Санкт–Петербургский государственный университет

Отчет по учебной 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 1 курса, группы 23.М04-мм:
Сатановский А. Д.

Научный руководитель:
Луцив Д.В.

г. Санкт – Петербург
2024

Содержание

1 Введение

Неразрушающий контроль – контроль надёжности основных рабочих свойств и параметров объекта или отдельных его элементов/узлов, не требующий выведения объекта из работы либо его демонтажа.

Целью использования неразрушающего контроля в промышленности является надёжное выявление опасных дефектов изделий.

Область применения – промышленность.

Область неразрушающего контроля достаточно большая. Направление неразрушающего контроля при помощи радиографии развивается на базе НИПК Электрон.

В рамках работы планируется реализовать и протестировать:

- Систему позиционирования – ПО, обеспечивающее перемещение аппаратного комплекса
- Систему безопасности – ПО, обеспечивающее безопасность аппаратного комплекса
- Пульт управления системой позиционирования – ПО, обеспечивающее возможность взаимодействия аппаратного пульта управления с системой позиционирования 5

2 Постановка цели и задач

Целью работы является создание и реализация архитектуры ПО управления системой неразрушающего контроля. А именно, модулями системы позиционирования, системы безопасности, пульта управления системой позиционирования.

Задачи:

- Создание архитектуры системы позиционирования
- Создание архитектуры системы безопасности
- Создание архитектуры пульта управления системой позиционирования
- Выбор протоколов взаимодействия между клиентом - сервером - контроллером
- Выбор библиотек для разработки
- Реализация и тестирование

3 Функциональные требования

Функциональные требования, предъявляющиеся системе:

- При включении питания модуль должен автоматически запускаться
- При включении должна быть led индикация
- Модуль принимает команду от клиента, исполняет, формирует ответ и передает его клиенту
- Если возникает ошибка, клиент должен получить ошибку в ответ
- Формирует статус системы и передает подключенным клиентам

4 Нефункциональные требования

Данные нефункциональные требования должны быть выполнены при разработке:

- Отказоустойчивость. Каждый модуль располагается на Raspberry Pi [1]
- Надежность. Модуль должен вести журнал событий
- Гибкость. Система настраивается через конфигурационные файлы
- Соответствие стандартам предприятия. Взаимодействие между клиент - сервером должна происходить по TCP в формате Gcode Marlin [2]
- При подключении более одного клиента, каждый запрос должен обрабатываться в порядке очереди.

5 Архитектура и разработка

5.1 Аппаратно-программная конфигурация

Работа серверной части происходит на устройстве Raspberry Pi4, обладающем следующими характеристиками:

- Процессор: Cortex-A72 (ARM v8)
- ОЗУ: 8gb
- ОС: Ubuntu 22.04
- SSD: 64gb

5.2 Выбор библиотек

Разработка ведется на языке C++. Зафиксированы следующие библиотеки, удовлетворяющие требованиям:

- Для возможности конфигурации используется классическая библиотека **json**, позволяющая настраивать систему через файлы, что позволяет не компилировать программу повторно [3]
- Для работы с форматом **Gcode** используется форк **gpr** [4]
- Работу с сетью обеспечивает библиотека **clsocket** [5]
- Модуль логгирования – библиотека, основанная на фреймворке **QT** [6; 7]
- Фреймворк Unit-тестирования **googltest** [8]
- Переиспользуемые локальные библиотеки

5.3 Архитектура взаимодействия

Структурно архитектуру серверной части можно представить следующим образом:

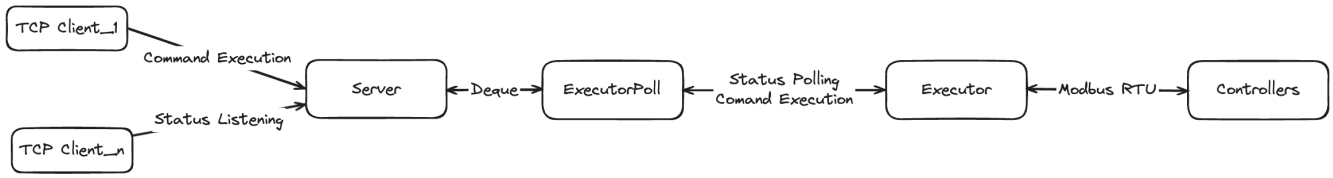


Рис. 1: **Архитектура взаимодействия серверной части**

Программно реализуются следующие модули:

