НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____ 2026 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення

на тему: «Програмне забезпечення для емуляції поведінки роботизованого маніпулятора Puma560 на основі ROS2»

Виконав:
студент IV курсу, групи ТВ-21
Зіолковський Дмитро Володимирович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
к.т.н., доцент Сарибога Г.В. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
к.т.н., доцент Гуменний Д. О. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2026

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____ 2026 р.

ЗАВДАННЯ
на дипломну роботу студенту

Зіолковський Дмитро Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи: «Програмне забезпечення для емуляції поведінки роботизованого маніпулятора Puma560 на основі ROS2»
керівник роботи Сарибога Ганна Володимирівна
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «__» _____ 2026 р. №_____

2. Строк подання студентом роботи «10» червня 2026 р.
3. Вихідні дані до роботи: емулація робота Puma560 в симуляторі Gazebo; зв'язок ROS2 та STM32F411; емуляція робота на STM32F411.
4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): аналіз існуючих рішень; проектування архітектури ПЗ; симуляція робота в Gazebo, зв'язок ROS2 та STM32F411; емуляція робота на STM32F411; тестування та аналіз результатів.
5. Перелік ілюстративного матеріалу: Структура нод та топіків ROS2 у запущеному стані; Графічний інтерфейс керування суглобами; Графічне вікно RViz2 у робочому стані; Графічне вікно Gazebo у робочому стані; Діаграма потоків даних TCP-клієнта; Діаграма алгоритмів прошивки мікроконтролера;
6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	При-мітка
1	Отримання завдання	31.10.2025	
2	Дослідження предметної області	01.11.2025 – 30.11.2025	
3	Дослідження існуючих рішень	01.12.2025 – 31.12.2025	
4	Постановка вимог до проєктування системи	01.01.2026 – 31.01.2026	
5	Розробка програмного продукту	01.02.2026 – 03.05.2026	
6	Тестування	04.05.2026 – 10.05.2026	
7	Захист програмного продукту	11.05.2026 – 15.05.2026	
8	Оформлення дипломної роботи	18.05.2026 – 31.05.2026	
9	Передзахист	01.06.2026 – 05.06.2026	
10	Захист	15.06.2026 – 19.06.2026	

Студент _____ Дмитро ЗІОЛКОВСЬКИЙ

(підпис) (ім’я, ПРІЗВИЩЕ)

Керівник роботи _____ Ганна САРИБОГА

(підпис) (ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Робота містить: вступ, 4 розділи, висновки, перелік умовних позначень, список використаних джерел (не наведений у тексті) та додатки. Загальний обсяг роботи становить 48 сторінок, ілюстрована 6 рисунками.

Об’єкт дослідження – процес розробки програмного забезпечення для емуляції поведінки роботизованого маніпулятора.

Предмет дослідження – застосування сучасної системи керування ROS2 разом із засобами моделювання фізики (Gazebo) та цифрових двійників, а також інтеграція з мікроконтролерними пристроями.

Мета роботи – закріплення та практичне застосування теоретичних знань при розв’язанні інженерно-програмних завдань створення гібридної системи емуляції маніпулятора PUMA560, що поєднує віртуальне середовище та фізичний пристрій на базі STM32F4.

Методи дослідження та реалізації – використання фреймворку ROS2 (вузли, топики, сервіси, контролери), симулятора Gazebo з фізичним двигуном Bullet Featherstone, засобів візуалізації RViz2 та rqt, програмування мікроконтролера STM32F411 на мові C з бібліотекою libopencm3, розробка асинхронного TCP-клієнта на C++20 з Boost.Asio, інтеграція кастомного Hardware Interface у фреймворк ros2_control.

Основні результати роботи.

– Опрацьовано науково-технічну літературу та документацію з ROS2, Gazebo, моделі PUMA560 та засобів інтеграції з мікроконтролерами; проведено порівняльний аналіз існуючих систем і методів керування.

– Розгорнуто середовище симуляції на базі ROS2 Jazzy та Gazebo Harmonic, налаштовано модель PUMA560 у форматі URDF/SDF, запущено мости між ROS2 та Gazebo, реалізовано посуглобове керування через графічний інтерфейс.

– Розроблено кастомний Hardware Interface для ros2_control з інтегрованим TCP-клієнтом (Boost.Asio, корутини), який забезпечує асинхронний двосторонній обмін даними через UART–TCP міст (CH32V208).

– Створено прошивку для мікроконтролера STM32F411, що включає драйвери семисегментного дисплея (SPI), АЦП з DMA, обробку кнопки (переривання), UART-драйвер із кадруванням, періодичне кодування та відправлення масиву з 6 чисел, фільтр гістерезису та апаратний CRC32.

– Реалізовано два режими роботи системи: повну симуляцію в Gazebo (без фізичного обладнання) та гібридну емуляцію з підключеним STM32F4, який імітує сенсори та відображає керуючі команди.

– Проведено комплексне тестування, підтверджено коректну передачу даних між ROS2 і мікроконтролером, візуалізацію стану маніпулятора в RViz2 та Gazebo, а також керування через графічний інтерфейс.

Практичне значення – розроблена система може бути використана для подальших досліджень алгоритмів керування промисловими роботами, проведення навчальних лабораторних робіт, а також як база для віртуального введення в експлуатацію роботизованих комплексів без ризику пошкодження реального обладнання.

Висновки – усі поставлені завдання виконано в повному обсязі, мету дипломної роботи досягнуто. Отримано практичний досвід роботи з сучасним інструментарієм робототехніки (ROS2, Gazebo, STM32) та створення цілісної програмної системи емуляції роботизованого маніпулятора.

КЛЮЧОВІ СЛОВА: ЕМУЛЯЦІЯ, РОБОТИЗОВАНИЙ МАНІПУЛЯТОР, PUMA560, ROS2, GAZEBO, ЦИФРОВИЙ ДВІЙНИК, СИМУЛЯЦІЯ ФІЗИКИ, ROS2_CONTROL, HARDWARE INTERFACE, STM32F4, МІКРОКОНТРОЛЕР, TCP-КЛІЄНТ, UART, BOOST.ASIO, СЕМИСЕГМЕНТНИЙ ДИСПЛЕЙ, RVIZ, КЕРУВАННЯ СУГЛОБАМИ.

ABSTRACT

The work contains: introduction, 4 chapters, conclusions, list of symbols, 15 references, and 2 appendices. The total volume of the work is 48 pages, illustrated with 6 figures.

Object of research – the process of software development for emulating the behaviour of a robotic manipulator.

Subject of research – the application of the modern ROS2 control system together with physics simulation tools (Gazebo) and digital twins, as well as integration with microcontroller devices.

Objective of the work – consolidation and practical application of theoretical knowledge in solving engineering and software tasks of creating a hybrid emulation system for the PUMA560 manipulator that combines a virtual environment and a physical device based on STM32F4.

Research and implementation methods – use of the ROS2 framework (nodes, topics, services, controllers), the Gazebo simulator with the Bullet Featherstone physics engine, RViz2 and rqt visualisation tools, STM32F411 microcontroller programming in C with the libopencm3 library, development of an asynchronous TCP client in C++20 with Boost.Asio, integration of a custom Hardware Interface into the ros2_control framework.

Main results of the work.

– Scientific and technical literature and documentation on ROS2, Gazebo, the PUMA560 model, and microcontroller integration tools were reviewed; a comparative analysis of existing control systems and methods was conducted.

– A simulation environment based on ROS2 Jazzy and Gazebo Harmonic was deployed, the PUMA560 model in URDF/SDF format was configured, bridges between ROS2 and Gazebo were launched, and joint-level control via a graphical interface was implemented.

– A custom Hardware Interface for ros2_control with an integrated TCP client (Boost.Asio, coroutines) was developed, which provides asynchronous bidirectional data exchange via a UART–TCP bridge (CH32V208).

– Firmware for the STM32F411 microcontroller was created, including drivers for a seven-segment display (SPI), ADC with DMA, button handling (interrupt), UART driver with framing, periodic encoding and transmission of an array of 6 numbers, hysteresis filter, and hardware CRC32.

– Two operating modes of the system were implemented: full simulation in Gazebo (without physical hardware) and hybrid emulation with the connected STM32F4, which simulates sensors and displays control commands.

– Comprehensive testing was performed; correct data transfer between ROS2 and the microcontroller, visualisation of the manipulator state in RViz2 and Gazebo, as well as control via the graphical interface were confirmed.

Practical significance – the developed system can be used for further research of control algorithms for industrial robots, for conducting educational laboratory work, and as a basis for virtual commissioning of robotic complexes without the risk of damaging real equipment.

Conclusions – all assigned tasks have been fully completed, the goal of the thesis has been achieved. Practical experience was gained in working with modern robotics tools (ROS2, Gazebo, STM32) and in creating an integrated software system for emulating a robotic manipulator.

KEYWORDS: EMULATION, ROBOTIC MANIPULATOR, PUMA560, ROS2, GAZEBO, DIGITAL TWIN, PHYSICS SIMULATION, ROS2_CONTROL, HARDWARE INTERFACE, STM32F4, MICROCONTROLLER, TCP CLIENT, UART, BOOST.ASIO, SEVEN-SEGMENT DISPLAY, RVIZ, JOINT CONTROL.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ROS2	Robot Operating System 2 (Робототехнічна операційна система 2) — фреймворк для розробки програмного забезпечення роботів.
Gazebo	Симулятор фізики та 3D-середовища для моделювання роботів.
PUMA560	Programmable Universal Machine for Assembly 560 — класична модель промислового роботизованого маніпулятора з 6 ступенями свободи.
URDF	Unified Robot Description Format — формат файлу для опису кінематики, динаміки та візуальної моделі робота.
SDF	Simulation Description Format — формат для опису сцен та об’єктів у симуляторі Gazebo.
RViz	ROS Visualization — інструмент для тривимірної візуалізації даних у ROS.
rqt	ROS Qt-based GUI framework — фреймворк для створення графічних інтерфейсів користувача в ROS.
PID	Proportional-Integral-Derivative (Пропорційно-інтегрально-диференційний) — класичний алгоритм керування.
STM32F4	Серія 32-бітних мікроконтролерів від STMicroelectronics на ядрі ARM Cortex-M4.
UART	Universal Asynchronous Receiver-Transmitter — універсальний асинхронний приймач-передавач для послідовної комунікації.
TCP	Transmission Control Protocol — один з основних протоколів передачі даних в інтернеті.
SPI	Serial Peripheral Interface — послідовний інтерфейс для синхронної передачі даних між мікроконтролером і периферією.

