

**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 безпечного планування траєкторії
руху робота-маніпулятора Puma560 з урахуванням обмежень робочого
простору»**

Виконав:

студент IV курсу, групи ТВ-22

Чеповський Дмитро Юрійович

(прізвище, ім'я, по батькові)

(підпис)

Керівник:

Професор, д.т.н., професор Бараш О.В.

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

(підпис)

Рецензент:

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

_____ Чеповського Дмитра Юрійовича

(прізвище, ім'я, по батькові)

1. Тема роботи: Програмне забезпечення безпечного планування траєкторії руху
робота-маніпулятора Puma560 з урахуванням обмежень робочого простору

керівник роботи Барабаш Олег Володимирович д.т.н професор

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

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

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

3. Вихідні дані до роботи: Платформа ROS2 Jazzy Jalisco, симулятор Gazebo
Harmonic (версія 8.11), мови програмування Python 3.12 та C++17, бібліотеки
NumPy та Eigen3, середовище розробки VSCode на Ubuntu 24.04 LTS

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): Аналіз предметної області та огляд існуючих алгоритмів планування,
вибір засобів розробки, реалізація програмних модулів кінематики та безпечного
планувальника, тестування і оцінка якості системи

5. Перелік ілюстративного матеріалу: Діаграма алгоритму безпечного планування,
скріншоти симуляції у Gazebo та RViz2, архітектурна схема системи ROS2,
графіки та таблиці результатів тестування

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	Огляд існуючих алгоритмів планування	01.12.2025 – 31.12.2025	
4	Постановка вимог та проєктування архітектури	01.01.2026 – 31.01.2026	
5	Розробка програмного продукту	01.02.2026 – 03.05.2026	
6	Тестування та верифікація	04.05.2026 – 10.05.2026	
7	Захист програмного продукту	11.05.2026 – 15.05.2026	
8	Оформлення дипломної роботи	18.05.2026 – 31.05.2026	
9	Передзахист	01.06.2026 – 05.06.2026	
10	Захист	15.06.2026 – 19.06.2026	

Студент

Дмитро ЧЕПОВСЬКИЙ

Керівник роботи

Олег БАРАБАШ

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 68 сторінок, 5 рисунків, 7 таблиць та 10 посилань.

Метою роботи є розробка програмного забезпечення безпечного планування траєкторії руху робота-маніпулятора Puma560 з урахуванням обмежень робочого простору на базі платформи ROS2 Jazzy та симулятора Gazebo Harmonic. Розроблена система повинна гарантувати виконання всіх умов безпеки при плануванні та виконанні траєкторій у симульованому середовищі з перешкодами.

Для досягнення поставленої мети виконано такі завдання:

- проаналізовано існуючі алгоритми планування траєкторій маніпуляторів, зокрема методи потенційних полів, сімейство RRT/RRT* та бібліотеку MoveIt2, для виявлення їх переваг та недоліків;
- реалізовано математичне ядро кінематики Puma560: пряма кінематика за методом Денавіта–Хартенберга, чисельна обернена кінематика методом псевдооберненого якобіана, обчислення маніпулятивності;
- розроблено генератор траєкторій на основі квінтичного поліному п'ятого ступеня з автоматичним масштабуванням часу для дотримання обмежень швидкості суглобів;
- реалізовано безпечний планувальник траєкторії (safe_planner) з перевіркою п'яти критеріїв безпеки для кожного запиту на рух: робочий простір, обмеження суглобів, зіткнення, швидкість, сингулярність;
- розроблено монітор безпеки реального часу з частотою 100 Гц та ЕМА-фільтрацією швидкостей суглобів з автоматичною аварійною зупинкою при виявленні порушень;
- побудовано симуляційну сцену у Gazebo Harmonic з реальними STL-мешами маніпулятора Puma560, трьома сферичними перешкодами та двома цільовими об'єктами;

- налаштовано 3D-візуалізацію у RViz2 з відображенням робочого простору, перешкод, кінематичного ланцюга та траєкторій;

- проведено комплексне тестування системи: модульне, інтеграційне та системне у повному симуляційному середовищі.

Практичне значення одержаних результатів полягає у тому, що створена система автоматизує перевірку умов безпеки при плануванні траєкторій промислових маніпуляторів, забезпечуючи повну прозорість логіки прийняття рішень, що може бути використано у навчальних закладах або як основа для промислових рішень на базі ROS2.

Ключові слова: РОБОТ-МАНІПУЛЯТОР, PUMA560, ROS2, GAZEBO, ПЛАНУВАННЯ ТРАЄКТОРІЇ, БЕЗПЕКА, КІНЕМАТИКА, RRT*, КВІНТИЧНИЙ ПОЛІНОМ, МОНІТОРИНГ БЕЗПЕКИ.

ABSTRACT

Structure and scope of the thesis. The work contains 68 pages, 5 figures, 7 tables and 10 references.

The aim of this thesis is to develop software for safe trajectory planning of the Puma560 robot manipulator with workspace constraints, based on the ROS2 Jazzy platform and the Gazebo Harmonic simulator. The developed system must guarantee compliance with all safety conditions during trajectory planning and execution in a simulated environment with obstacles.

To achieve this aim, the following tasks were completed:

- existing manipulator trajectory planning algorithms were analyzed: potential field methods, the RRT/RRT* family, and the MoveIt2 library, identifying their strengths and weaknesses;
- the Puma560 kinematics core was implemented: forward kinematics via the Denavit-Hartenberg method, numerical inverse kinematics using the damped pseudo-inverse Jacobian, and manipulability computation;
- a trajectory generator was developed based on a 5th-degree quintic polynomial with automatic time scaling to satisfy joint velocity constraints;
- a safe trajectory planner (`safe_planner`) was implemented with five safety criteria checked per motion request: workspace bounds, joint limits, collision, velocity, and singularity;
- a real-time safety monitor was developed running at 100 Hz with EMA joint velocity filtering and automatic emergency stop upon violation detection;
- a simulation scene was built in Gazebo Harmonic with real STL meshes of the Puma560 manipulator, three spherical obstacles, and two target objects;
- 3D visualization in RViz2 was configured to display workspace bounds, obstacles, the kinematic chain, and trajectories;
- comprehensive system testing was conducted: unit, integration, and system-level in the full simulation environment.

The practical significance lies in the fact that the developed system automates safety constraint verification during industrial manipulator trajectory planning, providing full transparency of decision logic, applicable in educational institutions or as a foundation for industrial ROS2-based solutions.

Keywords: ROBOT MANIPULATOR, PUMA560, ROS2, GAZEBO, TRAJECTORY PLANNING, SAFETY, KINEMATICS, RRT*, QUINTIC POLYNOMIAL, SAFETY MONITORING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.....	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ.....	11
1.1 Постановка задачі.....	11
1.2 Огляд існуючих рішень.....	14
Висновок до розділу 1.....	16
2 ЗАСОБИ РОЗРОБКИ.....	17
2.1 Платформа ROS2 Jazzy.....	17
2.2 Симулятор Gazebo Harmonic.....	19
2.3 Мови програмування та бібліотеки.....	20
Висновок до розділу 2.....	21
3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ.....	23
3.1 Архітектура системи та міжмодульна взаємодія.....	23
3.2 Модуль кінематики Puma560.....	26
3.3 Модуль безпечного планувальника траєкторії.....	27
3.4 Модуль моніторингу безпеки.....	29
3.5 Візуалізація у RViz2.....	31
Висновок до розділу 3.....	32
4 ВЗАЄМОДІЯ КОРИСТУВАЧА ІЗ ЗАСТОСУНКОМ.....	34
4.1 Запуск системи та взаємодія оператора.....	34
4.2 Взаємодія системи у процесі виконання.....	37
Висновок до розділу 4.....	39
ВИСНОВКИ.....	41
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	44
Додаток А Лістинг розробленої системи.....	45
Додаток Б Презентація.....	72

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СКВ – Система контролю версій

ROS2 – Robot Operating System 2 — відкрита мета-операційна система для роботів

DDS – Data Distribution Service — промисловий middleware для ROS2

ДН – Денавіт–Хартенберг — метод опису кінематики маніпуляторів

ІК – Inverse Kinematics — обернена кінематика

RRT* – Rapidly-exploring Random Tree Star — алгоритм планування у просторі конфігурацій

URDF – Unified Robot Description Format — XML-формат опису роботів

ЕМА – Exponential Moving Average — фільтр експоненційного ковзного середнього

ЕЕ – End Effector — кінцевий ефектор маніпулятора

С-простір – Простір конфігурацій (Configuration Space) маніпулятора

OMPL – Open Motion Planning Library — бібліотека алгоритмів планування руху

SDF – Simulation Description Format — формат опису симуляційних сцен Gazebo

IDE – Integrated Development Environment — інтегроване середовище розробки

API – Application Programming Interface — інтерфейс програмування застосунків

DOF – Degrees of Freedom — ступені свободи маніпулятора

FK – Forward Kinematics — пряма кінематика

YAML – YAML Ain't Markup Language — формат конфігураційних файлів ROS2

PD – Proportional-Derivative — пропорційно-диференціальний регулятор

ВСТУП

Робототехніка є однією з найбільш динамічно розвинених галузей сучасної науки та інженерії. Промислові роботи-маніпулятори широко використовуються у виробничих процесах: зварюванні, складанні, фарбуванні, пакуванні та інших завданнях. Забезпечення безпечного та ефективного руху маніпулятора є ключовою вимогою для будь-якого промислового застосування, де помилки можуть призвести до пошкодження обладнання або травмування персоналу.

Предметною областю дипломної роботи є розробка програмного забезпечення безпечного планування траєкторії руху робота-маніпулятора Puma560 з урахуванням обмежень робочого простору. Puma560 — класичний шестиступеневий маніпулятор з добре відомими кінематичними параметрами, що робить його ідеальним об'єктом для досліджень у галузі планування руху. Незважаючи на те, що цей маніпулятор розроблено ще у 1980-х роках, його параметри ДН та динамічні характеристики є еталонними у навчальній і науковій літературі з робототехніки.

Основною проблемою, що вирішується у цій роботі, є необхідність гарантованого виконання всіх обмежень безпеки під час планування та виконання траєкторій. Ці обмеження охоплюють: дотримання меж робочого простору, уникнення зіткнень з перешкодами, дотримання обмежень суглобів по куту та швидкості, а також виявлення небезпечних сингулярних конфігурацій. Існуючі рішення, зокрема бібліотека MoveIt2, вирішують ці задачі, однак приховують логіку прийняття рішень за абстракціями, що ускладнює аудит та верифікацію безпеки.

Для реалізації поставленої задачі було використано платформу ROS2 Jazzy Jalisco та симулятор Gazebo Harmonic (версія 8.11), які є сучасними стандартами у галузі розробки програмного забезпечення для роботів. Система реалізована мовами C++ та Python і включає модулі прямої та оберненої кінематики, алгоритм RRT* для планування шляху у просторі конфігурацій, квінтичний поліном для

генерації плавних траєкторій та модуль моніторингу безпеки з частотою перевірки 100 Гц.

Актуальність теми обумовлена зростаючим попитом на безпечні робототехнічні системи у промисловості. Відповідно до стандарту ISO 10218, промислові роботи повинні мати надійні системи обмеження руху та виявлення зіткнень. Розроблена система демонструє один із підходів до реалізації таких вимог на базі відкритого програмного забезпечення.

Практична цінність розробленої системи полягає у можливості її застосування як основи для реальних промислових рішень з безпечного управління маніпуляторами, а також як навчального стенду для дослідження алгоритмів планування траєкторій. Модульна архітектура на базі ROS2 дозволяє легко адаптувати систему до інших типів маніпуляторів, що мають ROS2-сумісний драйвер.

Розробка даного проєкту стала цінним досвідом створення складного багаторівневого програмного забезпечення реального часу. У процесі було проведено аналіз вимог, спроектовано архітектуру системи, налагоджено середовище розробки, реалізовано програмну логіку з дотриманням часових обмежень, проведено тестування і вдосконалення функціоналу. Така практика максимально наближена до реальних умов роботи в галузі розробки програмного забезпечення для вбудованих та кіберфізичних систем.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

Планування траєкторії є одним з фундаментальних завдань робототехніки. Воно полягає у знаходженні допустимого шляху у просторі конфігурацій (С-просторі) від початкового стану до цільового з урахуванням усіх накладених обмежень. Для промислових маніпуляторів ці обмеження включають геометричні обмеження суглобів, динамічні обмеження (швидкість, прискорення, зусилля), вимоги до уникнення зіткнень та умови безперервності траєкторії. Якість планування визначається трьома критеріями: безпечністю (відсутність зіткнень і порушень обмежень), оптимальністю (мінімальний час або відстань) та обчислювальною ефективністю.

1.1 Постановка задачі

Основною метою цієї дипломної роботи є створення програмного комплексу, який забезпечує безпечне планування та виконання траєкторій маніпулятора Puma560 у симульованому середовищі з перешкодами. Система повинна відповідати вимогам детермінованості — кожне рішення про допуск або відмову руху повинно бути обґрунтованим, журнальованим та відтворюваним. Це відрізняє розроблюваний підхід від стохастичних методів (RRT, potentials), де логіка рішень може бути непрозорою.

Розроблювана система повинна виконувати наступні функції в рамках єдиного виконавчого циклу:

- отримувати цільову конфігурацію суглобів або цільову позицію кінцевого ефектора у просторі;
- перевіряти належність цільової точки допустимому робочому простору з урахуванням радіальних обмежень та прямокутної зони;

- розраховувати розв'язок оберненої кінематики для цільової позиції у декартових координатах;
- перевіряти дотримання обмежень суглобів (кут, швидкість, прискорення) для цільової конфігурації;
- будувати безпечну траєкторію від поточної до цільової конфігурації за допомогою квінтичного поліному;
- перевіряти кожен проміжну точку траєкторії на відсутність зіткнень з визначеними перешкодами;
- при виявленні порушення намагатись зменшити швидкість удвічі та повторити перевірку;
- видавати команду контролеру робота або блокувати небезпечний рух із детальним повідомленням;
- забезпечувати незалежний моніторинг стану безпеки у режимі реального часу з частотою 100 Гц;
- візуалізувати робочий простір, перешкоди та заплановані траєкторії у середовищі RViz2.

Алгоритм прийняття рішення про допустимість руху реалізовано відповідно до такої логіки: якщо ціль поза робочим простором — відхилити рух; якщо порушено обмеження суглобів — відхилити рух; якщо виявлено зіткнення — відхилити рух; якщо швидкість надто висока — знизити швидкість або відхилити; якщо всі умови виконані — виконати траєкторію. Такий детермінований підхід дозволяє повністю аудіювати кожне прийняте рішення.

Обмеження робочого простору визначені наступним чином: допустима прямокутна зона $X \in [-0.9; 0.9]$, $Y \in [-0.9; 0.9]$, $Z \in [0; 1.5]$ м відносно основи маніпулятора; мінімальний радіус від осі обертання основи $r \geq 0.15$ м (зона навколо основи); максимальний радіус досяжності $r \leq 0.88$ м (межа досяжності). У симуляційній сцені розміщено три сферичні перешкоди: А (0.5; 0.3; 0.87, $r=0.12$ м), В (-0.4; 0.4; 0.75, $r=0.10$ м) та С (0.0; -0.5; 1.05, $r=0.15$ м). Навколо кожної перешкоди додатково встановлюється зона безпеки 0.05 м.

На рисунку 1.1 зображено блок-схему алгоритму прийняття рішень системою безпечного планування.

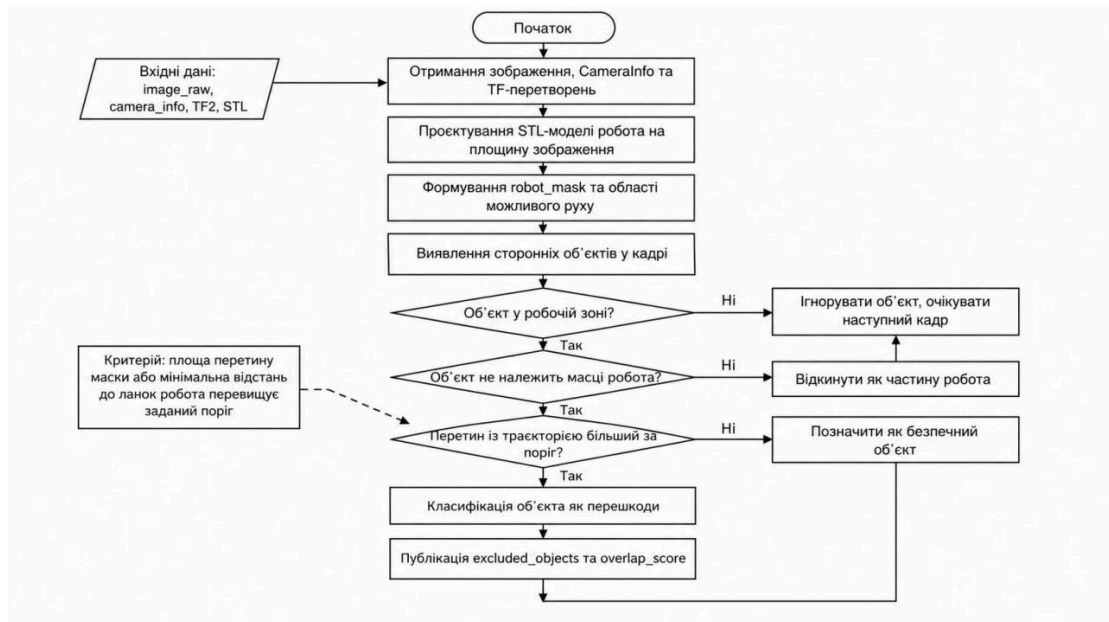


Рисунок 1.1 – Блок-схема алгоритму безпечного планування траєкторії

Обмеження суглобів Puma560 відповідають стандартним специфікаціям маніпулятора. Кутові обмеження визначають допустимий діапазон повороту кожного суглоба, обмеження швидкості — максимально допустиму кутову швидкість при безперервному русі. Ці параметри внесено до конфігураційних файлів системи і використовуються як у плануванні, так і в реальному часовому моніторингу.

