

**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 на основі нечіткої логіки для керування
рухами робота-маніпулятора в умовах невизначеності.»**

Виконав:

Студент IV курсу, групи ТВ-22

Турніцький Владислав Анатолійович

(прізвище, ім'я, по батькові)

(підпис)

Керівник:

Професор, д.т.н., професор Барабаш О.В

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент:

Професор, д.т.н., професор Отрох С.І.

(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»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Олександр КОВАЛЬ

«___» _____ 2026р.

ЗАВДАННЯ

на дипломну роботу студенту

_____ Турніцького Владислава Анатолійовича

(прізвище, ім'я, по батькові)

1. Тема роботи: Програмне забезпечення на основі нечіткої логіки для керування
рухами робота-маніпулятора в умовах невизначеності

керівник роботи Барабаш Олег Володимирович д.т.н професор

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «28» травня 2026 р. № 1992-с

2. Строк подання студентом роботи «10» червня 2026 р.

3. Вихідні дані до роботи: мова програмування Python, середовище розробки Visual
Studio Code, середовище симуляції Gazebo, фреймворк ROS 2

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно
розробити): провести аналіз існуючих методів керування просторовим рухом,
спроектувати математичну модель нечіткого контролера, синтезувати базу правил
та обрати алгоритм виведення та розробити архітектуру обчислювального ROS 2
вузла.

5. Перелік ілюстративного матеріалу: кінематична схема маніпулятора Puma 560,
графіки належності вхідних та вихідних лінгвістичних змінних, база правил,
тривимірна нелінійна поверхня, графік перехідного процесу позиціонування ланки.

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	31.10.2025	
2	Дослідження предметної області	01.11.2025 – 31.11.2025	
3	Дослідження існуючих рішень	01.12.2025 – 31.12.2025	
4	Постановка вимог до проектування 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

(підпис)

Владислав ТУРНІЦЬКИЙ

(ім'я, ПРИЗВИЩЕ)

Керівник роботи

(підпис)

Олег БАРАБАШ

(ім'я, ПРИЗВИЩЕ)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 63 сторінок, 13 рисунків, 2 таблиці, 2 додатки та 12 посилань.

Метою роботи є розробка програмного забезпечення на основі нечіткої логіки для керування рухами робота-маніпулятора в умовах невизначеності.

Для досягнення цієї мети було виконано наступні етапи:

- проведено аналіз існуючих методів керування просторовим рухом роботів-маніпуляторів;
- обґрунтовано вибір технологічного стеку інструментів розробки;
- створено архітектуру обчислювального ROS 2 вузла керування та її програмно реалізовано;
- проведено комплексне тестування розробленого програмного забезпечення, побудовано аналітичні графіки перехідних процесів та оцінено показники якості регулювання;
- створено інструкцію з розгортання та керування для експлуатації розробленого програмного комплексу.

Практичне значення одержаних результатів полягає в розробці робастичного програмного модуля інтелектуального керування рухами робота-маніпулятора Puma 560 в умовах параметричної невизначеності без використання складної аналітичної математичної моделі об'єкта.

Ключові слова: нечітка логіка, робот-маніпулятор, Puma 560, система керування просторовим рухом, стан параметричної невизначеності, фазифікації, база правил, дефазифікація, алгоритм Мамдані.

ABSTRACT

Structure and scope of the thesis. The work contains 63 pages, 13 figures, 2 tables, 2 appendices and 12 references.

The purpose of the work is to develop software based on fuzzy logic for controlling the movements of a robotic manipulator under conditions of uncertainty.

To achieve this goal, the following steps were taken:

- an analysis of existing methods for controlling the spatial motion of robotic manipulators was conducted;
- the selection of the technology stack of development tools was justified;
- the architecture of the ROS 2 computational control node was created and implemented in software;
- comprehensive testing of the developed software was conducted, analytical graphs of transient processes were constructed, and control quality indicators were evaluated;
- deployment and management instructions were created for the operation of the developed software complex.

The practical value of the results obtained lies in the development of a robust software module for the intelligent control of the Puma 560 robotic manipulator under conditions of parametric uncertainty, without the use of a complex analytical mathematical model of the system.

Keywords: fuzzy logic, robotic manipulator, Puma 560, spatial motion control system, parametric uncertainty, fuzzification, rule base, defuzzification, Mamdani algorithm.

ЗМІСТ

	стор.
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.....	8
ВСТУП.....	9
1 ПОСТАНОВКА ЗАДАЧІ ТА ТЕХНІЧНІ ВИМОГИ ДО СИСТЕМИ КЕРУВАННЯ МАНІПУЛЯТОРОМ.....	11
1.1 Опис об'єкта керування та предметної області.	11
1.2 Проблема невизначеності у просторовому позиціюванні.....	15
1.3 Формування вимог до програмної системи	16
1.4 Постановка задачі дослідження та проектування.	17
Висновок до розділу 1	19
2 ОГЛЯД СУЧАСНИХ МЕТОДІВ КЕРУВАННЯ ТА ОБҐРУНТУВАННЯ ІНТЕЛЕКТУАЛЬНОГО ПІДХОДУ	20
2.1 Огляд існуючих методів керування просторовим рухом.....	20
2.1.1 Метод керування на основі ПІД-Контролера	21
2.1.2 Метод обчислюваного обертального моменту	22
2.1.3 Метод керування зі застосуванням нейронних мереж	24
2.2 Використання нечіткої логіки для керування в умовах невизначеності	26
2.3 Порівняльний аналіз методів керування та обґрунтування вибору	28
Висновок до розділу 2	30
3 ОБҐРУНТУВАННЯ ВИБОРУ ТА ОГЛЯД ПРОГРАМНО-ТЕХНІЧНОГО ІНСТРУМЕНТАРІЮ ДЛЯ СТВОРЕННЯ СИСТЕМИ КЕРУВАННЯ.....	31
3.1 Мова програмування Python як засіб розробки.....	31
3.2 Екосистема бібліотек Python для розробки регулятора.....	32
3.3 Інтегроване середовище розробки Visual Studio Code.....	34
3.4 Екосистема ROS 2 для інтеграції компонентів керування	35
3.5 Використання симулятора Gazebo для верифікації руху маніпулятора.....	36
Висновок до розділу 3	36
4 ПРОЕКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ КЕРУВАННЯ МАНІПУЛЯТОРОМ.....	38
4.1 Проектування структури розробленого ROS-модуля.....	38

4.2 Розробка ОО-архітектури програмного модуля.....	40
4.3 Опис алгоритму роботи головного циклу керування.....	43
4.4 Проєктування динамічної взаємодії компонентів системи	44
Висновок до розділу 4.....	47
5 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА МОДЕЛЮВАННЯ РОБОТИ МАНІПУЛЯТОРА PUMA 560	49
5.1 Синтез логічної моделі керування	49
5.2 Аналіз результатів моделювання нечіткого регулятора	55
5.3 Порядок розгортання та керівництво користувача програмним комплексом.....	58
Висновок до розділу 5.....	59
ВИСНОВКИ.....	61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	63
ДОДАТОК А.....	64
ДОДАТОК Б.....	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

EOM	Електронна обчислювальна машина
LTS	Long Term Support
ARM	Advanced RISC Machine
ПІД	Пропорційно-інтегрально-диференціальний
ПК	Персональний комп'ютер

ВСТУП

В сучасному світі роботи-маніпулятори набувають дедалі більшої популярності. Це зумовлено початком активного використання їх у різних областях: від навантажувальних робіт до спецтехніки для лісозаготівлі. Також важливим аспектом актуальності є автоматизація промисловості: такі системи надають можливість автоматизувати складні рухи для взаємодії з важкими вантажами або в небезпечних умовах. Окрім того, використання таких роботів прискорює процес виробництва та збільшує продуктивність.

Через таку популярність навички створення програмного забезпечення для керування та обслуговування робота маніпулятора стали дуже затребуваними. Однією з найскладніших проблем при роботі з маніпулятором є задача керування ним. Труднощі виникають через те, що сам по собі маніпулятор знаходиться у стані невизначеності.

Стан невизначеності – це комплекс нелінійних фізичних факторів, що діють на маніпулятор під час роботи. До таких факторів належать постійна зміна моменту інерції, вплив гравітаційних сил та тертя в суглобах. Через нелінійність природи цих явищ, звичайні лінійні регулятори не в змозі правильно керувати рухами робота. Тому доцільним є використання інтелектуального методу, зокрема нечіткої логіки.

Нечітка логіка – це розділ математики, який відмовляється від бінарного поділу на користь ступеня належності. Також вона використовує лінгвістичні змінні, що робить її механізми дуже схожими до людського мислення. Завдяки цьому методи з використанням нечіткої логіки є дуже ефективними при вирішенні проблеми стану невизначеності.

Метою цієї роботи є створення програмного модуля на основі нечіткої логіки для підвищення точності та якості управління роботом маніпулятором Puma 560 в умовах невизначеності.

Об'єктом дослідження є процеси керування рухом багатоланкових роботів-маніпуляторів в умовах параметричної та неструктурованої невизначеності. А предмет дослідження цієї роботи є методи та алгоритми нечіткої логіки для синтезу робастних систем керування електроприводами робота

У результаті виконання роботи очікується отримати повністю працездатний програмний вузол для середовища ROS 2, який використовує методи нечіткої логіки та усуває проблеми класичних регуляторів. Розроблений модуль може бути використаним і адаптованим для керування реальним фізичним маніпулятором Puma 560.

1 ПОСТАНОВКА ЗАДАЧІ ТА ТЕХНІЧНІ ВИМОГИ ДО СИСТЕМИ КЕРУВАННЯ МАНІПУЛЯТОРОМ

На початку виконання будь-якої наукової роботи, пов'язаної з розробкою програмного забезпечення, потрібно пройти декілька етапів підготовки. Першим завданням є технічний огляд кіберфізичної системи, для якої відбувається розробка. Цей етап потрібен для кращого розуміння апарату, з яким доведеться працювати протягом усього досягнення поставленої цілі.

Після повного розуміння основної робочої системи настає крок детального аналізу предметної області та існуючої проблематики. На цьому етапі досліджується, чим саме зумовлена поставлена в умові роботи проблема та чому її подолання є критично важливим для роботи усієї системи.

Наступним кроком у підготовці до написання програмного продукту є формування технічних вимог до майбутньої розробленої системи. Їх формування скоротить загальний час створення програми та зменшить кількість допущених помилок під час розробки.

Кінцевим етапом підготовки до розробки є постановка фінальної задачі та визначення критеріїв якості. Цей етап потрібен для чіткого розуміння очікуваного результату та формування об'єктивних метрик, за якими буде оцінюватися успішність розробленого програмного рішення.

1.1 Опис об'єкта керування та предметної області.

Об'єктом дослідження обрано модель промислового робота-маніпулятора Puma560. Цей робот є антропоморфним, тобто його будова подібна до будови руки людини. Він складається з елементів двох типів: ланок (links), що є аналогом людських кісток й представлені у вигляді металевих труб та зчленувань (joints), які є аналогом суглобів й вони означають електричні мотори, які крутять ланки.

Маніпулятор Puma560 складається з 6 зчленувань, що забезпечує 6 ступенів свободи (DOF). Це надає можливість виведення робочого органу в будь-яку точку

робочої зони з довільною орієнтацією. Ці зчленування прийнято ділити на дві підсистеми: позиціонування та орієнтації. Кінематична схема маніпулятора з позначенням розташування осей обертання та відповідних ланок наведена на Рисунку 1.1.

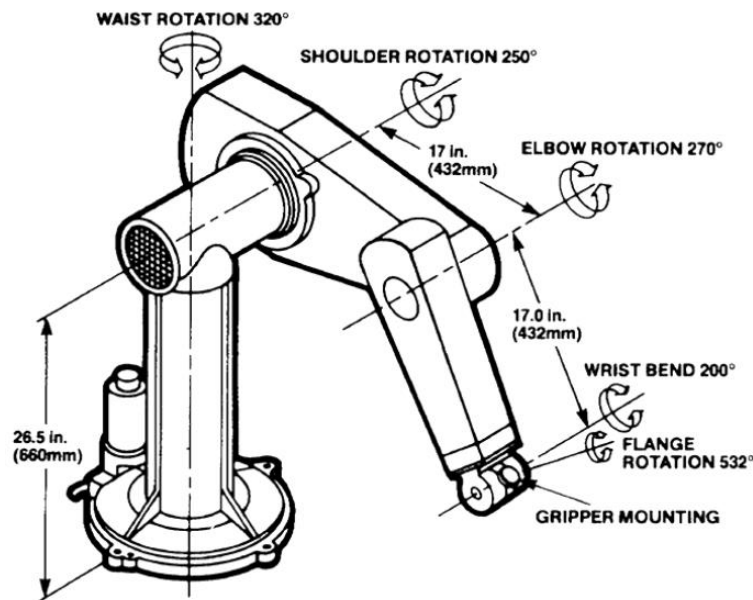


Рисунок 1.1 – Кінематична схема маніпулятора Puma560 [9]

Підсистема позиціонування складається з трьох різних зчленувань, що відповідають за переміщення центру кисті робота у задані координати простору (x, y, z). Перше зчленування називається талія (waist), вона здійснює базове обертання колони, тобто обертається навколо вертикальної осі Z. Другим зчленуванням є «плече» (shoulder), яке відповідає за обертання навколо горизонтальної осі, чим забезпечує підйом та опускання плеча. Останнє зчленування називається «лікоть» (elbow). Воно обертається навколо осі, що є паралельною осі плеча, та визначає радіус досяжності маніпулятора.

Підсистема орієнтації складається з трьох інших зчленувань осі, яких перетинаються в одній точці, чим утворюють сферичну кисть. Задача цієї підсистеми створення певного нахилу в просторі інструменту в кисті. Всі три зчленування цієї структури носять назву «кисть» (wrist), але оскільки кожен з них відповідає за свій кут Ейлера то до назви додається назва кута. Перше зчленування

«кисть» Roll здійснює обертання кисті навколо поздовжньої осі передпліччя. Цей рух дозволяє змінити площину дії наступного суглоба. Зчленування, що відповідає за кут з назвою Pitch, обертає навколо поперечної осі, завдяки чому відбуваються нахили вниз та вгору. «Кисть» Yaw обертається навколо вертикальної осі, чим крутить сам диск, до якого кріпиться інструмент.

Для математичного моделювання кінематики руху роботу Puma560 буде використано таблицю параметрів Денавіта-Хартенберга. Цей метод вважається стандартним та універсальним оскільки надає можливість точно обчислити положення кожної ланки робота відносно попередньої застосовуючи лише 4 параметра. Перелік даних чотирьох параметрів:

1. Довжина ланки (a_i): Відстань між осями z_{i-1} та z_i вздовж осі x_i .
2. Кут скручування ланки (α_i): Кут повороту осі z_i відносно z_{i-1} навколо осі x_i .
3. Зміщення ланки (d_i): Відстань між осями x_{i-1} та x_i вздовж осі z_{i-1} .
4. Кут повороту (θ_i): Кут повороту осі x_i відносно x_{i-1} навколо осі z_{i-1} .

Для обертальних суглобів це змінна величина.

Відповідно до методу Денавіта-Хартенберга, узагальнена матриця перетворення однорідних координат A_i , що описує перехід від системи координат $(i-1)$ -ї ланки до i -ї ланки, має наступний вигляд на формулі 1.1.

$$A_i = \begin{pmatrix} \cos \theta_i & -\sin \theta_i \cos \alpha_i & \sin \theta_i \sin \alpha_i & \cos \theta_i \\ \sin \theta_i & \cos \theta_i \cos \alpha_i & -\cos \theta_i \sin \alpha_i & \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad (1.1)$$

де θ_i – кут повороту відповідної ланки,

α_i – кут скручування відповідної ланки,

d_i – зміщення відповідної ланки.

Загальне положення та орієнтація робочого органу маніпулятора відносно базової системи координат визначається шляхом послідовного множення матриць перетворення для всіх шести ланок та описується за формулою 1.2.

$$T_{total} = A_1 * A_2 * A_3 * A_4 * A_5 * A_6, \quad (1.2)$$

де T_{total} – загальна матриця перетворення однорідних координат.

Числові значення параметрів Денавіта-Хартенберга для робота Puma560 наведені в Таблиці 1.1. Підставивши їх у рівняння вище буде отримана кінетична модель керування маніпулятора.

Таблиця 2.1. – Параметри Денавіта-Хартенберга для робота Puma560.

