Національний університет кораблебудування
імені адмірала Макарова

Навчально-науковий інститут автоматики та електротехніки
(назва інституту, факультету)

кафедра електричної інженерії суднових та роботизованих комплексів
(назва кафедри)

Допущений до захисту

зав. кафедри ЕІС та РК
(скорочена назва кафедри)

_____ Костенко Д.В.
(підпис) (прізвище та ініціали

" 16 " 12 2025 р.

Кваліфікаційна робота

на здобуття ступеня вищої освіти

магістр

на тему Розробка засобів керування мобільними об'єктами у 3D-просторі

Виконав: студент групи 6361м

Спеціальності

141 «Електроенергетика, електротехніка та електромеханіка»

спеціалізації «Електричні системи і комплекси транспортних засобів»

Студент Дячков А.П. (підпис)
(прізвище та ініціали)

Керівник к.т.н., доцент Бабкін Г.В. (підпис)
(прізвище та ініціали)

Рецензент к.т.н., доцент Клочков О.П. (підпис)
(прізвище та ініціали)

м. Миколаїв – 2025 р.

Міністерство освіти і науки України
Національний університет кораблебудування
імені адмірала Макарова

Інститут, факультет Навчально-науковий інститут автоматики та електротехніки
Кафедра електричної інженерії суднових та роботизованих комплексів
Рівень вищої освіти магістр
Галузь знань 14 «Електрична інженерія»
(шифр і назва)
Спеціальність 141 «Електроенергетика, електротехніка та електромеханіка»
(шифр і назва)
Освітньо-професійна програма «Електричні системи і комплекси транспортних засобів»
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Гарант освітньої програми
_____ Костенко Э.В.
(підпис) (прізвище та ініціали)
"01" 10 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу
на здобуття ступеня вищої освіти магістр

Дячков Андрій Петрович
(прізвище, ім'я, по батькові студента)
1. Тема кваліфікаційної роботи Розробка засобів керування мобільними об'єктами у 3D-просторі

Керівник кваліфікаційної роботи Бабкін Георгій Володимирович, канд. техн. наук, доцент НУК
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «14» 10 2025 року №1217уч
2. Строк подання студентом кваліфікаційної роботи
3. Вихідні дані кваліфікаційної роботи

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

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Електронна презентація

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
ОП	Бабкін Г. В.		
ОНС	Бабкін Г. В.		

7. Дата видачі завдання на кваліфікаційну роботу «_15__» вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	*Видача завдання. Розроблення змісту роботи. Узгодження завдання по спец. розділам*		
2	*Розробка розділу 1. Розробка розділу 2.*		
3	*Розробка розділу 3. Розробка розділу 4.*		
4	*Розробка розділу 5. Затвердження додатків.*		
5	*Розробка висновків та переліку посилань. Розробка реферату та вступу.*		
6	*Затвердження висновків, переліку посилань, реферату та вступу. Оформлення ПЗ та презентації.*		
7	*Підготовка доповіді. Захист МР.*		

Студент _____ *Дячков А. П.*
(підпис) (прізвище та ініціали)

Керівник
кваліфікаційної роботи _____ *Бабкін Г. В.*
(підпис) (прізвище та ініціали

АНОТАЦІЯ

У роботі виконано аналіз існуючих засобів моделювання робототехнічних систем та алгоритмів систем керування. Розроблено структуру взаємодії моделей робототехнічних систем та систем керування, створено приклад програмного забезпечення для комплексу інструментальних засобів моделювання робототехнічних систем та інструментальних засобів моделювання вбудованих алгоритмів систем керування.

Ключові слова: робототехнічні системи; алгоритми керування; моделювання; мобільні роботи; нечітке керування

ABSTRACT

The robot has analyzed the main features of modeling robotic systems and algorithms of heating systems. The structure of the interaction of models of robotic systems and control systems has been broken down, a software application has been created for a complex of instrumental features for modeling robotic systems and instrumental features for modeling the implementation of algorithms for control systems.

Keywords: robotic systems; control algorithms; modeling; mobile robots; fuzzy control

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ППО – проміжне програмне забезпечення

РТО – робототехнічний об'єкт

РТС – робототехнічна система

ROS – Robot Operating System

ВСТУП

Проектування та реалізація складних робототехнічних систем є довготривалими, трудомісткими та дорогими процесами. Рішення, які одержуються в результаті застосування звичних методів розробки, не завжди відповідають споконвічно заявленим вимогам, а іноді можуть бути зовсім не реалізовані на практиці через невизначеність параметрів функціонування системи чи середовища експлуатації.

Останнім часом зростає інтерес до створення єдиного технологічного процесу проектування та розробки робототехнічних систем на основі існуючих та застосовуваних окремо етапів реалізації вищезгаданих систем.

Перспективним напрямом у цій галузі є процес, у якому паралельно відбуваються розробка об'єкта управління та розробка системи управління. У цій роботі пропонується варіант реалізації програмної частини, що поєднує інструментальні засоби розробки об'єкта та його системи управління на етапі моделювання.

Мета роботи: Зниження трудомісткості проектування складних робототехнічних систем із вбудованими пристроями управління шляхом розробки комплексу засобів САПР, що поєднує інструментальні засоби моделювання робототехнічного комплексу та моделювання вбудованих алгоритмів систем керування.

Завдання, які вирішуються у роботі:

– аналіз існуючих засобів моделювання робототехнічних об'єктів та систем управління;

– розробка бібліотеки моделей робототехнічного об'єкту;

– розробка бібліотеки алгоритмів нечіткого керування мобільним об'єктом;

– реалізація програмної частини, що поєднує інструментальні засоби моделювання робототехнічної системи та моделювання системи керування.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ДЛЯ ПРОЕКТУВАННЯ РОБОТОТЕХНІЧНИХ СИСТЕМ. ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальні засади розробки робототехнічних систем

Робототехнічна система (РТС) являє собою сукупність роботів (робототехнічних об'єктів – РТО), автоматичних та/або автоматизованих пристроїв та іншого обладнання, інформаційно та функціонально пов'язаних між собою в єдине ціле (2). У цьому роботі під робототехнічної системою розумітиметься сукупність робототехнічного об'єкта управління та її системи управління.

Проектування та розробка будь-яких РТС є тривалим, трудомістким і дорогим процесом. Класична теорія автоматичного управління визначає конкретні етапи у процесі дослідження та синтезу РТС (рис.1.1).

За структурою процесу видно, що він є ітераційним. Це дозволяє під час проектування повертатися на попередні етапи, що дає можливість коригування моделі, виконання повторного синтезу системи управління або зміни поставленого раніше завдання в цілому. Також залежно від конкретної ситуації чи вимог, що пред'являються може змінюватися значимість кожного етапу проектування. Найскладнішим у реалізації РТС є створення такої математичної моделі об'єкта управління, яка буде адекватна реальному об'єкту, що моделюється, і такої системи управління, яка буде відповідати заявленим вимогам і поставленому завданню. При цьому разом їх мета - заощадити час розробки та знизити ризики виходу з ладу результативної технічної реалізації.

1.2 Принципи створення моделей робототехнічних систем

Будь-яку математичну модель РТО чи РТС загалом можна описати з допомогою аналітичного моделювання. Однак, данне завдання є трудомісткім.

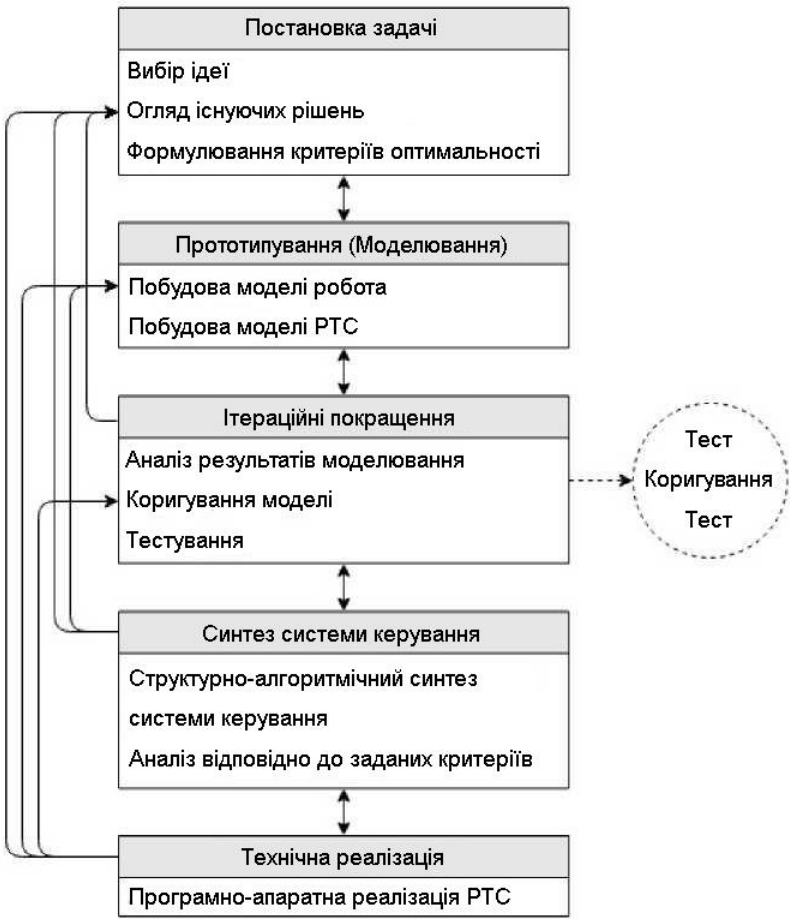

Рисунок 1.1 – Структура процесу дослідження та синтезу РТС

Це полягає в тому, що для опису адекватної моделі навіть найпростішого об'єкта необхідно скласти сукупність аналітичних виразів для системи рівнянь руху заданого об'єкта з урахуванням його фізичних властивостей, системи рівнянь взаємодії із зовнішнім середовищем його експлуатації та системи рівнянь взаємодії об'єкта управління з його системою управління.

На (рис.1.2) наведено приклад розрахунку рівнянь руху спрощеної моделі робота на ходулях у момент, коли він спирається на одну ногу [5]. З одного боку, результати таких обчислень дозволяють прогнозувати поведінку системи з заданою точністю. З іншого боку, крім складності та трудомісткості складання та розрахунку систем рівнянь алгебри, такий підхід має й інші вагомі недоліки:

– внесення будь-яких змін до вимог до об'єкта, його параметрів або постановки завдання в цілому призведе до повторного перерахунку раніше складених систем рівнянь і навіть повторного складання цих рівнянь;

– основна частина створення моделі з використанням такого методу відбувається розробниками вручну, що підвищує ризик виникнення помилок, які можуть бути виявлені лише на етапах технічної реалізації;

– в більшості випадків для прискорення процесу обчислень, обробки результатів і помилок, що виникають, потрібен колектив розробників;

– для реалізації РТС реального часу стає важко виконання розрахунків систем рівнянь у такому обсязі.

$$\left.\begin{aligned}
&\alpha_1\theta + \alpha_8\ddot{\varphi}\sin(\varphi-\theta) - \alpha_2\psi\cos(\psi-\theta) = \\
&= -\alpha_8\dot{\varphi}_2\cos(\varphi-\theta) - \alpha_2\psi^2\sin(\psi-\theta) + \alpha_3\sin\theta - u_1, \\
&\alpha_8\ddot{\theta}\sin(\varphi-\theta) + \alpha_4\ddot{\varphi} + \alpha_3\ddot{\psi}\sin(\psi-\varphi) = \\
&= \alpha_8\dot{\theta}^2\cos(\varphi-\theta) - \alpha_9\dot{\psi}^2\cos(\psi-\varphi) - \alpha_5\cos\varphi + u_1 - u_2, \\
&-\alpha_2\ddot{\theta}\cos(\varphi-\theta) + \alpha_9\ddot{\varphi}\sin(\psi-\varphi) + \alpha_6\ddot{\psi} = \\
&= \alpha_2\dot{\theta}^2\sin(\varphi-\theta) + \alpha_9\dot{\varphi}^2\cos(\psi-\varphi) - \alpha_7\sin\psi + u_2, \\
&\beta_1\ddot{\eta} - \beta_2\ddot{\xi}\cos(\xi-\eta) = -\beta_2\dot{\xi}^2\sin(\xi-\eta) + \beta_3\sin\eta - u_3, \\
&\beta_2\ddot{\eta}\cos(\xi-\eta) - \beta_4\ddot{\xi} = -\beta_2\dot{\eta}^2\sin(\xi-\eta) + \beta_5\sin\xi - u_3,
\end{aligned}\right\}$$

$$\left.\begin{aligned}
&\alpha_1 = I_r + M_r\left(q_r^2 + l^2\right) + M_b l^2, \\
&\alpha_2 = M_r p_r l, \\
&\alpha_3 = \left\{M_r\left(q_r + l\right) + M_b l\right\} g, \\
&\alpha_4 = I_b + 4M_r r^2 + M_b r^2, \\
&\alpha_5 = \left(2M_r + M_b\right) gr, \\
&\alpha_6 = I_r + M_r p_r^2, \\
&\alpha_7 = M_r gp, \\
&\alpha_8 = \left\{M_b + 2M_r\right\} rl, \\
&\alpha_9 = 2M_r p_r r_r, \\
&\beta_1 = I_p + M_p\left(q_p^2 + l^2\right), \\
&\beta_2 = M_p p_p l, \\
&\beta_3 = M_p\left(q_p + l\right) q, \\
&\beta_4 = I_p + M_p p_p^2, \\
&\beta_5 = M_p gp_p.
\end{aligned}\right\}$$

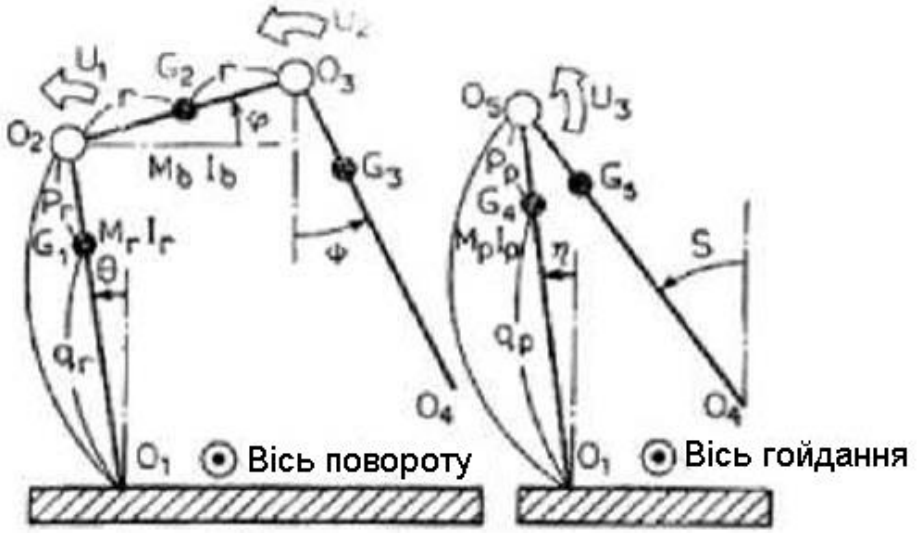

Рисунок 1.2 – Приклад аналітичного опису простої моделі робота

В даний час популярністю користується так зване візуальне конструювання (віртуальне моделювання), яке дає можливість мінімізувати і тимчасові витрати на розрахунки моделей об'єкта та системи управління та матеріальні витрати на подальшу технічну реалізацію.

Реалізувати візуальне конструювання можна за допомогою різних симуляторів та пакетів моделювання. Їхнє основне завдання – знизити часові витрати на аналітичний опис фізичних властивостей моделей РТО за рахунок свого програмного забезпечення, що автоматизує цей процес.

Таким чином, при проектуванні віртуальних моделей помилки розробника перестають носити фатальний характер, на усунення їх та їх наслідків потрібно набагато менше тимчасових та матеріальних витрат, ніж при застосуванні аналітичного підходу.

Також більшість РТС мають схожі структури та властивості, а значить можуть бути частково чи повністю запозичені один в одного. І відповідно створення віртуальних відкритих і поповнюваних бібліотек з математичними моделями об'єктів (бібліотек готових рішень) дозволить прискорити процес проектування РТС надалі, що є ще одним плюсом на користь віртуального конструювання. Звичайно, всі ці позитивні показники актуальні за умови, що розробник "довіряє" розрахункам бібліотек готових методів усередині засобів моделювання.

1.3 Огляд засобів віртуального моделювання робототехнічних систем

Середовища розробки роботів стали відігравати все більш важливу роль у дослідженнях у галузі робототехніки в цілому і зокрема в розробці архітектур для мобільних роботів. На сьогоднішній день існує велика кількість програмних пакетів, що дозволяють моделювати та досліджувати роботу будь-яких РТС, а також вплив передбачуваного зовнішнього середовища експлуатації на неї. Будь-який пакет віртуального моделювання включає графічний і фізичний движки.

Графічний движок (англ, graphics engine; система рендерингу чи «візуалізатор») – це проміжне програмне забезпечення (ППО). Його основним завданням є візуалізація (рендеринг) дво- чи тривимірної комп'ютерної графіки. Графічні двигуни можуть працювати як у реальному часі, так і витрачати по кілька десятків годин на виведення одного зображення. Це залежить від того, який використовується процесор: графічний процесор відеокарти або центральний процесор[1].

До найвідоміших вільних графічних двигунів відносяться:

– OGRE – об'єктно-орієнтований графічний двигун, написаний на C++ [35]. Двигун є багатофункціональним, оскільки за його допомогою можна створювати ігри різних жанрів та інші додатки, не пов'язані з іграми. Підтримується рендеринг як через Direct3D9[42], і через OpenGL [37]. Двигун має досить потужне співтовариство підтримки, велику документацію та навчальні приклади на багатьох мовах;

– Irrlicht – графічний двигун, який використовує можливості OpenGL і DirectX, написаний на C++ [21];

– GLScene – OpenGL-орієнтований графічний двигун для Delphi [14].

Фізичний движок (англ, physics engine) – програма чи підпрограма, яка здійснює моделювання фізичних законів реального світу у віртуальному просторі ЗО-сцени із заданим ступенем точності. Сучасні фізичні движки симулюють лише деякі закони фізики, проте, з часом і прогресу список «підтримуваних» законів збільшується. Зараз більшість фізичних двигунів

можуть симулювати такі фізичні явища і стани: динаміку абсолютно твердого і деформованого тіла, динаміку рідин і газів, поведінку тканин і мотузок (троси, канати і т.д.).

На практиці фізичний движок дозволяє наповнити віртуальний 3D-простір статичними і динамічними об'єктами – тілами (англ. body), вказати деякі загальні закони взаємодії тіл та середовища у просторі, наближені до фізичних, задаючи при цьому характер та ступінь взаємодій. Розрахунок взаємодії тіл між собою та з середовищем двигун бере на себе, наближаючи фізичну модель отримуваної системи до реальної та передаючи уточнені геометричні дані в графічний двигун. На (рис.1.3) показана схема взаємодії фізичного та графічного двигунів.

Якщо простого набору об'єктів, що взаємодіють за певними законами у віртуальному просторі, недостатньо, через неповне наближення фізичної моделі до реальної, можна додавати до тіл зв'язку (англ, joint, «з'єднання»). Зв'язки представють обмеження об'єктів фізики, кожне з яких може накладатися на одне або два тіла [1].