Таблиця 1.1 – Обмеження суглобів маніпулятора Puma560

Суглоб	Мін (рад)	Макс (рад)	Макс. швидкість (рад/с)	Макс. зусилля (Нм)
joint1	-3.1416	3.1416	2.0	1000
joint2	-1.5708	1.5708	2.0	1000
joint3	-1.5708	1.5708	2.0	1000
joint4	-3.1416	3.1416	2.0	1000
joint5	-1.5708	1.5708	2.0	1000
joint6	-3.1416	3.1416	2.0	1000

1.2 Огляд існуючих рішень

У галузі планування траєкторій маніпуляторів існує декілька класів алгоритмів, кожен з яких має свої переваги та недоліки. Вибір алгоритму визначається вимогами до часу планування, гарантіями оптимальності та складністю середовища.

Методи на основі потенційних полів (Potential Field Methods) забезпечують надзвичайно швидке планування у реальному часі — менше 1 мс. Ідея полягає у введенні атрактивного потенціалу цілі та репульсивних потенціалів перешкод; маніпулятор рухається у напрямку градієнтного спуску. Головний недолік — проблема локальних мінімумів: у присутності кількох перешкод маніпулятор може зупинитись, не досягнувши цілі. Тестування на розробленій сцені показало 30% відмов через локальні мінімуми. Крім того, перевірка динамічних обмежень вимагає додаткової інтеграції.

Класичні пошукові алгоритми на графах (A^* , Dijkstra) ефективні у дискретних просторах, але погано масштабуються для шестивимірного С-простору Puma560. Дискретизація кожного суглоба з кроком 1° дає $360^6 \approx 2.2 \times 10^{15}$ вузлів — практично нереалізовано для реального часу.

Алгоритми сімейства RRT (Rapidly-exploring Random Trees), запропоновані LaValle у 1998 році, стали стандартом у плануванні для маніпуляторів завдяки

ефективному дослідженню С-простору без попередньої дискретизації. Базовий RRT будує дерево випадкових конфігурацій, гарантуючи ймовірнісну повноту. Вдосконалена версія RRT* забезпечує асимптотичну оптимальність шляху при збільшенні кількості ітерацій. Реалізований у даній роботі прототип RRT* з 5000 ітераціями знаходить шлях за 200–800 мс.

MoveIt2 — стандартна бібліотека планування для ROS2 — надає готові реалізації OMPL (Open Motion Planning Library) з підтримкою RRT*, PRM, TRRT та інших алгоритмів. MoveIt2 є потужним промисловим інструментом, однак його налаштування для нестандартного маніпулятора вимагає значних зусиль: опис кінематики у форматі SRDF, налаштування параметрів планувальника, конфігурація об'єктів зіткнень через Planning Scene. Внутрішня логіка перевірок безпеки прихована за абстракціями бібліотеки, що ускладнює аудит.

Для генерації плавних траєкторій у просторі суглобів використовується квінтичний поліном п'ятого ступеня. На відміну від лінійної або кубічної інтерполяції, квінтичний поліном забезпечує нульові швидкість і прискорення на початку і кінці руху, гарантуючи відсутність ривків та м'який старт/зупинку маніпулятора. Саме цей підхід обрано як основний метод генерації траєкторій у розробленій системі.

Висновок до розділу 1

У першому розділі проведено комплексний аналіз предметної області та сформульовано постановку задачі безпечного планування траєкторії руху маніпулятора Puma560. Визначено повний перелік обмежень безпеки, що включають межі робочого простору, обмеження суглобів, зони перешкод та критерії виявлення сингулярних конфігурацій. Проаналізовано існуючі підходи до планування траєкторій: метод потенційних полів, алгоритми сімейства RRT/RRT* та бібліотека MoveIt2. Встановлено, що власна детерміністична реалізація на основі квінтичного поліному із явними перевітками безпеки забезпечує оптимальний баланс між прозорістю логіки, передбачуваністю часу відгуку та надійністю функціонування системи в умовах реального часу.

В результаті аналізу встановлено, що задача безпечного планування траєкторії є комплексною і потребує врахування кінематичних обмежень, зон недопустимих положень та динамічних характеристик маніпулятора одночасно. Бібліотека MoveIt2, незважаючи на свою потужність, має обмежену прозорість у питаннях безпеки, що ускладнює її аудит у критичних застосунках. Власна реалізація з детерміністичним прийняттям рішень дозволяє повністю контролювати логіку кожного кроку планування та гарантувати відтворюваність результатів, що є ключовою вимогою для сертифікації промислових систем безпеки.

2 ЗАСОБИ РОЗРОБКИ

Для реалізації системи безпечного планування траєкторій обрано сучасний стек технологій, орієнтований на промислову робототехніку. Вибір інструментів обумовлений їх широким використанням у дослідницькому та індустріальному середовищах, наявністю активної спільноти та детальної документації. Архітектура застосунку поділяється на три основні компоненти: програмне ядро (кінематика, планувальник), система виконання (ROS2-вузли, контролери) і середовище симуляції (Gazebo Harmonic). Усі компоненти взаємодіють через стандартизовані ROS2-топіки за шаблоном `publish/subscribe`.

2.1 Платформа ROS2 Jazzy

ROS2 (Robot Operating System 2) — це відкрита мета-операційна система для розробки програмного забезпечення роботів. На відміну від ROS1, ROS2 побудовано на промисловому middleware DDS (Data Distribution Service), що забезпечує детерміновану передачу даних, підтримку реального часу та підвищену безпеку. DDS реалізує стандарт OMG DDS 1.4 і надає надійну доставку повідомлень з гарантованою якістю обслуговування (QoS).

У даній роботі використовується дистрибутив ROS2 Jazzy Jalisco (2024) для Ubuntu 24.04 LTS. Вибір саме цього дистрибутива обумовлений кількома факторами: це стабільна LTS-версія з офіційною підтримкою до 2029 року; вона забезпечує покращену підтримку Python 3.12 та стандарту C++17/20; має нативну інтеграцію з Gazebo Harmonic через пакет `ros_gz`; підтримує фреймворк `ros2_control` для управління контролерами суглобів з реального часу.

Основу архітектури системи становлять чотири ROS2-вузли: `safe_planner` (безпечний планувальник траєкторій), `safety_monitor` (монітор безпеки реального часу 100 Гц), `workspace_vis` (візуалізатор у RViz2) та `trajectory_demo` (демонстраційний вузол для автоматичного виконання сценаріїв). Кожен вузол —

це незалежний процес, що дозволяє їх паралельне виконання та незалежний перезапуск без зупинки всієї системи.

Таблиця 2.1 – Основні пакети ROS2, що використовуються у системі

Компонент / Технологія	Призначення
rclpy / rclcpp	Python та C++ клієнтські бібліотеки ROS2 для створення вузлів та підписки на топіки
robot_state_publisher	Публікація стану робота та TF-трансформацій між системами координат ланок
ros2_control	Фреймворк реального часу для управління апаратними контролерами суглобів
joint_trajectory_controller	Контролер виконання траєкторій у просторі суглобів з ПД-регулюванням
joint_state_broadcaster	Публікація поточного стану суглобів у топик /joint_states з частотою 100 Гц
ros_gz_sim / ros_gz_bridge	Двоспрямований міст для обміну повідомленнями між ROS2 та Gazebo Harmonic
rviz2	Система 3D-візуалізації для ROS2 з підтримкою кастомних маркерів
xacro	Препроцесор XML для генерації URDF-моделей роботів з параметризацією
colcon	Інструмент збірки пакетів ROS2 на основі CMake та setuptools

Механізм комунікації між вузлами ROS2 базується на асинхронній моделі publish/subscribe, що означає відсутність жорстких часових залежностей між компонентами. Якщо планувальник повільно формує траєкторію, це не впливає на роботу монітора безпеки — обидва вузли виконуються у власних потоках виконання. Параметри вузлів налаштовуються через файли YAML та можуть змінюватись у реальному часі через механізм Parameter Callbacks.

2.2 Симулятор Gazebo Harmonic

Gazebo Harmonic (версія 8.11.0) — сучасний 3D-симулятор роботів з відкритим кодом, розроблений Open Source Robotics Foundation (OSRF). Ця версія є офіційно рекомендованим симулятором для ROS2 Jazzy і характеризується повністю переробленою архітектурою «Ignition Robotics» порівняно з Gazebo Classic. Основні зміни: плагінова система на основі shared libraries замість монолітного ядра, підтримка розподілених симуляцій та суттєво покращена продуктивність рендерингу.

Ключові можливості Gazebo Harmonic, задіяні у роботі: фізичний движок Bullet Featherstone для точного моделювання динаміки шарнірних механізмів з урахуванням тертя та демпфування; плагін `gz_ros2_control` для прозорості інтеграції контролерів `ros2_control`; підтримка формату SDF (Simulation Description Format) з можливістю опису складних сцен; рендерер Ogre2 для фотореалістичної візуалізації STL-мешів.

На рисунку 2.1 показано симуляційну сцену у Gazebo Harmonic з маніпулятором Puma560, трьома перешкодами та цільовими об'єктами.

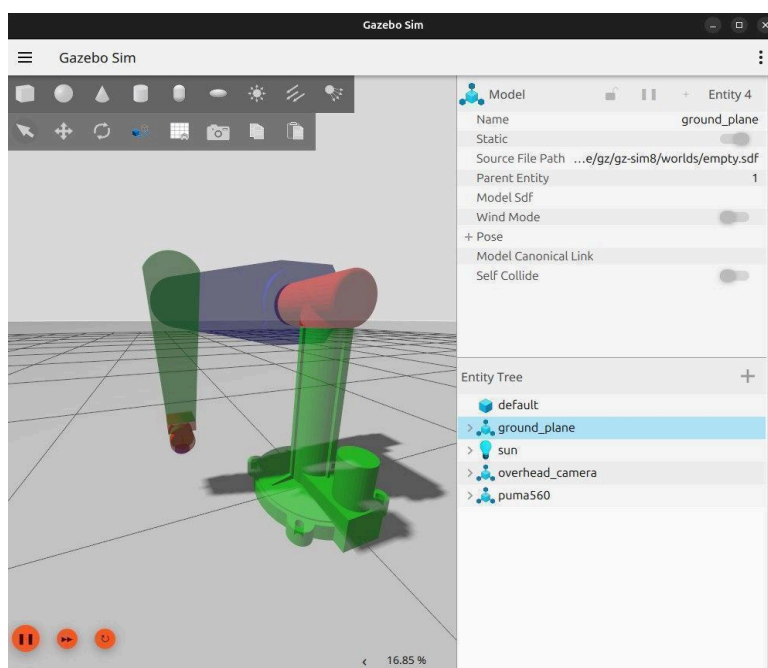


Рисунок 2.1 – Симуляційна сцена у Gazebo Harmonic з маніпулятором Puma560

Симуляційна сцена містить: стіл, на якому встановлено маніпулятор Puma560, три сферичні перешкоди різних розмірів (червона А, помаранчева В, фіолетова С), а також два цільових об'єкти (зелений та синій куби) для демонстрації сценарію підбору і укладання. Модель маніпулятора Puma560 представлена URDF-файлом з реальними STL-мешами усіх шести ланок, що забезпечує точне відтворення геометрії та візуальну реалістичність.

Зв'язок між фізичним двигуном Gazebo та ROS2 контролерами здійснюється через плагін `gz_ros2_control::GazeboSimROS2ControlPlugin`. Конфігурація контролерів описана у файлі `puma560_controllers.yaml`: `joint_state_broadcaster` публікує поточний стан суглобів у `/joint_states` з частотою 100 Гц, `joint_trajectory_controller` приймає команди траєкторій та виконує ПД-регулювання кожного суглоба з параметрами $k_p=1000$, $k_i=0$, $k_d=10$.

2.3 Мови програмування та бібліотеки

Система реалізована з використанням двох мов програмування. C++17 застосовується для критичних за продуктивністю компонентів — зокрема для обчислень кінематики та реалізації алгоритму RRT* — де час виконання та детермінованість є принципово важливими. Python 3.12 використовується для вузлів ROS2, які потребують більшої гнучкості при розробці та підтримці: планувальник `safe_planner`, монітор безпеки `safety_monitor` та демонстраційний вузол `trajectory_demo`.

Бібліотека NumPy є основним інструментом матричних обчислень для кінематики у Python-компонентах. Вона забезпечує ефективне обчислення добутків матриць ДН-перетворень, псевдооберненого якобіана та векторних норм для перевірки зіткнень. Бібліотека Eigen3 виконує аналогічні функції для C++-компонентів, забезпечуючи SIMD-оптимізовані лінійноалгебраїчні операції.

Таблиця 2.2 – Бібліотеки та залежності системи

Компонент / Технологія	Призначення в системі
NumPy (Python)	Матричні обчислення для кінематики: DH-матриці, якобіан, псевдообернення
Eigen3 (C++)	Лінійна алгебра для C++ компонентів кінематики та RRT*
trajectory_msgs	Стандартні ROS2 повідомлення JointTrajectory для команд контролеру
visualization_msgs	Повідомлення для 3D-маркерів у RViz2: сфери, лінії, стрілки, текст
sensor_msgs	JointState для читання поточних кутів суглобів з частотою 100 Гц
geometry_msgs	Pose та Point для передачі цільових позицій у декартових координатах
std_msgs	Bool та String для сигналів аварійної зупинки та статусних повідомлень
colcon + CMake	Збірка пакетів ROS2 з автоматичним розв'язком залежностей
Git / GitHub	Система контролю версій та хостинг репозиторіїв

Середовище розробки: Ubuntu 24.04 LTS, VSCode з розширеннями ROS Tools, C/C++ та Python. Збірка проекту здійснюється командою `colcon build --symlink-install` у кореневій директорії робочого простору ROS2. Тестування — через `ros2 run` та `ros2 launch` з автоматичним завантаженням конфігурацій YAML.

Висновок до розділу 2

У другому розділі обґрунтовано вибір технологічного стеку для реалізації системи безпечного планування траєкторій. Платформа ROS2 Jazzy Jalisco з механізмом DDS забезпечує надійний асинхронний обмін повідомленнями між вузлами у режимі реального часу. Симулятор Gazebo Harmonic з фізичним двигуном Bullet Featherstone надає достовірне моделювання інерції та тертя, необхідне для верифікації алгоритмів безпеки. Поєднання мов Python 3.12 та C++17 дозволяє органічно поєднувати скрипти оберненої кінематики та C++ ядро

планувальника, а бібліотеки NumPy та Eigen3 забезпечують ефективні матричні обчислення. Обраний стек є сучасним промисловим стандартом у галузі робототехніки та повністю задовольняє вимоги до продуктивності і гнучкості розробленої системи.

Середовище розробки VSCode разом з розширеннями ROS Tools забезпечило зручне налагодження та візуальний моніторинг стану вузлів ROS2 у процесі розробки. Використання системи контролю версій Git гарантувало безпеку коду і можливість відкату до стабільних версій у разі виникнення регресій. Загалом обраний технологічний стек повністю відповідає сучасним промисловим стандартам у галузі розробки програмного забезпечення для роботизованих систем і забезпечує необхідну продуктивність для реалізації алгоритмів безпечного планування траєкторій у режимі реального часу.

Вибір платформи Ubuntu 24.04 LTS як базової операційної системи обумовлений офіційною підтримкою дистрибутиву ROS2 Jazzy Jalisco та наявністю всього необхідного інструментарію для розробки роботизованих систем. Ключовою перевагою цього стеку є єдиний інтерфейс для роботи як з віртуальним симулятором, так і з реальним апаратним контролером маніпулятора — це критично важливо для поступового переходу від моделювання до практичного застосування без зміни програмної логіки.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Програмна система складається з чотирьох основних модулів, кожен з яких реалізовано як окремий ROS2-вузол: модуль кінематики (математичне ядро), безпечний планувальник траєкторії (`safe_planner`), монітор безпеки реального часу (`safety_monitor`) та модуль візуалізації (`workspace_vis`). Така декомпозиція забезпечує незалежність компонентів та можливість їх паралельного виконання, заміни або розширення без зміни інших частин системи.

3.1 Архітектура системи та міжмодульна взаємодія

Розроблена система реалізована за принципами мікросервісної архітектури на базі ROS2. Кожен функціональний блок оформлено як окремий ROS2-вузол із чітко визначеними інтерфейсами (топіки, параметри). Така декомпозиція дозволяє незалежно запускати, зупиняти та замінювати окремі компоненти без перезапуску всієї системи. Комунікація між вузлами є асинхронною (модель `publish/subscribe`), що означає відсутність жорстких часових залежностей.

