Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)
Моделювання сенсорної системи мобільного робота у симуляторі CoppeliaSim
(тема)

Виконав:
здобувач 2 року навчання,
групи КТРСм-24-1
Ростислав КРИВЧУН
(власне ім'я, прізвище)

Спеціальність 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Комп'ютеризовані та робототехнічні системи
(повна назва освітньої програми)

Керівник доц. Артем БРОННІКОВ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри КІТАР _____ Ігор НЕВЛЮДОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Я, Кривчун Ростислав Валерійович, як здобувач вищої освіти ХНУРЕ, розумію та підтримую політику закладу з академічної доброчесності. Я не надавала і не одержувала недозволену допомогу під час підготовки кваліфікаційної роботи. Я не використовував штучний інтелект для підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

«19.12.2025»  Ростислав КРИВЧУН

(підпис)

Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки

Рівень вищої освіти другий (магістерський)

Спеціальність 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Комп'ютеризовані та робототехнічні системи
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Кривчуну Ростиславу Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Моделювання сенсорної системи мобільного робота у симуляторі CoppeliaSim

затверджена наказом університету від 10.11. 2025 р. № 1018Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 26.12.2025 р.

3. Вихідні дані до роботи Комп'ютерне моделювання
Мобільний робот
Сенсорна система (датчикі лінії, кольору)
Симулятор CoppeliaSim

4. Перелік питань, що потрібно опрацювати в роботі
Вступ
Аналіз сучасного стану систем моделювання робототехнічних систем
Обґрунтування використання Coppeliasim для дослідження сенсорної системи
Моделювання мобільного робота у симуляторі CoppeliaSim
Моделювання сенсорної системи мобільного робота у симуляторі CoppeliaSim
Дослідження переміщення модельованого робота у середовищі CoppeliaSim
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) слайди у форматі PowerPoint у кількості 12 слайдів з розширенням .pptx.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз технічного завдання	03.09-08.09.2025	виконано
2	Аналіз літератури за темою	09.09-18.09.2025	виконано
3	Виконання розділу 1 Аналіз сучасного стану систем моделювання робототехнічних систем	19.09-27.09.2025	виконано
4	Виконання розділу 2 Обґрунтування використання Coppeliasim для дослідження сенсорної системи	28.09-04.10.2025	виконано
5	Виконання розділу 3 Моделювання мобільного робота у симуляторі CoppeliaSim	05.10-05.11.2025	виконано
6	Виконання розділу 4 Моделювання сенсорної системи мобільного робота у симуляторі CoppeliaSim	07.11-28.11.2025	виконано
7	Виконання розділу 5 Дослідження переміщення модельованого робота у середовищі CoppeliaSim	29.11-14.12.2025	
8	Оформлення пояснювальної записки	14.12-18.12.2025	виконано
9	Подання роботи на рецензію	20.12.2025	виконано
10	Подання роботи на підпис зав. кафедри	21.12.2025	виконано
11	Подання атестаційної роботи в ЕК	26.12.2025	виконано

Дата видачі завдання 1.09.2025 р.

Здобувач _____ Ростислав КРИВЧУН
(підпис) (ПІБ студента)

Керівник роботи _____ доц. Артем БРОННІКОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 94 с., 2 табл., 82 рис., 2 дод., 34 джерел.

МОДЕЛЮВАННЯ, МОБІЛЬНИЙ РОБОТ, COPPELIASIM, СЕНСОРНА СИСТЕМА, КЕРУВАННЯ.

Метою даної кваліфікаційної роботи є підвищення ефективності розробки мобільних роботів за рахунок розробки та дослідження комп'ютерної моделі у симуляторі CoppeliaSim для забезпечення його автономної навігації та взаємодії з віртуальним середовищем.

Об'єкт дослідження – процес моделювання сенсорної системи мобільного робота та її функціонування у віртуальному середовищі симулятора CoppeliaSim.

Предмет дослідження – модель, алгоритми та програмне забезпечення для налаштування та оцінки ефективності віртуальної моделі в CoppeliaSim для автономної навігації.

Проведено аналіз літератури з моделювання у робототехніці, аналіз сучасних систем моделювання, дослідження сенсорної системи у симуляторі CoppeliaSim, розроблено програмне забезпечення для моделювання сенсорної системи мовою Lua у симуляторі CoppeliaSim та провести експериментальні дослідження з розробленою моделлю.

ABSTARCT

Explanatory note: 94 p., 2 tab., 82 fig., 2 app., 34 sources.

MODELING, MOBILE ROBOT, COPPELIASIM, SENSOR SYSTEM, CONTROL.

The purpose of this qualification work is to increase the efficiency of mobile robot development by developing and researching a computer model in the CoppeliaSim simulator to ensure its autonomous navigation and interaction with the virtual environment.

The object of research is the process of modeling the sensor system of a mobile robot and its functioning in the virtual environment of the CoppeliaSim simulator.

The subject of the study is the model, algorithms, and software for setting up and evaluating the effectiveness of the virtual model in CoppeliaSim for autonomous navigation.

The analysis of the literature on modeling in robotics, analysis of modern modeling systems, study of the sensor system in the CoppeliaSim simulator, development of software for modeling the sensor system in Lua in the CoppeliaSim simulator and experimental studies with the developed model were carried out.

ЗМІСТ

Вступ 10
1 Аналіз сучасного стану систем моделювання робототехнічних систем 12
1.1 Загальне значення моделювання у робототехніці 12
1.2 Переваги та недоліки моделювання 16
1.3 Класифікація систем моделювання робототехнічних систем 19
1.4 Огляд ключових систем моделювання та їх характеристики 21
1.4.1 ROS (Robot Operating System) 21
1.4.2 MATLAB/Simulink 22
1.4.3 Webots 23
1.4.4 Adams (MSC Adams) 23
1.4.5 CoppeliaSim 24
1.5 Висновки до розділу 1 25
2 Обґрунтування використання Coppeliasim для дослідження сенсорної системи 26
2.1 Датчики наближення (Proximity sensor) у CoppeliaSim 26
2.2 Датчик бачення (Vision sensor) у CoppeliaSim 29
2.3 Теорія автоматичного керування роботом на основі датчика наближення 34
2.4 Висновки за розділом 2 37
3 Моделювання мобільного робота у симуляторі CoppeliaSim 38
3.1 Розроблення моделі робота з диференціальним приводом 38
3.2 Розроблення програми керування простою моделлю робота 43
3.3 Отримання даних і їх візуалізація 45
3.4 Висновки до розділу 3 49
4 Моделювання сенсорної системи мобільного робота у симуляторі CoppeliaSim 50
4.1 Моделювання слідування за лінією 50
4.1.1 Моделювання керування швидкістю 50

4.1.2 Моделювання сенсору лінії 52
4.1.3 Алгоритм переміщення по лінії 57
4.2 Моделювання слідування стіною за допомогою датчиків наближення 59
4.2.1 Моделювання оточення робота 59
4.2.2 Моделювання сенсорів наближення 62
4.2.3 Реалізація алгоритму руху стіною за допомогою сенсорів наближення 64
4.3 Висновки до розділу 4 68
5 Дослідження переміщення модельованого робота у середовищі CoppeliaSim 69
5.1 Дослідження руху складною траєктроїєю з використанням датчиків лінії 69
5.2 Дослідження руху стіною з використанням датчиків наближення 76
5.3 Охорона праці 84
5.4 Висновки до розділу 5 85
Висновки 87
Перелік джерел посилання 90
Додаток А Код програми 95
Додаток Б Демонстраційний матеріал 104

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

АР – автономний робот;

ЖЦ – життєвий цикл;

МР – мобільний робот;

РОС – розподілені обчислювальні системи;

РП – робочий простір;

РС – робототехнічна система;

Digital twins – цифрові двійники;

HiL – Hardware-in-the-Loop;

ROS (Robot Operating System;

SiL – Software-in-the-Loop.

ВСТУП

Сьогодні світ все більше орієнтується на автономні системи, де мобільні роботи (МР) відіграють одну з ключових ролей.

У епоху Індустрії 4.0 та 5.0, застосування роботів значно зростає – від складів і промисловості до медицини та побуту. Головною потребою є їхня автономність та ефективність.

Основою будь-якої автономної системи є її сенсорна підсистема, що дозволяє отримувати інформацію про робочий простір, місцезнаходження, оточення, перешкоди та цілі і потрім буде використано навігації, уникнення зіткнень, виконання завдань та взаємодії з оточуючими об'єктами.

Створення та тестування таких систем є дорогим, дуже довгим та достатньо небезпечним процесом, бо прототипи робототехнічних систем вимагають значних інвестицій у матеріали та виробництво, а помилки у програмному чи апаратному забезпеченні можуть призвести до непередбачуваних ситуацій.

Щоб уникнути таких витрат і пов’язаних ризиків, створюються спеціальні програми-симулятори. Одним з таких є CoppeliaSim – інструмент для розробки та тестування складних робототехнічних систем та їхніх сенсорних підсистем. Моделювання сенсорів у віртуальному середовищі CoppeliaSim не тільки дозволяє оцінити їх ефективність у різних сценаріях, але й оптимізувати розміщення сенсорів, їхні параметри та алгоритми обробки даних, що є невід'ємною частиною проектування будь-якого мобільного робота.

Метою даної кваліфікаційної роботи є підвищення ефективності розробки мобільних роботів за рахунок розробки та дослідження комп’ютерної моделі у симуляторі CoppeliaSim для забезпечення його автономної навігації та взаємодії з віртуальним середовищем.

Об'єкт дослідження – процес моделювання сенсорної системи мобільного робота та її функціонування у віртуальному середовищі симулятора CoppeliaSim.

Предмет дослідження – модель, алгоритми та програмне забезпечення для налаштування та оцінки ефективності віртуальної моделі в CoppeliaSim для автономної навігації.

Для досягнення поставленої мети потрібно вирішити наступні задачі:

– провести аналіз літератури з моделювання у робототехніці;

– провести аналіз сучасних систем моделювання;

– провести дослідження сенсорної системи у симуляторі CoppeliaSim;

– розробити програмне забезпечення для моделювання сенсорної системи мовою Lua у симуляторі CoppeliaSim та провести експериментальні дослідження з розробленою моделлю;

– розглянути питання охорони праці при роботі з роботами;

– оформити пояснювальну записку згідно з [1] та [2].

Отримані результати роботи можна віднести до Цілі сталого розвитку 9 “Промисловість, інновації та інфраструктура”, а саме п. 9.4 “Сприяти 12 прискореному розвитку високо- та середньо-високотехнологічних секторів переробної промисловості, які формуються на основі використання ланцюгів «освіта – наука – виробництво» та кластерного підходу за напрямами: розвиток інноваційної екосистеми”, індикатор 9.4.1.

Матеріали роботи пройшли апробацію у [3].

1 АНАЛІЗ СУЧАСНОГО СТАНУ СИСТЕМ МОДЕЛЮВАННЯ РОБОТОТЕХНІЧНИХ СИСТЕМ

1.1 Загальне значення моделювання у робототехніці

Моделювання відіграє ключову роль у життєвому циклі (ЖЦ) робототехнічних систем, за рахунок того, що воно дозволяє розробникам починати ефективніше працювати з роботами від початкової ідеї створення до кінцевого впровадження. Воно створює віртуальне середовище, де можна проводити тестування, аналіз та оптимізацію різних показників системи, уникаючи потреби у дорогих та часозатратних фізичних моделях-прототипах [3] (рис. 1.1).

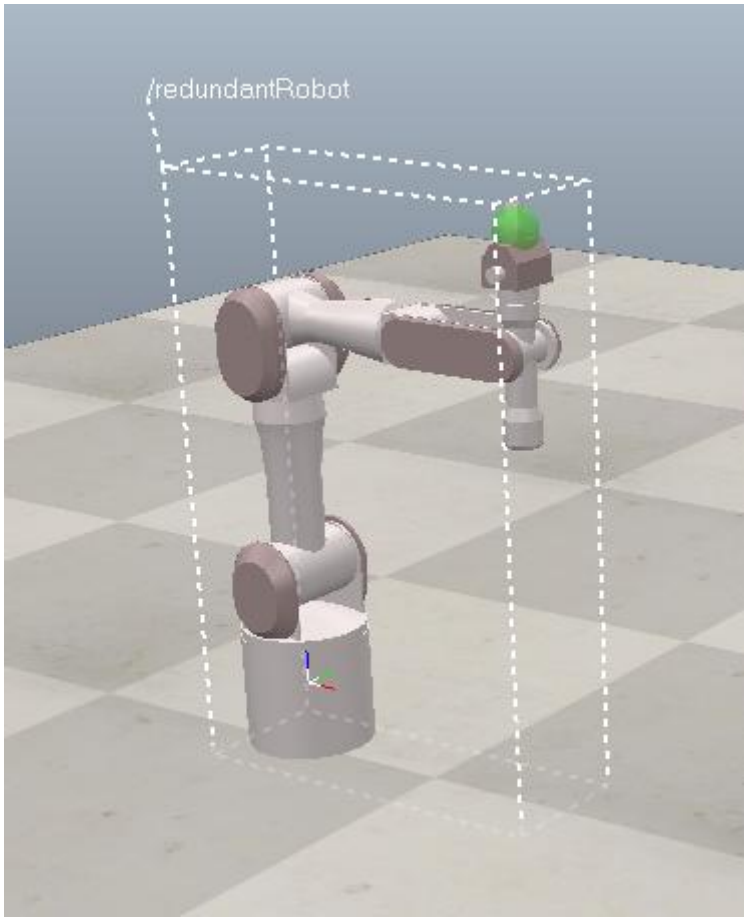

Рисунок 1.1 – Моделювання у робототехніці

На першому етапі (створення концепції та проектування), моделювання дає змогу швидко перевіряти різні ідеї для створення робота, наприклад, створити правильну кінематичну та динамічну структуру. Також є можливість проводити експерименти з різними конфігураціями, розмірами та ступенями свободи. Симуляції допомагають оцінити ефективність таких складових робота, як двигуни, сенсори, приводи тощо ще до як будуть витрачені кошти на закупівлі, виявити потенційні проблеми – обмеження робочого простору (РП), зіткнення чи нестійкість роботів на початкових етапах, що зменшує кількість помилок [4]. Також, моделювання надає інструменти для візуалізації роботи, що в значній мірі впливають на розуміння поведінки робота і сприяють ефективній комунікації в команді [5].

На наступному етапі (тестування та верифікація), моделювання є незамінним для безпечного тестування робота в небезпечних або екстремальних умовах, які були б ризикованими чи неможливими для реального пристрою, наприклад, тестування в небезпечному для людини середовищі або тестування на відмову. Це дозволяє швидко провести тестування та внести зміни у програмне чи апаратне забезпечення для зменшення неочікуваних проблем [6]. Розробники можуть тестувати та налаштовувати складні алгоритми управління (такі як ПІД-регулятори, алгоритми планування траєкторій та навігації) у віртуальному середовищі, прискорюючи процес налагодження ще до завантаження на реального робота. Завдяки моделюванню можна провести імітацію даних з різних сенсорів (камер, LiDAR, ультразвукових датчиків тощо), для розроблення та тестування цих розроблених алгоритмів без реальних роботів. Також, симуляції можуть генерувати величезні обсяги даних для навчання нейронних мереж та алгоритмів машинного навчання, що особливо цінно, коли реальні дані важко або дорого отримати [7].

На наступному етапі (оптимізація), є можливість змінювати й оптимізувати параметри робота та його програм керування для досягнення максимальної продуктивності – чи то швидкість, точність або

енергоефективність. Таким чином, навіть алгоритми планування траєкторій можуть бути оптимізовані в самій моделі, ще до завантаження в реального робота, щоб уникати перешкоди та мінімізувати час й енергоспоживання. Коєфіцієнти ПІД-регуляторів, параметри фільтрації, налаштування сенсорів, можуть бути точно відкалібровані та задані ще у віртуальному середовищі – це скоротить час налагодження на реальному обладнанні. Також моделювання допомагає виявити слабкі місця в конструкції, якщо вага цих роботів завелика, а також у програмному забезпеченні, що може підвищити продуктивність та уникнути збоїв ще до впровадження [8].

Останній етап (розгортання та експлуатація) дозволяє використання симуляторів для того, щоб навчити операторів роботів, що дозволить набиратися практики в безпечному віртуальному середовищі перед роботою з реальним обладнанням. Для автономних роботів (АР) моделювання дозволяє планувати та перевіряти складні місії, враховуючи потенційні перешкоди та динаміку середовища. Концепція цифрового двійника (віртуальної копії реального робота, digital twins) [9] стає дедалі важливішою, особливо у епоху Індустрії 4.0 та 5.0, і дозволяє стежити та прогнозувати поведінку робота в режимі реального часу, використовуючи дані з фізичного робота для оновлення його віртуальної моделі. Завдяки моделюванню можна також прогнозувати знос компонентів, планувати профілактичне обслуговування та мінімізувати час простою [10].

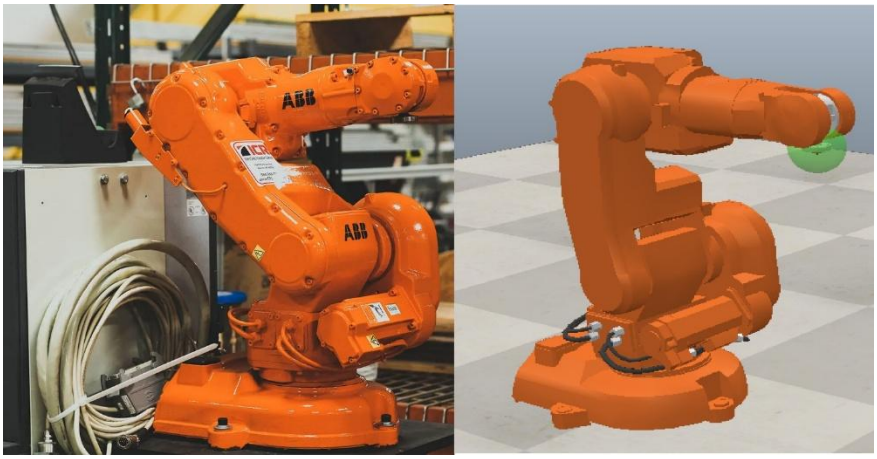

Рисунок 1.2 – Цифровий двійник у робототехніці

Таким чином, моделювання є не просто допоміжним інструментом, а невід'ємною частиною сучасного процесу розробки робототехнічних систем. Воно дозволяє значно скоротити витрати, прискорити розробку, підвищити надійність та безпеку, а також забезпечити високу продуктивність роботів у реальному світі [11].

Моделювання роботів, хоча й є потужним інструментом, стикається з кількома ключовими проблемами, що впливають на його ефективність.

По-перше, критично важливою є точність моделей, тому що модель є спрощеним представленням робота, тому потрібно уважно слідкувати за фізичними властивостями матеріалів, фізичними параметрами (маса, коефіцієнт тертя тощо), моделюванням сенсорів (шум, калібрування), а також враховувати динаміку роботів, бо це може привести до значних розбіжностей між симуляцією та реальним світом [12].

По-друге, необхідно забезпечити реалізм симуляції – якість фізичного двигуна для достовірного відтворення фізичних параметрів (наприклад, параметри зіткнення і тертя), реалістичне моделювання складного навколишнього середовища і РП, включати, наприклад, динамічні перешкоди та різні рівні освітлення. До того ж, для деяких застосувань вимагається виконання симуляції в реальному часі, що часто змушує шукати компроміс між деталізацією та продуктивністю, а також враховувати випадкові фактори, як-от шуми та зміни умов.

По-третє, значною проблемою є потреба у відповідних обчислювальних ресурсах – складні й деталізовані моделі з точними фізичними розрахунками та тривимірним рендерингом вимагають потужних комп’ютерів. Для паралельних симуляцій, що використовуються в машинному навчанні або для тестування великої кількості роботів, потрібні розподілені обчислювальні системи (РОС) та великі обсяги пам'яті для даних.

Останньою проблемою є інтеграція різних аспектів робототехніки: механіки, електроніки та програмного забезпечення. Це вимагає міждисциплінарних знань і часто пов'язано з необхідністю об'єднання різних

спеціалізованих інструментів. Забезпечення коректної взаємодії між змодельованою апаратною частиною та реальним або віртуальним програмним забезпеченням, а також синхронізація та обмін даними між різними симуляторами є складними завданнями, що вимагають ретельної координації.

1.2 Переваги та недоліки моделювання

Моделювання суттєво знижує витрати та прискорює розробку робототехнічних систем (РС). Створення, модифікація та тестування фізичних прототипів роботів призводить до додаткових витрат – фізичних та часових. Віртуальне ж середовище дозволяє проводити численні ітерації проектування, тестування та оптимізації зі значно меншими витратами., що дозволяє швидко виявляти та виправляти помилки та недоліки, адже конструктивні недоліки або проблеми з управлінням можна виявити ще на початкових стадіях, коли їх виправлення коштує найменше [13].

Крім того, моделювання значно підвищує безпеку і дає змогу тестувати роботів у небезпечних або надзвичайно ризикованих умовах, таких як забруднені середовища, великі висоти, сценарії відмов чи зіткнення, що було б неможливо або занадто ризиковано з реальним обладнанням. Можна проводити імітації таких ситуацій, що обов’язково призведуть до відмови для виявлення обмежень системи та розробки майбутніх дій для реагування на надзвичайні випадки без ризику пошкодження дорогого обладнання.

У віртуальному середовищі легко змінювати параметри робота, середовища чи алгоритмів керування, тестувати величезну кількість різних сценаріїв, що фізично неможливо в реальному світі. Головною перевагою є точна відтворюваність експериментів – при завданні одних і тих же вхідних даних та умов, на виході отримується той самий результат, що критично для налагодження та порівняння різних алгоритмів, на відміну від реального світу з його невеликими, але непередбачуваними впливами. Також, якщо дозволяє

залізо, є можливість одночасного запуску декількох симуляцій з різними вхідними даними і умовами для прискорення процесу оптимізації та випробувань [14].

Моделювання дозволяє проводити комплексний аналіз та оптимізацію. У симуляції доступні будь-які внутрішні змінні та стани робота чи середовища, які важко виміряти в реальності (наприклад, точне положення ланок чи внутрішні сигнали контролера), надаючи глибоке розуміння поведінки системи, що дозволяє налаштовувати алгоритми управління та параметри робота крок за кроком для досягнення необхідного результату. Більше того, симуляції генерують величезні обсяги високоякісних, анотованих даних, необхідних для навчання нейронних мереж та алгоритмів машинного навчання, вирішуючи проблему браку реальних даних.

Також моделювання є інструментом для навчання та розвитку майбутніх операторів роботів, дозволяючи їм набувати досвіду в безпечному та контрольованому віртуальному середовищі без ризику пошкодження обладнання чи травмування. Для розробників це забезпечує швидку ітерацію: нові ідеї та алгоритми можна швидко реалізувати, протестувати та отримати негайний зворотний зв'язок, що стимулює інновації та прискорює цикл розробки.

Однією з ключових проблем є розрив між симуляцією та реальністю (Sim-to-Real Gap) [15] (рис. 1.3), бо поведінка робота у віртуальному середовищі може відрізнятися від його дій у фізичному світі через неточності моделей через спрощення фізичних законів, неідеальні компоненти роботів або ігнорування дрібних недоліків конструкції, таких як люфти чи деформації. Також віртуальні сенсори не можуть в достатній мірі відтворити шуми, дрейф та інші параметри реальних датчиків, а симуляція рідко враховує всі випадкові зовнішні фактори (температурні коливання, нерівності поверхні, повітряні потоки). Цьому можна запобігти додавши рандомізацію симуляції для навчання стійкіших алгоритмів, адаптацію в реальності шляхом донавчання на фізичному роботі, трансферне навчання, постійне вдосконалення моделей

сенсорів та фізичних моделей, а також розробку методів оцінки невизначеності для стійкіших алгоритмів керування.

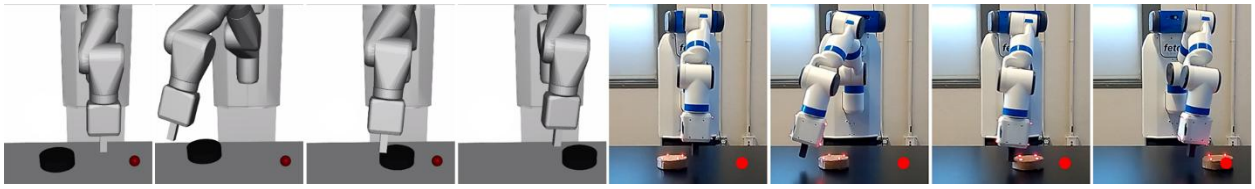

Рисунок 1.3 – Принцип Sim-to-Real

При моделюванні критично важливими є перевірка коректності реалізації моделі відповідно до специфікацій (верифікація), а також перевірка чи адекватно відображає реальну систему для конкретного застосування (валідація)[16]. Найбільш надійний метод –порівняння поведінки змодельованого робота з експериментальними даними реального робота, хоча також використовуються експертна оцінка, аналітичні рішення та тестування на межах. Ці ітеративні, хоч і затратні процеси, є необхідними для довіри до результатів моделювання.

Крім того, реалістичні симуляції (рис. 1.4) робототехнічних систем вимагають значних обчислювальних ресурсів.

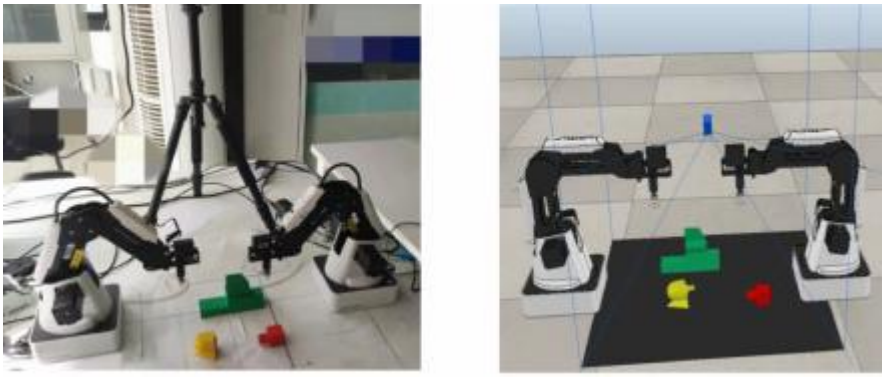

Рисунок 1.4 – Симуляції роботів

Деталізовані моделі роботів зі складною кінематикою та численними сенсорами, разом з точними фізичними розрахунками (особливо для контакту та тертя) та реалістичним рендерингом, вимагають потужних процесорів та

графічних карт. Завдання, такі як навчання машинного навчання або тестування великих флотів роботів, часто вимагають тисяч паралельних симуляцій, що змушує використовувати розподілені системи та хмарні платформи, постійно балансуючи між деталізацією моделі та швидкістю обчислень [17].

Нарешті, значним викликом є складність інтеграції різних аспектів робототехніки: механіки, електроніки та програмного забезпечення. Сучасні роботи є комплексними мехатронними системами, і їх моделювання часто вимагає об'єднання даних з різноманітних спеціалізованих інструментів, що вимагає міждисциплінарних знань. Застосування косимуляції, де кілька симуляторів працюють паралельно, обмінюючись даними, а також забезпечення коректної взаємодії між змодельованим апаратним забезпеченням та реальним програмним забезпеченням (у моделях Software-in-the-Loop або Hardware-in-the-Loop) [18-19], становить значну технічну складність, яка постійно вдосконалюється через розвиток уніфікованих стандартів та покращення інтерфейсів.

1.3 Класифікація систем моделювання робототехнічних систем

Системи моделювання робототехнічних систем можна класифікувати за кількома ключовими ознаками, що дозволяє краще зрозуміти їхній широкий спектр та застосовність для різних етапів розробки та досліджень, а також допомагає вибрати найбільш відповідний інструмент для конкретного завдання.

