Санкт–Петербургский государственный университет

Отчет по преддипломной производственной практике “Разработка ПО управления системой неразрушающего контроля”

Уровень образования: **магистратура**
Направление: **02.04.03 “Математическое обеспечение и администрирование информационных систем ”**
Основная образовательная программа: **ВМ.5665.2019 “Математическое обеспечение и администрирование информационных систем”**

Студент 2 курса, группы 23.М04-мм:
Сатановский А.Д.

Научный руководитель:
к.ф-м.н., доцент кафедры системного программирования
Луцив Д.В.

Рецензент:
Ведущий инженер-программист “НИПК Электрон”
Департамента неразрушающего контроля
Попович И.В.

г. Санкт – Петербург
2025

Содержание

1 Введение

Неразрушающий контроль – это метод контроля надёжности основных рабочих свойств и параметров объекта или его отдельных элементов без необходимости вывода объекта из эксплуатации или его демонтажа.

Основные цели неразрушающего контроля в промышленности:

- Выявление опасных дефектов изделий
- Обеспечение безопасности эксплуатации оборудования
- Предупреждение аварийных ситуаций

Области применения включают:

- Промышленное производство
- Энергетику
- Транспортную инфраструктуру
- Авиационно-космическую отрасль

Одним из перспективных направлений является радиографический метод неразрушающего контроля, который активно развивается на базе НИПК "Электрон".

В рамках данного проекта реализованы и протестированы следующие компоненты:

- Система позиционирования – программное обеспечение для управления перемещением аппаратного комплекса
- Система безопасности – ПО, обеспечивающее безопасную работу оборудования
- Пульт управления системой позиционирования – интерфейс для взаимодействия оператора с системой позиционирования (рис. 12)

2 Постановка цели и задач

Цель работы – разработка и реализация архитектуры программного обеспечения для системы управления неразрушающим контролем, включающей модули позиционирования, безопасности и пульта управления.

Основные задачи:

1. Разработка архитектурных решений:
   - Системы позиционирования
   - Системы безопасности
   - Пульта управления
2. Выбор протоколов взаимодействия между компонентами системы (клиент-сервер-контроллер)
3. Подбор библиотек для разработки
4. Реализация архитектуры и бизнес-логики
5. Тестирование (ручное и автоматизированное с использованием Gitlab CI/CD)
6. Создание пакетов для развертывания

3 Функциональные требования

Система должна удовлетворять следующим функциональным требованиям:

- Автоматический запуск модуля при включении питания с индикацией готовности (LED)
- Прием, обработка и ответ на команды от клиентских приложений
- Унифицированный формат сообщений об ошибках
- Регулярная рассылка статуса системы подключенным клиентам, включая:
  - Состояние контроллеров
  - Текущие координаты
  - Статус выполнения команд

4 Нефункциональные требования

- **Отказоустойчивость**: Каждый модуль работает на отдельном устройстве Raspberry Pi [1]
- **Надежность**: Ведение детального журнала событий
- **Гибкость**: Настройка системы через конфигурационные файлы
- **Стандартизация**: Взаимодействие по TCP с использованием формата Gcode Marlin [2]
- **Очередность обработки**: Запросы от нескольких клиентов обрабатываются последовательно
- **Автозапуск**: Управление запуском через supervisor

5 Архитектура и разработка

5.1 Аппаратно-программная конфигурация

Серверная часть системы работает на платформе Raspberry Pi 4 со следующей конфигурацией:

- **Процессор**: 4-ядерный Cortex-A72 (ARM v8) с тактовой частотой 1.5 ГГц
- **Оперативная память**: 8 ГБ LPDDR4
- **Операционная система**: Ubuntu Server 22.04 LTS
- **Накопитель**: SSD 64 ГБ (SATA через USB 3.0)
- **Сетевые интерфейсы**: Gigabit Ethernet, Wi-Fi 5 (802.11ac)

Данная конфигурация обеспечивает:

- Высокую производительность для обработки команд в реальном времени
- Стабильную работу сетевых компонентов системы
- Достаточный объем памяти для работы всех модулей
- Надежное хранение конфигурационных данных и логов

5.2 Выбор библиотек и инструментов

Для реализации системы на языке C++ был выбран следующий набор библиотек:

Таблица 1: Используемые библиотеки и их назначение

Библиотека	**Назначение**	**Ключевые особенности**
JSON [3]	Работа с конфигурацией	Поддержка JSON-файлов, простота использования
GPR [4]	Обработка G-кода	Форк с доработками, интеграция с тестами
clsocket [5]	Сетевое взаимодействие	Кроссплатформенность, поддержка TCP
QT Logger [6]	Логирование	Интеграция с QT
GTest/GMock [7]	Модульное тестирование	Полноценный фреймворк для unit-тестов

5.3 Протоколы взаимодействия

Система использует два основных протокола:

1. **Протокол обмена данными с контроллерами**:

- Бинарный формат сообщений
- Поддержка Modbus RTU
- Контроль целостности данных (CRC32)

2. **G-code протокол**:

- Текстовый формат команд
- Поддержка стандарта Marlin [2]
- Расширение для специфичных команд системы

Команда	Данные master	Адрес, байт	Код функции	Данные, байт	Значения данных	CRC, байт	Данные slave	Адрес, байт	Код функции	Данные, байт	Значения данных	CRC, байт
Задать адрес устройства	Адрес	1	10 (0x0A)	3	1 байт: 0x00; 2 байт: 0x01; 3 байт: адрес от 0x01 до 0xF7	2	Установленный адрес	1	10 (0x0A)	3	1 байт: 0x00; 2 байт: 0x01; 3 байт: адрес от 0x01 до 0xF7	2