Ланка (i)	Кут повороту (θ), рад	Зміщення (d), см	Довжина (a), см	Кут скручування (α), рад
1	A_1	0	0	$\frac{\pi}{2}$
2	A_2	0	0.4318	0
3	A_3	0.15	0.0203	$-\frac{\pi}{2}$
4	A_4	0.4318	0	$\frac{\pi}{2}$
5	A_5	0	0	$-\frac{\pi}{2}$
6	A_6	0	0	0

Для повноцінного керування роботом потрібно вирішити дві кінематичні задачі: пряму та обернену. Пряма задача кінематики полягає у визначенні позиції та орієнтації в просторі робочого органу на основі відомих значень кутів повороту суглобів та геометричних параметрів з таблиці Денавіта-Хартенберга. Формула розв’язку прямої задачі кінематики наведена вище для значення T_{total} . Головною

характеристикою цієї задачі є те, що вона завжди має рішення і це рішення завжди єдине.

Обернена задача кінематики, навпаки, полягає у визначення необхідних значень кутів повороту суглобів для того, щоб робочий інструмент опинився в заданій точці в просторі із необхідною орієнтацією. У загальному вигляді математична модель оберненої задачі описується нелінійним векторним рівнянням, наведеним у формулі 1.3.

$$Q = f^{-1}(X, Y, Z, Roll, Pitch, Yaw), \quad (1.3)$$

де X, Y, Z – цільові координати,

$Roll, Pitch, Yaw$ – цільові кути орієнтації інструменту.

Вирішення цієї задачі є набагато складнішим, ніж у випадку завдання прямої кінематики. Така складність викликана тим, що рівняння містять тригонометричні функції, що унеможлиблює просте алгебраїчне вираження змінних. Також може існувати декілька наборів значень, що задовольняють умови задачі. Окрім цього, рішень може не існувати, тобто маніпулятор не може опинитися в заданій позиції. Для маніпулятора з 6 ступенями свободи, такого як Puma 560, не існує єдиної аналітичної формули для розрахунку цих кутів через складність геометричних зв'язків. Тому для розв'язання цієї задачі застосовуються чисельні ітераційні методи, такі як метод Ньютона-Рафсона або Левенберга-Марквардта.

1.2 Проблема невизначеності у просторовому позиціюванні

Для коректного тлумачення терміна «невизначеність» скористаємося статтею Жан-Жак Слотін «The Robust Control of Robot Manipulators» [3]. Автор статті описує невизначеність, як різницю між значеннями математично створеної моделі та

значеннями реального об'єкта. Також невизначеність класифікується на два типи: параметрична та неструктурована.

Параметрична невизначеність позначає неточності у значеннях змінних, що ми використовуємо при обчисленнях у формулах. Такими значеннями є неточність інерції ланок маніпулятора, геометричних розмірів або маси корисного навантаження. Оскільки точні значення невідомі, для обчислень використовуються межі в які реальний параметр точно входить і які є наскільки можливо малими.

Неструктурована невизначеність позначає фізичні процеси, якими було знехтувано при створенні фінального рівняння керування, через навмисне нехтування ними чи складність в підрахунку. Прикладами таких явищ є важкість моделювання тертя в суглобах, високочастотні резонансні коливання ланок маніпулятора та часові затримки в передачі сигналу.

Наслідки невизначеності є очевидними. Це виникнення похибки відстеження траєкторії руху маніпулятора, неправильної компенсації нелінійності рівнянь, створення нестійких систем та неможливість проаналізувати потенційну траєкторію руху маніпулятора.

1.3 Формування вимог до програмної системи

Перед початком виконання завдання на цю роботу та розроблення будь-яких програмних модулів необхідно сформулювати чіткий перелік вимог. Вимоги прийнято ділити на дві групи: функціональні (що система має робити) та нефункціональні (як система має функціонувати). Було виділено такі функціональні вимоги:

- Збір телеметрії. Програмний вузол повинен отримувати значення телеметрії про поточну позицію та швидкість всіх суглобів зчитуючи канал даних.
- Коректне отримання цільового значення. Програмний продукт повинен отримувати коректні цільові координати всіх суглобів за допомогою спеціального іменованого топіку.

- Використання нечіткого регулятора. Програмний вузол повинен використовувати математичний апарат нечіткої логіки при обчисленні керуючих впливів для маніпулятора.

- Генерація команд. Програма повинна видавати відповідні команди для маніпулятору у власний створений іменний канал даних.

Список нефункціональних вимог наступний:

- Частота 100 Гц. Програмний вузол повинен працювати з частотою 100 Герц, тобто щосекунди цикл обчислення команди має виконуватися 100 разів.

- Точність. Статична помилка при русі робота у кінцевій точці має наближатися до нуля.

- Стабільність. Контролер повинен повністю усувати явище автоколивань та мікровібрацій при утриманні цільової позиції.

- Якість перехідного процесу. Система повинна забезпечувати плавний вихід на задану координату з мінімальним або нульовим перерегулюванням.

- Кросплатформеність. Створене програмне забезпечення повинно мати змогу функціонувати в середовищі ОС Linux Ubuntu із дистрибутивом ROS 2 та підтримувати однакові інтерфейси взаємодії як для віртуальної моделі, так і для реального маніпулятора.

1.4 Постановка задачі дослідження та проєктування.

Підбиваючи підсумки, об'єктом дослідження цієї дипломної роботи є робот-маніпулятор Puma 560 із 6 ступенями свободи. На початку виконання роботи було надано його математичну модель та налаштоване середовище Gazebo, яке імітує стан невизначеності.

Таким чином, метою цієї дипломної роботи стає розробка програмного забезпечення на основі нечіткої логіки для керування наданим роботом-маніпулятором Puma 560 для подолання стану параметричної невизначеності.

Критерії ефективності розробленої системи узгоджуються з функціональними та нефункціональними вимогами. Зокрема, до них належать: точність пересування маніпулятора, відсутність коливань та мінімізація часу досягнення цільової позиції.

Розробка програмної системи для керування промисловим роботом-маніпулятором є доволі комплексним та складним процесом. Тому для пришвидшення та оптимізації створення необхідного програмного забезпечення було сформовано список задач на роботу. Перелік цих завдань наступний:

1. Розробка математичної моделі нечіткого контролера. Ця задача включає визначення вхідних та вихідних змінних регулятора. Також створення експертних лінгвістичних термінів та функцій належності з універсальними множинами. Окрім того на цьому етапі обирається метод нечіткого виведення згідно з яким будуть відбуватися процеси фазифікації та дефазифікації.

2. Синтез бази експертних правил. В межах цього завдання створюється база правил з використанням створених раніше лінгвістичних термінів. За допомогою цієї бази правил на основі вхідних значень буде обчислюватися вихідне у вигляді лінгвістичного терміну.

3. Розробка програмного модуля нечіткого контролера. Цей модуль буде реалізований мовою програмування Python з використанням бібліотеки scikit-fuzzy.

4. Реалізація вузла нечіткого керування для середовища ROS 2. Для виконання цієї задачі потрібно буде створити програмний скрипт підписавши вузол, який здійснюватиме підписку на топіки з телеметрією та цільовими кутами, а також публікацію керуючих команд у власний канал даних. Нова позиція обчислюється з використанням створеного до цього нечіткого контролера.

5. Інтегрування з моделлю робота та тюнінг системи. Розроблені до цього програмні модулі імпортуються до системи маніпулятора. Після збірки модуля він тестується та система докручується для точної зупинки та усунення явища перерегулювання.

Висновок до розділу 1

У розділі 1 було проведено попередню підготовку до практичного виконання дипломної роботи. Здійснено комплексний аналіз предметної області та окреслено ключові аспекти цієї роботи. Також в цьому розділі було сформовано вимоги до програмного забезпечення та постановлені задачі дослідження.

В першому пункті було проаналізовано об'єкт керування, яким є робот-маніпулятор, як складну кіберфізичну систему. Було наведено базовий опис наданого робота Puma 560. Досліджено структуру всіх його шести ланок, розглянуто параметри Денавіта-Хартенберга та використання узагальненої матриці для розв'язання задачі прямої кінематики.

Під час подальшого аналізу було детально розглянуто явище параметричної невизначеності в умовах якої має працювати маніпулятор. Окрім цього, було проаналізовано, які саме проблеми виникають під час просторового керування роботизованими системами через цей стан.

Важливим етапом роботи стало формування функціональних та нефункціональних вимог. Отримані вимоги стануть основою критеріїв ефективності та повноти створення програмного модуля.

На завершення розділу були визначені об'єкт та мета дослідження, а також сформовано перелік задачі до цієї роботи. Завдяки отриманим результатам можна краще оптимізувати план виконання всієї роботи, та окрім того, бути впевненим, що в подальшому виконання всіх задач дослідження буде повністю реалізовано.

2 ОГЛЯД СУЧАСНИХ МЕТОДІВ КЕРУВАННЯ ТА ОБҐРУНТУВАННЯ ІНТЕЛЕКТУАЛЬНОГО ПІДХОДУ

За десятиліття розвитку робототехніки було створено широкий спектр методів керування роботами-маніпуляторами, кожен зі своїми перевагами та недоліками. Вибір оптимального для поточних умов методу є критично важливим, адже робот-маніпулятор є складною нелінійною динамічною системою зі змінними параметрами, й потребує стійкості, точності та надійної роботи в реальному часі. Існуючі в теорії керування маніпуляторами методи можна розділити на три основні групи: класичні лінійні методи, методи на основі моделі та адаптивні й інтелектуальні методи.

2.1 Огляд існуючих методів керування просторовим рухом

До класичних лінійних методів належать PID регулятори та їх модифікації. Вони ефективно вирішують прості задачі через низьку обчислювальну складність та простоту налаштувань. Однак ця особливість робить їх неефективним вибором для задач, де динаміка руху маніпулятора є суттєво нелінійною або піддається значним зовнішнім збуренням.

Методи керування на основі моделі використовують математичний опис динаміки маніпулятора для прогнозування поведінки системи та обчислення керуючих впливів на неї. До цих способів належать метод обчислюваного моменту та метод лінеаризації зворотним зв'язком. Вони здатні забезпечити високу точність для складних нелінійних завдань, проте побудова абсолютно точної математичної моделі та її обчислення в реальному часі є ресурсомісткими задачами, що не завжди можливо реалізувати на практиці.

Адаптивні та інтелектуальні методи, до них належать адаптивне та робастне керування. Яскравими представниками цієї групи є нейронні мережі та нечітка логіка. Ці методи застосовуються в умовах невизначеності, коли точна модель об'єкта невідома або змінюється у часі.

2.1.1 Метод керування на основі ПІД-Контролера

Одним з перших, базових та найпоширеніших методів контролювання робіт маніпуляторів є правило під назвою PID (Proportional, Integral, Derivative) [1]. Базовий принцип роботи цього методу полягає в обчисленні значення різниці між цільовим значенням та поточним, що називається помилкою та спробам звести його до мінімуму. Цей контролер є закритим циклом що передає та обчислює значення по колу. Для зменшення помилки використовують формулу 2.1 яка складається з трьох складових (коефіцієнтів пропорційної, інтегральної та диференціальної помилки):

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}, \quad (2.1)$$

де $u(t)$ – керуючий вплив (вихідний сигнал регулятора) у момент часу,

K_p – коефіцієнти підсилення пропорційної складової,

$e(t)$ – похибка в момент часу,

K_i – коефіцієнти підсилення інтегральної складової,

$\int_0^t e(\tau) d\tau$ – інтеграл похибки (накопичена помилка),

K_d – коефіцієнти підсилення диференційної складової,

$\frac{de(t)}{dt}$ – похідна похибки (швидкість зміни помилки).

Пропорційний коефіцієнт: Це значення пропорційне до значення справжньої помилки. В залежності від нього регулюється швидкість роботи контролера. Якщо значення мале – робот зміщується повільно, а якщо високе то нестабільно швидко.

Інтегральний коефіцієнт: Це значення похибки є сумою усіх попередніх помилок. Її призначення усунення постійних невеликих відхилень, які можуть бути викликані тертям або зміщенням. Великі значення інтегральної похибки ймовірно є причиною коливань маніпулятора.

Диференційний коефіцієнт: Це значення є односторонньою протилежністю інтегральної похибки і пропорційна різниці в зміні похибки в кожен момент часу. З допомогою неї регулюється швидкості зміни майбутньої похибки, що запобігає утворенню коливань. Також потрібно розуміти, що значення цього коефіцієнту дуже чутливе до шуму, тому воно не має бути надто великим.

Недоліками цього методу є те, що він має фіксовану лінійну структуру в той час як робота маніпулятора в реальному світі описується нелінійним чином. Ці коефіцієнти не здатні описати поведінку маніпулятора в інших положеннях з іншою вагою відмінною від налаштованих на початку. Окрім того, цей метод розглядає кожен суглоб системи як окремо незалежну систему, що є помилково в реальному світі, адже існує динамічна взаємодія між цими суглобами.

Виходячи з перелічених недоліків PID-контролер вважається застарілим методом в сучасній робототехніці незважаючи на його простоту [5]. Це також підтверджують експериментальні дослідження Гані [10], де порівняльний аналіз показав, що при зміні маси вантажу класичний PID-регулятор дає похибку на 15–20% більшу, ніж інтелектуальні методи керування.

2.1.2 Метод обчислюваного обертального моменту

Метод обчислюваного обертального моменту (Computed Torque Control) можна вважати одним з фундаментальних для управління роботом-маніпулятором [1]. Головна ідея цього методу полягає в обчисленні динамічних сил, що діють на робота, з метою їх повної компенсації. Важливо враховувати, що точність керування залежить не лише від механічної моделі ланок, а й від динаміки електричних перетворювачів. Зокрема, особливості електромеханічного перетворення енергії в приводах роботів та параметри двигунів безпосередньо впливають на загальну стійкість системи [7].

На початку обчислень система є нелінійною для того, щоб це виправити використовується процес лінеаризації зворотним зв'язком (Feedback Linearization).

Під час виконання цього процесу вираховуються сили інерції, відцентрові сили та сили Коріоліса, а також гравітаційне навантаження. Для проведення цього кроку використовується формула 2.2.

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau, \quad (2.2)$$

де $M(q)$ – матриця інерції,

q – вектор положень ланок,

$\ddot{q}$ – вектор кутових прискорень ланок,

$C(q, \dot{q})$ – матриця відцентрових та коріолісових сил,

$\dot{q}$ – вектор кутових швидкостей ланок,

$G(q)$ – вектор гравітаційних сил,

τ – вектор обертальних моментів.

Після того, як система стає лінійною, можна переходити до обчислення значень, що потрібні для пересування робота в ідеальному (незалежному) стані. Для цього можна використати вже раніше згаданий PID-контролер. Але в цьому випадку нам буде потрібна лише його спрощена версія, що не використовує значення інтегрального коефіцієнту під назвою PD-контролер. Причина з якої використовується спрощена версія контролера полягає в тому, що завдяки компенсації нелінійності об'єкт керування поводить себе як ідеальна ланка, і інтегральна складова для усунення статичної помилки теоретично не потрібна.

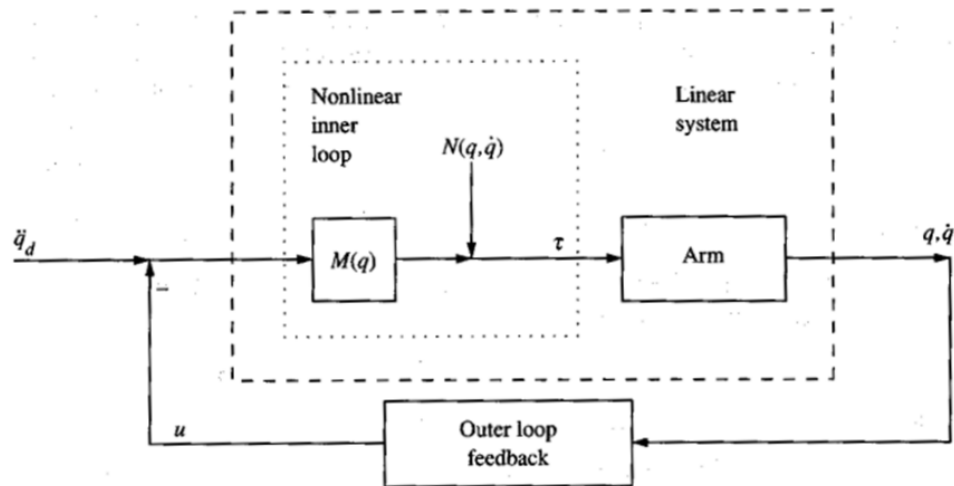


Рисунок 2.1. – Схема роботи методу обчислюваного обертового моменту, що показує внутрішній і зовнішній контури. [2]

Головним недоліком цього методу є його залежність від точності описаних фізичних властивостей маніпулятора [6]. Навіть мінімальні відхилення від правдивих значень можуть призвести до некоректного позиціонування робота. А в реальному світі не є можливим абсолютно правильно визначити параметри тертя або миттєву зміну ваги навантаження.