- Модуль Server. Отвечает за обработку подключений клиентов по сети. Получает сообщения и передает их в модуль ExecutorPoll. Передает статус системы.
- Модуль ExecutorPoll. Валидирует сообщения (в соответствии с заданными regex), переводит сообщение из формата Gcode в байт-код. Передает байт-код классу Executor. Выполняет Status Polling.
- Модуль Executor. Реализует интерфейс взаимодействия с контроллерами - это может быть Serial Port RS485, Ethernet и другие. Пересылает байт-пакет контроллерам и получает ответ (если необходимо). Проверяет наличие ошибки в соответствии с протоколом Modbus RTU.

В общем и целом, логика работы такой архитектуры сводится к следующим стадиям:

- Передать статус подключенному клиенту
- Получить Gcode команды от клиента, провалидировать
- Распарсить команду, перевести в байт-код
- Передать байт-код контроллеру, получить ответ, обработать ответ. Если есть ошибка - обработать ошибку.

- Сформировать Gcode из полученного байт-кода в нужном формате
- Передать сформированный Gcode обратно клиенту

При этом в системе позиционирования, безопасности и в пульте управления стадия валидации, парсинга, перевода в байт-код и формирования ответного Gcode может быть своя и зависит от набора gcode команд в протоколе для конкретной системы.

Status polling также может быть свой для вышеперечисленных систем. Таким образом, изменению подвержен только класс ExecutorPoll, где реализуется основная логика системы с специфичными протоколами.

Интерфейс подключения к контроллеру может быть произвольным. Для этого достаточно добавить поддержку нужного интерфейса в класс $interface\_t$.

5.4 Пример выполнения Gcode команды для системы позиционирования

Пусть необходимо выполнить относительное перемещение системы позиционирования по оси X на 50.5 мм, Y на 25.3 мм, Z на 22.4 со скоростью 100 мм/мин. На уровне TCP команда в формате Gcode выглядит как G1 F100 X50.5 Y-25.3 Z22.4, где $G1$ - команда линейного перемещения.

На уровне Modbus RTU команда транслируется в следующие байт пакеты:

- установить скорость 100 мм/мин для оси X
- установить скорость 100 мм/мин для оси Y
- установить скорость 100 мм/мин для оси Z
- относительное перемещение оси X на 50.5 мм
- относительное перемещение оси Y на -25.3 мм
- относительное перемещение оси Z на 22.4 мм