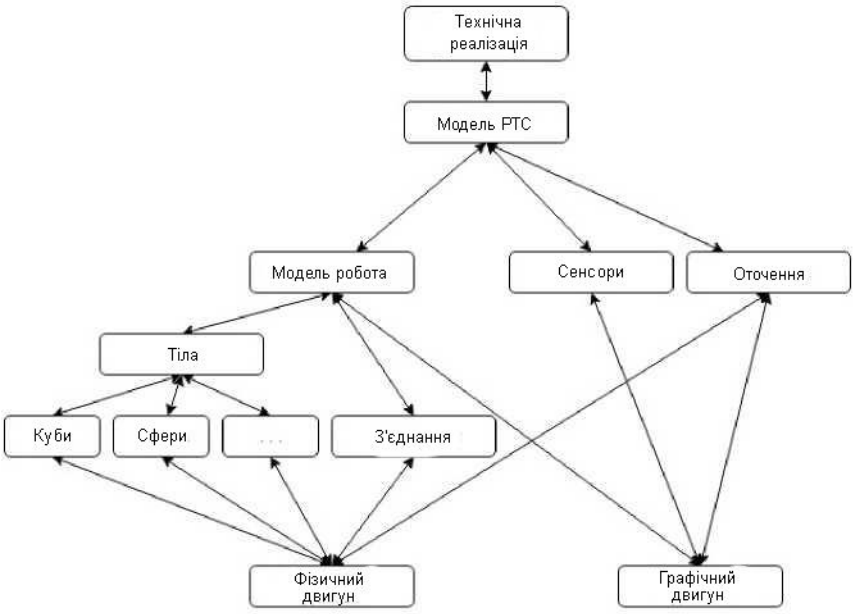

Рисунок 1.3 – Схема взаємодії фізичного та графічного двигунів

Сучасні фізичні двигуни:

– PhysX (PhysX SDK) – раніше відомий як NovodeX. В даний час належить

Nvidia [32]. Використовується в ігрових двигунах, наприклад, Unity[50]. Є безкоштовні плагіни для тривимірних програм;

– Havok – популярне та відоме комерційне рішення. Лінійка продуктів Havok включає не тільки сам фізичний двигун, але і спеціальні інструменти для AI, анімації та поведінки [18];

– Open Dynamics Engine (ODE) – простий старий безкоштовний двигун, розроблений Расселом Смітом, з відкритим вихідним кодом [34];

– Bullet Physics Library – фізичний двигун реального часу, який поширюється під вільною ліцензією zlib. Використовується в комп'ютерних іграх, фільмах, програмах тривимірного моделювання, як компонент інших ігрових двигунів і в багатьох інших специфічних випадках [9].

Пакети моделювання відрізняються своїми можливостями, які залежать від використовуваних у сукупності графічного та фізичного двигунів. Відповідно, від вибору середовища моделювання залежить, наскільки складну та реалістичну модель робота або цілої робототехнічної системи можна реалізувати. Нижче наведено деякі популярні пакети моделювання.

Gazebo

Програмний пакет Gazebo [15] – симулятор з відкритим вихідним кодом розроблений компанією Open Source Robotics Foundation для моделювання роботи роботів в оточенні зовнішнього середовища. Програмний комплекс дозволяє тестувати алгоритми, проектувати роботів, проводити регресійне тестування та навчати систему штучного інтелекту з використанням реалістичних сценаріїв.

Gazebo має доступ до кількох високопродуктивних фізичних двигунів, включаючи ODE і Bullet. Це дозволяє імітувати кінематику та динаміку взаємодії об'єктів між собою та із середовищем експлуатації, а також отримувати та передавати правдоподібні показання з датчиків та інших пристроїв. А як графічний движок Gazebo використовує OGRE, що забезпечує реалістичну візуалізацію навколишнього середовища, включаючи освітлення, тіні та текстури. Завдяки зручним програмним та графічним інтерфейсам розробник

може створити свою власну модель робота з використанням SDF формату XML, який описує об'єкти та середовища для симуляторів роботів, візуалізації та управління. Спочатку розроблений як частина робота-симулятора Gazebo, SDF був створений з урахуванням потреб наукових досліджень у галузі робототехніки. За минулі роки SDF став стабільним, надійним та розширюваним форматом, здатним описувати всі аспекти роботів, статичних та динамічних об'єктів, освітлення, місцевості та законів фізики. Також програмувати можна мовами C/C++ та Python. На даний момент Gazebo повноцінно розроблений тільки для Unix-подібних систем. Однак, розробники надають докладний опис для самостійного збирання пакета під Windows, а також розробляють готову повноцінну версію для даної операційної системи. Надається повністю безкоштовно.

На основі робіт [3, 4, 49] можна переконатися, що цей пакет використовується багатьма розробниками РТС у різних галузях. Самі творці Gazebo як приклади для досліджень у вільному доступі пропонують кілька моделей реальних роботів, наприклад: PR2. Pioneer 2 DX. iRobot Create, та TurtleBot.

У роботі [8] використовується віртуальна модель Willow Garage Personal Robot 2 (PR2), що існує в Gazebo, з його сенсорною системою (рис.1.4) яка полегшує симуляцію пересування робота та сенсорних вимірювань для SLAM (simultaneous localization and mapping – метод одночасної локалізації та побудови карти) та задач навігації. Результати, описані у статті, демонструють вірність моделюваної тривимірної кімнати по відношенню до отриманої від робота лазерної системи ROS-карті та здійсненність SLAM на основі ROS з мобільним роботом-симулятором Gazebo для його використання в тривимірному середовищі на основі камери. Також у роботі детально розглянуто моделювання локомоції робота PR2. даних датчиків та алгоритму SLAM.

V-REP

V-Rep – безкоштовний та відкритий кросплатформовий програмний комплекс від компанії Coppelia Robotics створений для розробки реалістичних

моделей роботів на базі універсальної архітектури [51]. Він не має центрального модуля, а складається із відносно самостійних функціональних груп, які можна використовувати за потребою.

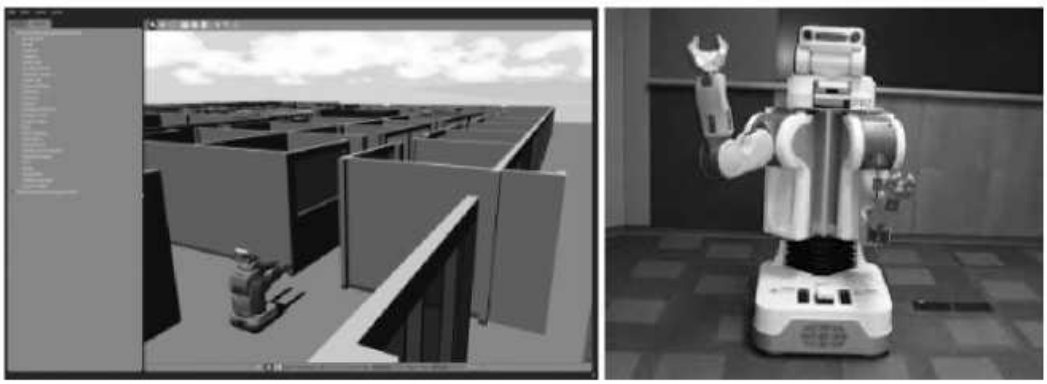

Рисунок 1.4. – Модель робота PR2 у симуляторі Gazebo (ліворуч) та його технічна реалізація (праворуч)

Симулятор V-Rep з інтегрованим середовищем розробки заснований на розподіленій архітектурі управління: кожен об'єкт/модель може індивідуально керуватися за допомогою вбудованого скрипту, плагіна, віддаленого клієнта API або інтерфейсу користувача. Це робить V-Rep універсальним та придатним для застосування з кількома роботами. Всі об'єкти, що програмуються в цій системі, моделюються в реальному з погляду фізичних законів світі, що має гравітацію, що надає можливість захоплення предметів, відтворення зіткнень, реалізації роботи датчиків відстані, відео датчиків тощо.

Крім цього, V-Rep використовується для розробки алгоритмів, моделювання автоматизації виробництва, прототипування та перевірки, навчання в галузі робототехніки, віддаленого моніторингу, подвійної перевірки безпеки і т.д. Однак спочатку ядро системи працює тільки з мовою Lua, а для використання інших мов програмування потрібно написати невелику допоміжну програму цією мовою.

У [39, 46, 52] можна знайти різні варіанти застосування цього пакета моделювання. Так, наприклад, у роботі [28] розглянуто створення програмного середовища на основі RL Python, яке орієнтоване на проведення експериментів з різними роботами та середовищами для різних роботизованих задач поряд з

різними стандартними варіаціями алгоритму RL (алгоритму навчання з підкріпленням) та методами вибору дій. Програмне забезпечення, зване RL-ROBOT, може використовуватися автономно з фізично реалістичним симулятором робота V-Rep або операційною системою робота (ROS). Це проект із відкритим вихідним кодом, призначений для полегшення реалізації нових завдань для віртуальних та реальних роботів. Розробниками було проведено серію експериментів із роботом Giraff (рис.1.5) який оснащений двома диференціальними приводами, двома колесами та лазерним далекоміром Hokuyo.

Рисунок1.5 – Модель робота PR2 у симуляторі Gazebo (ліворуч) та його технічна реалізація (праворуч)

Giraff може вивчити ті самі завдання, що були розроблені для модельованого мобільного робота в V-Rep. просто вказавши фізичні параметри завдання. І тут RL-ROBOT запускає вузол ROS. надаючи доступ до реальних датчиків та виконавчих механізмів.

OpenHRP

OpenHRP – це віртуальна платформа для гуманоїдного робота, що складається з симулятора гуманоїдних роботів та бібліотеки управління рухом для них[38]. Очікується, що OpenHRP забезпечить підтримку досліджень гуманоїдної робототехніки на програмному та апаратному забезпеченні з відкритою архітектурою завдяки уніфікації контролерів та перевіреній

узгодженості між симулятором та справжнім роботом-гуманоїдом. Симулятор побудований на основі CORBA (Common Object Request Broker Architecture) [22], яка є стандартною програмною архітектурою, а контролер робота в реальному часі працює на ART-Linux[24]що є розширенням Linux в реальному часі. Крім того, симулятор та контролери є "білими ящиками". Користувач може реалізувати контролер із використанням довільної мови у довільній операційній системі, якщо він має прив'язку CORBA (наприклад: C/C++, Java). Оскільки реалізовано уніфікацію контролерів та узгодженість між модельованими та реальними роботами, OpenHRP може бути корисною віртуальною платформою для гуманоїдної робототехніки, на якій можуть бути розроблені різні фундаментальні технології. Віртуалізація платформи є дуже важливою для ефективного успадкування бібліотеки програмного забезпечення від одного обладнання до іншого, і це спільне використання робить розробку більш ефективною.

На OpenHRP розроблено генератор моделей ходьби, який заснований на тривимірному лінійному інвертованому маятниковому режимі. Відмінна риса цього методу генерації в порівнянні з іншими [31] полягає в тому, що він здатний генерувати шаблон із дуже високою швидкістю, оскільки він не використовує ітеративний розрахунок. Використовуючи цю функцію, він може генерувати як автономний режим, а й онлайн за необхідності. Розроблений контролер застосовується до апаратної платформи HRP-1S для реального робота, який показано на рис.1.6) [55].

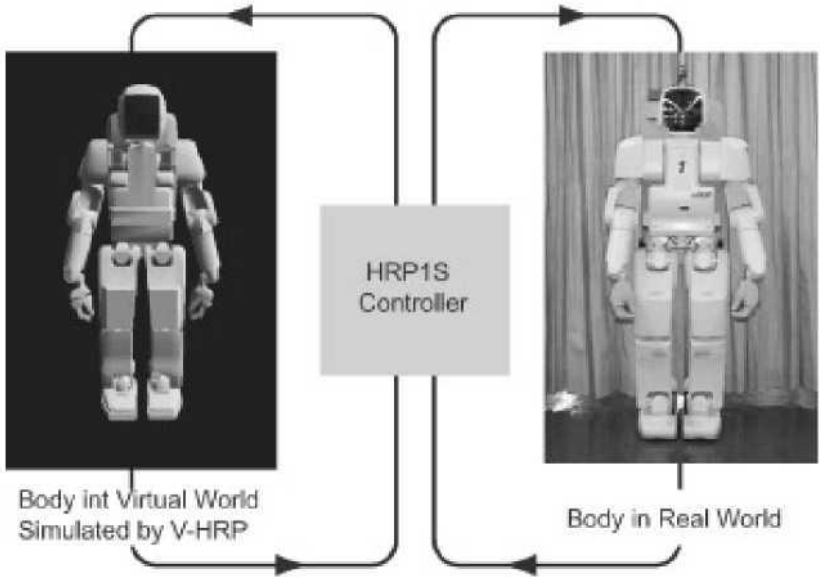

Рисунок 1.6 – Віртуальна та реальна моделі робота-гуманоїда на платформі HRP-1S

HRP-1S має висоту 1600 мм. ширину 600 мм та вага 99 кг без урахування батарей. Він управляється процесорною платою VME з урахуванням Intel Pentinmlll. яка знаходиться в рюкзаку робота. ART-Linux працює на ній. Ця програма не потребує портування контролера завдяки механізму уніфікації контролера OpenHRP. Цей механізм реалізується шляхом запровадження адаптера, який узагальнює інтерфейси тіла апаратного забезпечення та тіла програмного забезпечення у світі моделювання.

Тому ця програма виконується простою заміною адаптера для тіла, що моделюється на адаптер для реального, на (рис.1.7) показаний результат. Обидва кути нахилу тіла і сила реакції ґрунту показують поведінку, яка дуже схожа на симуляцію.

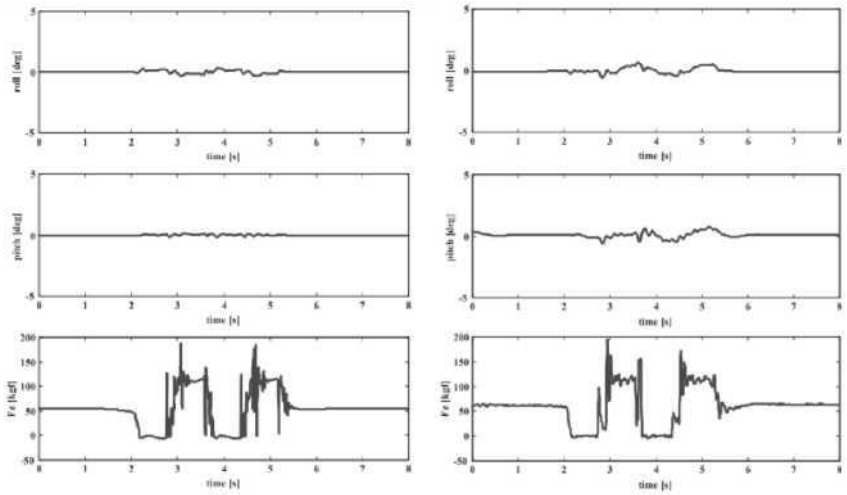

Рисунок 1.7 – Графіки кута нахилу тіла та сила реакції ґрунту при роботі віртуальної моделі (ліворуч) та при реальній моделі (праворуч)

Webots

Webots – це симулятор роботів з відкритим вихідним кодом, який надає повне середовище розробки для моделювання, програмування та симуляції роботів[53]. Тисячі установ по всьому світу використовують його для наукових досліджень та навчання. Webots був розроблений у Федеральному технологічному інституті Швейцарії у Лозанні, ретельно протестований, добре документований та постійно підтримується з 1996 року. Це ефективне рішення для швидкого отримання професійних результатів, включаючи:

– розробку складних робототехнічних систем – швидке прототипування з нуля;

– моделювання автономних автомобілів – оснащених датчиками та взаємодіючих з навколишнім середовищем;

– перевірку нових досліджень у галузі робототехніки – глибоке навчання, еволюційні алгоритми, мультиагент тощо.

Webots пропонує кілька варіантів програмування для управління віртуальними моделями роботів багатьма мовами програмування: мовою C – API функції; на C++, Python, Java та ROS – API документація; для використання протоколу TCP/IP – інтерфейс із будь-якою системою контролера робота, що працює на віддаленій машині. Подібно до програмування робота, користувач

може запрограмувати супервізора симуляції, який буде автоматизувати такі речі під час прогону симуляції, як переміщення об'єктів, скидання положень роботів, збереження траєкторії руху робота і т. д. Також можна реалізувати свій або доповнити існуючий фізичний плагін, щоб налаштувати динамік сили або моменти, що крутять (вітер, обурення). Всі ці API доступні безпосередньо з інтегрованого середовища розробки Webots (IDE), яке включає редактор вихідного коду і редактор дерева сцен.

Однією із запропонованих моделей роботів у бібліотеці Webots є робот саламандр, керований моделлю спинного мозку (рис.1.8, ліворуч). Розробники в [20] пропонують нейронні механізми для модуляції швидкості, напряму та типу ходи, які актуальні для всіх чотирилапих. Вони передбачають, що коливальні центри кінцівок мають нижчі власні частоти, ніж коливальні центри тіла, і представляють біологічні дані, що підтверджують це.

Модель імітує центральний генератор патернів (CPG, тобто ланцюг, здатний генерувати скоординовані патерни ритмічної нейронної активності), що лежать в основі локомоції в спинному мозку. Оі реалізовано у вигляді системи зв'язаних нелінійних генераторів і працює на мікроконтролері на борту робота-амфібії.

Робот поєднує в собі три режими пересування: плавання, повзання та ходьба. Прості сигнали приводу передаються по бездротовій мережі від ноутбука до роботи для модуляції локомоції. Як підтвердження своїх здогадів і результатів відповідних віртуальних моделей. багато розробників створювали реальні технічні реалізації даного робота і демонстрували самі завдання й досягнення результатів, як у симуляторе (рис.1.8. праворуч).

ARGoS призначений для моделювання складних експериментів за участю великих груп роботів різних типів. ARGoS – це перший симулятор сімейств роботів, який одночасно ефективний (швидка робота з багатьма роботами) та гнучкий (легко налаштовується для конкретних експериментів) [6].

В ARGoS фізичні двигуни – це просто плагіни. Користувач може вибрати, який фізичний двигун використовуватиме симуляції. Крім того, фізичний

простір може бути розділений на кілька областей, кожна з яких управляється певним фізичним двигуном. Іншими словами, ARGoS дозволяє моделювати декілька двигунів. Коли роботи переміщаються навколишнім середовищем, вони автоматично передаються від двигуна до двигуна. Навколишнє середовище - це також плагін, який реалізує відповідні алгоритми для імітації засобів зв'язку роботів (наприклад, дальність і радіус дії, Wi-Fi і т.д.). Середовище можна як ще один тип двигуна, який ARGoS запускає для детального моделювання. Контролери роботів використовують спеціальний API для роботів, званий інтерфейсом управління, – той же API, що реалізований на реальному роботі. Це дозволяє плавно переходити від моделювання до реальності.

