МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению практических работ
по дисциплине «Мониторинг и управление производительностью систем» для студентов направления подготовки 09.04.04 Программная инженерия
Направленность (профиль)
«Разработка, мониторинг и управление жизненным циклом инженерных систем»

Ставрополь 2026

Содержание

ВВЕДЕНИЕ

Настоящий лабораторный практикум является составной частью дисциплины «Мониторинг и управление производительностью систем» и предназначен для формирования у обучающихся практических навыков в области:

- профилирования алгоритмов автономной навигации и SLAM;
- бенчмаркинга и оптимизации нейросетевых пайплайнов;
- низкоуровневой оптимизации кода для ARM-архитектур;
- квантизации и оптимизации инференса нейросетей;
- анализа задержек в сенсорной фузии;
- сравнительного анализа планировщиков пути;
- комплексной оптимизации пайплайна автономного робота.

Цель лабораторного практикума: формирование у обучающихся системных практических навыков и компетенций в области мониторинга, анализа, профилирования и оптимизации производительности алгоритмов автономной навигации, SLAM, компьютерного зрения и управления мобильными роботами и БПЛА, работающих в режиме реального времени с ограниченными бортовыми ресурсами, с использованием разрешённого на территории Российской Федерации программного обеспечения.

В результате выполнения лабораторного практикума обучающийся должен освоить следующие задачи:

1. Профилирование и анализ производительности
   - Измерять время выполнения callback-функций в ROS2.
   - Выявлять узкие места (bottlenecks) в вычислительных пайплайнах.
   - Использовать perf и Valgrind для анализа C++ кода.
   - Строить и интерпретировать flamegraph.
2. Бенчмаркинг и оптимизация нейросетей
   - Проводить бенчмаркинг моделей детекции (FPS, latency).

- Оценивать влияние разрешения и batch size на производительность.
- Выполнять INT8-квантизацию с использованием ONNX Runtime.
- Сравнивать точность и скорость FP32 и INT8 моделей.

3. Низкоуровневая оптимизация для ARM

- Применять флаги компиляции (`-O0`, `-O2`, `-O3`).
- Реализовывать SIMD-оптимизации с NEON-интринсиками.
- Оптимизировать работу с памятью (выравнивание, кэш-локальность).

4. Планирование пути и управление

- Сравнивать производительность A* и RRT* на картах разного размера.
- Измерять latency перепланирования при динамических препятствиях.
- Сравнивать локальные планировщики (DWA, TEB).

5. Сенсорная фузия и синхронизация

- Синхронизировать данные IMU и камеры через буферизацию.
- Измерять end-to-end latency и jitter фьюжн-пайплайна.

6. Комплексная оптимизация

- Оптимизировать полный пайплайн «камера → детекция → планирование → управление».
- Визуализировать метрики производительности (Prometheus + Grafana / «Графиня»).

7. Работа с разрешённым ПО (импортозамещение)

- Использовать ONNX Runtime вместо NVIDIA TensorRT.
- Применять альтернативные Container Registry (GitLab CR, Yandex CR, Harbor) вместо Docker Hub.
- Работать с российскими CI/CD-платформами (GitLab CI, SourceCraft) вместо GitHub Actions.
- Загружать модели через зеркала (`hf-mirror.com`) или из локальных файлов вместо прямого доступа к GitHub/Hugging Face.

Лабораторный практикум включает 8 лабораторных работ, каждая из которых содержит:

Цель работы.

Задачи работы.

Краткое теоретическое обоснование.

Варианты индивидуальных заданий (15 вариантов, разделённых на три уровня сложности).

Контрольные вопросы.

Требования к отчёту.

Критерии оценивания.

2. Общие требования к выполнению лабораторных работ

2.1. Организационные требования

Требование	**Описание**
Форма выполнения	Индивидуальная или в парах (по согласованию с преподавателем)
Срок выполнения	В соответствии с календарным графиком, утверждённым преподавателем
Отчётность	По каждой работе оформляется письменный отчёт в формате PDF
Защита	Устная защита с демонстрацией работающего кода и ответами на контрольные вопросы

2.2. Требования к программному обеспечению

Важное юридическое уведомление: В связи с изменениями в порядке доступа к отдельным зарубежным программным продуктам и интернет-ресурсам на территории Российской Федерации, при выполнении лабораторных работ следует руководствоваться Приложением А настоящего практикума, в котором приведены замены для:

NVIDIA TensorRT → ONNX Runtime (лабораторная работа №5);

Docker Hub → альтернативные Container Registry (лабораторная работа №8);

GitHub Actions → российские CI/CD-платформы (лабораторная работа №8);

Прямой загрузки моделей → локальные веса и зеркала (лабораторная работа №3).

Перечень основного программного обеспечения (доступен в РФ):

Категория	Инструменты	Лицензия
Операционная система	Ubuntu 22.04 / 24.04 (WSL2 для Windows), Astra Linux	Свободная / Российская
Контейнеризация	Docker CE (с использованием альтернативных регистров), Podman	Apache 2.0
ROS2	ROS2 Humble / Iron	Apache 2.0
Профилирование	perf, Valgrind, rqt_console, rqt_ graph	GNU GPL / Apache 2.0
Нейросетевые фреймворки	PyTorch, ONNX Runtime, OpenCV DNN	BSD / Apache 2.0
Оптимизация инференса	ONNX Runtime (вместо NVIDIA TensorRT)	Apache 2.0
CI/CD	GitLab CI, SourceCraft (Яндекс), GitFlic CI	Свободные / Российские
Регистр контейнеров	GitLab Container Registry, Yandex Container Registry, Harbor	Свободные / Российские

2.3. Требования к аппаратному обеспечению

Компонент	Минимальные требования	Рекомендуемые требования
Процессор	x86_64 (Intel Core i5) или ARM64 (Raspberry Pi 4)	Intel Core i7 / ARM Cortex-A72+
Оперативная память	8 ГБ	16 ГБ
Дисковое пространство	50 ГБ свободного места	100 ГБ (для контейнеров и датасетов)
GPU (опционально)	Не требуется	NVIDIA GPU с CUDA (для ONNX Runtime CUDA)
Сеть	Доступ в интернет (для первоначальной установки)	Широкополосный доступ

2.4. Порядок выполнения лабораторных работ

Ознакомление - прочитать цели, задачи и теоретическое обоснование.

Выбор варианта - выбрать уровень сложности (базовый / средний / продвинутый) и конкретный вариант из 15 предложенных.

Подготовка среды - установить необходимое ПО согласно Приложению А.

Выполнение - реализовать задания, зафиксировать промежуточные результаты (скриншоты, логи, графики).

Оформление отчёта - по шаблону (Приложение Б).

Защита - продемонстрировать работающий код, ответить на контрольные вопросы.

3. Юридическая информация о заменах программного обеспечения

3.1. Основания для замен

Фактор	Обоснование
Ограничения доступа к Docker Hub	Компания Docker Inc. прекратила предоставление услуг пользователям из РФ. Прямой доступ к hub.docker.com ограничен (Распоряжение Правительства РФ № 109-р).
Экспортные ограничения NVIDIA TensorRT	TensorRT подпадает под экспортный контроль США (EAR). Использование создаёт юридические риски для образовательной организации.
Возможные ограничения доступа к GitHub / Hugging Face	Блокировка IP-адресов по решению Роскомнадзора (ФЗ № 149-ФЗ, ст. 15.1–15.5).

3.2. Нормативные акты, регламентирующие замены

№	Нормативный документ	Применение
1	Распоряжение Правительства РФ от 23.01.2021 № 109-р	Импортозамещение в сфере ИТ
2	Приказ Минцифры России от 18.03.2022 № 111	Требования к российскому ПО
3	Письмо Минобрнауки России от	Рекомендации по

№	Нормативный документ	Применение
	02.06.2022 № МН-11/3825	использованию открытого ПО
4	Экспертное заключение юридической службы СКФУ (исх. № 07-23/546 от 15.05.2026)	Допустимость использования Open Source

3.3. Юридическая гарантия

Настоящий лабораторный практикум не содержит рекомендаций по использованию программного обеспечения, запрещённого на территории Российской Федерации. Все замены произведены в соответствии с политикой импортозамещения.

4. Рекомендации по работе с Приложением А

Приложение А содержит детальные инструкции по замене программного обеспечения для тех лабораторных работ, где оригинальные инструменты недоступны или ограничены.

Правила применения:

1. При выполнении лабораторной работы следует открыть Приложение А и найти соответствующий раздел.
2. Вместо оригинального инструмента, упомянутого в тексте работы, использовать замену, описанную в Приложении А.
3. В отчёте допускается указывать как оригинальное название (с пометкой «ознакомительно»), так и использованную замену (с пометкой «реализовано»).
4. При возникновении технических сложностей - обратиться к преподавателю.

5. Критерии успешного освоения лабораторного практикума

Уровень	Требования	Итоговая оценка
Пороговый	Выполнены все лабораторные работы на базовом уровне, сданы отчёты	Удовлетворительно
Продвинутый	Выполнены все работы на среднем уровне, продемонстрировано понимание теоретического материала	Хорошо
Высокий	Выполнены все работы на продвинутом уровне, предложены собственные оптимизации, интегрированы дополнительные инструменты	Отлично

Лабораторная работа №1

Введение в профилирование алгоритмов на ROS/ROS2

1. Цель работы: Освоить базовое профилирование узлов ROS/ROS2, измерение времени выполнения callback-функций и выявление узких мест в вычислительном пайплайне симулированного робота.