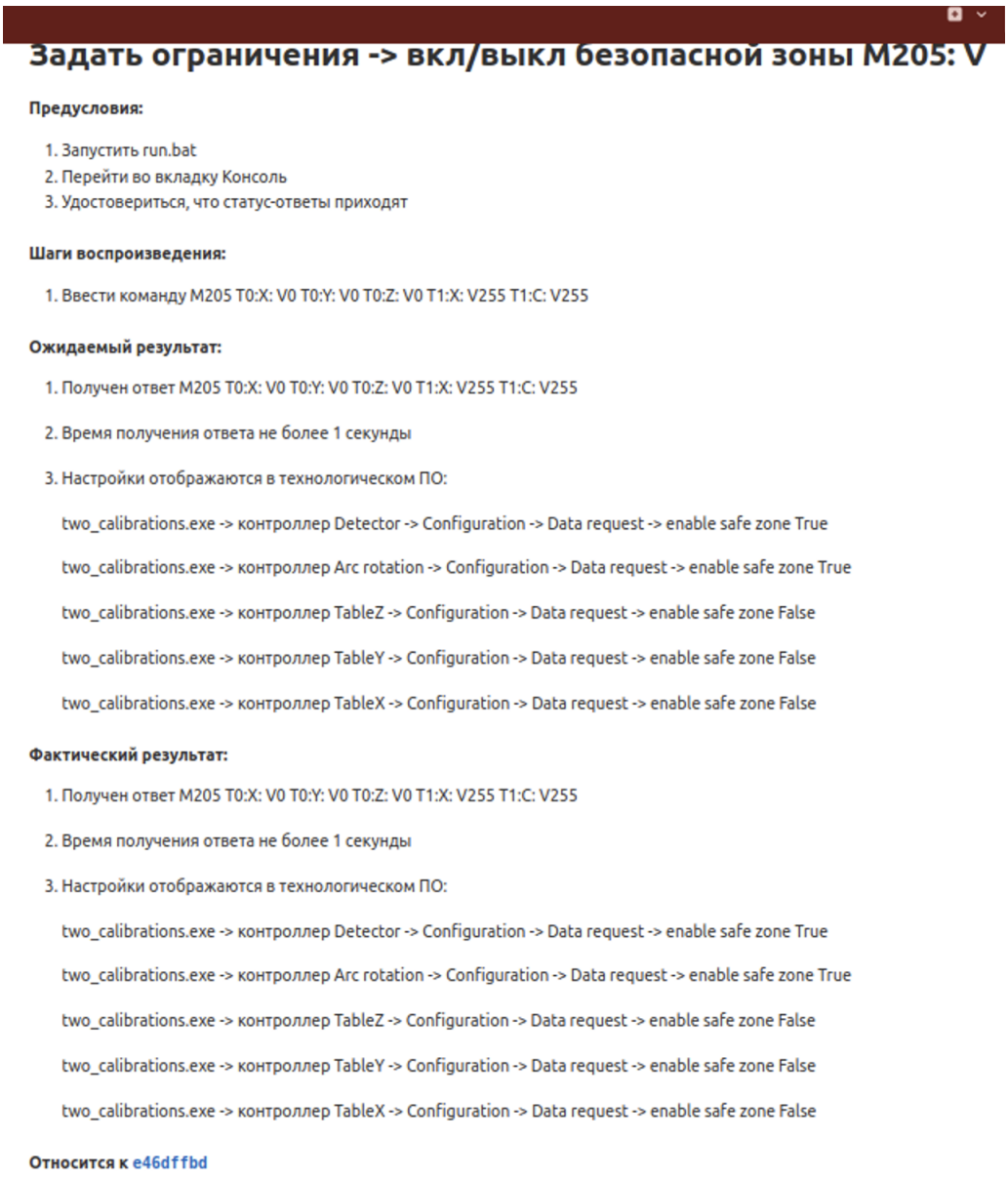
При возникновении ошибок на уровне Modbus RTU посылается команда на экстренную остановку. Экстренную остановку также можно произвести программно gcode командой $M112$.

6 Тестирование

6.1 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.

Для тестирование созданы тест-кейсы в локальном GitLab.



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. 2: **Пример тест-кейса для ручного тестирования.**

6.2 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 [8]. 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-gcode-conv - тесты на создание байт-пакетов из Gcode
- test-gpr - тесты на парсинг Gcode
- test-regex - тесты на сопоставление Gcode в regex
- test-serial - тесты сериал порта, контрольную сумму CRC32

Тесты постоянно разрабатываются. Сейчас покрытие тестами оценивается в 40

6.3 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 **ERROR: BAD MESSAGE FORMAT**

6.4 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 **ERROR: BAD MESSAGE FORMAT**. Соединение при этом не рвется.

7 Алгоритм удержания точки интереса для системы позиционирования

Точка интереса – это точка стола, которая находится в центре области видимости детектора на момент начала перемещения.

Для системы позиционирования комплекса AXI определены такие команды как G0 - линейное перемещение в точку, G6 старт вращения со скоростью для осей X, Y, Z, (стол) A(детектор), C(дуга).

Wx_0 и Wy_0 – это точка центра системы координат относительно которой вращается дуга и детектор. Данная точка считается центром мировых координат;

Tx_0 и Ty_0 – это точка центра системы координат стола. Для осей X и Y в системе позиционирования это точка с координатами 0, 0; Левее и ниже данной точки стол перемещаться не может;

PS_x и PS_y – это координаты центра стола системы позиционирования. Данные координаты соответствуют координатам текущего положения X и Y в системе позиционирования.