Рис. 1: Структура бинарной команды установки адреса контроллера

Старт вращения со скоростью	**G0 F[Value] [Ось] [Value]** **F[Value]** – скорость перемещение mm/s. Устанавливается как номинальная скорость для всех осей, перечисленных в строке или до следующей F в этой строке. **[Ось]:** A, B, C, X, Y, Z Value: mm ***Usage:*** ***Command:*** *G0 F100 X20.1 – для оси X* ***Answer:*** *G0 F100 X20.1 – Подтверждение выполнения команды* **F[Value]** – скорость вращения град/с. Устанавливается как номинальная скорость для всех осей, перечисленных в строке или до следующей F в этой строке. Знак для оси определяет направление движения **[Ось]:** A, B, C, X, Y, Z Value: degree ***Usage:*** ***Command:*** *G0 F10 X20.1 – вокруг оси X* ***Answer:*** *G0 F10 X20.1 – Подтверждение выполнения команды*

Рис. 2: Пример G-code команды линейного перемещения (G1)

5.4 Архитектура системы

Архитектура серверной части построена по модульному принципу, ниже представлены графики выполнеия команды и получения статуса (рис. 3, 4):

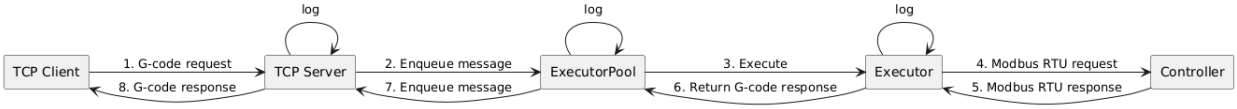


Рис. 3: Диаграмма выполнения команды

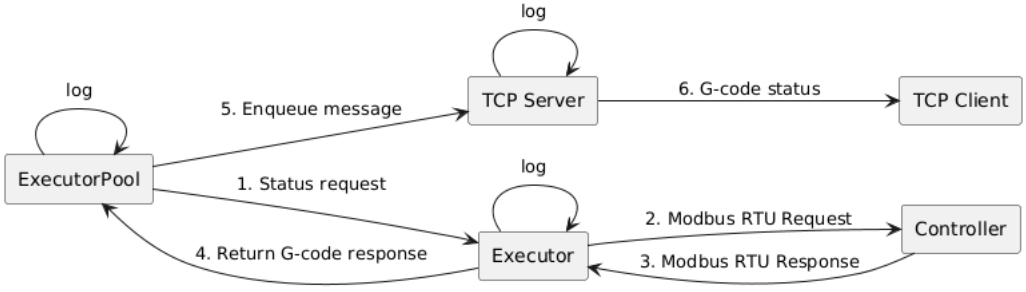


Рис. 4: Диаграмма получения статуса

5.4.1 Основные модули

1. **Модуль Server**:

- Обработка TCP-подключений клиентов
- Маршрутизация сообщений
- Рассылка статуса системы
- Поддержка множества одновременных подключений

2. **Модуль ExecutorPoll**:

- Валидация входящих сообщений (регулярные выражения)
- Трансляция G-code в бинарные пакеты
- Периодический опрос статуса контроллеров
- Реализация бизнес-логики (удержание точки интереса)

3. **Модуль Executor**:

- Преобразование команд между форматами
- Работа с интерфейсами (RS485, Ethernet)
- Проверка контрольных сумм (CRC32)
- Обработка ошибок Modbus RTU

5.4.2 Жизненный цикл обработки команды

1. Клиент отправляет G-code команду через TCP
2. Сервер принимает и передает команду в ExecutorPoll
3. ExecutorPoll валидирует команду и преобразует в бинарный формат
4. Executor отправляет пакет контроллеру
5. Контроллер выполняет команду и возвращает ответ
6. Ответ преобразуется обратно в G-code формат
7. Клиент получает результат выполнения

5.5 Пример выполнения команды

Рассмотрим выполнение команды линейного перемещения:

```
G1 F100 X50.5 Y-25.3 Z22.4
```

Этапы выполнения:

1. Парсинг команды G1 (линейное перемещение)
2. Установка скорости 100 мм/мин для всех осей
3. Отправка команд относительного перемещения:
   - Ось X: +50.5 мм
   - Ось Y: -25.3 мм
   - Ось Z: +22.4 мм
4. Контроль выполнения каждой операции
5. Возврат результата клиенту

При возникновении ошибки выполняется:

- Прерывание выполнения команды
- Отправка команды аварийной остановки (M112)
- Формирование сообщения об ошибке

6 Формат ошибок

6.1 Структура сообщения об ошибке

Система использует унифицированный формат сообщений об ошибках следующей структуры:

$$Error : [ErrorCode]"Description"$$

где:

- `ErrorCode` — 32-битное целое число без знака (uint32_t) в десятичном формате
- `Description` — текстовое описание ошибки на английском языке

6.2 Семантика кода ошибки

Код ошибки имеет следующую структуру:

Таблица 2: Структура кода ошибки

Байт 3	Байт 2	Байт 1	Байт 0
Уникальный код	Уникальный код	Приоритет	Локализация

6.2.1 Локализация ошибки (байт 0)

Определяет компонент системы, в котором возникла ошибка:

Таблица 3: Коды локализации ошибок

Код	Описание
1	Ошибка сервера Примеры: потеря соединения, ошибка CRC, несоответствие размера пакета
2	Ошибка оконечного устройства (ОУ) Примеры: сбой датчика, отсутствие калибровки, аварийное состояние
3	Некорректные параметры Примеры: недопустимые координаты, превышение скорости
4	Ошибка формата G-code Примеры: неправильный синтаксис, несколько команд в одном сообщении

6.2.2 Приоритет ошибки (байт 1)

Определяет критичность ошибки:

Таблица 4: Уровни приоритета ошибок

Код	Тип ошибки
0	Информационное сообщение
1	Устранимая ошибка Возможность автоматического или ручного восстановления
2	Предупреждение Команда не выполнена, но система работоспособна
3	Критическая ошибка Требуется вмешательство оператора, работа невозможна

6.2.3 Уникальный код ошибки (байты 2-3)

16-битный идентификатор конкретной ошибки, определяемый разработчиком компонента.

