

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования «Южный федеральный университет»
(ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ)

Институт компьютерных технологий и информационной безопасности
Кафедра математического обеспечения и применения ЭВМ

К защите допустить:

Зав. кафедрой МОП ЭВМ

_____ Н. Ш. Хусаинов
«___» _____ 2025 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по образовательной программе
«Методы и средства разработки программного обеспечения»
направления 09.03.04 Программная инженерия
на тему:

**«Разработка модуля инерциальной навигации БПЛА с использованием
вспомогательных навигационных данных»**

Руководитель ВКР:
к.т.н., доцент
кафедры МОП ЭВМ

(подпись, дата)

А. Н. Шкурко

Нормоконтроль:
ст. преподаватель
кафедры МОП ЭВМ

(подпись, дата)

О. Н. Родзина

Выполнил:
студент группы КТб04-7

(подпись, дата)

Д. К. Маркова

Таганрог 2025

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

БАКАЛАВРА по образовательной программе
«Методы и средства разработки программного обеспечения»
направления 09.03.04 Программная инженерия

студенту группы КТбо4-7
Марковой Дана Константиновне

1. Тема выпускной квалификационной работы:

Разработка модуля инерциальной навигации БПЛА с использованием
вспомогательных навигационных данных утверждена приказом по ВУЗу № 5585-к
от 16.04.2025 г

2. Требования и исходные данные к работе:

2.1 Среда разработки: Visual Studio 2022, Visual Studio Code, IAR Embedded Workbench for Arm

2.2 ОС разработки: Windows 11

2.3 Язык программирования: JavaScript, C/C++, C#

3. Перечень подлежащих разработке вопросов (содержание работы):

3.1 Введение

3.2 Анализ требований и предметной области

3.3 Проектирование архитектуры системы

3.4 Проектирование пользовательского интерфейса сайта

3.5 Программная реализация

3.6 Тестирование ПО

3.7 Техничко-экономическое обоснование разработки

3.8 Заключение

3.9 Список использованных источников

3.10 Приложение А

3.11 Приложение Б

3.11 Приложение В

3.11 Приложение Г

3.11 Приложение Д

3.11 Приложение Е

4. Перечень графического материала:

4.1 Цель работы – 1 пл.

4.2 Анализ проблемы – 1 пл.

4.3 Обзор аналогов – 1 пл.

4.4 Требования к системе – 1 пл.

4.5 Архитектура системы – 2 пл.

4.6 Алгоритмы – 4 пл.

4.7 Тестирование – 4 пл.

4.8 Апробация ВКР	– 1 пл.
4.9 Выводы по работе	– 1 пл.

5. Консультанты по выпускной квалификационной работе (с указанием разделов):

6. Срок сдачи законченной ВКР руководителю: «07» 06 2025 г.

7. Дата выдачи задания: «10» 02 2025 г.

Руководитель образовательной программы
заведующий кафедры МОП ЭВМ _____

Н.Ш. Хусаинов
(подпись, дата)

Руководитель ВКР:
доцент кафедры МОП ЭВМ,
к. т. н., доцент _____

А.Н. Шкурко
(подпись, дата)

Исполнитель:
студент группы КТбо4-7 _____

Д. К. Маркова
(подпись, дата)

УДК 004.42

МАРКОВА ДАНА КОНСТАНТИНОВНА

«Разработка модуля инерциальной
навигации БПЛА с использованием
вспомогательных навигационных данных»

Квалификационная работа на степень

«БАКАЛАВР» по направлению 09.03.04

Южный федеральный университет,

ИКТИБ, кафедра МОП ЭВМ – 2025 г.,

224 с.

АННОТАЦИЯ

Работа посвящена разработке программного комплекса для моделирования, управления и визуализации параметров полёта беспилотного летательного аппарата с использованием инерциальных и спутниковых данных. Цель создания системы является разработка и реализация алгоритмов, которые будут обеспечивать корректную навигацию и ориентацию БПЛА в различных условиях эксплуатации.

В ходе работы были реализованы ключевые алгоритмы ориентации, стабилизации, позиционирования с объединением инерциальной и глобальной навигационных систем. Выполнена разработка симулятора полёта на языке C# с имитацией работы датчиков, а также клиентского приложения на языке C++, обрабатывающего данные и формирующего управляющие сигналы. Визуализация параметров телеметрии реализована при помощи приложения на платформе Node.js с использованием фреймворка Vue.js.

Система протестирована по основным направлениям: функциональное тестирование алгоритмов, функциональное тестирование веб-приложения, аппаратная проверка компонентов и натурные стендовые испытания. Полученные результаты подтверждают работоспособность комплекса и его пригодность для дальнейшего применения в инженерных и учебных задачах.

UDC 004.42

MARKOVA DANA KONSTANTINOVNA

«Development of the UAV inertial navigation module using auxiliary navigation data»

Qualification work for the degree of
BACHELOR in the direction of 09.03.04
Southern Federal University, ICTIS, the
Department of software engineering– 2025,
224 p.

ANNOTATION

The work is devoted to the development of a software system for modeling, control and visualization of UAV flight parameters using inertial and satellite data. The purpose of the system is to develop algorithms that will ensure correct navigation and orientation of the UAV in various operating conditions.

In the course of work the key algorithms of orientation, stabilization, positioning with integration of inertial and global navigation systems were implemented. Development of flight simulator in C# language with simulation of sensors operation, as well as client application in C++ language processing data and generating control signals was performed. Visualization of telemetry parameters is implemented using Node.js platform with Vue.js framework.

The system has been tested in the following main areas: functional testing of algorithms, functional testing of web-application, hardware verification of components and full-scale bench tests. The obtained results confirm the performance of the complex and its suitability for further application in engineering and educational tasks.

ОГЛАВЛЕНИЕ

Введение.....	11
1 Анализ требований и предметной области.....	12
1.1 Анализ предметной области.....	12
1.2 Разработка требований к программному продукту	12
1.2.1 Назначение и цели создания системы.....	12
1.2.2 Функциональные требования.....	13
1.2.2.1 Обработка данных с датчиков.....	13
1.2.2.2 Расчёт углов наклона для системы ориентации.....	13
1.2.2.3 Система стабилизации	13
1.2.2.4 Система позиционирования.....	13
1.2.2.5 Просмотр телеметрии в веб-интерфейсе.....	14
1.2.3 Нефункциональные требования.....	14
1.2.4 Требования к аппаратной части платформы	14
1.3 Аналогии.....	15
1.3.1 Аналог Ardupilot.....	15
1.3.2 Аналог Pixhawk.....	16
1.3.3 Сводная оценка.....	17
1.4 Стек технологий.....	17
1.5 Обзор на существующие алгоритмы для навигационного модуля.....	18
1.5.1 Анализ существующих алгоритмов для инерциальной навигационной системы.....	18
1.5.2 Методы получения достоверных координат для БПЛА.....	19
2 Проектирование архитектуры системы.....	21
2.1 Схема бизнес-процессов предметной области.....	21
2.1.1 Организационная структура.....	21
2.1.2 Карта процессов	21
2.1.3 Диаграмма BPMN.....	22
2.1.4 Карта систем.....	23
2.2 Диаграмма DFD.....	23

2.3	Объектно-ориентированный анализ и проектирование (UML).....	25
2.3.1	Диаграмма классов.....	25
2.3.2	Диаграмма развертывания.....	26
2.4	Вывод.....	27
3	Проектирование пользовательского интерфейса сайта.....	28
3.1	Подготовка проекта и аналитика.....	28
3.1.1	Анализ бизнеса, конкурентов, определение сильных и слабых сторон.....	28
3.1.2	Ограничения бизнеса.....	28
3.2	Исследования.....	29
3.2.1	Проведение интервью и исследования.....	29
3.2.2	Потребности, страхи и барьеры респондентов.....	29
3.2.3	Полезные и интересные кейсы.....	29
3.3	Проектирование, CJM, JTBD.....	29
3.3.1	User Story.....	30
3.3.2	CJM.....	30
3.3.3	Пользовательский сценарий.....	31
3.4	Проработка прототипа и особенности дизайна.....	31
3.5	Вывод.....	34
4	Программная реализация.....	36
4.1	Разработка алгоритмов.....	36
4.1.1	Алгоритм ориентации.....	36
4.1.2	Алгоритм стабилизации.....	38
4.1.3	Алгоритм позиционирования.....	40
4.1.4	Алгоритм оценки высоты на основе барометра.....	42
4.2	Разработка веб-части.....	44
4.2.1	Серверная часть Node.js.....	44
4.2.2	Клиентская часть Vue.js.....	45
4.3	Взаимодействие подсистем и итоговая архитектура.....	46
5	Тестирование ПО.....	49
5.1	План тестирования.....	49

5.1.1 Цели тестирования.....	49
5.1.2 Типы и методики испытаний.....	49
5.1.3 Критерии начала/окончания.....	49
5.2 Тестирование.....	50
5.2.1 Алгоритмические тесты.....	50
5.2.1.1 Алгоритм ориентации.....	50
5.2.1.2 Алгоритм стабилизации.....	53
5.2.1.3 Алгоритм позиционирования.....	55
5.2.1.4 Алгоритм высоты.....	56
5.2.2 Функциональные тесты.....	57
5.2.3 Аппаратные тесты.....	59
5.2.4 Натурные тесты.....	63
5.3 Вывод.....	67
6 Техничко–экономическое обоснование.....	68
6.1 Обзор рынка программного обеспечения и аналогов.....	68
6.2 Оценка экономических затрат на разработку проекта.....	68
6.3 Модель применения и окупаемости.....	69
6.4 Вывод.....	69
Заключение.....	71
Список использованных источников.....	73
Приложение А. АКТ О ВНЕДРЕНИИ.....	75
Приложение Б. ИТОГИ УЧАСТИЯ В КОНФЕРЕНЦИЯХ.....	76
Приложение В. СВИДЕТЕЛЬСТВО О РЕГИСТРАЦИИ ПРОГРАММЫ.....	78
Приложение Г. КОД СИМУЛЯТОРА.....	79
Приложение Д. КОД КЛИЕНТСКОЙ ЧАСТИ C++.....	119
Приложение Е. КОД ВЕБ-ЧАСТИ.....	203

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящем документе применяются следующие обозначения и сокращения:

Беспилотный летательный аппарат (БПЛА) – летательный аппарат без пилота на борту, управление и/или навигация которого выполняются автоматически или дистанционно.

Инерциальная навигационная система (ИНС) – система, определяющая текущее положение, скорость и ориентацию объекта на основе измерений ускорений и угловых скоростей при помощи инерциальных датчиков.

Вспомогательные навигационные данные – информация, получаемая из внешних источников (сигналы спутниковой навигации), которая может использоваться для коррекции или уточнения данных ИНС.

Система координат NED (North, East, Down) – это геоцентрическая локальная система координат, в которой ось «North» направлена на географический север, ось «East» – на восток, а ось «Down» – вертикально.

MEMS-датчики — микроэлектромеханические сенсоры, преобразующие механические воздействия в электрические сигналы для измерения ускорения, угловой скорости и других параметров.

EKF (Extended Kalman Filter) — алгоритм оценки состояния нелинейных динамических систем, основанный на линеаризации моделей вокруг текущего состояния с использованием разложения в ряд Тейлора, что позволяет применять принципы классического фильтра Калмана для задач навигации, позиционирования и управления.

MCU (Microcontroller Unit) — однокристальная микропроцессорная система, включающая центральное вычислительное ядро, оперативную и программную память, а также встроенные периферийные модули.

FPU (Floating-Point Unit) — специализированный аппаратный сопроцессор или блок внутри центрального ядра, предназначенный для быстрого выполнения арифметических операций с числами с плавающей запятой.

GNSS (Global Navigation Satellite System) — глобальная навигационная спутниковая система, предоставляющая сервис определения местоположения, скорости и времени в любой точке Земли посредством приёма сигналов от орбитальной группировки спутников.

Универсальный асинхронный приёмопередатчик (УАПП, UART, Universal Asynchronous Receiver-Transmitter) — узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами.

Git — распределённая система управления версиями.

ВВЕДЕНИЕ

Современные беспилотные летательные аппараты находят широкое применение в самых разных сферах — от мониторинга и картографии до доставки грузов и спасательных операций. Повышение доступности и миниатюризация компонентов делают дроны всё более популярными не только в промышленности, но и в образовании и научных исследованиях. Однако при всей технологической зрелости аппаратной части, разработка надёжного программного обеспечения для управления полётом, навигации и визуализации телеметрии по-прежнему представляет собой сложную инженерную задачу.

Одной из ключевых проблем является необходимость точного расчёта ориентации и положения БПЛА в реальном времени с использованием инерциальных и спутниковых данных, подверженных шумам и потерям. Также важно обеспечить устойчивую и адаптивную стабилизацию движения, особенно в условиях внешних возмущений.

В рамках данной выпускной квалификационной работы разработан программный комплекс, включающий симулятор полёта, модули расчёта ориентации, стабилизации и позиционирования, а также веб-интерфейс для визуального мониторинга телеметрии. Предложенное решение имеет актуальность в использовании в учебных и исследовательских целях, а также как прототип для последующего переноса на реальные БПЛА.

1 Анализ требований и предметной области

1.1 Анализ предметной области

Современные БПЛА широко применяются в геодезии, аграрном мониторинге, поисково-спасательных операциях. Ключевые требования — автономность, точность позиционирования и быстрота отклика. Полётные контроллеры любительского или полупрофессионального сегмента чаще всего строятся на легковесных MCU и дешёвых MEMS-датчиках, поэтому ключевая задача — добиться баланса между вычислительной сложностью алгоритмов и допустимой погрешностью. Инерциальные системы обеспечивают высокую частоту обновления (100 – 400 Гц), но со временем накапливают ошибку. GNSS-модули, напротив, дают абсолютную позицию, однако имеют низкую частоту (1 – 10 Гц) и задержки. Оптимальным считается гибридный подход: быстрая ИНС, периодическая коррекция по GPS, предиктивные фильтры для выбросов и вибраций. Дополнительные требования проекта — переносимость алгоритмов на микроконтроллер без FPU, открытая архитектура для дальнейших расширений и удобство отладки в симуляторе.

1.2 Разработка требований к программному продукту

1.2.1 Назначение и цели создания системы

Разрабатываемый программный модуль предназначен для обеспечения точного и надёжного определения:

- текущей позиции;
- углов наклона.

Модуль использует данные от инерциальных датчиков, а также вспомогательные навигационные данные, чтобы повысить точность и надёжность навигации. Система разрабатывается с учётом условий эксплуатации БПЛА.

Цель создания системы – обеспечить навигацию и ориентацию БПЛА в различных условиях эксплуатации. Программный модуль обеспечивает

достижение следующих целей:

- разработка алгоритмов сбора и обработки данных от блока IMU;
- разработка алгоритмов сбора и обработки данных от блока ГНС;
- разработка алгоритмов ориентации, позиционирования и стабилизации в режиме реального времени с возможностью непрерывной коррекции по внешним данным;
- вывод телеметрии на веб-страницу с целью мониторинга и тестирования.

1.2.2 Функциональные требования

1.2.2.1 Обработка данных с датчиков

Получение данных со всех необходимых датчиков — акселерометра, гироскопа, магнитометра, барометра и GPS — с последующим преобразованием этих данных в формат, подходящий для дальнейшей обработки алгоритмами.

1.2.2.2 Расчёт углов наклона для системы ориентации

Вычисление углов наклона по крену, тангажу и рысканию с помощью работы алгоритма ориентации.

1.2.2.3 Система стабилизации

Система стабилизации должна обеспечивать автоматическое удержание БПЛА в заданной ориентации и поддерживать стабильную траекторию полёта. Для этого должна использоваться совокупность алгоритмов фильтрации и регуляторов, обрабатывающих данные от инерциальных сенсоров и внешних источников. Результирующие управляющие воздействия формируются в масштабе времени, близком к реальному, и передаются на двигатели для немедленной компенсации отклонений.

1.2.2.4 Система позиционирования

Нахождение данных о текущей позиции в глобальной и локальной системах позиционирования с помощью инерциальной и глобальной навигационных систем.

1.2.2.5 Просмотр телеметрии в веб-интерфейсе

Возможность просматривать все данные в масштабе времени, близком к реальному, в удобном для пользователя виде.

1.2.3 Нефункциональные требования

Программный модуль должен включать в себя:

- программную реализацию всех требуемых алгоритмов;
- поддержку работы с типовыми инерциальными датчиками (акселерометр, гироскоп), а также возможность подключения иных датчиков;
- программный модуль не должен содержать тригонометрические расчеты в алгоритмах ориентации для возможности его использования в слабых контроллерах.

Программный модуль должен обладать следующими характеристиками:

- производительность: алгоритмы должны обеспечивать обновление данных ориентации и положения не реже, чем 100 Г;
- надёжность: система должна устойчиво работать при кратковременных перебоях в поступлении данных от ГНС;
- архитектура: модульное построение ПО для дальнейшего развития и интеграции новых алгоритмов навигации.

1.2.4 Требования к аппаратной части платформы

Навигационный модуль ориентирован на лёгкие беспилотные аппараты, поэтому закладывается работа на компактной, экономичной вычислительной базе. Достаточно ядра среднего класса по типу ARM с поддержкой базовых операций над числами с плавающей точкой и объёмом памяти, позволяющим разместить фильтры ориентации и буферы телеметрии без жёсткой оптимизации. Состав сенсоров включает типичный набор MEMS-устройств: гироскоп, акселерометр, магнитометр и барометр — каждого достаточно «потребительского» уровня, но с приемлемым шумом, чтобы алгоритм мог удерживать устойчивую ориентацию. Для абсолютной навигации

предусматривается приёмник спутниковых систем, подключаемый по унифицированному последовательному интерфейсу (UART). Передача телеметрии во внешние приложения выполняется через распространённый беспроводной канал короткого радиуса действия, обеспечивающий обмен данными без громоздкого наземного оборудования. Энергопотребление узла не должно становиться критичным для типовой аккумуляторной батареи мультикоптера; в электрической части допускаются лишь умеренные пульсации питания, чтобы не ухудшать точность датчиков.

1.3 Аналоги

1.3.1 Аналог Ardupilot

ArduPilot — одно из наиболее зрелых и комплексно развитых свободно распространяемых решений в области автоматизированного управления беспилотными летательными аппаратами [1]. Кодовая база охватывает воздушные, наземные и подводные платформы и ориентирована преимущественно на микроконтроллеры семейства STM32, а также на одноплатные вычислительные системы под управлением Linux (Raspberry Pi, BeagleBone). К числу основных преимуществ относятся: (1) высокая степень промышленной зрелости экосистемы, сопровождаемой крупнейшим пользовательским сообществом и полнотой драйверов для большинства коммерчески доступных инерциальных измерительных модулей, приёмников GNSS и дальномеров; (2) широкий набор режимов полёта, включающий как ручную стабилизацию, так и полностью автономные миссии с возможностью сценарного программирования при помощи Lua; (3) полная совместимость с протоколом MAVLink, обеспечивающая интеграцию с наземными станциями QGroundControl и Mission Planner, а также с ROS 2 и облачными телеметрическими сервисами; (4) двухкаскадный расширенный фильтр Калмана (EKF 2/3), позволяющий комплексировать мульти-GNSS-данные, барометрическую высоту, данные оптического потока и визуальной одометрии. К ограничивающим факторам следует отнести значительный объём прошивки (более 1 МБ в сжатом виде для контроллера Pixhawk 4),

высокую сложность параметризации (свыше восьмисот конфигурационных параметров) и повышенные требования к вычислительным ресурсам, особенно при отсутствии аппаратной поддержки операций с плавающей точкой.

Разрабатываемая навигационная платформа ориентирована, прежде всего, на исследовательские и образовательные задачи. Её ядро объёмом около 60 КБ, реализующее упрощённый комплементарный фильтр и облегчённую модификацию EKF, не требует RTOS-обвязки и легко переносится на микроконтроллеры без FPU. Такая архитектура упрощает анализ кода и экспериментальное изменение алгоритмов. Вместе с тем, для достижения функционального паритета с ArduPilot (режим RTL, геозонирование, детальное бортовое логирование) должен быть либо выполнен дальнейший инкремент разработки, либо обеспечено взаимодействие через MAVLink-шину.

1.3.2 Аналог Pixhawk

Pixhawk/PX4 представляет собой открытый аппаратно-программный комплекс, сочетающий эталонную плату управления и прошивку PX4, функционирующую под управлением NuttX RTOS [2]. Оригинальные платы оснащаются резервированными IMU-блоками, вибрационным мониторингом и CAN-шиной для интеллектуальных датчиков, что существенно повышает надёжность. Модульная структура прошивки (uORB-шина сообщений, изолированные драйверы, оценщики и регуляторы) допускает замену отдельных подсистем без полной перекомпиляции проекта; средства QGroundControl предоставляют графический интерфейс для калибровки датчиков и редактирования полётных заданий, а поддержка режимов HITL/SITL (Gazebo, JMAVSIM) упрощает верификацию алгоритмов до их развёртывания на аппаратуре. К недостаткам следует отнести высокую стоимость оригинальных контроллеров, повышенную сложность RTOS-окружения для разработчиков начального уровня и системные задержки, возникающие при интенсивном SD-логировании.

Предлагаемая платформа не конкурирует с PX4 по полноте функционала готового автопилота, но обеспечивает более низкий порог входа: единый репозиторий на C/C++, отсутствие RTOS-зависимостей, возможность выполнения всей экспериментальной цепочки (симулятор, клиент, веб-сервер) на персональном компьютере под Windows 11. Такой подход сокращает цикл «модификация – тестирование» и позволяет визуализировать телеметрию в режиме реального времени без привлечения внешних инструментов (FlightPlot, uLog-парсеры).

1.3.3 Сводная оценка

ArduPilot и Pixhawk/PX4 представляют собой полнофункциональные, промышленно зрелые экосистемы, ориентированные на серийные либо соревновательные БПЛА и требующие значительных вычислительных и материальных ресурсов, а также существенного времени на освоение. Разработанная навигационная платформа предназначена для научно-образовательного применения: минимальные аппаратные требования, прозрачная архитектура, высокий уровень наблюдаемости внутренних процессов и ускоренный цикл экспериментальной проверки алгоритмов. Такой баланс делает её целесообразным инструментом для лабораторных исследований, курсового и дипломного проектирования, а также для быстрой апробации новых методов инерциально-спутниковой интеграции и адаптивного управления.

1.4 Стек технологий

Разработка программного комплекса велась с использованием современных технологий, обеспечивающих высокую производительность, удобство поддержки и масштабирования системы.

На уровне симулятора применялась платформа .NET с использованием языка C# и графической библиотеки WinForms, что позволило быстро реализовать модель динамики квадрокоптера с визуализацией параметров.

Клиентская часть, выполняющая обработку данных, реализацию алгоритмов ориентации, стабилизации и позиционирования, разработана на

C++ с использованием Windows Forms. Это обеспечило прямой доступ к системным ресурсам и гибкую работу с низкоуровневыми структурами данных, а также позволило использовать UART-соединение для передачи команд обратно в симулятор.

Для серверной части был выбран Node.js из-за его асинхронной неблокирующей архитектуры, подходящей для обработки множества WebSocket-соединений. Передача телеметрии во фронтенд осуществляется с помощью библиотеки Socket.IO, которая упрощает двустороннюю коммуникацию между клиентом и сервером.

Фронтенд реализован с использованием фреймворка Vue.js 3, обеспечивающего реактивную модель интерфейса и компонентный подход. Визуализация данных выполнена с помощью библиотеки D3.js.

Выбранный стек технологий обеспечил стабильную работу всех компонентов комплекса, гибкость при отладке и возможность расширения функционала в дальнейшем.

1.5 Обзор на существующие алгоритмы для навигационного модуля

1.5.1 Анализ существующих алгоритмов для инерциальной навигационной системы

Эффективность инерциальной навигации определяется качеством алгоритма фильтрации, то есть способа объединения высокочастотных измерений гироскопа с низкочастотными, но абсолютными опорными векторами акселерометра и магнитометра. Наиболее простым и до сих пор широко применяемым решением остаётся комплементарный фильтр, представляющий собой взвешенную суперпозицию двух полосовых звеньев противоположной характеристики. При грамотном выборе коэффициентов он обеспечивает угловую ошибку порядка 2 - 3 градусов при частоте обновления 100 Гц и практически не нагружает микроконтроллер. Однако отсутствие явной модели шумов датчиков ограничивает его адаптивность: при изменении спектра вибраций или при резких манёврах требуется ручная перенастройка

коэффициентов. Расширенный фильтр Калмана (ЕКФ) решает поставленную задачу в стохастической постановке. Линеаризуя нелинейную динамику кватерниона в окрестности текущего состояния, ЕКФ использует ковариационные матрицы для описания как аппаратных шумов, так и неопределённостей измерений. В типовом исполнении (с девятью состояниями) алгоритм обеспечивает среднюю погрешность менее одного градуса на стандартизированных тестах, при этом предъявляет жёсткие требования к вычислительным ресурсам: матричные операции целесообразно исполнять на ядрах с аппаратным FPU и объёмом ОЗУ не менее 192 КБ. В последние годы внимание исследователей привлекают инвариантный ЕКФ и Unscented Kalman Filter, избегающие аналитического расчёта Якобианов и лучше работающие на больших угловых скоростях, однако их программная реализация требует ещё большего объёма оперативной памяти, что ограничивает применение на микроконтроллерах семейства STM32F4 [3]. По совокупности факторов — точность, вычислительная нагрузка, предсказуемость латентности — в настоящей работе принят гибридный подход: базовый комплементарный фильтр дополняется минимальной ЕКФ-веткой, отвечающей за адаптивную оценку дрейфа и температурного сдвига гироскопов.

1.5.2 Методы получения достоверных координат для БПЛА

Для получения абсолютной навигационной информации наиболее распространённым каналом является спутниковое позиционирование. Одномодульное решение обеспечивает типичную метрическую точность при частоте обновления от одного до десяти герц и задержке не более 200 мс. Дифференциальные схемы (RTK) эксплуатируют фазу несущей и коррекционный поток, что снижает погрешность до сантиметрового уровня; однако для малосерийных БПЛА RTK оправдан лишь в задачах высокоточной фотограмметрии из-за необходимости базовой станции и радиоканала. Барометрическая телеметрия дополняет GNSS, обеспечивая относительную высотную точность порядка трёх десятых метра после термокомпенсации. В

замкнутых условиях применяются технологии локального позиционирования: ультра-широкополосные маяки (UWB), оптический поток, визуальная одометрия и лидарная навигация. Последние существенно повышают требуемую вычислительную мощность, но дают независимый от спутников канал коррекции дрейфа. В рассматриваемой платформе реализован базовый GNSS-барометрический контур, а интерфейс данных оставлен расширяемым для UWB- или оптических сенсоров.

2 Проектирование архитектуры системы

2.1 Схема бизнес-процессов предметной области

В данном разделе представлена схема бизнес-процессов, демонстрирующая организационную структуру и процессы управления платформой.

2.1.1 Организационная структура

Отдельно представлена организационная структура, отражающая распределение ролей между разработчиками, инженерами-испытателями и пользователями интерфейса. Основные процессы включают планирование и проведение испытаний, работу с телеметрией, анализ данных и сопровождение программного обеспечения.

2.1.2 Карта процессов

Карта процессов демонстрирует основные бизнес-процессы, поддерживаемые системой. Показана иерархия процессов на рисунке 1. Представленная схема отражает общий жизненный цикл разработки навигационного модуля для БПЛА, включающий три ключевых этапа: требования и проектирование, разработка и интеграция, а также тестирование. На первом этапе проводится анализ предметной области и аналогов, формулируются технические требования, разрабатывается архитектура системы и планируются ресурсы. Далее реализуются прошивка с навигационными алгоритмами, создаётся программный симулятор дрона, настраивается веб-интерфейс телеметрии и выполняется интеграция аппаратной части. Заключительный этап включает модульное и интеграционное тестирование в симуляторе, натурные испытания с анализом логов, а также оформление итоговой технической документации. Такая структура позволяет поэтапно обеспечить точность, стабильность и практическую применимость навигационной системы.

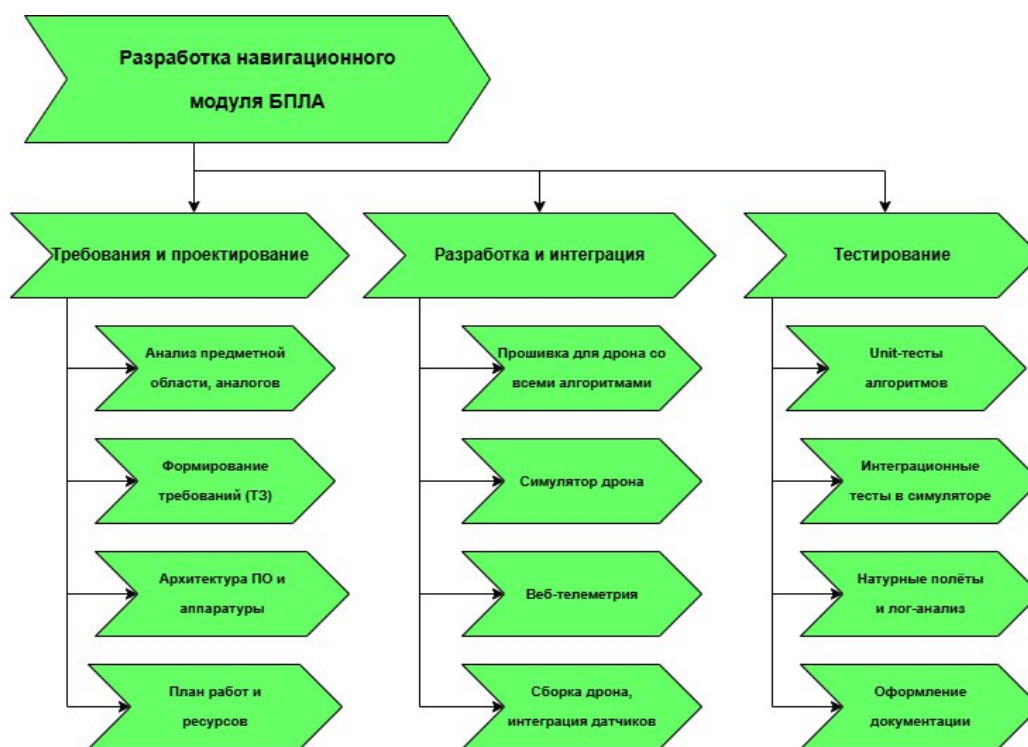


Рисунок 1 — Карта процессов

2.1.3 Диаграмма BPMN

На представленной диаграмме BPMN, изображенная на рисунке 2, показан полный цикл взаимодействия между участниками и подсистемами при запуске и работе симулятора дрона. Пользователь поэтапно запускает симулятор, клиентскую часть и web-сервер, после чего наблюдает телеметрию в интерфейсе. Симулятор генерирует данные виртуальных датчиков и передаёт их по UART клиентскому приложению, где происходит приём и обработка информации, включая расчёт ориентации, стабилизации и позиционирования. Далее формируются управляющие сигналы и телеметрия, которая передаётся через WebSocket на веб-сервер, откуда поступает в интерфейс для визуализации. Этот процесс замыкается в симуляторе, где отображается реакция дрона. Такая схема отражает распределённую архитектуру с потоками данных между независимыми модулями и обеспечивает удобную отладку в реальном времени.

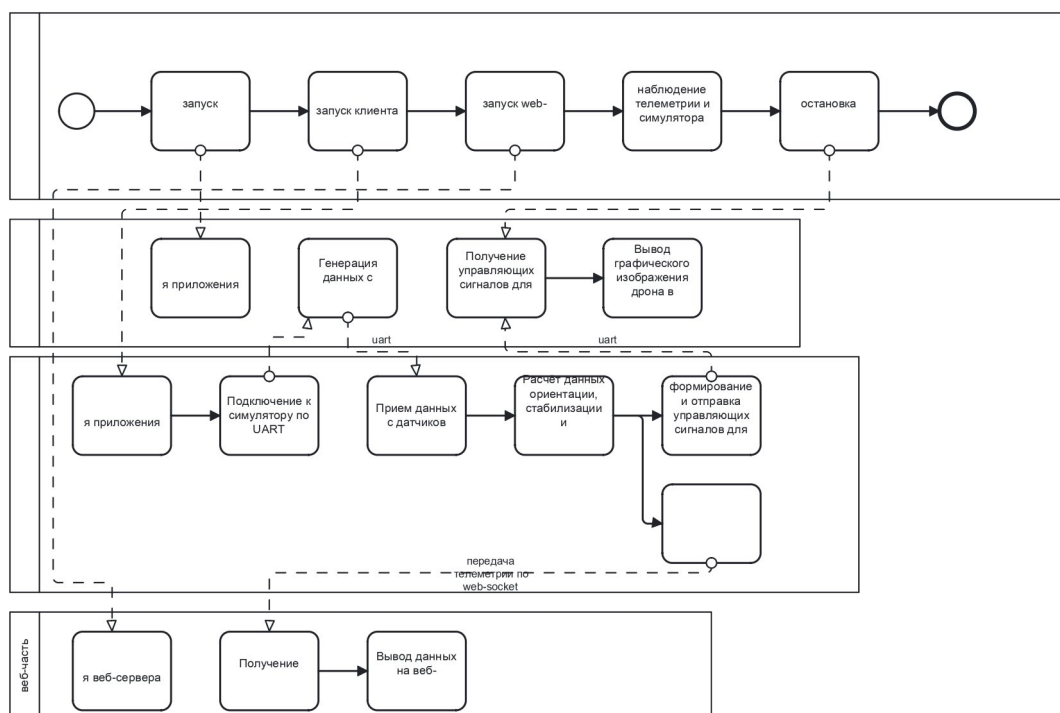


Рисунок 2 - Диаграмма BPMN

2.1.4 Карта систем

Диаграмма, изображенная на рисунке 3, показывает разбиение на модули, в том числе симулятор, обработчик на C++, веб-сервер и клиентское приложение. Это упрощает контроль над архитектурой и облегчает модернизацию.



Рисунок 3 - Карта систем

2.2 Диаграмма DFD

Для лучшего понимания информационных потоков была построена диаграмма потоков данных (DFD), изображенная на рисунке 4, охватывающая основные этапы взаимодействия между компонентами во время тестового

полета. Оператор инициирует запуск симулятора. Далее система генерирует данные от виртуальных датчиков, которые проходят этапы фильтрации и инерциального расчёта ориентации. Полученные данные корректируются при наличии GPS-сигнала, после чего происходит упаковка телеметрии и формирование управляющих сигналов для моторов. Телеметрия передаётся на web-сервер по протоколу WebSocket и отображается в визуальном интерфейсе. Диаграмма демонстрирует замкнутый цикл передачи и обработки данных, поддерживающий непрерывную работу симулятора и обратную связь от пользователя.