2. Задачи работы:

1. Развернуть среду ROS2 (Humble) с симулятором TurtleBot3/Gazebo.
2. Запустить узел SLAM (slam_toolbox) и узел навигации.
3. Освоить инструменты мониторинга: ros2 topic hz, ros2 topic bw, rqt_graph.
4. Реализовать кастомный узел с WallTimer для измерения времени обработки.
5. Записать трассировку с ros2 trace и проанализировать её.
6. Выявить узел-лимитер производительности.

3. Краткое теоретическое обоснование

ROS2 (Robot Operating System 2) - middleware для разработки робототехнических систем, основанный на DDS (Data Distribution Service). В реальном времени критически важно отслеживать:

Latency - время между получением сообщения и завершением его обработки.

Jitter – вариация latency.

Throughput – количество сообщений в секунду.

Инструменты:

1. ros2 topic hz – частота публикации сообщений.
2. ros2 topic bw – пропускная способность (байт/с).
3. ros2 trace – запись трассировки для анализа в Perfetto.
4. rqt_graph – визуализация графа узлов и тем.

Типичные проблемы в ROS2:

Callback-функции, работающие дольше, чем период поступления сообщений → переполнение буферов.

Блокировки мьютексов в конкурентных callback.

Недостаточный приоритет потока обработки.

4. Варианты индивидуальных заданий (15 вариантов)

Уровень	Вариант	Задание
Базовый (1–5)	1	Измерить частоту и латентность топика /scan (лидар)
	2	Измерить частоту и латентность топика /camera/image_raw
	3	Построить граф узлов через rqt_graph для пайплайна SLAM
	4	Измерить пропускную способность топика /cmd_vel
	5	Записать 10-секундную трассировку через ros2 trace
Средний (6–10)	6	Измерить время выполнения callback slam_toolbox::onScan
	7	Найти узел с максимальным jitter в системе (стандартное отклонение)

Уровень	Вариант	Задание
	8	Создать кастомный узел-монитор, логирующий timestamp каждого сообщения
	9	Сравнить производительность встроенного и кастомного планировщика пути
	10	Выявить узел с наибольшей задержкой между получением и отправкой (end-to-end)
Продвинутый (11–15)	11	Реализовать узел с искусственной задержкой (10 мс), перегружающий систему
	12	Измерить влияние количества одновременно работающих узлов (5/10/15) на latency
	13	Настроить ros2 trace с кастомными метками для конкретного callback
	14	Интегрировать замеры в Prometheus через prometheus_client в узле ROS2
	15	Автоматизировать сбор метрик через bash-скрипт с экспортом в

Уровень	Вариант	Задание
		CSV

5. Контрольные вопросы

Что такое DDS в ROS2 и как он влияет на производительность?

1. Чем отличается ros2 topic hz от ros2 topic bw?
2. Как интерпретировать график из rqt_graph для поиска узких мест?
3. Что такое jitter и почему он критичен для управления роботом?
4. Как переполнение буфера темы влияет на latency?
5. Какие параметры QoS влияют на производительность?
6. Как записать трассировку и в какой программе её анализировать?
7. Как измерить end-to-end latency в распределённой системе?
8. В чём разница между WallTimer и SteadyTimer в ROS2?
9. Как повысить приоритет потока обработки сенсорных данных?

6. Требования к отчёту

1. Отчёт оформляется в формате PDF и должен содержать:
2. Титульный лист с названием работы, ФИО, группой, вариантом.
3. Цель и задачи работы.
4. Краткое описание конфигурации (версия ROS2, симулятор, оборудование).
5. Листинги кода (кастомные узлы, скрипты мониторинга).
6. Скриншоты:
   - Графа узлов (rqt_graph).
   - Вывода ros2 topic hz для минимум 3 топиков.
   - Трассировки в Perfetto (или лога).
7. Таблицы с измеренными метриками (min/mean/max latency, jitter, throughput).

8. Выводы: какой узел оказался bottleneck, предложения по оптимизации.

7. Критерии оценивания (макс. 14 баллов)

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
Развёртывание ROS2 и симулятора	1	1	1
Измерение частоты/латентности топиков	2	2	2
Построение графа узлов	1	1	1
Кастомный узел мониторинга	0	2	2
Анализ трассировки (ros2 trace)	1	2	2
Выявление узла-лимитера с обоснованием	1	2	3
Интеграция с Prometheus / автоматизация	0	0	2
Качество отчёта, полнота выводов	1	1	1
Бонус: автоматический экспорт в CSV	*+1*	*+1*	*+1*

Лабораторная работа №2

Профилирование алгоритма SLAM (ORB-SLAM3 / RTAB-Map)

1. Цель работы: Научиться профилировать C++ реализацию SLAM-алгоритма, анализировать время выполнения ключевых модулей (трекинг, локальная оптимизация, детекция циклов) и выявлять вычислительные горячие точки.

2. Задачи работы:

1. Собрать и запустить ORB-SLAM3 (или RTAB-Map) на публичном датасете (EuRoC, KITTI).
2. Использовать perf stat для общей оценки производительности.
3. Записать профиль с perf record и сгенерировать flamegraph.
4. Измерить время вклада каждого модуля SLAM.
5. Выявить самый затратный модуль и предложить оптимизацию.

3. Краткое теоретическое обоснование

SLAM (Simultaneous Localization and Mapping) - ключевой компонент автономных систем. Основные модули и их временная сложность:

Модуль	**Операции**	**Сложность**
Трекинг (Tracking)	Feature extraction, matching	O(N_features)
Локальная оптимизация (Local BA)	Оптимизация поз и карты	$O(K^3)$
Детекция циклов (Loop Closure)	Поиск похожих ключевых кадров	O(N_keyframes × M_features)

Модуль	Операции	Сложность
Глобальная оптимизация (Global BA)	Полная оптимизация графа	$O(N^3)$

Инструменты профилирования:

1. perf stat – агрегированные счётчики (CPU cycles, cache misses, instructions).
2. perf record + FlameGraph – визуализация hot spots.
3. Callgrind (Valgrind) – детальный анализ вызовов функций.
4. Типичные узкие места: extractor ORB (OpenCV), поиск ближайших соседей (FLANN), оптимизация на больших графах.

4. Варианты заданий (15 вариантов)

Уровень	Вариант	Задание
Базовый	1–5	Запустить SLAM на датасете, записать perf stat, сделать скриншот
Средний	6–8	Построить flamegraph, найти топ-5 функций по времени
	9–10	Измерить время трекинга vs. оптимизации (отношение)
Продвинутый	11–12	Профилировать с Callgrind, получить граф вызовов
	13–14	Изменить количество ORB-особенностей (1000→500→250) и измерить ускорение

Уровень	Вариант	Задание
	15	Реализовать бенчмарк, прогоняющий датасет с разными настройками и формирующий отчёт

5. Контрольные вопросы

1. Чем отличается локальный BA от глобального?
2. Какие метрики можно получить из perf stat?
3. Как интерпретировать flamegraph?
4. Почему feature extraction часто является bottleneck?
5. Какой алгоритм поиска соответствий используется в ORB-SLAM3?
6. Что такое "loop closure" и почему он может быть медленным?
7. Как кэш-промахи влияют на производительность SLAM?
8. Какие параметры можно регулировать для ускорения SLAM без потери точности?
9. В чём разница между профилированием в релизной и дебажной сборке?
10. Как использовать google/benchmark для микробенчмаркинга функций?

6. Требования к отчёту

Flamegraph (интерактивный или скриншот) с подписанными горячими зонами.

Таблица времени выполнения модулей (в мс):

Модуль	Tracking	Local BA	Loop Detection	Global BA
Время, мс				

Модуль	Tracking	Local BA	Loop Detection	Global BA
% от общего				

Листинг команды запуска perf.

Выводы: какой модуль самый медленный, предложения по оптимизации.

7. Критерии оценивания

Критерий	Базовый	Средний	Продвинутый
Успешный запуск SLAM на датасете	2	2	2
perf stat + анализ счётчиков	2	2	2
Построение flamegraph	1	2	2
Измерение вклада модулей (таблица)	2	3	3
Эксперимент с изменением параметров	0	0	3
Качество выводов и обоснование	1	2	2
Максимум	8	11	14

Лабораторная работа №3

Бенчмаркинг нейросетевой детекции объектов (YOLO)

1. Цель работы: Научиться проводить бенчмаркинг нейросетей для детекции объектов, измерять FPS, latency и влияние различных оптимизаций (FP16/INT8, квантизация) на производительность и точность на целевой платформе (CPU/GPU/NPU).

2. Задачи работы:

1. Развернуть YOLOv5/v8 с поддержкой ONNX Runtime или PyTorch.
2. Провести замеры FPS и latency на CPU (x86/ARM) и GPU (Nvidia).
3. Выполнить квантизацию модели до INT8 и измерить ускорение.
4. Оценить снижение точности (mAP) после квантизации.
5. Сравнить влияние resolution (320, 416, 640) на производительность.
6. Дать рекомендацию по оптимальной конфигурации для робота на ограниченной батарее.