6.3 Пример обработки ошибки

```
Входная команда:
G0 F10 X100

Ответ системы:
G0 F10 X100 Error: 66306 "Permissible excess current value"
```

Разбор кода ошибки 66306 (0x00010302 в hex):

- **Байт 0**: 0x02 → Ошибка оконечного устройства (ОУ)
- **Байт 1**: 0x03 → Критическая ошибка (уровень 3)
- **Байты 2-3**: 0x0001 → Уникальный код ошибки 1 (превышение тока)

6.4 Рекомендации по обработке ошибок

- Для ошибок уровня 3 (критических) необходимо:
  - Прекратить выполнение текущих команд
  - Перевести систему в безопасное состояние
  - Уведомить оператора
- Ошибки уровня 2 должны записываться в лог с пометкой WARNING
- Ошибки формата (код 4) должны сопровождаться примером корректного синтаксиса
- Все ошибки должны сохраняться в системном журнале

7 Алгоритм удержания точки интереса для системы позиционирования

7.1 Определение и основные понятия

Точка интереса — это фиксированная точка на столе позиционирования, которая должна оставаться в центре области видимости детектора при любых перемещениях системы.

Система использует следующие команды управления:

- `G0` — линейное перемещение в заданную точку
- `G6` — вращение с заданной скоростью (для осей X,Y,Z — стол; A — детектор; C — дуга)

7.2 Системы координат

В системе определены три ключевые точки:

- Wx_0, Wy_0 — центр мировой системы координат (точка вращения дуги и детектора)
- Tx_0, Ty_0 — центр системы координат стола (0,0 в координатах позиционирования)
- PS_x, PS_y — текущее положение центра стола

7.3 Функциональные требования

7.3.1 Общие требования

1. Функциональность должна быть конфигурируемой через параметр в файле конфигурации сервера
2. При активации функции система должна автоматически корректировать положение стола при выполнении команд G0/G6

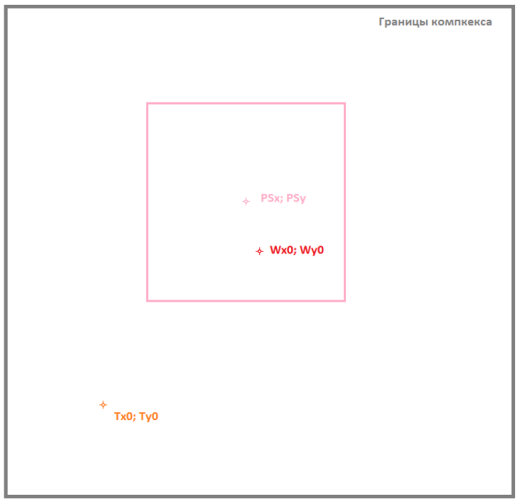


Рис. 5: **Системы координат комплекса AXI.** Ось Z для всех систем координат совпадает. И направлена снизу вверх.

7.3.2 Обработка команды G0

- Если команда не затрагивает оси X/Y, но изменяет Z/A/C:
  1. Рассчитать необходимое смещение стола по осям X/Y
  2. Выполнить коррекцию одновременно с основным перемещением
- При наличии изменений по X/Y в команде — коррекция не выполняется

7.3.3 Обработка команды G6

- При вращении по Z/A/C:
  1. Выполнять периодическую коррекцию X/Y (с настраиваемым интервалом)
  2. После остановки — финальная точная коррекция положения
- Ответ отправляется только после завершения всей последовательности

7.3.4 Особенности реализации

- Ответ содержит только явно указанные в команде оси
- Допускается реализация в виде отдельного микросервиса
- Обязательно наличие комплексных тестов:
  - Проверка отдельных осей
  - Комбинации осей
  - Точность финального позиционирования

7.4 Требования к тестированию

- Проверка граничных условий
- Тестирование комбинаций осей
- Проверка точности позиционирования
- Тестирование с разными скоростями перемещения
- Проверка реакции на ошибки позиционирования

8 Алгоритм контроля положения системы позиционирования в границах цифровых пределов

8.1 Общее описание

Алгоритм обеспечивает контроль допустимого рабочего пространства системы позиционирования, предотвращая выход за установленные границы. Функциональность реализована как дополнительный защитный механизм, работающий параллельно с аппаратными концевыми выключателями.

8.2 Функциональные требования

8.2.1 Активация функциональности

- Управление функцией осуществляется через параметр конфигурации:

```
{
  "enable_ccdc": true|false
}
```

- Возможные варианты значений:
  - `true`/`1` — функция активирована
  - `false`/`0` — функция деактивирована

Если функциональность контроля положения Системы позиционирования в границах цифровых пределов – не включена, то при работе сервера никаких дополнительных проверок выполняться не должно.

8.2.2 Условия срабатывания

Контроль пределов выполняется при:

1. Получении команды стоп вращение `M76`
2. Завершении выполнения команд:
   - `G0` (линейное перемещение)
   - `G62` (специальное позиционирование)
3. Старте системы (инициализационная проверка)

8.2.3 Процедура проверки

1. Запрос текущих значений цифровых пределов для всех осей
2. Верификация полученных значений:
   - Проверка на наличие значений
   - Валидация (max $>$ min)
3. Сравнение текущего положения с допустимыми пределами
4. Действия при выходе за границы:
   - Запись в системный журнал
   - Перевод системы в безопасный режим

9 Алгоритм повышения точности позиционирования

9.1 Общее описание

Алгоритм обеспечивает повышенную точность позиционирования за счет двухэтапного перемещения:

1. Быстрое перемещение в целевую область
2. Точная коррекция положения на сниженной скорости

9.2 Функциональные требования

9.2.1 Активация функциональности

- Управление через параметр конфигурации:

```
{
  "enable_increased_positioning_accuracy": true,
  "axis_settings": {
    "X": {
      "positioning_accuracy": 0.005,
      "correction_speed": 1.5
    }
  }
}
```