ADC	Analog-to-Digital Converter (Аналогово-цифровий перетворювач, АЦП) — пристрій для перетворення аналогового сигналу в цифровий код.
DMA	Direct Memory Access — прямий доступ до пам’яті, що дозволяє периферійним пристроям обмінюватися даними без участі процесора.
CRC	Cyclic Redundancy Check (Циклічний надлишковий код) — алгоритм для перевірки цілісності даних.
COBS	Consistent Overhead Byte Stuffing — метод кадрування даних для надійної передачі по послідовних каналах.
libopencm3	Бібліотека для роботи з периферією мікроконтролерів на базі ARM Cortex-M з відкритим кодом.
Hardware Interface	Компонент у складі ros2_control, який забезпечує низькорівневий доступ до апаратного забезпечення (зчитування даних з сенсорів та надсилання команд на приводи).
ros2_control	Фреймворк у ROS2 для керування роботами, що розділяє високорівневі контролери та низькорівневий доступ до апаратури.
microROS	Версія ROS2, оптимізована для роботи на мікроконтролерах з обмеженими ресурсами.
CH32V208	Мікроконтролер від компанії WCH, який використовується як апаратний перетворювач UART–TCP.
Boost.Asio	Бібліотека C++ для асинхронного введення-виведення та роботи з мережею.
Потенціометр	Змінний резистор, який у цій роботі використовується для емуляції сигналу датчика положення суглоба.
Енкодер	Датчик, який перетворює кут повороту вала в електричний сигнал. У роботі його роботу емулює мікроконтролер.

ВСТУП

Об’єктом дипломної роботи є процес розробки програмного забезпечення для емуляції поведінки роботизованого маніпулятора, а предметною областю – застосування сучасної системи керування ROS2 разом із засобами моделювання фізики та цифрових двійників.

Актуальність теми зумовлена необхідністю створення віртуальних середовищ для тестування алгоритмів керування промисловими роботами без ризику пошкодження реального обладнання. Використання ROS2 та симуляторів фізики дозволяє пришвидшити розробку, знизити витрати та забезпечити високий рівень повторюваності експериментів.

Метою дипломної роботи є аналіз теоретичного матеріалу та закріплення практичних вмінь, отриманих студентом у закладі вищої освіти. Для досягнення цієї мети було поставлено такі завдання:

- опрацювати джерел за темою дипломної роботи;
- розробити ключові програмні модулі системи емуляції;
- налаштувати середовище симуляції;
- підключення зовнішніх мікроконтролерних пристроїв до ROS2.

Шляхи реалізації передбачають використання ROS2, симулятора Gazebo, мови програмування C/C++, а також інструментів для роботи з мікроконтролерами. Основним результатом дипломної роботи має стати набуття цілісного уявлення про етапи створення програмного забезпечення для емуляції роботизованих систем.

1 МЕТА, ПОСТАНОВКА ЗАДАЧІ ТА ОСНОВНІ ЗАВДАННЯ

1.1 Мета

Метою дипломної роботи є закріплення та практичне застосування теоретичних знань, при розв’язанні конкретних інженерно-програмних завдань у рамках теми дипломної роботи «Програмне забезпечення для емуляції поведінки роботизованого маніпулятора PUMA560 на основі ROS2». Крім того, мета передбачає набуття навичок роботи з сучасним інструментарієм робототехніки (ROS2, Gazebo, STM32) та формування цілісної програмної системи,здатної емулювати поведінку маніпулятора у віртуальному середовищі з можливістю підключення зовнішніх фізичних пристроїв.

1.2 Постановка задачі

Для досягнення поставленої мети необхідно виконати комплекс робіт, які охоплюють теоретичну підготовку, практичне програмування, налаштування симуляційного середовища та інтеграцію всіх компонентів у єдину систему. Задача дипломної роботи полягає в наступному:

– опрацювати науково-технічну літературу та документацію за темою;

– розгорнути середовище ROS2 і Gazebo, налаштувати модель маніпулятора PUMA560 для симуляції фізики та керування;

– розробити ключові програмні модулі (взаємодія з периферією, синхронізація цифрового двійника з фізичною моделлю);

– освоїти програмування мікроконтролера STM32F4 для виконання завдань збору даних, обробки сигналів енкодера, індикації та обміну даних;

– налагодити двосторонній канал зв’язку між ROS2 та мікроконтролером;

– провести комплексне тестування та налагодження всієї системи.

1.3 Основні завдання

Опрацювання літератури та розробка модуля взаємодії ROS2 з мікроконтролером.

– Вивчити матеріали щодо ROS2, Gazebo, моделі PUMA560, принципів емуляції роботизованих систем.

– Реалізувати канал зв’язку ROS2 та мікроконтролера.

– Забезпечити приймання від мікроконтролера стану робота та передачу цих даних у ROS2.

– Забезпечити відпарвку команд від ROS2 до мікроконтролера.

– Забеспечити емітацію роботи всіх частин робота та відправку цих даних на мікроконтролері.

– Забезбечити приймання команд на мікроконтролера та виведленя іх на дисплей.

Розробка модуля синхронізації цифрового двійника та фізичної моделі.

– Дослідити роботу прогарамно-апараного модуля Ethernet(uart-tcp(сервер) перетворувач).

– Реалізувати асинхроний tcp-кліент для ітеграції у ROS-контроллер, для прийнятя стану робота для цифровгого двійника та відправки команд роботу від системи керування.

– Резізація емуляції родботи робота на мікроконтролері: считування положення потенціометра, вивід даних на 7-сігментний дісплей, перемикання по кнопці номера суглоба, прийом та пердача даних по uart з первіркою цілосності пакету.

Висновки до розділу 1

У першому розділі чітко сформульовано мету роботи – закріплення практичних навичок і створення цілісної системи емуляції PUMA560 на базі ROS2.

Поставлено конкретні інженерно-програмні завдання, які охоплюють усі ключові етапи: від теоретичного опрацювання джерел до налагодження двостороннього зв’язку між ROS2 і мікроконтролером. Така структура забезпечує логічну послідовність виконання роботи та дозволяє комплексно підійти до розв’язання проблеми.

2 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ, МЕТОДІВ ТА МОДЕЛЕЙ

2.1 Опис предметної області

Предметна область дипломної роботи охоплює процеси моделювання, емуляції та керування поведінкою промислових роботизованих маніпуляторів із використанням сучасних засобів програмування та симуляції. Основними об’єктами є роботизований маніпулятор, його цифровий двійник (віртуальна модель, що імітує фізичну поведінку), програмне забезпечення для емуляції, алгоритми керування, а також допоміжні фізичні пристрої (мікроконтролери, сенсори). Суб’єктами виступають розробники, дослідники, викладачі, студенти та фахівці з віртуального введення в експлуатацію. Ключові процеси включають моделювання маніпулятора, симуляцію його руху з урахуванням фізичних ефектів, емуляцію поведінки, синтез алгоритмів керування (положенням, швидкістю, силою), інтеграцію з розподіленими системами керування та організацію обміну даними між віртуальним середовищем і реальними пристроями.

2.2 Аналіз існуючих систем симуляції роботизованих маніпуляторів

Для моделювання та емуляції роботизованих маніпуляторів на сьогодні існує низка програмних систем, які відрізняються функціональністю, продуктивністю, ступенем інтеграції з ROS2 та сферою застосування.

Gazebo є найбільш поширеним симулятором у ROS-спільноті. Він забезпечує високоточну симуляцію фізики (декілька фізичних двигунів на вибір), підтримує датчики, колізії, а також має глибоку інтеграцію з ROS2 через спеціальні мости. Його основні переваги – відкритий код, активна підтримка та продуктивність, достатня для задач реального часу. Недоліком є дещо застаріла візуалізація.

Webots – відкритий симулятор, що має нативну підтримку ROS2. Він зручний для освітніх цілей і швидкого прототипування, але поступається Gazebo в гнучкості налаштувань фізики.

CoppeliaSim (раніше V-REP) пропонує розширені можливості для моделювання складних кінематичних ланцюгів, підтримує різні фізичні двигуни та інтегрується з ROS2 через вузли. Використовується переважно в дослідженнях алгоритмів керування.

Unity + ROS – відносно новий напрямок, що базується на ігровому рушії Unity. Забезпечує фотореалістичну графіку, підтримку віртуальної реальності, проте вимагає більше обчислювальних ресурсів і має меншу поширеність у промислових додатках.

З огляду на тему дипломної роботи, Gazebo обрано як базовий симулятор завдяки його стандартності в екосистемі ROS2, зрілості та можливості легкого підключення зовнішніх модулів.

2.3 Аналіз методів керування маніпуляторами

1. Маніпулятор PUMA560 є класичним об’єктом для відпрацювання алгоритмів керування. У літературі описано декілька підходів, які умовно можна поділити на базові та вдосконалені.

2. Класичний PID-регулятор є найпростішим і найпоширенішим методом для керування положенням кожного суглоба окремо. Він забезпечує прийнятну точність при незмінних умовах навантаження, але має обмежену стійкість до зовнішніх збурень та варіацій параметрів маніпулятора. Більшість навчальних та експериментальних систем на базі PUMA560 використовують саме PID.

