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 4

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 и передает клиенту

4 Нефункциональные требования

Данные нефункциональные требования должны быть выполнены при разработке:

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

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++14. 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]

5.3 Архитектура

Структурно архитектуру серверной части можно представить следующим образом:

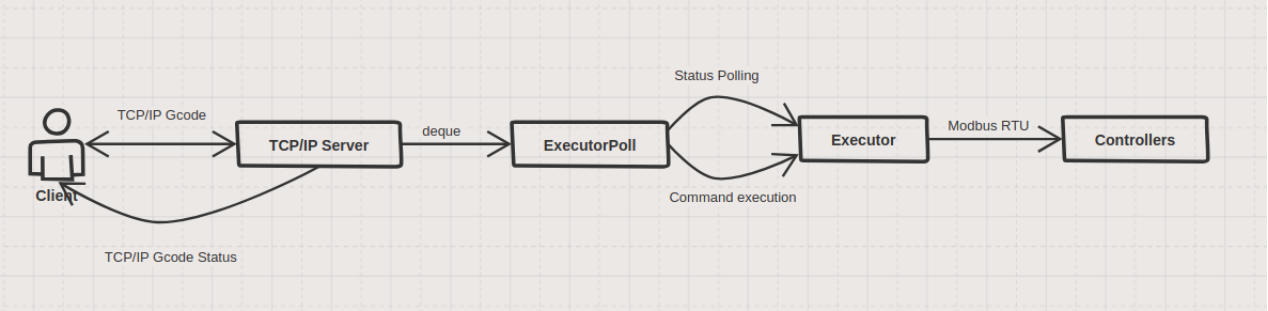

Рис. 1: **Архитектура серверной части**

Программно реализуются следующие классы:

- Класс Server. Отвечает за обработку подключений клиентов по сети. Передает сообщения в ExecutorPoll
- Класс ExecutorPoll. Валидирует сообщения (в соответствии с заданными regex), переводит сообщение из формата Gcode в байт-код. Передает байт-код классу Executor. Выполняет status polling.
- Класс Executor. Реализует интерфейс взаимодействия с контроллерами - это может быть Serial Port RS485, Ethernet, Moxa UPORT 1150 и другие. Пересылает байты контроллерам и получает ответ (если необходимо).

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

- Получить Gcode команды от клиента, провалидировать
- Спарсить команду и перевести в байт-код
- Передать байт-код контроллерам, получить ответ, обработать ответ. Если есть ошибка - обработать ошибку.
- Сформировать Gcode из полученного байт-кода

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

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

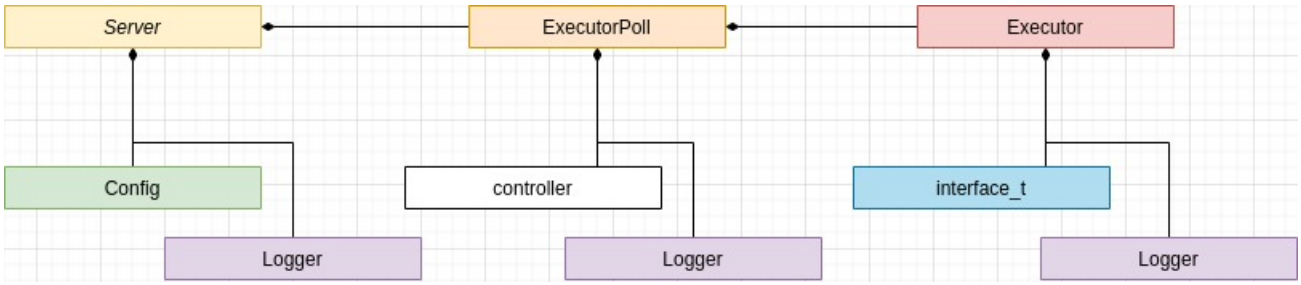

Рис. 2: **UML диаграмма классов**

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 командой M112.