- Значения по умолчанию:
  - Точность: 0.01 мм/град
  - Скорость коррекции:
    - 2 мм/с - линейные оси (X,Y,Z)
    - 1 град/с - вращательные оси (A,B,C)

9.2.2 Логика работы

1. После выполнения команды G0:
   - Сравнение текущего положения с целевым
   - Расчет отклонения $\Delta = |P_{current} - P_{target}|$
2. Условия коррекции:
$$\Delta > \epsilon_{axis} \tag{1}$$
   где ϵ_{axis} - заданная точность для оси
3. Процедура коррекции:
   - Установка пониженной скорости
   - Выполнение перемещения $P_{current} \rightarrow P_{target}$
   - Восстановление рабочей скорости

9.2.3 Ключевые особенности

- Однократное выполнение коррекции после основной команды
- Независимые параметры для каждой оси
- Сохранение состояния скоростей
- Журналирование процесса коррекции

9.3 Пример работы

Для оси X с параметрами:

- Требуемая позиция: 100.00 мм
- Достигнутая позиция: 99.98 мм
- Точность: 0.01 мм
- Скорость коррекции: 1 мм/с

Система выполнит:

1. Обнаружит отклонение 0.02 мм $>$ 0.01 мм
2. Установит скорость 1 мм/с
3. Выполнит перемещение на +0.02 мм
4. Вернет рабочую скорость

10 Файлы конфигурации

Система использует два основных типа конфигурационных файлов:

- `master_config.json` - главный файл настроек сервера
- `T.json` - файлы конфигурации контроллеров

10.1 Главный конфигурационный файл (master_config.json)

10.1.1 Пример полной конфигурации

Ниже приведен пример конфигурации для системы позиционирования master_config.json


```
{
  "master": {
    "control_console": {
      "host": "",
      "port": 4002,
      "usage": "on"
    },
    "server": {
      "log_path": "/var/log/server",
      "host": "",
      "port": 4000,
      "status_port": 4001,
      "status_pooling_ms": 1000
    },
    "executor": {
      "log_path": "/var/log/server",
      "modbus-interface": {
        "port_devices": "/dev/ttyS0",
        "baud_rate": 9600,
        "parity": "N",
        "usage": "on"
      }
    },
    "executor_pool": {
      "log_path": "/var/log/server",
      "status_pooling_ms": 100
    },
    "enable_ccdc": "off",
    "enable_increased_positioning_accuracy": "off",
    "model_math_usage": "off",
    "model_math_tuning_ms": 100,
    "model_params": {
      "m_tableCenter": {
        "x": 254.688,
        "y": 271.377,
        "z": 10.0
      },
      "m_tableSize": {
        "w": 450.0,
        "h": 450.0
      },
      "m_arcRadius": 606,
      "m_fdd": 600.0,
      "m_detectorSize": {
        "w": 140.0,
        "h": 116.5
      },
      "m_height_above_table_mm": 8
    },
```

```
    "server_version": "v0.0.8",
    "modbus_protocol": "v18",
    "gcode_protocol": "v1.11"
  }
}
```

10.2 Конфигурация контроллеров (T.json)

Ниже приведен фрагмент для конфигурации работы с контроллерами. В системе Gcode контроллер обозначается при помощи тега (tag) и имени оси (X, Y, Z, A, B, C) T0.json:

```
{
    "slaves": {
        "A": {
            "brake_condition": "off",
            "conf_el": {
                "cyc_rev_rel_enc": 4096,
                "freq_poll_ms": 1,
                "inversion_absolute_enc": "yes",
                "inversion_increment_enc": "no",
                "inversion_pot": "no",
                "presence_absolute_enc": "no",
                "presence_brake": "no",
                "presence_end_sensors": "no",
                "presence_increment_enc": "no",
                "presence_pot": "yes"
            },
            "conf_motor": {
                "engine_inversion": "yes",
                "microstep_mode": "off",
                "mm_rev_frac_part": 1,
                "mm_rev_whole_part": 0,
                "mode_work_stepper": "micro_step",
                "pairs_bldc": 1,
                "type_motor": "bldc"
            },
            "correction_speed": 1,
            "displacement": "absolute",
            "home_position": 0.0,
            "home_position_status": 0,
            "limit": {
                "boost_speed_bldc": 0,
                "enable_position_limit": "yes",
                "enable_safe_zone": "no",
                "max_current": 0,
                "max_extremal_boost": 6000,
                "max_pos": 57,
                "max_speed_bldc": 3000,
                "min_current": 1,
                "min_pos": 1,
                "minimum_motor_shaft_frequency": 0,
                "speed_reduction_mm": 0,
                "speed_reduction_pers_max": 0
            },
            "main_init": {
                "baud_rate": 9600,
                "firmware_version": "none",
                "parity": "none",
                "slave_address": 10,
                "usage": "yes"
            },
```

```
            "position_accuracy": null,
            "speed_linear_default": 10,
            "speed_linear_move": 0.0
        }
    },
    "tag": "T0"
}
```

11 Состав пакета и процесс установки

11.1 Формат пакета

Система распространяется в виде пакета `server*.tar.gz`, содержащего:

Таблица 5: Структура пакета сервера

Путь	**Назначение**
bin/server	Исполняемый файл сервера
files/AXI(NDT)/	Конфигурации для различных стендов
files/master_config.json	Основной конфигурационный файл
lib/	Статические библиотеки (*.a)
scripts/run.sh	Основной скрипт запуска
configs/server.conf	Конфиг для supervisor
configs/gpio.conf	Конфигурация GPIO
scripts/requirements.sh	Установка зависимостей

