Санкт–Петербургский государственный университет

Группа 24.М71-мм

КИСЕЛЕВ Владимир Александрович

Отчет по учебной практике
в форме «Производственное задание»

«Высокопроизводительный симулятор системы управления роем роботов одним пультом»

Научный руководитель:
доктор физико-математических наук,
профессор кафедры системного программирования,
Граничин Олег Николаевич

Санкт-Петербург
2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Использование роев беспилотных роботов в последние годы привлекает внимание многих исследователей. Задачи, решаемые роями беспилотных летательных систем, возникают в самых разных областях человеческой деятельности: от агропромышленности и поисковых операций до мониторинга инфраструктуры и сервисов доставки [1; 2]. Одной из ключевых задач в этой области является разработка и тестирование алгоритмов управления, позволяющих рою выполнять скоординированные действия [3].

Тестирование алгоритмов на реальных роботах требует больших временных и финансовых затрат, а также сопряжено с риском повреждения дорогостоящего оборудования. В связи с этим, первоначальная отладка и настройка алгоритмов проводятся в программных симуляторах. Симуляторы предоставляют контролируемую, повторяемую и безопасную среду для разработки, позволяя легко масштабировать количество агентов и собирать детальные метрики их производительности. Однако многие существующие симуляторы сталкиваются с ограничениями производительности при моделировании большого количества роботов, что создает разрыв между симуляцией и реальными условиями.

1 ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Целью работы является разработка высокопроизводительного, масштабируемого симулятора для исследования и тестирования алгоритмов управления роем беспилотных летательных аппаратов.

Основными задачами работы являются:

1. Проанализировать существующие подходы к симуляции роевых систем и выявить их ограничения.
2. Определить функциональные и нефункциональные требования к симулятору.
3. Спроектировать архитектуру симулятора, ориентированную на высокую производительность и масштабируемость.
4. Реализовать ядро симулятора, обеспечивающее управление жизненным циклом большого количества виртуальных роботов.
5. Разработать инструментарий для автоматической генерации конфигураций и полетных параметров для симуляции больших роев.
6. Провести тестирование производительности и масштабируемости разработанного симулятора.

2 ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Для эффективного проектирования симулятора роя роботов необходимо рассмотреть существующие подходы, инструменты и вызовы в этой области.

2.1 Существующие решения для симуляции роя

Существует множество программных инструментов, предназначенных для моделирования и симуляции поведения роевых систем. В работе «Survey of Simulators for Aerial Robots» [4] приводится детальный анализ и классификация существующих симуляторов для воздушных роботов. Авторы выделяют несколько ключевых критериев для их сравнения: реалистичность физической модели, поддержка программно-аппаратного моделирования (SIL/HIL), масштабируемость, доступность и поддержка многоагентных систем.

2.1.1 Традиционные симуляторы и их ограничения

Симуляторы, такие как Gazebo и AirSim, ориентированы на достижение высокой реалистичности 3D-визуализации и точного моделирования физики. Однако, как отмечается в [4], их высокая вычислительная сложность часто становится узким местом при попытке симуляции десятков или сотен роботов одновременно, что делает их менее подходящими для исследования именно роевых алгоритмов.

2.1.2 Симуляторы для обучения с подкреплением (Reinforcement Learning)

В последние годы активно развиваются симуляторы, ориентированные на задачи обучения с подкреплением. В работе «Learning to Fly — a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control» [5] представлен `gym-pybullet-drones` — среда на основе физического движка PyBullet, предназначенная для обучения

как одиночных, так и многоагентных систем управления квадрокоптерами. Среда обеспечивает детерминированность, необходимую для воспроизводимости RL-экспериментов.

Другим примером является QuadSwarm [6], модульный симулятор, также ориентированный на RL. Его ключевая особенность — прямое управление тягой моторов, что позволяет обучать низкоуровневые контроллеры. QuadSwarm оптимизирован для высокой производительности и способен генерировать до 30 000 отсчетов в секунду на одном ядре процессора, однако он использует упрощенную модель аэродинамики.