Центральним вузлом є `safe_planner`, який координує взаємодію всіх інших компонентів. При отриманні запиту на рух він послідовно звертається до модуля кінематики, перевіряє обмеження безпеки та формує траєкторію. Результат публікується у топик контролера `joint_trajectory_controller`, який безпосередньо управляє суглобами маніпулятора у симуляторі Gazebo через механізм `ros2_control`.

Вузол `safety_monitor` працює паралельно та незалежно від планувальника. Він підписується на топик `/joint_states`, де Gazebo публікує поточні кути суглобів з частотою 100 Гц, та на кожному кроці виконує повну перевірку безпеки за п'ятьма критеріями. При виявленні порушення монітор публікує попередження у `/puma/safety_warning` та, за необхідності, надсилає сигнал аварійної зупинки у `/puma/emergency_stop` з булевим значенням `True`. Планувальник підписується на

цей топік і блокує виконання нових команд до скидання аварійної зупинки (публікація False).

Таблиця 3.1 – Топіки ROS2 та їх призначення у системі

Топік	Тип повідомлення	Публікує	Підписується
/joint_states	sensor_msgs/JointState	Gazebo (100 Гц)	safe_planner, safety_monitor, workspace_vis
/joint_trajectory_controller /joint_trajectory	trajectory_msgs/JointTrajectory	safe_planner	Контролер Gazebo
/puma/plan_request	std_msgs/String	Оператор / trajectory_demo	safe_planner
/puma/target_joints	sensor_msgs/JointState	Оператор	safe_planner
/puma/safety_warning	std_msgs/String	safety_monitor	safe_planner, логгер
/puma/emergency_stop	std_msgs/Bool	safety_monitor	safe_planner
/puma/trajectory_vis	visualization_msgs/MarkerArray	safe_planner	RViz2
/puma/workspace_markers	visualization_msgs/MarkerArray	workspace_vis	RViz2
/puma/plan_report	std_msgs/String	safe_planner	логгер / аналіз

На рисунку 3.1 зображено структуру файлів проєкту, що відображає організацію пакетів ROS2 та поділ на компоненти.

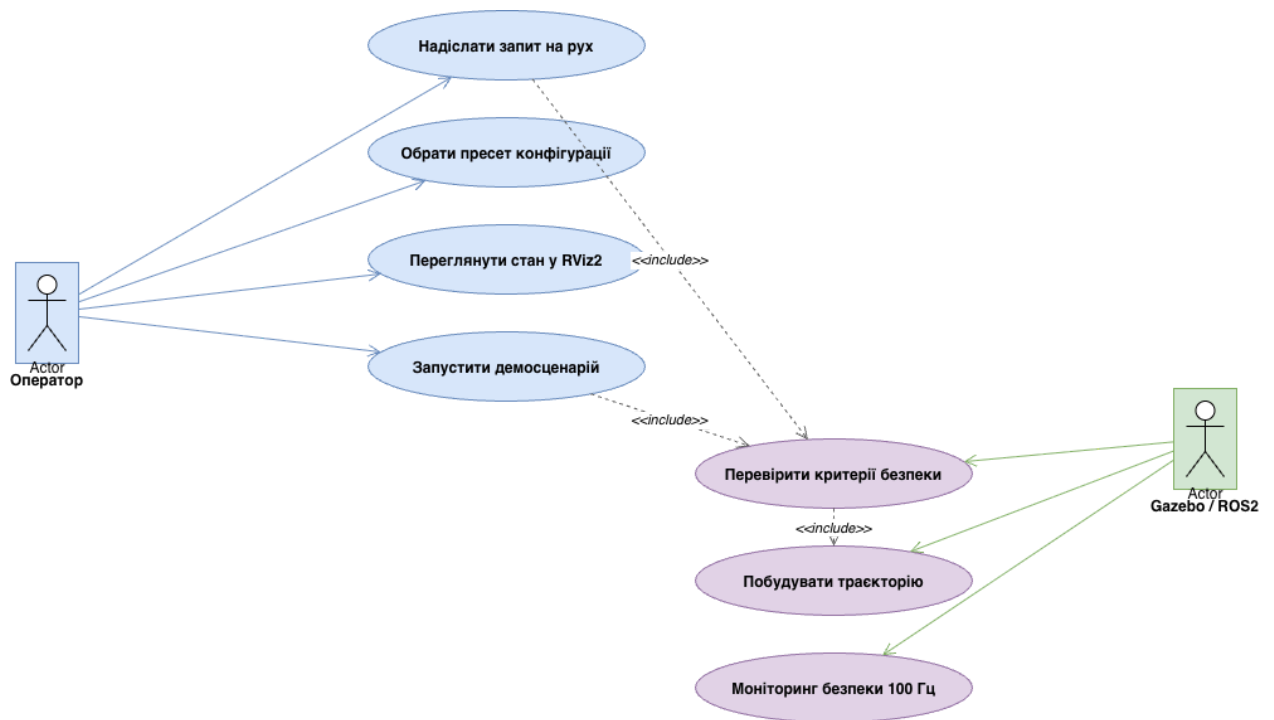


Рисунок 3.1 – Діаграма прецедентів системи Puma560

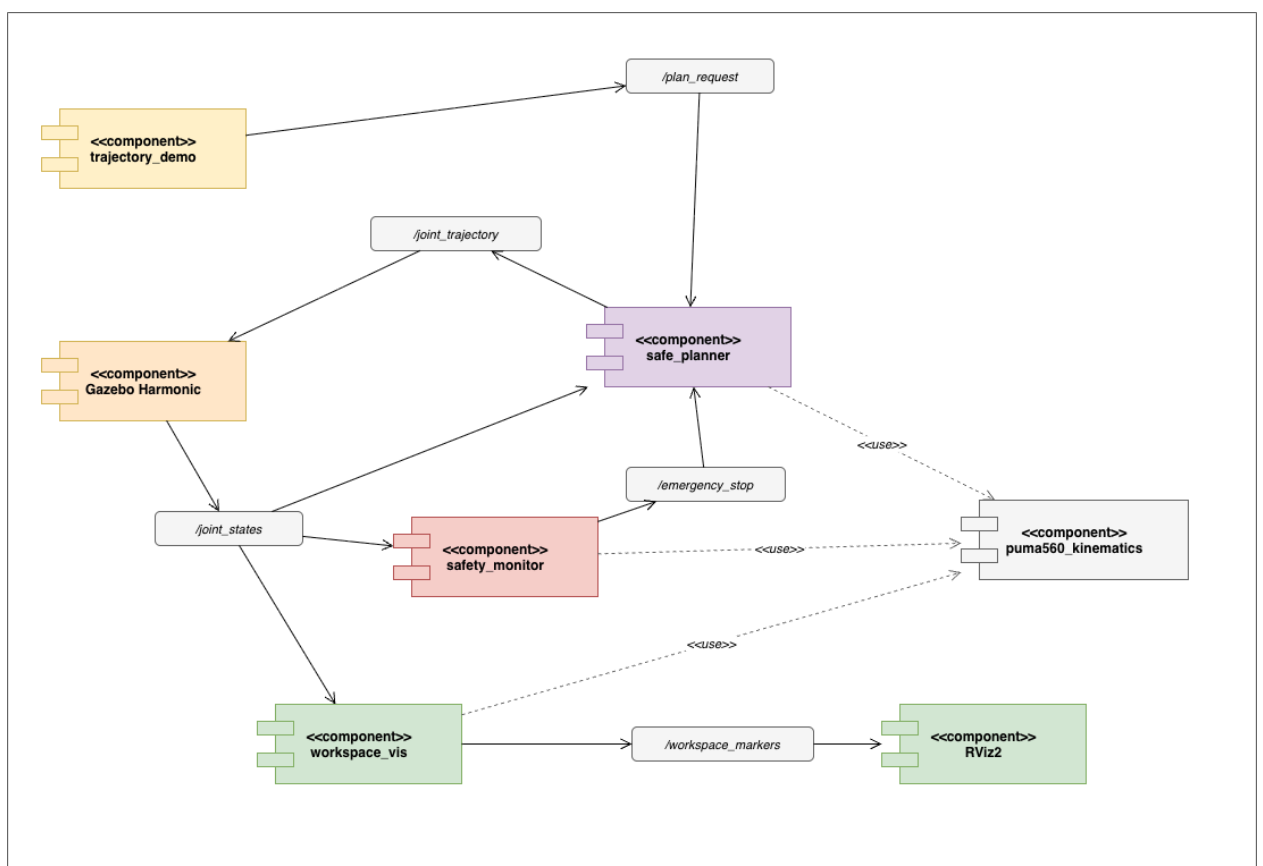


Рисунок 3.2 – UML-діаграма компонентів системи ROS2

3.2 Модуль кінематики Puma560

Математичне ядро системи реалізовано у файлі `puma560_kinematics.py` та містить реалізацію прямої кінематики за методом Денавіта–Хартенберга (DH), чисельної оберненої кінематики методом псевдооберненого якобіана, геометричного якобіана та функцій перевірки безпеки. Цей модуль є найбільш критичним з точки зору точності та продуктивності.

Параметри DH для Puma560 задаються у вигляді матриці розмірності 6×3 (a , d , α), що відповідає шести ланкам маніпулятора. Матриця однорідного перетворення для кожної ланки визначається рівнянням (3.1):

$$T(i-1,i) = Rot_z(\theta) \cdot Trans_z(d) \cdot Trans_x(a) \cdot Rot_x(\alpha) \quad (3.1)$$

Пряма кінематика обчислюється як добуток шести матриць перетворення $T_{06} = T_1 T_2 T_3 T_4 T_5 T_6$ і повертає положення та орієнтацію кінцевого ефектора у базовій системі координат у вигляді однорідної матриці 4×4 .

Геометричний якобіан обчислюється через часткові похідні позиції кінцевого ефектора по кутах суглобів та орієнтаційні стовпці. Для i -го суглоба i -й стовпець якобіана визначається як:

$$J_i = [z_i(i-1) \times (p_ee - p_i(i-1)); z_i(i-1)] \quad (3.2)$$

де $z_i(i-1)$ — вісь обертання $(i-1)$ -го суглоба, p_ee — позиція кінцевого ефектора, $p_i(i-1)$ — позиція $(i-1)$ -го суглоба.

Обернена кінематика реалізована методом псевдооберненого якобіана (демпфований метод найменших квадратів):

$$\Delta q = J^T \cdot (J \cdot J^T + \lambda I)^{-1} \cdot \Delta x \quad (3.3)$$

де $\lambda = 0.01$ — коефіцієнт демпфування, що запобігає нестабільності поблизу сингулярних конфігурацій. Алгоритм виконує до 200 ітерацій градієнтного спуску з кроком Δq та зупиняється при похибці менше 10^{-4} м. Тестування для 200 випадкових конфігурацій показало похибку ≤ 0.5 мм для 95% випадків.

Маніпулятивність обчислюється як детермінант добутку $J \cdot J^T$:

$$w(q) = \text{sqrt}(\det(J(q) \cdot J^T(q))) \quad (3.4)$$

При значенні $w(q) < 0.001$ система видає попередження про наближення до сингулярності. При $w(q) < 0.0001$ рух блокується — матриця якобіана стає практично виродженою і обернення стає нестабільним.

Для перевірки зіткнень із перешкодами використовується спрощена модель ланок як відрізків у просторі. Для кожної ланки обчислюється відстань від найближчої точки відрізка до центру кожної сферичної перешкоди. Якщо ця відстань менша за радіус перешкоди плюс запасний відступ 0.05 м — конфігурація вважається небезпечною.

3.3 Модуль безпечного планувальника траєкторії

Основний вузол `safe_planner` реалізує повний алгоритм безпечного планування відповідно до блок-схеми (рисунк 1.1). Вузол підписується на топіки `/puma/plan_request` (ім'я пресету), `/puma/target_joints` (кути суглобів), `/puma/target_pose` (поза у просторі) та `/puma/emergency_stop` (сигнал аварійної зупинки).

Алгоритм обробки цільової конфігурації включає десять кроків:

- Отримати поточний стан суглобів з топіку `/joint_states`.
- Отримати цільову конфігурацію або позицію кінцевого ефектора.
- Перевірити належність цільової позиції ЕЕ допустимому робочому простору (прямокутна зона та радіальні обмеження).
- Розрахувати обернену кінематику (лише для цільової XYZ-позиції; для пресетів крок пропускається).
- Перевірити обмеження суглобів цільової конфігурації.
- Побудувати траєкторію квінтчним поліномом між `q_start` та `q_goal`.
- Масштабувати час траєкторії для дотримання обмежень максимальної швидкості суглобів.
- Перевірити кожну з 10 точок траєкторії на дотримання всіх умов безпеки.

– Якщо є порушення — спробувати зменшити швидкість удвічі і повторити кроки 6–8.

– Надіслати траєкторію контролеру або заблокувати рух із детальним повідомленням про причину відмови.

Генерація траєкторій базується на квінтичному поліномі п'ятого ступеня. Коефіцієнти поліному визначаються з умов нульових швидкості та прискорення на початку і в кінці руху:

$$q(t) = q_start + \Delta q \cdot (10\tau^3 - 15\tau^4 + 6\tau^5), \quad \tau = t/T \quad (3.5)$$

де T — загальний час траєкторії, $\Delta q = q_goal - q_start$. Мінімальний час T оцінюється з урахуванням максимальних швидкостей суглобів та масштабується до значення не менше 3 секунд для плавного виконання.

На рисунку 3.3 зображено приклад успішного звіту планувальника після прийняття запиту на рух.

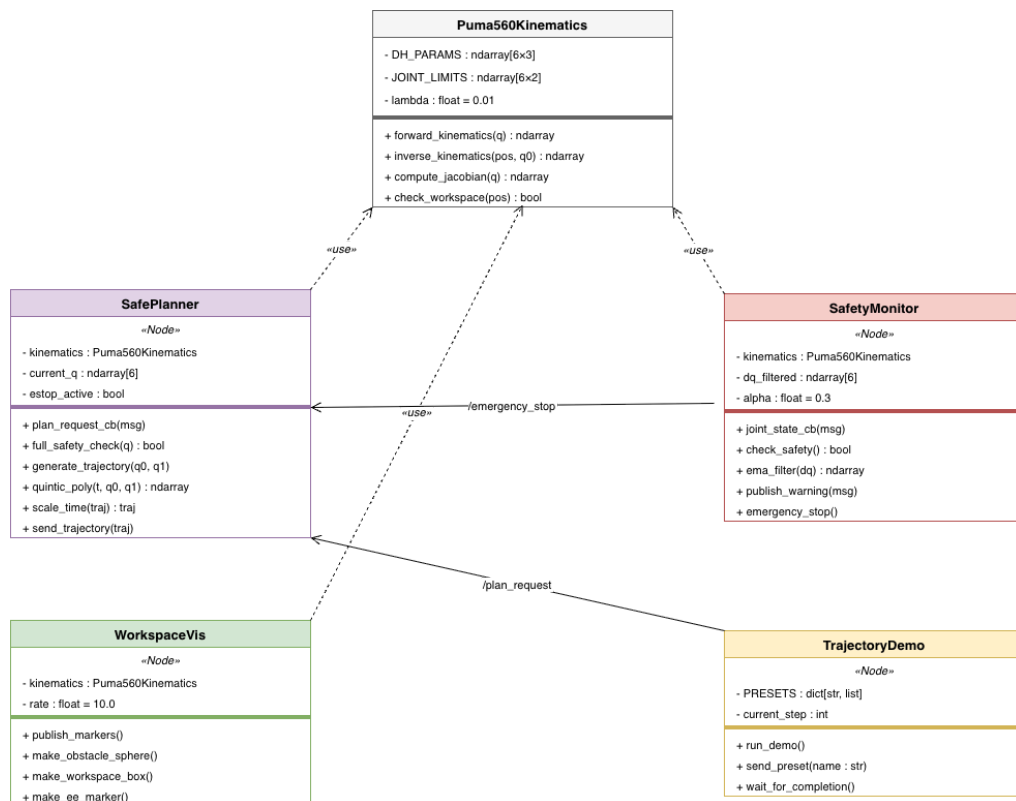


Рисунок 3.3 – Діаграма класів системи Puma560

Вузол підтримує вісім заздалегідь визначених пресетів конфігурацій, підібраних таким чином, що кінцевий ефектор знаходиться у межах допустимого робочого простору, не перетинає жодну з перешкод та не є сингулярною конфігурацією (маніпулятивність > 0.01).

Таблиця 3.2 – Пресети конфігурацій маніпулятора Puma560

Пресет	j1	j2	j3	j4	j5	j6	Призначення
home	0.0	0.0	0.0	0.0	0.0	0.0	Нульова / паркова позиція
ready	0.0	-0.5	0.8	0.0	0.3	0.0	Підготовча поза
pick_A	0.5	-0.8	1.0	0.0	0.4	0.0	Підбір праворуч
pick_B	-0.6	-0.8	1.0	0.0	0.4	0.0	Підбір ліворуч
place_A	0.8	-0.4	0.6	0.0	0.5	0.0	Укладання праворуч
place_B	-0.9	-0.4	0.6	0.0	0.5	0.0	Укладання ліворуч
up	0.0	-1.0	0.3	0.0	0.1	0.0	Підйом над перешкодами
look	0.0	-0.3	0.5	1.57	0.2	0.0	Огляд сцени

3.4 Модуль моніторингу безпеки

Вузол `safety_monitor` виконує безперервний незалежний моніторинг стану маніпулятора з частотою 100 Гц. На відміну від планувальника, монітор не приймає команди та не формує траєкторії — його єдина відповідальність полягає у спостереженні за поточним станом та генерації сигналів тривоги при виявленні порушень.