6 Результаты и дальнейшая работа

Разработка ведется 2 года, за это время:

- Реализована система позиционирования. Сейчас она находится в тестировании. Для ручного тестирования описаны основные сценарии, занесены в локальный gitlab
- Написан веб-пульт для ручного тестирования 3. Он умеет отправлять команды и получать статус сервера. Тестировщику позволяет бесшовно общаться с модулями
- Для сборки, пакетирования и тестирования развернут gitlab CI/CD [8]. На выходе получаем автоматически оттестированный пакет, который можно развернуть на стенде для дальнейшего ручного тестирования 5, 6.

В дальнейшем планируется:

- Разработать систему безопасности и пульт управления
- Провести итерацию рефакторинга системы позиционирования. Возможно, вынести status polling в отдельный класс
- Подключить пульт управления напрямую к системе позиционирования, провести smoke testing
- Научиться тестировать под санитайзерами. Сделать покрытие дополнительными unit тестами
- Провести замеры производительности, fuzzy-testing

Список литературы

1. Raspberry Pi4. — Accessed: 2023-01-07. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.
2. Gocde Marlin Documentation. — Accessed: 2023-01-07. https://marlinfw.org/meta/gcode/.
3. Json library. — Accessed: 2023-01-07. https://github.com/nlohmann/json.
4. Gcode parsing library. — Accessed: 2023-01-07. https://github.com/childhoodisend/gpr.
5. Socket library. — Accessed: 2023-01-07. https://github.com/DFHack/clsocket.
6. Logger library. — Accessed: 2023-01-07. https://gitlab.com/childhoodisend/qt-logger.
7. Logger library. — Accessed: 2023-01-07. https://doc.qt.io/qt-5/.
8. Gitlab CI/CD documentation. — Accessed: 2023-01-07. https://docs.gitlab.com/ee/ci/.

