping / Antoni Rosinol, Marcus Abate, Yun Chang, Luca Carlone // IEEE Intl. Conf. on Robotics and Automation (ICRA). — 2020. — URL: <https://github.com/MIT-SPARK/Kimera>.
- [18] MIT-SPARK linter. — URL: <https://github.com/MIT-SPARK/linter/tree/master> (дата обращения: 20 мая 2025 г.).

- [19] Nathan Aaron. Loosely Coupled & Tightly Coupled INS & GNSS [2024 Guide].— 2024.— URL: <https://pointonnav.com/news/loose-vs-tight-coupling-gnss/>.
- [20] ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial, and Multimap SLAM / Carlos Campos, Richard Elvira, Juan J. Gomez Rodriguez et al. // [IEEE Transactions on Robotics](#). — 2021. — . — Vol. 37, no. 6. — P. 1874–1890. — URL: <http://dx.doi.org/10.1109/TR0.2021.3075644>.
- [21] Pull-request with GNSS pipeline.— URL: <https://github.com/MIT-SPARK/Kimera-VIO/pull/258> (дата обращения: 20 мая 2025 г.).
- [22] SLAM Overview: From Single Sensor to Heterogeneous Fusion / Weifeng Chen, Chengjun Zhou, Guangtao Shang et al. // [Remote Sensing](#). — 2022. — 11. — Vol. 14. — P. 6033.
- [23] Stereo Visual Simultaneous Localization And Mapping. — URL: <https://docs.exponenta.ru/vision/ug/stereo-visual-simultaneous-localization-mapping.html> (дата обращения: 5 мая 2025 г.).
- [24] Vicon: Award Winning Motion Capture Systems | Tracker.— URL: <https://www.vicon.com/software/tracker/> (дата обращения: 5 мая 2025 г.).
- [25] Vision-based SLAM: монокулярный SLAM.— URL: <https://habr.com/ru/companies/singularis/articles/277537/> (дата обращения: 23 января 2025 г.).
- [26] Vision-based SLAM: стерео- и depth-SLAM.— URL: <https://habr.com/ru/companies/singularis/articles/279035/> (дата обращения: 23 января 2025 г.).
- [27] Visual and Visual-Inertial SLAM: State of the Art, Classification, and Experimental Benchmarking / Myriam Servières, Valerie Renaudin, Alexis Dupuis, Nicolas Antigny // [Journal of Sensors](#).—

2021. — . — Vol. 2021. — P. 054828. — URL: <https://hal.science/hal-03187856>.
- [28] Your code editor. Redefined with AI. — URL: <https://code.visualstudio.com/> (дата обращения: 20 мая 2025 г.).
- [29] Yu Yongqiang Lu; Hongjie Ma; Edward Smart; Hui. Real-Time Performance-Focused Localization Techniques for Autonomous Vehicle: A Review. — URL: <https://ieeexplore.ieee.org/document/9435134> (дата обращения: 20 ноября 2024 г.).
- [30] A comparison of loop closing techniques in monocular SLAM / Brian Williams, Mark Cummins, José Neira et al. // [Robotics and Autonomous Systems](#). — 2009. — 12. — Vol. 57. — P. 1188–1197.
- [31] i2Nav WHU. IC-GVINS. — URL: <https://github.com/i2Nav-WHU/IC-GVINS> (дата обращения: 30 апреля 2025 г.).
- [32] kleo-53/Kimera-VIO. — URL: <https://github.com/kleo-53/Kimera-VIO> (дата обращения: 20 мая 2025 г.).
- [33] A review of visual SLAM for robotics: evolution, properties, and future applications / Basheer Al-Tawil, Thorsten Hempel, Ahmed Abdelrahman, Ayoub Al-Hamadi // [Frontiers in Robotics and AI](#). — 2024. — Vol. Volume 11 - 2024. — URL: <https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2024.1347985>.
- [34] rpng. MINS. — URL: <https://github.com/rpng/MINS> (дата обращения: 30 апреля 2025 г.).
- [35] uzh rpg. rpg_svo_pro_gps. — URL: https://github.com/uzh-rpg/rpg_svo_pro_gps (дата обращения: 30 апреля 2025 г.).
- [36] xiaohong huang. RTK-Visual-Inertial-Navigation. — URL: <https://github.com/xiaohong-huang/RTK-Visual-Inertial-Navigation> (дата обращения: 30 апреля 2025 г.).

- [37] Локализация – Искусственный интеллект. — URL: <http://www.rriai.org.ru/lokalizatsiya.html> (дата обращения: 22 декабря 2024 г.).
- [38] Тахеометр Leica MS50 (1"). — URL: <https://www.geooptic.ru/product/leica-ms50-1?srsltid=AfmB0opcW3hUzxaYDCriu04JKM8RseCklU9pee0AkSN1le-ReY06d4gR> (дата обращения: 5 мая 2025 г.).