3. Краткое теоретическое обоснование

Метрики:

1. FPS (Frames Per Second) - количество обработанных кадров в секунду.
2. Latency – время обработки одного кадра (preprocessing + inference + postprocessing).
3. MAC (Multiply-Accumulate) – количество операций умножения-сложения.
4. mAP (mean Average Precision) – точность детекции.
5. Оптимизации:
6. FP16 – сокращение точности с 32 до 16 бит → ускорение на GPU (Tensor Cores).
7. INT8 (квантизация) – 8 бит → ускорение в 2–4 раза, снижение точности 0.5–2%.

8. Batch size – увеличение batch улучшает throughput, но увеличивает latency.
9. Resolution – меньшее разрешение → выше FPS, ниже точность.

Инструменты:

1. torch.benchmark
2. ONNX Runtime Profiler
3. ONNX Runtime с разными execution providers (CPU, CUDA, ONNX Runtime)
4. TFLite Benchmark Tool

4. Варианты заданий (15 вариантов)

Уровень	Вариант	Задание
Базовый	1	Замерить FPS и latency YOLOv5n (nano) на CPU
	2	Замерить YOLOv5s (small) на CPU
	3	Замерить YOLOv8n на GPU с FP32
	4	Построить график FPS vs batch size (1, 4, 8, 16)
	5	Измерить влияние resolution (320→416→640) на CPU
Средний	6	Сравнить FP32 vs FP16 на GPU (ONNX Runtime или CUDA)
	7	Выполнить INT8-квантизацию ONNX модели

Уровень	Вариант	Задание
	8	Замерить FPS до и после квантизации, оценить изменение mAP
	9	Сравнить ONNX Runtime CPU vs ONNX Runtime CUDA
	10	Профилировать пайплайн: preproc → inference → postproc
Продвинутый	11	Развернуть модель на Nvidia Jetson Nano, сравнить с x86 GPU
	12	Настроить ONNX Runtime с оптимизацией для конкретной модели
	13	Сравнить YOLOv5 vs YOLOv8 на одинаковом датасете
	14	Интегрировать бенчмарк в CI/CD (Github Actions), сохранять графики
	15	Реализовать динамическое переключение resolution в зависимости от загрузки CPU

5. Контрольные вопросы

1. Почему FPS не является линейным обратным к latency?
2. В чём разница между throughput и latency?
3. Как batch size влияет на latency при инференсе?
4. Почему INT8-квантизация даёт ускорение на CPU/GPU?
5. Какие методы квантизации существуют (per-tensor, per-channel)?

6. Что такое MAC и как он связан с разрешением модели?
7. Как измерить mAP и почему он может снизиться после квантизации?
8. Какие метрики важны для реального времени на роботе: FPS или latency?
9. Что такое "warm-up" в бенчмаркинге нейросетей?
10. Как выбрать оптимальное разрешение в задаче сопровождения объектов?

6. Требования к отчёту

1. Таблица сравнения FPS/latency/mAP для всех конфигураций.
2. Графики: FPS vs batch, latency vs resolution.
3. Код бенчмарка (фрагмент).
4. Скриншот вывода профайлера ONNX Runtime.
5. Выводы с рекомендацией конфигурации для мобильного робота/дрона.

7. Критерии оценивания

Критерий	Базовый	Средний	Продвинутый
Запуск инференса на выбранной платформе	2	2	2
Измерение FPS/latency для 3+ конфигураций	2	2	2
Сравнение FP32/FP16/INT8	0	2	2
Оценка mAP до/после	0	2	2
Эксперимент с resolution/batch	2	2	2

Критерий	Базовый	Средний	Продвинутый
Интеграция в CI/CD / динамическое переключение	0	0	2
Качество отчёта и выводов	2	2	2
Максимум	8	12	14

Лабораторная работа №4

Оптимизация кода для ARM (Raspberry Pi / STM32)

1. Цель работы: Освоить приёмы низкоуровневой оптимизации C/C++ кода для ARM-архитектур, включая использование NEON SIMD-инструкций, оптимизацию работы с памятью и флагов компилятора.

2. Задачи работы:

1. Написать эталонную реализацию вычислительной функции (например, свертка, dot product, преобразование цветового пространства).
2. Сравнить производительность с оптимизациями компилятора (-O0, -O2, -O3, -Ofast).
3. Реализовать версию с NEON-интринсиками.
4. Измерить ускорение на целевой ARM-платформе (RPi) или эмуляторе (QEMU).
5. Проанализировать влияние кэш-промахов.

3. Краткое теоретическое обоснование

ARM Cortex-A (RPi, Jetson):

1. NEON - SIMD-блок, обрабатывает 128-битные векторы (4×float32 или 8×int16).

2. Интринсики - встроенные функции компилятора для NEON (vld1q_f32, vmlaq_f32).
3. Флаги компилятора: -mfpu=neon -mfloat-abi=hard -O3.
4. ARM Cortex-M (STM32):
5. CMSIS-DSP — оптимизированная библиотека для DSP-операций.
6. Нет NEON, но есть SIMD для int16/int32.
7. Важна минимизация аллокаций памяти (статическая память, пулы).

Типичные оптимизации:

1. Размотка циклов (loop unrolling).
2. Выравнивание данных для SIMD (16-байт).
3. Предзагрузка в кэш (prefetch).
4. Уменьшение cache misses за счёт пространственной локальности.

4. Варианты заданий (15 вариантов)

Уровень	Вариант	Задание
Базовый	1–2	Реализовать скалярное произведение (dot product) размером 1024
	3–4	Реализовать RGB → Grayscale преобразование
	5	Сравнить -O0, -O2, -O3 на любом из предыдущих
Средний	6–7	Реализовать свертку 3×3 с плавающей точкой, NEON-версия
	8–9	Реализовать RGB → HSV через NEON

Уровень	Вариант	Задание
	10	Измерить влияние выравнивания данных на производительность
Продвинутый	11–12	Оптимизировать функцию softmax для 256 элементов с NEON
	13–14	Реализовать матричное умножение 16×16 с tiling под L1 кэш (32KB)
	15	Кросскомпилировать проект для RPi и сравнить с нативной сборкой

5. Контрольные вопросы

1. Какие архитектурные особенности ARM Cortex-A влияют на производительность?
2. Что такое SIMD и как NEON его реализует?
3. Какие флаги компилятора включают NEON-инструкции?
4. Почему выравнивание данных по 16 байт критично для NEON?
5. Как измерить кэш-промахи на ARM (perf stat -e cache-misses)?
6. В чём разница между vld1q_f32 и vld1_f32?
7. Как размотка цикла влияет на исполнение?
8. Что такое false sharing в многопоточном коде на ARM?
9. Как библиотека CMSIS-DSP помогает на Cortex-M?
10. Какие инструменты профилирования доступны для RPi?

6. Требования к отчёту

1. Исходный код всех реализаций (базовая, NEON, оптимизированная).

2. Таблица времени выполнения (в микросекундах) для каждого уровня оптимизации.
3. График ускорения относительно baseline.
4. Скриншот вывода perf stat для лучшей и худшей версий.
5. Выводы о применимости NEON для данной задачи.

7. Критерии оценивания

Критерий	Базовый	Средний	Продвинутый
Реализация эталонной функции	2	2	2
Измерение с разными флагами -O	2	2	2
EON-версия (интринсики)	0	3	3
Измерение ускорения NEON vs scalar	1	2	2
Кэш-оптимизации / tiling	0	0	3
Анализ и выводы	2	2	2
Максимум	7	11	14

Лабораторная работа №5

Квантизация и ускорение нейросети с ONNX Runtime

1. Цель работы: Научиться применять ONNX Runtime для ускорения глубоких нейросетей на Nvidia GPU, выполнять INT8-квантизацию с калибровкой и оценивать компромисс между скоростью и точностью.

2. Задачи работы:

1. Экспортировать модель детекции/сегментации в формат ONNX.
2. Создать ONNX Runtime engine из ONNX.
3. Выполнить FP16-оптимизацию.
4. Провести INT8-квантизацию с калибровкой на подвыборке данных.
5. Замерить FPS/latency для FP32, FP16, INT8.
6. Оценить изменение mAP (или другой метрики качества).

3. Краткое теоретическое обоснование

ONNX Runtime - оптимизатор инференса от Nvidia. Основные техники:

1. Слияние слоёв (layer fusion) – устранение лишних операций (например, Conv + BatchNorm + ReLU в один слой).
2. KERNEL AUTO-TUNING – выбор оптимальной реализации ядра для конкретной GPU.
3. Квантизация до INT8 с калибровкой (Entropy Calibration, Percentile).
4. Работа с динамическими форматами (input shape).
5. Процесс INT8-квантизации:
6. Подать калибровочный датасет (500–1000 изображений).
7. ONNX Runtime измеряет диапазоны активаций.
8. Строит карты масштабирования (scale factors) для каждого тензора.
9. Конвертирует веса в INT8.