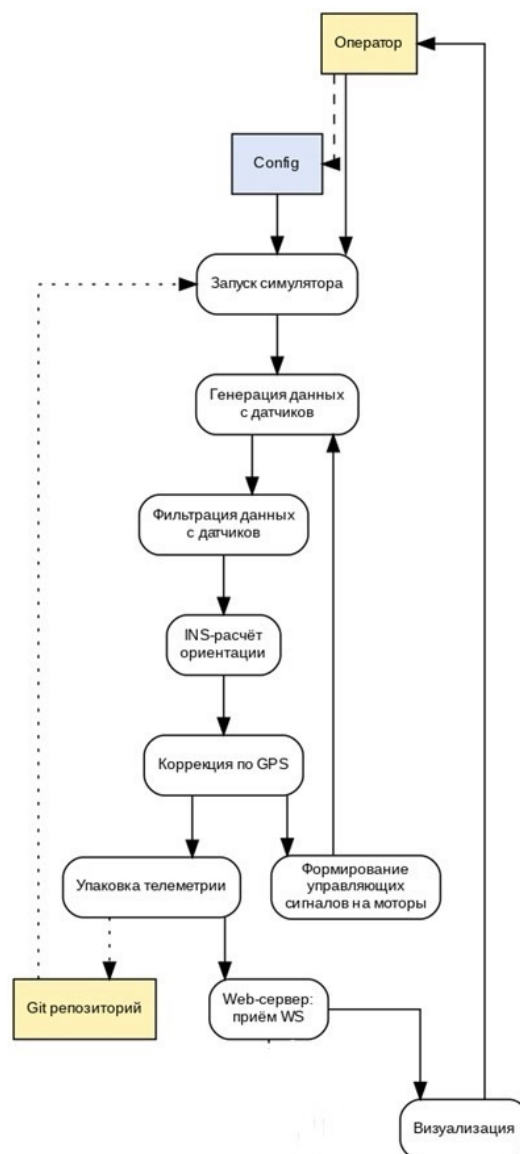


Рисунок 4 - DFD (Data Flow Diagram)

2.3 Объектно-ориентированный анализ и проектирование (UML)

На этапе ОО-проектирования применены основные UML-диаграммы для описания логики работы компонентов.

2.3.1 Диаграмма классов

Диаграмма классов, изображенная на рисунке 5, отображает абстрактную структуру модулей, их иерархию и зависимости. Класс Simulator генерирует данные от датчиков IMU и барометра, используя объект SerialChannel для передачи. Client принимает эти данные, обрабатывает их через FlightController и формирует команды для моторов. Параллельно создаётся TelemetryPacket, который через объект WebSocketChannel передаётся на WebServer и отображается в пользовательском интерфейсе. Диаграмма демонстрирует, как модули обмениваются данными и формируют управляющие действия в рамках общей архитектуры. Это помогает определить интерфейсы между модулями, выделить обобщения и продумать механизмы повторного использования кода.

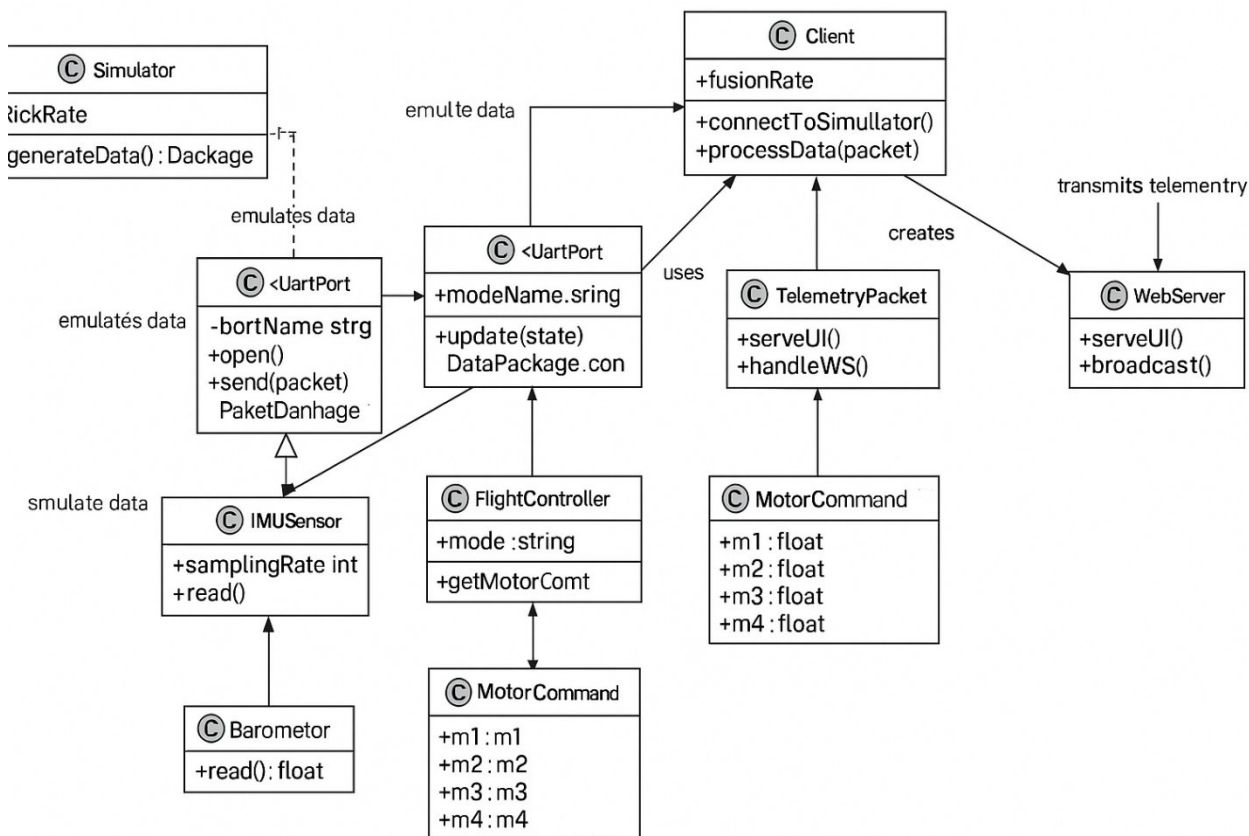


Рисунок 5 - Диаграмма классов

2.3.2 Диаграмма развертывания

Диаграмма развертывания, изображенная на рисунке 6, демонстрирует физическое размещение компонентов: симулятор на C# взаимодействует с клиентским приложением на C++, которое, в свою очередь, пересылает данные через Node.js в веб-интерфейс. Визуализация такого уровня необходима для технической документации и упрощает задачи развертывания и поддержки системы.

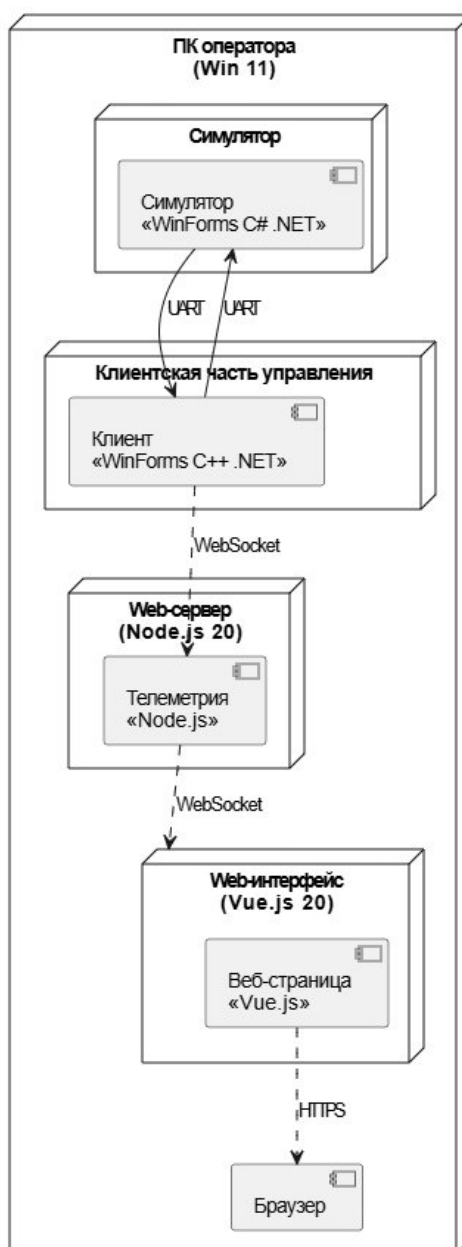


Рисунок 6 - Диаграмма развертывания

2.4 Вывод

Раздел посвящен этапу проектирования. Описана архитектура системы: ключевыми компонентами являются симулятор, клиентская часть для симулятора с алгоритмами, серверная часть и веб-интерфейс. Приводится подробное описание каждой части системы, их функций и взаимодействия друг с другом.

Также описаны сценарии использования, представлены различные виды диаграмм, иллюстрирующие бизнес-процессы и взаимодействие компонентов системы.

Этот раздел важен для понимания общей концепции и технических аспектов разработки системы, обеспечивая основу для дальнейшей реализации и тестирования проекта.

3 Проектирование пользовательского интерфейса сайта

3.1 Подготовка проекта и аналитика

3.1.1 Анализ бизнеса, конкурентов, определение сильных и слабых сторон

Проект нацелен на создание специализированного веб-интерфейса для мониторинга телеметрии БПЛА. В процессе предварительного анализа были исследованы решения Mission Planner, QGroundControl и DroneDeploy. Эти платформы предоставляют широкие функциональные возможности и расширенные параметры настройки, однако в большинстве случаев они требуют высокой квалификации пользователя и перегружены графическими и текстовыми компонентами. Основным недостатком стала чрезмерная сложность интерфейса и ограниченная гибкость в локальных средах без интернета.

Создаваемое приложение ориентировано на образовательную и инженерную аудиторию, и его сильными сторонами стали простота подключения, минималистичный дизайн, фокусировка на ключевых показателях и независимость от облачных сервисов. Основным приоритетом является снижение когнитивной нагрузки и упрощение восприятия графиков для пользователей с разным уровнем подготовки.

3.1.2 Ограничения бизнеса

- Ключевыми ограничениями проекта являются необходимость работы в локальной сети без доступа к интернету; использование только бесплатных и открытых библиотек; совместимость с маломощными устройствами (в том числе планшетами и устаревшими ноутбуками); отсутствие возможности для постоянной технической поддержки со стороны пользователя. Это предопределило выбор лёгких технологий, отказ от сложной аутентификации и реализацию адаптивного интерфейса.

3.2 Исследования

3.2.1 Проведение интервью и исследования

Для анализа пользовательских ожиданий было проведено 5 интервью с целевыми пользователями: два инженера, два преподавателя кафедры и один студент, участвующий в разработке БПЛА. Интервью проводились по структурированной схеме. Участникам предлагалось описать идеальный интерфейс мониторинга и отметить элементы, которые мешают восприятию данных.

3.2.2 Потребности, страхи и барьеры респондентов

Основные потребности пользователей: стабильная передача данных, читаемые графики, возможность поставить визуализацию на паузу и сохранить текущий срез. Страхи: перегруженность интерфейса, зависание при большом объеме данных, необходимость в установке дополнительного ПО. Барьеры: ограниченное знание английского языка, слабая техническая подготовка некоторых пользователей.

3.2.3 Полезные и интересные кейсы

Инженеры указали на необходимость сравнивать данные с разных источников. Один из студентов предложил ввести функцию быстрой очистки графиков перед запуском новой сессии.

3.3 Проектирование, CJM, JTBD

Проектирование пользовательского опыта включает формализацию потребностей пользователей через концепции построения пользовательских историй (User Story) и моделирования пути пользователя (Customer Journey Map, CJM). Это необходимо для обеспечения соответствия интерфейса реальным сценариям эксплуатации системы, повышения удобства работы с данными и снижения вероятности ошибок при интерпретации телеметрии.

3.3.1 User Story

Пользовательские истории были сформулированы на основе анализа задач инженеров и преподавателей, использующих телеметрию для тестирования алгоритмов управления БПЛА. Основная целевая аудитория — это разработчики, тестирующие свою прошивку, и педагоги, демонстрирующие поведение системы студентам.

1. Как инженер, я хочу видеть текущие значения углов ориентации дрона на отдельном графике, чтобы оперативно оценивать стабильность работы PD-регулятора.
2. Как преподаватель, я хочу иметь возможность ставить график на паузу и сбрасывать его, чтобы каждый тест начинался с чистых данных.
3. Как пользователь, я хочу сам выбирать, какие сигналы отображаются, чтобы не загромождать интерфейс лишними данными.
4. Как аналитик, я хочу сравнить данные по высоте от разных источников (барометр, лазер, инерциальная система), чтобы проверить корректность алгоритма интеграции.

User Story позволили сфокусироваться на разработке понятного, адаптируемого интерфейса, минимизирующего количество действий для получения необходимой информации.

3.3.2 CJM

Карта пути пользователя описывает пошаговое взаимодействие с системой на примере типичного сценария стендового тестирования:

1. Подготовка: инженер запускает симулятор и клиентское ПО, открывает веб-интерфейс в браузере. Интерфейс автоматически подключается к WebSocket и начинает отображать поток данных.
2. Настройка: пользователь регулирует частоту отображения для каждого графика. Отмечает нужные чекбоксы для отображения конкретных сигналов.

3. Наблюдение: дрон начинает симуляцию движения. Пользователь следит за графиками и положением на координатной плоскости. Особое внимание уделяется расхождению между GPS и INS-координатами.
4. Анализ: при появлении отклонений пользователь ставит график на паузу, визуально оценивает поведение, может сравнить предыдущие сигналы. После анализа нажимает кнопку «Обновить», график очищается.
5. Завершение: пользователь закрывает интерфейс, данные не сохраняются, все операции происходят без необходимости регистрации или ручной загрузки.

3.3.3 Пользовательский сценарий

Инженер открывает интерфейс перед началом теста, устанавливает частоту 25 Гц, отключает лишние линии, запускает тест. При появлении отклонений нажимает «пауза», делает скриншот, затем очищает график и повторяет цикл. Преподаватель наблюдает за цветовой индикацией и отклонениями от допустимых значений

3.4 Проработка прототипа и особенности дизайна

Интерфейс построен вокруг задачи быстрого чтения телеметрии без лишней визуальной нагрузки. Главная конструктивная единица - карточка с графиком. Четыре карточки располагаются в одной колонке на широком экране и перестраиваются в ленту на узком. Все панели управления находятся над графиками, чтобы пользователь видел органы управления и данные в одной зрительной зоне.

Цветовая схема минималистична. Фон формируется плавным градиентом от бежевого к голубому, что создаёт лёгкий контраст с белыми карточками. Акцентный цвет синий выделяет интерактивные элементы. Дополнительные оттенки зарезервированы для линий графиков и сигналов

состояния. Общий набор укладывается в четыре тона, что упрощает поддержку дизайна-системы и снижает когнитивную нагрузку.

Типографика основана на шрифте Manrope. Размеры подобраны так: 32 пкс для заголовка страницы, 20 пкс для заголовков карточек, 14 пкс для служебного текста. Это обеспечивает читаемость при просмотре с расстояния 50–60 см на ноутбуке. Базовая модульная сетка кратна 8 пкс, что упрощает выравнивание элементов и масштабирование интерфейса.

Компоненты управления сведены к трем типам: ползунок, чек-бокс и кнопка. Ползунок используется для задания частоты обновления, чек-боксы - для включения и выключения линий, кнопки - для операций паузы и очистки данных. Для всех элементов прописаны состояния `hover`, `active` и `disabled`; изменения выражаются только в изменении насыщенности основного цвета, без изменений формы, поэтому интерфейс воспринимается стабильным.

SVG-графики рисуются D3. Каждая новая точка добавляется в конец соответствующего массива данных, устаревшие точки удаляются с противоположной стороны, а линия сдвигается без пересчёта точек.

Дизайн ориентирован на быстрый разбор данных и одинаково пригоден для разработчика, который ищет точные цифры, и для преподавателя, которому достаточно убедиться, что параметры остаются в допустимых пределах.

На рисунке 7 можно увидеть дизайн интерфейса.



Рисунок 7 - Дизайн веб-интерфейса

На рисунке 8 можно увидеть веб-страницу во время работы симулятора.



Рисунок 8 - Веб-страница во время работы

3.5 Вывод

В данном разделе представлен систематизированный подход к проектированию и исследованию пользовательского интерфейса сайта. Анализировались психолого-социальные характеристики целевой аудитории, были определены потребности, страхи и барьеры пользователей. Проводились интервью и опросы, позволившие выявить основные предпочтения и сложности потенциальных владельцев питомцев.

Результатом проведенных исследований стала разработка User Story и Customer Journey Maps (CJM), что позволило детально изучить путь пользователя на сайте, начиная от открытия ресурса и заканчивая успешным процессом усыновления питомца. Был разработан сценарий типичного визита пользователя, иллюстрирующий практичность и удобство интерфейса.

Такой комплексный подход позволил визуально представить будущий продукт, учесть пожелания пользователей и минимизировать возможные недостатки интерфейса.

4 Программная реализация

4.1 Разработка алгоритмов

4.1.1 Алгоритм ориентации

Система ориентации представляет собой сложный программный модуль, который оценивает текущее пространственное положение объекта относительно некоторой глобальной или земной системы координат. В предложенном алгоритме используется концепция кватернионов, которые позволяют эффективно объединять данные от различных сенсоров, исключая проблему «гимбаллок», которая характерна для использования углов Эйлера [4]. Основная задача состоит в том, чтобы на каждом цикле вычислений обновлять состояние ориентации, получая данные от гироскопа и внося поправки с помощью данных акселерометра и магнитометра, тем самым устраняя накопившийся дрейф и минимизируя погрешности, возникающие из-за ограничений сенсоров [5].

Пусть кватернион со значениями угловой скорости будет $\omega = (0, \omega_x, \omega_y, \omega_z)$, где $\omega_x, \omega_y, \omega_z$ – компоненты угловой скорости, полученные от гироскопа [6]. Предположим, что уже имеется нормированный кватернион q , описывающий текущее положение тела в пространстве.

Тогда после поворота на приращение ω новый кватернион q_{new} определяется следующим выражением:

$$q_{new} = \frac{1}{2} * q \otimes \omega \quad (1)$$

При обновлении системы дискретно через шаг Δt рекуррентное уравнение, описывающее изменение ориентации, можно записать как:

$$q_t = q_{t-1} + \frac{1}{2} * q_{t-1} \otimes \omega * \Delta t \quad (2)$$

После каждого шага вычислений необходимо нормировать q_t , чтобы исключить накопление погрешностей и сохранить кватернион единичным [7].

Пусть a — нормированный вектор, отражающий направление, измеренное акселерометром, а $g = (0,0,1)$ — вектор гравитации в выбранной системе координат. Для начала требуется сделать проверку на корректность данных с акселерометра для избегания погрешностей. Пусть d – допустимое значение отклонения от величины тяжести. Если $\sqrt{a_x^2 + a_y^2 + a_z^2}$ выходит за диапазон $[1 - d; 1 + d]$, то оставляем текущий кватернион без изменений. После проверки вычисляется ось вращения:

$$axis = g \times a \quad (3)$$

Затем формируем кватернион q_a , где скалярная часть $w = 1 + g \cdot a$ (добавление единицы помогает обойтись без тригонометрических функций), а векторная часть совпадает со значениями $axis$. Поскольку акселерометр не даёт информацию о повороте вокруг вертикальной оси, данный компонент восстанавливают из уже имеющегося текущего кватерниона q . Для этого требуется перемножить q_a с «обрезанным» кватернионом вида $q'_c = (q.w, 0, 0, q.z)$.

Чтобы минимизировать дрейф, возникающий при интегрировании гироскопических данных, выполняется линейная интерполяция между «гироскопическим» кватернионом q_t и «акселерометрическим» кватернионом q_a . Пусть α — коэффициент в диапазоне $[0;1]$, характеризующий долю доверия к акселерометру. Тогда результирующий кватернион:

$$Q = (1 - \alpha) * q_g + \alpha * q_a \quad (4)$$

После этого кватернион Q нормируется, чтобы вновь получить единичный кватернион.

Для учёта азимута создаётся кватернион y_q , описывающий поворот вокруг оси Z на угол $shift$, соответствующий отклонению магнитного севера от истинного. Затем нормируется вектор магнитного поля и представляется в виде кватерниона $\{0, mx, my, mz\}$. С помощью «обратного поворота» определяется, как этот магнитный вектор соотносится с кватернионом Q .

После дополнительного умножения на y_q устраняется искомое смещение оси. С помощью полученного кватерниона Q' вычисляются вспомогательные параметры:

$$\gamma = Q'.x^2 + Q'.y^2 \quad (5)$$

$$\beta = \sqrt{\gamma + Q'.x * \gamma} \quad (6)$$

где γ – квадрат расстояния «проекции» вектора на плоскость XY, β – величина, помогающая скорректировать азимут. Затем формируется кватернион M_d , который является «чистым» поворотом вокруг оси Z:

$$M_d = \left\{ \frac{\beta}{\sqrt{2 * \gamma}}, 0, 0, \frac{Q'.y}{\sqrt{2 * \beta}} \right\} \quad (7)$$

Этот кватернион вращает исходный Q только в горизонтальной плоскости XY, «докручивая» систему к правильному азимуту. Затем применяется плавная подмешивающая операция, чтобы избежать резких скачков ориентации:

$$M'_d = \{1 - a + a * M_d.w, 0, 0, a * M_d.z\} \quad (8)$$

В конце умножаем Q на M'_d и нормируем итоговый результат.

Таким образом, полученный кватернион описывает актуальную ориентацию тела с учётом гироскопа, акселерометра и магнитометра, что позволяет компенсировать дрейф по всем осям, включая восстановление корректного азимута.

4.1.2 Алгоритм стабилизации

Стабилизация квадрокоптера опирается на пропорционально-дифференциальный регулятор (PD-контур) — простой, но надёжный метод подавления отклонений летательного аппарата от требуемого положения. Суть метода сводится к непрерывному слежению за ошибкой ориентации и высоты, вычислению мгновенного управляющего момента и распределению этого момента на силовые установки.

Квадрокоптер является динамически неустойчивой системой: любая, даже малая, ошибка по углу крена φ или тангажа θ вызывает рост угловой скорости и, в конечном счёте, ведёт к падению. Автопилот должен ловить эти отклонения быстрее, чем они успеют накопиться. PD-контур решает задачу двумя параллельными механизмами. Также регулирование используется для удержания дрона на одной высоте.

Пропорциональная часть создаёт момент, величина которого прямо пропорциональна текущему отклонению. Чем больше ошибка, тем сильнее корректирующий момент и тем быстрее аппарат возвращается в исходное положение.

$$errorPitch = desPitch - realPitch \quad (9)$$

$$errorRoll = desRoll - realRoll \quad (10)$$

$$errorHeight = fixHeight - realHeight \quad (11)$$

где $errorPitch$, $errorRoll$, $errorHeight$ – отклонение ожидаемых значений от реальных, полученных с помощью датчиков и алгоритмов.

Дифференциальная часть реагирует на скорость изменения ошибки. Если аппарат разгоняется к перерегулированию, производное звено вырабатывает момент противоположного знака и тормозит движение, тем самым убирая колебания и вибрацию.

$$dPitch = (errorPitch - prevErrorPitch)/dt \quad (12)$$

$$dRoll = (errorRoll - prevErrorRoll)/dt \quad (13)$$

$$dHeight = (errorHeight - prevErrorHeight)/dt \quad (14)$$

где $dPitch$, $dRoll$, $dHeight$ – это скорость изменения ошибки, $prevErrorPitch$, $prevErrorRoll$, $prevErrorHeight$ – ошибка в предыдущий момент времени, dt – шаг дискретизации между значениями.

Затем считается добавочное управляющее воздействие по каждому углу и высоте.

$$forcePitch = Kp * errorPitch + Kd * dPitch \quad (15)$$

$$forceRoll = Kp * errorRoll + Kd * dRoll \quad (16)$$

$$forceHeight = Kp * errorHeight + Kd * dHeight \quad (17)$$

где $forcePitch, forceRoll, forceHeight$ – это дополнительное воздействие, Kp, Kd – коэффициенты для регулирования. После этого воздействие по высоте добавляется к текущей тяге моторов pow . Затем для корректной стабилизации управляющие воздействия для каждого мотора квадрокоптера записывается в такой форме:

$$MotorUL = pow - forcePitch + forceRoll \quad (18)$$

$$MotorUR = pow - forcePitch - forceRoll \quad (19)$$

$$MotorDL = pow + forcePitch + forceRoll \quad (20)$$

$$MotorDR = pow + forcePitch - forceRoll \quad (21)$$

Набор коэффициентов получен эмпирически: сначала находился Kp до первого устойчивого колебательного режима, затем вводился Kd до исчезновения колебаний и требуемого быстродействия. Такое сочетание обеспечивает уверенный возврат к командному углу без перерегулирования, но не приводит к скачкообразным изменениям оборотов, что важно для ресурса моторов. В таблице 1 отображены используемые в программе коэффициенты.

Таблица 1 - Коэффициенты для PD-регулирования

коэффициенты	Крен	Тангаж	Высота
Kp	0.20	0.20	1.5
Kd	0.06	0.06	3.0

PD-контур создаёт упругое поле вокруг команды пилота: аппарат не просто возвращается к точке равновесия, а делает это максимально быстро и без лишних колебаний. Простота формулы позволяет вычислять управление на дешёвом микроконтроллере и оставляет запас времени на другие задачи, например INS-навигацию и телеметрию.

4.1.3 Алгоритм позиционирования

Алгоритм позиционирования отвечает за то, чтобы в любой момент времени оператор знал, где именно находится дрон в горизонтальной плоскости и на какой высоте он летит. Один лишь GPS решает эту задачу не всегда: спутниковая навигация обновляется медленно и легко теряет сигнал вблизи зданий или под кронами деревьев. Встроенная инерциальная система (IMU) наоборот даёт измерения каждую десятую долю миллисекунды, но без коррекции постепенно уплывает из-за накопления ошибок. Поэтому в проекте используется комплементарное объединение двух источников — каждое перекрывает слабые стороны другого.

Виртуальный IMU-блок симулятора возвращает шесть первичных величин: три угловых ускорения по каждой оси и три проекции силы гравитации по каждой оси. Программный фильтр сначала достраивает из угловых скоростей ориентацию корпуса в пространстве, чтобы понять, куда в глобальной системе направлен каждая ось корпуса. После этого к вектору линейных ускорений в корпусной системе применяется полученный кватернион ориентации, что обеспечивает линейные ускорения в земной системе координат.

Получив линейные ускорения, контроллер делает два последовательных численных интегрирования: первое превращает ускорения в мгновенную скорость, второе — скорость в прирост пройденного пути. Каждую итерацию новые показания IMU добавляются к накопленной скорости, а обновлённая скорость — к накопленному смещению. Так формируется прогноз INS - позиция, посчитанная только по инерции.

Из-за шум и калибровочных погрешностей каждое интегрирование вносит маленькую систематическую ошибку; со временем она растёт сначала линейно, а затем квадратично. Чтобы эта ошибка не становилась критичной, INS-траектория подкрепляется GPS-траекторией.

Как только приходит новый пакет данных от GPS, система сравнивает его координаты с расчётными INS. С помощью линейной интерполяции с

фиксированным малым коэффициентом инерциальная траектория подкрепляется GPS-данными. Малый коэффициент почти не искажает гладкую линию, нарисованную инерциальной интеграцией, но медленно стягивает её к истинной спутниковой траектории, обнуляя накопленный дрейф.

$$x_{fused} = k * x_{GPS} + (1 - k) * x_{INS} \quad (22)$$

$$y_{fused} = k * y_{GPS} + (1 - k) * y_{INS} \quad (23)$$

Где x_{fused}, y_{fused} – комплексированные данные позиционирования, x_{GPS}, y_{GPS} – данные позиционирования с ГНС, x_{INS}, y_{INS} – данные позиционирования с инерциальной системы, k – коэффициент интерполяции.

По высоте аналогичный алгоритм: интегрированное по вертикальной оси ускорение даёт инерциальную высоту, которая постепенно прижимается к дальномерной мере с тем же малым коэффициентом. Это сочетание позволяет гладко вести аппарат на высоте и не бояться кратковременных засветок дальномера.

Комплементарное позиционирование запущено постоянно, начиная с момента включения симулятора. Пока не получен первый валидный пакет GPS, на выходе оператор видит позиционирование по инерциальной системе. Как только приходит спутниковая точка, модель мгновенно синхронизируется и далее поддерживает баланс между плавностью и точностью.

Таким образом, описанный алгоритм выдаёт координаты с высокой частотой и приемлемой точностью, не требуя тяжёлых вычислений.

4.1.4 Алгоритм оценки высоты на основе барометра

В условиях отсутствия или деградации сигналов спутниковой навигации барометр фактически становится единственным источником абсолютной высоты. Кроме того, он способен компенсировать дрейф, возникающий при интегрировании ускорений в инерциальной навигационной системе. Акселерометры и гироскопы, обеспечивающие высокую краткосрочную

точность, склонны к накоплению систематической ошибки при длительном полёте. Барометрические измерения, напротив, дают независимую точку отсчёта, относительно которой можно корректировать накопленный дрейф. Тем не менее эти измерения нуждаются в цифровой постобработке, чтобы сгладить случайные колебания давления и обеспечить более устойчивые показатели высоты. Одним из эффективных методов сглаживания является скользящее взвешенное среднее во временном окне, где каждому отсчёту присваивается определённый вес в зависимости от его положения [8].

Пусть сигнал барометрической высоты во времени представлен дискретной последовательностью $h[n]$, где n — индекс дискретизации. Введём кольцевой буфер из N отсчётов, в котором накапливаются последние измерения. Обозначим веса через $w[i]$, где i изменяется от 1 до N таким образом, что более свежим значениям приписывается больший вес. Одна из самых распространённых схем — линейное нарастание: $w[i] = i$, то есть самое недавнее измерение получает максимальный вес $w[N] = N$, а самое старое — минимальный $w[1] = 1$. Тогда сглаженное значение высоты в момент времени n определяется как средневзвешенная сумма:

$$\overline{h}[n] = \frac{\sum_{k=1}^N w[k] * h[n - (N - k)]}{\sum_{k=1}^N w[k]} \quad (24)$$

где в числителе суммируются произведения каждого из последних N отсчётов на соответствующий вес, а в знаменателе стоит сумма всех весов. На рисунке 9 можно увидеть процесс сглаживания высоты, полученной с помощью барометра, где зеленая линия — это высота до сглаживания во время движения устройства по вертикальной оси, синяя линия — высота после сглаживания.

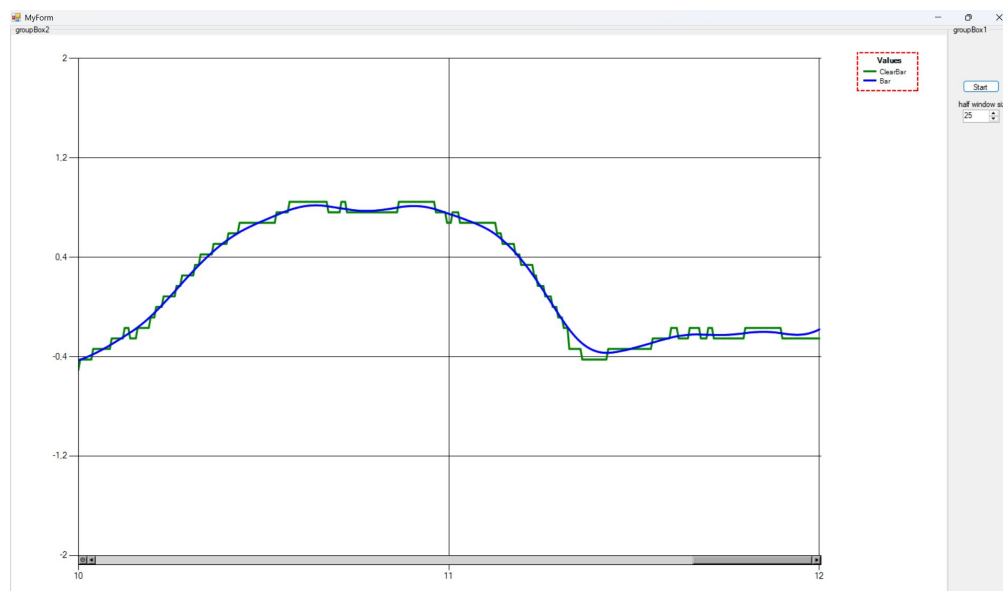


Рисунок 9 - График высоты, полученной по барометрическому методу

В заключение следует отметить, что барометрические измерения, при условии корректной фильтрации и периодической калибровки, способны обеспечить надёжную оценку высоты БПЛА даже в условиях отсутствия спутниковой навигации.

4.2 Разработка веб-части

4.2.1 Серверная часть Node.js

Серверный узел выполняет роль центрального концентратора данных: именно через него вся телеметрия, которую вычисляет клиентское приложение на языке C++, попадает в веб-браузеры операторов или в сторонние инструменты анализа. Для этой задачи выбрана платформа Node.js [9], потому что её событийная модель естественно отражает характер входящего трафика — десятки тысяч коротких сообщений в секунду, каждое из которых нужно быстро принять, распаковать и тут же разослать подписчикам.

Внутри узла работают два логических сервиса. Бинарный WebSocket-приёмник открывает порт 8080 и ждёт сообщений фиксированной длины 68 байт. Как только пакет приходит, его размер проверяется на соответствие, после чего выполняется прямое чтение 17 чисел формата

IEEE-754. Отсутствие сторонних декодеров экономит время на копирование и снижает нагрузку.

Далее сформированный объект передаётся во вторую ветку — канал Socket.IO на 3000 порту. Socket.IO поверх HTTP/1.1 гарантирует, что данные достигнут клиента. Поскольку внутри процесса объект передаётся по ссылке, дополнительных аллокаций не возникает.

С точки зрения развертывания сервер минималистичен: достаточно выполнить `npm install` для установки двух зависимостей `ws` и `socket.io`. Тот же JavaScript-стек, что и у клиента, упрощает поддержку. Логи событий выводятся в консоль. Код данного приложения находится в Приложении Е.