2.1.3 Метод керування зі застосуванням нейронних мереж

Нейронні мережі також є дуже популярним методом для автоматизації керування роботами маніпуляторами [2]. Причиною цього є те, що основними обмеженнями в управлінні є вимога до рівняння, щоб воно було лінійним відносно параметрів, що означає потребу в точній математичній моделі, яку важко отримати на практиці.

Саме цю задачу вирішують нейронні мережі, а саме вони займаються апроксимацією будь якої нелінійної функції динаміки з довільною точністю. Цей спосіб не потребує глибоких знань фізики та параметри робота, мережа обчислює результат динамічно з використанням вагів системи, які так само налаштовуються в реальному часі з використанням результату помилки. Цей процес називається

навчання (Tuning). При цьому закони налаштування ваг формуються на основі теорії стійкості Ляпунова, що гарантує обмеженість похибки та безпеку системи під час навчання.

Зазвичай для виконання задач управління роботом-маніпулятором використовуються мережі двох типів: двошарові перцептрони (MLP) або мережі радіально-базисних функцій (RBF). MLP – це класична нейронна мережа, яка працює за принципом скалярного добутку. Кожен нейрон цієї мережі бере вхідні сигнали, множить їх на ваги, сумує і пропускає через функцію активації. Завдяки цьому мережа навіть без великої кількості нейронів здатна швидко знаходити закономірності та виконувати процес лінеаризації. Але така мережа потребує значної кількості часу для налаштування вагів.

Мережа RBF працює за принципом подібності, що робить її дуже популярною в адаптивному керуванні. Кожен нейрон даної мережі обчислює евклідову відстань між вхідним сигналом та центром власної зони відповідальності. Після чого використовує отримане значення в функції Гауссіан для отримання виходу нейрону. Цей тип мереж є популярним через високу швидкість навчання, можливість донавчання нейронів, які потрібні та відповідності кожного нейрону за локальні області простору станів.

Після того, як система була приведена до лінійного виду застосовується PD-контролер, як і під час використання методу обчислюваного обертального моменту. І так само в цій системі цей контролер обраховує подальші рухи роботи.

Попри всі переваги нейронних мереж вони так само мають недоліки. Першим з них є потреба в донавчанні (tuning) мережі в режимі реального часу. Цей процес створює значне обчислювальне навантаження і для прискорення їх швидкості потрібні значні ресурси. Окрім цього ще існує проблема під назвою чорна скриня (Black Box). Суть цієї проблеми в тому, що хоч доступ до вагів мережа у нас є, але логіка їх призначення залишається непрозорою. А отже це ускладнює прогнозування дій маніпулятора в певних ситуаціях та гарантування безпеки роботи системи.

2.2 Використання нечіткої логіки для керування в умовах невизначеності

Усі попередньо розглянуті методи контролю рухами робота-маніпулятора мають недоліки через явище невизначеності, яке також було розглянуто вище. Для розв'язання проблеми невизначеності було розроблено розділ математики під назвою нечітка логіка. Ця логіка, на відміну від класичної не розділяє вислови на істину або хибя, а визначає рівень приналежності до певної множини зі значенням від 0 до 1. Також це дозволяє використовувати лінгвістичні змінні для опису невизначеності.

Щоб зрозуміти, яким саме чином використати нечітку логіку для створення системи контролеру звернемось до книги Тімоті Россі «Fuzzy Logic with Engineering Applications» [4]. Також враховано сучасні методи реалізації контролерів, які були систематизовані у новітній роботі «Introduction to Fuzzy Sets, Fuzzy Logic, and Fuzzy Control Systems» [12]. Загальна структура такого контролера зображена на рисунку 2.2. Він описує чотири етапи що мають пройти дані в контролері.

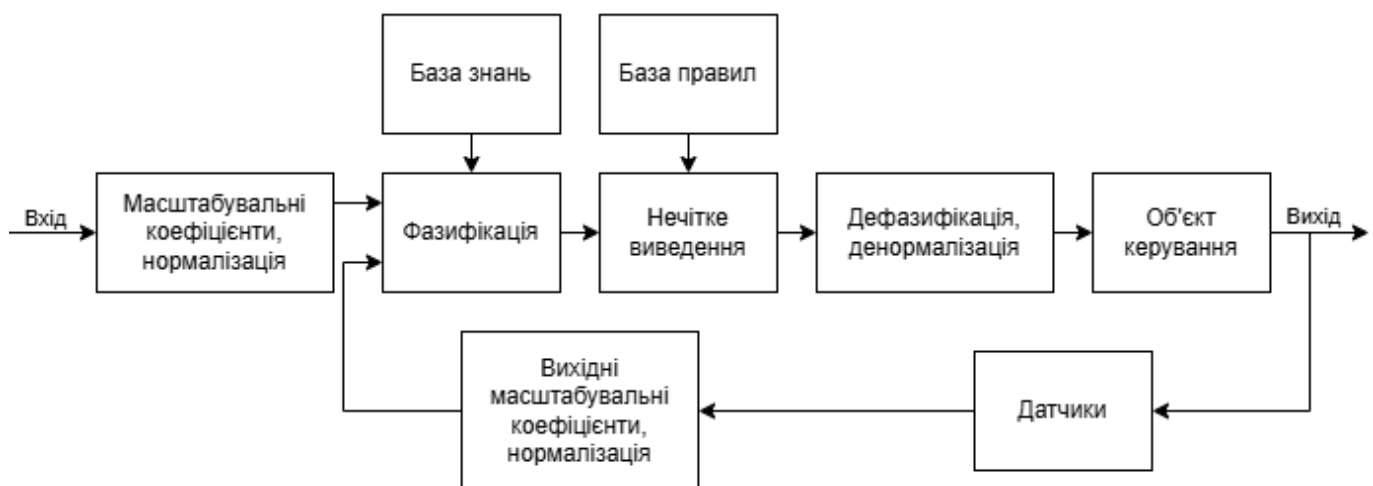


Рисунок 2.2 – Схема блоку системи управління на основі нечіткої логіки. [4]

Перший етап – Фазифікація. Це процес перетворення отриманого числа у нечітке значення за допомогою функції належності. Ця функція визначає ступінь приналежності конкретного числового значення до певної лінгвістичної змінної. Другим етапом є База правил (Rule Base), що представлена у вигляді матриці, де зберігаються знання у форматі «ЯКЩО... ТО...», де рядки та стовпці відповідають станам входних змінних. Ефективність інтелектуальної системи безпосередньо залежить від якості лінгвістичних правил, що описують стратегію керування [11]. Третім етапом є Механізм логічного виведення (Inference Engine) – це процес прийняття рішень контролером. Якщо певне правило спрацювало на значення 0.8 то і відповідна дія матиме такий самий коефіцієнт. Та останнім четвертим етапом є процес дефазифікації. Він приймає на вхід набір нечітких значень, а на вихід має повернути число, що отримане за певним алгоритмом з цих значень.

Наразі існує два найпоширеніших методів логічного виведення: Мамдані (Mamdani) та Сугено (Takagi-Sugeno).

Метод Мамдані для перетворення входних значень у нечіткі використовує правило «ЯКЩО входне значення "ознака", то вихідне значення "ознака"». При умові що дві ознаки пов'язанні логічним «І» використовується операція мінімуму, а якщо логічним «АБО» – операція максимуму. Для дефазифікації використовуються метод центру ваги. Оскільки для використання цього методу потрібно обраховувати площу отриманої фігури він є доволі ресурсномістким процесом.

Метод Сугено був створений, щоб виправити недоліки з складними обчисленнями методу Мамдані. Різниця цього методу, що він використовує правило «ЯКЩО входне значення "ознака", то вихідне значення = $f(x,y)$ ». Тобто замість графіку в якості виходу використовується математична формула або просто число. Це дозволяє під час процесу дефазифікації просто обрахувати середнє зважене значення результатів усіх правил. Це робить метод набагато швидшим, але через зміну правила він втрачає можливість виводу у вигляді лінгвістичної змінної, що зробило метод більш «людиноподібним».

Разом з тим, значна увага в сучасних дослідженнях використання нечіткої логіки приділяється автоматизації налаштування таких систем. У сучасних роботах [8] детально розглядається застосування природоподібних алгоритмів оптимізації. Зокрема, досліджується ефективність налаштування нечітких контролерів за допомогою алгоритму рою частинок (PSO) та алгоритму гравітаційного пошуку (GSA). Це підтверджує, що сучасний розвиток нечіткої логіки рухається в бік гібридних схем, де структура контролера залишається нечіткою, а його параметри налаштовуються алгоритмічно.

Проте, незалежно від методу налаштування, ключовою перевагою нечітких регуляторів залишається їхня природна здатність компенсувати невизначеність без точної математичної моделі.

2.3 Порівняльний аналіз методів керування та обґрунтування вибору

На основі детального розгляд усіх основних методів керування роботами маніпуляторами було проведено їх порівняльний аналіз. Для систематизації розглянутих підходів та вибору оптимального методу для даної роботи було проведено порівняльний аналіз їхніх переваг та недоліків, який наведено в Таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика методів керування маніпуляторами

Назва методу	Переваги	Недоліки	Сфера застосування
PID-регулятор	Простота алгоритму, висока точність та стабільність, швидкодія обчислення.	Обмеженість лінійними системами, низька точність при високій швидкості, погано адаптується при зміні навантаження.	Промислові роботи з низькими швидкостями, прості механізми.

Продовження таблиці 2.1.

Метод обчислюваного оберտального моменту (СТС)	Висока точність відстеження траєкторії, повна компенсація динаміки робота.	Вимагає абсолютно точної математичної моделі, чутливий до похибок моделювання.	Прецизійні роботи в лабораторних умовах.
Нейронні мережі (Neural Networks)	Здатність до навчання, апроксимація будь-якої нелінійної функції.	Потребують великого обсягу даних та часу для навчання, проблема «чорної скринька» (Black Box).	Складні автономні роботи, що функціонують у змінному середовищі
Нечітка логіка (Fuzzy Logic)	Інтуїтивна зрозумілість, не вимагає точної математичної моделі, стійкість до шумів та невизначеності.	Складність налаштування бази правил при великій кількості входів, відсутність строгих методів аналізу стійкості.	Системи з неповними або невизначеними параметрами, задачі, що вимагають плавного керування.

За результатами проведеного порівняльного аналізу можна констатувати, що оптимальним методом для керування роботом маніпулятором Puma 560 в умовах невизначеності у даній роботі є використання нечіткої логіки.

Цей вибір обґрунтовується декількома факторами. По-перше, суттєвою нелінійністю моделі маніпулятора Puma 560. Це робить використання класичного PID-регулятора неефективним, оскільки він не здатний забезпечити високу точність при роботі з нелінійними системами, що змінюють свої параметри в просторі.

По-друге, відсутня точна моделі маніпулятора, оскільки у рамках даної роботи немає точних даних про моменти інерції та тертя у кожному шарнірі. Через це використання методу обчислювального оберտального моменту не є можливим, тому що воно потребує максимально точної моделі.

По-третє, метод нечіткої логіки дозволяє легко імплементувати евристичні правила керування, на відміну від нейронних мереж, які потребують тривалого

навчання та страждають від проблеми інтерпретованості прийнятих рішень («чорна скринька»).

Отже, використання нечіткого контролера є найбільш доцільним компромісом між складністю реалізації та якістю керування для поставленої задачі.

Висновок до розділу 2

У розділі 2 було проведено огляд наявних методів керування просторовими рухами робота-маніпулятора та вибір оптимального варіанта для цієї дипломної роботи.

На початку цього розділу було проаналізовано три базових методи керування робототехнічними системами, а саме системи з використанням ПД-регуляторів, метод обертального моменту та регулятори на основі нейронних мереж. Розглянуто алгоритми роботи цих методів, а також їх переваги та недоліки.

Наступним кроком стало дослідження математичного апарату нечіткої логіки, як основного методу керування для розробки системи. Були проаналізовані всі етапи роботи нечіткого регулятора від фазифікації до дефазифікації. Окрім того, було розібрано базові методи нечіткого виведення: Мамдані та Сугено.

В кінці розділу було виконано порівняння всіх наведених методів, виділено всі переваги та недоліки, а також найбільш ефективний тип регулятора для керування роботом-маніпулятором. Після вибору було обґрунтовано чому саме нечітка логіка є найкращою для подолання негативного впливу параметричної невизначеності.

3 ОБҐРУНТУВАННЯ ВИБОРУ ТА ОГЛЯД ПРОГРАМНО-ТЕХНІЧНОГО ІНСТРУМЕНТАРІЮ ДЛЯ СТВОРЕННЯ СИСТЕМИ КЕРУВАННЯ

Для створення програмного забезпечення, що відповідає меті роботи, а саме керування просторовими рухами робота маніпулятора в умовах невизначеності з використанням нечіткої логіки, потрібно сформувати найкращий стек технологій. Правильно підібрані інструментальні засоби безпосередньо впливають на швидкість розробки, гнучкість системи та можливості її подальшого масштабування. У цьому розділі наведено детальне обґрунтування вибору мови програмування, спеціалізованих математичних бібліотек, екосистеми керування та середовища фізичної симуляції. Комплексний підхід до вибору цих компонентів гарантує стабільну роботу системи в умовах динамічних збурень та високу точність позиціювання.

3.1 Мова програмування Python як засіб розробки

Основним інструментом розробки будь якого програмного продукту є мова програмування, тому напочатку потрібно визначитися саме з нею. Для створення інтелектуальної системи керування було обрано Python, як основну мову програмування.

Вибір саме цієї мови був ретельно продуманий. Сучасні роботи маніпулятори використовують дві мови програмування C++ та Python. Але різниця в тому, що C++ використовується для розробки низькорівневих драйверів, взаємодії з апаратним забезпеченням, такими як датчиками або енкодерами та забезпечення керування приводами в режимі жорсткого реального часу.

Натомість Python використовується для реалізації різноманітних оптимізаційних алгоритмів застосовується для реалізації високорівневої логіки («мозку» робота), алгоритмів комп'ютерного зору, планування траєкторій руху та систем штучного інтелекту. Оскільки головним завданням даної роботи є розробка

інтелектуального програмного модуля на базі нечіткої логіки, зважаючи на описане архітектурне розділення, найбільш доцільним та ефективним інструментом є саме мова Python. Вагомою перевагою також є низький поріг входження та висока швидкість прототипування, що дозволяє інженеру зосередитися на математичній моделі регулятора, а не на складнощах синтаксису.

3.2 Екосистема бібліотек Python для розробки регулятора

Окрім цього, мова програмування Python також займає провідні місця в розробці проєктів, що пов'язані з використанням штучного інтелекту, машинного навчання, комп'ютерного зору та аналізу даних. Причинами її лідерства є її головна особливість, а саме незліченно велика кількість сторонніх бібліотек. Так вони беруть на себе реалізацію механізмів найнижчого рівня, що дозволяє сфокусуватися на основному завданні.

Першою обраною бібліотекою була `scikit-fuzzy`. Це спеціалізована бібліотека, яка реалізує алгоритм нечіткої логіки. Її головними можливостями є:

- Генерація різних функцій належності. Бібліотека дозволяє використовувати трикутні, трапецієподібні, гаусові, дзвоноподібні та сигмоїдальні функції належності.
- Використання різних логічних операцій. Під час моделювання нечіткого виведення надається можливість використовувати логічні оператори «І», «АБО» та «НЕ».
- Вибір методу дефазифікації. Для конвертація нечітких множин у чіткі числа пропонується на вибір методи центроїда, бісектриси, найменшого/середнього/найбільшого з максимумів.
- Підтримка кластеризації Fuzzy C-Means: Поділ даних на кластери, де один об'єкт може частково належати до кількох груп одночасно.

Окрім всіх цих можливостей, головною перевагою бібліотеки `scikit-fuzzy` є повна підтримка класичного алгоритму нечіткого виведення Мамдані. Також код

написаний з використанням цієї бібліотеки повністю інтегрується зі стеком SciPy, а також MATLAB. Використання готових та оптимізованих функцій позбавляє необхідності створювати алгоритм нечіткого виведення з нуля, тим самим мінімізуючи ризик виникнення прихованих програмних помилок.

