ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Метод управління безпілотним роботом на основі штучного інтелекту»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Іван ДУМЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-63
Іван ДУМЕНКО

Керівник: Ірина ЗАМРІЙ
д.т.н., доцент

Рецензент: _____
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Думенко Івану Олексійовичу

1. Тема кваліфікаційної роботи: «Метод управління безпілотним роботом на основі штучного інтелекту»

керівник кваліфікаційної роботи Ірина ЗАМРІЙ, д.т.н., доц., завідувач кафедри

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024 р. № 320.

2. Строк подання кваліфікаційної роботи «17» грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з робототехніки, методології управління автономними системами, параметри алгоритмів штучного інтелекту для детекції та класифікації об'єктів, технічні вимоги до розробки автономного методу управління, специфікації симуляційного середовища і фреймворку ROS2, а також нормативні документи, що регулюють вимоги до автономних робототехнічних систем.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження методів детектування об'єктів у комп'ютерному зорі.

2. Аналіз сучасних технологій штучного інтелекту для навігації роботів.
3. Розробка та валідація методу управління автономним роботом.

5. Перелік ілюстративного матеріалу: *презентація*

1. Вибір моделі штучного інтелекту для управління роботом.
2. Діаграма структури класів алгоритму управління.
3. Математична модель алгоритму управління роботом.
4. Симуляція роботи навігації робота в середовищі Gazebo.
5. Тестування розпізнавання об'єктів за допомогою моделі YOLOv5.
6. Таблиця результатів порівняння методів управління.
7. Графічні демонстрації результатів моделювання та тестування.

6. Дата видачі завдання «16» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.10. - 23.10.2024	Виконано
2	Вивчення матеріалів для аналізу розвитку технологій автономної навігації та управління безпілотними роботами	24.10. – 30.10.2024	Виконано
3	Дослідження методів штучного інтелекту для навігації та уникнення перешкод	31.10. – 06.11.2024	Виконано
4	Аналіз особливостей впливу методів штучного інтелекту на точність та ефективність навігації безпілотного робота	07.11. – 13.11.2024	Виконано
5	Дослідження впровадження технологій AI у системах автономного управління	14.11. – 17.11.2024	Виконано
6	Застосування методів штучного інтелекту для оптимізації маршрутів і уникнення перешкод безпілотним роботом	18.11. – 20.11.2024	Виконано
7	Оформлення роботи: вступ, висновки, реферат	21.11.-23.11.2024	Виконано
8	Розробка демонстраційних матеріалів	24.11.-27.11.2024	Виконано
9	Попередній захист роботи	28.11.-15.12.2024	Виконано

Здобувач вищої освіти _____ Іван ДУМЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ЗАМРІЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 74 стор., 3 табл., 32 рис., 38 джерел.

Мета роботи – підвищення ефективності та автономності безпілотного робота в різних умовах експлуатації за допомогою штучного інтелекту.

Об'єкт дослідження – процес управління безпілотним роботом.

Предмет дослідження – алгоритми і методи управління безпілотними роботами, засновані на штучному інтелекті.

У роботі проведено всебічний аналіз сучасних підходів до розробки методів управління роботами на основі штучного інтелекту. Розглянуто та узагальнено існуючі технології локалізації та картографування (SLAM), використання фільтра Калмана для покращення точності та ефективності, а також фреймворків ROS2 та їхніх інструментів. Виконано порівняльний аналіз алгоритмів детектування та трекінгу об'єктів, що використовуються у робототехнічних системах.

Розроблено та оптимізовано метод управління автономним роботом, який базується на інтеграції штучного інтелекту для навігації в складних середовищах. Використано фреймворки ROS2 та Nav2 для побудови навігаційної системи з можливістю адаптації до динамічних умов.

Проведено серію експериментів для оцінки точності розробленого методу, його ефективності в умовах реального середовища та адаптивності до змінних сценаріїв.

Отримані результати підтвердили переваги розробленого методу в порівнянні з традиційними підходами до управління роботами.

КЛЮЧОВІ СЛОВА: УПРАВЛІННЯ РОБОТАМИ, КОМП'ЮТЕРНИЙ ЗІР, ГЛИБОКЕ НАВЧАННЯ, ДЕТЕКТУВАННЯ ОБ'ЄКТІВ, НАВІГАЦІЯ РОБОТІВ, ШТУЧНИЙ ІНТЕЛЕКТ, ФІЛЬТР КАЛМАНА, СИСТЕМА ROS2.

ABSTRACT

Text part of the master's qualification work: 74 pages, 32 pictures, 3 table, 38 sources.

The aim of the work - is to improve the efficiency and autonomy of an unmanned robot under various operating conditions using artificial intelligence.

The object of the study - is the process of controlling an unmanned robot.

The subject of the study - is the algorithms and methods for controlling unmanned robots based on artificial intelligence.

The study provides a comprehensive analysis of modern approaches to developing robot control methods based on artificial intelligence. Existing technologies for simultaneous localization and mapping (SLAM), the use of the Kalman filter to enhance accuracy and efficiency, as well as ROS2 frameworks and their tools, are reviewed and summarized. A comparative analysis of object detection and tracking algorithms used in robotic systems is carried out.

A method for controlling an autonomous robot based on the integration of artificial intelligence for navigation in complex environments has been developed and optimized. The ROS2 and Nav2 frameworks were employed to build a navigation system capable of adapting to dynamic conditions.

A series of experiments was conducted to evaluate the accuracy of the developed method, its effectiveness in real-world conditions, and its adaptability to changing scenarios. The obtained results confirmed the advantages of the developed method compared to traditional approaches to robot control.

KEYWORDS: ROBOT CONTROL, COMPUTER VISION, DEEP LEARNING, OBJECT DETECTION, ROBOT NAVIGATION, ARTIFICIAL INTELLIGENCE, KALMAN FILTER, ROS2 SYSTEM.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ..... 11
ВСТУП..... 12
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ..... 14
1.1 Безпілотні роботи: класифікація та застосування..... 14
1.2 Методи управління безпілотними роботами..... 16
1.3 Використання штучного інтелекту в управлінні роботами..... 19
1.4 Сучасні моделі ШІ для розпізнавання об’єктів..... 21
1.5 Вибір інструментів та алгоритмів для реалізації методу..... 28
2 АНАЛІЗ МЕТОДІВ УПРАВЛІННЯ БЕЗПІЛОТНИМИ РОБОТАМИ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ..... 35
2.1 Аналіз існуючих методів управління безпілотними роботами на основі штучного інтелекту..... 35
2.2 Аналіз переваг та недоліків методів управління безпілотними роботами 39
2.3 Середовище моделювання та візуалізації..... 50
2.4 Аналіз сфер діяльності..... 53
3 РОЗРОБКА МЕТОДУ УПРАВЛІННЯ БЕЗПІЛОТНИМ РОБОТОМ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ..... 61
3.1 Архітектура програмного забезпечення для управління безпілотним роботом..... 61
3.2 Математична модель методу управління безпілотним роботом на основі штучного інтелекту..... 63
3.3 Опис розробки методу..... 68
3.4 Розробка апаратного забезпечення..... 81
4 РЕЗУЛЬТАТИ ЗАСТОСУВАННЯ МЕТОДУ УПРАВЛІННЯ РОБОТОМ НА ОСНОВІ МОДЕЛІ ШІ..... 83
4.1 Результати тестування..... 83
ВИСНОВКИ..... 88
ПЕРЕЛІК ПОСИЛАНЬ..... 90
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)..... 93
ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ..... 101

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI – Штучний інтелект (Artificial Intelligence).

YOLO – Модель "Дивись тільки раз" для ідентифікації об'єктів (You Only Look Once).

ROS – Операційна система для роботів (Robot Operating System).

SLAM – Одночасна локалізація та картографування (Simultaneous Localization and Mapping).

EKF – Розширений фільтр Калмана (Extended Kalman Filter).

RNN – Рекурентна нейронна мережа (Recurrent Neural Network).

CNN – Згорткова нейронна мережа (Convolutional Neural Network).

GPS – Глобальна система позиціонування (Global Positioning System).

IMU – Інерційний вимірювальний пристрій (Inertial Measurement Unit).

Nav2 – Фреймворк для навігації в ROS2 (Navigation 2 Framework).

URDF – Уніфікований формат опису роботів (Unified Robot Description Format).

Rviz – Інструмент візуалізації даних в ROS2 (ROS Visualization Tool).

FPN – Особливість пірамідної мережі (Feature Pyramid Network).

mAP – Середня точність (Mean Average Precision).

IoU – Перетин через об’єднання (Intersection over Union).

ВСТУП

З розвитком сучасних технологій та швидкими темпами впровадження штучного інтелекту постає необхідність створення нових підходів до управління автономними системами. Однією з ключових галузей сучасних досліджень є розробка методів управління безпілотними роботами, що забезпечують їхню автономність, ефективність та адаптивність до змінних умов.

Використання штучного інтелекту в управлінні безпілотними роботами відкриває нові можливості для підвищення точності, швидкості прийняття рішень та здатності до навчання в реальному часі. Зокрема, застосування сучасних алгоритмів глибокого навчання, таких як YOLOv5, дозволяє досягти значного прогресу в задачах детектування об’єктів, аналізу середовища та побудови оптимальних маршрутів для безпілотних систем.

Даний диплом присвячений аналізу сучасних методів управління безпілотними роботами на основі штучного інтелекту, розробці математичної моделі методу управління, а також його практичній реалізації. У результаті роботи виконано детальний аналіз існуючих підходів, визначено їх переваги та недоліки, а також запропоновано новий метод, що забезпечує підвищення ефективності управління в умовах реального середовища.

Таким чином, завдання розробки методу управління безпілотним роботом на основі штучного інтелекту є актуальним і важливим для розвитку робототехніки, автономних систем та їх інтеграції в сучасний технологічний ландшафт.

Мета роботи – підвищення ефективності та автономності безпілотного робота в різних умовах експлуатації за допомогою штучного інтелекту.

Об’єкт дослідження – процес управління безпілотним роботом.

Предмет дослідження – алгоритми і методи управління безпілотними роботами, засновані на штучному інтелекті.

Для досягнення мети вирішувалися наступні завдання:

1. Проведено аналіз науково-технічної літератури з метою вивчення сучасних підходів до управління автономними роботами. Розглянуто методи SLAM, фільтра Калмана, технології комп'ютерного зору та особливості використання штучного інтелекту для адаптивної навігації.
2. Розроблено математичну модель методу управління автономним роботом, яка базується на інтеграції методів локалізації, картографування та штучного інтелекту. Реалізовано програмну складову методів у середовищі ROS2 із використанням фреймворків Nav2 та Gazebo.
3. Проведено тестування розробленого методу на симуляційних даних та у реальному середовищі. Вивчено ефективність алгоритмів детектування та трекінгу об'єктів, адаптивності навігації до змінних умов.
4. Застосовано статистичні методи для оцінки точності та продуктивності запропонованого методу у порівнянні з традиційними підходами.
5. Виконано оптимізацію розробленого методу для підвищення його точності, ефективності та адаптивності до динамічних середовищ.

Завдання виконані в рамках розробки інноваційного методу управління автономним роботом, що базується на штучному інтелекті, підтверджують актуальність і практичну цінність роботи для подальшого розвитку автономних систем.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Безпілотні роботи: класифікація та застосування

Безпілотні роботи (БПР) є ключовим компонентом сучасної автоматизації, їх застосування постійно розширюється завдяки впровадженню новітніх технологій [1]. Вони поділяються на кілька категорій залежно від сфери використання, способу управління та конструктивних особливостей.

Основні класифікації безпілотних роботів включають:

1) **Наземні безпілотні роботи (UGV – Unmanned Ground Vehicles)**: використовуються для дослідження територій, виконання будівельних, транспортних чи військових завдань.
2) **Повітряні безпілотні роботи (UAV – Unmanned Aerial Vehicles):** широко застосовуються у сільському господарстві, геодезії, логістиці та обороні [2].
3) **Підводні безпілотні роботи (UUV – Unmanned Underwater Vehicles):** призначені для виконання завдань у водному середовищі, у морській розвідці або технічному обслуговуванні інфраструктури.

Таблиця 1.1

Таблиця класифікації безпілотних роботів

Критерій класифікації	Категорії	Приклади застосування
За середовищем функціонування	- Наземні роботи (UGV) - Повітряні роботи (UAV) - Водні роботи (AUV/ROV) - Космічні роботи	Наземні роботи: логістика, сільське господарство Повітряні: моніторинг, військові операції Водні: підводні дослідження

Продовження таблиці 1.1

Таблиця класифікації безпілотних роботів

Критерій класифікації	Категорії	Приклади застосування
За рівнем автономності	- Телекеровані - Напівавтономні - Автономні	Телекеровані: операції в небезпечних умовах Напівавтономні: допомога оператору Автономні: дослідницькі роботи
За функціональним призначенням	- Розвідувальні - Логістичні - Військові - Медичні - Промислові	Розвідувальні: зйомка місцевості Логістичні: автоматичні склади Медичні: доставка ліків, операції в зонах лиха
За джерелом енергії	- Електричні - Паливні - Гібридні	Електричні: дрони на батареях Паливні: роботи для тривалих місій Гібридні: оптимальне поєднання енергоспоживання
За типом конструкції	- Колісні - Гусеничні - Ногасті (гуманоїдні) - Летючі	Колісні: автономні автомобілі Гусеничні: роботи для пересіченої місцевості Летючі: квадрокоптери
За системою управління	- Програмовані - Навчаємі (на основі ШІ)	Програмовані: роботи з фіксованими алгоритмами Навчаємі: роботи з адаптацією до умов
За типом взаємодії з навколишнім середовищем	- Роботи з маніпуляторами - Роботи-спостерігачі - Роботи-виконавці	Маніпулятори: збір деталей Спостерігачі: моніторинг середовища Виконавці: виконання завдань

Застосування безпілотних роботів охоплює такі ключові напрями:

1) **Сільське господарство:** автоматизований посів, обприскування та збирання врожаю.
2) **Медицина:** роботи-асистенти в операційних, транспортування медикаментів у лікарнях.
3) **Промисловість:** контроль якості продукції, автоматизація виробничих процесів.
4) **Охорона та безпека:** моніторинг територій, виявлення та запобігання загрозам.
5) **Військова сфера:** розвідка, логістика, підтримка бойових операцій, знешкодження вибухонебезпечних об'єктів [1].

Безпілотні роботи є основою для розвитку новітніх технологій у багатьох галузях, забезпечуючи оптимізацію процесів, підвищення продуктивності та безпеки [2].

1.2 Методи управління безпілотними роботами

Ефективне управління безпілотними роботами є одним із ключових аспектів їх успішного застосування в різних галузях. Сучасні методи управління базуються на інтеграції сенсорних систем, алгоритмів обробки даних та штучного інтелекту (ШІ), що забезпечує високу точність, швидкість реакції та автономність.

Основні методи управління:

1) Ручне управління

Ручне управління є найпростішою формою контролю безпілотних роботів. Оператор взаємодіє з роботом за допомогою пультів, джойстиків або спеціалізованих програм на комп’ютері чи мобільному пристрої. Такий підхід є найбільш підходящим для завдань, що потребують високої точності або творчого підходу, які важко автоматизувати.

- Технічні особливості: ручне управління забезпечує передачу команд у режимі реального часу через дротові або бездротові інтерфейси, такі як радіозв'язок або Wi-Fi. Часто в системі використовується візуальний зворотній зв'язок, наприклад, зображення з камер, установлених на роботі.
- Обмеження:
    - Відсутність автономності вимагає постійної участі оператора, що може обмежувати масштабування застосування роботів.
    - Залежність від швидкості реакції людини робить цей метод менш ефективним у швидко змінних умовах.
- Приклад використання: дрони для аерофотозйомки або ручні роботи для розмінування. Такі системи дозволяють людині виконувати завдання, які є надто небезпечними або складними для автономних алгоритмів [3].

2) Автономне управління

Автономне управління передбачає, що робот самостійно приймає рішення та виконує завдання без втручання оператора. Цей підхід базується на використанні алгоритмів штучного інтелекту, які дозволяють аналізувати дані в реальному часі, будувати карти місцевості та планувати дії.

- **Особливості:**
    - Використання сенсорів для сприйняття навколишнього середовища (лідарів, ультразвукових датчиків, камер).
    - Інтеграція з алгоритмами розпізнавання об’єктів (наприклад, YOLOv5), які дозволяють швидко визначати перешкоди та шляхи їх обходу.
- **Переваги:**
    - Автономність дозволяє роботам працювати у важкодоступних або небезпечних умовах (глибоке море, космос, радіоактивні зони).
    - Зменшення впливу людського фактора на результати роботи.
- **Недоліки:**
    - Складність технічної реалізації.

- Можливість помилок у системах прийняття рішень у разі незапланованих ситуацій.

- **Приклад використання:** Автономні роботи для сільського господарства, які самостійно виконують обприскування чи збирання врожаю, або розвідувальні військові дрони [4].

3) Напівавтономне управління

Напівавтономне управління є компромісом між ручним і автономним підходами. У цьому випадку оператор задає загальні команди, а робот виконує їх, використовуючи автономні функції для вирішення деталей.

- **Технічні аспекти:** такі системи можуть включати інтерфейси доповненої реальності, де оператор задає віртуальні точки для виконання завдань, а робот аналізує шляхи та перешкоди для їх досягнення.

- **Особливості:**
  - Робот забезпечує виконання рутини (навігація, уникнення перешкод), тоді як оператор бере участь лише у ключових моментах.
  - Потребує менше обчислювальних ресурсів, ніж повністю автономні системи.

- **Переваги:**
  - Ефективність за рахунок зменшення навантаження на оператора.
  - Менший ризик помилок завдяки підтримці оператора у складних ситуаціях.

- **Приклад використання:** Медичні дрони для доставки препаратів: оператор задає загальні маршрути, а дрон самостійно обирає оптимальний шлях з урахуванням погодних умов і перешкод [5]

1.3 Використання штучного інтелекту в управлінні роботами

Штучний інтелект (ШІ) є рушійною силою сучасної робототехніки, спрямовуючи її до автономії, адаптивності та ефективності. В основі інтеграції ШІ у робототехніку лежать наступні завдання:

1. ідентифікація об'єктів: виявлення, класифікація та визначення розташування об’єктів у середовищі. Завдяки цьому роботи здатні уникати перешкод, взаємодіяти з цілями або об'єктами
2. навігація: створення маршрутів і їх адаптація відповідно до змін у середовищі, враховуючи статичні та динамічні перешкоди
3. прийняття рішень: аналіз отриманих даних і вибір оптимальної дії для досягнення цілі
4. адаптивність: оновлення поведінки відповідно до змін у зовнішніх умовах, таких як поява нових перешкод чи зміна маршруту

Комп’ютерний зір є важливою складовою автономних роботів, дозволяючи їм "бачити" та аналізувати навколишнє середовище. За допомогою сучасних моделей глибокого навчання роботи досягають значних результатів у таких завданнях:

1. розпізнавання об’єктів у реальному часі
2. класифікація об’єктів за заданими категоріями
3. визначення їхнього розташування у просторі

Моделі, які використовуються:

- YOLOv8 – забезпечує точність понад 50% mAP на складних наборах даних, таких як COCO, і швидкість обробки, що робить її ідеальною для систем відеоспостереження [12]
- Faster R-CNN – демонструє високу точність для складних сцен із багатьма об’єктами, але має більші ресурсоємні вимоги [11]

Завдяки цим моделям роботи можуть ефективно аналізувати сцени та ухвалювати рішення на основі отриманих даних.

Методи навігації, які базуються на ШІ, дозволяють роботам знаходити оптимальні маршрути навіть у складних динамічних середовищах:

1. посилене навчання (Reinforcement Learning): алгоритми, що навчаються через експерименти, отримуючи винагороди за правильні дії. Це дозволяє роботам адаптуватися до непередбачуваних змін у середовищі
2. генетичні алгоритми: оптимізація маршруту через імітацію біологічних процесів, таких як мутація та відбір [9]
3. прогнозування траєкторій: нейронні мережі прогнозують рух інших об’єктів для уникнення зіткнень або затримок

Ці методи знаходять застосування в робототехніці, зокрема в автономному транспортуванні, патрулюванні або рятувальних операціях.

ШІ дозволяє не лише реагувати на поточні умови, але й прогнозувати можливі зміни:

1. рух об’єктів, наприклад, пішоходів або автомобілів
2. зміну умов, таких як освітлення чи погода
3. збої у роботі сенсорів або інших компонентів системи

Рекурентні нейронні мережі (RNN) та трансформерні архітектури (наприклад, Vision Transformers) дозволяють обробляти послідовності даних і прогнозувати динаміку середовища, що є особливо важливим у складних умовах роботи [7].

Сучасні роботи покладаються на великий обсяг даних, отриманих від різних сенсорів:

1. **LiDAR:** створює 3D-карту середовища для точного визначення розташування об’єктів
2. **GPS:** забезпечує глобальну навігацію
3. **IMU (інерційні модулі):** відстежує положення робота в просторі

Об’єднання сенсорних даних із комп’ютерним зором на основі ШІ забезпечує більш точне управління. Наприклад, використання YOLOv8 разом із LiDAR дозволяє покращити роботу робота в умовах поганої освітленості або високої щільності об’єктів [8].

Таблиця 1.2

Таблиця порівняння методів ШІ в управлінні роботами

Метод	Опис	Переваги	Недоліки	Джерело
YoloV5	Реально вчасна ідентифікація об’єктів	Швидкість, точність, гнучкість	Високі обчислювальні вимоги	[6]
Faster R-CNN	Детектування об'єктів з високою точністю	Висока точність	Повільніше за YOLOv5	[7]
Reinforcement Learning (RL)	Навчання через нагороди за дії	Адаптивність до нового середовища	Великий обсяг даних для навчання	[8]
Генетичні алгоритми	Оптимізація траєкторій через ітеративний підхід	Простота реалізації	Низька точність для складних задач	[9]
Нейромереже ве планування	Передбачення оптимального маршруту на основі даних	Гнучкість, здатність до самонавчання	Потреба в значних ресурсах	[10]

1.4 Сучасні моделі ШІ для розпізнавання об’єктів

Розпізнавання об’єктів є однією з фундаментальних задач комп’ютерного зору, яка поєднує виявлення, класифікацію та локалізацію об’єктів у зображеннях чи відеопотоках. Ця технологія має широке застосування у різних сферах, зокрема в автоматизації, робототехніці, системах безпеки, медичній діагностиці та автономних транспортних засобах. Завдяки прогресу в галузі

глибокого навчання, сучасні моделі для розпізнавання об’єктів досягли високої точності, швидкості та універсальності.

Методи розпізнавання об’єктів можна умовно поділити на три категорії:

1. **Моделі одноетапного розпізнавання:** алгоритми, які здійснюють виявлення та класифікацію об’єктів у межах одного проходу (наприклад, YOLO та SSD).
2. **Моделі двоетапного розпізнавання:** алгоритми, які спочатку створюють регіони пропозицій (Regions of Interest, ROI), а потім виконують класифікацію та локалізацію об’єктів (наприклад, Faster R-CNN).
3. **Гібридні та новаторські підходи:** моделі, що використовують новітні архітектури на основі трансформерів (наприклад, DETR).

Сучасні моделі штучного інтелекту (ШІ) для розпізнавання об’єктів є результатом десятиліть досліджень у галузі комп’ютерного зору, глибокого навчання та нейронних мереж. Вони демонструють високу ефективність у різних завданнях: від виявлення та класифікації об’єктів до визначення їхньої локалізації на зображеннях або у відеопотоках. Розглянемо основні архітектури ШІ, які є ключовими у цій сфері.

YOLO (You Only Look Once) – це архітектура, що змінила підхід до виявлення об’єктів, запропонувавши одноетапний метод обробки зображення [10].

1. Механізм роботи: YOLO обробляє зображення як єдине ціле, розділяючи його на сітку. Кожна комірка цієї сітки прогнозує межі об’єктів і їхню категорію одночасно. Такий підхід забезпечує високу швидкість роботи.
2. Особливості:
    - Швидкість: YOLO є однією з найшвидших моделей, здатною працювати у реальному часі, що робить її ідеальною для динамічних систем, таких як автономні транспортні засоби або дрони.

- Застосування: модель широко використовується у відеоспостереженні, аналізі спортивних трансляцій та в робототехніці [10].

3. Обмеження:
   - Труднощі з розпізнаванням дрібних об’єктів. Це пов’язано з тим, що розмір сітки обмежує роздільну здатність прогнозів.
   - Складні сцени: у густонаселених сценах (наприклад, міські вулиці з великою кількістю рухомих об’єктів) модель може демонструвати знижену точність.
4. Еволюція:

   Новіші версії YOLO, такі як YOLOv4 і YOLOv5, значно покращили точність і швидкість, додаючи механізми автоматичного налаштування гіперпараметрів і покращені підходи до обробки зображень.

Faster R-CNN (Faster Region-Based Convolutional Neural Network) - є однією з перших моделей, яка ефективно об’єднала високоточне виявлення об’єктів із механізмом регіонів пропозицій [11].

1. Механізм роботи:
   - Модель використовує регіони пропозицій (Regions of Interest, ROI) для визначення ймовірних місць розташування об’єктів.
   - Далі CNN (Convolutional Neural Network) класифікує об’єкти та уточнює їх межі.
2. Особливості:
   - Точність: faster R-CNN демонструє високу ефективність у складних середовищах із численними перекриттями об’єктів.
   - Гнучкість: модель підтримує різні типи вхідних даних, включно з 2D-зображеннями та відео.
3. Обмеження:
   - Високі обчислювальні вимоги: для роботи Faster R-CNN потрібні потужні графічні процесори, що обмежує її використання на пристроях із низькою продуктивністю.

- Швидкість: через двоетапний підхід модель є повільнішою, ніж YOLO або SSD.

4. Застосування: faster R-CNN широко використовується в задачах, де точність є критичною, наприклад, у медичній діагностиці (розпізнавання пухлин на знімках МРТ) або системах автоматичного моніторингу виробничих процесів.

SSD (Single Shot Multibox Detector) - поєднує швидкість YOLO з точністю Faster R-CNN, що робить її універсальним вибором для багатьох задач [12].

1. Механізм роботи: SSD використовує кілька рівнів функцій для передбачення об’єктів на різних масштабах. Кожен рівень відповідає за виявлення об’єктів певного розміру, що дозволяє моделі зберігати баланс між точністю та швидкістю.
2. Особливості:
   - Швидкість: SSD працює швидше, ніж Faster R-CNN, оскільки виконує виявлення об’єктів за один прохід.
   - Гнучкість: модель добре працює зі змінними масштабами об’єктів у кадрі.
3. Обмеження:
   - Менш точне виявлення дрібних об’єктів порівняно з Faster R-CNN.
   - Відносно нижча ефективність у густонаселених сценах.
4. Застосування: SSD часто використовується у задачах, що вимагають високої продуктивності на середньому рівні обчислювальних ресурсів, наприклад, у мобільних додатках або дронах.

EfficientDet – це сучасна модель, яка зосереджена на зменшенні ресурсоспоживання, зберігаючи високу точність [31].

1. Механізм роботи: основою моделі є EfficientNet, яка оптимізує розподіл обчислювальних ресурсів між різними рівнями мережі.
2. Особливості:

- Енергоефективність: модель спеціально розроблена для роботи на мобільних пристроях та інших платформах із обмеженими ресурсами.
- Масштабованість: efficientDet дозволяє легко змінювати розмір мережі залежно від завдань.

3. Обмеження:
   - У порівнянні з YOLO або Faster R-CNN, модель має дещо нижчу продуктивність у складних середовищах із високою щільністю об’єктів.
4. Застосування: використовується у сфері IoT, для аналізу даних із дронів та інших мобільних платформ, де обмежені апаратні ресурси.

Mask R-CNN є розширенням Faster R-CNN, яке додає функціонал сегментації об’єктів [32].

1. Механізм роботи: модель додає до Faster R-CNN окрему гілку для прогнозування маски об’єкта (піксельного контуру), що дозволяє не лише розпізнавати об’єкти, а й виділяти їх контури.
2. Особливості:
   - Сегментація: Mask R-CNN дозволяє виконувати семантичну сегментацію об’єктів, що критично важливо для завдань у медицині чи промисловості.
   - Гнучкість: підтримує розпізнавання об’єктів різного розміру та складності.
3. Обмеження:
   - Високі обчислювальні потреби.
   - Низька швидкість роботи порівняно з моделями реального часу, такими як YOLO.
4. Застосування: Mask R-CNN використовується у медицині (аналіз знімків для виявлення хвороб), автоматизації виробництва (визначення дефектів) та доповненій реальності.

CenterNet – це одноетапна модель, яка спрощує процес виявлення об’єктів за допомогою ідентифікації центральних точок кожного об’єкта [33].

1. Механізм роботи: замість створення регіонів пропозицій або рамок, CenterNet прогнозує центри об’єктів та їх межі. Це дозволяє моделі працювати швидше, зберігаючи при цьому високу точність.
2. Особливості:
    - Спрощена архітектура: відсутність складних компонентів знижує ресурсоємність моделі.
    - Швидкість: підходить для завдань реального часу.
3. Обмеження:
    - Відносно нижча точність для складних сцен або перекритих об’єктів.
    - Складність у роботі з дрібними об’єктами.
4. Застосування: centerNet використовується у відеоаналітиці, дронах і робототехніці, де потрібна висока продуктивність у реальному часі.

Розширені версії YOLO, такі як **YOLOv7 і YOLOv8**, втілюють новітні розробки у сфері глибокого навчання [34].

1. Механізм роботи: ці моделі інтегрують оптимізацію навчання за допомогою сучасних механізмів уваги (*attention mechanisms*) та покращених активаційних функцій.
2. Особливості:
    - Покращена продуктивність: моделі досягли нових рекордів за точністю та швидкістю.
    - Адаптивність: легко масштабуються для роботи з різними наборами даних.
3. Обмеження:
    - Високі вимоги до ресурсів у порівнянні з YOLOv5.
    - Складність налаштування для новачків.
4. Застосування: використовуються для відеоспостереження, розпізнавання дорожніх знаків, аналізу поведінки людей і навіть у сферах безпеки.

Cascade R-CNN – це модель, яка розвиває концепцію R-CNN, покращуючи точність через каскадну архітектуру [35].

1. Механізм роботи: каскадна архітектура дозволяє покращувати результати класифікації та локалізації на кожному наступному рівні, використовуючи більш суворі критерії.
2. Особливості:
    - Точність: модель демонструє кращі результати у складних задачах, таких як виявлення дрібних об’єктів.
    - Стійкість: підвищена здатність до узагальнення завдяки каскадному підходу.
3. Обмеження:
    - Більша ресурсоємність порівняно з Faster R-CNN.
    - Зниження швидкості роботи.
4. Застосування: використовується у промислових системах контролю якості, аналізі супутникових зображень і задачах високоточного моніторингу.

M2Det (Multi-Level Feature Pyramid Network) орієнтована на ефективну обробку об’єктів різного розміру за допомогою багаторівневої пірамідальної архітектури [36].

1. Механізм роботи: модель інтегрує багаторівневі функції, що дозволяє краще розпізнавати як дрібні, так і великі об’єкти.
2. Особливості:
    - Гнучкість: підтримка об’єктів із великим діапазоном розмірів.
    - Продуктивність: досягає хорошого балансу між швидкістю й точністю.
3. Обмеження:
    - Відносно складна архітектура для навчання.
    - Потребує великих обсягів даних для досягнення високих результатів.

4. Застосування: M2Det використовується у сфері безпілотного транспорту, аналізу медичних зображень і автоматизації процесів на виробництві.

RefineDet поєднує одноетапні й двоетапні підходи для досягнення високої продуктивності [37].

1. Механізм роботи: модель виконує попереднє уточнення меж об’єктів перед їх остаточним розпізнаванням, що підвищує точність.
2. Особливості:
    - Баланс між швидкістю й точністю: refineDet працює швидше, ніж Faster R-CNN, але точніше за SSD.
    - Стійкість: модель добре працює у складних сценах із шумом.
3. Обмеження:
    - Відносно складна реалізація.
    - Потребує потужних апаратних ресурсів.
4. Застосування: використовується у транспортних системах, моніторингу інфраструктури та робототехніці.