3. Гібридні та нечіткі регулятори (Fuzzy-PID, нечіткий PD+I) дозволяють покращити якість керування за наявності невизначеностей. Дослідження показують, що гібридний Fuzzy-PID контролер забезпечує кращу збіжність і меншу

перерегуляцію порівняно з класичним PID. Такі методи потребують налаштування функцій належності та бази правил, що ускладнює їх реалізацію.

3. Адаптивні та робастні методи (ковзний режим, обчислювальний момент) використовуються для компенсації динаміки маніпулятора. Вони забезпечують високу точність навіть при змінному навантаженні, але вимагають значного обсягу обчислень і повної моделі динаміки.

2.4 Аналіз засобів взаємодії ROS2 з мікроконтролерами

1. **microROS** – офіційне рішення для вбудованих систем. Воно реалізує базові ROS2-конструкції (publisher, subscriber, сервіси) на мікроконтролері, працює поверх FreeRTOS. microROS дозволяє інтегрувати пристрій як повноцінний вузол ROS2, але потребує значних ресурсів (близько 200 КБ RAM, 1 МБ Flash) та складної процедури налаштування. Для багатьох задач, особливо на платах з обмеженою пам’яттю, це є надлишковим.

2. **ros2_control** з Hardware Interface – це стандартний фреймворк ROS2 для керування роботами, який розділяє високорівневі контролери та низькорівневий доступ до апаратури. Hardware Interface – це драйвер, що реалізує методи read() (зчитування даних з сенсорів) та write() (надсилання команд на приводи) через фізичний канал зв’язку (UART, CAN, USB тощо) або симулятор. Контролери (PID, траєкторій) виконуються на хост-комп’ютері в ROS2. Для мікроконтролера не потрібні RTOS чи microROS – достатньо простого протоколу обміну по UART, вимоги до ресурсів мінімальні. Переваги: чітке розділення обов’язків, легке перемикання між симуляцією та реальним пристроєм (через gazebo_ros2_control), повторне використання готових контролерів, низькі вимоги до апаратної частини. Недоліки: необхідність самостійної розробки Hardware Interface та протоколу обміну, додаткові затримки.

Висновки до розділу 2

У розділі виконано ґрунтовний аналіз існуючих систем симуляції (Gazebo, Webots, CoppeliaSim, Unity+ROS), методів керування маніпуляторами (PID, Fuzzy-PID, адаптивні) та засобів інтеграції ROS2 з мікроконтролерами (microROS, ros2_control). Визначено, що для задач роботи найбільш доцільним є використання Gazebo як стандартного симулятора в екосистемі ROS2, класичного PID-регулятора для простоти реалізації та підходу з кастомним Hardware Interface замість microROS через надмірні ресурсні вимоги останнього та відсутність контролю цілісності даних на транспортному рівні.

3 ОБГРУНТУВАННЯ ОБРАНИХ РІШЕНЬ

3.1 Вибір середовища симуляції

Попри наявність альтернатив, таких як Webots, CoppeliaSim та Unity+ROS, основну увагу приділено Gazebo. Цей вибір базується на наступних критеріях:

Нативна інтеграція з ROS2: Gazebo є стандартним симулятором для екосистеми ROS2. Він підтримує прямий обмін повідомленнями через ROS2-мости, що дозволяє використовувати стандартні інтерфейси типу sensor_msgs, geometry_msgs та trajectory_msgs без додаткових прошарків адаптації.

Підтримка фізичних двигунів: Gazebo надає вибір фізичних двигунів (ODE, Bullet, Simbody, DART). Для задач динаміки PUMA560 критично важливо мати точний розрахунок інерційних характеристик, тертя в суглобах та реакцій на зовнішні впливи. Можливість налаштування кроку симуляції (step size) дозволяє синхронізувати фізику з реальним часом (Real-Time Factor), що необхідно для подальшого підключення реальних алгоритмів керування без внесення фазових запізнень.

Сумісність з ros2_control: Наявність пакету gazebo_ros2_control дозволяє легко підключати стандартні контролери безпосередньо до віртуальної моделі. Це забезпечує уніфікацію керування: код, написаний для симуляції, з мінімальними змінами може бути перенесений на реальну апаратну платформу.

Робота з URDF: Оскільки модель PUMA560 постачається у форматі URDF, Gazebo забезпечує найкращу сумісність для його парсингу, додавання тегів інерційних параметрів та візуалізації.

3.2 Вибір архітектури керування в ROS2

При побудові каналу зв’язку між ROS2 та мікроконтролером було відкинуто варіант використання microROS на користь класичного ros2_control з кастомним Hardware Interface. Це обґрунтовується наступними чинниками:

Контроль цілісності даних: У дослідженні було виявлено, що microROS при роботі поверх TCP/UDP не гарантує перевірки цілісності кожного окремого пакета на рівні транспортного протоколу в умовах високого навантаження. Для керування маніпулятором PUMA560 (6 ступенів свободи) втрата або спотворення навіть одного значення кута суглоба може призвести до аварійної поведінки. Натомість кастомний Hardware Interface дозволяє реалізувати фреймовий протокол UART із включенням стартового байту, перевіркою розміру пакету та 16-бітною контрольною сумою (CRC-16).

Зменшення вимог до ресурсів мікроконтролера: microROS потребує операційної системи реального часу (RTOS), близько 200 КБ RAM та понад 1 МБ Flash-пам’яті. Плата STM32F4 (навіть з великим обсягом пам’яті) при використанні RTOS ускладнює налагодження периферії та підвищує енергоспоживання. Запропонований підхід (Hardware Interface) вимагає лише простої UART-периферії та таймерів, що дозволяє писати прошивку навіть у «голому» режимі (bare-metal) з бібліотекою libopencm3, що значно спрощує відлагодження.

Гнучкість контролерів: У підході ros2_control всі обчислення PID/PD регуляторів відбуваються на хост-комп’ютері (в середовищі ROS2), а не на STM32. Це дозволяє змінювати коефіцієнти підсилення контролерів на льоту (online tuning) через параметричні сервери (ros2 param set), не перепрошиваючи мікроконтролер.

3.3 Вибір апаратної платформи та інструментарію програмування

STM32F4: вибір саме цієї серії зумовлений наявністю апаратного блоку підрахунку імпульсів, великою кількістю 16-бітних та 32-бітних таймерів для генерації ШІМ-сигналів, а також вбудованим FPU (Cortex-M4), що дозволяє швидко обробляти обчислення швидкості та позиції з плаваючою комою.

libopencm3: відмова від стандартної HAL (Hardware Abstraction Layer) від ST та використання libopencm3 обґрунтована швидкістю роботи коду та прозорістю доступу до регістрів. HAL генерує багато зайвих перевірок, що сповільнює обробку переривань від енкодерів. libopencm3 надає оптимізований низькорівневий доступ, що критично при емуляції енкодера (де потрібно формувати ШІМ з мінімальним джиттером) та при одночасному опитуванні потенціометра і кнопки.

3.4 Вибір методу емуляції сенсорів

У реальному промисловому роботі для зворотного зв’язку використовуються дорогі абсолютні або інкрементальні енкодери. У межах цієї роботи обрано метод програмної емуляції енкодера замість його фізичного підключення. Це рішення має наступні переваги:

Безпека та керованість: Емуляція дозволяє задавати будь-яку частоту обертання (від 0 до максимуму) за допомогою потенціометра без ризику пошкодження двигунів чи механіки, що особливо важливо на етапі налагодження алгоритмів обробки сигналів.

Гнучкість налаштувань: Кількість імпульсів на оберт (PPR) можна легко змінювати програмно, що дозволяє тестувати стійкість алгоритмів оцінки швидкості до різної роздільної здатності енкодерів.

Мінімізація апаратних витрат: Емуляція реалізується за допомогою двох ШІМ-каналів (з фазовим зсувом 90°) на тій самій STM32F4, що усуває потребу в зовнішньому генераторі сигналів або реальному оптичному енкодері.

3.5 Вибір комунікаційної архітектури

Вибір каналу UART-TCP та паралельного TCP-клієнта: Для передачі даних між STM32 та ROS2 обрано схему STM32 (UART) -> готовий перетворювач UART-TCP (сервер) -> TCP-клієнт у ROS2. Використання готового перетворювача дозволяє абстрагуватися від складності мережевих стеків на мікроконтролері (не потрібно реалізовувати lwIP на STM32), що зменшує обсяг прошивки та підвищує стабільність. TCP-клієнт, реалізований в окремому паралельному потоці всередині ROS2-вузла, забезпечує асинхронний прийом даних, не блокуючи основний цикл обробки управлінських команд . При цьому пакет з 6 значень (відповідає 6-ти ступеням свободи PUMA560) є оптимальним за розміром для передачі по UART та забезпечує актуальну частоту оновлення стану суглобів.

Висновки до розділу 3

Обґрунтування обраних рішень виконано з урахуванням практичних вимог до системи. Вибір Gazebo обумовлено його нативною інтеграцією з ROS2, підтримкою сучасних фізичних двигунів та сумісністю з ros2_control. Відмова від microROS на користь ros2_control із кастомним Hardware Interface дозволила зменшити вимоги до пам’яті мікроконтролера, спростити налагодження та забезпечити перевірку цілісності пакетів через власний протокол. Вибір STM32F4 обґрунтований наявністю необхідної периферії та FPU, а використання libopencm3 – швидкодією та прозорістю доступу до регістрів. Емуляція енкодера замість фізичного підключення підвищує безпеку та гнучкість налаштувань, а комунікація через UART–TCP міст забезпечує стабільність і мінімальне навантаження на МК.

4 ОПИС РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ

4.1 Загальна архітектура системи

Система підтримує два взаємовиключними режими роботи, які використовуються окремо залежно від наявності фізичного обладнання.