За призначенням, або типом моделювання, ці системи фокусуються на різних аспектах робототехнічної системи. Фундаментальним є кінематичне [20] та динамічне моделювання [21], яке дозволяє аналізувати рухи робота: кінематика відповідає на питання про положення без урахування сил, тоді як динаміка враховує маси, інерцію, тертя та керуючі сили, аналізуючи, як саме рухається робот під їхнім впливом. Такі симулятори, як Gazebo, CoppeliaSim,

Simscape Multibody та MSC Adams, є типовими прикладами [22]. Оскільки сучасні роботи покладаються на сенсори, моделювання сенсорів дозволяє розробляти та тестувати алгоритми сприйняття без використання дорогого фізичного обладнання. Це включає імітацію камер (з реалістичним світлом і шумами для комп'ютерного зору), LiDAR (для хмар точок та картографування), ультразвукових датчиків, IMU та дальномірів. Gazebo і CoppeliaSim відзначаються широкими можливостями у цій сфері. Моделювання взаємодії з навколишнім середовищем зосереджується на контакті робота з віртуальним світом, імітуючи фізичні взаємодії, такі як контакт, тертя та деформації, а також дозволяючи створювати складні сцени з об'єктами та динамічним освітленням для тестування навігаційних алгоритмів. Окремо виділяється моделювання програмного забезпечення (Software-in-the-Loop, SiL) [23], де код контролера тестується у взаємодії з віртуальною моделлю робота без реального обладнання, що забезпечує високу швидкість тестування та відсутність ризиків. Інтеграція ROS-вузлів з Gazebo або запуск коду в Simulink є поширеними прикладами. Більш просунутим є моделювання апаратного забезпечення (Hardware-in-the-Loop, HiL) [24], що є проміжним етапом, де реальні апаратні компоненти (наприклад, контролер) взаємодіють з віртуальною моделлю, дозволяючи виявляти проблеми із затримками та синхронізацією ще до повного складання.

За рівнем абстракції системи моделювання різняться за деталізацією та фокусом. Низькорівневе (фізичне) моделювання зосереджується на максимально детальному відтворенні фізичних властивостей, враховуючи точні маси, інерцію, пружні властивості матеріалів, тертя та теплові ефекти [25]. Цей рівень використовується для тонкого налаштування механічної конструкції, аналізу навантажень та точного моделювання динаміки руху, як у деяких аспектах Gazebo, Simscape Multibody та MSC Adams. На противагу цьому, високорівневе (поведінкове) моделювання абстрагується від надмірних фізичних деталей, фокусуючись на логіці керування та загальній взаємодії систем. Тут роботи моделюються як чорні скриньки з певними входами та

виходами [26], а основна увага приділяється алгоритмам планування місій, навігації та прийняття рішень, що ідеально підходить для тестування складної логіки керування, симуляції великої кількості роботів та аналізу взаємодії людина-робот. Симуляційні середовища для мультироботних систем та деякі компоненти ROS є прикладами цього рівня абстракції.

1.4 Огляд ключових систем моделювання та їх характеристики

Вибір правильної системи моделювання є вирішальним етапом у розробці робототехнічних систем, оскільки кожен інструмент має свої унікальні особливості, сильні та слабкі сторони.

1.4.1 ROS (Robot Operating System)

ROS (Robot Operating System) разом з Gazebo є одним з найпопулярніших рішень для досліджень та open-source проектів. ROS – це не класична ОС, а радше фреймворк для розробки програмного забезпечення, що спрощує створення розподілених систем керування роботами завдяки набору бібліотек та інструментів для комунікації між компонентами (рис. 1.5) [27].

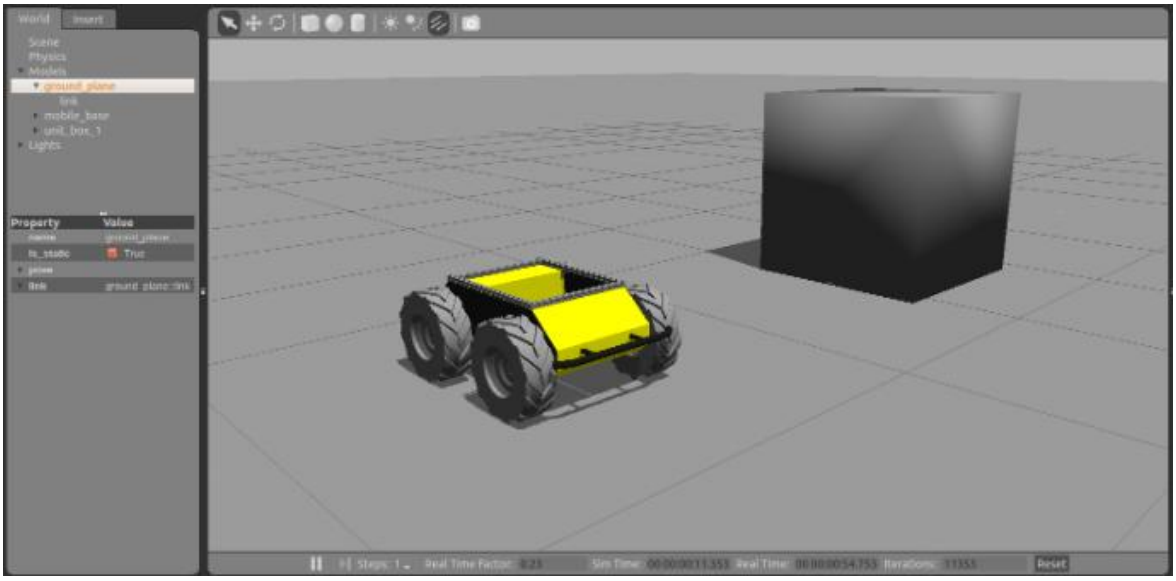

Рисунок 1.5 – Інтерфейс ROS та Gazebo

Gazebo, потужний 3D-симулятор, тісно інтегрується з ROS, пропонуючи реалістичний фізичний двигун для моделювання динаміки руху, зіткнень та взаємодії з оточенням. Він імітує широкий спектр сенсорів (камери, LiDAR, ультразвук) і підтримує стандартні формати URDF/SDF для опису роботів. Перевагами цього тандему є велике та активне ком'юніті, висока модульність та реалістична фізика. Однак, він має високий поріг входу для новачків і значну обчислювальну складність для великих сцен.

1.4.2 MATLAB/Simulink

MATLAB/Simulink від MathWorks – це всеосяжна платформа, особливо популярна в академічному середовищі та для розробки систем керування. Вона дозволяє моделювати різноманітні динамічні системи, включаючи кінематику, динаміку, електроніку та складні системи керування, використовуючи спеціалізовані пакети, як-от Robotics System Toolbox та Simscape Multibody (рис. 1.6) для детальних багатотільних моделей [28]. Зручний графічний інтерфейс для блокового моделювання, широкі можливості аналізу та легка інтеграція з апаратними платформами (Arduino, Raspberry Pi) є її основними перевагами. Проте, це комерційний продукт з високою вартістю ліцензії.

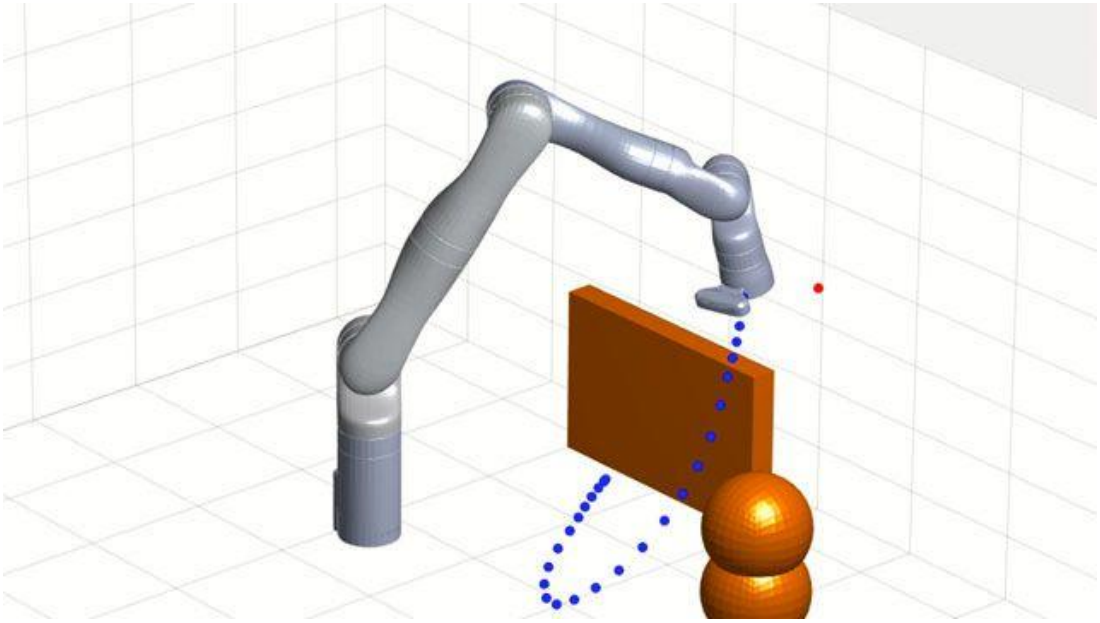

Рисунок 1.6 – MATLAB/Simulink для робототехніки

1.4.3 Webots

Webots – це симулятор з відкритим вихідним кодом, розроблений спеціально для мобільних роботів та маніпуляторів. Він інтегрується з ROS, підтримує моделювання різноманітних сенсорів та має власний фізичний двигун. Цей інструмент особливо зручний для освітніх цілей завдяки простому інтерфейсу та гарній документації. Однак, він може бути менш гнучким для дуже специфічних та складних індустріальних завдань порівняно з Gazebo [29] (рис. 1.7).

Рисунок 1.7 – Інтерфейс Webots

1.4.4 Adams (MSC Adams)

Нарешті, Adams (MSC Adams) є провідним професійним інструментом для багатотільного динамічного аналізу. Це високоспеціалізований інструмент, що дозволяє моделювати складні механізми та їхню взаємодію з надзвичайно високою точністю, фокусуючись на динаміці руху механічних систем, силах та навантаженнях. Його перевагами є висока точність фізичних розрахунків та

потужні можливості оптимізації конструкцій [30]. Проте, Adams є дуже дорогим комерційним продуктом, складним для освоєння та більше орієнтованим на механіку, що робить його менш придатним для розробки програмного забезпечення чи моделювання складних сенсорних систем (рис. 1.8).

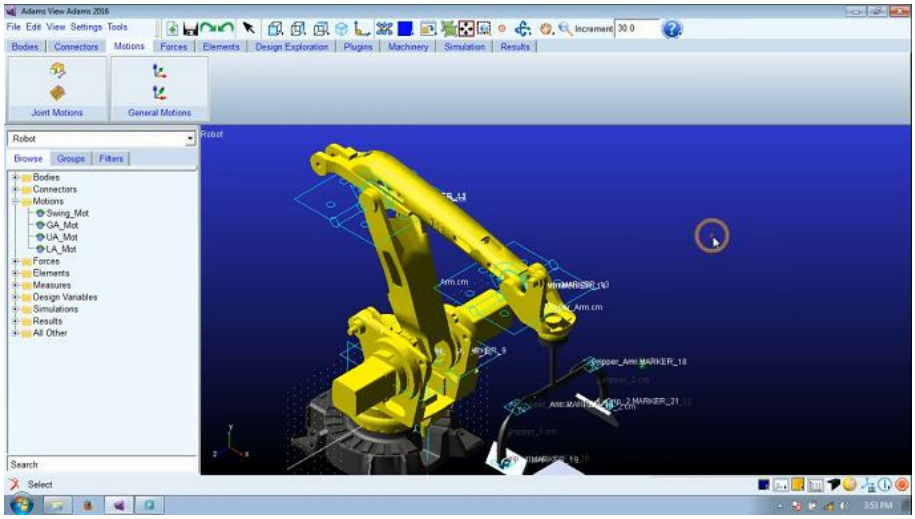

Рисунок 1.8 – Інтерфейс Adams

1.4.5 CoppeliaSim

CoppeliaSim (раніше V-REP) [31] – це універсальний робототехнічний симулятор, що пропонує гнучке та потужне середовище. Він має вбудований фізичний двигун для реалістичної взаємодії, широку бібліотеку готових моделей та підтримує різні мови програмування (Lua, Python, C++), дозволяючи писати скрипти для керування симуляцією безпосередньо в середовищі. Серед переваг – висока гнучкість, кросплатформенність та відмінні засоби візуалізації, хоча його ком'юніті менше, ніж у ROS/Gazebo.

Вибір симулятора завжди залежить від конкретних потреб проєкту: його складності, бюджету, необхідного рівня реалізму та основних цілей моделювання, чи це механічний дизайн, розробка алгоритмів управління, навчання ШІ або підготовка операторів (рис. 1.9).

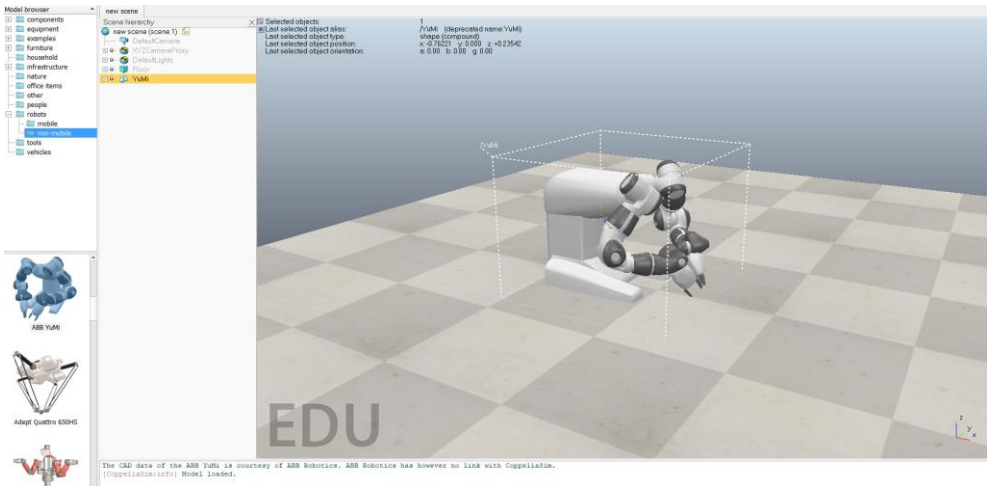

Рисунок 1.9 – Інтерфейс CoppeliaSim

1.5 Висновки до розділу 1

В процесі написання 1 розділу було проведено аналіз літератури за темою, а саме проаналізовано такі напрямки:

– загальне значення моделювання у робототехніці;

– переваги та недоліки моделювання;

– класифікація систем моделювання робототехнічних систем;

– огляд ключових систем моделювання та їх характеристики.

Провівши аналіз систем моделювання (ROS, Matlab, Adams, Webots, CoppeliaSim), було прийнято рішення використати CoppeliaSim тому що він є найкращим симулятором для робототехніки, особливо якщо потрібні гнучкість, кросплатформність, універсальність та широкі можливості програмування (Lua, C++, Python) без прив'язки до екосистеми ROS чи високих витрат на купівлю цих систем моделювання.

2 ОБҐРУНТУВАННЯ ВИКОРИСТАННЯ COPPELIASIM ДЛЯ ДОСЛІДЖЕННЯ СЕНСОРНОЇ СИСТЕМИ

2.1 Датчики наближення (Proximity sensor) у CoppeliaSim

Датчики наближення у симуляторі CoppeliaSim мають наступні форми (рис. 2.1).

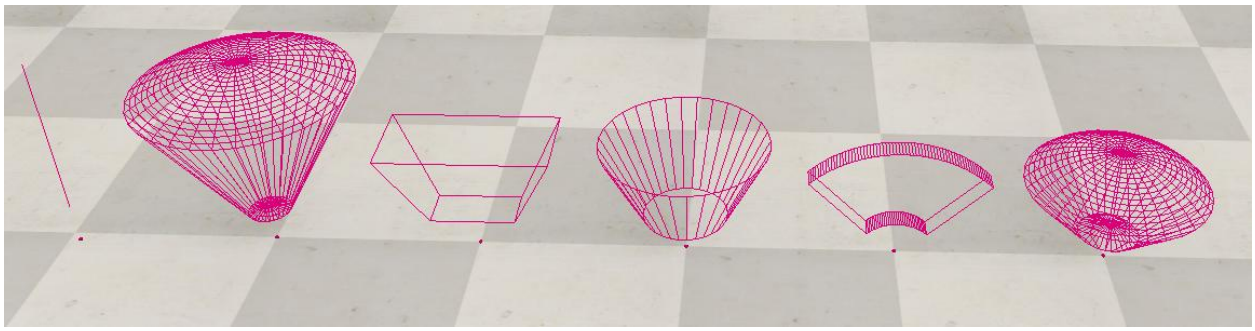

Рисунок 2.1 – Види датчиків наближення

Існує шість основних типів датчиків наближення, кожен з яких можна гнучко налаштовувати для різних завдань.

– променеві датчики чудово підходять для простого моделювання або імітації лазерного далекоміра, оскільки вони є найшвидшими;

– рандомізовані променеві датчики функціонують аналогічно променевим, але випадковим чином сканують конусоподібний об'єм і візуально схожі на конусні;

– для моделювання датчиків з прямокутним об'ємом виявлення ідеально підходять пірамідальні датчики, які також є дуже швидкими;

– циліндричні датчики чудово підходять для моделювання датчиків з обертовим об'ємом виявлення, зберігаючи при цьому високу швидкість;

– дискові датчики дозволяють точно моделювати датчики, що сканують по колу – їхня обчислювальна складність може дещо зростати залежно від обраної точності та режиму роботи;

– конусні датчики пропонують найточніше моделювання більшості датчиків наближення, хоча, як і дискові, можуть бути трохи більш вимогливими до обчислювальних ресурсів в залежності від обраної точності та режиму роботи.

Основними командами для роботи з сенсорами є [32]:

– sim.handleProximitySensor – обробляє (виконує зондування тощо) зареєстрований об'єкт датчика наближення;

– sim.readProximitySensor – зчитує стан датчика наближення. Ця функція не виконує детектування, а лише зчитує результат попереднього виклику sim.handleProximitySensor;

– sim.resetProximitySensor – очищає стан виявлення, колір виявлення, сегменти виявлення тощо об'єкта датчика наближення;

– sim.checkProximitySensor – перевіряє, чи виявив датчик наближення вказаний об'єкт;

– sim.createProximitySensor – створює датчик наближення.

Кожен з типів датчиків має свої власні налаштування спрацьовування, на рис. 2.2- та рис. 2.3 наведено два з таких варіантів налаштування.

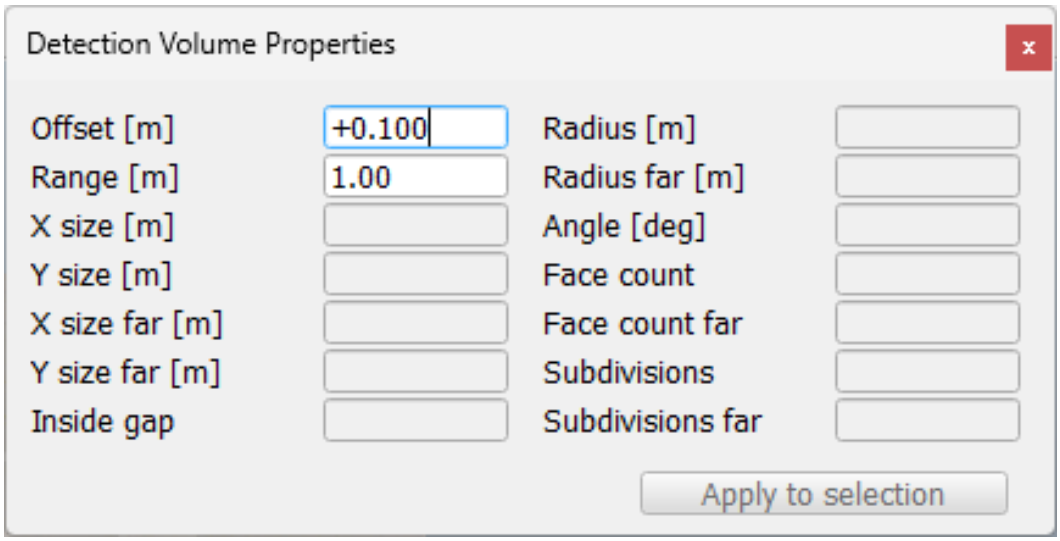

Рисунок 2.2 – Параметри детектування для променевого датчика

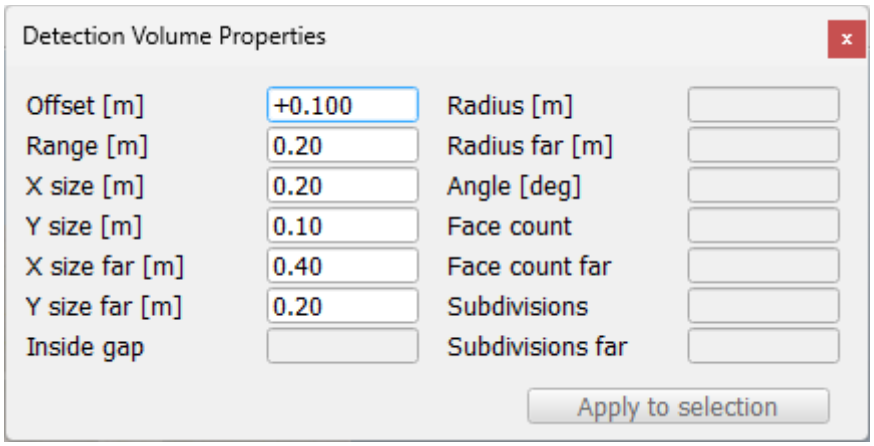

Рисунок 2.3 – Параметри детектування з прямокутним об'ємом виявлення

Для доцільності датчики були повернуті та додатні перешкоди у вигляді кубів (рис. 2.4).

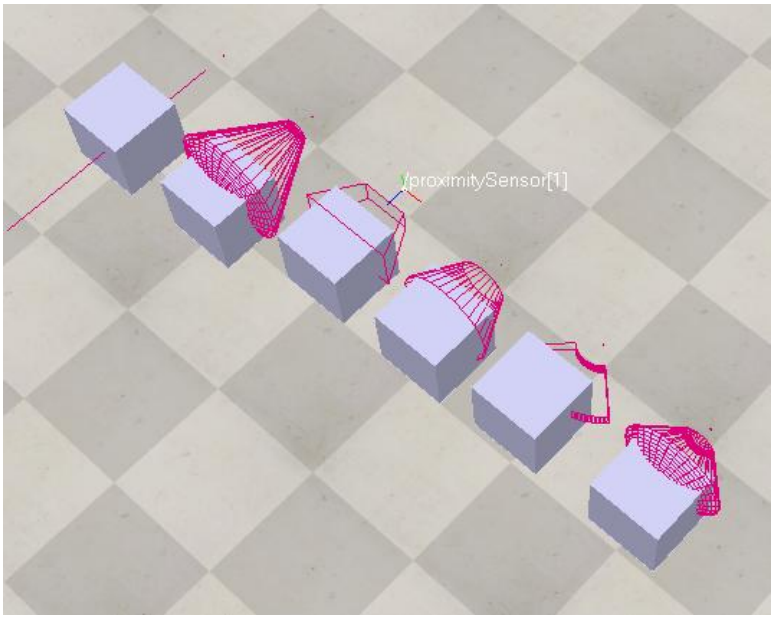

Рисунок 2.4 – Сцена з датчиками та перешкодами

Щоб отримати інформацію з датчиків, потрібно додати змінні датчиків у функцію sysCall_init()

```
sensor0 = sim.getObject('/proximitySensor[0]')
sensor1 = sim.getObject('/proximitySensor[1]')
sensor2 = sim.getObject('/proximitySensor[2]')
sensor3 = sim.getObject('/proximitySensor[3]')
```

```
sensor4 = sim.getObject('/proximitySensor[4]')
sensor5 = sim.getObject('/proximitySensor[5]')
```

А у функції sysCall_sensing() спочатку зчитати дані з датчиків (рис. 2.5):

```
flag0 = 0 p0 = 0.0
flag1 = 0 p1 = 0.0
flag2 = 0 p2 = 0.0
flag3 = 0 p3 = 0.0
flag4 = 1 p4 = 0.28352780511281
flag5 = 1 p5 = 0.43055146302467
```

Рисунок 2.5 – отрманя данних з датчиків

```
flag0,p0=sim.readProximitySensor(sensor0)
flag1,p1=sim.readProximitySensor(sensor1)
flag2,p2=sim.readProximitySensor(sensor2)
flag3,p3=sim.readProximitySensor(sensor3)
flag4,p4=sim.readProximitySensor(sensor4)
flag5,p5=sim.readProximitySensor(sensor5)
```

Потім вивести їх на екран:

```
print('flag0 = '..flag0..' '..'p0 = '..p0)
print('flag1 = '..flag1..' '..'p1 = '..p1)
print('flag2 = '..flag2..' '..'p2 = '..p2)
print('flag3 = '..flag3..' '..'p3 = '..p3)
print('flag4 = '..flag4..' '..'p4 = '..p4)
print('flag5 = '..flag5..' '..'p5 = '..p5)
```

2.2 Датчик бачення (Vision sensor) у CoppeliaSim

Датчики зору, які є видимими об'єктами, працюють дуже схоже на об'єкти камер: вони відображають об'єкти, що перебувають у їхньому полі

зору, і запускають виявлення, якщо задані пороги перевищені або не досягнуті. Датчики зору слід використовувати замість датчиків наближення переважно тоді, коли колір, світло або структура відіграють важливу роль у процесі виявлення (наприклад, інфрачервоні датчики або, загалом, датчики, чутливі до світла (камери тощо)). Однак, залежно від графічної карти, на якій працює програма, або від складності об'єктів сцени, датчики зору можуть працювати трохи повільніше, ніж датчики наближення.

Датчики зору бувають двох різних типів і можуть бути налаштовані для різних цілей (рис. 2.6):

– ортографічний проекційний тип: поле зору орфографічного проекційного типу має прямокутну форму. Вони добре підходять для інфрачервоних датчиків близького радіусу дії або лазерних далекомірів;

– перспективний проекційний тип: поле зору перспективних проекційних датчиків зору має трапецієподібну форму. Вони добре підходять для датчиків камерного типу.

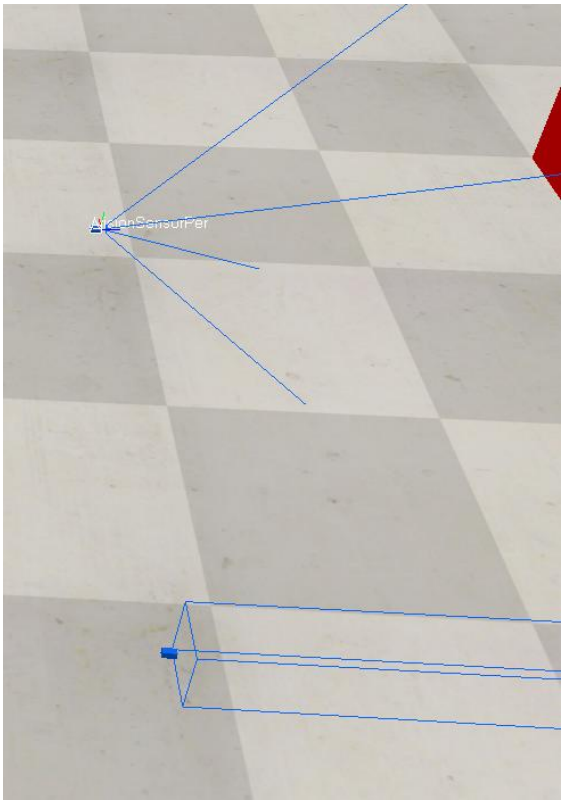

Рисунок 2.6 – Датчик зору

Датчики зору дуже потужні в тому сенсі, що їх можна використовувати різними і гнучкими способами. Наприклад, їх можна використовувати для відображення нерухомих або рухомих зображень із зовнішніх додатків або плагінів. Плагіни також можуть надавати індивідуальні алгоритми обробки зображень, а також алгоритми оцінки (наприклад, умови спрацьовування).