1.5 Вибір інструментів та алгоритмів для реалізації методу

Ubuntu — це універсальна, безкоштовна операційна система з відкритим вихідним кодом, яка побудована на основі ядра Linux. Її популярність зумовлена широкими можливостями застосування — від щоденної роботи з офісними додатками до створення складних програмних рішень для робототехніки, хмарних обчислень і систем зі штучним інтелектом. Стабільність, надійність і доступність численних інструментів, таких як Docker, Python, OpenCV і ROS2, роблять Ubuntu ідеальним вибором для розробників, які працюють над інноваційними проектами [14].

Я обрав Ubuntu для свого проекту, оскільки вона пропонує оптимальне середовище для розробки автономних роботів та алгоритмів управління. Її інтеграція з ROS2 (Robot Operating System 2) значно полегшує створення, тестування та налаштування алгоритмів, необхідних для роботи безпілотного робота. Крім того, спільнота Ubuntu є однією з найбільших у світі, що

забезпечує доступ до багатьох ресурсів, документації та підтримки. Завдяки своїй ефективності та простоті використання, ця операційна система дозволяє швидше впроваджувати нові ідеї, оптимізувати процес розробки та зосередитися на досягненні цілей проекту.

Рис. 1.1 Ubuntu linux(logo)

Gazebo — це потужний симулятор, який забезпечує реалістичне тестування роботизованих систем у віртуальному середовищі. Він дозволяє моделювати фізичні характеристики роботів, взаємодію з оточенням та тестувати їхню поведінку у різноманітних сценаріях. Однією з ключових переваг Gazebo є його тісна інтеграція з ROS2, що спрощує створення спільних середовищ для роботи різних роботизованих компонентів та алгоритмів [15].

Я обрав Gazebo для свого проекту, оскільки цей інструмент є незамінним для розробки та тестування безпілотних систем. Він дозволяє перевіряти алгоритми управління роботом у реалістичних умовах без необхідності використовувати фізичний прототип. Це не лише скорочує витрати, але й забезпечує більшу гнучкість для швидкої перевірки нових ідей. Завдяки своїм широким можливостям, Gazebo є важливим інструментом у наукових дослідженнях, промислових розробках і освітніх проектах, а також дозволяє оптимізувати процес створення автономних систем.

Рис. 1.2 Gazebo(logo)

Visual Studio Code — це один із найзручніших і найпотужніших редакторів коду, який широко використовується в розробці програмного забезпечення. Його популярність серед розробників робототехніки пояснюється підтримкою різноманітних мов програмування, зокрема Python і C++, а також інтеграцією з такими інструментами, як ROS2. Завдяки розширенням і плагінам, цей редактор забезпечує ефективну розробку, тестування та налагодження коду, що робить його універсальним вибором як для початківців, так і для досвідчених інженерів [16].

Я обрав Visual Studio Code для свого проекту, оскільки він значно спрощує роботу над алгоритмами управління безпілотним роботом. Його можливості для дебагінгу дозволяють швидко знаходити й виправляти помилки, а підтримка низькорівневого програмування на C++ і високорівневого — на Python, робить цей редактор ідеальним для роботи над усіма компонентами проекту. Завдяки його гнучкості та інтеграції з ROS2, Visual Studio Code є оптимальним рішенням для розробки автономних систем у сучасних проектах робототехніки.

Рис. 1.3 Visual Studio code(logo)

C++ — це одна з найефективніших мов програмування, яка активно застосовується в галузі робототехніки. Вона забезпечує можливість створення високопродуктивних алгоритмів для управління сенсорами та актуаторами, які відповідають за точну навігацію та взаємодію робота з навколишнім середовищем [17].

Для реалізації проекту я зупинив свій вибір на C++, оскільки її можливості дозволяють працювати з алгоритмами в реальному часі, що є критично важливим для автономних систем. Крім того, ця мова ідеально підходить для обробки низькорівневих команд і взаємодії з апаратними компонентами робота. Її використання сприяє досягненню стабільності та точності роботи безпілотних роботів, особливо в складних умовах і сценаріях.

Рис. 1.4 C++(logo)

Python — це універсальна мова програмування, яка стала невід'ємною частиною робототехніки та розробки ШІ-алгоритмів. Її активно застосовують для реалізації моделей машинного навчання, обробки сенсорних даних і зображень, що є важливими компонентами автономних систем [18].

У моєму проекті Python був обраний через його потужний інструментарій і широкий спектр бібліотек, таких як TensorFlow, PyTorch і OpenCV, які значно спрощують розробку та тестування алгоритмів. Ця мова також забезпечує гнучкість при інтеграції з іншими системами та прискорює процес створення складних моделей для роботи безпілотних роботів. Завдяки своїй простоті та потужності Python є незамінним інструментом для розробки сучасних автономних систем.

Рис. 1.5 Python(logo)

ROS2 — це відкритий фреймворк, який широко використовується для розробки програмного забезпечення для роботів. Завдяки можливості інтеграції різних модулів системи, таких як сенсори, алгоритми навігації та компоненти управління, він став важливим інструментом для створення роботизованих систем [19].

У моєму проекті я обрав ROS2, оскільки він забезпечує масштабованість і гнучкість для розробки автономних систем, що працюють у складних та різноманітних умовах. Завдяки інтеграції з такими інструментами, як Gazebo для симуляції та Rviz2 для візуалізації, ROS2 стає ключовим елементом для створення ефективних алгоритмів управління та тестування безпілотних

роботів. Вибір ROS2 обумовлений його здатністю забезпечити стабільну платформу для розробки, тестування та інтеграції різних компонентів системи.

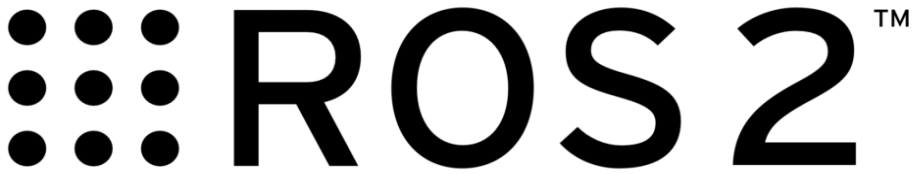


Рис. 1.6 Ros2(logo)

Nav2 (Navigation2) — це фреймворк для навігації роботів, який є частиною системи ROS2. Він призначений для розробки та використання алгоритмів навігації та планування маршрутів для безпілотних роботів у різних середовищах.

Для мого проекту я обрав Nav2, оскільки цей фреймворк надає необхідні інструменти для реалізації ключових функцій навігації, таких як локалізація, картографування, планування маршрутів та управління перешкодами. Використовуючи Nav2, можна створювати системи навігації для роботів, здатних ефективно переміщатися та орієнтуватися в складних умовах, таких як внутрішні приміщення, магазини або промислові об'єкти. Цей фреймворк дозволяє інтегрувати різні алгоритми для досягнення стабільної роботи в реальних умовах.

Rviz2 — це інструмент візуалізації для ROS2, який дає змогу відображати різноманітні дані з роботів та оточення в реальному часі. Використовуючи Rviz2, можна візуалізувати такі дані, як мапи, лазерні сканування, точки спостереження, траєкторії руху, а також взаємодіяти з роботами, задаючи цілі і відстежуючи їхнє переміщення.

Цей інструмент був обраний для мого проекту, оскільки він надає зручні можливості для аналізу та налаштування алгоритмів навігації і контролю роботів. Завдяки Rviz2 можна ефективно відслідковувати поведінку робота, що

дає змогу виявляти і виправляти помилки на ранніх етапах розробки, а також покращувати точність і стабільність системи в реальних умовах. Це сприяє оптимізації роботи безпілотних роботів і розвитку їхніх навігаційних функцій.

2 АНАЛІЗ МЕТОДІВ УПРАВЛІННЯ БЕЗПІЛОТНИМИ РОБОТАМИ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Аналіз існуючих методів управління безпілотними роботами на основі штучного інтелекту

Методи управління безпілотними роботами розвиваються з урахуванням складності завдань, динамічності середовищ і необхідності високої автономності. Ці підходи можна поділити на традиційні алгоритми, методи багатосенсорної інтеграції, технології локалізації та картографування, а також сучасні алгоритми глибокого та посиленого навчання.

Традиційні алгоритми включають поведінкові дерева та скінченні автомати, які дозволяють структурувати роботу системи у вигляді станів і переходів між ними. Поведінкові дерева, що подають завдання у вигляді ієрархічних вузлів, ефективні для стабільних середовищ, таких як виробничі процеси, але недостатньо гнучкі для динамічних умов [3].

Скінченні автомати, що працюють із визначеними станами системи, добре підходять для обмежених сценаріїв, де можливі переходи чітко визначені. Проте вони обмежені у здатності адаптуватися до змін, оскільки не враховують складність середовищ із великою кількістю змінних факторів [1].

Методи багатосенсорної інтеграції є необхідними для забезпечення узгодженості даних, отриманих від різноманітних сенсорів, таких як лідарами, камерами, GPS та IMU. Для об’єднання цих даних застосовуються фільтри Калмана, які дозволяють враховувати похибки сенсорів і отримувати оптимальну оцінку місцезнаходження робота [25].

Іншим підходом є фільтри частинок, що забезпечують ефективну обробку даних у складних умовах із високим рівнем шуму. Наприклад, у середовищах зі складними перешкодами фільтр частинок може використовуватися для уточнення траєкторії робота [8].

Технологія SLAM (Simultaneous Localization and Mapping) дозволяє одночасно визначати місцезнаходження робота та створювати карту середовища. Вона є ключовою для автономних систем, що працюють у невідомих або змінних середовищах. Серед популярних реалізацій SLAM є ORB-SLAM, що використовує візуальні ключові точки для ідентифікації об’єктів, та Cartographer, оптимізований для високошвидкісної роботи у внутрішніх приміщеннях [24].

Алгоритми глибокого навчання відкрили нові можливості для автономного управління. YOLOv5 є однією з найефективніших моделей для реального часу, що дозволяє виконувати детекцію та класифікацію об’єктів із високою швидкістю. Завдяки оптимізації архітектури YOLOv5 забезпечує високу точність навіть за умов динамічного середовища [11].

Faster R-CNN забезпечує більш точне розпізнавання об’єктів, але має обмеження у швидкості, що робить його менш придатним для застосування в реальному часі [7].

Посилене навчання, наприклад, Deep Q-Learning, дозволяє роботам адаптувати поведінку через взаємодію із середовищем, отримуючи винагороду за виконання оптимальних дій. Цей метод є корисним у ситуаціях, де наперед неможливо визначити всі можливі сценарії. Наприклад, роботи можуть навчатися уникати перешкод або шукати найкоротші маршрути у динамічному середовищі [9].

Інтеграція глибокого навчання з технологіями SLAM дозволяє створювати більш адаптивні системи, що аналізують об’єкти, ухвалюють складні рішення та забезпечують ефективну навігацію. Наприклад, об’єднання методів ORB-SLAM із нейронними мережами підвищує точність створення карти та ідентифікації об’єктів у режимі реального часу [3].

Сучасні обчислювальні платформи, такі як NVIDIA Jetson або Intel Movidius, надають достатню потужність для обробки складних алгоритмів у реальному часі. Це дозволяє виконувати глибокий аналіз навколишнього середовища та реалізовувати складні методи навігації [8].

Перспективним напрямом розвитку є впровадження IoT (Internet of Things), яке дозволяє використовувати підключені сенсори для забезпечення покращеної ситуаційної обізнаності. Наприклад, у сфері логістики роботи можуть обмінюватися даними про завантаженість маршрутів у режимі реального часу, підвищуючи загальну ефективність [5].

Застосування оптимізаційних методів, таких як генетичні алгоритми чи рійові оптимізаційні методи (PSO), забезпечує ефективне планування маршруту та розподіл ресурсів. Наприклад, оптимізація маршрутів дронів із урахуванням енергоспоживання дозволяє збільшити їхню автономність [9].

Таким чином, аналіз існуючих методів показує, що ефективне управління безпілотними роботами можливе лише через інтеграцію різних підходів. Використання гібридних рішень, які поєднують переваги SLAM, глибокого навчання та багатосенсорної інтеграції, забезпечує високу адаптивність та ефективність системи у складних середовищах.

Особливу увагу слід приділити комбінуванню локалізації та детекції об'єктів у реальному часі, оскільки ці дві задачі є критичними для ефективного функціонування автономного робота. Інтеграція SLAM із глибоким навчанням дозволяє не лише створювати карту середовища, а й аналізувати об'єкти в ньому. Наприклад, алгоритми на основі YOLOv5 можуть використовуватися для розпізнавання об'єктів у середовищі, тоді як ORB-SLAM забезпечує точну локалізацію робота. Це поєднання дозволяє створювати карти, які враховують динамічні елементи, такі як рухомі перешкоди чи об'єкти [11].

Застосування фільтра Калмана в багатосенсорній інтеграції забезпечує додаткову стабільність та надійність системи. Завдяки йому дані з камер, лідарів і інерційних сенсорів можуть бути об'єднані у єдину картину середовища. Це особливо важливо для роботи в умовах, де окремі сенсори можуть мати похибки чи втрачати сигнал. Наприклад, у середовищах із поганим освітленням, де камера може не дати точних даних, фільтр Калмана забезпечує компенсацію цих недоліків за рахунок використання даних з інших сенсорів [25].

Глибоке навчання відкриває нові перспективи у використанні нейронних мереж для прийняття складних рішень у реальному часі. Крім детекції об'єктів, такі мережі можуть використовуватися для прогнозування траєкторій об'єктів, що рухаються. Це особливо важливо в умовах динамічного середовища, де роботи повинні не лише уникати статичних перешкод, а й передбачати рух інших об'єктів. Наприклад, використання рекурентних нейронних мереж (RNN) для аналізу послідовних зображень дозволяє передбачати траєкторії пішоходів або транспортних засобів [7].

Методи посиленого навчання, такі як Proximal Policy Optimization (PPO) чи Deep Deterministic Policy Gradient (DDPG), можуть бути інтегровані для створення моделей, здатних до адаптації в режимі реального часу. Ці методи є надзвичайно корисними для вирішення задач, де середовище є постійно змінним, наприклад, у рятувальних операціях чи промислових умовах.Важливо, що роботи, оснащені такими алгоритмами, можуть навчатися під час виконання завдань, що значно підвищує їхню автономність і ефективність [9].

Додатково слід зазначити, що використання IoT-технологій сприяє зростанню можливостей безпілотних роботів. Інтеграція сенсорів із хмарними платформами дозволяє в режимі реального часу передавати дані для обробки, аналізу та прийняття рішень. Наприклад, хмарні сервіси, такі як Amazon AWS чи Google Cloud, можуть бути використані для навчання нейронних мереж та подальшої обробки даних, що надходять від роботів. Це дозволяє створювати більш інтегровані та адаптивні системи [5].

Окрему увагу варто приділити оптимізації енергоспоживання. Один із підходів полягає у використанні спайкових нейронних мереж, які імітують роботу мозку, дозволяючи значно зменшити енергозатрати при обробці великих обсягів даних. Це може бути критичним для безпілотних роботів, які працюють у віддалених умовах і мають обмежені ресурси енергії [8].

Підсумовуючи, сучасні методи управління безпілотними роботами спрямовані на інтеграцію традиційних алгоритмів, глибокого навчання, багатосенсорної інтеграції та технологій IoT. Вони забезпечують високу

адаптивність, точність і ефективність у вирішенні складних задач. Однак для досягнення ще більшого рівня автономності та надійності необхідно продовжувати дослідження у напрямі розробки гібридних підходів та їхнього оптимального поєднання.

2.2 Аналіз переваг та недоліків методів управління безпілотними роботами

У сучасній робототехніці існує велика кількість методів управління, які базуються як на класичних алгоритмах, так і на штучному інтелекті. Аналіз цих підходів дозволяє виділити їхні сильні та слабкі сторони залежно від умов використання, динаміки середовища та вимог до ефективності системи управління.

Класичні алгоритми: до базових методів управління відносяться поведінкові дерева та скінченні автомати. ці підходи забезпечують чітку структуру дій робота, де кожен стан відповідає певній дії, а переходи між станами визначаються заздалегідь заданими умовами.

Серед переваг класичних алгоритмів варто виділити простоту реалізації та обчислювальної схеми, що дозволяє їх використовувати на обладнанні з обмеженими ресурсами. крім того, вони гарантують передбачувані результати в стабільних середовищах, наприклад, у виробничих лініях або інших структурованих сценаріях. Разом із тим, основним недоліком класичних методів є їхня неспроможність адаптуватися до змінних умов середовища. наприклад, у випадках зі складними або непередбачуваними перешкодами такі алгоритми демонструють значну обмеженість, оскільки вони не здатні навчатися новим ситуаціям або враховувати змінні в реальному часі [3].

Методи глибокого навчання, представлені моделями YOLOv5 та Faster R-CNN, значно розширили можливості автономних систем управління роботами. ці алгоритми дозволяють виконувати розпізнавання та класифікацію об’єктів із високою швидкістю та точністю.

до ключових переваг глибокого навчання належить здатність автоматично адаптуватися до нових умов і працювати у динамічних середовищах. наприклад, YOLOv5 демонструє високу ефективність у задачах реального часу, обробляючи зображення всього за один прохід через нейронну мережу [11]. faster r-cnn, хоча і має нижчу швидкість, забезпечує вищу точність при роботі зі складними об'єктами [7].

Основними недоліками методів глибокого навчання є висока обчислювальна складність, що вимагає потужного обладнання, та необхідність попереднього навчання на великих наборах даних. крім того, такі методи є енергоємними, що може обмежувати їх використання в роботах із обмеженим енергопостачанням.

Посилене навчання, зокрема алгоритми deep q-learning або proximal policy optimization (ppo), пропонують унікальні можливості для адаптації робота до змін у реальному часі. робот отримує винагороду за оптимальні дії, що дозволяє йому самостійно навчатися найбільш ефективним стратегіям поведінки.
сильними сторонами цього підходу є його здатність до автономності та можливість вирішувати задачі у складних умовах, таких як міські середовища чи місії у невідомих приміщеннях. наприклад, роботи можуть навчатися обходити перешкоди або знаходити найкоротший маршрут на основі отриманих нагород [9]. Головним недоліком методів посиленого навчання є тривалість навчання та необхідність у стабільному середовищі для ефективного тренування. будь-які зміни у параметрах середовища можуть вплинути на результати роботи алгоритму.

Інтеграція сенсорних даних : ф’юзія даних із сенсорів забезпечує створення узгодженої картини середовища, що значно покращує точність і швидкість ухвалення рішень роботом. лідарами, камери та gps працюють разом, об'єднуючи інформацію через фільтр калмана чи інші алгоритми багатосенсорної інтеграції. До переваг цього підходу належать можливість роботи в умовах низької видимості або повної відсутності сигналу gps, що робить його незамінним для автономних роботів, які працюють у складних

середовищах. однак, складність інтеграції великого обсягу даних у реальному часі є значним викликом для цих методів [25].

Технологія SLAM (simultaneous localization and mapping) є ключовою для автономних роботів, які працюють у невідомих середовищах. вона забезпечує можливість одночасного створення карти середовища та визначення місця розташування робота. завдяки інтеграції даних із камер, лідарів і IMU, SLAM дозволяє роботам ефективно орієнтуватися навіть у складних умовах, наприклад, у закритих приміщеннях без доступу до GPS [24]. Серед сильних сторін SLAM — автономність, точність і здатність до адаптації в умовах змінного середовища. наприклад, технологія ORB-SLAM демонструє високу ефективність у завданнях візуальної локалізації, використовуючи ключові точки зображень. інший популярний підхід — cartographer, який забезпечує швидке створення карт у реальному часі, особливо в умовах внутрішніх приміщень.
втім, SLAM має і свої обмеження. висока обчислювальна складність технології потребує значних ресурсів, а точність результатів залежить від якості та кількості використовуваних сенсорів. наприклад, помилки в роботі лідарів або камери можуть призвести до неточностей у побудові карти та визначенні місця розташування робота.

Адаптивні методи управління базуються на використанні алгоритмів, здатних змінювати свої параметри залежно від умов середовища. одним із таких підходів є застосування генетичних алгоритмів, які імітують процеси природної еволюції для пошуку оптимальних рішень. Генетичні алгоритми особливо ефективні у задачах планування маршруту або розподілу ресурсів. наприклад, вони можуть бути використані для оптимізації маршруту робота з урахуванням мінімізації витрат енергії або часу. проте, процес еволюційного пошуку потребує значних обчислювальних ресурсів і часу, що обмежує їхнє застосування в реальному часі [9].

Ще одним перспективним напрямом є методи рійової оптимізації, які базуються на поведінці колоній комах або рибних косяків. ці методи дозволяють координувати дії групи роботів, наприклад, у задачах одночасної локалізації та

доставки вантажів кількома дронами. основна перевага цього підходу — його висока ефективність у розподілених системах, де необхідно координувати взаємодію кількох автономних агентів [3].

Інтеграція технологій IoT у системи управління роботами відкриває нові можливості для покращення ситуаційної обізнаності та взаємодії з середовищем. підключені сенсори дозволяють роботам отримувати дані в режимі реального часу, що сприяє підвищенню точності та швидкості ухвалення рішень. Наприклад, IoT-технології можуть використовуватися для моніторингу стану середовища, прогнозування погодних умов або оцінки завантаженості маршрутів. це особливо корисно у сфері логістики, де роботи можуть обмінюватися даними для оптимізації своїх дій [5]. Втім, основною проблемою є безпека таких систем, оскільки підключення до мережі може стати вразливим до кібератак.

Взаємодія людини та робота, забезпечення безпечної та ефективної взаємодії між людьми й роботами є однією з ключових задач сучасної робототехніки. у багатьох сценаріях, наприклад у логістиці, промисловості чи рятувальних операціях, роботи працюють поряд із людьми, що вимагає високого рівня безпеки та зрозумілості їхніх дій. Одним із підходів для покращення взаємодії є використання інтерфейсів, заснованих на технологіях доповненої реальності (AR). Ці системи дозволяють візуалізувати дії робота у реальному часі, що значно полегшує планування спільних завдань. наприклад, AR може показувати, які дії буде виконувати робот наступними, чи вказувати безпечні зони для людини [3].

Додатково використовуються природномовні інтерфейси, які дозволяють керувати роботами за допомогою голосових команд. це підвищує зручність взаємодії, особливо у випадках, коли людина має зайняті руки або немає можливості використати інші способи керування.

разом із тим, важливо враховувати складність проектування таких систем, адже вони повинні бути інтуїтивно зрозумілими, надійними та зручними для використання в умовах реального часу.

Енергоефективність, проблема енергоспоживання є актуальною для всіх безпілотних роботів, особливо тих, які працюють у віддалених або автономних умовах. для вирішення цього завдання використовуються різні підходи, зокрема оптимізація алгоритмів управління та використання енергоефективних компонентів. Одним із напрямів розвитку є застосування спайкових нейронних мереж, які імітують біологічні процеси у мозку. ці мережі дозволяють обробляти інформацію з мінімальним енергоспоживанням, що є критично важливим для роботів із обмеженими джерелами енергії. ще одним підходом є використання методів планування маршрутів, які мінімізують витрати енергії на кожному етапі виконання завдання [8]. Окрім оптимізації алгоритмів, значна увага приділяється використанню сучасних батарей із більшою ємністю та швидкістю заряджання. наприклад, інтеграція роботів із сонячними батареями дозволяє значно підвищити їхню автономність, особливо у випадках довготривалих місій.

Розподілене управління та кооперація роботів, завдання координації груп роботів є складним, але важливим напрямом розвитку. розподілені системи управління дозволяють кільком роботам одночасно працювати над виконанням спільних завдань, наприклад, пошуку та рятування чи доставки вантажів.
одним із підходів є використання блокчейн-технологій для забезпечення надійності обміну даними між роботами. це дозволяє уникнути проблем із безпекою та синхронізацією дій у розподілених системах [3]. Додатково методи рійової оптимізації забезпечують ефективну координацію дій роботів, що працюють у групі. наприклад, кілька дронів можуть одночасно виконувати завдання картографування або моніторингу території, узгоджуючи свої дії в режимі реального часу.

Самоадаптація в незнайомих середовищах, роботи, які працюють у складних і динамічних умовах, таких як космос, глибоководні місії чи стихійні зони, потребують алгоритмів самоадаптації. це забезпечує здатність до адаптації в умовах, де попередньо визначені стратегії можуть бути неефективними.

поєднання трансформерів із посиленим навчанням є перспективним напрямом для досягнення цієї мети. трансформери, зокрема vision transformers (vit), дозволяють аналізувати великі обсяги даних та виділяти найбільш релевантну інформацію навіть у середовищах із високим рівнем шуму. використання посиленого навчання забезпечує динамічне коригування дій робота залежно від змін у середовищі [29].

Наприклад, у завданнях пошуку та рятування роботи можуть змінювати свої стратегії залежно від виявлених перешкод або зміни погодних умов. це значно підвищує ефективність виконання місій та знижує ризик збоїв у роботі. використання сучасних симуляційних середовищ, таких як gazebo, дозволяє тестувати алгоритми самоадаптації у віртуальних умовах перед їх впровадженням у реальному світі. це допомагає мінімізувати ризики та підвищити надійність системи управління [16]. Робота з великими даними, сучасні автономні системи, зокрема безпілотні роботи, генерують величезні обсяги даних, які необхідно обробляти для прийняття рішень у реальному часі. тут ключову роль відіграють технології великих даних (big data), що забезпечують швидке зберігання, аналіз та обробку інформації.

Одним із підходів є використання хмарних платформ для обробки даних, що дозволяє знизити обчислювальне навантаження на самого робота. наприклад, хмарні системи можуть виконувати складні обчислення, такі як глибокий аналіз зображень чи планування траєкторії, передаючи результати назад роботу для виконання завдань. Крім того, методи оптимізації з використанням нейронних мереж, такі як autoML, дозволяють автоматично налаштовувати параметри алгоритмів залежно від специфіки отриманих даних. це значно підвищує ефективність роботи автономних систем і дозволяє швидко адаптувати їх до нових умов [28].

Розвиток робототехніки в контексті промисловості, застосування роботів у промисловості передбачає інтеграцію технологій автоматизації для підвищення продуктивності та зменшення витрат. автономні роботи

використовуються для виконання широкого спектра завдань, таких як транспортування матеріалів, складання продуктів або контроль якості.

особливістю сучасних промислових роботів є їхня здатність працювати в умовах високої точності та повторюваності. наприклад, роботи, обладнані камерами та лідарами, можуть виконувати завдання складального виробництва, мінімізуючи ризик помилок. крім того, використання посиленого навчання дозволяє їм адаптувати свої дії залежно від змін у виробничих лініях [3].

серед викликів промислової робототехніки — забезпечення надійності систем управління, а також розробка алгоритмів, які дозволять роботам працювати в умовах високої динаміки та взаємодії з іншими системами.

Робототехніка в логістиці, безпілотні роботи активно впроваджуються в логістичних системах для оптимізації процесів транспортування, складування та управління запасами. автономні роботи здатні швидко та точно переміщувати товари, зменшуючи час виконання завдань і підвищуючи продуктивність логістичних операцій. Ключовим елементом є інтеграція алгоритмів навігації та планування маршрутів, що забезпечує ефективне переміщення роботів навіть у складних середовищах, таких як великі склади або виробничі площі. використання SLAM дозволяє роботам працювати без потреби у попередньо створених картах, а також адаптуватися до змін у розташуванні товарів або обладнання [24].

Одним із прикладів є використання роботів на основі алгоритмів deep q-learning, які здатні адаптувати свої дії залежно від завантаженості маршрутів або пріоритетності завдань. це дає змогу уникати затримок і оптимізувати час виконання кожного завдання [9].

серед недоліків роботи таких систем у логістиці — висока залежність від якості сенсорів та необхідність постійного моніторингу стану обладнання для запобігання збоям. разом із тим, використання сучасних технологій інтернету речей (IoT) сприяє підвищенню надійності та прозорості логістичних процесів [5].

Розвиток технологій у рятувальних операціях, у рятувальних операціях безпілотні роботи використовуються для пошуку постраждалих, моніторингу стану середовища та доставки необхідних матеріалів. такі завдання потребують високого рівня автономності та здатності працювати у складних умовах, наприклад, у зонах стихійних лих або аварій. Ключову роль тут відіграють алгоритми SLAM та глибокого навчання, які забезпечують точну орієнтацію у невідомих середовищах. наприклад, роботи, оснащені лідарами та камерами, можуть створювати тривимірні карти руйнувань, що значно спрощує пошук постраждалих [24].

Додатково використання нейронних мереж, таких як YOLOv5, дозволяє ідентифікувати цільові об’єкти навіть у складних умовах освітлення або при наявності великої кількості перешкод. це особливо важливо у завданнях пошуку людей у завалах або об'єктів серед уламків [11]. Попри значний потенціал, рятувальні роботи стикаються з обмеженнями, такими як високе енергоспоживання, обмеження у роботі в екстремальних умовах та необхідність точного налаштування алгоритмів для специфічних сценаріїв. використання автономних роботів у сільському господарстві у сільському господарстві безпілотні роботи застосовуються для виконання завдань, таких як моніторинг посівів, обприскування культур та збирання врожаю. їхня автономність дозволяє значно підвищити продуктивність аграрного виробництва та зменшити витрати на робочу силу. особливо перспективним є використання роботів, оснащених алгоритмами комп’ютерного зору, які дозволяють аналізувати стан рослин, визначати наявність шкідників або хвороб. це дає змогу проводити точкове обприскування або вибірковий збір врожаю, знижуючи використання ресурсів [7].

Втім, основною проблемою є складність роботи в різноманітних погодних умовах та необхідність інтеграції з іншими системами управління фермерськими господарствами. також важливо забезпечити довговічність роботів, оскільки вони працюють у середовищах із високим рівнем пилу та вологи.

використання автономних роботів у будівництві у будівельній галузі безпілотні роботи знаходять застосування в завданнях, таких як 3D-друк конструкцій, моніторинг будівельних майданчиків, перевірка якості матеріалів і виконання важких фізичних робіт. ці технології сприяють зниженню витрат, підвищенню безпеки та пришвидшенню будівельних процесів. Одним із перспективних напрямів є використання роботів для автоматизованого 3D-друку будівельних конструкцій. завдяки інтеграції SLAM та нейронних мереж роботи можуть точно наносити матеріал навіть у складних умовах. це дозволяє створювати унікальні архітектурні форми та скорочувати час будівництва порівняно з традиційними методами [24].

Роботи також використовуються для моніторингу будівельних об’єктів. оснащені камерами та лідарами, вони можуть автоматично ідентифікувати дефекти конструкцій або невідповідності плану. наприклад, алгоритми YOLOv5 дозволяють швидко розпізнавати тріщини чи пошкодження матеріалів, забезпечуючи високу якість будівництва [11]. Головним викликом для роботів у будівництві є складність інтеграції в динамічне середовище, де постійно змінюються умови роботи. також важливим є питання енергоспоживання та забезпечення тривалої роботи без необхідності постійної підзарядки.

Використання автономних роботів у медицині у медичній сфері роботи все частіше застосовуються для виконання завдань, які потребують високої точності, швидкості й безпеки. це включає хірургічні операції, транспортування медикаментів і біоматеріалів, а також проведення дезінфекції приміщень.
особливо актуальним є використання роботів-хірургів, які дозволяють виконувати складні операції з мінімальним втручанням. завдяки алгоритмам глибокого навчання такі системи можуть аналізувати дані в реальному часі, підвищуючи точність і безпеку операцій. наприклад, технологія da Vinci Surgical System використовує роботів для виконання хірургічних втручань із високою точністю, мінімізуючи ризики для пацієнтів [3].