Наступною фундаментально важливою бібліотекою стала rclpy (ROS 2 Client Library for Python). Це базова бібліотека для інтеграції Python-скриптів до екосистеми ROS 2. Вона використовується при створенні ROS 2 вузлу, який використовуватиме механізм публікації для надсилання команд роботу в спеціальний канал даних. Також бібліотека забезпечує механізми підписки для безперервного отримання телеметрії про поточний стан суглобів маніпулятора.

Крім того, інструментарій rclpy дозволяє використовувати внутрішні таймери для забезпечення стабільної частоти виконання головного циклу керування (100 Гц), а також автоматично конвертує отримані ROS-повідомлення у формат стандартних об'єктів Python, що значно спрощує їх подальшу математичну обробку. Завдяки використанню промислового стандарту під капотом ROS 2, бібліотека rclpy забезпечує високу надійність передачі пакетів без втрат, що є критично важливим для керування реальним обладнанням.

Також було використано дві допоміжні бібліотеки NumPy та Matplotlib. Бібліотека Matplotlib надає доступ до створення різноманітних двовимірних та тривимірних графіків. Саме за допомогою цієї бібліотеки будуть створені всі графіки функцій належності, нелінійну 3D-поверхню керування та бази правил для зображення у звітах та на презентаціях.

NumPy (Numerical Python) – це бібліотека з відкритим кодом, що призначена для ефективних наукових та інженерних обчислень. Наразі вона є фундаментальною для будь яких наукових робіт, які включають області Data Science, штучного інтелекту та робототехніки. Її ключові можливості це забезпечення доступу до потужного багатовимірного масиву, до векторизованих обчислень, до механізму, що дозволяє проводити арифметичні операції над масивами різних, але сумісних розмірів та до функцій лінійної алгебри. Внутрішня

архітектура бібліотеки реалізована мовою C, що забезпечує високу швидкість всіх обрахунків.

У межах даної роботи NumPy є головним інструментом для роботи математичного ядра контролера: бібліотека використовується для генерації масивів універсальних множин, а також для виконання швидких матричних операцій під час агрегації нечітких правил.

3.3 Інтегроване середовище розробки Visual Studio Code

Окрім мови програмування також було обрано інтегроване середовище розробки, ним стало Visual Studio Code. Це середовище є одним з найпопулярніших не тільки при розробці мовою Python, а взагалі у програмуванні. Така популярність спричинена можливістю встановлення різноманітних розширень для роботи з будь-якими інструментами розробки від мови програмування до менеджера бази даних.

Також це середовище має зручну консоль та зручні засоби відлагодження. Наявність потужного дебаггера дозволяє покроково виконувати алгоритм нечіткого виведення, відстежувати значення змінних у реальному часі та виявляти логічні помилки. Легку інтеграцію Git в проект з панеллю подальшої роботи з ним. Водночас VS Code має вбудований графічний інтерфейс для роботи з Git, що спрощує відстеження змін у коді та дозволяє безпечно експериментувати з різними версіями алгоритмів.

До того ж це середовище розробки також має підтримку платформи Jupyter Notebooks, що ідеально підходить для побудови та перевірки різних графіків. Важливою перевагою є також підтримка інтелектуального автодоповнення коду та статичних аналізаторів, які допомагають підтримувати високий стандарт написання коду при мінімальному споживанні системних ресурсів ПК.

3.4 Екосистема ROS 2 для інтеграції компонентів керування

ROS 2 – це проміжне програмне забезпечення для розробки складного робототехнічного забезпечення на базі розподілених систем. Незважаючи на назву, ROS 2 є не окремою операційною системою, а набором інструментів, бібліотек і протоколів, які запускаються поверх стандартних ОС.

Архітектура ROS 2 складається з декількох компонентів, а саме вузла, топіку, сервісу та дії. Вузол (Node) – це програма, що виконає певне одне завдання в системі. Таким завданням може бути зчитування даних звідкись, побудова маршруту або керування моторами.

Топік (Topic) – це іменовані канали для передачі потокових даних. Вони працюють за моделлю «видавець-підписник». В цій моделі видавець публікує дані в топік, а підписник чекає та отримує дані з цього каналу.

Сервіс (Service) – це механізм комунікації між вузлами, побудований за архітектурою «клієнт-сервер». На відміну від топіків, які забезпечують безперервний та односпрямований потік даних, сервіси використовуються для виконання короткочасних дискретних операцій. Цей протокол є оптимальним для задач, що не потребують постійного моніторингу, наприклад, для одноразового зчитування конфігурації.

Дія (Action) – це комунікаційний протокол, спеціально розроблений для виконання довготривалих та складних завдань. Дія складається з трьох елементів цілі, зворотного зв'язку та кінцевого результату. На відміну від сервісів, дії є асинхронними. У сучасній робототехніці саме протокол дій є базовим стандартом для керування просторовими траєкторіями руху.

Для імплементації розробленого в ході виконання роботи нечіткий регулятор до маніпулятору Puma 560 буде створений спеціальний керуючий вузол. Цей вузол буде оперувати системою топіків для отримання інформації про поточну позицію робота та для надсилання обчислених команд руху у контролер траєкторії.

3.5 Використання симулятора Gazebo для верифікації руху маніпулятора

Gazebo – це потужне середовище для симуляції роботів в тривимірному просторі. Цей симулятор, окрім точної візуалізації кінематики маніпулятора, також забезпечує достовірне моделювання впливу різних нелінійних фізичних явищ. Саме ці фактори представляють стан невизначеності, який має компенсувати розроблений регулятор з використанням методів нечіткої логіки.

Також величезною перевагою симулятора Gazebo є його легка інтеграція з ROS 2. Для цього використовуються пакет плагінів `gazebo_ros2_control`. Вони виступають мостом між симуляцією та контролером, дозволяючи в режимі реального часу зчитувати поточний стан суглобів та передавати керуючі зусилля безпосередньо на віртуальні приводи маніпулятора.

У межах цієї роботи для тестування функціонування нечіткого регулятора буде використовуватися модель, запущена в симуляторі Gazebo. За допомогою цього середовища можна без ризиків провести налаштування внутрішніх параметрів нечіткого контролера, зокрема база правил та функції належності. Більше того, використання віртуального середовища є доцільним з огляду на безпеку, оскільки повністю виключає ризик механічного пошкодження реального маніпулятора під час відлагодження системи. Крім того, цей симулятор надає забезпечує наочну демонстрацію фінальної версії продукту в умовах, максимально наближених до реальних фізичних процесів.

Висновок до розділу 3

У розділі 3 було проведено вибір основного технологічного стеку для розробки програмного модуля на основі нечіткої логіки для керування роботом-маніпулятором в умовах параметричної невизначеності.

На першому етапі було обґрунтовано вибір найважливішого інструмента створення програмного забезпечення – мову програмування. Під час аналізу було

виявлено, що для робототехнічних систем використовуються дві мови C++ та Python. Було досліджено переваги, недоліки та прикладні сфери застосування цих мов. На основі отриманої інформації було прийнято рішення, що мова Python – це найкращий вибір для цієї роботи.

Після вибору мови програмування наступним найбільш логічним кроком стала підготовка екосистеми бібліотек для мови Python. Були описані чотири бібліотеки для різних потреб та етапів роботи програми. Для комфортного використання алгоритмів нечіткої логіки обрано було `scikit-fuzzy`, а для інтеграції з середовищем ROS 2 – `rclpy`. Також були визначені бібліотеки `numpy` та `matplotlib` для підготовки графіків та складних обчислень.

Наступним етапом став вибір інтегрованого середовища розробки. Після дослідження всіх можливостей середовища Visual Studio Code було вирішено продовжити розробку саме в цьому середовищі.

Завершальним етапом вибору стеку інструментів стали інструменти для інтеграції та математичного моделювання. Для забезпечення взаємодії між компонентами системи було обрано архітектурний каркас ROS 2, що є сучасним стандартом у галузі робототехніки.

Для моделювання робота маніпулятора та навколишнього світу відповідає середовище симуляції Gazebo. Головною перевагою якого є повна симуляція всіх фізичних процесів, що викликані різними фізичними силами, такими як інерція, тертя суглобів або сила тяжіння. Це особливо важливо для цієї дипломної роботи, адже всі ці явища утворюють параметричну невизначеність.

4 ПРОЕКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ КЕРУВАННЯ МАНІПУЛЯТОРОМ

Перед початком розробки будь якого програмного забезпечення потрібно якісно спроектувати надійну структуру для майбутньої системи. Правильно обрана архітектура дозволяє мінімізувати кількість потенційних помилок та надає можливість легко розширення функціоналу системи. На основі обґрунтованого у попередньому розділі технологічного стеку, в межах цього розділу здійснюється створення системи, як окремих функціональних блоків.

4.1 Проектування структури розробленого ROS-модуля

Робот-маніпулятор Puma 560 це комплексна система, що складається з багатьох різних програмних сутностей, які виконують кожні свої завдання та за допомогою певних алгоритмів та протоколів комунікують між собою. Одним із завдань цієї роботи було правильна інтеграція розробленого програмного забезпечення в цю систему. Для розуміння логіки роботи системи схема інформаційної взаємодії розробленого вузла з іншими компонентами наведена на рисунку 4.1.

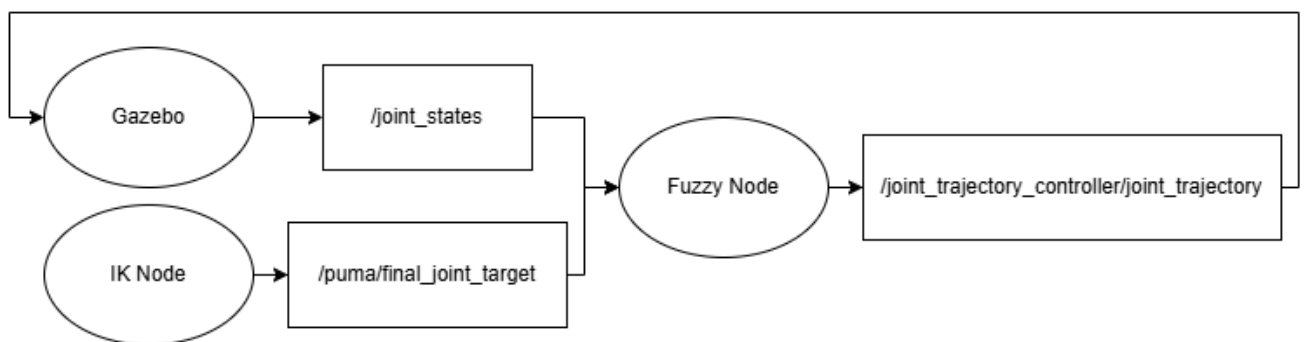


Рисунок 4.1. – Схема інформаційної взаємодії вузлів та топіків у середовищі ROS2

На наведеній схемі відображено структурне поєднання розробленого програмного модуля з іншими елементами екосистеми. Центральним елементом є

вузол нечіткого керування (Fuzzy Node), який працює в режимі підписника і зчитує дані про стан суглобів та цільові кути. На основі цих значень алгоритм формує команди для переміщення ланок, які публікуються у відповідний вихідний канал даних для відпрацювання симулятором.

Важливою частиною архітектури є замикання контуру керування через зовнішні компоненти. Вузол оберненої кінематики (IK Node) забезпечує перерахунок декартових координат у кути повороту.

Розроблений вузол має взаємодіяти з трьома топіками для повної реалізації свого функціоналу. Перелік назв цих каналів даних з описом їх призначення наведений нижче:

- `/joint_states` – на цей топік нечіткий вузол підписується та завдяки ньому він отримує значення телеметрії маніпулятора. В цей канал даних постійно приходять значення з частотою вказаною в налаштуваннях робота.

- `/puma/final_joint_target` – на цей топік нечіткий вузол підписується та завдяки ньому він отримує значення цільових кутів для кожної з 6 ланок маніпулятора Puma 560. В цей канал дані публікуються кожен раз коли користувачем задається нова позиція для робота та зазвичай цільові кути передає вузол оберненої кінематики.

- `/joint_trajectory_controller/joint_trajectory` – у цей топік розроблений вузол публікує команди для маніпулятора. В функції зворотного виклику обчислюються ці команди із застосуванням нечіткого контролера та повертаються у форматі об'єкту `JointTrajectoryPoint`.

Така архітектура взаємодії дозволяє забезпечити стабільний обмін даними між інтелектуальним контролером та віртуальною моделлю маніпулятора, гарантуючи плавність траєкторії руху.

4.2 Розробка ОО-архітектури програмного модуля

Для реалізації керуючого програмного модуля було застосовано об'єктно-орієнтований підхід. Розроблене програмне забезпечення для керування рухами робота-маніпулятора складається з двох класів: FuzzyController та FuzzyNode. Для наочного представлення структури програмного забезпечення, його атрибутів, методів та зв'язків між компонентами на рисунку 4.2 наведено UML-діаграму класів.

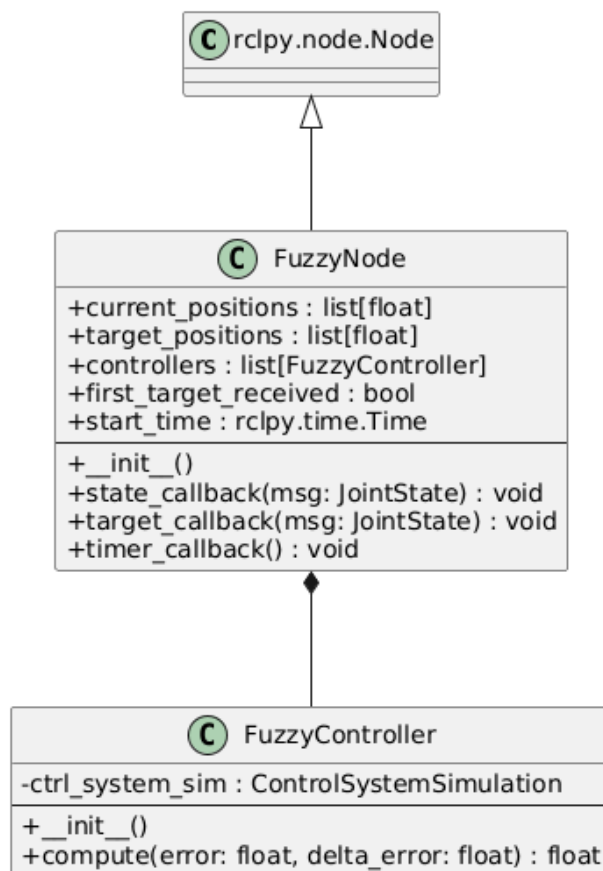


Рисунок 4.2. – UML-діаграма класів програмного модуля керування

Як видно з наведеної діаграми, головний клас `FuzzyNode` успадковує базовий функціонал від стандартного класу `rclpy.node.Node`, що дозволяє йому інтегруватися в екосистему ROS 2. Процес запуску вузла відбувається в функції `main()` скрипта, що містить код класу вузла. Цей клас складається з конструктора та трьох методів зворотного виклику.

В конструкторі спочатку оголошуються назви всіх шести суглобів робота-маніпулятора, створюються поля для зберігання поточних значень швидкості та позицій всіх ланок, а також їх цільові значення. Також для кожного суглоба маніпулятора створюється окремий екземпляр класу `FuzzyController`. Це відображає децентралізований підхід до керування роботом. Хоча всі контролери використовують єдину математичну модель, вони функціонують паралельно та обчислюють керуючі сигнали для своєї осі індивідуально. Така архітектура задовольняє критерії ефективності роботи системи та надає можливість для майбутнього масштабування до впровадження ієрархічної нечіткої логіки. Після цього ініціалізуються два поля з валідаційними значеннями часу старту роботи вузла та булеве значення чи була отримана ціль. Крім того, тут відбувається підписка на топіки `/joint_states` та `/puma/final_joint_target` для отримання телеметрії та цільових позицій відповідно. Одночасно з цим створюється об'єкт публікації у топик `/joint_trajectory_controller/joint_trajectory` для надсилання команд роботу-маніпулятору. Для роботи з цими топіками потрібні методи зворотного виклику, що були створені далі.