Несмотря на высокую производительность в генерации данных для обучения, такие симуляторы часто жертвуют реалистичностью моделирования автопилота, заменяя его упрощенной моделью или непосредственно управляя силами. Это может приводить к расхождению между поведением в симуляции и на реальном оборудовании (Sim-to-Real Gap).

2.1.3 SwarmLab: a MATLAB Drone Swarm Simulator

SwarmLab, разработанный на базе MATLAB, представляет собой инструмент для быстрого прототипирования алгоритмов роя [7]. Он предоставляет встроенные инструменты для анализа и визуализации, однако его производительность ограничена возможностями интерпретируемого языка MATLAB, а также он не поддерживает интеграцию с реальными прошивками автопилотов, что ограничивает реалистичность симуляции.

2.2 Инструменты, используемые в проекте

Для достижения поставленной цели в работе используются следующие ключевые технологии, позволяющие совместить высокую производительность и реалистичность моделирования.

2.2.1 Язык программирования Go

В качестве основного языка для разработки симулятора был выбран Go (Golang). Этот выбор обусловлен рядом ключевых преимуществ, крити-

чески важных для создания высокопроизводительных и масштабируемых систем:

- **Встроенная поддержка конкурентности:** Модель конкурентности в Go, основанная на горутинах (goroutines) и каналах (channels), позволяет удобно и эффективно управлять тысячами параллельных операций, упрощая их реализацию;
- **Высокая производительность:** Go является компилируемым языком, что обеспечивает производительность, сравнимую с C++, но с лаконичным синтаксисом и автоматическим управлением памятью;
- **Эффективное управление памятью:** Сборщик мусора в Go оптимизирован для минимизации задержек, что важно для приложений, работающих в режиме, близком к реальному времени.

2.2.2 ArduPilot SITL (Software-in-the-Loop)

ArduPilot — это многофункциональная открытая прошивка для автопилотов, получившая широкое распространение в сообществе [8]. Режим SITL позволяет запускать бинарный код прошивки ArduPilot на персональном компьютере, что обеспечивает полную симуляцию поведения автопилота без необходимости использования реального оборудования. SITL получает данные от симулятора физики и возвращает команды для моторов, позволяя тестировать все высокоуровневые функции автопилота, такие как навигация, выполнение миссий и отказоустойчивость, в полностью программной среде.

2.2.3 QEMU+KVM и эмуляция ESP32

QEMU (Quick Emulator) — это универсальный эмулятор, способный запускать операционные системы и программы, предназначенные для одной архитектуры процессора, на другой [9]. Для достижения максимальной производительности QEMU может использовать гипервизор KVM (Kernel-based Virtual Machine), который является частью ядра Linux. Связка QEMU+KVM позволяет выполнять гостевой код с практически натив-

ной скоростью, используя аппаратные расширения виртуализации (Intel VT-x или AMD-V) [10]. Именно эта связка используется в нашем симуляторе для работы в среде Ubuntu.

Выбор QEMU+KVM в качестве инструмента для эмуляции микроконтроллеров ESP32 обусловлен его высокой производительностью по сравнению с альтернативами и гибкостью. В отличие от VirtualBox, который ориентирован в первую очередь на настольную виртуализацию архитектуры x86, или проприетарных решений от VMware, QEMU является открытым инструментом с широкой поддержкой различных архитектур, включая Xtensa, используемую в ESP32.

Параметр	**QEMU (чистая эмуляция)**	**QEMU + KVM**	**VirtualBox**	**VMware**
Тип гипервизора	Эмулятор + виртуализация	Тип 1	Тип 2	Тип 2
Производительность	Низкая	Почти нативная	Средняя	Высокая
Лицензия	Open Source	Open Source	Open Source	Проприетарная
Поддержка архитектур	Широкая (ARM, RISC-V...)	Широкая (ARM, RISC-V...)	В основном x86	В основном x86
Устройства	Ограничена	Требует доп. настройки	Хорошая	Хорошая

Таблица 1 — Сравнение гипервизоров и эмуляторов

В контексте данной работы эмуляция ESP32 с помощью QEMU позволяет запускать и отлаживать прошивки, предназначенные для этих микроконтроллеров, в виртуальной среде. Это дает возможность тестировать алгоритмы децентрализованного взаимодействия, которые выполняются на бортовых микроконтроллерах, без использования реальных физических устройств.