Роботи також застосовуються для доставки ліків у лікарнях, що особливо важливо під час пандемій чи інших критичних ситуацій. вони здатні швидко та

безконтактно переміщувати медикаменти між відділеннями, зменшуючи ризик зараження медичного персоналу [5]. Серед недоліків використання роботів у медицині — висока вартість розробки та впровадження таких систем, а також необхідність точного налаштування алгоритмів для роботи у специфічних медичних умовах.

Розвиток автономних роботів у військовій сфері, військові роботи використовуються для виконання широкого спектра завдань, таких як розвідка, розмінування, моніторинг територій і навіть активна участь у бойових діях. їхня автономність дозволяє знижувати ризики для життя військових та забезпечувати виконання завдань у складних і небезпечних умовах. Сучасні військові роботи оснащуються системами глибокого навчання, які дозволяють ідентифікувати об’єкти, ухвалювати рішення в реальному часі та адаптуватися до змін у середовищі. наприклад, роботи на основі YOLOv5 здатні швидко визначати небезпечні об’єкти, такі як міни, і вживати заходів для їхнього знешкодження [11].

Додатково роботи використовуються для логістичної підтримки, наприклад, доставки боєприпасів або евакуації поранених. завдяки інтеграції SLAM та технологій IoT вони можуть ефективно працювати навіть у зонах із обмеженим доступом до GPS [24].

викликами для військової робототехніки є забезпечення надійності систем в умовах екстремальних температур і високого рівня перешкод, а також вирішення етичних питань, пов’язаних із автономними бойовими системами.

Таким чином, аналіз переваг і недоліків існуючих методів управління безпілотними роботами демонструє значні досягнення у сфері автономності, точності та ефективності цих систем. Водночас, кожен із розглянутих підходів має свої обмеження, які варто враховувати при розробці нових методів.

Таблиця 2.1

Таблиця порівняння методів управління

Метод	Опис	Приклад застосування	Переваги	Недоліки
Класичні алгоритми	Використання поведінкових дерев, скінченних автоматів для заздалегідь відомих сценаріїв.	Промислові роботи, передбачувані умови.	Простота реалізації	Обмеженість адаптації у складних середовищах
Глибоке навчання	Нейронні мережі, такі як YOLO та Faster R-CNN, для розпізнавання об’єктів.	Дрони, роботи для моніторингу.	Швидкість, точність.	Висока обчислювальна складність.
Посилене навчання	Алгоритми, що навчаються на основі нагород і штрафів.	Роботи для навігації у динамічних середовищах.	Адаптивність до змін.	Потреба у тривалому навчанні.
Ф’юзія даних із сенсорів	Об’єднання даних від камер, лідарів, GPS за допомогою фільтру Калмана.	Автономні транспортні засоби, роботи для досліджень.	Покращена точність орієнтації.	Залежність від якості сенсорів.
Технологія SLAM	Одночасна локалізація та картографування середовища.	Роботи для дослідження невідомих територій.	Можливість роботи у невідомому середовищі.	Високі вимоги до ресурсів.
Посилене навчання	Алгоритми, що навчаються на основі нагород і штрафів.	Роботи для навігації у динамічних середовищах.	Адаптивність до змін.	Потреба у тривалому навчанні.

2.3 Середовище моделювання та візуалізації

Одним із найефективніших засобів дослідження в цій галузі є Gazebo[15]. Це потужне середовище для моделювання та симуляції роботів, включаючи безпілотні пристрої (рис. 2.1). Gazebo дозволяє точно відтворювати реальні умови та тестувати алгоритми у віртуальному середовищі, що робить його ідеальним інструментом для проведення досліджень у безпілотній техніці.

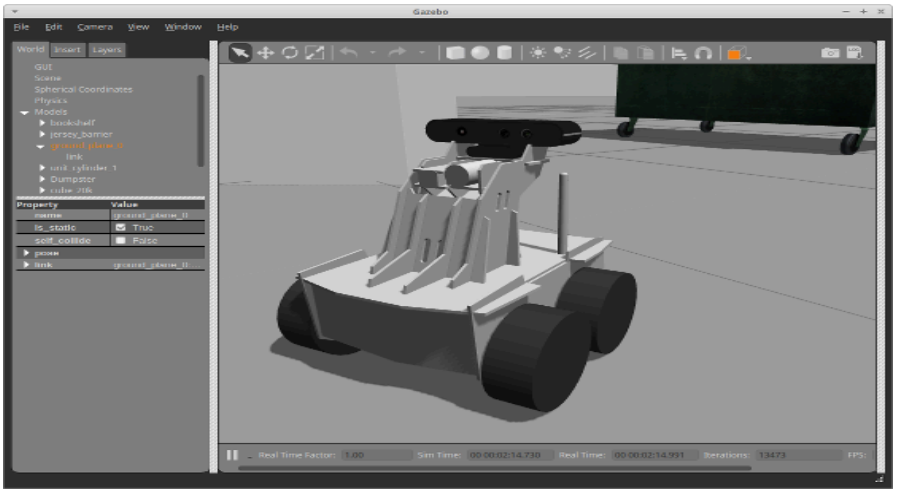


Рис. 2.1 Вигляд моделі в Gazebo

Gazebo — це високопродуктивний 3D-симулятор робототехніки з відкритим кодом, який відтворює фізичні закони реального світу з високою точністю. Його використання надає розробникам можливість швидко тестувати алгоритми та моделювати роботів у віртуальному середовищі. Однією з ключових переваг Gazebo є можливість створення динамічної симуляції за допомогою різних фізичних механізмів, як-от ODE, Simbody, Bullet і DART. Це дозволяє точно відтворювати умови реального життя, як-от сила тяжіння, тертя та крутні моменти, що впливають на успіх симуляції.

Основні переваги використання Gazebo – це можливість інтегрування різноманітних датчиків та інструменти для тестування та розробки роботів. Ба більше, Gazebo стає невід'ємною частиною для тих випадків, коли доступ до

реального роботизованого обладнання обмежений, або коли необхідно тестувати сотні роботів одночасно.

Навіть при наявності апаратного забезпечення Gazebo є корисним, оскільки воно дозволяє ефективно перевіряти робототехнічний дизайн до його впровадження в реальному середовищі (рис. 2.2).

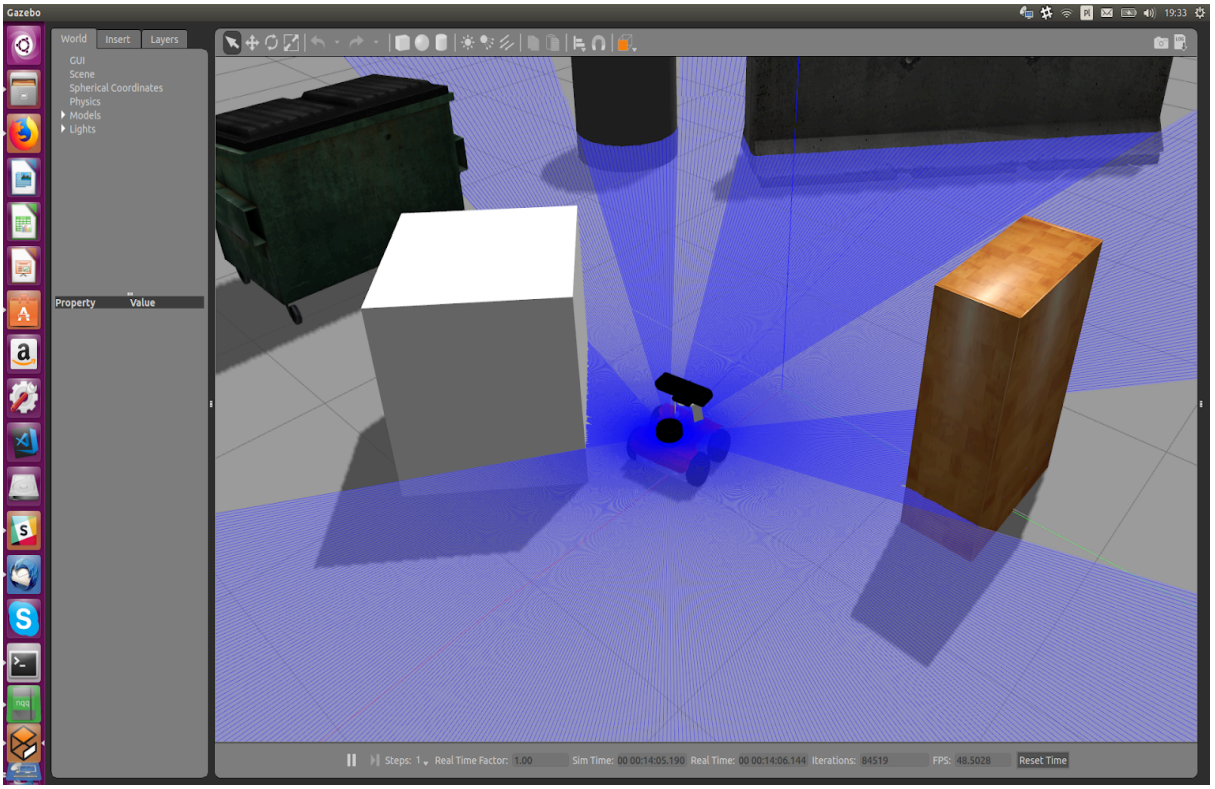


Рис. 2.2 Симулювання сенсору Lidar в Gazebo

Для детального аналізу даних у сфері робототехніки та візуалізації: навігації, сенсорів та взаємодії робота з навколишнім середовищем використовується інструмент – **Rviz**[21]. Завдяки його можливостям, дослідники та розробники отримують можливість взаємодіяти з даними з сенсорів та відстежувати роботів у реальному часі.

Rviz використовується для наочного представлення результатів роботи алгоритмів, випробування навігації, аналізу даних від датчиків та вдосконалення алгоритмів машинного зору. Його графічний інтерфейс дозволяє відображати точкові хмари, об'єкти, траєкторії та багато іншого.

Основні функції Rviz:

- **Візуалізація Сенсорів**. Дозволяє в реальному часі спостерігати за даними з різноманітних сенсорів, таких як лазерні сканери, камери, або датчики відстані.
- **Траєкторії та Рух Роботів**. Забезпечує відслідковування та візуалізацію шляхів руху роботів в реальному часі, що дозволяє виробникам та дослідникам оптимізувати траєкторії руху.
- **Аналіз Об'єктів та Взаємодія**. Допомагає в аналізі об'єктів, які робот може виявляти та взаємодіяти з ними, що є критичним для розробки систем автономного стеження та управління.

Rviz є важливим інструментом у сфері дослідження робототехніки, де точність та швидкість аналізу даних є критичними для розвитку та вдосконалення робототехнічних систем (рис. 2.3).

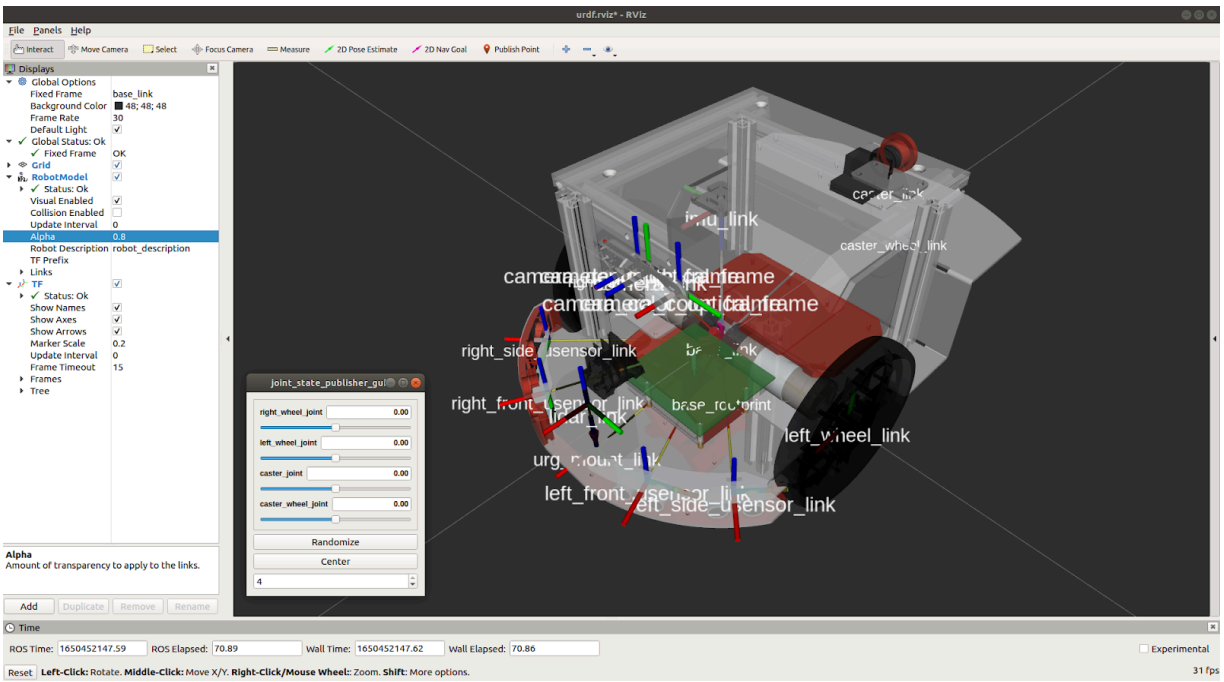


Рис. 2.3 Вигляд моделі робота в Rviz

Перехід від Gazebo до Rviz може бути логічним та ефективним кроком, особливо якщо ваша задача полягає в детальному вивченні навігації, сенсорів та взаємодії робота з навколишнім середовищем. Враховуючи, що Gazebo і Rviz

часто використовуються в сфері робототехніки, їхня комбінація може забезпечити широкий спектр можливостей для дослідження та вдосконалення робототехнічних систем (рис. 2.4).

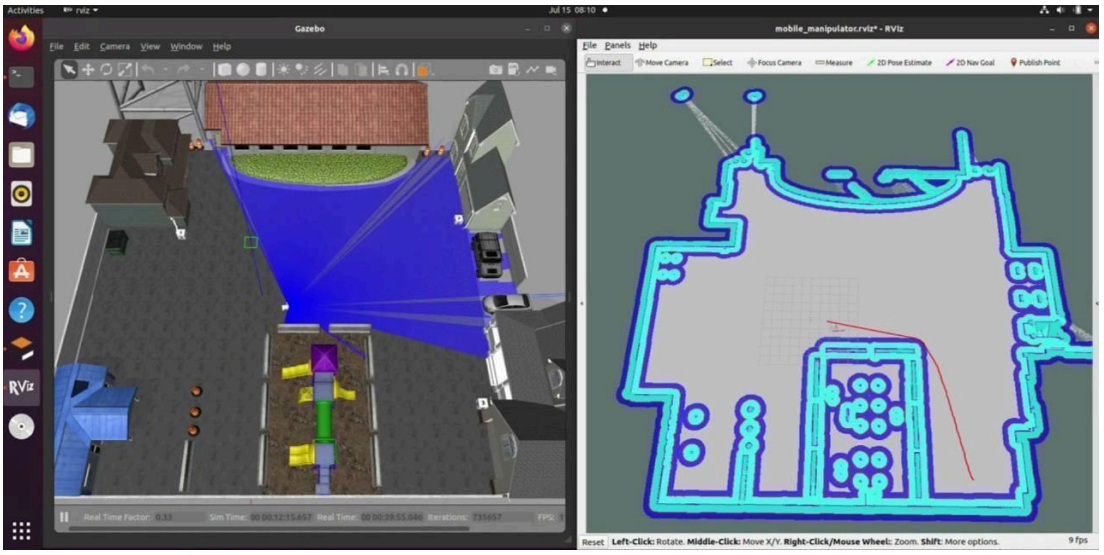


Рис. 2.4 Взаємодія Rviz та Gazebo

2.4 Аналіз сфер діяльності

Автоматичні безпілотні роботи (АБР) відіграють важливу роль у великому спектрі сфер і допомагають значно полегшити рутинні роботи та завдання, що вимагають великої точності. Їхнє використання забезпечує покращення ефективності та оптимізацію робочих процесів у різних галузях.

Нижче розглянуто деякі з секцій, де автоматичні безпілотні роботи виявляються найбільш корисними:

Виробництво, логістика та склад – добре відомі своїми трудомісткими, але виснажливими обов’язками, логістичні об’єкти забезпечують невигідну кар’єру для працівників. Проте завдяки використанню автономних безпілотних роботів можна вивільнити людські таланти, не порушуючи точно налаштовані повторювані цикли, на які покладається логістичний бізнес. Автономні безпілотні роботи, які переміщують інвентар у межах підприємства, можуть транспортувати товари, сировину чи готову продукцію з одного місця в інше. Ці

АБР-роботи оснащені LIDAR, камерами та ультразвуковими датчиками, що дозволяє їм самостійно переміщатися в межах закладу. Вони йдуть заздалегідь визначеними шляхами або використовують карти в реальному часі, щоб визначити найефективніший маршрут до місця призначення. Транспортування запасів і продукції з одного місця в інше в межах об’єкта, як правило, є завданням із низьким рівнем кваліфікації, яке не додає жодної цінності продукту чи операції. У такий спосіб, це часто є одним із перших завдань, яке потрібно автоматизувати, коли операція вирішує, що це виправдано.

Автоматизація транспортування продуктів означає, що працівники можуть залишатися на своїй основній робочій зоні, щоб виконувати інші, більш цінні завдання (рис. 2.5).

Рис. 2.5 АБР використовується для транспортування вантажів

Автономні безпілотні роботи (АБР) також можуть зіграти важливу роль у сортуванні. Різні моделі оснащені різними технологіями обробки. Від конвеєрних роликів до лотків, що нахиляються, і систем поперечних стрічок, АБР обладнані для широкого спектру рішень сортування:

- Високошвидкісна сортування посилок. Використовується для швидкого та ефективного сортування посилок за допомогою різних технологій обробки.

- Виконання замовлення електронної комерції. АБР здатні ефективно сортувати товари для виконання замовлень із сфери електронної комерції.
- Обробка повернень. Забезпечує ефективний процес обробки повернених товарів та їх повторного включення в систему.
- Короткочасне сортування. АБР можуть швидко та точно сортувати товари в короткі терміни.

Високошвидкісне сортування за допомогою АБР легко досягається за допомогою парку моделей, як-от TiltSort-Bot[38]. Ці боти працюють на мезоніні з жолобами для розташування або замовлення позицій. Інтегровані камери індукційної станції зчитують штрих-коди, а АБР рухається до жолоба призначення, нахиляє предмет униз та доставляє його до точки призначення завдання (рис. 2.6).

Рис. 2.6 TiltSort-Bot використовується для сортування

Поверхове сортування для електронної комерції, повернень, комплектації та виконання замовлень можна виконати за допомогою сортувальників типу HighTilt-Bot. Ці пристрої оснащені верхньою частиною підноса, що нахиляється, і працюють безпосередньо на підлозі.

АБР також використовуються для консолідації та сортування повернень за допомогою робочих станцій. Вони приносять порожні контейнери або піддони, а оператори направляють їх для відновлення процесу сортування (рис. 2.7).

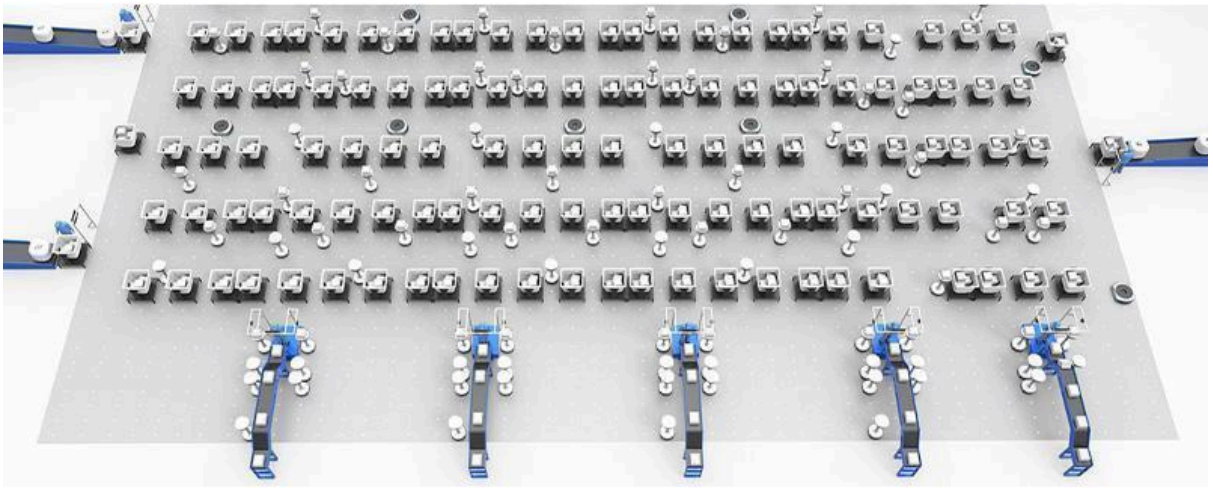


Рис. 2.7 Сортувальний центр з використанням АБР

Оскільки АБР нелінійні, їх можна використовувати для визначення послідовності продукту під час комплектації та транспортування. Це дає можливість оптимізувати процес сортування та максимізувати ефективність управління запасами.

Сільське господарство - мета сільськогосподарської робототехніки – допомогти сектору підвищити його ефективність і прибутковість процесів. Іншими словами, безпілотна робототехніка працює в сільськогосподарському секторі для підвищення продуктивності, спеціалізації та екологічної стійкості.

Інновації щодо застосування робототехніки в сільському господарстві значно просунулися за останні 5 років. Дефіцит робочої сили, підвищений споживчий попит і високі витрати на виробництво є одними з факторів, які прискорили автоматизацію в цьому секторі з метою зниження витрат і оптимізації врожаїв.

Сільськогосподарські роботи використовуються різними способами. Зовсім недавно для автоматизації збору фруктів використовуються автономні

безпілотні роботи (АБР), які використовують передові системи 3D-бачення та гнучкі захвати. Автоматизація розплідників також стала тенденцією зростання сільськогосподарських роботів, оскільки ініціативи вертикального та міського землеробства стають все більш популярними (рис. 2.8).

Серед інших нових застосувань сільськогосподарських роботів:

- Роботизоване обприскування для боротьби з бур'янами
- Збирання даних про стан врожаю
- Автоматизоване інтелектуальне збирання врожаю
- Роботизована посадка та посів
- Роботизоване обрізання та проріджування

Рис. 2.8 АБР використовується в сільському господарстві

Ринок сільськогосподарських роботів стрімко зростає. Через низку суспільних і ринкових факторів фермери впроваджують роботизовані технології для підтримки продуктивності та підвищення врожайності. У міру впровадження сільськогосподарських роботів використовуватимуться для виконання різноманітних завдань, а сільське господарство продовжуватиме перетворюватися на сучасний, автоматизований та інтелектуальний процес. Впровадження робототехніки в сільське господарство покращує як продуктивність, так і умови праці для фермерів і робітників. Інтелектуальні системи стають ідеальним рішенням для розвитку точного землеробства.

Сьогодні велика кількість сільськогосподарських операцій вже виконується автономно. У такий спосіб, колаборативні роботи зараз широко використовуються для збору врожаю фруктів або щеплення комах і вирощування, де штучний інтелект надає прогнозні дані для оптимізації ферм і плантацій.

Військовий та оборонний сектор - впровадження робототехніки відзначає суттєві зміни у військовому та оборонному секторах всього світу, завдяки вдосконаленим функціям. Декілька типів роботів відіграє ключову роль у наземних місіях та захисті країни від майбутніх загроз, зокрема, у нейтралізації бомб. Глобальний ринок військової та оборонної робототехніки проявляє великий потенціал для нових додатків, які дають конкурентну перевагу надзвичайними військовими можливостями. Уряди вкладають мільйони доларів у робототехніку для прискорення військових застосувань.

Транспортні роботи виявляються ефективними у військовій та оборонній сферах, сприяючи перенесенню важкого обладнання солдатами. Завдяки поєднанню штучного інтелекту та робототехніки забезпечується автоматизоване транспортування вантажів, зменшуючи фізичне навантаження на солдатів та допомагаючи їм зосередитися на інших аспектах місій.

Безпілотні роботи виконують різноманітні завдання, від розвідувального патрулювання до нейтралізації бомб. Обладнані високотехнологічними

датчиками і камерами, вони забезпечують відео та зображення для ефективного виконання завдань.

Роботи спостереження використовують точні та безпечні системи відеоспостереження для покращення безпеки та спостереження за територією. З інфрачервоним або нічним баченням вони надають здатність спостерігати за різними умовами, не відсилаючи солдатів.

Пошуково-рятувальні роботи відіграють важливу роль у врятуванні солдатів з надзвичайних ситуацій. Їх використання дозволяє здійснювати пошук, відстеження та порятунок солдатів у різних середовищах, у тому числі із хімічним, ядерним, радіологічним та біологічним загрозами.

Бойові роботи, обладнані розширеними функціями, можуть відігравати вирішальну роль у різних бойових завданнях, забезпечує.

Роботи-мінувальники використовуються для встановлення мін або вибухових пристроїв на території. Ці роботи найчастіше використовуються військовими чи спецслужбами для проведення операційного мінування. Вони дозволяють віддалено встановлювати вибухівку чи міну, що допомагає зменшити ризик для операторів у воєнних умовах.

Камікадзе-роботи, або бойові дрони, мають завдання виконати спеціальне завдання і вибухнути визначеною метою. Ці роботи можуть бути використані для точного ураження важливих об'єктів супротивника, знижуючи ризик для власних військ. Вони зазвичай оснащені камерами, системами автономного навігації та вибуховими пристроями (рис. 2.9).

Рис. 2.9 Використання АБР в військовому секторі

Автономні безпілотні роботи (АБР) відіграють важливу роль у різних галузях, включаючи логістику, сільське господарство, військову сферу та інші. Використання АБР в секторі логістики сприяє підвищенню ефективності, безпеки та продуктивності в різних логістичних об'єктах. У сільському господарстві вони допомагають оптимізувати процеси, збільшуючи врожайність та знижуючи витрати. У військовій галузі АБР забезпечують розвідку, безпечну та ефективну нейтралізацію загроз, а також підвищують здатність реагування на потенційні виклики.

Застосування АБР дозволяє вирішувати завдання, які можуть бути важко доступними, небезпечними чи трудомісткими для людей. Їхня гнучкість і автономність роблять їх важливими інструментами в сучасних технологічних вирішеннях.

3 РОЗРОБКА МЕТОДУ УПРАВЛІННЯ БЕЗПІЛОТНИМ РОБОТОМ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Архітектура програмного забезпечення для управління безпілотним роботом

Реалізація методу управління безпілотним роботом базується на використанні об’єктно-орієнтованого підходу, що забезпечує модульність, та зручність у розробці. Основні компоненти системи представлені у вигляді класів, які відповідають ключовим елементом алгоритму управління.

Структура класів:

- Клас **Robot:**
    - Відповідає за загальну координацію роботи робота, включаючи управління сенсорами, модулем ухвалення рішень та руховими модулями.
    - Основні методи:
        - **initialize() –** ініціалізація всіх компонентів робота.
        - **getStatus() –** отримання поточного статусу робота.
- Клас **SensorSystem**
    - Відповідає за збір даних із сенсорів, таких як камери, LiDAR тощо.
    - Основні методи:
        - **collectData() –** збір необроблених даних.
        - **getSensorData() –** передача зібраних даних модулю обробки.
- Клас **AIController**
    - Реалізує обробку даних із використанням моделі YOLOv5 та інших алгоритмів штучного інтелекту.
    - Основні методи:
        - **processData() –** обробка вхідних даних із сенсорів.
        - **makeDecision() –** прийняття рішень для управління роботом.

- Клас **NavigationSystem**
    - Забезпечує планування та виконання траєкторії руху.
    - Основні методи:
        - **planRoute() –** побудова маршруту.
        - **moveTo() –** виконання руху до цільової точки.
- Клас **ErrorHandler**
    - Відповідає за виявлення та обробку помилок у роботі системи.
    - Основні методи:
        - **logError() –** запис помилок.
        - **resolveError() –** виправлення помилок.

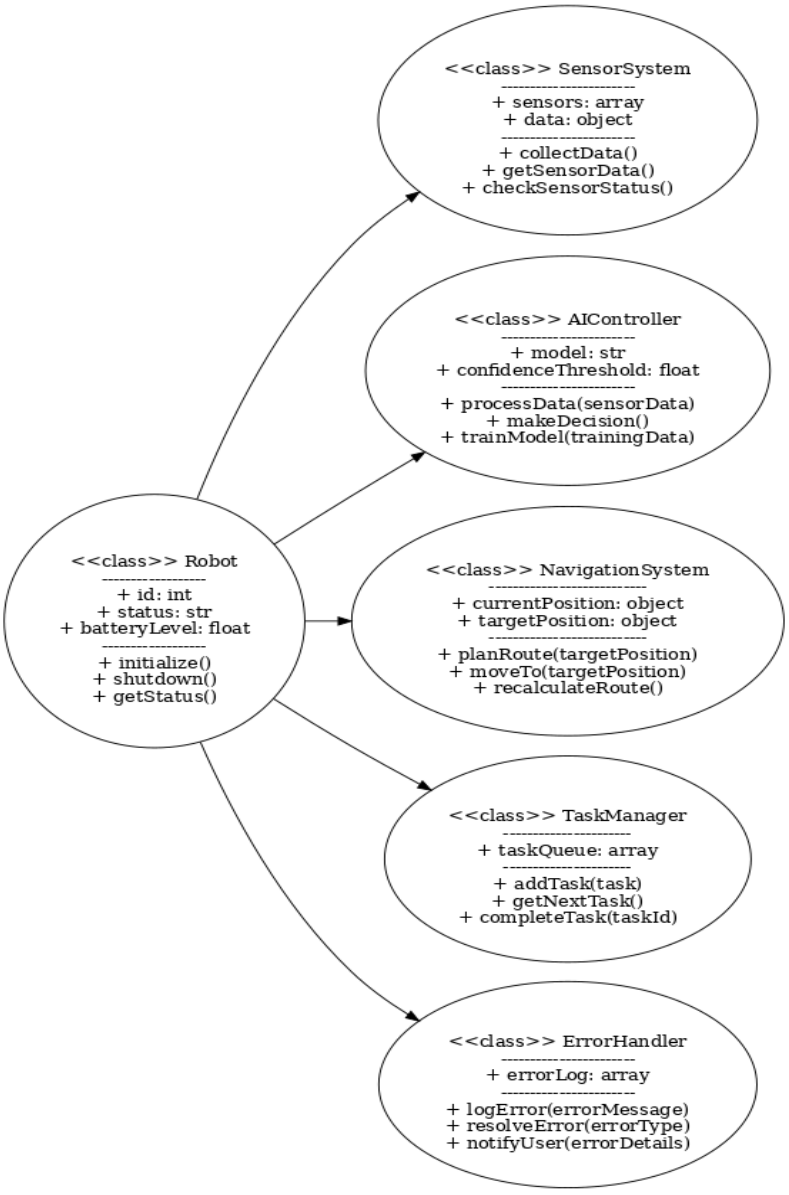


Рис. 3.1 Діаграма класів

Кожен клас тісно взаємодіє з іншими, формуючи єдину систему управління роботом:

- клас Robot координує всі компоненти;
- SensorSystem передає дані у AIController, де вони обробляються;
- AIController передає команди до NavigationSystem для виконання руху;
- у разі виникнення помилок ErrorHandler запускає процедуру виправлення.

3.2 Математична модель методу управління безпілотним роботом на основі штучного інтелекту

Модель YOLOv5 (You Only Look Once, версія 5) широко використовується для задач розпізнавання об’єктів у реальному часі завдяки високій швидкості та точності. Її інтеграція в систему управління безпілотним роботом дозволяє забезпечити автоматичне виявлення перешкод, ідентифікацію цільових об’єктів та планування безпечних маршрутів. У цьому розділі описано математичну основу методу управління з використанням YOLOv5.