Метод зворотного виклику `state_callback()` працює разом із об'єктом-підписником на топик `/joint_states`. Цей метод отримує аргумент `message` типу `JointState`, що містить всю інформацію про поточну телеметрію. Робота методу представлена циклом, який виконується для кожного елементу в масиві `name` з об'єкту `message`. Після отримання імені перевіряється, чи воно належить до переліку імен суглобів. Якщо ім'я пройшло перевірку, програма визначає його індекс у списку, отримує відповідні значень швидкості та позиції, після чого зберігає ці значення до спеціальних полів створених раніше у конструкторі класу.

Другий метод зворотного виклику `target_callback()` використовується підписником на топик `/puma/final_joint_target`. На початку виконання методу відбувається валідаційна перевірка, чи був вузол створений раніше за надсилання цільової позиції. Дані про час відправки цільового значення отримуються з метаданих аргументу `message`. Ця перевірка потрібна для відсікання застарілих або

некоректних повідомлень (наприклад, від попередніх сесій симуляції), що дозволяє гарантувати обробку лише актуальних цільових координат у реальному часі. Після успішного проходження валідації, змінюється значення поля, яке зберігає інформацію про те чи було отримане цільове значення. Далі запускається цикл, який шукає відповідність назв суглобів маніпулятора з іменами в масиві `message.name`. Коли такі збіги знаходяться то зчитується значення записаної позиції та зберігається воно у вигляді цільового значення.

Останній створений метод зворотного виклику `timer_callback()` працює з публікатором у топіка `/joint_trajectory_controller/joint_trajectory` та здійснює обчислення керуючих зусиль за допомогою екземплярів нечіткого контролера й формує фінальні команди для публікації в канал зв'язку. Цей метод містить основну керуючу логіку та не отримує жодного аргументу. Опис його алгоритму наведений у наступному підрозділі 4.3.

Клас `FuzzyController` інкапсулює математичний апарат нечіткої логіки. Зв'язок між структурними елементами реалізовано через відношення композиції: клас `FuzzyNode` містить список із шести незалежних екземплярів класу `FuzzyController`, по одному для керування кожним суглобом робота Puma 560.

Його метод `__init__()` є конструктором класу та відповідає за визначення вхідних та вихідних змінних нечіткого контролера, лінгвістичних термів, функцій належності та бази правил. Після ініціалізації всіх цих параметрів створюється об'єкт системи нечіткого виведення за допомогою спеціалізованої бібліотеки `scikit-fuzzy`.

Метод `compute()` отримує два аргументи вхідних змінних поточних швидкості та позиції ланки. На основі цих параметрів алгоритм запускає метод, створеного в конструкторі об'єкту налаштованого апарату нечіткої логіки, для обчислення вихідного значення нечіткого регулятора. Отримане значення є результатом роботи методу та повертається як обчислений керуючий сигнал для конкретного суглоба маніпулятора.

4.3 Опис алгоритму роботи головного циклу керування

За створення керуючих команд для переміщення відповідає метод `timer_callback()`. Цей метод є циклічним та виконується з частотою 100 Герц. Для наочного представлення логіки обчислення керуючих зусиль на рисунку 4.3 наведено блок-схему відповідного алгоритму.

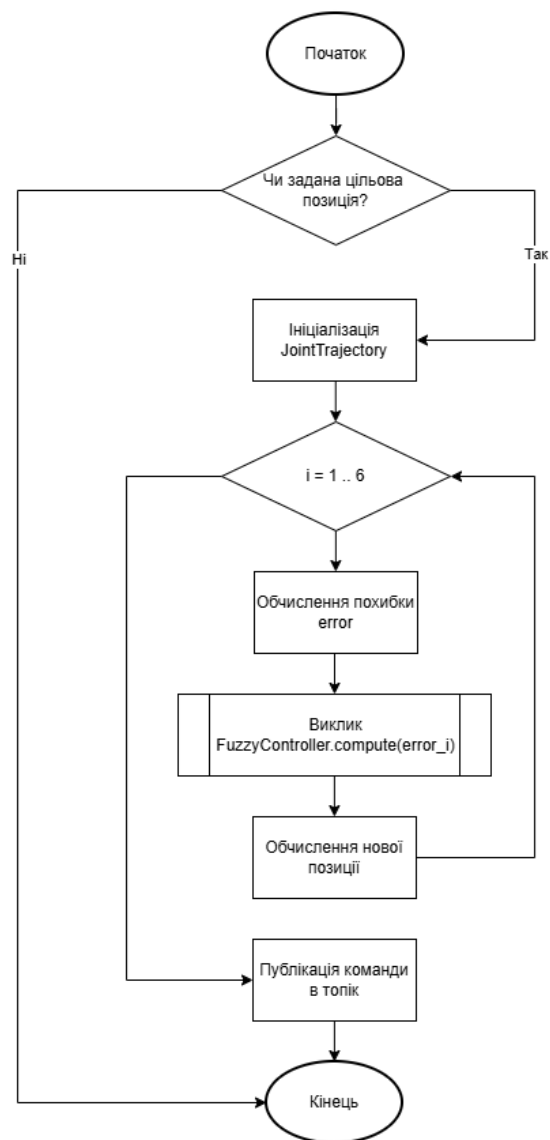


Рисунок 4.3. Блок-схема алгоритму

Як показано на наведеній схемі, алгоритм розпочинає роботу з перевірки наявності цільової позиції. Ця умова потрібно, щоб система переконалась в тому що цільова позиція була задана. Вона слугує програмним запобіжником від неконтрольованого руху при запуску. Якщо цільова позиція задана успішно, далі створюється та налаштовуються два поржні об'єкт типу JointTrajectory та JointTrajectoryPoint, які слугуватимуть базовими контейнерами для формування вихідного керуючого повідомлення.

На наступному етапі запускається цикл на 6 ітерацій для кожного суглоба. На кожному кроці обраховується спочатку помилка – різниця між цільовим і поточним кутом. Потім це значення помилки передається в екземпляр нечіткого контролера цієї ланки та вираховується значення нової позиції за допомогою отриманого від регулятора числа.

Після розрахунку нових позицій для всіх шести ланок, формується фінальна команда для досягнення цих точок протягом наступних 10 мілісекунд, яка публікується у відповідний топик для виконання симулятором.

4.4 Проєктування динамічної взаємодії компонентів системи

Спроектвана система складається з декількох компонентів, які мають взаємодіяти один з одним для реалізації поставленої задачі. Етап планування правильного зв'язку між компонентами є критичним для цього продукту, адже головний модуль має працювати з частотою 100 Гц, що є однією із ключових нефункціональних вимог. Для повного аналізу взаємодій компонентів програмного комплексу було розроблено UML-діаграму послідовності, що зображено на рисунку 4.4. Вона відображає основні інформаційні потоки та послідовність роботи кожного компонента системи під час виконання однієї повної ітерації головного циклу керування.

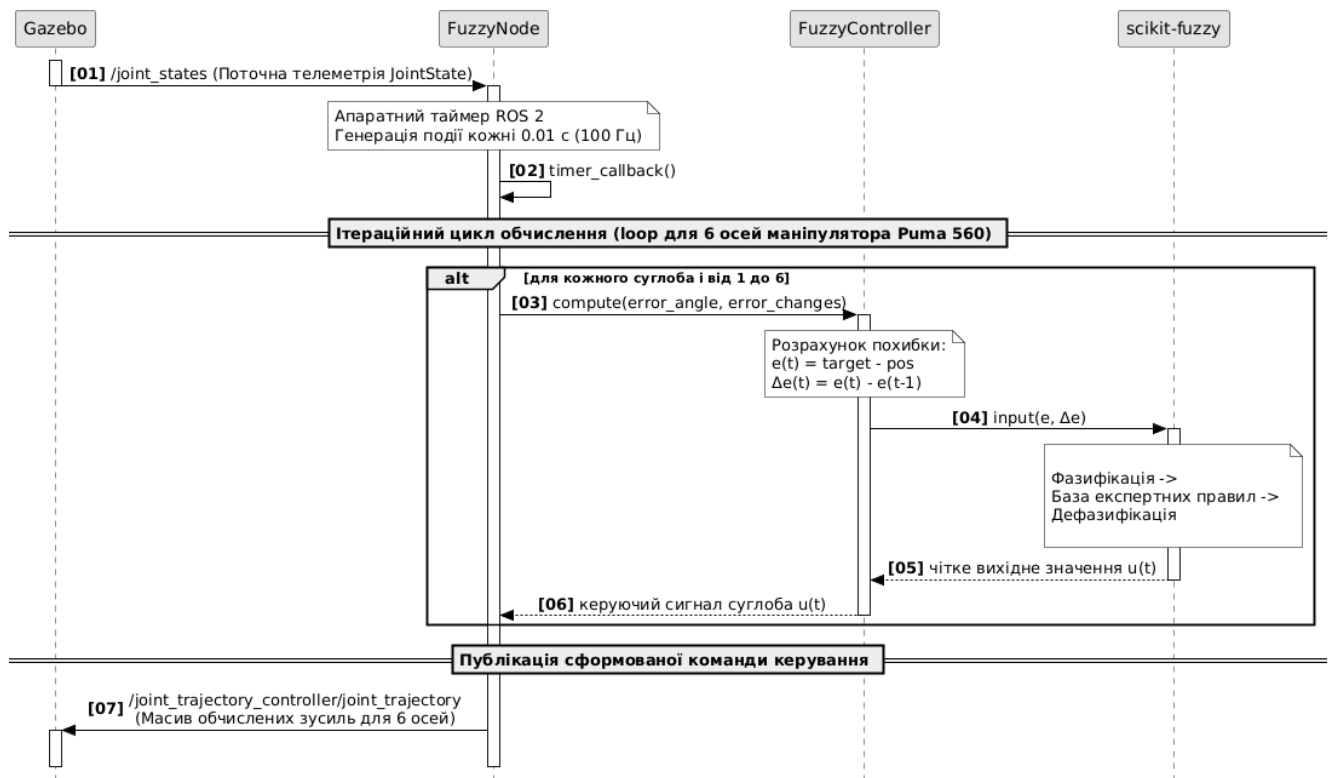


Рисунок 4.4. – UML-діаграма послідовності

Як видно з наведеної діаграми, у процесі взаємодії беруть участь чотири основних компоненти системи:

- Gazebo – цей компонент виступає в ролі навколишнього світу. Симулятор імітує впливи навколишнього середовища, передає дані телеметрії робота-маніпулятора далі по системі та приймає результати обчислень у вигляді команд для об'єкта керування.

- FuzzyNode – це власноруч розроблений компонент, який є частиною екосистеми ROS 2. Він здійснює асинхронне зчитування даних наданих йому з інших компонентів та відповідає за генерування та публікацію фінальних набір команд керування роботом-маніпулятором.

- FuzzyController – це також створений в рамках цієї роботи компонент системи. Його основне призначення є визначення конфігурації математичного апарата нечіткої логіки, приймати значення для обчислень та повертати результат обчислень у головний вузол.

- Scikit-fuzzy – це компонент, що представлений бібліотекою сторонніх розробників. Вона реалізує безпосередні математичні операції нечіткого виведення (фазифікацію, композицію та дефазифікацію), забезпечуючи високу швидкість обробки даних.

Повну послідовність дій зображену на діаграмі на рисунку 4.4 можна описати у вигляді семи основних етапів:

1. Перший етап життєвого циклу даних полягає в передачі значень телеметрії роботу від симулятора Gazebo до ROS 2 вузла FuzzyNode за допомогою топіка `/joint_states`.

2. Під час другого кроку запускається апаратний таймер ROS 2 для генерації команди кожні 0.01 секунди. Протягом цього кроку запускається функція `timer_callback()`, яка працює циклічно до моменту ручного переривання команди. Етапи з третього по шостий представляють одну ітерацію циклу обчислень команди для суглоба.

3. Крок номер 3 складається з обчислення помилок та запуску методу `compute()` всередині вузла FuzzyNode, який передає виконання до третього компонента системи – FuzzyController.

4. На наступному кроці відбуваються процеси фазифікації, композиції за базою експертних правил та дефазифікації для отримання чіткого числового результату. Ці процеси проходять в компоненті `scikit-fuzzy`;

5. По завершенню етапу обчислення потік виконання разом із чітким вихідним значенням повертається від обчислювального ядра до відповідного об'єкта FuzzyController,

6. Шостий крок повертає сигнал керування суглоба до компонента FuzzyNode. Після виконання кроків з третього по шостий для кожного з суглобів генерується наступна команда керування.

7. Останній етап під номером сім відповідає за публікацію згенерованої команди до топіка `/joint_trajectory_controller/joint_trajectory`. Разом із цим потік

виконання повертається до компонента Gazebo, в якому і відбудеться симуляція руху робота-маніпулятором.

Таким чином, послідовність, зображена на UML-діаграмі рисунку 4.4, повністю відповідає спроектованому алгоритму роботи та взаємозв'язку різних компонентів системи керування. Це забезпечує стабільну та швидку обробку інформаційних потоків, що є критично важливо для виконання ключової нефункціональної вимоги щодо підтримки частоти дискретизації контуру керування на рівні 100 Гц.

Висновок до розділу 4

У розділі 4 було реалізовано об'єктно-орієнтовану архітектуру ROS-вузла керування на базі апарату нечіткої логіки. Розроблений алгоритм головного циклу гарантує стабільну взаємодію із симулятором Gazebo та точне відпрацювання заданих просторових траєкторій маніпулятора.

Першим етапом стало проектування програмного вузла в системі ROS 2. Проаналізовано необхідні для взаємодії топіки та визначено специфікацію вхідних і вихідних топіків. Окрім того, було розроблено схему інформаційної взаємодії вузлів та топіків у середовищі ROS2 для забезпечення децентралізованого обміну даними між обчислювальним модулем та середовищем симуляції.

Після цього була розроблена об'єктно-орієнтована архітектура програмного модуля. Визначено ключові програмні сутності, що необхідно буде розробити та візуалізовано зв'язки між ними на UML-діаграмі класів програмного модуля керування. Опис архітектури охоплює детальну характеристику внутрішніх методів класів, їхніх вхідних параметрів та механізмів міжкласової взаємодії.

Наступним кроком став детальний опис роботи головного таймерного циклу програмного забезпечення для керування. Цей метод є найважливішою частиною класу FuzzyNode та спрацьовує з частотою 100 Гц. Для кращого розуміння алгоритму роботи цього методу була створена блок-схема перебігу всіх дій. В ній

чітко зображені всі валідаційні перевірки та цикли для обробки одразу шести ланок робота-маніпулятора.

Наостанок було спроектовано динамічну взаємодію всіх компонентів створеної системи керування роботом-маніпулятором. Спочатку було описано всі чотири основні елементи програмного продукту між якими відбувається перебіг потоків даних додатка. До цих компонентів належать: симулятор Gazebo, ROS 2 вузол FuzzyNode, апарат конфігурації нечіткої логіки FuzzyController та бібліотека для математичних обчислень scikit-fuzzy. Щоб зобразити взаємодію між цими частинами системи було створено UML-діаграму послідовності циклу керування. Розроблена послідовність складається із семи різних кроків, протягом яких потік виконання проходить через усі описані компоненти та завершується початком руху робота-маніпулятора.

5 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА МОДЕЛЮВАННЯ РОБОТИ МАНІПУЛЯТОРА PUMA 560

У цьому розділі розглядається шлях практичної розробки логічної моделі керування роботом-маніпулятором. Детально описується процес створення нечіткого регулятора від етапу фазифікації вхідних даних до фінальної дефазифікації керуючого сигналу.

5.1 Синтез логічної моделі керування

Створення математичної моделі керування з використанням нечіткої логіки було одним з ключових завдань цієї роботи. Процес проектування розпочався з визначення вхідних та вихідних змінних. Як змінні на вхід було обрано похибку позиціонування та швидкість її похибки. Такий вибір був обґрунтований специфікою роботи маніпулятора Puma 560 та математичними характеристиками нечіткої логіки. Саме поточного кута та поточної швидкості руху ланки надсилає в телеметрії робот-маніпулятор до спеціалізованого топіка. Тому обрати моменти в суглоб як вихідні сигнали було б неможливо в межах саме цієї роботизованої системи.

Окрім того, збільшення кількості вхідних змінних призводить до збільшення правил в базі правил у геометричній прогресії, тому додавання третього вхідного значення яким була б сума помилок є недоцільною витратою ресурсів всієї системи обрахунків. Через це вирішено, що оптимальною кількістю вхідних значень для цієї роботи є дві змінні.