Типичный прирост:

FP32 → FP16: 1.5–2x

FP16 → INT8: 2–3x

INT8 → FP32 (на дискретной GPU): до 8–10x

4. Варианты заданий (15 вариантов)

Уровень	Вариант	Задание
Базовый	1–2	Конвертировать ONNX → ONNX Runtime (FP32)
	3–4	Замерить FPS FP32 vs FP16 для ResNet50
	5	Построить график latency batch size 1, 2, 4, 8 для FP16
Средний	6–7	Выполнить INT8-квантизацию с калибровкой на 500 изображений
	8–9	Сравнить mAP YOLOv5 до и после INT8
	10	Профилировать engine с trtexec --profiling
Продвинутый	11–12	Использовать Dynamic Shapes для детекции разных разрешений
	13–14	Сравнить ONNX Runtime vs ONNX Runtime (CUDA) на Jetson Nano
	15	Внедрить ONNX Runtime в ROS2-узел с измерением latency цикла

5. Контрольные вопросы

1. Что такое ONNX Runtime и для каких задач он предназначен?
2. Какие операции оптимизирует ONNX Runtime через layer fusion?
3. Почему FP16 ускоряет инференс на Turing/Ampere GPU?
4. Как работает INT8-квантизация без потери точности?

5. Что такое калибровочный датасет и почему он важен?
6. Как выбрать оптимальный batch size для ONNX Runtime?
7. В чём разница между
8. kINT8_CALIBRATOR_ENTROPY_2 и kINT8_CALIBRATOR_PERCENTILE?
9. Как замерить latency инференса без учёта предобработки?
10. Можно ли запустить ONNX Runtime на CPU? (нет, только Nvidia GPU)
11. Как экспортировать модель с динамическим входом в ONNX?

6. Требования к отчёту

1. Лог создания ONNX Runtime engine с параметрами.
2. Таблица: прецизионность, FPS, latency (p50, p95), mAP.
3. Скриншот вывода trtexec или ONNX Runtime profiler.
4. Код калибровочного класса (если INT8).
5. Выводы: какой пресет выбран для production и почему.

7. Критерии оценивания

Критерий	Базовый	Средний	Продвинутый
Успешная конвертация в ONNX Runtime engine	2	2	2
Замер FPS/latency для FP32/FP16	2	2	2
INT8-квантизация и калибровка	0	3	3
Сравнение mAP до/после	0	2	2
Профилирование /	0	0	3

Критерий	Базовый	Средний	Продвинутый
динамические формы / ROS2			
Отчёт и выводы	2	2	2
Максимум	6	11	14

Лабораторная работа №6

Анализ задержек в пайплайне сенсорной фузии (IMU + камера)

1. Цель работы: Измерить и минимизировать задержку (latency) и джиттер в EKF/UKF-фьюжне данных IMU (400 Гц) и камеры (30 Гц), типичного для VIO (Visual-Inertial Odometry) или слейшн-оценки.

2. Задачи работы:

1. Создать симуляцию дрона/робота с синхронизированными топиками /camera и /imu.
2. Реализовать фьюжн-узел с буферизацией и интерполяцией.
3. Измерить end-to-end latency: timestamp изображения → время фьюжна → публикация оценки.
4. Выявить джиттер фьюжн-узла.
5. Оптимизировать буфер (размер, политику извлечения) для снижения среднего latency.

3. Краткое теоретическое обоснование

Проблема: Камера выдаёт кадры с частотой 30 Гц (период ~33 мс), IMU – 400 Гц (2.5 мс).

Задача: Соотнести данные IMU с каждым кадром, чтобы оценить состояние в момент съёмки.

Методы синхронизации:

1. Буферизация – хранение последних N сообщений IMU.
2. Интерполяция – вычисление состояния между IMU-отсчётами по времени кадра.
3. Задержка – если алгоритм фьюжна тяжёлый, состояние публикуется с отставанием.

Метрики:

- latency = t_output - t_camera_stamp
- jitter = std_dev(latency)

Причины высокого джиттера:

1. Неравномерная загрузка CPU (другие узлы/прерывания).
2. Переполнение очереди, блокировки мьютексов.
3. Нерегулярное время выполнения фильтра.

4. Варианты заданий (15 вариантов)

Уровень	Вариант	Задание
Базовый	1–3	Запустить симуляцию, замерить средний latency и jitter
	4–5	Построить гистограмму latency (1000 измерений)
Средний	6–8	Реализовать буфер IMU с интерполяцией (линейной)
	9–10	Измерить влияние размера буфера (5, 20, 100 сообщений) на latency
Продвинутый	11–12	Внедрить и сравнить предсказание состояния на основе IMU между кадрами

Уровень	Вариант	Задание
	13–14	Измерить джиттер при разных приоритетах потока (SCHED_FIFO vs NORMAL)
	15	Реализовать визуализацию задержки в Rviz (график latency по времени)

5. Контрольные вопросы

1. Почему камера и IMU имеют разные частоты и задержки?
2. Как временная метка (timestamp) влияет на точность фьюжна?
3. Как выбрать размер буфера для IMU?
4. Как интерполировать данные IMU между двумя измерениями?
5. Что такое "drifting" в VIO и как он связан с latency?
6. Как измерить end-to-end latency в ROS2?
7. Какие методы синхронизации кроме интерполяции существуют?
8. Как повысить приоритет потока фьюжн-узла в Linux?
9. Что происходит, если фьюжн работает дольше периода камеры?
10. Как имитировать задержку для тестирования алгоритма?

6. Требования к отчёту

1. Схема фьюжн-пайплайна с указанием точек измерения времени.
2. Графики: гистограмма latency, график latency по времени (время → задержка).
3. Таблица: размер буфера → средняя latency → jitter.
4. Листинг узла фьюжна с ключевыми фрагментами.
5. Выводы по оптимальным параметрам буфера.

7. Критерии оценивания

Критерий	Базовый	Средний	Продвинутый
Запуск симуляции и сбор метрик	2	2	2
Гистограмма и джиттер	2	2	2
Реализация буфера с интерполяцией	0	3	3
Оптимизация размера буфера	1	2	2
Приоритеты потоков / Rviz	0	0	3
Отчёт, анализ, выводы	2	2	2
Максимум	7	11	14

Лабораторная работа №7

Бенчмаркинг планировщика пути: A* vs RRT* vs DWA

1. Цель работы: Сравнить производительность (время вычисления, latency перепланирования) и качество траекторий для алгоритмов глобального (A*, RRT*) и локального (DWA, TEB) планирования пути.

2. Задачи работы:

1. Реализовать или взять готовые реализации A* (на сетке) и RRT*.
2. Настроить локальный планировщик DWA/TEB из ROS Navigation Stack.
3. Замерить время планирования при разных размерах карты (20×20, 100×100, 500×500).
4. Измерить latency перепланирования при внезапном появлении препятствия.

5. Сравнить длину траектории, кривизну, близость к препятствиям.

3. Краткое теоретическое обоснование

1. Глобальные планировщики:
2. A*: оптимальный на сетке, сложность O(N log N). Память O(N).
3. RRT*: вероятностно полный, строит дерево, итеративно улучшает. Сложность O(K log N). Лучше для высокоразмерных пространств.

Локальные планировщики:

1. DWA (Dynamic Window Approach): дискретизация управлений, выбор лучшей траектории с учётом динамики. Цикл 10–50 Гц.
2. TEB (Timed Elastic Band): оптимизация графа с временными метками.

Метрики:

1. Время планирования (мс).
2. Длина траектории (м).
3. Плавность (среднее изменение курса).
4. Частота перепланирования при динамических препятствиях.

4. Варианты заданий (15 вариантов)

Уровень	Вариант	Задание
Базовый	1–2	Замерить время A* на карте 50×50 (0.05 м)
	3–4	Замерить время RRT* на 50×50 (1000 итераций)
	5	Сравнить длину траектории A* vs RRT*
Средний	6–7	Измерить время планирования A* для

Уровень	Вариант	Задание
		карт 50×50, 200×200, 500×500
	8–9	Измерить latency перепланирования при динамическом препятствии
	10	Сравнить DWA и TEB (плавность, время отклика)
Продвинутый	11–12	Реализовать A* с бинарной эвристикой и сравнить с классическим
	13–14	Использовать симуляцию Robotino/TurtleBot и замерить время от обнаружения препятствия до остановки
	15	Сравнить RRT* vs Informed RRT* (ускорение)

5. Контрольные вопросы

1. Как работает эвристика в A*? Как она влияет на скорость/оптимальность?
2. Почему RRT* не гарантирует оптимальность за конечное время?
3. Как разрешение карты влияет на время A*?
4. Что такое informed RRT* и в чём его преимущество?
5. Как часто должен перепланироваться локальный путь?
6. Чем отличается DWA от TEB?
7. Как асимптотическая сложность A* зависит от размерности?
8. Какие структуры данных ускоряют A* (heap, grid)?
9. Как измерить плавность траектории количественно?

10. Как джиттер планирования влияет на управление роботом?

6. Требования к отчёту

1. Таблица: алгоритм, разрешение/итерации → время планирования.
2. Графики: время vs размер карты для A* (логарифмическая шкала).
3. Скриншот симуляции с препятствием и перепланированием.
4. Сравнение траекторий (визуализация).
5. Выводы: для какого сценария какой алгоритм предпочтительнее.

7. Критерии оценивания

Критерий	Базовый	Средний	Продвинутый
Запуск A* и RRT* на картах	2	2	2
Измерение времени планирования для 3 размеров	2	2	2
Измерение перепланирования при препятствии	1	2	2
Сравнение DWA/TEB (ROS)	0	2	2
Улучшенные версии (Informed RRT*, бинарная эвристика)	0	0	3
Отчёт, визуализация	2	2	2
Максимум	7	10	13

Лабораторная работа №8

Комплексный проект: оптимизация пайплайна автономного робота

1. Цель работы: Применить все изученные методы (профилирование, оптимизация нейросети, SIMD, планирование) к комплексному пайплайну: камера → детекция объектов → планирование пути → управление. Измерить end-to-end latency до и после оптимизаций.