Основні функції датчику:

– sim.handleVisionSensor – обробляє (виконує зондування тощо) об'єкт датчика зору. По-перше, очищає попередні обчислені дані обробки зображення. По-друге, зчитує зображення. По-третє, виконує обробку зображення за допомогою функцій зворотного виклику;

– sim.readVisionSensor – зчитує стан датчика зору. Ця функція не виконує детектування, а лише зчитує результат попереднього виклику sim.handleVisionSensor;

– sim.resetVisionSensor – скидає стан виявлення тощо об'єкта датчика наближення;

– sim.checkVisionSensor – перевіряє, чи виявив датчик зору вказаний об'єкт. Функції зворотного виклику зору будуть викликані на отриманому зображенні. Також перевизначається стан видимості сутності, якщо сутність є об'єктом;

– sim.getVisionSensorImg та sim.setVisionSensorImg – зчитує та встановлює зображення з та на датчика зору;

– sim.getVisionSensorRes та sim.setVisionSensorRes – повертає або встановлює відповідно роздільну здатність датчика зору;

– sim.createVisionSensor – створює датчик зору.

Кожен з датчиків також має налаштування (рис. 2.9).

Render mode (Режим рендерингу) – визначає, як датчик генерує зображення. Near / far clipping plane [m] – ближня / дальня площина відсікання, Persp. angle [deg] / ortho. size [m] – кут перспективи / ортогональний розмір. Якщо датчик працює в ортогональному режимі (без перспективи), це значення

визначатиме розмір поля огляду в метрах. Resolution X / Y – роздільна здатність за шириною та висотою.

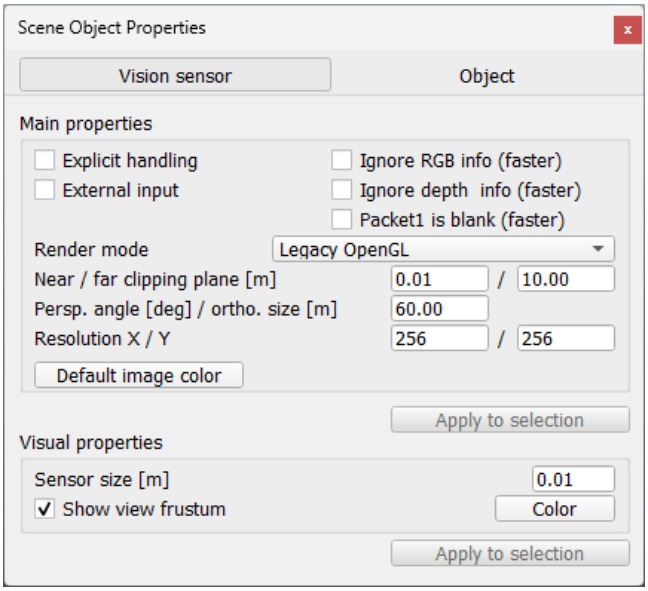

Рисунок 2.7 – Налаштування датчика зору

Потрібно змінити сцену та додати 3 куби різного кольору (рис. 2.8).

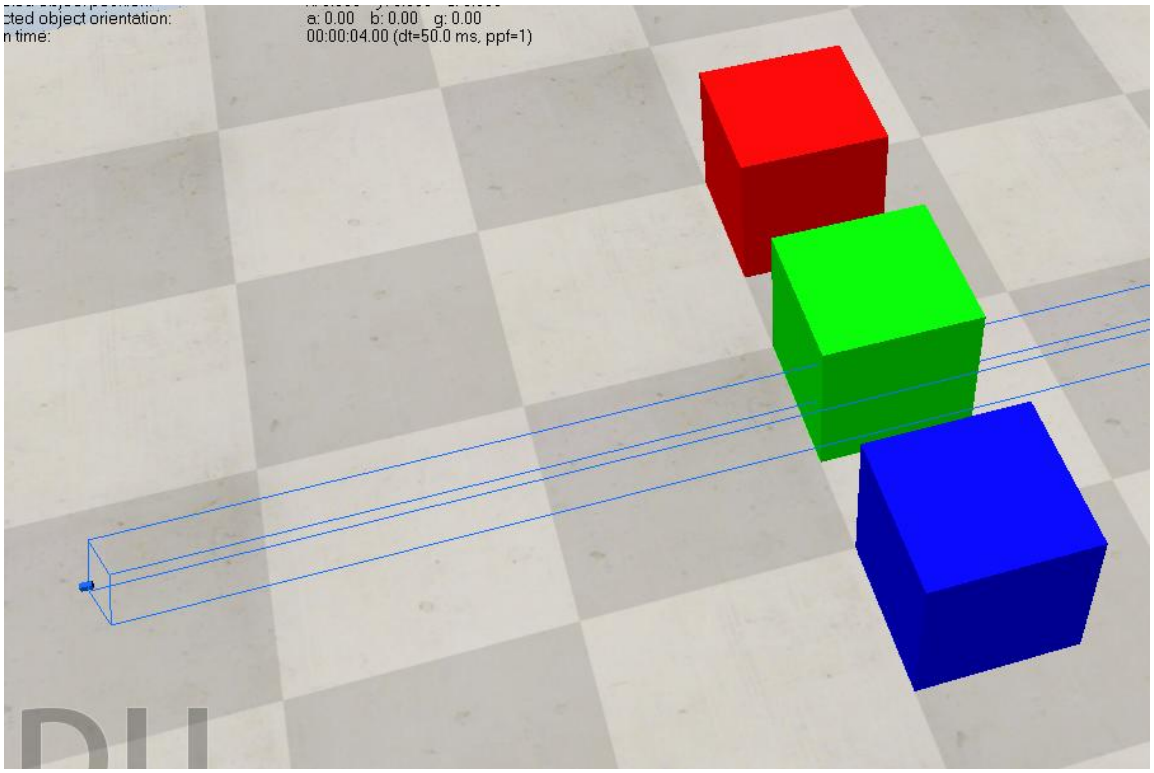

Рисунок 2.8 – Змінена сцена з доданими кубами різного кольору

Для того, щоб отримати інформацію, в тілі функції sysCall_init() потрібно оглосити змінну датчика:

vision = sim.getObject('/visionSensorOr').

Потім потрібно всередині функції sysCall_sensing() використати команду:

result, data_sensor = sim.readVisionSensor(vision).

Тепер змінна data_sensor зберігає всю інформацію про кольори та яскравість світла на датчику.

Наступним кроком потрібно її вивести у консоль:

print('Minimum intensity, r,g,b, depth: ' .. data_sensor[1].. ' ' ..data_sensor[2].. ' ' ..data_sensor[3].. ' ' ..data_sensor[4]..' ' .. data_sensor[5])

print('Maximum intensity, r,g,b, depth: ' .. data_sensor[6].. ' ' ..data_sensor[7].. ' ' ..data_sensor[8].. ' ' ..data_sensor[9]..' ' .. data_sensor[10])

print('Average intensity, r,g,b, depth: ' .. data_sensor[11].. ' ' ..data_sensor[12].. ' ' ..data_sensor[13].. ' ' ..data_sensor[14]..' ' .. data_sensor[15])

Отримаємо показання для кожного з кольорів (рис. 2.9 – рис. 2.11).

```
Minimum intensity, r,g,b, depth: 0.0 0.0 0.0 0.0 0.12212211638689
Maximum intensity, r,g,b, depth: 0.25490196078431 0.047058823529412 0.67843137254902 0.047058823529412 1.0
Average intensity, r,g,b, depth: 0.062745098039216 0.011764705882353 0.16862745098039 0.011764705882353 0.7805083990097
```

Рисунок 2.9 – Показання для зеленого кольору

```
Minimum intensity, r,g,b, depth: 0.0 0.0 0.0 0.0 0.12462461739779
Maximum intensity, r,g,b, depth: 0.25490196078431 0.67843137254902 0.047058823529412 0.047058823529412 1.0
Average intensity, r,g,b, depth: 0.1921568627451 0.50588235294118 0.035294117647059 0.035294117647059 0.3436244726181
```

Рисунок 2.10 – Показання для червоного кольору

```
Minimum intensity, r,g,b, depth: 0.0 0.0 0.0 0.0 0.10960961133242
Maximum intensity, r,g,b, depth: 0.25490196078431 0.047058823529412 0.047058823529412 0.67843137254902 1.0
Average intensity, r,g,b, depth: 0.1921568627451 0.035294117647059 0.035294117647059 0.50588235294118 0.33206593990326
```

Рисунок 2.11 – Показання для синього кольору

Слід зауважити, що дані трохи неточні, тому що впливає загальне освітлення сцени і те що відстань між сенсором і об'єктом достатньо велика.

2.3 Теорія автоматичного керування роботом на основі датчика наближення

В системі присутні такі елементи:

– об'єкт керування – робот ($W_R(s)$);

– датчик наближення – $W_s(s)$;

– регулятор – $W_c(s)$.

Швидкість робота пропорційна вхідному сигналу від контролера, а відстань є інтегралом швидкості. Тобто, якщо контролер видає сигнал *U(s),* який визначає швидкість, то зміна положення (відстані) *ΔX(s)* буде:

$$\Delta X(s) = \frac{K_R}{s} U(s) \tag{2.1}$$

де K_R – коефіцієнт підсилення робота, що перетворює керуючий сигнал на швидкість. Для простоти він включає в себе всі динамічні властивості двигуна та кінематики робота, які перетворюють керуючий сигнал на зміну відстані. Якщо U(s) – це команда швидкості, то $K_R = 1$

Отже, передавальна функція робота (від керуючого сигналу до зміни відстані) матиме наступний вигляд:

$$W_R(s) = \frac{1}{s} \tag{2.2}$$

Датчик наближення зазвичай має швидку динаміку порівняно з роботом. Для спрощення, ми можемо моделювати його як пропорційний елемент з певним коефіцієнтом підсилення K_s:

$$W_s(s) = K_s \tag{2.3}$$

де K_s – коефіцієнт перетворення фізичної відстані в сигнал датчика (наприклад, вольти або цифрові одиниці). Якщо датчик ідеально передає відстань в тих же одиницях, то $K_s = 1$.

Передавальна функція ПІД-регулятора в області Лапласа виглядає так:

$$W_c(s) = \frac{K_d s^2 + K_p s + K_i}{s} \tag{2.4}$$

Тоді передавальна функція розімкненої системи матиме вигляд:

$$W_{\text{р}}(s) = \frac{K_s(K_d s^2 + K_p s + K_i)}{s^2} \tag{2.5}$$

Замкненої системи відповідно:

$$W_{\text{з}}(s) = \frac{K_s(K_d s^2 + K_p s + K_i)}{s^2 + K_s(K_d s^2 + K_p s + K_i)} \tag{2.6}$$

У випадку, коли $K_p = 5,\ K_i = 1, K_d = 0{,}5$, отримаємо таку передавальну функцію:

$$W_{\text{з}}(s) = \frac{0{,}5s^2 + 5s + 1}{1{,}5s^2 + 5s + 1}$$

Наступним кроком перевіримо систему на стійкість. Спершу побудуємо перехідну характеристику системи (рис. 2.12) [33].

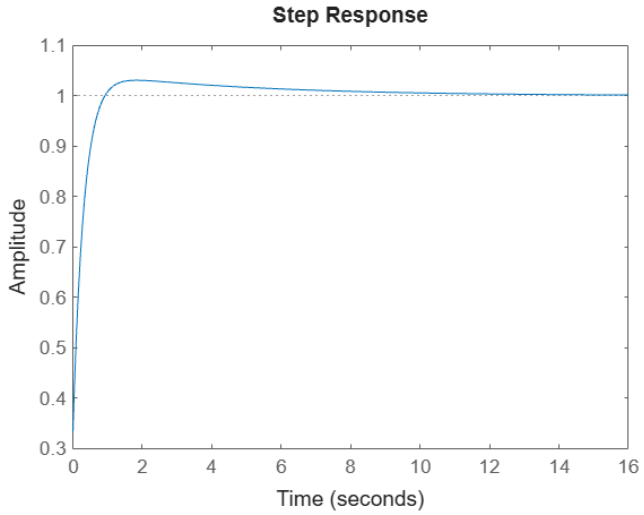

Рисунок 2.12 – Перехідна характеристика системи

Наступним кроком перевіримо стійкість за кореневим критерієм та побудуємо графік (рис. 2.13).

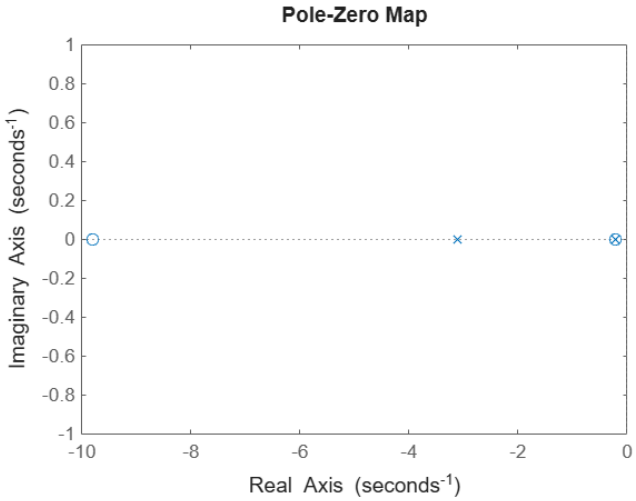

Рисунок 2.13 – Корені системи

Так як всі корені є лівими, то система є стійкою за кореневим критерієм.

Далі перевіримо стійкість за критерієм Найквіста. Кількість правих коренів дорівнює нулю, отже за критерієм система не повинна охоплювти точку (-1, j0) жодного разу. Побудуємо графік (рис. 2.14).

Система є стійкою за критерієм Найквіста, годограф жодного разу не охоплює відповідну точку.

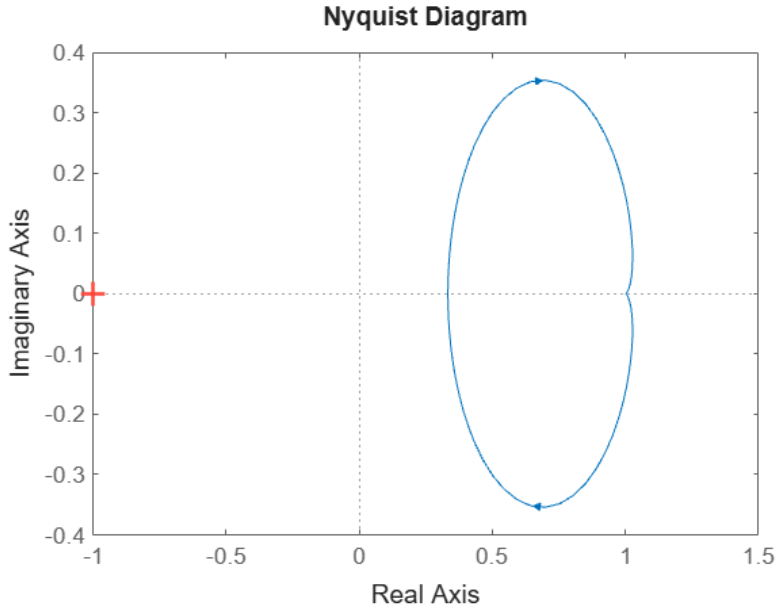

Рисунок 2.14 – Годограф Найквіста

2.4 Висновки за розділом 2

В ході виконання 2 розділу було розглянуто можливості моделювання сенсорів у симуляторі CoppeliaSim.

Було розглянуто 2 типи датчиків:

– датчики наближення;

– датчики зору.

Також проведено роботу кожного з датчиків та отримано програмні результати, що демонструють можливість подальшого моделювання сенсорної системи роботів.

Було проведено розрахунки теорії автоматичного керування роботом на основі датчика наближення, отримано передавальні функції та перевірено їх на стійкість.

3 МОДЕЛЮВАННЯ МОБІЛЬНОГО РОБОТА У СИМУЛЯТОРІ COPPELIASIM

3.1 Розроблення моделі робота з диференціальним приводом

Для створення моделі потрібно створити новий об’єкт типу кубоїд, що буде використовуватись як корпус робота (рис. 3.1-3.3) з параметрами довжини – 0,5 м, ширини – 0,3 м, висоти – 0,05 м.

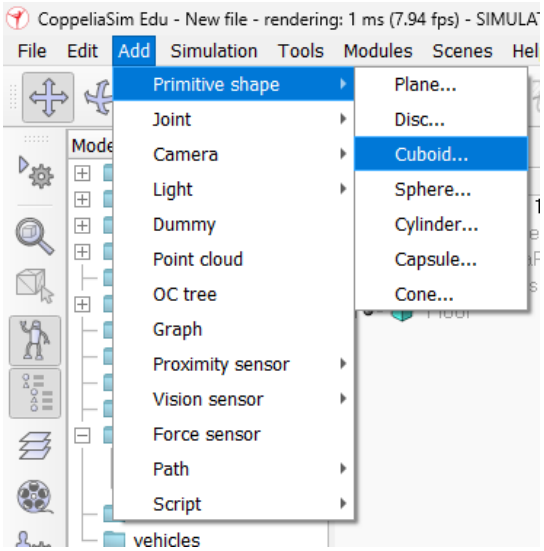

Рисунок 3.1 – Додавання кубоїду

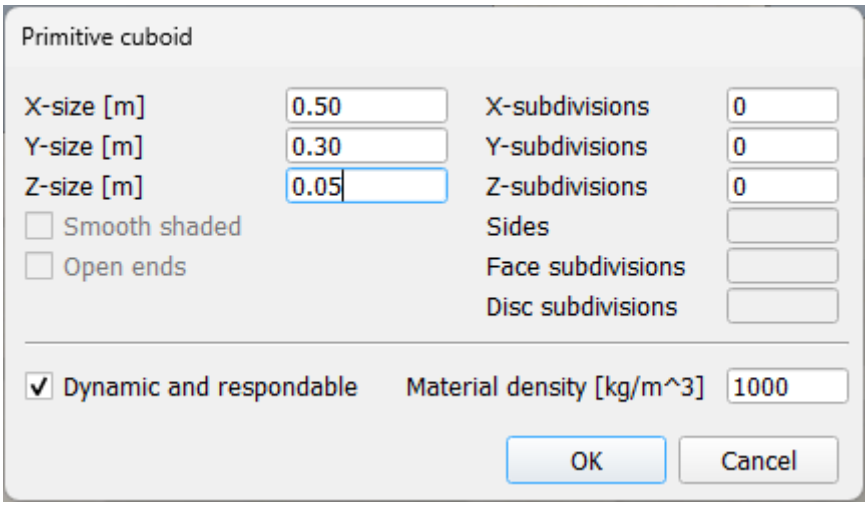

Рисунок 3.2 – Параметри кубоїду

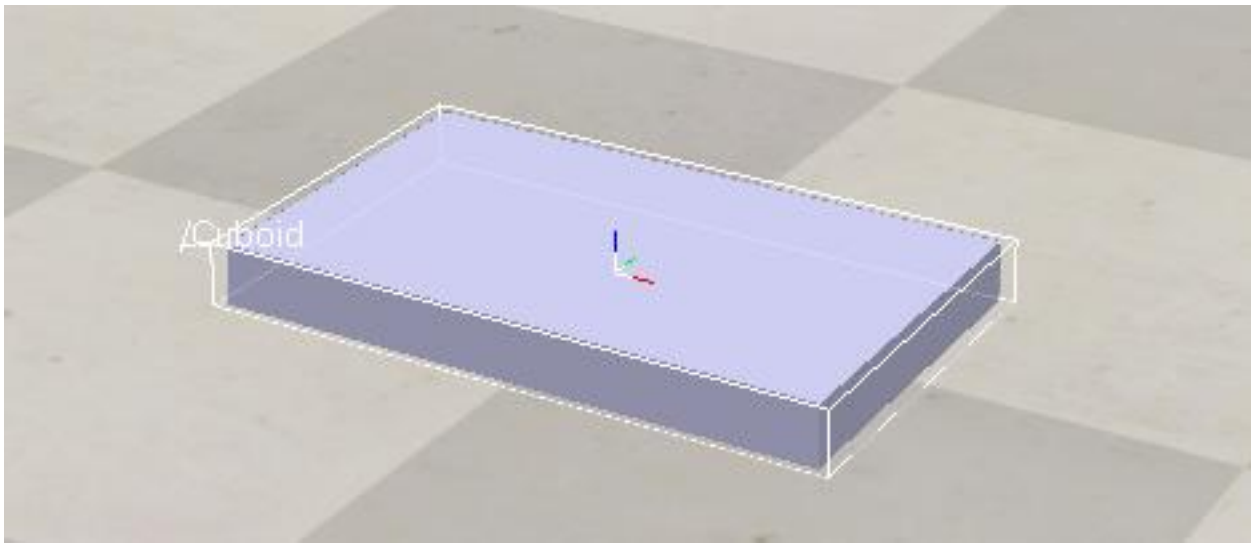

Рисунок 3.3 – Створений кубоїд

Наступним кроком потрібно в налаштуваннях об’єкту змінити його назву та колір (рис. 3.4).

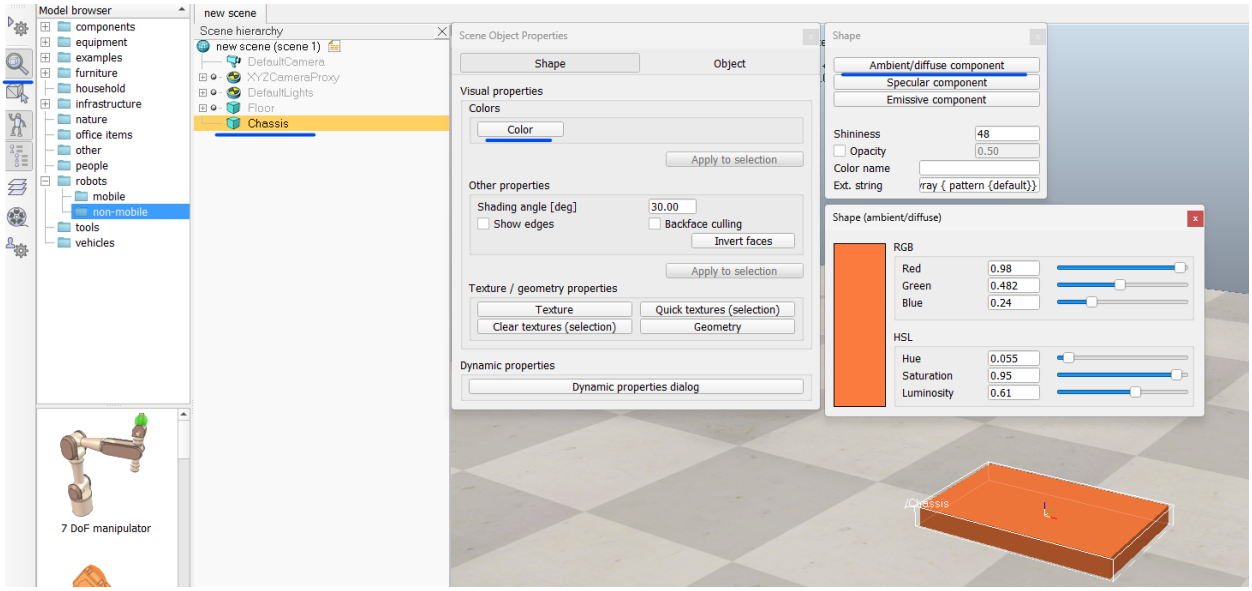

Рисунок 3.4 – Змінені налаштування кубоїду

Далі потрібно його підняти над сценою по осі z (рис. 3.5) на 0,1 м.

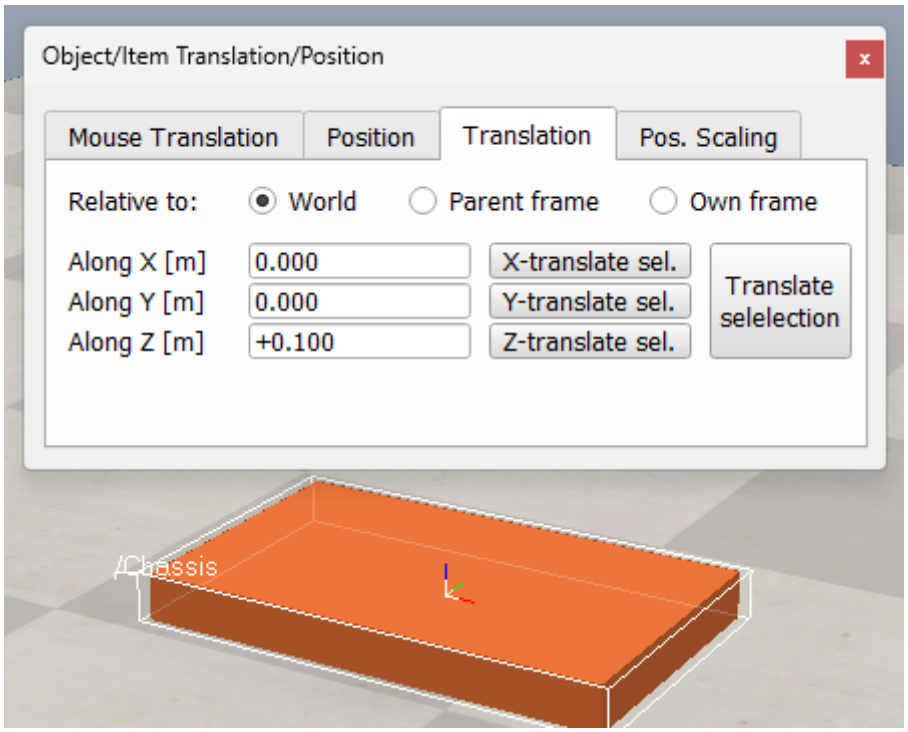

Рисунок 3.5 – Підняття кубоїду

Потрібно створити колеса, для цього треба додати Revolute_joint та відразу перейменувати його у Joint_right, що буде відповідати за правий двигун робота (рис. 3.6).

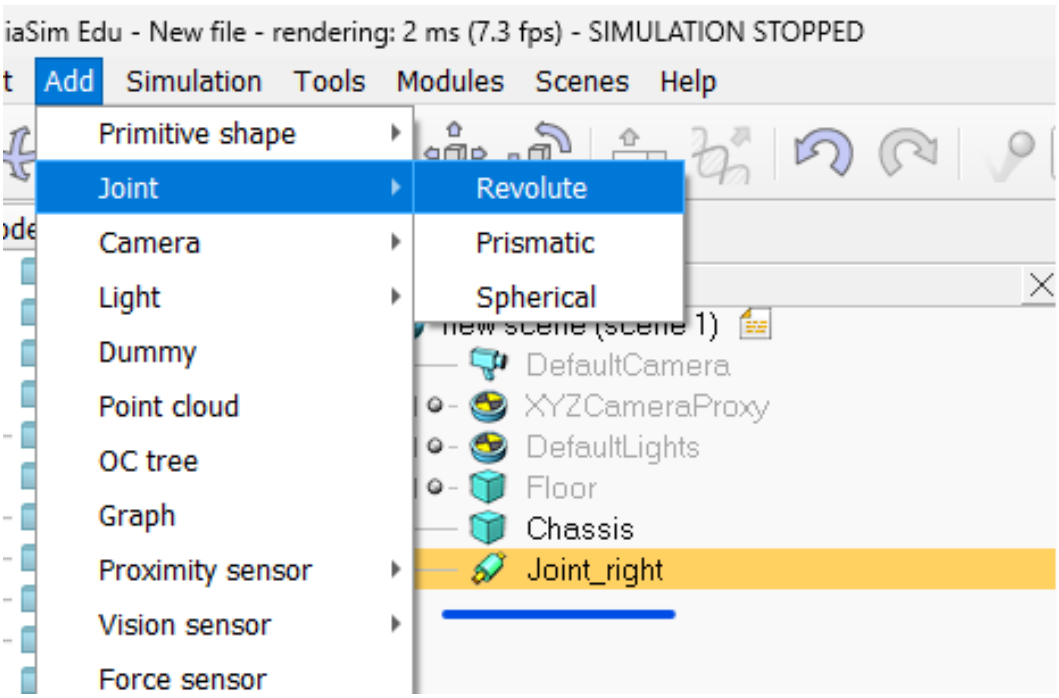

Рисунок 3.6 – Створення двигуна для робота

Для більшої наочності потрібно його перенести, а також змінити візуальні дані (рис. 3.7).