3 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

3.1 Архитектура системы управления роем роботов

В основе симулятора предлагается используется трёхуровневую архитектуру (см. Рисунок 1), аналогичную предложенной в исследованиях [11—13]. Эта архитектура распределяет задачи управления между различными уровнями системы.

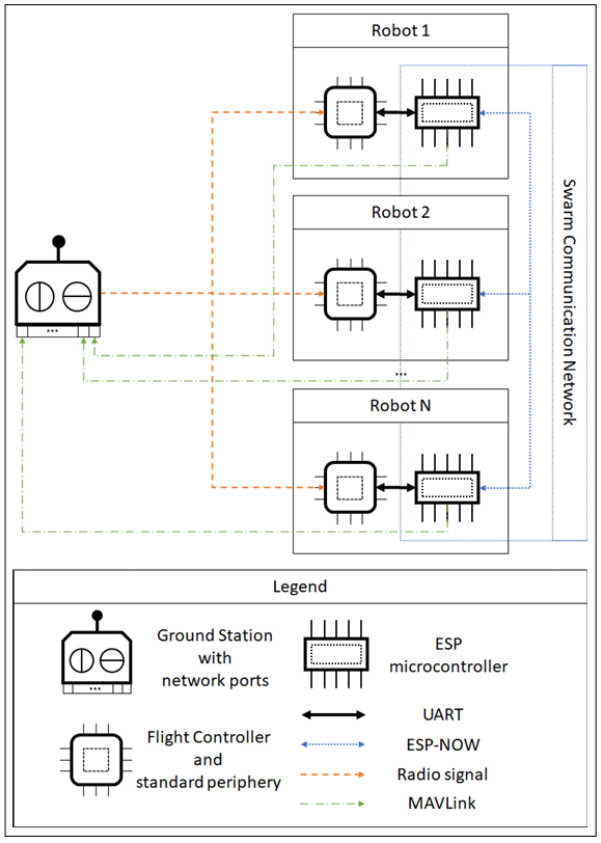

Рисунок 1 — Трёхуровневая архитектура

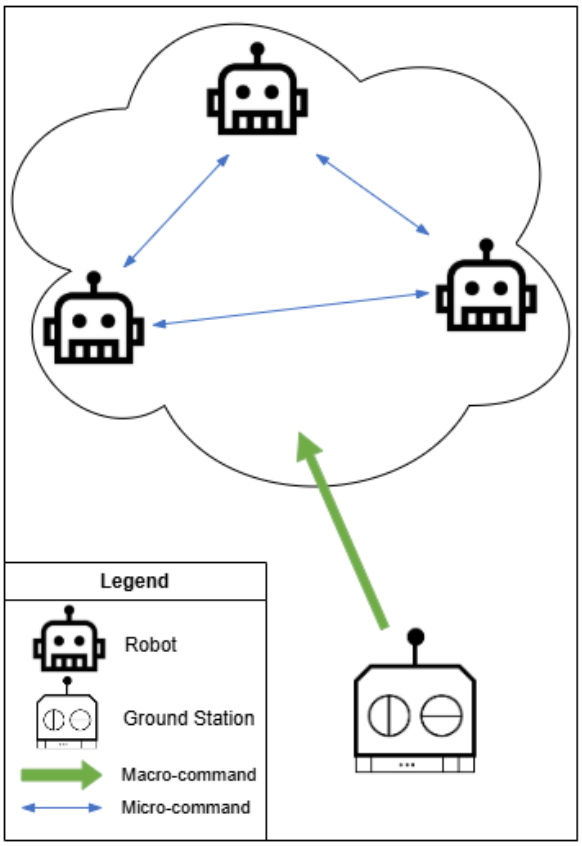

Рисунок 2 — Микро- и макрокоманды

1. **Автопилот (полётный контроллер):** нижний уровень. Отвечает за стабилизацию, навигацию и непосредственное управление полетом робота.
2. **Микроконтроллер (ESP):** средний уровень. Осуществляет связь с автопилотом, обрабатывает микрокоманды от других роботов и реализует логику децентрализованного управления.

