Санкт–Петербургский государственный университет

Группа 24.М71-мм

КИСЕЛЕВ Владимир Александрович

Отчет по учебной практике
в форме «Производственное задание»

«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 (а в некоторых случаях – наземных или надводных) систем, возникают в самых разных областях человеческой деятельности: агропромышленности, для поисковых операций, в мониторинге фауны, климатологии, картографии, для проверки и инспектирования инфраструктуры и в сервисах доставки [1] [2]. Одной из основных задач является задача поддержания строгой формации роем роботов, что требует эффективных алгоритмов управления и взаимодействия [3].

1 ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Целью работы является разработать архитектуру системы управления роем роботов при помощи одного пульта и симулятора для такой системы.

Основными задачами работы являются:

1. Проанализировать существующие решения для управления роем роботов.
2. Изучить существующие симуляторы для роевого управления.
3. Спроектировать архитектуру системы управления роем.
4. Определить функциональные и нефункциональные требования к системе и симулятору.
5. Разработать симулятор, реализующий предложенную архитектуру.
6. Провести тестирование разработанного симулятора для проверки соответствия функциональным требованиям и оценки его работоспособности.

2 ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Для эффективного управления роем роботов и их симуляции необходимо рассмотреть существующие подходы и инструменты.

2.1 Методы управления роем роботов

Для поддержания формации роя роботов может применяться два основных подхода: централизованный и децентрализованный [4].

2.1.1 Централизованный метод

Централизованный метод подразумевает наличие базовой станции, с которой взаимодействует каждый из роботов, получая свои задачи. Базовая станция собирает данные от всех роботов роя и определяет необходимые позиции для каждого из них. Однако централизованный подход имеет ряд недостатков: с увеличением числа роботов в рое возрастает вычислительная и коммуникационная нагрузка на базовую станцию, что ограничивает общую производительность системы [3] [4].

2.1.2 Децентрализованный метод

При децентрализованном подходе каждый робот обязан самостоятельно определять свое требуемое местоположение. Обычно в рамках такого метода роботы имеют возможность вести локальное взаимодействие друг с другом и обмениваться необходимыми данными. Одним из способов децентрализованного поддержания формации являются алгоритмы консенсуса [5] [6]. При таком подходе каждый робот в рое принимает решения о движении, опираясь на информацию о состоянии своих соседей, что способствует согласованному поведению всего роя [6] [7]. Децентрализованные подходы более масштабируемы и устойчивы к отказам отдельных узлов или базовой станции.

2.2 Существующие решения для симуляции роя

Существует один программный инструмент в открытом доступе, предназначенный для моделирования и симуляции поведения роевых систем.

2.2.1 SwarmLab: a MATLAB Drone Swarm Simulator

SwarmLab, разработанный Сориа, Скиано и Флореано, представляет собой симулятор на базе MATLAB, который упрощает прототипирование, настройку, отладку и анализ производительности алгоритмов роя [8]. SwarmLab стремится предоставить стандартизированные процессы и показатели для количественной оценки производительности и надежности роевых алгоритмов, особенно в контексте летающих роботов.

Ключевые особенности SwarmLab:

- Среда MATLAB: SwarmLab полностью написан на MATLAB, скриптовом языке, который обеспечивает высокий уровень абстракции и встроенные инструменты для проектирования, управления, анализа и визуализации.
- Модульная архитектура: программное обеспечение имеет объектно-ориентированную структуру с такими компонентами, как параметры для роботов, роя и окружающей среды, классы `Drone` и `Swarm`, графические классы для визуализации и примеры сценариев для помощи пользователям.
- Алгоритмы роя: SwarmLab включает в себя реализации двух децентрализованных алгоритмов для навигации роя: алгоритм Ольфати-Сабера и алгоритм Вашархеи, что позволяет пользователям сравнивать и анализировать их эффективность.
- Анализ производительности: симулятор предоставляет инструменты для сбора данных, анализа производительности и визуализации поведения роя, включая такие метрики, как безопасность расстояния между агентами и вне агентов, связность и порядок.

В отличие от подхода SwarmLab, в котором используется специализированная среда симуляции в MATLAB, данная работа направлена на со-

здание симулятора роя роботов, интегрированного с симуляцией Software-In-The-Loop (SITL), что обеспечивает более реалистичное моделирование поведения роботов.