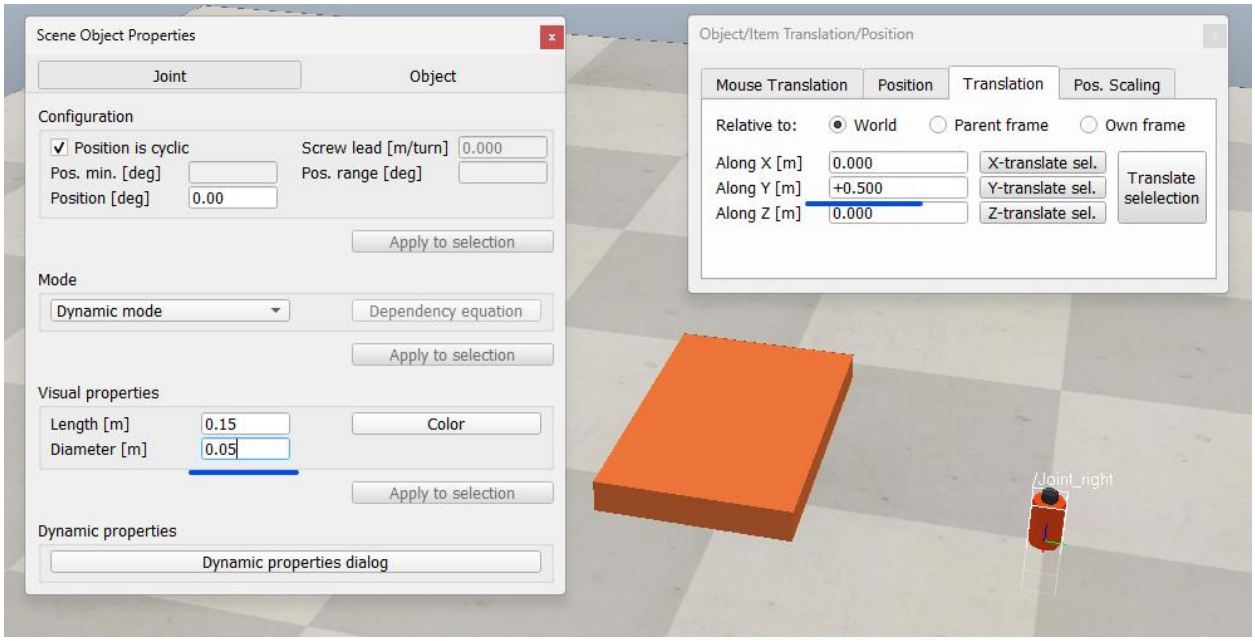

Рисунок 3.7 – Зміни двигуна для робота

Далі потрібно додати колесо яке буде представлено у вигляді циліндру з діаметром 0,2 м, товщиною – 0,05 м (рис. 3.8).

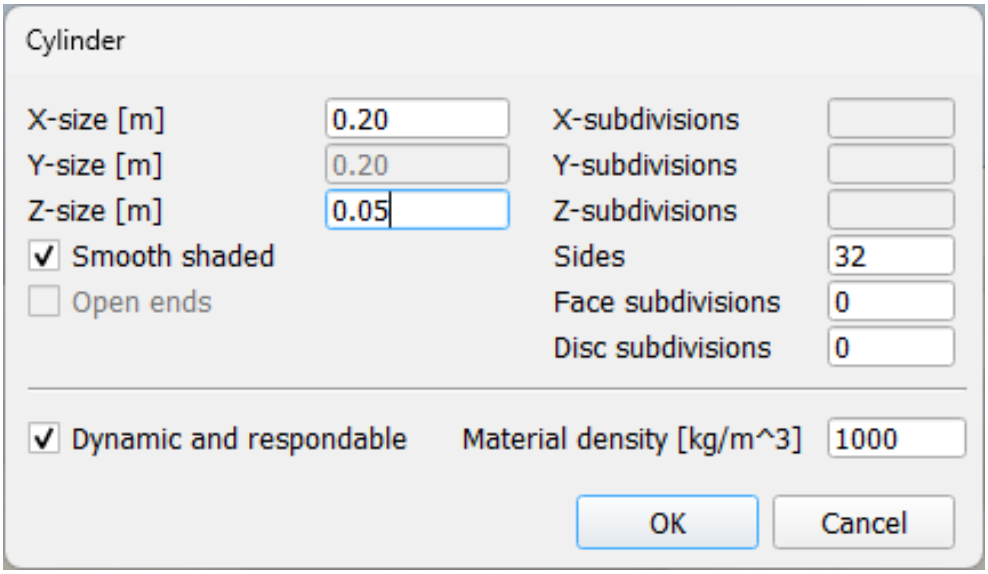

Рисунок 3.8 – Параметри колеса робота

Наступним кроком потрібно зв’язати колесо та мотор, для цього його потрібно перетягнути у ієрархії сцени циліндр-колесо на мотор-з’єднання, а також виставити параметри позиціонування на Parent frame, а також усі координати у 0 м, щоб вийшло колесо з двигуном (рис. 3.9).

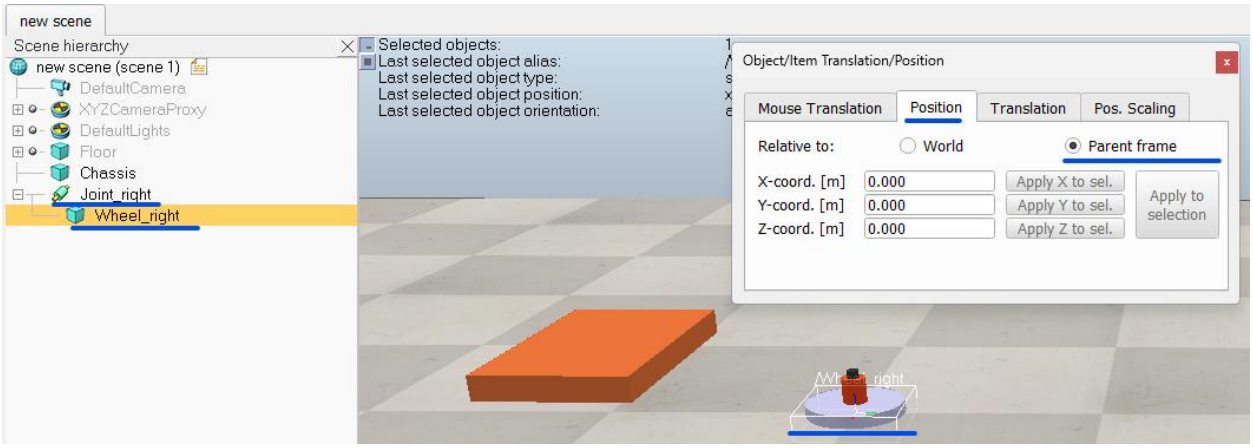

Рисунок 3.9 – Маніпуляції з колесом робота

Наступним кроком виконаємо перенесення колеса у ієрархії для того, щоб воно залежало від корпусу робота та його поворот відповідним чином, як показано на рис. 3.10.

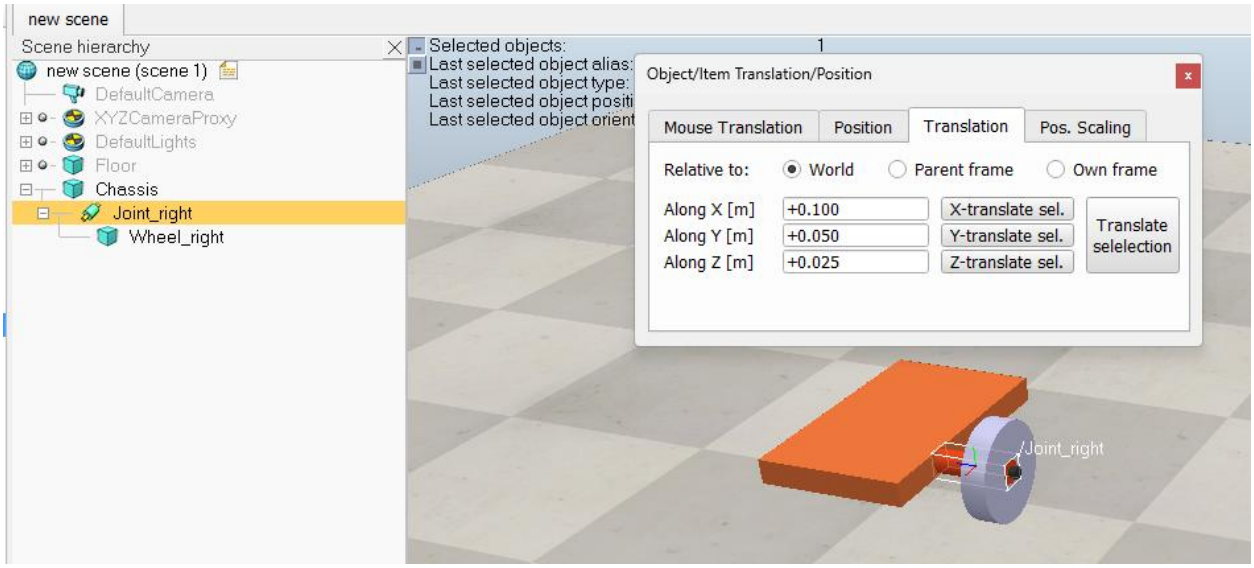

Рисунок 3.10 – Положення колеса робота

Аналогічним чином потрібно створити і додати друге колесо (рис. 3.11), а також змінити їх кольори.

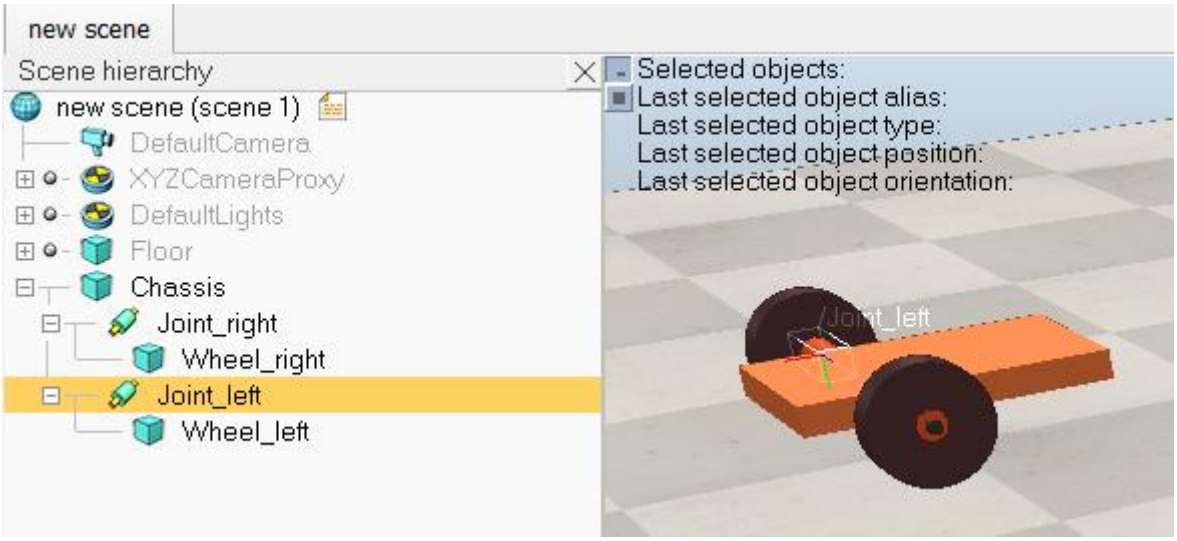

Рисунок 3.11 – Додавання другого колеса робота

3.2 Розроблення програми керування простою моделлю робота

Потрібно додати скрипт керування мовою Lua (рис. 3.12), відкрити його для написання програми.

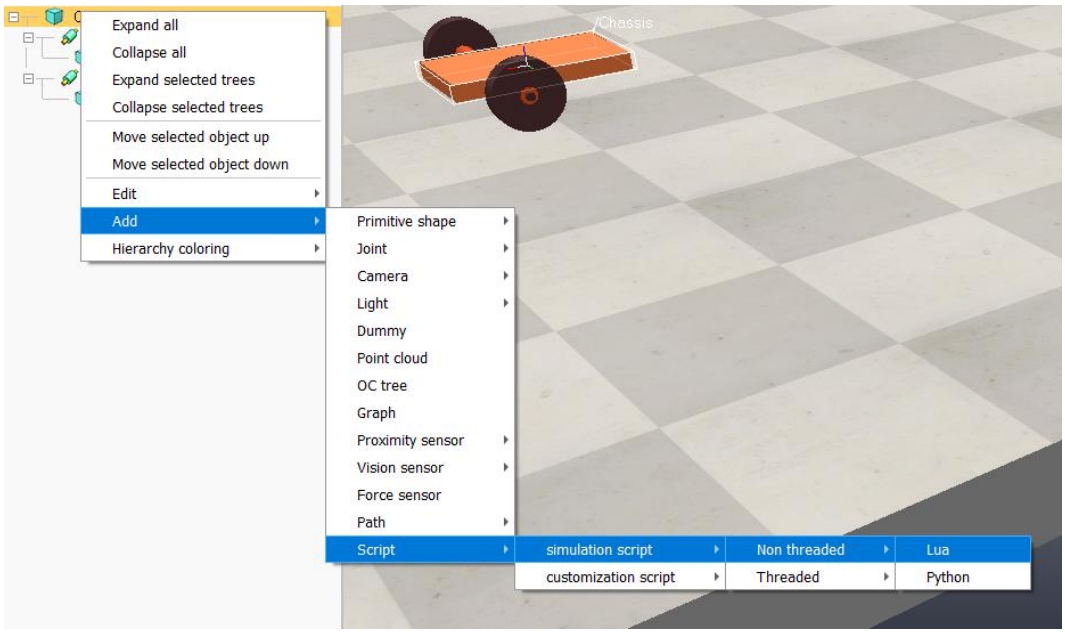

Рисунок 3.12 – Додавання скрипту керування

Першим кроком потрібно обрати створені двигуни, тобто створити відповідні змінні:

joint_left = sim.getObject('/Joint_left')
joint_right = sim.getObject('/Joint_right').

Далі встановити параметри керування двигуном, а також встановити його швидкість для правого колеса:

sim.setObjectInt32Param(joint_right, 2000,1)
sim.setJointTargetVelocity(joint_right, 0.5).

Аналогічним чином створити і для лівого колеса, лише з відмінністю у параметрі швидкості, бо колесо буде крутитися у зворотному напрямку:

sim.setObjectInt32Param(joint_left, 2000,1)
sim.setJointTargetVelocity(joint_left, -0.5).

Ця проста програма дозволяє роботу рухатись уперед (рис. 3.13).

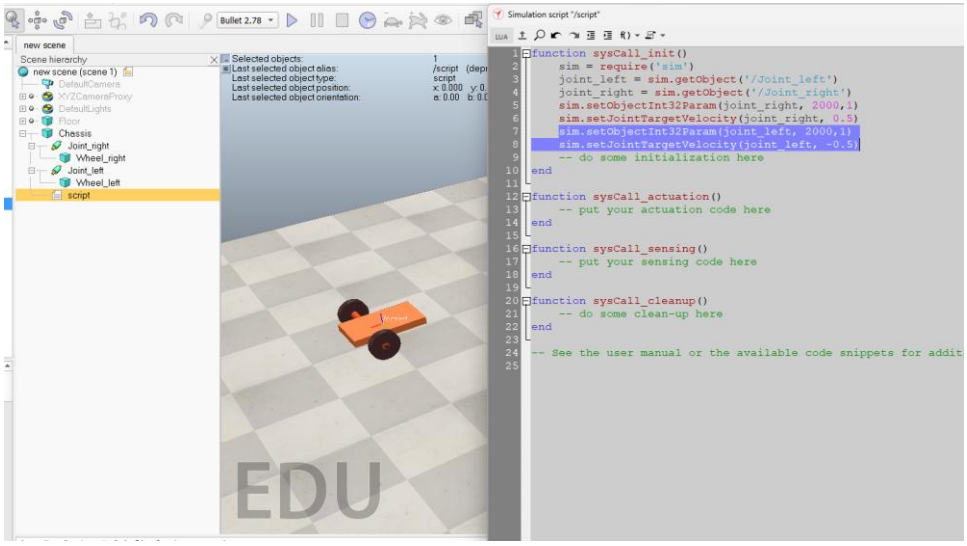

Рисунок 3.13 – Програма керування роботом для руху вперед

3.3 Отримання даних і їх візуалізація

Ці дані отримуються програмно. Спочатку потрібно створити об’єкт робота, або його корпусу Chassis:

chassis = sim.getObject('/Chassis').

Далі, в середині функції sysCall_sensing(), що відповідає за сенсори робота потрібно додати код для отримання позиціонування робота у просторі модельованого світу:

chassis_position = sim.getObjectPosition(chassis, sim.handle_world).

Параметр sim.handle_world використовується для отримання абсолютного позиціонування.

Для того, щоб вивести розташування робота за осями x та y, можна використати наступні функції:

print ('x=' .. chassis_position[1])
print ('y=' .. chassis_position[2]).

Тоді отримаємо наступну інформацію (рис. 3.14).

Для того, щоб потримати інформацію про швидкість та кутову швидкість робота, можна використати функцію:

chassis_linearVelocity, chassis_angularVelocity = sim.getObjectVelocity(chassis).

Виведення даних доступно за наступними командами (рис. 3.15):

```
print ('lV=' .. chassis_linearVelocity[1])
print ('aV=' .. chassis_angularVelocity[1]).
```

```
y=-0.21861067530847
x=-1.2018574342942
y=-0.21944832537756
x=-1.203771299377
y=-0.22032036087675
x=-1.2056983907962
y=-0.22118958131111
x=-1.20763507756
y=-0.22204032357063
x=-1.2095736785319
y=-0.22289163172261
x=-1.2115198855615
y=-0.22374921962868
x=-1.213442640213
y=-0.22460005410552
```

Рисунок 3.14 – Дані про позиціонування робота

```
aV=0.00018077960985865
lV=-0.044175157006578
aV=0.00020272375124814
lV=-0.044177275254209
aV=0.00014534602835849
lV=-0.044081662977362
aV=-0.0001178345415351
lV=-0.044085781531362
aV=-4.6923226183468e-05
lV=-0.044244898309904
aV=0.00032142912537186
lV=-0.044431624332602
aV=0.00028820383069133
lV=-0.043985774544493
aV=8.7179152547405e-06
```

Рисунок 3.15 – Дані про швидкість робота

Також є можливість слідкувати за параметрами робота, використовуючи графіки. Для цього потрібно додати об’єкт Graph (рис. 3.16).

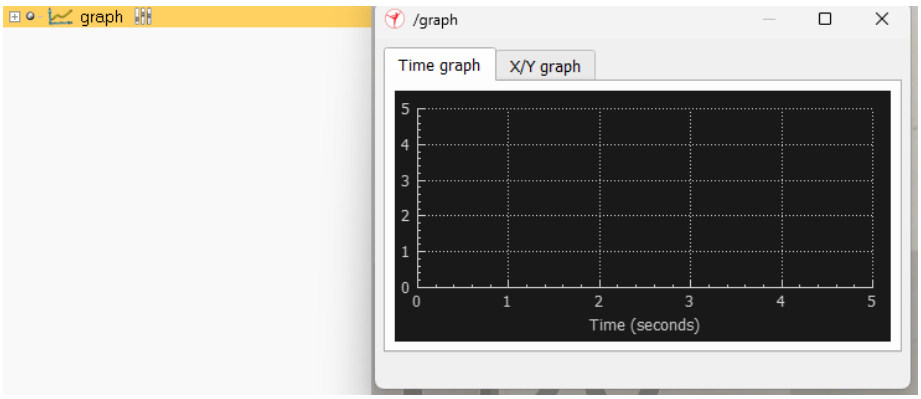

Рисунок 3.16 – Доданий графік

Далі потрібно додати об’єкт графіку:

graph = sim.getObject('/graph').

Щоб дані графіку змінювались, потрібно додати зміну швидкості. Спочатку потрібно створити змінну часу симуляції у функції sysCall_actuation():

time = sim.getSimulationTime().

Наступним кроком зробити умову щоб робот змінював швидкість лівого мотору через 3 секунди роботи:

```
if (time > 3) then
   sim.setJointTargetVelocity(joint_left, -0.2)
end.
```

Далі потрібно створити у функції sysCall_init() об’єкт потокового отримання даних:

chassis_lv = sim.addGraphStream(graph, 'vx', 'm/s', 0, {1,0,0}).

В даному випадку за віссю x буде час у секундах, за віссю y – зміна швидкості, назва графіку – vx, одиниці вимірювання – m/s, графік буде червоного кольору – {1,0,0}.

Результати представлені на рис. 3.17.

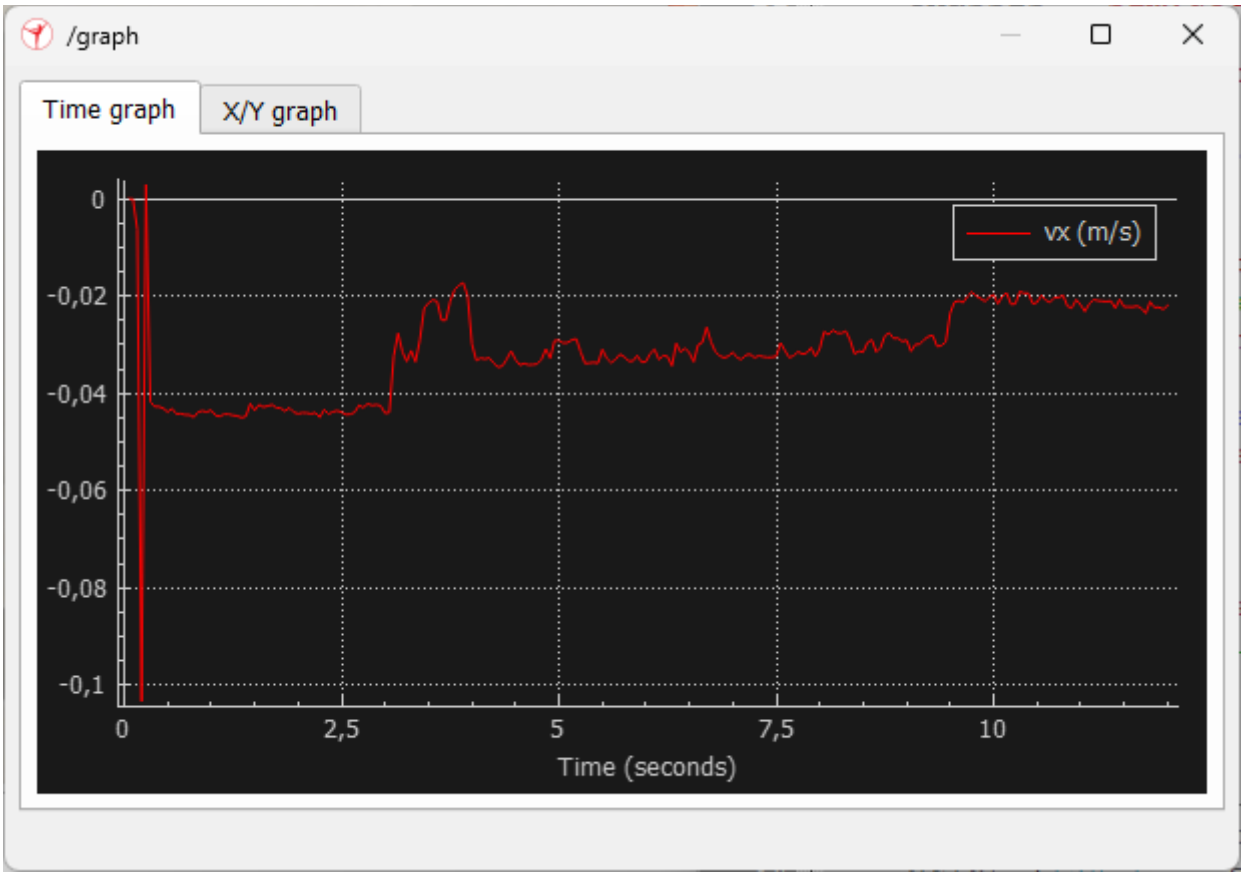

Рисунок 3.17 – Графік зміни швидкості робота за віссю x

Потрібно додати аналогічний код для відображення швидкості за віссю y.

Створюємо іншу зміну потоку зеленого кольору:

chassis_vy = sim.addGraphStream(graph, 'vy', 'm/s', 0, {0,1,0}).

Також передаємо до графіка значення швидкості за віссю y:

sim.setGraphStreamValue(graph, chassis_vy, chassis_linearVelocity[2])

Результат зміни швидкостей показано на рис. 3.18.

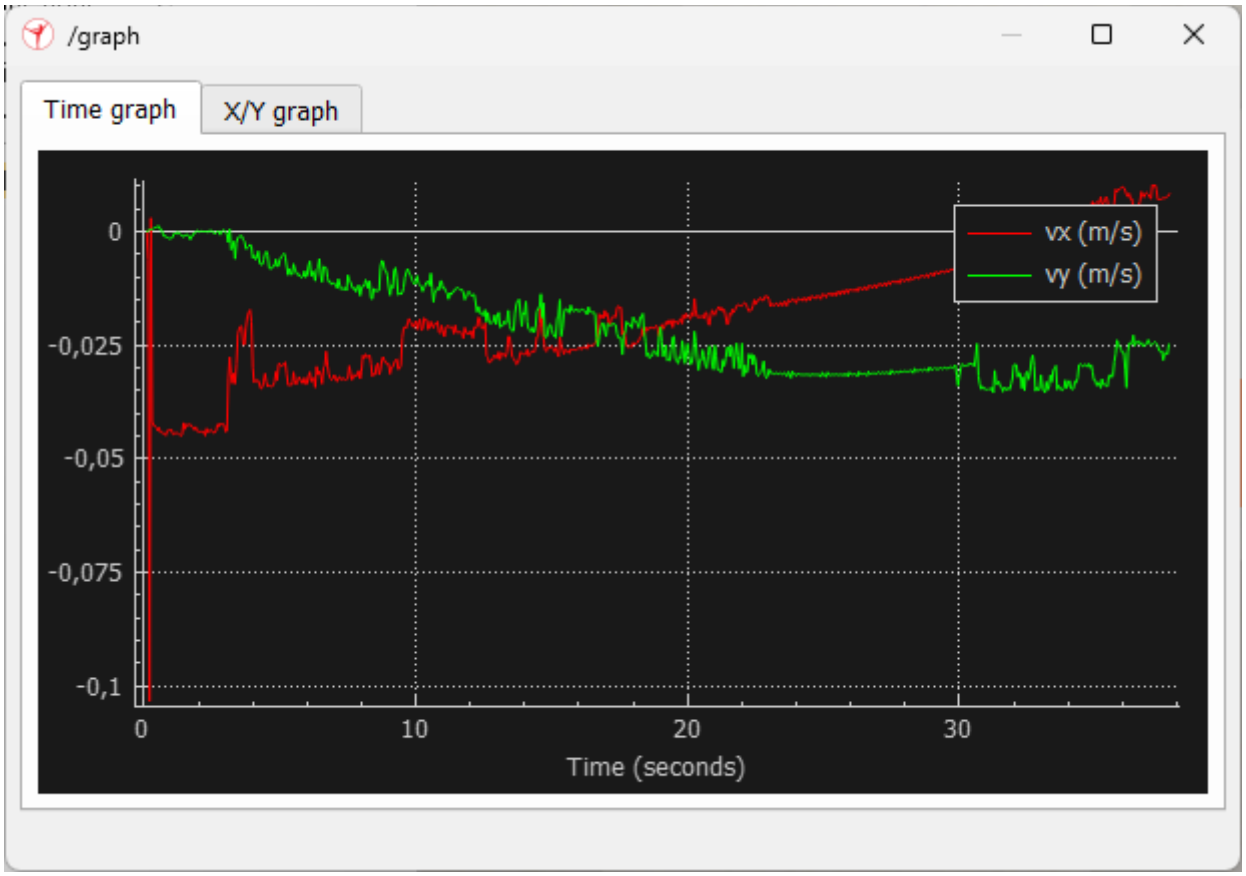

Рисунок 3.18 – Графік зміни швидкості робота за віссю x та y

3.4 Висновки до розділу 3

В результаті 3 розділу з використанням симулятора CoppeliaSim було побудовано модель робота з диференціальним приводом, а також досліджено можливість його програмування, отримання інформації з сенсорної системи, а також побудови графіків залежності його швидкостей лівого та правого коліс та відповідних моторів.