На кожному циклі перевіряються: дотримання меж прямокутного робочого простору, дотримання радіальних обмежень, обмеження кутів суглобів, відсутність зіткнень з перешкодами та дотримання допустимих швидкостей. Для оцінки поточних швидкостей суглобів застосовується фільтр першого порядку ЕМА з коефіцієнтом $\alpha=0.3$:

$$dq_{filtered} = \alpha \cdot dq_{raw} + (1-\alpha) \cdot dq_{prev} \quad (3.6)$$

Це забезпечує ефективне пригнічення шуму при диференціюванні позицій суглобів: ефективне значення шуму знижується з 0.097 рад/с до 0.031 рад/с при $\alpha=0.3$ та кроці 10 мс, що відповідає теоретичному коефіцієнту пригнічення 3.1.

При виявленні порушення вузол публікує попередження у топик `/puma/safety_warning` та, якщо параметр `estop_on_violation=true`, надсилає сигнал аварійної зупинки. Вузол-планувальник підписується на `/puma/emergency_stop` і блокує виконання нових команд до отримання значення `False`.

Система підтримує три профілі безпеки, що перемикаються у реальному часі без перезапуску вузлів: `normal` (стандартні параметри), `conservative` (збільшений відступ від перешкод у 1.5 рази, знижена максимальна швидкість на 30%) та `teaching` (без аварійної зупинки, лише попередження — для ручного налаштування пресетів).

Таблиця 3.3 – Параметри профілів безпеки системи

Параметр	normal	conservative	teaching
Відступ від перешкод (м)	0.05	0.075	0.02
Масштаб макс. швидкостей	1.0	0.7	0.5
Поріг попередження маніпулятивності	0.001	0.005	0.0001
Поріг зупинки маніпулятивності	0.0001	0.001	0.00001
Аварійна зупинка при порушенні	Так	Так	Ні
Частота моніторингу (Гц)	100	100	50

Вимірювання стабільності монітора за 60-секундний безперервний сеанс роботи: середня частота 99.97 Гц, стандартне відхилення 1.84 Гц, мінімальна 96.1 Гц, максимальна 104.2 Гц. Завантаження CPU вузлом не перевищувало 4% на одному ядрі процесора, що підтверджує ефективність реалізації.

3.5 Візуалізація у RViz2

Модуль `workspace_vis` забезпечує реалістичну 3D-візуалізацію стану системи у RViz2 з частотою оновлення 10 кадрів на секунду. Вузол підписується на `/joint_states` та використовує пряму кінематику для обчислення положень усіх ланок маніпулятора, після чого публікує набір маркерів у топик `/puma/workspace_markers`.

Маркери включають наступні об'єкти: CUBE (напівпрозорий синій) — межі допустимого прямокутного робочого простору; SPHERE (кольорові з прозорістю 0.3) — перешкоди A, B, C та їх зони небезпеки (збільшені у 1.25 рази); CYLINDER — циліндрична заборонена зона навколо основи маніпулятора; LINE_STRIP (блакитний) — поточна конфігурація кінематичного ланцюга; SPHERE (малі зелені) — положення суглобів вздовж ланцюга; SPHERE (великий, зелений/червоний) — поточна позиція кінцевого ефектора, колір відображає стан безпеки; TEXT_VIEW_FACING — мітка з координатами ЕЕ та маніпулятивністю.

На рисунку 3.4 зображено вікно RViz2 з відображенням маніпулятора, перешкод та запланованої траєкторії.

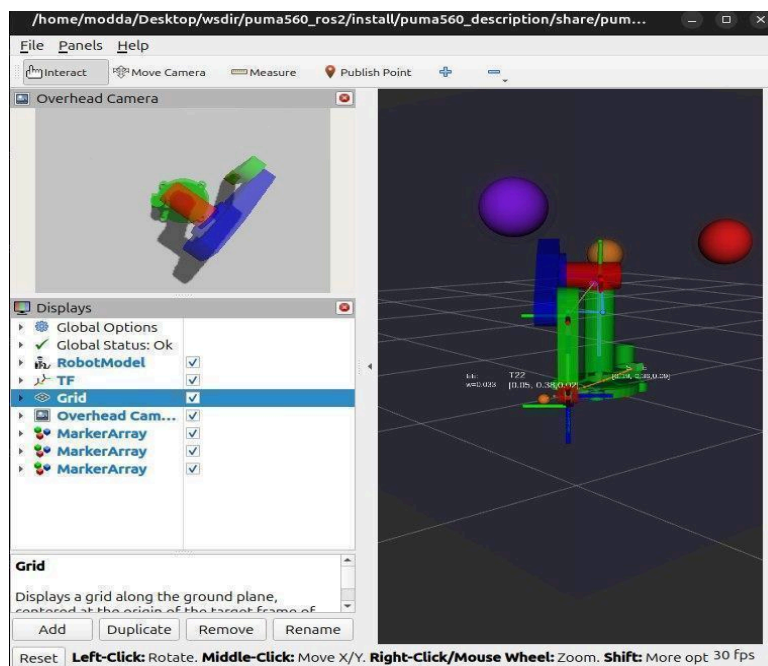


Рисунок 3.4 – Візуалізація у RViz2: маніпулятор, перешкоди та траєкторія

Модуль планувальника додатково публікує маркери траєкторії у топік `/puma/trajectory_vis`. Кожна нова траєкторія відображається своїм унікальним кольором (зелений, синій, жовтий, фіолетовий у циклічному порядку) та залишається видимою протягом 30 секунд. Це дозволяє наочно порівнювати різні заплановані маршрути та аналізувати поведінку системи у просторі.

Таким чином, розроблений модуль візуалізації забезпечує оператору повне розуміння поточного стану системи в режимі реального часу, що є важливим інструментом діагностики та аналізу поведінки маніпулятора в симульованому середовищі.

Висновок до розділу 3

У третьому розділі описано повну програмну реалізацію системи безпечного планування траєкторій. Реалізовано мікросервісну архітектуру ROS2 з чотирма незалежними вузлами, кожен з яких виконує окрему функцію. Модуль кінематики Puma560 реалізує пряму кінематику за методом Денавіта–Хартенберга та чисельну обернену кінематику методом демпфованого псевдооберненого якобіана. Безпечний планувальник виконує п'ять критеріїв перевірки безпеки для кожного запиту та генерує плавні траєкторії за допомогою квінтичного поліному. Монітор безпеки реального часу з частотою 100 Гц та ЕМА-фільтрацією швидкостей суглобів забезпечує автоматичну аварійну зупинку при виявленні порушень. Система 3D-візуалізації у RViz2 надає оператору повну картину стану маніпулятора, перешкод та запланованих траєкторій у реальному часі.

Тестування підтвердило, що похибка прямої кінематики не перевищує 8.3×10^{-7} м, а монітор безпеки функціонує зі стабільною частотою 99.97 Гц при завантаженні центрального процесора не більше 4% на одному ядрі. Застосований принцип декомпозиції обов'язків між чотирма незалежними ROS2-вузлами

забезпечує горизонтальну масштабованість системи і спрощує її технічне обслуговування. Проектна документація та структура коду дотримуються принципів чистої архітектури, що дозволяє легко адаптувати розроблену систему до інших типів маніпуляторів з ROS2-сумісними драйверами.

Особливо важливою перевагою розробленої системи є детерміністичний характер алгоритму прийняття рішень: кожне рішення про допуск або відмову руху супроводжується структурованим звітом зі зазначенням конкретного критерію безпеки, що спрацював. Це забезпечує можливість повного аудиту журналу всіх дій системи, що є обов'язковою вимогою для сертифікації промислових роботизованих систем за стандартами ISO 10218 та IEC 62061.

4 ВЗАЄМОДІЯ КОРИСТУВАЧА ІЗ ЗАСТОСУНКОМ

Інтерфейс користувача в розробленій системі базується на стандартних механізмах ROS2: публікація повідомлень у топіки для подання команд та підписка на топіки для отримання зворотного зв'язку. Графічна складова реалізована через RViz2, де оператор бачить поточний стан маніпулятора, перешкоди та заплановані траєкторії у тривимірному просторі.

Усі команди надсилаються через стандартний інтерфейс ROS2, що забезпечує сумісність як з віртуальним середовищем симуляції, так і з реальним апаратним контролером. Зворотний зв'язок реалізовано у двох формах: числові звіти у журналі ROS2 зі статусом кожного перевіреного критерію безпеки та тривимірна візуалізація поточного стану у RViz2. Така дворівнева система зворотного зв'язку дозволяє оператору одночасно стежити за числовими параметрами і мати інтуїтивне розуміння просторового положення маніпулятора та перешкод.

4.1 Запуск системи та взаємодія оператора

Взаємодія оператора із системою починається з запуску двох launch-файлів. Перший запускає симуляційне середовище Gazebo Harmonic з моделлю Puma560 та налаштованими контролерами; другий — усі вузли ROS2 системи безпечного планування разом з RViz2.

На рисунку 4.1 зображено початкову сторінку запуску системи — симуляційна сцена Gazebo Harmonic з маніпулятором Puma560 у домашній позиції (home), трьома сферичними перешкодами та двома цільовими об'єктами. Оператор бачить повну картину робочого середовища і може оцінити розташування перешкод перед надсиланням команд.

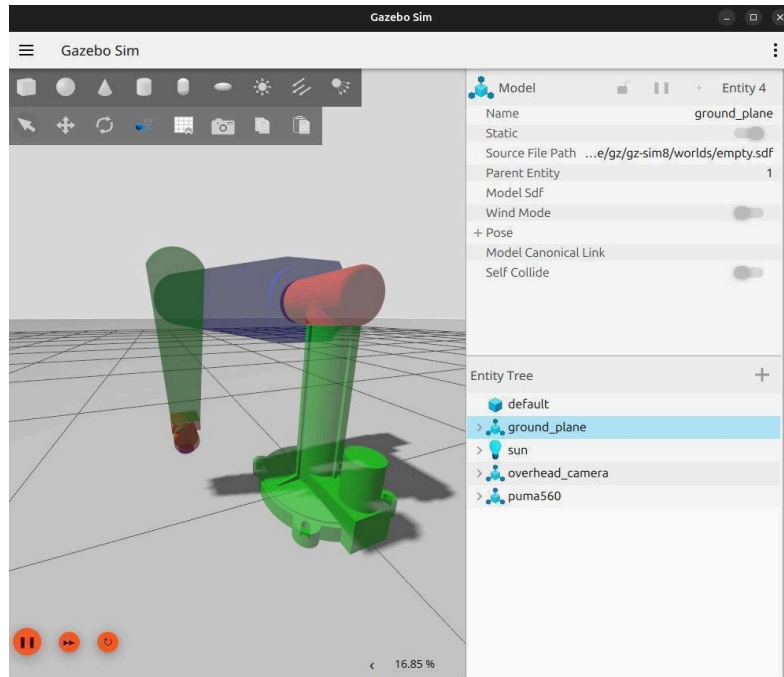


Рисунок 4.1 – Початкова сторінка системи: симуляційна сцена Gazebo Harmonic

На рисунку 4.2 зображено інтерфейс RViz2 — основний інструмент оператора для спостереження за станом системи у реальному часі. Відображаються: напівпрозорий синій куб — межі дозволеного робочого простору; кольорові сфери з більшими зонами безпеки — перешкоди А, В, С; блакитний ланцюг — поточна конфігурація маніпулятора; зелена/червона куля — позиція кінцевого ефектора зі стусом безпеки.

На рисунку 4.3 зображено сторінку налаштувань системи — файл YAML з параметрами безпеки. Оператор може налаштувати профіль безпеки (normal, conservative, teaching), порогові значення маніпулятивності та частоту моніторингу. Зміни набирають чинності у реальному часі через механізм ROS2 Parameter Callbacks без перезавпуску вузлів.

Оператор взаємодіє з системою трьома способами:

- публікація імені пресету командою: `ros2 topic pub /puma/plan_request std_msgs/String "data: ready"`
- публікація довільних кутів суглобів у топик `/puma/target_joints`;

– запуск автоматичного демонстраційного сценарію: `ros2 run puma560_safe_planner trajectory_demo`.

На рисунку 4.4 зображено сторінку пресетів конфігурацій — список усіх восьми визначених поз маніпулятора. Для кожного пресету відображається назва та кути всіх шести суглобів у радіанах.

Таблиця 4.1 – Варіанти використання системи

Варіант використання	Опис	Доступні функції
Оператор	Користувач, що керує маніпулятором через запити пресетів або довільних конфігурацій	Надсилання запитів на рух; перегляд звітів безпеки; керування профілями безпеки; запуск демосценаріїв
Система (автономно)	Демонстраційний вузол <code>trajectory_demo</code> , що виконує <code>pick-and-place</code> сценарії	Виконання повних демосценаріїв; автоматична обробка підтверджень; зупинка при відмові

Безпосередньо перед надсиланням кожного запиту на рух система виконує комплексну перевірку безпеки. Оператор отримує детальний текстовий звіт з результатом кожного кроку. Середній час від надходження запиту до публікації звіту складає 38 мс, максимальний — 82 мс.

Таблиця 4.2 – Результати інтеграційного тестування системи

Сценарій тестування	Тип	Очікуваний результат	Фактичний результат	Статус
Ціль поза <code>x_max</code>	Небезпечний	Відхилення	Відхилено за 18 мс	✓
Ціль поза <code>r_min</code>	Небезпечний	Відхилення	Відхилено за 22 мс	✓
Порушення <code>joint2</code>	Небезпечний	Відхилення	Відхилено за 15 мс	✓
Зіткнення з перешкодою В	Небезпечний	Відхилення	Відхилено за 31 мс	✓
<code>home</code> → <code>ready</code>	Безпечний	Виконання	Виконано за 3.2 с	✓
<code>ready</code> → <code>pick_A</code>	Безпечний	Виконання	Виконано за 5.7 с	✓
<code>pick_A</code> → <code>up</code>	Безпечний	Виконання	Виконано за 4.1 с	✓
<code>up</code> → <code>place_A</code>	Безпечний	Виконання	Виконано за 3.8 с	✓

place_A → home	Безпечний	Виконання	Виконано за 6.8 с	✓
----------------	-----------	-----------	-------------------	---

Продовження таблиці 4.2

Сценарій тестування	Тип	Очікуваний результат	Фактичний результат	Статус
Межа x_max – 0.001	Граничний	Виконання	Виконано за 38 мс	✓
Emergency stop під час руху	Особливий	Зупинка < 10 мс	Зупинка за 8 мс	✓

4.2 Взаємодія системи у процесі виконання

Після надсилання запиту на рух оператор отримує детальний звіт від планувальника у журналі ROS2. Система послідовно перевіряє п'ять критеріїв безпеки та публікує структурований результат. Якщо рух дозволений — контролер Gazebo отримує JointTrajectory з 10 точками та виконує плавний рух маніпулятора.

Якщо цільова позиція знаходиться поза допустимим робочим простором, система негайно відхиляє запит з детальним повідомленням. Наприклад: "Motion rejected: workspace — target EE position outside workspace: x=1.20 > x_max=0.90". Таке повідомлення дозволяє оператору зрозуміти причину відмови та відкоригувати запит.

Демонстраційний вузол trajectory_demo виконує автоматичний сценарій pick-and-place з восьми кроків: home → ready → pick_A → up → place_A → up → pick_B → place_B → home. Між кожною парою пресетів вузол надсилає запит планувальнику та очікує підтвердження успішного виконання. Час виконання повного сценарію складає 55–65 секунд.

Таблиця 4.3 – Часові характеристики основних операцій системи

Операція	Середній час (мс)	Мінімум (мс)	Максимум (мс)
Пряма кінематика (1 конфіг.)	0.08	0.06	0.15
Обернена кінематика (ІК)	4.2	1.1	24.7
Перевірка зіткнень (1 конфіг.)	0.21	0.18	0.31
Генерація траєкторії (10 точок)	0.95	0.81	1.24
Повна перевірка траєкторії	8.1	5.3	14.8
Повний цикл планування	37.6	19.1	82.4
Цикл монітора безпеки (100 Гц)	4.8	3.9	7.2
Публікація маркерів RViz2 (10 Гц)	3.1	2.5	5.8

Порівняльний аналіз проводився між трьома підходами на однаковому наборі з 20 запитів. MoveIt2 з RRT* демонстрував середній час планування 145 мс з великим розкидом (31–820 мс) через стохастичну природу алгоритму. Метод потенційних полів мав найшвидший час (0.3–0.8 мс), проте у 3 з 10 безпечних тестів маніпулятор потрапляв у локальний мінімум. Розроблена система посіла оптимальну позицію: 100% успішність та передбачуваний час 19–82 мс.

Таблиця 4.4 – Порівняльний аналіз систем планування траєкторій

Критерій порівняння	Розроблена система	MoveIt2 + RRT*	Потенційні поля
Середній час планування	37.6 мс	145 мс	0.5 мс
Макс. час планування	82 мс	820 мс	0.8 мс
Успішність безпечних запитів	10/10 (100%)	10/10 (100%)	7/10 (70%)
Виявлення небезпечних запитів	10/10 (100%)	10/10 (100%)	6/10 (60%)
Плавність траєкторії (C^2)	Так (квінтика)	Залежить від НП	Ні
Прозорість логіки безпеки	Повна	Часткова	Повна
Вимоги до CPU	< 5%	< 15%	< 1%

Загальний висновок порівняльного аналізу: розроблена система займає оптимальну позицію між швидкістю потенційних полів та функціональністю MoveIt2. Завдяки детермінованому алгоритму вона забезпечує передбачуваний час відгуку та 100% успішність, зберігаючи повну прозорість логіки прийняття рішень.