3 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

3.1 Архитектура системы управления роем роботов

Для снижения нагрузки на базовую станцию и обеспечения масштабируемости в данной работе предлагается использовать трёхуровневую архитектуру (см. Рисунок 1) с использованием макро- и микрокоманд (см. Рисунок 2). Эта архитектура сочетает преимущества централизованного и децентрализованного подходов.

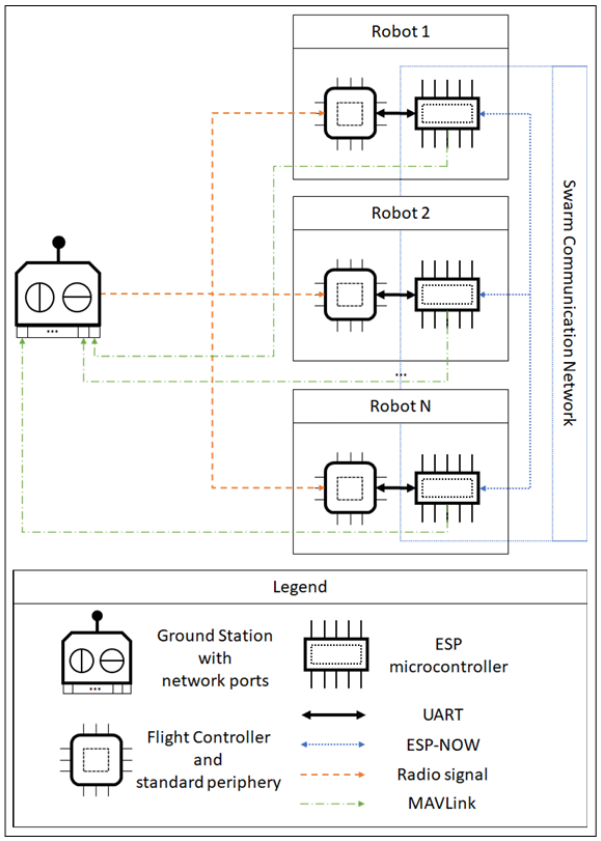

Рисунок 1 — Трёхуровневая архитектура

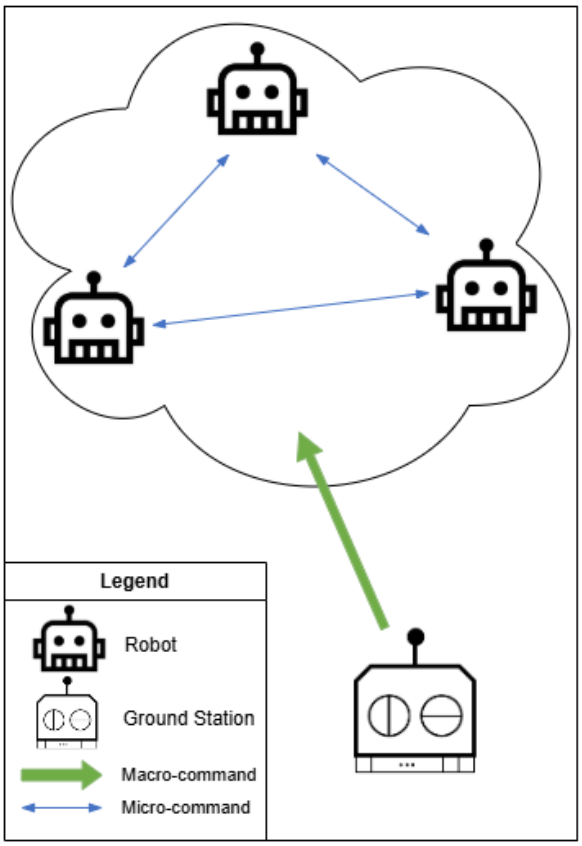

Рисунок 2 — Микро- и макрокоманды

Микрокоманда — сообщение, которое один робот отправляет своему соседу и в котором содержится информация о состоянии робота-отправителя (например, его позиция, скорость). Используются для локальной координации и поддержания строя на основе децентрализованных алгоритмов (например, алгоритмов консенсуса).