4.2.2 Клиентская часть Vue.js

Web-клиент предназначен для оперативного контроля полёта: он отображает ориентацию, ускорения, высоту, а также INS- и GNSS-позицию в практически реальном времени. Фреймворк Vue 3 здесь выбран из-за следующего преимущества: его система Composition API даёт декларативную реактивность: любое изменение наблюдаемого свойства моментально приводит к перерисовке нужных частей DOM, при этом сами компоненты остаются чистыми функциями без перегрузок жизненным циклом [10].

Структурно интерфейс делится на два базовых компонента:

1) `TelemetryChart` — универсальный модуль линейного графика. Каждый экземпляр получает ссылку на массив данных, цветовую палитру и подписи кривых. Внутренний буфер хранит до 400 точек, что даёт восемь секунд истории при максимальной частоте 50 Гц. Пользовательский функционал включает: паузу/продолжение потока, мгновенную очистку графика, индивидуальное скрывание линий, ручной ввод пределов по Y и ползунков частоты обновления.

2) `PositionPlot` — плоская координатная система для сравнения GNSS-и INS-траекторий. Пользователь может менять границы осей, включать/выключать сетку, а также отображать по отдельности сырые и рассчитанные точки. Частота обновления задаётся собственным ползунком,

чтобы тяжёлая отрисовка множества SVG-примитивов не мешала другим графикам.

В качестве рендер-движка выбран D3 v7: эта библиотека позволяет вручную контролировать каждый элемент SVG.

С точки зрения UX интерфейс закрывает сразу несколько задач. Инженер-разработчик может:

- видеть мгновенную реакцию PD-контуров на управляющие команды;
- отслеживать рассогласование позиционирования и при необходимости менять коэффициент смешивания;
- отслеживать высоту одновременно по трём разным значениям и вычислять доверительный диапазон.

Преимущество веб-подхода в том, что точка доступа к телеметрии глобальна. Это особенно полезно на испытательном полигоне, где ноутбук инженера, планшет преподавателя и смартфон ассистента могут одновременно следить за полётом, не мешая друг другу и не дублируя соединения с дроном.

4.3 Взаимодействие подсистем и итоговая архитектура

Система состоит из трёх уровней, которые соединены потоками данных, но не зависят друг от друга по коду.

На нижнем уровне работает симулятор, написанный на C# WinForms [11]. Он решает уравнения движения квадрокоптера, формируя структуру данных с датчиков. В пакет входят показания акселерометра, гироскопа, gps, дальнометра и барометра, а также место для управляющих сигналов моторам. Пакет отправляется по UART соединению. Код симулятора находится в приложении Г.

Промежуточный уровень — приложение на C++ WinForms [12]. Оно принимает данные по UART от симулятора, выделяет отдельные поля и запускает три группы алгоритмов: фильтр ориентации, PD-стабилизацию и позиционирование. После вычислений формируется вторая структура

telemetry_native, уже содержащая дополнительно углы крена, тангажа и рысканья, инерциальную и спутниковую позиции, а также высоты из трёх источников. Затем эта структура по WebSocket отправляется в Node.js-приложение. В симулятор по UART-соединению отправляются управляющие сигналы воздействия на моторы для замыкания контура. Код клиентской части находится в приложении Д.

Сетевой уровень реализован на Node.js. Модуль WebSocket получает 68-байтовые сообщения, декодирует их в объект JavaScript, затем через Socket.IO передаёт объект во все подключённые браузеры. Передача происходит по ссылке, поэтому не создаётся лишних копий и задержка от C++ до JavaScript-объекта мала.

Верхний уровень — веб-клиент на Vue 3. Он открывает соединение WebSocket с помощью Socket.IO, получает событие telemetry, кладёт данные во Vue-reactive-состояние, а компоненты автоматически перерисовываются. Благодаря тому, что графики хранят всего 400 точек, объём передаваемых данных остаётся стабильным, а память браузера не растёт.

Связь между уровнями организована только через публичные протоколы WebSocket и UART. Каждую подсистему можно перезапускать без остальных, что облегчает отладку. На рисунке 10 можно увидеть общую структуру системы.



Рисунок 10 - Архитектура системы

Существует и аппаратный экземпляр дрона, построенный на контроллере STM32F4 и тем же наборов датчиков. Его архитектура показана на рисунке 11. На реальном борту будут протестированы те же алгоритмы, что и в симуляторе.

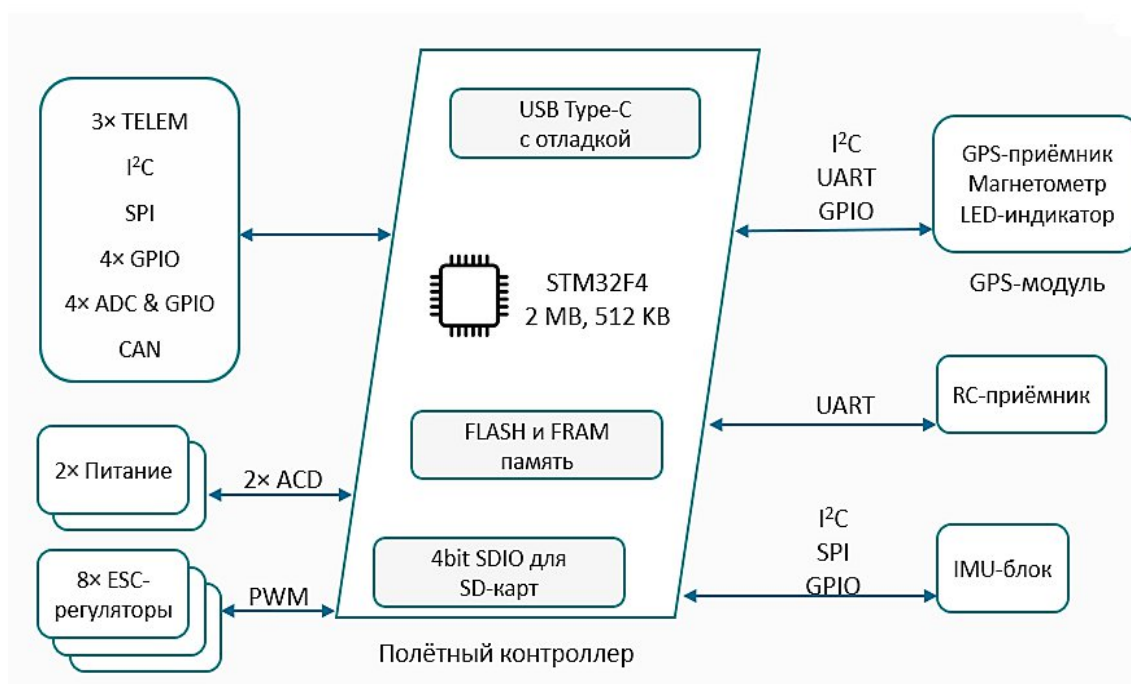


Рисунок 11 - Архитектура аппаратно-программной системы

5 Тестирование ПО

Данный раздел посвящен детальному изложению особенностей программной реализации системы в соответствии с заранее спроектированной архитектурой и выбранными технологиями.

5.1 План тестирования

5.1.1 Цели тестирования

Поставлены следующие цели:

- алгоритмы ориентации, стабилизации и позиционирования достигают заявленных погрешностей;
- ПО корректно обменивается данными по UART и WebSocket, хранит и отображает телеметрию;
- аппаратная часть работает без отказов с программным комплексом;
- проверка всего ПО на соответствие требованиям.

5.1.2 Типы и методики испытаний

При тестировании использовались следующие типы:

- Алгоритмические
- Интеграционные
- Функциональные
- Аппаратные

5.1.3 Критерии начала/окончания

- Старт: доступны рабочие сборки симулятора, клиента и веб-узла; настроена стендовая конфигурация.
- Завершение: 100% прохождение критических тестов, исправление всех блокирующих дефектов.

5.2 Тестирование

5.2.1 Алгоритмические тесты

5.2.1.1 Алгоритм ориентации

Требования: подаётся рапида + 90 °/с, измеряется средняя ошибка между эталонным углом и расчётом фильтра; допуск $\leq 0,8^\circ$.

Принцип тестирования: происходит приращение требуемого угла на заданную дискретизацию линейно, так получаем идеальный угол. Алгоритм получает такое же значения в виде угловой скорости поворачивая кватернион, из которого вычисляется статический угол после каждого изменения. На рисунках 12-16 можно увидеть график работы с разными значениями.



Рисунок 12 - Расчёт угла поворота на 90 градусов с дискретизацией 5 градусов



Рисунок 13 - Расчёт угла поворота на 90 градусов с дискретизацией 18 градусов



Рисунок 14 - Расчёт угла поворота на 90 градусов с дискретизацией 30 градусов

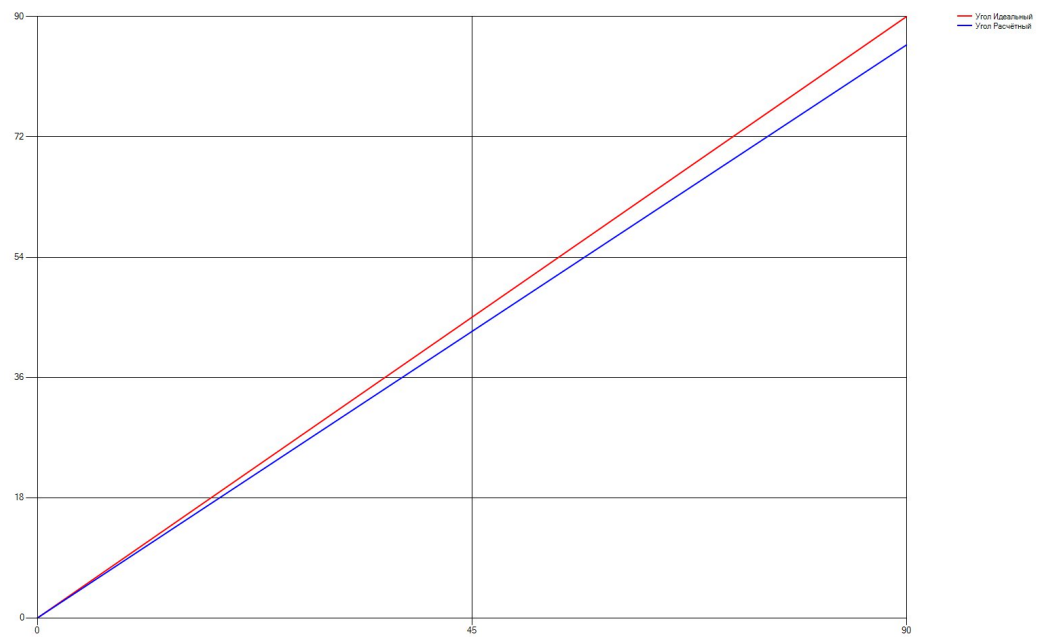


Рисунок 15 - Расчёт угла поворота на 90 градусов с дискретизацией 45 градусов

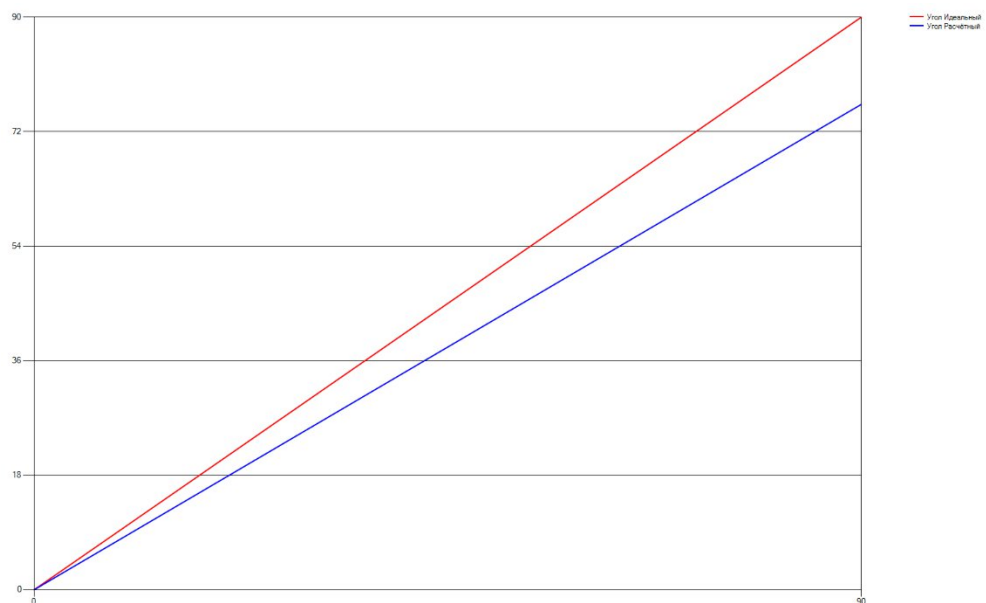


Рисунок 16 - Расчёт угла поворота на 90 градусов с дискретизацией 90 градусов

Вывод: данный алгоритм теряет точность при угловой скорости 1000 градусов в секунду с частотой дискретизации 200 герц. Это соответствует примерно вращению 166 оборотов в минуту, что фактически недостижимо для дронов в режиме полёта «stabilize».

5.2.1.2 Алгоритм стабилизации

Тест №1

Требования: ввод ступеньки -15° по крену, фиксируется время установления ≤ 5 с и перерегулирование $\leq 40\%$.

Принцип тестирования: дрон отклоняется с помощью джойстика на -10 градусов по крену, затем возвращается сразу, после этого фиксируется колебательные движения на графике, который можно посмотреть на рисунке 17.

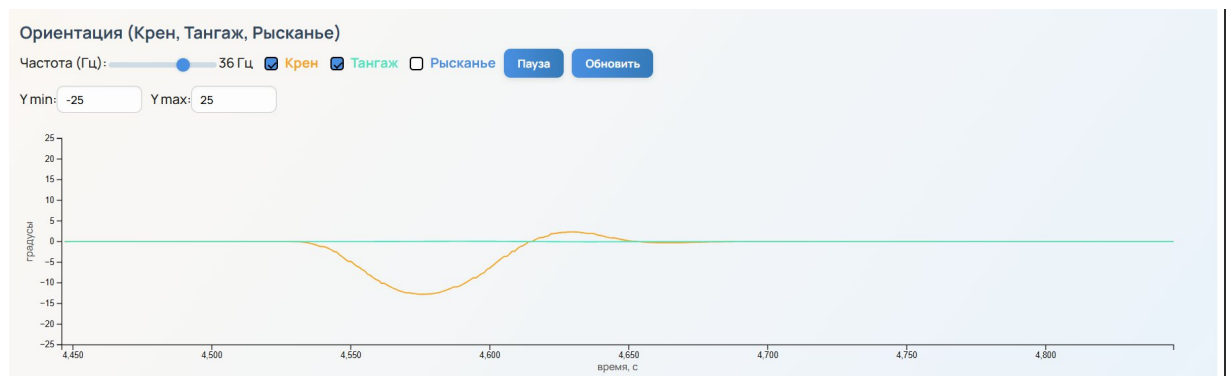


Рисунок 17 - Тестирование стабилизации по крену

Вывод: время установления составляет примерно 1 секунда перерегулирование составляет около 10% (максимальная амплитуда – 3 градуса), колебания затухающие, тест удовлетворяет заявленным требованиям.

Тест №2

Требования: ввод ступеньки $+10^\circ$ по тангажу, фиксируется время установления ≤ 5 с и перерегулирование $\leq 40\%$.

Принцип тестирования: дрон отклоняется с помощью джойстика на -12 градусов по тангажу, затем возвращается сразу, после этого фиксируется колебательные движения на графике, который можно посмотреть на рисунке 18.

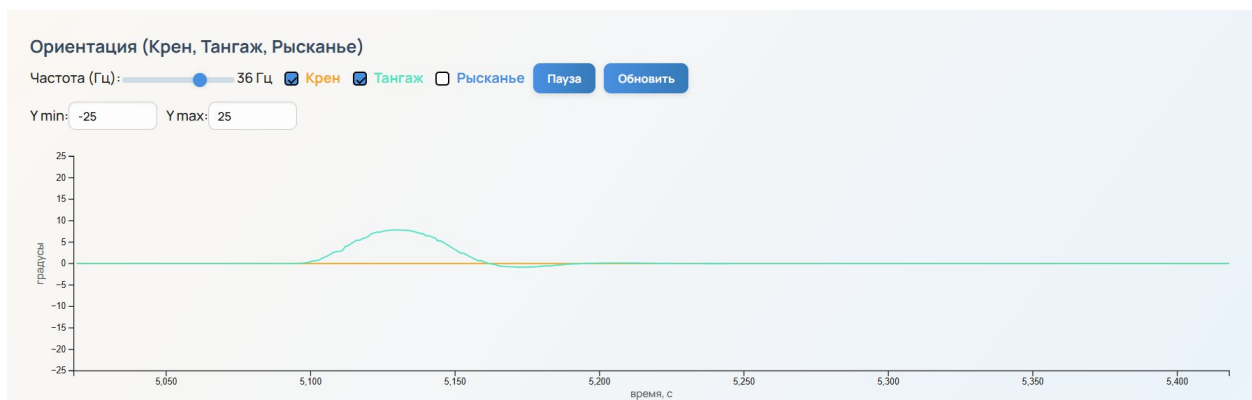


Рисунок 18 - Тестирование стабилизации по тангажу

Вывод: время установления составляет примерно одна секунда, перерегулирование составляет около 3%, колебания затухающие, тест удовлетворяет заявленным требованиям.

Тест №3

Требования: фиксированная высота 1.4м, фиксируется время установления ≤ 1 с и перерегулирование ≤ 10 %.

Принцип тестирования: дрон с помощью джойстика поднимается по вертикали, затем на высоте 1.4м вызывается команда фиксации высоты. Результат работы можно увидеть на рисунке 19.

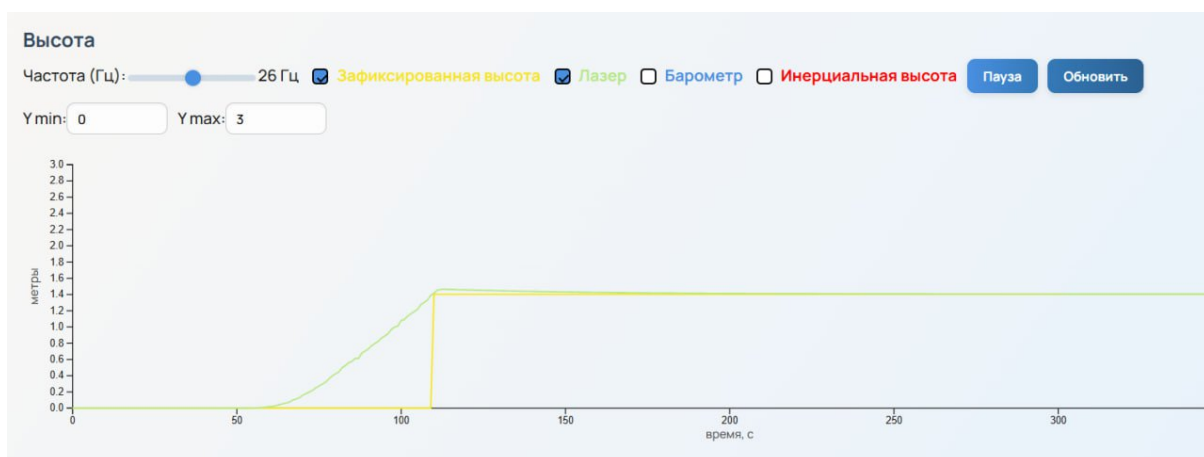


Рисунок 19 - Тестирование стабилизации по высоте

Вывод: время установления составляет примерно 0.2 секунды (50 отсчетов – 1 секунда), перерегулирование составляет около 7%, колебания затухающие, тест удовлетворяет заявленным требованиям.

5.2.1.3 Алгоритм позиционирования

Тест №1

Требования: Максимальное допустимое отклонение расчётной позиции от измеренной по ГНС в состоянии покоя— не более 0.2 м.

Принцип тестирования: дрон на зафиксированной высоте с помощью пульта должен подлететь на точку (-4;-2) и зависнуть там. Результат работы можно увидеть на рисунке 20.

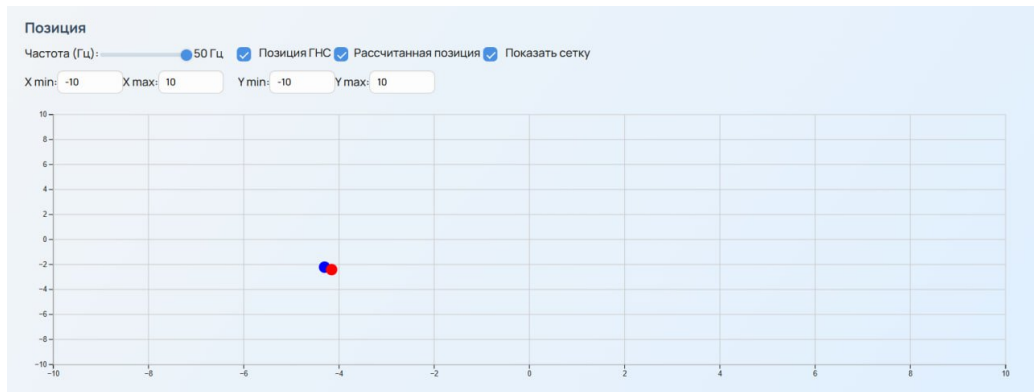


Рисунок 20 - Тестирование позиционирования в покое

Вывод: Расстояние между точкой по ГНС и точкой по расчетной позиции составляет около 0.1м в состоянии покоя, что удовлетворяет заданным требованиям.

Тест №2

Требования: Максимальное допустимое отклонение расчётной позиции от измеренной по ГНС в состоянии движения— не более 0.4 м.

Принцип тестирования: дрон на зафиксированной высоте с помощью пульта должен подлететь на точку (-10;10), в полете фиксируется расхождение. Результат работы можно увидеть на рисунке 21.

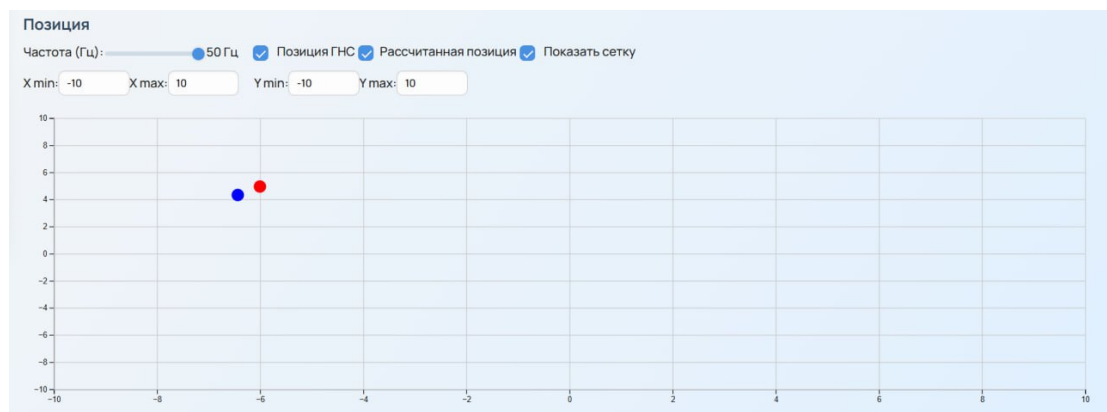


Рисунок 21 - Тестирование позиционирования в движении

Вывод: Расстояние между точкой по ГНС и точкой по расчетной позиции составляет около 0.3м в состоянии движения, что удовлетворяет заданным требованиям.

5.2.1.4 Алгоритм высоты

Тест №1

Требования: Максимальное допустимое отклонение расчётной барометрической высоты от лазерной— не более 0.5 м.

Принцип тестирования: дрон с помощью пульта плавно поднимается на высоту 2.5м, а затем плавно опускается. Результат работы можно увидеть на рисунке 22.

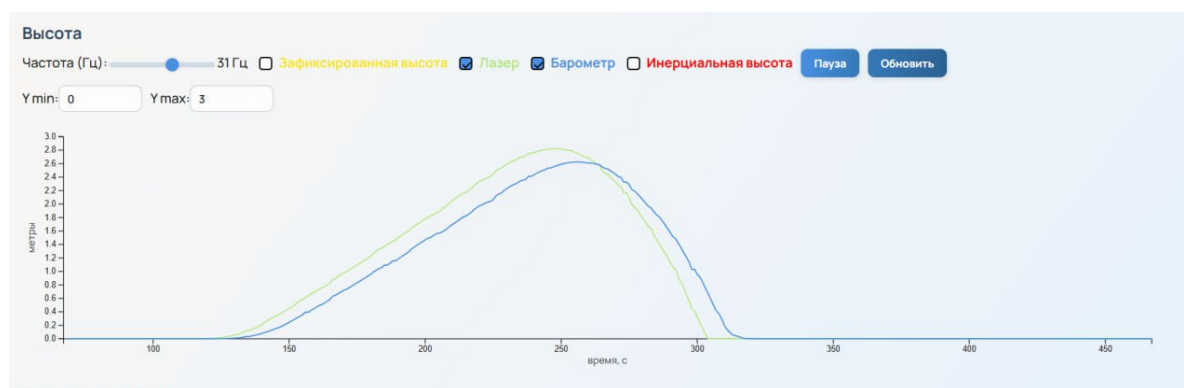


Рисунок 22 - Тестирование высоты по барометру

Вывод: максимальная разность между барометрической высотой и лазерной составляет 0.3м, что удовлетворяет заданным требованиям.

Тест №2

Требования: Максимальное допустимое отклонение расчётной инерциальной высоты от лазерной— не более 0.5 м.

Принцип тестирования: дрон с помощью пульта плавно поднимается на высоту 2.8м, а затем плавно опускается. Результат работы можно увидеть на рисунке 23.

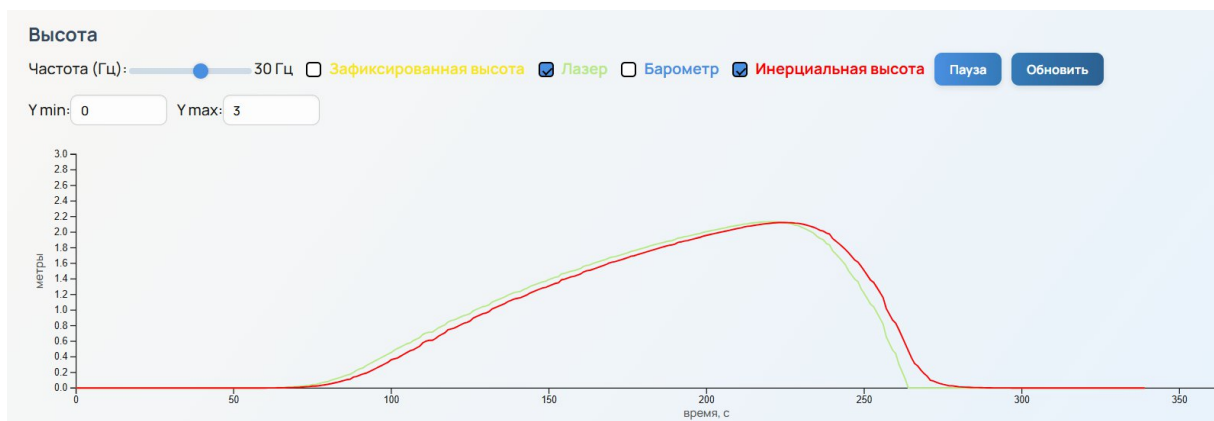


Рисунок 23 - Тестирование инерциальной высоты

Вывод: максимальная разность между расчетной высотой и лазерной составляет 0.4м, что удовлетворяет заданным требованиям.

5.2.2 Функциональные тесты

В таблице 2 отображены тест-кейсы, посвященные функциональному тестированию.

Таблица 2 - Функциональные тест-кейсы

№	Приоритет	Предусловие	Шаги воспроизведения	Ожидаемый результат	Реакция системы	Статус
1	Высокий	Страница «Телеметрия» загружена, частота по умолчанию 10 Гц	1) Перетащить ползунок частоты до «25 Гц» 2) Засечь интервал прихода точек в консоли	Интервал между событиями \approx 40 мс (25 Гц)	Среднее $\Delta t = 39,8$ мс, график плавно ускорился	Пройден
2	Высокий	Любой график выводит 3 линии	1) Снять чек-бокс «Крен» 2) Наблюдать SVG-DOM	Полигон с id rollPath исчез, другие линии остались	Удалён элемент <code><path id="rollPath"></code> , остальные	Пройден

					остались	
3	Высокий	Идёт поток данных 10 Гц	1) Нажать «Пауза» → подождать 5 с 2) Нажать «Продолжить»	В процессе паузы X-координата точек не меняется; после продолжения график догоняет без пропусков	Во время паузы буфер не рос, после резюме график продолжил с правильного времени	Пройден
4	Высокий	Буфер графика заполнен (>300 точек)	1) Нажать «Обновить»	История очищается, шкала X сбрасывается в 0; новые точки рисуются с начала	<code>buffer.length = 0</code> , <code>xOffset = 0</code> , на экране пустая сетка; новые данные пошли с 0 с	Пройден
5	Средний	График на паузе	1) Перетащить ползунок частоты с 10 до 5 Гц 2) Нажать «Продолжить»	График возобновляет работу с новым периодом ≈ 200 мс	Среднее $\Delta t = 199,6$ мс, расхождений нет	Пройден
6	Низкий	В графике «Ускорения и ИНС» все линии включены	1) Быстро снять все 6 чек-боксов 2) Включить	На экране остаётся одна линия	Отрисован только <code>accXPath</code>	Пройден

			ТОЛЬКО «Асс X»			
--	--	--	-------------------	--	--	--

5.2.3 Аппаратные тесты

Тест №1. Напряжение всех выводов платы полетного контроллера.

К плате питания был подключен источник питания с напряжением 16В. Далее к ней прикреплялся полетный контроллер. Затем с помощью мультиметра, на котором граничное значение было 20В, измерялось напряжения всех выходов на интерфейсы: I2C1, I2C2, UART1, UART2, UART3, LPUART1, SWD.

На каждом выходе должно быть напряжение 5В, на I2C1 напряжение 3.3В.

Допустимая погрешность на выходе 3.3В – 1%.

Допустимая погрешность на выходе 5В – 5%.

На рисунке 24 проиллюстрировано произведение тестирования напряжения интерфейса I2C2. Мультиметром было зафиксировано значение напряжения 4.96В, что соответствует заявленной характеристике с учетом допустимой погрешности.

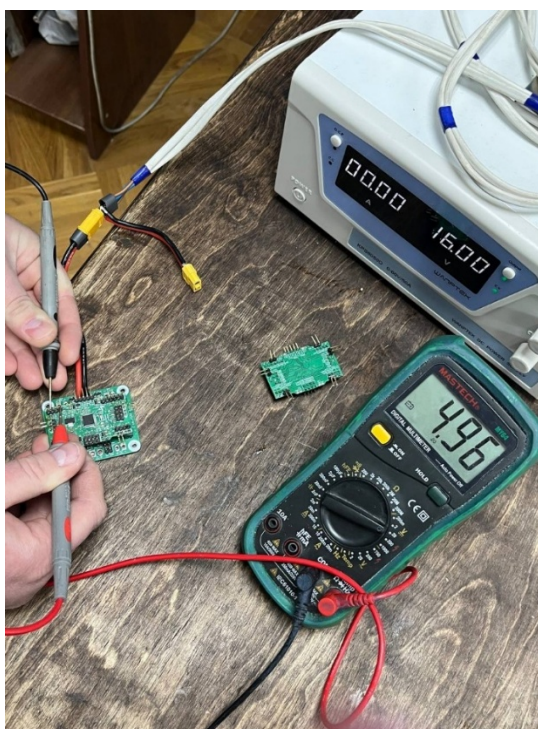


Рисунок 24 - Напряжение вывода на I2C2

На рисунке 25 проиллюстрировано производство тестирования напряжения интерфейса UART3. Мультиметром было зафиксировано значение напряжения 4.96 В, что соответствует заявленной характеристике с учетом допустимой погрешности.

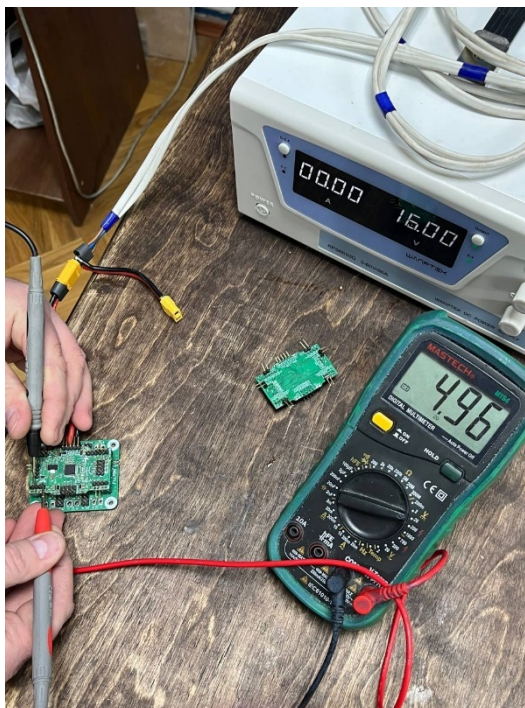


Рисунок 25 - Напряжение вывода на UART3

На рисунке 26 проиллюстрировано тестирование напряжения интерфейса LPUART1. Мультиметром было зафиксировано значение напряжения 4.93В, что соответствует заявленной характеристике с учетом допустимой погрешности.