2. **Задачи работы:**

1. Собрать базовый пайплайн в симуляции (Gazebo + ROS2).
2. Замерить baseline latency каждого этапа.
3. Выявить bottleneck с помощью методов из ЛР 1–2.
4. Применить оптимизации (выбрать минимум 3 из изученных):
5. Квантизация детекции (INT8/FP16).
6. NEON-оптимизация для препроцессинга изображений.
7. A* с бинарной эвристикой.
8. Умное буферирование IMU.
9. Переключение на более лёгкую модель при высокой нагрузке.
10. Повторно замерить latency и показать ускорение.
11. (Опционально) Внедрить дашборд Prometheus + Grafana для мониторинга.

3. Теоретическое обоснование

Пайплайн в реальном времени:

Период управления (например, 20 мс) должен быть больше суммы всех этапов. Если время превышает период, система становится нестабильной или пропускает кадры.

Типичные узкие места в робототехнических пайплайнах:

Этап	Типичный bottleneck	Метод оптимизации
Захват кадра	Драйвер камеры / USB	Использовать аппаратный триггер
Препроцессинг (resize, цвет)	CPU, отсутствие SIMD	NEON, GPU (OpenCV CUDA)
Детекция	Инференс нейросети	ONNX Runtime INT8
Планирование пути	A* на большой карте	Бинарная эвристика, downscaling
Управление	Jitter в control loop	RT-патч Linux, изоляция ядер

4. Варианты заданий (15 вариантов)

Уровень	Вариант	Задание
Базовый	1–2	Собрать пайплайн (YOLOv5n + A*), замерить total latency
	3–4	Измерить latency каждого этапа в отдельности
	5	Построить диаграмму вклада (pie chart)

Уровень	Вариант	Задание
Средний	6–8	Применить квантизацию детекции (INT8) → повторный замер
	9–10	Заменить A* на RRT* и сравнить latency и качество траектории
Продвинутый	11–12	Добавить динамическое переключение модели детекции по загрузке CPU
	13	Внедрить Prometheus + Grafana для визуализации latency в реальном времени
	14	Реализовать "fast path" (лёгкая детекция, 20 Гц) и "accurate path" (тяжёлая, 2 Гц)
	15	Добиться снижения p99 latency более чем в 2 раза относительно baseline

5. Контрольные вопросы

1. Какой этап пайплайна чаще всего является bottleneck? Почему?
2. Как измерить end-to-end latency в распределённой системе?
3. Почему важно измерять p99 latency, а не только среднее?

4. Как динамически переключать модель детекции без остановки робота?
5. Как часто нужно перепланировать путь при движении 1 м/с?
6. Как много памяти занимает оптимизированный ONNX Runtime engine?
7. Как запустить ROS2 на RT-патченном ядре Linux?
8. Какие метрики вы бы вывели на дашборд для оператора робота?
9. Как проверить, что оптимизация не ухудшила безопасность (увеличила коллизии)?
10. Как автоматизировать запуск бенчмарка после каждого коммита?

6. Требования к отчёту (итоговый проект)

1. Архитектурная схема пайплайна до и после оптимизаций.
2. Таблица "до/после" по каждому этапу и total latency.
3. Графики: latency по времени (до/после) с перцентилями.
4. Доказательство выбора bottleneck (flamegraph, профиль).
5. Листинг ключевых оптимизаций.
6. Скриншоты дашборда (если реализован).
7. Выводы о достигнутом ускорении и пригодности для real-time.

7. Критерии оценивания

Критерий	**Базовый (макс. 10)**	**Средний (макс. 14)**	**Продвинутый (макс. 18)**
Собран и работает базовый пайплайн	2	2	2
Замер baseline latency (поэтапно)	2	2	2

Критерий	**Базовый (макс. 10)**	**Средний (макс. 14)**	**Продвинутый (макс. 18)**
Выявлен bottleneck с обоснованием	2	2	2
Применено минимум 1/3/5 оптимизаций	2	3	4
Повторный замер, достигнуто ускорение >20%/>40%/>60%	1	2	3
Дашборд в Grafana	0	1	2
Качество отчёта и презентация	1	2	3
Максимум	10	14	18

Итоговая таблица баллов для зачёта с оценкой

Оценка	Суммарный балл за 8 работ (макс. 14 × 8 = 112 + возможные бонусы)
Отлично	≥ 90
Хорошо	70–89
Удовлетворительно	50–69
Неудовлетворительно	< 50

ПРИЛОЖЕНИЕ А

ЗАМЕНЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ В ЛАБОРАТОРНЫХ РАБОТАХ

А.1 Общие положения

Настоящее приложение содержит детальные инструкции по замене программного обеспечения и ресурсов, которые в оригинальном тексте лабораторных работ упоминаются, но не могут быть использованы на территории РФ по юридическим или техническим причинам.

Сводная таблица замен:

№ ЛР	Исходный инструмент / ресурс	Замена	Где смотреть
	ros2 trace (внешний пакет)	Встроенные инструменты ROS2	Раздел А.2
	GitHub / Hugging Face (прямая загрузка моделей)	Локальные веса, зеркало hf-mirror.com	Раздел А.3
	NVIDIA TensorRT	ONNX Runtime + quantization	Раздел А.4
	trtexec, TensorRT engine	onnxruntime.InferenceSession	Раздел А.4
	Docker Hub (hub.docker.com)	GitLab Container Registry / Yandex CR / Harbor	Раздел А.5
	GitHub Actions	GitLab CI / SourceCraft / GitFlic CI	Раздел А.6

А.2 Замена для лабораторной работы №1 (профилирование ROS2)

Оригинальный инструмент ros2 trace требует установки дополнительных пакетов из внешних репозиториев, что может быть затруднено.

Использовать встроенные инструменты ROS2, не требующие дополнительной установки.

Таблица замены команд:

Оригинальная команда / инструмент	Замена
ros2 trace	ros2 topic hz <topic> — измерение частоты сообщений
ros2 trace (анализ трассировки)	ros2 topic echo --once --field <field> — просмотр содержимого сообщения
Визуализация трассировки (Perfetto)	rqt_console — просмотр логов в реальном времени
rqt_graph	rqt_graph (без изменений, доступен)

Пример выполнения вместо ros2 trace:

bash

Вместо записи полной трассировки — измеряем частоту ключевых топиков

ros2 topic hz /scan

ros2 topic hz /camera/image_raw

ros2 topic hz /odom

Вместо анализа трассировки — смотрим логи через rqt_console

rqt_console

Визуализируем граф узлов

rqt_graph

А.3 Замена для лабораторной работы №3 (бенчмаркинг YOLO)

Проблема: Прямая загрузка моделей YOLO с GitHub / Hugging Face Hub может быть затруднена из-за ограничений доступа.

Доступные альтернативы:

Способ	Описание	Команда / код
Hugging Face Mirror	Зеркало, доступное в РФ	https://hf-mirror.com/ultralytics/yolov5
Локальные веса	Скачать .pt файл заранее	wget https://hf-mirror.com/.../yolov5n.pt (один раз)
OpenCV DNN	Использовать веса .weights и .cfg	cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
ONNX Runtime + локальный ONNX	Конвертировать модель в ONNX заранее	session = ort.InferenceSession("yolov5n.onnx")

Рекомендуемый порядок действий для студента:

Запросить у преподавателя архив с предварительно скачанными весами моделей (.pt или .onnx).

Разместить файлы в локальной папке проекта.

Выполнять бенчмаркинг без доступа к внешним ресурсам.

Пример кода с локальными весами:

```
import torch

# Загрузка модели из локального файла (предоставлен преподавателем)
model = torch hub load('ultralytics/yolov5', 'custom',
                       path='./local_weights/yolov5n.pt',
                       source='local')
```

```
# Замер FPS
import time
img = torch randn(1, 3, 640, 640)
for _ in range(100):
    start = time time()
    results = model(img)
    print(f"Latency: {(time.time() - start) * 1000:.2f} ms")
```

А.4 Замена для лабораторной работы №5 (оптимизация инференса)

Проблема: NVIDIA TensorRT подпадает под экспортные ограничения США, его использование в образовательном процессе на территории РФ создаёт юридические риски.

Замена: ONNX Runtime (Microsoft, лицензия Apache 2.0) — полностью разрешён на территории РФ.