Пример установочного скрипта запуска сервера с помощью supervisor install_run.sh, принимает путь к server*.tar.gz:

```
#!/bin/bash

# install & run
# ---------
# ARGUMENT $1 = prefix --path to server*.tar.gz
# run server with supervisorctl

set -ex

if [ -z $1 ] ; then
  echo "First parameter needed!" && exit 1;
fi

prefix=$1
echo $prefix
cd $prefix

# server stage:unpack
echo "server stage:unpack"
if [ -d "/builds/server" ]; then
  OLD_DIR_TMP=$(mktemp -d "${TMPDIR:-/builds/}server.XXXXXXXXXXX")
  echo "copying old server to ${OLD_DIR_TMP}"
  cp -r /builds/server/* ${OLD_DIR_TMP}
  rm -rf /builds/server/
fi
mkdir -p /builds/server/
tar -xzvf server*tar.gz -C /builds/server/ --strip-components=1
cd /builds/server/
# shellcheck disable=SC2010
cd "`ls | grep server*tar.gz | head -n1`"

echo `pwd`/lib/* >> /etc/ld.so.conf.d/libc.conf
```

```
cp `pwd`/lib/* /usr/local/lib
ldconfig

service supervisor stop
service supervisor start

# allow ports
echo "allow ports"
update-alternatives --list iptables
update-alternatives --set iptables /usr/sbin/iptables-legacy
ufw allow 4000
ufw allow 4001
ufw allow 4002

# mkdir log folder
mkdir -p /var/log/server
mkdir -p /var/log/gpio
touch /var/log/server/out.log /var/log/server/err.log /var/log/gpio/out.log /var/log/gpio/err.log

# copy supervisor server config and reread/update service
echo "copy supervisor server config and reread/update service"
cp server.conf /etc/supervisor/conf.d
cp gpio.conf /etc/supervisor/conf.d
supervisorctl reread
supervisorctl update

# server access rights
echo "server access rights"
chmod +x main_run.sh files/*/run.sh

# server stage:run
# echo "server stage:run"
supervisorctl restart server || true
```

11.2 Конфигурация сервисов

11.2.1 Файл server.conf для supervisor

```
[program:server]
# path to server directory
directory=/builds/server
command=/bin/bash main_run.sh bin/app files/AXI
user=root
autostart=true
autorestart=true
stdout_logfile=/var/log/server/out.log
stderr_logfile=/var/log/server/err.log
```

11.3 Процесс обновления

1. Остановка работающего сервера
2. Проверка целостности пакета
3. Создание резервной копии
4. Распаковка новой версии
5. Валидация конфигураций
6. Запуск обновлённой версии

12 Автоматизация

Система использует GitLab CI/CD для организации end-to-end процесса разработки. Основные компоненты:

- **Главный пайплайн** - координирует выполнение дочерних пайплайнов
- **Дочерние пайплайны** - выполняются на специфичных раннерах (x86_64, arm64)
- **Артефактные хранилища** - пакеты, логи тестов, метафайлы

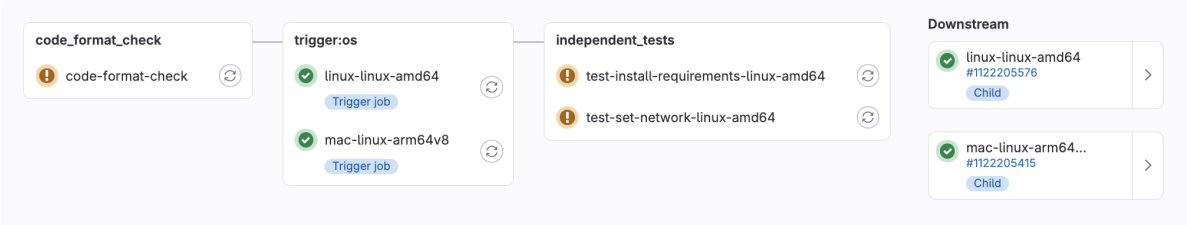


Рис. 6: Архитектура системы CI/CD

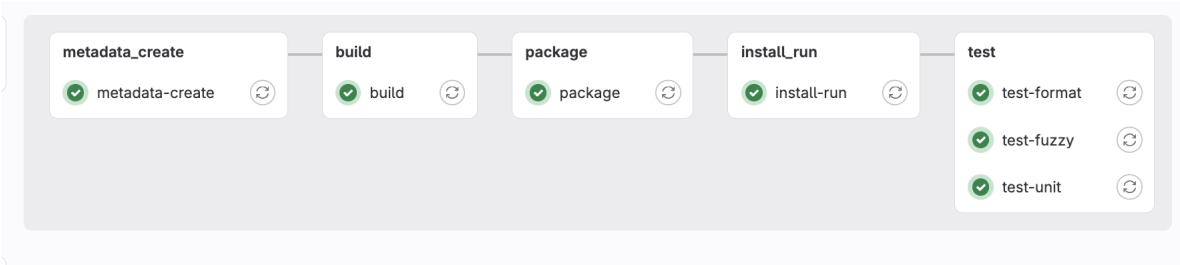


Рис. 7: Пример дочернего пайплайна

12.1 Этапы обработки

Таблица 6: Этапы CI/CD пайплайна

Этап	Задачи	Артефакты
Генерация метаинформации	Создание версии пакета	Файл версии сервера
Сборка	Компиляция, линковка	Библиотеки, бинарники
Тестирование	Unit, интеграционные тесты	Отчеты
Пакетирование	Создание tar.gz пакетов	Установочные пакеты