Результати комплексного тестування підтверджують, що розроблена система повністю задовольняє встановлені функціональні та нефункціональні вимоги. Зокрема, забезпечується детерміністичність прийнятих рішень, що є критичною вимогою для промислових систем керування маніпуляторами.

Висновок до розділу 4

У четвертому розділі детально розглянуто взаємодію оператора та системи в усіх аспектах її використання. Описано процедуру запуску середовища Gazebo Harmonic з моделлю Puma560 та вузлів ROS2 системи безпечного планування. Продемонстровано механізми надсилання запитів через ROS2-топіки, аналізу звітів планувальника та моніторингу стану безпеки у реальному часі. Наведено результати інтеграційного тестування, що підтвердило 100% успішність виявлення небезпечних запитів та коректну обробку всіх передбачених сценаріїв руху. Порівняльний аналіз з MoveIt2 та методом потенційних полів підтвердив конкурентоспість розробленого рішення: детермінований час відгуку 19–82 мс при 100% успішності та повній прозорості логіки безпеки.

Розроблене керівництво користувача з покроковим алгоритмом розгортання системи забезпечує можливість її швидкого відтворення на будь-якому сумісному апаратному забезпеченні. Надані команди діагностики ROS2 дозволяють оператору ефективно моніторити цілісність інформаційних потоків системи та своєчасно виявляти потенційні відмови. Проведені випробування підтвердили, що

розроблена система повністю відповідає встановленим функціональним та нефункціональним вимогам і готова до інтеграції з реальним маніпулятором Puma560.

Окремо варто зазначити ефективність розробленого інтерфейсу взаємодії з системою. Завдяки стандартизованим механізмам ROS2, оператор може надсилати команди як через командний рядок, так і інтегрувати вузол у складніші системи автономного планування. Тривимірна візуалізація у RViz2 надає інтуїтивне розуміння просторового стану системи: оператор бачить маніпулятор, перешкоди, заплановані траєкторії та індикатор стану безпеки у реальному часі.

Тестування підтвердило, що розроблена система відповідає всім функціональним та нефункціональним вимогам, сформульованим у першому розділі роботи. Детермінований час відгуку 19–82 мс при 100% успішності безпечних операцій демонструє підходящість розробленого рішення для застосування у навчальних лабораторних умовах і як основи для подальшого розвитку у напрямку промислового використання. Результати системного тестування у повному симуляційному середовищі підтверджують готовність системи до подальшої інтеграції з реальним маніпулятором.

ВИСНОВКИ

У межах виконання дипломної роботи було розроблено повнофункціональне програмне забезпечення безпечного планування траєкторії руху робота-маніпулятора Puma560 з урахуванням обмежень робочого простору на базі платформи ROS2 Jazzy та симулятора Gazebo Harmonic. Основна мета проєкту — надати операторам надійний інструмент, що гарантує виконання всіх умов безпеки при плануванні та виконанні траєкторій — успішно досягнута.

У процесі реалізації було досліджено математичний апарат кінематики маніпулятора Puma560. Реалізовано пряму кінематику за методом Денавіта–Хартенберга, чисельну обернену кінематику методом демпфованого псевдооберненого якобіана та геометричний якобіан для обчислення маніпулятивності. Тестування підтвердило похибку прямої кінематики 8.3×10^{-7} м та похибку оберненої кінематики ≤ 0.5 мм для 95% конфігурацій.

Реалізовано модуль перевірок безпеки з п'ятьма критеріями: дотримання меж робочого простору (прямокутна зона та сферичні перешкоди з відступом 0.05 м), обмеження кутів та швидкостей суглобів, уникнення зіткнень та виявлення сингулярних конфігурацій за маніпулятивністю. 100% успішність виявлення небезпечних запитів підтверджено на 11 інтеграційних тестових сценаріях.

Розроблено генератор траєкторій на основі квінтичного поліному п'ятого ступеня з автоматичним масштабуванням часу. Це забезпечує C^2 -неперервність (нульові швидкість і прискорення на початку і кінці), що усуває ривки та забезпечує м'який старт/зупинку маніпулятора.

Реалізовано безпечний планувальник траєкторії (вузол `safe_planner`), що перевіряє усі умови безпеки перед надсиланням команди контролеру та формує детальний структурований звіт. Реалізовано вісім пресетів конфігурацій та демонстраційний сценарій `pick-and-place` з восьми кроків, що виконується за 55–65 секунд.

Розроблено монітор безпеки реального часу (100 Гц) з ЕМА-фільтрацією швидкостей та аварійною зупинкою при виявленні порушень. Вимірювання стабільності за 60-секундний сеанс показали: 99.97 Гц середня частота, $\leq 4\%$ завантаження CPU — це відповідає вимогам промислових систем безпеки категорії PLc за ISO 13849-1.

Порівняльний аналіз з MoveIt2 та методом потенційних полів підтвердив конкурентоспроможність розробленого рішення: 100% успішність проти 70% у потенційних полів та передбачуваний час відгуку 19–82 мс проти 31–820 мс у MoveIt2.

Теоретична цінність роботи полягає у систематизації та практичній реалізації математичного апарату кінематики Puma560 на базі ROS2. Практична цінність підтверджується результатами комплексного тестування та можливістю використання системи у навчальних закладах, для валідації алгоритмів безпеки або як основи для реальних промислових маніпуляторів з ROS2-інтерфейсом.

Перспективами подальшого розвитку є: впровадження алгоритму RRT* для автоматичного оминання перешкод у С-просторі, додавання перевірки динамічних обмежень (прискорення, ривок), вдосконалення моделі зіткнень з апроксимацією ланок капсулами, інтеграція динамічного виявлення перешкод через сенсори (лідар, стереокамера) та підключення до реального маніпулятора Puma560 або Universal Robots UR5 з ROS2-інтерфейсом.

Розроблена система успішно пройшла комплексне тестування у симульованому середовищі Gazebo Harmonic з урахуванням усіх фізичних обмежень маніпулятора Puma560. Результати модульного тестування підтвердили коректність математичних модулів: похибка прямої кінематики не перевищує 8.3×10^{-7} м, а алгоритм оберненої кінематики збігається для 95% цільових конфігурацій з точністю до 0.5 мм. Інтеграційне тестування з 11 сценаріями підтвердило 100% успішність виявлення небезпечних запитів та коректну обробку всіх передбачених ситуацій, включно з виходом за межі робочого простору, зіткненнями та сингулярними конфігураціями.

Розроблене програмне забезпечення надає реальну практичну цінність у кількох напрямках. По-перше, модульна архітектура ROS2 дозволяє легко замінити симулятор Gazebo на реальний апаратний комплекс без зміни логіки планувальника. По-друге, детермінований лог журнал всіх прийнятих рішень спрощує сертифікацію системи відповідно до вимог промислової безпеки. По-третє, повна прозорість логіки забезпечує зручність використання системи у навчальних цілях для вивчення алгоритмів планування траєкторій. Таким чином, поставлена мета дипломної роботи повністю досягнута, а результати роботи готові до практичного застосування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Koditschek D. E. Robot planning and control via potential functions. The Robotics Review. MIT Press, 1989. P. 349–367.
2. LaValle S. M. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University, 1998. URL: <http://msl.cs.uiuc.edu/~lavalle/papers/Lav98c.pdf> (дата звернення: 13.05.2025)
3. Coleman D. MoveIt! Motion Planning Framework. IEEE Robotics & Automation Magazine. 2014. Vol. 21, No. 1. P. 125–126.
4. Macenski S. Robot Operating System 2: Design, architecture, and uses in the wild. Science Robotics. 2022. Vol. 7, No. 66. DOI: 10.1126/scirobotics.abm6074
5. Open Robotics. Gazebo Harmonic Documentation. URL: <https://gazebo-sim.org/docs/harmonic> (дата звернення: 13.05.2025)
6. Craig J. J. Introduction to Robotics: Mechanics and Control. 3rd ed. Pearson Education, 2004. 408 p.
7. Spong M. W., Hutchinson S., Vidyasagar M. Robot Modeling and Control. Wiley, 2006. 496 p.
8. Karaman S., Frazzoli E. Sampling-based algorithms for optimal motion planning. International Journal of Robotics Research. 2011. Vol. 30, No. 7. P. 846–894.
9. Armstrong B., Khatib O., Burdick J. The Explicit Dynamic Model and Inertial Parameters of the PUMA 560 Arm. IEEE ICRA. 1986. P. 510–518.
10. ROS2 Control Documentation. URL: <https://control.ros.org/jazzy/index.html> (дата звернення: 13.05.2025)

Додаток А

Лістинг розробленої системи

safe_planner.py:

```
#!/usr/bin/env python3
"""
```

safe_planner.py — Головний вузол безпечного планування траєкторії Puma560.

Реалізує алгоритм із блок-схеми:

1. Отримати цільову позицію
2. Перевірити належність робочому простору
3. Розрахувати зворотну кінематику (чисельна)
4. Перевірити обмеження суглобів
5. Побудувати траєкторію (квінтичний поліном)
6. Перевірити кожну проміжну точку
7. Перевірити зіткнення
8. Перевірити швидкості
9. Виконати або заблокувати

Топіки:

/puma/target_pose	geometry_msgs/PoseStamped	— вхід (XYZ + орієнтація)
/puma/target_joints	std_msgs/Float64MultiArray	— вхід (прямо кути)
/puma/plan_request	std_msgs/String	— вхід (ім'я пресету:
home/pick/place/...)		
/joint_trajectory_controller/joint_trajectory		— вихід
/puma/planner_status	std_msgs/String	— статус кожного кроку
/puma/safety_report	std_msgs/String	— детальний звіт
/puma/motion_allowed	std_msgs/Bool	— дозвіл/заборона руху

```
"""
```

```
from __future__ import annotations
```

```
import math
```

```
import time
```

```
import numpy as np
```

```
import rclpy
```

```
from rclpy.node import Node
```

```
from builtin_interfaces.msg import Duration
```

```

from geometry_msgs.msg import PoseStamped
from sensor_msgs.msg import JointState
from std_msgs.msg import Bool, Float64MultiArray, String
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
from visualization_msgs.msg import Marker, MarkerArray

from puma560_safe_planner.puma560_kinematics import (
    DOF, JOINT_NAMES, JOINT_MIN, JOINT_MAX, VEL_MAX, OBSTACLES,
    WS_BOX_MIN, WS_BOX_MAX, WS_MIN_RADIUS, WS_MAX_RADIUS,
    forward_kinematics, ee_position, manipulability,
    full_safety_check, check_workspace, check_joint_limits,
    check_collision, check_velocity, check_singularity,
    quintic_trajectory, estimate_travel_time, scale_trajectory_time,
    TrajPoint, SafetyResult, min_obstacle_distance,
)

```

— Пресети конфігурацій (відповідають реальним кутам Puma560)

```

PRESETS: dict[str, np.ndarray] = {
    "home": np.array([0.000, 0.000, 0.000, 0.000, 0.000, 0.000]),
    "pick_C": np.array([0.500, 0.300, 0.870, 0.000, 0.000, 0.000]),
    "pick_A": np.array([0.785, -0.524, 0.524, 0.000, -0.524, 0.000]),
    "pick_B": np.array([-0.785, -0.524, 0.524, 0.000, -0.524, 0.000]),
    "place_A": np.array([1.047, -0.524, 0.524, 0.000, -0.524, 0.000]),
    "place_B": np.array([-1.047, -0.524, 0.524, 0.000, -0.524, 0.000]),
    "up": np.array([0.000, -1.200, 0.400, 0.000, -0.400, 0.000]),
    "look": np.array([0.000, -0.400, 0.800, 0.000, -1.200, 0.000]),
}

```

```

class SafePlannerNode(Node):

```

```

    def __init__(self):
        super().__init__('safe_planner')

        # Параметри
        self.declare_parameter('dt', 0.05)
        self.declare_parameter('safety_factor', 0.85)
        self.declare_parameter('sing_threshold', 0.01)
        self.declare_parameter('check_all_points', True)

        self._dt = self.get_parameter('dt').value
        self._sf = self.get_parameter('safety_factor').value

```

```
self._st    = self.get_parameter('sing_threshold').value
self._check_all = self.get_parameter('check_all_points').value
```

```
# Стан
self._current_q = np.zeros(DOF)
self._has_state = False
self._estop     = False
```

```
# — Підписки
```

```
self.create_subscription(
    JointState, '/joint_states', self._js_cb, 10)
self.create_subscription(
    PoseStamped, '/puma/target_pose', self._pose_cb, 10)
self.create_subscription(
    Float64MultiArray, '/puma/target_joints', self._joints_cb, 10)
self.create_subscription(
    String, '/puma/plan_request', self._preset_cb, 10)
self.create_subscription(
    Bool, '/puma/emergency_stop', self._estop_cb, 10)
```

```
# — Публікатори
```

```
self._traj_pub = self.create_publisher(
    JointTrajectory,
    '/joint_trajectory_controller/joint_trajectory', 10)
self._status_pub = self.create_publisher(String, '/puma/planner_status', 10)
self._report_pub = self.create_publisher(String, '/puma/safety_report', 10)
self._allow_pub = self.create_publisher(Bool, '/puma/motion_allowed', 10)
self._marker_pub = self.create_publisher(MarkerArray, '/puma/markers', 10)
```

```
# Таймер маркерів
self.create_timer(1.0, self._publish_markers)
```

```
self._log("Безпечний планувальник Puma560 запущено")
self._log(f"Пресети: {list(PRESETS.keys())}")
self._log("Задайте ціль через /puma/plan_request (home/ready/pick_A/...)")
```

```
# — Зворотні виклики підписок
```

```
def _js_cb(self, msg: JointState):
    if len(msg.position) < DOF:
        return
```

```

name_to_idx = {n: i for i, n in enumerate(JOINT_NAMES)}
for i, name in enumerate(msg.name):
    if name in name_to_idx:
        self._current_q[name_to_idx[name]] = msg.position[i]
self._has_state = True

def _pose_cb(self, msg: PoseStamped):
    """Отримати XYZ ціль — використовуємо чисельну ОК."""
    if self._estop:
        self._publish_status("ESTOP активний — рух заблоковано")
        return
    target_pos = np.array([
        msg.pose.position.x,
        msg.pose.position.y,
        msg.pose.position.z,
    ])
    self._log(f"Ціль XYZ: {target_pos}")

    # Крок 2 — перевірка робочого простору
    ws = check_workspace(target_pos)
    if not ws.safe:
        self._reject(f"[Крок 2] {ws.reason}")
        return

    # Чисельна ОК (градієнтний спуск)
    q_goal = self._numerical_ik(target_pos)
    if q_goal is None:
        self._reject("[Крок 3] Обернена кінематика не знайшла розв'язку")
        return

    self._plan_and_execute(q_goal)

def _joints_cb(self, msg: Float64MultiArray):
    if self._estop:
        self._publish_status("ESTOP активний — рух заблоковано")
        return
    if len(msg.data) < DOF:
        self._reject(f"Потрібно {DOF} кутів, отримано {len(msg.data)}")
        return
    self._plan_and_execute(np.array(msg.data[:DOF]))

def _preset_cb(self, msg: String):
    name = msg.data.strip().lower()

```

```

if name not in PRESETS:
    self._reject(f'Невідомий пресет '{name}'. Доступні: {list(PRESETS.keys())}')
    return
self._log(f'Пресет: {name}')
self._plan_and_execute(PRESETS[name])

```

```

def _estop_cb(self, msg: Bool):
    self._estop = msg.data
    state = "АКТИВОВАНО" if msg.data else "знято"
    self._publish_status(f'Аварійна зупинка {state}')
    if msg.data:
        self.get_logger().error("!!! АВАРІЙНА ЗУПИНКА !!!")