3. **Базовая станция (пульт управления):** верхний уровень. Взаимодействует с автопилотами, визуализирует состояние роя, позволяет оператору задавать макрокоманды и контролировать выполнение миссии.

Такая архитектура позволяет распределить вычислительную нагрузку: критичные по времени задачи стабилизации решаются на автопилоте, задачи локальной координации — на микроконтроллере, а общее целеполагание и оркестрация — на базовой станции.

3.2 Требования к симулятору

На основе анализа существующих решений и целей проекта были сформулированы функциональные и нефункциональные требования к разрабатываемому симулятору.

3.2.1 Функциональные требования

- Обеспечение полного взаимодействия с автопилотом и базовой станцией по протоколу MAVLink;
- Симуляция обмена данными между базовой станцией и роботами и непосредственно между роботами;
- Детальное моделирование сети с имитацией взаимной видимости роботов, учётом помех, задержек и временных сбоев при передачи данных и настроек параметров сети;
- Поддержка запуска программного кода для микроконтроллера на эмуляторах;
- Моделирование физического окружения роботов;
- Сбор ключевых метрик симуляции;
- Наличие инструментов для анализа и визуализации собранных логов.

Требования на будущее развитие

- Возможность сохранения и загрузки полных сценариев симуляции в стандартных форматах.

3.2.2 Нефункциональные требования

- Поддержка одновременной работы не менее 128 роботов;
- Обеспечение совместимости с MAVProxy и другими стандартными GCS (Groud Control Station);
- Модульность и расширяемость архитектуры симулятора.

Требования на будущее развитие

- Улучшение масштабируемости для поддержки тысяч роботов;
- Интеграция с продвинутыми 3D-симуляторами и фреймворками для более реалистичной визуализации и моделирования сенсоров.

3.3 Архитектура Go-симулятора

В основе симулятора лежит ядро, написанное на языке Go. Выбор Go обусловлен его высокой производительностью, встроенной поддержкой конкурентности (горутины и каналы) и эффективным управлением памятью, что является критически важным для симуляции большого числа агентов.

Архитектура симулятора построена по модульному принципу:

- **Управляющие скрипты:** внешний слой оркестрации на `bash`, отвечающий за подготовку среды, запуск и управление жизненным циклом всех внешних процессов (SITL, MAVProxy, QEMU).
- `Simulator`: центральный компонент, отвечающий за управление жизненным циклом симуляции. Он инициализирует всех роботов, управляет их запуском и остановкой, а также агрегирует данные телеметрии;
- `Drone`: представляет собой абстракцию одного робота. Каждый экземпляр этого модуля работает в отдельной горутине, что обеспечивает параллельную обработку данных для каждого агента. Модуль отвечает за взаимодействие с экземпляром SITL и эмулятором QEMU;

– `Mavlink`: обеспечивает взаимодействие с ArduPilot SITL по протоколу MAVLink. Использует библиотеку `gomavlib` для парсинга и отправки сообщений.;
– `Network`: симулирует сетевое взаимодействие между роботами, позволяя моделировать задержки и потери пакетов для более реалистичного тестирования алгоритмов;
– **Генератор конфигураций:** утилита, позволяющая автоматически создавать JSON-конфигурации для роя с заданным количеством роботов, их начальными позициями и параметрами сети;
– **Генератор параметров:** утилита на основе Go-шаблонов для генерации файлов параметров ArduPilot, что позволяет уникально сконфигурировать каждый робот в рое.

Каждый робот в симуляторе запускает два основных процесса: экземпляр ArduPilot SITL и, опционально, экземпляр QEMU для эмуляции микроконтроллера. Go-симулятор выступает в роли оркестратора, управляя всеми этими процессами и обеспечивая обмен данными между ними через TCP, UDP и UDS (Unix Domain Sockets). Ядро симулятора, написанное на Go, имеет многокомпонентную архитектуру, где каждый модуль выполняет чётко определённую роль. Рассмотрим их подробнее.