12.2 Конфигурация пайплайна

Пример `.gitlab-ci.yml`:

```
include:
  - local: "ci/stages.yml"
  - local: "ci/variables.yml"

code-format-check:
  tags:
    - shared
    - amd64
  stage: code_format_check
  image: $SERVER_IMAGE
  allow_failure: true
  script:
    - export TESTS_DIR=`pwd`/tests/code_format_check
```

```
    - chmod +x ${TESTS_DIR}/run_format_check.sh
    - /bin/bash ${TESTS_DIR}/run_format_check.sh
  artifacts:
    when: on_failure
    paths:
      - format_check_error_list.txt
      - format_check_patch.diff

mac-linux-arm64v8:
  stage: trigger:os
  rules:
    - if: $ELECTRON_LOCAL == "false"
      when: always
  trigger:
    include: "ci/pipelines/mac-linux-arm64v8/common.yml"
    strategy: depend
  variables:
    PARENT_PIPELINE_ID: $CI_PIPELINE_ID

linux-linux-amd64:
  stage: trigger:os
  trigger:
    include: "ci/pipelines/linux-linux-amd64/common.yml"
    strategy: depend
  variables:
    PARENT_PIPELINE_ID: $CI_PIPELINE_ID

# --- independent_tests ---
test-install-requirements-linux-amd64:
  tags:
    - shared
    - amd64
  needs: [ ]
  stage: independent_tests
  image: $SERVER_IMAGE
  script:
    - chmod +x scripts/requirements.sh
    - ./scripts/requirements.sh
  allow_failure: true

test-set-network-linux-amd64:
  tags:
    - shared
    - amd64
  needs: [ ]
  stage: independent_tests
  image: $SERVER_IMAGE
  script:
    - chmod +x scripts/set-network.sh
    - ./scripts/set-network.sh scripts
  allow_failure: true
# --- independent_tests ---
```

13 Тестирование

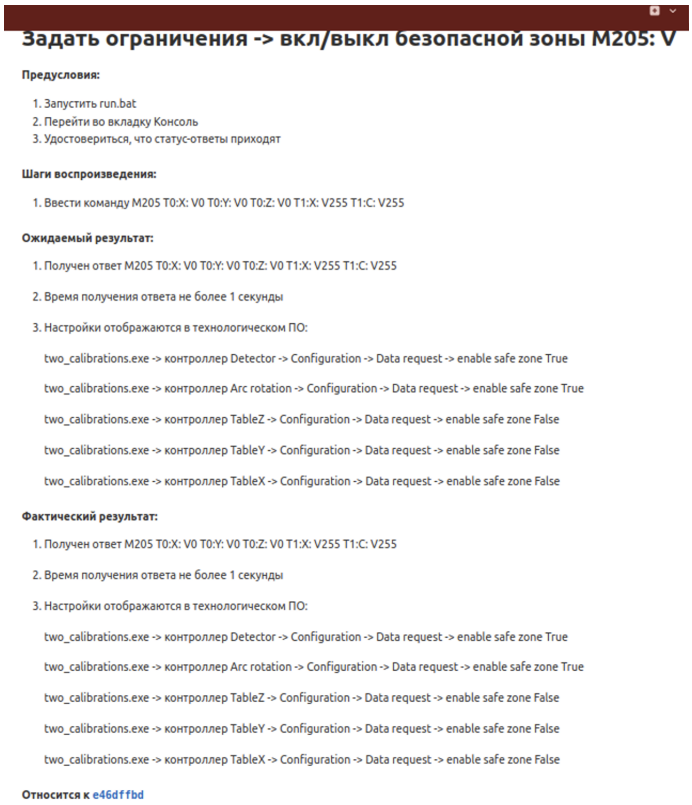
Система использует комплексный подход к тестированию, включающий несколько взаимодополняющих методов: ручное тестирование, unit-тестирование, format и fuzzy тестирование

13.1 Ручное тестирование

Ручное тестирование – это процесс поиска "багов"при помощи сил человека. Тестировщик имитирует реальные действия пользователя и старается охватить максимум функций системы и найти ошибки.

13.1.1 Процесс и инструменты

- Реализовано через систему тест-кейсов в GitLab
- Используется чек-лист из 50+ сценариев
- Включает проверку:
  - Основных функций позиционирования
  - Граничных условий
  - Обработки ошибок



Задать ограничения -> вкл/выкл безопасной зоны M205: V

Предусловия:

1. Запустить run.bat
2. Перейти во вкладку Консоль
3. Удостовериться, что статус-ответы приходят

Шаги воспроизведения:

1. Ввести команду M205 T0:X: V0 T0:Y: V0 T0:Z: V0 T1:X: V255 T1:C: V255

Ожидаемый результат:

1. Получен ответ M205 T0:X: V0 T0:Y: V0 T0:Z: V0 T1:X: V255 T1:C: V255
2. Время получения ответа не более 1 секунды
3. Настройки отображаются в технологическом ПО:

   two_calibrations.exe -> контроллер Detector -> Configuration -> Data request -> enable safe zone True

   two_calibrations.exe -> контроллер Arc rotation -> Configuration -> Data request -> enable safe zone True

   two_calibrations.exe -> контроллер TableZ -> Configuration -> Data request -> enable safe zone False

   two_calibrations.exe -> контроллер TableY -> Configuration -> Data request -> enable safe zone False

   two_calibrations.exe -> контроллер TableX -> Configuration -> Data request -> enable safe zone False

Фактический результат:

1. Получен ответ M205 T0:X: V0 T0:Y: V0 T0:Z: V0 T1:X: V255 T1:C: V255
2. Время получения ответа не более 1 секунды
3. Настройки отображаются в технологическом ПО:

   two_calibrations.exe -> контроллер Detector -> Configuration -> Data request -> enable safe zone True

   two_calibrations.exe -> контроллер Arc rotation -> Configuration -> Data request -> enable safe zone True

   two_calibrations.exe -> контроллер TableZ -> Configuration -> Data request -> enable safe zone False

   two_calibrations.exe -> контроллер TableY -> Configuration -> Data request -> enable safe zone False

   two_calibrations.exe -> контроллер TableX -> Configuration -> Data request -> enable safe zone False

Относится к e46dffbd

Рис. 8: **Пример тест-кейса**

13.2 Unit-тестирование

Unit-тестирование –это разновидность тестирования в программной разработке, которое заключается в проверке работоспособности отдельных функциональных модулей, процессов или частей кода приложения.