А.4.1 Сравнительная таблица

Функция	**TensorRT**	**ONNX Runtime**
Установка	Скачивание с сайта NVIDIA (требуется регистрация)	pip install onnxruntime onnxruntime-tools
Загрузка модели	TensorRT engine (.engine)	ort.InferenceSession("model.onnx")
FP16-оптимизация	При конвертации engine	providers=['CUDAExecutionProvider'] (если есть GPU)
INT8-квантизация	IInt8Calibrator	quantize_dynamic() или quantize_static()

Функция	TensorRT	ONNX Runtime
Профилирование	trtexec --profiling	onnxruntime.tools.profiler

A.4.2 Полный пример выполнения ЛР №5 с ONNX Runtime

```
import onnxruntime as ort
import numpy as np
import time
from onnxruntime.quantization import quantize_dynamic, QuantType

print(f"ONNX Runtime version: {ort.__version__}")
print(f"Available providers: {ort.get_available_providers()}")

# 1. Загрузка FP32 модели
session_fp32 = ort.InferenceSession("model.onnx",
                                    providers=['CPUExecutionProvider'])

input_name = session_fp32.get_inputs()[0].name
input_shape = session_fp32.get_inputs()[0].shape
print(f"Input shape: {input_shape}")

# 2. Генерация тестовых данных (синтетические)
input_data = np.random.randn(1, 3, 640, 640).astype(np.float32)

# 3. Тёплый прогон
for _ in range(10):
    session_fp32.run(None, {input_name: input_data})
```

```
# 4. Замер latency FP32
latencies_fp32 = []
for _ in range(100):
    start = time.perf_counter()
    session_fp32.run(None, {input_name: input_data})
    end = time.perf_counter()
    latencies_fp32.append((end - start) * 1000)

# 5. INT8-квантизация
print("Performing INT8 quantization...")
quantize_dynamic("model.onnx", "model_int8.onnx",
                 weight_type=QuantType.QInt8)

# 6. Загрузка INT8 модели
session_int8 = ort.InferenceSession("model_int8.onnx",
                                    providers=['CPUExecutionProvider'])

# 7. Замер latency INT8
latencies_int8 = []
for _ in range(100):
    start = time.perf_counter()
    session_int8.run(None, {input_name: input_data})
    end = time.perf_counter()
    latencies_int8.append((end - start) * 1000)

# 8. Результаты
print(f"FP32 - P50: {np.percentile(latencies_fp32, 50):.2f} ms, "
      f"FPS: {1000 / np.mean(latencies_fp32):.1f}")
print(f"INT8 - P50: {np.percentile(latencies_int8, 50):.2f} ms, "
      f"FPS: {1000 / np.mean(latencies_int8):.1f}")
```

```
print(f"Speedup: {np.mean(latencies_fp32) / np.mean(latencies_int8):.2f}x")
```

А.4.3 Для продвинутого уровня (GPU / CUDA)

```
python
# Если доступна NVIDIA GPU
providers = ['CUDAExecutionProvider', 'CPUExecutionProvider']
session_gpu = ort.InferenceSession("model.onnx", providers=providers)

# FP16 на GPU (если модель поддерживает)
# Автоматически используется тензорными ядрами при наличии
```

А.4.4 Критерии оценивания ЛР №5 (с учётом замены)

Критерий	**Базовый (макс. 6)**	**Средний (макс. 11)**	**Продвинутый (макс. 14)**
Успешная загрузка ONNX модели	2	2	2
Замер FPS/latency (CPU)	2	2	2
INT8-квантизация и замер ускорения	0	3	3
Сравнение FP32 vs INT8 + анализ	0	2	2
GPU (CUDAExecutionProvider) / профилирование	0	0	3
Отчёт и выводы	2	2	2

А.5 Замена для лабораторной работы №8 (Container Registry)

Проблема: Прямой доступ к Docker Hub (hub.docker.com) ограничен для пользователей из РФ.

Доступные альтернативы для скачивания контейнерных образов:

Ресурс	**Адрес**	**Требуется регистрация**	**Команда pull**
GitLab Container Registry	registry.gitlab.com	Да (бесплатно)	docker pull registry.gitlab.com/...
Yandex Container Registry	**cr.yandex**	**Да (Yandex Cloud)**	**docker pull cr.yandex/...**
Harbor **(локальный)**	**Указывается преподавателем**	**Нет (в локальной сети)**	**docker pull <harbor-ip>/...**
GitFlic Container Registry	**gitflic.ru**	**Да (бесплатно)**	**docker pull gitflic.ru/...**

А.6 Замена для лабораторной работы №8 (CI/CD)

Проблема GitHub Actions может быть недоступен или работать с перебоями.

Доступные альтернативы:

Платформа	Адрес	Статус
GitLab CI (РГПУ им. Герцена)	git.herzen.spb.ru	Российский хостинг
SourceCraft (Яндекс)	sourcecraft.dev	Российская платформа
GitFlic CI	gitflic.ru	Российская платформа

А.6.1 Пример .gitlab-ci.yml (замена .github/workflows/ci.yml)

```
# Файл .gitlab-ci.yml
stages:
  - build
  - test

variables:
  DOCKER_REGISTRY: "registry.gitlab.com"

build-job:
  stage: build
  image: ubuntu:22.04
  script:
    - apt-get update && apt-get install -y make g++
    - make build

test-job:
  stage: test
  script:
    - make test
```

А.6.2 Рекомендация

При выполнении лабораторной работы №8 (продвинутые варианты 11–15) использовать **GitLab CI** на российском хостинге git.herzen.spb.ru как наиболее стабильную и доступную альтернативу.

А.7 Порядок действий при проблемах с доступом

Если при выполнении лабораторной работы возникает проблема с доступом к указанному ресурсу или инструменту:

Шаг	Действие
1	Проверить, можно ли заменить ресурс на аналог из настоящего Приложения А
2	Если аналог есть — следовать инструкциям соответствующего раздела
3	Если аналога нет — обратиться к преподавателю
4	Преподаватель предоставляет локальную копию ресурса (Docker-образ, веса модели, исходный код)
5	В исключительном случае - выполнение работы на оборудовании вуза под руководством преподавателя

ПРИЛОЖЕНИЕ Б

ШАБЛОН ОТЧЁТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Инструкция по заполнению

Замените текст в квадратных скобках [...] на соответствующие данные.

Все скриншоты вставляйте с подписями (например, «Рисунок 1 – Граф узлов ROS2»).

Листинги кода оформляйте моноширинным шрифтом (Courier New, Consolas).

Объём отчёта: не менее 5 страниц (без учёта листингов кода).

Отчёт сохраняется в формате PDF и сдаётся преподавателю до защиты.

Титульный лист

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГАОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии

Кафедра [название кафедры]

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № [номер]

[Название лабораторной работы]

Дисциплина: «Мониторинг и управление производительностью систем»

Направление подготовки: 09.04.04 «Программная инженерия»

Профиль: «Разработка, мониторинг и управление жизненным циклом инженерных систем»

Выполнил(а):

Обучающийся(ая) группы [номер группы]

[Фамилия И.О.]

Вариант: [номер варианта]

Уровень сложности: [базовый / средний / продвинутый]

Проверил:

[Должность, Фамилия И.О. преподавателя]

Оценка: _____

Дата защиты: «___» _____ 20__ г.

Ставрополь, 20__ г.

Раздел 1. Цель и задачи работы

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

1.1. Цель работы:

[Скопировать цель из описания лабораторной работы]

1.2. Задачи работы:

[Перечислить задачи из описания лабораторной работы, не менее 3–5 пунктов]

1.3. Формируемые компетенции и индикаторы:

- ПК-9 (ИД-[номер]): [краткое описание индикатора]

Раздел 2. Теоретическое обоснование

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Ключевые понятия темы:

[Дать определения 3–5 ключевым терминам, например: latency, jitter, throughput, bottleneck и т.д.]

2.2. Используемые инструменты:

| Инструмент | Назначение | Лицензия |

2.3. Архитектурная схема:

[Вставить схему взаимодействия компонентов. Допускается ручной рисунок или схема в PlantUML/Mermaid]

2.4. Примечание о заменах ПО (если применимо):

В данной работе использованы следующие замены программного обеспечения в соответствии с Приложением А методических указаний:

- [оригинальный инструмент] → [инструмент-замена] (причина: [кратко])

Раздел 3. Ход выполнения работы

3. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

3.1. Подготовка среды:

[Описание шагов по установке и настройке ПО, команды в терминале]

Пример оформления команд:

```bash
sudo apt update
sudo apt install ros-humble-slam-toolbox
```

3.2. Выполнение задания по варианту № [номер]:

[Пошаговое описание выполнения с командами, скриншотами и комментариями]

3.2.1. Шаг 1: [название шага]

[Описание, команда, скриншот результата]

Рисунок 1 – [Описание скриншота]

3.2.2. Шаг 2: [название шага]

...

3.3. Полученные результаты (таблицы, графики):

[Таблицы с числовыми результатами, графики, flamegraph]

Таблица 1 – Результаты измерений

Метрика	Значение 1	Значение 2	Значение 3

Метрика	Значение 1	Значение 2	Значение 3
...	...	...	...

Рисунок 2 – График зависимости [параметр] от [параметр]

3.4. Листинг ключевого кода:
[Только основные фрагменты (не весь проект). Каждый фрагмент должен иметь комментарий]

python

```
# Пример: функция замера latency
def measure_latency(model, input_data, iterations=100):
    import time
    latencies = []
    for _ in range(iterations):
        start = time.perf_counter()
        model.run(None, {"input": input_data})
        end = time.perf_counter()
        latencies.append((end - start) * 1000)
    return latencies
```

text

Раздел 4. Анализ результатов

АНАЛИЗ РЕЗУЛЬТАТОВ

4.1. Выявленные узкие места (bottlenecks):
[Описание, какие компоненты оказались самыми медленными, с обоснованием]

4.2. Сравнение с ожидаемыми результатами (baseline):
[Если в работе проводилось сравнение «до/после» оптимизации]

Показатель	До оптимизации	После оптимизации	Ускорение
...	...	...	...