4 МОДЕЛЮВАННЯ СЕНСОРНОЇ СИСТЕМИ МОБІЛЬНОГО РОБОТА У СИМУЛЯТОРІ COPPELIASIM

4.1 Моделювання слідування за лінією

4.1.1 Моделювання керування швидкістю

Для моделювання прийнято рішення використовувати робот Line Tracer (рис. 4.1).

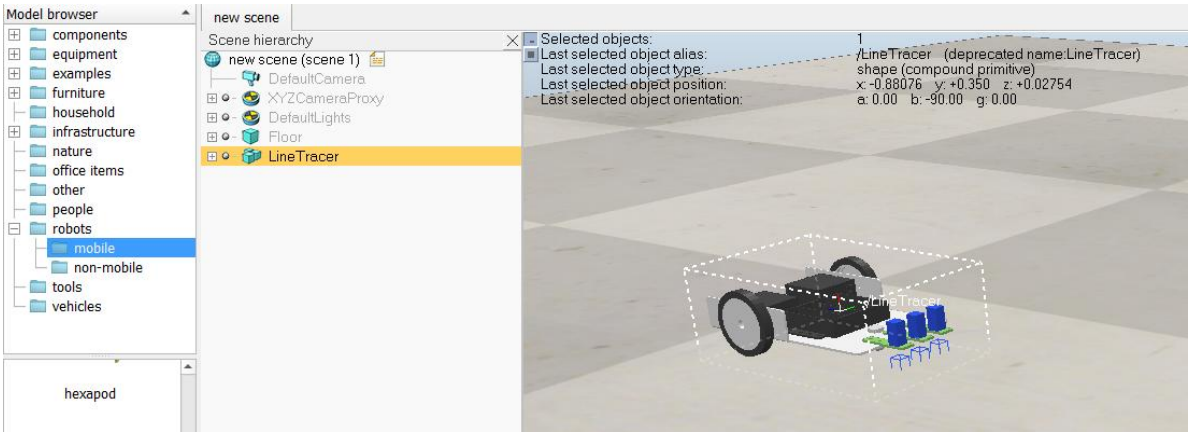

Рисунок 4.1 – Робот для моделювання

Далі потрібно змінити швидкості лівого та правого коліс (рис. 4.2).

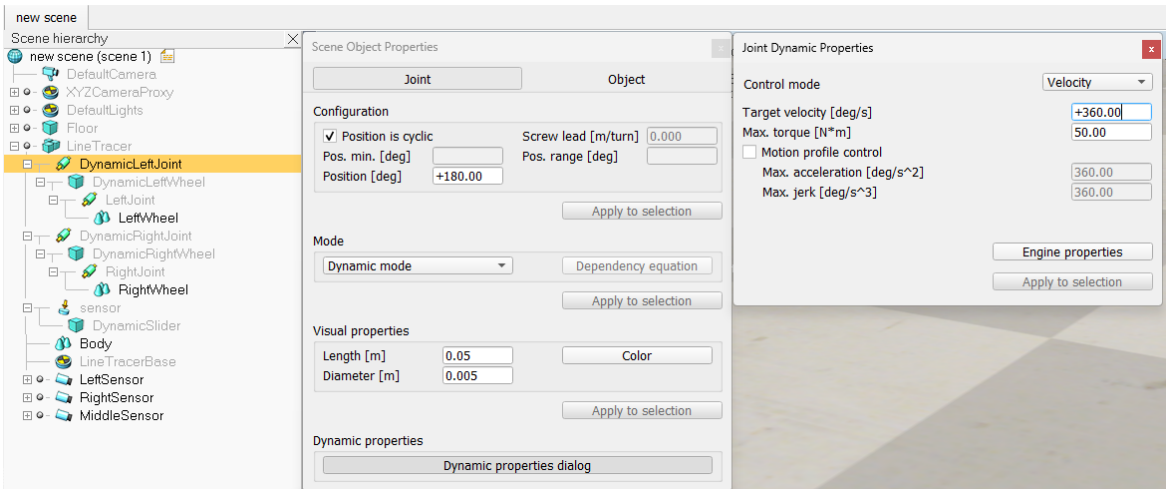

Рисунок 4.2 – Зміна швидкостей робота

Аналогічним чином потрібно виставити і для другого мотору швидкість у 360 градусів / с – тобто один повний оберт осі мотора.

Далі потрібно додати скрипт керування. Пустий має вигляд, представлений на рис. 4.3.

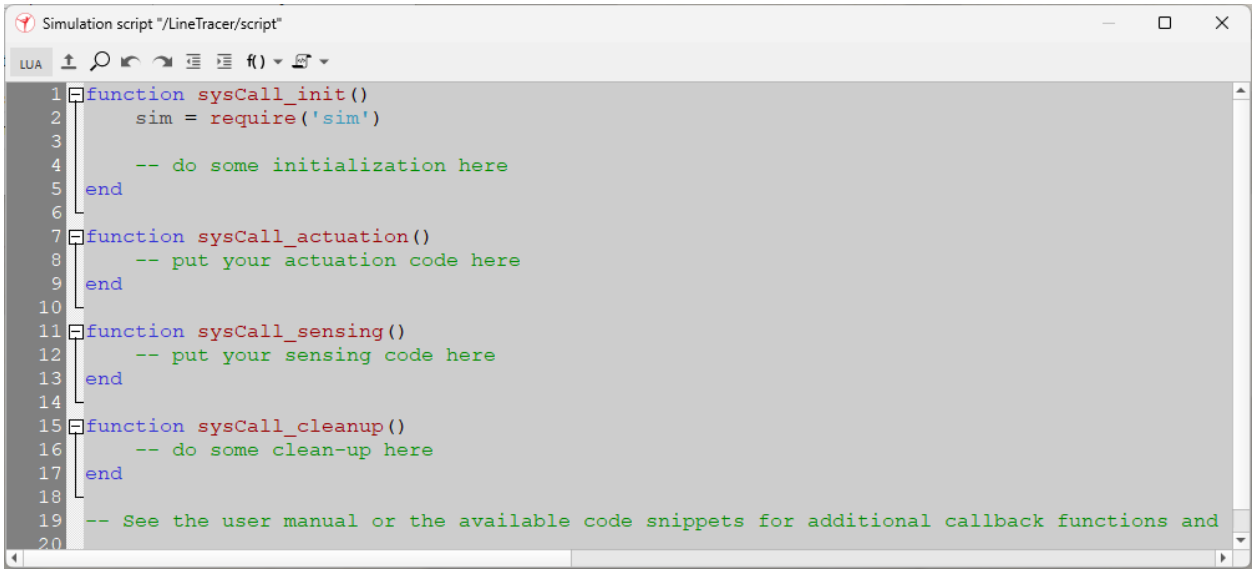

Рисунок 4.3 – Пустий скрипт керування

Наступною дією потрібно обрати двигуни робота, для цього в середині функції sysCall_init() додати наступний код для створення двох змінних двигунів:

leftJoint = sim.getObject('/DynamicLeftJoint')
rightJoint = sim.getObject('/DynamicRightJoint').

Щоб привести робота дію в середині функції sysCall_actuation() для нескінченого виконання потрібно додати код для встановлення швидкостей лівого та правого моторів відповідно:

sim.setJointTargetVelocity(leftJoint, 2)
sim.setJointTargetVelocity(rightJoint, 2).

Цей код дозволить роботу рухатись прямолінійно нескінчену кількість обертів (рис. 4.4).

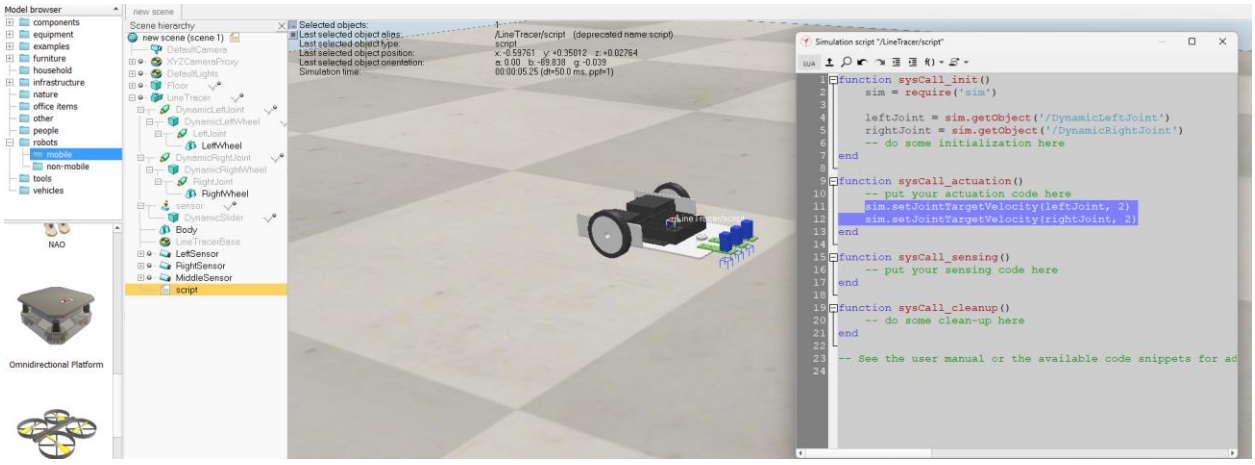

Рисунок 4.4 – Робот та скрипт керування

4.1.2 Моделювання сенсору лінії

Робот обладнаний 3 датчиками лінії (Vision Sensor), що представлені на рис. 4.5.

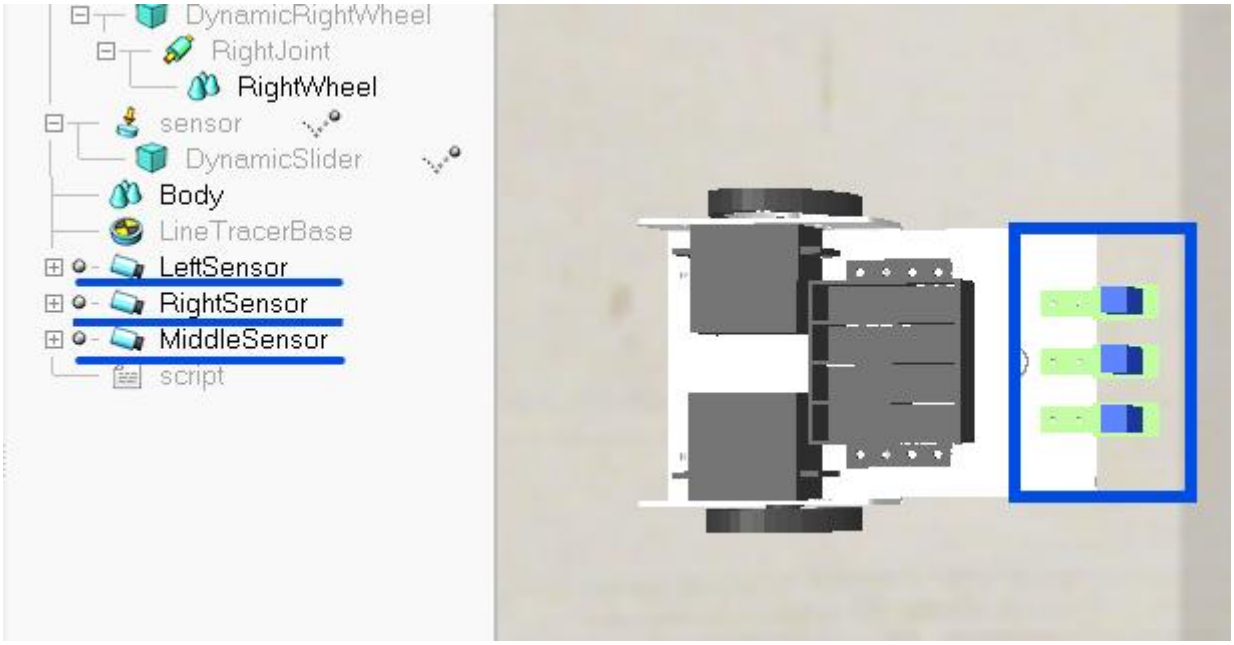

Рисунок 4.5 – Датчики лінії робота

Спочатку потрібно створити змінні сенсорів – дляя лівого, правого та центрального:

```
left_sensor = sim.getObject('/LeftSensor')
right_sensor = sim.getObject('/RightSensor')
middle_sensor = sim.getObject('/MiddleSensor').
```

Для того, щоб отримати дані з лівого датчика потрібно використати наступний код у середині функції sysCall_sensing(), що відповідає за сенсори робота:

```
result, data_left = sim.readVisionSensor(left_sensor).
```

Наступним кроком буде перевірка того, чи є дані, що отримуються з датчика нульовими:

```
if (data_left ~= nil) then …
```

Далі потрібно вивести всі дані з датчиків у консоль за допомогою команд:

```
print('Minimum intensity, r,g,b, depth: ' .. data_left[1]..' ' ..data_left[2]..' ' .. data_left[3]..' ' .. data_left[4]..' ' .. data_left[5])
print('Maximum intensity, r,g,b, depth: ' .. data_left[6]..' ' .. data_left[7].. ' ' ..data_left[8].. ' ' ..data_left[9].. ' ' ..data_left[10])
print('Average intensity, r,g,b, depth: ' .. data_left[11].. ' ' ..data_left[12].. ' ' ..data_left[13].. ' ' ..data_left[14]..' ' .. data_left[15]).
```

Тут у першому рядку виводиться інформація про мінімальну інтенсивність, кольори у форматі червоного, зеленого та сильного, а також про глибину яскравості.

Другий рядок служить для виведення максимальних значень, третій – для середніх значень.

Приклад показання з датчиків показано на рис. 4.6.

```
Minimum intensity, r,g,b, depth: 0.81960784313725 0.84705882352941 0.82745098039216 0.7921568627451 0.1683653742075
Maximum intensity, r,g,b, depth: 0.83529411764706 0.85882352941176 0.84313725490196 0.81176470588235 0.1689191609621
Average intensity, r,g,b, depth: 0.82745098039216 0.85098039215686 0.83529411764706 0.8 0.1686422675848

Minimum intensity, r,g,b, depth: 0.81960784313725 0.84705882352941 0.82745098039216 0.78823529411765 0.16841788589954
Maximum intensity, r,g,b, depth: 0.83529411764706 0.86274509803922 0.84313725490196 0.8078431372549 0.16898675262928
Average intensity, r,g,b, depth: 0.82745098039216 0.85098039215686 0.83529411764706 0.79607843137255 0.16870231926441

Minimum intensity, r,g,b, depth: 0.82352941176471 0.84705882352941 0.83137254901961 0.78823529411765 0.16847069561481
Maximum intensity, r,g,b, depth: 0.83529411764706 0.86274509803922 0.84313725490196 0.80392156862745 0.16905416548252
Average intensity, r,g,b, depth: 0.82745098039216 0.85490196078431 0.83529411764706 0.79607843137255 0.16876244544983

Minimum intensity, r,g,b, depth: 0.82352941176471 0.85098039215686 0.83529411764706 0.78823529411765 0.16852279007435
Maximum intensity, r,g,b, depth: 0.83529411764706 0.86274509803922 0.84313725490196 0.8078431372549 0.16912038624287
Average intensity, r,g,b, depth: 0.83137254901961 0.85882352941176 0.83921568627451 0.79607843137255 0.16882163286209

Minimum intensity, r,g,b, depth: 0.82352941176471 0.85490196078431 0.83529411764706 0.78823529411765 0.16857387125492
Maximum intensity, r,g,b, depth: 0.83921568627451 0.86666666666667 0.84313725490196 0.81176470588235 0.1691849976778
Average intensity, r,g,b, depth: 0.83137254901961 0.85882352941176 0.83921568627451 0.8 0.16887944936752

Minimum intensity, r,g,b, depth: 0.82745098039216 0.85490196078431 0.83529411764706 0.78823529411765 0.16862328350544
Maximum intensity, r,g,b, depth: 0.83921568627451 0.86666666666667 0.84705882352941 0.81176470588235 0.1692471653223
Average intensity, r,g,b, depth: 0.83137254901961 0.85882352941176 0.83921568627451 0.8 0.16893525421619

Minimum intensity, r,g,b, depth: 0.82352941176471 0.85098039215686 0.83921568627451 0.7843137254902 0.16867078840733
Maximum intensity, r,g,b, depth: 0.83921568627451 0.86666666666667 0.85098039215686 0.81176470588235 0.16930647194386
Average intensity, r,g,b, depth: 0.83137254901961 0.85882352941176 0.84313725490196 0.8 0.16898867487907
```

Рисунок 4.6 – Приклад показання з лівого датчику

Потрібно розробити аналогічний код для отримання середньої інформації з правого та центрального датчика:

```
result, data_right = sim.readVisionSensor(right_sensor)
result, data_middle = sim.readVisionSensor(middle_sensor).
```

Також виводити цю інформацію (рис. 4.7):

```
print('Average right intensity, r,g,b, depth: ' .. data_right[11].. ' ' ..data_right[12].. ' ' ..data_right[13].. ' ' ..data_right[14]..' ' .. data_right[15])
print('Average middle intensity, r,g,b, depth: ' .. data_middle[11].. ' ' ..data_middle[12].. ' ' ..data_middle[13].. ' ' ..data_middle[14]..' ' .. data_middle[15])
```

```
Average left intensity, r,g,b, depth: 0.83529411764706 0.85490196078431 0.83921568627451 0.81176470588235 0.16928668320179
Average right intensity, r,g,b, depth: 0.83529411764706 0.85490196078431 0.84313725490196 0.81176470588235 0.16883216798306
Average middle intensity, r,g,b, depth: 0.83529411764706 0.85882352941176 0.84313725490196 0.8078431372549 0.16905942559242

Average left intensity, r,g,b, depth: 0.83529411764706 0.85490196078431 0.83921568627451 0.81176470588235 0.16928043961525
Average right intensity, r,g,b, depth: 0.83529411764706 0.85490196078431 0.84313725490196 0.8078431372549 0.16885809600353
Average middle intensity, r,g,b, depth: 0.83529411764706 0.85882352941176 0.84313725490196 0.8078431372549 0.16906923055649

Average left intensity, r,g,b, depth: 0.83529411764706 0.85490196078431 0.83921568627451 0.8156862745098 0.16926783323288
Average right intensity, r,g,b, depth: 0.83529411764706 0.85490196078431 0.84313725490196 0.81176470588235 0.16887980699539
Average middle intensity, r,g,b, depth: 0.83529411764706 0.85882352941176 0.84313725490196 0.8078431372549 0.1690738350153
```

Рисунок 4.7 – Приклад показання з усіх датчиків

Створимо просту площину розміром в довжину 0,5 м., шириною 0,2 м. (рис. 4.8) для отримання даних з неї.

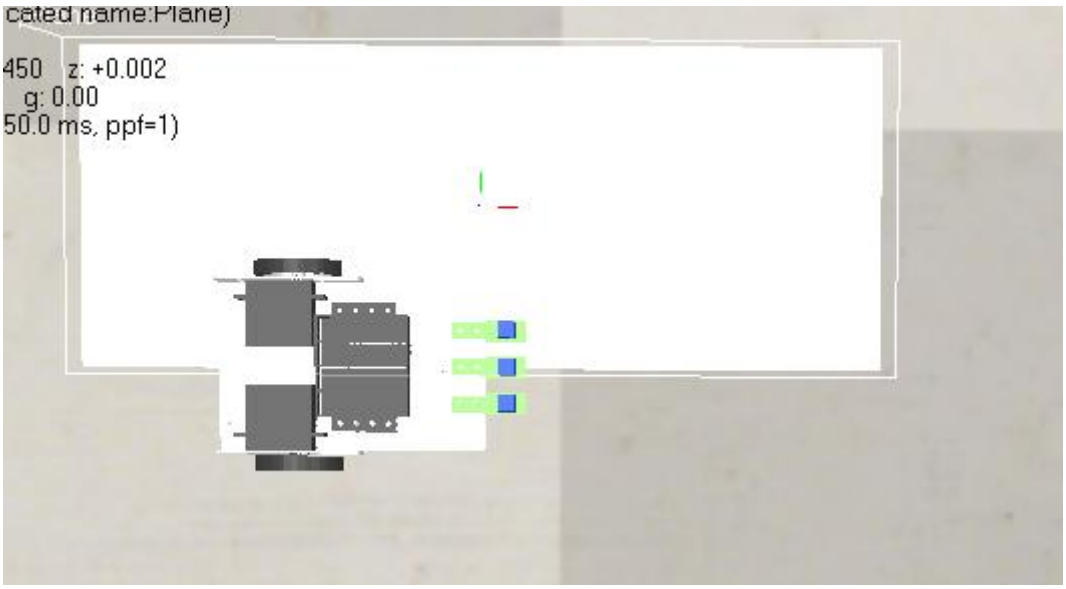

Рисунок 4.8 – Створення площини

Для того щоб подивитися на зміни показань, ротрібно запустити програму та отримати дані (рис. 4.9).

```
Average left intensity, r,g,b, depth: 1.0 1.0 1.0 1.0 0.12866035103798
Average right intensity, r,g,b, depth: 0.83137254901961 0.85882352941176 0.83529411764706 0.8 0.16816574335098
Average middle intensity, r,g,b, depth: 0.91372549019608 0.92549019607843 0.91764705882353 0.89803921568627 0.14841301739216
```

Рисунок 4.9 – Дані з датчиків під час руху по площини

Далі потрібно додати візуалізацію того, що бачить сенсор, для цього потрібно додати Floating View, настинуши правою кнопкою миші по сцені та обравши потрібний пункт (рис. 4.10).

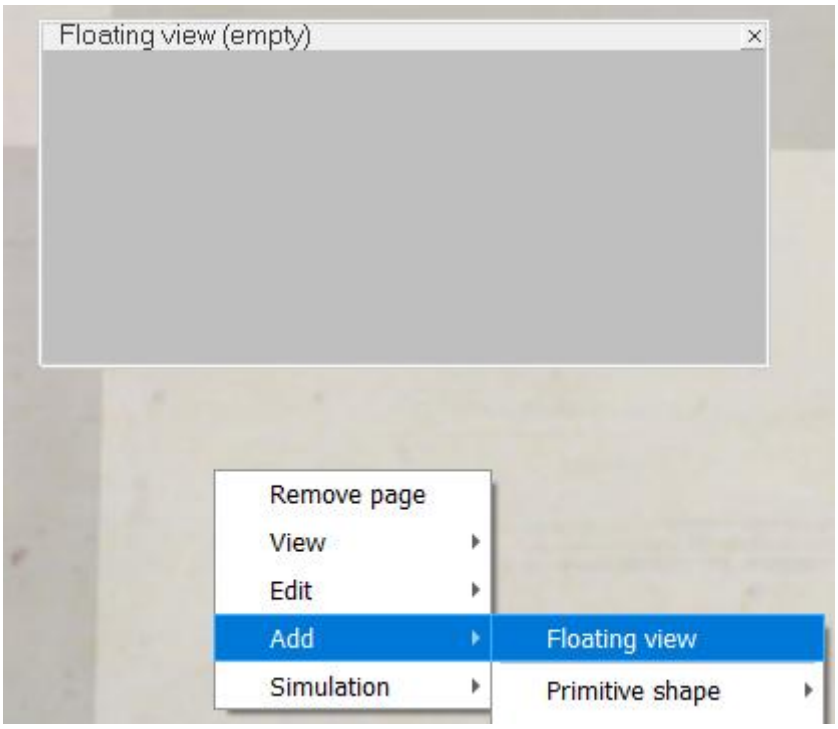

Рисунок 4.10 – Додавання Floating View

Наступним кроком є отримання даних з сенсору у це вікно, для цього потрібно обрати сенсор у ієрархії об’єктів та обрати певний пункти меню, як на рис. 4.11.

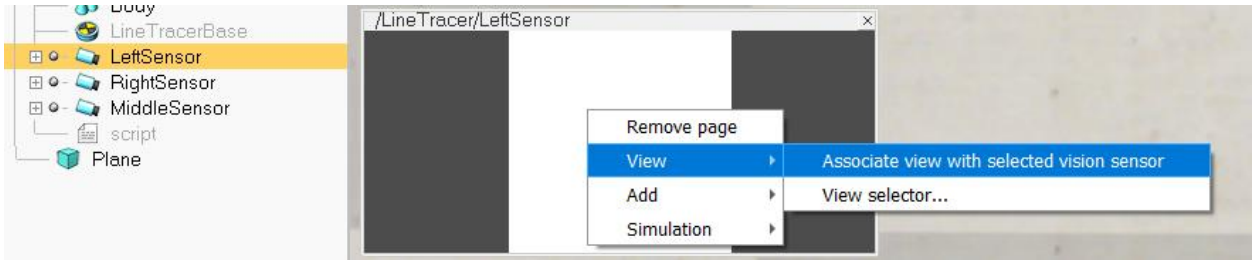

Рисунок 4.11 – Налаштування Floating View для лівого сенсору

Аналогічним чином потрібно зробити і для правого сенсору.

Тоді під час руху можна спостерігати наступні параметри в консолі та візуальні складові, що представлені на рис. 4.12.

Рисунок 4.12 – Візуалізація роботи датчиків

4.1.3 Алгоритм переміщення по лінії

Для створення програми руху по лінії, потрібно додати шлях переміщення (Path) з наступними налаштуваннями (рис. 4.13), а саме – збільшити розмір лінії-шляху, генерувати саму форму лінії, зробити її червоним кольором. Також потрібно змінити висоту цієї лінії-траси до 0,1 м.

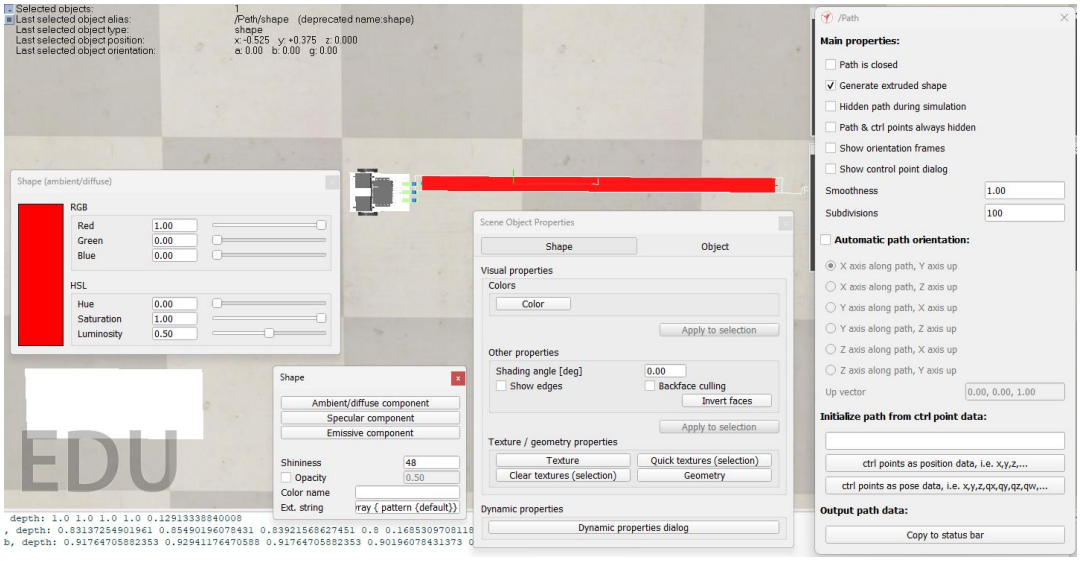

Рисунок 4.13 – Додавання лінії-шляху

Наступним кроком є створення програми для руху лінією.