Макрокоманда — сообщение от базовой станции, которое рассылается в виде broadcast-сообщения всем доступным роботам или целевой

группе. Задает общий характер движения или задачу для всего роя (например, переместиться в заданную область, начать выполнение миссии).

В предложенной архитектуре макрокоманда рассылается с базовой станции всем доступным роботам, задавая общее направление или цель, а микрокоманды, которыми обмениваются соседние роботы, позволяют им корректировать свое взаимное расположение внутри группы, нивелируя ошибки, возникшие из-за различных погрешностей, и поддерживая заданную формацию даже при неидеальной связи с базовой станцией.

Трёхуровневость системы управления роем роботов подразумевает под собой следующее разделение обязанностей:

1. **Автопилот (полётный контроллер):** нижний уровень. Отвечает за стабилизацию, навигацию и непосредственное управление полетом робота на основе команд с верхних уровней. В нашей симуляции эту роль выполняет ArduPilot SITL.
2. **Микроконтроллер (ESP):** средний уровень. Осуществляет связь с автопилотом через COM-порт, принимает и обрабатывает микрокоманды от других роботов в сети, реализует логику децентрализованного управления (например, алгоритм консенсуса) и передает скорректированные управляющие воздействия автопилоту.
3. **Базовая станция (пульт управления):** верхний уровень. Взаимодействует с автопилотами роботов, визуализирует состояние роя, позволяет оператору задавать макрокоманды и контролировать выполнение миссии. В симуляторе представлена управляющим приложением (GUI/геймпад или скрипт эксперимента).

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

3.2 Требования к симулятору

Чтобы иметь условия, приближенные к реальным, симулятор должен эмулировать работу реальной системы, включая моделирование сети,

окружения, базовой станции, в связи с чем были составлены списки функциональных и нефункциональных требований.

3.2.1 Функциональные требования

- Поддержка протокола MAVLink для коммуникации роботов и базовой станцией;
- Симуляция взаимодействия между базовой станцией и роботами;
- Симуляция обмена данными между роботами через интерфейс, соответствующий коммуникационному модулю;
- Моделирование окружения, в котором перемещаются роботы;
- Логирование метрик симуляции;
- Визуализация результатов полётов.

Требования на будущее развитие:

- Детальное моделирование коммуникационной сети:
  - симуляция взаимной видимости роботов основываясь на расстоянии между ними и возможных препятствиях;
  - учет случайных помех, задержек и временных сбоев при передаче данных;
  - настройка параметров сети (пропускная способность, дальность, уровень потерь) для определения доступности и качества передачи данных между роботами;
- Поддержка доставки команд от пульта ко всем SITL-роботам с симуляцией видимости пульта;
- Поддержка запуска программного кода для микроконтроллера (например, ESP) на эмуляторах (например, Wokwi, QEMU) или интеграция с реальным оборудованием (Hardware-in-the-Loop);
- Импорт/экспорт сценариев полёта в стандартных форматах (например, JSON, YAML, MAVLink Mission).

3.2.2 Нефункциональные требования

- Поддержка одновременной работы как минимум четырёх роботов без существенной деградации скорости симуляции реального времени;
- Совместимость с MAVProxy;
- Модульность и расширяемость архитектуры симулятора.

Требования на будущее развитие:

- Улучшение масштабируемости для поддержки десятков и сотен роботов;
- Совместимость и интеграция с продвинутыми 3D симуляторами (например, Gazebo, jMAVSim) и фреймворками (например, ROS2) для более реалистичной визуализации и моделирования сенсоров.

4 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ СИМУЛЯТОРА

На основе спроектированной архитектуры и требований был разработан симулятор системы управления роем роботов.

4.1 Реализация симулятора

Симулятор системы управления роем роботов разработан с использованием языка программирования Python и управляется с помощью bash-скриптов для запуска и настройки окружения. Система предназначена для моделирования поведения группы роботов при управлении с единого интерфейса, как в интерактивном режиме, так и в режиме автоматизированных экспериментов.