Вихідною змінною, що генерується є керуючий сигнал – це нормований приріст позиції, який визначає напрямок та інтенсивність руху ланки на кожному кроці. Нормованість цього значення обумовлена розділенням сфер відповідальності між різними модулями програмної системи, а отже перетворення в реальну команду відбувається на рівні виконавчих драйверів у вузлові. Визначення інтенсивності

руху відбувається за допомогою значення модуля керуючого сигналу, а напрямку – залежно від знаку обчисленого числа (додатного або від’ємного).

Після вибору вхідних та вихідних значень на цьому етапі також були визначені універсальні множини, тобто числові межі для цих значень за допомогою евристичного способу. Перелік цих меж наступний:

- Для помилки кута межі обрані від $-\pi$ до π рад.
- Для швидкості межі обрані були від -2 до 2 рад/с.
- Для керуючого сигналу від -1 до 1

Другим етапом синтезу математичної моделі було створення лінгвістичних термів, до яких із певним ступенем належали б значення вхідних чисел. Щоб визначити цю міру використовуються функції належності. Для цього нечіткого контролеру було обрано трикутну функцію належності. Перелік всіх лінгвістичних термів наведено нижче:

- NB (Negative Big) – Велика від’ємна
- NS (Negative Small) – Мала від’ємна
- Z (Zero) – Нуль
- PS (Positive Small) – Мала додатна
- PB (Positive Big) – Велика додатна

Вибір трикутної форми функцій належності обумовлене її високою обчислювальною здатністю та лінійним характером схилів, що спрощує операції перетину та об’єднання нечітких множин на етапі фазифікації. Графічне зображення експертних функцій належності для вхідних лінгвістичних змінних помилки позиції та швидкості зміни цієї похибки для суглобів маніпулятора зображено на рисунках 5.1.

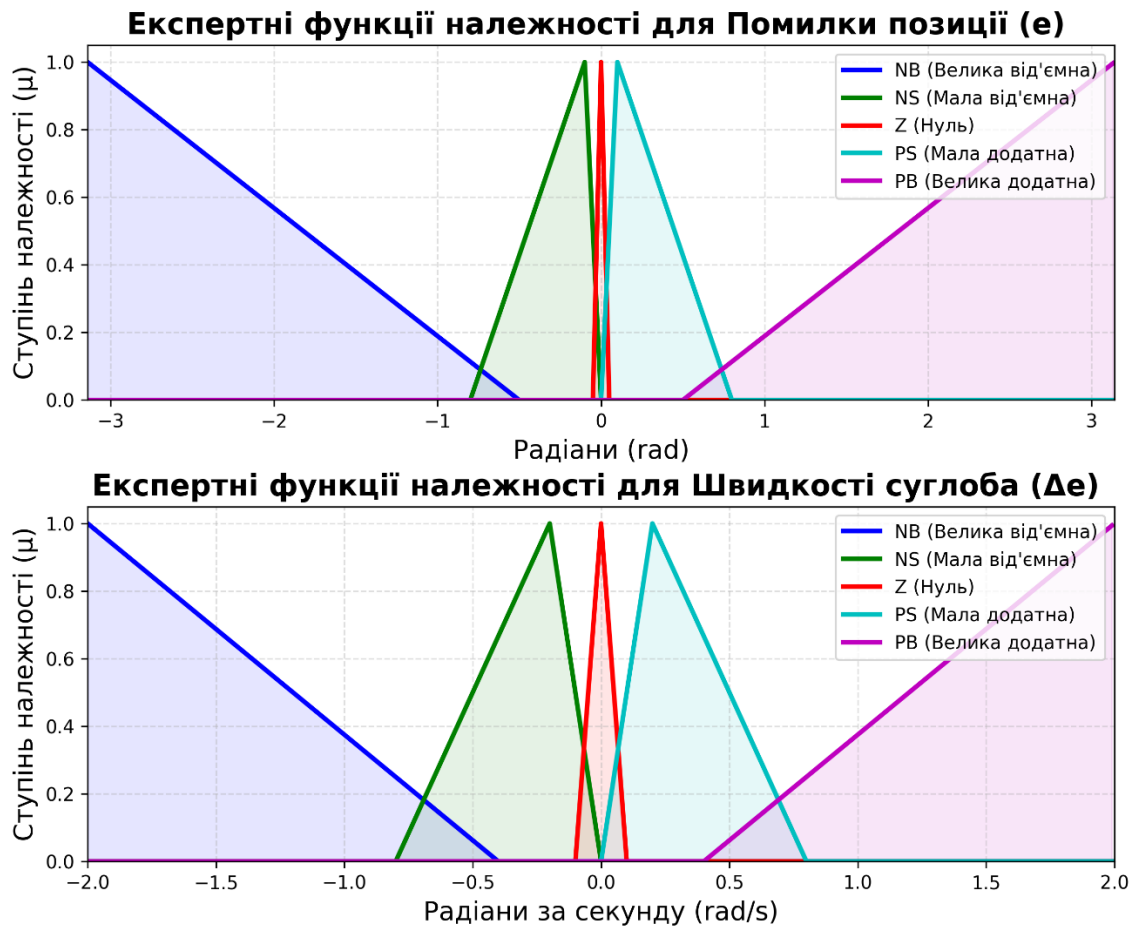


Рисунок 5.1 – Експертні функції належності для вхідних значень

На двох графіках зображено по п'ять трикутників, кожен з яких перетинається із сусіднім, завдяки чому і реалізовується механізм нечіткої логіки згідно з принципами якої певні значення можуть одночасно належати до декількох лінгвістичних термів. Також трикутники не є рівнобедреними, значення вершини так само було задано, як і межі термів, за допомогою евристичного методу. Аналогічний графік функції належності був створений і для вихідного значення керуючого сигналу та зображений на рисунку 5.2.

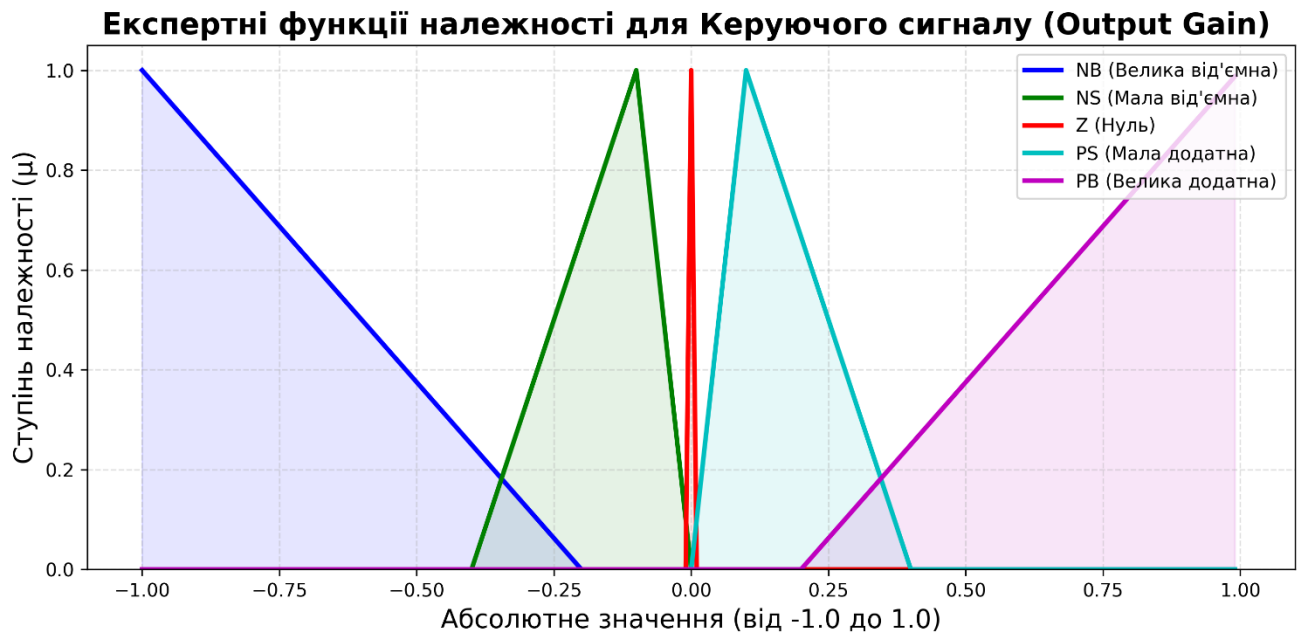


Рисунок 5.2 – Експертна функція належності для вихідного значення

Отриманий графік експертної функції належності для вихідного значення є подібним до двох графіків вхідних значень. Головною особливістю цього графіку є дуже вузький трикутник нуля. Це архітектурне рішення було прийнято для того, щоб уникнути явища перерегулювання та мікро тремтіння ланки.

Третім етапом створення моделі нечіткого контролера є синтез бази експертних правил. Кількість правил в цій базі обчислюється за формулою 5.1, тобто в цьому випадку 25 правил. Кожне правило описує яким має бути вихідний результат на основі лінгвістичних значень вхідних параметрів. Зображення повної бази правил продемонстровано на рисунку 5.3.

$$N = m^n = 5^2, \quad (5.1)$$

де m – кількість лінгвістичних термів кожної змінної;

n – кількість вхідних каналів керування.

	Velocity NB	Velocity NS	Velocity Z	Velocity PS	Velocity PB
Error NB	NB	NB	NB	NB	NB
Error NS	PS	Z	NS	NS	NB
Error Z	PB	PS	Z	NS	NB
Error PS	PB	PS	PS	Z	NS
Error PB	PB	PB	PB	PB	PB

Рисунок 5.3. База правил

Проаналізувавши синтезовану базу експертних правил можна зробити висновок, що вона орієнтована на забезпечення високої швидкодії робота-маніпулятора до цілі та екстрене зупинення біля цільової позиції з зупинкою прямо на цільовому значенні. Завдяки таким правилам система працюватиме швидко, точно та без проблем з тремтінням ланок. На основі отриманої бази правил було згенеровано нелінійну поверхню керування, яка візуалізує всю синтезовану нечітку логіку у тривимірному просторі. Цю поверхню зображено на рисунку 5.4.

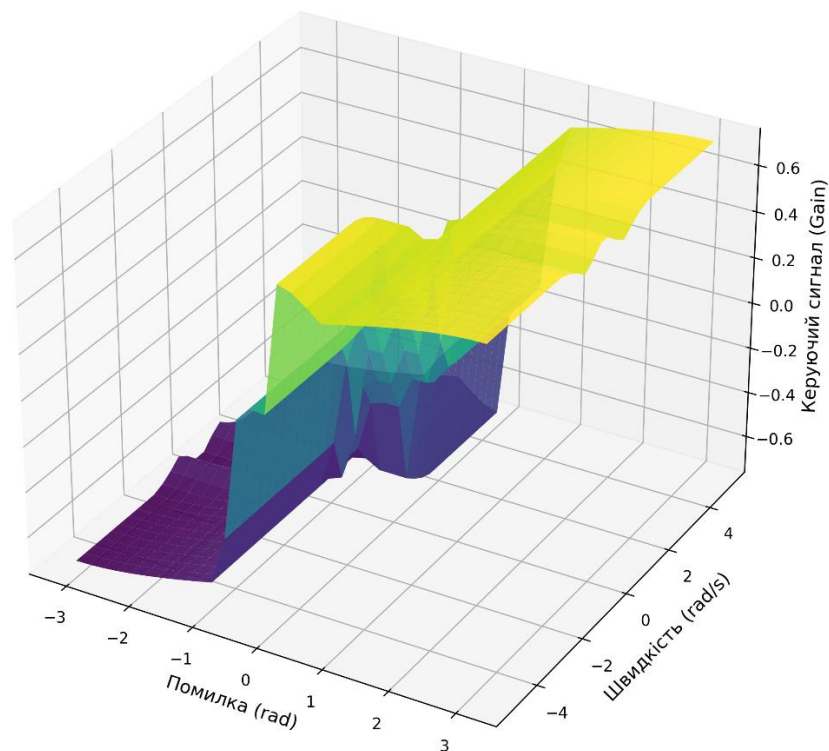


Рисунок 5.4. Нечітка поверхня керування

Отримана нелінійна поверхня керування розташована в тривимірному просторі, де виміри представлені помилкою позиції, поточної швидкості та керуючого сигналу. Після проведення візуального аналізу можна стверджувати, що поверхня складається з двох плато на краях графіків та різних висотах (значеннях керуючого сигналу), а також ці два плато об'єднуються одним схилом. Ці два плато характеризують стабільну максимальну швидкість, коли робот-маніпулятор знаходиться далеко від цілі. А схил між ними відображає зону дії вузького трикутника терми «Z» та означає миттєве зменшення швидкості при наближенні до заданої координати.

Наступним етапом став вибір інструментів для реалізації процесу нечіткого виведення. Як базовий алгоритм було обрано алгоритм Мамдані. Його головними перевагами є висока інтерпретабельність, що забезпечує прозорість правил через їх запис у інтуїтивно зрозумілій формі. Також вагомою перевагою є відповідність людському мисленню через використання в ньому лінгвістичних змінних, що зрозумілі будь якій людині. Крім того, цей алгоритм вважається універсальним та гнучким, що дозволяє адаптувати його під широке коло завдань та надає можливості для точного налаштування.

Першим кроком нечіткого виведення є процес фазифікації. Цей процес означає перетворення числових значень у значення нечіткої множини. Його головне завдання визначити ступінь належності отриманого на вході числа до кожної з лінгвістичних термів. В процесі фазифікації використовуються обрані раніше функції належності, завдяки чому визначення ступеня відбувається швидко та точно. Через це перетворення проходять абсолютно всі числа, що йдуть на вхід до нечіткого апарата.

Другим етапом є активація правил за допомогою логічного «І». Алгоритм застосовує отримані лінгвістичні значення до бази правил, щоб отримати вихідне значення. Оскільки активація використовує оператор перетину «І», то ступінь належності нового значення обирається як мінімальне значення серед ступенів

істинності антецедентів. За цим мінімальним рівнем система «зрізає» верхівку у відповідного вихідного трикутника керуючого сигналу.

Третім пунктом алгоритму нечіткого виведення є агрегація частин. Для отримання результатів регулятора використовується оператор максимуму. Він накладає всі «зрізані» фігури одна на одну і формує їхній загальний зовнішній контур. Тобто, якщо дві фігури взяті з одного лінгвістичного трикутника тоді обирається найбільший відсоток належності.

Останнім етапом є процес дефазифікації. Він є протилежністю до фазифікації і полягає в отриманні чіткого числа з нечіткого значення. Щоб отримати чітке значення з агрегованої фігури, обрано метод центру ваги. Цей метод є беззаперечним стандартом у робототехніці та системах керування рухом. Його перевага полягає в тому, що він враховує площу всіх без винятку активованих правил, завдяки чому зміна сигналу завжди відбувається плавно. Це критично важливо для даної роботи, оскільки дозволяє уникнути високочастотних вібрацій, перерегулювання та надмірних ударних навантажень на механічні редуктори суглобів під час динамічної зміни траєкторії.

5.2 Аналіз результатів моделювання нечіткого регулятора

Для перевірки ефективності розробленого нечіткого регулятора було проведено моделювання маніпулятора Puma 560 у середовищі Gazebo. Головною метою стало тестування здатності системи відпрацьовувати стрибкоподібну зміну координати та плавно переміщувати ланку в задане положення.

Щоб візуально зобразити результати керування роботом-маніпулятором за допомогою створеного нечіткого контролера було проведено тестування в середовищі Gazebo з використанням засобу тривимірної візуалізації Rviz та згенеровано графік перехідного процесу. Отримані дані дозволяють оцінити ключові показники якості системи керування, а саме перерегулювання, час

регулювання та статичну похибку. Спершу було побудовано графіки позиціювання ланки, що зображений на рисунку 5.5.