3.3.1 Управляющие скрипты

Запуск и управление всей системой осуществляется через `bash`-скрипты (`run_golang_simulation.sh` и `run_golang_simulation_-headless.sh`). Они выполняют следующие функции:

1. **Подготовка среды:** скрипты проверяют наличие конфигурационных файлов, собирают исполняемый файл Go-симулятора и создают директории для логов.
2. **Запуск внешних процессов:** для каждого робота, описанного в `config.json`, скрипты запускают и конфигурируют экземпляры ArduPilot SITL, MAVProxy и, при необходимости, QEMU. Для оптимизации производительности реализовано распределение этих процессов по ядрам CPU с помощью `taskset`.

3. **Управление жизненным циклом:** скрипты сохраняют PID всех запущенных процессов и обеспечивают их корректное завершение при получении сигнала `SIGINT`, вызывая функцию очистки.
4. **Режимы запуска:** поддерживается два режима: с графическими терминалами (`xterm`) для каждого процесса для целей отладки, и полностью фоновый ("headless") режим для проведения масштабных экспериментов, где весь вывод перенаправляется в лог-файлы.

3.3.2 Модуль `Simulator`

Это центральный компонент приложения. Его жизненный цикл выглядит следующим образом:

1. **Инициализация:** при запуске, `Simulator` читает `config.json`, инициализирует симулятор сети `Network` и создает экземпляры всех роботов `Drone` в соответствии с конфигурацией.
2. **Запуск:** `Simulator` последовательно запускает симулятор сети и вызывает метод `Start()` для каждого экземпляра робота.
3. **Управление симуляцией:** симулятор периодически выводя статус роя и выполняет предопределённый сценарий.
4. **Завершение работы:** при получении сигнала о завершении, `Simulator` инициирует корректную остановку всех роботов и внутренних модулей.

3.3.3 Модуль `Drone`

Каждый робот является независимой сущностью, работающей в собственной горутине. Это ключевой элемент, обеспечивающий параллелизм. `Drone` инкапсулирует в себе всю логику, связанную с одним агентом роя.

- **Состояния и позиция:** хранит актуальное состояние и позицию робота, которая постоянно обновляется данными из MAVLink;
- **Коммуникационные каналы:** имеет два основных внешних соединения:
  - **MAVLink-соединение:** UDP-соединение с экземпляром MAVProxy. Через этот канал `Drone` получает телеметрию и

отправляет высокоуровневые команды управления. За чтение отвечает горутина `mavlinkReader`, за отправку команд — `controlHandler`;

- **Соединение для данных:** TCP или Unix-сокет, который подключается либо напрямую к порту `serial5` в SITL, либо к QEMU. Этот канал предназначен для обмена произвольными данными между роботами в рамках реализации роевых алгоритмов. Горутина `dataHandler` непрерывно читает данные из этого сокета и отправляет их в симулятор сети. Горутина `messageProcessor` принимает сообщения от симулятора сети и записывает их в этот сокет.

– **Внутренние каналы:** обмен данными с другими модулями происходит через Go-каналы: `messageTx` для отправки сообщений в сеть, `messageRx` для получения сообщений из сети, и `controlCh` для получения команд от `Simulator`.

3.3.4 Модуль `Network`

Этот модуль имитирует физическую среду распространения радиосигнала между роботами.

1. **Обработка сообщений:** сообщения, отправленные одним из роботов, поступают в общую очередь. Воркеры разбирают эту очередь и для каждого сообщения определяют его судьбу.
2. **Пространственное индексирование:** для быстрой и эффективной симуляции используется R-Tree. Позиции всех роботов периодически обновляются в этом дереве. При обработке сообщения от отправителя, симулятор делает запрос к R-Tree для получения списка всех роботов, находящихся в радиусе слышимости.
3. **Моделирование потерь:** для каждого робота-получателя, находящегося в пределах максимальной дальности, вычисляется вероятность доставки сообщения. Эта вероятность зависит от расстояния и базового процента потерь пакетов. Затем, с помощью генератора случайных чисел, принимается решение, будет ли сообщение "доставлено"или "потеряно".