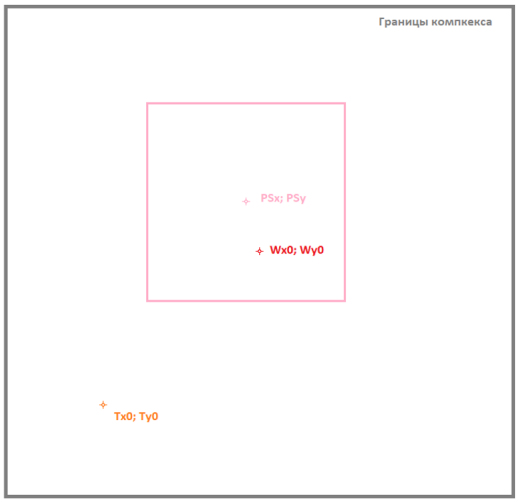


Рис. 3: **Системы координат комплекса AXI.**Ось Z для всех систем координат совпадает. И направлена снизу вверх.

7.1 Функциональные требования

Планируется разработка следующего алгоритма для системы позиционирования с следующими функциональными требованиями:

- Включение функциональности удержания точки интереса, должно быть настраиваемым через файл конфигурации сервера.

- Если функциональность удержания точки интереса включена в конфигурационном файле, то при изменении (перемещении) системы позиционирования с использованием команд G0 и/или G6 система должна выполнять коррекцию положения стола в горизонтальной плоскости
- Корректировка положения стола при обработке команды G0. При получении команды G0 в которой нет изменения координат по осям X и/или Y, но есть изменения для любой другой оси (Z, A или C). Система позиционирования должна рассчитать требуемое перемещение стола по осям X и Y и выполнить их вместе с основным перемещением системы позиционирования.
- Корректировка положения стола при обработке команды G6 в режиме следования стола за точкой интереса. При выполнении перемещения по осям Z, A и/или C необходимо с определённой периодичностью (периодичность должна настраиваться в файле конфигурации) выполнять перемещение стола (оси X и Y) таким образом, чтобы система позиционирования обеспечивала следование точки интереса за текущими перемещениями системы позиционирования.

  После остановки перемещения выполняемого по команде G6, необходимо обеспечить финальное позиционирование системы (перемещение стола по осям X и Y) таким образом, чтобы точка фокуса оказалась в центре области видимости детектора.
- При завершении перемещения по команде G6, ответ о завершении выполнения команды должен отправляться только после завершения финальной корректировки положения стола по осям X b Y.
- В ответе о завершении выполнения команд G0 и G6 (при успешном завершении) должны содержаться только те оси, которые были указаны в команде G0 и/или G6 при её получении сервером.
- В случае получения команды G0 и/или G6, в которой указано перемещение хотя бы по одной из осей X и/или Y – корректировка положения стола для удержания точки интереса выполняться не должна.
- Допускается реализация всей функциональности описанной в данных требования в виде отдельного микро сервиса, но с соблюдением всех изложенных в требованиях требований
- Реализация всех требований, должна иметь набор тестов, покрывающих все требования для перемещения как по отдельным осям, так и по нескольким осям одновременно. Все тесты должны проверять корректность координат удерживаемой точки интереса после завершения перемещения.

8 Результаты и дальнейшая работа

За время разработки были выполнены следующие этапы:

- Реализована система позиционирования. Сейчас она находится в тестировании. Для ручного тестирования описаны основные сценарии, занесены в локальный Gitlab
- Написан веб-пульт для ручного тестирования 4. Он умеет отправлять команды и получать статус сервера. Тестировщику позволяет бесшовно общаться с модулями
- Реализован сервер пульта управления, пройдено ручное тестирование. Пульт подключался напрямую к системе позиционирования.
- Для сборки, пакетирования и тестирования развернут gitlab CI/CD [9]. На выходе получаем автоматически оттестированный пакет, который можно развернуть на стенде для дальнейшего ручного тестирования 6, 7.
- Проведен рефакторинг системы позиционирования в модуле Server. Создан базовый класс, куда вынесена общая функицональность, связанная с передачей и получением сообщений, поддержкой подключенных клиентов.