Разработанный симулятор сопоставляется с трехуровневой архитектурой следующим образом:

- Уровень автопилота моделируется экземплярами ArduPilot SITL;
- Уровень микроконтроллера моделируется TCP-соединением между экземплярами SITL и логикой пересылки данных в классе `Drone`. Реализация алгоритмов консенсуса предполагается либо в прошивке ArduPilot (используя Lua-скрипты или модификацию C++ кода), либо в будущем при интеграции с эмуляторами ESP;
- Уровень базовой станции реализуется через MAVProxy для связи и классы `DroneControlGUI` (управление с геймпада) или `ExperimentRunner` (автоматические сценарии) для отправки макрокоманд.

4.1.1 Компоненты симулятора

В основе архитектуры симулятора лежат следующие ключевые компоненты:

1. **SITL (Software-in-the-Loop):** используется ArduCopter SITL для симуляции полётной динамики, сенсоров и автопилота каждого отдельного робота. Запуск и конфигурация нескольких эк-

экземпляров SITL автоматизированы скриптом `run_simulation.sh`, который считывает параметры (ID, порты, начальные позиции) из файла `config.json`. Каждый экземпляр SITL запускается в отдельном окне терминала (`xterm`) для наглядности.

2. **MAVProxy:** скрипт `mavproxy.py` используется для управления MAVLink соединениями между экземплярами SITL и управляющим приложением. Он выступает в роли маршрутизатора, позволяя централизованно общаться со всеми роботами.
3. **Drone (`app/drone.py`):** класс, инкапсулирующий представление одного робота в симуляции. Он отвечает за:
   - установление и поддержание MAVLink соединения со своим экземпляром SITL для отправки команд управления (RC-каналы, режимы полета, параметры, arm/disarm) и приёма телеметрии;
   - установление TCP-соединения, имитирующего порт SERIAL5 ArduPilot, для обмена данными между роботами;
   - логирование данных о состоянии робота в CSV-файлы для последующего анализа;
   - запуск фоновых потоков для чтения данных из MAVLink и по TCP-соединению (SERIAL5);
   - пересылку данных, полученных по TCP, другим роботам в сети напрямую через их TCP-сокеты (метод `forward_data`). Эта реализация представляет собой *идеализированную сеть* без задержек и потерь.
4. **NetworkSimulator (`app/network.py`):** класс, который инициализирует объекты `Drone` на основе конфигурации из `config.json`. Он также содержит конфигурационные параметры сети (максимальная дальность, базовая потеря пакетов) и логику для симуляции эффектов сети (задержки, потери пакетов в зависимости от расстояния). *Примечание: в текущей реализации активная симуляция сетевых эффектов из этого класса не используется. В будущем планируется перенести логику обмена сообщениями между роботами в этот класс для детального моделирования сети.*

5. **DroneControlGUI** (`app/gamepad.py`)**:** класс, реализующий интерактивный режим управления (`gui` режим) с помощью геймпада. Позволяет пользователю в реальном времени управлять одним или несколькими роботами, отправляя команды через MAVLink соединения объектов `Drone`.
6. **ExperimentRunner** (`app/experiment_runner.py`)**:** класс, отвечающий за автоматическое выполнение экспериментов (`experiment` режим). Он позволяет задавать параметры автопилота роботов, запускать предопределенные сценарии и сохранять логи для анализа эффективности различных алгоритмов или конфигураций.
7. **Data Processor** (`app/proff.py`)**:** набор функций для постобработки и анализа данных, собранных во время экспериментов. Включает чтение логов, объединение данных с нескольких роботов, расчет ошибок формации, визуализацию траекторий и метрик производительности с использованием `matplotlib`, `scipy` и `pandas`.
8. **Orchestration Scripts** (`run_simulation.sh`, `run_all_experiments.sh`)**:**
   - `run_simulation.sh`: запускает один экземпляр симуляции, включая все необходимые компоненты (SITL, MAVProxy, основное приложение Python) с нужным режимом (`gui` или `experiment`). Также содержит логику для корректного завершения всех процессов;
   - `run_all_experiments.sh`: автоматизирует запуск серии экспериментов. Итерируется по заданным наборам параметров и номерам запусков, вызывая `run_simulation.sh experiment` для каждой комбинации и выполняя очистку между запусками.