Спочатку потрібно отримати дані інтенсивності для лівого та правого сенсорів та записати їх у функції sysCall_actuation(). Також слід написати умову стосовно того, що датчики отримують якісь значення і не є нульовими, лише тоді отримувати з них інформацію:

```
if (data_left ~= nil) then
    intensity_left = data_left[11]
    intensity_right = data_right[11]
end.
```

Для того щоб робот рухався лінією, потрібно додати декілька умов і тоді функція sysCall_actuation() матиме наступний вигляд:

```
function sysCall_actuation()
    -- put your actuation code here
    v = 2
    dv = 2

    sim.setJointTargetVelocity(leftJoint, v)
    sim.setJointTargetVelocity(rightJoint, v)
    if (data_left ~= nil) then
    intensity_left = data_left[11]
    intensity_right = data_right[11]

        if (intensity_right> 0.5 and intensity_left < 0.5) then
            print('Turn right')
            sim.setJointTargetVelocity(rightJoint, v+dv)
        end
        if (intensity_left> 0.5 and intensity_right < 0.5) then
```

```
            print('Turn left')
            sim.setJointTargetVelocity(leftJoint, v+dv)
        end
    end
end
```

Спочатку було створено 2 змінні – одна для завдання швидкості, інша – за її зміну (пропорційна та диференційна складові регулятору). Далі додано умову для правого датчика:

```
if (intensity_right> 0.5 and intensity_left < 0.5) then
            print('Turn right')
            sim.setJointTargetVelocity(rightJoint, v+dv)
        end
```

Якщо інтенсивність з правого датчика більше за 0,5, а інтенсивність лівого менше за 0,5, то встановлюємо швидкість правого мотору на суму двох змінних, що відповідають за швидкість і її зміну.

Аналогічним чином відбувається зміна швидкості і лівого мотора та колеса відповідно до показань з датчиків.

4.2 Моделювання слідування стіною за допомогою датчиків наближення

4.2.1 Моделювання оточення робота

Для моделювання було обрано робота, представленого на рис. 4.14.

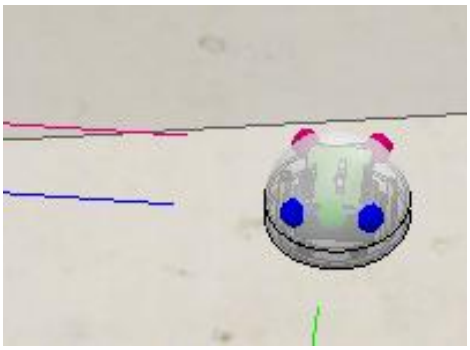

Рисунок 4.14 – Обраний робот для моделювання

Цей робот має 5 датчиків наближення – 1 спереду робота, 4 – по боках, при чому 2 знаходяться у передній частині, 2 у задній частині робота.

Кожен з бокових датчиків має наступні налаштування (рис. 4.15):

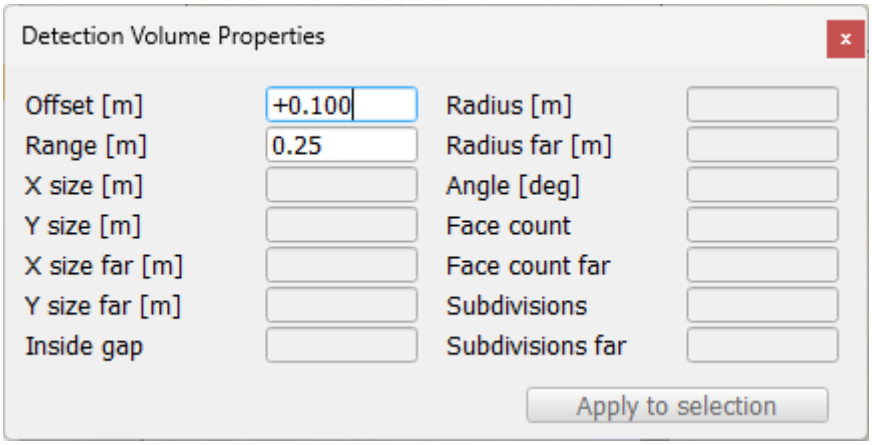

Рисунок 4.15 – Налаштування датчиків наближення

На даному рискну Offset – дальність спрацювання датчика, Range – дальність роботи датчика.

Передній датчик має змінену дальність роботи до 0,2 м.

Далі для того, щоб змоделювати стіну, потрібно додати кубоїд (Cuboid) з наступними параметрами, що представлені на рис. 4.16.

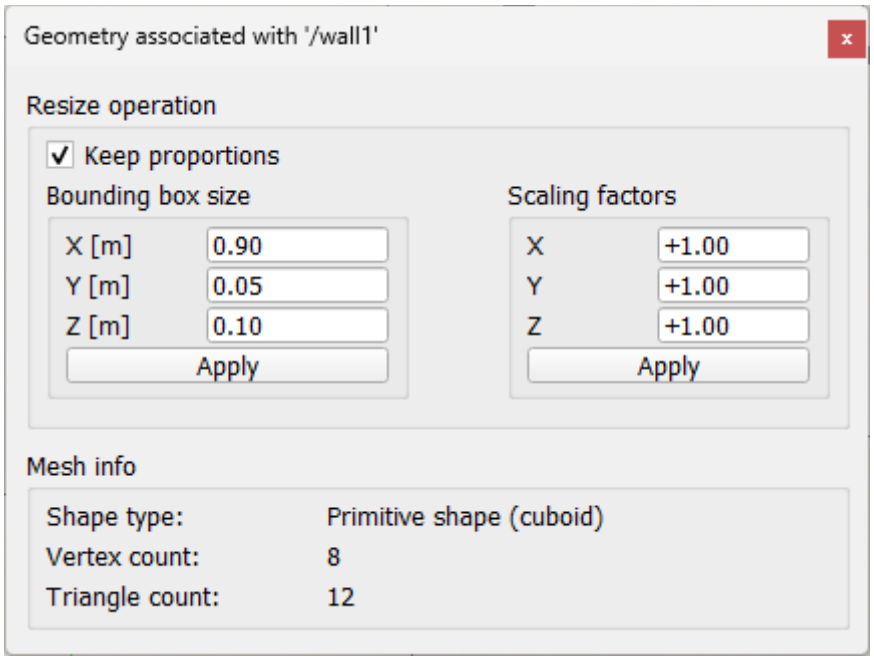

Рисунок 4.16 – Додавання кубоїду-стіни

Тоді модель буде мати наступний вигляд (рис. 4.17).

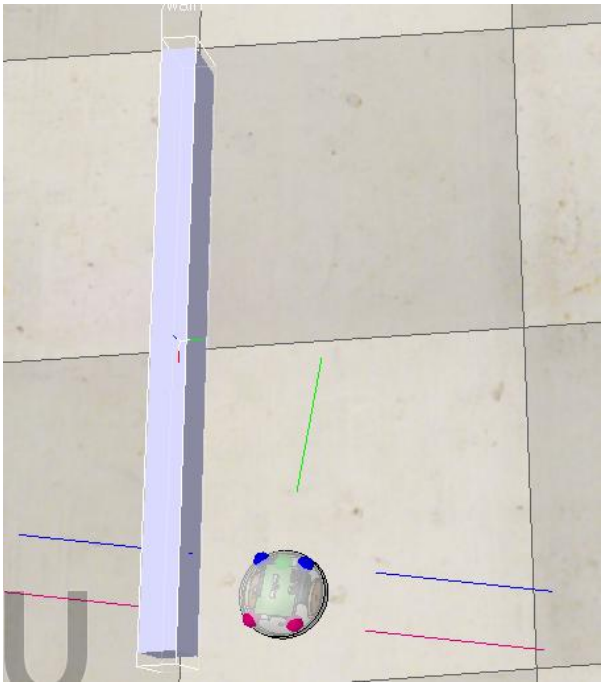

Рисунок 4.17 – Робот та стіна у сцені

4.2.2 Моделювання сенсорів наближення

Для того щоб провести моделювання, потрібно створити змінні, що відповідають за датчики та мотори в середині функції sysCall_init():

```
lmotor=sim.getObjectHandle("./lumibot_leftMotor")
rmotor=sim.getObjectHandle("./lumibot_rightMotor")
sensorLF=sim.getObjectHandle('./LeftFront')
sensorLR=sim.getObjectHandle('./LeftRear')
sensorRF=sim.getObjectHandle('./RightFront')
sensorRR=sim.getObjectHandle('./RightRear')
sensorF=sim.getObjectHandle('./Forward').
```

Щоб отримати значення з датчиків (спочатку для лівого датчика), в середині функції sysCall_sensing(), створимо команду:

```
flag1,LF=sim.readProximitySensor(sensorLF).
```

В даному коді flag1 відповідає за те, чи є стіна перед датчиком (0 або 1), LF – зберігає саме значення з сенсору.

Команда sim.readProximitySensor дозволяє отримати інформацію з сенсору.

Аналогічним чином потрібно створити команди і для правого та центрального датчика:

```
flag2,LR = sim.readProximitySensor(sensorLR)
flag3,F= sim.readProximitySensor(sensorF).
```

Робот має два сенсори збоку (рис. 4.18). На основі даних з них можна побудувати алгоритм руху вздовж стіни.

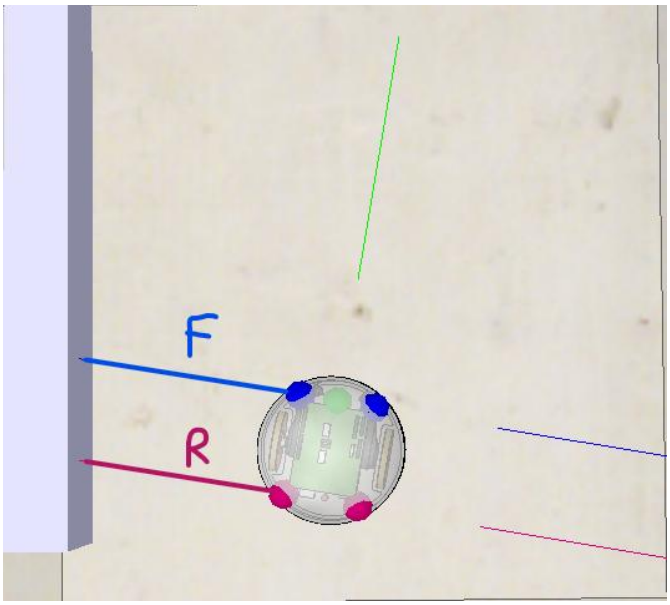

Рисунок 4.18 – Датчики робота, що розташовані ліворуч

Для переміщення прямо, потрібно знайти середнє значення avg показників з датчиків, а також різницю в показаннях dif для корегування маршруту рисканням:

$$avg = \frac{F+R}{2}, \tag{4.1}$$

$$dif = F - R \tag{4.2}$$

У цих формулах F – показання з переднього датчика, R – показання з заднього датчика.

На основі цих формул можна створити наступну умову:

```
if (flag1==0 and flag2 == 1) then
    avg = LR
    diff = 0
    elseif (flag1==1 and flag2 == 0) then
    avg = LF
    diff = 0
    elseif (flag1 ==1 and flag2== 1) then
```

```
    avg = 0.5*(LF+LR)
    diff = LF - LR
    else
    avg = avg_default
    diff = 0
end
```

Якщо тільки лівий задній датчик щось виявляє (flag1 дорівнює 0, що означає, що LF не виявив, а flag2 дорівнює 1, що означає, що LR виявив), то avg встановлюється на відстань LR, а diff дорівнює 0.

Якщо тільки лівий передній датчик щось виявляє, то avg дорівнює відстані LF, а diff дорівнює 0.

Якщо обидва ліві передні та лівий задній датчики щось виявили, avg обчислюється як середнє арифметичне їхніх відстаней, а diff – як різниця між цими відстанями формули (4.1) та (4.2). Значення diff може бути використано для логіки стеження за стіною або вирівнювання (наприклад, якщо diff додатне, робот нахилений ліворуч, якщо від'ємне – праворуч відносно стіни).

Якщо жодна з наведених вище умов не виконується (наприклад, обидва прапорці дорівнюють 0, що означає відсутність виявлення, або неочікувану комбінацію), avg за замовчуванням дорівнює 0.15 (з avg_default), а diff дорівнює 0.

4.2.3 Реалізація алгоритму руху стіною за допомогою сенсорів наближення

Для алгоритму переміщення відстань повинна бути в певному проміжку між мінімальною (min_d) та максимальною (max_d), щоб робот міг повертати без помилок, рискання (yaw) має бути між мінімальним (min) і максимальним (max) відсіканням рискання (повороти ліворуч-праворуч), від'їжджати, якщо передній датчик виявить відсічення max_fwd.

Спочатку потрібно створити змінні для цих параметрів:

```
min_d = 0.15
max_d = 0.25
yaw_cutoff= 0.005
fwd_cutoff= 0.25
avg_default= 0.15
fwd_default = 100
v = 0.5
dv = 0.5
v_sharp = 1
v_straight= 2
avg = avg_default
diff = 0
fwd = fwd_default.
```

Тут min_d і max_d – це бажаний діапазон відстаней до стіни, в якому робот намагатиметься рухатися. Якщо відстань менша за min_d, він занадто близько, якщо більша за max_d, він занадто далеко. yaw_cutoff – поріг для корекції курсового кута (повороту). Якщо різниця між показаннями лівого переднього та лівого заднього датчиків перевищує це значення, робот вважається повернутим відносно стіни і потребує корекції. fwd_cutoff – поріг для переднього датчика. Якщо відстань до об'єкта спереду менша за це значення, робот вважає, що він наближається до стіни або перешкоди. v_sharp – швидкість для різкого повороту (наприклад, при наближенні до стіни спереду). v_straight – швидкість для прямолінійного руху.

В тілі функції sysCall_actuation() потрібно додати такий код. Спочатку робот повинен рухатись вперед з максимальною швидкістю:

```
sim.setJointTargetVelocity(lmotor,v_straight)
```

sim.setJointTargetVelocity(rmotor,v_straight).

Далі потрібно додати умову, що має наступний вигляд:

```
if (fwd < fwd_cutoff) then
        print('going toward the wall, turn right')
        sim.setJointTargetVelocity(lmotor,v_sharp)
        sim.setJointTargetVelocity(rmotor,0)
    elseif (fwd > fwd_cutoff) then
        if (avg > max_d) then
            print('going away from the wall, turn left')
            sim.setJointTargetVelocity(lmotor,v-dv)
            sim.setJointTargetVelocity(rmotor,v)
        elseif (avg < min_d) then
            print('going toward the wall, turn right')
            sim.setJointTargetVelocity(lmotor,v)
            sim.setJointTargetVelocity(rmotor,v-dv)
        elseif(avg > min_d and avg<max_d) then
            if (diff> yaw_cutoff) then --LF>LR
            print('yaw correction: , turn left')
            sim.setJointTargetVelocity(lmotor,v-dv)
            sim.setJointTargetVelocity(rmotor,v)
            elseif (diff < -yaw_cutoff) then --LF<LR
            sim.setJointTargetVelocity(lmotor,v)
            sim.setJointTargetVelocity(rmotor,v-dv)
            end
        end
end
```

Якщо передній датчик показує, що робот занадто близько до перешкоди спереду (тобто він прямує до стіни), то він різко повертає праворуч. Лівий мотор працює на v_sharp, а правий зупиняється, щоб здійснити швидкий поворот.

У випадку коли попереду вільно, то робот підтримує відстань до стіни збоку.

Якщо робот занадто далеко від стіни (середня відстань більша за max_d), він повертає ліворуч, щоб наблизитися. Лівий мотор повільніший (v-dv), а правий швидший (v).

Якщо робот занадто близько до стіни (середня відстань менша за min_d), він повертає праворуч, щоб віддалитися. Лівий мотор швидший (v), а правий повільніший (v-dv).

Якщо робот знаходиться в бажаному діапазоні відстані до стіни, то починається корекція курсового кута (yaw correction):

Якщо diff знаходиться в межах yaw_cutoff (тобто diff між -yaw_cutoff і yaw_cutoff), це означає, що робот рухається паралельно стіні. У цьому випадку, судячи з коду, швидкість моторів повертається до початкового v_straight, що забезпечує прямолінійний рух, але це неявно, оскільки немає else для diff умови.

Також у функцію sysCall_sensing() було додано умову для датчику, що знаходиться попереду робота:

```
if (flag3== 1) then
    fwd = F
    else
    fwd = fwd_default
    end
```

Це означає наступне: якщо передній датчик бачить об'єкт, fwd встановлюється як відстань до цього об'єкта, у випадку коли передній датчик

нічого не бачить, fwd встановлюється на значення за замовчуванням (fwd_default), що, по суті, означає рух вперед.

4.3 Висновки до розділу 4

В результаті виконання 4 розділу було проведено моделювання роботи сенсорної системи роботів для руху лінією (датчики лінії), а також для руху стіною (датчики наближення), а саме:

– побудовано модель з роботами;

– проведено побудову програмного забезпечення для руху роботів;

– проведено моделювання та програмування алгоритмів руху лінією та стіною.

Використовуючи запропоновані алгоритми, роботи проходять траєкторії без помилок за умови правильного налаштування параметрів. Експериментальні дослідження з цими алгоритмами наведено у розділі 5.

5 ДОСЛІДЖЕННЯ ПЕРЕМІЩЕННЯ МОДЕЛЬОВАНОГО РОБОТА У СЕРЕДОВИЩІ COPPELIASIM

5.1 Дослідження руху складною траєктроїєю з використанням датчиків лінії

Для проведення досліджень, потрібно створити новий шлях-траєкторію. Для цього потрібно використати Path – Closed Path, а далі змінити розташування ключових точок цього шляху (рис. 5.1).

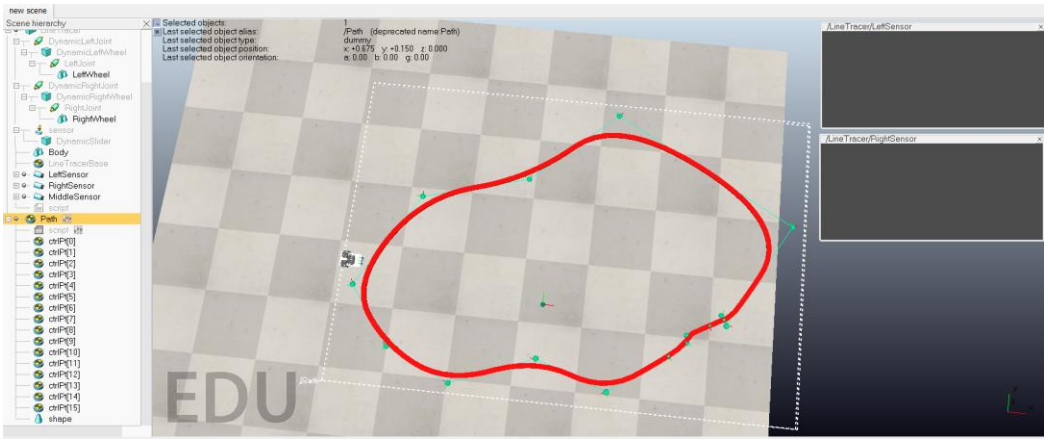

Рисунок 5.1 – Шлях переміщення робота

Під час запуску програми робот проходить всю трасу, рухаючись лінією (рис. 5.2).

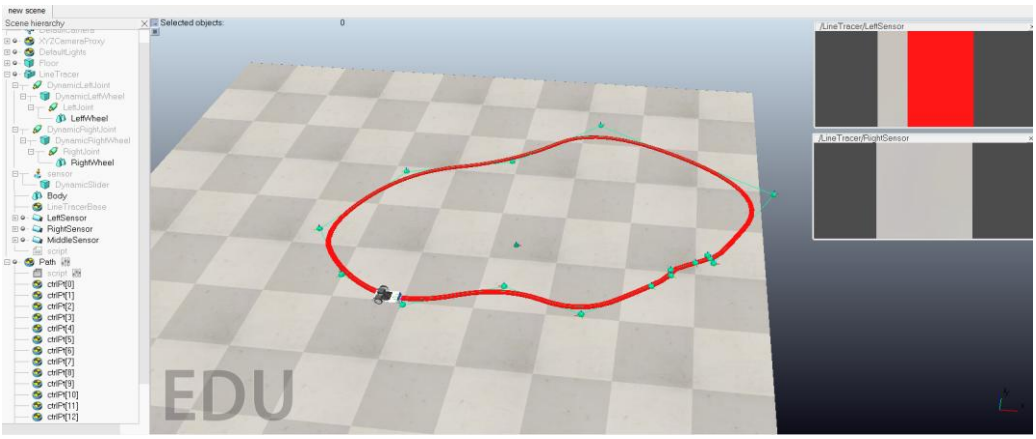

Рисунок 5.2 – Проходження шляху робота

Проведемо дослідження з різною швидкістю та зміною швидкості (пропорційна та диференційна складові регулятору). Результати експериментальних досліджень представлені у табл. 5.1.

Таблиця 5.1 – Результати проходження роботом траси

Номер етапу експерименту	Швидкість	Зміна швидкості	Результат проходження
1	2	2	так
2	3	5	так
3	3	20	ні
4	5	2	так
5	10	5	так
6	12	15	ні
7	13	10	так
8	14	15	ні
9	15	10	так
10	20	15	ні
11	20	15	так
12	20	2	ні
13	20	10	так
14	25	10	так
15	25	20	ні
16	30	10	так
17	30	15	ні
18	35	10	ні
19	35	15	ні
20	40	10	ні

Провівши експеримент, можна зробити декілька висновків:

– за низьких показників пропорційної та диференційної складових, робот проходить шлях без помилок;

– у випадку коли швидкість (пропорційна складова) стає більшою, а приріст швидкості (диференційна складова) є малою, то робот з’їжджає з траси;

– при зміні швидкості більше ніж на 30 м./с., за будь-яких значень диференційної складової, робот вже перестає бачити трасу, датчики не встигають аналізувати інформацію;

– коли швидкість менша за приріст, то робот також починає губити трасу, що негативно позначається на його роботі.

Розглянемо зміну швидкостей робота при різних показниках пропорційної та диференційної складової. Результати представлено на рис. 5.3 – рис. 5.11.

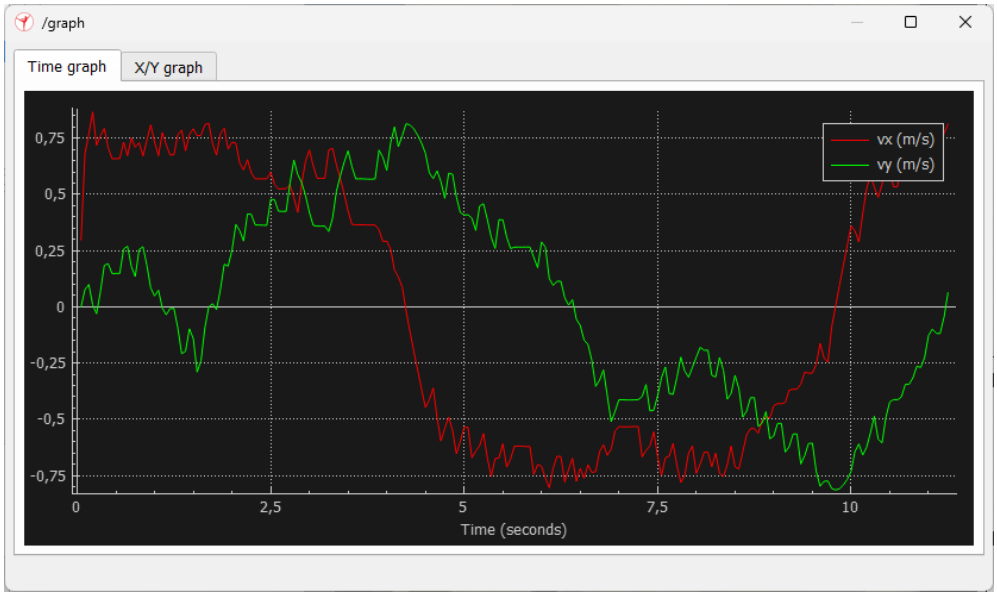

Рисунок 5.3 – Зміна швидкостей при пропорційній складовій – 25, диференційній – 10

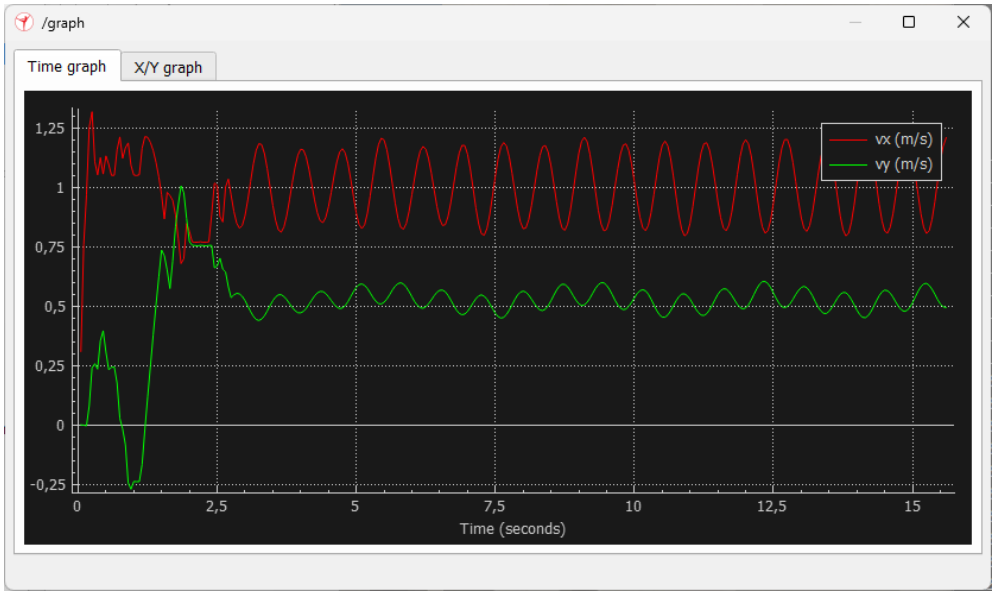

Рисунок 5.4 – Зміна швидкостей при пропорційній складовій – 40, диференційній – 10