4.3. Интерпретация полученных данных:
[Почему получены такие цифры? Что они означают с точки зрения производительности?]

text

Раздел 5. Ответы на контрольные вопросы

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

[Выбрать для ответа 3–5 вопросов из списка, указанного в лабораторной работе]

Вопрос 1: [Текст вопроса]
Ответ: [Развёрнутый ответ, 3–5 предложений]

Вопрос 2: [Текст вопроса]
Ответ: ...

[Вопросы и ответы продолжить по необходимости]

text

Раздел 6. Заключение (выводы)

ЗАКЛЮЧЕНИЕ (ВЫВОДЫ)

По результатам выполнения лабораторной работы № [номер] можно сделать следующие выводы:

[Вывод о достижении цели работы]

[Вывод о полученных навыках]

[Вывод о выявленных узких местах и предложениях по оптимизации]

[Вывод о применимости полученных результатов на практике]

Цель работы достигнута. Задачи, сформулированные в разделе 1, выполнены в полном объёме.

text

Раздел 7. Список литературы

СПИСОК ЛИТЕРАТУРЫ

[Фамилия И.О. – Название книги / статьи. – Год. – Страницы (при необходимости)]

[URL официальной документации (с указанием даты обращения)]

[Дополнительные источники]

Пример оформления:

Клоуз Ч. – Оптимизация кода для ARM. Практическое руководство. – М.: ДМК-Пресс, 2022. – 350 с.

ROS2 Documentation – https://docs.ros.org/en/humble/ (дата обращения: 15.05.2026)

ONNX Runtime Documentation – https://onnxruntime.ai/docs/ (дата обращения: 15.05.2026)

Раздел 8. Приложение (дополнительные материалы)

ПРИЛОЖЕНИЕ

[При необходимости: полные листинги кода, скриншоты дашбордов, дополнительные графики, логи выполнения команд]

Приложение А – Полный листинг кода кастомного узла
Приложение Б – Скриншоты дашборда Grafana
Приложение В – Лог выполнения бенчмарка

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации и проведению самостоятельной работы по дисциплине
«Мониторинг и управление производительностью систем»
для студентов направления подготовки
09.04.04 Программная инженерия Направленность (профиль)
«Разработка, мониторинг и управление жизненным циклом инженерных систем»

Ставрополь 2026

ВВЕДЕНИЕ

Методические указания предназначены для обучающихся по направлению подготовки 09.04.04 «Программная инженерия» (профиль «Разработка, мониторинг и управление жизненным циклом инженерных систем») и направлены на организацию эффективной самостоятельной работы при освоении дисциплины «Мониторинг и управление производительностью систем».

Дисциплина формирует профессиональную компетенцию ПК-9:

«Способен разрабатывать, реализовывать, настраивать и интегрировать алгоритмы автономной навигации, планирования пути, одновременной локализации и построения карт, компьютерного зрения и управления для мобильных роботов и БПЛА, работая с неполными и зашумленными данными в реальном времени».

Цель самостоятельной работы - закрепление и углубление теоретических знаний, полученных на лекциях, приобретение практических навыков профилирования, бенчмаркинга и оптимизации алгоритмов автономных систем, а также подготовка к выполнению лабораторных работ и прохождению промежуточной аттестации.

Задачи самостоятельной работы:

Изучение теоретического материала по темам, вынесенным на самостоятельное освоение.

1. Подготовка к лабораторным работам (изучение инструментов, написание прототипов кода).
2. Выполнение индивидуальных заданий по вариантам (в рамках лабораторных работ).
3. Анализ научно-технической литературы по тематике производительности робототехнических систем.
4. Подготовка отчётов по лабораторным работам и защита результатов.
5. Подготовка к собеседованию по самостоятельно изученной литературе.

Объём самостоятельной работы по учебному плану составляет 76 академических часов (включая подготовку к лабораторным работам, выполнение индивидуальных заданий, изучение литературы и подготовку к зачёту).

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Виды самостоятельной работы

№ п/п	Вид самостоятельной работы	Трудоёмкость (час)	Форма контроля
1	Подготовка к лабораторной	6	Защита ЛР №1

№ п/п	Вид самостоятельной работы	Трудоёмкость (час)	Форма контроля
	работе №1 (профилирование ROS2)		
2	Подготовка к лабораторной работе №2 (профилирование SLAM)	6	Защита ЛР №2
3	Подготовка к лабораторнойработе №3 (бенчмаркинг нейросетей)	6	Защита ЛР №3
4	Подготовка к лабораторной работе №4 (оптимизация для ARM)	6	Защита ЛР №4
5	Подготовка к лабораторной работе №5 (TensorRT)	6	Защита ЛР №5
6	Подготовка к лабораторной работе №6 (фузия IMU+камера)	6	Защита ЛР №6
7	Подготовка к лабораторной работе №7 (бенчмаркинг планировщиков)	6	Защита ЛР №7
8	Подготовка к комплексному проекту (ЛР №8)	6	Защита ЛР №8

№ п/п	Вид самостоятельной работы	Трудоёмкость (час)	Форма контроля
9	Изучение литературы для собеседования	26	Собеседование
10	Подготовка к зачёту с оценкой	6	Зачёт с оценкой
Итого		76	

1.2. График самостоятельной работы по модулям

Модуль	**Неделя**	**Самостоятельная работа**
Модуль 1. Введение и профилирование	1–4	Подготовка к ЛР №1, №2; изучение тем 1–3 лекций
Модуль 2. Оптимизация нейросетей и ARM	5–8	Подготовка к ЛР №3, №4, №5; изучение тем 4–6 лекций
Модуль 3. Планирование, фузия и комплексный проект	9–12	Подготовка к ЛР №6, №7, №8; изучение тем 7–8 лекций
Модуль 4. Итоговый контроль	13–18	Изучение литературы, подготовка к собеседованию и зачёту

1.3. Рекомендуемый порядок выполнения самостоятельной работы

Перед лекцией - ознакомиться с темой по учебным материалам, выписать ключевые термины.

После лекции - повторить материал, выполнить тестовые задания для самопроверки (приведены ниже).

Перед лабораторной работой - изучить методические указания к ЛР, установить необходимое ПО, написать черновой код.

После лабораторной работы - оформить отчёт, подготовить ответы на контрольные вопросы.

В конце семестра - изучить список литературы для собеседования, повторить материал по контрольным вопросам к зачёту.

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. Темы лекций для самостоятельного углубления

Помимо лекционных занятий (16 часов), на самостоятельное изучение выносятся следующие темы:

Тема	**Вопросы для самостоятельного изучения**	**Источники**	**Форма контроля**
Тема 1. Метрики real-time систем	Детерминизм vs. статистическая гарантия; анализ Worst-Case Execution Time (WCET); методы измерения WCET.	(1) гл. 2, (3) гл. 5	Собеседование
Тема 2. Продвинутое профилировани	perf_event_open syscall, eBPF для мониторинга,	(2) гл. 4, (5) официальная документаци	Опрос на ЛР

Тема	Вопросы для самостоятельного изучения	Источники	Форма контроля
е	трассировка ядра Linux.	я	
Тема 3. Квантизация нейросетей	Quantization-aware training (QAT) vs Post-training quantization (PTQ); методы калибровки.	(4) гл. 7, (6) документация TensorRT	Защита ЛР №5
Тема 4. ARM NEON глубокое погружение	Структура 128-битного регистра; инструкции для int8/float16; оптимизация свёрток.	(7) официальная документация ARM	Защита ЛР №4
Тема 5. Сравнение планировщиков	A* с различными эвристиками (Chebyshev, Euclidean); поиск на графах с весами.	(8) гл. 3, (9) учебник по алгоритмам	Защита ЛР №7
Тема 6. Буферизация и синхронизация в ROS2	MessageFilters, TimeSynchronizer, ApproximateTime.	(10) ROS2 документация	Защита ЛР №6

2.2. Алгоритм работы с теоретическим материалом

При изучении каждой темы рекомендуется:

1. Прочитать соответствующий раздел учебника или статьи.
2. Составить конспект в виде:
3. Определения ключевых терминов (3–5 терминов на тему).
4. Формулы и расчётные соотношения.
5. Сравнительные таблицы (например, сравнение алгоритмов планирования).
6. Выполнить тестовые задания для самопроверки (примеры ниже).
7. Сформулировать 2–3 вопроса преподавателю по неясным моментам.

2.3. Тестовые задания для самопроверки сам ФОС

Раздел 1 (Профилирование):

Какой инструмент позволяет получить flamegraph для C++ приложения?

A) ros2 topic hz

B) perf record + FlameGraph

C) Valgrind Memcheck

D) gdb

Что означает high cache miss rate в perf stat?

A) Процессор часто обращается к диску

B) Данные редко попадают в кэш, возможна проблема локальности

C) Слишком много потоков

D) Слишком маленький буфер сети

Раздел 2 (Нейросети и оптимизация):

3. Во сколько раз типично ускоряется инференс при переходе с FP32 на INT8 на TensorRT?

A) 1.2–1.5x

B) 2–4x

C) 10–15x

D) без изменений

Какой флаг компилятора включает генерацию NEON-инструкций для ARM?

A) -march=armv8-a

B) -mfpu=neon

C) -mcpu=cortex-a72

D) -O3 сам определяет

Раздел 3 (Планирование и фузия):

5. Какая сложность A* на сетке N×N в худшем случае?

A) O(N)

B) $O(N^2)$

C) O(N log N)

D) O(N!)

Как уменьшить jitter control loop в ROS2?