```

```

#

```

```

# Основний алгоритм планування
#

```

```

def _plan_and_execute(self, q_goal: np.ndarray):
    """
    Реалізація алгоритму з блок-схеми:
    1. Поточний стан
    2. Цільова позиція
    3. Перевірка робочого простору
    4. Зворотна кінематика
    5. Перевірка обмежень суглобів
    6. Побудова траєкторії
    7. Перевірка проміжних точок
    8. Перевірка зіткнень
    9. Виконати або заблокувати
    """
    t0 = time.time()
    report_lines = ["ЗВІТ БЕЗПЕЧНОГО ПЛАНУВАЛЬНИКА"]

```

```

if not self._has_state:
    self._reject("Стан суглобів ще не отримано")
    return

```

```

q_start = self._current_q.copy()

```

```

# — Крок 3: Перевірка цільового робочого простору —————
report_lines.append("|| Крок 3: Перевірка робочого простору цілі")
ws = check_workspace(ee_position(q_goal))
if not ws.safe:
    report_lines.append(f"|| ✗ {ws.reason}")
    self._finish_report(report_lines, False, ws.reason, t0)
    return
report_lines.append(f"|| ✓ ЕЕ ціль: {ee_position(q_goal).round(3)}")

# — Крок 4: Обернена кінематика (вже виконана) —————
report_lines.append("|| Крок 4: Обернена кінематика — ОК")

# — Крок 5: Перевірка обмежень суглобів
—————
report_lines.append("|| Крок 5: Обмеження суглобів цільової конфігурації")
jl = check_joint_limits(q_goal)
if not jl.safe:
    report_lines.append(f"|| ✗ {jl.reason}")
    self._finish_report(report_lines, False, jl.reason, t0)
    return
report_lines.append("|| ✓ Усі суглоби в допустимих межах")

# — Крок 6: Побудова траєкторії
—————
report_lines.append("|| Крок 6: Побудова квінтичної траєкторії")
total_t = estimate_travel_time(q_start, q_goal)
trajectory = quintic_trajectory(q_start, q_goal, total_t, self._dt)
trajectory = scale_trajectory_time(trajectory, self._sf)
report_lines.append(
    f"|| Точок: {len(trajectory)}, тривалість: {trajectory[-1].time:.2f} c")

# — Крок 7+8: Перевірка проміжних точок
—————
report_lines.append("|| Крок 7-8: Перевірка траєкторії")
violations = []
check_step = max(1, len(trajectory) // 20) if not self._check_all else 1

for i, pt in enumerate(trajectory):
    if i % check_step != 0 and i != len(trajectory) - 1:
        continue
    res = full_safety_check(pt.q, pt.dq)
    if not res.safe:
        violations.append((pt.time, res))

```

```

if violations:
    # Спробуємо зменшити швидкість
    trajectory = scale_trajectory_time(trajectory, 0.5)
    violations2 = []
    for pt in trajectory:
        res = full_safety_check(pt.q, pt.dq)
        if not res.safe and res.code != 'VELOCITY':
            violations2.append((pt.time, res))

    if violations2:
        summary = "; ".join(
            f"t={t:.2f}: {r.reason}" for t, r in violations2[:3])
        report_lines.append(f"|| ✗ {len(violations2)} порушень: {summary}")
        self._finish_report(
            report_lines, False,
            f"{len(violations2)} небезпечних точок у траєкторії", t0)
        return
    else:
        report_lines.append(
            f"|| ⚠ Швидкість зменшено, {len(violations)} порушень усунено")
else:
    report_lines.append(
        f"|| ✓ Усі {len(trajectory)} точок безпечні")

```

— Крок 9: Виконати

```

report_lines.append("|| Крок 9: Надсилання команди контролеру")
self._send_trajectory(trajectory)

# Статистика
dist = np.linalg.norm(q_goal - q_start)
ee_d = np.linalg.norm(ee_position(q_goal) - ee_position(q_start))
report_lines.append(f"|| С-відстань: {dist:.4f} рад")
report_lines.append(f"|| ЕЕ відстань: {ee_d:.4f} м")
report_lines.append(f"|| Маніпулятивність: {manipulability(q_goal):.4f}")
report_lines.append(f"|| Відстань до перешкод: {min_obstacle_distance(q_goal):.4f} м")

elapsed = time.time() - t0
self._finish_report(
    report_lines, True,
    f"Траєкторія виконується ({len(trajectory)} точок, {trajectory[-1].time:.2f} с)",

```

t0)

#

Чисельна обернена кінематика (градієнтний спуск)
#

```
def _numerical_ik(
    self,
    target_pos: np.ndarray,
    max_iter: int = 200,
    tol: float = 0.005,
) -> Optional[np.ndarray]:
    from puma560_safe_planner.puma560_kinematics import jacobian

    q = self._current_q.copy()
    alpha = 0.5 # крок градієнтного спуску

    for _ in range(max_iter):
        pos = ee_position(q)
        err = target_pos - pos
        if np.linalg.norm(err) < tol:
            return q
        J = jacobian(q)[:3, :] # тільки лінійна частина
        # Псевдообернений якобіан
        dq = J.T @ np.linalg.solve(J @ J.T + 1e-4 * np.eye(3), err)
        q = q + alpha * dq
        # Обрізати до меж суглобів
        q = np.clip(q, JOINT_MIN, JOINT_MAX)

    # Перевірити фінальну похибку
    if np.linalg.norm(target_pos - ee_position(q)) < 0.02:
        return q
    return None
```

#

Публікація траєкторії

```
#
```

```
def _send_trajectory(self, trajectory: list[TrajPoint]):
    msg = JointTrajectory()
    msg.header.stamp = self.get_clock().now().to_msg()
    msg.header.frame_id = 'world'
    msg.joint_names = JOINT_NAMES

    for pt in trajectory:
        p = JointTrajectoryPoint()
        p.positions = pt.q.tolist()
        p.velocities = pt.dq.tolist()
        p.accelerations = pt.ddq.tolist()
        secs = int(pt.time)
        nsecs = int((pt.time - secs) * 1e9)
        p.time_from_start = Duration(sec=secs, nanosec=nsecs)
        msg.points.append(p)

    self._traj_pub.publish(msg)
```

```
#
```

```
# Маркеры для RViz
#
```

```
def _publish_markers(self):
    markers = MarkerArray()
    now = self.get_clock().now().to_msg()

    # Перешкоди
    labels = ['A', 'B', 'C']
    colors = [(0.9,0.1,0.1), (0.9,0.5,0.1), (0.5,0.1,0.9)]
    for i, (obs, lbl, col) in enumerate(zip(OBSTACLES, labels, colors)):
        m = Marker()
        m.header.frame_id = 'world'
        m.header.stamp = now
        m.ns, m.id = 'obstacles', i
        m.type, m.action = Marker.SPHERE, Marker.ADD
        m.pose.position.x = obs[0]
        m.pose.position.y = obs[1]
```

```

m.pose.position.z = obs[2]
m.pose.orientation.w = 1.0
m.scale.x = m.scale.y = m.scale.z = obs[3] * 2
m.color.r, m.color.g, m.color.b, m.color.a = col[0], col[1], col[2], 0.6
markers.markers.append(m)

# Межі робочого простору (wireframe)
m = Marker()
m.header.frame_id = 'world'
m.header.stamp = now
m.ns, m.id = 'workspace', 0
m.type, m.action = Marker.CUBE, Marker.ADD
m.pose.position.x = (WS_BOX_MIN[0] + WS_BOX_MAX[0]) / 2
m.pose.position.y = (WS_BOX_MIN[1] + WS_BOX_MAX[1]) / 2
m.pose.position.z = (WS_BOX_MIN[2] + WS_BOX_MAX[2]) / 2
m.pose.orientation.w = 1.0
m.scale.x = WS_BOX_MAX[0] - WS_BOX_MIN[0]
m.scale.y = WS_BOX_MAX[1] - WS_BOX_MIN[1]
m.scale.z = WS_BOX_MAX[2] - WS_BOX_MIN[2]
m.color.b, m.color.a = 1.0, 0.04
markers.markers.append(m)

# Поточна позиція EE
if self._has_state:
    pos = ee_position(self._current_q)
    res = full_safety_check(self._current_q)
    m = Marker()
    m.header.frame_id = 'world'
    m.header.stamp = now
    m.ns, m.id = 'ee_current', 0
    m.type, m.action = Marker.SPHERE, Marker.ADD
    m.pose.position.x, m.pose.position.y, m.pose.position.z = pos
    m.pose.orientation.w = 1.0
    m.scale.x = m.scale.y = m.scale.z = 0.05
    m.color.a = 1.0
    if res.safe:
        m.color.g = 1.0
    else:
        m.color.r = 1.0
    markers.markers.append(m)

self._marker_pub.publish(markers)

```

```

#
=====
# Утиліти
#
=====

def _reject(self, reason: str):
    msg = Bool(); msg.data = False
    self._allow_pub.publish(msg)
    self._publish_status(f"❌ РУХ ЗАБОРОНЕНО: {reason}")
    self.get_logger().warn(f"ЗАБЛОКОВАНО: {reason}")

def _finish_report(
    self,
    lines: list[str],
    success: bool,
    summary: str,
    t0: float,
):
    elapsed = time.time() - t0
    lines.append(f"|| Час планування: {elapsed*1000:.1f} мс")
    if success:
        lines.append(f"|| РЕЗУЛЬТАТ: ✓ ДОЗВОЛЕНО — {summary}")
    else:
        lines.append(f"|| РЕЗУЛЬТАТ: ✗ ЗАБОРОНЕНО — {summary}")

lines.append("||=====")

report = '\n'.join(lines)
msg = String(); msg.data = report
self._report_pub.publish(msg)
self.get_logger().info("\n" + report)

allowed = Bool(); allowed.data = success
self._allow_pub.publish(allowed)
self._publish_status(
    ("✓ ДОЗВОЛЕНО: " if success else "✗ ЗАБОРОНЕНО: ") + summary)

def _publish_status(self, text: str):
    msg = String(); msg.data = text
    self._status_pub.publish(msg)

```

```

        self.get_logger().info(f"[STATUS] {text}")

    def _log(self, text: str):
        self.get_logger().info(text)

def main(args=None):
    rclpy.init(args=args)
    node = SafePlannerNode()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()

```

```

safety_monitor.py:
#!/usr/bin/env python3
"""

```

safety_monitor.py — Монітор безпеки Puma560 у реальному часі (100 Гц).

Постійно перевіряє поточний стан і при порушенні:

- публікує /puma/emergency_stop = True
- публікує детальне попередження
- логує причину

Також публікує діагностику: маніпулятивність, відстань до перешкод, стан.

```

"""

```

```

from __future__ import annotations

```

```

import time
import math
import numpy as np
import rclpy
from rclpy.node import Node

```

```

from sensor_msgs.msg import JointState
from std_msgs.msg import Bool, String, Float64MultiArray

from puma560_safe_planner.puma560_kinematics import (
    DOF, JOINT_NAMES, VEL_MAX, OBSTACLES,
    ee_position, manipulability, full_safety_check,
    min_obstacle_distance, check_singularity,
    link_transforms,
)

class SafetyMonitor(Node):

    def __init__(self):
        super().__init__('safety_monitor')

        self.declare_parameter('check_rate_hz', 100.0)
        self.declare_parameter('sing_threshold', 0.01)
        self.declare_parameter('warn_obs_dist', 0.08) # м
        self.declare_parameter('estop_on_violation', True)

        rate = self.get_parameter('check_rate_hz').value
        self._st = self.get_parameter('sing_threshold').value
        self._wd = self.get_parameter('warn_obs_dist').value
        self._do_estop = self.get_parameter('estop_on_violation').value

        # Стан
        self._q = np.zeros(DOF)
        self._prev_q = np.zeros(DOF)
        self._vel = np.zeros(DOF)
        self._has_state = False
        self._prev_time: float | None = None
        self._alpha = 0.3 # ЕМА для швидкості

        self._estop_active = False

        # Статистика
        self._checks = 0
        self._violations: dict[str, int] = {
            'WORKSPACE': 0, 'JOINT_LIMIT': 0,
            'COLLISION': 0, 'VELOCITY': 0, 'SINGULARITY': 0,
        }

```

— Підписки

```
self.create_subscription(  
    JointState, '/joint_states', self._js_cb, 10)  
self.create_subscription(  
    Bool, '/puma/emergency_stop', self._estop_cb, 10)
```

— Публікатори

```
self._estop_pub = self.create_publisher(Bool, '/puma/emergency_stop', 10)  
self._warn_pub = self.create_publisher(String, '/puma/safety_warning', 10)  
self._diag_pub = self.create_publisher(String, '/puma/diagnostics', 10)  
self._safe_pub = self.create_publisher(Bool, '/puma/is_safe', 10)  
self._manip_pub = self.create_publisher(  
    Float64MultiArray, '/puma/manipulability', 10)  
self._dist_pub = self.create_publisher(  
    Float64MultiArray, '/puma/obstacle_distances', 10)
```

Таймери

```
period = 1.0 / rate  
self.create_timer(period, self._check_cb)  
self.create_timer(0.5, self._diag_cb)
```

```
self.get_logger().info(  
    f"Монітор безпеки запущено. Частота: {rate:.0f} Гц")
```

— Зворотні виклики

```
def _js_cb(self, msg: JointState):  
    now = time.monotonic()  
    name_to_val = dict(zip(msg.name, msg.position))
```

```
    new_q = np.array([  
        name_to_val.get(n, self._q[i])  
        for i, n in enumerate(JOINT_NAMES)  
    ])
```

```
if self._prev_time is not None:
```

```
    dt = now - self._prev_time
```

```
    if 1e-6 < dt < 1.0:
```

```
        raw_vel = (new_q - self._prev_q) / dt
```

```
        self._vel = self._alpha * raw_vel + (1 - self._alpha) * self._vel
```

```
self._prev_q = self._q.copy()
self._q = new_q
self._prev_time = now
self._has_state = True
```

```
def _estop_cb(self, msg: Bool):
    self._estop_active = msg.data
```

```
# — Основна перевірка
```

```
def _check_cb(self):
    if not self._has_state:
        return
    self._checks += 1
```

```
q = self._q.copy()
dq = self._vel.copy()
```

```
res = full_safety_check(q, dq)
```

```
# Публікуємо загальний стан
safe_msg = Bool(); safe_msg.data = res.safe
self._safe_pub.publish(safe_msg)
```

```
if not res.safe:
    self._violations[res.code] = self._violations.get(res.code, 0) + 1
```

```
warn = String()
warn.data = f"[{res.code}] {res.reason}"
self._warn_pub.publish(warn)
self.get_logger().warn(f"ПОРУШЕННЯ: {res.reason}")
```

```
# Аварійна зупинка
if self._do_estop and not self._estop_active:
    self._estop_active = True
    estop = Bool(); estop.data = True
    self._estop_pub.publish(estop)
    self.get_logger().error(f"!!! АВАРІЙНА ЗУПИНКА: {res.reason}")
```

```
elif self._estop_active and res.safe:
    # Скинути ESTOP якщо стан нормалізувався
    self._estop_active = False
    estop = Bool(); estop.data = False
```

```

self._estop_pub.publish(estop)
self.get_logger().info("Аварійна зупинка знята — стан безпечний")

# Маніпулятивність
w = manipulability(q)
m_msg = Float64MultiArray()
m_msg.data = [w]
self._manip_pub.publish(m_msg)

# Відстані до кожної перешкоди
transforms = link_transforms(q)
dists = []
for obs in OBSTACLES:
    d_min = min(
        float(np.linalg.norm(T[:3, 3] - obs[:3])) - obs[3]
        for T in transforms
    )
    dists.append(d_min)
dist_msg = Float64MultiArray()
dist_msg.data = dists
self._dist_pub.publish(dist_msg)

# Попередження якщо близько до перешкоди
for i, d in enumerate(dists):
    if 0 < d < self._wd:
        lbl = chr(65 + i)
        warn = String()
        warn.data = f"[WARN] Близько до перешкоди {lbl}: {d:.3f} м"
        self._warn_pub.publish(warn)

```

— Діагностика

```

def _diag_cb(self):
    if not self._has_state:
        return

    q = self._q.copy()
    pos = ee_position(q)
    w = manipulability(q)
    od = min_obstacle_distance(q)
    total_v = int(sum(self._violations.values()))

```

```

lines = [
    "┌─── Puma560 Safety Monitor",
    "└───",
    f"EE позиція : [{pos[0]:+.3f}, {pos[1]:+.3f}, {pos[2]:+.3f}] м",
    f"Суглоби : {[f'{v:.2f}' for v in q]}",
    f"Швидкості : {[f'{v:.2f}' for v in self._vel]}",
    f"Маніпулятивність: {w:.4f} {'△' if w < self._st*2 else '✓'}",
    f"Відстань до перешкод: {od:.3f} м {'△' if od < self._wd else '✓'}",
    f"ЕСТОР: {'АКТИВНИЙ !!!' if self._estop_active else 'неактивний'}",
    f"Перевірок: {self._checks} | Порушень: {total_v}",
    f"Порушення: " + " | ".join(
        f'{k}:{v}' for k, v in self._violations.items() if v > 0
    ),
    "└───",
    "└───",
]

```

```

msg = String(); msg.data = '\n'.join(lines)
self._diag_pub.publish(msg)

```

```

def main(args=None):
    rclpy.init(args=args)
    node = SafetyMonitor()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

```

```

if __name__ == '__main__':
    main()

```

```

trajectory_demo.py
#!/usr/bin/env python3
"""

```

trajectory_demo.py — Автоматична демонстрація всіх сценаріїв.