В дальнейшем планируется:

- Разработать алгоритм удержания точки интереса
- Разработать систему безопасности
- Провести дальнейший рефакторинг системы позиционирования. Собрать интерфейсную библиотеку для переиспользования в других серверах.
- Дальнейшая интеграция с ПО АРМ Оператора. ПО является главным управляющим элементом для разрабатываемых систем.

- Научиться тестировать под санитайзерами. Сделать покрытие дополнительными unit тестами. Увеличить процент покрытия
- Провести замеры производительности, fuzzy-testing
- Сделать Deploy на стенды для автоматического обновления
- Поднять Registry для хранения пакетов и Docker образов

Список литературы

1. Raspberry Pi4. — Accessed: 2024-06-13. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.
2. Gcode Marlin Documentation. — Accessed: 2024-06-13. https://marlinfw.org/meta/gcode/.
3. Json library. — Accessed: 2024-06-13. https://github.com/nlohmann/json.
4. Gcode parsing library. — Accessed: 2024-06-13. https://github.com/childhoodisend/gpr.
5. Socket library. — Accessed: 2024-06-13. https://github.com/DFHack/clsocket.
6. Logger library. — Accessed: 2024-06-13. https://gitlab.com/childhoodisend/qt-logger.
7. Logger library. — Accessed: 2024-06-13. https://doc.qt.io/qt-5/.
8. GoogleTest Framework. — Accessed: 2024-06-13. https://github.com/google/googletest.
9. Gitlab CI/CD documentation. — Accessed: 2024-06-13. https://docs.gitlab.com/ee/ci/.