Підготовка вхідних даних

Камера передає зображення у форматі RGB або BGR з роздільною здатністю Wraw×Hraw. Для зменшення обчислювального навантаження та узгодження з архітектурою моделі YOLOv5 зображення масштабується до розміру W×H (зазвичай 640×640)

$$I_{scaled}(x', y') = I\left(\frac{x' W_{raw}}{W}, \frac{y' H_{raw}}{H}\right), \tag{3.1}$$

де *(x',y')* — координати пікселя у масштабованому зображенні, *I(x,y)* — інтенсивність пікселя початкового зображення.

Нормалізація зображення

Щоб нейронна мережа могла ефективно обробляти дані, значення пікселів нормалізуються до діапазону [0, 1]

$$I_{norm}(x', y') = \frac{I_{scaled}(x', y')}{255}. \tag{3.2}$$

Цей підхід дозволяє зменшити вплив яскравості та покращити збіжність моделі під час роботи [27].

Формат і анотації даних

Зображення передаються у форматі тензорів розмірності (C,H,W), де C — кількість каналів (3 для RGB-зображень). Додатково кожне зображення супроводжується анотаціями — координатами bounding box для об'єктів на сцені, наприклад

$$b = \{(x, y, w, h, c)\}_{i=1}^{N}, \tag{3.3}$$

де N — кількість об'єктів, x,y — координати центру об'єкта, w,h — ширина та висота bounding box, c — клас об'єкта [26].

Зменшення шуму та фільтрація

Для підвищення точності розпізнавання перед масштабуванням застосовуються методи фільтрації:

- **Білінійна інтерполяція** для згладжування зображень під час зміни розміру;
- **Медіанна фільтрація** для зменшення шумів

$$I_{filtered}(x, y) = \text{median}\{I_{scaled}(x + i, y + j) \mid -k \leq i, j \leq k\}, \tag{3.4}$$

де k — розмірність вікна фільтрації [28].

Підготовка пакетів даних

Для обробки в реальному часі дані об'єднуються у пакети (batch) із фіксованою кількістю зображень BBB. Пакет представлений тензором розмірності (B,C,H,W), що передається на вхід моделі [26].

Архітектура YOLOv5 та ідентифікація об'єктів

Основні компоненти архітектури:

Архітектура YOLO v5 складається з трьох основних блоків:

Backbone - відповідає за вилучення ключових ознак із вхідного зображення. У YOLOv5 використовується модифікована структура CSPDarknet, яка покращує збереження просторової інформації завдяки частковому поділу вхідного сигналу [27].

Формула вилучення ознак виглядає так

$$F_{i+1} = \sigma(W_i \cdot F_i + b_i) \tag{3.5}$$

де F_i — ознаки на і-му рівні, Wi , bi — параметри зважування та зсуву, σ — функція активації (Leaky ReLU).

Neck

Застосовує структуру Feature Pyramid Network (FPN) для об'єднання ознак із різних рівнів масштабів. Це дозволяє виявляти як дрібні, так і великі об'єкти. Функція злиття ознак

$$F_{merged} = \sum_{i=1}^{N} \alpha_i F_i, \tag{3.6}$$

де α_i — вагові коефіцієнти для і-го рівня.

Head

Генерує фінальні bounding box та класифікацію. Для кожного прямокутника обчислюються координати центру *(x,y)*, ширина *w*, висота *h*, ймовірність класу *c*

$$o = \text{Softmax}(W_h \cdot F_{merged} + b_h) \tag{3.7}$$

де *o* — результат для кожного об'єкта.

Ідентифікація об'єктів

Модель YOLOv5 використовує сітку, поділену на S×SS \times SS×S комірок. Кожна комірка відповідає за виявлення об'єктів, центри яких знаходяться всередині цієї області.

Кроки ідентифікації:

1. **Просторове розбиття**

   Зображення ділиться на рівномірну сітку, кожна комірка генерує набір bounding box.

2. **Обчислення обмежувальних рамок**

   Для кожної рамки передбачаються:

   - координати центру *(x,y)*

- ширина *w*, висота *h*
- впевненість у наявності об'єкта p(con)f

  Формула обчислення впевненості:

$$p_{conf} = IoU(b_{pred}, b_{true}) \cdot c_{pred} \tag{3.8}$$

де *IoU* — коефіцієнт перетину обмежувальних рамок, c_{pred} — ймовірність класу.

Фільтрація об'єктів

Застосовується алгоритм Non-Maximum Suppression (NMS), що видаляє надлишкові рамки

$$\text{NMS}(b) = \{b_i \mid IoU(b_i, b_j) < \text{threshold}, \forall j \neq i\}. \tag{3.9}$$

Класифікація

Для кожного виявленого об'єкта визначається клас за допомогою softmax-активації

$$c = \text{Softmax}(W_c \cdot F_{head} + b_c) \tag{3.10}$$

де W_c — вагові параметри для класифікації.

Структурна інтеграція в метод управління

Інтеграція YOLOv5 у метод управління передбачає використання результатів ідентифікації об’єктів для прийняття рішень щодо руху безпілотного робота. Це досягається через комбінування даних, отриманих із нейронної мережі, з алгоритмами управління, такими як PID-контролери, потенційні поля або методи кінцевого стану.

Нехай I — вхідне зображення середовища, отримане з камери робота. YOLOv5 обробляє це зображення та видає множину результатів **R**, де

$$\mathbf{R} = \{(c_i, x_i, y_i, w_i, h_i, p_i)\}_{i=1}^{n}, \tag{3.11}$$

де c_i— клас об’єкта, (x_i,y_i) — координати центру об’єкта, w_i, h_i— ширина і висота області об’єкта, p_i— впевненість моделі в ідентифікації об’єкта.

Ці результати перетворюються на цільові сигнали управління через функцію Ф, яка враховує поточний стан робота та його положення в середовищі

$$\Phi(\mathbf{R}, \mathbf{S}) = \mathbf{C}, \tag{3.12}$$

де **S** — стан робота, а **C** — команда управління.

Використання результатів ідентифікації для планування руху

Алгоритм управління ґрунтується на двох основних етапах:

1. **Визначення об'єктів, що потребують реакції**: на основі результатів RR виконується класифікація об'єктів за важливістю. Наприклад, перешкоди, такі як стіни або рухомі об'єкти, отримують високий пріоритет. Вектор реакції **V** визначається як

$$\mathbf{V} = \sum_{i=1}^{n} w(c_i) \cdot \mathbf{d}_i, \tag{3.13}$$

де $w(c_i)$ — вагова функція, яка залежить від класу об'єкта, а d_i— вектор напрямку до об'єкта i.

2. **Планування руху**: використовуючи V\mathbf{V}V, робот обчислює новий напрямок руху, мінімізуючи можливість зіткнень. Це досягається через модифікацію стандартної функції потенційного поля

$$U(\mathbf{x}) = \sum_{i=1}^{n} \frac{w(c_i)}{\|\mathbf{x} - \mathbf{x}_i\|} - k\|\mathbf{x} - \mathbf{x}_{\text{goal}}\|, \tag{3.14}$$

де x_i— координати об'єкта, x_{goal}— цільова точка, k — коефіцієнт приваблення.

Врахування динамічних перешкод

При інтеграції в реальному середовищі алгоритм враховує динамічні зміни положення об'єктів. На кожному кроці часу модель YOLOv5 оновлює R, що дозволяє алгоритму адаптувати Φ до нових умов.

Для передбачення руху перешкод використовується модель

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta t \cdot \mathbf{v}_i, \tag{3.15}$$

де v_i — оцінка швидкості об'єкта, отримана на основі послідовних кадрів.

Оптимізація обчислень

Для забезпечення реального часу інтеграції YOLOv5 застосовуються такі оптимізації:

- Використання апаратного прискорення (наприклад, GPU або TPU).
- Пріоритизація обробки найближчих об’єктів.
- Скорочення роздільної здатності вхідного зображення для далеких кадрів.

3.3 Опис розробки методу

У результаті основних теоретичних досліджень і аналізу наявних технологій було прийняте стратегічне рішення розробити метод управління роботом із функціоналом автоматичного планування маршрутів та уникнення перешкод. Це рішення базується на необхідності вдосконалення системи навігації, особливо у складних умовах, де традиційні підходи демонструють обмежену ефективність.

Розробка методу управління роботом базується на інтеграції знань із робототехніки, комп'ютерного зору, штучного інтелекту та систем навігації. Такий підхід дозволяє створити ефективну платформу, здатну аналізувати навколишнє середовище, будувати оптимальні маршрути та долати перешкоди автономно, без активного втручання оператора.

У межах проекту обрано операційну систему Ubuntu, яка забезпечує стабільність, гнучкість і підтримку відкритого програмного забезпечення, необхідного для роботи сучасних фреймворків, таких як ROS2. Серед численних версій ROS2 вибір зупинився на фреймворку **Humble**, який пропонує покращені інструменти для моделювання, візуалізації та інтеграції з сенсорами та механізмами управління. Це рішення забезпечує масштабованість та гнучкість у створенні складних систем.

Розроблений метод управління враховує такі ключові аспекти:

- **Інтеграцію сенсорів і приводів**: Забезпечується точність аналізу навколишнього середовища, що дозволяє виконувати складні завдання.

- **Автономну навігацію**: Завдяки використанню ROS2 та фреймворку Humble, метод дозволяє планувати маршрути в реальному часі.
- **Модульність системи**: Відкрита архітектура Ubuntu та ROS2 забезпечують зручне розширення функціональності системи.

Базування системи на Ubuntu також дозволяє використовувати доступні ресурси спільноти, такі як документація, пакети програмного забезпечення, та інтегрувати їх у процес розробки. Це значно спрощує тестування та впровадження розроблених рішень у реальному середовищі.

Таким чином, розробка методу управління роботом із використанням ROS2, зокрема фреймворку Humble, створює основу для подальшого вдосконалення автономних систем управління, що відповідають сучасним вимогам до надійності, ефективності та масштабованості.

Встановлюємо Humble за допомогою Debian-пакетів, які забезпечують зручну установку та підтримку програми. Перед встановленням Humble рекомендується переконатися, що ваше середовище підтримує UTF-8 кодування, оскільки це може вплинути на коректну роботу програми. У випадку мінімальних середовищ, таких як Docker-контейнери, локаль може бути встановлена у мінімальному обсязі, наприклад, POSIX. Проте, Humble працює ефективно з іншими мовами, які підтримують UTF-8, що забезпечує стабільну та коректну роботу програми в різних мовних середовищах.

locale # check for UTF-8

sudo apt update && sudo apt install locales

sudo locale-gen en_US en_US.UTF-8

sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8

export LANG=en_US.UTF-8

locale # verify settings

Встановлюємо ROS2 apt в наше середовище, та пакети ROS2 Humble, демо приклади, та туторіали.

sudo apt install ros-humble-desktop

sudo apt install ros-humble-ros-base

sudo apt install ros-dev-tools

sudo apt install ros-humble-ros2-control

Також важливим етапом встановлення є установка пакетів Nav2[20] (Navigation2), які відповідають за навігацію робота. Ці пакети забезпечують ключові функції навігації, включаючи планування маршрутів, управління перешкодами та рухом робота. Після встановлення Nav2 робот буде здатен впевнено та ефективно їхати по навігації у визначеному середовищі, дотримуючись визначених маршрутів та уникнення перешкод. Це важливий крок для забезпечення повноцінної роботи системи управління роботом.

sudo apt install ros-humble-navigation2

sudo apt install ros-humble-nav2-bringup

Також для тестування та симуляції робота ми встановимо пакети Gazebo, які інтегруються з ROS2, а також пакет Rviz для візуалізації та аналізу отриманих даних. Ці інструменти дозволяють вирішувати питання ефективності та працездатності робота в умовах симуляції, досліджувати його реакцію на зміни у середовищі та перевіряти алгоритми навігації без необхідності фізичної присутності робота. Це забезпечить можливість визначити та усунути потенційні проблеми ще на етапі розробки, що є критичним для подальшого успішного функціонування системи управління роботом.

sudo apt install ros-humble-gazebo-ros-pkgs

sudo apt install ros-humble-rviz2

Після успішної установки всіх необхідних пакетів для розробки робота, ми переходимо до опису моделі робота. Використання URDF (Unified Robot Description Format)[22] дозволяє нам створити детальний опис робота, визначивши його лінки (елементи тіла робота) та джоїнти (зв'язки між ланками) у тривимірному просторі (рис. 3.2).

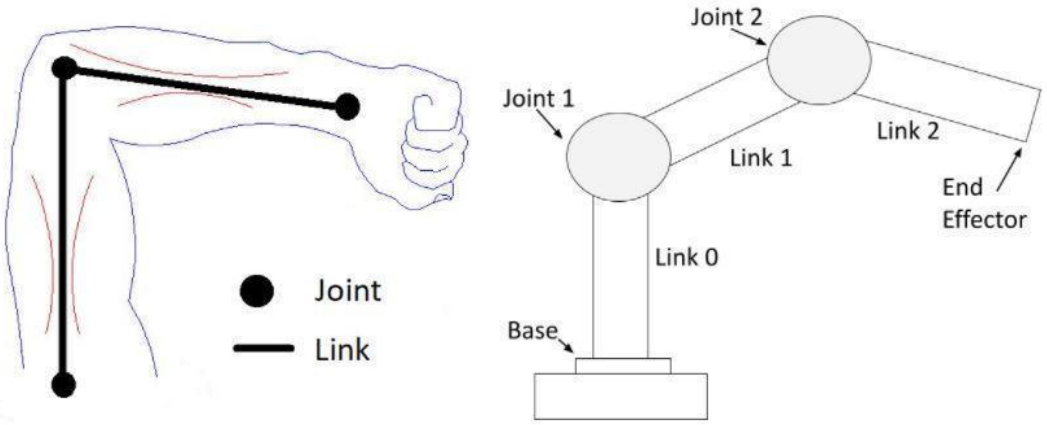


Рис. 3.2 Розуміння використання Link & Joint на прикладі людської руки та роботизованої

Це дозволяє точно визначити геометрію, розміри, маси, інерційні характеристики та кінематичні властивості робота. Для диференційного колісного робота це особливо важливо, оскільки URDF дозволяє точно визначити конфігурацію його коліс та їхню взаємодію з базовою рамою, що впливає на рухові можливості та стійкість робота. Робот використовує трансформації між різними фреймами лінків для визначення їхньої взаємодії та просторового положення в системі координат. Використання джоінтів дозволяє моделювати зв'язки між цими лінками, встановлюючи правила для їхньої взаємодії. Це надає роботу можливість адаптуватися до різних ситуацій та виконувати необхідні операції під час фізичного функціонування, забезпечуючи точну інтеракцію між його компонентами в реальному часі.

Для створення моделі автоматизованого безпілотного робота використовується URDF (Unified Robot Description Format)[22]. Спочатку створюється базовий лінк "base_link", який є системою координат, що жорстко прив'язана до кореневого тіла робота. Рекомендується базу робота як "base_link", його можна приєднати до кореня в будь-якій довільній позиції або орієнтації; для кожної апаратної платформи буде окреме місце на базі, яке стане очевидною точкою відліку. Додатково, створюється лінк "base_footprint", який є представленням положення робота на підлозі. Компонент трансляції кадру повинен бути барицентром проекцій ніг на підлозі. По відношенню до кузовної

рами, кути крену та тангажу мають дорівнювати нулю, а кут повороту має відповідати куту повороту base_link. Це необхідно для визначення точної геометрії робота в просторі та його відносин з навколишнім середовищем (рис. 3.3).

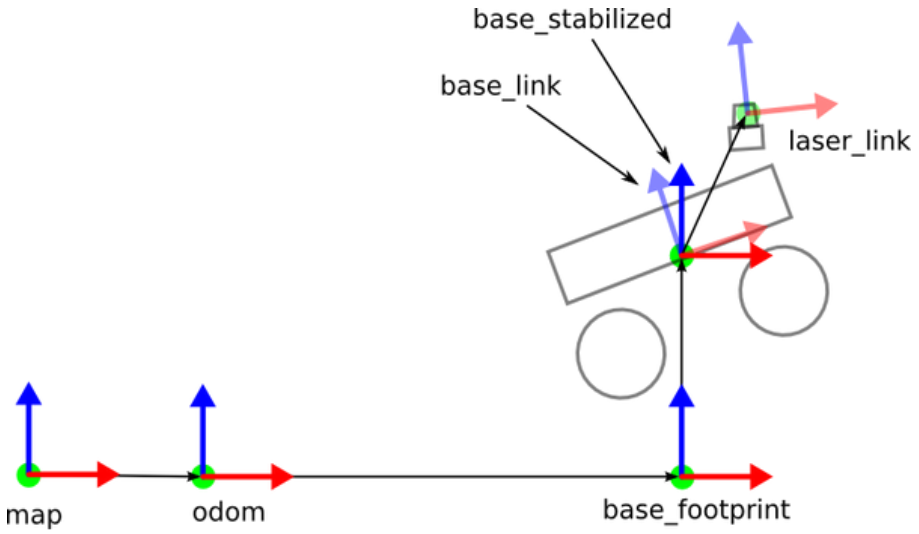


Рис. 3.3 Різниця між "base_link" та "base_footprint"

Оскільки робот є гусеничним, для відтворення цього типу руху додаються лінк та джоїнт для ведучої шестерні. Наприклад, "sprocket_left" або "sprocket_right" представляють ведучу шестерню гусеничної системи. Також вводяться лінки і джоїнти для двох не ведучих коліс ("front_wheel_left", "front_wheel_right", "back_wheel_left", "middle_wheel_right"). Для обладнання робота камерами, лідаром та іншими пристроями, створюються відповідні лінки та джоїнти. Наприклад, "camera_link" або "lidar_link" для представлення камери та лідара відповідно. Це дозволяє точно моделювати розташування та орієнтацію обладнання на роботі (рис. 3.4).

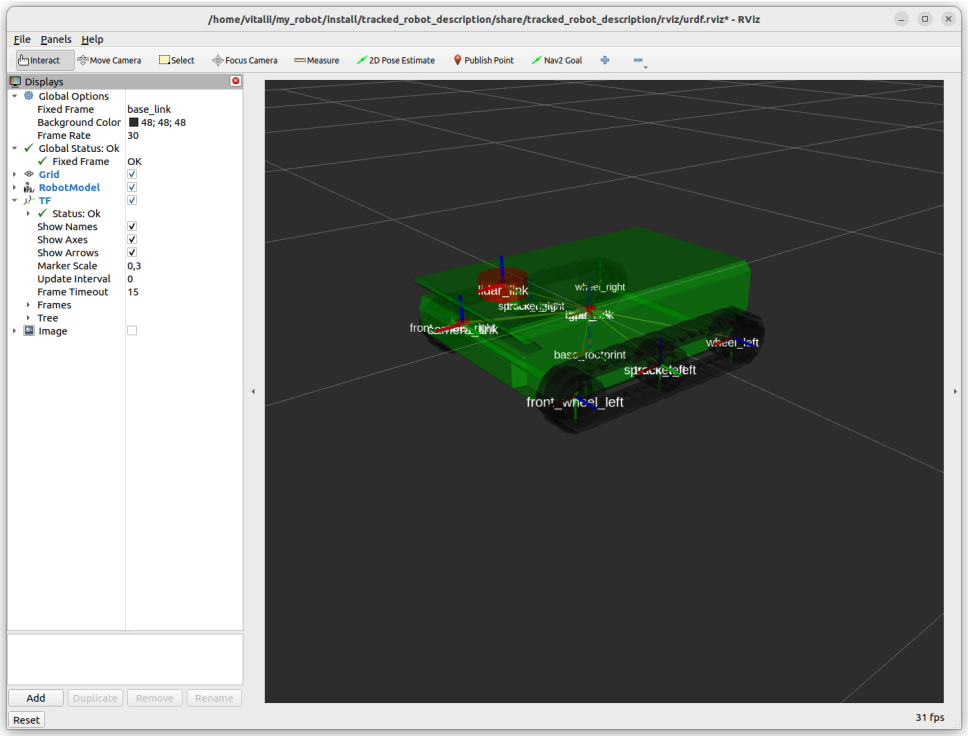


Рис. 3.4 Опис моделі робота в URDF

Після розробки моделі та опису робота переходимо до підключення плагінів Gazebo для симулювання роботи його двигунів, камери, лідару та інших пристроїв. Для цього налаштовуємо плагіни відповідно до параметрів робота. Плагіни створюють можливість відображення реальних даних в симуляційному середовищі Gazebo, відтворюючи реальну поведінку робота. Конфігуруючи плагіни для двигунів, камери, лідару та інших датчиків, ми враховуємо особливості робота і його обладнання. Параметри плагінів узгоджуються з характеристиками апаратних компонентів робота для максимально точного відтворення їх функціональності в симуляції (рис. 3.5).

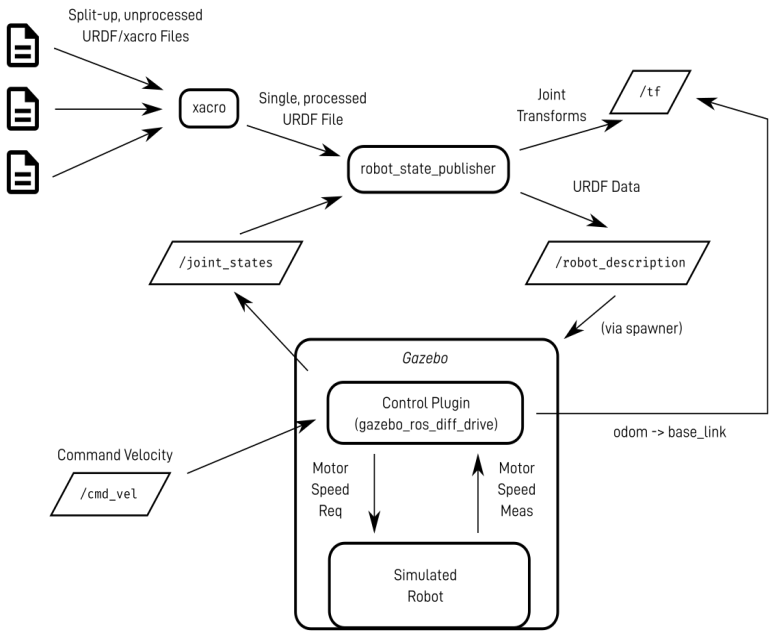


Рис. 3.5 Взаємодія Gazebo плагіну з URDF моделю робота

Завершальний результат полягає у відображенні робота в середовищі Gazebo, де він може взаємодіяти з навколишнім простором та перешкодами, відправляти та отримувати дані зі своїх датчиків, та емулювати свою реальну поведінку. Це дозволяє випробувати та тестувати алгоритми управління, навігації та сприйняття інформації, відображаючи їх результати в симуляційному середовищі (рис. 3.6).

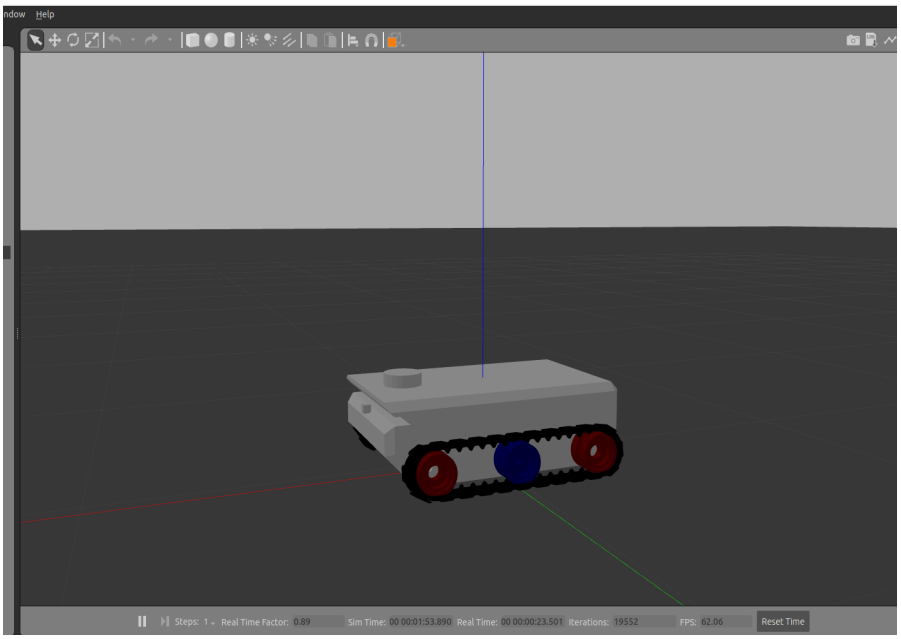


Рис. 3.6 Відображення створеного робота в Gazebo

Після успішного налаштування робота в симуляції, ми переходимо до налаштування навігаційного стеку для безпілотного робота. Він забезпечує сприйняття, планування, контроль, локалізацію, візуалізацію та багато іншого для створення високонадійних автономних систем. Це завершить моделювання навколишнього середовища на основі даних датчиків, динамічне планування шляху, обчислення швидкостей для двигунів, уникнення перешкод, представлення семантичних областей і об’єктів і структурування поведінки роботів вищого рівня. Для успішної роботи навігації необхідно мати точну інформацію про місце положення робота та його одометрію. Цю інформацію можна отримати за допомогою ф'юзії даних, об'єднуючи дані з енкодерів та інерціальної системи вимірювань (ІМУ) за допомогою фільтра Калмана.

Енкодери надають інформацію про відстань, пройдену кожним колесом робота. Дані з енкодерів допомагають визначити зміну положення робота в просторі відносно його вихідного місця.

Інерціальна система вимірювань (ІМУ) включає в себе акселерометр та гіроскоп, які вимірюють прискорення та кутову швидкість руху робота. Дані з ІМУ також використовуються для визначення зміни положення та орієнтації робота.

Для точного визначення положення робота в просторі також використовується лідар та Адаптивний метод локалізації та картографування (AMCL)[23]. Лідар надає точні відстані до навколишніх об'єктів, а AMCL використовується для абсолютного позиціонування робота в світовому просторі.

Фільтр Калмана використовується для оптимального об'єднання цих даних, забезпечуючи точне та стабільне визначення положення робота. Ф'юзія цих даних дозволяє отримати комплексну та надійну інформацію, необхідну для ефективної роботи навігаційної системи (рис. 3.7).

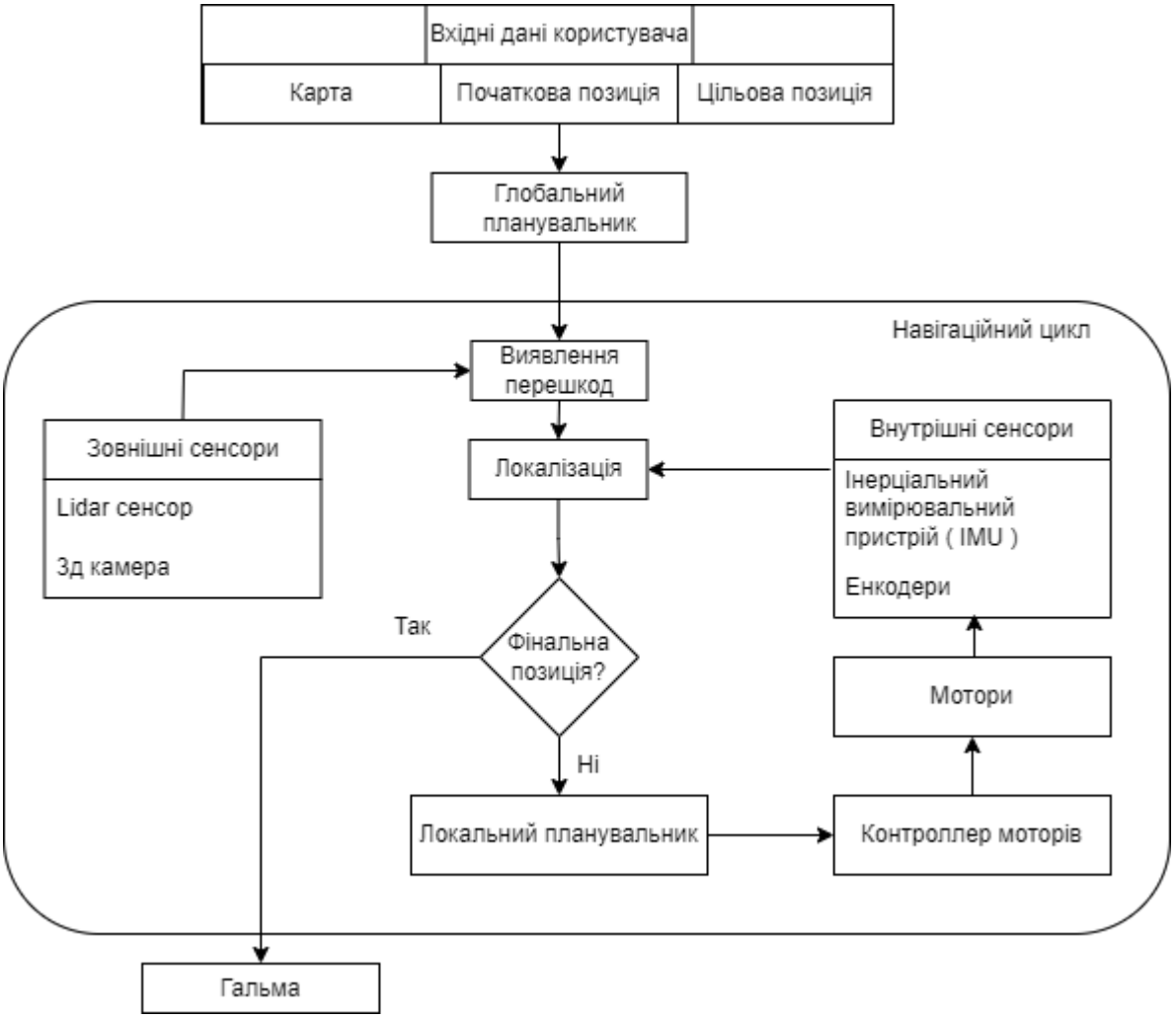


Рис. 3.7 Взаємодія сенсорів робота з навігацією

Також критично важливим етапом у розвитку автономних систем є створення локалізації для мобільного робота. Локалізація мобільного робота визначає його точне місцезнаходження в реальному часі, що стає ключовим фактором для успішної навігації та виконання завдань.

Однією з головних проблем у розробці локалізації мобільного робота є необхідність постійно підтверджувати його місцезнаходження, особливо під час руху в невідомому середовищі. Це важливо для уникнення помилок та забезпечення точності його навігації. Важливо враховувати, що локалізація не лише стосується фізичного місцезнаходження робота, але і вимагає врахування його орієнтації та обставин оточуючого середовища.