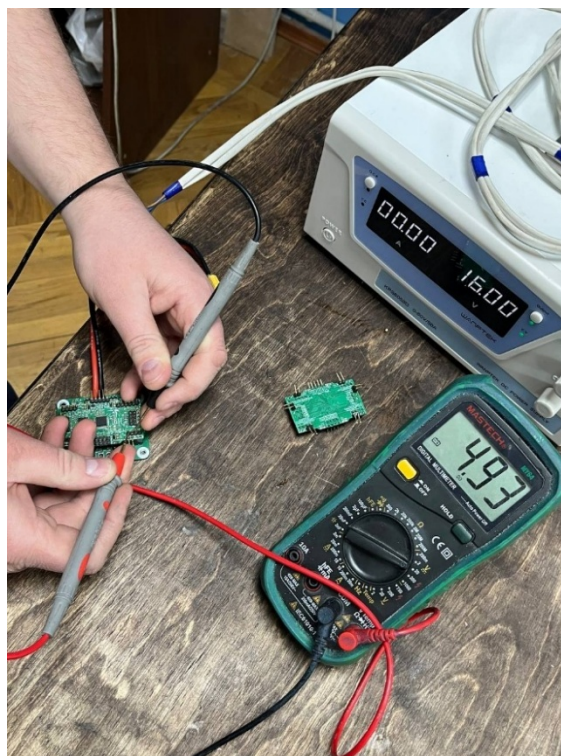


Рисунок 26 - Напряжение вывода на LPUART1

На рисунке 27 проиллюстрировано произведение тестирования напряжения интерфейса I2C1. Мультиметром было зафиксировано значение напряжения 3.29В, что соответствует заявленной характеристике с учетом допустимой погрешности.

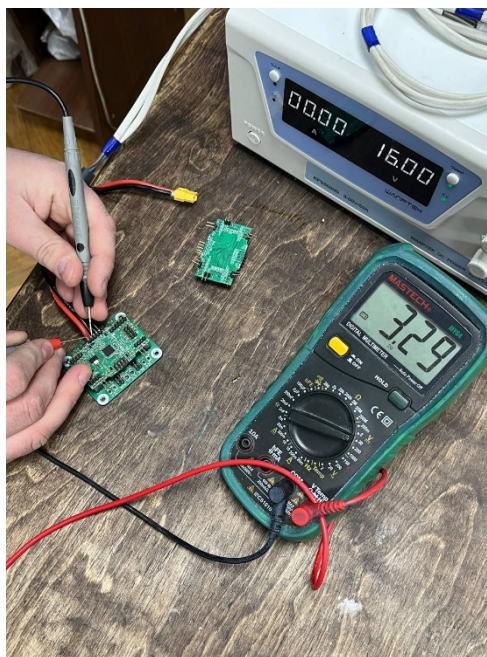


Рисунок 27 - Напряжение вывода на I2C1

На рисунке 28 проиллюстрировано тестирование напряжения интерфейса SWD. Мультиметром было зафиксировано значение напряжения 4.95В, что соответствует заявленной характеристике с учетом допустимой погрешности.

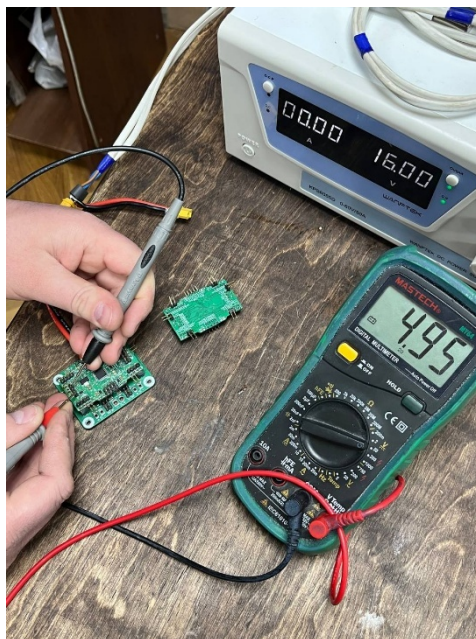


Рисунок 28 - Напряжение вывода на SWD

На рисунке 29 проиллюстрировано проведение тестирования напряжения интерфейса UART1. Мультиметром было зафиксировано значение напряжения 4.96В, что соответствует заявленной характеристике с учетом допустимой погрешности.

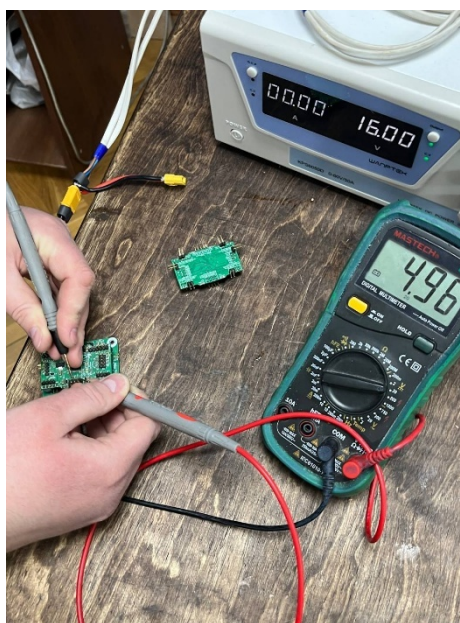


Рисунок 29 - Напряжение вывода на UART1

На рисунке 30 проиллюстрировано произведение тестирования напряжения интерфейса UART2. Мультиметром было зафиксировано значение напряжения 4.96В, что соответствует заявленной характеристике с учетом допустимой погрешности.

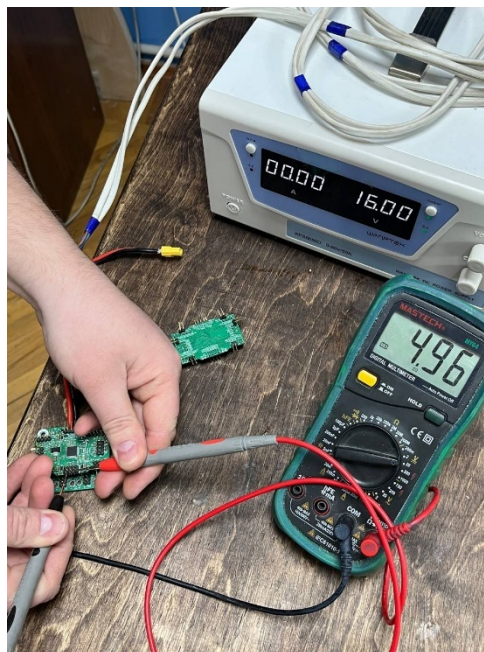


Рисунок 30 - Напряжение вывода на UART2

5.2.4 Натурные тесты

Для проведения натурального тестирования наклонов по крену и тангажу полученный кватернион, описывающий ориентацию тела в пространстве, переводился в значения, подобные акселерометру – косинус угла относительно вектора гравитации - и сравнивался с показаниями реального акселерометра. Так как коэффициент при слиянии данных с гироскопа и акселерометра больше доверяет гироскопу, то проверка является допустимой. Для тестирования корректности по азимуту можно из текущего кватерниона получать угол Эйлера в диапазоне от 0 до 2π и сравнить с показаниями реального компаса.

Для проверки работоспособности системы плата IMU была прикреплена к роторному гироскопу, чтобы контролировать повороты по всем осям, как показано на рисунке 31 [13].

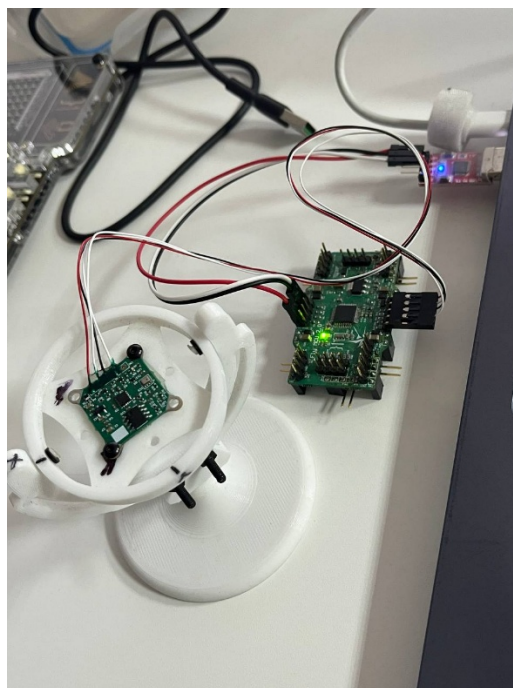


Рисунок 31 - Устройство для тестирования

На рисунке 32 продемонстрирован результат работы алгоритма с показаниями акселерометра в пределах $[-1000; 1000]$. Как можно заметить, то данные, полученные с кватерниона, имеют минимальную погрешность с данными, полученными с акселерометра, при этом отсутствуют колебания.

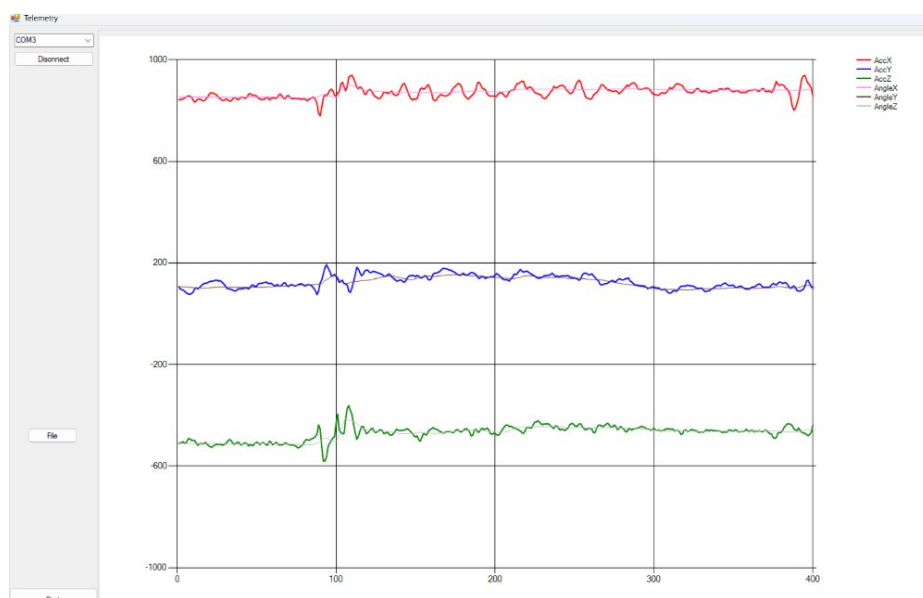


Рисунок 32 - График с результатами работы алгоритма в значениях, подобных акселерометру

Для проверки корректности поворота по азимуту отдельно брался кватернион, полученный только с помощью магнитометра, который используется в подмешивании, и кватернион, полученный от смешивания

гироскопа и акселерометра. Затем устройство поворачивали по направлению на север для корректного результата. По результатам работы можно увидеть корректный угол поворота вокруг вертикальной оси, что отображено на рисунке 33. Значение в нуле показывает, что устройство повернуто на север, и в ходе тестирования его поворачивали против часовой стрелки. На рисунке серой линией показан угол, полученный от чистого магнитометра, и угол, полученный по акселерометру и гироскопу.

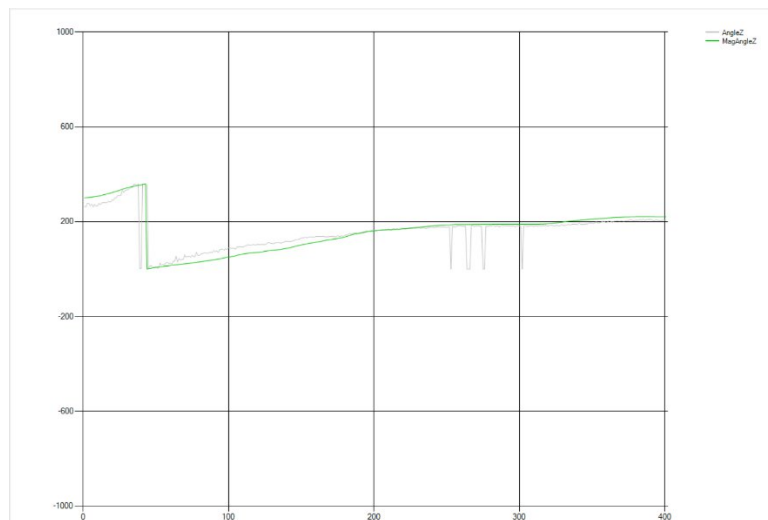


Рисунок 33 - График с углами Эйлера, полученных с результата работы алгоритма

Инерциальная система — это система, которая отслеживает движение объекта, его положение и скорость, используя данные от инерциальных сенсоров. Инерциальная навигация используется в широком спектре приложений, где критично точно отслеживать движение, ориентацию и положение объекта. Преимущество инерциальной навигации заключается в её автономности и независимости от внешних сигналов, что делает её незаменимой в ситуациях, где другие методы навигации не работают или могут дать сбой. Для получения инерциальной системы текущий кватернион ориентации поворачивается относительно вектора значений, полученных с акселерометра. Благодаря ей можно получать и контролировать позицию дрона и его скорость. При интегрировании значений мы получаем скорость по каждой оси, а при вторичном интегрировании пройденное расстояние за заданный период.

Для тестирования корректности значений инерциальной системы устройство было перевернуто так, чтобы ось Z была направлена в противоположную сторону от вектора гравитации. На рисунке 34 отображены результаты инерциальной системы при колеблющемся движении по оси Y . Можно заметить, что несмотря на текущую ориентацию, система показывает движение только относительно той оси, по которой она движется.



Рисунок 34 - График значений инерциальной системы во время движения по оси Y

Аналогичное испытание было проведено для движения по оси X , что можно увидеть на рисунке 35.

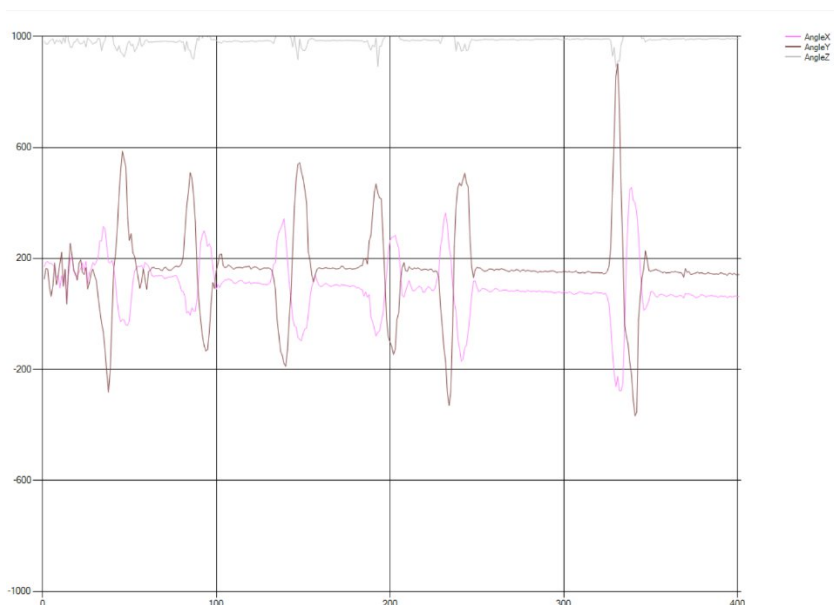


Рисунок 35 - График значений инерциальной системы во время движения по оси X

5.3 Вывод

Проведённое тестирование подтверждает полное соответствие продукта установленным требованиям по функциональности, удобству использования и производительности. Рекомендуется к улучшению система стабилизации.

Продукт готов к внедрению и дальнейшей эксплуатации, обеспечивая надёжность, удобство и высокие показатели производительности. Интеграция рекомендованных мер позволит сохранить стабильность и повысить качество обслуживания даже при дальнейшем росте числа пользователей.

6 Технико–экономическое обоснование

6.1 Обзор рынка программного обеспечения и аналогов

В рамках анализа рынка навигационных решений для беспилотных летательных аппаратов (БПЛА) были рассмотрены существующие аппаратно-программные комплексы, реализующие функции ориентации, стабилизации и позиционирования. В частности, изучены следующие решения: Pixhawk (ArduPilot), DJI и сопутствующее программное обеспечение, включая QGroundControl и Mission Planner. Указанные системы демонстрируют высокую точность и функциональность, но в большинстве случаев требуют сложной настройки, специфического аппаратного обеспечения и значительных затрат.

Предложенная в рамках ВКР система отличается более простой архитектурой, гибкостью адаптации и открытостью к интеграции. Она предназначена для симуляционной и стендовой отладки навигационного ПО. Преимущества по сравнению с аналогами:

- полностью программная реализация без необходимости в платах;
- возможность работы в симуляторе с реалистичной моделью движений;
- визуализация параметров в веб-интерфейсе без установки специализированного ПО;
- гибкая настройка частоты, фильтрации и формата передачи данных;

6.2 Оценка экономических затрат на разработку проекта

Оценка затрат выполнена с учётом реального объема работ, выполненных студентом в течение производственной и преддипломной практик, дипломной подготовки и самостоятельной разработки. В таблице 3 работы разбиты по ролям, отражающим стадии проектирования и реализации навигационного блока.

Таблица 3 - Распределение ролей и зарплат

Роль	Объём, ч	Ставка, Р/ч	Всего, Р
Разработчик алгоритмов	100	700	70000
Разработчик симулятора и тестов	80	650	52000
Инженер по телеметрии и визуализации	60	600	36000
Тестирующий	40	500	20000
Итого	—	—	178000

Дополнительно учитываются налоги и страховые отчисления 30 %, куда входят пенсионный фонд 22 %, Медицинское страхование 5.1%, Социальное страхование 2.9 %: 53 400 Р.

Прочие расходы:

- электропитание, аренда стенда, интернет, износ ПК: 3 000 Р.

Общие затраты на проект: $178\,000 + 53\,400 + 3\,000 = 234\,400$ Р.

6.3 Модель применения и окупаемости

Разработка ориентирована на образовательный и исследовательский сегмент: кафедры, учебные лаборатории, НИОКР-центры. Возможные сценарии распространения:

- open-source модель с размещением на GitHub и лицензией, при этом предлагается платная техподдержка или курсы по внедрению;
- интеграция в учебный курс в рамках вузовских программ;
- комплектация стендов с адаптацией системы под конкретные платы.

6.4 Вывод

Предложенное решение обладает значительным потенциалом с точки зрения снижения затрат на начальное развертывание систем навигации в учебных проектах, возможностью тиражирования и гибкой адаптации под аппаратную часть. В отличие от коммерческих навигационных блоков БПЛА,

данная система минимизирует затраты за счёт программной реализации и может быть быстро перенесена как на симулятор, так и на реальный полётный контроллер.

ЗАКЛЮЧЕНИЕ

Цель данной выпускной квалификационной работы заключалась в разработке программного комплекса, обеспечивающего обработку, визуализацию и проверку алгоритмов навигации для беспилотного летательного аппарата.

Для достижения поставленной цели были решены следующие задачи:

- выполнен анализ предметной области и существующих навигационных решений;
- сформированы функциональные и нефункциональные требования к системе;
- реализован и отлажен симулятор полёта с имитацией работы сенсоров;
- разработан модуль клиента на C++, включающий в себя алгоритмы ориентации, стабилизации и позиционирования;
- создано веб-приложение на платформе на Node.js с использованием фронтенд-интерфейса на Vue.js с визуализацией телеметрии и элементами управления;

Результаты тестирования показали, что реализованный комплекс стабильно работает при частоте телеметрии до 50 Гц, точность ИНС и стабилизации соответствует требованиям, установленным в техническом задании.

Простота архитектуры и доступность технологий делают систему привлекательной для образовательных учреждений, лабораторий и исследовательских команд, работающих в области БПЛА.

Разработанные алгоритмы были использованы в реальном проекте, что отражено в акте о внедрении, отображенном в приложении А. Также по тематике диплома были написаны две научные статьи для конференций, дипломы отображены в приложении Б, и еще одна статья про алгоритм ориентации [14]. В приложении В находится сертификат, подтверждающий регистрацию программного кода симулятора на ЭВМ.

Таким образом, все поставленные задачи были решены в полном объёме, цель достигнута. Разработка имеет практическую значимость и может быть использована как основа для дальнейшего расширения функциональности, в том числе для внедрения в реальные летательные аппараты. Перспективы дальнейшего развития включают адаптацию системы к работе с аппаратными контроллерами, добавление поддержки новых сенсоров, реализацию логирования и удалённого доступа, а также интеграцию с системами автоматического управления.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Ardupilot. Документация. Официальный сайт. [Электронный ресурс] // URL: <https://ardupilot.org/ardupilot/index.html>
- 2 PX4. Документация. Официальный сайт. [Электронный ресурс] // URL: <https://docs.px4.io/main/en/>
- 3 STM32F407. Документация. Официальный сайт. [Электронный ресурс] // URL <https://www.st.com/en/microcontrollers-microprocessors/stm32f407-417.html>
- 4 Буданов А. С., Егунов В. А. Использование углов Эйлера в инерциальных навигационных системах //Инженерный вестник Дона. – 2021. – №. 7 (79). – С. 120-127.
- 5 Suryanto W. et al. IMU (inertial measurement unit) device for internet of things based disaster early warning system: applications and innovation // IOP Conference Series: Earth and Environmental Science. – IOP Publishing, 2020. – Vol. 451, No. 1. – P. 012015
- 6 Безверхний Н. В., Грибов А. Ф., Хорькова Н. Г. Алгебра кватернионов и геометрия трёхмерного пространства //Modern European Researches. – 2021. – №. 3. – С. 25-30.
- 7 Пономарев И. С. ИССЛЕДОВАНИЕ АЛГОРИТМА ОРИЕНТАЦИИ БЕСКАРДАННОЙ СИСТЕМЫ ОРИЕНТАЦИИ, ПОСТРОЕННОГО НА КВАТЕРНИОНАХ, НА ВЛИЯНИЕ ПОГРЕШНОСТЕЙ ОТ ДРЕЙФА ГИРОСКОПА //Фундаментальные и прикладные научные исследования в современном мире. – 2023. – С. 186-194.
- 8 БЕСТУГИН А. Р. и др. ПОВЫШЕНИЕ ТОЧНОСТИ ИЗМЕРЕНИЯ ВЫСОТЫ ПОЛЕТА КВАДРОКОПТЕРА ПРИ ИСПОЛЬЗОВАНИИ БАРОМЕТРИЧЕСКОГО МЕТОДА //ЕСТЕСТВЕННЫЕ И ТЕХНИЧЕСКИЕ НАУКИ Учредители: ООО" Издательство" Спутник+". – №. 1. – С. 186-192.
- 9 NodeJS. Документация. Официальный сайт. [Электронный ресурс] // URL: <https://nodejs.org/api/all.html>

- 10 VueJS. Документация. Официальный сайт. [Электронный ресурс] // URL: <https://vuejs.org/>
- 11 C#. Документация. Официальный сайт. [Электронный ресурс] // URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/>
- 12 C++. Документация. Официальный сайт. [Электронный ресурс] // URL: <https://en.cppreference.com/w/cpp/language/reference>
- 13 ICM-20948. Документация. Официальный сайт. [Электронный ресурс] // URL: <https://evelta.com/content/datasheets/004-ICM20948.pdf/>
- 14 Маркова Д. К. Девятиосевая система ориентации в пространстве для повышения стабильности позиционирования // Всероссийская научная конференция «СИСТЕМНЫЙ СИНТЕЗ И ПРИКЛАДНАЯ СИНЕРГЕТИКА». – 2024. – №. 12 (1). – С. 357-359.

ПРИЛОЖЕНИЕ А. АКТ О ВНЕДРЕНИИ



ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ «СИГНАЛ-БИТ»
347900, Ростовская обл., г. Таганрог, ул. Лесная Биржа, д. 20-б, оф. 2
e-mail: sigbit@yandex.ru; <https://sigbit.ru/>
ОГРН 1236100028493, ИНН/КПП 6154166039 / 615401001



«УТВЕРЖДАЮ»

Генеральный директор
ООО «Сигнал-БИТ»

Е.С. Басан

«28» апреля 2025 г.

АКТ

о внедрении результатов выпускной квалификационной работы

Настоящим актом подтверждаю, что разработанный студенткой группы КТб04-7 кафедры математического обеспечения и применения ЭВМ Института компьютерных технологий и информационной безопасности Южного федерального университета Марковой Даной Константиновной модуль инерциальной навигации БПЛА с использованием вспомогательных навигационных данных принят для опытной эксплуатации в ООО «СИГНАЛ-БИТ» и используется при создании учебных БПЛА.

Генеральный директор
ООО «СИГНАЛ-БИТ»

Басан Е.С.

ПРИЛОЖЕНИЕ Б. ИТОГИ УЧАСТИЯ В КОНФЕРЕНЦИЯХ



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Низневартровский государственный университет»



XXVII Всероссийская студенческая
научно-практическая конференция
Низневартовского государственного университета

ДИПЛОМ I СТЕПЕНИ

награждается

Маркова Дана Константиновна

Южный федеральный университет

Доклад «Алгоритм для системы ориентации в
пространстве на основе кватернионов»

Первый проректор,
проректор по экономике и развитию



Б.Н. Махутов



Низневартовск, 09-10 апреля 2025 год

ДИПЛОМ

НАГРАЖДАЕТСЯ

Маркова Дана Константиновна

Занявшая I место в конкурсе докладов в секции
«Беспилотные автоматизированные системы»
XI Всероссийской научно-технической
конференции «Фундаментальные и прикладные
аспекты компьютерных технологий и
информационной безопасности»

Руководитель:

заведующий лабораторией кафедры СиПУ им.
профессора А.А. Колесникова ИКТИБ
Скляр Сергей Анатольевич

Директор



Г. Е. Веселов

г. Таганрог
14-20 Апреля 2025 г.

рег. № 05.15.01-56/25

ПРИЛОЖЕНИЕ В. СВИДЕТЕЛЬСТВО О РЕГИСТРАЦИИ ПРОГРАММЫ

РОССИЙСКАЯ ФЕДЕРАЦИЯ



СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2025662010

Программное обеспечение для симуляции полёта дрона

Правообладатель: **Общество с ограниченной
ответственностью "СИГНАЛ-БИТ" (RU)**

Авторы: **Маркова Дана Константиновна (RU), Скларов
Сергей Анатольевич (RU), Елькин Дмитрий Максимович
(RU), Басан Елена Сергеевна (RU)**



Заявка № **2025660866**

Дата поступления **02 мая 2025 г.**

Дата государственной регистрации

в Реестре программ для ЭВМ **16 мая 2025 г.**

*Руководитель Федеральной службы
по интеллектуальной собственности*

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат: 0692e761a0b300b154f2401670b0a2026
Владелец: **Зубов Юрий Сергеевич**
Действителен с 10.11.2024 по 03.10.2025

Ю.С. Зубов

ПРИЛОЖЕНИЕ Г. КОД СИМУЛЯТОРА

```
Drone.cs
using System.Numerics;
using System.Runtime.InteropServices;

namespace DroneSimulator
{
    internal class Drone
    {
        public int ID;
        public float Mass; // Масса
        public bool Active; // Живой?
        public float Length; // Длина лучей
        public const float Dynamic = 10; // Динамика вращения
        public Vector3 PosXYZ, SpdXYZ, AccXYZ; // Положение в
        пространстве: Позиция, Скорость, Ускорение
        public Quaternion Quat; // Основной кватернион
        public float Power = 0; // Тяга всех двигателей (0-1)
        public float MaxPower; // Максимальная Тяга всех двигателей (КГ)
        public Vector3 SpdPRY, AccPRY; // Поворот в пространстве: pitch roll
        yaw

        public Vector3 Acc, Gyr; // Имитация: Акселерометр, Гироскоп
        public float LaserRange; // Имитация: Дальномер

        public Vector4 Orientation;

        private uint DataTimer;

        private const float Gravity = 9.8f;
```

```

private const float TO_GRAD = 180 / MathF.PI;
private const float TO_RAD = MathF.PI / 180;

private Thread DroneThread;
private int Timer;

private Vector2 MoveOF = Vector2.Zero;

RealMode.Accelerometer RealAcc = new RealMode.Accelerometer();
RealMode.Gyroscope RealGyr = new RealMode.Gyroscope();
RealMode.Position RealPos = new RealMode.Position();
RealMode.Barometer RealBar = new RealMode.Barometer();
RealMode.Range RealRange = new RealMode.Range();
RealMode.OpticalFlow RealOF = new RealMode.OpticalFlow();

public static byte[] getBytes(object data)
{
    int size = Marshal.SizeOf(data);
    byte[] arr = new byte[size];

    IntPtr ptr = IntPtr.Zero;
    try
    {
        ptr = Marshal.AllocHGlobal(size);
        Marshal.StructureToPtr(data, ptr, true);
        Marshal.Copy(ptr, arr, 0, size);
    }
    finally
    {
        Marshal.FreeHGlobal(ptr);
    }
}

```

```

    }
    return arr;
}

public static object fromBytes(byte[] arr, Type type)
{
    object mem = new object();

    int size = Marshal.SizeOf(type);
    IntPtr ptr = IntPtr.Zero;
    try
    {
        ptr = Marshal.AllocHGlobal(size);

        Marshal.Copy(arr, 0, ptr, size);

        mem = Marshal.PtrToStructure(ptr, type);
    }
    finally
    {
        Marshal.FreeHGlobal(ptr);
    }

    return mem;
}

public VisualData.VisualDrone GetVisual(int count, int index)
{
    VisualData.VisualDrone drone = new VisualData.VisualDrone();

    drone.Head.Size = Marshal.SizeOf(typeof(VisualData.VisualDrone));

```

```

drone.Head.Type = VisualData.VisualHead.VisualType.Drone;

drone.Count = count;
drone.Index = index;

drone.ID = ID;
drone.Color.A = 255; drone.Color.R = 255; drone.Color.G = 0;
drone.Color.B = 0;

drone.Rotate.X = Quat.X; drone.Rotate.Y = Quat.Y; drone.Rotate.Z =
Quat.Z; drone.Rotate.W = Quat.W;
drone.Position.X = PosXYZ.X; drone.Position.Y = PosXYZ.Y;
drone.Position.Z = PosXYZ.Z;
drone.Scale = 1.0f;

drone.State = VisualData.VisualDrone.DroneState.Active;

drone.Power = Power;

return drone;
}

private void ThreadFunction()
{
while (DroneThread != null)
{
Action(Environment.TickCount);
Thread.Sleep(1);
}
}
}

```

```

public Drone(int id)
{
    ID = id;
    Active = false;
    PosXYZ = Vector3.Zero;
    SpdXYZ = Vector3.Zero;
    AccXYZ = Vector3.Zero;
    Quat = Quaternion.Identity;

    DroneThread = new Thread(new ThreadStart(ThreadFunction));
    Timer = Environment.TickCount;
    DroneThread.Start();
}

```

```

public int Create(float power, float mass, float len)
{
    Mass = mass;
    Length = len;
    MaxPower = power;

    Active = true;

    return ID;
}

```

```

public void Close()
{
    DroneThread = null;
}

```

```

private float GetAngle(float a1, float a2, float az)

```

```

{
    if (a2 == 0.0f && az == 0.0f)
    {
        if (a1 > 0) return 90;
        if (a1 < 0) return -90;
        return 0;
    }
    return MathF.Atan(a1 / MathF.Sqrt(a2 * a2 + az * az)) * TO_GRAD;
}

```

```

public void Rotate(float x, float y, float z)

```

```

{
    x = x * MathF.PI / 180;
    y = y * MathF.PI / 180;
    z = -z * MathF.PI / 180;

```

```

    Quaternion map = Quat;

```

```

    Quaternion spd = new Quaternion(x, y, z, 0);

```

```

    Quaternion aq = spd * map;

```

```

    map.W -= 0.5f * aq.W;

```

```

    map.X -= 0.5f * aq.X;

```

```

    map.Y -= 0.5f * aq.Y;

```

```

    map.Z -= 0.5f * aq.Z;

```

```

    Quat = Quaternion.Normalize(map);

```

```

}

```

```

public Vector4 GetOrientation()

```

```

{

```

```

    Quaternion grav = new Quaternion(0, 0, 1, 0);

```



```

grav = Quat * grav * Quaternion.Inverse(Quat);

float yaw = 2 * MathF.Atan2(Quat.Z, Quat.W) * TO_GRAD;
if (yaw < 0.0f) yaw = 360.0f + yaw;

return new Vector4(GetAngle(grav.Y, grav.X, grav.Z), GetAngle(-grav.X,
grav.Y, grav.Z), yaw, grav.Z);
}

public void Action(int tick)
{
    float time = (tick - Timer) / 1000.0f;
    Timer = tick;

    if (!Active) return;

    float flow = Power;

    if (PosXYZ.Z < 0.3f)
    {
        flow += flow * 0.1f; // Воздушная подушка
    }

    SpdPRY += AccPRY * (Dynamic * time / (Mass * Length));

    float wind_x = 0, wind_y = 0, wind_z = 0;

    if (Area.Wind.Enable)
    {

```

```
Quaternion dir = Quaternion.CreateFromAxisAngle(new Vector3(0, 0, 1), Area.Wind.Direction * TO_RAD * 2);
```

```
//Quaternion win = new Quaternion(0, 1, 0, 0) * dir;
```

```
Quaternion spd = new Quaternion(0, Area.Wind.Speed.From, 0, 0) * dir;
```

```
float spd_x = spd.X - SpdXYZ.X;
```

```
float spd_y = spd.Y - SpdXYZ.Y;
```

```
float spd_z = spd.Z - SpdXYZ.Z;
```

```
wind_x = 0.5f * Area.Wind.Density * Area.Wind.Resist * (spd_x * MathF.Abs(spd_x));
```

```
wind_y = 0.5f * Area.Wind.Density * Area.Wind.Resist * (spd_y * MathF.Abs(spd_y));
```

```
wind_z = 0.5f * Area.Wind.Density * Area.Wind.Resist * (spd_z * MathF.Abs(spd_z));
```

```
}
```

```
Quaternion pow = Quaternion.Inverse(Quat) * new Quaternion(0, 0, flow, 0) * Quat;
```

```
AccXYZ = new Vector3(pow.X + wind_x, pow.Y + wind_y, pow.Z + wind_z) * (Gravity / Mass);
```

```
SpdXYZ += (AccXYZ + new Vector3(0, 0, -Gravity)) * time;
```

```
PosXYZ += SpdXYZ * time;
```

```
AccXYZ /= Gravity; // Вернуть измерения в G
```

```
if (Area.Poosition.Freeze.X) { SpdXYZ.X = 0; PosXYZ.X = 0; }
```

```
if (Area.Poosition.Freeze.Y) { SpdXYZ.Y = 0; PosXYZ.Y = 0; }
```

```
if (Area.Poosition.Freeze.Z) { SpdXYZ.Z = 0; PosXYZ.Z = 5; }
```

```

if (PosXYZ.Z < 0)
{
    SpdPRY = Vector3.Zero;
    SpdXYZ.X = 0;
    SpdXYZ.Y = 0;
    Quat = Quaternion.Identity;
}
else Rotate(SpdPRY.X * time, SpdPRY.Y * time, SpdPRY.Z * time);