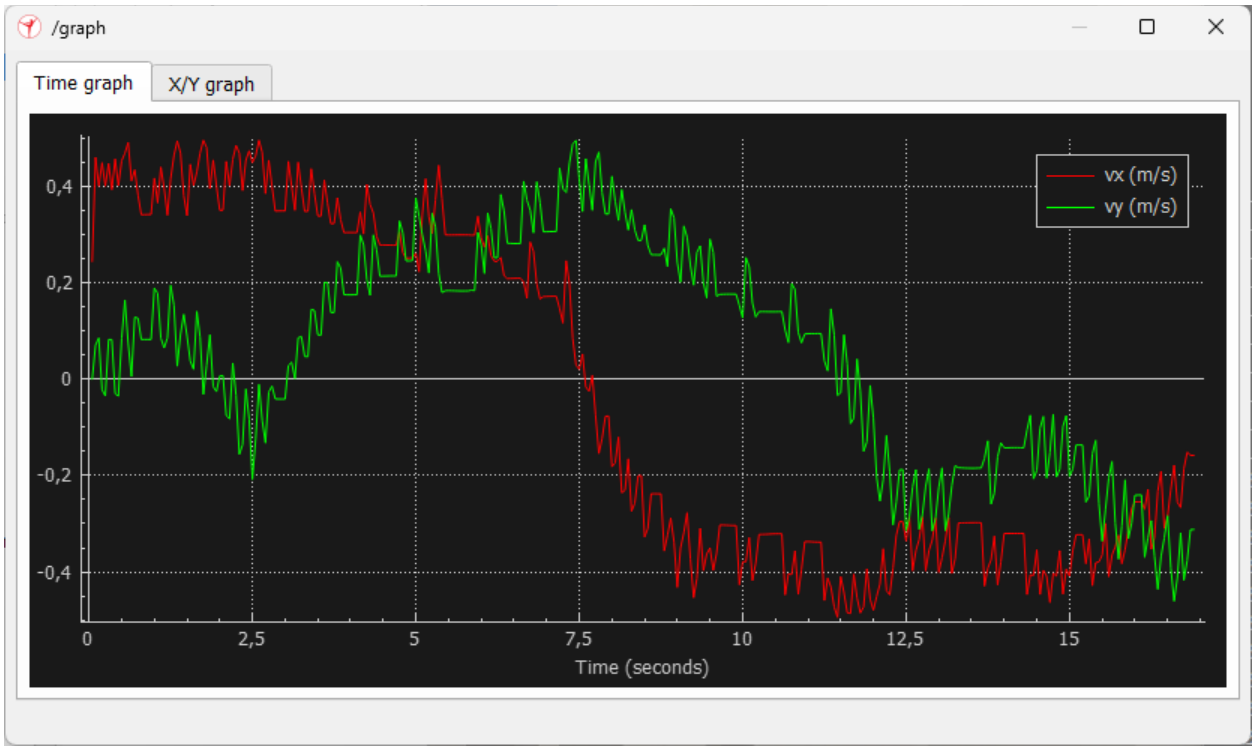

Рисунок 5.5 – Зміна швидкостей при пропорційній складовій – 13, диференційній – 10

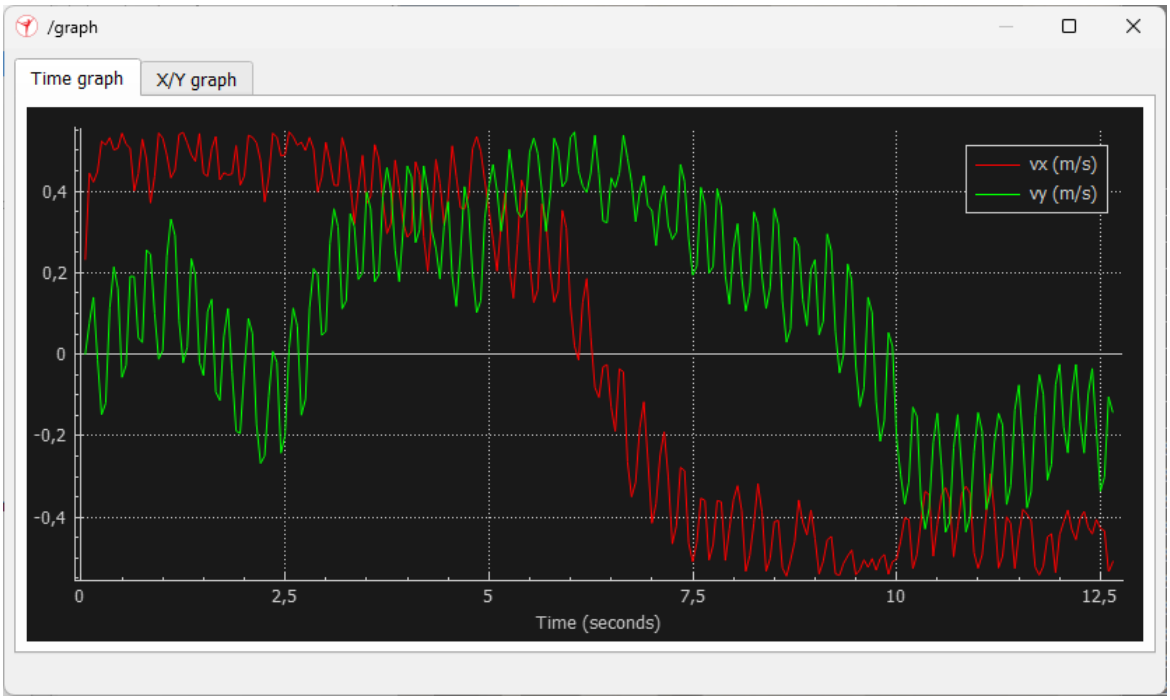

Рисунок 5.6 – Зміна швидкостей при пропорційній складовій – 12, диференційній – 15

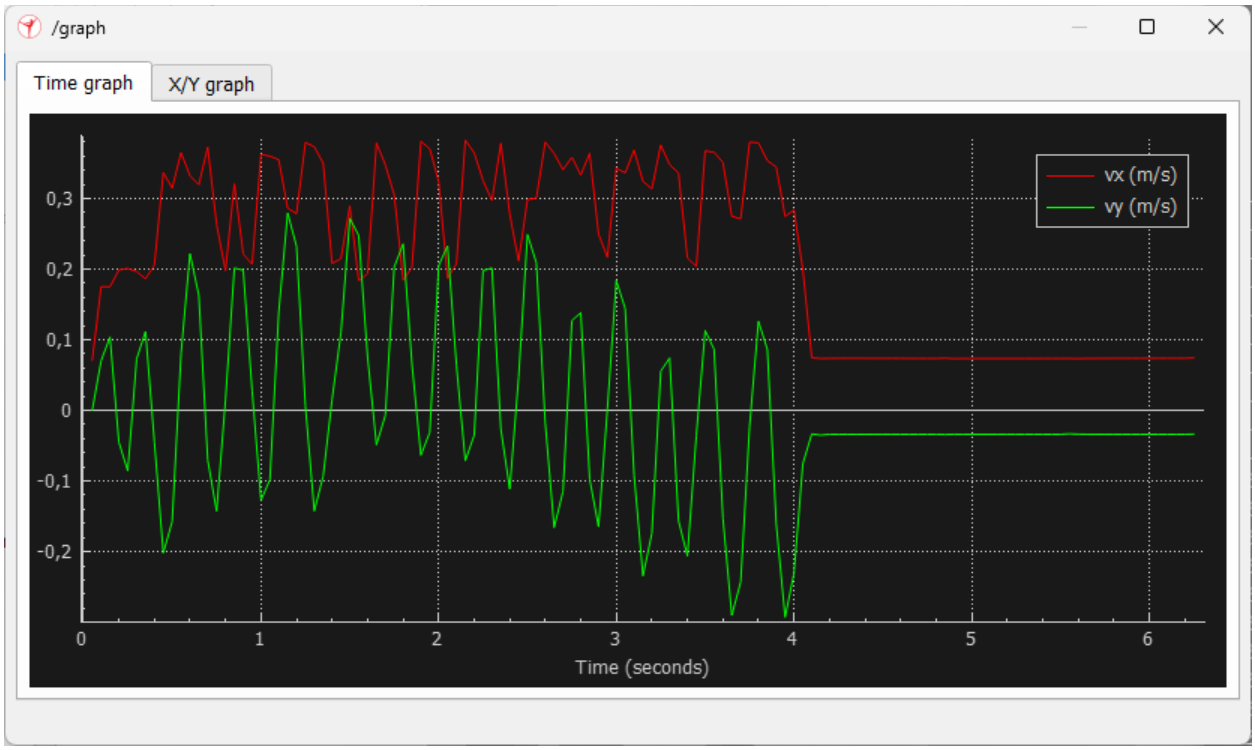

Рисунок 5.7 – Зміна швидкостей при пропорційній складовій – 3, диференційній – 20

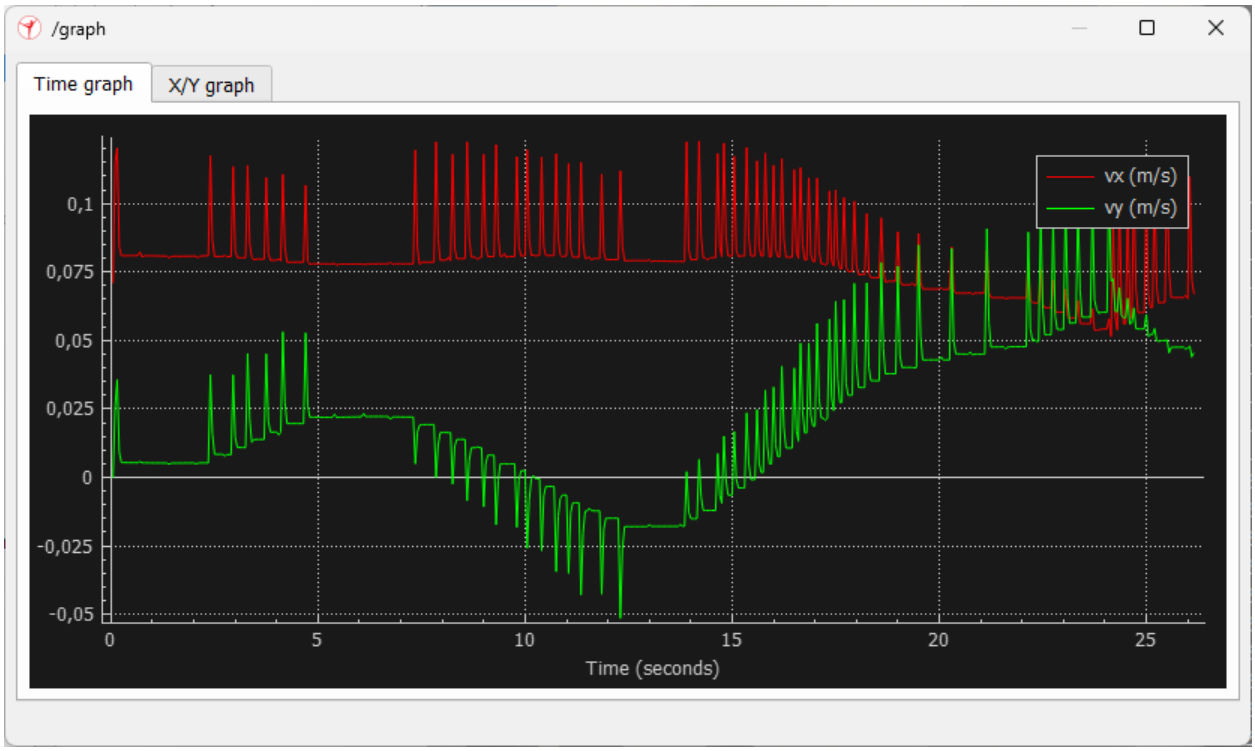

Рисунок 5.8 – Зміна швидкостей при пропорційній складовій – 3, диференційній – 3

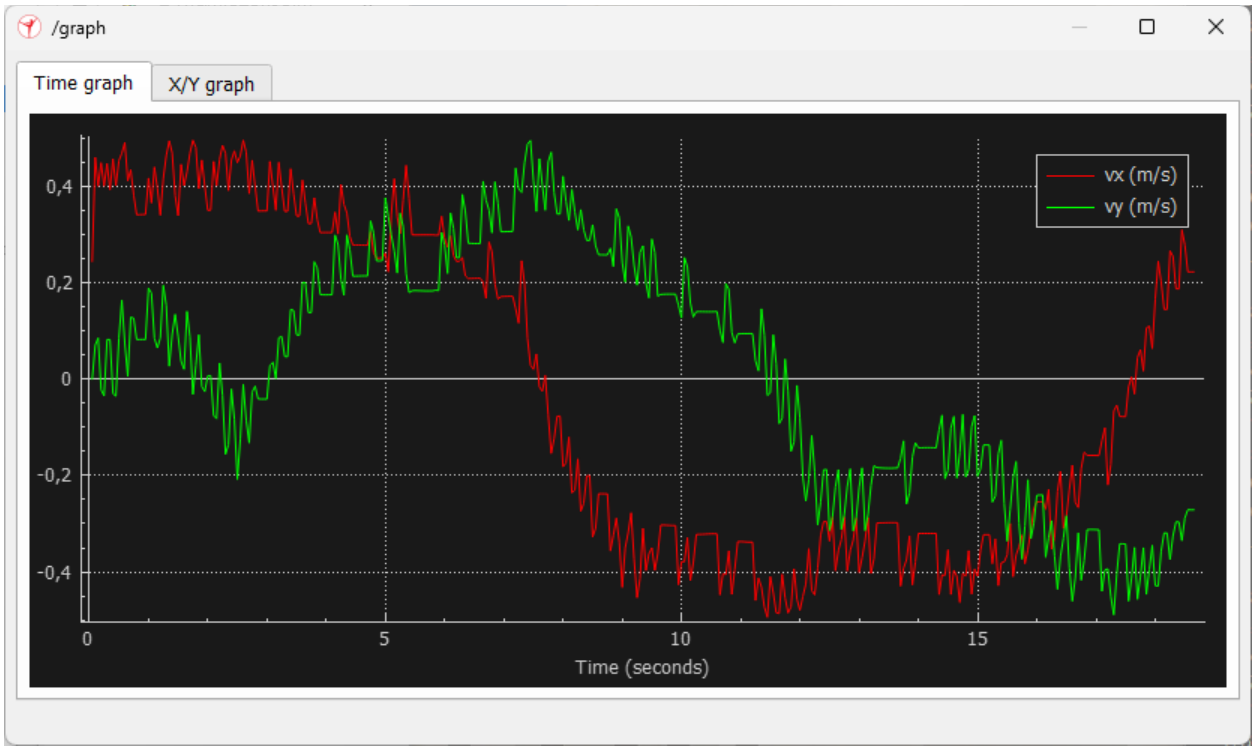

Рисунок 5.9 – Зміна швидкостей при пропорційній складовій – 15, диференційній – 15

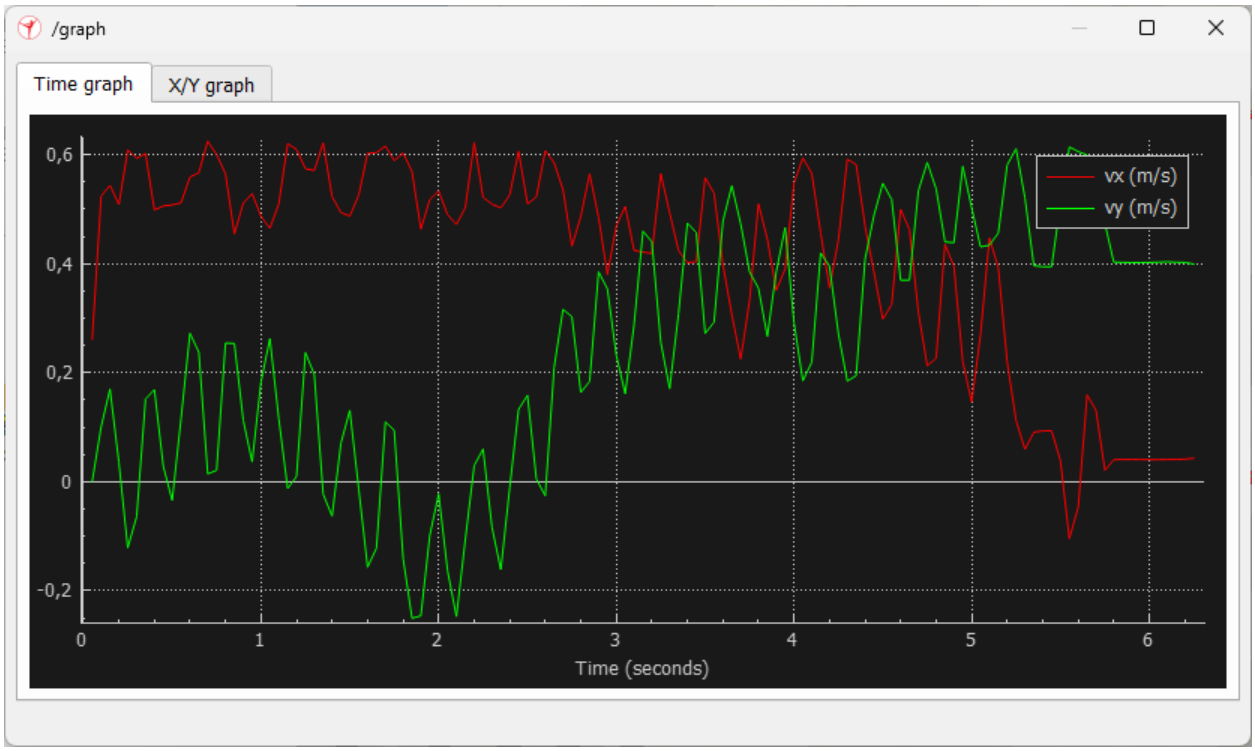

Рисунок 5.10 – Зміна швидкостей при пропорційній складовій – 15, диференційній – 15

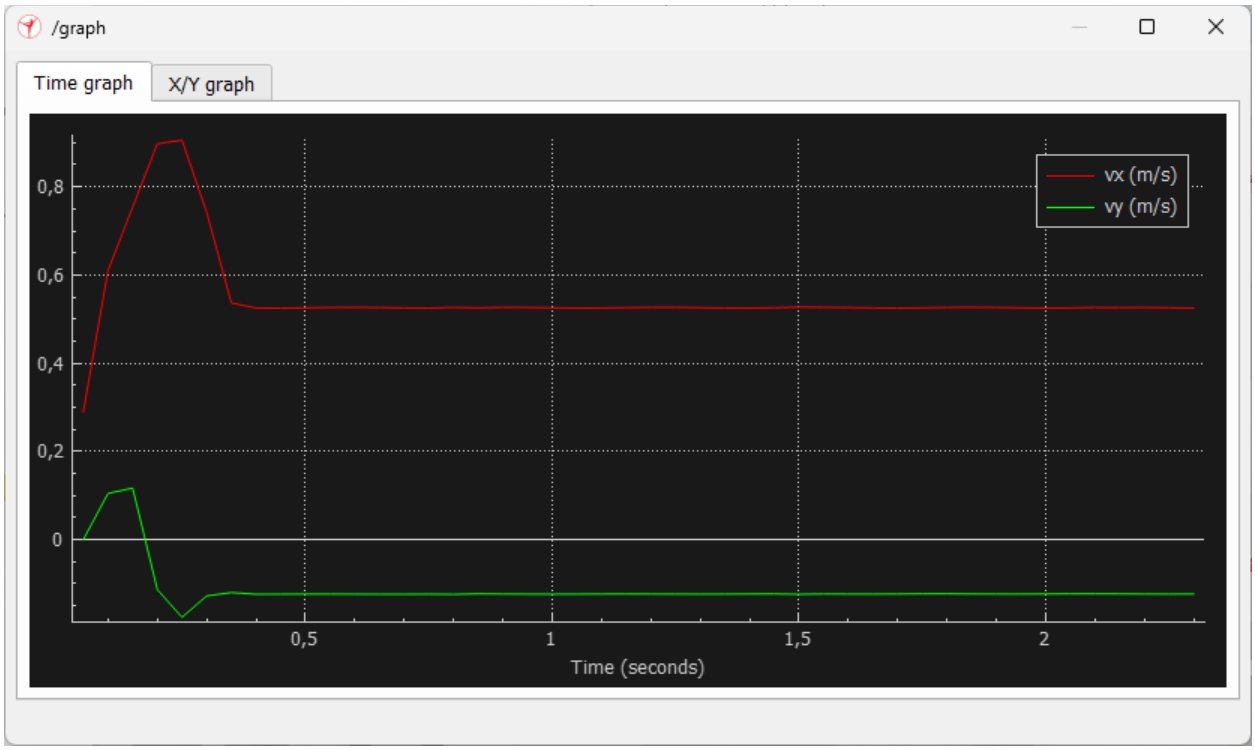

Рисунок 5.11 – Зміна швидкостей при пропорційній складовій – 20, диференційній – 30

Виходячи з графіків можна зробити висновки:

– швидкість обох коліс постійно змінюється, що вказує на нерівномірний рух робота;

– швидкість лівого колеса демонструє часті та різкі сплески (піки) і падіння для підтримки руху заданим маршрутом;

– ліве колесо рухається вперед більшу частину часу і з більшою швидкістю;

– при некоректних заданих значеннях пропорційної та інтегральної складової, швидкості стають прямою лінією (рис. 5.7, 5.9 - 5.11).

5.2 Дослідження руху стіною з використанням датчиків наближення

Для дослідження потрібно змінити траєкторію, а саме зробити квадратну, щоб імітувати рух робота у реальних умовах (наприклад, робот-пилосос виконує поставлені перед ним дії).

Дослідження будуть проводитись на траєкторії, представленій на рис. 5.12.

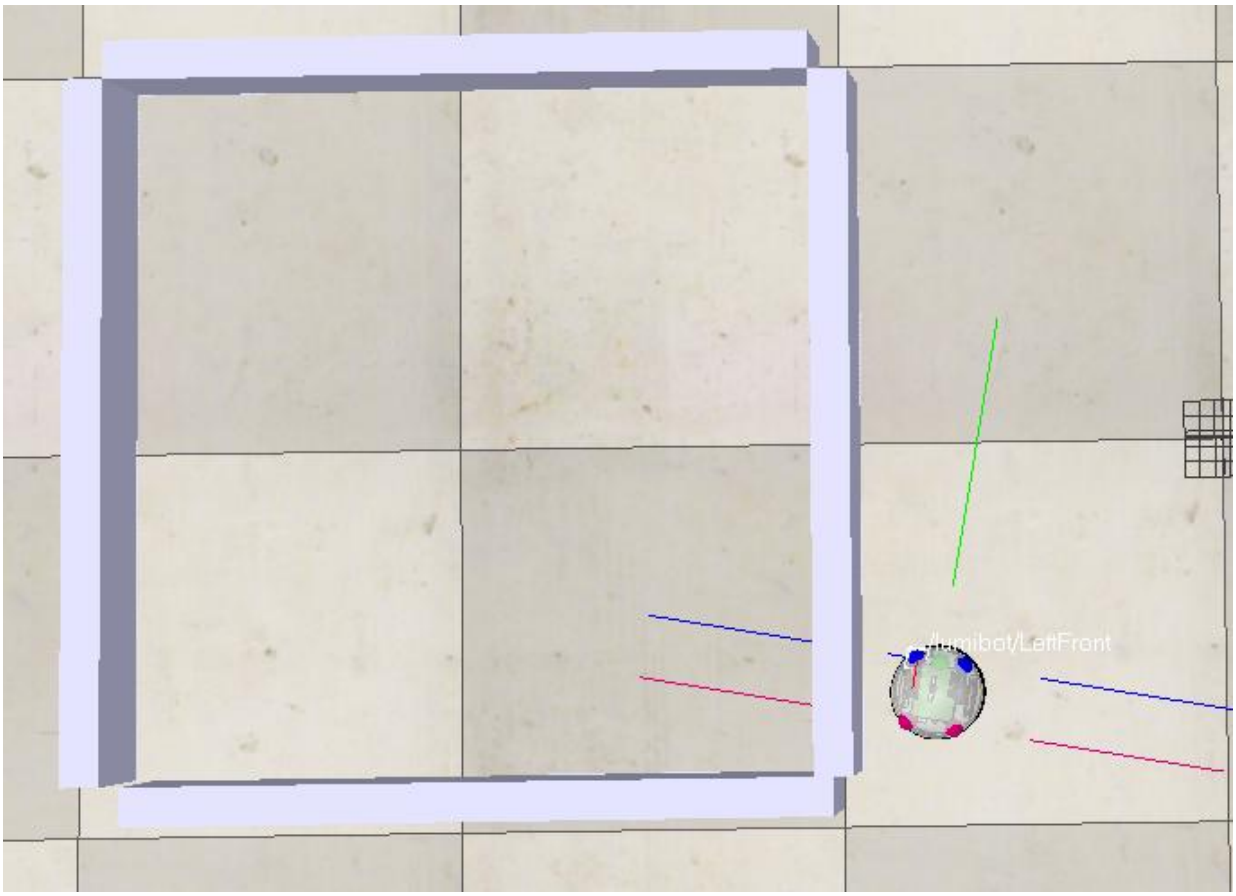

Рисунок 5.12 – Траєкторія для дослідження

Проведемо дослідження з різною швидкістю та зміною швидкості (пропорційна та диференційна складові регулятору), а також кутового руху і руху вперед за замовчуванням. Результати дослідження представлені у табл. 5.2.

Таблиця 5.2 – Результати досліджень рухом стіною

Номер етапу експерименту	v	dv	v_sharp	v_straight	Результат проходження
1	0,5	0,5	1	2	так
2	1	1	2	4	так
3	2	2	4	2	так
4	5	5	5	5	ні
5	10	5	10	5	так
6	10	10	10	20	ні
7	10	20	20	20	ні
8	20	10	10	20	ні
9	20	20	10	15	ні
10	25	20	20	10	ні
11	25	15	20	15	ні
12	30	10	20	10	ні
13	30	20	25	20	ні
14	30	10	25	10	так
15	30	15	30	10	ні
16	40	10	40	10	ні
17	40	20	30	20	ні
18	50	10	20	10	ні
19	60	20	30	20	ні
20	70	10	20	10	ні

Провівши експерименти можна зробити наступні висновки:

– низькі швидкості (v=0.5, 1, 2, 5) дають успішні результати, але співвідношення v та dv є критичним для стабільності;

– на відносно низькій швидкості, якщо dv дорівнює v (тобто одне колесо зупиняється, а інше рухається з v), це може призводити до занадто різких поворотів або нестабільності, що призводить до невдачі;

– робот успішно переміщується вздовж стіни на низьких швидкостях (до 2-5 одиниць v);

– збільшення швидкості понад 5-10 одиниць v різко знижує успішність проходження, що вказує на потребу у більш точному налаштуванні параметрів для динамічніших режимів;

– занадто великі зміни параметрів (великі dv або v_sharp відносно v) можуть призводити до нестабільності руху.

Розглянемо зміну швидкостей робота при різних показниках параметрів таблиці 5.2. Результати представлено на рис. 5.13 – рис. 5.23.