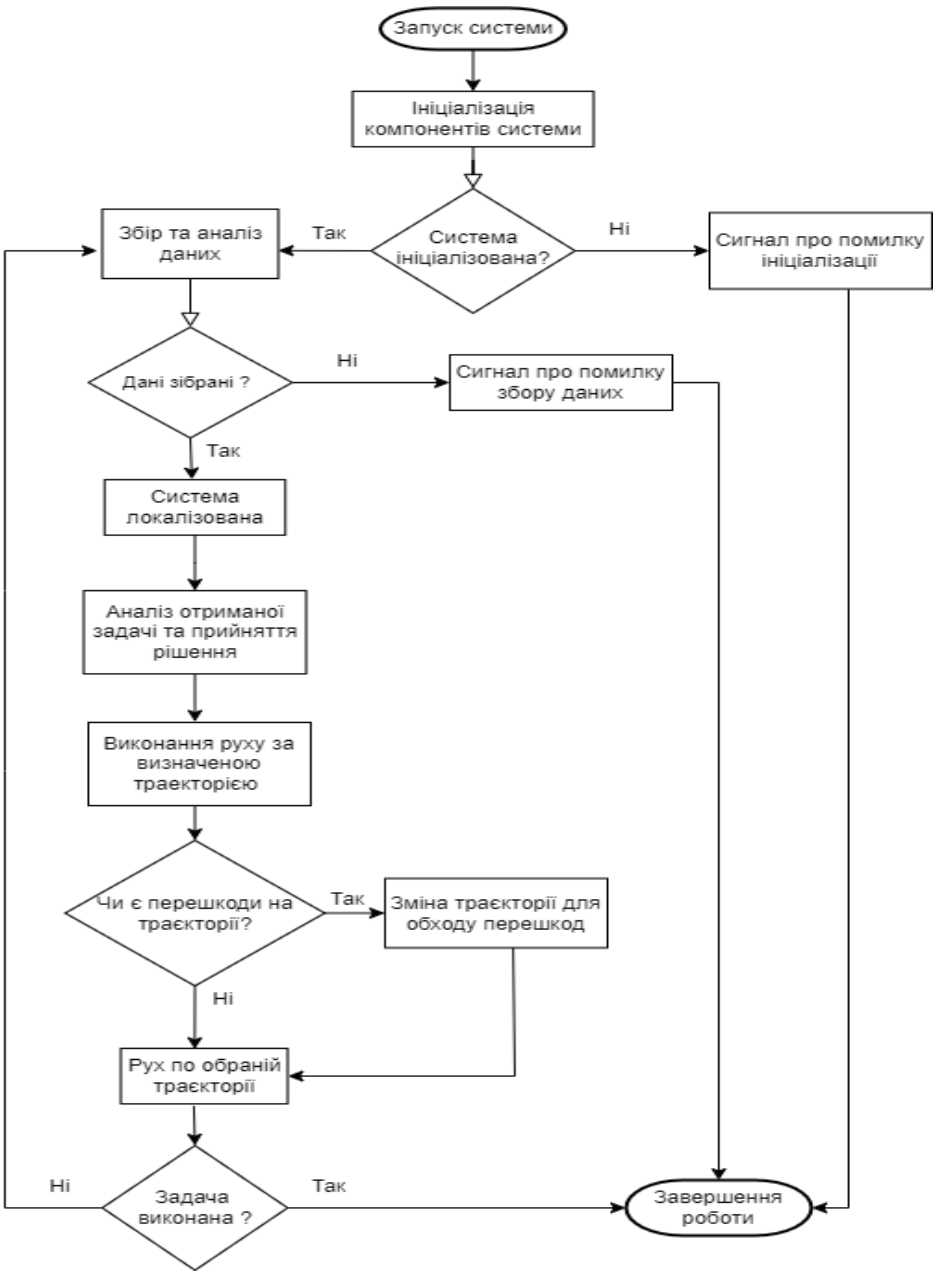


Рис. 3.8 Блок схема алгоритму управління роботом

Сучасні тенденції у розвитку локалізації мобільних роботів свідчать про важливість впровадження технології одночасної локалізації та картографування (SLAM)[24]. Це комплексне рішення, яке дозволяє роботу ефективно орієнтуватися в невідомому середовищі, використовуючи дані зі спостережень та карт. SLAM ставить під сумнів не лише питання локалізації, але й задачу створення точної карти оточуючого простору, що робить його важливим інструментом у сфері робототехніки.

При реалізації SLAM, першочерговою стає проблема локалізації, оскільки вона визначає ефективність всього процесу. Вдосконалення технології локалізації в SLAM є ключовим напрямком робіт, оскільки це впливає на якість навігації та загальну продуктивність мобільного робота (рис. 3.9).

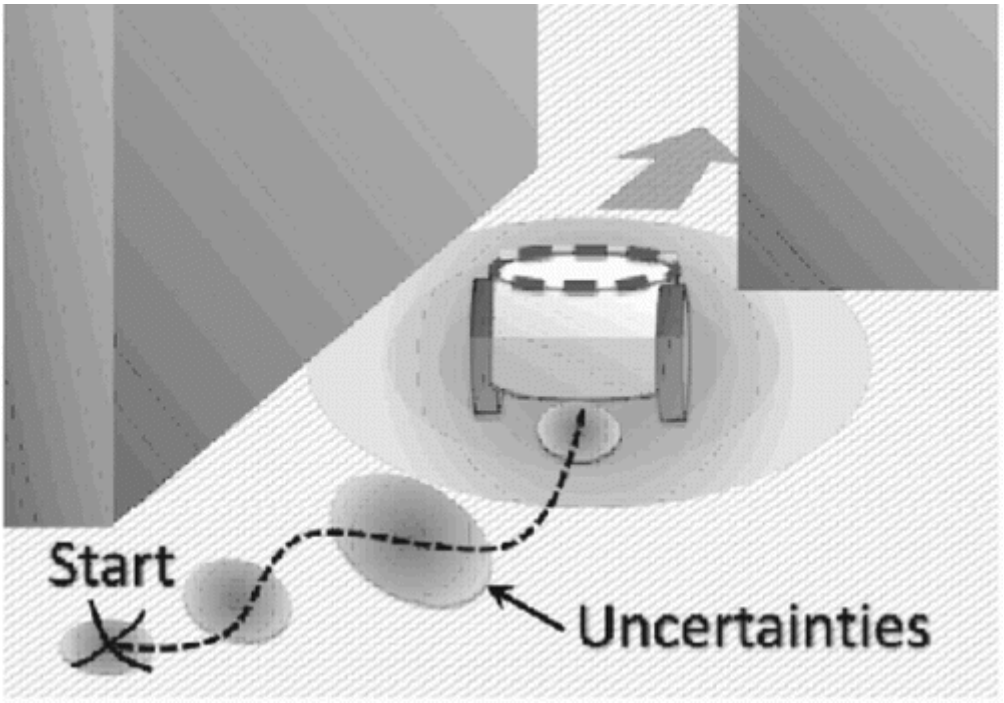


Рис. 3.9 Локалізація автономного безпілотного робота

Розглянемо локалізацію мобільного робота на базі розширеного фільтра Калмана (EKF)[25] за умови періодичних вимірювань Для досягнення високої точності та надійності визначення позиції таких роботів, використання розширеного фільтра Калмана є ключовим аспектом.

EKF – це алгоритм, що базується на математичних принципах фільтрації, і використовується для прогнозування позиції автономного безпілотного робота. В основі цього методу лежить обробка вимірювань від інерціально-магнітного пристрою (IMU), датчиків-енкодерів та GPS, що дозволяє отримувати високоточні дані про рух та відстані пройдені роботом.

Інерціально-магнітний пристрій вимірює прискорення та кутову швидкість робота, а енкодери фіксують обертання коліс та визначають відстань, пройдену роботом. Зібрані дані стають вхідними для EKF, який, в свою чергу,

використовує математичні моделі руху та розгортає їх для передбачення майбутньої позиції робота. Однією з вагомих переваг використання EKF є його здатність коректно враховувати невизначеності та шуми вимірювань. Цей алгоритм дозволяє роботу ефективно пристосовуватися до змін у середовищі та уникати систематичних помилок (рис. 3.10).

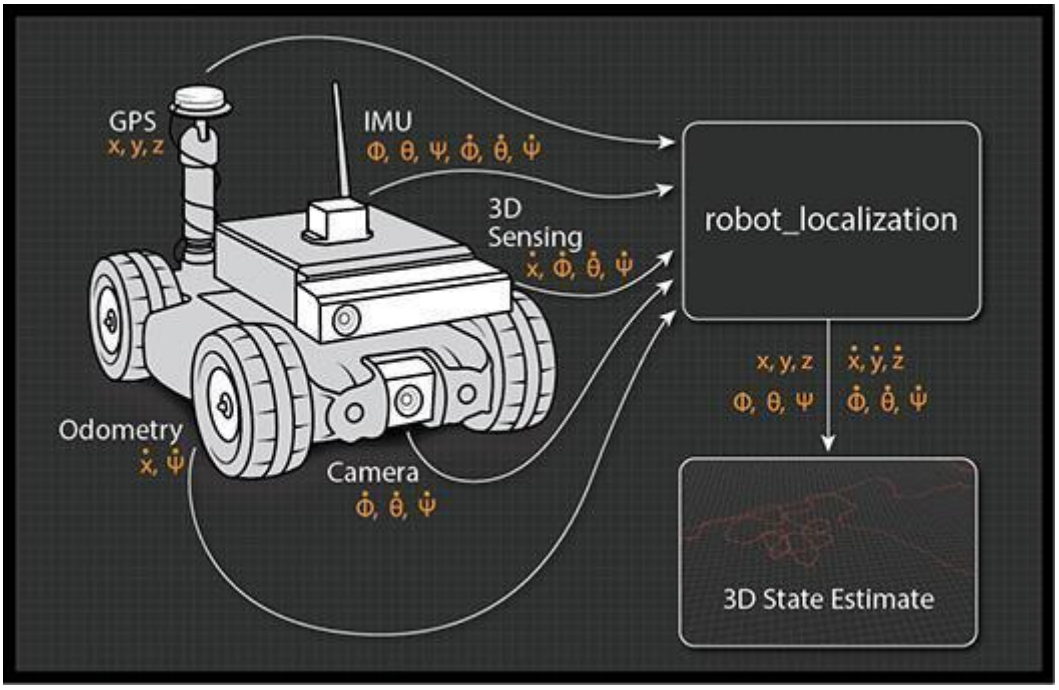


Рис. 3.10 Фільтрація даних для локалізації

У розширеному фільтрі Калмана (EKF) гнучкість визначення моделей переходу стану та спостережень дозволяє використовувати нелінійні функції стану. Важливою характеристикою EKF є те, що ці функції не обов'язково повинні бути лінійними, а можуть представляти собою диференційовані функції.

На кожному кроці часу оцінюються дані з поточними прогнозованими станами. Ці матриці часткових похідних використовуються в рівняннях фільтра Калмана. Цей процес суттєво покращує нелінійну функцію навколо поточної оцінки, забезпечуючи ефективне використання EKF для прогнозування та корекції стану системи в умовах нелінійності та шумів вимірювань (рис. 3.11).

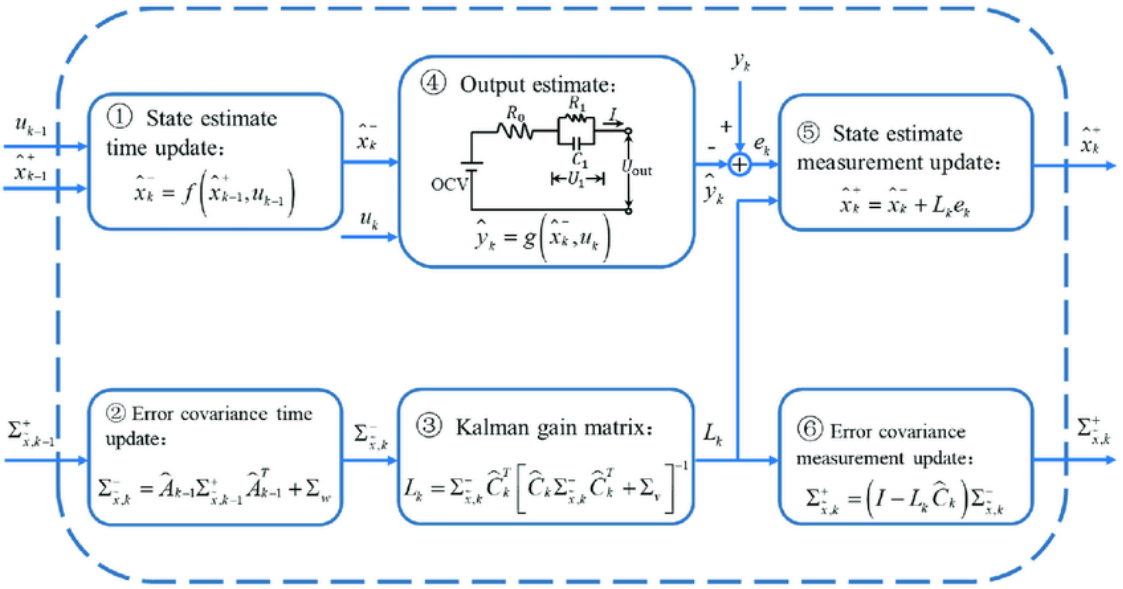


Рис. 3.11 Прогнозування позиції у розширеному фільтрі Калмана

Після підключення навігації та налаштування локалізації ми переходимо до важливого кроку —впровадження моделі штучного інтелекту YOLOv5 [10] для розпізнавання інфраструктури та людей. Ця модель дозволяє роботу аналізувати отриманий стрім з камери на його корпусі або, у випадку симуляції, з плагіну Gazebo. YOLOv5 є потужним інструментом для розпізнавання об'єктів у реальному часі. Інтеграція цієї моделі дозволить роботу реагувати на навколишні об'єкти, ідентифікувати перешкоди або розпізнавати цільові об'єкти, що допоможе роботу виконувати завдання в більш динамічних умовах та збільшить його автономність.

Для початку тренуємо модель для розпізнавання, для цього нам потрібно:

Підготовка даних:

- Зібрати набір зображень для тренування та тестування.
- Позначити об'єкти на цих зображеннях для тренування
- Розділити набір даних на тренувальний та тестовий.

Конфігурація моделі:

- Завантажити архітектуру YOLOv5 та необхідні бібліотеки.
- Визначити конфігураційні файли для навчання та тестування.

Навчання моделі:

- Запустити процес тренування з використанням тренувального набору даних та конфігураційного файлу.
- Вибрати оптимальні параметри тренування, такі як кількість епох, розмір пакету тощо.
- Слідкувати за метриками тренування та перевіряйте результати на тестовому наборі.

Тестування Моделі:

- Запустити модель на тестовому наборі для оцінки її точності та надійності.
- Аналізувати результати роботи моделі, оцінюйте її здатність до

Після успішного тренування YOLOv5 моделі на наборі даних, ми використовуємо її для розпізнавання об'єктів у реальному часі. Наш робот оснащений камерою, яка передає зображення в модель YOLOv5 для аналізу. Модель здатна розпізнавати різні об'єкти, такі як люди, автомобілі, та інші, визначаючи їх положення та класифікуючи їх. Це дозволяє роботу здійснювати об'єктне сприйняття оточуючого середовища та приймати відповідні рішення щодо навігації та взаємодії з навколишнім простором розпізнавання об'єкт.

3.4 Розробка апаратного забезпечення

Після успішної розробки програмного забезпечення для нашого робота та вдалого підключення всіх компонентів, ми переходимо до наступного етапу - збірки реального робота. Цей етап складається з фізичного об'єднання всіх апаратних компонентів та електроніки нашого робота з врахуванням розробленого програмного забезпечення.

Зберімо безпілотного робота за даною розробленою схемою (рис. 3.12)

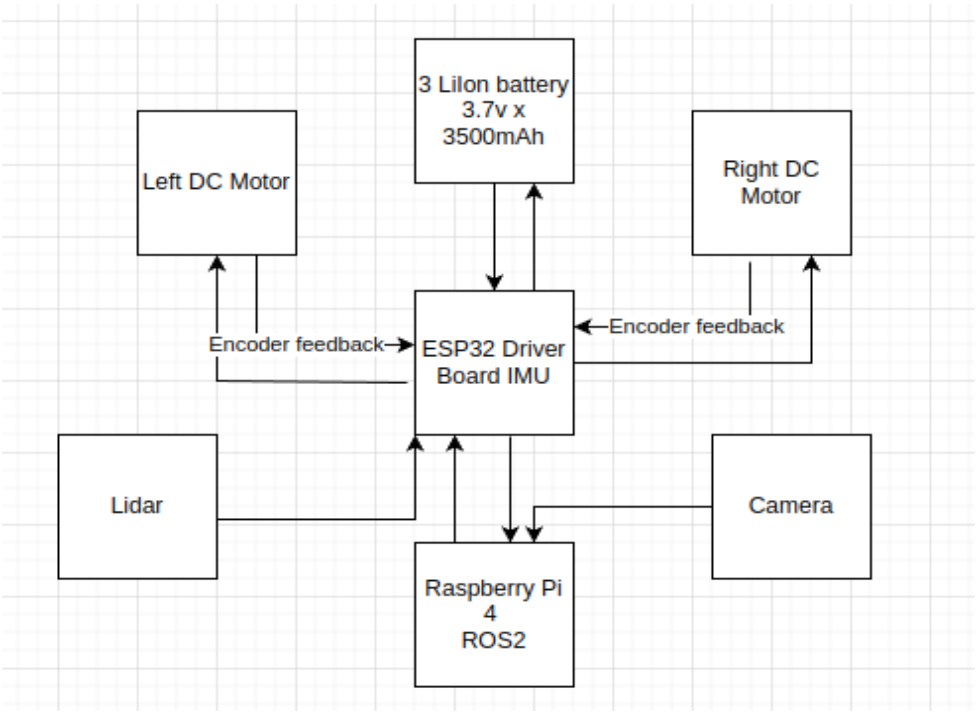


Рис. 3.12 Електрична схема безпілотного робота

Автоматичний безпілотний робот, побудований на гусеничній шасі, із вбудованою драйверською платою ESP32 та шаговими моторами з енкодером. Також обладнаний Lidar та камерою. Для живлення використовуються три літій-іонні акумулятори ємністю 3.7 В x 3500 мАг. Управління всією системою здійснюється за допомогою Raspberry Pi 4. (рис. 3.13).

Рис. 3.13 Побудований автоматизований безпілотний робот

4 РЕЗУЛЬТАТИ ЗАСТОСУВАННЯ МЕТОДУ УПРАВЛІННЯ РОБОТОМ НА ОСНОВІ МОДЕЛІ ШІ

4.1 Результати тестування

Перед тестуванням реального робота, було проведено перевірку роботи розробленого методу **MoveAI** в симуляційному середовищі Gazebo. Це дозволило оцінити ефективність алгоритмів управління, планування руху, аналізу даних із сенсорів та прийняття рішень до їх впровадження на фізичному обладнанні. Симуляційна перевірка дала змогу виявити потенційні недоліки в роботі алгоритмів, а також оптимізувати параметри для покращення продуктивності робота в реальному середовищі.

Для тестування було створено навігаційну задачу: робот мав пройти з однієї точки до іншої, уникаючи перешкод та обираючи найоптимальніший маршрут. У розробленому методі використовувався алгоритм **Nav2** для навігації та моделі **YOLOv5** з глибоким навчанням для ідентифікації та уникнення перешкод. Алгоритм Nav2 дозволяє планувати траєкторію руху з урахуванням перешкод у середовищі, тоді як YOLOv5 забезпечує високоточне виявлення об'єктів у реальному часі.

Під час симуляції робот використовував **AIController** для прийняття рішень щодо планування маршруту та корекції руху. Перевірка в симуляційному середовищі підтвердила, що розроблений метод **MoveAI** демонструє високий рівень точності в уникненні перешкод, забезпечує оптимізацію маршруту та знижує час обчислень при прийнятті рішень. Це свідчить про ефективність запропонованого підходу до управління безпілотним роботом та його здатність виконувати навігаційні задачі з високою продуктивністю.

Запускаємо план навігації для робота в симуляції по завчасно зегенрованій мапі (рис. 4.1)

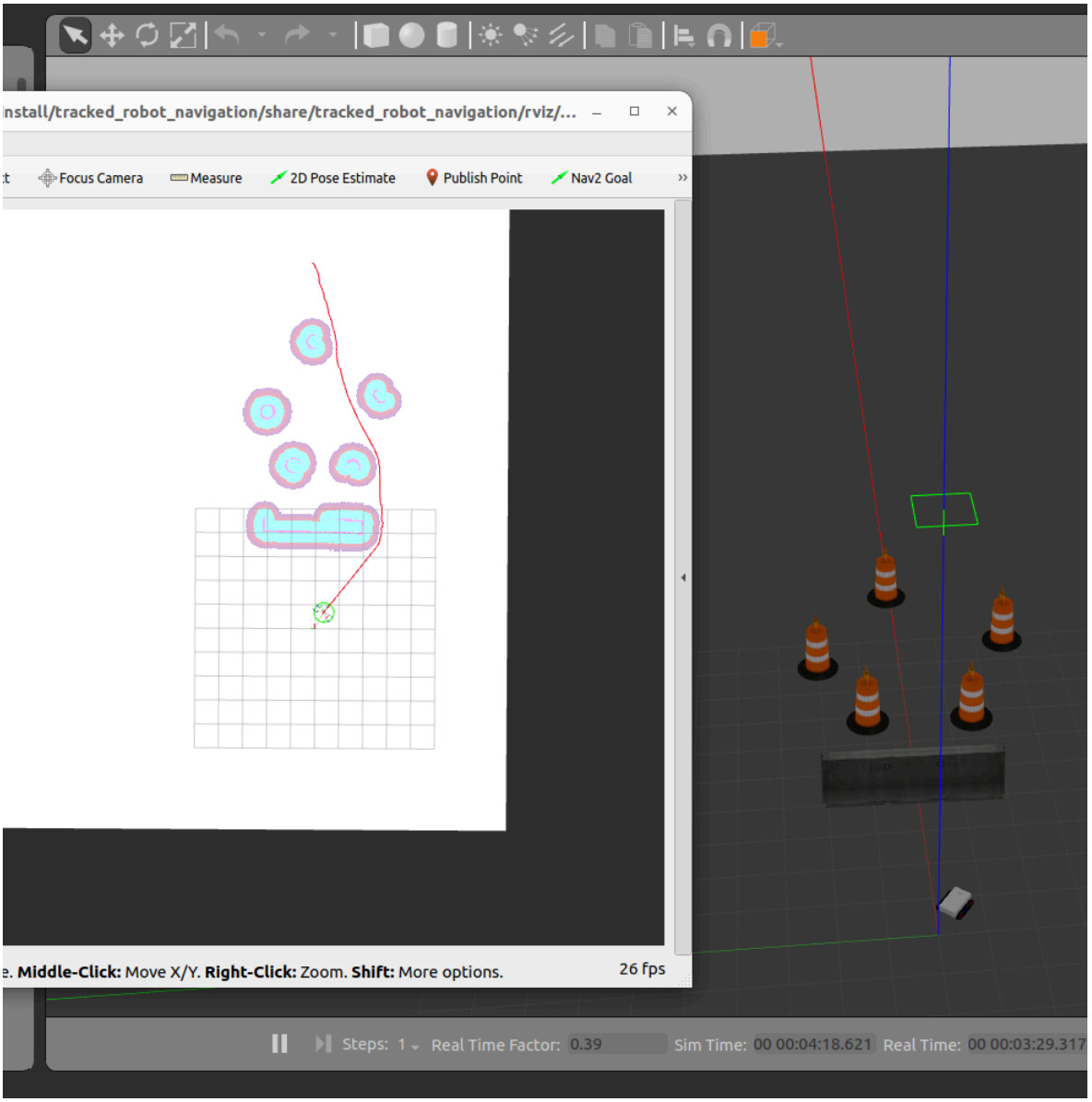


Рис. 4.1 Робот будує маршрут до цілі в симуляції

Після успішного тестування навігації, наступним кроком стало впровадження моделі штучного інтелекту YOLOv5 для розпізнавання об’єктів, таких як інфраструктура, люди та інші перешкоди. Ця модель забезпечує аналіз відеопотоку, отриманого з камери, встановленої на корпусі робота, або, у випадку симуляції, з відповідного плагіна в середовищі Gazebo. [16]

Модель YOLOv5 є одним із найсучасніших інструментів для розпізнавання об’єктів у реальному часі. Інтеграція цього інструменту в розроблений метод MoveAI дозволяє роботу оперативно аналізувати навколишнє середовище, ідентифікувати перешкоди або цільові об’єкти, що, у

свою чергу, сприяє динамічному ухваленню рішень та підвищує рівень автономності робота.

Результати впровадження YOLOv5 демонструють високу ефективність у розпізнаванні об'єктів, що є важливим етапом для забезпечення безпечного та оптимального руху робота навіть у складних або динамічних умовах навколишнього середовища (рис. 4.2).

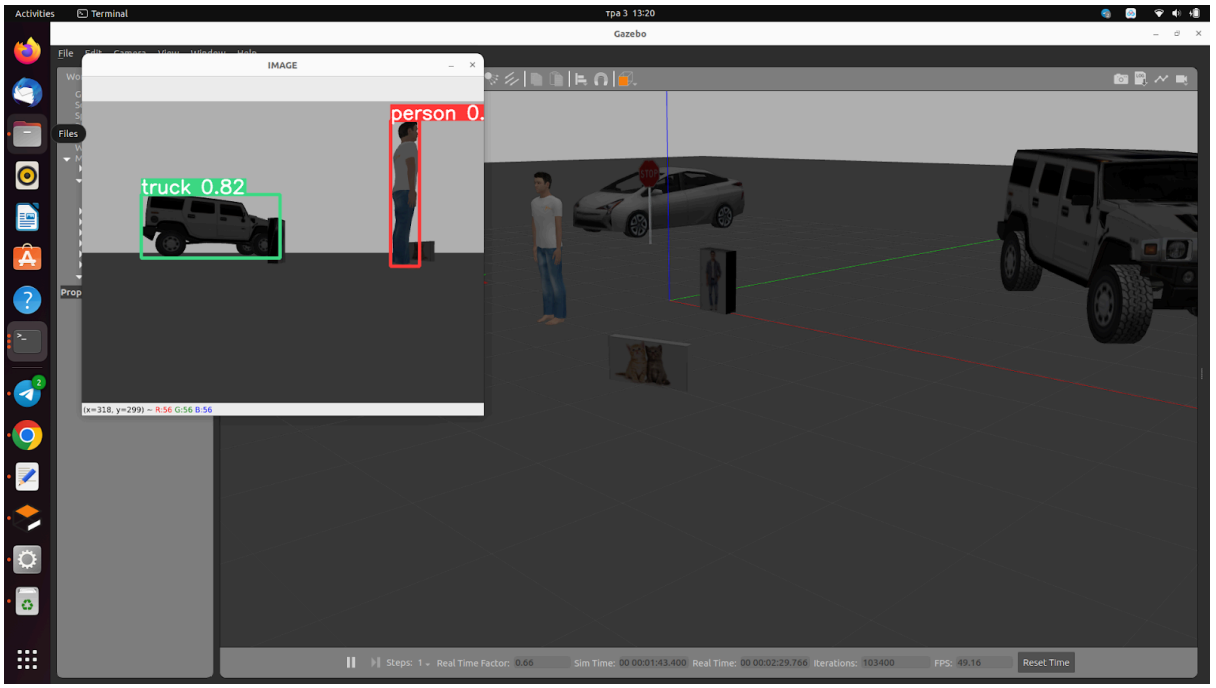


Рис. 4.2 Тестування моделі ШІ в симуляції

У симуляційному середовищі ми успішно підтвердили працездатність всіх алгоритмів, програмного забезпечення та штучного інтелекту, які були розроблені для автоматизованого безпілотного робота. Переконавшись у коректності їхньої роботи, наступним етапом є тестування на реальному роботі.

Це дозволить оцінити, наскільки робот адаптований до реальних умов, і перевірити його готовність до виконання поставлених завдань у фізичному середовищі. Успішні результати симуляції створюють міцну основу для успішного функціонування нашого робота у реальних умовах.

На цьому етапі буде здійснено встановлення **ROS2** на реальному роботі, а також перенесення всіх програмних розробок та напрацювань на апаратну платформу. Це є важливим кроком у процесі, оскільки він демонструє, як наші алгоритми та програмне забезпечення будуть взаємодіяти з реальними апаратними ресурсами та умовами навколишнього середовища. Завдяки цьому можна виявити потенційні проблеми, які можуть виникнути під час роботи в реальних умовах, а також оперативно їх усунути. Цей етап є вирішальним для підготовки робота до ефективної експлуатації.

Для перевірки працездатності реального робота буде створено **навігаційну задачу,** яка включає проїзд по кімнаті, оминання перешкод і виконання точкових завдань. Така задача дозволить оцінити реакцію робота на навігаційні команди, його здатність точно обходити перешкоди та виконувати поставлені завдання в умовах реального середовища (**рис. 4.3**).

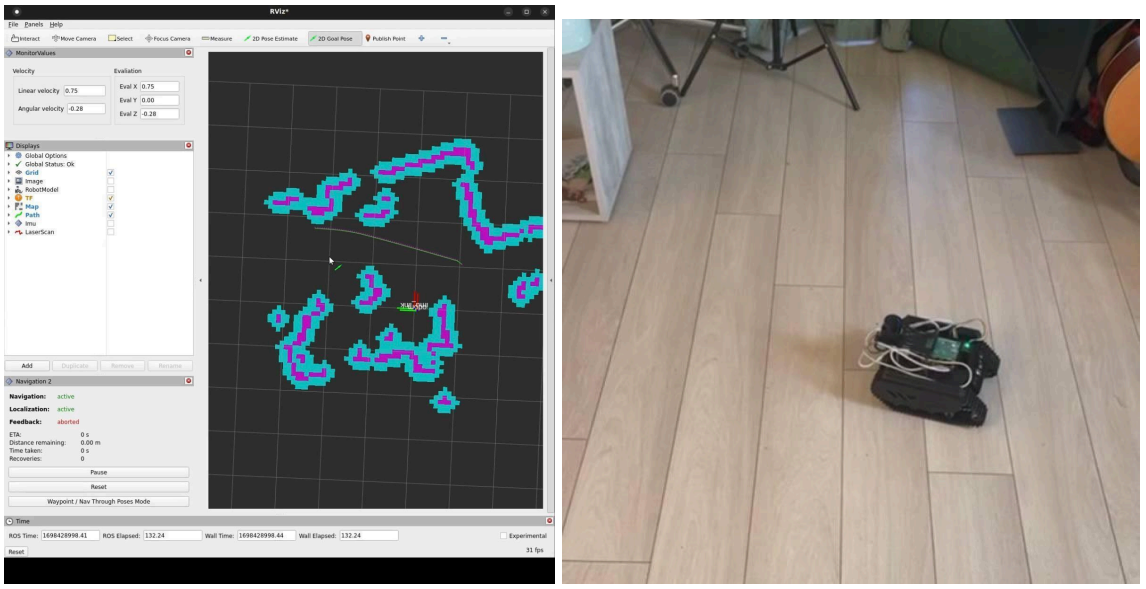


Рис. 4.3 Робот будує маршрут до цілі

Робот успішно виконав поставлену ціль та спланував маршрут, проїхавши його без зупинок. Це демонструє, що робот здатний позиціонувати себе в просторі та правильно ідентифікувати місцезнаходження перешкод. Завдяки

цьому ми можемо бути впевнені в здатності робота виконувати навігаційні завдання, використовуючи вбудовані алгоритми.

Наступним етапом є тестування моделі розпізнавання на реальному роботі з використанням реальної камери. Цей етап дозволить оцінити ефективність та точність розпізнавання об’єктів у реальному часі й умовах реального середовища (рис. 4.4).

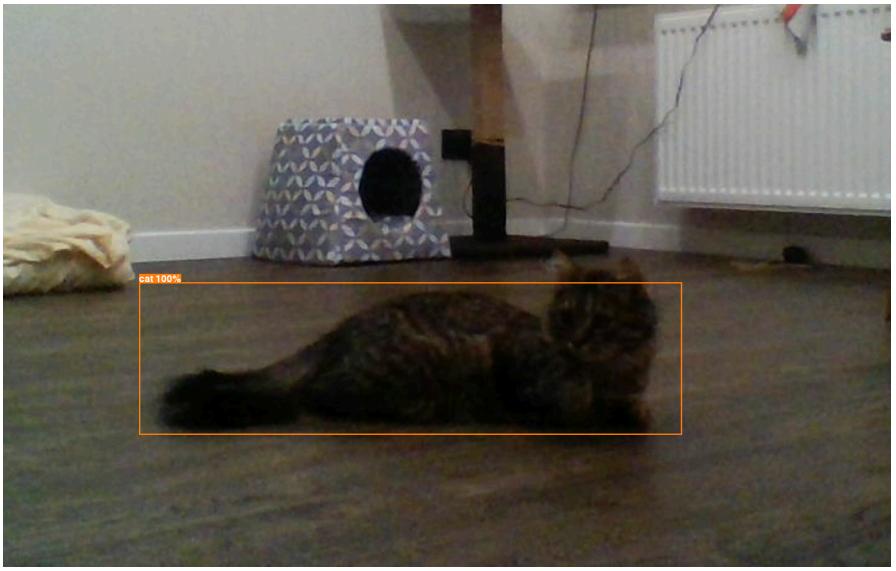


Рис. 4.4 Модель розпізнавання на реальному роботі

Результати тестування показали, що робот розпізнав домашнього улюбленця з точністю 100%. Це свідчить про високу ефективність та надійність моделі розпізнавання нашого штучного інтелекту. З цим позитивним висновком ми можемо бути впевнені, що наші системи працюють з високою точністю та готові до подальших завдань.