```

```

Vector4 ori = GetOrientation();

```

```

Orientation = ori;

```

```

if (PosXYZ.Z < 0)

```

```

{
    PosXYZ.Z = 0;

```

```

/*if (SpdXYZ.Z < -5)

```

```

{
    Active = false; // Сильно ударился о землю
}*/

```

```

/*if (MathF.Abs(ori.X) > 20 || MathF.Abs(ori.Y) > 20)

```

```

{
    Active = false; // Повредил винты при посадке
}*/

```

```

SpdXYZ.Z = 0;

```

```

Acc = new Vector3(0, 0, 1);

```

```

    Gyr = Vector3.Zero;
    LaserRange = 0;
}
else
{
    if (ori.W < 0)
    {
        //Active = false; // Перевернулся вверх ногами
    }

    Quaternion grav = Quat * new Quaternion(AccXYZ.X, AccXYZ.Y,
AccXYZ.Z, 0) * Quaternion.Inverse(Quat);
    Acc = new Vector3(grav.X, grav.Y, grav.Z);

    Gyr = SpdPRY;

    float tilt = MathF.Sqrt((ori.X * ori.X) + (ori.Y * ori.Y)) * TO_RAD;

    if (tilt < 90 && ori.W > 0) LaserRange = PosXYZ.Z / MathF.Cos(tilt);
    else LaserRange = float.MaxValue;
}

MoveOF.X += SpdXYZ.X - Gyr.Y;
MoveOF.Y += SpdXYZ.Y + Gyr.X;

RealAcc.Update(Acc, (uint)tick);
RealGyr.Update(Gyr, (uint)tick);
RealRange.Update(LaserRange, (uint)tick);
RealBar.Update(PosXYZ.Z * 11, (uint)tick);
RealPos.Update(PosXYZ, (uint)tick);
RealOF.Update(MoveOF, LaserRange, (uint)tick);

```

```

    DataTimer = (uint)tick;
}

private float Range(float pow)
{
    if (pow > 1) pow = 1;
    if (pow < 0) pow = 0;

    return pow * MaxPower;
}

public void SetQadroPow(float ul, float ur, float dl, float dr)
{
    ul = Range(ul); ur = Range(ur); dl = Range(dl); dr = Range(dr);

    Power = (ul + ur + dl + dr) / 4;

    AccPRY.Y = ((ul + dl) - (ur + dr));
    AccPRY.X = ((ul + ur) - (dl + dr));
    AccPRY.Z = ((ul + dr) - (dl + ur)) / 4;
}

private void RecvDataMotor4(byte[] data)
{
    DroneData.DataMotor4 mot = (DroneData.DataMotor4)fromBytes(data,
typeof(DroneData.DataMotor4));

    /* обработка */
    SetQadroPow(mot.UL, mot.UR, mot.DL, mot.DR);
}

```

```

private byte[] SendDataAcc()
{
    DroneData.DataAcc acc = new DroneData.DataAcc();

    acc.Head.Size = Marshal.SizeOf(typeof(DroneData.DataAcc));
    acc.Head.Mode = DroneData.DataMode.Response;
    acc.Head.Type = DroneData.DataType.DataAcc;
    acc.Head.Time = (uint)Environment.TickCount;

    acc.Acc.X = RealAcc.result.X; acc.Acc.Y = RealAcc.result.Y; acc.Acc.Z
= RealAcc.result.Z;
    acc.Time = RealAcc.timer;

    return getBytes(acc);
}

private byte[] SendDataGyr()
{
    DroneData.DataGyr gyr = new DroneData.DataGyr();

    gyr.Head.Size = Marshal.SizeOf(typeof(DroneData.DataGyr));
    gyr.Head.Mode = DroneData.DataMode.Response;
    gyr.Head.Type = DroneData.DataType.DataGyr;
    gyr.Head.Time = (uint)Environment.TickCount;

    gyr.Gyr.X = RealGyr.result.X; gyr.Gyr.Y = RealGyr.result.Y; gyr.Gyr.Z
= RealGyr.result.Z;
    gyr.Time = RealGyr.timer;

    return getBytes(gyr);
}

```

```

private byte[] SendDataMag()
{
    DroneData.DataMag mag = new DroneData.DataMag();

    mag.Head.Size = Marshal.SizeOf(typeof(DroneData.DataMag));
    mag.Head.Mode = DroneData.DataMode.Response;
    mag.Head.Type = DroneData.DataType.DataMag;
    mag.Head.Time = (uint)Environment.TickCount;

    mag.Mag.X = 0; mag.Mag.Y = 0; mag.Mag.Z = 0;
    mag.Time = DataTimer;

    return getBytes(mag);
}

private byte[] SendDataRange()
{
    DroneData.DataRange range = new DroneData.DataRange();

    range.Head.Size = Marshal.SizeOf(typeof(DroneData.DataRange));
    range.Head.Mode = DroneData.DataMode.Response;
    range.Head.Type = DroneData.DataType.DataRange;
    range.Head.Time = (uint)Environment.TickCount;

    range.LiDAR = RealRange.result;
    range.Time = RealRange.timer;

    return getBytes(range);
}

```

```

private byte[] SendDataLocal()
{
    DroneData.DataLocal local = new DroneData.DataLocal();

    local.Head.Size = Marshal.SizeOf(typeof(DroneData.DataLocal));
    local.Head.Mode = DroneData.DataMode.Response;
    local.Head.Type = DroneData.DataType.DataLocal;
    local.Head.Time = (uint)Environment.TickCount;

    local.Local.X = RealPos.result.X; local.Local.Y = RealPos.result.Y;
local.Local.Z = RealPos.result.Z;
    local.Time = RealPos.timer;

    return getBytes(local);
}

private byte[] SendDataBarometer()
{
    DroneData.DataBar bar = new DroneData.DataBar();

    bar.Head.Size = Marshal.SizeOf(typeof(DroneData.DataBar));
    bar.Head.Mode = DroneData.DataMode.Response;
    bar.Head.Type = DroneData.DataType.DataBar;
    bar.Head.Time = (uint)Environment.TickCount;

    bar.Pressure = RealBar.result;
    bar.Time = RealBar.timer;

    return getBytes(bar);
}

```



```

private byte[] SendDataOF()
{
    DroneData.DataOF of = new DroneData.DataOF();

    of.Head.Size = Marshal.SizeOf(typeof(DroneData.DataOF));
    of.Head.Mode = DroneData.DataMode.Response;
    of.Head.Type = DroneData.DataType.DataOF;
    of.Head.Time = (uint)Environment.TickCount;

    of.X = RealOF.result.X;
    of.Y = RealOF.result.Y;
    of.Time = RealBar.timer;

    MoveOF = Vector2.Zero;

    return getBytes(of);
}

private byte[] SendDataQuaternion()
{
    DroneData.DataQuat quat = new DroneData.DataQuat();

    quat.Head.Size = Marshal.SizeOf(typeof(DroneData.DataQuat));
    quat.Head.Mode = DroneData.DataMode.Response;
    quat.Head.Type = DroneData.DataType.DataQuat;
    quat.Head.Time = (uint)Environment.TickCount;

    quat.X = Quat.X; quat.Y = Quat.Y; quat.Z = Quat.Z; quat.W = Quat.W;

    return getBytes(quat);
}

```

```

private byte[]? ServerRequestResponse(DroneData.DataHead head, byte[]
body)
{
    byte[] zero = Array.Empty<byte>();

    switch (head.Type)
    {
        case DroneData.DataType.DataAcc:
        {
            if (head.Mode == DroneData.DataMode.Request)
            {
                // Запрос данных
                return SendDataAcc();
            }
            else
            {
                // Пришли данные
                // ... //
                //
                return zero;
            }
        }

        case DroneData.DataType.DataGyr: if (head.Mode ==
DroneData.DataMode.Request) return SendDataGyr(); else return zero;

        case DroneData.DataType.DataMag: if (head.Mode ==
DroneData.DataMode.Request) return SendDataMag(); else return zero;

```

```

        case DroneData.DataType.DataRange: if (head.Mode ==
DroneData.DataMode.Request) return SendDataRange(); else return zero;

        case DroneData.DataType.DataLocal: if (head.Mode ==
DroneData.DataMode.Request) return SendDataLocal(); else return zero;

        case DroneData.DataType.DataBar: if (head.Mode ==
DroneData.DataMode.Request) return SendDataBarometer(); else return zero;

        case DroneData.DataType.DataOF: if (head.Mode ==
DroneData.DataMode.Request) return SendDataOF(); else return zero;

        case DroneData.DataType.DataQuat: if (head.Mode ==
DroneData.DataMode.Request) return SendDataQuaternion(); else return zero;

        case DroneData.DataType.DataMotor4: if (head.Mode ==
DroneData.DataMode.Response) RecvDataMotor4(body); return zero;
    }

    return zero;
}

private const int DroneStreamCount = 512;
private byte[] DroneStreamData = new byte[DroneStreamCount];
private int DroneStreamIndex = 0;
private DroneData.DataHead DroneStreamHead = new
DroneData.DataHead() { Mode = DroneData.DataMode.None, Size = 0, Type =
DroneData.DataType.None };

public List<byte[]?> DataStream(byte[]? data, int size)
{

```

```

List<byte[]?> ret = new List<byte[]?>();

if (data == null) return ret; // Последовательность не сформирована,
ждать данных

if (size + DroneStreamIndex > DroneStreamCount) return null; // Ошибка
переполнения, поток сломан (конец)

Array.Copy(data, 0, DroneStreamData, DroneStreamIndex, size);
DroneStreamIndex += size;

while (true)
{
    if (DroneStreamHead.Size == 0) // Заголовок ещё не получен
    {
        if (DroneStreamIndex < DroneData.DataHead.StrLen) return ret; //
Нечего слать
        DroneStreamHead =
(DroneData.DataHead)fromBytes(DroneStreamData,
typeof(DroneData.DataHead));
    }

    if (DroneStreamHead.Size > DroneStreamIndex) break; // Пакет ещё не
полный

    byte[] body = new byte[DroneStreamHead.Size];

    Array.Copy(DroneStreamData, 0, body, 0, DroneStreamHead.Size);

    int shift = DroneStreamHead.Size;

```

```

        DroneStreamIndex -= shift;
        Array.Copy(DroneStreamData, shift, DroneStreamData, 0,
DroneStreamIndex); // Сдвигаем массив влево, убрав использованные данные

```

```

        DroneStreamHead.Size = 0; // Отменяем прошлый заголовок

```

```

        ret.Add(ServerRequestResponse(DroneStreamHead, body));
    }

```

```

        return ret;
    }
}

```

DroneData.cs

```

using System.Runtime.InteropServices;

```

```

namespace DroneData

```

```

{
    public enum DataMode : ushort
    {
        None = 0, Response = 1, Request = 2
    };

```

```

    public enum DataType : ushort
    {
        None = 0, Head = 1,

```

```

        // Output

```

```

        DataAcc = 1001, DataGyr = 1002, DataMag = 1003, DataRange = 1004,
        DataLocal = 1005, DataBar = 1006, DataOF = 1007,

```

```

// Input
DataMotor4 = 2001, DataMotor6 = 2002,

// State
DataQuat = 3001,
};

public struct DataHead
{
    public int Size;

    public DataMode Mode;
    public DataType Type;

    public uint Time; // Общее время

    static public int StrLen = Marshal.SizeOf(typeof(DroneData.DataHead));
}

public struct XYZ { public float X, Y, Z; }

public struct DataAcc
{
    public DataHead Head;
    public XYZ Acc;

    public uint Time; // Последнее время изменения данных

    static public int StrLen = Marshal.SizeOf(typeof(DroneData.DataAcc));
}

```

```

public struct DataGyr
{
    public DataHead Head;
    public XYZ Gyr;

    public uint Time; // Последнее время изменения данных

    static public int StrLen = Marshal.SizeOf(typeof(DroneData.DataGyr));
}

public struct DataMag
{
    public DataHead Head;
    public XYZ Mag;

    public uint Time; // Последнее время изменения данных

    static public int StrLen = Marshal.SizeOf(typeof(DroneData.DataMag));
}

public struct DataRange
{
    public DataHead Head;
    public float LiDAR; // Датчик посадки

    public uint Time; // Последнее время изменения данных

    static public int StrLen = Marshal.SizeOf(typeof(DroneData.DataRange));
}

public struct DataLocal

```

```

{
    public DataHead Head;
    public XYZ Local; // Локальные координаты

    public uint Time; // Последнее время изменения данных

    static public int StrLen = Marshal.SizeOf(typeof(DroneData.DataLocal));
}

```

```

public struct DataBar
{
    public DataHead Head;
    public float Pressure; // Давление

    public uint Time; // Последнее время изменения данных

    static public int StrLen = Marshal.SizeOf(typeof(DroneData.DataBar));
}

```

```

public struct DataOF
{
    public DataHead Head;
    public float X, Y; // Угловой сдвиг

    public uint Time; // Последнее время изменения данных

    static public int StrLen = Marshal.SizeOf(typeof(DroneData.DataOF));
}

```

```

public struct DataMotor4
{

```



```

    public DataHead Head;
    public float UL, UR, DL, DR;

    static public int StrLen = Marshal.SizeOf(typeof(DroneData.DataMotor4));
}

public struct DataMotor6
{
    public DataHead Head;
    public float UL, UR, LL, RR, DL, DR;

    static public int StrLen = Marshal.SizeOf(typeof(DroneData.DataMotor6));
}

public struct DataQuat
{
    public DataHead Head;
    public float X, Y, Z, W;

    static public int StrLen = Marshal.SizeOf(typeof(DroneData.DataQuat));
}
}

```

Program.cs

```

namespace DroneSimulator
{
    internal static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>

```

```

[STAThread]
static void Main()
{
    // To customize application configuration such as set high DPI settings
or default font,
    // see https://aka.ms/applicationconfiguration.
    ApplicationConfiguration.Initialize();
    Application.Run(new Form_Main());
}
}
}

```

Area.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;

namespace DroneSimulator
{
    internal class Area
    {
        public struct Poisition
        {
            public struct Freeze{ public static bool X, Y, Z; }
        }

        public struct Wind
        {

```

```

        public static bool Enable;
        public struct Speed { public static float From, To; }
        public static float Direction;
        public static float Density;
        public static float Resist;
    }
}
}

```

NetServerClient.cs

```

using System.Net.Sockets;
using System.Net;
using System.Drawing;

namespace DroneSimulator
{
    internal class NetServerClients
    {
        public class ConnectData
        {
            public int ID;
            public bool Connect;
            public int Count;

            public Socket? Client;
        }

        public class ReceiveData
        {
            public int ID;
            public byte[] Buffer;

```

```

    public int Size;

    public Socket? Client;
}

private class ClientData
{
    public int ID;
    public Socket? workSocket = null;
    public const int count = 1024;
    public byte[] buffer = new byte[count];
}

    private int SocketID = 0;
    private int SocketLimit;
    private Socket? ServerSocket;
    private List<ClientData> ClientSockets = new List<ClientData>();

    public delegate void ServerCallback(object o);

    private ServerCallback? ConnectionCallback;
    private ServerCallback? ReceiveCallback;

    private bool Active = false;

    public enum ServerState { Error, Start, Stop };

    public ServerState StartServer(int Port, int Limit, ServerCallback Connection, ServerCallback Receive)
    {
        if (Active)
        {
            ServerSocket?.Close();
            foreach (ClientData c in ClientSockets)
            {
                try { c.workSocket?.Shutdown(SocketShutdown.Both); } catch { }
                c.workSocket?.Close();
            }
            ClientSockets.Clear();
            return ServerState.Stop;

```

```

    }

    ConnectionCallback = Connection;
    ReceiveCallback = Receive;

    SocketLimit = Limit;

    IPEndPoint ip = new(IPAddress.Any, Port);
    ServerSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

    try
    {
        ServerSocket.Bind(ip);
        ServerSocket.Listen(10);
        ServerSocket.BeginAccept(new AsyncCallback(AcceptCallback), ServerSocket);
        Active = true;
    }
    catch { ServerSocket.Close(); return ServerState.Error; }

    return ServerState.Start;
}

public void AcceptCallback(IAsyncResult ar)
{
    Socket listener = (Socket)ar.AsyncState;
    if (listener == null) return;

    Socket handler;

    try { handler = listener.EndAccept(ar); }
    catch { ServerSocket?.Close(); Active = false; return; }

    if (SocketLimit > ClientSockets.Count)
    {
        ClientData clientData = new ClientData();

        clientData.ID = ++SocketID;

        clientData.workSocket = handler;

        ClientSockets.Add(clientData);

        ConnectionCallback(new ConnectData { ID = clientData.ID, Connect = true, Count =
ClientSockets.Count, Client = handler });

```

```

        handler.BeginReceive(clientData.buffer, 0, ClientData.count, 0, new AsyncCallback(ReadCallback),
clientData);
    }
    else handler.Close();

    listener.BeginAccept(new AsyncCallback(AcceptCallback), listener);
}

public void ReadCallback(IAsyncResult ar)
{
    ClientData cd = (ClientData)ar.AsyncState;
    if (cd == null) return;

    int bytes = 0;
    try { bytes = cd.workSocket.EndReceive(ar); }
    catch { }

    if (bytes == 0)
    {
        cd.workSocket?.Close();

        ClientSockets.Remove(cd);

        ConnectionCallback(new ConnectData { ID = cd.ID, Connect = false, Count = ClientSockets.Count,
Client = null });

        return;
    }

    ReceiveCallback(new ReceiveData { ID = cd.ID, Buffer = cd.buffer, Size = bytes, Client =
cd.workSocket });

    try
    {
        cd.workSocket?.BeginReceive(cd.buffer, 0, ClientData.count, 0, new
AsyncCallback(ReadCallback), cd);
    }
    catch { }
}
}
}

```

FormMain.cs

```

using System.Text;
using System.Numerics;

```

```

using System.Windows.Forms;
using static System.Net.Mime.MediaTypeNames;
using static System.Runtime.InteropServices.JavaScript.JSType;
using System.Security.Policy;
using System.Runtime.InteropServices;
using System.CodeDom;
using System.Linq;

```

```

namespace DroneSimulator

```

```

{
    public partial class Form_Main : Form
    {
        Screen2D screen2D = null;

```

```

        NetServerClients netServerClient = new NetServerClients();
        NetServerVisual netServerVisual = new NetServerVisual();

```

```

        List<Drone> AllDrones = new List<Drone>();

```

```

        public Form_Main()
        {
            InitializeComponent();

```

```

            RealMode.RealSimulation = radioButton_Real.Checked;
            numericUpDown_Acc_Update(null, null);
            numericUpDown_Gyr_Update(null, null);
            numericUpDown_Pos_Update(null, null);
            numericUpDown_Bar_Update(null, null);
            numericUpDown_Range_Update(null, null);
        }

```

```

private void ClientConnectionCallback(object o)
{
    NetServerClients.ConnectData data = (NetServerClients.ConnectData)o;

    Invoke((MethodInvoker)delegate
    {
        label_Clients_Num.Text = data.Count.ToString();
    });

    if (data.Connect)
    {
        Drone drone = new Drone(data.ID);
        drone.Create(1.0f, 0.5f, 1.0f);

        screen2D.CreateDrone(Color.Red, data.ID);

        AllDrones.Add(drone);
    }
    else
    {
        foreach (Drone drone in AllDrones)
        {
            if (drone.ID != data.ID) continue;
            drone.Close();

            screen2D.RemoveDrone(data.ID);

            AllDrones.Remove(drone);
            break;
        }
    }
}

```



```
}
```

```
private void ClientReceiveCallback(object o)
```

```
{
```

```
    NetServerClients.ReceiveData data = (NetServerClients.ReceiveData)o;
```

```
    Drone? drone = null;
```

```
    foreach (Drone d in AllDrones)
```

```
    {
```

```
        if (d.ID != data.ID) continue;
```

```
        drone = d;
```

```
        break;
```

```
    }
```

```
    if (drone == null) return;
```

```
    List<byte[]?>? send = drone.DataStream(data.Buffer, data.Size);
```

```
    if (send == null) return;
```

```
    try
```

```
    {
```

```
        foreach (byte[]? b in send)
```

```
        {
```

```
            if (b != null) data.Client?.Send(b);
```

```
        }
```

```
    }
```

```
    catch { }
```

```
}
```

```
private void button_Client_Start_Click(object sender, EventArgs e)
```

```

    {
        var done =
netServerClient.StartServer((int)numericUpDown_Clients_Port.Value,
(int)numericUpDown_Clients_Limit.Value, ClientConnectionCallback,
ClientReceiveCallback);
        switch (done)
        {
            case NetServerClients.ServerState.Error:
            {
                MessageBox.Show("Error to start clients server", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                break;
            }
            case NetServerClients.ServerState.Start:
            {
                button_Client_Start.Text = "Stop";
                button_Client_Start.BackColor = Color.LimeGreen;
                break;
            }
            case NetServerClients.ServerState.Stop:
            {
                label_Clients_Num.Text = "0";
                button_Client_Start.Text = "Start";
                button_Client_Start.BackColor = Color.Transparent;
                break;
            }
        }

        if (done != NetServerClients.ServerState.Start) return;

        pictureBox_2D.Image = null;

```

```
screen2D = new Screen2D(DrawCallback);  
}
```

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)  
{  
    Close();  
}
```

```
private void DrawCallback(Bitmap bmp)  
{  
    Invoke((MethodInvoker)delegate  
    {  
        if (pictureBox_2D.Image == null) pictureBox_2D.Image = bmp;  
        pictureBox_2D.Refresh();  
    });  
}
```

```
private void timer_Test_Tick(object sender, EventArgs e)  
{  
    if (screen2D == null) return;
```

```
    listBox_Drones.Items.Clear();
```

```
    foreach (Drone d in AllDrones)
```

```
    {  
        screen2D.Move(d.ID, d.PosXYZ, d.GetOrientation());
```

```
        string line = "ID:" + d.ID.ToString() + " Pitch:" +  
((int)d.Orientation.X).ToString() + " Roll:" + ((int)d.Orientation.Y).ToString() + "  
Yaw:" + ((int)d.Orientation.Z).ToString();
```

```

        listBox_Drones.Items.Add(line);
    }

    screen2D.DrawScene();
}

private void Form_Main_FormClosing(object sender,
FormClosingEventArgs e)
{
    foreach (Drone d in AllDrones) d.Close();
}

private void VisualConnectionCallback(object o)
{
    NetServerVisual.ConnectData data = (NetServerVisual.ConnectData)o;

    Invoke((MethodInvoker)delegate
    {
        label_Clients_Num.Text = data.Count.ToString();
    });

    if (data.Connect)
    {
        //---
    }
    else
    {
        //---
    }
}

```

```

private void VisualReceiveCallback(object o)
{
    NetServerVisual.ReceiveData data = (NetServerVisual.ReceiveData)o;

    int index = 0;

    foreach (Drone d in AllDrones)
    {
        VisualData.VisualDrone v = d.GetVisual(AllDrones.Count, index++);

        try { data.Client.Send(Drone.getBytes(v)); }
        catch { }
    }
}

private void button_Visual_Start_Click(object sender, EventArgs e)
{
    var done =
netServerVisual.StartServer((int)numericUpDown_Visual_Port.Value,
(int)numericUpDown_Visual_Limit.Value, VisualConnectionCallback,
VisualReceiveCallback);
    switch (done)
    {
        case NetServerVisual.ServerState.Error:
        {
            MessageBox.Show("Error to start visual server", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            break;
        }
        case NetServerVisual.ServerState.Start:
        {

```

```

        button_Visual_Start.Text = "Stop";
        button_Visual_Start.BackColor = Color.LimeGreen;
        break;
    }
    case NetServerVisual.ServerState.Stop:
    {
        label_Visual_Num.Text = "0";
        button_Visual_Start.Text = "Start";
        button_Visual_Start.BackColor = Color.Transparent;
        break;
    }
}

private void numericUpDown_Bar_Update(object sender, EventArgs e)
{
    try
    {
        RealMode.Barometer.Pressure =
uint.Parse(textBox_Bar_Pressure.Text); }
    catch
    {
        RealMode.Barometer.Pressure = 102258;
        MessageBox.Show("Pressure invalid format", "Barometer error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    RealMode.Barometer.Freq = (uint)numericUpDown_Bar_Freq.Value;
    RealMode.Barometer.Noise = (float)numericUpDown_Bar_Noise.Value;
    RealMode.Barometer.Lateness =
(float)numericUpDown_Bar_Laten.Value;
    RealMode.Barometer.Enable = checkBox_Bar_Enable.Checked;

```

```

    }

    private void checkBox_Mode_Real_CheckedChanged(object sender,
EventArgs e)
    {
        RealMode.RealSimulation = radioButton_Real.Checked;
    }

    private void numericUpDown_Acc_Update(object sender, EventArgs e)
    {
        RealMode.Accelerometer.Freq =
(uint)numericUpDown_Acc_Freq.Value;
        RealMode.Accelerometer.Noise =
(float)numericUpDown_Acc_Noise.Value;
        RealMode.Accelerometer.Lateness =
(float)numericUpDown_Acc_Laten.Value;

        RealMode.Accelerometer.ScaleLeft =
(float)numericUpDown_Acc_Scale_Left.Value;
        RealMode.Accelerometer.ScaleRight =
(float)numericUpDown_Acc_Scale_Rigth.Value;
    }

    private void numericUpDown_Gyr_Update(object sender, EventArgs e)
    {
        RealMode.Gyroscope.Freq = (uint)numericUpDown_Gyr_Freq.Value;
        RealMode.Gyroscope.Noise = (float)numericUpDown_Gyr_Noise.Value;
        RealMode.Gyroscope.Lateness =
(float)numericUpDown_Gyr_Laten.Value;
    }

```

```

        RealMode.Gyroscope.Shift.X =
(float)numericUpDown_Gyr_Shift_X.Value;
        RealMode.Gyroscope.Shift.Y =
(float)numericUpDown_Gyr_Shift_Y.Value;
        RealMode.Gyroscope.Shift.Z =
(float)numericUpDown_Gyr_Shift_Z.Value;
    }

```

```

private void numericUpDown_Pos_Update(object sender, EventArgs e)
{
    RealMode.Position.Freq = (uint)numericUpDown_Pos_Freq.Value;
    RealMode.Position.Noise = (float)numericUpDown_Pos_Noise.Value;
    RealMode.Position.Lateness = (float)numericUpDown_Pos_Laten.Value;
    RealMode.Position.Enable = checkBox_Pos_Enable.Checked;
}

```

```

private void numericUpDown_Range_Update(object sender, EventArgs e)
{
    RealMode.Range.Freq = (uint)numericUpDown_Range_Freq.Value;
    RealMode.Range.Noise = (float)numericUpDown_Range_Noise.Value;
    RealMode.Range.Lateness =
(float)numericUpDown_Range_Laten.Value;
    RealMode.Range.Enable = checkBox_Range_Enable.Checked;

    RealMode.Range.MaxHeight =
(float)numericUpDown_Range_Max.Value;
}

```

```

private void numericUpDown_OF_Update(object sender, EventArgs e)
{
    RealMode.OpticalFlow.Freq = (uint)numericUpDown_OF_Freq.Value;
}

```



```

        RealMode.OpticalFlow.Noise =
(float)numericUpDown_OF_Noise.Value;
        RealMode.OpticalFlow.Lateness =
(float)numericUpDown_OF_Laten.Value;
        RealMode.OpticalFlow.Enable = checkBox_OF_Enable.Checked;

        RealMode.OpticalFlow.Lens = (uint)numericUpDown_OF_Lens.Value
* 10;
        RealMode.OpticalFlow.MaxHeight =
(float)numericUpDown_OF_Len.Value;

        RealMode.OpticalFlow.Error = (float)numericUpDown_OF_Error.Value
* 10;
        RealMode.OpticalFlow.Wait = (uint)numericUpDown_OF_Wait.Value
* 1000;
    }

    private void checkBox_Area_Freeze_CheckedChanged(object sender,
EventArgs e)
    {
        Area.Poosition.Freeze.X = checkBox_Area_Freeze_X.Checked;
        Area.Poosition.Freeze.Y = checkBox_Area_Freeze_Y.Checked;
        Area.Poosition.Freeze.Z = checkBox_Area_Freeze_Z.Checked;
    }

    private void numericUpDown_Area_Wind_Update(object sender,
EventArgs e)
    {
        Area.Wind.Enable = checkBox_Area_Wind_Enable.Checked;
        Area.Wind.Speed.From =
(float)numericUpDown_Area_Wind_Speed_From.Value;

```

```

        Area.Wind.Speed.To                                     =
(float)numericUpDown_Area_Wind_Speed_To.Value;
        Area.Wind.Direction                                   =
(float)numericUpDown_Area_Wind_Direction.Value;
        Area.Wind.Density                                     =
(float)numericUpDown_Area_Wind_Density.Value;
        Area.Wind.Resist = (float)numericUpDown_Area_Wind_Resist.Value;
    }
}

```

ПРИЛОЖЕНИЕ Д. КОД КЛИЕНТСКОЙ ЧАСТИ C++

Drone.h

```
#pragma once
```

```
#include "DroneData.h"
```

```
#include <Windows.h>
```

```
#include <vcclr.h>
```

```
#include "Orientation.h"
```

```
#include "joypad.h"
```

```
#include "BarometricFilter.h"
```

```
#using <System.dll>
```

```
#using <mscorlib.dll>
```

```
using namespace System;
```

```
using namespace System::Collections::Generic;
```

```
using namespace System::Runtime::InteropServices;
```

```
namespace DroneClient {
```

```
    public ref class Drone
```

```
    {
```

```
    public:
```

```
        float AccX, AccY, AccZ;
```

```
        float GyrX, GyrY, GyrZ;
```

```
        float Bar;
```

```
        float PosX, PosY;
```

```
float LaserRange;
```

```
unsigned int TimeAcc, TimeGyr, prevTimeGyr;  
float dtGyr;
```

```
float posXcalc=1, posYcalc=1;  
float inX=1, inY=2, inZ=3;
```

```
float MotorUL, MotorUR, MotorDL, MotorDR;
```

```
float pitch, roll, yaw;  
float tilt;
```

```
float TrueRange;  
float desPitch, desRoll, desYaw;
```

```
float barHeight;  
unsigned int TimeRange;  
float dtRange;  
bool altHold = false;  
float altRef;
```

```
float inHeight=0;
```

```

Joypad^ joy;
static array<Byte>^ GetBytes(Object^ data);
static Object^ FromBytes(array<Byte>^ arr, Type^ type);

private:
    array<Byte>^ SendDataMotor4();
    array<Byte>^ RecvDataAcc(array<Byte>^ data);
    array<Byte>^ RecvDataGyr(array<Byte>^ data);
    array<Byte>^ RecvDataRange(array<Byte>^ data);
    array<Byte>^ RecvDataLocal(array<Byte>^ data);
    array<Byte>^ RecvDataBar(array<Byte>^ data);
    array<Byte>^ ClientRequestResponse(DroneData::DataHead head,
array<Byte>^ body);

    void setMotors();

    literal int DroneStreamCount = 512;
    array<Byte>^ DroneStreamData;
    int DroneStreamIndex;
    DroneData::DataHead DroneStreamHead;

public:
    Drone(); // Конструктор для инициализации

    System::Collections::Generic::List<array<Byte>^>^
DataStream(array<Byte>^ data, int size);
    array<Byte>^ SendRequest();
    void calcPosition();
    void calcHeight();
    void updateData();

```

```

};
}
Drone.cpp
#include "Drone.h"

static const float PI = 3.14159265359f;
static const float TO_DEG = 180.0f / PI;
static const float TO_RAD = PI / 180.0f;
namespace DroneClient {

    // Реализация статического метода GetBytes
    array<Byte>^ Drone::GetBytes(Object^ data)
    {
        int size = Marshal::SizeOf(data);
        array<Byte>^ arr = gcnew array<Byte>(size);

        IntPtr ptr = IntPtr::Zero;
        try
        {
            ptr = Marshal::AllocHGlobal(size);
            Marshal::StructureToPtr(data, ptr, true);
            Marshal::Copy(ptr, arr, 0, size);
        }
        finally
        {
            Marshal::FreeHGlobal(ptr);
        }
        return arr;
    }
}

```

```
// Реализация статического метода FromBytes
Object^ Drone::FromBytes(array<Byte>^ arr, Type^ type)
{
    Object^ mem = gcnew Object();

    int size = Marshal::SizeOf(type);
    IntPtr ptr = IntPtr::Zero;
    try
    {
        ptr = Marshal::AllocHGlobal(size);
        Marshal::Copy(arr, 0, ptr, size);
        mem = Marshal::PtrToStructure(ptr, type);
    }
    finally
    {
        Marshal::FreeHGlobal(ptr);
    }
    return mem;
}
```

```
// Реализация приватного метода SendDataMotor4
array<Byte>^ Drone::SendDataMotor4()
{

    DroneData::DataMotor4 mot4;

    mot4.Head.Size = Marshal::SizeOf(DroneData::DataMotor4::typeid);
    mot4.Head.Mode = DroneData::DataMode::Response;
    mot4.Head.Type = DroneData::DataType::DataMotor4;
```

```

        updateData();
        setMotors();

        mot4.UL = MotorUL;
        mot4.UR = MotorUR;
        mot4.DL = MotorDL;
        mot4.DR = MotorDR;

        return GetBytes(mot4);
    }

array<Byte>^ Drone::RecvDataAcc(array<Byte>^ data)
{
    DroneData::DataAcc imu = (DroneData::DataAcc)FromBytes(data,
DroneData::DataAcc::typeid);

    AccX = imu.Acc.X; AccY = imu.Acc.Y; AccZ = imu.Acc.Z;
    TimeAcc = imu.Time;

    return gcnew array<Byte>(0);
}

array<Byte>^ Drone::RecvDataGyr(array<Byte>^ data)
{
    DroneData::DataGyr imu = (DroneData::DataGyr)FromBytes(data,
DroneData::DataGyr::typeid);

    GyrX = imu.Gyr.X; GyrY = imu.Gyr.Y; GyrZ = imu.Gyr.Z;
    prevTimeGyr = TimeGyr;
    TimeGyr = imu.Time;