4.1.2 Принцип работы

Процесс запуска и работы симулятора выглядит следующим образом:

1. Скрипт `run_simulation.sh` считывает конфигурацию из `config.json` (количество роботов, их ID, UDP и TCP порты, начальные позиции).
2. Для каждого робота запускается отдельный процесс ArduCopter SITL в `xterm`, настроенный на соответствующие порты и параметры.
3. Запускается `mavproxy.py`, который подключается ко всем SITL экземплярам и предоставляет точки подключения для управляющего приложения.
4. Запускается основной скрипт `app/simulator.py` с указанием режима (`--mode gui` или `--mode experiment`).
5. `app/simulator.py` создает экземпляр `NetworkSimulator`, который, в свою очередь, создает и инициализирует объекты `Drone` для каждого робота из конфигурации.
6. Каждый объект `Drone` устанавливает MAVLink-соединение со своим SITL и TCP-соединение для имитации SERIAL5. Запускаются потоки для чтения MAVLink и данных с TCP-порта SERIAL5.
7. В зависимости от режима:
   - **GUI Mode:** запускается `DroneControlGUI`, позволяющий взаимодействовать с геймпадом. `DroneControlGUI` преобразует ввод в MAVLink-команды (RC Override, смена режима, arm/disarm и т.д.) и отправляет их выбранным роботам через их объекты `Drone`;
   - **Experiment Mode:** запускается `ExperimentRunner`. Он устанавливает параметры роботов через MAVLink, выполняет последовательность действий, ожидает завершения и сохраняет логи.
8. Обмен данными между роботами происходит по TCP. Данные, полученные одним роботом, пересылаются другим через метод `forward_data` класса `Drone` напрямую в их TCP-сокеты.
9. Скрипт `run_all_experiments.sh` позволяет автоматизировать запуск множества экспериментов с различными параметрами и повторениями.

10. По завершении экспериментов, скрипты из `app/proff.py` могут быть использованы для анализа собранных логов и визуализации результатов.

4.2 Результаты работы симулятора

Для проверки работоспособности симулятора и демонстрации его возможностей было проведено несколько серий экспериментов. В качестве примера рассматривается задача поддержания формации роем из четырёх роботов при движении по заданному маршруту.

Сценарий эксперимента:

1. Рой из четырёх роботов стартует из начальных позиций, заданных в `config.json`.
2. Используя режим `experiment`, роботам дается команда на взлет и занятие определенной формации на заданной высоте.
3. Рою дается макрокоманда двигаться по определенному маршруту.
4. Во время движения роботы должны поддерживать заданную формацию, используя алгоритмы, реализованные в прошивке ArduPilot (в данном случае, сравнивались встроенные алгоритмы LVP и ALVP, переключаемые параметром `Swarm_XY_ALGO`).
5. Симуляция выполняется в течение заданного времени или до достижения конечной точки маршрута.
6. Процесс повторяется несколько раз для каждого набора параметров (LVP и ALVP) для сбора статистики (с использованием `run_all_experiments.sh`).

Сбор и обработка данных: Во время симуляции данные о положении (широта, долгота, высота) и курсе каждого робота логируются в CSV-файлы классом `Drone`. После завершения серий экспериментов, данные обрабатываются с помощью скриптов из `app/proff.py`:

- данные из разных логов объединяются по временным меткам;
- рассчитываются метрики качества поддержания формации, в формате средней абсолютной ошибки (MAE) от идеальной геометрии формации;

– строятся графики траекторий роботов и графики изменения ошибки формации во времени.

Примеры результатов:

На Рисунке 3 показаны траектории движения четырёх роботов для двух сравниваемых алгоритмов (LVP — сплошные линии, ALVP — пунктирные).