Unit-тестирование позволяет избежать ошибок или быстро исправить их при обновлении или дополнении ПО новыми компонентами, не тратя время на проверку программного обеспечения целиком.

Во всех проектах применяется фреймворк GoogleTest [7]. Можно сказать, что он является стандартом индустрии и имеет хорошо развитое сообщество.

В проекте присутствуют сдедующие тесты:

- test-algorithms – тесты самописных алгоритмов
- test-config-update – тесты на обновление конфигурации
- test-errros – тесты на формат ошибок
- test-executor-pool – тесты на класс ExecutorPool
- test-gcode-conv – тесты на создание байт-пакетов из Gcode
- test-gpr – тесты на парсинг Gcode
- test-regex – тесты на сопоставление Gcode в regex. Для каждого ПО regex свои, согласно протоколу общения в Gcode формате
- test-serial – тесты сериал порта, вычисление контрольной суммы CRC32

Тесты постоянно разрабатываются. Сейчас покрытие тестами оценивается в 42%.

13.3 Format-тестирование

В проекте написан скрипт для тестирование входящих сообщений от клиента. Сервер сопоставляет входящие сообщения с набором допустимых regex и присылает ответ.

Скрипт сравнивает ответ сервера с эталонными значениями.

Например, при сообщении с неправильным форматом ответ от сервера будет содержать ошибку

13.4 Fuzzy-тестирование

Fuzzy-тестирование –техника тестирования программного обеспечения, заключающаяся в передаче приложению на вход неправильных, неожиданных или случайных данных.

В проекте реализован скрипт для такого тестирования. На осовании regex для проверки формата сообщения, скрипт генерирует входные данные и пересылает их серверу. Ожидается, что сервер обработает входные данные с ошибкой на формат сообщения. Соединение при этом не рвется.

14 Использование брокера сообщений

Брокер сообщений — это промежуточное программное обеспечение, которое обеспечивает обмен сообщениями между компонентами системы, соблюдая принципы асинхронности, декомпозиции и отказоустойчивости.

14.1 Архитектурные преимущества

Внедрение брокера сообщений обеспечивает следующие ключевые преимущества для системы:

- **Асинхронная коммуникация**:
  - Отправители и получатели работают независимо
  - Возможность буферизации сообщений
- **Масштабируемость**:
  - Горизонтальное масштабирование компонентов
  - Балансировка нагрузки между потребителями
- **Отказоустойчивость**:
  - Сохранение сообщений при сбоях
  - Автоматическое восстановление соединений

14.2 Выбор технологий

Для интеграции рассматриваются следующие решения:

Таблица 7: Сравнение брокеров сообщений

Технология	**Протокол**	**Преимущества**	**Ограничения**
RabbitMQ	AMQP	Гибкая маршрутизация	Требует изучения
Apache Kafka	Собственный	Высокая пропускная способность	Сложность администрирования
Redis	Pub/Sub	Простота интеграции	Нет persistence

14.3 Ключевые компоненты системы:

- **Публикаторы**:
  - Система позиционирования (команды, статусы)
  - Пульт управления (команды)
  - Система безопасности (команды, статусы)
  - АРМ оператора (команды)
- **Подписчики**:
  - АРМ оператора (статусы)
  - Внешние интеграции

14.4 Ожидаемые результаты

- Увеличение отказоустойчивости системы
- Снижение задержек межмодульного взаимодействия
- Упрощение добавления новых компонентов

15 Результаты

В ходе разработки достигнуты следующие результаты:

- Полностью реализована система позиционирования, включающая:
  - Основные функции управления перемещением
  - Регулярное тестирование на рабочих стендах
  - Более 150 Unit тестов
- Разработан веб-пульт для тестирования с функциями:
  - Отправка команд управления
  - Получение и отображение статуса системы
  - Простое переключение между модулями по IP-адресу
- Реализованы и протестированы ключевые серверные компоненты:
  - Сервер пульта управления (интегрирован с системой позиционирования)
  - Сервер системы безопасности
- Проведена оптимизация архитектуры:
  - Выделен базовый класс для общей функциональности
  - Унифицирована работа с TCP-соединениями
  - Улучшена обработка сообщений
- Внедрены алгоритмы управления:

  - Удержание точки интереса
  - Контроль рабочих пределов
  - Повышение точности позиционирования
- Настроена CI/CD система:
  - Автоматическая сборка и тестирование
  - Генерация установочных пакетов
  - Поддержка разных архитектур (x86_64, arm64)
- Развернут Docker Registry для хранения образов

16 Планы развития

На ближайшую перспективу запланированы следующие работы:

- Модернизация архитектуры:
  - Внедрение брокера сообщений
  - Создание библиотеки общего кода
- Повышение качества:
  - Тестирование с санитайзерами
  - Увеличение покрытия unit-тестами
  - Профилирование производительности
- Инфраструктурные улучшения:
  - Автоматическое развертывание на стендах
  - Организация хранилища пакетов
  - Улучшение CI/CD пайплайнов