4. **Доставка:** если сообщение "доставлено оно отправляется в канал `messageRx` соответствующего робота-получателя.

Такая архитектура позволяет эффективно симулировать сложные сетевые взаимодействия в большом рое, обеспечивая при этом высокую производительность за счет параллелизма и эффективных алгоритмов поиска соседей.

4 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ СИМУЛЯТОРА

4.1 Реализация симулятора

Симулятор разработан на языке Go и управляется с помощью bash-скриптов. Основной скрипт `run_golang_simulation.sh` отвечает за запуск всех компонентов системы в правильной последовательности.

4.1.1 Принцип работы

Процесс запуска и работы симулятора выглядит следующим образом:

1. Пользователь генерирует конфигурационный файл `config.json` с помощью утилиты `config-generator`, указывая количество роботов, их формацию и параметры сети.
2. С помощью утилиты `param-generator` создаются индивидуальные файлы параметров для каждого робота.
3. Запускается скрипт `run_golang_simulation.sh`. Он считывает конфигурацию и для каждого робота запускает в отдельном терминале экземпляр ArduPilot SITL и MAVProxy с соответствующими параметрами.
4. При необходимости, для каждого робота также запускается экземпляр QEMU с прошивкой для ESP32.
5. Запускается основное приложение Go-симулятора.
6. Go-симулятор создает горутину для каждого робота. Каждая горутина устанавливает MAVLink-соединение со своим экземпляром SITL и TCP-соединение с эмулятором QEMU.
7. Симулятор начинает передавать данные телеметрии и команды управления между компонентами, симулируя полет роя.
8. Взаимодействие с симулятором может происходить либо в автоматическом режиме (выполнение сценария), либо в интерактивном режиме через подключение GCS. Для этого симулятор может маршрутизировать MAVLink-трафик через стандартный MAVProxy. Разработка собственного высокопроизводительно-

го MAVProxy-компонента на Go находится в процессе и является одним из направлений дальнейшего развития проекта.

4.1.2 Ключевые технические решения

- **Конкурентность:** Использование горутин для каждого робота позволяет эффективно утилизировать многоядерные процессоры и обрабатывать данные от сотен роботов параллельно;
- **Производительность сети:** В рамках данной работы в репозиторий ArduPilot была добавлена возможность подключения к SITL через UDS. Это позволило уменьшить накладные расходы на пересылку сообщений между SITL и симулятором на одном хосте на 6%. В будущем планируется расширить эту поддержку на QEMU и MAVProxy-компонент;
- **Оптимизация на уровне ОС:** Для минимизации переключений контекста и снижения взаимного влияния процессов друг на друга, реализована привязка каждого процесса (SITL, MAVProxy, QEMU) к определённым ядрам CPU. Это обеспечивает более стабильную и предсказуемую производительность при симуляции большого количества роботов;
- **Эффективное моделирование сети:** Для масштабирования симуляции сетевого взаимодействия был внедрён алгоритм пространственного индексирования на основе R-Tree. Вместо полного перебора всех роботов (сложность O(N^2)) для определения соседей, R-Tree позволяет выполнять поиск за время, близкое к O(log N), что значительно снижает вычислительную нагрузку на модуль сети при увеличении числа агентов.

4.2 Результаты работы симулятора

Тестирование симулятора проводилось для оценки его производительности и масштабируемости. Основной метрикой была способность системы поддерживать симуляцию в реальном времени при увеличении количества роботов. Симулятор был протестирован в нескольких режимах:

– **Direct Mode:** Прямое взаимодействие Go-симулятора с ArduPilot SITL;
– **Headless Mode:** Запуск SITL без графических терминалов для снижения накладных расходов;
– **QEMU Mode:** полная симуляция с запуском SITL и QEMU для каждого робота;

Для тестирования были использованы две тестовые конфигурации оборудования.

Конфигурация 1:

– **CPU:** Intel Core i5-11400 @ 2.6GHz;
– **RAM:** 16 ГБ DDR4;
– **ОС:** Windows 11 с WSL2 (Ubuntu 22.04).

Результаты для 30 роботов на Конфигурации 1 представлены в Таблице 2.