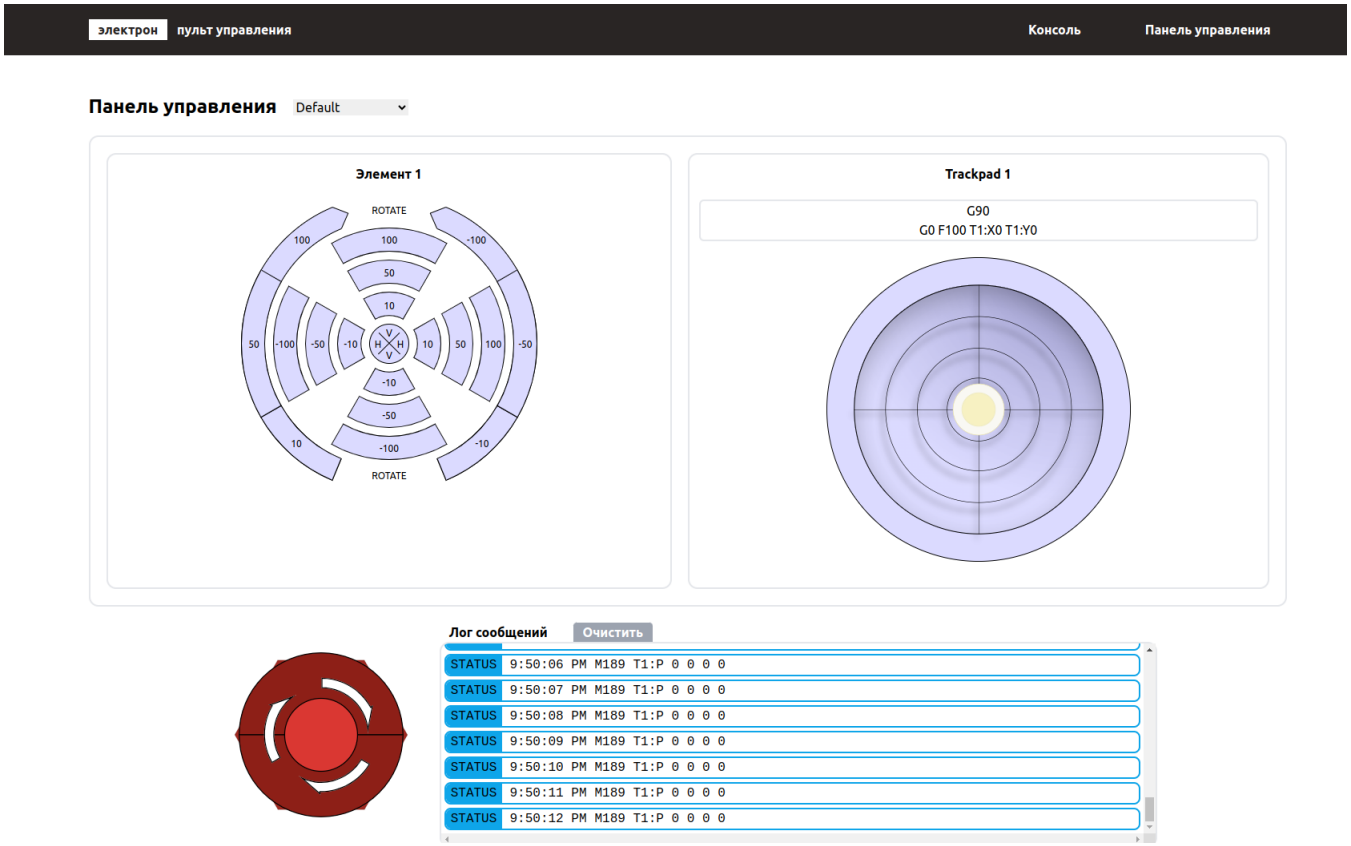

Рис. 3: **Стендовый пульт управления.** Эмулирует аппарутную часть. Способен слать команды и получать статус от системы позиционирования, безопасности, пульта управления.

Рис. 4: **Аппаратный пульт управления системой позиционирования.**

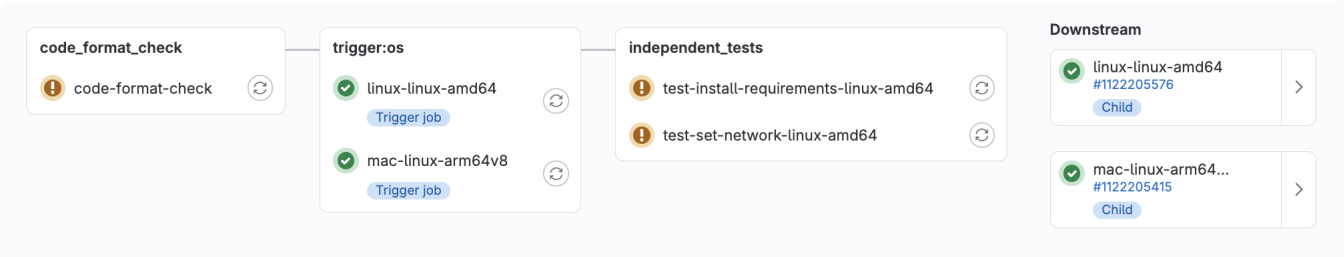

Рис. 5: **Пример Gitlab CI/CD пайплайна.** Сборка происходит в дочерних пайплайнах для архитектур **x86_64** и **arm64**.

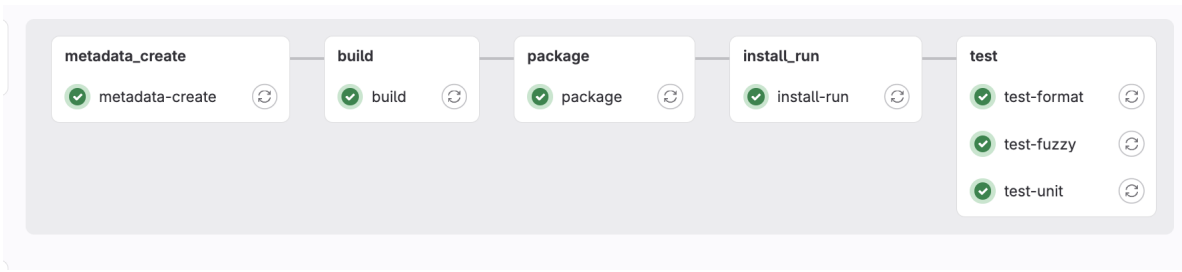

Рис. 6: **Пример дочернего пайплайна.**