Режим 1 – Повна симуляція (без фізичного пристрою). У цьому режимі вся поведінка маніпулятора PUMA560 емулюється виключно програмними засобами.

– Gazebo – симулятор фізики, який моделює динаміку маніпулятора, взаємодію з оточенням, колізії та тертя. Модель PUMA560 налаштована та інтегрована через URDF/SDF.

– RViz – інструмент візуалізації для відображення стану маніпулятора в тривимірному просторі, отримання трансформацій tf2.

– Графічне вікно керування суглобами (наприклад, rqt_joint_trajectory_controller або власний GUI), яке дозволяє оператору в реальному часі задавати бажані кути для кожного суглоба або траєкторії руху.

У режимі 1 відсутній будь-який фізичний підключений пристрій. Команди з GUI надходять безпосередньо до контролерів ROS2 (наприклад, joint_trajectory_controller), які через ros2_control та стандартний gazebo_ros2_control (або прямий ros_gz_bridge) керують моделлю в Gazebo. Зворотний зв’язок (поточні позиції суглобів) отримується з симуляції та відображається в RViz.

Режим 2 – Гібридна емуляція з фізичним пристроєм. У цьому режимі Gazebo не використовується. Замість нього фізичну поведінку «робота» імітує зовнішній пристрій на базі STM32F4.

– STM32F4 – мікроконтролер, який виконує роль фізичної емуляції: зчитує потенціометр для імітація кутів суглобів робота, керує семисегментним дисплеєм, для відображення отриманих команд, обробляє кнопку ядка оберая суглоб який імітується, а також приймає та надсилає дані через UART.

– Готовий програмно-апаратний модуль UART–TCP-сервер – пристрій на базі CH32V208, який перетворює UART-пакети в TCP-пакети та навпаки. Виконує роль мосту між послідовним портом STM32F4 та Ethernet мережею.

– Hardware Interface з TCP-клієнтом – кастомний компонент ros2_control, реалізований на стороні ROS2. Він містить TCP-клієнт на основі boost::asio, який встановлює з'єднання з UART–TCP-сервером. Цей інтерфейс реалізує методи read() (отримання даних від STM32F4: 6 значень з потенціометра) та write() (надсилання команд до STM32F4: 6 чисел для відображення на дисплеї). Таким чином, ros2_control отримує зворотний зв'язок від фізичного пристрою та передає йому керуючі впливи.

– RViz – використовується для візуалізації стану, але цей стан формується на основі даних, що надходять від STM32F4 (через Hardware Interface), а не від Gazebo.

– Графічне вікно керування суглобами – те саме, що в режимі 1. Команди від оператора надходять до контролерів ROS2, далі через Hardware Interface та TCP-клієнт – на UART–TCP-сервер, потім по UART на STM32F4, який виводить отримані числа на дисплей.

4.2 Launch-файл для запуску симуляційного середовища

Для автоматизації запуску всіх необхідних вузлів у режимі повної симуляції створено launch-файл (формат XML, сумісний з ROS2 Jazzy). Він послідовно виконує наступні дії.

Визначення глобальних змінних.

```
<let name="use_sim_time" value="true"/>
<let name="robot_description" value="$(command 'xacro $(find-pkg-share puma560_description)/urdf/
puma560_robot.urdf.xacro')"/>
<let name="camera_description" value="$(find-pkg-share puma560_description)/world/
overhead_camera.sdf"/>
```

Лістинг 4.1 — Визначення глобальних змінних

use_sim_time=true – всі вузли використовують симуляційний час Gazebo замість системного.

robot_description – команда xacro генерує URDF-опис маніпулятора PUMA560 з параметризованого xacro-файлу.

camera_description – шлях до SDF-файлу навісної камери.

Вузол публікації кінематичних трансформацій.

```
<node pkg="robot_state_publisher" exec="robot_state_publisher" name="robot_state_publisher">
    <param name="robot_description" value="$(var robot_description)"/>
    <param name="use_sim_time" value="$(var use_sim_time)"/>
</node>
```

Лістинг 4.2 — Запуск вузола публікації кінематичних трансформацій

Цей вузол читає URDF, обчислює перетворення між суглобами на основі поточних кутів та публікує їх. Необхідний для коректної роботи RViz.

Запуск Gazebo Harmonic.

```
<set_env name="GZ_SIM_RESOURCE_PATH" value="$(find-pkg-share puma560_description)/../../share/"/>
<include file="$(find-pkg-share ros_gz_sim)/launch/gz_sim.launch.py">
    <arg name="gz_args" value="-v 4 -r empty.sdf --physics-engine gz-physics-bullet-featherstone-
plugin"/>
</include>
```

Лістинг 4.3 — Запуск Gazebo Harmonic

Змінна середовища GZ_SIM_RESOURCE_PATH вказує Gazebo, де шукати моделі, світ та sdf-файли.

Запускається Gazebo з пустим світом (empty.sdf) та фізичним двигуном Bullet Featherstone (оптимізований для гіллястих кінематичних ланцюгів, таких як PUMA560).

Спавн моделей у світі Gazebo.

```
<node pkg="ros_gz_sim" exec="create" name="spawn_robot" output="screen"
      args="-topic robot_description -name puma560"/>
<node pkg="ros_gz_sim" exec="create" name="spawn_camera" output="screen"
      args="-file $(var camera_description) -name overhead_camera -x 0 -y 0 -z 2 -R 0 -P 1.57 -Y 0"/
>
```

Лістинг 4.4 — Спавн моделей у світі Gazebo

– Перший вузол створює екземпляр маніпулятора з топика robot_description та дає йому ім’я puma560.

– Другий вузол створює камеру, розташовуючи її на висоті 2 м над роботом з нахилом вниз, щоб спостерігати за робочою зоною.

запустк мостів між ROS2 та Gazebo.

```
<node pkg="ros_gz_bridge" exec="parameter_bridge" name="camera_bridge"
      args="/overhead_camera/image_raw@sensor_msgs/msg/Image@gz.msgs.Image"/>
<node pkg="ros_gz_bridge" exec="parameter_bridge" name="clock_bridge"
      args="/clock@rosgraph_msgs/msg/Clock[gz.msgs.Clock"/>
```

Лістинг 4.5 — Мости між ROS2 та Gazebo

camera_bridge – перетворює повідомлення зображення з камери з формату Gazebo (gz.msgs.Image) у стандартний ROS2-топік sensor_msgs/Image.

clock_bridge – передає симуляційний час з Gazebo у ROS2 (топік /clock), забезпечуючи синхронізацію часу.

Запуск контролерів.

```
<node pkg="controller_manager" exec="spawner" name="joint_state_broadcaster_spawner"
      args="-c controller_manager joint_state_broadcaster"/>
<node pkg="controller_manager" exec="spawner" name="joint_trajectory_controller_spawner"
      args="-c controller_manager joint_trajectory_controller"/>
```

Лістинг 4.6 — Запуск контролерів

controller_manager запускає два стандартні контролери:

joint_state_broadcaster – публікує поточні позиції, швидкості та зусилля суглобів у топік /joint_states.

joint_trajectory_controller – відповідає за відпрацювання траєкторій, отриманих через топік /joint_trajectory_controller/command, та подає керуючі сигнали на модель Gazebo через ros2_control та gazebo_ros2_control.

Запуск контролерів та графічних інструментів.

```
<node pkg="rqt_joint_trajectory_controller" exec="rqt_joint_trajectory_controller"
name="joint_controller_gui">
    <param name="use_sim_time" value="$(var use_sim_time)"/>
</node>
<node pkg="rviz2" exec="rviz2" name="rviz2"
      args="-d $(find-pkg-share puma560_description)/config/rviz.yaml"/>
```

Лістинг 4.7 — Запуск графічних інструментів

rqt_joint_trajectory_controller – графічний інтерфейс, який дозволяє оператору вручну задавати цільові кути суглобів (або завантажувати траєкторії) і запускати їх виконання.

rviz2 – візуалізатор стану робота, завантажує збережену конфігурацію (панелі, фіксований кадр, відображення моделі та траєкторій).

Усі графічні інструменти rqt та rviz2 отримують параметр use_sim_time=true, що синхронізує їх відображення із симуляційним часом Gazebo.

Порядок роботи.

Після виконання launch-файлу.

– Відкривається вікно Gazebo з порожнім світом, куди завантажується модель PUMA560.

– RViz показує тривимірну сцену з тим самим роботом, а також доступні трансформації.

– Користувач отримує графічний інтерфейс для керування суглобами. Рухаючи повзунки, він задає кути суглобів, які через контролер передаються в Gazebo, і маніпулятор відтворює рух.

– Камера в Gazebo транслює зображення, яке можна використовувати для алгоритмів технічного зору.