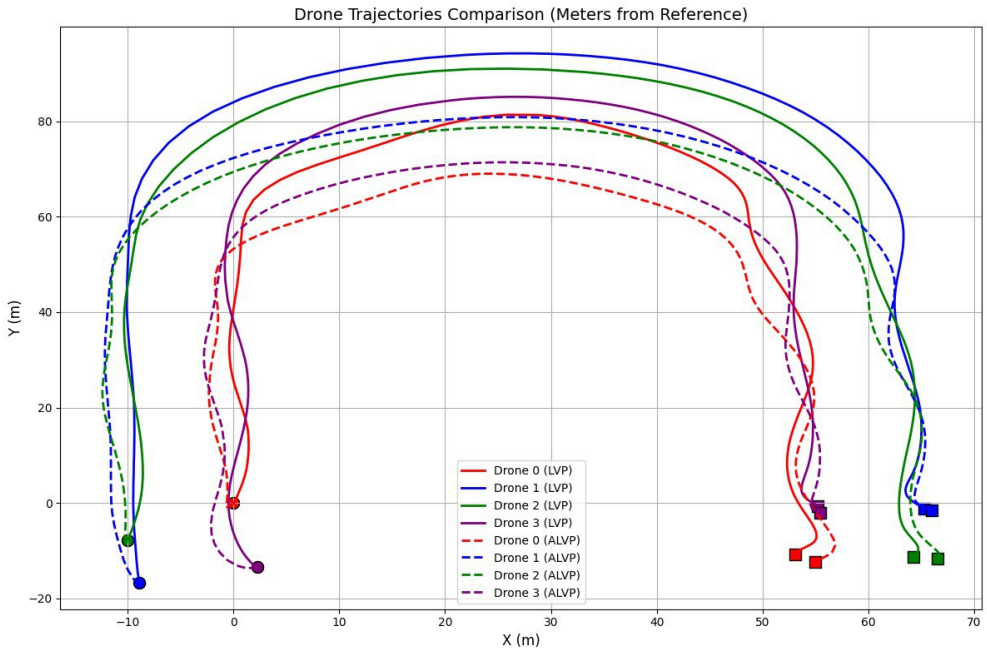

Рисунок 3 — Сравнение траекторий роботов при использовании алгоритмов LVP и ALVP

На Рисунке 4 представлен график изменения ошибки поддержания формации во времени для обоих алгоритмов. Анализ графика показывает, что алгоритм ALVP имеет лучшую скорость сходимости по сравнению с алгоритмом LVP.

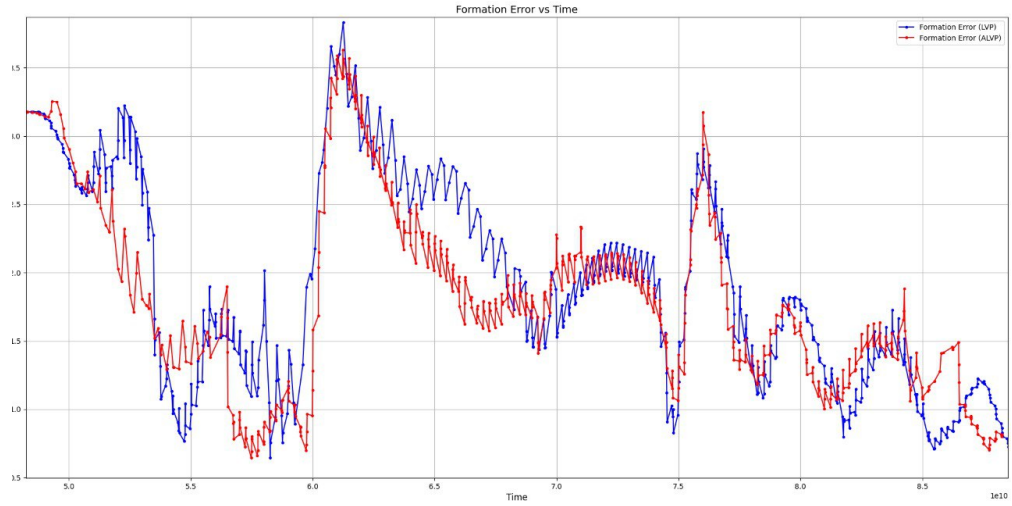

Рисунок 4 — График ошибки поддержания формации во времени

Анализ результатов: Проведенные эксперименты демонстрируют, что разработанный симулятор позволяет:

- Успешно моделировать полет группы роботов под управлением ArduPilot SITL;
- Задавать различные сценарии и параметры для экспериментов;
- Собирать необходимые данные для анализа поведения роя;
- Сравнивать эффективность различных алгоритмов управления формацией (на примере LVP и ALVP);
- Визуализировать результаты для наглядного представления.