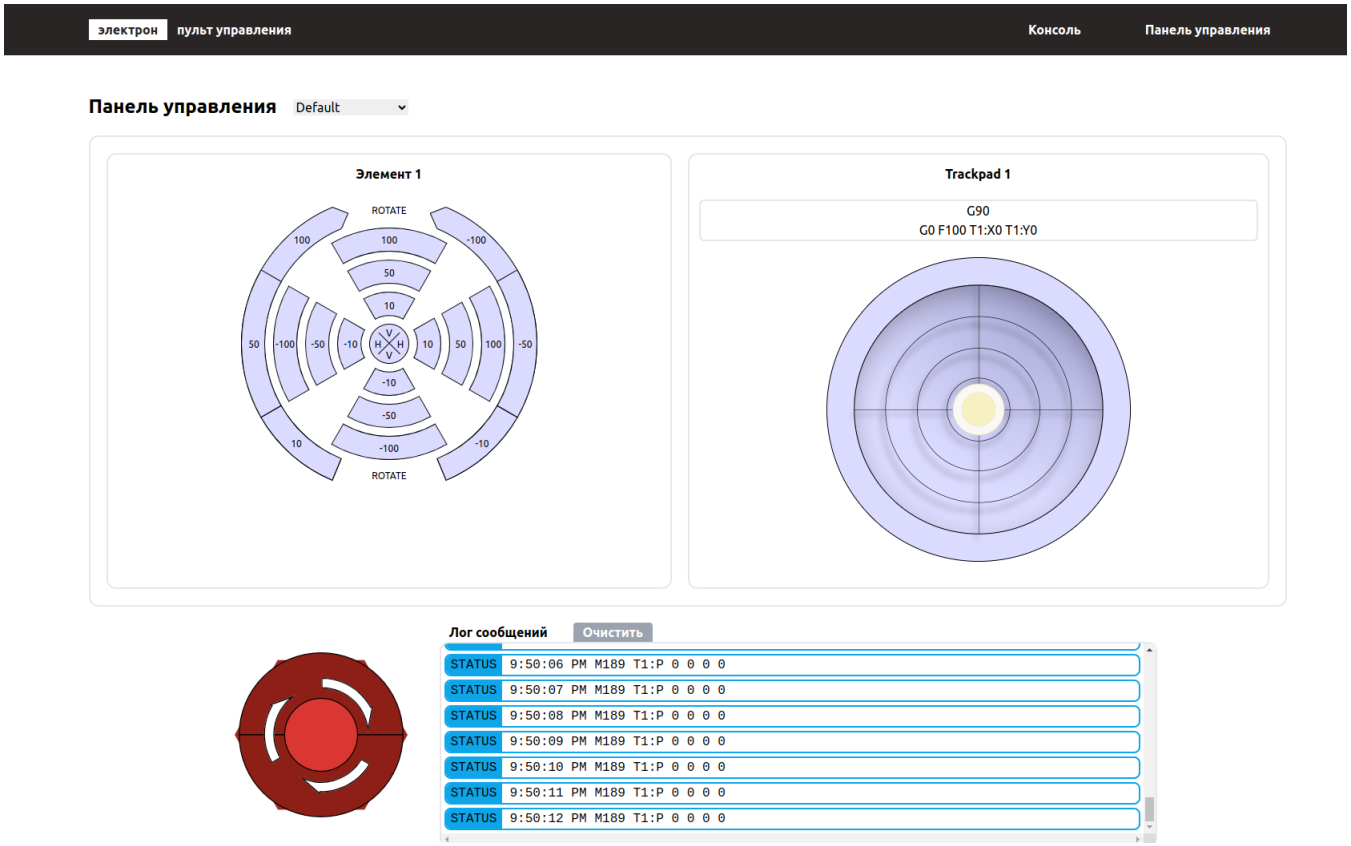


Рис. 4: **Стендовый пульт управления.** Эмулирует аппарутную часть. Способен слать команды и получать статус от системы позиционирования, безопасности, пульта управления.

Рис. 5: **Аппаратный пульт управления системой позиционирования.**

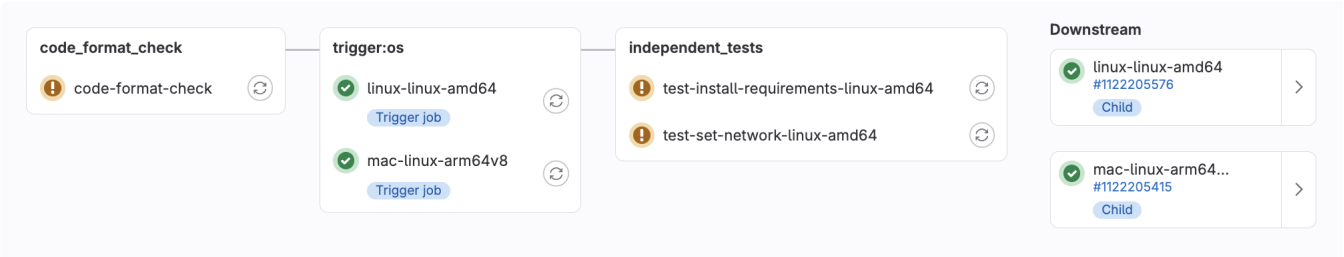


Рис. 6: **Пример Gitlab CI/CD пайплайна.** Сборка происходит в дочерних пайплайнах для архитектур **x86_64** и **arm64**.

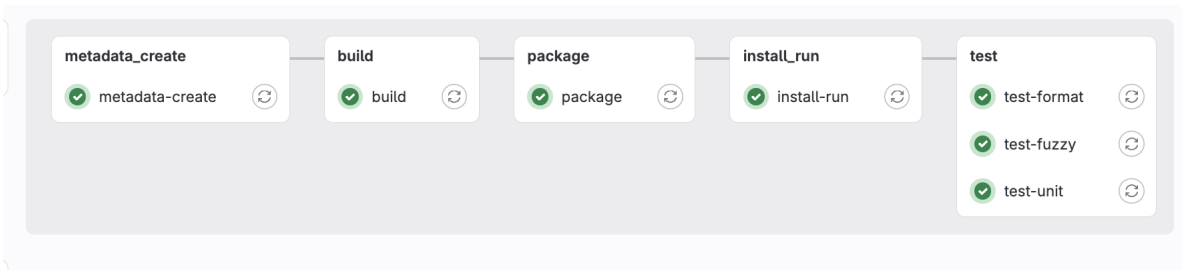


Рис. 7: **Пример дочернего пайплайна.**