Цей launch-файл є центральним для режиму 1 (повна симуляція) і не потребує наявності фізичного обладнання.

```
<?xml version="1.0"?>

<launch>

    <let name="use_sim_time" value="true"/>


    <let name="robot_description"

         value="$(command 'xacro $(find-pkg-share puma560_description)/urdf/

puma560_robot.urdf.xacro')"/>


    <let name="camera_description"

         value="$(find-pkg-share puma560_description)/world/overhead_camera.sdf"/>


    <node pkg="robot_state_publisher" exec="robot_state_publisher" name="robot_state_publisher">

        <param name="robot_description" value="$(var robot_description)"/>

        <param name="use_sim_time" value="$(var use_sim_time)"/>

    </node>
```

```
    <set_env name="GZ_SIM_RESOURCE_PATH" value="$(find-pkg-share puma560_description)/../../share/"/
>


    <include file="$(find-pkg-share ros_gz_sim)/launch/gz_sim.launch.py">
        <arg name="gz_args" value="-v 4 -r empty.sdf --physics-engine gz-physics-bullet-
featherstone-plugin"/>
    </include>


    <node pkg="ros_gz_sim" exec="create" name="spawn_robot" output="screen"
          args="-topic robot_description -name puma560"/>


    <node pkg="ros_gz_sim" exec="create" name="spawn_camera" output="screen"
          args="-file $(var camera_description) -name overhead_camera -x 0 -y 0 -z 2 -R 0 -P 1.57 -Y
0"/>


    <node pkg="ros_gz_bridge" exec="parameter_bridge" name="camera_bridge"
          args="/overhead_camera/image_raw@sensor_msgs/msg/Image@gz.msgs.Image"/>


    <node pkg="ros_gz_bridge" exec="parameter_bridge" name="clock_bridge"
          args="/clock@rosgraph_msgs/msg/Clock[gz.msgs.Clock"/>


    <node pkg="controller_manager" exec="spawner" name="joint_state_broadcaster_spawner"
          args="-c controller_manager joint_state_broadcaster"/>


    <node pkg="controller_manager" exec="spawner" name="joint_trajectory_controller_spawner"
          args="-c controller_manager joint_trajectory_controller"/>


    <node pkg="rqt_joint_trajectory_controller" exec="rqt_joint_trajectory_controller"
name="joint_controller_gui">
        <param name="use_sim_time" value="$(var use_sim_time)"/>
    </node>
```

```
    <node pkg="rviz2" exec="rviz2" name="rviz2"

          args="-d $(find-pkg-share puma560_description)/config/rviz.yaml"/>

</launch>
```

Лістинг 4.7 — Повний launch-файл

4.3 Launch-файл для роботи з фізичним пристроєм

Для режиму гібридної емуляції (без Gazebo, з реальним STM32F4) використовується окремий launch-файл.

Відсутнє Gazebo. Весь блок запуску Gazebo (include, spawn_robot, spawn_camera, bridges) вилучено. Це перемикає систему з віртуальної симуляції на роботу з реальним апаратним забезпеченням.

Явний запуск ros2_control_node. У симуляційному режимі контролери запускалися через spawner, але сам менеджер контролерів вже був активований Gazebo-мостом. У режимі реального пристрою необхідно окремо запустити вузол ros2_control_node, який буде керувати кастомним Hardware Interface.

Завантаження конфігурації контролерів (controllers.yaml) Цей файл визначає: які контролери будуть активні, тип апаратного інтерфейсу, спільні налаштування (частота оновлення, команди інтерфейсу).

4.4 Робочій стан програмного забезпечення

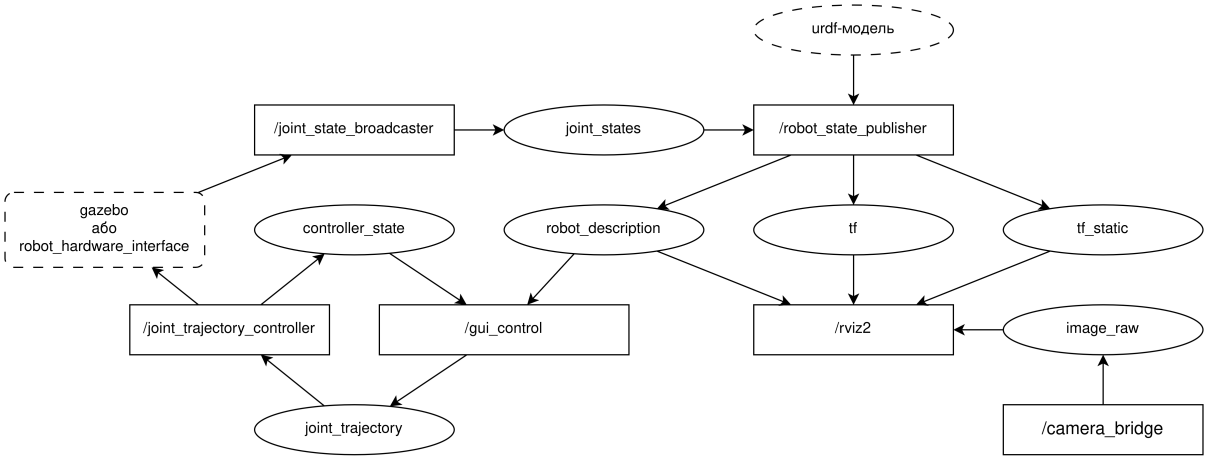

Рисунок 4.1 — Структура нод та топіків ROS2 у запущеному стані.

На рисунку 4.1 зображено структуру нод та топіків ROS2 у запущеному стані. Зображенео які ноди читають та пишуть які саме ноди.

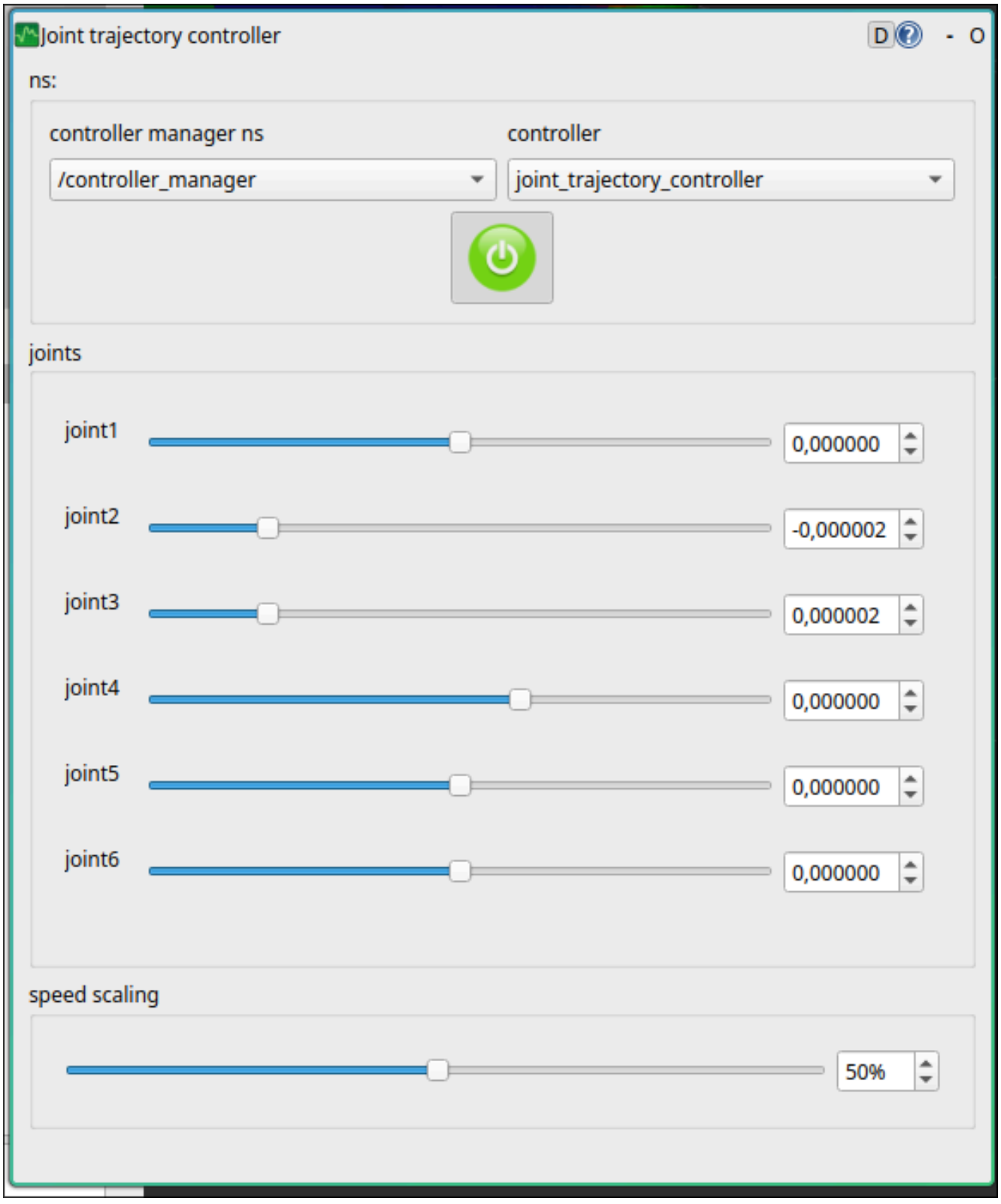

Рисунок 4.2 — Графічний інтерфейс керування суглобами.

На рисунку 4.2 зображено графічне відко завдяки якого можна відстежити поточний стан сугдобів та надати команди до емулацій у який стан перевести кожен суглоб.