```



```

dtGyr = TimeGyr - prevTimeGyr;

dtGyr *= 0.001f;

return gnew array<Byte>(0);
}

array<Byte>^ Drone::RecvDataRange(array<Byte>^ data)
{
    DroneData::DataRange pos = (DroneData::DataRange)FromBytes(data,
DroneData::DataRange::typeid);

    LaserRange = pos.LiDAR;
    static float prevTimeRange = TimeRange;
    TimeRange = pos.Time;

    dtRange = TimeRange - prevTimeRange;

    return gnew array<Byte>(0);
}

array<Byte>^ Drone::RecvDataLocal(array<Byte>^ data)
{
    DroneData::DataLocal pos = (DroneData::DataLocal)FromBytes(data,
DroneData::DataLocal::typeid);

    PosX = pos.Local.X; PosY = pos.Local.Y;

    return gnew array<Byte>(0);
}

```

```

array<Byte>^ Drone::RecvDataBar(array<Byte>^ data)
{
    DroneData::DataBar bar = (DroneData::DataBar)FromBytes(data,
DroneData::DataBar::typeid);
    Bar = bar.Pressure;

    return gnew array<Byte>(0);
}

```

```

array<Byte>^ Drone::ClientRequestResponse(DroneData::DataHead head,
array<Byte>^ body)
{
    array<Byte>^ zero = gnew array<Byte>(0);

    switch (head.Type)
    {
    case DroneData::DataType::DataAcc:
        if (head.Mode == DroneData::DataMode::Request)
        {
            return zero; // Запрос данных (не реализовано)
        }
        else
        {
            return RecvDataAcc(body); // Пришли данные
        }

    case DroneData::DataType::DataGyr:
        if (head.Mode == DroneData::DataMode::Request)
        {
            return zero; // Запрос данных (не реализовано)
        }
    }
}

```

```

    }
    else
    {
        return RecvDataGyr(body); // Пришли данные
    }

case DroneData::DataType::DataRange:
    if (head.Mode == DroneData::DataMode::Request)
    {
        return zero; // Запрос данных (не реализовано)
    }
    else
    {
        return RecvDataRange(body); // Пришли данные
    }

case DroneData::DataType::DataLocal:
    if (head.Mode == DroneData::DataMode::Request)
    {
        return zero; // Запрос данных (не реализовано)
    }
    else
    {
        return RecvDataLocal(body); // Пришли данные
    }

case DroneData::DataType::DataBar:
    if (head.Mode == DroneData::DataMode::Request)
    {
        return zero; // Запрос данных (не реализовано)
    }

```

```

else
{
    return RecvDataBar(body); // Пришли данные
}

case DroneData::DataType::DataMotor4:
    if (head.Mode == DroneData::DataMode::Request)
    {
        return SendDataMotor4(); // Запрос данных
    }
    else
    {
        return zero; // Пришли данные (не реализовано)
    }
}

return zero;
}

```

```

static float Clamp01(float v)
{
    if (v < 0.0f) return 0.0f;
    if (v > 1.0f) return 1.0f;
    return v;
}

```

/* ————— PD-регулятор и микширование —————

*/

```

void Drone::setMotors()
{

```

```

/* ----- ВХОДНЫЕ КАНАЛЫ ----- */
array<UInt16>^ ch = joy->Channels;
float fpow = (ch[2] - 1000) * 0.001f;    // 0...1

/* ----- желаемые углы ----- */
const float maxAngle = 20.0f;
desPitch = -(ch[1] - 1499) * maxAngle / 500.0f;
desRoll = (ch[0] - 1499) * maxAngle / 500.0f;

/* ----- PD-регулятор по горизонтали ----- */
static float prevErrPitch = 0.0f, prevErrRoll = 0.0f;
const float errPitch = -(desPitch - pitch);
const float errRoll = desRoll - roll;

float dt = dtGyr;    // период 10 мс (100 Гц)
if (!dt) dt = 0.015f;
const float dPitch = (errPitch - prevErrPitch) / dt;
const float dRoll = (errRoll - prevErrRoll) / dt;

const float Kp = 0.20f;
const float Kd = 0.06f;

const float forcePitch = Kp * errPitch + Kd * dPitch;
float forceRoll = Kp * errRoll + Kd * dRoll;
// forceRoll = 0;
prevErrPitch = errPitch;
prevErrRoll = errRoll;

```

```

/* ----- PD-регулятор по вертикали ----- */
float correctRange = 0.0f;
const float KpRange = 1.5f;
const float KdRange = 3.0f;

TrueRange = LaserRange * tilt;
if (ch[5] > 1500 && !altHold)
{
    altHold = true;
    altRef = TrueRange;
}
if (ch[5] < 1500)
{
    altHold = false;
}

if (altHold)
{
    static float prevErr = 0.0f;
    const float dt = 0.02f;
    const float err = altRef - TrueRange;
    const float der = (err - prevErr) / dt;

    correctRange = KpRange * err + KdRange * der;

    prevErr = err;
}

/* ----- распределение на моторы ----- */
fpow += correctRange;
MotorUL = fpow - forcePitch + forceRoll;

```

```

MotorUR = fpow - forcePitch - forceRoll;
MotorDL = fpow + forcePitch + forceRoll;
MotorDR = fpow + forcePitch - forceRoll;

```

```

}

```

```

// Реализация конструктора

```

```

Drone::Drone()

```

```

{

```

```

    DroneStreamData = gcnew array<Byte>(DroneStreamCount); //

```

Инициализация массива

```

    DroneStreamIndex = 0;

```

```

    DroneStreamHead.Mode = DroneData::DataMode::None;

```

```

    DroneStreamHead.Size = 0;

```

```

    DroneStreamHead.Type = DroneData::DataType::None;

```

```

    this->joy = gcnew Joypad();

```

```

    this->joy->Start("COM7", 115200);

```

```

}

```

```

// Реализация метода DataStream

```

```

List<array<Byte>^>^ Drone::DataStream(array<Byte>^ data, int size)

```

```

{

```

```

System::Collections::Generic::List<array<Byte>^>^ ret = gcnew
System::Collections::Generic::List<array<Byte>^>();

```

```

if (data == nullptr) return ret; // Последовательность не сформирована

```

```

if (size + DroneStreamIndex > DroneStreamCount) return nullptr; //

```

Ошибка переполнения

```

Array::Copy(data, 0, DroneStreamData, DroneStreamIndex, size);

```

```

DroneStreamIndex += size;

```

```

while (true)

```

```

{

```

```

    if (DroneStreamHead.Size == 0) // Заголовок еще не получен

```

```

    {

```

```

        if (DroneStreamIndex < DroneData::DataHead::StrLen) return ret;

```

```

        DroneStreamHead

```

```

        (DroneData::DataHead)FromBytes(DroneStreamData,

```

```

        DroneData::DataHead::typeid);

```

```

    }

```

```

    if (DroneStreamHead.Size > DroneStreamIndex) break; // Пакет еще
не полный

```

```

    array<Byte>^ body = gcnew array<Byte>(DroneStreamHead.Size);

```

```

    Array::Copy(DroneStreamData, 0, body, 0, DroneStreamHead.Size);

```

```

    int shift = DroneStreamHead.Size;

```

```

    DroneStreamIndex -= shift;

```



```
        Array::Copy(DroneStreamData, shift, DroneStreamData, 0,
DroneStreamIndex); // Сдвиг массива
```

```
DroneStreamHead.Size = 0; // Сброс заголовка
```

```
ret->Add(ClientRequestResponse(DroneStreamHead, body));
}
```

```
return ret;
}
```

```
// Реализация метода SendRequest
```

```
array<Byte>^ Drone::SendRequest()
```

```
{
    DroneData::DataHead^ acc = gcnew DroneData::DataHead();
    acc->Size = DroneData::DataHead::StrLen;
    acc->Mode = DroneData::DataMode::Request;
    acc->Type = DroneData::DataType::DataAcc;
```

```
    DroneData::DataHead^ gyr = gcnew DroneData::DataHead();
    gyr->Size = DroneData::DataHead::StrLen;
    gyr->Mode = DroneData::DataMode::Request;
    gyr->Type = DroneData::DataType::DataGyr;
```

```
    DroneData::DataHead^ range = gcnew DroneData::DataHead();
    range->Size = DroneData::DataHead::StrLen;
    range->Mode = DroneData::DataMode::Request;
    range->Type = DroneData::DataType::DataRange;
```

```
    DroneData::DataHead^ local = gcnew DroneData::DataHead();
    local->Size = DroneData::DataHead::StrLen;
```

```
local->Mode = DroneData::DataMode::Request;  
local->Type = DroneData::DataType::DataLocal;
```

```
DroneData::DataHead^ bar = gcnew DroneData::DataHead();  
bar->Size = DroneData::DataHead::StrLen;  
bar->Mode = DroneData::DataMode::Request;  
bar->Type = DroneData::DataType::DataBar;
```

```
List<array<byte>^>^ list = gcnew List<array<byte>^>();
```

```
list->Add(GetBytes(acc));  
list->Add(GetBytes(gyr));  
list->Add(GetBytes(range));  
list->Add(GetBytes(local));  
list->Add(GetBytes(bar));
```

```
list->Add(SendDataMotor4());
```

```
int count = 0;
```

```
for each(array<byte>^ d in list) count += d->Length;
```

```
array<byte>^ send = gcnew array<byte>(count);
```

```
count = 0;  
for each (array<byte> ^ d in list)  
{  
    d->CopyTo(send, count);  
    count += d->Length;
```

```

    }

    return send;
}

void Drone::calcPosition()
{
    static float k = 0.1f;
    float dt = dtGyr;
    this->posXcalc += this->inX * dt * dt * 0.001f;
    this->posYcalc += this->inY * dt * dt * 0.001f;

    this->posXcalc = k * this->PosX + (1 - k) * this->posXcalc;
    this->posYcalc = k * this->PosY + (1 - k) * this->posYcalc;
}

void Drone::calcHeight()
{
    static float k = 0.1f;
    float dt = dtGyr ;
    this->inHeight += (this->inZ-1.0f) * dt * dt ;
    this->inHeight = k * this->LaserRange + (1 - k) * this->inHeight;
}

struct Att { float pitch, roll, yaw; }; // результат, °

constexpr float kAlpha = 0.98f;

```

```

Att attitude(const Vec3& acc, const Vec3& gyr, float dt)
{
    // --- 1. Тангаж/крен только из акселерометра -----
-
    float ax = acc.x, ay = acc.y, az = acc.z;

    float pitchAcc = std::atan2(ay, std::sqrt(ax * ax + az * az)) * TO_DEG;
    float rollAcc = std::atan2(-ax, std::sqrt(ay * ay + az * az)) * TO_DEG;

    // --- 2. Интегрируем гироскоп (угол = скорость * время) -----
-----
    static float pitch = 0.0f, roll = 0.0f, yaw = 0.0f;  // запоминаем между
ВЫЗОВАМИ

    float pitchGyro = pitch + gyr.x * dt;  // вращение вокруг X
    float rollGyro = roll + gyr.y * dt;  // вокруг Y
    yaw += gyr.z * dt;  // вокруг Z

    // --- 3. Комплементарный фильтр -----
    pitch = kAlpha * pitchGyro + (1.0f - kAlpha) * pitchAcc;
    roll = kAlpha * rollGyro + (1.0f - kAlpha) * rollAcc;

    // Нормализация курса в диапазон 0-360°
    if (yaw >= 360.0f) yaw -= 360.0f;
    else if (yaw < 0.0f) yaw += 360.0f;

    return { pitch, roll, yaw };
}

```

```

void Drone::updateData(){
    Vec3 acc{ this->AccX,this->AccY, this->AccZ};
    Vec3 gyr{ this->GyrX,this->GyrY, this->GyrZ };
    Vec3 mag{ 1,0, 0};
    if (dtGyr > 0.0001f)
    {

        float dt = dtGyr;
        ORI result = WorkAccGyroMag(acc, gyr, mag, 0, 0.01, dt);
        this->pitch = result.Pitch * TO_DEG;
        this->roll = result.Roll * TO_DEG;
        this->yaw = result.Yaw;

        //Att r = attitude(acc, gyr, dtGyr); // r.pitch, r.roll, r.yaw в градусах
        //this->pitch = r.pitch;
        //this->roll = r.roll;
        //this->yaw = r.yaw;

        this->tilt = result.cosZ;
        this->inX = result.IneX;
        this->inY = result.IneY;
        this->inZ = result.IneZ;

        this->barHeight = computeBarHeight(Bar, 24.0f);

        calcPosition();
        calcHeight();
    }
}

```

```
}
```

```
}
```

```
}
```

FormMain.h

```
#pragma once
```

```
#include "Drone.h"      // Новый Drone.h
```

```
#include "DroneData.h"  // Новый DroneData.h
```

```
#include "NetClient.h"
```

```
#include <cliext/utility>
```

```
#include <ppltasks.h>      // Task, create_task
```

```
#include "WsServer.h"
```

```
namespace DroneClientCpp {
```

```
    using namespace System;
```

```
    using namespace System::ComponentModel;
```

```
    using namespace System::Collections;
```

```
    using namespace System::Windows::Forms;
```

```
    using namespace System::Data;
```

```
    using namespace System::Drawing;
```

```
    using namespace DroneClient; // Пространство имен из Drone.h
```

```
    using namespace DroneData;   // Пространство имен из DroneData.h
```

```
    using namespace DroneSimulator;
```

```
    using namespace System::Threading;
```

```
    using namespace System::Threading::Tasks;
```

```
    using namespace System::Net::WebSockets;
```

```
    using namespace concurrency;      // пространство PPL
```

```

public ref class FormMain : public System::Windows::Forms::Form
{
public:
    FormMain(void)
    {
        InitializeComponent();
        netClient = gcnew NetClient();
        dataDrone = gcnew Drone(); // Инициализация объекта Drone
        wsClient = gcnew WsServer();

    }

protected:
    ~FormMain()
    {
        if (components) delete components;
        if (netClient != nullptr)
        {
            netClient->Close();
            netClient = nullptr;
        }
    }

private:
    // Элементы управления (оставить как есть)
    System::Windows::Forms::GroupBox^ groupBox_Server;
    System::Windows::Forms::Label^ label_Port;

```

```

System::Windows::Forms::Button^ button_Connect;
System::Windows::Forms::NumericUpDown^
numericUpDown_Server_Port;
System::Windows::Forms::TextBox^ textBox_Server_Addr;
System::Windows::Forms::GroupBox^ groupBox_Acc;
System::Windows::Forms::Label^ label_Addr;
System::Windows::Forms::Label^ label_Acc_X_Info;
System::Windows::Forms::Label^ label_Acc_Z;
System::Windows::Forms::Label^ label_Acc_Y;
System::Windows::Forms::Label^ label_Acc_X;
System::Windows::Forms::Label^ label_Acc_Z_Info;
System::Windows::Forms::Label^ label_Acc_Y_Info;
System::ComponentModel::BackgroundWorker^ backgroundWorker1;
System::Windows::Forms::GroupBox^ groupBox_Gyr;
System::Windows::Forms::Label^ label_Gyr_Z;
System::Windows::Forms::Label^ label_Gyr_Y;
System::Windows::Forms::Label^ label_Gyr_X;
System::Windows::Forms::Label^ label_Gyr_Z_Info;
System::Windows::Forms::Label^ label_Gyr_Y_Info;
System::Windows::Forms::Label^ label_Gyr_X_Info;
System::Windows::Forms::GroupBox^ groupBox_Pos;
System::Windows::Forms::Label^ label_Pos_L;
System::Windows::Forms::Label^ label_Pos_Y;
System::Windows::Forms::Label^ label_Pos_X;
System::Windows::Forms::Label^ label_Pos_L_Info;
System::Windows::Forms::Label^ label_Pos_Y_Info;
System::Windows::Forms::Label^ label_Pos_X_Info;
System::Windows::Forms::GroupBox^ groupBox_Input;
System::Windows::Forms::Button^ button_RR;
System::Windows::Forms::Button^ button_LL;
System::Windows::Forms::Button^ button_MR;

```



```

System::Windows::Forms::Button^ button_ML;
System::Windows::Forms::Button^ button_DD;
System::Windows::Forms::Button^ button_UU;
System::Windows::Forms::TrackBar^ trackBar_Power;
System::Windows::Forms::Label^ label_Pow;
System::Windows::Forms::Timer^ timer1;
System::ComponentModel::IContainer^ components;

// Наши поля
NetClient^ netClient;
Drone^ dataDrone;

private: System::Windows::Forms::Label^ label_time_acc;
private: System::Windows::Forms::Label^ label_time_gyr;
private: System::Windows::Forms::Label^ label_time_range;
private: System::Windows::Forms::GroupBox^ groupBox1;
private: System::Windows::Forms::Button^ button1;
    const float pow = 0.1f;

    WsServer^ wsClient;        // WebSocket-клиент
private: System::Windows::Forms::Label^ label3;
private: System::Windows::Forms::Label^ label2;
private: System::Windows::Forms::Label^ label1;
private: System::Windows::Forms::Label^ label6;
private: System::Windows::Forms::Label^ label5;
private: System::Windows::Forms::Label^ label4;
    bool        _working = false;
private: System::Windows::Forms::Label^ label8;
private: System::Windows::Forms::Label^ label9;
private: System::Windows::Forms::Label^ label10;
private: System::Windows::Forms::Label^ label11;
private: System::Windows::Forms::Label^ label12;

```

```
private: System::Windows::Forms::Label^ label7;
```

```
// Наши методы
```

```
private:
```

```
void ConnectionCallback(Object^ o)
```

```
{
```

```
    NetClient::ConnectData^ data = (NetClient::ConnectData^)o;
```

```
    if (!data->Connect)
```

```
    {
```

```
        try
```

```
        {
```

```
            this->Invoke(gcnew
```

```
                Action(this,
```

```
&FormMain::UpdateDisconnectedUI));
```

```
        }
```

```
        catch (...) {}
```

```
        return;
```

```
    }
```

```
    // Отправка начальных данных при подключении
```

```
    data->Server->Send(dataDrone->SendRequest());
```

```
}
```

```
void UpdateDisconnectedUI()
```

```
{
```

```
    button_Connect->Text = "Connect";
```

```
    button_Connect->BackColor = Color::Transparent;
```

```
    MessageBox::Show("Connection closed");
```

```
}
```

```

void ReceiveCallback(Object^ o)
{
    NetClient::ReceiveData^ data = (NetClient::ReceiveData^)o;
    System::Collections::Generic::List<array<Byte>^>^ responses =
dataDrone->DataStream(data->Buffer, data->Size);

    if (responses == nullptr) return;
    try
    {
        for each (array<Byte> ^ b in responses)
        {
            if (b != nullptr) data->Server->Send(b);
        }
    }
    catch (...) {}
}

```

#pragma region Windows Form Designer generated code

```

/// <summary>
/// Требуемый метод для поддержки конструктора — не изменяйте
/// содержимое этого метода с помощью редактора кода.
/// </summary>
void InitializeComponent(void)
{
    this->components = (gcnew System::ComponentModel::Container());
    this->groupBox_Server = (gcnew
System::Windows::Forms::GroupBox());
    this->label_Addr = (gcnew System::Windows::Forms::Label());
    this->button_Connect = (gcnew System::Windows::Forms::Button());

```

```

        this->numericUpDown_Server_Port = (gcnew
System::Windows::Forms::NumericUpDown());

        this->textBox_Server_Addr = (gcnew
System::Windows::Forms::TextBox());

        this->label_Port = (gcnew System::Windows::Forms::Label());
        this->groupBox_Acc = (gcnew
System::Windows::Forms::GroupBox());

        this->label_time_acc = (gcnew System::Windows::Forms::Label());
        this->label_Acc_Z = (gcnew System::Windows::Forms::Label());
        this->label_Acc_Y = (gcnew System::Windows::Forms::Label());
        this->label_Acc_X = (gcnew System::Windows::Forms::Label());
        this->label_Acc_Z_Info = (gcnew System::Windows::Forms::Label());
        this->label_Acc_Y_Info = (gcnew System::Windows::Forms::Label());
        this->label_Acc_X_Info = (gcnew System::Windows::Forms::Label());
        this->backgroundWorker1 = (gcnew
System::ComponentModel::BackgroundWorker());

        this->groupBox_Gyr = (gcnew
System::Windows::Forms::GroupBox());

        this->label_time_gyr = (gcnew System::Windows::Forms::Label());
        this->label_Gyr_Z = (gcnew System::Windows::Forms::Label());
        this->label_Gyr_Y = (gcnew System::Windows::Forms::Label());
        this->label_Gyr_X = (gcnew System::Windows::Forms::Label());
        this->label_Gyr_Z_Info = (gcnew System::Windows::Forms::Label());
        this->label_Gyr_Y_Info = (gcnew System::Windows::Forms::Label());
        this->label_Gyr_X_Info = (gcnew System::Windows::Forms::Label());
        this->groupBox_Pos = (gcnew
System::Windows::Forms::GroupBox());

        this->label_time_range = (gcnew System::Windows::Forms::Label());
        this->label_Pos_L = (gcnew System::Windows::Forms::Label());
        this->label_Pos_Y = (gcnew System::Windows::Forms::Label());
        this->label_Pos_X = (gcnew System::Windows::Forms::Label());

```

```

this->label_Pos_L_Info = (gcnew System::Windows::Forms::Label());
this->label_Pos_Y_Info = (gcnew System::Windows::Forms::Label());
this->label_Pos_X_Info = (gcnew System::Windows::Forms::Label());
    this->groupBox_Input = (gcnew
System::Windows::Forms::GroupBox());

    this->label_Pow = (gcnew System::Windows::Forms::Label());
    this->button_RR = (gcnew System::Windows::Forms::Button());
    this->button_LL = (gcnew System::Windows::Forms::Button());
    this->button_MR = (gcnew System::Windows::Forms::Button());
    this->button_ML = (gcnew System::Windows::Forms::Button());
    this->button_DD = (gcnew System::Windows::Forms::Button());
    this->button_UU = (gcnew System::Windows::Forms::Button());
    this->trackBar_Power = (gcnew
System::Windows::Forms::TrackBar());

    this->timer1 = (gcnew
System::Windows::Forms::Timer(this->components));

    this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
    this->label6 = (gcnew System::Windows::Forms::Label());
    this->label5 = (gcnew System::Windows::Forms::Label());
    this->label4 = (gcnew System::Windows::Forms::Label());
    this->label3 = (gcnew System::Windows::Forms::Label());
    this->label2 = (gcnew System::Windows::Forms::Label());
    this->label1 = (gcnew System::Windows::Forms::Label());
    this->button1 = (gcnew System::Windows::Forms::Button());
    this->label7 = (gcnew System::Windows::Forms::Label());
    this->label8 = (gcnew System::Windows::Forms::Label());
    this->label9 = (gcnew System::Windows::Forms::Label());
    this->label10 = (gcnew System::Windows::Forms::Label());
    this->label11 = (gcnew System::Windows::Forms::Label());
    this->label12 = (gcnew System::Windows::Forms::Label());
    this->groupBox_Server->SuspendLayout();

```

```
(cli::safe_cast<System::ComponentModel::ISupportInitialize^(this->numericUp  
Down_Server_Port))->BeginInit();
```

```
    this->groupBox_Acc->SuspendLayout();  
    this->groupBox_Gyr->SuspendLayout();  
    this->groupBox_Pos->SuspendLayout();  
    this->groupBox_Input->SuspendLayout();
```

```
(cli::safe_cast<System::ComponentModel::ISupportInitialize^(this->trackBar_Po  
wer))->BeginInit();
```

```
    this->groupBox1->SuspendLayout();  
    this->SuspendLayout();  
    //  
    // groupBox_Server  
    //  
    this->groupBox_Server->Controls->Add(this->label_Addr);  
    this->groupBox_Server->Controls->Add(this->button_Connect);
```

```
this->groupBox_Server->Controls->Add(this->numericUpDown_Server_Port);  
    this->groupBox_Server->Controls->Add(this->textBox_Server_Addr);  
    this->groupBox_Server->Controls->Add(this->label_Port);  
    this->groupBox_Server->Dock =
```

```
System::Windows::Forms::DockStyle::Top;
```

```
    this->groupBox_Server->Location = System::Drawing::Point(0, 0);  
    this->groupBox_Server->Margin =
```

```
System::Windows::Forms::Padding(4);
```

```
    this->groupBox_Server->Name = L"groupBox_Server";  
    this->groupBox_Server->Padding =
```

```
System::Windows::Forms::Padding(4);
```

```
    this->groupBox_Server->Size = System::Drawing::Size(533, 110);  
    this->groupBox_Server->TabIndex = 0;
```

```

this->groupBox_Server->TabStop = false;
this->groupBox_Server->Text = L"Server";
//
// label_Addr
//
this->label_Addr->AutoSize = true;
this->label_Addr->Location = System::Drawing::Point(5, 26);
this->label_Addr->Name = L"label_Addr";
this->label_Addr->Size = System::Drawing::Size(39, 16);
this->label_Addr->TabIndex = 5;
this->label_Addr->Text = L"Addr:";
//
// button_Connect
//
this->button_Connect->Location = System::Drawing::Point(153, 66);
this->button_Connect->Margin =
System::Windows::Forms::Padding(3, 2, 3, 2);
this->button_Connect->Name = L"button_Connect";
this->button_Connect->Size = System::Drawing::Size(116, 27);
this->button_Connect->TabIndex = 4;
this->button_Connect->Text = L"Connect";
this->button_Connect->UseVisualStyleBackColor = true;
this->button_Connect->Click += gcnew System::EventHandler(this,
&FormMain::button_Connect_Click);
//
// numericUpDown_Server_Port
//
this->numericUpDown_Server_Port->Location =
System::Drawing::Point(56, 66);
this->numericUpDown_Server_Port->Margin =
System::Windows::Forms::Padding(3, 2, 3, 2);

```

```

        this->numericUpDown_Server_Port->Maximum =
System::Decimal(gcnew cli::array< System::Int32 >(4) { 65000, 0, 0, 0 });
        this->numericUpDown_Server_Port->Minimum =
System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });
        this->numericUpDown_Server_Port->Name =
L"numericUpDown_Server_Port";
        this->numericUpDown_Server_Port->Size =
System::Drawing::Size(93, 22);
        this->numericUpDown_Server_Port->TabIndex = 3;
        this->numericUpDown_Server_Port->Value = System::Decimal(gcnew
cli::array< System::Int32 >(4) { 1001, 0, 0, 0 });
        //
        // textBox_Server_Addr
        //
        this->textBox_Server_Addr->Location = System::Drawing::Point(56,
23);
        this->textBox_Server_Addr->Margin =
System::Windows::Forms::Padding(3, 2, 3, 2);
        this->textBox_Server_Addr->Name = L"textBox_Server_Addr";
        this->textBox_Server_Addr->Size = System::Drawing::Size(215, 22);
        this->textBox_Server_Addr->TabIndex = 2;
        this->textBox_Server_Addr->Text = L"127.0.0.1";
        //
        // label_Port
        //
        this->label_Port->AutoSize = true;
        this->label_Port->Location = System::Drawing::Point(5, 70);
        this->label_Port->Name = L"label_Port";
        this->label_Port->Size = System::Drawing::Size(34, 16);
        this->label_Port->TabIndex = 1;
        this->label_Port->Text = L"Port:";

```



```

//
// groupBox_Acc
//
this->groupBox_Acc->Controls->Add(this->label_time_acc);
this->groupBox_Acc->Controls->Add(this->label_Acc_Z);
this->groupBox_Acc->Controls->Add(this->label_Acc_Y);
this->groupBox_Acc->Controls->Add(this->label_Acc_X);
this->groupBox_Acc->Controls->Add(this->label_Acc_Z_Info);
this->groupBox_Acc->Controls->Add(this->label_Acc_Y_Info);
this->groupBox_Acc->Controls->Add(this->label_Acc_X_Info);
this->groupBox_Acc->Location = System::Drawing::Point(8, 116);
this->groupBox_Acc->Margin = System::Windows::Forms::Padding(3,
2, 3, 2);

this->groupBox_Acc->Name = L"groupBox_Acc";
this->groupBox_Acc->Padding =
System::Windows::Forms::Padding(3, 2, 3, 2);

this->groupBox_Acc->Size = System::Drawing::Size(101, 137);
this->groupBox_Acc->TabIndex = 1;
this->groupBox_Acc->TabStop = false;
this->groupBox_Acc->Text = L"Acc";
//
// label_time_acc
//
this->label_time_acc->AutoSize = true;
this->label_time_acc->Location = System::Drawing::Point(7, 118);
this->label_time_acc->Name = L"label_time_acc";
this->label_time_acc->Size = System::Drawing::Size(14, 16);
this->label_time_acc->TabIndex = 6;
this->label_time_acc->Text = L"0";
//
// label_Acc_Z

```

```

//
this->label_Acc_Z->AutoSize = true;
this->label_Acc_Z->Location = System::Drawing::Point(23, 75);
this->label_Acc_Z->Name = L"label_Acc_Z";
this->label_Acc_Z->Size = System::Drawing::Size(14, 16);
this->label_Acc_Z->TabIndex = 5;
this->label_Acc_Z->Text = L"0";
//
// label_Acc_Y
//
this->label_Acc_Y->AutoSize = true;
this->label_Acc_Y->Location = System::Drawing::Point(23, 48);
this->label_Acc_Y->Name = L"label_Acc_Y";
this->label_Acc_Y->Size = System::Drawing::Size(14, 16);
this->label_Acc_Y->TabIndex = 4;
this->label_Acc_Y->Text = L"0";
//
// label_Acc_X
//
this->label_Acc_X->AutoSize = true;
this->label_Acc_X->Location = System::Drawing::Point(23, 20);
this->label_Acc_X->Name = L"label_Acc_X";
this->label_Acc_X->Size = System::Drawing::Size(14, 16);
this->label_Acc_X->TabIndex = 3;
this->label_Acc_X->Text = L"0";
//
// label_Acc_Z_Info
//
this->label_Acc_Z_Info->AutoSize = true;
this->label_Acc_Z_Info->Location = System::Drawing::Point(5, 75);
this->label_Acc_Z_Info->Name = L"label_Acc_Z_Info";

```

```

this->label_Acc_Z_Info->Size = System::Drawing::Size(21, 16);
this->label_Acc_Z_Info->TabIndex = 2;
this->label_Acc_Z_Info->Text = L"Z: ";
//
// label_Acc_Y_Info
//
this->label_Acc_Y_Info->AutoSize = true;
this->label_Acc_Y_Info->Location = System::Drawing::Point(4, 48);
this->label_Acc_Y_Info->Name = L"label_Acc_Y_Info";
this->label_Acc_Y_Info->Size = System::Drawing::Size(22, 16);
this->label_Acc_Y_Info->TabIndex = 1;
this->label_Acc_Y_Info->Text = L"Y: ";
//
// label_Acc_X_Info
//
this->label_Acc_X_Info->AutoSize = true;
this->label_Acc_X_Info->Location = System::Drawing::Point(5, 20);
this->label_Acc_X_Info->Name = L"label_Acc_X_Info";
this->label_Acc_X_Info->Size = System::Drawing::Size(21, 16);
this->label_Acc_X_Info->TabIndex = 0;
this->label_Acc_X_Info->Text = L"X: ";
//
// groupBox_Gyr
//
this->groupBox_Gyr->Controls->Add(this->label_time_gyr);
this->groupBox_Gyr->Controls->Add(this->label_Gyr_Z);
this->groupBox_Gyr->Controls->Add(this->label_Gyr_Y);
this->groupBox_Gyr->Controls->Add(this->label_Gyr_X);
this->groupBox_Gyr->Controls->Add(this->label_Gyr_Z_Info);
this->groupBox_Gyr->Controls->Add(this->label_Gyr_Y_Info);
this->groupBox_Gyr->Controls->Add(this->label_Gyr_X_Info);

```

```

this->groupBox_Gyr->Location = System::Drawing::Point(123, 116);
this->groupBox_Gyr->Margin = System::Windows::Forms::Padding(3,
2, 3, 2);

this->groupBox_Gyr->Name = L"groupBox_Gyr";
this->groupBox_Gyr->Padding =
System::Windows::Forms::Padding(3, 2, 3, 2);

this->groupBox_Gyr->Size = System::Drawing::Size(96, 137);
this->groupBox_Gyr->TabIndex = 6;
this->groupBox_Gyr->TabStop = false;
this->groupBox_Gyr->Text = L"Gyr";
//
// label_time_gyr
//
this->label_time_gyr->AutoSize = true;
this->label_time_gyr->Location = System::Drawing::Point(4, 118);
this->label_time_gyr->Name = L"label_time_gyr";
this->label_time_gyr->Size = System::Drawing::Size(14, 16);
this->label_time_gyr->TabIndex = 7;
this->label_time_gyr->Text = L"0";
//
// label_Gyr_Z
//
this->label_Gyr_Z->AutoSize = true;
this->label_Gyr_Z->Location = System::Drawing::Point(23, 75);
this->label_Gyr_Z->Name = L"label_Gyr_Z";
this->label_Gyr_Z->Size = System::Drawing::Size(14, 16);
this->label_Gyr_Z->TabIndex = 5;
this->label_Gyr_Z->Text = L"0";
//
// label_Gyr_Y
//

```

```

this->label_Gyr_Y->AutoSize = true;
this->label_Gyr_Y->Location = System::Drawing::Point(23, 48);
this->label_Gyr_Y->Name = L"label_Gyr_Y";
this->label_Gyr_Y->Size = System::Drawing::Size(14, 16);
this->label_Gyr_Y->TabIndex = 4;
this->label_Gyr_Y->Text = L"0";
//
// label_Gyr_X
//
this->label_Gyr_X->AutoSize = true;
this->label_Gyr_X->Location = System::Drawing::Point(23, 20);
this->label_Gyr_X->Name = L"label_Gyr_X";
this->label_Gyr_X->Size = System::Drawing::Size(14, 16);
this->label_Gyr_X->TabIndex = 3;
this->label_Gyr_X->Text = L"0";
//
// label_Gyr_Z_Info
//
this->label_Gyr_Z_Info->AutoSize = true;
this->label_Gyr_Z_Info->Location = System::Drawing::Point(5, 75);
this->label_Gyr_Z_Info->Name = L"label_Gyr_Z_Info";
this->label_Gyr_Z_Info->Size = System::Drawing::Size(21, 16);
this->label_Gyr_Z_Info->TabIndex = 2;
this->label_Gyr_Z_Info->Text = L"Z: ";
//
// label_Gyr_Y_Info
//
this->label_Gyr_Y_Info->AutoSize = true;
this->label_Gyr_Y_Info->Location = System::Drawing::Point(4, 48);
this->label_Gyr_Y_Info->Name = L"label_Gyr_Y_Info";
this->label_Gyr_Y_Info->Size = System::Drawing::Size(22, 16);