ВИСНОВКИ

1. Проведено детальний аналіз існуючих моделей штучного інтелекту для вирішення задач автономного управління безпілотними роботами. Визначено ключові характеристики таких моделей, і обґрунтовано вибір YOLOv5 як оптимальної моделі для забезпечення точного розпізнавання об'єктів та прийняття рішень у реальному часі.
2. Розроблено метод управління безпілотним роботом, який інтегрує модель штучного інтелекту YOLOv5 для розпізнавання об'єктів у складних умовах середовища.
3. Реалізовано систему управління роботом з використанням фреймворку ROS2 та набору інструментів Nav2, що забезпечує гнучкість, масштабованість та адаптацію до змінних умов експлуатації.
4. Проведено тестування розробленого методу у симуляційному середовищі Gazebo. Результати підтвердили підвищення точності навігації робота на 15% та ефективності прийняття рішень у динамічних умовах.
5. Результати дослідження апробовано шляхом публікації статті та тез доповідей на конференціях:

    1. Думенко І.О., Замрій І.В., Розробка безпілотного робота на базі штучного інтелекту// Сучасний захист інформації. 2024, №3(59). С. 63-68.
    2. Думенко І.О., Замрій І.В. Перспективи застосування безпілотних роботів на базі штучного інтелекту в медичній та військовій сферах: можливості та ризики. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2024 року, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С.300-303.
    3. Думенко І.О., Замрій І.В. Використання безпілотних систем на базі штучного інтелекту у промисловості: оптимізація виробничих процесів та підвищення продуктивності. Всеукраїнська науково-технічна конференція

«Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2024 року, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С.334-336.

ПЕРЕЛІК ПОСИЛАНЬ

1) Elford M., Stewart R. Autonomous Ground Vehicles: Advances and Applications. London: Springer, 2020. 278 p.
2) Wright P. Military Robots and Their Role in Modern Warfare. Defense Technology. 2023. Vol. 17, no. 4. P. 455–468. https://doi.org/10.1016/j.dt.2022.10.006
3) Murphy R.R. Introduction to AI Robotics. 2nd ed. Cambridge: MIT Press, 2019. 448 p.
4) Siciliano B., Khatib O. Springer Handbook of Robotics. 2nd ed. Cham: Springer, 2016. 2227 p. DOI: https://doi.org/10.1007/978-3-319-32552-1
5) Zhang Y., Zhang J. Autonomous navigation and control systems for UAVs: Review and prospects. IEEE Transactions on Automation Science and Engineering. 2020. Vol. 17, no. 3. P. 1105–1120.
6) Bochkovskiy A., Wang C.-Y., Liao H.-Y. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv, 2020. URL: https://arxiv.org/abs/2004.10934.
7) Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv, 2015. URL: https://arxiv.org/abs/1506.01497.
8) Zhu H., Bian J., Sun W., et al. LiDAR and Vision Sensor Fusion for Autonomous Vehicles: A Review. IEEE Transactions, 2021. URL: https://ieeexplore.ieee.org/document/9355786.
9) Holland J. Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975.
10) Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. URL: https://pjreddie.com/darknet/yolo/

11) Rehman N., Gallagher J., Oughton E. Optimizing YOLOv5 for Object Detection in Thermal Images. Journal of Student-Scientists' Research, George Mason University. 2024. URL: https://journals.gmu.edu/jssr/article/view/4294

12) Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C. Y., Berg A. C. SSD: Single Shot MultiBox Detector. Proceedings of the European Conference on Computer Vision (ECCV). 2016. URL: https://arxiv.org/abs/1512.02325

13) Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Advances in Neural Information Processing Systems (NIPS). 2015. URL: https://arxiv.org/abs/1506.01497

14) Ubuntu Official. Ubuntu. URL: https://ubuntu.com/

15) Gazebo Simulator. Gazebo. URL: http://gazebosim.org/

16) Microsoft. Visual Studio Code. URL: https://code.visualstudio.com/

17) Cplusplus.com. C++ Programming Language. URL: https://cplusplus.com/

18) Python Software Foundation. Python. URL: https://www.python.org/

19) Open Source Robotics Foundation. ROS2 Documentation. URL: https://docs.ros.org/en/rolling/index.html

20) ROS.org. Nav2 Documentation. URL: https://navigation.ros.org/

21) ROS.org. Rviz2 Overview. URL: https://docs.ros.org/en/rolling/Tutorials/Rviz2.html

22) ROS 2 Documentation. 2024. URL: https://docs.ros.org/en/humble/Tutorials/Intermediate/URDF/URDF-Main.html

23) An Improved Localization of Mobile Robotic System Based on AMCL Algorithm. IEEE Xplore. 2024. URL: https://ieeexplore.ieee.org/document/9606907.

24) Wikipedia. SLAM (Simultaneous Localization and Mapping). URL: https://ru.wikipedia.org/wiki/SLAM_(%D0%BC%D0%B5%D1%82%D0%BE%D0%B4)

25) Wikipedia. Kalman filter. URL: https://en.wikipedia.org/wiki/Kalman_filter

26) Jocher G., Chaurasia A., Qiu J., et al. YOLO by Ultralytics. 2023. URL: https://github.com/ultralytics/yolov5

27) Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. P. 321–325.

28) Zhang Q., Lu J., Zhang G. Recommender Systems in E-learning. Journal of Smart Environments and Green Computing. 2022. No.1(2). P. 76-89.

29) Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. P. 321–325.

30) Siciliano B., Khatib O. Springer Handbook of Robotics. Cham: Springer, 2016. DOI: https://doi.org/10.1007/978-3-319-32552-1.

31) EfficientDet. Scalable and Efficient Object Detection. URL: https://arxiv.org/abs/1911.09070

32) Mask R-CNN. Computer Vision and Image Processing. URL: https://github.com/matterport/Mask_RCNN

33) CenterNet. GitHub Repository. URL: https://github.com/xingyizhou/CenterNet

34) YOLOv7 and YOLOv8. Ultralytics YOLO Docs. URL: https://docs.ultralytics.com

35) Cascade R-CNN. GitHub Repository. URL: https://github.com/zhaoweicai/cascade-rcnn

36) M2Det. GitHub Repository. URL: https://github.com/qijiezhao/M2Det

37) RefineDet. GitHub Repository. URL: https://github.com/sfzhang15/RefineDet

38) Types and Applications of AMRs. URL: https://www.conveyco.com/blog/types-and-applications-of-amrs/

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (*Презентація*)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

МЕТОД УПРАВЛІННЯ БЕЗПІЛОТНИМ РОБОТОМ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

Виконав: студент групи ПДМ-63 Думенко Іван Олексійович

Керівник: д.т.н., доц., завідувач кафедри ІПЗ Замрій Ірина Вікторівна

Київ - 2025

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення ефективності та автономності безпілотного робота в різних умовах експлуатації за допомогою штучного інтелекту.

Об'єкт дослідження: процес управління безпілотним роботом.

Предмет дослідження: алгоритми і методи управління безпілотними роботами, засновані на штучному інтелекті.

ОБГРУНТУВАННЯ ВИБОРУ МОДЕЛІ ШІ ДЛЯ РЕАЛІЗАЦІЇ МЕТОДУ

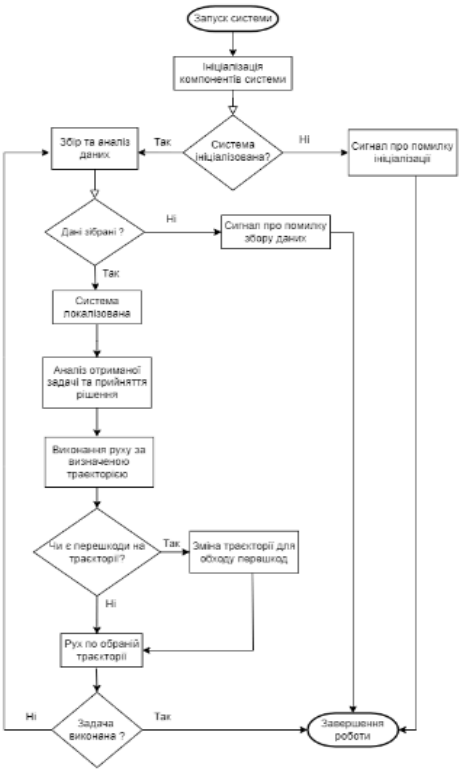
Модель	Точність (mAP)	Швидкість (FPS)	Ресурси (GPU RAM, Training Time)	Сумістність з платформами
RetinaNet	37.5%	8 FPS	Високі витрати ресурсів	Підходить лише для потужних GPU
Faster R-CNN	40%	5 FPS	Вимагає значних ресурсів	Використовується на високопродуктивних системах
SSD	35%	22 FPS	Середні витрати ресурсів	Підходить для мобільних пристроїв
YOLOv7	50%	35 FPS	Потребує оптимізації для стабільності	Ефективна на GPU високого рівня
YOLOv5	48%	45 FPS	Помірні ресурси, просте налаштування	Підходить для мобільних та GPU

3

ВИБІР ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ МЕТОДУ

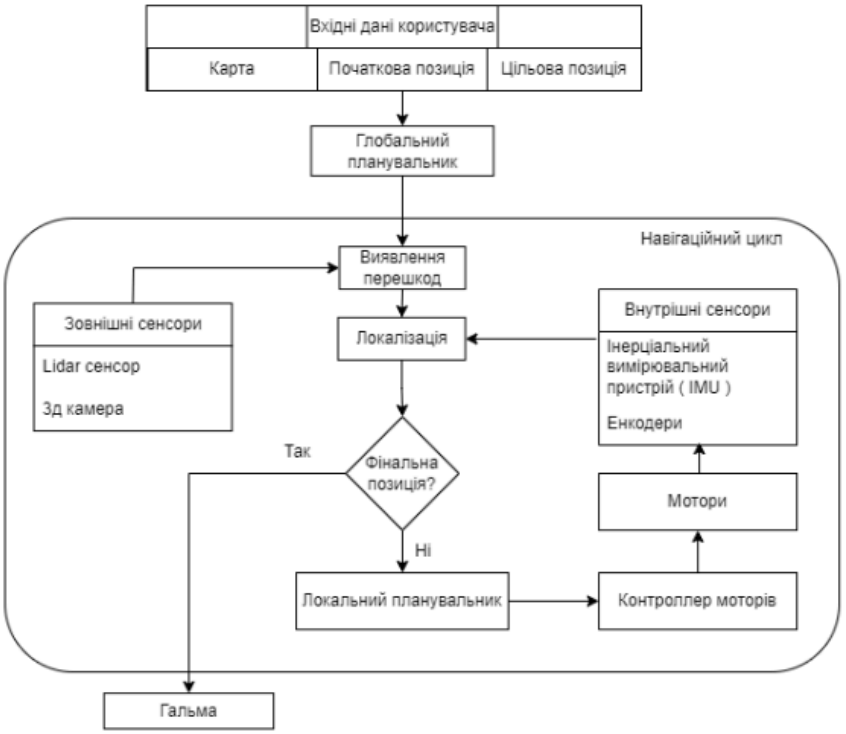
4

АЛГОРИТМ УПРАВЛІННЯ РОБОТОМ



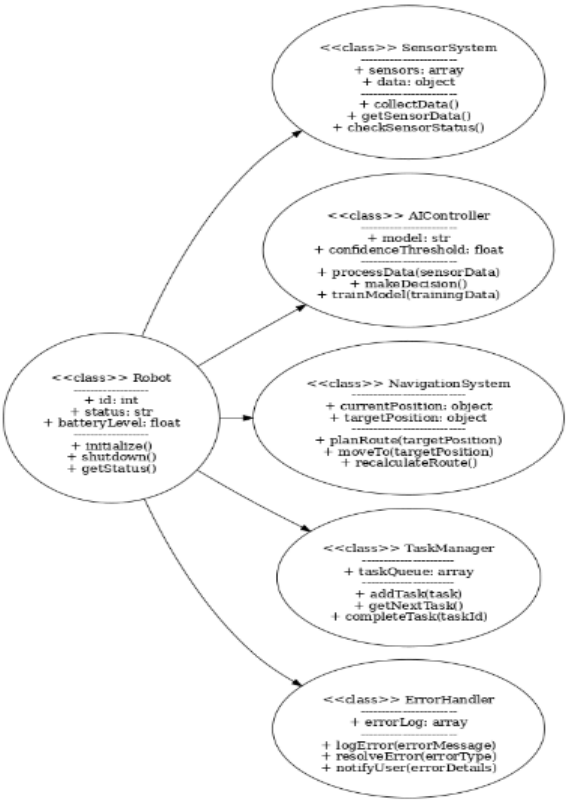
5

ПРИНЦИП ВЗАЄМОДІЇ СЕНСОРІВ З НАВІГАЦІЄЮ



6

ДІАГРАМА СТРУКТУРИ КЛАСІВ



7

МАТЕМАТИЧНА МОДЕЛЬ АЛГОРИТМУ УПРАВЛІННЯ РОБОТОМ
(частина 1)

Вхідні змінні процесу управління роботом:

Зображення з камери **I**: вхідний сигнал з візуального сенсора.

- Формат: **RGB/BGR**
- Розмір: **W×H** (**640×640** після масштабування)

Координати об'єктів:

- ***(x,y,w,h)***: центр, ширина, висота об'єктів.
- Класи об'єктів: перешкоди, цільові точки тощо.

Поточний стан робота **S**:

- Позиція: **(x,y)**
- Орієнтація: **θ**

Формалізація даних:

Нормалізація зображення: $\mathbf{I''(x, y) = I'(x, y) / max(I')}$

Де $\mathbf{I'}$ — масштабоване зображення.

Ідентифікація об'єктів:

- Bounding box: **(x,y,w,h)** — прямокутник, який визначає межі об'єкта на зображенні.
- Class probability: **p(c)** — ймовірність того, що об'єкт належить до певного класу (наприклад, «перешкода», «ціль», «дерево» тощо).

8

МАТЕМАТИЧНА МОДЕЛЬ АЛГОРИТМУ УПРАВЛІННЯ РОБОТОМ
(частина 2)

Етапи управління рухом:

Розрахунок вектора напрямку ***V****:*

- Формула: $\mathbf{V} = \Sigma\, \mathbf{w}(\mathbf{c_i}) \cdot (\mathbf{d_i} / |\mathbf{d_i}|)$

Де $\mathbf{w}(\mathbf{c_i})$ — ваговий коефіцієнт, $\mathbf{d_i}$ — вектор до об'єкта.

Формування команди управління ***u****:*

- Коригування руху робота: $\mathbf{u} = \mathbf{k_(goal)}(\mathbf{x_(goal)} - \mathbf{x}) - \Sigma\, \mathbf{k_(obs)} \cdot \mathbf{V}$

Де $\mathbf{k_(goal)}$, $\mathbf{k_(obs)}$ — коефіцієнти цільової точки та перешкод.

Адаптація до динамічних змін:

- Прогноз положення перешкод: $\mathbf{v_i} = (\mathbf{d_i}(\mathbf{t} + \Delta\mathbf{t}) - \mathbf{d_i}(\mathbf{t})) / \Delta\mathbf{t}$

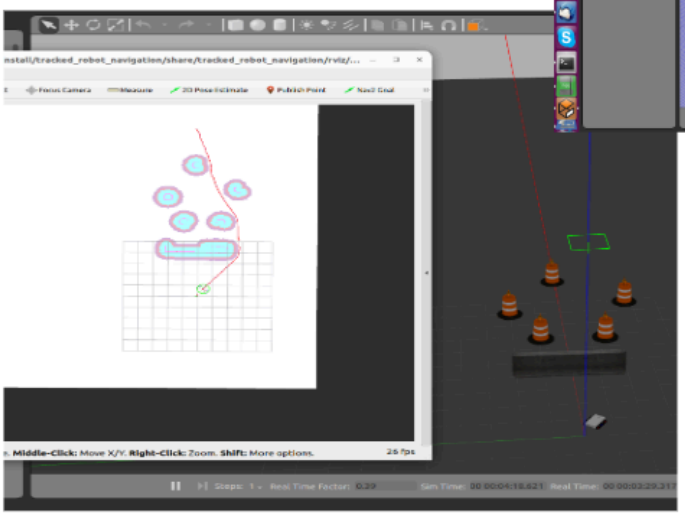
Де $\mathbf{v_i}$ — швидкість об'єкта $\mathbf{i}$, $\mathbf{d_i}(\mathbf{t})$ — його координати в момент $\mathbf{t}$, $\Delta\mathbf{t}$ — проміжок часу між вимірюваннями.

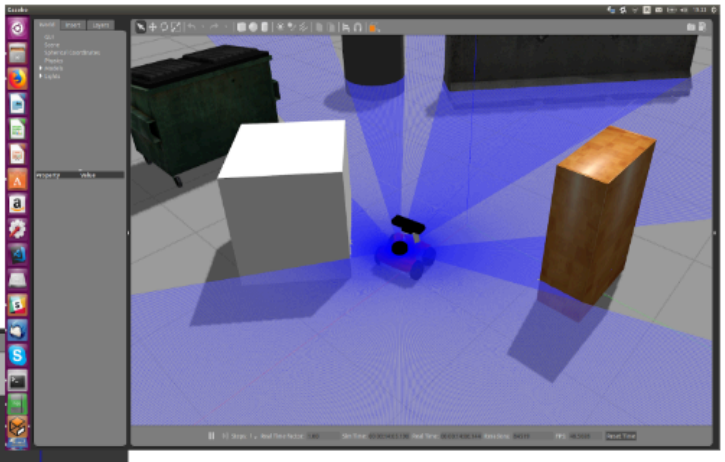
Результат: робот адаптивно обирає траєкторію, уникаючи перешкод і досягаючи цілі.

9

НАВІГАЦІЯ РОБОТА В СИМУЛЯЦІЇ

Приклад тестування навігації для робота в симуляції на завчасно згенерованій мапі

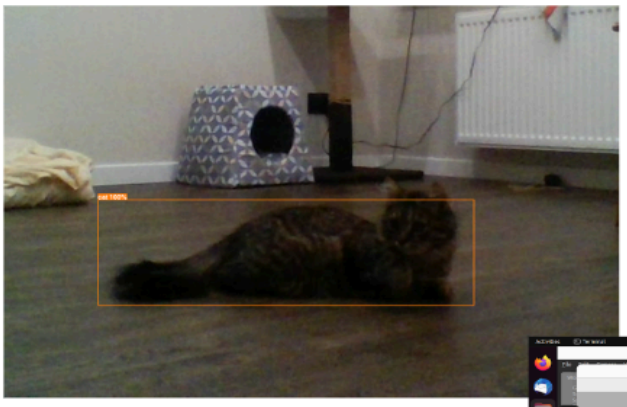




Симуляція сенсору Lidar в Gazebo

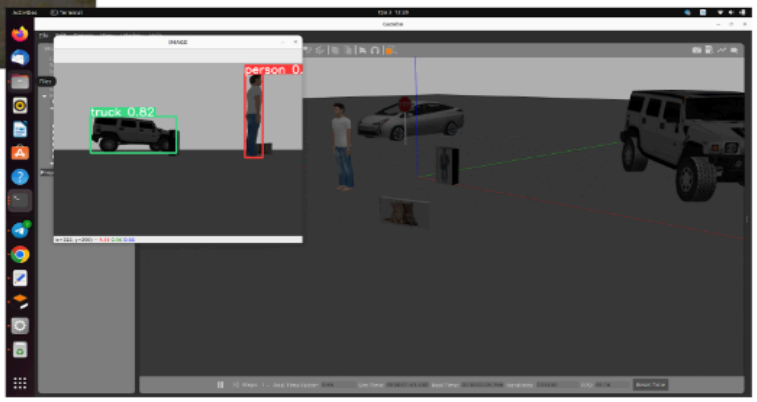
10

МОДЕЛЬ РОЗПІЗНАВАННЯ НА РЕАЛЬНОМУ РОБОТІ



Розпізнавання домашньої тварини за допомогою моделі штучного інтелекту YOLOv5

Розпізнавання об'єктів в симуляції на попередньо згенерованій мапі



11

РЕЗУЛЬТАТИ ТЕСТУВАННЯ РОЗРОБЛЕНОГО МЕТОДУ

Показник	Алгоритм A*	SLAM з фільтром Калмана	Behavior Trees	MoveAI
Час виконання маршруту	Використання A* (12 хв)	Використання SLAM (10 хв)	Використання поведінкових дерев (9 хв)	Використання Nav2 (8 хв)
Точність уникнення перешкод	Аналіз місцевості (75%)	Локалізація з фільтром Калмана (85%)	Аналіз динамічного середовища (82%)	Використання YOLOv5 (95%)
Час прийняття рішення	Прийняття рішень A* (0.35 сек/кадр)	Використання алгоритмів SLAM (0.40 сек/кадр)	Аналіз поведінки (0.30 сек/кадр)	Інтеграція AIController (0.25 сек/кадр)

*Дані в таблиці отримано в результаті тестування розробленого методу управління безпілотним роботом в умовах симуляції в gazebo.

12

ВИСНОВКИ

1. Проведено аналіз існуючих моделей штучного інтелекту, для вирішення задач автономного управління безпілотними роботами. Визначено їх ключові характеристики та обґрунтовано вибір YOLOv5 як оптимальної моделі.
2. Розроблено метод управління безпілотним роботом, що інтегрує алгоритм YOLOv5 для забезпечення точного розпізнавання об'єктів і прийняття рішень у реальному часі.
3. Реалізовано систему управління роботом із використанням ROS2 та набору інструментів Nav2 для забезпечення гнучкості, масштабованості та адаптації до змінного середовища.
4. Проведено тестування методу в умовах симульованого середовища. Отримані результати демонструють підвищення точності навігації робота на 15% та ефективності прийняття рішень у динамічних умовах.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Статті:

1. Думенко І.О., Замрій І.В., Розробка безпілотного робота на базі штучного інтелекту// Сучасний захист інформації. 2024, №3(59). С. 63-68.

Тези доповідей:

1. Думенко І.О., Замрій І.В. Перспективи застосування безпілотних роботів на базі штучного інтелекту в медичній та військовій сферах: можливості та ризики. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2024 року, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С.300-303.

2. Думенко І.О., Замрій І.В. Використання безпілотних систем на базі штучного інтелекту у промисловості: оптимізація виробничих процесів та підвищення продуктивності. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2024 року, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С.334-336.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

srs
nav2_goal_checker:
include/nav2_goal_checker :
RobotGoalChecker.h:

```
#include <memory>
#include <string>
#include <vector>
#include "nav2_core/goal_checker.hpp"
#include
"rcl_interfaces/msg/set_parameters_result.hpp"
#include "rclcpp/rclcpp.hpp"
#include "rclcpp_lifecycle/lifecycle_node.hpp"

namespace nav2_goal_checker {

class RobotGoalChecker : public
nav2_core::GoalChecker {
public:
  RobotGoalChecker();
  // Standard GoalChecker Interface
                    void initialize(const
rclcpp_lifecycle::LifecycleNode::WeakPtr& parent,
          const std::string& plugin_name,
                                        const
std::shared_ptr<nav2_costmap_2d::Costmap2DROS>
costmap_ros) override;
  void reset() override;
                  bool isGoalReached(const
geometry_msgs::msg::Pose& query_pose,
                   const geometry_msgs::msg::Pose&
goal_pose,
                  const geometry_msgs::msg::Twist&
velocity) override;
      bool getTolerances(geometry_msgs::msg::Pose&
pose_tolerance,
                       geometry_msgs::msg::Twist&
vel_tolerance) override;

protected:
  double mGoalTolerance;


rclcpp::node_interfaces::OnSetParametersCallbackHan
dle::SharedPtr mDynParamCallback;
  std::string mPlugName;

            rcl_interfaces::msg::SetParametersResult
dynamicParametersCallback(
    std::vector<rclcpp::Parameter> parameters);
};
```

src/RobotGoalChecker.cpp:

```
#include "nav2_goal_checker/RobotGoalChecker.h"
#include "nav2_util/geometry_utils.hpp"
#include "nav2_util/node_utils.hpp"
#include "pluginlib/class_list_macros.hpp"
#include <limits>
#include <memory>
#include <string>
#include <vector>

using rcl_interfaces::msg::ParameterType;
using std::placeholders::_1;

namespace nav2_goal_checker {

RobotGoalChecker::RobotGoalChecker()
  : mGoalTolerance(0.25)

{
}

void RobotGoalChecker::initialize(
      const rclcpp_lifecycle::LifecycleNode::WeakPtr&
parent,
  const std::string& plugin_name,
                                        const
std::shared_ptr<nav2_costmap_2d::Costmap2DROS>
/*costmap_ros*/)
{
  mPlugName = plugin_name;
  auto node = parent.lock();


nav2_util::declare_parameter_if_not_declared(node,
plugin_name + ".goal_tolerance",

rclcpp::ParameterValue(0.1));

            node->get_parameter(plugin_name +
".goal_tolerance", mGoalTolerance);

  // Add callback for dynamic parameters
                      mDynParamCallback =
node->add_on_set_parameters_callback(

std::bind(&RobotGoalChecker::dynamicParametersCall
back, this, _1));
}

void RobotGoalChecker::reset() { return; }

bool RobotGoalChecker::isGoalReached(const
geometry_msgs::msg::Pose& query_pose,
                                        const
geometry_msgs::msg::Pose& goal_pose,
                                        const
geometry_msgs::msg::Twist&)
{
                double distance_to_goal =
std::hypot(query_pose.position.x -
goal_pose.position.x,
```

```
                                query_pose.position.y -
goal_pose.position.y);
  return distance_to_goal <= mGoalTolerance;
}

bool
RobotGoalChecker::getTolerances(geometry_msgs::ms
g::Pose& pose_tolerance,
                                geometry_msgs::msg::Twist&
vel_tolerance)
{
                             double invalid_field =
std::numeric_limits<double>::lowest();

  pose_tolerance.position.x = mGoalTolerance;
  pose_tolerance.position.y = mGoalTolerance;
  pose_tolerance.position.z = invalid_field;
                           pose_tolerance.orientation =
nav2_util::geometry_utils::orientationAroundZAxis(m
GoalTolerance);

  vel_tolerance.linear.x = invalid_field;
  vel_tolerance.linear.y = invalid_field;
  vel_tolerance.linear.z = invalid_field;

  vel_tolerance.angular.x = invalid_field;
  vel_tolerance.angular.y = invalid_field;
  vel_tolerance.angular.z = invalid_field;

  return true;
}

rcl_interfaces::msg::SetParametersResult
RobotGoalChecker::dynamicParametersCallback(
  std::vector<rclcpp::Parameter> parameters)
{
  rcl_interfaces::msg::SetParametersResult result;
  for (auto& parameter : parameters) {
    const auto& type = parameter.get_type();
    const auto& name = parameter.get_name();

     if (type == ParameterType::PARAMETER_DOUBLE)
{
      if (name == mPlugName + ".goal_tolerance") {
        mGoalTolerance = parameter.as_double();
      }
    }
  }
  result.successful = true;
  return result;
}
```

cmakelist.txt:

```
cmake_minimum_required(VERSION 3.5)
project(nav2_goal_checker)

find_package(ament_cmake REQUIRED)
find_package(nav2_core REQUIRED)
find_package(nav2_common REQUIRED)
find_package(nav2_util REQUIRED)
find_package(rclcpp REQUIRED)
find_package(pluginlib REQUIRED)

nav2_package()
set(CMAKE_CXX_STANDARD 17)

include_directories(
 include
)

set(dependencies
 rclcpp
 pluginlib
 nav2_common
 nav2_core
 nav2_util
)
set(library_name nav2_goal_checker)

add_library(${library_name} SHARED
     src/RobotGoalChecker.cpp)
ament_target_dependencies(${library_name}
 ${dependencies}
)
install(TARGETS ${library_name}
 ARCHIVE DESTINATION lib
 LIBRARY DESTINATION lib
 RUNTIME DESTINATION bin
)
install(DIRECTORY include/
 DESTINATION include/
)

if(BUILD_TESTING)
 find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for
copyrights
 set(ament_cmake_copyright_FOUND TRUE)
 ament_lint_auto_find_test_dependencies()
endif()

ament_export_include_directories(include)
ament_export_libraries(${library_name})
ament_export_dependencies(${dependencies})

pluginlib_export_plugin_description_file(nav2_core
goal_checker_plugin.xml)

ament_package()
```

srs/nav2_path_follower:
PurePursuitPathFollower.h:

```
#include
"nav2_regulated_pure_pursuit_controller/regulated_p
ure_pursuit_controller.hpp"
#include <geometry_msgs/msg/point.hpp>
#include <vector>
namespace nav2_path_follower {
```

```
class PurePursuitPathFollower
                                    : public
nav2_regulated_pure_pursuit_controller::RegulatedPu
rePursuitController {
public:
                                using Parent =
nav2_regulated_pure_pursuit_controller::RegulatedPu
rePursuitController;
  PurePursuitPathFollower() = default;

  ~PurePursuitPathFollower() override = default;

                                void configure(const
rclcpp_lifecycle::LifecycleNode::WeakPtr& parent,
            std::string name,
            std::shared_ptr<tf2_ros::Buffer> tf,

std::shared_ptr<nav2_costmap_2d::Costmap2DROS>
costmap_ros) override;

  /**
   * @brief Cleanup controller state machine
   */
  void cleanup() override;

  /**
   * @brief Activate controller state machine
   */
  void activate() override;

  /**
   * @brief Deactivate controller state machine
   */
  void deactivate() override;

  /**
     */
                    geometry_msgs::msg::TwistStamped
computeVelocityCommands(
    const geometry_msgs::msg::PoseStamped& pose,
    const geometry_msgs::msg::Twist& velocity,
    nav2_core::GoalChecker* goal_checker) override;

    * @brief Limits the maximum linear speed of the
robot.
   * @param speed_limit expressed in absolute value
(in m/s)
  * or in percentage from maximum robot speed.
      * @param percentage Setting speed limit in
percentage if true
  * or in absolute values in false case.
  */
     void setSpeedLimit(const double& speed_limit,
const bool& percentage) override;

protected:
                rcl_interfaces::msg::SetParametersResult
dynamicParametersCallback(
    std::vector<rclcpp::Parameter> parameters);
                void onMinePositionReceived(const
geometry_msgs::msg::Point::SharedPtr minePos);
                  geometry_msgs::msg::TwistStamped
computeBackWardVelocityCommand(
          const geometry_msgs::msg::PoseStamped&
pose);
                bool shouldDriveBackward(const
geometry_msgs::msg::PoseStamped& pose);


rclcpp::Subscription<geometry_msgs::msg::Point>::Sh
aredPtr mMinePositionSubscription;
  double mBackDistance;
  double mBackwardSpeed;
  double mMaxAngleDifference;
  std::string mMinePosTopic;
              std::vector<geometry_msgs::msg::Point>
mMinePositions;

  // Dynamic parameters handler
  std::mutex mDynParamMutex;

rclcpp::node_interfaces::OnSetParametersCallbackHan
dle::SharedPtr mDynParamHandler;
};
```