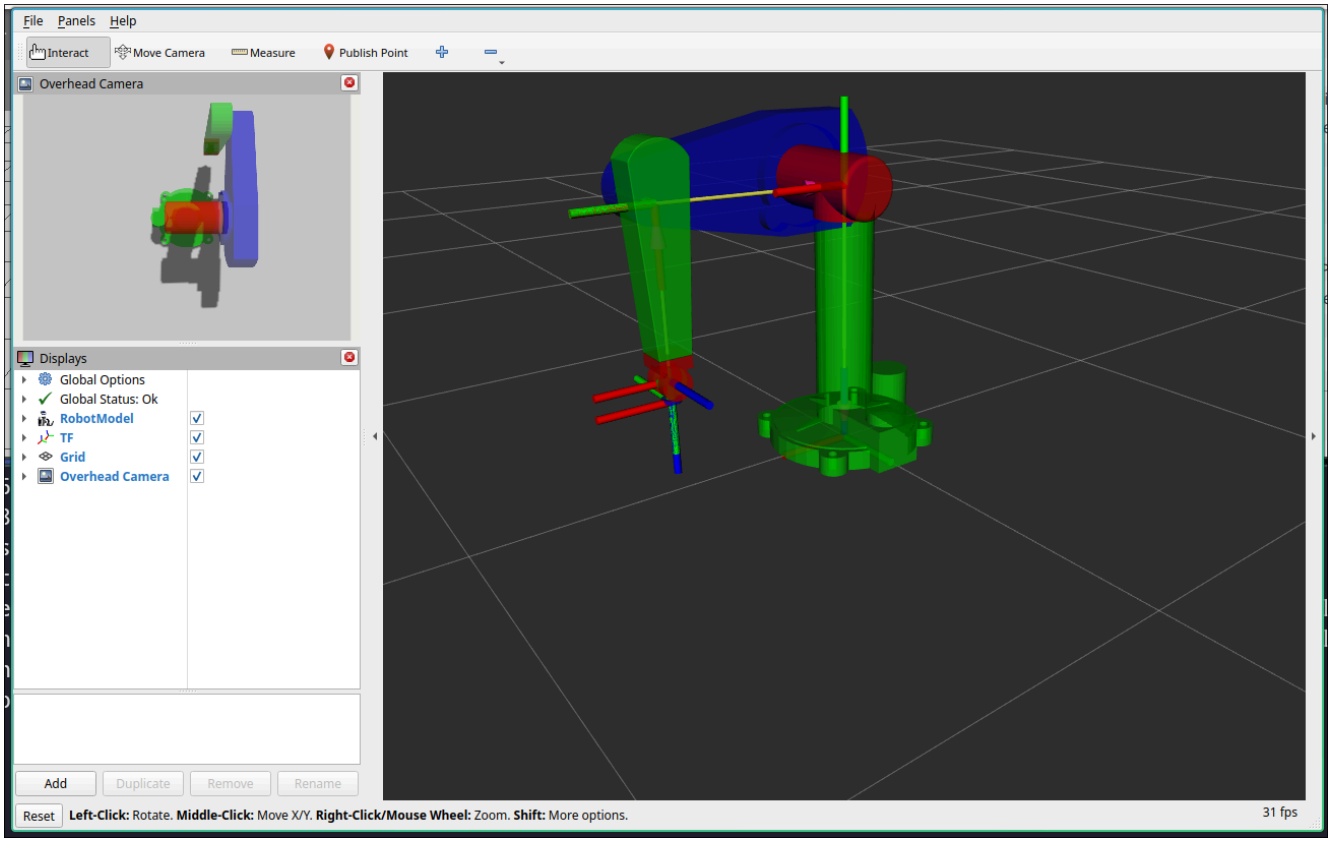

Рисунок 4.3 — Графічне вікно Rviz2 у робочому стані.

На рисунку 4.3 зображено вікно Rviz2 в якому відображаеться модель та скелет робота у поточному стані, також відображаеться зображення з відео камери.

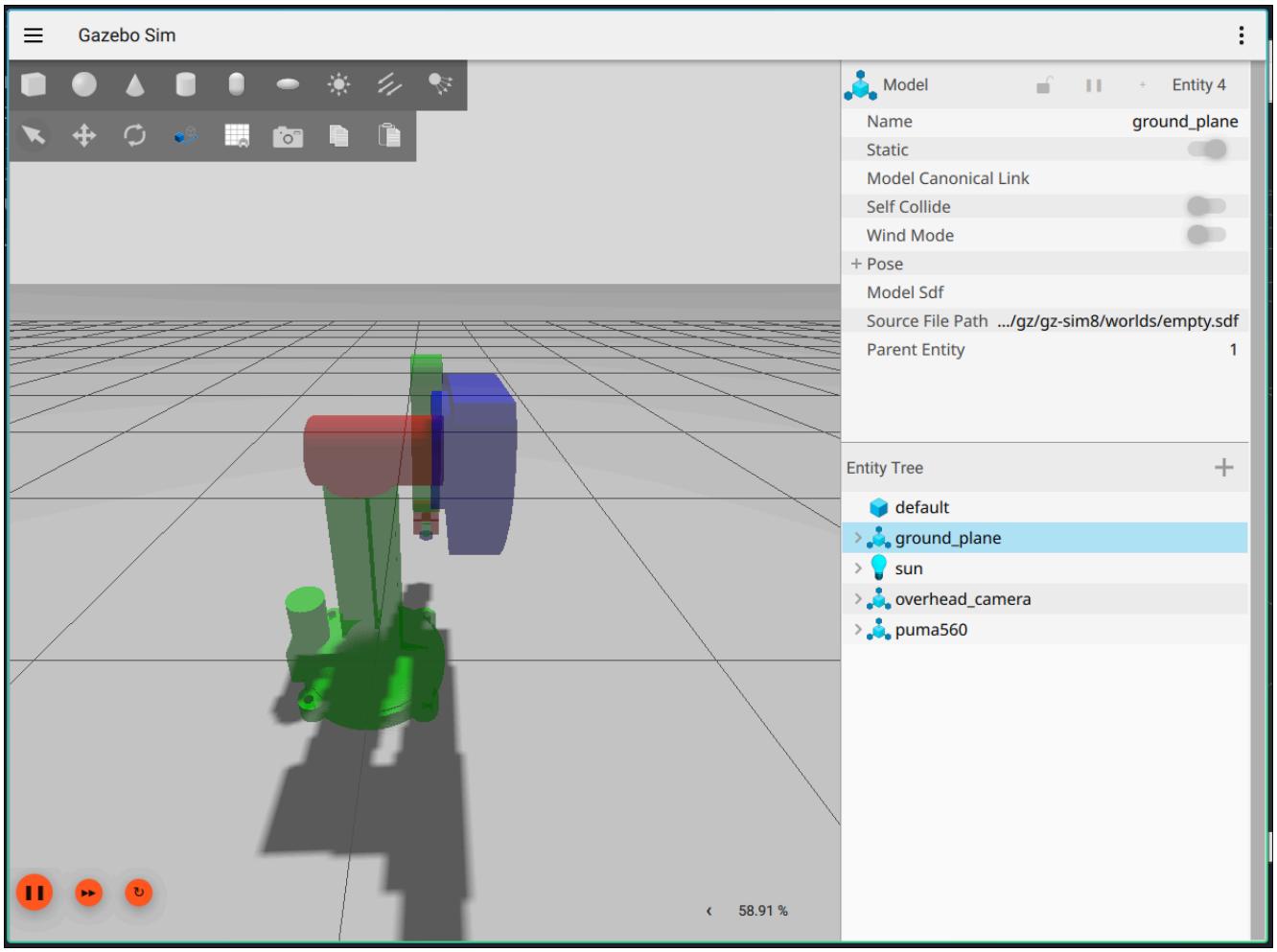

Рисунок 4.4 — Графічне вікно Gazebo у робочому стані.

На рисунку 4.4 зображено вікно Gazebo в якому відображається емуляція робота у поточному стані.

4.5 Реалізація TCP-клієнта

TCP-клієнт є ключовим компонентом кастомного Hardware Interface. Він забезпечує двосторонній асинхронний обмін даними між ROS2 та UART–TCP-сервером (CH32V208). Клієнт реалізований на мові C++20 з використанням бібліотеки Boost.Asio та корутин (C++20 coroutines), що дозволяє писати асинхронний код у лінійному стилі. Діагармрма потоків даних комунікації зображенна на рисунку 4.5.

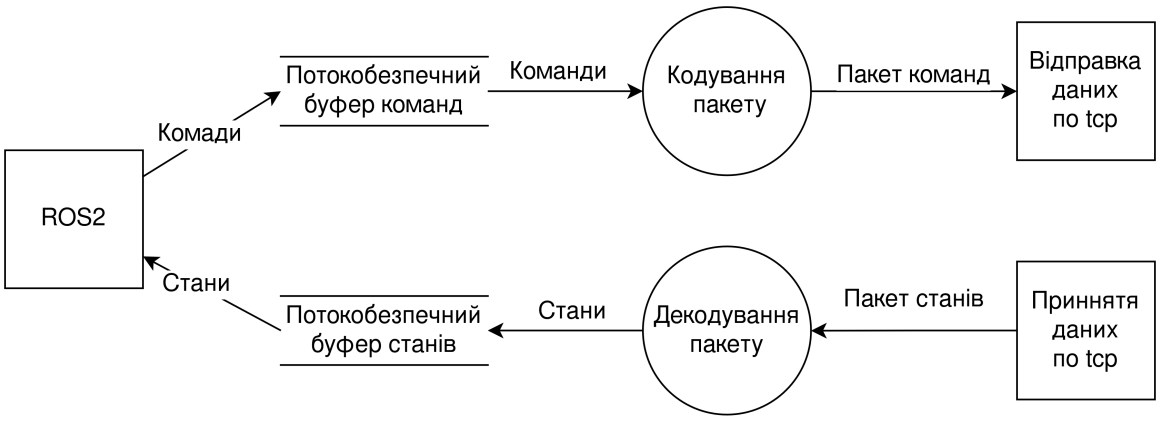

Рисунок 4.5 — Діаграмма потокікв даних tcp-клієнта.

Клас tcp_client інкапсулює всі необхідні компоненти для роботи з TCP-з'єднанням.

– `boost::asio::io_context io` - контекст асинхронних операцій

– `boost::asio::ip::tcp::socket socket` - TCP-сокет

– `boost::asio::experimental::concurrent_channel<void(boost::system::erro`
`std::array<float, 6>)> send_channel` - канал для передачі даних у корутину-відправник

– `std::array<float, 6> recv_buf` - буфер отриманих даних

– `std::mutex recv_mtx` - м'ютекс для доступу до recv_buf

– `std::jthread worker` - робочий потік, якиму віюувается вся роботаз з TCP

– `bool running` - прапорець стану

Метод start(host, port).

– Перевіряє, чи клієнт вже запущено.

– Запускає корутину connect(host, port) за допомогою boost::asio::co_spawn. Ця корутина асинхронно резолвить ім'я хоста, встановлює TCP-з'єднання та породжує корутини sender() і receiver().