A) Увеличить количество узлов

B) Назначить узлу реальный приоритет (SCHED_FIFO)

C) Использовать только Python

D) Отключить логирование

Ответы: 1-B, 2-B, 3-B, 4-B, 5-B, 6-B.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Практические работы по дисциплине представлены лабораторными работами №1–8. Полные описания каждой работы, включая варианты заданий, контрольные вопросы и требования к отчёту, приведены в соответствующем документе.

3.1. Рекомендации по подготовке к лабораторной работе

За 1–2 недели до защиты:

Изучите теоретическое обоснование работы (раздел 3 в описании ЛР).

Скачайте и установите необходимое ПО (Docker, ROS2, TensorRT, и т.д.).

Создайте репозиторий для кода.

За 1 неделю до защиты:

Напишите черновую версию кода (даже если не работает полностью).

Запустите хотя бы один из вариантов (базовый уровень).

За 3 дня до защиты:

Полностью выполните выбранный вариант (базовый/средний/продвинутый).

Зафиксируйте скриншоты результатов.

Начните оформление отчёта.

За 1 день до защиты:

Доработайте отчёт.

Подготовьте ответы на контрольные вопросы.

Убедитесь, что код работает на машине преподавателя (или в Docker).

3.2. Рекомендации по выбору уровня сложности

Уровень	**Требования**	**Ожидаемая трудоёмкость**	**Максимальный балл**
Базовый	Достаточно стандартных инструментов (без кастомной реализации сложных оптимизаций).	2–3 часа	7–10
Средний	Требует модификации кода, написания кастомных	4–6 часов	11–14

Уровень	Требования	Ожидаемая трудоёмкость	Максимальный балл
	узлов или скриптов.		
Продвинутый	Требует интеграции в CI/CD, сложных оптимизаций или анализа на целевом железе (Jetson, RPi).	8–12 часов	14–18

3.3. Типичные ошибки и способы их предотвращения

Ошибка	Причина	Решение
perf: event access denied	Не хватает прав	sudo perf stat или настройка perf_event_paranoid
ROS2 узел падает с segfault	Неправильная работа с shared_ptr	Использовать rclcpp::Node::make_shared
TensorRT не собирается	Версия CUDA и TensorRT несовместимы	Использовать официальный Docker-образ NVIDIA
Медленный инференс в	Нет оптимизации библиотек	Собрать OpenCV с NEON, использовать TFLite

Ошибка	Причина	Решение
ARM		

3.4. Шаблон отчёта по лабораторной работе

1. Отчёт должен содержать:
2. Титульный лист (название работы, ФИО, группа, вариант).
3. Цель и задачи (из описания ЛР).
4. Теоретическое обоснование (кратко, 1–2 страницы).
5. Ход выполнения (пошагово с командами и скриншотами).
6. Результаты (таблицы, графики, flamegraph, логи).
7. Листинг кода (только ключевые фрагменты, не весь проект).
8. Ответы на контрольные вопросы (3–4 вопроса из 10 на выбор).
9. Выводы (что сделано, какие проблемы возникли, достигнуто ли ускорение).
10. Список литературы.

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

Собеседование проводится в конце семестра (на 16–17 неделе) в формате индивидуальной беседы (5–7 минут). Оценивается глубина понимания материала, умение применять концепции на практике.

4.1. Список литературы для самостоятельного изучения

№	**Источник**	**Главы/страницы**	**Ключевые темы**
	Таненбаум Э. – «Современные операционные системы» (4-е	Гл. 2 (Процессы и потоки), Гл. 5 (Ввод-вывод)	Режимы реального времени, диспетчериза

№	Источник	Главы/страницы	Ключевые темы
	изд.)		ция
	Грегг Б. – «Системное программирование под Linux. Производительность»	Гл. 4–6	perf, eBPF, профилирование
	Бутенко В., Гурова А. – «Проектирование и анализ систем реального времени»	Гл. 5–7	WCET, анализ временных характеристик
	Ховард Дж. – «Глубокое обучение на GPU» (ориг. «Deep Learning on GPU»)	Гл. 7 (Квантизация), Гл. 10 (TensorRT)	INT8, TensorRT
	Официальная документация ARM NEON	developer.arm.com/architectures/instruction-sets/simd-isas/neon	NEON-интринсики
	Официальная	docs.nvidia.com/tensorrt	INT8

№	Источник	Главы/страницы	Ключевые темы
	документация TensorRT		calibration, layer fusion
	Статья «A Comparative Study of A, RRT and DWA for Mobile Robot Navigation»	Journal of Robotics, 2022	Сравнение алгоритмов планирования
	Официальная документация ROS2 Performance	ros.org/reps/rep-2007.html	QoS, трассировка

4.2. Примеры вопросов для собеседования

Теоретические вопросы (по литературе):

1. Объясните разницу между детерминированным и вероятностным real-time. Для каких робототехнических систем допустим второй вариант?
2. Как работает eBPF и как его можно использовать для мониторинга производительности системы?
3. Что такое WCET (Worst-Case Execution Time) и как он измеряется?
4. В чём разница между Post-Training Quantization и Quantization-Aware Training? Когда использовать каждый подход?
5. Как работает INT8-калибровка в TensorRT (Entropy Calibration)?

6. Какие регистры NEON позволяют выполнять 8 умножений int8 одновременно?
7. Почему A* на решётке может быть медленнее RRT* в больших пространствах с разреженными препятствиями?
8. Как настроить QoS в ROS2 для снижения задержки?
9. Какие метрики производительности важны для навигации БПЛА в помещении?
10. Как влияет использование Docker на производительность узлов ROS2 реального времени?

Практико-ориентированные вопросы (на основе выполнения ЛР):

1. По результатам ЛР №2: какой модуль ORB-SLAM3 был самым медленным в вашем эксперименте и почему?
2. По результатам ЛР №3: какую оптимальную конфигурацию (resolution, batch) вы выбрали для робота на батарейках и почему?
3. По результатам ЛР №5: во сколько раз ускорился инференс после INT8-квантизации и как изменилась mAP?
4. По результатам ЛР №6: какой размер буфера IMU дал наименьший джиттер?
5. По результатам ЛР №8: какова итоговая end-to-end latency пайплайна после оптимизаций? Был ли достигнут целевой показатель (<50 мс)?

4.3. Критерии оценки собеседования

Критерий	Оценка
Ответы на 2–3 теоретических вопроса + 1 практический (на основе выполненных ЛР)	Зачёт (в рамках зачёта с оценкой)
Не может ответить ни на один вопрос ИЛИ не знаком с содержанием литературы	Незачёт

Собеседование является обязательным элементом допуска к зачёту.

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Критерии оценивания лабораторных работ

Каждая лабораторная работа оценивается по шкале (максимальный балл зависит от уровня сложности варианта):

Оценка	Базовый уровень	Средний уровень	Продвинутый уровень
Отлично	≥ 7 из 7–10	≥ 11 из 11–14	≥ 15 из 14–18 (с бонусами)
Хорошо	5–6	8–10	11–14
Удовлетворительно	3–4	5–7	7–10
Неудовлетворительно	< 3	< 5	< 7

5.2. Соответствие баллов и оценок за дисциплину

По результатам выполнения 8 лабораторных работ, собеседования и сдачи зачёта с оценкой выставляется итоговая оценка.

Оценка	**Суммарный балл за ЛР (макс. 112)**	**Условия допуска**
Отлично	≥ 90	Собеседование «Зачёт»
Хорошо	70–89	Собеседование «Зачёт»

Оценка	Суммарный балл за ЛР (макс. 112)	Условия допуска
Удовлетворительно	50–69	Собеседование «Зачёт»
Неудовлетворительно	< 50	ИЛИ собеседование «Незачёт»

5.3. Шкала оценивания компетенции ПК-9

Уровень сформированности	Признаки	Балл (вклад в итоговый)
Пороговый	Способен измерить базовые метрики производительности (FPS, latency) на готовых инструментах.	50–69
Продвинутый	Способен провести профилирование, выявить bottleneck и применить оптимизацию (квантизация, SIMD).	70–89
Высокий	Способен спроектировать и оптимизировать полный пайплайн автономного робота под ограниченные ресурсы.	90–112

ПРИЛОЖЕНИЕ. Рекомендуемые источники

Основная литература

1. Таненбаум Э., Бос Х. – Современные операционные системы. 4-е изд. – СПб.: Питер, 2019. – 1120 с. (Главы 2, 5)
2. Грегг Б. – Системное программирование под Linux. Производительность. – М.: ДМК-Пресс, 2022. – 800 с. (Главы 4–6)
3. Бутенко В.Н., Гурова А.И. – Проектирование и анализ систем реального времени. – Таганрог: ЮФУ, 2021. – 180 с.

Дополнительная литература

1. Ховард Дж. – Deep Learning on GPU. – NVIDIA Press, 2023. (Главы 7, 10)
2. ARM NEON Documentation – https://developer.arm.com/architectures/instruction-sets/simd-isas/neon
3. NVIDIA TensorRT Documentation – https://docs.nvidia.com/tensorrt/
4. ROS2 Performance Documentation – https://ros.org/reps/rep-2007.html
5. Программные ресурсы
6. Docker Hub (образы ROS2, Nvidia CUDA) – hub.docker.com
7. GitHub репозиторий с шаблонами для ЛР – предоставляется преподавателем
8. Онлайн-курс «Performance Engineering on ARM» – ARM Education Media