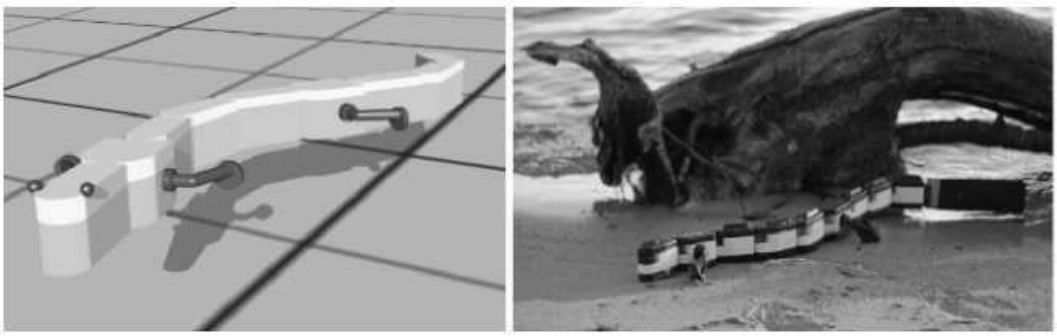

Рисунок 1.8 – Робот саламандр, керований моделлю спинного мозку Webots

Основний цикл симуляції в ARGoS розподіляється на кількох потоках: головний і підлеглий. Головний потік призначає завдання підлеглим. Крім того, ARGoS пропонує два методи для розподілу роботи по потоках: складання розкиду та h-dispatch. Перший забезпечує кращі результати, коли завдання мають аналогічні обчислювальні витрати, наприклад, для експериментів за участю великих скупчень ідентичних роботів. Друге, коли завдання дуже різноманітні, коли в експерименті беруть участь дуже різні роботи. Важливим аспектом є те, що паралелізм вбудований в ядро ARGoS, але він повністю прозорий для розробника плагінів. Таким чином, при створенні нового плагіна розробнику не потрібно впоратися з арбітражем загальних ресурсів.

ARGoS був основним інструментом моделювання роботів у наступних

європейських проектах: ASCENS, H2SWARM, E-SWARM та Swamiix [41, 12, 40].

Також ARGoS був офіційним симулятором проекту Swamianoid [48]. Swarmanoid складається з численних (близько 60) гетерогенних, динамічно пов'язаних невеликих автономних роботів трьох типів: eye-bot – бот. відповідальний за огляд, пошук та орієнтування, foot-bot – бот. відповідальний за сканування відстані. земну опору та батарею для інших ботів, і hand-bot – бот. що відповідає за перенесення та сортування, а також пересування вертикальними поверхнями. Завдання Swarmanoid знайти один або кілька цільових об'єктів, розташованих в 2D або в 3D-подорожі кімнати. Таке завдання може бути розбите на наступні підзавдання, які виконуються асинхронно та/або послідовно роботами, що діють у середовищі: пошук предметів, збирання/схоплювання. транспортування, складування/організація. На (рис.1.9) представлені моделі трьох типів роботів у симуляторі.

На (рис.1.10) реальні моделі роботів зліва направо: foot-bot. hand-bot та eye-bot.

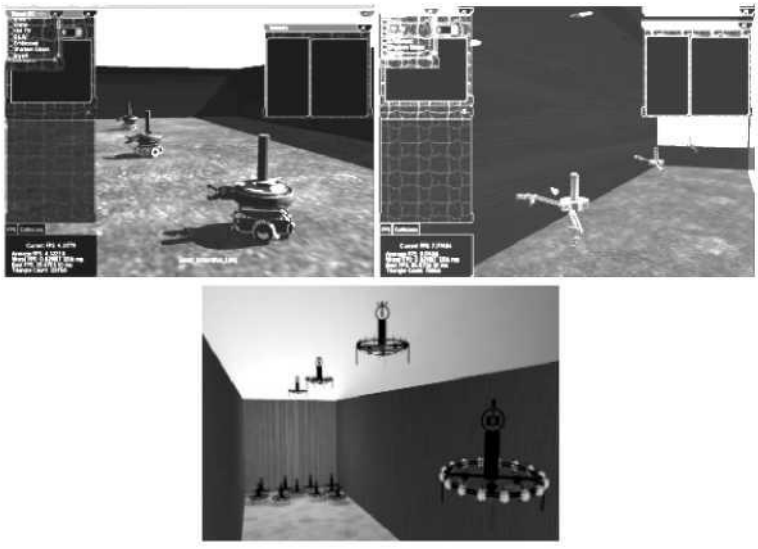

Рисунок 1.9 – Моделі foot-bot. hand-bot та eye-bot у симуляторі ARGoS

Для того, щоб розробникам можна було плавно перейти від моделювання роботів та їх роботи до реальних технічних реалізацій та експериментів, у

пакетах моделювання можна використовувати той же API. що й реального робота. Також під час створення свого робота необхідно реалізовувати його архітектуру, протокол обміну повідомленнями, драйвера для систем управління тощо. з нуля, а потім об'єднати їх у єдину систему робота. У результаті така система буде придатна лише для поодинокого випадку або цілком ідентичних роботів. Щоразу це ускладнює процес розробки. Тому більшість розробників прагне використання більш загальних коштів, наприклад. ROS (див.[4.8.28.13]).

Рисунок 1.10 – Реальні моделі роботів зліва направо: foot-bot. hand-bot та eye-bot

ROS та ROS 2.0

Robot Operating System (ROS) – це набір програмних бібліотек та інструментів, які дозволяють створювати програми для роботів. Від драйверів до найсучасніших алгоритмів. ROS спочатку був спроектований так, щоб бути якомога розподіленішим і модульнішим, щоб користувачі могли використовувати ROS так, як вони хочуть. Модульність ROS дозволяє вибирати, які частини є корисними, а які частини будуть реалізовані розробником самостійно.

Надаючи програмам управління свої інтерфейси, бібліотеки та готові програми, ROS виступає як «операційна система». Для програм взаємодії та управління роботом вже реалізовані драйвера, що дозволяють єдиним чином працювати з багатьма пристроями (джойстиками, GPS, камерами, контролерами тощо). Також у ROS вже містяться допоміжні бібліотеки та програми для роботів (перетворення систем координат, утиліти для візуалізації даних, систему для налагодження (дотування), SLAM та багато іншого). Таким чином, для типових

завдань робототехніки можна використовувати готові модулі (пакети та стеки), дописуючи лише необхідний функціонал [10, 43].

ROS є операційною системою з відкритим вихідним кодом для роботів, що широко використовується і швидко розвивається в співтоваристві робототехніки. Однак, працюючи в Linux, ROS не дає гарантій забезпечення реального часу, тоді як саме ці завдання потрібні в багатьох додатках роботів, таких як управління рухом роботів. Можна скористатися різними плагінами, що приводять ROS до роботи в реальному часі, але їхня достовірність не завжди може бути справжньою. Тому для усунення цієї проблеми, а також для вирішення нових завдань розробники працюють над новою версією ROS – ROS 2.0 [44].

ROS 2.0 – версія, яка розширює початкову концепцію (спочатку призначену виключно для дослідних цілей) та спрямована на надання розподіленого та модульного рішення для ситуацій, у яких беруть участь команди роботів, систем реального часу чи виробничих середовищ.

У ROS 2.0 використовується перероблена архітектура системи [29]. На (рис.1.11) показані архітектури двох поколінь ROS та відмінності між ними:

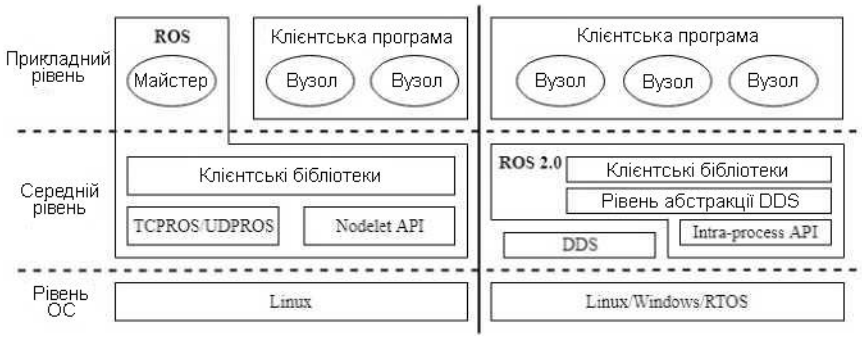

Рисунок 1.11 – Архітектури ROS та ROS 2.0

– Рівень операційної системи ROS побудований поверх систем Linux, ROS 2.0 вносить зміни, які підтримують системи, створені у тому числі Linux, Windows, Mac, RTOS, і навіть голе залізо без операційної системи.

– Середній рівень

Одним із найбільш важливих понять у ROS є обчислення «вузлів» у графі, що дозволяє розробникам паралельно розробляти функціональні модулі з малим зв'язком та полегшувати вторинне мультиплексування. Система зв'язку ROS базується на TCPROS/UDPROS. ROS 2.0 використовує Data Distribution Service (DDS) – це стандартне рішення для публікації/отримання даних у розподілених системах реального часу. ROS 2.0 забезпечує реалізацію DDS на рівні абстракції і користувачам не потрібно звертати увагу на основного постачальника DDS. В архітектурі ROS Nodelet та TCPROS/UDPROS є паралельними рівнями, які можуть забезпечити оптимізований режим передачі даних для кількох вузлів в одному процесі. Аналогічний метод передачі також зберігається в ROS 2.0, званої «внутрішньопроцесним», який також залежить від DDS.

– Прикладний рівень

ROS значною мірою залежить від Майстра ROS, і цілком можливо, що коли Майстер відключиться, вся система зіткнеться з дилемою. В архітектурі ROS 2.0 Майстер відсутній, а між вузлами використовується механізм виявлення, щоб допомогти один одному встановити з'єднання.

У [16] розповідається про експериментальну установку, що досліджує придатність ROS 2.0 для роботизованих програм реального часу. Зв'язок ROS 2.0 оцінюється у разі апаратного міжкомпонентного комунікаційного обладнання, що працює поверх Linux. Дослідники перевірили та вивчили затримки найгіршого випадку та охарактеризували зв'язок ROS 2.0 для додатків реального часу. Результати показують, що правильна конфігурація середовища ROS 2.0 забезпечує "м'який" реальний час, а в ряді випадків – і "жорсткий".

Крім моделювання функціонування роботів у їхньому довкіллі, часто виникає необхідність вивчити модель поведінки створеного робота на основі того чи іншого алгоритму управління. Деякі пакети моделювання, розглянуті вище, дозволяють аналізувати поведінку роботів на основі простих алгоритмів, заданих без сторонніх засобів, але це не завжди швидкий, зручний та інтуїтивно зрозумілий спосіб. Тому для вирішення задачі аналізу алгоритмів управління як додаткові засоби САПР використовуються такі пакети, як MATLAB або OpenAI.

MatLab

MATLAB – це високорівнева мова та інтерактивне середовище для програмування, чисельних розрахунків та візуалізації результатів. За допомогою MATLAB можна аналізувати дані, розробляти алгоритми, створювати моделі та програми. MATLAB у порівнянні з традиційними мовами програмування (C/C++, Java, Pascal, FORTRAN) дозволяє на порядок скоротити час вирішення типових завдань та значно спрощує розробку нових алгоритмів. MATLAB широко використовується в таких областях, як обробка сигналів та зв'язок, обробка зображень та відео, системи управління, автоматизація тестування та вимірювань тощо. MATLAB з Robotics System Toolbox забезпечує алгоритми та апаратні засоби зв'язку для розробки додатків для автономних мобільних роботів [30].

Наприклад, Gazebo MATLAB підключається через інтерфейс ROS, до V-Rep – через ROS або remote API, а до Webots просто через команду в терміналі (командному рядку). У Linux інсталятор MATLAB зазвичай пропонує додати символічне посилання на сценарій запуску MATLAB. Це добрий варіант, щоб зробити його глобально доступним, що підходить безпосередньо для Gazebo та інших unix-придатних пакетів моделювання.

У [4, 47] можна докладно дізнатись про застосування MATLAB у зв'язці з іншими пакетами моделювання роботів. На (рис.1.12) показаний приклад зв'язку MATLAB та пакетів моделювання на основі ROS [54].

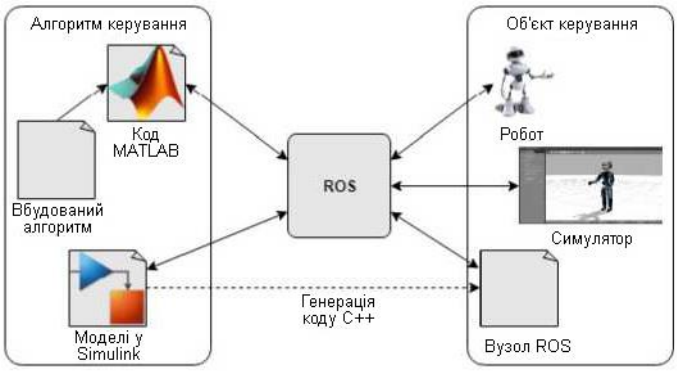

Рисунок 1.12 – Зв'язок MATLAB та пакетів моделювання на основі ROS

OpenAI

OpenAI – некомерційна дослідницька компанія із Сан-Франциско. що займається штучним інтелектом (1111) мета якої розвивати відкритий, дружній 1111 та забезпечити, щоб 1111 приносив користь всьому людству [36].

Зокрема. OpenAI Gym – інструментарій для розробки та порівняння алгоритмів навчання з підкріпленням. Він підтримує навчання агентів усьому, від прогулянок до ігор у понг чи пінбол. Бібліотека Gym – це набір тестових завдань, середовищ, які можна використовувати для розробки своїх алгоритмів навчання із підкріпленням. Ці середовища мають загальний інтерфейс, що дозволяє писати загальні алгоритми.

Алгоритм навчання з підкріпленням (Reinforcement Learning – RL) вивчає, як ІІ може навчитися досягати цілей у складному, невизначеному середовищі. Це цікаво з двох причин:

– RL є дуже загальним, що охоплює всі проблеми, які включають ухвалення послідовності рішень;

– Алгоритми RL почали досягати добрих результатів у багатьох складних умовах.

Але дослідження алгоритмів RL уповільнюються двома факторами:

– Необхідність покращення показників. У контрольованому навчанні прогрес зумовлювався великими позначеними наборами даних. У RL найближчим еквівалентом була б велика та різноманітна колекція середовищ. Однак, існуючі колекції RL-середовищ з відкритим вихідним кодом не мають достатньої різноманітності, і їх часто важко навіть налаштувати і використовувати.

– Відсутність стандартизації середовищ, які у публікаціях. Тонкі відмінності у визначенні проблеми, такі як функція винагороди або набір дій можуть кардинально змінити складність завдання. Ця проблема ускладнює відтворення опублікованих досліджень та порівняння результатів різних робіт.

Розробники в GpenAI намагаються вирішити обидві проблеми за допомогою бібліотек у Gym [17]. У [7, 56] наведено приклади використання

зв'язки OpenAI Gym, ROS та Gazebo (рис.1.13) і результати стосовно таких робіт, як наприклад: Tmtlebot, Erlo-Rover і Eric-Copter.

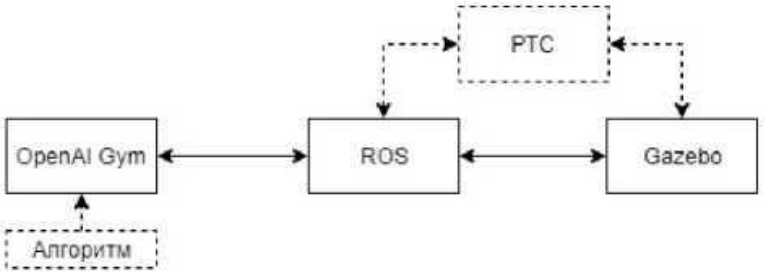

Рисунок 1.13 – Алгоритм взаємодії OpenAI Gym, ROS та Gazebo

1.4 Постановка задачі

Об'єднання різних етапів розробки робототехнічних систем у єдиний технологічний цикл трудомістке. Тому на фінальних етапах створення проекту підвищується ризик отримання помилок, неврахованих раніше. Такі помилки можуть призвести до великих матеріальних та тимчасових витрат.

Виникає потреба у виборі оптимального підходу до проектування РТС, у якому організовано паралельне виконання процесів створення конструкції, програмних засобів та апаратного забезпечення. Метою даної є зниження трудомісткості при проектуванні складних РТС із вбудованими пристроями управління шляхом розробки та комплексування інструментальних засобів моделювання робототехнічної системи та інструментальних засобів моделювання вбудованих алгоритмів систем управління.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– розробити структурні та функціональні схеми для створення моделей РТО та моделей систем управління;

– розробити додаткове програмне забезпечення;

– створити приклад готового рішення РТС;

– виконати експериментальні дослідження системи.

РОЗДІЛ 2
РОЗРОБКА СТРУКТУРНИХ І ФУНКЦІОНАЛЬНИХ СХЕМ

2.1 Вибір засобів моделювання

Порівняння різних пакетів моделювання за основними технічними характеристиками, параметрами та можливостями при спільній роботі з іншими САПР у подробицях розглядаються в багатьох роботах, наприклад [33, 11].

Зважаючи на завдання поставлені в цій роботі, основними критеріями для вибору пакета моделювання РТО стали:

– можливості фізичного та графічного двигунів;

– відкритий вихідний код із можливістю його вдосконалення;

– безкоштовне використання;

– використання популярних мов програмування;

– наявність документації та бібліотеки готових рішень з прикладами роботів, датчиків, сенсорів тощо;

– можливість роботи з різними типами роботів;

– можливість створення/редагування моделей робота;

– можливість створення/редагування динамічних сцен;

– доступна інтеграція з ROS;

– можливість взаємодії з іншими засобами САПР (можливо через ROS).

Проведені автором дослідження та аналіз (див. табл. 2.1) показали. що всім перерахованим вище критеріям відповідає симулятор Gazebo.

Як додаткові засоби САПР були обрані:

– ROS початкової версії, оскільки вона на цьому етапі надійніша – має повну підтримку всіх необхідних для роботи пакетів на відміну від ROS 2.0.

– OpenAI Gym, оскільки є просто додатковою бібліотекою, не вимагає великих апаратних ресурсів і має достатньо готових рішень як для об'єктів, так і для сцен.

Таблиця 2.1 – Порівняння пакетів моделювання

	Пакети моделювання				
Характеристики	Gazebo	V-Rep	OpenHRP	Webots	ARGos
Графічні двигуни	OGRE	власний	JavaSD	OGRE	власний
Фізичні двигуни	ODE, Bullet Simbodv. DART	ODE, Bullet Vortex, Newton	ODE та власний	власний на основі ODE	власні
Відкритий вихідний код	Так	Зд	Тільки OpenHRPS	Зд	Так
Безкоштовне ПЗ	Зд	Зд	Зд	Ні	Так
Операційні системи	Linux. MacOS (Windows)	Windows, Linux. MacOS	Windows. Linux	Windows. Linux. MacOS	Linux. MacOS
Документація	Cl	Зд	Частково	Зд	Частково
Підтримувані мови програмування	C/C++, Python	Lua (C/C++, Python, Java, Matlab, Octave)	C/C++, Java	C/C++, Python, Java, Matlab	C++
Підтримуване зв'язок} тоше ПЗ для роботів	ROS, Player, Sockets	ROS, Sockets	ROS. OpenRTMAIST	ROS, Sockets, NaoQI	ROS
Бібліотеки готових рішень	Так	Зд	Зд	Зд	Ні
Типи роботів, що підтримуються	Усі	Усі, крім водних	Усі, крім водних та повітряних	Усі	Усі
Редаговані моделі роботів	Зд	Зд	Зд	Зд	Зд
Редаговані моделі динамічних сцен	Зд	Ні	Ні	Ні	Ні

2.2 Структура засобів моделювання робототехнічних систем

Вибраний засіб для моделювання РТС, пакет Gazebo підтримує простий API для додавання об'єктів - моделей, та необхідні плагіни для взаємодії з клієнтськими програмами. Gazebo включає кілька додаткових компонентів (рис.2.1):

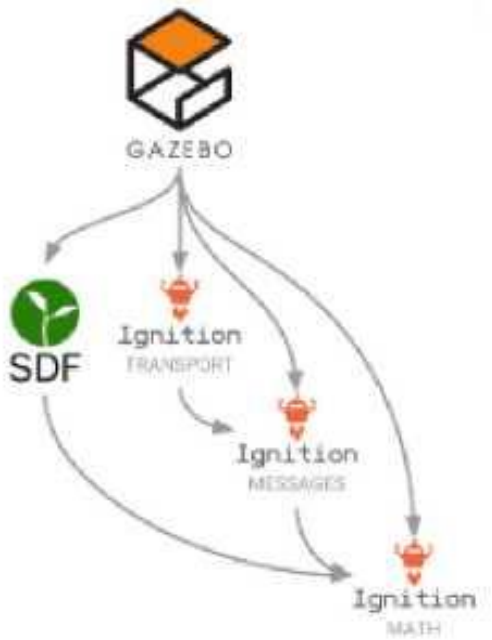

Рисунок 2.1 – Архітектура взаємодії компонентів Gazebo: SDFomiat та Ignition Robotics

– SDFormat – формат XML [45], за допомогою якого можна описати всі аспекти робота, будь то колісний робот або антропоморфний робот. На додаток до кінематичних та динамічних атрибутів, для робота можуть бути визначені датчики, властивості поверхні, текстури, тертя суглобів та багато інших властивостей. Ці функції дозволяють використовувати SDF для моделювання, візуалізації, планування руху та керування роботом. Також SDF надає засоби для визначення різних середовищ.