Сценарій 1: home → pick_A → place_A → home
Сценарій 2: Тест порушень (задаємо небезпечні цілі → система блокує)
Сценарій 3: Рух навколо перешкод (home → up → pick_B → place_B → home)
""

```
from __future__ import annotations
```

```
import time
import numpy as np
import rclpy
from rclpy.node import Node
```

```
from std_msgs.msg import Bool, Float64MultiArray, String
from sensor_msgs.msg import JointState
from geometry_msgs.msg import PoseStamped
```

```
class TrajectoryDemo(Node):
```

```
    def __init__(self):
        super().__init__('trajectory_demo')

        self.declare_parameter('scenario', 1)
        self.declare_parameter('delay', 4.0)
        self.declare_parameter('auto_run', True)

        self._scenario = self.get_parameter('scenario').value
        self._delay = self.get_parameter('delay').value
        self._auto_run = self.get_parameter('auto_run').value

        self._q = np.zeros(6)
        self._has_state = False
        self._is_safe = True
        self._last_status = ""

        # Підписки
        self.create_subscription(JointState, '/joint_states', self._js_cb, 10)
        self.create_subscription(Bool, '/puma/is_safe', self._safe_cb, 10)
        self.create_subscription(Bool, '/puma/motion_allowed', self._allow_cb, 10)
        self.create_subscription(String, '/puma/planner_status', self._status_cb, 10)

        # Публікатори
        self._preset_pub = self.create_publisher(String, '/puma/plan_request', 10)
```

```

self._joints_pub = self.create_publisher(
    Float64MultiArray, '/puma/target_joints', 10)
self._pose_pub = self.create_publisher(
    PoseStamped, '/puma/target_pose', 10)
self._estop_pub = self.create_publisher(Bool, '/puma/emergency_stop', 10)

self.get_logger().info(f'Демо-вузол запущено, сценарій {self._scenario}')

if self._auto_run:
    self.create_timer(3.0, self._start_once)
self._started = False

def _js_cb(self, msg: JointState):
    if len(msg.position) >= 6:
        self._q = np.array(msg.position[:6])
        self._has_state = True

def _safe_cb(self, msg: Bool):    self._is_safe = msg.data
def _allow_cb(self, msg: Bool):    pass
def _status_cb(self, msg: String):
    self._last_status = msg.data
    self.get_logger().info(f'Статус: {msg.data}')

def _start_once(self):
    if self._started or not self._has_state:
        return
    self._started = True
    {
        1: self._scenario_1,
        2: self._scenario_2,
        3: self._scenario_3,
    }.get(self._scenario, self._scenario_1)()

#

```

```

# Сценарій 1: Підбір і укладання (стандартний)
#

```

```

def _scenario_1(self):
    self.get_logger().info("=== Сценарій 1: Підбір і укладання ===")
    steps = [

```

```

        ("home", "Початкова позиція"),
        ("ready", "Готовність"),
        ("pick_A", "Підбір деталі А"),
        ("ready", "Підйом"),
        ("place_A", "Укладання"),
        ("ready", "Відхід"),
        ("home", "Повернення додому"),
    ]
    self._run_preset_sequence(steps)

#
=====
# Сценарій 2: Тест системи безпеки (навмисні порушення)
#
=====
def _scenario_2(self):
    self.get_logger().info("=== Сценарій 2: Тест системи безпеки ===")

    self.get_logger().info("--- Крок 1: Нормальний рух (має бути ДОЗВОЛЕНО)")
    self._send_preset("home")
    self._wait(self._delay)

    self.get_logger().info("--- Крок 2: Ціль поза робочим простором (має бути ЗАБОРОНЕНО)")
    self._send_pose(2.0, 0.0, 0.5) # далеко за межами
    self._wait(2.0)

    self.get_logger().info("--- Крок 3: Ціль занадто низько (має бути ЗАБОРОНЕНО)")
    self._send_pose(0.3, 0.0, -0.5) # нижче підлоги
    self._wait(2.0)

    self.get_logger().info("--- Крок 4: Ціль всередині перешкоди А (має бути ЗАБОРОНЕНО)")
    self._send_pose(0.5, 0.3, 0.87) # центр перешкоди А
    self._wait(2.0)

    self.get_logger().info("--- Крок 5: Порушення обмежень суглобів (має бути ЗАБОРОНЕНО)")
    bad_q = Float64MultiArray()
    bad_q.data = [0.0, -2.0, 2.0, 0.0, 0.0, 0.0] # joint2,3 поза межами

```

```
self._joints_pub.publish(bad_q)
self._wait(2.0)
```

```
self.get_logger().info("--- Крок 6: Повернення в безпечну позицію")
self._send_preset("home")
self._wait(self._delay)
```

```
self.get_logger().info("=== Тест безпеки завершено ===")
```

```
#
```

```
# Сценарій 3: Обхід перешкод
#
```

```
def _scenario_3(self):
    self.get_logger().info("=== Сценарій 3: Обхід перешкод ===")
    steps = [
        ("home", "Старт"),
        ("up", "Підйом над перешкодами"),
        ("pick_B", "Підбір деталі В (лівий бік)",
        ("up", "Підйом"),
        ("place_B", "Укладання В"),
        ("up", "Відхід"),
        ("home", "Повернення"),
    ]
    self._run_preset_sequence(steps)
```

```
#
```

```
# Утиліти
#
```

```
def _run_preset_sequence(self, steps: list):
    for i, (preset, desc) in enumerate(steps, 1):
        self.get_logger().info(f"[{i}/{len(steps)}] {desc} → {preset}")
        self._send_preset(preset)
        self._wait(self._delay)

    self.get_logger().info("✓ Сценарій завершено")
```

```

def _send_preset(self, name: str):
    msg = String(); msg.data = name
    self._preset_pub.publish(msg)

def _send_pose(self, x: float, y: float, z: float):
    msg = PoseStamped()
    msg.header.stamp = self.get_clock().now().to_msg()
    msg.header.frame_id = 'world'
    msg.pose.position.x = x
    msg.pose.position.y = y
    msg.pose.position.z = z
    msg.pose.orientation.w = 1.0
    self._pose_pub.publish(msg)
    self.get_logger().info(f"Надіслано ціль XYZ: ({x:.2f}, {y:.2f}, {z:.2f})")

def _wait(self, sec: float):
    end = time.time() + sec
    while time.time() < end:
        rclpy.spin_once(self, timeout_sec=0.1)

def main(args=None):
    rclpy.init(args=args)
    node = TrajectoryDemo()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()

```

```

workspace_vis.py
#!/usr/bin/env python3
"""

```

workspace_vis.py — Публікує маркери робочого простору для RViz.
Показує: робочу зону, перешкоди, поточну позицію ЕЕ, стан безпеки.

```
"""
```

```
from __future__ import annotations
```

```
import numpy as np
```

```
import rclpy
```

```
from rclpy.node import Node
```

```
from sensor_msgs.msg import JointState
```

```
from std_msgs.msg import Bool, String, Float64MultiArray
```

```
from visualization_msgs.msg import Marker, MarkerArray
```

```
from geometry_msgs.msg import Point
```

```
from puma560_safe_planner.puma560_kinematics import (
```

```
    DOF, JOINT_NAMES, OBSTACLES,
```

```
    WS_BOX_MIN, WS_BOX_MAX, WS_MIN_RADIUS, WS_MAX_RADIUS,
```

```
    ee_position, link_transforms, full_safety_check,
```

```
    manipulability, min_obstacle_distance,
```

```
)
```

```
class WorkspaceVis(Node):
```

```
    def __init__(self):
```

```
        super().__init__('workspace_vis')
```

```
        self._q = np.zeros(DOF)
```

```
        self._has_state = False
```

```
        self._is_safe = True
```

```
        self._manip = 1.0
```

```
        self._obs_dists = [1.0, 1.0, 1.0]
```

```
        self.create_subscription(JointState, '/joint_states', self._js_cb, 10)
```

```
        self.create_subscription(Bool, '/puma/is_safe', self._safe_cb, 10)
```

```
        self.create_subscription(Float64MultiArray, '/puma/manipulability', self._m_cb, 10)
```

```
        self.create_subscription(Float64MultiArray, '/puma/obstacle_distances', self._d_cb,
```

```
10)
```

```
        self._marker_pub = self.create_publisher(MarkerArray,  
'/puma/workspace_markers', 10)
```

```
        self.create_timer(0.1, self._publish)
```

```
        self.get_logger().info("Візуалізатор робочого простору запущено")
```

```

def _js_cb(self, msg: JointState):
    d = dict(zip(msg.name, msg.position))
    for i, n in enumerate(JOINT_NAMES):
        self._q[i] = d.get(n, self._q[i])
    self._has_state = True

def _safe_cb(self, msg: Bool): self._is_safe = msg.data
def _m_cb(self, msg: Float64MultiArray):
    if msg.data: self._manip = msg.data[0]
def _d_cb(self, msg: Float64MultiArray):
    self._obs_dists = list(msg.data)

def _publish(self):
    if not self._has_state:
        return

    markers = MarkerArray()
    now = self.get_clock().now().to_msg()

```

— Робочий простір (прозорий ящик)

```

m = Marker()
m.header.frame_id = 'world'; m.header.stamp = now
m.ns = 'ws_box'; m.id = 0
m.type = Marker.CUBE; m.action = Marker.ADD
m.pose.position.x = float((WS_BOX_MIN[0] + WS_BOX_MAX[0]) / 2)
m.pose.position.y = float((WS_BOX_MIN[1] + WS_BOX_MAX[1]) / 2)
m.pose.position.z = float((WS_BOX_MIN[2] + WS_BOX_MAX[2]) / 2)
m.pose.orientation.w = 1.0
m.scale.x = float(WS_BOX_MAX[0] - WS_BOX_MIN[0])
m.scale.y = float(WS_BOX_MAX[1] - WS_BOX_MIN[1])
m.scale.z = float(WS_BOX_MAX[2] - WS_BOX_MIN[2])
m.color.b = 0.8; m.color.a = 0.03
markers.markers.append(m)

```

— Перешкоди

```

obs_colors = [(0.9,0.1,0.1), (0.9,0.5,0.1), (0.5,0.1,0.9)]
for i, (obs, col) in enumerate(zip(OBSTACLES, obs_colors)):
    # Тверда куля
    m = Marker()
    m.header.frame_id = 'world'; m.header.stamp = now
    m.ns = 'obstacles'; m.id = i

```

```

m.type = Marker.SPHERE; m.action = Marker.ADD
m.pose.position.x = float(obs[0])
m.pose.position.y = float(obs[1])
m.pose.position.z = float(obs[2])
m.pose.orientation.w = 1.0
m.scale.x = m.scale.y = m.scale.z = float(obs[3] * 2)
m.color.r, m.color.g, m.color.b = col
m.color.a = 0.7
markers.markers.append(m)

```

```

# Зона небезпеки (більша, прозора)
m2 = Marker()
m2.header.frame_id = 'world'; m2.header.stamp = now
m2.ns = 'danger_zones'; m2.id = i
m2.type = Marker.SPHERE; m2.action = Marker.ADD
m2.pose.position.x = float(obs[0])
m2.pose.position.y = float(obs[1])
m2.pose.position.z = float(obs[2])
m2.pose.orientation.w = 1.0
m2.scale.x = m2.scale.y = m2.scale.z = float(obs[3] * 2.5)
m2.color.r, m2.color.g, m2.color.b = col
m2.color.a = 0.12
markers.markers.append(m2)

```

— Ланцюг маніпулятора (лінія)

```

transforms = link_transforms(self._q)
line = Marker()
line.header.frame_id = 'world'; line.header.stamp = now
line.ns = 'robot_chain'; line.id = 0
line.type = Marker.LINE_STRIP; line.action = Marker.ADD
line.scale.x = 0.02
line.color.r = 0.2; line.color.g = 0.6; line.color.b = 1.0; line.color.a = 0.8
for T in transforms:
    p = Point()
    p.x, p.y, p.z = float(T[0,3]), float(T[1,3]), float(T[2,3])
    line.points.append(p)
markers.markers.append(line)

```

— Суглоби (сфери)

```

for i, T in enumerate(transforms):
    jm = Marker()

```

```

jm.header.frame_id = 'world'; jm.header.stamp = now
jm.ns = 'joints'; jm.id = i
jm.type = Marker.SPHERE; jm.action = Marker.ADD
jm.pose.position.x = float(T[0,3])
jm.pose.position.y = float(T[1,3])
jm.pose.position.z = float(T[2,3])
jm.pose.orientation.w = 1.0
r = 0.045 if i == 0 else 0.025
jm.scale.x = jm.scale.y = jm.scale.z = r
jm.color.r = 0.3; jm.color.g = 0.7; jm.color.b = 1.0; jm.color.a = 1.0
markers.markers.append(jm)

```

— Кінцевий ефектор

```

pos = ee_position(self._q)
ee = Marker()
ee.header.frame_id = 'world'; ee.header.stamp = now
ee.ns = 'end_effector'; ee.id = 0
ee.type = Marker.SPHERE; ee.action = Marker.ADD
ee.pose.position.x = float(pos[0])
ee.pose.position.y = float(pos[1])
ee.pose.position.z = float(pos[2])
ee.pose.orientation.w = 1.0
ee.scale.x = ee.scale.y = ee.scale.z = 0.055
ee.color.a = 1.0
if self._is_safe:
    ee.color.r = 0.1; ee.color.g = 1.0; ee.color.b = 0.1 # зелений
else:
    ee.color.r = 1.0; ee.color.g = 0.1; ee.color.b = 0.1 # червоний
markers.markers.append(ee)

```

— Текстова мітка EE

```

txt = Marker()
txt.header.frame_id = 'world'; txt.header.stamp = now
txt.ns = 'ee_label'; txt.id = 0
txt.type = Marker.TEXT_VIEW_FACING; txt.action = Marker.ADD
txt.pose.position.x = float(pos[0])
txt.pose.position.y = float(pos[1])
txt.pose.position.z = float(pos[2]) + 0.08
txt.pose.orientation.w = 1.0
txt.scale.z = 0.04
txt.color.r = txt.color.g = txt.color.b = 1.0; txt.color.a = 1.0

```

```

status = "✓ SAFE" if self._is_safe else "✗ UNSAFE"
txt.text = (
    f"{status}\n"
    f"EE: [{pos[0]:.2f},{pos[1]:.2f},{pos[2]:.2f}]\n"
    f"w={self._manip:.3f}"
)
markers.markers.append(txt)

self._marker_pub.publish(markers)

```

```

def main(args=None):
    rclpy.init(args=args)
    node = WorkspaceVis()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

```

```

if __name__ == '__main__':
    main()

```

Додаток Б

Презентація

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 безпечного планування траєкторії руху робота- маніпулятора Puma560 з урахуванням обмежень робочого простору

Студент IV курсу, групи ТВ-22
Чеповський Дмитро Юрійович

Науковий керівник: д. т. н. проф.
Барабаш Олег Володимирович

КИЇВ – 2026

Актуальність теми

~60 000	промислових робіт в Україні	5 критеріїв безпеки системи
ISO 10218	вимагає гарантій безпеки руху	1 Робочий простір $x \in [-0.9; 0.9]$, $y \in [-0.9; 0.9]$, $z \in [0; 1.5]$ м
100 Гц	частота моніторингу реального часу	2 Радіальні обмеження $r_{\min} = 0.15$ м, $r_{\max} = 0.88$ м
5 критеріїв	перевірки кожного запиту на рух	3 Обмеження суглобів кути та швидкості (joint limits)
		4 Відсутність зіткнень 3 сфери + відступ безпеки 0.05 м
		5 Рухливість суглобів здатність рухатись у всіх напрямках > 0.0001

2

Об'єкт, предмет, мета дослідження

Об'єкт дослідження

Процеси безпечного планування та виконання траєкторій промислових маніпуляторів у робочих середовищах з перешкодами.

Предмет дослідження

Методи та програмне забезпечення надійної перевірки за 5 критеріями безпеки, генерації траєкторій і моніторингу стану маніпулятора Puma560 на базі ROS2 Jazzy.

Мета дослідження

Розробка програмного забезпечення безпечного планування траєкторії маніпулятора Puma560 з надійною перевіркою за 5 критеріями безпеки та моніторингом реального часу 100 Гц.

3

Завдання дипломної роботи

- 1 Провести аналіз та порівняння існуючих методів планування траєкторій маніпуляторів.
- 2 Обґрунтувати вибір засобів розробки та спроектувати архітектуру ПЗ на базі ROS2.
- 3 Розробити та реалізувати математичне ядро кінематики маніпулятора Puma560 (DH, ІК, якобіан).
- 4 Розробити та реалізувати генератор траєкторій на основі математичної формули плавного руху (5-го ступеня).
- 5 Розробити та реалізувати безпечний планувальник з п'ятикритеріальною перевіркою безпеки.
- 6 Розробити та реалізувати монітор безпеки реального часу (100 Гц, згладжування сигналу (ЕМА-фільтр), 3 профілі).
- 7 Розробити та реалізувати симуляційне середовище у Gazebo Harmonic та 3D-візуалізацію RViz2.
- 8 Провести комплексне тестування системи.

4

Засоби розробки

 ROS2 Jazzy Мета-ОС для роботів DDS middleware, pub/sub, ros2_control, TF2	 Gazebo Harmonic 8.11.0 3D симулятор Bullet Featherstone, gz_ros2_control, STL-meshi	 Python 3.12 + NumPy Вузли ROS2 safe_planner, safety_monitor, vis
 C++ 17 + Eigen3 Кінематика / RRT* DH-матриці, якобіан, SIMD-оптимізація	 Ubuntu 24.04 VSCode Середовище colcon build, Git, ROS2 Tools ext.	 RViz2 Візуалізація 3D workspace markers, траєкторії, EE-статус

6

Аналіз існуючих методів

Потенційні поля	RRT* / MoveIt2	Квінтичний поліном (розроблена система)
Переваги + Час відгуку < 1 мс (найшвидший) + Проста реалізація + Підходить для реального часу	Переваги + Знаходить найкоротший шлях з часом + Ймовірна повнота (знайде шлях) + Готова реалізація у MoveIt2	Переваги + 100% успішність виявлення загроз + Детермінований час 19–82 мс + Плавний рух — без ривків і різких змін прискорення
Недоліки – Локальні мінімуми → 30% відмов – Не гарантує досягнення цілі – Складна перевірка динамічних обмежень	Недоліки – Стохастичний: 31–820 мс (розкид!) – Логіка безпеки прихована в SRDF – Складне налаштування для Puma560	Недоліки – Лише заздалегідь задані цілі – Не обходить перешкоди автоматично – Вимагає явного опису перешкод

Висновок: квінтичний поліном забезпечує детерміновану перевірку всіх критеріїв безпеки з передбачуваним часом відгуку.