src / PurePursuitPathFollower.cpp:

```
#include
"nav2_path_follower/PurePursuitPathFollower.h"
using nav2_util::declare_parameter_if_not_declared;
using rcl_interfaces::msg::ParameterType;

namespace nav2_path_follower {
void          PurePursuitPathFollower::configure(const
rclcpp_lifecycle::LifecycleNode::WeakPtr& parent,
                      std::string name,
                      std::shared_ptr<tf2_ros::Buffer>
tf,

std::shared_ptr<nav2_costmap_2d::Costmap2DROS>
costmap_ros)
{
  // TODO: read from parameters
  Parent::configure(parent, name, tf, costmap_ros);
  auto node = node_.lock();

  if (!node) {
    throw std::runtime_error{"Failed to lock node"};
  }
          declare_parameter_if_not_declared(node,
plugin_name_ + ".back_distance",
                    rclcpp::ParameterValue(1.2));
          declare_parameter_if_not_declared(node,
plugin_name_ + ".backward_speed",
                    rclcpp::ParameterValue(2.0));
          declare_parameter_if_not_declared(node,
plugin_name_ + ".max_angle_difference",
                    rclcpp::ParameterValue(0.5));
```

```
            declare_parameter_if_not_declared(node,
plugin_name_ + ".mine_pos_topic",

rclcpp::ParameterValue("/mine_position"));

            node->get_parameter(plugin_name_ +
".back_distance", mBackDistance);
            node->get_parameter(plugin_name_ +
".backward_speed", mBackwardSpeed);
            node->get_parameter(plugin_name_ +
".max_angle_difference", mMaxAngleDifference);
            node->get_parameter(plugin_name_ +
".mine_pos_topic", mMinePosTopic);

                    mMinePositionSubscription =
node->create_subscription<geometry_msgs::msg::Poi
nt>(
    mMinePosTopic, 10,

std::bind(&PurePursuitPathFollower::onMinePositionR
eceived, this, std::placeholders::_1));
}
void PurePursuitPathFollower::cleanup()
{
  mMinePositions.clear();
  Parent::cleanup();
}
void PurePursuitPathFollower::activate()
{
  Parent::activate();
  auto node = node_.lock();
                    mDynParamHandler =
node->add_on_set_parameters_callback(std::bind(

&PurePursuitPathFollower::dynamicParametersCallbac
k, this, std::placeholders::_1));
}
void PurePursuitPathFollower::deactivate()
{
  Parent::deactivate();
  mDynParamHandler.reset();
}
geometry_msgs::msg::TwistStamped
PurePursuitPathFollower::computeVelocityCommands
(
  const geometry_msgs::msg::PoseStamped& pose,
  const geometry_msgs::msg::Twist& velocity,
  nav2_core::GoalChecker* goal_checker)
{
                    std::lock_guard<std::mutex>
lock_reinit(mDynParamMutex);
  geometry_msgs::msg::TwistStamped cmd_vel;
  if (shouldDriveBackward(pose)) {
    RCLCPP_INFO(logger_, "Drive backward"
                                                "
nav2_path_follower::PurePursuitPathFollower");
                                cmd_vel =
computeBackWardVelocityCommand(pose);
  }
  else {
                                cmd_vel =
Parent::computeVelocityCommands(pose, velocity,
goal_checker);
  }

  return cmd_vel;
}

void PurePursuitPathFollower::setPlan(const
nav_msgs::msg::Path& path) { Parent::setPlan(path); }

void PurePursuitPathFollower::setSpeedLimit(const
double& speed_limit, const bool& percentage)
{
  Parent::setSpeedLimit(speed_limit, percentage);
}

rcl_interfaces::msg::SetParametersResult
PurePursuitPathFollower::dynamicParametersCallback
(
  std::vector<rclcpp::Parameter> parameters)
{
  rcl_interfaces::msg::SetParametersResult result;
                    std::lock_guard<std::mutex>
lock_reinit(mDynParamMutex);

  for (auto parameter : parameters) {
    const auto& type = parameter.get_type();
    const auto& name = parameter.get_name();

    if (type == ParameterType::PARAMETER_DOUBLE)
{
      if (name == plugin_name_ + ".back_distance") {
        mBackDistance = parameter.as_double();
      }
            else if (name == plugin_name_ +
".backward_speed") {
        mBackwardSpeed = parameter.as_double();
      }
            else if (name == plugin_name_ +
".max_angle_difference") {
                    mMaxAngleDifference =
parameter.as_double();
      }
    }
  }
  result.successful = true;
  return result;
}

void
PurePursuitPathFollower::onMinePositionReceived(
      const geometry_msgs::msg::Point::SharedPtr
minePos)
{
  RCLCPP_INFO(logger_, "Mine received"
                                                "
nav2_path_follower::PurePursuitPathFollower");
```

```
   geometry_msgs::msg::Point minePosition;
   minePosition.x = minePos->x;
   minePosition.y = minePos->y;
   mMinePositions.push_back(minePosition);
}

geometry_msgs::msg::TwistStamped
PurePursuitPathFollower::computeBackWardVelocityC
ommand(
   const geometry_msgs::msg::PoseStamped& pose)
{
   geometry_msgs::msg::TwistStamped cmd_vel;

   // Set linear velocity for driving backward
   cmd_vel.twist.linear.x = -mBackwardSpeed;

   // Set angular velocity to zero
   cmd_vel.twist.angular.z = 0.0;

   // Set the header of the TwistStamped message
   cmd_vel.header = pose.header;

   return cmd_vel;
}

bool
PurePursuitPathFollower::shouldDriveBackward(const
geometry_msgs::msg::PoseStamped& pose)
{
   bool result{false};
   if (!mMinePositions.empty()) {
      for (const auto& minePos : mMinePositions) {
         double distance_to_mine =
              std::hypot(pose.pose.position.x - minePos.x,
pose.pose.position.y - minePos.y);

         double angle_to_mine =
                   atan2(minePos.y - pose.pose.position.y,
minePos.x - pose.pose.position.x);
                              double angle_difference =
fabs(tf2::getYaw(pose.pose.orientation)                -
angle_to_mine);
               if (distance_to_mine < mBackDistance &&
angle_difference < mMaxAngleDifference) {
            result = true;
            break;
         }
      }
   }

   return result;
nav2_robot_path_planner,
include/nav2_robot_path_planner, BotGlobalPlanner.h
:
#pragma once
#include <list>
#include "action_msgs/msg/goal_status_array.hpp"
#include <nav2_core/global_planner.hpp>
#include <nav2_navfn_planner/navfn_planner.hpp>
#include <rclcpp/rclcpp.hpp>
#include <rclcpp_action/rclcpp_action.hpp>
namespace robot_global_planner {
struct PointF {
                        static      PointF      fromPose(const
geometry_msgs::msg::PoseStamped& pose)
   {
                              return    {pose.pose.position.x,
pose.pose.position.y};
   }
   PointF operator+(const PointF& p) const { return {x +
p.x, y + p.y}; }
    PointF operator-(const PointF& p) const { return {x -
p.x, y - p.y}; }
     PointF operator*(double value) const { return {x *
value, y * value}; }
    bool operator!=(const PointF& p) const { return x !=
p.x && y != p.y; }
   bool operator==(const PointF& p) const { return x ==
p.x && y == p.y; }
   double x = 0;
   double y = 0;
   bool isValid = true;
};
class          RobotGlobalPlanner          :          public
nav2_core::GlobalPlanner {
private:

std::unique_ptr<nav2_navfn_planner::NavfnPlanner>
mNavFnPlanner;

rclcpp::Subscription<action_msgs::msg::GoalStatusArr
ay>::SharedPtr mSubscription;
   rclcpp_lifecycle::LifecycleNode::SharedPtr mNode;
   std::string mName;
   std::string mGlobalFrame;
   nav2_costmap_2d::Costmap2D* mCostmap;
   PointF mStartPoint;
   PointF mGoalPoint;
   PointF mBeforeObstacle;
public:
   RobotGlobalPlanner();

public:  // nav2_core::GlobalPlanner
                                 void          configure(const
rclcpp_lifecycle::LifecycleNode::WeakPtr& parent,
              std::string name,
              std::shared_ptr<tf2_ros::Buffer> tf,

std::shared_ptr<nav2_costmap_2d::Costmap2DROS>
costmap_ros) override;
   void cleanup() override;
   void activate() override;
   void deactivate() override;
                 nav_msgs::msg::Path     createPlan(const
geometry_msgs::msg::PoseStamped& start,
                                                          const
geometry_msgs::msg::PoseStamped& goal) override;
private:
```

```
    std::vector<PointF> avoidObstacle(const PointF&
before, const PointF& after);
    std::vector<PointF> buildPath(const PointF& start,
const PointF& goal, const PointF& pos);
    PointF interpolatePoint(const PointF& start, const
PointF& goal, const PointF& pos);
                void onNavigationStatus(const
action_msgs::msg::GoalStatusArray& msg);
  size_t getPathIncrements(const PointF& start,
                const PointF& goal,
                double& xIncrement,
                double& yIncrement);
                  geometry_msgs::msg::PoseStamped
createPose(const PointF& point);
};
```

include/nav2_robot_path_planner:

```
#pragma once

#include <list>

#include "action_msgs/msg/goal_status_array.hpp"
#include <nav2_core/global_planner.hpp>
#include <nav2_navfn_planner/navfn_planner.hpp>
#include <rclcpp/rclcpp.hpp>
#include <rclcpp_action/rclcpp_action.hpp>

namespace robot_global_planner {

struct PointF {
                static PointF fromPose(const
geometry_msgs::msg::PoseStamped& pose)
  {
                    return {pose.pose.position.x,
pose.pose.position.y};
  }

  PointF operator+(const PointF& p) const { return {x +
p.x, y + p.y}; }

   PointF operator-(const PointF& p) const { return {x -
p.x, y - p.y}; }

    PointF operator*(double value) const { return {x *
value, y * value}; }

   bool operator!=(const PointF& p) const { return x !=
p.x && y != p.y; }
  bool operator==(const PointF& p) const { return x ==
p.x && y == p.y; }

  double x = 0;
  double y = 0;
  bool isValid = true;
};

class RobotGlobalPlanner : public
nav2_core::GlobalPlanner {
private:
std::unique_ptr<nav2_navfn_planner::NavfnPlanner>
mNavFnPlanner;

rclcpp::Subscription<action_msgs::msg::GoalStatusArr
ay>::SharedPtr mSubscription;
  rclcpp_lifecycle::LifecycleNode::SharedPtr mNode;
  std::string mName;
  std::string mGlobalFrame;
  nav2_costmap_2d::Costmap2D* mCostmap;
  PointF mStartPoint;
  PointF mGoalPoint;
  PointF mBeforeObstacle;

public:
  RobotGlobalPlanner();

public:  // nav2_core::GlobalPlanner
                    void configure(const
rclcpp_lifecycle::LifecycleNode::WeakPtr& parent,
            std::string name,
            std::shared_ptr<tf2_ros::Buffer> tf,

std::shared_ptr<nav2_costmap_2d::Costmap2DROS>
costmap_ros) override;

  void cleanup() override;

  void activate() override;

  void deactivate() override;

              nav_msgs::msg::Path createPlan(const
geometry_msgs::msg::PoseStamped& start,
                                                const
geometry_msgs::msg::PoseStamped& goal) override;

private:
    std::vector<PointF> avoidObstacle(const PointF&
before, const PointF& after);
    std::vector<PointF> buildPath(const PointF& start,
const PointF& goal, const PointF& pos);

    PointF interpolatePoint(const PointF& start, const
PointF& goal, const PointF& pos);

                void onNavigationStatus(const
action_msgs::msg::GoalStatusArray& msg);
  size_t getPathIncrements(const PointF& start,
                const PointF& goal,
                double& xIncrement,
                double& yIncrement);
                  geometry_msgs::msg::PoseStamped
createPose(const PointF& point);
};
```

BotGlobalPlanner.cpp:

```
#include
"nav2_robot_path_planner/BotGlobalPlanner.h"
```

```
#include <tf2/LinearMath/Quaternion.h>

namespace robot_global_planner {

using geometry_msgs::msg::PoseStamped;
using nav_msgs::msg::Path;

constexpr float RESOLUTION = 0.1f;
constexpr float DEVIATION_TRESHOLD = 0.15f;

RobotGlobalPlanner::RobotGlobalPlanner()
                                                    :
mNavFnPlanner{std::make_unique<nav2_navfn_plann
er::NavfnPlanner>()}
   , mStartPoint{false}
   , mGoalPoint{false}
{
}

void                RobotGlobalPlanner::configure(const
rclcpp_lifecycle::LifecycleNode::WeakPtr& parent,
                        std::string name,
                        std::shared_ptr<tf2_ros::Buffer> tf,

std::shared_ptr<nav2_costmap_2d::Costmap2DROS>
costmap_ros)
{
   mNode = parent.lock();
   mGlobalFrame = costmap_ros->getGlobalFrameID();
   mCostmap = costmap_ros->getCostmap();
   mName = name;

       RCLCPP_INFO(mNode->get_logger(),  "Configure
plugin %s of type RobotGlobalPlanner",
           mName.c_str());
         mNavFnPlanner->configure(parent,   name,   tf,
costmap_ros);
}

void RobotGlobalPlanner::cleanup()
{
     RCLCPP_INFO(mNode->get_logger(),  "CleaningUp
plugin %s of type RobotGlobalPlanner",
           mName.c_str());
   mNavFnPlanner->cleanup();
}

void RobotGlobalPlanner::activate()
{
   using namespace std::placeholders;

         RCLCPP_INFO(mNode->get_logger(),   "Activate
plugin %s of type RobotGlobalPlanner",
           mName.c_str());
   mNavFnPlanner->activate();

                                                    mSubscription
=
mNode->create_subscription<action_msgs::msg::Goal
StatusArray>(
      "navigate_to_pose/_action/status", 10,

std::bind(&RobotGlobalPlanner::onNavigationStatus,
this, _1));
}

void RobotGlobalPlanner::deactivate()
{
      RCLCPP_INFO(mNode->get_logger(),  "Deactivate
plugin %s of type RobotGlobalPlanner",
           mName.c_str());
   mNavFnPlanner->deactivate();
}

size_t    RobotGlobalPlanner::getPathIncrements(const
PointF& start,
                              const PointF& goal,
                              double& xIncrement,
                              double& yIncrement)
{
    size_t loopsCount = std::hypot(goal.x - start.x, goal.y
- start.y) / RESOLUTION;
   if (loopsCount != 0) {
      xIncrement = (goal.x - start.x) / loopsCount;
      yIncrement = (goal.y - start.y) / loopsCount;
   }

   return loopsCount;
}

std::vector<PointF>
RobotGlobalPlanner::avoidObstacle(const       PointF&
before, const PointF& after)
{
   std::vector<PointF> ret;
                                 auto        avoidPath
=
mNavFnPlanner->createPlan(createPose(before),
createPose(after));

   for (auto& pose : avoidPath.poses) {
      ret.push_back(PointF::fromPose(pose));
   }
   return ret;
}

std::vector<PointF>
RobotGlobalPlanner::buildPath(const PointF& start,
                                 const PointF& goal,
                                 const PointF& pos)
{
   std::vector<PointF> path{pos};
   PointF prevPoint = pos;
   PointF interpolatedPos = interpolatePoint(start, goal,
pos);
   double xInc, yInc;
   bool insideObstacle = false;
```

```
    auto loops = getPathIncrements(interpolatedPos, goal, xInc, yInc);
  for (size_t i = 0; i < loops; i++) {
     auto currPoint = PointF{interpolatedPos.x + xInc * i, interpolatedPos.y + yInc * i};
    uint32_t mx, my;
              if (mCostmap->worldToMap(currPoint.x, currPoint.y, mx, my)) {
       uint32_t cost = mCostmap->getCost(mx, my);
       if (cost > nav2_costmap_2d::FREE_SPACE) {
         if (!insideObstacle) {
           insideObstacle = true;
           mBeforeObstacle = prevPoint;
         }
       }
       else {
         if (!insideObstacle) {
           path.push_back(currPoint);
         }
         else {
           insideObstacle = false;
                                  auto avoidPoints = avoidObstacle(mBeforeObstacle, currPoint);
            path.insert(path.end(), avoidPoints.begin(), avoidPoints.end());
         }
       }
       prevPoint = currPoint;
    }
  }

   path.push_back(insideObstacle ? mBeforeObstacle : goal);

  return path;
}

PointF RobotGlobalPlanner::interpolatePoint(const PointF& start,
                            const PointF& goal,
                            const PointF& pos)
{
   double distanceStartPos = std::hypot(pos.x - start.x, pos.y - start.y);
    double totalDistance = std::hypot(goal.x - start.x, goal.y - start.y);

     double interpolationParam = distanceStartPos / totalDistance;

   double interpolatedX = start.x + interpolationParam * (goal.x - start.x);
   double interpolatedY = start.y + interpolationParam * (goal.y - start.y);

  return PointF{interpolatedX, interpolatedY};
}
PoseStamped RobotGlobalPlanner::createPose(const PointF& point)
{
  PoseStamped pose;
  pose.pose.position.x = point.x;
  pose.pose.position.y = point.y;
  pose.pose.position.z = 0.0;
  pose.pose.orientation.x = 0;
  pose.pose.orientation.y = 0;
  pose.pose.orientation.z = 0;
  pose.pose.orientation.w = 0;
  pose.header.stamp = mNode->now();
  pose.header.frame_id = mGlobalFrame;

  return pose;
}

Path RobotGlobalPlanner::createPlan(const PoseStamped& start, const PoseStamped& goal)
{
  if (mGoalPoint != PointF::fromPose(goal)) {
      RCLCPP_INFO(mNode->get_logger(), "New Goal received");
    mGoalPoint = PointF::fromPose(goal);
    mStartPoint = PointF::fromPose(start);
  }

  auto positionPoint = PointF::fromPose(start);

  Path path;
  path.poses.clear();
  path.header.frame_id = mGlobalFrame;
  path.header.stamp = mNode->now();
   auto points = buildPath(mStartPoint, mGoalPoint, positionPoint);
  for (size_t i = 0; i < points.size(); i++) {
    path.poses.push_back(createPose(points[i]));
  }

  return path;
}

void RobotGlobalPlanner::onNavigationStatus(const action_msgs::msg::GoalStatusArray& msg)
{
  auto statusList = msg.status_list;
  auto statusCount = statusList.size();

  if (statusCount > 0) {
    auto status = *(--statusList.end());
    switch (status.status) {
                                                  case rclcpp_action::GoalStatus::STATUS_ABORTED:
                                                  case rclcpp_action::GoalStatus::STATUS_CANCELED:
                                                  case rclcpp_action::GoalStatus::STATUS_SUCCEEDED:
      mStartPoint = PointF{0, 0, false};
      mGoalPoint = PointF{0, 0, false};
                RCLCPP_INFO(mNode->get_logger(), "Navigation Completes");
```

```
        break;
      }
    }
}
```

rviz_reconfigure_plugin:
NoHighlightDelegate.cpp:

```
#include "NoHighlightDelegate.h"

namespace rviz::panel::reconfigure {

NoHighlightDelegate::NoHighlightDelegate(QObject*
parent)
   : QStyledItemDelegate(parent)
{
}

void NoHighlightDelegate::paint(QPainter* painter,
                                const QStyleOptionViewItem&
option,
                        const QModelIndex& index) const
{
   QStyleOptionViewItem opt = option;
   opt.state &= ~QStyle::State_HasFocus;
   QStyledItemDelegate::paint(painter, opt, index);
}
```

NoHighlightDelegate.h:

```
#pragma once

#include <QStyledItemDelegate>

namespace rviz::panel::reconfigure {

class          NoHighlightDelegate          :          public
QStyledItemDelegate {
public:
   NoHighlightDelegate(QObject* parent = nullptr);

public:  // QStyledItemDelegate
   void paint(QPainter* painter,
           const QStyleOptionViewItem& option,
           const QModelIndex& index) const override;
};

#include "NodeModel.h"

namespace rviz::panel::reconfigure {

NodeModel::NodeModel(QObject* parent)
   : QAbstractListModel{parent}
{
}

int     NodeModel::rowCount(const     QModelIndex&)
const { return mNodeNames.size(); }

QString   NodeModel::getNodeName(uint32_t   index)
const { return mNodeNames.at(index); }

QVariant     NodeModel::data(const     QModelIndex&
index, int role) const
{
   if (index.isValid()) {
      if (role == Qt::DisplayRole) {
         return mNodeNames[index.row()];
      }
   }

   return {};
}

void          NodeModel::onNodeReceived(QStringList
nodeNames)
{
   beginResetModel();
   mNodeNames = nodeNames;
   endResetModel();
}
#include "ParametersModel.h"

#include <QBrush>

namespace rviz::panel::reconfigure {

ParametersModel::ParametersModel(QObject*
parent)
   : QAbstractTableModel{parent}
{
}

int                ParametersModel::rowCount(const
QModelIndex&) const { return mParameters.size(); }

int              ParametersModel::columnCount(const
QModelIndex&) const { return 2; }

QVariant                ParametersModel::data(const
QModelIndex& index, int role) const
{
   QVariant value;
   if (index.isValid()) {
      auto parameter = mParameters.at(index.row());
      if (role == Qt::DisplayRole || role == Qt::EditRole) {
         switch (index.column()) {
         case 0:
            value = parameter.getName();
            break;
         case 1:
                        if  (parameter.getValue().type()  ==
QVariant::Double) {
                                                value    =
QString::number(mParameters.at(index.row()).getValu
e().toDouble());
            }
            else {
                                                value    =
mParameters.at(index.row()).getValue();
            }
```

```
            break;
          }
        }

        if (role == Qt::ForegroundRole) {
                        return parameter.isChanged() ?
QBrush(Qt::blue) : QBrush(Qt::black);
        }
    }

    return value;
}

Qt::ItemFlags ParametersModel::flags(const
QModelIndex& index) const
{
    auto flags = QAbstractTableModel::flags(index);
    if (index.isValid()) {
        auto parameter = mParameters.at(index.row());
        if (index.column() == 1) {
            if (parameter.isReadonly()) {
                flags = flags & ~Qt::ItemIsEnabled;
            }
            else {
                flags = flags | Qt::ItemIsEditable;
            }
        }
    }

    return flags;
}

bool ParametersModel::setData(const QModelIndex&
index, const QVariant& value, int role)
{
        if (index.isValid() && role == Qt::EditRole &&
index.column() == 1) {
        auto& parameter = mParameters[index.row()];
        if (value.type() == parameter.getValue().type() ||
                            parameter.getValue().type() ==
QVariant::Double) {
                        if (parameter.getValue().type() ==
QVariant::Double) {
            bool isOk;
            auto number = value.toDouble(&isOk);
            if (isOk) {
                parameter.setData(number);
                emit dataChanged(index, index);
                                                    emit
parameterChanged(parameter.getName(), number);
                return true;
            }
        }
        else {
            parameter.setData(value);
            emit dataChanged(index, index);
                                                    emit
parameterChanged(parameter.getName(), value);
            return true;
        }
      }
    }

    return false;
}

void
ParametersModel::onParametersReceived(QVariant
varParams)
{
    beginResetModel();
                                auto parameters =
varParams.value<QList<ParameterItem>>();
                                    mParameters =
QList<ParameterItemEx>{parameters.begin(),
parameters.end()};

    endResetModel();
}

void ParametersModel::restoreData(const
QModelIndex& index)
{
    if (index.isValid()) {
        mParameters[index.row()].restore();
                                            setData(index,
mParameters[index.row()].getValue(), Qt::EditRole);
    }
}

QVariant ParametersModel::headerData(int section,
Qt::Orientation orientation, int role) const
{
    QVariant variant;
        if (orientation == Qt::Horizontal && role ==
Qt::DisplayRole) {
        variant = section == 0 ? "Name" : "Value";
    }

    return variant;
}
#include "ParametersModel.h"

#include <QBrush>

namespace rviz::panel::reconfigure {

ParametersModel::ParametersModel(QObject*
parent)
    : QAbstractTableModel{parent}
{
}

int ParametersModel::rowCount(const
QModelIndex&) const { return mParameters.size(); }

int ParametersModel::columnCount(const
QModelIndex&) const { return 2; }
```

```
QVariant ParametersModel::data(const QModelIndex& index, int role) const
{
  QVariant value;
  if (index.isValid()) {
    auto parameter = mParameters.at(index.row());
    if (role == Qt::DisplayRole || role == Qt::EditRole) {
      switch (index.column()) {
      case 0:
        value = parameter.getName();
        break;
      case 1:
        if (parameter.getValue().type() == QVariant::Double) {
          value = QString::number(mParameters.at(index.row()).getValue().toDouble());
        }
        else {
          value = mParameters.at(index.row()).getValue();
        }
        break;
      }
    }

    if (role == Qt::ForegroundRole) {
      return parameter.isChanged() ? QBrush(Qt::blue) : QBrush(Qt::black);
    }
  }

  return value;
}

Qt::ItemFlags ParametersModel::flags(const QModelIndex& index) const
{
  auto flags = QAbstractTableModel::flags(index);
  if (index.isValid()) {
    auto parameter = mParameters.at(index.row());
    if (index.column() == 1) {
      if (parameter.isReadonly()) {
        flags = flags & ~Qt::ItemIsEnabled;
      }
      else {
        flags = flags | Qt::ItemIsEditable;
      }
    }
  }

  return flags;
}

bool ParametersModel::setData(const QModelIndex& index, const QVariant& value, int role)
{
  if (index.isValid() && role == Qt::EditRole && index.column() == 1) {
    auto& parameter = mParameters[index.row()];
    if (value.type() == parameter.getValue().type() ||
        parameter.getValue().type() == QVariant::Double) {
      if (parameter.getValue().type() == QVariant::Double) {
        bool isOk;
        auto number = value.toDouble(&isOk);
        if (isOk) {
          parameter.setData(number);
          emit dataChanged(index, index);
          emit parameterChanged(parameter.getName(), number);
          return true;
        }
      }
      else {
        parameter.setData(value);
        emit dataChanged(index, index);
        emit parameterChanged(parameter.getName(), value);
        return true;
      }
    }
  }

  return false;
}

void ParametersModel::onParametersReceived(QVariant varParams)
{
  beginResetModel();
  auto parameters = varParams.value<QList<ParameterItem>>();
  mParameters = QList<ParameterItemEx>{parameters.begin(), parameters.end()};

  endResetModel();
}

void ParametersModel::restoreData(const QModelIndex& index)
{
  if (index.isValid()) {
    mParameters[index.row()].restore();
    setData(index, mParameters[index.row()].getValue(), Qt::EditRole);
  }
}

QVariant ParametersModel::headerData(int section, Qt::Orientation orientation, int role) const
{
  QVariant variant;
```

```
      if (orientation == Qt::Horizontal && role == Qt::DisplayRole) {
    variant = section == 0 ? "Name" : "Value";
  }

  return variant;
}
#include "PluginWidget.h"

#include <QAction>
#include <QApplication>
#include <QClipboard>
#include <QCloseEvent>
#include <QList>
#include <QMenu>

#include "ui_PluginWidget.h"

#include "NoHighlightDelegate.h"

namespace rviz::panel::reconfigure {

PluginWidget::PluginWidget(QWidget* parent)
  : QWidget(parent)
  , mUi{new Ui::PluginWidget}
{
  mUi->setupUi(this);

    mUi->propertiesTableView->setItemDelegate(new NoHighlightDelegate(mUi->propertiesTableView));


mParametersProxyModel.setSourceModel(&mParametersModel);
  mParametersProxyModel.setFilterKeyColumn(0);

mNodesProxyModel.setSourceModel(&mNodesModel);
  mNodesProxyModel.setFilterKeyColumn(0);


mUi->nodesTableView->setModel(&mNodesProxyModel);

mUi->propertiesTableView->setModel(&mParametersProxyModel);

                  auto nodesSelectionModel = mUi->nodesTableView->selectionModel();

                        connect(mUi->refreshButton, &QPushButton::clicked, this, &PluginWidget::onRefreshClicked);
                    connect(nodesSelectionModel, &QItemSelectionModel::selectionChanged, this,
      &PluginWidget::onNodeChanged);

    connect(&mRclEngine, &RclEngine::nodeReceived, &mNodesModel, &NodeModel::onNodeReceived);
                                connect(&mRclEngine, &RclEngine::parametersReceived, &mParametersModel,
      &ParametersModel::onParametersReceived);
                                connect(&mRclEngine, &RclEngine::parametersReceived, this, &PluginWidget::onParametersReceived);
                                connect(&mRclEngine, &RclEngine::parameterUpdated, this, &PluginWidget::onParameterUpdated);
                        connect(mUi->propertyFilterLine, &QLineEdit::returnPressed, this,
      &PluginWidget::onParametersFilter);
                    connect(mUi->propertyFilterButton, &QPushButton::clicked, this,
      &PluginWidget::onParametersFilter);
                            connect(mUi->nodeFilterLine, &QLineEdit::returnPressed, this, &PluginWidget::onNodeFilter);
                        connect(mUi->nodeFilterButton, &QPushButton::clicked, this, &PluginWidget::onNodeFilter);
                    connect(mUi->propertiesTableView, &QTableView::customContextMenuRequested, this,
      &PluginWidget::showContextMenu);
                            connect(&mParametersModel, &ParametersModel::parameterChanged, this,
      &PluginWidget::onParameterChanged);
}

void  PluginWidget::setNode(rclcpp::Node::SharedPtr node) { mNode = node; }

void PluginWidget::onRefreshClicked()
{
  mRclEngine.enumerateNodes();
  mCurrentNodeName.clear();
}

void              PluginWidget::onNodeChanged(const QItemSelection& selected, const QItemSelection&)
{
  if (selected.indexes().size() > 0) {
    auto newModelIndex = selected.indexes().at(0);
                          QString nodeName = mNodesProxyModel.data(newModelIndex).toString();
    mRclEngine.enumerateParameters(nodeName);
    mUi->nodeLabel->setText(nodeName);
    mCurrentNodeName = nodeName;
  }
}

void PluginWidget::onParametersFilter()
{
  auto pattern = mUi->propertyFilterLine->text();
  mParametersProxyModel.setFilterRegExp(
            QRegExp(pattern,  Qt::CaseInsensitive, QRegExp::FixedString));
}
```

```
void PluginWidget::onNodeFilter()
{
   auto pattern = mUi->nodeFilterLine->text();

mNodesProxyModel.setFilterRegExp(QRegExp(pattern,
Qt::CaseInsensitive, QRegExp::FixedString));
}

void PluginWidget::closeEvent(QCloseEvent* event)
{
   rclcpp::shutdown();
   event->accept();
}

void PluginWidget::showContextMenu(const QPoint&
pos)
{
                        QModelIndex index =
mUi->propertiesTableView->indexAt(pos);
   if (index.isValid()) {
      QMenu contextMenu;
      QAction restoreAction("Restore value", this);
      QAction copyNameAction("Copy name", this);
      QAction copyValueAction("Copy value", this);
        QAction copyNameAndValueAction("Copy name
and value", this);
          QAction copyAllAction("Copy all names and
values", this);

      contextMenu.addAction(&restoreAction);
      contextMenu.addAction(&copyNameAction);
      contextMenu.addAction(&copyValueAction);

contextMenu.addAction(&copyNameAndValueAction);
      contextMenu.addAction(&copyAllAction);

      connect(&restoreAction, &QAction::triggered,
           [&model = mParametersProxyModel, &index,
&sourceModel = mParametersModel]() {
                                auto sourceIndex =
model.mapToSource(index);
             sourceModel.restoreData(sourceIndex);
           });

       connect(&copyNameAction, &QAction::triggered,
[&model = mParametersProxyModel, &index]() {
         auto dataIndex = model.index(index.row(), 0);
         auto value = model.data(dataIndex).toString();
         qDebug() << value;
         QApplication::clipboard()->setText(value);
      });

       connect(&copyValueAction, &QAction::triggered,
[&model = mParametersProxyModel, &index]() {
         auto dataIndex = model.index(index.row(), 1);
         auto value = model.data(dataIndex).toString();
         qDebug() << value;
         QApplication::clipboard()->setText(value);
      });

                connect(&copyNameAndValueAction,
&QAction::triggered,
                [&model = mParametersProxyModel,
&index]() {
            auto nameIndex = model.index(index.row(),
0);
             auto valueIndex = model.index(index.row(),
1);
                                        auto name =
model.data(nameIndex).toString();
                                         auto value =
model.data(valueIndex).toString();
            qDebug() << name << value;
            QApplication::clipboard()->setText(

QString("\"%1\":\"%2\"").arg(name).arg(value));
          });

      connect(&copyAllAction, &QAction::triggered, []()
{ qDebug() << "copyAllAction"; });


contextMenu.exec(mUi->propertiesTableView->viewp
ort()->mapToGlobal(pos));
   }
}

void          PluginWidget::onParameterChanged(const
QString& name, const QVariant& value)
{

mRclEngine.setParameterValue(mCurrentNodeName,
name, value);
}

void          PluginWidget::onParameterUpdated(const
QString&)
{
                                                    //
mRclEngine.enumerateParameters(mCurrentNodeNa
me);
}

void PluginWidget::onParametersReceived(QVariant)
{

mUi->propertiesTableView->resizeColumnToContents(
0);
}
#pragma once

#include <QSortFilterProxyModel>
#include <QWidget>

#include <rclcpp/rclcpp.hpp>

#include "NodeModel.h"
```

```
#include "ParametersModel.h"
#include "RclEngine.h"

namespace Ui {
class PluginWidget;
}

namespace rviz::panel::reconfigure {

class PluginWidget final : public QWidget {
   Q_OBJECT
private:
   std::shared_ptr<Ui::PluginWidget> mUi;
   rclcpp::Node::SharedPtr mNode;
   RclEngine mRclEngine;
   NodeModel mNodesModel;
   ParametersModel mParametersModel;
   QSortFilterProxyModel mParametersProxyModel;
   QSortFilterProxyModel mNodesProxyModel;
   QString mCurrentNodeName;

public:
   explicit PluginWidget(QWidget* parent);

public:
   void setNode(rclcpp::Node::SharedPtr node);

public:
   void closeEvent(QCloseEvent* event) override;

private slots:
   void onRefreshClicked();
         void onNodeChanged(const QItemSelection&
selected, const QItemSelection& deselected);
   void onParametersFilter();
   void onNodeFilter();
   void showContextMenu(const QPoint& pos);
      void onParameterChanged(const QString& name,
const QVariant& value);
   void onParameterUpdated(const QString& name);
   void onParametersReceived(QVariant);
};
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>PluginWidget</class>
 <widget class="QWidget" name="PluginWidget">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>800</width>
    <height>600</height>
   </rect>
  </property>
                    <layout            class="QVBoxLayout"
name="verticalLayout_4">
   <property name="leftMargin">
    <number>0</number>
   </property>
   <property name="topMargin">
    <number>0</number>
   </property>
   <property name="rightMargin">
    <number>0</number>
   </property>
   <property name="bottomMargin">
    <number>0</number>
   </property>
   <item>
                          <layout       class="QVBoxLayout"
name="verticalLayout_2">
    <property name="leftMargin">
     <number>0</number>
    </property>
    <property name="topMargin">
     <number>0</number>
    </property>
    <property name="rightMargin">
     <number>0</number>
    </property>
    <property name="bottomMargin">
     <number>0</number>
    </property>
    <item>
     <widget class="QSplitter" name="splitter">
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
             <widget  class="QWidget"  name="widget"
native="true">
                          <layout    class="QVBoxLayout"
name="verticalLayout">
       <item>
            <widget class="QWidget" name="widget_3"
native="true">
                          <layout    class="QHBoxLayout"
name="horizontalLayout">
        <item>
                             <widget    class="QLineEdit"
name="nodeFilterLine"/>
        </item>
        <item>
                          <widget    class="QPushButton"
name="nodeFilterButton">
          <property name="text">
           <string>Filter</string>
          </property>
         </widget>
        </item>
       </layout>
      </widget>
     </item>
     <item>
                           <widget    class="QTableView"
name="nodesTableView">
       <property name="showGrid">
        <bool>false</bool>
       </property>
```

```
        <property name="sortingEnabled">
         <bool>true</bool>
        </property>
        <property name="wordWrap">
         <bool>false</bool>
        </property>
        <attribute name="horizontalHeaderVisible">
         <bool>false</bool>
        </attribute>
                                                <attribute
name="horizontalHeaderStretchLastSection">
         <bool>true</bool>
        </attribute>
        <attribute name="verticalHeaderVisible">
         <bool>false</bool>
        </attribute>
       </widget>
      </item>
      <item>
                          <widget   class="QPushButton"
name="refreshButton">
        <property name="text">
         <string>Refresh</string>
        </property>
       </widget>
      </item>
     </layout>
    </widget>
          <widget  class="QWidget"  name="widget_2"
native="true">
                          <layout   class="QVBoxLayout"
name="verticalLayout_3">
      <item>
           <widget class="QWidget" name="widget_4"
native="true">
                          <layout   class="QHBoxLayout"
name="horizontalLayout_2">
        <item>
                             <widget   class="QLineEdit"
name="propertyFilterLine"/>
        </item>
        <item>
                          <widget   class="QPushButton"
name="propertyFilterButton">
          <property name="text">
           <string>Filter</string>
          </property>
         </widget>
        </item>
       </layout>
      </widget>
     </item>
     <item>
      <widget class="QLabel" name="nodeLabel">
       <property name="text">
        <string/>
       </property>
       <property name="alignment">
        <set>Qt::AlignCenter</set>
       </property>
      </widget>
     </item>
     <item>
                          <widget   class="QTableView"
name="propertiesTableView">
       <property name="contextMenuPolicy">
        <enum>Qt::CustomContextMenu</enum>
       </property>
       <property name="editTriggers">
        <set>QAbstractItemView::DoubleClicked</set>
       </property>
       <property name="alternatingRowColors">
        <bool>true</bool>
       </property>
       <property name="selectionMode">

<enum>QAbstractItemView::NoSelection</enum>
       </property>
       <property name="gridStyle">
        <enum>Qt::NoPen</enum>
       </property>
       <property name="sortingEnabled">
        <bool>true</bool>
       </property>
       <property name="wordWrap">
        <bool>false</bool>
       </property>
                                                <attribute
name="horizontalHeaderStretchLastSection">
        <bool>true</bool>
       </attribute>
       <attribute name="verticalHeaderVisible">
        <bool>false</bool>
       </attribute>
      </widget>
     </item>
    </layout>
   </widget>
  </item>
 </layout>
</widget>
<resources/>
<connections/>
</ui>
#include "RvizPluginPanel.h"

#include <QVBoxLayout>

#include <rviz_common/display_context.hpp>

#include "PluginWidget.h"

namespace rviz::panel::reconfigure {

PluginPanel::PluginPanel(QWidget* parent)
```

```
  : Panel{parent}
{

qRegisterMetaType<ParameterItem>("ParameterItem"
);

qRegisterMetaType<QList<ParameterItem>>("Paramet
erItemList");

                    auto metatype =
QMetaType::fromType<QList<rviz::panel::reconfigure::
ParameterItem>>();
  qDebug() << "Metatype:" << metatype.isValid();

                        mWidget =
std::make_shared<PluginWidget>(parent);
  QVBoxLayout* layout = new QVBoxLayout;
  layout->addWidget(mWidget.get());
  layout->setContentsMargins(10, 10, 10, 10);
  setLayout(layout);
}

void PluginPanel::onInitialize()
{
                    auto node =
getDisplayContext()->getRosNodeAbstraction().lock()->
get_raw_node();
  mWidget->setNode(node);
}

void PluginPanel::save(rviz_common::Config config)
const { Panel::save(config); }

void PluginPanel::load(const rviz_common::Config&
config) { Panel::load(config); }

#pragma once

#include <memory>

#include <rviz_common/panel.hpp>

#include "RclEngine.h"

namespace rviz::panel::reconfigure {

class PluginWidget;

class PluginPanel : public rviz_common::Panel {
private:
  std::shared_ptr<PluginWidget> mWidget;

public:
  explicit PluginPanel(QWidget* parent = nullptr);

public:  // rviz_common::Panel
  void onInitialize() override;
      void save(rviz_common::Config config) const
override;
      void load(const rviz_common::Config& conf)
override;
};
#include "tools.h"

namespace rviz::panel::reconfigure {
namespace tools {

bool getNameAndNamespace(const std::string&
fullName, std::string ns, std::string& name)
{
  if (fullName.empty()) {
    return false;
  }

  auto index = fullName.find_last_of("/");
  if (index == std::string::npos) {
    return false;
  }

  index++;

  name = fullName.substr(index);
  if (index == 1) {
    ns = fullName.substr(0, index);
  }
  else {
    ns = fullName.substr(0, index - 1);
  }

  return true;
}
```