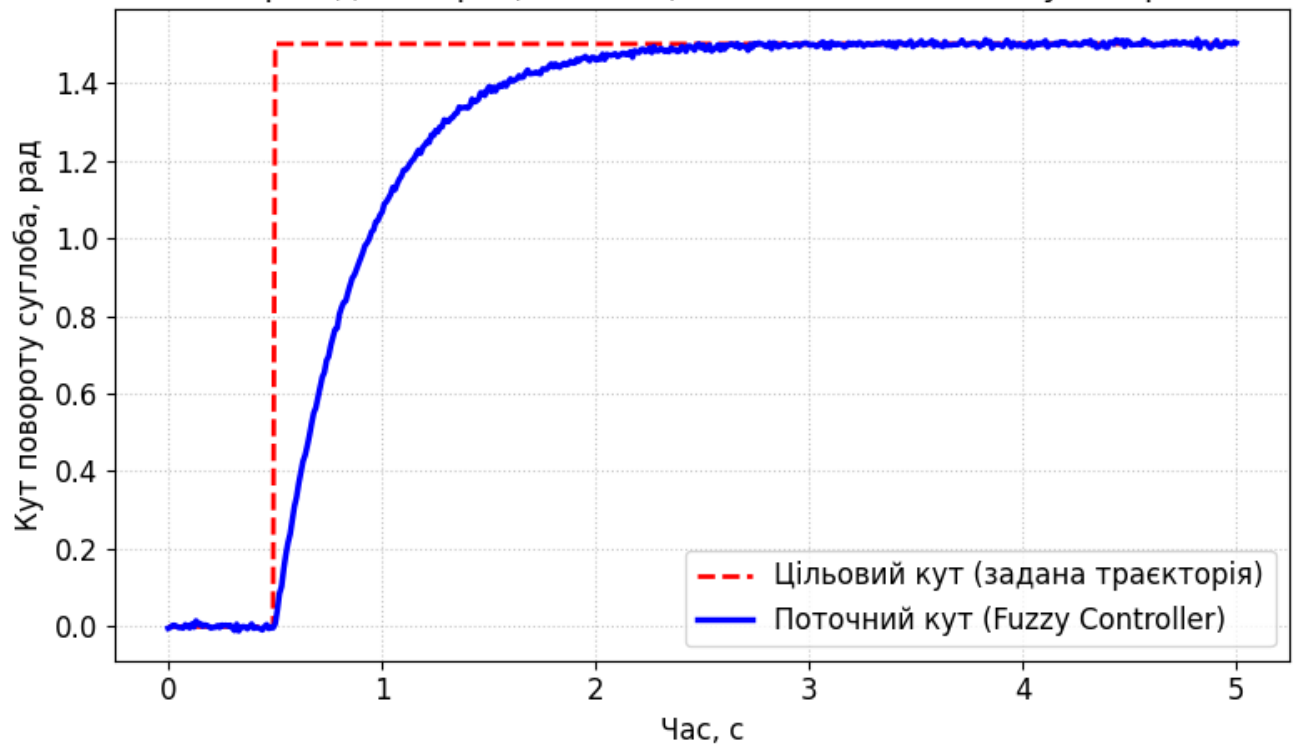


Рисунок 5.5. – Перехідний процес позиціювання ланки маніпулятора

Як видно з графіка на рисунку 5.5, розроблений регулятор успішно відпрацьовує заданий керуючий вплив. Крива поточного кута плавно наближається до цільової уставки без виникнення автоколивань та перерегулювання. Час переходного процесу становить близько двох секунд, що повністю задовольняє вимоги до динаміки просторового переміщення маніпулятора даного класу. Це підтверджує правильність обраної бази правил та коректність налаштування функцій належності.

Варто зазначити, що коли похибка зменшується і є досить маленькою, а швидкість її зміни є високою, то вся система суглобів робота починає різко гальмувати, завдяки чому ланка не пролітає повз цілі. Причиною такої поведінки є правильно розроблена база правил, яка й мала на меті забезпечити таке агресивне гальмування. Саме це дозволило отримати таку модель руху, за якої крива графіку підходить до цілі знизу і повністю уникає небезпечні перенавантаження на мотори

маніпулятора. Таким чином, результати симуляції доводять високу якість роботи запропонованої системи керування.

Також важливою особливістю лінії графіку є її сильні коливання на етапі усталеного режиму. Ці райони були представлені на тривимірній нечіткій поверхні у формі плато. Вони не є дефектами контролера, а характеризують вплив зовнішніх фізичних величини на саму систему. Розроблений нечіткий регулятор успішно утримує суглоб у межах зони статичної точності.

Другою частиною аналізу стало використання моделі робота-маніпулятора в симуляторі Gazebo для імітації справжнього руху робота з усіма параметричними невизначеностями, що існують в реальному світі. На рисунку 5.6 зображено позицію робота-маніпулятора в просторі у початковому стані та після переміщення у цільову точку.

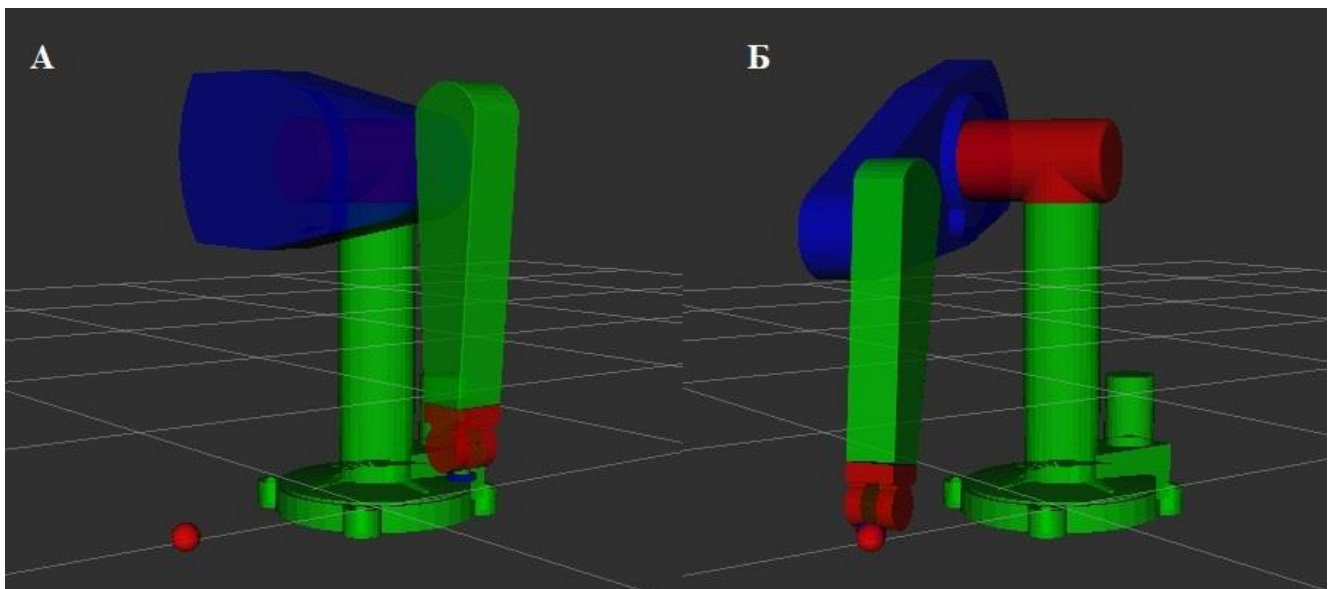


Рисунок 5.6. – Рух робота в просторі

Для створення рисунку було використано додатковий тривимірний візуалізатор під назвою Rviz, що є частиною фреймворку ROS 2. Він використовує всі дані з симулятора Gazebo та візуалізує їх у власному середовищі. Завдяки використанню Rviz з'явилась можливість створити червону сферу, яка стала маркером пункту призначення робота-маніпулятора.

Окрім того, використання візуалізатора Rviz дозволила проводити моніторинг зіткнень у реальному час та перевіряти досяжність робочої зони маніпулятора без ризику зламати реальне обладнання.

На першій половині зображення продемонстровано початкову позицію робота Puma 560 та цільову точку в просторі. На другій половині відображено кінцеву фазу руху та успішне досягнення червоної сфери кінцевим ефектором маніпулятором. Той факт, що контролер без математичної моделі самого маніпулятора зміг ідеально вийти на ціль, доводить його робастність

5.3 Порядок розгортання та керівництво користувача програмним комплексом

Оскільки розроблене програмне забезпечення є доволі комплексною системою, доцільно було б вказати необхідне програмне середовище для його запуск та описати алгоритм дій для розгортання створеного програмного комплексу для керування рухами робота.

Основні системні вимоги до програмного та апаратного оточення нечіткого регулятора наступні:

- Операційна система: Ubuntu 22.04 LTS;
- Фреймворк робототехніки: ROS 2 Humble;
- Апаратна платформа: ПК з використанням симулятора Gazebo для комплексного моделювання або одноплатний комп'ютер Raspberry Pi 4/5;
- Залежності: інтерпретатор Python та набір Python-бібліотек описаних у підпункті 3.2.

Далі наведено алгоритм дій для інсталяції та збирання розробленого проєкту на новій машині:

1. Підготовка робочого простору: `mkdir -p ~/ros2_ws/src`
2. Клонування до нього коду розробленого пакету: `git clone`

3. Компіляція пакету за допомогою команди `colcon build --packages-select puma560_fuzzy_control`

4. Оновлення системних змінних середовища командою `source install/setup.bash`

5. Запуск зібраного регулятора: `ros2 run puma560_fuzzy_control fuzzy_node`

При роботі з фреймворком ROS2 існує декілька корисних команд для моніторингу система та діагностики помилок. Перелік основних з них наведено нижче:

- Перевірка списку активних топіків: `ros2 topic list`
- Отримання поточної телеметрії: `ros2 topic echo /joint_states`
- Аналіз стану обчислювального вузла: `ros2 node info`

Таким чином, розроблене керівництво оператора та сформований покроковий алгоритм розгортання забезпечують швидку інсталяцію, збирання та запуск системи керування на будь-якій сумісній EOM або ARM-платі. Наявність уніфікованих команд діагностики в межах екосистеми ROS 2 дозволяє оператору здійснювати моніторинг цілісності інформаційних потоків у реальному часі, що спрощує процес відлагодження, технічного обслуговування та подальшої модернізації програмного комплексу.

Висновок до розділу 5

У розділі 5 було успішно синтезовано логічну модель нечіткого керування та проведено її верифікацію. Результати моделювання перехідних процесів підтвердили високу точність позиціонування, відсутність перерегулювання та загальну стабільність роботи розробленого програмного забезпечення. Окрім того, всі результати візуалізовано на відповідних графіках.

Першим етапом цього розділу був синтез логічної моделі керування. Під час цього кроку були обрані вхідні та вихідні змінні, функція належності, створені

лінгвістичні терми, синтезована база правил та обрано алгоритм нечіткого виведення. Були створені графіки для функцій належності, а також візуальне зображення таблиці бази правил. Окрім того, була створена поверхня нечіткого керування у тривимірному просторі.

Наступним кроком став аналіз роботи розробленого нечіткого регулятора. Для цього спочатку зображено рух маніпулятора на графіку перехідного процесу позиціювання ланки робота. Після детального розбору графіку виконано тестування програмного модуля для керування маніпулятором в симуляторі Gazebo з використанням візуалізатора RViz. Результати перевірки задовольнили базові критерії до створеної системи.

Останньою частиною цього розділу стала інструкція з розгортання та експлуатації програмного модуля. В ній вказано необхідний набір програмного та апаратного оточення. Також описано алгоритм дій для інсталяції та запуску нечіткого регулятора. В доповнення до того було надано перелік базових команд ROS 2 для аналізу результатів.

ВИСНОВКИ

Під час виконання цієї дипломної роботи було розроблено програмне забезпечення на основі нечіткої логіки для керування рухами робота-маніпулятора Puma 560 в умовах невизначеності.

Спочатку було проведено опрацювання літератури та інших джерел за темою роботи. Проаналізовані інші методи керування, такі як ПІД-регулятор, метод обчислювального моменту та нейронні мережі. Також обґрунтовано доцільність використання визначеного стеку засобів, а саме ROS 2, Gazebo та спеціалізовані бібліотеки для мови Python.

Після цього було спроектовано математичну модель нечіткого контролера. Були визначені вхідні та вихідні змінні, створені лінгвістичні терми, обрано трикутну функцію належності, синтезовано базу експертних правил, алгоритм Мамдані для нечіткого виведення та метод центру ваги для дефазифікації. Математичний апарат реалізовано програмно із використанням мови програмування Python та бібліотеки scikit-fuzzy.

Наступним етапом стало створення ROS 2 вузла для керування роботом з використанням розробленого нечіткого регулятора. Цей вузол працює з трьома топіками робота Puma 560. Він підписується на топіки `/joint_states` та `/puma/final_joint_target` для отримання даних про телеметрію та цільові кути. Та публікує у топік `/joint_trajectory_controller/joint_trajectory` команди для переміщення ланок робота.

В кінці роботи проведено комплексне тестування і налаштування розробленого програмного модуля в інтегрованому середовищі Gazebo спільно з інструментом 3D-візуалізації RViz. Після цього були створені аналітичні графіки, що підтвердили високу точність позиціювання ланок, повну відсутність явища перерегулювання та оптимальну швидкість досягнення цільової точки, що доводить стабільність розробленої системи.

Таким чином, після виконання цієї дипломної роботи було успішно створено програмне забезпечення для керування маніпулятором Puma 560 на основі нечіткої логіки в умовах невизначеності. Створений програмний комплекс демонструє високу ефективність у задачах просторового керування роботами-маніпуляторами. Створене програмне забезпечення надає можливість інтегрувати розроблені ROS-модулі як на локальних робочих станціях, так і на вбудованих мікропроцесорних платформах для керування реальними фізичними об'єктами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Craig J. J. Introduction to Robotics: Mechanics and Control. 3rd ed. Upper Saddle River : Pearson Prentice Hall, 2005. 400 p.
2. Lewis F. L., Jagannathan S., Yesildirek A. Neural Network Control of Robot Manipulators and Nonlinear Systems. London : Taylor & Francis, 1999. 480 p.
3. Slotine J.-J. E. The Robust Control of Robot Manipulators. The International Journal of Robotics Research. 1985. Vol. 4, no. 2. P. 49–64.
4. Ross T. J. Fuzzy Logic with Engineering Applications. 3rd ed. Chichester : John Wiley & Sons, 2010. 585 p.
5. Lynch K. M., Park F. C. Modern Robotics: Mechanics, Planning, and Control. Cambridge : Cambridge University Press, 2017. 540 p.
6. Liu J., Wang X. Advanced Sliding Mode Control for Mechanical Systems. Beijing : Springer, 2018. 300 p.
7. Ткачук В. І. Електромеханотроніка : підручник. Львів : Видавництво Львівської політехніки, 2021. 420 с.
8. Precup R.-E., David R.-C. Nature-Inspired Optimization Algorithms for Fuzzy Controlled Servo Systems. Oxford : Butterworth-Heinemann, 2019. 250 p.
9. Niku S. B. Introduction to Robotics: Analysis, Control, Applications. 3rd ed. New York : Wiley, 2020. 480 p.
10. Ghani N. M. A., Othman A., Hashim A. A. A., Nasir A. N. K. Comparative Analysis of PID and Fuzzy Logic Controllers for Position Control in Double-Link Robotic Manipulators. Journal of Intelligent Systems and Control. 2023. Vol. 2, No. 4. P. 183–196.
11. Литвин В. В., Пасічник В. В., Яцишин Ю. В. Інтелектуальні системи : підручник. Львів : «Новий Світ-2000», 2019. 406 с.
12. Chen G., Pham T. T. Introduction to Fuzzy Sets, Fuzzy Logic, and Fuzzy Control Systems. Boca Raton : CRC Press, 2019. 320 p.