– Ignition Robotics – це набір бібліотек з відкритим вихідним кодом, які включають все необхідне для роботизованого моделювання [19]. Мета Ignition –

спростити моделювання та дати розробникам можливість використовувати моделювання у своїй повсякденній роботі, у системах безперервної інтеграції або навіть у програмах машинного навчання. Бібліотеки Ignition Robotics створені у зв'язку з необхідністю кардинально оновити та реструктурувати Gazebo для використання сучасних методів розробки та шаблонів проектування. ґрунтуючись на відгуках спільноти, спостережуваних потребах та цілях розробників.

Докладніша архітектура пакета моделювання показана на (рис.2.2). Вона являє собою два процеси, об'єднані між собою за допомогою бібліотек:

– gzserver – запускає фізичний цикл і генерує дані датчиків;

– gzelient – забезпечує взаємодію з користувачем та візуалізацію симуляції.

Запуск даних процесів подає запити та команди до файлів з описом моделей роботів, сенсорів, датчиків та навколишнього середовища у форматі SDF/XML: physics sim. sensor gen та client gui. Додатково можна реалізувати власні програми (plugins), необхідні для роботи системи різними мовами програмування, наприклад XML/C/Python. Всі файли з описом у свою чергу звертаються до бібліотек Ignition Robotics через API.

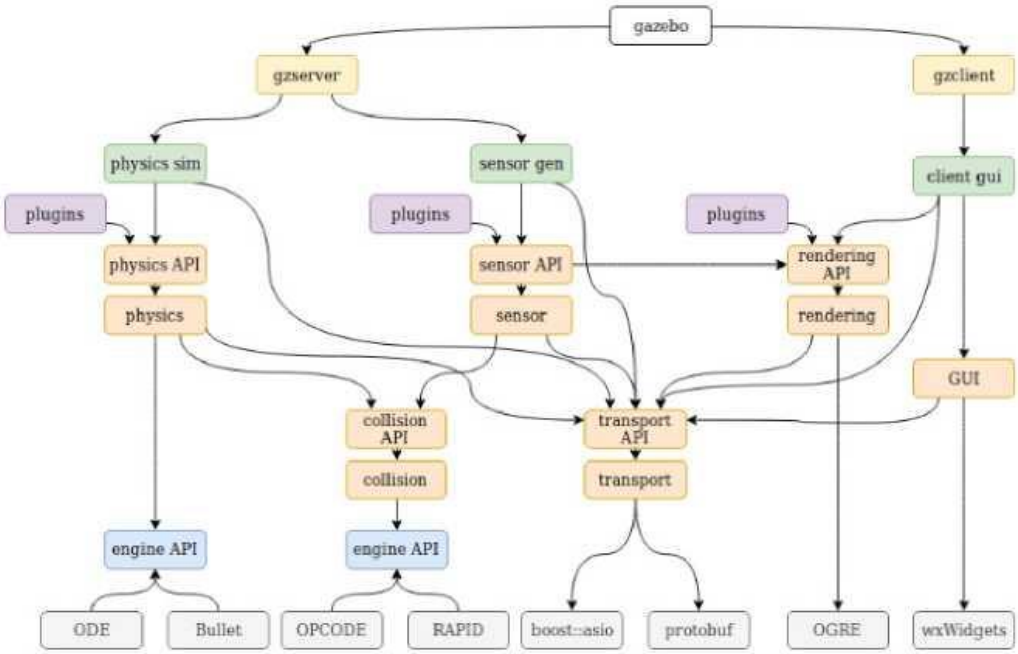

Рисунок 2.2 – Архітектура Gazebo

Окрім цього. Gazebo використовує низку сторонніх бібліотек для обліку законів фізики, візуалізації та рендерингу. Gazebo використовує ODE (Open Dynamics Engine). Bullet, Simbody та Dynamic Animation and Robotics Toolkit (DART) для моделювання динаміки та кінематики, пов'язаних із зчленованими твердими тілами. OpenGL і GLUT використовуються як інструменти візуалізації за умовчанням. Щоб забезпечити зв'язок між різними компонентами, Gazebo використовує відкритий код Google Protobuf для серіалізації повідомлень та boost:ASIG для механізму транспортування. Gazebo використовує OGRE як бібліотеку рендерингу для тривимірних сцен як у графічних, так і сенсорних бібліотеках.

У цьому пакеті моделювання створити модель необхідного об'єкта чи сцени можна кількома способами:

– за допомогою примітивів та стандартних бібліотечних моделей, доступних прямо з робочої області симулятора (Menu-> Model Editor);

– за допомогою програмного опису у форматі SDF (XML), що дає безліч можливостей, як було сказано вище;

– за допомогою додаткових програмних продуктів для ЗП-моделювання з можливістю полігонування трикутником (наприклад: Autodesk 3dsMax або Blender) створити графічну оболонку моделі і додати як об'єкт у файл у форматі SDF.

Для того, щоб перехід від симуляції РТС до реальних робіт був гнучким і швидким, не вимагав змін у програмній частині, необхідно використовувати програмне забезпечення, яке можна використовувати як на моделях, так і на реальних прототипах. Таким програмним ланкою у цій роботі обраний ROS.

Процеси в ROS називаються вузлами і кожен вузол може спілкуватися з іншими через теми. З'єднання між вузлами управляється майстром і слідує наступному процесу (рис.2.3):

– перший вузол (Node 1) повідомляє майстра (Master), що має дані для обміну (Advertising);

– другий вузол (Node 2) повідомляє майстра (Master), що хоче мати доступ до даних (Subscriber);

– з'єднання між двома вузлами (Topic) створено;

– перший вузол може надсилати дані другому (Publication->Callback).

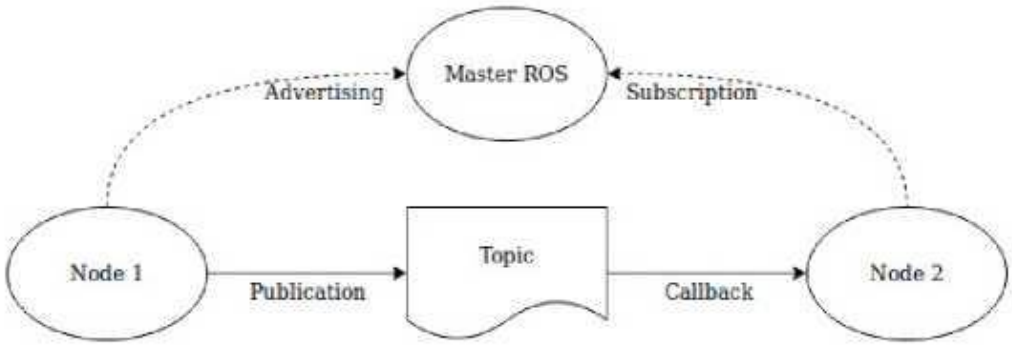

Рисунок 2.3 – Архітектура ROS

Вузол (Node) це файл пакету ROS, що виконується. Вузли ROS використовують бібліотечні бібліотеки ROS для зв'язку з іншими вузлами.

Для реалізації будь-якого вузла ROS необхідно створити його опис однією з мов програмування: Python. З .Lips. Java або Lua, які підтримують реалізацію ROS. за допомогою клієнтських бібліотек ROS (наприклад: raspy для Python і rosepp для C).

Вузли можуть публікувати повідомлення або отримувати підписавшись на тему (Topic). Вузол, що публікує дані, називається видавцем (Publisher), а вузол, який підписується на дані, називається передплатником (Subscriber). Один і той самий вузол може бути як видавцем. так і передплатником. Повідомлення, надіслані на теми (Topics), здебільшого стандартизовані, що робить систему гнучкою.

ROS дозволяє здійснювати міжмашинну взаємодію: вузли. працюють на різних машинах, але мають доступ до одного і того ж майстра, можуть безперешкодно спілкуватися один з одним і з користувачем. На (рис.2.4) показано схему взаємодії ROS з моделлюваною моделлю робота в пакеті Gazebo (ліворуч) та з реальним обладнанням (праворуч).

На схемі видно, що з використанням ROS та коректно реалізованим програмним забезпеченням можна створювати керування для моделей роботів та без додаткових програмних кодів переносити на реальне обладнання.

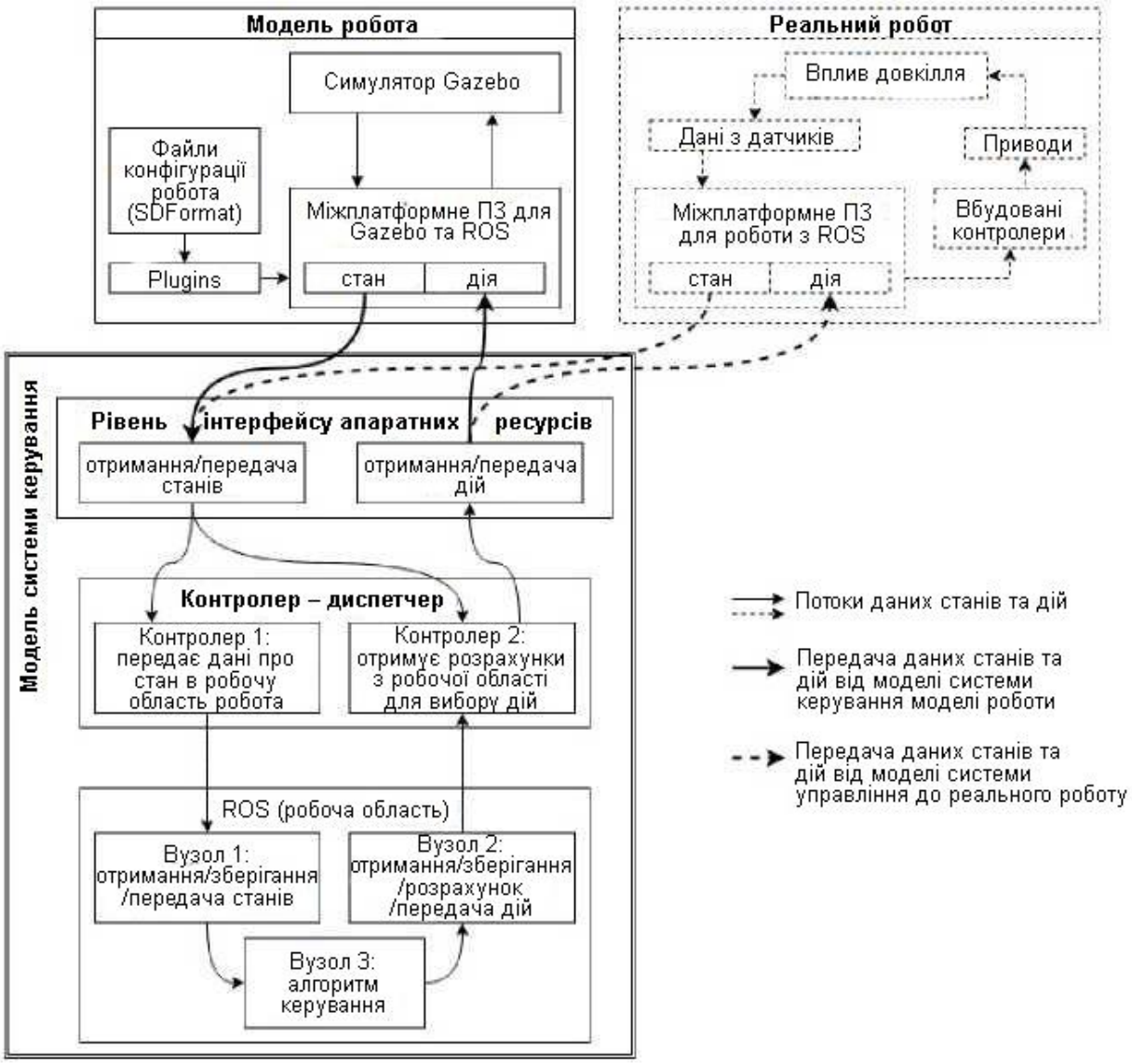

Рисунок 2.4 – Схема взаємодії ROS з Gazebo та реальним обладнанням

2.3 Структура засобів моделювання алгоритмів системи керування

OpenAI надає повний набір бібліотек Reinforcement Learning, які дозволяють навчати програмних агентів завданням, щоб агенти могли самостійно вивчити, як найкраще виконати завдання. Для реалізації системи управління РТС було обрано бібліотеку OpenAI Gym. OpenAI Gym це бібліотека

Python, яка надає API для розробки та порівняння RL-алгоритмів з величезною кількістю віртуальних середовищ. RL навчання складається із двох компонентів (рис.2.5) [25, 26]:

– агент, який вирішує, які дії повинен зробити робот:

– середовище, що є набір всіх станів, які постійно змінюються і на які намагається впливати агент при виконанні обраної дії.

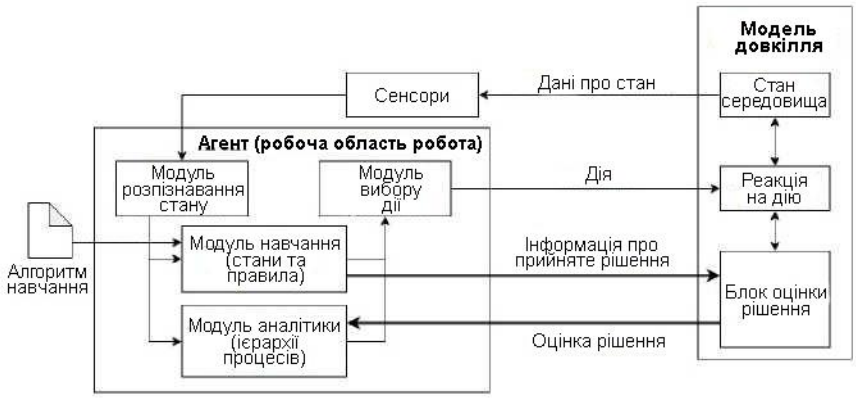

Рисунок 2.5 – Архітектура RL-алгоритму

У своєму робочому просторі агент використовує модель для ухвалення рішення про дію. Для поточного стану робота модель зіставляє можливі дії із припущеннями про те. наскільки гарною може бути кожна дія (у навчанні з підкріпленням це називається винагородою).

Спочатку модель не має уявлення про те. які дії є найкращими, та її припущення зазвичай неправильні. Коли агент навчається максимізувати потенційну винагороду, яку він може отримати, модель покращується, і її здогади про те. які дії хороші, покращуються.

Після навчання моделі RL-алгоритму створюється робоча область робота, і таку отриману модель можна розгорнути реальному роботі. Для того щоб це розгортання проходило без глобальних змін з боку програмування, а також для можливості заміни як моделі самого алгоритму, так і моделі оточення, необхідно додати в архітектуру алгоритму ROS (рис.2.6).

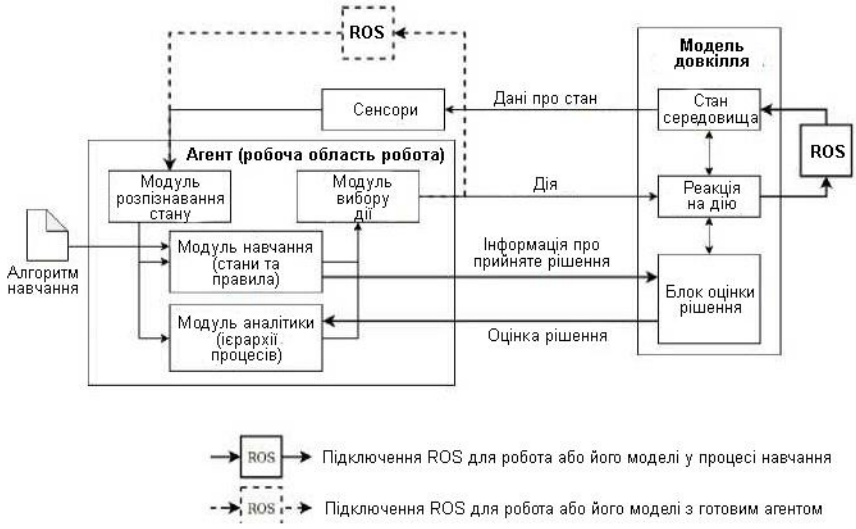

Рисунок 2.6 – Архітектура RL-алгоритму з ROS

2.4 Загальна структура розробленої системи

Для того щоб створити загальну структуру на основі окремих схем, розглянутих на (рис.2.4–2.6). необхідно визначити конкретну роль кожної складової стосовно поставленої у цій роботі задачі:

– Файл із програмним описом алгоритму навчання, роботу якого необхідно вивчити чи перевірити на моделі робота.

– OpenAI Gym – інструментарій, що використовується для розробки та проектування агента RL, який приймає автономні рішення про повороти, управління швидкістю тощо.

– ROS – середовище розробки додатків для роботів, що надає просту абстракцію для взаємодії моделі алгоритму управління та моделі робототехнічної системи.

– Gazebo – симулятор, який візуалізує стани та дії моделі робота на основі роботи алгоритму, розраховує поведінку моделі навколишнього середовища та передає дані алгоритму управління. Gazebo також може імітувати зображення з камери, що подаються агенту RL-алгоритму.

– Файл із описом моделі робота та моделі навколишнього середовища.

На основі цих компонентів отримуємо загальну структуру комплексу інструментальних засобів моделювання робототехнічних систем, що

розробляється, і вбудованих алгоритмів систем управління (рис.2.7).