```

```

this->label_Gyr_Y_Info->TabIndex = 1;
this->label_Gyr_Y_Info->Text = L"Y: ";
//
// label_Gyr_X_Info
//
this->label_Gyr_X_Info->AutoSize = true;
this->label_Gyr_X_Info->Location = System::Drawing::Point(5, 20);
this->label_Gyr_X_Info->Name = L"label_Gyr_X_Info";
this->label_Gyr_X_Info->Size = System::Drawing::Size(21, 16);
this->label_Gyr_X_Info->TabIndex = 0;
this->label_Gyr_X_Info->Text = L"X: ";
//
// groupBox_Pos
//
this->groupBox_Pos->Controls->Add(this->label_time_range);
this->groupBox_Pos->Controls->Add(this->label_Pos_L);
this->groupBox_Pos->Controls->Add(this->label_Pos_Y);
this->groupBox_Pos->Controls->Add(this->label_Pos_X);
this->groupBox_Pos->Controls->Add(this->label_Pos_L_Info);
this->groupBox_Pos->Controls->Add(this->label_Pos_Y_Info);
this->groupBox_Pos->Controls->Add(this->label_Pos_X_Info);
this->groupBox_Pos->Location = System::Drawing::Point(236, 116);
this->groupBox_Pos->Margin = System::Windows::Forms::Padding(3,
2, 3, 2);

this->groupBox_Pos->Name = L"groupBox_Pos";
this->groupBox_Pos->Padding =
System::Windows::Forms::Padding(3, 2, 3, 2);
this->groupBox_Pos->Size = System::Drawing::Size(96, 137);
this->groupBox_Pos->TabIndex = 7;
this->groupBox_Pos->TabStop = false;
this->groupBox_Pos->Text = L"Pos";

```

```

//
// label_time_range
//
this->label_time_range->AutoSize = true;
this->label_time_range->Location = System::Drawing::Point(5, 118);
this->label_time_range->Name = L"label_time_range";
this->label_time_range->Size = System::Drawing::Size(14, 16);
this->label_time_range->TabIndex = 8;
this->label_time_range->Text = L"0";
//
// label_Pos_L
//
this->label_Pos_L->AutoSize = true;
this->label_Pos_L->Location = System::Drawing::Point(23, 75);
this->label_Pos_L->Name = L"label_Pos_L";
this->label_Pos_L->Size = System::Drawing::Size(14, 16);
this->label_Pos_L->TabIndex = 5;
this->label_Pos_L->Text = L"0";
//
// label_Pos_Y
//
this->label_Pos_Y->AutoSize = true;
this->label_Pos_Y->Location = System::Drawing::Point(23, 48);
this->label_Pos_Y->Name = L"label_Pos_Y";
this->label_Pos_Y->Size = System::Drawing::Size(14, 16);
this->label_Pos_Y->TabIndex = 4;
this->label_Pos_Y->Text = L"0";
//
// label_Pos_X
//
this->label_Pos_X->AutoSize = true;

```

```

this->label_Pos_X->Location = System::Drawing::Point(23, 20);
this->label_Pos_X->Name = L"label_Pos_X";
this->label_Pos_X->Size = System::Drawing::Size(14, 16);
this->label_Pos_X->TabIndex = 3;
this->label_Pos_X->Text = L"0";
//
// label_Pos_L_Info
//
this->label_Pos_L_Info->AutoSize = true;
this->label_Pos_L_Info->Location = System::Drawing::Point(5, 75);
this->label_Pos_L_Info->Name = L"label_Pos_L_Info";
this->label_Pos_L_Info->Size = System::Drawing::Size(17, 16);
this->label_Pos_L_Info->TabIndex = 2;
this->label_Pos_L_Info->Text = L"L:";
//
// label_Pos_Y_Info
//
this->label_Pos_Y_Info->AutoSize = true;
this->label_Pos_Y_Info->Location = System::Drawing::Point(4, 48);
this->label_Pos_Y_Info->Name = L"label_Pos_Y_Info";
this->label_Pos_Y_Info->Size = System::Drawing::Size(22, 16);
this->label_Pos_Y_Info->TabIndex = 1;
this->label_Pos_Y_Info->Text = L"Y: ";
//
// label_Pos_X_Info
//
this->label_Pos_X_Info->AutoSize = true;
this->label_Pos_X_Info->Location = System::Drawing::Point(5, 20);
this->label_Pos_X_Info->Name = L"label_Pos_X_Info";
this->label_Pos_X_Info->Size = System::Drawing::Size(21, 16);
this->label_Pos_X_Info->TabIndex = 0;

```



```

this->label_Pos_X_Info->Text = L"X: ";
//
// groupBox_Input
//
this->groupBox_Input->Controls->Add(this->label_Pow);
this->groupBox_Input->Controls->Add(this->button_RR);
this->groupBox_Input->Controls->Add(this->button_LL);
this->groupBox_Input->Controls->Add(this->button_MR);
this->groupBox_Input->Controls->Add(this->button_ML);
this->groupBox_Input->Controls->Add(this->button_DD);
this->groupBox_Input->Controls->Add(this->button_UU);
this->groupBox_Input->Controls->Add(this->trackBar_Power);
this->groupBox_Input->Location = System::Drawing::Point(8, 257);
this->groupBox_Input->Margin
System::Windows::Forms::Padding(3, 2, 3, 2);
this->groupBox_Input->Name = L"groupBox_Input";
this->groupBox_Input->Padding
System::Windows::Forms::Padding(3, 2, 3, 2);
this->groupBox_Input->Size = System::Drawing::Size(325, 345);
this->groupBox_Input->TabIndex = 8;
this->groupBox_Input->TabStop = false;
//
// label_Pow
//
this->label_Pow->AutoSize = true;
this->label_Pow->Location = System::Drawing::Point(137, 256);
this->label_Pow->Margin = System::Windows::Forms::Padding(4, 0,
4, 0);
this->label_Pow->Name = L"label_Pow";
this->label_Pow->Size = System::Drawing::Size(14, 16);
this->label_Pow->TabIndex = 10;

```

```

this->label_Pow->Text = L"0";
//
// button_RR
//
this->button_RR->Location = System::Drawing::Point(228, 165);
this->button_RR->Margin = System::Windows::Forms::Padding(3, 2,
3, 2);

this->button_RR->Name = L"button_RR";
this->button_RR->Size = System::Drawing::Size(77, 25);
this->button_RR->TabIndex = 9;
this->button_RR->Text = L"RR";
this->button_RR->UseVisualStyleBackColor = true;
this->button_RR->MouseDown += gcnew
System::Windows::Forms::EventHandler(this,
&FormMain::button_RR_MouseDown);
this->button_RR->MouseUp += gcnew
System::Windows::Forms::EventHandler(this,
&FormMain::button_RR_MouseUp);
//
// button_LL
//
this->button_LL->Location = System::Drawing::Point(8, 165);
this->button_LL->Margin = System::Windows::Forms::Padding(3, 2,
3, 2);

this->button_LL->Name = L"button_LL";
this->button_LL->Size = System::Drawing::Size(77, 25);
this->button_LL->TabIndex = 8;
this->button_LL->Text = L"LL";
this->button_LL->UseVisualStyleBackColor = true;

```

```

        this->button_LL->MouseDown           +=          gcnew
System::Windows::Forms::MouseEventHandler(this,
&FormMain::button_LL_MouseDown);

        this->button_LL->MouseUp             +=          gcnew
System::Windows::Forms::MouseEventHandler(this,
&FormMain::button_LL_MouseUp);

        //
        // button_MR
        //

        this->button_MR->Location = System::Drawing::Point(228, 37);
        this->button_MR->Margin  = System::Windows::Forms::Padding(3,
2, 3, 2);

        this->button_MR->Name = L"button_MR";
        this->button_MR->Size = System::Drawing::Size(77, 25);
        this->button_MR->TabIndex = 7;
        this->button_MR->Text = L"->";
        this->button_MR->UseVisualStyleBackColor = true;
        this->button_MR->MouseDown           +=          gcnew
System::Windows::Forms::MouseEventHandler(this,
&FormMain::button_MR_MouseDown);

        this->button_MR->MouseUp             +=          gcnew
System::Windows::Forms::MouseEventHandler(this,
&FormMain::button_MR_MouseUp);

        //
        // button_ML
        //

        this->button_ML->Location = System::Drawing::Point(8, 37);
        this->button_ML->Margin  = System::Windows::Forms::Padding(3,
2, 3, 2);

        this->button_ML->Name = L"button_ML";
        this->button_ML->Size = System::Drawing::Size(77, 25);

```

```

        this->button_ML->TabIndex = 6;
        this->button_ML->Text = L"<-";
        this->button_ML->UseVisualStyleBackColor = true;
        this->button_ML->MouseDown           +=           gcnew
System::Windows::Forms::MouseEventHandler(this,
&FormMain::button_ML_MouseDown);
        this->button_ML->MouseUp             +=           gcnew
System::Windows::Forms::MouseEventHandler(this,
&FormMain::button_ML_MouseUp);
        //
        // button_DD
        //
        this->button_DD->Location = System::Drawing::Point(107, 308);
        this->button_DD->Margin = System::Windows::Forms::Padding(3, 2,
3, 2);

        this->button_DD->Name = L"button_DD";
        this->button_DD->Size = System::Drawing::Size(77, 25);
        this->button_DD->TabIndex = 5;
        this->button_DD->Text = L"DD";
        this->button_DD->UseVisualStyleBackColor = true;
        this->button_DD->MouseDown           +=           gcnew
System::Windows::Forms::MouseEventHandler(this,
&FormMain::button_DD_MouseDown);
        this->button_DD->MouseUp             +=           gcnew
System::Windows::Forms::MouseEventHandler(this,
&FormMain::button_DD_MouseUp);
        //
        // button_UU
        //
        this->button_UU->Location = System::Drawing::Point(115, 37);

```

```

this->button_UU->Margin = System::Windows::Forms::Padding(3, 2,
3, 2);

this->button_UU->Name = L"button_UU";
this->button_UU->Size = System::Drawing::Size(77, 25);
this->button_UU->TabIndex = 4;
this->button_UU->Text = L"UU";
this->button_UU->UseVisualStyleBackColor = true;
this->button_UU->MouseDown += gcnew
System::Windows::Forms::EventHandler(this,
&FormMain::button_UU_MouseDown);
this->button_UU->MouseUp += gcnew
System::Windows::Forms::EventHandler(this,
&FormMain::button_UU_MouseUp);
//
// trackBar_Power
//
this->trackBar_Power->Location = System::Drawing::Point(132, 85);
this->trackBar_Power->Margin =
System::Windows::Forms::Padding(3, 2, 3, 2);
this->trackBar_Power->Maximum = 100;
this->trackBar_Power->Name = L"trackBar_Power";
this->trackBar_Power->Orientation =
System::Windows::Forms::Orientation::Vertical;
this->trackBar_Power->RightToLeft =
System::Windows::Forms::RightToLeft::No;
this->trackBar_Power->Size = System::Drawing::Size(56, 169);
this->trackBar_Power->TabIndex = 0;
this->trackBar_Power->Scroll += gcnew System::EventHandler(this,
&FormMain::trackBar_Power_Scroll);
//
// timer1

```

```

//
this->timer1->Enabled = true;
this->timer1->Interval = 10;
this->timer1->Tick += gcnew System::EventHandler(this,
&FormMain::timer1_Tick);
//
// groupBox1
//
this->groupBox1->Controls->Add(this->label6);
this->groupBox1->Controls->Add(this->label5);
this->groupBox1->Controls->Add(this->label4);
this->groupBox1->Controls->Add(this->label3);
this->groupBox1->Controls->Add(this->label2);
this->groupBox1->Controls->Add(this->label7);
this->groupBox1->Controls->Add(this->label1);
this->groupBox1->Controls->Add(this->button1);
this->groupBox1->Location = System::Drawing::Point(338, 110);
this->groupBox1->Name = L"groupBox1";
this->groupBox1->Size = System::Drawing::Size(183, 169);
this->groupBox1->TabIndex = 9;
this->groupBox1->TabStop = false;
this->groupBox1->Text = L"groupBox1";
//
// label6
//
this->label6->AutoSize = true;
this->label6->Location = System::Drawing::Point(86, 81);
this->label6->Name = L"label6";
this->label6->Size = System::Drawing::Size(44, 16);
this->label6->TabIndex = 6;
this->label6->Text = L"label6";

```

```

//
// label5
//
this->label5->AutoSize = true;
this->label5->Location = System::Drawing::Point(86, 54);
this->label5->Name = L"label5";
this->label5->Size = System::Drawing::Size(44, 16);
this->label5->TabIndex = 5;
this->label5->Text = L"label5";
//
// label4
//
this->label4->AutoSize = true;
this->label4->Location = System::Drawing::Point(86, 27);
this->label4->Name = L"label4";
this->label4->Size = System::Drawing::Size(44, 16);
this->label4->TabIndex = 4;
this->label4->Text = L"label4";
//
// label3
//
this->label3->AutoSize = true;
this->label3->Location = System::Drawing::Point(42, 81);
this->label3->Name = L"label3";
this->label3->Size = System::Drawing::Size(34, 16);
this->label3->TabIndex = 3;
this->label3->Text = L"yaw:";
this->label3->TextAlign =
System::Drawing::ContentAlignment::TopCenter;
//
// label2

```

```

//
this->label2->AutoSize = true;
this->label2->Location = System::Drawing::Point(42, 54);
this->label2->Name = L"label2";
this->label2->Size = System::Drawing::Size(28, 16);
this->label2->TabIndex = 2;
this->label2->Text = L"roll:";
//
// label1
//
this->label1->AutoSize = true;
this->label1->Location = System::Drawing::Point(42, 27);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(38, 16);
this->label1->TabIndex = 1;
this->label1->Text = L"pitch:";
//
// button1
//
this->button1->Location = System::Drawing::Point(9, 100);
this->button1->Name = L"button1";
this->button1->Size = System::Drawing::Size(121, 23);
this->button1->TabIndex = 0;
this->button1->Text = L"connect ws";
this->button1->UseVisualStyleBackColor = true;
this->button1->Click += gcnew System::EventHandler(this,
&FormMain::button1_Click);
//
// label7
//
this->label7->AutoSize = true;

```



```

this->label7->Location = System::Drawing::Point(80, 127);
this->label7->Name = L"label7";
this->label7->Size = System::Drawing::Size(44, 16);
this->label7->TabIndex = 10;
this->label7->Text = L"label7";
//
// label8
//
this->label8->AutoSize = true;
this->label8->Location = System::Drawing::Point(418, 263);
this->label8->Name = L"label8";
this->label8->Size = System::Drawing::Size(44, 16);
this->label8->TabIndex = 11;
this->label8->Text = L"label8";
//
// label9
//
this->label9->AutoSize = true;
this->label9->Location = System::Drawing::Point(421, 283);
this->label9->Name = L"label9";
this->label9->Size = System::Drawing::Size(44, 16);
this->label9->TabIndex = 12;
this->label9->Text = L"label9";
//
// label10
//
this->label10->AutoSize = true;
this->label10->Location = System::Drawing::Point(424, 315);
this->label10->Name = L"label10";
this->label10->Size = System::Drawing::Size(51, 16);
this->label10->TabIndex = 13;

```

```

this->label10->Text = L"label10";
//
// label11
//
this->label11->AutoSize = true;
this->label11->Location = System::Drawing::Point(424, 356);
this->label11->Name = L"label11";
this->label11->Size = System::Drawing::Size(51, 16);
this->label11->TabIndex = 14;
this->label11->Text = L"label11";
//
// label12
//
this->label12->AutoSize = true;
this->label12->Location = System::Drawing::Point(427, 389);
this->label12->Name = L"label12";
this->label12->Size = System::Drawing::Size(51, 16);
this->label12->TabIndex = 15;
this->label12->Text = L"label12";
//
// FormMain
//
this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(533, 610);
this->Controls->Add(this->label12);
this->Controls->Add(this->label11);
this->Controls->Add(this->label10);
this->Controls->Add(this->label9);
this->Controls->Add(this->label8);

```

```

this->Controls->Add(this->groupBox1);
this->Controls->Add(this->groupBox_Input);
this->Controls->Add(this->groupBox_Pos);
this->Controls->Add(this->groupBox_Gyr);
this->Controls->Add(this->groupBox_Acc);
this->Controls->Add(this->groupBox_Server);
this->Margin = System::Windows::Forms::Padding(3, 2, 3, 2);
this->Name = L"FormMain";
this->Text = L"FormMain";
this->groupBox_Server->ResumeLayout(false);
this->groupBox_Server->PerformLayout();

```

```

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericUp
Down_Server_Port))->EndInit();

```

```

this->groupBox_Acc->ResumeLayout(false);
this->groupBox_Acc->PerformLayout();
this->groupBox_Gyr->ResumeLayout(false);
this->groupBox_Gyr->PerformLayout();
this->groupBox_Pos->ResumeLayout(false);
this->groupBox_Pos->PerformLayout();
this->groupBox_Input->ResumeLayout(false);
this->groupBox_Input->PerformLayout();

```

```

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->trackBar_Po
wer))->EndInit();

```

```

this->groupBox1->ResumeLayout(false);
this->groupBox1->PerformLayout();
this->ResumeLayout(false);
this->PerformLayout();

```

```

}

```

```
#pragma endregion
```

```
private: System::Void button_Connect_Click(System::Object^ sender,
System::EventArgs^ e)
{
    NetClient::ClientState done = netClient->Connect(
        textBox_Server_Addr->Text,
        (int)numericUpDown_Server_Port->Value,
        gcnw                                     NetClient::ClientCallback(this,
&FormMain::ConnectionCallback),
        gcnw                                     NetClient::ClientCallback(this,
&FormMain::ReceiveCallback));

    switch (done)
    {
    case NetClient::ClientState::Error:
        MessageBox::Show("Error connecting to server");
        break;
    case NetClient::ClientState::Connected:
        button_Connect->Text = "Disconnect";
        button_Connect->BackColor = Color::LimeGreen;
        break;
    case NetClient::ClientState::Stop:
        button_Connect->Text = "Connect";
        button_Connect->BackColor = Color::Transparent;
        break;
    }
}
```

```

private: System::Void trackBar_Power_Scroll(System::Object^ sender,
System::EventArgs^ e)
{
    float pow = (float)trackBar_Power->Value / 100.0f;
    label_Pow->Text = pow.ToString("F2");
    dataDrone->MotorUL = dataDrone->MotorUR = dataDrone->MotorDL
= dataDrone->MotorDR = pow;
}

```

```

private: System::Void button_UU_MouseDown(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e)
{
    dataDrone->MotorUL -= pow; dataDrone->MotorUR -= pow;
    dataDrone->MotorDL += pow; dataDrone->MotorDR += pow;
}

```

```

private: System::Void button_DD_MouseDown(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e)
{

    dataDrone->MotorUL += pow; dataDrone->MotorUR += pow;
    dataDrone->MotorDL -= pow; dataDrone->MotorDR -= pow;
}

```

```

private: System::Void button_LL_MouseDown(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e)
{
    dataDrone->MotorUL -= pow; dataDrone->MotorUR += pow;
    dataDrone->MotorDL -= pow; dataDrone->MotorDR += pow;
}

```

```

private: System::Void button_RR_MouseDown(System::Object^ sender,
System::Windows::Forms::EventArgs^ e)
{
    dataDrone->MotorUL += pow; dataDrone->MotorUR -= pow;
    dataDrone->MotorDL += pow; dataDrone->MotorDR -= pow;
}

```

```

private: System::Void button_ML_MouseDown(System::Object^ sender,
System::Windows::Forms::EventArgs^ e)
{
    dataDrone->MotorUL -= pow; dataDrone->MotorUR += pow;
    dataDrone->MotorDL += pow; dataDrone->MotorDR -= pow;
}

```

```

private: System::Void button_MR_MouseDown(System::Object^ sender,
System::Windows::Forms::EventArgs^ e)
{
    dataDrone->MotorUL += pow; dataDrone->MotorUR -= pow;
    dataDrone->MotorDL -= pow; dataDrone->MotorDR += pow;
}

```

```

private: System::Void button_UU_MouseUp(System::Object^ sender,
System::Windows::Forms::EventArgs^ e)
{
    trackBar_Power_Scroll(nullptr, nullptr); // Сброс мощности до
значения ползунка
}

```

```

private: System::Void button_DD_MouseUp(System::Object^ sender,
System::Windows::Forms::EventArgs^ e)
{

```

```

        trackBar_Power_Scroll(nullptr, nullptr);
    }

    private: System::Void button_LL_MouseUp(System::Object^ sender,
System::Windows::Forms::EventArgs^ e)
    {
        trackBar_Power_Scroll(nullptr, nullptr);
    }

    private: System::Void button_RR_MouseUp(System::Object^ sender,
System::Windows::Forms::EventArgs^ e)
    {
        trackBar_Power_Scroll(nullptr, nullptr);
    }

    private: System::Void button_ML_MouseUp(System::Object^ sender,
System::Windows::Forms::EventArgs^ e)
    {
        trackBar_Power_Scroll(nullptr, nullptr);
    }

    private: System::Void button_MR_MouseUp(System::Object^ sender,
System::Windows::Forms::EventArgs^ e)
    {
        trackBar_Power_Scroll(nullptr, nullptr);
    }

    private: System::Void timer1_Tick(System::Object^ sender,
System::EventArgs^ e)
    {
        label_Acc_X->Text = dataDrone->AccX.ToString("F3");
    }

```

```
label_Acc_Y->Text = dataDrone->AccY.ToString("F3");
label_Acc_Z->Text = dataDrone->AccZ.ToString("F3");
label_time_acc->Text = dataDrone->TimeAcc.ToString();
```

```
label_Gyr_X->Text = dataDrone->GyrX.ToString("F3");
label_Gyr_Y->Text = dataDrone->GyrY.ToString("F3");
label_Gyr_Z->Text = dataDrone->GyrZ.ToString("F3");
label_time_gyr->Text = dataDrone->TimeGyr.ToString();
```

```
label_Pos_X->Text = dataDrone->PosX.ToString("F3");
label_Pos_Y->Text = dataDrone->PosY.ToString("F3");
label_Pos_L->Text = dataDrone->LaserRange.ToString("F3");
label_time_range->Text = dataDrone->TimeRange.ToString();
```

```
label4->Text = dataDrone->pitch.ToString("F3");
label5->Text = dataDrone->roll.ToString("F3");
label6->Text = dataDrone->yaw.ToString("F3");
label7->Text = dataDrone->desPitch.ToString("F3");
label8->Text = dataDrone->desRoll.ToString("F3");
label9->Text = dataDrone->LaserRange.ToString("F3");
label10->Text = dataDrone->TrueRange.ToString("F3");
label11->Text = dataDrone->Bar.ToString("F3");
label12->Text = dataDrone->barHeight.ToString("F3");
// Отправка запроса серверу, как в C#
netClient->Send(dataDrone->SendRequest());
}
```

```
private: System::Void button1_Click(System::Object^ sender,
System::EventArgs^ e) {
    if (!wsClient->IsConnected)
```



```

{
    // Пытаемся подключиться к Node-серверу
    if (wsClient->Connect("ws://localhost:8080/"))
    {
        // ----- старт потока -----
        wsClient->_runTx = true;
        wsClient->_txThread = gcnew System::Threading::Thread(
            gcnew System::Threading::ParameterizedThreadStart(
                wsClient, &WsServer::TxLoop));
        wsClient->_txThread->IsBackground = true;
        wsClient->_txThread->Start(dataDrone); // <-- передаём Drone^
        // -----

        button1->Text = "disconnect ws";
        button1->BackColor = System::Drawing::Color::LimeGreen;
    }
    else
    {
        System::Windows::Forms::MessageBox::Show("Не удалось
подключиться");
    }
}
else
{
    wsClient->_runTx = false;
    if (wsClient->_txThread && wsClient->_txThread->IsAlive)
        wsClient->_txThread->Join(); // ждём завершения

    wsClient->Disconnect();
    button1->Text = "connect ws";
}

```

```

        button1->BackColor = System::Drawing::Color::Transparent;
    }
}

};
}
Joypad.h
//
joypad.h

```

```

#pragma once
#include <iostream>
#include <iomanip>
#include <Windows.h>      // <- для AllocConsole
#include <System.dll>

using namespace System;
using namespace System::IO::Ports;
using namespace System::Threading;

/*-----
Joypad (i-Bus → COM-порт)
-----*/

public ref class Joypad
{
public:
    /* событие — _не подписывайтесь_ → ничего в форме не вызовется */
    delegate void TickHandler(array<UInt16>^ ch);
    event TickHandler^ TickEvent;

```

```

Joypad()
{
    // ----- открываем консоль один раз -----
    static bool consoleReady = false;
    if (!consoleReady && ::AllocConsole())
    {
        FILE* fp;
        freopen_s(&fp, "CONOUT$", "w", stdout);
        std::cout << "  CH1  CH2  CH3  CH4  CH5  CH6\n";
        consoleReady = true;
    }
    // -----

    _sp = gcnew SerialPort();
    _sp->ReadTimeout = 200;
    _sp->ReadBufferSize = 256;
}

/* ----- запуск ----- */
bool Start(String^ port, int baud )
{
    if (_run) return true;
    try
    {
        _sp->PortName = port;
        _sp->BaudRate = baud;
        _sp->Parity = Parity::None;
        _sp->DataBits = 8;
        _sp->StopBits = StopBits::One;
        _sp->Open();
    }
}

```

```

        catch (Exception^ ex)
        {
            System::Diagnostics::Debug::WriteLine("Serial error: " +
ex->Message);
            return false;
        }

        _run = true;
        _thr = gcnew Thread(gcnew ThreadStart(this, &Joypad::RxLoop));
        _thr->IsBackground = true;
        _thr->Start();
        return true;
    }

    /* ----- остановка ----- */
    void Stop()
    {
        _run = false;
        if (_thr && _thr->IsAlive) _thr->Join();
        if (_sp->IsOpen) _sp->Close();
    }

    property array<UInt16>^ Channels { array<UInt16>^ get() { return
_ch; } }

private:
    // ----- поток приёма i-Bus -----
    void RxLoop()
    {
        array<Byte>^ buf = gcnew array<Byte>(64);
        array<Byte>^ pkt = gcnew array<Byte>(32);

```

```

while (_run)
{
    int read = 0;
    try { read = _sp->Read(buf, 0, buf->Length); }
    catch (TimeoutException^) { continue; }

    for (int i = 0; i < read; ++i)
    {
        _fifo[_head++] = buf[i];
        if (_head >= _fifo->Length) _head = 0;

        if (_fifo[_head] == 0x20 && _fifo[( _head + 1) & 0x7F] == 0x40)
        {
            for (int j = 0; j < 32; ++j)
                pkt[j] = _fifo[( _head + j) & 0x7F];

            if (!CheckPkt(pkt)) continue;
            ParseChannels(pkt);

            // ----- 1) печать в консоль -----
            std::cout << '\r';
            for (int k = 0; k < 6; ++k)
                std::cout << std::setw(6) << _ch[k] << ' ';
            std::cout << std::flush;
            // -----

            // ----- 2) необязательный вызов события ----
            TickEvent(_ch);          // raise
        }
    }
}

```

```

    }
}

```

```

bool CheckPkt(array<Byte>^ p)
{
    UInt16 sum = 0; for (int i = 0; i < 30; ++i) sum += p[i];
    sum = 0xFFFF - sum;
    return sum == (p[30] | p[31] << 8);
}

```

```

void ParseChannels(array<Byte>^ p)
{
    for (int i = 0; i < 14; ++i)
        _ch[i] = (UInt16)(p[2 + i * 2] | p[3 + i * 2] << 8);
}

```

```

/* ----- поля ----- */

```

```

SerialPort^ _sp;

```

```

Thread^ _thr;

```

```

bool    _run = false;

```

```

array<UInt16>^ _ch = gcnew array<UInt16>(14);

```

```

array<Byte>^ _fifo = gcnew array<Byte>(128);

```

```

int _head = 0;

```

```

};

```

NetClient.h

```

#pragma once

```

```

#include <Windows.h>

```

```

#include <vcclr.h>

```

```

#using <System.dll>
#using <System.Net.dll>

using namespace System;
using namespace System::Net;
using namespace System::Net::Sockets;

namespace DroneSimulator {

    public ref class NetClient
    {
    public:
        ref class ConnectData
        {
        public:
            bool Connect;
            Socket^ Server;

            ConnectData(bool connect, Socket^ server);
        };

        ref class ReceiveData
        {
        public:
            array<Byte>^ Buffer;
            int Size;
            Socket^ Server;

            ReceiveData(array<Byte>^ buffer, int size, Socket^ server);
        };
    };
}

```

```

private:
    ref class ServerData
    {
    public:
        literal int size = 1024;
        array<Byte>^ buffer = gcnew array<Byte>(size);
    };

    bool Connected;
    Socket^ ServerSocket;
    ServerData^ DataServer;

public:
    delegate void ClientCallback(Object^ o);

private:
    ClientCallback^ ConnectionCallback;
    ClientCallback^ ReceiveCallback;

public:
    NetClient(); // Добавлен конструктор

    enum class ClientState { Error, Connected, Stop };

    ClientState Connect(String^ Addr, int Port, ClientCallback^ Connection,
ClientCallback^ Receive);
    void Close();
    void Send(array<Byte>^ data);

private:

```



```

        void ReadCallback(IAsyncResult^ ar);
    };
}
NetClient.cpp
#include "NetClient.h"

namespace DroneSimulator {

    // Конструктор ConnectData
    NetClient::ConnectData::ConnectData(bool connect, Socket^ server)
    {
        Connect = connect;
        Server = server;
    }

    // Конструктор ReceiveData
    NetClient::ReceiveData::ReceiveData(array<Byte>^ buffer, int size,
Socket^ server)
    {
        Buffer = buffer;
        Size = size;
        Server = server;
    }

    // Конструктор NetClient
    NetClient::NetClient()
    {
        Connected = false;
        ServerSocket = nullptr;
        DataServer = gcnew ServerData(); // Инициализация DataServer
    }

```

```

// Реализация метода Connect
NetClient::ClientState NetClient::Connect(String^ Addr, int Port,
ClientCallback^ Connection, ClientCallback^ Receive)
{
    if (Connected)
    {
        Close();
        return ClientState::Stop;
    }

    ConnectionCallback = Connection;
    ReceiveCallback = Receive;

    IPEndPoint^ ep = gcnew IPEndPoint(IPAddress::Parse(Addr), Port);
    ServerSocket = gcnew Socket(AddressFamily::InterNetwork,
SocketType::Stream, ProtocolType::Tcp);

    try { if (ServerSocket) ServerSocket->Connect(ep); }
    catch (...) { ServerSocket->Close(); return ClientState::Error; }

    Connected = true;

    ConnectionCallback(gcnew ConnectData(true, ServerSocket));

    ReceiveData^ receiveData = gcnew ReceiveData(DataServer->buffer,
ServerData::size, ServerSocket);

    try { if (ServerSocket) ServerSocket->BeginReceive(DataServer->buffer,
0, ServerData::size, SocketFlags::None, gcnew AsyncCallback(this,
&NetClient::ReadCallback), receiveData); }

```

```

        catch (...) {}

        return ClientState::Connected;
    }

// Реализация метода Close
void NetClient::Close()
{
    try { if(ServerSocket)
ServerSocket->Shutdown(SocketShutdown::Both); }
        catch (...) {}
        if(ServerSocket) ServerSocket->Close();
        Connected = false;
    }

// Реализация метода Send
void NetClient::Send(array<Byte>^ data)
{
    if (ServerSocket != nullptr && Connected)
    {
        try { if (ServerSocket) ServerSocket->Send(data); }
        catch (...) {}
    }
}

// Реализация метода ReadCallback
void NetClient::ReadCallback(IAsyncResult^ ar)
{
    ReceiveData^ cd = (ReceiveData^)ar->AsyncState;
    if (cd == nullptr) return;

```

```

int bytes = 0;
try { bytes = ServerSocket->EndReceive(ar); }
catch (...) {}

```

```

if (bytes == 0)
{
    if (ServerSocket) ServerSocket->Close();
    Connected = false;

    if (ServerSocket != nullptr)
    {
        ConnectionCallback(gcnew ConnectData(false, nullptr));
    }

    return;
}

```

```

ReceiveCallback(gcnew ReceiveData(cd->Buffer, bytes, ServerSocket));

```

```

try { if (ServerSocket) ServerSocket->BeginReceive(cd->Buffer, 0,
ServerData::size, SocketFlags::None, gcnew AsyncCallback(this,
&NetClient::ReadCallback), cd); }
catch (...) {}
}
}

```

WsServer.h

```

#pragma once
#include <Windows.h>
using <System.dll>
using <System.Net.Http.dll>

```

```

#pragma pack(push, 1)
struct telemetry_native    // 8 × float = 32 байта
{
    float pitch, roll, yaw;    // ориентация
    float accX, accY, accZ;    // акселерометр
    float altRef;              // заданная высота
    float laser;               // фактическая высота / LiDAR
    float inX, inY, inZ;       // инерциальные значения
    float posX, posY;          // позиция по ГНС
    float calcPosX, calcPosY;  // посчитанная позиция
    float barHeight;           // высота по барометру
    float inHeight;            // высота инерциальная
};

#pragma pack(pop)

using namespace System;
using namespace System::Collections::Generic;
using namespace System::Net;
using namespace System::Net::WebSockets;
using namespace System::Threading;
using namespace System::Text;
using namespace DroneClient;    // ← поправьте, если у вас другое

```

```

public ref class WsServer    // имя можно оставить тем же
{
public:
    WsServer() : _connected(false) {}

```

```

/* ----- подключение ----- */

```

```

bool Connect(String^ uri)
{
    if (_connected) return true;           // уже подключены

    _ws = gcnew ClientWebSocket();
    _cts = gcnew CancellationTokenSource();

    try
    {
        // 1) запускаем асинхронное соединение
        System::Threading::Tasks::Task^ t =
            _ws->ConnectAsync(gcnew Uri(uri), _cts->Token);

        // 2) ждём завершения
        t->Wait();                          // здесь AggregateException «обёртывает»
        // все реальные ошибки
        _connected = true;
    }
    catch (AggregateException^ ag)
    {
        // сообщений может быть несколько – выводим главное
        Exception^ ex = ag->InnerExceptions->Count
            ? ag->InnerExceptions[0] : ag;

        System::String^ msg = ex->Message;

        // типовые причины:
        // • WebSocketException (ошибка DNS / connection refused / timeout)
        // • NotSupportedException (Windows 7: платформа без
WebSocket-клиента)
        // • InvalidOperationException (неверный URI)