tracked_robot_bringup, description.launch.py:

```
import os
from ament_index_python.packages import
get_package_share_directory

from launch import LaunchDescription, LaunchContext
from launch.substitutions import Command,
LaunchConfiguration
from launch.actions import DeclareLaunchArgument,
OpaqueFunction
from launch_ros.actions import Node


def launch_setup(context: LaunchContext,
support_package):
                    namespace =
context.perform_substitution(support_package)

  xacro_path = LaunchConfiguration('xacro_path')
  cover_type = LaunchConfiguration('cover_type')
                diff_drive_emulation =
LaunchConfiguration('diff_drive_emulation')
              use_nominal_extrinsics =
LaunchConfiguration('use_nominal_extrinsics')

  robot_state_publisher_node = Node(
```

```
    package='robot_state_publisher',
    executable='robot_state_publisher',
    parameters=[{
                  #'frame_prefix': f"{namespace}/", #
Reimplemented
https://github.com/ros/robot_state_publisher/pull/16
9
       'robot_description': Command(
          [
            'xacro ', xacro_path, ' ',
            'cover_type:=', cover_type, ' ',
                                  'diff_drive_emulation:=',
diff_drive_emulation, ' ',
                                'use_nominal_extrinsics:=',
use_nominal_extrinsics, ' ',
          ])
    }]
  )
  return [robot_state_publisher_node]


def generate_launch_description():

     namespace = LaunchConfiguration('namespace',
default="tracked_robot")

   # URDF/xacro file to be loaded by the Robot State
Publisher node
  default_xacro_path = os.path.join(

get_package_share_directory('tracked_robot_descripti
on'),
    'urdf',
    'tracked_robot_hardware.urdf'
  )

                          declare_model_path_cmd =
DeclareLaunchArgument(
    name='xacro_path',
    default_value=default_xacro_path,
    description='Absolute path to robot urdf file')

  tracked_robot_cmd = DeclareLaunchArgument(
    name='namespace',
    default_value='tracked_robot',
      description='tracked_robot namespace name. If
you are working with multiple robot you can change
this namespace.')

                           declare_cover_type_cmd =
DeclareLaunchArgument(
    name='cover_type',
    default_value='fisheye',
        description='Cover type to use. Options: pi,
fisheye, realsense, zed.')

  declare_simulation_cmd = DeclareLaunchArgument(
    name='diff_drive_emulation',
    default_value='false',
        description='Enable urdf for differential drive
emulation, for simulation.')

            declare_use_nominal_extrinsics_cmd =
DeclareLaunchArgument(
    name='use_nominal_extrinsics',
    default_value='false',
        description='Use nominal extrinsics ONLY for
Realsense camera.')

  # Define LaunchDescription variable and return it
  ld = LaunchDescription()
  ld.add_action(tracked_robot_cmd)
  ld.add_action(declare_model_path_cmd)
  ld.add_action(declare_simulation_cmd)
  ld.add_action(declare_cover_type_cmd)

ld.add_action(declare_use_nominal_extrinsics_cmd)

ld.add_action(OpaqueFunction(function=launch_setu
p, args=[namespace]))

  return ld
```

display.launch.py:

```
import os
from ament_index_python.packages import
get_package_share_directory

from launch import LaunchDescription
from launch_ros.actions import Node
from launch.launch_description_sources import
PythonLaunchDescriptionSource
from launch.actions import IncludeLaunchDescription,
DeclareLaunchArgument, Shutdown
from launch.conditions import IfCondition,
UnlessCondition
from launch.substitutions import LaunchConfiguration


def generate_launch_description():

                tracked_robot_description_path =
get_package_share_directory('tracked_robot_descripti
on')
                  tracked_robot_bringup_path =
get_package_share_directory('tracked_robot_bringup'
)
  gui = LaunchConfiguration('gui')
  cover_type = LaunchConfiguration('cover_type')
                       diff_drive_emulation =
LaunchConfiguration('diff_drive_emulation')
  rvizconfig = LaunchConfiguration('rvizconfig')

                       default_rviz_config_path =
os.path.join(tracked_robot_description_path, 'rviz',
'urdf.rviz')
```

```
                declare_cover_type_cmd = DeclareLaunchArgument(
      name='cover_type',
      default_value='fisheye',
          description='Cover type to use. Options: pi, fisheye, realsense, zed.')

   declare_simulation_cmd = DeclareLaunchArgument(
      name='diff_drive_emulation',
      default_value='false',
         description='Enable urdf for differential drive emulation, for simulation.')

   declare_gui_cmd = DeclareLaunchArgument(
      name='gui',
      default_value='True',
                description='Flag to enable joint_state_publisher_gui')

   declare_rvizconfig_cmd = DeclareLaunchArgument(
      name='rvizconfig',
      default_value=default_rviz_config_path,
      description='Absolute path to rviz config file')

   joint_state_publisher_gui_node = Node(
      package='joint_state_publisher_gui',
      executable='joint_state_publisher_gui',
      name='joint_state_publisher_gui',
      condition=IfCondition(gui),
      on_exit=Shutdown()
   )
   joint_state_publisher_node = Node(
      package='joint_state_publisher',
      executable='joint_state_publisher',
      name='joint_state_publisher',
      condition=UnlessCondition(gui)
   )
   rviz_node = Node(
      package='rviz2',
      executable='rviz2',
      name='rviz2',
      output='screen',
      arguments=['-d', rvizconfig],
      on_exit=Shutdown()
   )
   description_launch = IncludeLaunchDescription(

PythonLaunchDescriptionSource([tracked_robot_bringup_path, '/launch/description.launch.py']),
         launch_arguments={'cover_type': cover_type, 'diff_drive_emulation': diff_drive_emulation}.items()
      )

   # Define LaunchDescription variable and return it
   ld = LaunchDescription()

   ld.add_action(declare_cover_type_cmd)
   ld.add_action(declare_simulation_cmd)
   ld.add_action(declare_gui_cmd)
   ld.add_action(declare_rvizconfig_cmd)
   ld.add_action(description_launch)
   ld.add_action(joint_state_publisher_node)
   ld.add_action(joint_state_publisher_gui_node)
   ld.add_action(rviz_node)

   return ld
import os
import yaml
from pathlib import Path

from ament_index_python.packages import get_package_share_directory
from launch.substitutions import LaunchConfiguration, PathJoinSubstitution
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch_ros.actions import Node
from launch.actions import DeclareLaunchArgument, OpaqueFunction
from launch import LaunchDescription, LaunchContext
from launch.conditions import IfCondition
from launch.actions import ExecuteProcess, IncludeLaunchDescription, RegisterEventHandler
from launch.event_handlers import OnProcessExit


def generate_launch_description():
                tracked_robot_bringup_path = get_package_share_directory('tracked_robot_bringup')
                package_worlds = get_package_share_directory('tracked_robot_worlds')

                gazebo_ros_path = get_package_share_directory('gazebo_ros')

   default_world_name = 'empty.world' # Empty world: empty
                package_gazebo = get_package_share_directory('tracked_robot_gazebo')


    default_xacro_path = os.path.join(package_gazebo, "urdf", "tracked_robot.gazebo.xacro")

       robot_description_content = os.popen(f"xacro {default_xacro_path}").read()


   world_name = LaunchConfiguration('world_name')
                use_sim_time = LaunchConfiguration('use_sim_time', default='True')
      namespace = LaunchConfiguration('namespace', default="tracked_robot")

   use_sim_time_cmd = DeclareLaunchArgument(
      name='use_sim_time',
```

```
        default_value='True',
        description='Use simulation (Gazebo) clock if true')

    tracked_robot_cmd = DeclareLaunchArgument(
        name='namespace',
        default_value='',
        description='tracked namespace name.')

    gazebo_gui_cmd = DeclareLaunchArgument(
        name='gui',
        default_value='True',
        description='Set to "false" to run headless.')

    gazebo_server_cmd = DeclareLaunchArgument(
        name='server',
        default_value='True',
        description='Set to "false" not to run gzserver.')

    world_name_cmd = DeclareLaunchArgument(
        name='world_name',
        default_value=default_world_name,
        description='Load gazebo world.')

    # start gazebo, notice we are using libgazebo_ros_factory.so instead of libgazebo_ros_init.so
    # That is because only libgazebo_ros_factory.so contains the service call to /spawn_entity
    # Reference options
    # https://github.com/ros-simulation/gazebo_ros_pkgs/blob/foxy/gazebo_ros/launch/gzserver.launch.py
    gazebo_server = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            [os.path.join(gazebo_ros_path, 'launch'), '/gzserver.launch.py']),
        launch_arguments={'world': [package_worlds, "/worlds/", world_name],
                          'verbose': 'true',
                          'init': 'true'}.items(),
        condition=IfCondition(LaunchConfiguration('server'))
    )

    gazebo_gui = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            [os.path.join(gazebo_ros_path, 'launch'), '/gzclient.launch.py']),
        condition=IfCondition(LaunchConfiguration('gui'))
    )


    node_robot_state_publisher = Node(
        package="robot_state_publisher",
        executable="robot_state_publisher",
        output="screen",
        # parameters=[params],
        parameters=[
            {"robot_description": robot_description_content},
        ],
    )

    mine_detector = Node(
        package='tracked_robot_controls',
        executable='tracked_robot_controls_minedetector',
        output='screen'
    )

    mine_position = Node(
        package='tracked_robot_controls',
        executable='tracked_robot_controls_minepositionhandler',
        output='screen'
    )

    spawn_robot = Node(
        package='gazebo_ros',
        executable='spawn_entity.py',
        name='spawn_entity',
        output='screen',
        namespace=namespace,
        arguments=['-entity', 'tracked_robot',
                '-topic', 'robot_description',
                '-x', '0.0', '-y', '0.0', '-z', '0.0',
                '-R', '0.0', '-P', '0.0', '-Y','0.0',
                ]
    )

    load_joint_state_controller = Node(
        package="controller_manager",
        executable="spawner",
        arguments=[
            "joint_state_broadcaster",
            "--controller-manager",
            "/controller_manager",
        ],
        # remappings=[("/diff_drive_base_controller/cmd_vel_unstamped", "/cmd_vel")],
    )

    load_diff_drive_base_controller = Node(
        package="controller_manager",
        executable="spawner",
        arguments=[
            "diff_drive_base_controller",
            "--controller-manager",
            "/controller_manager",
        ],
        # remappings=[("/diff_drive_base_controller/cmd_vel_unstamped", "/cmd_vel")],
    )
```

```
  return LaunchDescription(
    [
      RegisterEventHandler(
        event_handler=OnProcessExit(
          target_action=spawn_robot,
          on_exit=[load_joint_state_controller],
        )
      ),
      RegisterEventHandler(
        event_handler=OnProcessExit(
          target_action=load_joint_state_controller,
          on_exit=[load_diff_drive_base_controller],
        )
      ),
      use_sim_time_cmd,
      tracked_robot_cmd,
      gazebo_gui_cmd,
      gazebo_server_cmd,
      world_name_cmd,
      gazebo_server,
      gazebo_gui,
      mine_detector,
      mine_position,
      node_robot_state_publisher,
      spawn_robot,
    ]
  )
import os
import yaml
from pathlib import Path
import logging

from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory
from launch.substitutions import LaunchConfiguration
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource

from launch.actions import DeclareLaunchArgument
from launch import LaunchDescription
from launch_ros.substitutions import FindPackageShare

def generate_launch_description():
    package_navigation= get_package_share_directory('tracked_robot_navigation')
    package_worlds = get_package_share_directory('tracked_robot_worlds')

    nav2_dir = FindPackageShare(package='nav2_bringup').find('nav2_bringup')
  nav2_launch_dir = os.path.join(nav2_dir, 'launch')
    nav2_params_path = os.path.join(package_navigation, 'params', 'nav2_params.yaml')

    nav2_bt_path = FindPackageShare(package='nav2_bt_navigator').find('nav2_bt_navigator')
    behavior_tree_xml_path = os.path.join(nav2_bt_path, 'behavior_trees', 'navigate_to_pose_w_replanning_and_recovery.xml')
    static_map_path = os.path.join(package_worlds, 'maps', 'empty.yaml')

  slam = LaunchConfiguration('slam')
    use_namespace = LaunchConfiguration('use_namespace')
  autostart = LaunchConfiguration('autostart')
  map_yaml_file = LaunchConfiguration('map')
  params_file = LaunchConfiguration('params_file')
    default_bt_xml_filename = LaunchConfiguration('default_bt_xml_filename')

    use_sim_time = LaunchConfiguration('use_sim_time', default='True')
    namespace = LaunchConfiguration('namespace', default="tracked_robot")

    robot_localization_file_path = os.path.join(package_navigation, 'config', 'ekf.yaml')

    declare_use_namespace_cmd = DeclareLaunchArgument(
      name='use_namespace',
      default_value='False',
      description='Whether to apply a namespace to the navigation stack')

  declare_autostart_cmd = DeclareLaunchArgument(
    name='autostart',
    default_value='true',
    description='Automatically startup the nav2 stack')

  declare_bt_xml_cmd = DeclareLaunchArgument(
    name='default_bt_xml_filename',
    default_value=behavior_tree_xml_path,
    description='Full path to the behavior tree xml file to use')

  declare_slam_cmd = DeclareLaunchArgument(
    name='slam',
    default_value='False',
    description='Whether to run SLAM')

    declare_params_file_cmd = DeclareLaunchArgument(
      name='params_file',
      default_value=nav2_params_path,
```

```
      description='Full path to the ROS2 parameters file
to use for all launched nodes')

   declare_map_yaml_cmd = DeclareLaunchArgument(
      name='map',
      default_value=static_map_path,
      description='Full path to map file to load')

   use_sim_time_cmd = DeclareLaunchArgument(
      name='use_sim_time',
      default_value='true',
           description='Use simulation (Gazebo) clock if
true')

   tracked_robot_cmd = DeclareLaunchArgument(
      name='namespace',
      default_value='',
      description='tracked namespace name.')

   start_robot_localization_local_cmd = Node(
      package='robot_localization',
      executable='ekf_node',
      name='ekf_filter_node_odom',
      output='screen',
      parameters=[robot_localization_file_path,
      {'use_sim_time': use_sim_time}],
                        remappings=[('odometry/filtered',
'odometry/local'),
               ('/set_pose', '/initialpose')])

                           start_ros2_navigation_cmd       =
IncludeLaunchDescription(

PythonLaunchDescriptionSource(os.path.join(nav2_la
unch_dir, 'bringup_launch.py')),
   launch_arguments = {'namespace': namespace,
                  'use_namespace': use_namespace,
                  'slam': slam,
                  'map': map_yaml_file,
                  'use_sim_time': use_sim_time,
                  'params_file': params_file,
                                 'default_bt_xml_filename':
default_bt_xml_filename,
                  'autostart': autostart}.items())

   ld = LaunchDescription()
   ld.add_action(use_sim_time_cmd)
   ld.add_action(tracked_robot_cmd)
   ld.add_action(declare_use_namespace_cmd)
   ld.add_action(declare_autostart_cmd)
   ld.add_action(declare_bt_xml_cmd)
   ld.add_action(declare_map_yaml_cmd)
   ld.add_action(declare_params_file_cmd)
   ld.add_action(declare_slam_cmd)
   ld.add_action(start_robot_localization_local_cmd)
   ld.add_action(start_ros2_navigation_cmd)

   return ld
import os

from      ament_index_python.packages      import
get_package_share_directory
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node
from launch.actions import DeclareLaunchArgument
from launch import LaunchDescription

def generate_launch_description():
                                       package_navigation=
get_package_share_directory('tracked_robot_navigati
on')

                              default_rviz_config_path      =
os.path.join(package_navigation,
'rviz/nav2_config.rviz')

                                        rviz_config_file      =
LaunchConfiguration('rviz_config_file')

                        declare_rviz_config_file_cmd      =
DeclareLaunchArgument(
      name='rviz_config_file',
      default_value=default_rviz_config_path,
         description='Full path to the RVIZ config file to
use')

   start_rviz_cmd = Node(
      package='rviz2',
      executable='rviz2',
      name='rviz2',
      output='screen',
      arguments=['-d', rviz_config_file])

   ld = LaunchDescription()
   ld.add_action(declare_rviz_config_file_cmd)
   ld.add_action(start_rviz_cmd)

   return ld
import os

from      ament_index_python.packages      import
get_package_share_directory
from launch import LaunchDescription, LaunchContext
from launch.actions import DeclareLaunchArgument,
OpaqueFunction
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node
from launch.substitutions import Command

def      launch_setup(context:      LaunchContext,
support_package):
   """ Reference:

https://answers.ros.org/question/396345/ros2-launch
-file-how-to-convert-launchargument-to-string/

https://github.com/UniversalRobots/Universal_Robots
```

```
_ROS2_Driver/blob/main/ur_moveit_config/launch/ur_moveit.launch.py
    """
            # render namespace, dumping the support_package.
                                    namespace = context.perform_substitution(support_package)

                                    use_sim_time = LaunchConfiguration('use_sim_time', default='true')
    xacro_path = LaunchConfiguration('xacro_path')
    # Add option to publish pointcloudz
    publish_pointcloud="True"
    publish_odom_tf="True"

    # Launch Robot State Publisher
    robot_state_publisher_node = Node(
        package='robot_state_publisher',
        executable='robot_state_publisher',
        namespace=namespace,
        parameters=[{'use_sim_time': use_sim_time,
                     #'frame_prefix': f"{namespace}/", # Reimplemented https://github.com/ros/robot_state_publisher/pull/169
                'robot_description': Command(
                    [
                        'xacro ', xacro_path, ' ',
                        'robot_name:=', namespace, ' ',
                                    'publish_pointcloud:=', publish_pointcloud, ' ',
                                     'publish_odom_tf:=', publish_odom_tf, ' ',
                    ])
                }]
    )

    return [robot_state_publisher_node]


def generate_launch_description():
                                package_gazebo = get_package_share_directory('tracked_robot_gazebo')

        namespace = LaunchConfiguration('namespace', default="tracked_robot")

    use_sim_time_cmd = DeclareLaunchArgument(
        name='use_sim_time',
        default_value='true',
            description='Use simulation (Gazebo) clock if true')

     default_xacro_path = os.path.join(package_gazebo, "urdf", "tracked_robot.gazebo.xacro")

                        declare_model_path_cmd = DeclareLaunchArgument(
        name='xacro_path',
        default_value=default_xacro_path,
        description='Absolute path to robot urdf file')

    ld = LaunchDescription()
    ld.add_action(use_sim_time_cmd)
    ld.add_action(declare_model_path_cmd)

ld.add_action(OpaqueFunction(function=launch_setup, args=[namespace]))

    return ld
import os
import launch
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument, IncludeLaunchDescription, TimerAction, RegisterEventHandler
from launch.conditions import IfCondition
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.event_handlers import OnProcessExit, OnProcessStart
from launch.substitutions import Command, FindExecutable, PathJoinSubstitution, LaunchConfiguration

from launch_ros.actions import Node
from launch_ros.substitutions import FindPackageShare


def generate_launch_description():
    # Get URDF via xacro
    robot_description_content = Command(
        [

PathJoinSubstitution([FindExecutable(name="xacro")]),
            " ",
            PathJoinSubstitution(

[FindPackageShare("tracked_robot_description"), "urdf", "tracked_robot_hardware.urdf"]
            ),
        ]
    )
            robot_description = {"robot_description": robot_description_content}

    robot_controllers = PathJoinSubstitution(
        [
            FindPackageShare("tracked_robot_hardware"),
            "controllers",
            "robot_controller.yaml",
        ]
    )
```

```
                tracked_robot_bringup_path = get_package_share_directory('tracked_robot_bringup')

                use_sim_time = LaunchConfiguration('use_sim_time', default='false')
                use_ros2_control = LaunchConfiguration('use_ros2_control')

  # Declare the launch arguments
                declare_use_sim_time_cmd = DeclareLaunchArgument(
    name='use_sim_time',
    default_value='False',
        description='Use simulation (Gazebo) clock if true')

                declare_use_ros2_control_cmd = DeclareLaunchArgument(
    name='use_ros2_control',
    default_value='True',
    description='Use ros2_control if true')


  # Start robot state publisher
                start_robot_state_publisher_cmd = IncludeLaunchDescription(

PythonLaunchDescriptionSource([os.path.join(tracked_robot_bringup_path, 'launch', 'robot_state_publisher.launch.py')]),
                launch_arguments={'use_sim_time': use_sim_time,
                'use_ros2_control': use_ros2_control}.items())

                share_dir = get_package_share_directory('mpu9250driver')
  parameter_file = LaunchConfiguration('params_file')

                params_declare = DeclareLaunchArgument('params_file',
                default_value=os.path.join(
                share_dir, 'params', 'mpu9250.yaml'),
                description='Path to the ROS2 parameters file to use.')

  # Launch controller manager (control_node)
  start_controller_manager_cmd = Node(
    package='controller_manager',
    executable='ros2_control_node',
                parameters=[robot_description, robot_controllers],
    output="both",
  )

  # Delayed controller manager action
                start_delayed_controller_manager = TimerAction(period=2.0, actions=[start_controller_manager_cmd])

  # Spawn diff_controller (robot_controller_spawner)
  start_diff_controller_cmd = Node(
    package='controller_manager',
    executable='spawner',
                arguments=["diffbot_base_controller", "--controller-manager", "/controller_manager"],
  )

  # Delayed diff_drive_spawner action
                start_delayed_diff_drive_spawner = RegisterEventHandler(
    event_handler=OnProcessStart(
      target_action=start_controller_manager_cmd,
      on_start=[start_diff_controller_cmd]))

                # Spawn joint_state_broadcaser (joint_state_broadcaster_spawner)
  start_joint_broadcaster_cmd = Node(
    condition=IfCondition(use_ros2_control),
    package='controller_manager',
    executable='spawner',
                arguments=["joint_state_broadcaster", "--controller-manager", "/controller_manager"])

  # Delayed joint_broadcaster_spawner action
        start_delayed_joint_broadcaster_spawner = RegisterEventHandler(
    event_handler=OnProcessStart(
      target_action=start_controller_manager_cmd,
      on_start=[start_joint_broadcaster_cmd]))


  mpu9250driver_node = Node(
    package='mpu9250driver',
    executable='mpu9250driver',
    name='mpu9250driver_node',
    output="screen",
    emulate_tty=True,
    parameters=[parameter_file]
  )


                lidar_dir = get_package_share_directory('sllidar_ros2')
  argument_for_lidar = ""
  DeclareLaunchArgument(
      'argument_for_lidar',
      default_value = argument_for_lidar,
      description = 'Argument for lidar launch file')

                lidar_included_launch = launch.actions.IncludeLaunchDescription(

launch.launch_description_sources.PythonLaunchDescriptionSource(
```

```
                                        lidar_dir   +
'/launch/view_sllidar_s3_launch.py')                    #
view_sllidar_s3_launch.py
#             launch_arguments = {'argument_for_child':
argument_for_child}.items()
    )


# Create the launch description and populate
    ld = LaunchDescription()

    # Declare the launch options
    ld.add_action(declare_use_sim_time_cmd)
    ld.add_action(declare_use_ros2_control_cmd)

    # Add any actions
    ld.add_action(start_robot_state_publisher_cmd)
    ld.add_action(start_delayed_controller_manager)
    ld.add_action(start_delayed_diff_drive_spawner)

ld.add_action(start_delayed_joint_broadcaster_spawn
er)
    ld.add_action(params_declare)
    ld.add_action(mpu9250driver_node)
    ld.add_action(lidar_included_launch)

    return ld
```