Результаты тестирования подтверждают работоспособность симулятора и его пригодность для исследования алгоритмов управления роем роботов в условиях, приближенных к реальным с точки зрения автопилота, но с идеализированной моделью сети.

5 ВЫВОДЫ

В ходе выполнения данной учебной практики были получены следующие основные результаты:

1. Проведен анализ существующих подходов к управлению роем роботов (централизованный, децентрализованный) и существующих программных средств для их симуляции. Выявлены преимущества и недостатки различных подходов.
2. Разработана трехуровневая архитектура системы управления роем роботов, сочетающая централизованное задание макрокоманд с децентрализованной координацией на основе микрокоманд, что позволяет повысить масштабируемость и надежность системы.
3. Сформулированы функциональные и нефункциональные требования к симулятору, необходимому для тестирования и отладки алгоритмов управления в рамках предложенной архитектуры.
4. Реализован программный симулятор на языке Python с использованием ArduPilot SITL и MAVProxy. Симулятор позволяет моделировать полет группы роботов, управлять ими в интерактивном режиме (с геймпада) или в автоматическом режиме экспериментов, логировать данные и визуализировать результаты.
5. Проведено тестирование симулятора на примере задачи поддержания формации роем из четырёх роботов. Продемонстрирована возможность сравнения различных алгоритмов управления (LVP и ALVP), реализованных в ArduPilot.

Ограничения текущей реализации: Основным ограничением текущей версии симулятора является использование идеализированной модели сети для обмена микрокомандами между роботами. Это упрощение может влиять на достоверность симуляции алгоритмов, чувствительных к качеству связи.

Направления дальнейшего развития:

– Реализация детальной модели сети в классе `NetworkSimulator` с учетом расстояний, помех, задержек и потерь пакетов;

– Интеграция с эмуляторами микроконтроллеров или реальным оборудованием;
– Расширение функционала `ExperimentRunner` для поддержки более сложных сценариев и автоматического анализа результатов;
– Интеграция с 3D-визуализаторами (Gazebo/jMAVSim) для более наглядного представления и моделирования сенсоров;
– Повышение масштабируемости симулятора для поддержки большего числа роботов.

6 ЗАКЛЮЧЕНИЕ

В рамках настоящей работы была успешно решена поставленная цель: разработана архитектура системы управления роем роботов с использованием одного пульта и создан симулятор для её исследования.

В ходе работы были выполнены все поставленные задачи:

1. Проанализированы существующие подходы к управлению роем роботов.
2. Изучены существующие инструменты роевой симуляции.
3. Спроектирована трехуровневая архитектура системы управления.
4. Определены функциональные и нефункциональные требования к симулятору.
5. Разработан программный симулятор на Python, интегрированный с ArduPilot SITL, позволяющий моделировать поведение роя в различных режимах.
6. Проведено начальное тестирование симулятора, подтвердившее его работоспособность и пригодность для сравнения алгоритмов управления формацией.

Разработанный симулятор представляет собой гибкий инструмент для дальнейших исследований в области роевого управления роботами, отладки алгоритмов и проведения экспериментов в виртуальной среде перед их реализацией на реальном оборудовании. Полученные результаты и созданный программный продукт могут быть использованы в дальнейших научных и инженерных разработках, связанных с автономными системами.

Ссылка на github репозиторий, содержащий материалы учебной практики: 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. *Амелин К.*, *Амелина Н.*, *Граничин О.*, *Сергеев С.* Децентрализованное групповое управление роем автономных роботов без маршрутизации данных // РОБОТОТЕХНИКА И ТЕХНИЧЕСКАЯ КИБЕРНЕТИКА. — 2021. — Т. 9, 1(30). — С. 42—48.
5. *Ren W.* Consensus based formation control strategies for multi-vehicle systems //. Vol. 2006. — 07/2006. — 6 pp.
6. *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.
7. *Амелин К.*, *Граничин О.* Мультиагентное сетевое управление группой легких БПЛА // НЕЙРОКОМПЬЮТЕРЫ: РАЗРАБОТКА, ПРИМЕНЕНИЕ. — 2011. — № 6. — С. 64—72.
8. *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.