Режим работы	**Загрузка CPU**	**Загрузка RAM**
Direct Mode	100%	60%
Headless Mode	80%	60%
QEMU Mode	100%	80%

Таблица 2 — Результаты производительности для 30 роботов (Intel i5-11400)

Конфигурация 2:

– **CPU:** AMD Ryzen Threadripper 3990X (64 ядра) @ 2.2GHz;
– **RAM:** 194 ГБ DDR4;
– **ОС:** Ubuntu 20.04.

Результаты для 128 роботов на Конфигурации 2 представлены в Таблице 3.

Режим работы	**Загрузка CPU**	**Загрузка RAM**
Direct Mode	20%	5%
Headless Mode	18%	5%
QEMU Mode	22%	6%

Таблица 3 — Результаты производительности для 128 роботов (AMD Threadripper 3990X)

Результаты для 512 роботов на Конфигурации 2 представлены в Таблице 4.

Режим работы	**Загрузка CPU**	**Загрузка RAM**
Direct Mode	-	-
Headless Mode	90%	35%
QEMU Mode	99%	40%

Таблица 4 — Результаты производительности для 512 роботов (AMD Threadripper 3990X)

Результаты демонстрируют высокую масштабируемость симулятора. На высокопроизводительном оборудовании система способна моделировать более пятисот роботов, что подтверждает эффективность выбранной архитектуры и языка Go.

Дополнительно были проведены измерения задержки при передаче сообщений между роботами. Среднее значение в симуляторе составило 70 мс. Для сравнения, передача сообщения между двумя микроконтроллерами ESP32 с использованием протокола ESP-NOW в реальных условиях занимает около 100 мс. Более низкое значение задержки в симуляторе предоставляет резерв для будущего усложнения моделей сетевого взаимодействия без потери реалистичности.

5 ВЫВОДЫ

В ходе выполнения работы были решены все поставленные задачи и достигнута основная цель — создан высокопроизводительный, масштабируемый симулятор для исследования роевых систем.

Основные полученные результаты:

1. Проведен анализ существующих симуляторов, который показал необходимость создания специализированного решения, ориентированного на масштабируемость и интеграцию с реальными прошивками.
2. Спроектирована и реализована на языке Go архитектура симулятора, использующая конкурентные возможности языка для эффективного моделирования сотен роботов.
3. Обеспечена глубокая интеграция с ArduPilot SITL и эмуляции микроконтроллеров ESP32 через QEMU, что позволяет проводить тестирование алгоритмов в среде, максимально приближенной к реальной с точки зрения поведения автопилота.
4. Создан набор утилит для автоматизации настройки и запуска крупномасштабных экспериментов.
5. Тестирование подтвердило высокую производительность и масштабируемость симулятора.

Ограничения текущей реализации: Основным ограничением текущей реализации является использование стандартного MAVProxy для взаимодействия с GCS, поскольку разработка собственного высокопроизводительного компонента на Go еще не завершена. Это может создавать узкое место или большую нагрузку при работе с большим количеством роботов, особенно в интерактивном режиме.

Направления дальнейшего развития:

– Завершение разработки и интеграция собственного MAVProxy-компонента на Go для минимизации задержек и повышения производительности;
– Расширение поддержки UDS на эмулятор QEMU для дальнейшего снижения накладных расходов на межпроцессное взаимодействие;

- Дальнейшее улучшение масштабируемости симулятора для поддержки тысяч роботов за счёт тестирования и реализации следующих подходов:
  - Шардирование симуляционной области с использованием сеточной архитектуры для параллельной обработки сетевых взаимодействий;
  - Применение арен памяти (memory arenas) для минимизации задержек, вносимых сборщиком мусора Go;
  - Переход на высокопроизводительные сетевые библиотеки (netpoll) для снижения накладных расходов при обработке большого количества сетевых соединений;
  - Использование профильно-ориентированной оптимизации (PGO) для более точной настройки компилятора под реальные сценарии нагрузки.
- Интеграция с продвинутыми 3D-визуализаторами и фреймворками для более наглядного представления симуляции и моделирования работы сенсоров.