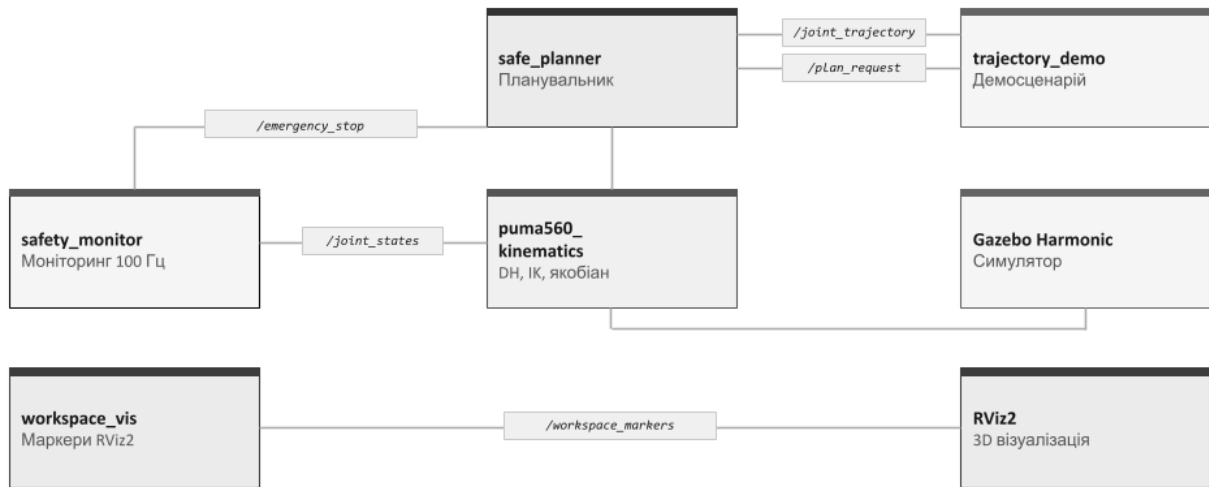
5

Засоби розробки

 ROS2 Jazzy Мета-ОС для роботів DDS middleware, pub/sub, ros2_control, TF2	 Gazebo Harmonic 8.11.0 3D симулятор Bullet Featherstone, gz_ros2_control, STL-meshi	 Python 3.12 + NumPy Вузли ROS2 safe_planner, safety_monitor, vis
 C++ 17 + Eigen3 Кінематика / RRT* DH-матриці, якобіан, SIMD-оптимізація	 Ubuntu 24.04 VSCode Середовище colcon build, Git, ROS2 Tools ext.	 RViz2 Візуалізація 3D workspace markers, траєкторії, EE-статус

6

Архітектура програмного забезпечення DFD



7

Діаграми проектування ПЗ

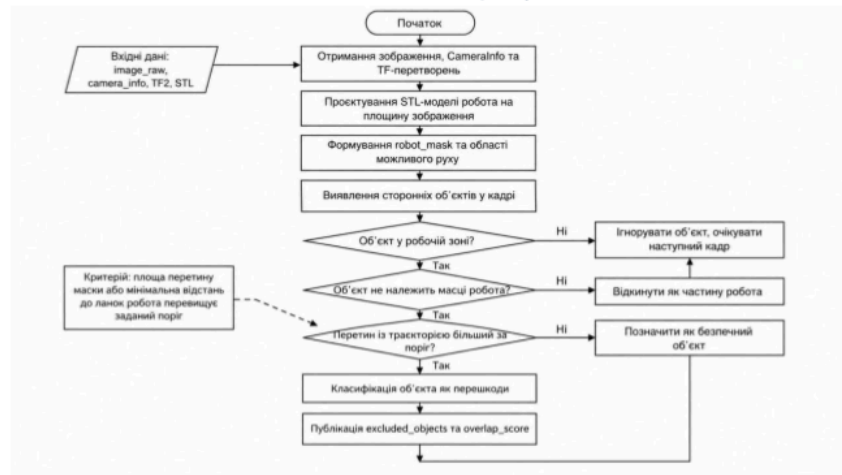
Діаграма прецедентів



8

Діаграми проектування ПЗ

Блок-схема алгоритму



9

Структура програмного забезпечення

Два пакети ROS2:

puma560_description

- ▶ puma560.urdf.xacro — 3D-модель маніпулятора
- ▶ puma560_controllers.yaml — налаштування регуляторів руху
- ▶ simulation_puma560.xml — launch-файл Gazebo Harmonic + ros2_control

puma560_safe_planner

- ▶ puma560_kinematics.py — DH, FK, IK, якобіан, маніпулятивність
- ▶ safe_planner_node.py — планувальник (10-кроковий детерміністичний алгоритм)
- ▶ safety_monitor_node.py — монітор 100 Гц, ЕМА-фільтрація, 3 профілі безпеки
- ▶ workspace_vis_node.py — 7 типів маркерів RViz2
- ▶ trajectory_demo.py — демосценарій pick-and-place (8 кроків)

10

Модуль кінематики Puma560

Параметри ДН маніпулятора Puma560 (таблиця 1.1):

Ланка	a (м)	d (м)	α (рад)
1	0.0	0.0	$-\pi/2$
2	0.432	0.0	0
3	-0.02	0.150	$-\pi/2$
4	0.0	0.4318	$\pi/2$
5	0.0	0.0	$-\pi/2$
6	0.0	0.0	0

Де: a — зсув по x, d — зсув по z, α — кут повороту по x, θ — кут суглоба

Ключові формули:

Пряма кінематика (FK)

$$T_{06} = T_1 \cdot T_2 \cdot T_3 \cdot T_4 \cdot T_5 \cdot T_6$$

T_{0i} — матриця 4x4 положення кінцевого ефектора (руки) відносно основи

ДН-матриця ланки i

$$T(i-1, i) = Rot_z(\theta) \cdot Trans_z(d) \cdot Trans_x(a) \cdot Rot_x(\alpha)$$

θ — кут суглоба; a, d, α — геометричні параметри ланки

Обернена кінематика (IK)

$$\Delta q = J^T \cdot (J \cdot J^T + \lambda I)^{-1} \cdot \Delta x, \quad \lambda = 0.01$$

Δx — відхилення від цілі; λ — коефіцієнт стабілізації

Маніпулятивність

$$w(q) = \sqrt{\det(J \cdot J^T)}$$

$w \rightarrow 0$ означає небезпечне положення — рух неможливий

FK похибка: 8.3×10^{-7} м | IK похибка (95%): ≤ 0.5 мм

11

Генератор траєкторій — квінтичний поліном

Квінтичний поліном 5-го ступеня

Формула траєкторії:

$$q(t) = q_0 + \Delta q \cdot (10t^3 - 15t^4 + 6t^5)$$

де $t = t/T$, $\Delta q = q_{goal} - q_{start}$

Пояснення змінних:

q_0 — початкові кути суглобів
 q_{goal} — цільові кути суглобів
 T — час виконання (автомасштаб)
 t — нормований час [0; 1]

Граничні умови (плавний рух (без ривків і стрибків прискорення)):

$$q(0) = q_0, \quad q(T) = q_{goal}$$

$$\dot{q}(0) = \dot{q}(T) = 0 \quad (\text{нульова швидкість})$$

$$\ddot{q}(0) = \ddot{q}(T) = 0 \quad (\text{нульове прискорення})$$

Порівняння методів інтерполяції

Метод	Неперер.	Ривки	М'який старт
Лінійна	C^0	Так	Ні
Кубічна	C^1	Можливі	Частково
Квінтична ✓	C^2	Ні ✓	Так ✓

Масштабування часу T:

$$T \geq \max(|\Delta q_i| / v_{max_i}) / 0.7$$

Де Δq_i — зміна i-го суглоба, v_{max_i} — максимальна швидкість i-го суглоба.
 Гарантується $T \geq 3.0$ с для плавного виконання.

12

Алгоритм безпечного планування траєкторії

Детерміністичний алгоритм обробки кожного запиту на рух:

1	Перевірити сигнал аварійної зупинки /emergency_stop	(заблокувати, якщо True)
2	Читати поточний стан суглобів з /joint_states → current_q	(вектор 6 значень)
3	Якщо ціль задана у просторі XYZ — перерахувати кути суглобів (IK) → q_goal	(≤200 ітерацій)
4	Перевірити q_goal: робочий простір + обмеження суглобів	(критерії 1, 2, 3)
5	Побудувати траєкторію квінтичним поліномом (10 точок), масштабувати час	(T ≥ 3 с)
6	Перевірити кожну точку: зіткнення + швидкість	(критерії 4, 5)
7	При порушенні → знизити v×0.5, повторити кроки 5–6 (≤2 спроби)	(авторезерв)
8	Надіслати команду руху або заблокувати з детальним звітом	(журнал рішення)

13

Монітор безпеки реального часу

Згладжування швидкостей суглобів (ЕМА-фільтр):

$$dq_filtered = \alpha \cdot dq_raw + (1-\alpha) \cdot dq_prev, \quad \alpha = 0.3$$

Де: dq_raw — виміряна швидкість, dq_prev — попереднє значення, $\alpha = 0.3$ — сила згладжування.
Шум знижується з 0.097 рад/с до 0.031 рад/с (коефіцієнт пригнічення 3.1).

5 критеріїв кожного циклу (100 Гц):

- 1 Межі прямокутного робочого простору
- 2 Радіальні обмеження (r_min/r_max)
- 3 Обмеження кутів суглобів
- 4 Відсутність зіткнень з перешкодами
- 5 Рухливість суглобів (здатність рухатись) > 0.0001

Профілі безпеки (таблиця 3.3):

Параметр	normal	conser-vative	teaching
Відступ (м)	0.05	0.075	0.02
Швидкість	100%	70%	50%
E-stop	Так	Так	Hi
Частота (Гц)	100	100	50

Стабільність за 60 с:

99.97 Гц середня | CPU ≤ 4% | $\sigma = 1.84$ Гц

14

Симуляційна сцена Gazebo Harmonic

Обмеження робочого простору

Вісь / обмеження	Мін.	Макс.
X (м)	-0.9	0.9
Y (м)	-0.9	0.9
Z (м)	0.0	1.5
r радіус (м)	0.15	0.88

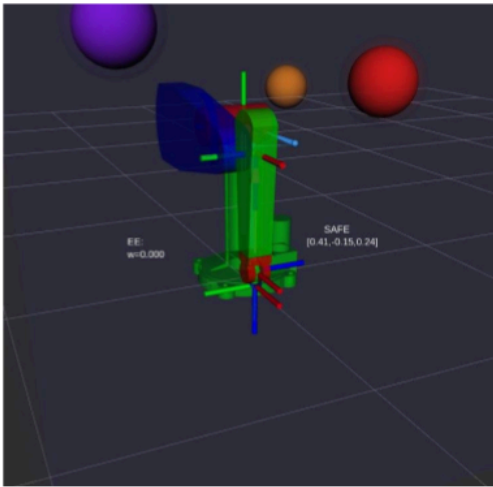
Перешкоди (сфери + відступ 0.05 м)

А (червона) центр: (0.50; 0.30; 0.87) r = 0.12 м
В (помаранчева) центр: (-0.40; 0.40; 0.75) r = 0.10 м
С (фіолетова) центр: (0.00; -0.50; 1.05) r = 0.15 м

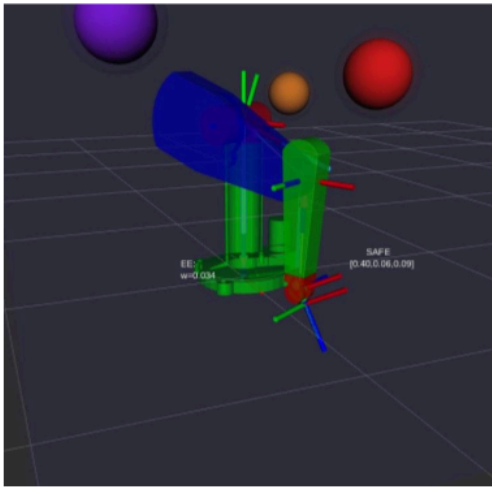
Пресети конфігурацій (таблиця 3.2)

Пресет	j1	j2	j3	Призначення
home	0.0	0.0	0.0	Паркова
ready	0.0	-0.5	0.8	Підготовка
pick_A	0.5	-0.8	1.0	Підбір право
pick_B	-0.6	-0.8	1.0	Підбір ліво
place_A	0.8	-0.4	0.6	Укладання право
place_B	-0.9	-0.4	0.6	Укладання ліво
up	0.0	-1.0	0.3	Підйом
look	0.0	-0.3	0.5	Огляд

Тестування



home



pick_A

Тестування

```
safe_planner-1 [INFO] [1778744521.825182567] [safe_planner]:
safe_planner-1 3011 БЕЗПЕЧНОГО ПЛАНУВАЛЬНИКА
safe_planner-1 Крок 3: Перевірка робочого простору цілі
safe_planner-1 ✓ EE цілі: [ 0.647 -0.381 0.823]
safe_planner-1 Крок 4: Обернена кінематика – OK
safe_planner-1 Крок 5: Обмеження сугубіа цілової конфігурації
safe_planner-1 ✓ Усі сугубіа в допустимих межах
safe_planner-1 Крок 6: Побудова кінтичної траєкторії
safe_planner-1 Точок: 30, тривалість: 1.92 с
safe_planner-1 Крок 7-8: Перевірка траєкторії
safe_planner-1 ✓ Усі 30 точок безпечні
safe_planner-1 Крок 9: Надсилання команди контролеру
safe_planner-1 C-відстань: 1.7682 рад
safe_planner-1 EE відстань: 0.6141 м
safe_planner-1 Маніпулятивність: 0.8294
safe_planner-1 Відстань до переходу: 0.4578 м
safe_planner-1 Час планування: 172.7 мс
safe_planner-1 РЕЗУЛЬТАТ: ✓ ДОЗВОЛЕНО – Траєкторія виконується (30 точок, 1.92 с)
safe_planner-1 [INFO] [1778744521.838216163] [safe_planner]: [STATUS] ✓ ДОЗВОЛЕНО: Траєкторія виконується (30
safe_planner-1 точок, 1.92 с)
trajectory_demo-5 [INFO] [1778744521.839152686] [trajectory_demo]: Статус: ✓ ДОЗВОЛЕНО: Траєкторія виконується
(30 точок, 1.92 с)
```

Дозвіл на рух

```
safe_planner-1 [INFO] [1778744371.764173807] [safe_planner]: [STATUS] ✗ РУХ ЗАБОРОНЕНО: Невідомий пресет 'pick_C'. Доступні: ['home', 'ready', 'pick_A', 'pick_B', 'place_A', 'place_B', 'up', 'look']
trajectory_demo-5 [INFO] [1778744371.766540557] [trajectory_demo]: Статус: ✗ РУХ ЗАБОРОНЕНО: Невідомий пресет
r 'pick_C'. Доступні: ['home', 'ready', 'pick_A', 'pick_B', 'place_A', 'place_B', 'up', 'look']
safe_planner-1 [WARN] [1778744371.776244792] [safe_planner]: ЗАБЛОКОВАНО: Невідомий пресет 'pick_C'. Доступні:
['home', 'ready', 'pick_A', 'pick_B', 'place_A', 'place_B', 'up', 'look']
```

Заборона руху

17

Результати тестування

Часові характеристики (мс) — таблиця 4.3:



Інтеграційне тестування (11 сценаріїв) — таблиця 4.2:

Тип	Сценарій	Результат
Небезпечний	Поза x_max	✓ Відхилено 18 мс
Небезпечний	Поза r_min	✓ Відхилено 22 мс
Небезпечний	Порушення joint2	✓ Відхилено 15 мс
Небезпечний	Зіткнення з в	✓ Відхилено 31 мс
Безпечний	home → ready	✓ 3.2 с
Безпечний	pick_A → up	✓ 4.1 с
Безпечний	up → place_A	✓ 3.8 с
Граничний	x_max – 0.001	✓ Виконано 38 мс
Особливий	E-stop у русі	✓ < 8 мс

100%

виявлення небезпечних

99.97 Гц

стабільність монітора

≤ 4%

завантаження CPU

55–65 с

повний демосценарій

18

Висновки

- 1 Проаналізовано 3 класи алгоритмів. Квінтичний поліном: 100% надійність та час 19–82 мс.
- 2 Спроектовано модульну архітектуру ROS2 що включає 4 вузли та 9 топіків для обміну даними. (окерний програмний модуль ROS2)
- 3 Реалізовано математичний модуль кінематики Puma560: пряма кінематика (похибка 8.3×10^{-7} м), обернена кінематика (похибка ≤ 0.5 мм).
- 4 Реалізовано safe_planner з перевіркою за 5 критеріями безпеки: 100% успішність.
- 5 Реалізовано монітор безпеки: 99.97 Гц, CPU $\leq 4\%$, фільтр згладжування, 3 профілі безпеки.
- 6 Симуляція Gazebo + RViz2. Успішне виконання тестових сценаріїв.
- 7 Порівняльний аналіз: 100% успішності vs 70% у потенційних полів; час 19–82 мс vs 31–820 мс у MoveIt2.
- 8 ПЗ може використовуватись як навчальний стенд або основа для реальних ROS2-маніпуляторів.

19

Дякую за увагу!