– Створює std::jthread, який викликає io.run(). Завдяки std::stop_callback, при запиті зупинки потоку викликається io.stop(), що коректно завершує всі асинхронні операції.

Метод stop().

– Викликає socket.close() через post в контексті io, щоб перервати будь-які очікуючі операції читання/запису.

– Запитує зупинку робочого потоку (worker.request_stop()).

– Скидає прапорець running.

Метод send(data).

– Використовує concurrent_channel (безпечний для багатьох продюсерів), щоб передати масив з 6 чисел у корутину sender().

– Канал має ємність 1 (send_channel(io, 1)). Якщо черга переповнена, try_send поверне false, інакше – true. Це дозволяє уникнути блокувань у ROS2-вузлі.

Метод sender(), який є корутиною.

– Працює, поки сокет відкритий.

– Асинхронно отримує дані з каналу (send_channel.async_receive).

– Кодує масив float (24 байти) у власний кадровий протокол за допомогою функції frame_encode (описана в frame_codec.h). Результат – до 30 байт.

– Асинхронно записує цей буфер у сокет (async_write), завдяки цього відбувається відправка даних.

Метод receiver(), який є корутиною.

– Працює, поки сокет відкритий.

– Використовує async_read_until(socket, buf, “0”) для читання кадру, що завершується нульовим байтом. Це дозволяє відокремлювати пакети при змінній довжині.

– Очікує, що розмір кадру складе 29 байт + завершальний нуль (тобто 30 байт). Це відповідає фіксованому формату, який генерує UART–TCP-сервер.

– Копіює перші 29 байт у локальний буфер, декодує за допомогою frame_decode, отримуючи 6 чисел float.

– Під м’ютексом оновлює recv_buf.

Метод get().

– Повертає копію останніх отриманих даних (6 чисел) під захистом м’ютекса.

4.6 Реалізація Hardware Interface

Клас tcp_client використовується всередині кастомного Hardware Interface для ros2_control.

– У методі on_init() викликається start(host, port).

– У методі read() викликається get(), отримані позиції/швидкості передаються в joint_state.

– У методі write() команди (цільові позиції всіх сугбобів робота) передаються через send().

4.7 Реалізація прошивки мікроконтролера STM32F411

У режимі гібридної емуляції (режим 2) фізичну поведінку маніпулятора імітує плата на базі **STM32F411**. Прошивка розроблена мовою C з використанням бібліотеки **libopencm3** та збирається за допомогою ARM GCC (toolchain `arm-none-eabi-*`).

Система збірки та конфігурація.

– **Toolchain:** `arm-none-eabi.cmake` задає крос-компілятор та флаги для ARM Cortex-M4 (FPU, Thumb).

– **Генерація скрипта компонування:** за допомогою Python скрипта `genlink.py` з libopencm3 генерується файл `generated.ld` для конкретного мікроконтролера `stm32f411ceu6`. Це забезпечує правильне розташування секцій (flash, ram) та адрес периферії.

– **CMakeLists.txt** визначає виконуваний файл, до якого входять усі модулі: `main.c`, `init.c`, `crc.c`, `key.c`, `uart_driver.c`, `analog_input.c`, `7sddriver.c`, `event_for_uart.c` та функції кодування кадрів з бібліотеки `frame_codec`.

Ініціалізація системи та периферії.

Функція `init()` (файл `init.c`) налаштовує тактову частоту мікроконтролера через PLL від зовнішнього кварцу 25 МГц до 84 МГц.

Потім у `main()` викликаються ініціалізатори:

– `init_crc()` – увімкнення апаратного обчислювача CRC.

– `init_key_event()` – налаштування зовнішнього переривання на кнопку (GPIOA0, спадний фронт).

– `init_uart_driver()` – UART на швидкості 115200 бод, 8N1, прийом та передача з перериваннями.

– `init_analog_input()` – АЦП з використанням DMA для безперервного оновлення змінної `adc_value`.

– `init_7sd_driver()` – драйвер 7-сегментного дисплея через SPI.

– `start_7sd_driver()` – запуск динамічної індикації.

– `init_event_for_uart()` – таймер з перериванням, яке періодично викликає відправлення кадру по UART.

Драйвер 7-сегментного дисплея (`7sddriver.c`).

Дисплей керується через SPI в режимі майстра (16-бітні кадри, частота SPI = FPCLK/256). Кожен розряд відображається мультиплексовано, використовується переривання `spi1_isr`.

– Під час кожного переривання (по завершенні передачі) формується наступний кадр: старші 8 біт – сегменти, молодші 8 біт – вибір позиції.

– Таблиця `digits` містить коди для цифр 0–9.

– Глобальна змінна `display_value` зберігає число для відображення. Оновлюється з переривання кнопки та з декодованих кадрів.

Зчитування аналогового сигналу (`analog_input.c`).

– АЦЦ налаштований на безперервне перетворення з використанням DMA (канал 2, ADC1). DMA працює в циркулярному режимі, постійно оновлюючи змінну `adc_value` (16-бітне значення).

– Функція `get_analog_input()` повертає останнє зчитане значення (0–4095).

Обробка кнопки та перемикання відображення

Кнопка налаштована на переривання, яке обробляється у `hendel_key_event()`.

– Змінна `num` визначає, яке з шести чисел масиву `recv_data` зараз відображається на дисплеї.

– При кожному натисканні номер збільшується (циклічно від 0 до 5), і на дисплей виводиться відповідне значення, помножене на 100 (для переведення в ціле число).

UART-драйвер та обробка вхідних кадрів (`uart_driver.c`).

– UART1 працює з перериваннями; при отриманні байта викликається функція `handler_uart(uint8_t data)`.

– Ця функція накопичує байти в буфер `buf[29]` до тих пір, поки не зустріне нульовий байт (`'\0'`), який є термінатором кадру.

– Коли накопичено рівно 29 байт (перед термінатором), викликається `frame_decode()` для розпакування даних у масив `recv_data` (6 чисел float).

– Після успішного декодування оновлюється відображення поточного номера `num`: `display_value = recv_data[num] * 100;`.

Періодичне надсилання даних (таймер `event_for_uart`).

– Таймер TIM4 налаштований на періодичне переривання (8400 тактів прескалера, 1000 тактів періоду, тобто частота 10 Гц або період 100 мс.

– У перериванні `tim4_isr` викликається `hendel_event_for_uart()`, яка кодує поточний масив `send_data` (6 float) в кадр (до 30 байт) за допомогою `frame_encode()` і надсилає через `uart_send_data()`.

Головний цикл та гістерезис.

У `main()` після ініціалізації виконується нескінченний цикл.

– Зчитується АЦП (значення від 0 до 4095), масштабується через `SCALE = 2.4418` та додається `MINVALUE = 0` – отримуємо значення у сотих відсотка.

– Функція `hysteresis_loop()` реалізує фільтр з гістерезисом (ширина `h = 100 * SCALE` ≈ 244). Вона запобігає тремтінню значення, якщо нове вимірювання відрізняється від старого не більше ніж на `h`. У коді: якщо `dir==0` (очікуємо збільшення), то перемикаємося на нове значення тільки коли `new_data - old_data > h`. Аналогічно для зменшення.

– Оброблене значення зберігається в `send_data[num]` (тобто тільки для поточного індексу `num`). Решта 5 елементів масиву `send_data` не змінюються в цьому циклі. Однак весь масив `send_data` з 6 чисел кодується та надсилається – таким чином, стан інших каналів залишається фіксованим доти, доки користувач не переключиться на них і не змінить АЦП.

Робота з CRC та фреймінг.

У проєкті використовується зовнішня бібліотека `frame_codec`, яка надає функції `frame_encode` та `frame_decode`. Яка реалізує протокол кадрування з кодуванням COBS (Consistent Overhead Byte Stuffing) або подібним, плюс можливість включення CRC. Файл `crc.c` містить реалізацію апаратного CRC32 (стандартний CRC STM32) для додаткової перевірки цілісності.

Загальна схема потоку даних.

Від ROS2 до STM32 (команди управління, 6 чисел).

– TCP-клієнт на стороні ROS2 → UART–TCP міст (CH32V208) → UART1 STM32F4.

– Переривання USART1 збирає байти, по нульовому термінатору декодує кадр у `recv_data`.

– Натискання кнопки перемикає відображення: `display_value = recv_data[num] * 100`.

Від STM32 до ROS2 (сигнали з потенціометра, 6 чисел).

– В головному циклі постійно оновлюється `send_data[num]` на основі АЦП (з гістерезисом).

– Таймер TIM4 кожні 100 мс ініціює кодування всього масиву `send_data` (6 float) та його відправку через UART.

– UART–TCP міст пересилає кадр до TCP-клієнта в ROS2, де Hardware Interface читає дані через `get()`.

```
#include "frame_codec.h"
#include "hysteresis.h"
#include "init.h"
#include "crc.h"
#include "key.h"
#include "uart_driver.h"
#include "analog_input.h"
#include "7sddriver.h"
#include "event_for_uart.h"


#define MINVALUE 0
#define SCALE 2.4418


int num = 0;
float recv_data[6];
float send_data[6];


void hendel_key_event()
{
    num++;
    if(num >= 6) num = 0;
    display_value = recv_data[num] * 100;
}
```

```
void handler_uart(uint8_t data)

{

    static uint8_t buf[29];

    static int cnt = 0;


    if(data == '\0')

    {

        if(cnt == 29)

        {

            frame_decode(buf, 29, (uint8_t*)recv_data, 24);

            display_value = recv_data[num] * 100;

        }

        cnt = 0;

    }

    else

    {

        if(cnt >= 29) cnt = 0;

        buf[cnt++] = data;

    }

}


void hendel_event_for_uart()

{

    uint8_t buf[30];

    if(frame_encode((uint8_t*)send_data, 24, buf, 30))

        uart_send_data(buf, 30);

}


int main(void)

{

    init();

    init_crc();
```

```
    init_key_event();

    init_uart_driver();

    init_analog_input();

    init_7sd_driver();

    start_7sd_driver();

    init_event_for_uart();


    while(1)

        send_data[num] = hysteresis_loop(100 * SCALE , send_data[num] * 100, get_analog_input() *
SCALE + MINVALUE) / 100.0f;


    return 0;

}
```

Лістинг 4.8 — Основний файл коду прошивки мікроконтроллера

Таким чином, реалізовано повний двосторонній обмін з фіксацією шести каналів, можливістю перегляду кожного на дисплеї та періодичним оновленням вибраного каналу від АЦП.