6 ЗАКЛЮЧЕНИЕ

В рамках настоящей работы была успешно решена поставленная цель: разработан высокопроизводительный, масштабируемый симулятор для исследования и тестирования алгоритмов управления роем беспилотных летательных аппаратов.

В ходе работы были выполнены все поставленные задачи:

1. Проанализированы существующие подходы к симуляции роевых систем.
2. Определены функциональные и нефункциональные требования к симулятору.
3. Спроектирована архитектура симулятора.
4. Реализовано ядро симулятора.
5. Разработан инструментарий для автоматической генерации конфигураций и полетных параметров.
6. Проведено тестирование производительности и масштабируемости разработанного симулятора.

Разработанный симулятор представляет собой масштабируемый и расширяемый инструмент для исследований в области роевой робототехники. Он позволяет преодолеть ограничения производительности, свойственные многим традиционным симуляторам, и обеспечивает высокую степень реализма за счет интеграции с программными и эмулируемыми версиями реального бортового оборудования. Созданный программный продукт может быть использован для разработки, тестирования и верификации широкого спектра алгоритмов управления роем.

Ссылка на github репозиторий, содержащий материалы работы: https://github.com/CroccoRush/swarm-simulator

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Bu Y.*, *Yan Y.*, *Yang Y.* Advancement Challenges in UAV Swarm Formation Control: A Comprehensive Review // Drones. — 2024. — July. — Vol. 8. — P. 320.
2. *Amala Arokia Nathan R.J.*, *Indrajit K.*, *Bimber O.* Drone swarm strategy for the detection and tracking of occluded targets in complex environments // Communications Engineering. — 2023. — Aug. — Vol. 2.
3. *Do H.*, *Hua H.*, *Nguyen M.*, [et al.]. Formation Control Algorithms for Multiple-UAVs: A Comprehensive Survey // EAI Endorsed Transactions on Industrial Networks and Intelligent Systems. — 2021. — June. — P. 170230.
4. *Dimmig C.*, *Silano G.*, *Mcguire K.*, [et al.]. Survey of Simulators for Aerial Robots // IEEE Robotics Automation Magazine. — 2023. — Nov. — Vol. PP.
5. *Panerati J.*, *Zheng H.*, *Zhou S.*, [et al.]. Learning to Fly—a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control. — 2021. — Mar.
6. *Huang Z.*, *Batra S.*, *Chen T.*, [et al.]. QuadSwarm: A Modular Multi-Quadrotor Simulator for Deep Reinforcement Learning with Direct Thrust Control. — 2023. — June.
7. *Soria E.*, *Schiano F.*, *Floreano D.* SwarmLab: a Matlab Drone Swarm Simulator // 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). — 2020. — P. 8005–8011.
8. SITL Simulator (Software in the Loop). — 2024. — Accessed: 2025-09-15. https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html.
9. QEMU User Documentation. — 2024. — Accessed: 2025-09-15. https://www.qemu.org/docs/master/system/introduction.html.

10. KVM vs. VMware: A comparison. — 2024. — Accessed: 2025-09-15. https://www.redhat.com/en/topics/virtualization/kvm-vs-vmware-comparison.

11. *Ren W.* Consensus based formation control strategies for multi-vehicle systems //. Vol. 2006. — 07/2006. — 6 pp.

12. *Амелин К.*, *Антал Е.*, *Васильев В.*, *(Амелина) Н.* АДАПТИВНОЕ УПРАВЛЕНИЕ АВТОНОМНОЙ ГРУППОЙ БЕСПИЛОТНЫХ ЛЕТАТЕЛЬНЫХ АППАРАТОВ // СТОХАСТИЧЕСКАЯ ОПТИМИЗАЦИЯ В ИНФОРМАТИКЕ. — 2009. — Т. 5, № 1—1. — С. 157—166.

13. *Амелин К.*, *Граничин О.* Мультиагентное сетевое управление группой легких БПЛА // НЕЙРОКОМПЬЮТЕРЫ: РАЗРАБОТКА, ПРИМЕНЕНИЕ. — 2011. — № 6. — С. 64—72.