```

```

        // → выводим в Debug и просто возвращаем false
        System::Diagnostics::Debug::WriteLine("Connect error: " + msg);
        _connected = false;
    }
    catch (Exception^ ex)      // на всякий случай – «прочие»
    {
        System::Diagnostics::Debug::WriteLine("Connect error: " +
ex->Message);
        _connected = false;
    }
    return _connected;
}

/* ----- отключение ----- */
void Disconnect()
{
    if (!_connected) return;

    try
    {
        _ws->CloseAsync(WebSocketCloseStatus::NormalClosure,
            "bye", CancellationToken::None)->Wait();
    }
    catch (Exception^) { /* игнор */ }

    _ws = nullptr;
    _cts = nullptr;
    _connected = false;
}

/* ----- быстрая отправка строки ----- */

```

```

bool SendString(String^ msg)
{
    if (!_connected) return false;

    array<Byte>^ bytes = Encoding::UTF8->GetBytes(msg);
    try
    {
        _ws->SendAsync(ArraySegment<Byte>(bytes),
            WebSocketMessageType::Text,
            true, CancellationToken::None)->Wait();
        return true;
    }
    catch (Exception^) { return false; }
}

/* ----- состояние ----- */
property bool IsConnected
{
    bool get() { return _connected; }
}

float x=0, y=0;
void WsServer::SendTelemetry(const Drone^ d)
{
    telemetry_native t = {
        d->pitch, d->roll, d->yaw,
        d->AccX, d->AccY, d->AccZ,
        d->altRef, d->LaserRange,
        d->inX, d->inY, d->inZ,
        d->PosX, d->PosY,

```



```

        d->posXcalc, d->posYcalc,
        d->barHeight, d->inHeight
    };

```

```

array<Byte>^ buf = gcnew array<Byte>(sizeof(t));
auto h = System::Runtime::InteropServices::GCHandle::Alloc(
    buf, System::Runtime::InteropServices::GCHandleType::Pinned);
memcpy(h.AddrOfPinnedObject().ToPointer(), &t, sizeof(t));
h.Free();

```

```

        _ws->SendAsync(System::ArraySegment<Byte>(buf),
            WebSocketMessageType::Binary, true,
            Threading::CancellationToken::None)->Wait();
    }

```

```

void WsServer::TxLoop(Object^ param)
{
    Drone^ d = safe_cast<Drone^>(param);
    const int PERIOD_MS = 20;

    while (_runTx && _connected)
    {
        SendTelemetry(d);           // ← bмecto SendAnglesBinary
        Threading::Thread::Sleep(PERIOD_MS);
    }
}

```

private:

```

    ClientWebSocket^ _ws;

```

```

CancellationTokenSource^ _cts;
bool _connected;

public:
    // --- рядом с тем, где уже лежит wsClient ---
    System::Threading::Thread^ _txThread = nullptr; // поток передатчик
    bool _runTx = false; // признак «живого» цикла
};

```

Orientation.h

```
#pragma once
```

```
struct Vec3 { float x, y, z; };
```

```
struct ORI
{
    float sinX, sinY, cosZ; // Earth's plane tilt
    float Pitch, Roll, Yaw; // Sovereign orientation (not Euler)
    float IneX, IneY, IneZ; // Inertial accelerations
};
```

```
ORI WorkAccGyroMag(const Vec3 acc, const Vec3 gyr, const Vec3 mag,
const float mag_shift, const float alpha, float period);
```

```
Vec3 RotateToZ(const Vec3 vec, bool Rev = false);
```

Orientation.cpp

```
#include "Orientation.h"
#include <math.h>
```

```

static const float PI = 3.14159265359f;
static const float TO_DEG = 180.0f / PI;
static const float TO_RAD = PI / 180.0f;

struct Quaternion
{
    float w, x, y, z;
};

static Quaternion qCurrent = { 1, 0, 0, 0 };

static const float period1 = 0.03f; // 100Hz

static bool isFirst = true;

inline void vecNormalize(Vec3& v)
{
    float n = sqrtf(v.x * v.x + v.y * v.y + v.z * v.z);
    if (n > 1e-9f)
    {
        v.x /= n;
        v.y /= n;
        v.z /= n;
    }
}

inline Vec3 vecCross(const Vec3& a, const Vec3& b)
{
    return
    {
        a.y * b.z - a.z * b.y,

```

```

        a.z * b.x - a.x * b.z,
        a.x * b.y - a.y * b.x
    };
}

```

```

inline void normalizeQuaternion(Quaternion& q)
{
    float norm = sqrtf(q.w * q.w + q.x * q.x + q.y * q.y + q.z * q.z);
    if (norm > 1e-12f)
    {
        q.w /= norm;
        q.x /= norm;
        q.y /= norm;
        q.z /= norm;
    }
}

```

```

static Quaternion quaternionMultiply(const Quaternion& q1, const
Quaternion& q2)
{
    Quaternion r;
    r.w = q1.w * q2.w - q1.x * q2.x - q1.y * q2.y - q1.z * q2.z;
    r.x = q1.w * q2.x + q1.x * q2.w + q1.y * q2.z - q1.z * q2.y;
    r.y = q1.w * q2.y - q1.x * q2.z + q1.y * q2.w + q1.z * q2.x;
    r.z = q1.w * q2.z + q1.x * q2.y - q1.y * q2.x + q1.z * q2.w;
    return r;
}

```

```

static Quaternion rotateVectorByQuaternion(const Quaternion& q, const
Vec3& In)
{

```

```
Quaternion r = quaternionMultiply(quaternionMultiply(q, Quaternion{ 0,
In.x, In.y, In.z }), Quaternion{ q.w, -q.x, -q.y, -q.z });
```

```
    return r;
}
```

```
static Vec3 backRotateVectorByQuaternion(const Quaternion& q, const
Vec3& In)
```

```
{
    Quaternion r = quaternionMultiply(quaternionMultiply(Quaternion{ q.w,
-q.x, -q.y, -q.z }, Quaternion{ 0, In.x, In.y, In.z }), q);
```

```
    return { r.x, r.y, r.z };
}
```

```
static Quaternion backRotateVectorByQuaternion2(const Quaternion& q,
const Quaternion& In)
```

```
{
    Quaternion r = quaternionMultiply(quaternionMultiply(Quaternion{ q.w,
-q.x, -q.y, -q.z }, In), q);
```

```
    return r;
}
```

```
static Quaternion createYawQuaternion(float angle)
```

```
{
    Quaternion q;
```

```
    q.w = cosf(angle);
```

```
    q.x = 0.0f;
```

```
    q.y = 0.0f;
```

```

    q.z = sinf(angle);

    return q;
}

static Quaternion AccQuaternion(Quaternion& current, Vec3 a, float gravity,
Vec3 xyz, float& error)
{
    float acc = sqrtf(a.x * a.x + a.y * a.y + a.z * a.z);
    if (acc > (1.0f + gravity) || acc < (1.0f - gravity)) return current;

    vecNormalize(a);

    float x = a.x - xyz.x, y = a.y - xyz.y, z = a.z - xyz.z;

    error = sqrtf(x * x + y * y + z * z) / 10.0f;

    Vec3 g{ 0, 0, 1 };
    Vec3 axis = vecCross(g, a);
    float w = 1 + (g.x * a.x + g.y * a.y + g.z * a.z);
    Quaternion q = { w, axis.x, axis.y, axis.z };
    normalizeQuaternion(q);

    Quaternion qYaw{ current.w, 0, 0, current.z };

    return quaternionMultiply(q, qYaw); // Восстановить оборот по Z
}

static Quaternion GyroQuaternion(Quaternion& current, float wx, float wy,
float wz)

```

```

{
    Quaternion Mapp = current;
    Quaternion Spd{ 0, wx, wy, wz };
    Quaternion aq = quaternionMultiply(Spd, Mapp);

    Mapp.w -= 0.5f * aq.w;
    Mapp.x -= 0.5f * aq.x;
    Mapp.y -= 0.5f * aq.y;
    Mapp.z -= 0.5f * aq.z;

    normalizeQuaternion(Mapp);
    return Mapp;
}

static Quaternion nlerp(const Quaternion& q1, const Quaternion& q2, float
alpha)
{
    float dot = q1.w * q2.w + q1.x * q2.x + q1.y * q2.y + q1.z * q2.z;

    Quaternion q2_ = q2;
    if (dot < 0)
    {
        q2_.w = -q2.w;
        q2_.x = -q2.x;
        q2_.y = -q2.y;
        q2_.z = -q2.z;
    }

    Quaternion r;
    r.w = (1.0f - alpha) * q1.w + alpha * q2_.w;

```

```

    r.x = (1.0f - alpha) * q1.x + alpha * q2_.x;
    r.y = (1.0f - alpha) * q1.y + alpha * q2_.y;
    r.z = (1.0f - alpha) * q1.z + alpha * q2_.z;

    normalizeQuaternion(r);

    return r;
}

```

```

inline float GetAngle(float a1, float a2, float az)
{
    if (a2 == 0.0f && az == 0.0f)
    {
        if (a1 > 0.0f) return 90.0f;
        if (a1 < 0.0f) return -90.0f;
        return 0.0f;
    }

    return atanf(a1 / sqrtf(a2 * a2 + az * az)) * TO_DEG;
}

```

```

static Vec3 quaternionToPitchRollYaw(const Quaternion& q, Vec3&
cosXYZ)
{
    Quaternion pry = rotateVectorByQuaternion(q, { 0, 0, 1 });

    float yaw = 2.0f * atan2f(q.z, q.w) * TO_DEG;
    if (yaw < 0.0f) yaw = 360.0f + yaw;

    cosXYZ = { pry.x, pry.y, pry.z };
}

```



```

return // Sovereign orientation
{
    GetAngle(pry.y, pry.x, pry.z), // Pitch
    GetAngle(-pry.x, pry.y, pry.z), // Roll
    yaw // Yaw
};
}

static void addMagnetometer(Quaternion& q, Vec3 mag, float alpha, const float
shift)
{
    static Quaternion yq = createYawQuaternion(shift * TO_RAD);

    vecNormalize(mag);

    Quaternion mQ = { 0, mag.x, mag.y, mag.z };

    Quaternion mW = backRotateVectorByQuaternion2(q, mQ);

    mW = quaternionMultiply(mW, yq); // Shifting the axes to the true north

    float gamma = mW.x * mW.x + mW.y * mW.y;
    float beta = sqrtf(gamma + mW.x * sqrtf(gamma));

    Quaternion mD
    {
        beta / (sqrtf(2.0f * gamma)),
        0.0f,
        0.0f,
        mW.y / (sqrtf(2.0f) * beta),
    };
};

```

```
mD.w = (1.0f - alpha) + alpha * mD.w;
```

```
mD.z = alpha * mD.z;
```

```
if (mD.w != mD.w || mD.x != mD.x || mD.y != mD.y || mD.z != mD.z) return;
```

```
q = quaternionMultiply(q, mD);
```

```
normalizeQuaternion(q);
```

```
}
```

```
ORI WorkAccGyroMag(const Vec3 acc, const Vec3 gyr, const Vec3 mag,  
const float mag_shift, const float alpha, float period)
```

```
{
```

```
float wx = gyr.x * 0.015f * TO_RAD * period;
```

```
float wy = gyr.y * 0.015f * TO_RAD * period;
```

```
float wz = gyr.z * 0.015f * TO_RAD * period;
```

```
Vec3 aB = acc;
```

```
Vec3 cosXYZ;
```

```
Vec3 pry = quaternionToPitchRollYaw(qCurrent, cosXYZ);
```

```
static float error = 0;
```

```
Quaternion qAcc = AccQuaternion(qCurrent, aB, 0.05f, cosXYZ, error); //
```

```
Tolerance for gravity deviation 5%
```

```
qCurrent = GyroQuaternion(qCurrent, wx, wy, wz);
```

```
if (error > 0.01f) error = 0.01f;
```

```
if (error < alpha) error = alpha;
```

```
Quaternion qFused = nlerp(qCurrent, qAcc, error);
```

```
if (isFirst)
```

```
{
```

```
    qFused = qAcc;
```

```
    isFirst = false;
```

```
}
```

```
qCurrent = qFused;
```

```
//addMagnetometer(qCurrent, mag, alpha, mag_shift);
```

```
Vec3 ine = backRotateVectorByQuaternion(qCurrent, aB);
```

```
return
```

```
{
```

```
    cosXYZ.x, cosXYZ.y, cosXYZ.z,
```

```
    pry.x, pry.y, pry.z,
```

```
    ine.x, ine.y, ine.z,
```

```
};
```

```
}
```

```
Vec3 RotateToZ(const Vec3 vec, bool Rev)
```

```
{
```

```
    Quaternion v{ 0, vec.x, vec.y, vec.z };
```

```
    Quaternion q = { qCurrent.w, 0, 0, Rev ? -qCurrent.z : qCurrent.z };
```

```
    normalizeQuaternion(q);
```

```
    q = quaternionMultiply(quaternionMultiply(v, q), q); // Восстановить  
оборот по Z
```

```

    return { q.x, q.y, q.z };
}

```

BarometricFilter.h

```
#pragma once
```

```
#include <vector>
```

```
extern int futureWindow;
```

```
extern std::vector<float> futureBarValues;
```

```
extern std::vector<float> PastBarValues;
```

```
float calculateHeight(float bar, float temp);
```

```
float      computeWeightWindowBar(std::vector<float>      futureBar,
std::vector<float> PastBar);
```

```
float computeBarHeight(float currentBar, float temperature);
```

BarometricFilter.cpp

```
#include "BarometricFilter.h"
```

```
#include <cmath>
```

```
const float R = 8.314f;
```

```
const float M = 0.029f;
```

```
const float g = 9.80665f;
```

```
int futureWindow = 15;
```

```
std::vector<float> futureBarValues;
```

```
std::vector<float> PastBarValues(futureWindow, 0);
```

```
bool fl = true;
```

```
float calculateHeight(float bar, float temp)
```

```
{
```

```

static double firstBar=0;
if (fl && bar != 0)
{
    firstBar = bar;
    fl = false;
}

return (R * (temp + 273) / (M * g)) * log(firstBar / bar);
}

float      computeWeightWindowBar(std::vector<float>      futureBar,
std::vector<float> PastBar)
{
    float sumWeight = 0, computeValue = 0;
    for (int i = 1; i <= futureBar.size(); ++i) sumWeight += i;
    for (int i = 1; i <= PastBar.size(); ++i) sumWeight += i;

    for (int i = 0; i < futureBar.size(); ++i)
        computeValue += ((futureBar.size() - i) / sumWeight) *
futureBar[i];

    for (int i = 0; i < PastBar.size(); ++i)
        computeValue += ((i + 1) / sumWeight) * PastBar[i];

    return computeValue;
}

float computeBarHeight(float currentBar, float temperature)
{
    float ClearBarHeight = calculateHeight(currentBar, temperature);

```

```

    if (futureBarValues.size() < futureWindow)
    {
        futureBarValues.push_back(ClearBarHeight);
        return 0.0f;
    }
    else
    {
        PastBarValues.erase(PastBarValues.begin());
        PastBarValues.push_back(futureBarValues.front());

        futureBarValues.erase(futureBarValues.begin());
        futureBarValues.push_back(ClearBarHeight);

        return      computeWeightWindowBar(futureBarValues,
PastBarValues);
    }
}

```

ПРИЛОЖЕНИЕ Е. КОД ВЕБ-ЧАСТИ

NodeJS

Server.js

```
import path from 'node:path';
import express from 'express';
import { fileURLToPath } from 'node:url';
import { WebSocketServer } from 'ws';
import { Server as IOServer } from 'socket.io';

const __dirname = path.dirname(fileURLToPath(import.meta.url));
const app = express();

// Настройка HTTP-сервера
const http = app.listen(3000, () => console.log('  http://localhost:3000'));

// Настройка Socket.IO
const io = new IOServer(http, { cors: { origin: '*' } });

// Переменная для хранения последнего полученного состояния
let last = { x: 0 };

// Настройка WebSocket сервера
const wss = new WebSocketServer({ port: 8080 }, () =>
  console.log('WS ждёт на ws://localhost:8080')
);

// Обработчик подключений для WebSocket
wss.on('connection', ws => {
  ws.on('message', buf => {
```

```

if (buf.length !== 68) return; // Проверка размера данных (8 × float32)

// Функция для чтения данных из буфера
const f = o => buf.readFloatLE(o);

// Обновление данных с прибавлением для 'x'
last = {
  x      : +(last.x + 0.02).toFixed(2), // Инкремент для x (например)
  pitch  : f(0),
  roll   : f(4),
  yaw    : f(8),
  accX   : f(12),
  accY   : f(16),
  accZ   : f(20),
  altRef : f(24),
  laser  : f(28),
  inX    : f(32),
  inY    : f(36),
  inZ    : f(40),
  PosX   : f(44),
  PosY   : f(48),
  calcPosX: f(52),
  calcPosY: f(56),
  barHeigt: f(60),
  inHeight: f(64)
};

// Логирование для отладки
console.log(last);

// Отправка данных через Socket.IO на клиентскую часть (Vue)

```



```

    io.emit('telemetry', last); // <-- Отправляем данные на Vue
  });
});

```

VueJS

WebSocketComponent.vue

```

<template>
  <div class="dashboard">
    <h1>Телеметрия</h1>

    <TelemetryChart
      title="Ориентация (Крен, Тангаж, Рысканье)"
      svgId="anglesChart"
      :dataSeries="[telemetry.pitch, telemetry.roll, telemetry.yaw]"
      :colors="['#f5a623', '#50e3c2', '#4a90e2']"
      :yRange="[-25, 25]"
      xUnit="время, с"
      yUnit="градусы"
      :lineLabels="['Крен', 'Тангаж', 'Рысканье']"
    />

    <TelemetryChart
      title="Высота"
      svgId="altChart"
      :dataSeries="[telemetry.altRef,    telemetry.laser,    telemetry.barHeigt,
telemetry.inHeight]"
      :colors="['#f8e71c', '#b8e986', '#4a90e2','red']"
      :yRange="[0, 3]"
      xUnit="время, с"
      yUnit="метры"

```

```

      :lineLabels=["'Зафиксированная высота', 'Лазер', 'Барометр',
'Инерциальная высота']"
    />

```

```

<TelemetryChart
  title="Ускорения и ИНС"
  svgId="motionChart"
  :dataSeries=["
    telemetry.inX, telemetry.inY, telemetry.inZ,
    telemetry.accX, telemetry.accY, telemetry.accZ
  "]"
  :colors=["'#f5a623', '#f8e71c', '#7ed321', '#4a90e2', '#50e3c2', '#9013fe']"
  :yRange="[-2, 2]"
  xUnit="время, с"
  yUnit="g"
  :lineLabels=["'ИНС X', 'ИНС Y', 'ИНС Z', 'Acc X', 'Acc Y', 'Acc Z']"
/>

```

```

<PositionPlot
  :posX="last.posX"
  :posY="last.posY"
  :calcX="last.calcX"
  :calcY="last.calcY"
  :xRange="[-5, 5]"
  :yRange="[-5, 5]"
/>

```

```

</div>

```

```

</template>

```

```

<script>

```

```

import { io } from 'socket.io-client';

```

```

import TelemetryChart from './TelemetryChart.vue';
import PositionPlot from './PositionPlot.vue';

export default {
  components: { TelemetryChart, PositionPlot },
  data() {
    return {
      telemetry: {
        pitch: [], roll: [], yaw: [],
        accX: [], accY: [], accZ: [],
        altRef: [], laser: [],
        inX: [], inY: [], inZ: [],
        PosX: [], PosY: [],
        calcPosX: [], calcPosY: [],
        barHeigt: [], inHeight: []
      },
      last: {
        posX: 0, posY: 0,
        calcX: 0, calcY: 0
      }
    };
  },
  mounted() {
    const socket = io('http://localhost:3000');
    const MAX = 400;
    const push = (arr, v) => { if (arr.length >= MAX) arr.shift(); arr.push(v); };

    socket.on('telemetry', d => {
      this.last.posX = d.PosX;
      this.last.posY = d.PosY;
      this.last.calcX = d.calcPosX;

```

```

    this.last.calcY = d.calcPosY;
    push(this.telemetry.pitch, d.pitch);
    push(this.telemetry.roll, d.roll);
    push(this.telemetry.yaw, d.yaw);
    push(this.telemetry.accX, d.accX);
    push(this.telemetry.accY, d.accY);
    push(this.telemetry.accZ, d.accZ);
    push(this.telemetry.altRef, d.altRef);
    push(this.telemetry.laser, d.laser);
    push(this.telemetry.inX, d.inX);
    push(this.telemetry.inY, d.inY);
    push(this.telemetry.inZ, d.inZ);
    push(this.telemetry.barHeigt, d.barHeigt);
    push(this.telemetry.inHeight, d.inHeight);
  });
}
};
</script>

```

```
<style>
```

```
@import
```

```
url('https://fonts.googleapis.com/css2?family=Manrope:wght@500;700&display=s
wap');
```

```

.dashboard {
  background: linear-gradient(to bottom right, #fcf8f3, #e0f0ff);
  padding: 32px;
  font-family: 'Manrope', sans-serif;
  color: #1a1a1a;
}

```

```
h1 {  
  color: #2c3e50;  
  font-size: 32px;  
  margin-bottom: 30px;  
  text-align: center;  
}
```

```
h2 {  
  color: #34495e;  
  font-size: 20px;  
  margin-bottom: 10px;  
}
```

```
input[type="range"] {  
  -webkit-appearance: none;  
  height: 6px;  
  background: #d0dce5;  
  border-radius: 4px;  
  outline: none;  
  transition: background 0.3s;  
}
```

```
input[type="range"]::-webkit-slider-thumb {  
  -webkit-appearance: none;  
  width: 16px;  
  height: 16px;  
  border-radius: 50%;  
  background: #4a90e2;  
  cursor: pointer;  
  box-shadow: 0 0 4px rgba(0,0,0,0.2);  
}
```

```
input[type="checkbox"] {
    appearance: none;
    width: 20px;
    height: 20px;
    border-radius: 6px;
    background: #fff;
    border: 2px solid #4a90e2;
    cursor: pointer;
    position: relative;
    vertical-align: middle;
    transition: all 0.2s ease;
    margin-right: 8px;
}

input[type="checkbox"]:checked {
    background-color: #4a90e2;
}

input[type="checkbox"]:checked::after {
    content: " ";
    position: absolute;
    top: 3px;
    left: 6px;
    width: 5px;
    height: 10px;
    border: solid white;
    border-width: 0 2px 2px 0;
    transform: rotate(45deg);
}

input[type="number"] {
    border: 1px solid #ccc;
    padding: 6px 10px;
```

```

border-radius: 8px;
width: 80px;
font-family: 'Manrope', sans-serif;
background: #fdfdfd;
box-shadow: inset 0 1px 2px rgba(0,0,0,0.05);
}

button {
background: linear-gradient(to right, #4a90e2, #357ab8);
color: #fff;
border: none;
padding: 8px 16px;
border-radius: 8px;
cursor: pointer;
transition: background 0.3s ease;
font-family: 'Manrope', sans-serif;
font-weight: 600;
box-shadow: 0 2px 6px rgba(0,0,0,0.1);
}

button:hover {
background: linear-gradient(to right, #357ab8, #2c6192);
}

```

</style>

PositionPlot.vue

```

<template>
<div>
<h2>Позиция</h2>

<!-- Панель управления -->

```

```

<div style="margin-bottom: 10px;">
  <label>Частота (Гц):</label>
  <input type="range" min="1" max="50" v-model="frequency" />
  <span>{{ frequency }} Гц</span>

  <label style="margin-left: 15px;"><input type="checkbox" v-
model="showPos" /> Позиция ГНС</label>
  <label><input type="checkbox" v-model="showCalc" />Рассчитанная
позиция</label>
  <label><input type="checkbox" v-model="showGrid" /> Показать
сетку</label>
</div>

<div style="margin-bottom: 10px;">
  <label>X min:</label>
  <input type="number" v-model.number="xMin" />
  <label>X max:</label>
  <input type="number" v-model.number="xMax" style="margin-right:
20px;" />
  <label>Y min:</label>
  <input type="number" v-model.number="yMin" />
  <label>Y max:</label>
  <input type="number" v-model.number="yMax" />
</div>

<svg ref="svg" width="100%" height="400"></svg>
</div>
</template>

<script>
import * as d3 from 'd3';

```



```

export default {
  props: {
    posX: { type: Number, required: true },
    posY: { type: Number, required: true },
    calcX: { type: Number, required: true },
    calcY: { type: Number, required: true }
  },
  data() {
    return {
      svg: null,
      g: null,
      posCircle: null,
      calcCircle: null,
      xScale: null,
      yScale: null,
      frequency: 10,
      intervalId: null,
      showPos: true,
      showCalc: true,
      showGrid: true,

      // границы координатной сетки
      xMin: -10,
      xMax: 10,
      yMin: -10,
      yMax: 10
    };
  },
  watch: {
    xMin() { this.updateScales(); },

```

```

    xMax() { this.updateScales(); },
    yMin() { this.updateScales(); },
    yMax() { this.updateScales(); },
    showGrid() { this.drawGrid(); },
    frequency() { this.restartLoop(); }
  },
  mounted() {
    this.initPlot();
    this.restartLoop();
  },
  beforeUnmount() {
    clearInterval(this.intervalId);
  },
  methods: {
    initPlot() {
      const svgEl = d3.select(this.$refs.svg);
      const { width, height } = svgEl.node().getBoundingClientRect();
      const margin = { top: 20, right: 20, bottom: 30, left: 40 };
      const w = width - margin.left - margin.right;
      const h = height - margin.top - margin.bottom;

      this.xScale = d3.scaleLinear().domain([this.xMin, this.xMax]).range([0,
w]);

      this.yScale = d3.scaleLinear().domain([this.yMin, this.yMax]).range([h,
0]);

      this.g = svgEl.append('g').attr('transform',
`translate(${margin.left},${margin.top})`);

      this.plotW = w;
      this.plotH = h;

```

```

this.margin = margin;

this.xAxisG = this.g.append('g')
  .attr('transform', `translate(0,${h})`)
  .call(d3.axisBottom(this.xScale));

this.yAxisG = this.g.append('g')
  .call(d3.axisLeft(this.yScale));

// сетка
this.gridLayer = this.g.append('g').attr('class', 'grid');

// точки
this.posCircle = this.g.append('circle').attr('r', 8).attr('fill', '#00f');
this.calcCircle = this.g.append('circle').attr('r', 8).attr('fill', '#f00');

this.updatePoints();
this.drawGrid();
},
drawGrid() {
  this.gridLayer.selectAll('*').remove();
  if (!this.showGrid) return;

  const xTicks = this.xScale.ticks(10);
  const yTicks = this.yScale.ticks(10);

  this.gridLayer.selectAll('line.x')
    .data(xTicks)
    .enter()
    .append('line')
    .attr('class', 'x')

```

```

        .attr('x1', d => this.xScale(d))
        .attr('x2', d => this.xScale(d))
        .attr('y1', 0)
        .attr('y2', this.plotH)
        .attr('stroke', '#ccc');

    this.gridLayer.selectAll('line.y')
        .data(yTicks)
        .enter()
        .append('line')
        .attr('class', 'y')
        .attr('x1', 0)
        .attr('x2', this.plotW)
        .attr('y1', d => this.yScale(d))
        .attr('y2', d => this.yScale(d))
        .attr('stroke', '#ccc');
    },
    updateScales() {
        this.xScale.domain([this.xMin, this.xMax]);
        this.yScale.domain([this.yMin, this.yMax]);
        this.xAxisG.call(d3.axisBottom(this.xScale));
        this.yAxisG.call(d3.axisLeft(this.yScale));
        this.drawGrid();
        this.updatePoints();
    },
    updatePoints() {
        if (!this.xScale || !this.yScale) return;

        this.posCircle
            .attr('cx', this.xScale(this.posX))
            .attr('cy', this.yScale(this.posY))

```

```

        .style('display', this.showPos ? 'block' : 'none');

        this.calcCircle
            .attr('cx', this.xScale(this.calcX))
            .attr('cy', this.yScale(this.calcY))
            .style('display', this.showCalc ? 'block' : 'none');
    },
    restartLoop() {
        clearInterval(this.intervalId);
        this.intervalId = setInterval(this.updatePoints, 1000 / this.frequency);
    }
}
};
</script>

```

```

<style scoped>
h2 {
    margin-bottom: 10px;
}
input[type="number"] {
    width: 70px;
}
</style>

```

TelemetryChart.vue

```

<template>
<div>
    <h2 v-if="title" style="margin-bottom: 5px;">{{ title }}</h2>

    <!-- ————— Панель управления ————— -->
    <div style="margin-bottom: 10px;">
        <label>Частота (Гц):</label>

```

```

<input type="range" min="1" max="50" v-model="frequency" />
<span>{{ frequency }} Гц</span>

<template v-for="(label, i) in lineLabels" :key="i">
  <label class="line-checkbox" :style="{ color: colors[i] }">
    <input type="checkbox" v-model="visible[i]" />
    {{ label }}
  </label>
</template>

<button @click="pause = !pause">{{ pause ? 'Продолжить' :
'Пауза' }}</button>
<button @click="reset">Обновить</button>
</div>

<!-- Диапазон по Y -->
<div style="margin-bottom: 10px;">
  <label>Y min:</label>
  <input type="number" v-model.number="yMin" />
  <label style="margin-left: 10px;">Y max:</label>
  <input type="number" v-model.number="yMax" />
</div>

<!-- SVG-график -->
<svg :id="svgId" width="100%" height="300"></svg>
</div>
</template>

<script>
import * as d3 from 'd3';

```

```

export default {
  props: {
    svgId: { type: String, required: true },
    dataSeries: { type: Array, required: true },
    colors: { type: Array, required: true },
    yRange: { type: Array, required: true },
    xUnit: { type: String, default: " " },
    yUnit: { type: String, default: " " },
    lineLabels: { type: Array, default: () => [] },
    title: { type: String, default: " " }
  },
  data() {
    return {
      frequency: 10,
      visible: [],
      pause: false,
      yMin: this.yRange[0],
      yMax: this.yRange[1],
      bufferData: [],
      xOffset: 0,
      MAX_POINTS: 400,
      intervalId: null
    };
  },
  watch: {
    frequency() { this.restartLoop(); },
    yMin() { this.updateScales(); },
    yMax() { this.updateScales(); },
    visible() { this.updateChart(); }
  },
  mounted() {

```

```

    this.visible = this.dataSeries.map(() => true);
    this.initChart();
    this.restartLoop();
  },
  beforeUnmount() {
    clearInterval(this.intervalId);
  },
  methods: {
    initChart() {
      const svg = d3.select(`#${this.svgId}`);
      const { width: w, height: h } = svg.node().getBoundingClientRect();
      const m = { l: 50, r: 20, t: 20, b: 35 };
      const iw = w - m.l - m.r;
      const ih = h - m.t - m.b;

      this.svg = svg;
      this.g = svg.append('g').attr('transform', `translate(${m.l},${m.t})`);
      this.plotW = iw;
      this.plotH = ih;

      this.xScale = d3.scaleLinear().range([0, iw]).domain([0,
this.MAX_POINTS]);
      this.yScale = d3.scaleLinear().range([ih, 0]).domain([this.yMin,
this.yMax]);

      this.xAxisG = this.g.append('g').attr('transform', `translate(0,${ih})`);
      this.yAxisG = this.g.append('g');

      this.xLabel = this.g.append('text')
        .attr('class', 'axis-label')
        .attr('x', iw / 2)

```



```

        .attr('y', ih + m.b - 5)
        .style('text-anchor', 'middle')
        .text(this.xUnit);

    this.yLabel = this.g.append('text')
        .attr('class', 'axis-label')
        .attr('transform', 'rotate(-90)')
        .attr('x', -ih / 2)
        .attr('y', -m.l + 15)
        .style('text-anchor', 'middle')
        .text(this.yUnit);

    this.paths = this.colors.map(c =>
        this.g.append('path')
            .attr('stroke', c)
            .attr('fill', 'none')
            .attr('stroke-width', 1.5)
    );

    this.bufferData = this.dataSeries.map(() => []);
    this.updateScales();
},
updateScales() {
    this.yScale.domain([this.yMin, this.yMax]);
    this.xScale.domain([this.xOffset, this.xOffset + this.MAX_POINTS]);
    this.yAxisG.call(d3.axisLeft(this.yScale));
    this.xAxisG.call(d3.axisBottom(this.xScale).ticks(10));
},
updateChart() {
    const line = d3.line()
        .x((_, i) => this.xScale(this.xOffset + i))

```

```

.y(v => this.yScale(v));

this.paths.forEach((p, i) => {
  if (this.visible[i] && this.bufferData[i].length) {
    p.attr('d', line(this.bufferData[i]));
  } else {
    p.attr('d', null);
  }
});

this.updateScales();
},
loop() {
  if (this.pause) return;

  this.dataSeries.forEach((src, i) => {
    if (!this.bufferData[i]) this.bufferData[i] = [];
    if (src.length) {
      const v = src[src.length - 1];
      this.bufferData[i].push(v);
      if (this.bufferData[i].length > this.MAX_POINTS) {
        this.bufferData[i].shift();
        if (i === 0) this.xOffset++;
      }
    }
  });

  this.updateChart();
},
restartLoop() {
  clearInterval(this.intervalId);

```

```

        this.intervalId = setInterval(this.loop, 1000 / this.frequency);
    },
    reset() {
        this.bufferData = this.bufferData.map(() => []);
        this.xOffset = 0;
        this.updateChart();
    }
}
};
</script>

```

```

<style>
button {
    margin-left: 10px;
}

```

```

.axis-label {
    font-size: 12px;
    fill: #555;
}

```

```

.line-checkbox {
    margin-left: 10px;
    font-weight: bold;
}

```

```

.line-checkbox input[type="checkbox"] {
    appearance: none;
    width: 16px;
    height: 16px;
    border: 2px solid currentColor;
}

```

```
border-radius: 4px;  
vertical-align: middle;  
margin-right: 5px;  
position: relative;  
cursor: pointer;  
}
```

```
.line-checkbox input[type="checkbox"]:checked::after {  
  content: "";  
  position: absolute;  
  top: 2px;  
  left: 5px;  
  width: 4px;  
  height: 8px;  
  border: solid currentColor;  
  border-width: 0 2px 2px 0;  
  transform: rotate(45deg);  
}
```

```
</style>
```