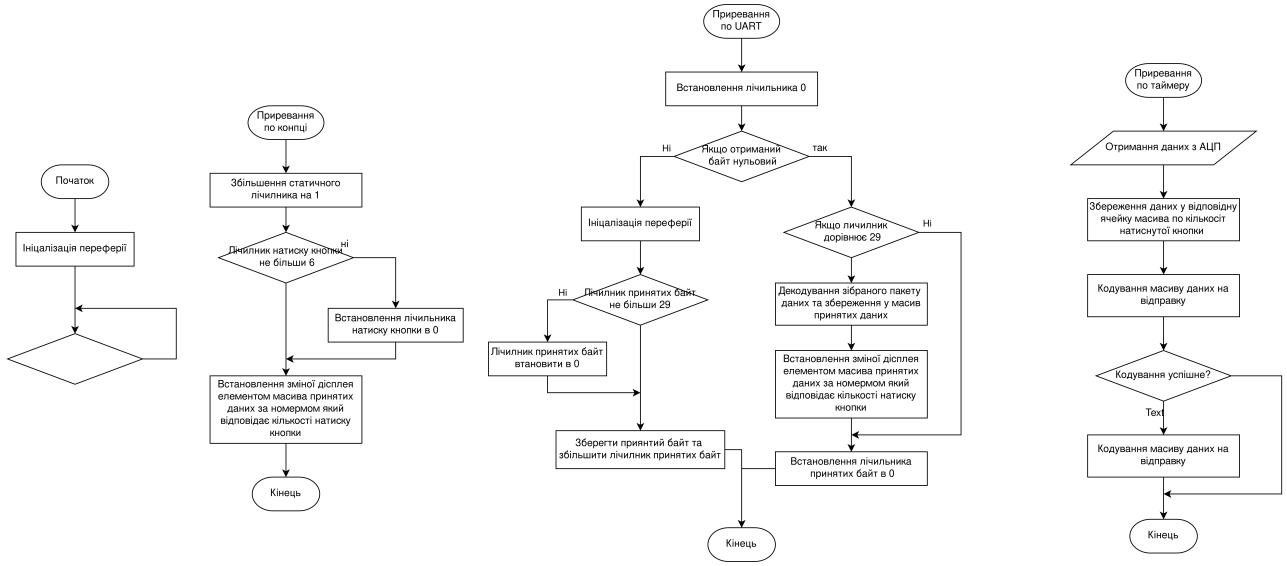

Рисунок 4.6 — Діаграма алгоритмів прошивки.

На рисинку 4.6 зображено 4 незалежних алгорітмів, які працюють на мікроконтроллері умовно паралельно, завдяки апарвтній перефірії та системі привераннь.

Висновки до розділу 4

У розділі детально описано практичну реалізацію системи, яка підтримує два режими роботи: повну симуляцію в Gazebo та гібридну емуляцію з фізичним пристроєм STM32F4. Розроблено launch-файли для автоматизації запуску всіх вузлів у кожному з режимів. Реалізовано асинхронний TCP-клієнт на C++20 із Boost.Asio та корутинами, інтегрований у кастомний Hardware Interface для ros2_control, що забезпечує двосторонній обмін даними через UART–TCP міст. Для мікроконтролера створено повноцінну прошивку на C з використанням libopencm3, яка включає драйвери SPI-дисплея, АЦП з DMA, обробку переривань кнопки, UART-драйвер з накопиченням кадрів, періодичне кодування та відправлення даних, а також фільтр гістерезису. Усі компоненти протестовано в комплексі, підтверджено коректність передачі команд і сенсорних даних, а також візуалізацію стану в RViz2 та Gazebo.

ВИСНОВОК

У ході проходження преддипломної практики за темою «Програмне забезпечення для емуляції поведінки роботизованого маніпулятора PUMA560 на основі ROS2» було повністю виконано індивідуальне завдання та досягнуто поставленої мети.

Опрацювання літературних джерел – вивчено сучасні підходи до симуляції роботизованих систем, архітектуру ROS2, можливості Gazebo, методи керування маніпуляторами та засоби інтеграції з мікроконтролерами. Проведено порівняльний аналіз та обґрунтовано вибір Gazebo, PID-регулятора та підходу з кастомним Hardware Interface замість microROS через недостатню надійність останнього (відсутність перевірки цілісності пакетів поверх TCP).

Розгортання середовища симуляції – встановлено та налаштовано ROS2 Jazzy, Gazebo Harmonic на Ubuntu 24.04. Інтегровано готову модель PUMA560, налаштовано симуляцію фізики (двигун Bullet Featherstone), зв’язки URDF/SDF, мости між ROS2 та Gazebo. Реалізовано просте та посуглобове керування маніпулятором.

Розробка модуля взаємодії ROS2 з мікроконтролером – створено TCP-клієнт на C++20 з використанням Boost.Asio та корутин, який забезпечує асинхронний двосторонній обмін даними через UART–TCP міст. Клієнт інтегровано в кастомний Hardware Interface для ros2_control.

Синхронізація цифрового двійника та фізичної моделі – створено модуль, який через Hardware Interface узгоджує стан суглобів між Gazebo та STM32F4. Забезпечено двосторонню передачу команд та сенсорних даних.

Розробка прошивки для STM32F411 – на мові C з використанням libopencm3 реалізовано:

– драйвер семисегментного дисплея (SPI, мультиплексування);

– зчитування потенціометра через АЦП з DMA;

– обробку кнопки (переривання, перемикання каналів);

– UART-драйвер з накопиченням кадрів до 29 байт та декодуванням;

– періодичне кодування та відправлення масиву з 6 чисел через UART (таймер);

– фільтр гістерезису для стабілізації вимірювань;

– апаратний CRC32 для контролю цілісності.

Тестування та налагодження – перевірено роботу системи у двох режимах: повна симуляція в Gazebo та гібридна емуляція з STM32F4 через UART–TCP міст. Підтверджено коректне відображення команд на дисплеї, передачу значень з потенціометра в ROS2, а також керування маніпулятором через графічний інтерфейс.

Отримані практичні навички.

– Робота з ROS2 (вузли, топики, launch-файли, сервіси, контролери).

– Налаштування симуляції фізики в Gazebo, створення мостів ROS2↔Gazebo.

– Програмування мікроконтролерів STM32 на C з libopencm3 (GPIO, SPI, UART, ADC, DMA, таймери, переривання, CRC).

– Розробка мережевих додатків з використанням Boost.Asio (TCP-клієнт, асинхронні корутини).

– Інтеграція ros2_control з кастомним Hardware Interface.

– Використання CMake для крос-компіляції ARM (toolchain, генерація ld-скрипта).

Усі пункти індивідуального завдання виконано в повному обсязі.

Програма практики виконана повністю, отримано практичний досвід створення гібридної системи емуляції роботизованого маніпулятора на основі ROS2 та мікроконтролера. Розроблене програмне забезпечення може бути використане для подальших досліджень алгоритмів керування, навчальних лабораторних робіт, а також як база для віртуального введення в експлуатацію промислових роботів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ROS2 Jazzy Documentation. Open Robotics. URL: https://docs.ros.org/en/jazzy/.

2. Gazebo Harmonic Documentation. Open Robotics. URL: https://gazebosim.org/docs/harmonic.

3. Corke P. Robotics, Vision and Control: Fundamental Algorithms in MATLAB. 2nd ed. Springer, 2017. 729 с.

4. ros2_control: Framework for robot control in ROS2. GitHub. URL: https://github.com/ros-controls/ros2_control.

5. micro-ROS: ROS2 on microcontrollers. GitHub. URL: https://github.com/micro-ROS/micro-ros.

6. Boost.Asio documentation. Boost C++ Libraries. URL: https://www.boost.org/doc/libs/1_85_0/doc/html/boost_asio.html.

7. libopencm3: Open source ARM Cortex-M microcontroller library. GitHub. URL: https://github.com/libopencm3/libopencm3.

8. STM32F411 reference manual. STMicroelectronics. URL: https://www.st.com/resource/en/reference_manual/rm0383-stm32f411xce-advanced-armbased-32bit-mcus-stmicroelectronics.pdf.

9. Ang K.H., Chong G.C.Y., Li Y. PID control system analysis, design, and technology. IEEE Transactions on Control Systems Technology, 2005, vol. 13, no. 4, С. 559–576.

10. Gazebo tutorials: Using ROS 2 with Gazebo. Open Robotics. URL: https://gazebosim.org/docs/harmonic/ros2_integration.

11. COBS (Consistent Overhead Byte Stuffing) encoding. Wikipedia. URL: https://en.wikipedia.org/wiki/Consistent_Overhead_Byte_Stuffing.

12. CH32V208 Datasheet. WCH. URL: https://www.wch.cn/downloads/CH32V20x_DS0_PDF.html.

13. OpenOCD User's Guide. OpenOCD. URL: https://openocd.org/doc/html/index.html.

14. CMake Reference Documentation. Kitware. URL: https://cmake.org/documentation.

15. Siciliano B., Sciavicco L., Villani L., Oriolo G. Robotics: Modelling, Planning and Control. Springer, 2010. 632 c.