Рис. 9: Взаимодействие компонент комплекса **в старой архитектуре**

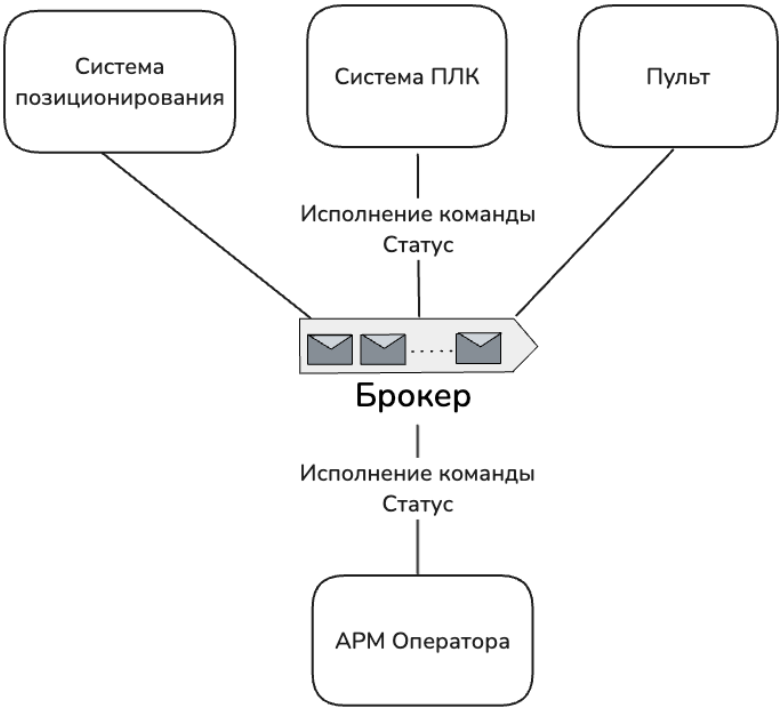


Рис. 10: Взаимодействие компонент комплекса **в новой архитектуре**

Список литературы

1. Raspberry Pi4. — Accessed: 2025-03-04. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.
2. Gcode Marlin Documentation. — Accessed: 2025-03-04. https://marlinfw.org/meta/gcode/.
3. Json library. — Accessed: 2025-03-04. https://github.com/nlohmann/json.
4. Gcode parsing library. — Accessed: 2025-03-04. https://github.com/childhoodisend/gpr.
5. Socket library. — Accessed: 2025-03-04. https://github.com/DFHack/clsocket.
6. Logger library. — Accessed: 2025-03-04. https://gitlab.com/childhoodisend/qt-logger.
7. GoogleTest Framework. — Accessed: 2025-03-04. https://github.com/google/googletest.

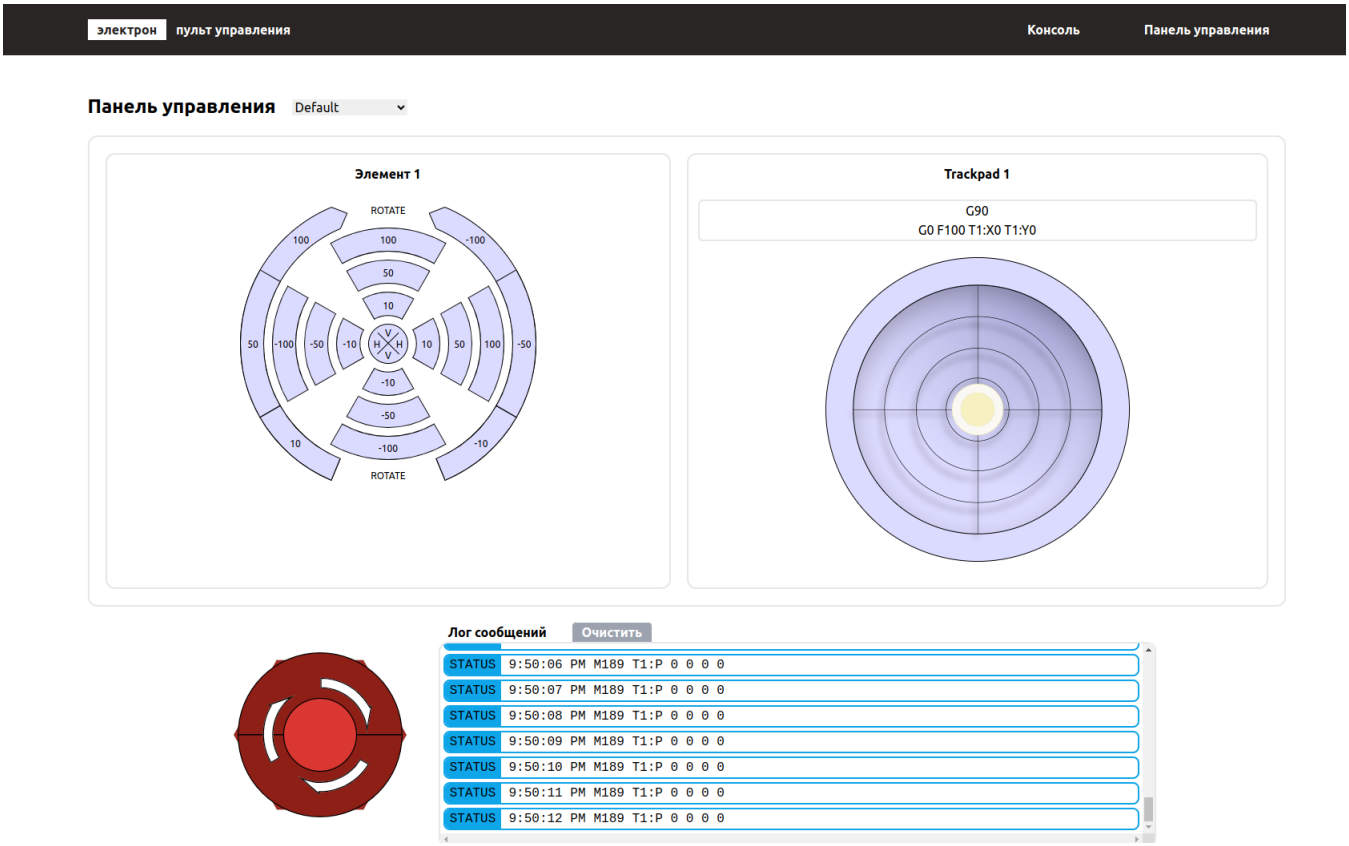


Рис. 11: **Стендовый пульт управления.** Эмулирует аппаратную часть. Способен слать команды и получать статус от системы позиционирования, безопасности, пульта управления.

Рис. 12: **Аппаратный пульт управления системой позиционирования.**