ДОДАТОК А Лістинг розробленої системи

Fuzzy_controller.py

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

class FuzzyController:
    def __init__(self):
        error_angle = ctrl.Antecedent(np.arange(-3.14, 3.14, 0.01), 'error_angle')
        error_changes = ctrl.Antecedent(np.arange(-2, 2, 0.01), 'error_changes')
        position_gain = ctrl.Consequent(np.arange(-1.0, 1, 0.01), 'position_gain')

        error_angle['NB'] = fuzz.trimf(error_angle.universe, [-3.14, -3.14, -0.5])
        error_angle['NS'] = fuzz.trimf(error_angle.universe, [-0.8, -0.1, 0.0])
        error_angle['Z'] = fuzz.trimf(error_angle.universe, [-0.05, 0.0, 0.05])
        error_angle['PS'] = fuzz.trimf(error_angle.universe, [0.0, 0.1, 0.8])
        error_angle['PB'] = fuzz.trimf(error_angle.universe, [0.5, 3.14, 3.14])

        error_changes['NB'] = fuzz.trimf(error_changes.universe, [-2.0, -2.0, -
0.4])
        error_changes['NS'] = fuzz.trimf(error_changes.universe, [-0.8, -0.2,
0.0])
        error_changes['Z'] = fuzz.trimf(error_changes.universe, [-0.1, 0.0,
0.1])
        error_changes['PS'] = fuzz.trimf(error_changes.universe, [0.0, 0.2,
0.8])
        error_changes['PB'] = fuzz.trimf(error_changes.universe, [0.4, 2.0,
2.0])

        position_gain['NB'] = fuzz.trimf(position_gain.universe, [-1.0, -1.0, -
0.2])
        position_gain['NS'] = fuzz.trimf(position_gain.universe, [-0.4, -0.1,
0.0])
        position_gain['Z'] = fuzz.trimf(position_gain.universe, [-0.05, 0.0,
0.05])
        position_gain['PS'] = fuzz.trimf(position_gain.universe, [0.0, 0.1,
0.4])
        position_gain['PB'] = fuzz.trimf(position_gain.universe, [0.2, 1.0,
1.0])
```

```

rules = [
    ctrl.Rule(error_angle['NB'] & error_changes['NB'],
position_gain['NB']),
    ctrl.Rule(error_angle['NB'] & error_changes['NS'],
position_gain['NB']),
    ctrl.Rule(error_angle['NB'] & error_changes['Z'],
position_gain['NB']),
    ctrl.Rule(error_angle['NB'] & error_changes['PS'],
position_gain['NB']),
    ctrl.Rule(error_angle['NB'] & error_changes['PB'],
position_gain['NB']),

    ctrl.Rule(error_angle['NS'] & error_changes['NB'],
position_gain['PS']),
    ctrl.Rule(error_angle['NS'] & error_changes['NS'],
position_gain['Z']),
    ctrl.Rule(error_angle['NS'] & error_changes['Z'],
position_gain['NS']),
    ctrl.Rule(error_angle['NS'] & error_changes['PS'],
position_gain['NS']),
    ctrl.Rule(error_angle['NS'] & error_changes['PB'],
position_gain['NB']),

    ctrl.Rule(error_angle['Z'] & error_changes['NB'],
position_gain['PB']),
    ctrl.Rule(error_angle['Z'] & error_changes['NS'],
position_gain['PS']),
    ctrl.Rule(error_angle['Z'] & error_changes['Z'], position_gain['Z']),
    ctrl.Rule(error_angle['Z'] & error_changes['PS'],
position_gain['NS']),
    ctrl.Rule(error_angle['Z'] & error_changes['PB'],
position_gain['NB']),

    ctrl.Rule(error_angle['PS'] & error_changes['NB'],
position_gain['PB']),
    ctrl.Rule(error_angle['PS'] & error_changes['NS'],
position_gain['PS']),
    ctrl.Rule(error_angle['PS'] & error_changes['Z'],
position_gain['PS']),

```

```

        ctrl.Rule(error_angle['PS'] & error_changes['PS'],
position_gain['Z']),
        ctrl.Rule(error_angle['PS'] & error_changes['PB'],
position_gain['NS']),

        ctrl.Rule(error_angle['PB'] & error_changes['NB'],
position_gain['PB']),
        ctrl.Rule(error_angle['PB'] & error_changes['NS'],
position_gain['PB']),
        ctrl.Rule(error_angle['PB'] & error_changes['Z'],
position_gain['PB']),
        ctrl.Rule(error_angle['PB'] & error_changes['PS'],
position_gain['PB']),
        ctrl.Rule(error_angle['PB'] & error_changes['PB'],
position_gain['PB']),
    ]

    # == 4. Creating model ==
    tipping_ctrl = ctrl.ControlSystem(rules)
    self.tipping = ctrl.ControlSystemSimulation(tipping_ctrl)

    def compute(self, error_angle, error_changes):
        self.tipping.input['error_angle'] = error_angle
        self.tipping.input['error_changes'] = error_changes
        self.tipping.compute()

        return self.tipping.output['position_gain']

```

Fuzzy_node.py

```

import rclpy
from rclpy.node import Node

from sensor_msgs.msg import JointState
from std_msgs.msg import Float64MultiArray
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
from builtin_interfaces.msg import Duration
from puma560_fuzzy_control.fuzzy_controller import FuzzyController

class FuzzyNode(Node):

```

```

def __init__(self):
    super().__init__('fuzzy_node')
    self.joint_names = ['j1', 'j2', 'j3', 'j4', 'j5', 'j6']

    self.current_positions = [0.0] * 6
    self.current_velocities = [0.0] * 6
    self.target_positions = [0.0] * 6

    self.start_time = self.get_clock().now()
    self.first_target_received = False #Zero-Target Jump Error

    self.controllers = [FuzzyController() for _ in range(6)]

    self.publisher = self.create_publisher(JointTrajectory,
'/joint_trajectory_controller/joint_trajectory', 10)

    self.sub_state = self.create_subscription(
        JointState, '/joint_states', self.state_callback, 10)
    self.sub_target = self.create_subscription(
        JointState, '/puma/final_joint_target', self.target_callback, 10)

    timer_period = 0.01
    self.timer = self.create_timer(timer_period, self.timer_callback)
    self.get_logger().info("The Puma 560 kinematic fuzzy controller has been
successfully launched!")


def state_callback(self, message: JointState):
    for i, name in enumerate(message.name):
        if name in self.joint_names:
            idx = self.joint_names.index(name)
            try:
                self.current_positions[idx] = message.position[i]
                self.current_velocities[idx] = message.velocity[i]
            except IndexError:
                pass


def target_callback(self, message: JointState):
    msg_time = rclpy.time.Time.from_msg(message.header.stamp)
    if msg_time < self.start_time:

```

```

        return

    self.first_target_received = True
    for i, name in enumerate(message.name):
        if name in self.joint_names:
            idx = self.joint_names.index(name)
            self.target_positions[idx] = message.position[i]

def timer_callback(self):
    if not self.first_target_received:
        return

    trajectory = JointTrajectory()
    trajectory.joint_names = self.joint_names

    destination_point = JointTrajectoryPoint()
    destination_point.positions = [0.0] * 6
    gain = 0.2
    for i in range(6):
        error = self.target_positions[i] - self.current_positions[i]
        if abs(error) < 0.005:
            destination_point.positions[i] = self.current_positions[i]
            continue
        output_value = self.controllers[i].compute(error,
self.current_velocities[i])
        new_position = self.current_positions[i] + (output_value * gain)
        destination_point.positions[i] = new_position

    destination_point.time_from_start = Duration(sec=0, nanosec=1000000000)

    trajectory.points.append(destination_point)
    self.publisher.publish(trajectory)

    #debugger
    self.get_logger().info(f"Target: {self.target_positions[0]:.2f} | Current:
{self.current_positions[0]:.2f} | Error: {self.target_positions[0] -
self.current_positions[0]:.2f}")

def main(args=None):
    rclpy.init(args=args)
    node = FuzzyNode()

```

```
try:
    rclpy.spin(node)
except KeyboardInterrupt:
    pass
finally:
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```



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

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

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ НЕЧІТКОЇ ЛОГІКИ ДЛЯ КЕРУВАННЯ РУХАМИ РОБОТА- МАНІПУЛЯТОРА В УМОВАХ НЕВИЗНАЧЕНОСТІ

виконав: студент групи ТВ-22 Турніцький Владислав Анатолійович

керівник: професор, д.т.н., професор Барабаш Олег Володимирович

2026

Актуальність теми

Актуальність теми: Сучасна промислова робототехніка стрімко еволюціонує в напрямку, де критичними вимогами стають адаптивність та універсальність маніпуляторів. Класичні методи керування втрачають ефективність в умовах невизначеності, коли змінюється маса вантажу або зношуються механізми, що призводить до вібрацій та втрати точності.

Впровадження інтелектуальних систем на базі нечіткої логіки дозволяє компенсувати ці нелінійні збурення без необхідності переналаштування системи. Такий підхід забезпечує стабільну роботу та високу точність позиціонування навіть при змінних навантаженнях.



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

2/14

Постановка задачі

Мета роботи

Розробка програмного забезпечення на основі нечіткої логіки для керування рухами робота-маніпулятора в умовах невизначеності

Завдання:

1. Провести аналіз існуючих рішень
2. Спроекувати архітектуру системи
3. Розробити UML-діаграму класів програмного модуля керування
4. Створити UML-діаграму послідовності взаємодії компонентів системи
5. Розробити програмний модуль нечіткого контролера
6. Синтезувати бази експертних правил
7. Провести тестування системи та отримати результати



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

3/14

Аналіз предметної області

Існуюче рішення	Перевага	Недолік
Класичний ПД-регулятор	Простота реалізації та низькі вимоги до комп'ютера	Чутливість до зміни конфігурації та виникає перерегулювання при збуреннях
Обчислювальний момент	Враховує динаміку робота та висока точність у стабільних умовах	Вимагає точної аналітичної моделі та втрачає точність через люфти та завади
Штучні нейронні мережі	Здатність до навчання та компенсація складних нелінійностей	Висока обчислювальна складність та явище «чорної скриньки»
Нечітка логіка	Робота без аналітичної моделі та висока робастність	Складність експертного налаштування функцій належності на етапі проектування



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

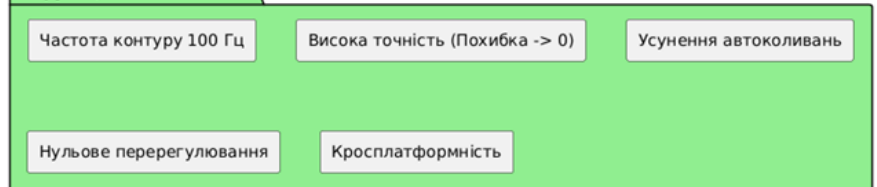
4/14

Проектування системи

Функціональні вимоги

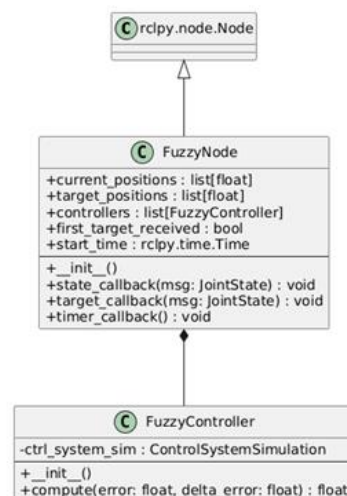


Нефункціональні вимоги



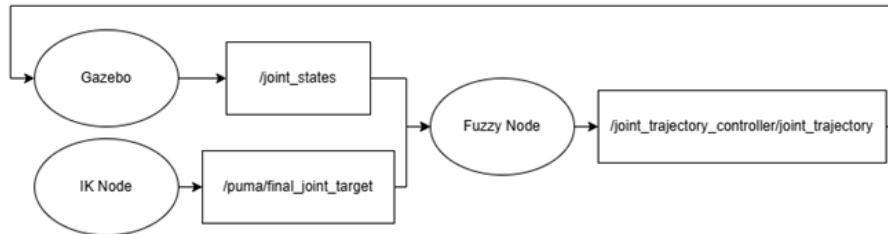
Розробка діаграми класів

UML-діаграма класів



Проектування архітектури системи

Схема інформаційної взаємодії вузлів та топіків у середовищі ROS2

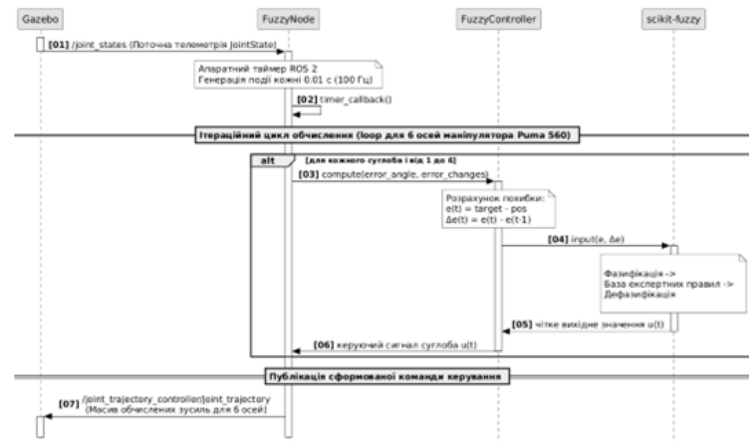


Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

7/14

Розробка діаграми послідовності процесів

UML-діаграма послідовності



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

8/14

Реалізація системи

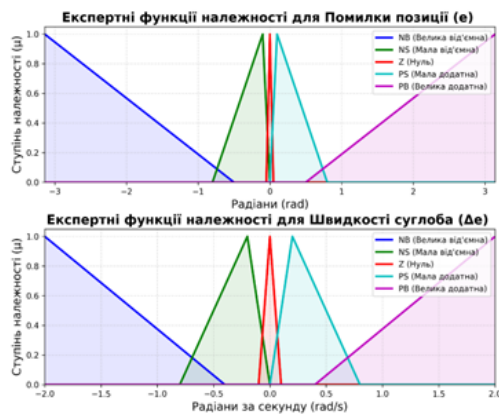


Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

9/14

Розробка програмного модуля

Експертні функції належності для вхідних значень



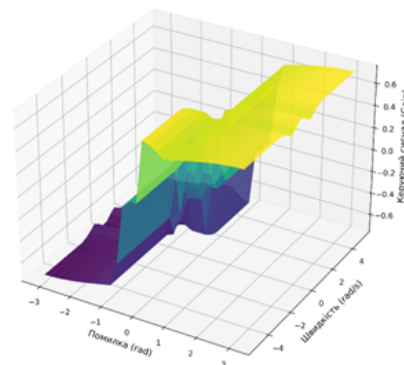
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

10/14

Синтез бази експертних правил

База експертних правил

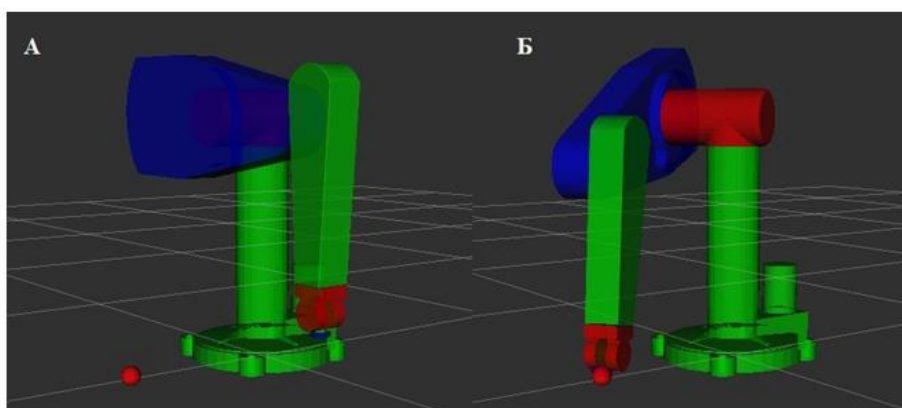
	Velocity NB	Velocity NS	Velocity Z	Velocity PS	Velocity PB
Error NB	NB	NB	NB	NB	NB
Error NS	PS	Z	NS	NS	NB
Error Z	PB	PS	Z	NS	NB
Error PS	PB	PS	PS	Z	NS
Error PB	PB	PB	PB	PB	PB



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

11/14

Тестування системи

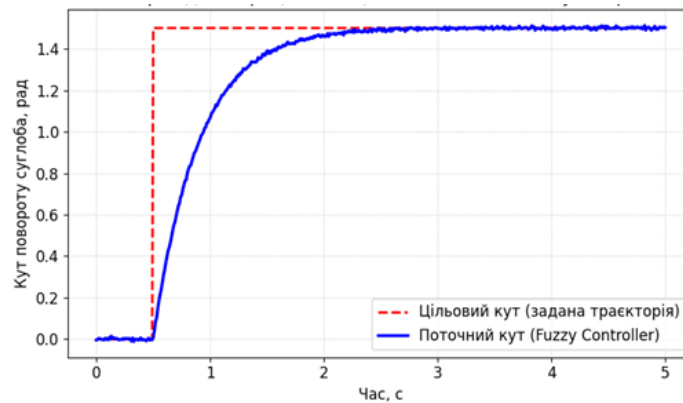


Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

12/14

Результати тестування системи

Перехідний процес позиціювання ланки маніпулятора



Висновки

У дипломній роботі повністю досягнуто поставлену мету та успішно **вирішено всі науково-практичні завдання:**

1. Проаналізовано існуючі рішення та обрано найкращий
2. Спроектовано архітектуру нечіткого контролера
3. Розроблено UML-діаграму класів програмного модуля керування
4. Створено UML-діаграму послідовності взаємодії компонентів системи
5. Реалізовано програмний модуль нечіткого контролера
6. Синтезовано повну базу експертних правил
7. Проведено тестування та отримання результатів система відповідає визначеним вимогам