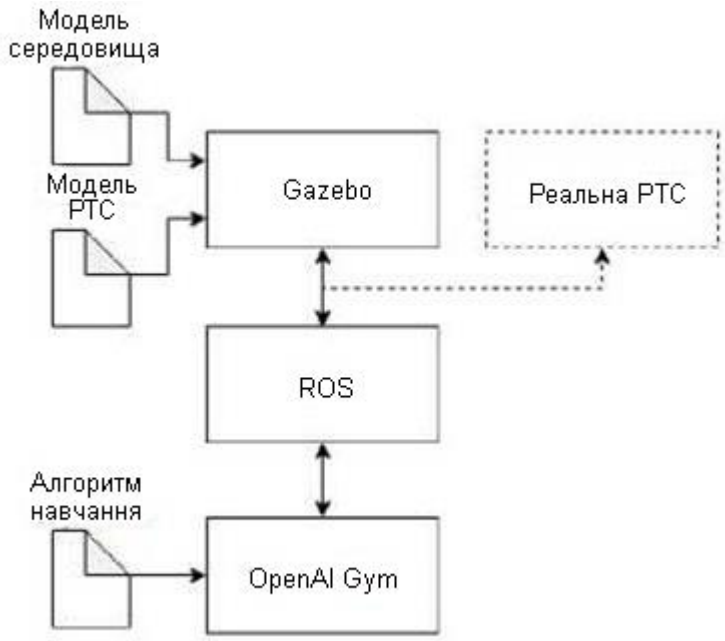

Рисунок 2.7 – Загальна структура комплексу інструментальних засобів моделювання РТС із вбудованими алгоритмами управління

Розглянемо цю схему, і навіть зв'язку між блоками докладніше (рис.2.8). Передавання даних між моделлю робота, в даному випадку, у пакеті Gazebo і моделлю системи управління можна забезпечити за допомогою пакета від розробників ROS під назвою gazebo_ros. Пакет включає готові приклади стандартних блоків, наприклад, для опису процесу передачі/отримання/редагування даних або опису сенсорів, моторів та інших динамічних компонентів, що реконфігуруються.

Для передачі даних від моделі системи управління до реального роботу існують аналогічні пакети: ros_api, rosbridge або rasbridge^suite залежно від необхідних можливостей.

Передача даних між OpenAI Gym та моделлю системи управління як для реального робота, так і для його моделі здійснюється також за рахунок спеціального пакету від розробників ROS з назвою openai^ros. Пакет надає загальну платформу для використання інфраструктури OpenAI Gym для навчання роботів за допомогою RL-алгоритмів, спрощуючи для розробника

спосіб зміни робота, завдань або алгоритмів навчання за збереження тієї ж структури

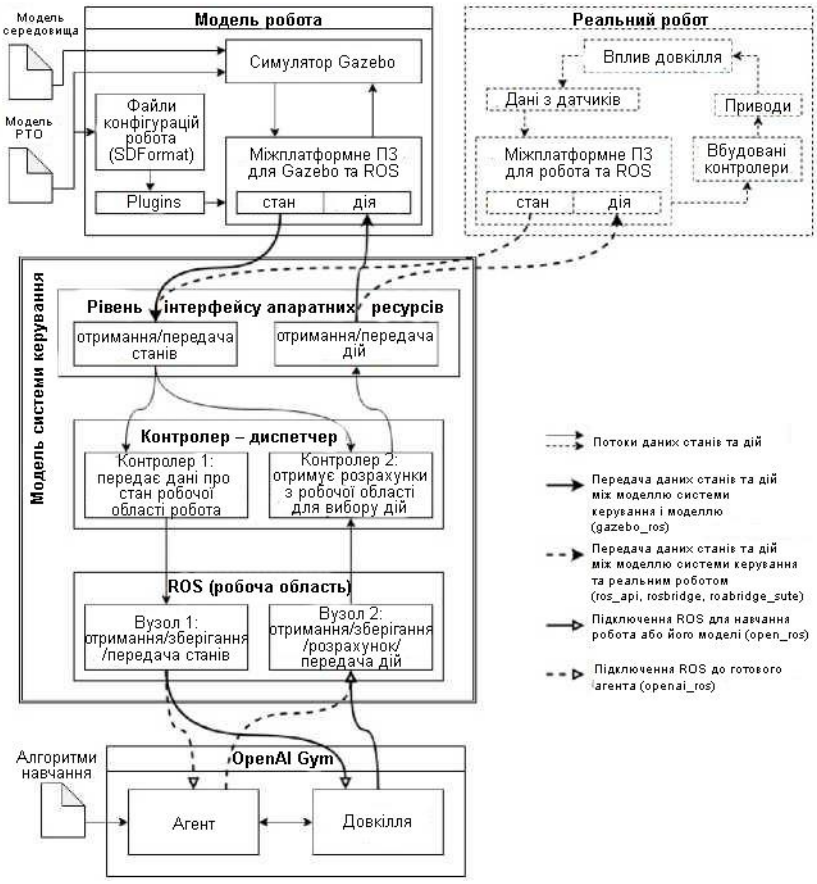

Рисунок 2.8 – Детальна структура комплексу інструментальних засобів моделювання РТС із вбудованими алгоритмами управління

Таким чином, навчання робота для виконання завдання зводиться до наступних кроків:

– вибір робота (його віртуальної моделі), який використовуватиметься;

– вибір середовища та завдання для комплексування рішення з готових варіантів або створення власних за допомогою наданих шаблонів;

– вибір робочої області робота на основі openai^ros із вже готових варіантів або створення своєї власної за допомогою наданих шаблонів;

– застосування алгоритмів навчання (може бути власною реалізацією або одним із базових у OpenAI);

– запуск РТС у середовищі моделювання чи реальних умовах.

Описана вище архітектура реалізації РТО і загалом РТС популярна серед багатьох розробників як початківців, так і давно працюючих у цьому напрямі [56]. Однак, у більшості випадків для кожної конкретної моделі робота створюється своє програмне забезпечення, індивідуально для кожної пишуться програмні коди, навіть якщо для подібної РТС вже було розроблено програмне забезпечення.

Автором даної роботи пропонується підхід, який на основі існуючої архітектури буде універсальним для однотипних моделей роботів, націлених на виконання однієї і тієї ж задачі, але мають різні параметри. Пропонується суттєво уніфікувати процес проектування від моменту отримання вхідних даних з робота (його моделі) та його навколишнього середовища (її моделі) до моменту отримання даних від системи управління про здійснення якоїсь дії. На (рис.2.9) виділено частину структури, яку пропонується зробити уніфікованою для однотипних моделей. Відповідно до такого підходу для однотипних моделей роботів на вхід системи подаватиметься сукупність параметрів, які впливають на рішення поставленої перед об'єктом управління завдання. Наприклад, це можуть бути масогабаритні показники або можливі максимальна та мінімальна швидкості об'єкта.

Далі на основі вже створеного раніше міжплатформного програмного забезпечення, моделі системи управління та готового агента або алгоритму навчання відбувається розрахунок дій та станів робототехнічної системи, які ми отримуємо на виході системи. Далі ці дані можуть аналізуватися розробниками з метою удосконалення РТС та подальших розробок.

Більше того, якщо розробник упевнений у правильності виконання завдань за допомогою розробленої системи управління і не потребує візуалізації віртуальної РТС, то він може аналізувати результати роботи агентів з різними вхідними даними, використовуючи можливості лише серверної частини Gazebo (gzserver), що знижує вимоги до ресурсів.

Таким чином, у пропонованому підході передбачається можливість реалізації та (або) перевірки розроблюваної РТС за наявності лише параметрів

робота, параметрів передбачуваного середовища його експлуатації (або їх віртуальних моделей) і раніше створеного блоку моделі системи управління з вбудованим навченим агентом. Крім цього, при такому підході забезпечується розпаралелювання процесів розробки роботів та систем управління (або їх моделей). Все це дає можливість знизити тимчасові витрати та трудомісткість розробки.

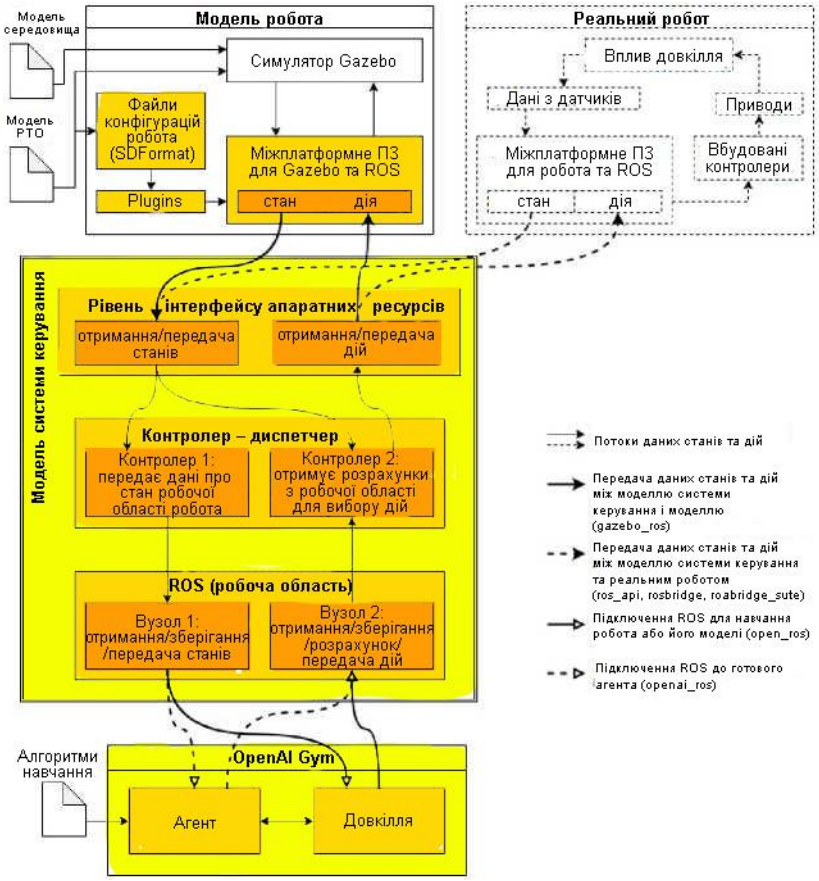

Рисунок 2.9 – Частина структури, яку пропонується уніфікувати

РОЗДІЛ 3
КОНФІГУРАЦІЯ ЗАСОБІВ МОДЕЛЮВАННЯ І ЗАСТОСУВАННЯ ПРОПОНУВАНОЇ СТРУКТУРИ НА ПРИКЛАДІ

3.1 Створення моделі об'єкта керування

Для створення моделей об'єктів керування автором цієї роботи було обрано пакет моделювання Gazebo. Цей пакет спочатку розроблений для Unix-подібних систем, відповідно має велику підтримку цих версій. Тому вся робота буде реалізовуватись на операційній системі Ubuntu 18.04 (Bionic Beaver).

Для створення моделі спочатку потрібно встановити сам пакет моделювання. Це можна зробити засобами:

– самостійно скомпілювати, слідуючи описаному посібнику від розробників, представленому на офіційному сайті [15];

– скачати готове складання з офіційного сайту;

– завантажити за допомогою менеджера пакетів Ubuntu;

– встановити повну версію пакетів ROS, яка містить у собі необхідні бібліотеки та симулятор Gazebo.

Так як у цій роботі також буде використовуватися ROS, було прийнято рішення виконати установку останнім способом. Для Ubuntu 18.04 підходить ROS Melodic. Приклад команд необхідних установки пакетів через термінал наведено на (рис.3.1).

У цьому випадку ініціалізація rosdep дозволяє встановлювати системні залежності для джерела, яке потрібно скомпілювати, і необхідний для запуску деяких основних компонентів ROS. Остання команда не є обов'язковою у тому випадку, якщо потрібне використання тільки стандартних пакетів ROS. У цьому роботі вона необхідна, оскільки планується реалізувати власні робочі компоненти.

Далі необхідно створити робочу область ROS на комп'ютері, де зберігатимуться доступні йому проекти. Для цього потрібно створити окрему

папку та підключити її до ROS (рис.3.2).

```
1  sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(
       lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.
       list'
2  sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --
       recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
3  sudo apt update
4  sudo apt install ros-melodic-desktop-full
5  apt search ros-melodic
6  sudo rosdep init
7  rosdep update
8  echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
9  source ~/.bashrc
10 sudo apt install python-rosinstall python-rosinstall-generator
       python-wstool build-essential
```

Рисунок 3.1 – Приклад програм для встановлення пакетів ROS із симулятором Gazebo

```
1  mkdir -p ~/catkin_ws/src
2  cd ~/catkin_ws/
3  catkin_make
4  source devel/setup.bash
```

Рисунок 3.2 – Команди для встановлення робочої області ROS

Перевірку наявності встановленого пакета моделювання Gazebo та його версії можна виконати через термінал, ввівши команду “gazebo” та двічі натиснути клавішу “Tab” на клавіатурі. При коррентному встановленню ROS в результаті будуть показані підказки – доступні в системі однойменні з введеної команди (рис.3.3).

Рисунок 3.3 – Перевірка доступності пакета моделювання Gazebo в системі

З рисунку видно, що у системі встановлено необхідний пакет Gazebo-9.9.0 – одна з останніх версій на момент написання цієї роботи.

При наявності в системі лише однієї версії пакета, його можна запустити через консоль за командою Gazebo, не вказуючи конкретний номер версії. Після

запуску пакета відкриється робоче вікно симулятора (рис.3.4).

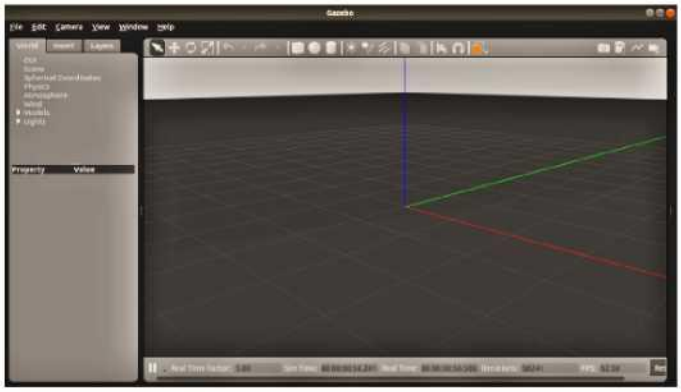

Рисунок 3.4 – Робоче вікно пакету Gazebo

Основну частину робочої області займає вікно, що демонструє віртуальний світ. Зліва знаходиться так зване дерево проекту. в якому відображено всі завантажені в цій сесії моделі, сцени і т.д. Зверху розташовується панель управління - панель швидкого доступу, де доступні всі дії, що використовуються і необхідні для роботи.

Як було зазначено раніше створити модель можна кількома способами. У цій роботі опис моделей буде реалізовуватися за допомогою файлів конфігурацій робота у форматі SDF.

Для перевірки можливості реалізації запропонованої в роботі структури САПР як приклад моделі об'єкта управління використовуватиметься простий чотириколісний робот.

Створюючи програмний опис моделі, необхідно мати уявлення про те. як вона виглядатиме. В даному випадку модель об'єкта управління – це чотириколісний мобільний робот (далі – гасесаг). зображення якого представлено (рис.3.5).

З рисунка видно, що модель складатиметься із простих елементів:

– один паралелепіпед як тіло моделі;

– чотири циліндри в якості чотирьох коліс.

Основний зміст для моделі – це файл model.config. який описує робота з деякими додатковими метаданими та файл model.sdf, який містить необхідні теги

для створення екземпляра моделі за допомогою Gazebo, пов'язаної з SDF.

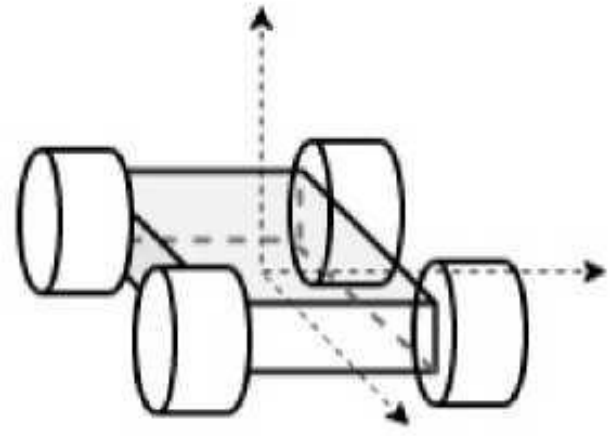

Рисунок 3.5 – Зображення майбутньої моделі об'єкта управління racecar

У папці /.gazebo/models/racecar^model створимо файл model.config, який має містити версії форматів xml та sdf та шлях до основного файлу з розширенням *.sdf. І в цій же папці створимо файл model.sdf, що містить опис окремих елементів моделі.

Кожен елемент можна описати двома варіантами:

– як елемент зіткнення, що визначає форму використання механізмом виявлення зіткнень;

– як візуальний елемент, що визначає форму використання механізмом рендеринга.

Для більшості випадків елементи зіткнення і візуальні елементи збігаються. Найбільш поширене використання для різних елементів зіткнення та візуальних елементів – мати спрощений елемент зіткнення у поєднанні з візуальним елементом, який використовує складну сітку. Це допоможе покращити продуктивність.

Також кожен елемент повинен мати унікальну ідентифікуючу ім'я-посилання, початкові позицію та положення, тип та розмір елемента.

На (рис.3.6) показано опис елемента у форматі sdf з прикладу тіла робота.

Щоб всі елементи взаємодіяли, необхідно описати зв'язок між ними. Для опису зв'язку використовується елемент joint, який повинен мати тип зв'язку,

унікальну назву, позицію та посилання на елементи, що зв'язуються. Приклад опису зв'язку між тілом гасесаге і одним колесом показаний (рис.3.7).

```
<link name='chassis'>
 <pose>0 0 0.1 0 0 0</pose>
 <collision name='collision'>
   <geometry>
     <box>
      <size>0.4 0.2 0.08</size>
     </box>
   </geometry>
 </collision>
 <visual name='visual'>
   <geometry>
     <box>
      <size>0.4 0.2 0.08</size>
     </box>
   </geometry>
  </visual>
 </link>
```

Рисунок 3.6 – Опис елемента у форматі sdf на прикладі тіла робота

```
<joint type="revolute" name="rare_left_wheel_hinge">
  <pose>0 0 -0.03 0 0 0</pose>
  <child>rare_left_wheel</child>
  <parent>chassis</parent>
  <axis>
    <xyz>0 1 0</xyz>
  </axis>
</joint>
```

Рисунок 3.7 – Опис зв'язку елементів на прикладі тіла гасесаг та колеса

Тепер можна запустити пакет моделювання Gazebo, у вкладці Insert вибрати створену модель гасесаг та подивитися, як вона виглядає (рис. 3.8). У вкладці World можна побачити всі елементи

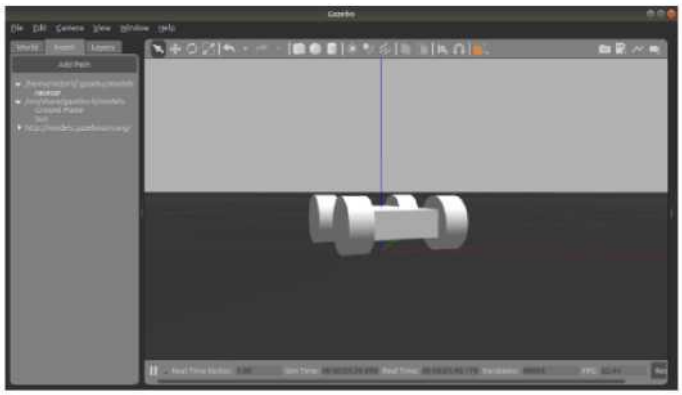

Рисунок 3.8 – Модель гасесаг у пакеті моделювання Gazebo

У більшості випадків для перевірки працездатності РТС, що розробляється, необхідно моделювати передбачуване середовище експлуатації (сцену). Її також можна реалізувати за допомогою конфігураційних файлів у форматі SDF. Для того щоб надалі сцену можна було використовувати для навчання моделі робота, створимо її у вигляді невеликого лабіринту під назвою maze (рис.3.9).

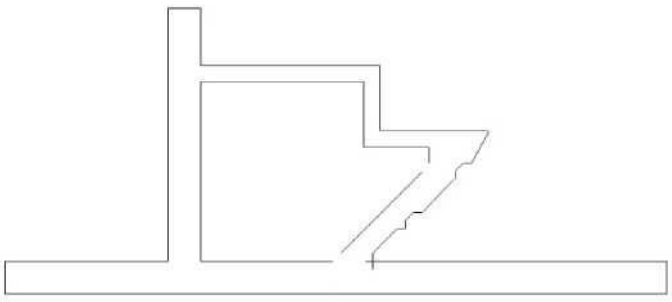

Рисунок 3.9 – Зображення передбачуваного лабіринту для сцени

Всі файли, які описують сцени, світи, навколишнє середовище для пакета моделювання Gazebo мають розширення *.world і зберігаються в окремій підпапці проекту з назвою worlds. Наповнення сцени складається з моделей об'єктів, які описуються також як було описано модель робота гасесаг.

У папці /.gazebo/models створимо окрему папку для моделі сцени під назвою maze_model, усередині якої створимо також два файли model.config і model.sdf з описом моделі лабіринту. Далі в папці /.gazebo/ створимо папку worlds із файлом racecare^maze.world і опишемо в ньому сцену, в якій буде створений раніше лабіринт. Крім моделей, які розробник хоче включити до сцени, необхідно прописати стандартні моделі: світло - "sun" та підлогу - "ground_plane" (рис.3.10). Повний текст файлів model.config, model.sdf та maze.world представлений у програмі.

```
<!— A global light source —>
    <include>
      <uri>model://sun</uri>
    </include>
<!— A ground plane —>
    <include>
      <uri>model://ground_plane</uri>
    </include>
<!— A maze model —>
    <include>
       <uri>model://maze</uri>
    </include>
```

Рисунок 3.10 – Опис сцени з лабіринтом для Gazebo

Запустити пакет моделювання Gazebo із заданим світом можна за допомогою команди, представленої на (рис.3.11).

```
gazebo /home/user/.gazebo/worlds/maze.world
```

Рисунок 3.11 – Команда для запуску певного світу у пакеті Gazebo

Додати до поточної сцени об'єкт управління, створений раніше можна, вибравши ого у вкладці Insert праворуч (рис. 3.12).

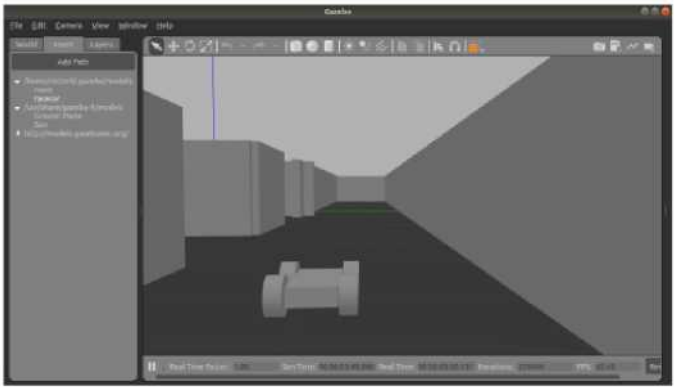

Рисунок 3.12 – Додавання моделі об'єкта керування до сцени

Оскільки пропонований автором підхід передбачає параметризовані моделі об'єктів управління, то цю конфігурацію файлів з описом моделі гасегсаг необхідно розбити на модулі - файли з розширеннями *. Gazebo і *. Хасго. У файлі racecar.gazebo необхідно описувати параметри елементів моделі, такі як коефіцієнти тертя, жорсткості та інші, що стосуються процесів взаємодії між

об'єктами, а також матеріал або колір елементів. Матеріали та кольори описуються у додатковому файлі materials, xacro.

У файлі racecar .xacro описуються залежності елементів один від одного, їх положення щодо один одного та положення у віртуальному світі, а також фізичні параметри елементів, наприклад, маса та інерція. Для реалізації параметризованої моделі необхідно створити файл macros.xacro, який містить опис характеристик елементів моделі, які планується змінювати.

Усі файли потрібно розмістити у папці .gazobo/urdf.

Таким чином, отримано параметризовану модель об'єкта управління racecar та модель сцени з його передбачуваним середовищем експлуатації – лабіринтом maze. Але в цій ситуації вони є просто моделями без будь-якої фізичної взаємодії. Така взаємодія, а також управління пропонується здійснити через ROS.

3.2 Створення моделі системи керування

Для прив'язки будь-якого проекту до ROS його необхідно визначити в робочій області catkin_ws/src, тобто створити папку з ім'ям проекту (у даному прикладі це буде також racecar_mobile_robot).

Папка проекту має відповідати декільком вимогам:

– пакет повинен містити сумісний із catkin файл package.xml. Цей файл package.xml містить метаінформацію про пакет;

– пакет має містити CMakeLists.txt, який використовує catkin. Якщо це метапакет catkin, то він повинен мати відповідний шаблон CMakeLists.txt;

– кожен пакет повинен мати власну папку. Це означає, що немає ні вкладених пакетів, ні декількох пакетів, які спільно використовують один і той же каталог.

За допомогою команд, що представлені на рис.3.13, буде створено пакет проекту з назвою car_mobile_robot, який вже міститиме package.xml та CMakeLists.txt, частково заповнені інформацією, яка була надана у параметрах

до команди catkin_create_pkg. У цьому випадку це назви трьох основних бібліотек, що часто використовуються.

```
cd ~/catkin_ws/src
catkin_create_pkg racecar_mobile_robot std_msgs rospy roscpp
cd ~/catkin_ws
catkin_make
```

Рисунок 3.13 – Команди для створення власного проекту у робочій області ROS

Найчастіше у великих проектах для файлів, що описують конфігурацію об'єктів або сцен, використовують окрему підпапку проекту. Створимо таку папку з назвою racecar^description/ і помістимо туди папки з файлами, що містять опис моделей об'єкта та лабіринту (models та urdf), а також папку з описом моделі лабіринту (рис.3.14).

```
cd ~/catkin_ws/src/racecar_mobile_robot/src
catkin_create_pkg racecar_description std_msgs rospy roscpp
mkdir racecar_description/models
mv /home/user/.gazebo/models/racecar_model/ /home/user/catkin_ws/src
     /racecar_mobile_robot/src/racecar_description/models/
mv /home/user/.gazebo/models/maze_model/ /home/user/catkin_ws/src/
     racecar_mobile_robot/src/racecar_description/models/
mv /home/user/.gazebo/urdf /home/user/catkin_ws/src/
     racecar_mobile_robot/src/racecar_description/
```

Рисунок 3.14 – Створення підпапки проекту та копіювання моделей

Файл з описом сцени maze.world також потрібно перемістити в робочу область проекту в ROS, але в окрему підпроект racecar^gazebo, файли в якому будуть ставитися безпосередньо до симулятора Gazebo. Так звані світи, що складаються зі сцен, також мають свою окрему папку worlds.

Для того, щоб запустити віртуальний світ зі сценою та моделлю одночасно та прив'язати їх до вузлів ROS, потрібно створити файли з розширенням *.launch для кожної моделі окремо та подібний загальний файл, який називають сценарієм. Таким чином, із сукупності реалізованих моделей об'єктів і сцен можна відтворювати різні сценарії їхньої взаємодії за допомогою команди roslaunch.

Інструмент roslaunch створений для запуску кількох вузлів ROS локально

та віддалено через SSH, а також для налаштування параметрів на сервері. Він включає параметри для автоматичного відновлення раніше призупинених або скасованих процесів. В якості параметрів команда roslaunch приймає один або кілька файлів конфігурації XML з розширенням *.launch, в яких вказуються параметри для встановлення та вузли для запуску, а також апаратні платформи, на яких вони повинні працювати.

Файли *.launch необхідно створювати у підпапці проекту ../rarecar^gazebo/launch. Кожен такий файл повинен включати початковий віртуальний світ і необхідні об'єкти або додатковий сцени. На рис.3.15 показаний приклад створення сцени з гасесагом у порожньому світі.

```
1 <arg name="world_name" default="racecar" />
2   <arg name="gui" default="true" />
3   <arg name="run_camera" default="false"/>
4   <include file="$(find gazebo_ros)/launch/empty_world.launch">
5   <arg name="world_name" value="$(find racecar_gazebo)/worlds/$(arg
        world_name).world"/>
6     <arg name="gui" value="$(arg gui)"/>
7   </include>
```

Рисунок 3.15 – Приклад опису файлу для запуску гасесаг у порожньому світі

Далі слід повторити складання (компіляцію) робочої області catkin _ws і при її успішному завершенні додати шляхи до всіх зібраних проектів в ROS за допомогою команди, показаної на рис.3.16. Цю команду необхідно прописати у системному файлі /.bashrc, щоб при відкритті нового терміналу всі скомпільовані проекти були доступні відразу.

```
1 . ~/catkin_ws/devel/setup.bash
```

Рисунок 3.16 – Команда для ініціалізації всіх проектів у галузі ROS

Тепер можна запустити весь проект із візуалізацією у пакеті моделювання Gazebo, використовуючи два термінали.

У першому терміналі необхідно ввести команду “roscore”, яка запустить набір вузлів та програм, що є попередніми умовами системи взаємодії вузлів

ROS.

У другому терміналі необхідно запустити проект через команду roslaunch і як параметри вказати потрібний проект, що містить файли з описом світів і сцен, і файл з розширенням *.launch (рис.3.17).

```
1 roslaunch racecar_gazebo racecar_maze.launch
```

Рисунок 3.17 – Команда для ініціалізації всіх проектів у галузі ROS

В результаті виконання останньої команди має відкритися робоча область пакету моделювання Gazebo з такою самою сценою, як на рис. 3.12.

Для того, щоб зі створеною моделлю об'єкта можна було взаємодіяти, необхідно додати пристрої зворотного зв'язку для отримання даних про навколишнє середовище та контролери для керування.

Як пристрої зворотного зв'язку будуть виступати камера і далекомір, які мають стандартні драйвера, описані в gazebo_ros_plugins. Для реалізації контролерів також можна використовувати відповідні стандартні варіанти, представлені в бібліотеці готових рішень gazebo_ros_plugins, які можна вносити додаткові зміни за необхідності.

Опис усіх додаткових пристроїв об'єкта керування та контролерів повинен знаходитись у тих же файлах, що й опис моделі в папці ../racocar_doscription/urdf/ (див. Додаток). Після того, як модель об'єкта повністю обладнана пристроями зворотного зв'язку та контролерами, можна знову скомпілювати проект та перевірити його (рис. 3.18).

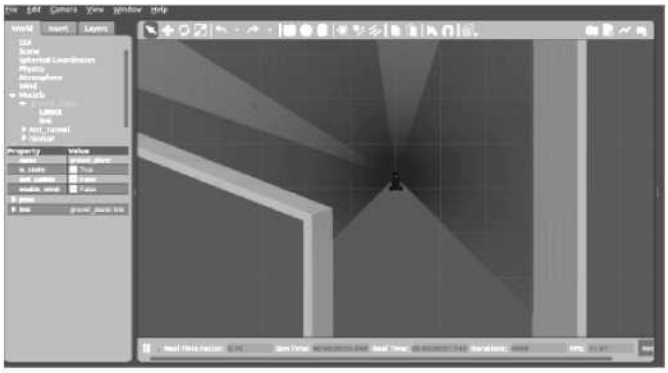

Рисунок 3.18 – Удосконалена модель об'єкта управління у пакеті Gazebo

Управління моделі об'єкта також описується за допомогою файлів – плагінів, які можуть бути реалізовані різними мовами програмування. У цій роботі ці файли будуть реалізовані мовою Python.

Модель системи управління розташовуватиметься в окремій підпапці проекту racocar_control/, яка в свою чергу розділятиметься на папки:

через загальний сценарій віртуального світу;

– папкаconfig/, в якій зберігається файл з параметрами для початкової конфігурації елементів керування:

– папка launch/, в якій зберігаються файли *.launch з описом контролерів системи керування для підключення до моделі

– папка scripts/, в якій зберігаються плагіни з описом можливостей контролерів, описом процесу отримання ними даних ззовні та описом їхньої реакції на будь-які дії з боку моделей об'єктів та сцен.

Опис необхідних контролерів у файлі racecar_control.launch для моделі об'єкта гасесаг та завантаження їх початкової конфігурації наведено на рис.3.19. У разі необхідно описати чотири приводи контролю швидкості – по одному на кожне колесо.

```
<!-- Load joint controller configurations from YAML file to
     parameter server -->
<rosparam file="$(find racecar_control)/config/racecar_control.yaml"
      command="load"/>
<!-- load the controllers -->
<node name="controller_manager" pkg="controller_manager" type="
     spawner" respawn="false"
output="screen" ns="/racecar" args="
     left_rear_wheel_velocity_controller
right_rear_wheel_velocity_controller
left_front_wheel_velocity_controller
     right_front_wheel_velocity_controller
left_steering_hinge_position_controller
right_steering_hinge_position_controller
                  joint_state_controller"/>
```

Рисунок 3.19 – Опис контролерів для коліс моделі racecar

У випадку, якщо необхідно реалізувати нові функції для стандартних пристроїв зворотного зв'язку або їх взаємодію з моделями сцени, створюються додаткові файли *.launch.

Для перевірки коректної реалізації моделі об'єкта управління та моделі системи управління з погляду їх взаємодії створимо варіант ручного управління для racecar. Суть даного управління у тому, що з отриманні сигналу з пульта управління, модель racecar буде переміщатися в потрібному напрямку. Як пульт управління буде виступати клавіатура комп'ютера (клавіші: w – вперед, s – назад, a – праворуч і d – праворуч). На рис.3.20 наведено приклад реалізації процесу прийняття сигналів з клавіатури.

```
keyBindings = {
  'w':(1,0),
  'd':(1,-1),
  'a':(1,1),
  's':(-1,0),
}
def getKey():
    tty.setraw(sys.stdin.fileno())
    select.select([sys.stdin], [], [], 0)
    key = sys.stdin.read(1)
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key
speed = 0.5
turn = 0.25
```

Рисунок 3.20 – Приклад опису процесу прийняття сигналів із клавіатури

Для цього потрібно створити два файли:

– keyboar_teleop.py, який приймає інформацію про сигнал, що надходить ззовні (натискання клавіші), визначає його і відповідну йому дію і передає команду на головний контролер моделі, тобто реалізує свого роду алгоритм

управління;

– servo_commands.py, який приймає команду та реалізує її на моделі об'єкта, розподіляючи за конкретними контролерами моделі (в даному випадку – за приводами коліс).

Дані файли необхідно підключити до сценарію роботи контролерів моделі об'єкта управління – файлу racecar_control.launch. Його, у свою чергу, необхідно підключити до загального сценарію роботи моделі об'єкта racecar_tunnel.launch. Після цього за допомогою команди roslaunch можна запустити останній файл і провести роботу реалізованої структури (рис.3.21).

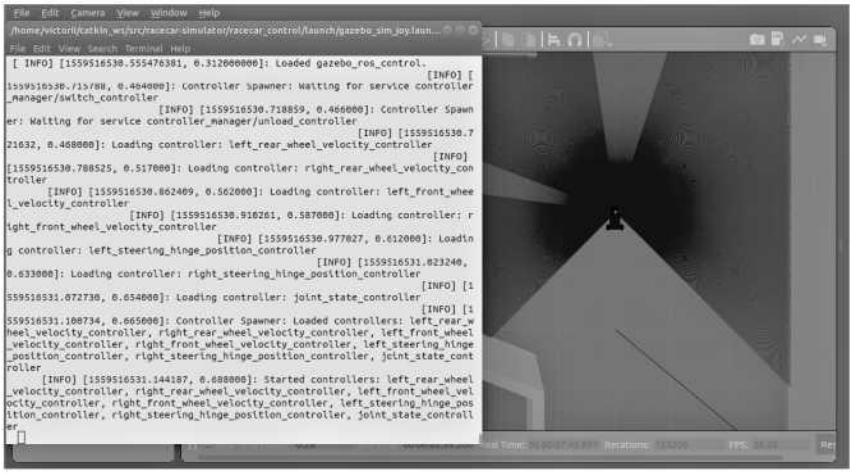

Рисунок 3.21 – Управління моделлю за допомогою клавіатури через ROS з візуалізацією"! у пакеті моделювання Gazebo

У результаті одержуємо модель робототехнічної системи, що складається з параметризованої моделі об'єкта управління та вбудованої моделі системи управління на основі ROS. згідно з структурою поданої в і. 2.2 на рис. 2.4.

3.3 Створення агенту на основі алгоритму керування

Керування об'єктами не завжди можливо за допомогою клавіатури. джойстика або іншого зовнішнього пристрою, доступного розробнику. Найбільший інтерес викликає реалізація автономного управління об'єкта з

можливістю своєчасного прийняття рішень та виконання поставлених завдань. Іншими словами, об'єкт повинен мати систему управління з вбудованим алгоритмом на основі бази знань.

Для реалізації моделі вбудованого алгоритму керування на основі набутих знань було обрано бібліотеку OpenAI Gym. Ця бібліотека надає створювати робочу область для роботів з урахуванням алгоритмів навчання з підкріпленням (RL-алгоритмів).

Навчання системи у цій роботі буде грунтуватися на інтеграції нечіткої логіки (fuzzy logic) з RL-алгоритмами. За допомогою fuzzy logic будуть описані стани та правила, на основі яких RL-алгоритм прийматиме рішення та навчатись [23]. Таке рішення зумовлене тим. що:

– нечітка логіка вводить простий, заснований на правилах виду IF X AND Y THEN Z підхід до вирішення проблеми управління замість спроб змоделювати систему математично:

– взаємодія двох методів може ефективно вирішити завдання навігації в складному невідомому середовищі, що змінюється.

На рис.3.22 показано структуру взаємодії двох методів.

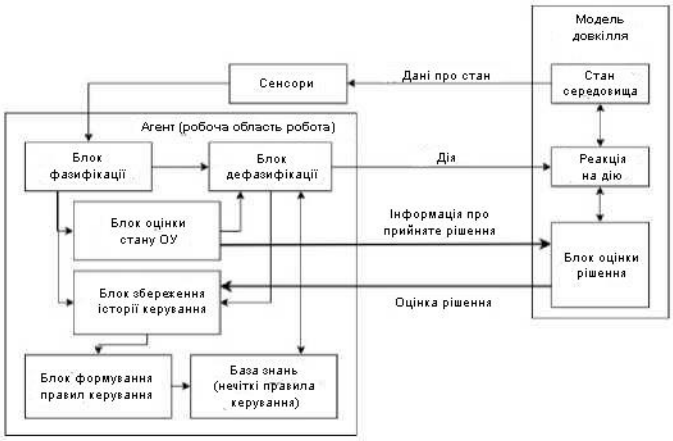

Рисунок 3.22 – Структура взаємодії fuzzy logic та RL-алгоритму

На рис.3.23 показаний приклад завдання правил у вигляді термів для швидкості обертання контролера колеса моделі гасесаг.

Рисунок 3.23 – Приклад завдання правил

Програмний опис правил і станів на основі нечіткої логіки можна реалізувати із застосуванням спеціальних бібліотек Python: Fuzzy або skfuzzy. На рис.3.24 показаний приклад опису термів для змінної. Додаток містить повний зміст файлу.

```
1  def tfm_generator(self, xmin, xmax):
2          x = (xmax - xmin)/2
3          NB = np.array([xmin, xmin, xmin+1/3*x], dtype = np.float)
4          NM = np.array([xmin, xmin+1/3*x, xmin+2/3*x], dtype = np.
               float)
5          NS = np.array([xmin+1/3*x, xmin+2/3*x, xmin+x], dtype = np.
               float)
6          ZE = np.array([xmin+2/3*x, xmin+x, xmax - 2/3*x], dtype = np
               .float)
7          PS = np.array([xmin+x, xmax-2/3*x, xmax-x/3], dtype = np.
               float)
8          PM = np.array([xmax-2/3*x, xmax-x/3, xmax], dtype = np.float
               )
9          PB = np.array([xmax - 1/3*x, xmax, xmax], dtype = np.float)
10         return [NB, NM, NS, ZE, PS, PM, PB]
```

Рисунок 3.24 – Приклад програмного опису термів змінної

Для параметризованої системи правила та стану повинні розраховуватися з параметрів, одержуваних від моделі об'єкта управління за його первинної ініціалізації.

Описав всі необхідні параметри через терми станів і описавши їм правила, необхідно створити робочу область для моделі об'єкта, у якій навчатиметься новий агент RL-алгоритму.

Для початку потрібно створити новий проект у робочій області ROS з назвою neuroracer, в якому буде розташована папка neuroracer_gym з файлами алгоритму навчання, файлом із початковими значеннями системи для ініціалізації бібліотеки openai_ros, файлами запуску сценаріїв навчання *.launch

та моделями навколишнього середовища.

Файли алгоритму навчання у разі – файли з описом правил і станів з урахуванням нечіткої логіки.

У файлі з описом моделі довкілля необхідно проводити ініціалізацію та перевірку взаємодії всіх контролерів, сенсорів та інших пристроїв, що взаємодіють зі світом (рис.3.25).

Файл сценарію навчання start.launch повинен включати файл з початковою конфігурацією системи, модель світу для навчання і назва майбутнього агенту. Для якісного навчання моделі об'єкта необхідно реалізувати достатню (на розсуд розробника) кількість сценаріїв навчання.

```
1 self._check_all_sensors_ready()
2 self._init_camera()
3 self.laser_subscription = rospy.Subscriber("/scan", LaserScan, self.
      _laser_scan_callback)
```

Рисунок 3.25 – Ініціалізація пристроїв зворотного зв'язку в моделі світу

Процес навчання займає певну кількість часу, що залежить від бажаних результатів, числа поставлених для навчання завдань та варіантів віртуальних сцен. На рис.3.26 показаний приклад графіків оцінки прийняття рішень агентом при проходженні навчання та його автономному тестуванні на моделі гасесаг у пакеті Gazebo.

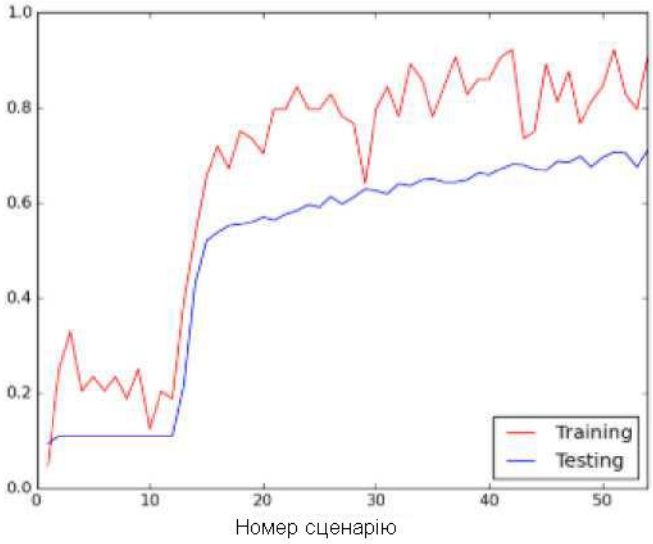

Рисунок 3.26 – Графіки оцінки прийняття рішень агентом під час навчання та при автономній роботі

З графіків видно, що з навчанні оцінка прийняття рішень має високі, але розкидані показники. У той час, як при виконанні автономної роботи агент працює з менш високою, але більш стабільною оцінкою.

Це пов'язано з тим. що в момент навчання алгоритм випробовує всілякі допустимі варіанти розвитку подій: від найкращого результату до найгіршого, таким чином створюючи собі робочу область із накопиченою бібліотекою знань.

Таким чином, на реальному прикладі було реалізовано запропоновану параметризовану структуру комплексного програмного забезпечення для однотипних моделей робототехнічних систем.

3.4 Експериментальні дослідженнярозробки

У цій роботі як складніший приклад об'єкта управління щодо досліджень пропонується розглянути модель автономного безлюдного підводного апарату (АНПА). Це з тим. що його середовище експлуатації динамічна, нестабільна і непередбачувана, і розв'язання завдань стабілізації управління у такій ситуації є цікавим дослідженням.

3.5 Реалізація моделі об'єкта управління та середовища його експлуатації

Найчастіше АНПА конструюються за модульною технологією, тобто складаються з різних взаємозамінних елементів (рис.3.27). Відповідно. такий об'єкт потребує параметризованого опису, щоб зберігати вбудований комплекс програмного забезпечення без серйозних змін.

Рисунок 3.27 – Модель АНПА

Для реалізації моделі АНПА так само, як і в попередньому прикладі, було створено опис моделі для пакета моделювання Gazebo, опис датчиків, опис контролерів для управління та плагінів для їх взаємодії. Віртуальний світ експлуатації даної моделі об'єкта – це модель водного простору (озеро, море, океан). Вода має фізичні властивості, відмінні від повітряного простору стандартно реалізованого в пакеті моделювання. Тому модель світу складається не тільки з компонентів, що описують візуальне уявлення світу, але й компонентів з описом взаємодій товщі води з об'єктами всередині неї та динамічних змін у просторі. На рис. 3.28 показано модель об'єкта серед моделювання.

Рисунок 3.28 – Модель АНПА у віртуальному світі, реалізовані у пакеті моделювання Gazebo

Далі необхідно створити модель ручного управління контролерами моделі,

щоб перевірити коректність взаємодії всіх реалізованих компонентів. На рис. 3.29 показаний приклад запущеної симуляції сценарію та паралельна зміна параметрів контролерів системи двигунів. Як видно з рисунка, при зміні швидкості обертання двигуна та зміні положення лопатей через панель ручної зміни параметрів об'єкт змінює свій стан у просторі.

3.6 Реалізація моделі системи управління із вбудованим алгоритмом автономного прийняття рішень

Як і в попередньому навчальному прикладі вбудований алгоритм управління буде заснований на зв'язці нечіткої логіки та алгоритму навчання з підкріпленням, тільки в даному випадку розрахунок станів, правил та дій має ґрунтуватися на уявленні тривимірної моделі світу. Тому що в даному випадку об'єкт управління може змінювати своє положення у просторі за трьома координатами.

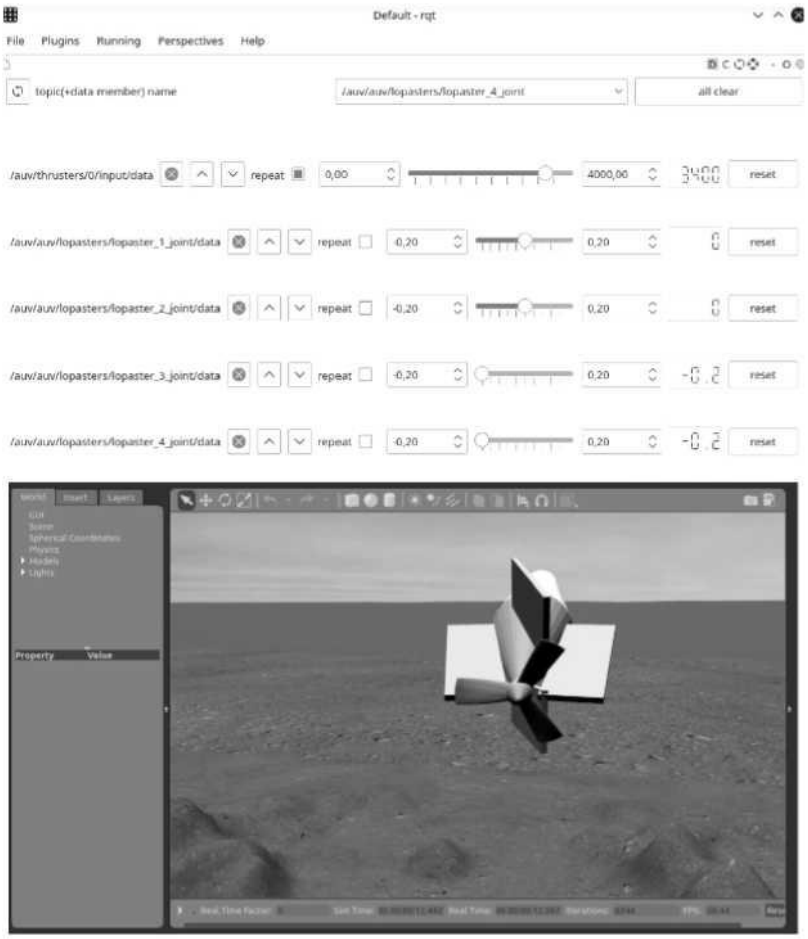

Рисунок 3.29 – Модель ручного керування контролерами моделі АНПА

Однак завдання тривимірного планування траєкторії може бути розбита на дві незалежні задачі планування горизонтальної та вертикальної площинах відповідно. У кожній площині способи обходу перешкод та пошуку цілей призначені для вирішення 2В-планування шляху, як це було реалізовано на прикладі п.

А виходи цих двох поведінки об'єднуються через додатковий контролер нечіткої логіки ваги для генерації остаточної команди руху [27].

Таким чином, можна створити модель системи управління аналогічно до попереднього варіанту, тільки в двох площинах. На рис. 3.30 показаний процес навчання моделі АНПА досягненню заданої координати простору з виведенням інформації про прийняте рішення у консолі.

Рисунок 3.30 – Процес навчання моделі АНПА досягненню заданої мети

Реалізована модель об'єкта управління з вбудованою моделлю навченого агента як робоча область системи автономного управління є прикладом роботи, яка може стати як навчальним проектом, так і початковим етапом вдосконалення інших завдань у різних напрямках.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ

Охорона праці – це соціально технічна дисципліна, що включає в себе: виробничу санітарію, техніку безпеки, пожежо- і вибухобезпеку, законодавство по охороні праці.

Основним законодавчим актом по охороні праці є Закон України «Про охорону праці», прийнятий Верховною Радою 14.10.1992 р., що складаються з восьми розділів:

– загальні положення;
– гарантії прав громадян на охорону праці;
– організація охорони праці;
– стимулювання охорони праці;
– державні міжгалузеві і галузеві нормативні акти про охорону праці;
– державне керування по охороні праці;
– державний нагляд і суспільний контроль над охороною праці;
– відповідальність працівників за порушення законодавства про охорону праці.

Безпечні умови праці повинні бути створені на базі сучасних досягнень наукиі техніки та підвищення культури виробництва. Трудове законодавство ставить за обов’язок адміністрації підприємств проводити інструктаж і регулярне навчання робітників безпечному виконанню робіт. На підприємствах чисельністю понад 50 людей створюються служби по охороні праці, в інших випадках функції цієї служби можуть виконувати особи, які пройшли перевірку знань по охороні праці. Підвищення безпеки праці приводить до зниження травматизму, і в такий спосіб до зниження витрат на виплату компенсацій за випадки травматизму і професійних захворювань, викликаних несприятливими умовами праці. Важливе місце в поліпшені умов праці займає санітарно-гігієнічний благоустрій промислових підприємств.

4.1 Аналіз шкідливих і небезпечних виробничих факторів, характерних при роботі електричного двигуна

Небезпечних виробничим фактором називається такий виробничий фактор, вплив якого на працюючого, у певних умовах приведе до травми або до іншого раптового погіршення здоров’я.

Шкідливим виробничим фактором називається такий виробничий фактор, вплив якого на працюючого, у певних умовах, приводить до захворювання і зниження працездатності.

Нещасний випадок – це випадок впливу на людину шкідливого виробничого фактору.

Професійне захворювання – це випадок впливу на людину шкідливого виробничого фактору.

Аналіз небезпечних і шкідливих виробничих факторів має на меті обґрунтувати конструктивні зміни в обладнанні, пристосуваннях, інструменті на робочому місці, удосконалювання організаційних і захисних заходів.

Найголовнішим небезпечним виробничим фактором , характерним як для всіх електроустановок, так і для електроприводів зокрема, незалежно від потужності і живлячої напруги, є ураження електричним струмом. Із загального числа нещасних випадків зі смертельним результатом основне місце займають нещасні випадки, що відбуваються в результаті ураження електричним струмом. Тому питанням електробезпеки необхідно приділяти велику увагу.

Основними причинами ураження електричним струмом при роботі електродвигуна є:

– випадковий дотик або наближення до струмоведучих частин, перебувають під напругою;

– поява напруги на металевих конструктивних частинах двигуна – корпусі, валу і т.д.;

– поява напруги на відключених струмоведучих частинах, на яких працюють люди, внаслідок помилкового включення двигуна.

Проходячи через організм, електричний струм виконує термічну, електролітичну і біологічну дії. Все це різноманіття дій електричного струму приводить до двох видів ураження: електричним травмам і електричним ударам.

Електричні травми – це чітко виражені місцеві ушкодження тканин організму, викликані впливом електричного струму і електричної дуги. Розрізняють наступні види електричних травм: електричний опік, електричні знаки, металізація шкіри і механічні ушкодження.

Електричний удар – це порушення живих тканин організму електричних струмом, що проходить через нього і супроводжується мимовільними судорожними скороченнями м’язів.

При роботі електродвигуна виникає електромагнітне поле (ЕМП), що не виявляться органами почуттів людини.

Ступінь впливу ЕМП на організм людини залежить від напруженості електричного і магнітного полів, щільності потоку енергії, тривалості впливу, частоти коливань, індивідуальних особливостей організму. Найпоширенішим засобом захисту від впливу ЕМП є екрани. Для зниження щільності потоку енергії використовують спеціальні поглиначі.

До шкідливих виробничих факторів, характерних для електродвигунів, відносяться шум і вібрація. Питання боротьби із шумом і вібрації мають велике значення у всіх областях механіки, особливо в машинобудуванні, на транспорті та в енергетиці.

В результаті дослідження із впливу шуму на організм людини встановлено, що в умовах шуму значно вповільнюється реакція, послабляється увага, швидко наступає втома. Ступінь шкідливого впливу шуму враховується при нормуванні, головним чином залежить від рівнів звукового тиску, частотного спектру шуму і тривалості впливу. Основні напрямки робіт зі зниження шуму: зниження шуму джерел, ослаблення коливальної енергії.

За результатами аналізу вібрації було відзначено, що при цьому порушуються функції центральної нервової і серцево-судинної систем, опорно-рухового апарата. Тривалий вплив вібрацій приводить до розвитку, так званної вібраційної хвороби, що може привести до часткової або повної втрати працездатності. Загальну вібрацію, що впливає на людину, можна знизити зменшенням рівня вібрації джерел, за рахунок зрівноважування обертових або мас , що рухаються. Застосовують також віброізоляцію джерела робочого місця.

4.2 Розрахунки захисного заземлення для проведення експериментів в лабораторії

Метою розрахунків захисного заземлення є знаходження заземлюючого пристрою з такими параметрами, щоб його опір розтікання був менше або дорівнював припустимому значенню.

Заземлюючий пристрій – це сукупність заземлювача – металевих привідників, що перебувають у безпосередньому контакті із землею.

Нижче приведено приклад розрахунку заземлюючого пристрою для електроустановки з номінальною напругою 220В.

Найбільш припустимий опір заземлюючого пристрою для електроустановки з напругою до 1000В $R_Д$ = 4 ОМ.

Питомий електричний опір ґрунту, у випадку наприклад для глини, ρ = 60Ом*м.

Вибираємо трубчастий або стрижневий заземлювач, з розмірами:

– довжина труби L = 2,5 м;

– зовнішній діаметр d = 0,07 м;

– глибина закладання труби, рівна відстані від поверхні землі до середини труби t = 2,05 м;

– глибина закладання труби від поверхні землі t_0 = 0,8 м.

Визначимо опір розтікання одного заземлювача:

$$R_1 = \frac{\rho}{2\pi \cdot L} \cdot \ln\left(\frac{2L}{d}\right) + \frac{1}{2}\ln\left(\frac{4t+L}{4t-L}\right),$$

$$R_1 = \frac{60}{2 \cdot 3.14 \cdot 2.5} \ln\left(\frac{2 \cdot 2.5}{0.07}\right) + \frac{1}{2}\ln\left(\frac{4*2.05+2.5}{4*2.05-2.5}\right) = 17.5 \text{ Ом.}$$

Оскільки опір заземлювача більше припустимого опору заземлюючого пристрою ($R_1 > R_Д$), то необхідно застосувати декілька заземлювачів з'єднаних паралельно.

Необхідне число паралельно з'єднаних заземлювачів розраховуєть за формулою:

$$n' = \frac{R_1}{\eta_В \eta_Д} = \frac{17{,}5}{4 \cdot 1} \approx 4 \text{ шт.}$$

де $\eta_В = 1$ – коефіцієнт використання заземлювачів, що враховує їхній взаємний вплив.

Фактичний коефіцієнт використання заземлювачів $\eta_Ф = 0{,}78$, при розміщенні заземлювачів по контуру і відстані між ними, рівному $a = 2L$.

Визначаємо фактичний опір вертикальних заземлювачів:

$$R_{ВФ} = \frac{R_1}{\eta_В \eta_Ф} = \frac{17{,}5}{4 \cdot 0{,}78} = 5{,}6 \text{ Ом.}$$

Для зв'язку вертикальних електродів і в якості самостійного горизонтального електрода використовуємо сталь круглого перерізу діаметром d = 0,01 м.

Довжина смуги визначається за наступною формулою:

$$L = 1{,}05 \cdot a \cdot n = 1{,}05 \cdot 5 \cdot 4 = 21 \text{ м.}$$

Глибину закладання смуги приймаємо рівній глибині закладання труби від поверхні $t=t_0=0,8$ м.

Опір розтікання сполучної смуги визначаємо за формулою:

$$R_{1П} = \frac{\rho}{2\pi L}\ln\left(\frac{L^2}{d \cdot t}\right) = \frac{60}{2\pi 21}\ln\left(\frac{21^2}{0,01 \cdot 0,8}\right) = 4,97 \text{ Ом.}$$

Коефіцієнт використання смуги $\eta_Г = 0,55$.

Опір сполучної смуги визначаємо за формулою:

$$R_П = \frac{R_{1П}}{\eta_Г} = \frac{4,97}{0,55} = 9,04 \text{ Ом.}$$

Фактичний опір заземлюючого пристрою, що складається з вертикального заземлювача і сполучної смуги, визначається як еквівалентний опір при паралельному з'єднані:

$$R_E = \frac{R_{ВФ} \cdot R_П}{R_{ВФ} + R_П} = \frac{5,6 \cdot 9,04}{5,6 + 9,04} = 3,46 \text{ Ом.}$$

В результаті ми отримали заземлюючий пристрій, що задовольняє умові $R_E \le R_Д$ (3,46Ом < 4 Ом).

4.3 Розробка заходів щодо забезпечення електробезпеки під час проведення експериментів з електричним двигуном в лабораторії

При розробці та тестуванні електричного двигуна завжди існує небезпека випадкового дотику до струмоведучих частин, тому слід дотримуватись правил техніки безпеки в лабораторії.

Застосування ізольованих проводів вже забезпечує достатній рівень захисту від напруги при дотику до них. Живлення до електродвигуна має підводитись спеціальним кабелем з міцною ізоляцією, кабель повинен проходити у недоступному місці або в кожусі, щоб виключити його пошкодження.

Для запобігання замикання на землю і інших пошкоджень ізоляції, при яких виникає небезпека ураження електричним струмом і виходу двигуна з ладу необхідно проводити профілактичні випробування і контроль опору ізоляції. При експлуатаційному контролі ізоляції проводиться вимір опору ізоляції при прийманні двигуна, після монтажу, періодично в строки або при виявлені дефектів.

Захисне відключення забезпечує автоматичне відключення установки при виникненні в ній небезпеки поразки електричним струмом. Захисне відключення буде здійснюватися при замиканні на землю і при несправностях пристрою захисного відключення.

Захисними засобами є:

– електровимірювальні кліщі і покажчики напруги;

– діелектричні гумові вироби і ізолюючі підставки;

– монтажний інструмент з ізольованими рукоятками;

– попереджувальні вивіски («Не вмикати працюють люди»), які вивішуються на відключених рубильниках для запобігання включенню установки.

Для забезпечення електробезпеки двигуна під час проведення експериментів в лабораторії застосовуються наступні технічні захисні заходи:

– дифиринційне реле (захист від випадкового дотику до струмоведучих частин);

– візуальний огляд (контроль і профілактика пошкоджень ізоляції);

– металеві заземлювачі (захисне заземлення);

– захисний автомат (захисне відключення);

– застосування електрозахистного електроінструменту і запобіжних пристосувань (діелектричні рукавиці, пасатижі, електричні запобіжники).

РОЗДІЛ 5
ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Сучасний стан розвитку наукової і технічної освіти суспільства дає змогу людству створювати такі машини та механізми різної складності, що реально приймають участь у вирішенні багатьох питань в процесі виробництва і споживання матеріальних благ, а також у дослідницькій діяльності, пов'язаних з розвідкою корисних копалин Світового океану, його освоєнням та знаходженнями, ідентифікацією затонулих об'єктів, що може передбачити їхнє підняття на поверхню.

Весь процес використання складних технічних систем людиною призводить до певного впливу на навколишнє середовище, а саме на гідросферу, літосферу і атмосферу. Цей вплив як правило є негативним і рівень цього негативу та запобігання йому вивчається наукою про охорону навколишнього середовища – одним з розділів екології, як загальної науки про взаємовідносини живих організмів зі всіма сферами природного походження та між собою. Наведемо поняття, яке характеризує охорону навколишнього середовища.

Охорона навколишнього середовища – сукупність адміністративних, правових, економічних, політичних заходів, які здійснюються з боку держави й суспільства, спрямоване на раціональне використання, відтворення та збереження природних ресурсів, а також намагання обмежити знищуючий вплив технічної діяльності людини на навколишнє середовище.

В основу формування державної екологічної політики України під час її формування як самостійної країни було покладено базовий принцип, згідно з яким екологічна безпека держави стає важливим елементом і складовою частиною національної безпеки. Цей принцип, було закріплено низкою нормативних актів у вигляді законів та документів, які базуються на основних засадах Конституції України, а також прийманням участі у створенні міжнародних конвенцій по охороні навколишнього середовища.

Для слідкування за дотримання всіх положень у сфері екології і втому числі охорони навколишнього середовища в Україні засноване Міністерство екологічної безпеки, яке здійснює державний контроль за дотриманням вимог законодавства з наведених питань, а також ядерної та екологічної безпеки і організації проведення державної екологічної експертизи.

Згідно постанови ВР України затверджено «Основні напрямки державної політики в області охорони навколишнього середовища, використання природних ресурсів і забезпечення екологічної безпеки».

Цей нормативний акт передбачає при освоєнні Світового океану, особливо Чорного та Азовського морів морськими технічними засобами України виконання ними природоохоронного законодавства України, природоохоронних вимого, встановлених міжнародними конвенціями, приймання участі у діяльності міжнародних і національних організацій по охороні навколишнього середовища з метою своєчасної підготовки підприємств морської галузі до виконання нових вимог, пов’язаних з охороною довкілля.

5.1 Забруднення навколишнього середовища при експлуатації судна-носія

Виробничі відходи утворюються в процесі експлуатації суднових енергетичних установок (СЕУ). Їх поділяють на тверді відходи (ганчір’я, тканина, синтетичні й паперові фільтри, дерево, резина, пароніт і т.п.) і рідинні відходи (шлам від сепараторів палива і мастил). Об’єм виробничих відходів залежить від потужності СЕУ і від стану механізмів. В середньому насудніскопичується10-20 кг твердих відходів на добу, які зберігаються в спеціальних ємностях, а потім здаються на беріг або спалюються в спеціальних печах інсинераторах. Рідкі відходи подають в інсинератори у розпиленому вигляді через спеціальні шламові форсунки.

Проблема запобігання забрудненню морського середовища сміттям може бути вирішена при використанні комбінованих печей, які знищують всі види відходів.

Основною причиною забруднення морського середовища в нормальних умовах експлуатації судна є скидання нафто наявних вод. Нафта та її продукти потрапляючи в море, розтікаються тонкою плівкою. Інтенсивність забруднення вод Світового океану нафтою в результаті судноплавства та експлуатації флоту складає 2 млн. тон на рік.

Для зменшення кількості нафтових вод судна обладнують танками чистого баласту, застосовують завантаження зверх залишків, мийку сирою нафтою і інші заходи.

Значним джерелом забруднення моря є нафтова вода СЕУ, які утворюються при експлуатації і характеризуються більш складним складом, з причин наявності в них нафтопродуктів різних видів, механічних домішок органічного і неорганічного походження. Скупчення нафто водяної суміші підлягає регулярному видаленню з машинних приміщень судна, згідно вимог міжнародної конвенції МАРПОЛ.

5.2 Розробка заходів щодо запобігання заходів забрудненню тепловими та газовими випаровуваннями судна-носія

Основними об’єктами негативного впливу СЕУ є водойми та атмосферне повітря. В процесі експлуатації СЕУ в повітря та водне середовище відбуваються викиди нафто наявних вод, відпрацьованих газів, тепла і виробничих відходів.

В продуктах неповного згоряння палива – випускних газах – основна частина шкідливих речовин є сажа, окис вуглецю, вуглеводень і альгідриди, її кількість обумовлена різними хімічними реакціями окислення палива впередполум’яний період згоряння і розширення газів.

Токсичність відпрацьованих газів визначається сортом палива та умовами його згоряння. Використання більш дешевого палива викликає підвищене забруднення навколишнього середовища. На номінальному режимі роботи двигунів внутрішнього згоряння 77% об’єму випускних газів складає азот, біля 13% кисень, по 5% вуглекислий газ та водяна пара. Вміст сажі, окису азоту, окис

вуглецю і вуглеводень не перевищують 0,5%. Зниження токсичності відпрацьованих газів відбувається по двом напрямкам – зменшенню шкідливості газів в процесі їх утворення і зменшення шкідливості випускних газів для суднових двигунів можна запропонувати наступний комплекс по зниженню шкідливості газів в процесі їх експлуатації.

Ефективних способом поліпшення екологічних показників двигунів внутрішнього згоряння (збільшується повнота згоряння, знижується димність) є зменшення концентрації оксидів азоту за рахунок уприскування води в трубопровід впускання повітря; застосування термофорсування, тобто попередньої обробки палива; збагачення киснем повітряного заряду; використання водопаливних емульсій; введення анти димних присипок до палива; своєчасне технічне обслуговування і ремонт паливної апаратури.

Понизити шкідливість випускних газів можна шляхом нейтралізації, очищення і утилізації, опалення шкідливих домішок, а також за допомогою різних пристроїв, встановлених у випускному тракті.

Рідинна нейтралізація заснована на проходженні випускних газів через шар рідини, яка стримує сажу і рідкі аерозолі, а також гасять іскри. Уловлювання іскри сажі може здійснюватися і в іскрогасниках, які виконують додаткову функцію глушників шуму. Загалом забруднення атмосфери СЕУ складає не більше 6 % в порівняні з іншими джерелами.

Теплові викиди СЕУ та їх влив на навколишнє середовище в даний час недостатньо вивчені. Але більше половини всієї теплової енергії, яка підводиться до СЕУ, віддається с відпрацьованими газами і рідиною охолодження в навколишнє середовище.

Усі теплові викиди СЕУ можна поділити на дві групи: тепло, яке поступає в атмосферу і тепло, яке віддається забортній воді. В першу групу входять викиди з відпрацьованими газами від головних та допоміжних двигунів, парогенераторів і від випромінювання нагрітих поверхонь обладнання. До другої групи відносяться викиди рідини, яка охолоджує двигуни, механізми, системи, валопроводи і тепло від тертя рушіїв.

Також проблемою забруднення навколишнього середовища є скидання сміття у відкриті води морів та океанів. Забороняється викидати в море (океан) незалежно від відстані до берега всі види пластмас, включаючи синтетичні троси, рибальські сітки і поліетиленові мішки для сміття. Для їхнього збору на судні встановлюється ємність, що закривається кришкою, а по приходу в порт це сміття здається на беріг.

В особливих районах забороняється викидати будь-яке сміття, за винятком харчових відходів на відстані не менш 12 миль від берега. До таких районів відносяться:

– Малаккська протока;

– Середземне, Чорне, Балтійське, Червоне, Північне, Карибське, Внутрішнє Японське моря;

– Перська та Мексиканська затоки;

– райони Антарктики, розташовані до півдня від 600-ї південної широти.

5.3 Висновки до розділу 5

1. Судноплавство є одним з основних джерел забруднення морського середовища.

2. Слід розрізняти негативний вплив на морське середовище судноплавства взагалі, і негативний вплив від використання різних підводних систем та підводних апаратів зокрема.

3. негативні прояви судноплавства можна класифікувати на ті, виникають у зв’язку з:

- технічною експлуатацією суден та підводних систем;
- використання підводних систем не за призначення;
- надзвичайними непереборними обставинами.

ВИСНОВКИ

У роботі вирішувалося завдання зниження трудомісткості процесу розробки робототехнічних систем з вбудованими алгоритмами систем управління шляхом розробки програмного забезпечення комплексного інструментального засобів моделювання робототехнічних систем і інструментальних засобів моделювання вбудованих алгоритмів систем управління, що параметризується. Окрім цього, результатом роботи стало розширення бібліотек готових рішень за рахунок реалізованих прикладів моделей об'єктів та моделей алгоритмів управління. Була розроблена навчальна модель чотириколісного робота з вбудованим алгоритмом управління, що працює у двокоординатній горизонтальній площині, та модель автономного безлюдного підводного апарату з вбудованим алгоритмом управління у тривимірному просторі на основі нечіткої логіки та навчання з підкріпленням.

З їхньою допомогою були проведені дослідження, що демонструють результативність та перспективність запропонованого підходу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. АКАДЕМІК [Електронний ресурс], Веб-сайт проекту АКАДЕМІК.– URL:https://dic.academic.ru/.

2. Бутаков Ст, Фаградянц І. Політехнічний термінологічний тлумачний словник Polyglossum. - Словникове видавництво ЕТС.

3. Клімчик А.А., Магід Є.А., Хусіанов Р.Р. Метод управління рухом двоногого крокуючого робота по довільній траєкторії. // Мехатроніка, автоматизація, керування. - 2018. - № 19. - С. 633-641.

4. Лавренов Р.О., Магід Е.А., Мацуно Ф. Розробка та імплементація сплайн-алгоритму планування шляху в середовищі ROS/Gazebo // Праці СПІІРАН. – 2019. – № 18. – С. 57-84.

5. Накано Еге. Введення у робототехніку. - М: Світ, 1988.

6. ARGoS [Електронний ресурс], Веб-сайт проекту ARGoS. - URL:https://www.argos-sim.info.

7. Acutronic robotics [Електронний ресурс], Веб-сайт проекту Acutronic robotics.– URL:https://acutronicrobotics.com/.

8. Afanasyev I., Sagitov A., Magid E. ROS-Based SLAM для Gazebo-Simulated Mobile Robot в Image-Based 3D Model of Indoor Environment // Advanced Concepts for Intelligent Vision Systems. - 2015.

18.05.2019) Bullet [Електронний ресурс], Веб-сайт проекту Bullet. - URL: https://pybullet.org/wordpress/.

9. Documentation ROS [Електронний ресурс], Documentation ROS. - URL:http://wiki.ros.org.

10. Feature and performance comparison of the V-REP, Gazebo

і ARGoS robot simulators / L. Pitonakova, M. Giuliani, A. Pipe, A. Winfield / / Lecture Notes in Computer Science. - 2018. - № 10965. - С. 357-368.

11. Ferrante E., Turgut AE, Duenez-Guzman E. Evolution of SelfOrganized Task Specialization in Robot Swarms // PLoS Computational Biology. ^2015.

12. Florea AG INTEGRATING V-REP SIMULATED MOBILE ROBOT INTO

ROS // UPB Sci. Bull. – 2018. – № 80.

13. GLScene [Електронний ресурс], Веб-сайт проекту GLScene.– URL:http://glscene.sourceforge.net/wikka/.

14. Gazebo [Електронний ресурс], Веб-сайт проекту Gazebo.^ URL:http://gazebosim.org.

15. Gutierrez C., San Juan L. Towards distributed and real-time framework для robots: Evaluation of ROS 2.0 communications for real-time robotic applications. // Erie Robotics SL - 2018.

16. Gym.OpenAI [Електронний ресурс], Веб-сайт проекту Gym.OpenAI.– URL:https://gym.openai.com.

17. Havok [Електронний ресурс], Веб-сайт проекту Havok. URL:https://www.havok.com/.

18. Ignition [Електронний ресурс], Веб-сайт проекту Ignition.– URL:https://ignitionrobotics.org.

19. Ijspeert AJ, Crespi A., Ryczko D. Від ходьби до їзди з Salamander Robot Driven by Spinal Cord Model // Science. - 2007. - № 315. - С. 1416-1420.

20. Irrlicht [Електронний ресурс], Веб-сайт проекту Irrlicht. – URL: http://irrlicht.sourceforge.net/.

21. Ishiwata Y., Matsui T. Development of Linux which has Advanced Real-Time Processing Function // 16th Annual Conference of

Robotics Society of Japan. - 1998. - С. 355-356.

22. Jin L., Chen P., Zhang R. Longitudinal velocity estimation базується на fuzzy логіці для електронної stability control system. // Advances in Mechanical Engineering. - 2017. - №9. - С. 1-12.

23. KANEHIRO F, FUJIWARA K., KAJITA S. Open Architecture Humanoid Robotics Platform. // International Conference on Robotics and Automation. - 2002.

24. Katayama K., Koshiishi T., Narihisa H. Reinforcement Learning Agents with Primary Knowledge Designed by Analytic Hierarchy Process // ACM symposium on Applied computing. - 2005. - № 05. - С. 14-21.

25. Li T., Meagher R., Davis W. Введення в реінформацію навчання з AWS

RoboMaker // AWS RoboMaker, Robotics. - 2019.

26. Liu S., Wei Y., Gao Y. 3D path planning for AUV using fuzzy logic // CSIP. - 2012.

27. Martinez-TenorA., Fernandez-Madrigal A. Нагороди про загальне підтвердження реінформації Learning for Multiple Robotic Tasks. // MAPIR. - 2017.

28. Maruyama Y., Kato S., Azumi T. Exploring the performance of ROS2. 2016 International Conference on Embedded Software // EMSOFT. – 2016. – С. 1-10.

29. MatLab [Електронний ресурс], Веб-сайт проекту MatLab. - URL:https://matlab.ru.

30. NAGASAKA K., INABA N. Dynamic Walking Pattern Generation для Humanoid Робот базується на Optimal Gradient Method // International Conference on Systems, Man, and Cybernetics. - 1999.

31. NVIDIA PHYSX SYSTEM SOFTWARE [Електронний ресурс], Веб-сайт NVIDIA PHYSX SYSTEM SOFTWARE. - URL:https://www.nvidia.ru/object/physx-9.19.0218-driver-ru.html.

32. Nogueira L. Comparative Analysis Between Gazebo and V-REP Robotic Simulators // School of Electrical and Computer Engineering University de Campinas. - 2014.

33. ODE [Електронний ресурс], Веб-сайт проекту ODE. - URL:https://www.ode.org/.

34. OGRE [Електронний ресурс], Веб-сайт проекту OGRE. - URL:https://www.ogre3d.org/.

35. OpenAI [Електронний ресурс], Веб-сайт проекту OpenAI. - URL:https://openai.com/.

36. OpenGL [Електронний ресурс], Веб-сайт проекту NVIDIA. URL:https://developer.nvidia.com/opengl.

37. OpenHRP3 [Електронний ресурс], Веб-сайт проекту OpenHRP3.– URL:https://fkanehiro.github.io/openhrp3-doc/ en/index.html.

38. Peralta E., Fabregas E., Farias G. Розвиток Khepera IV Library for V-REP Simulator. // IFAC-PapersOnLine. - 2016. - JY «49. – с. 81-86.

39. Pinciroli C., Talamali MS Simulating Kilobots within ARGoS: models and experimental validation // ANTS. - 2018. - С. 29-31.

40. Pinciroli C., Trianni V., O'Grady R. ARGoS: Modular, Parallel, Multi-Engine Simulator для Multi-Robot Systems // Swarm Intelligence. - 2017. - № 6. - С. 271-295.

41. Programming Guide for Direct3D 9 [Електронний ресурс], Веб-сайт проекту Microsoft. URL:https://docs.microsoft.com/en-us/windows/desktop/direct3d9/ dx9-graphics-programming-guide.

42. ROS [Електронний ресурс], Веб-сайт проекту ROS. - URL:https://www.ros.org.

43. ROS2 [Електронний ресурс], Веб-сайт проекту ROS. - URL:https://design.ros2.org/.

44. SDF [Електронний ресурс], Веб-сайт проекту SDF. - URL:http://sdformat.org.

45. Safeea M., Neto P. Human-robot Колісія озброєння для Industrial Robots: A V-REP заснований Solution // TE2018. - 2018. - № 25. - С. 81-86.

46. Spica R., Claudio G., Spindler G. Interfacing Matlab/Simulink with V-REP для Easy Development of Sensor-Based Control Algorithms for Robotic Platforms. // Conf. Workshop on Robotics and Automation on MATLAB/Simulink for Robotics Education and Research. 2014.

47. Swarmanoid [Електронний ресурс], Веб-сайт проекту Swarmanoid.– URL:http://www.swarmanoid.org.

48. Такая К., Асаї Т., Кромов V. Флорентін Смарандаче. Simulation environment for mobile robots testing using ROS and Gazebo. // ICSTCC. 2016.

49. Unity [Електронний ресурс], Веб-сайт проекту Unity. - URL:https://unity.com/ua.

50. V-REP [Електронний ресурс], Веб-сайт проекту V-REP. -

URL: http://www.coppeliarobotics.com.

51. Virtual Experiments Design для Роботів, базованих на V-REP. / M. Xie, D. Zhou, Y. Shi, R. Jia // IOP Conf. Series: Materials Science and Engineering. - 2018. -

№ 428.

52. Webots [Електронний ресурс], Веб-сайт проекту Webots.– URL:http://www.cyberbotics.com/webots.php.

53. Weeren A. Розробка Роботики Applications with MATLAB і Robotics System Toolbox. - MathWorks Benelux, 2015.

54. Yokoi K., Kanehiro F., Kaneko K. Honda Humanoid Robot Controlled by AIST Software // IEEERAS. - 2001.

55. Zamora I., Lopez NG, Vilches VN Extending the OpenAI Gym for robotics: a toolkit для reinforcement навчання шляхом використання ROS і Gazebo. - 2016.