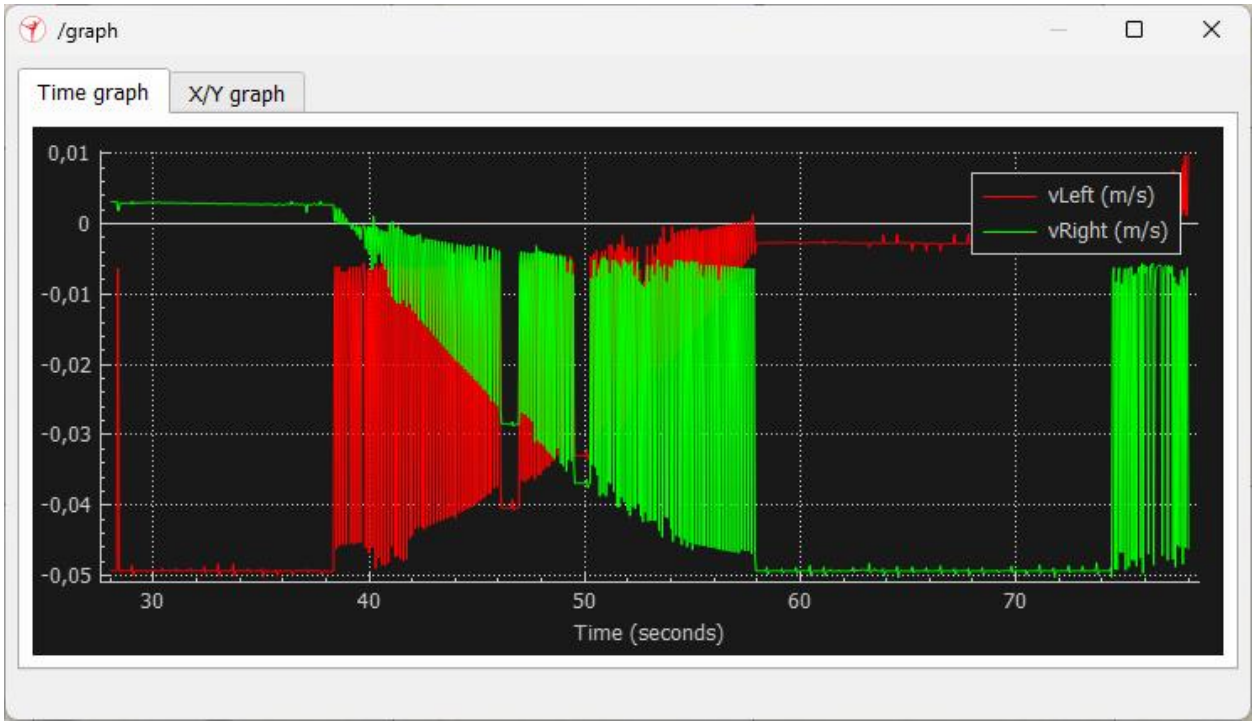

Рисунок 5.13 – Зміна швидкостей при v= 0,5, dv= 0,5, v_sharp = 1, v_straight=2

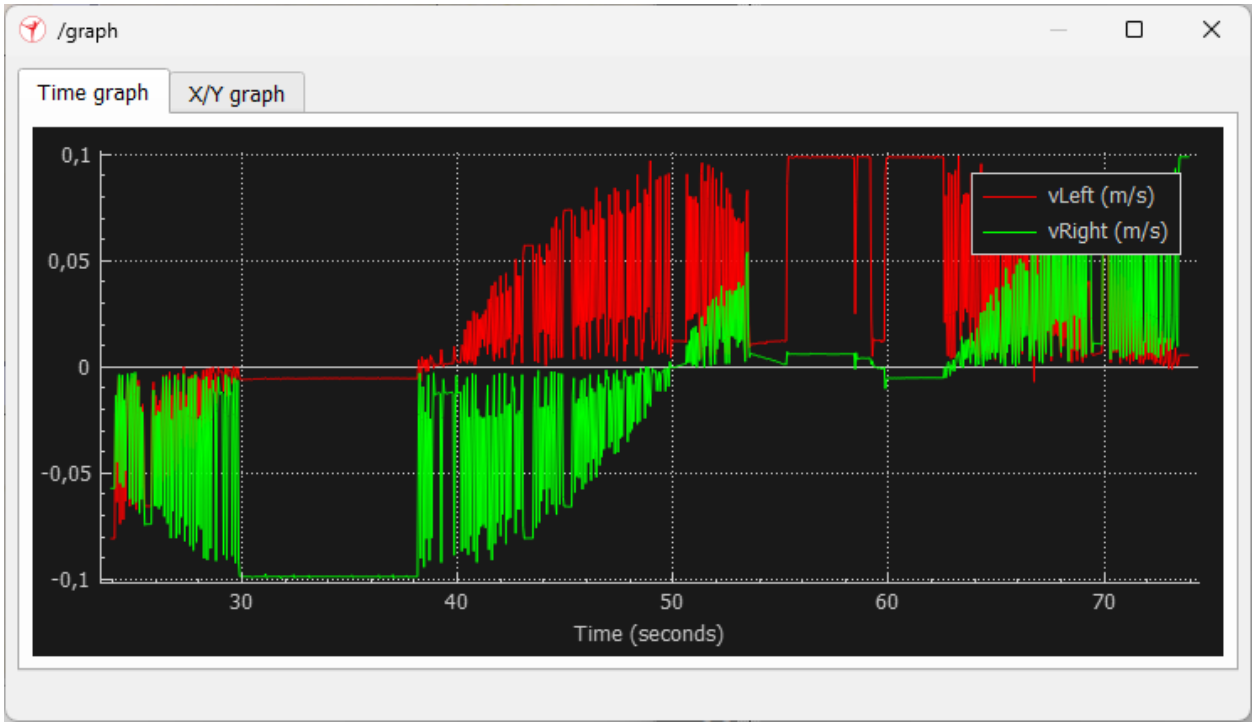

Рисунок 5.14 – Зміна швидкостей при v= 1, dv= 1, v_sharp = 2, v_straight=4

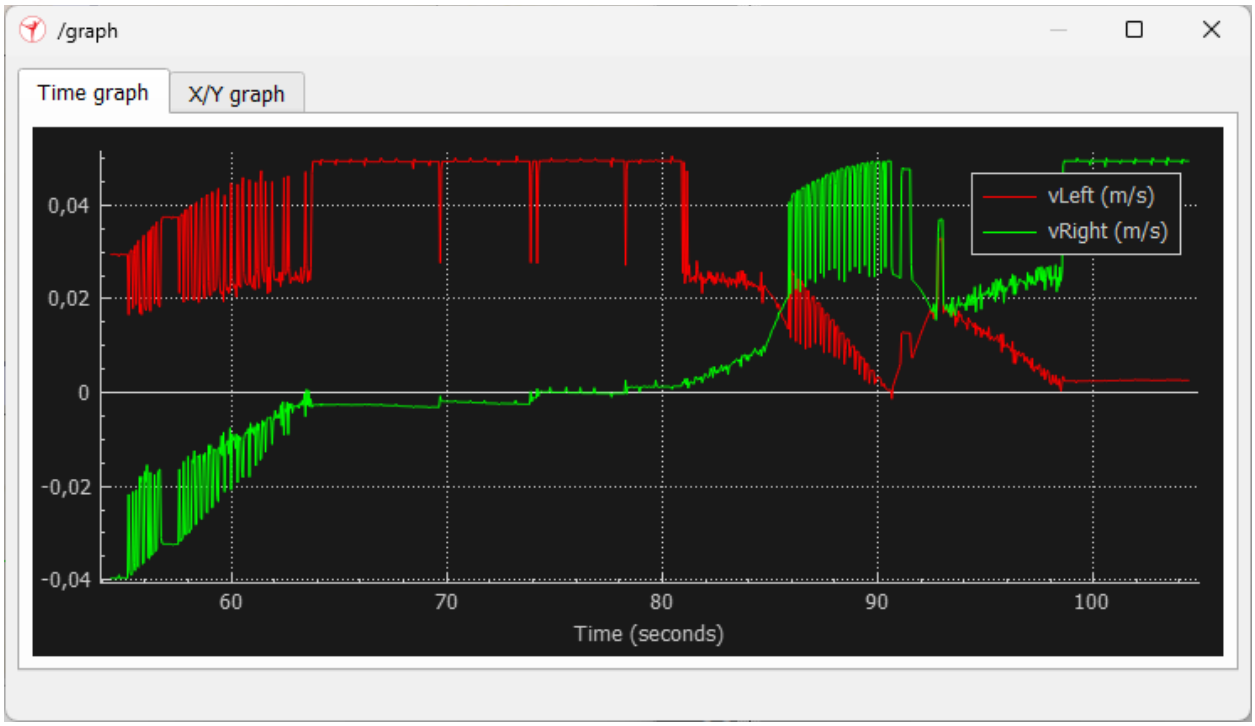

Рисунок 5.15 – Зміна швидкостей при v= 2, dv= 2, v_sharp = 4, v_straight=2

Рисунок 5.16 – Зміна швидкостей при v= 5, dv= 5, v_sharp = 5, v_straight=5

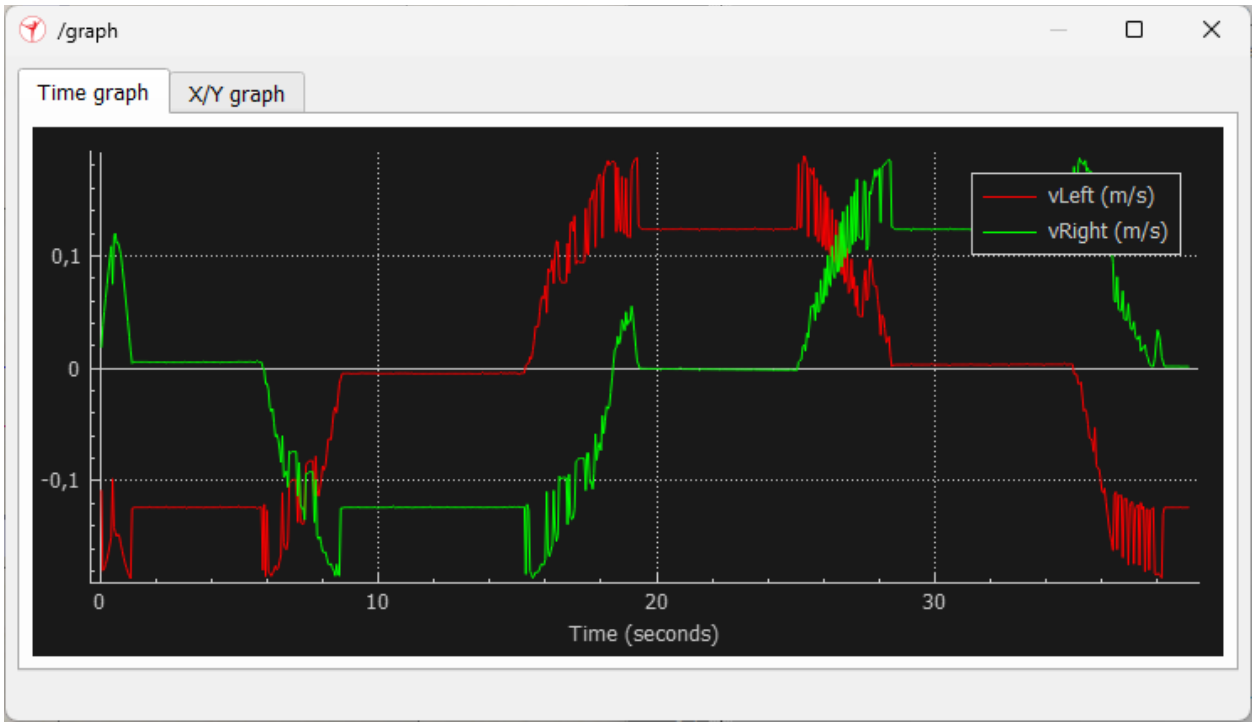

Рисунок 5.17 – Зміна швидкостей при v= 10, dv= 5, v_sharp = 10, v_straight=5

Рисунок 5.18 – Зміна швидкостей при v= 10, dv= 10, v_sharp = 10, v_straight=20

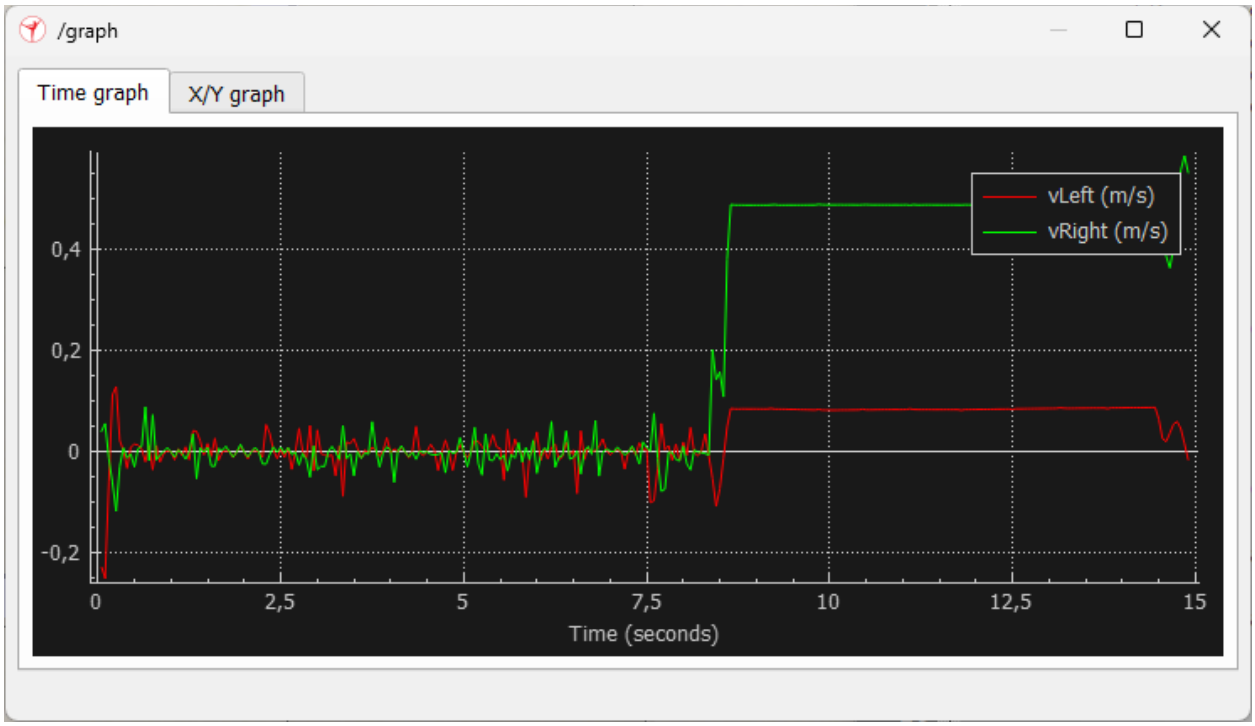

Рисунок 5.19 – Зміна швидкостей при v= 10, dv= 20, v_sharp = 20, v_straight=20

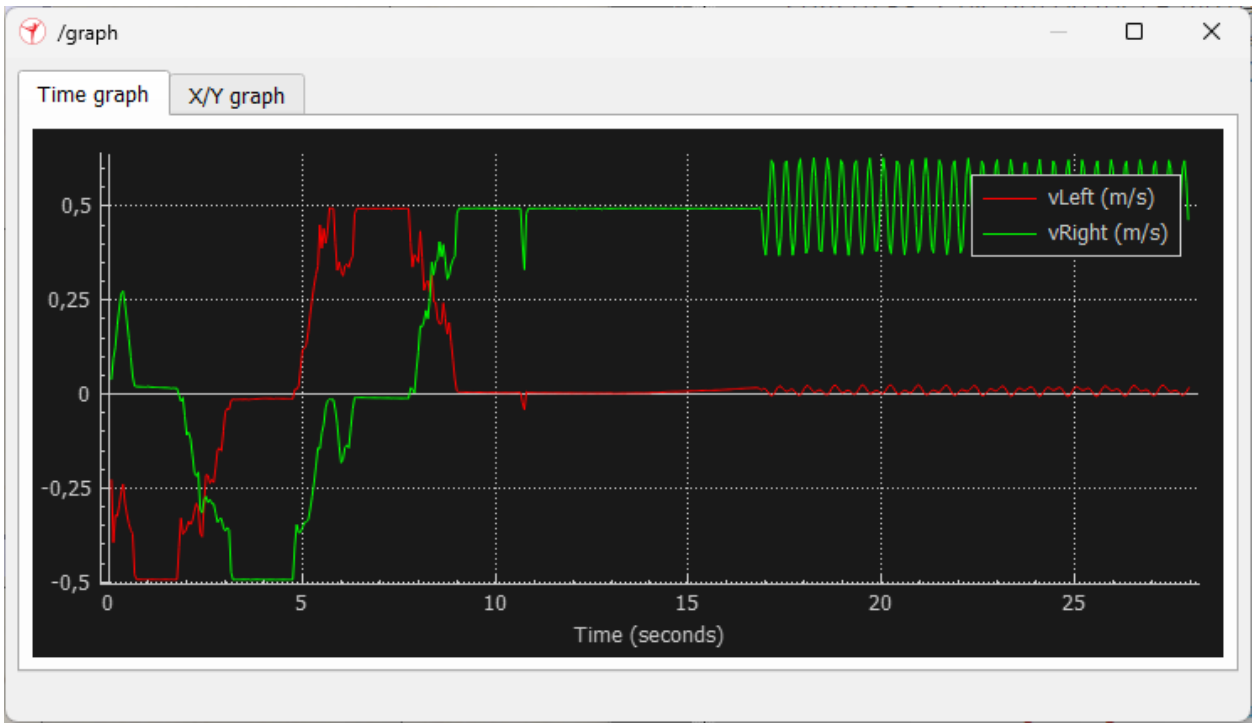

Рисунок 5.20 – Зміна швидкостей при v= 20, dv= 10, v_sharp = 10, v_straight=20

Рисунок 5.21 – Зміна швидкостей при v= 25, dv= 20, v_sharp = 10, v_straight=10

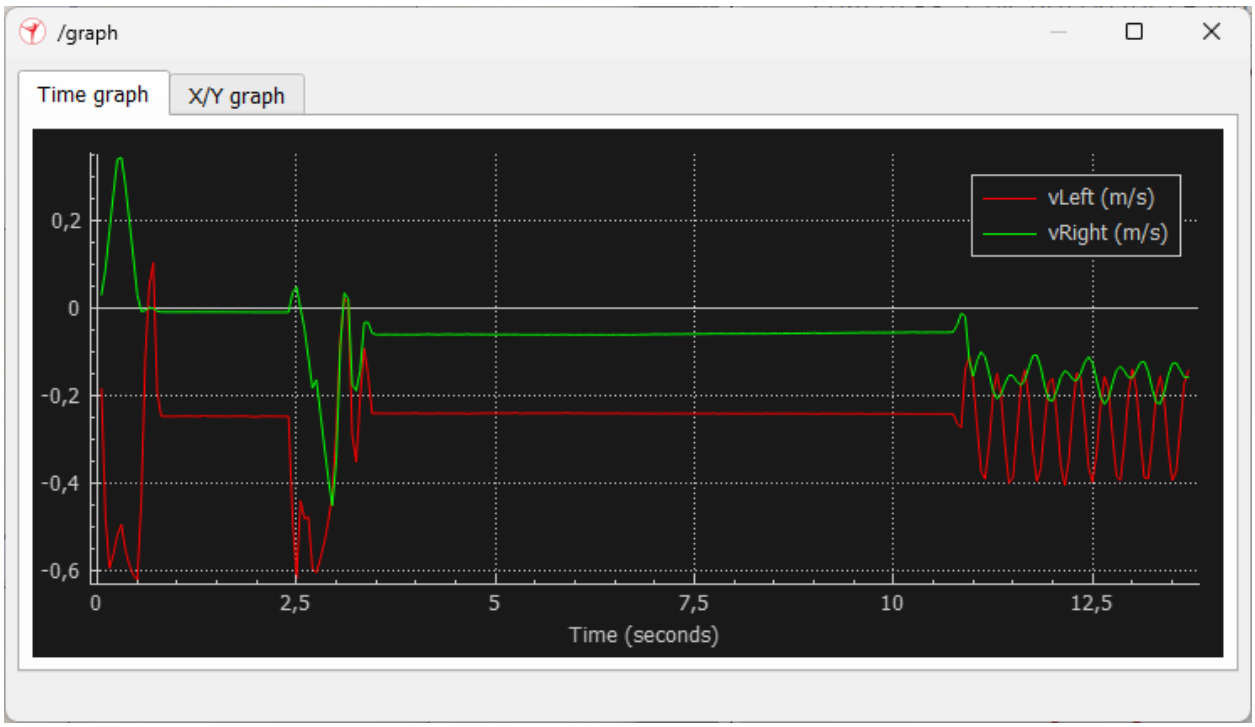

Рисунок 5.22 – Зміна швидкостей при v= 30, dv= 10, v_sharp = 20, v_straight=10

Рисунок 5.23 – Зміна швидкостей при v= 30, dv= 10, v_sharp = 25, v_straight=10

З графіків можна зробити наступні висновки:

– за низьких параметрів налаштування системи робот змінює швидкість дуже часто;

– якщо параметри системи є завеликими, то робот губить стіну і з’являється помилка переміщення, про що свідчить пряма горизонтальна лінія (рис. 5.14 – рис. 5.15, рис. 5.17 – рис. 5.23);

– при правильному налаштуванні параметрів системи робот навіть на великих швидкостях може рухатись стіною;

– при дуже великих налаштуваннях системи (коли v більше за 40) помилка руху робота з’являється вже на перших етапах руху, бо датчики не встигають аналізувати середовище і зміна швидкостей відбувається дуже сильно.

5.3 Охорона праці

До роботи з роботами допускаються виключно працівники, які пройшли спеціалізоване навчання, інструктаж з охорони праці та мають відповідну кваліфікацію. Перед початком роботи обов'язково ознайомтеся з інструкцією з експлуатації конкретної моделі робота та дотримуйтесь усіх рекомендацій виробника [34].

Усі роботизовані системи повинні бути належним чином обладнані засобами безпеки, такими як захисні огорожі, аварійні кнопки зупинки, а також світлові та звукові сигналізації. Зона роботи робота має бути чітко визначена та позначена відповідними попереджувальними знаками; доступ сторонніх осіб до неї суворо заборонений. Крім того, працівники повинні використовувати відповідні засоби індивідуального захисту (ЗІЗ), такі як захисні окуляри, рукавички та спеціальне взуття.

Перш ніж увійти в робочу зону робота, завжди переконайтеся, що він знаходиться в стані спокою або його робота заблокована. Категорично забороняється заходити в робочу зону, коли робот працює в автоматичному режимі, без попереднього відключення живлення або переведення його в режим безпечної зупинки.

При виконанні будь-яких робіт у межах робочої зони робота, включаючи налаштування, обслуговування або ремонт, обов'язковим є використання системи блокування енергії для запобігання несподіваному запуску.

Завжди пильно контролюйте рухи робота та будьте готові до можливих непередбачених ситуацій. Ніколи не розміщуйте частини тіла (руки, ноги, голову) в потенційних точках защемлення або удару. Суворо забороняється втручатися в роботу робота, намагатися зупинити його або змінити траєкторію його рухів вручну під час функціонування.

У разі виникнення будь-яких несправностей або відхилень у роботі робота, негайно зупиніть його за допомогою аварійної кнопки та повідомте свого безпосереднього керівника. Для програмування або налагодження

робота використовуйте спеціальні пульти управління, дотримуючись безпечної відстані. Після завершення роботи обов'язково переведіть робота в безпечний режим, вимкніть живлення та залиште робоче місце в належному стані.

Будь-які роботи з технічного обслуговування, ремонту або очищення робота можуть виконуватися виключно кваліфікованим персоналом. Ці дії можливі лише після повного відключення живлення робота та застосування процедур. Перед початком таких робіт переконайтеся у відсутності залишкової енергії в системах (наприклад, у пневматичних або гідравлічних).

Завжди використовуйте тільки справний інструмент та обладнання. Після завершення робіт з обслуговування переконайтеся, що всі захисні елементи встановлені на свої місця, і робот готовий до безпечної експлуатації.

У разі виникнення будь-якої аварійної ситуації, такої як несподіваний рух робота, задимлення або пожежа, негайно натисніть кнопку аварійної зупинки та дійте відповідно до плану ліквідації аварійних ситуацій.

Обов'язково повідомте про аварію безпосереднього керівника та, за необхідності, викличте відповідні служби (пожежну, медичну). Не намагайтеся самостійно усувати наслідки аварії, якщо це може загрожувати вашому життю або здоров'ю.

5.4 Висновки до розділу 5

В результаті написання 5 розділу було проведено експериментальні дослідження з роботами та сенсорною системо, запропонованою у розділі 4, а саме:

– дослідження руху складною траєктроїєю з використанням датчиків лінії;

– дослідження руху стіною з використанням датчиків наближення.

Також можна зробити декілька висновків під час руху лінією:

– за низьких показників пропорційної та диференційної складових, робот проходить шлях без помилок;

– у випадку коли швидкість (пропорційна складова) стає більшою, а приріст швидкості (диференційна складова) є малою, то робот з'їжджає з траси;

– при зміні швидкості більше ніж на 30 м./с., за будь-яких значень диференційної складової, робот вже перестає бачити трасу, датчики не встигають аналізувати інформацію;

– коли швидкість менша за приріст, то робот також починає губити трасу, що негативно позначається на його роботі.

Висновки під час руху стіною:

– низькі швидкості (v=0.5, 1, 2, 5) дають успішні результати, але співвідношення v та dv є критичним для стабільності;

– на відносно низькій швидкості, якщо dv дорівнює v (тобто одне колесо зупиняється, а інше рухається з v), це може призводити до занадто різких поворотів або нестабільності, що призводить до невдачі;

– робот успішно переміщується вздовж стіни на низьких швидкостях (до 2-5 одиниць v);

– збільшення швидкості понад 5-10 одиниць v різко знижує успішність проходження, що вказує на потребу у більш точному налаштуванні параметрів для динамічніших режимів;

– занадто великі зміни параметрів (великі dv або v_sharp відносно v) можуть призводити до нестабільності руху.

ВИСНОВКИ

Під час написання кваліфікаційної роботи було вирішено мету роботи та задачі:

– проведено аналіз літератури з моделювання у робототехніці;

– проведено аналіз сучасних систем моделювання;

– проведено дослідження сенсорної системи у симуляторі CoppeliaSim;

– розроблено програмне забезпечення для моделювання сенсорної системи мовою Lua у симуляторі CoppeliaSim та провести експериментальні дослідження з розробленою моделлю.

В процесі написання 1 розділу було проведено аналіз літератури за темою, та проаналізовано загальне значення моделювання у робототехніці, переваги та недоліки моделювання, класифікація систем моделювання робототехнічних систем, огляд ключових систем моделювання та їх характеристики.

Провівши аналіз систем моделювання (ROS, Matlab, Adams, Webots, CoppeliaSim), було прийнято рішення використати CoppeliaSim тому що він є найкращим симулятором для робототехніки, особливо якщо потрібні гнучкість, кросплатформність, універсальність та широкі можливості програмування (Lua, C++, Python) без прив'язки до екосистеми ROS чи високих витрат на купівлю цих систем моделювання

В ході виконання 2 розділу було розглянуто можливості моделювання сенсорів у симуляторі CoppeliaSim та 2 типах датчиків, таких як датчики наближення та датчики зору. Проведено моделювання роботи кожного з датчиків та отримано програмні результати, що демонструють можливість подальшого моделювання сенсорної системи роботів.

Також проведено розрахунки теорії автоматичного керування роботом на основі датчика наближення, отримано передавальні функції та перевірено їх на стійкість.

В результаті 3 розділу з використанням симулятора CoppeliaSim було побудовано модель робота з диференціальним приводом, а також досліджено можливість його програмування, отримання інформації з сенсорної системи, а також побудови графіків залежності його швидкостей лівого та правого коліс та відповідних моторів.

В результаті виконання 4 розділу було проведено моделювання роботи сенсорної системи роботів для руху лінією (датчики лінії), а також для руху стіною (датчики наближення). Використовуючи запропоновані алгоритми, роботи проходять траєкторії без помилок за умови правильного налаштування параметрів.

В результаті написання 5 розділу було проведено експериментальні дослідження з роботами та сенсорною системою.

За результатами можна зробити декілька висновків під час руху лінією:

– за низьких показників пропорційної та диференційної складових, робот проходить шлях без помилок;

– у випадку коли швидкість (пропорційна складова) стає більшою, а приріст швидкості (диференційна складова) є малою, то робот з'їжджає з траси;

– при зміні швидкості більше ніж на 30 м./с., за будь-яких значень диференційної складової, робот вже перестає бачити трасу, датчики не встигають аналізувати інформацію;

– коли швидкість менша за приріст, то робот також починає губити трасу, що негативно позначається на його роботі.

Висновки під час руху стіною:

– низькі швидкості (v=0.5, 1, 2, 5) дають успішні результати, але співвідношення v та dv є критичним для стабільності;

– на відносно низькій швидкості, якщо dv дорівнює v (тобто одне колесо зупиняється, а інше рухається з v), це може призводити до занадто різких поворотів або нестабільності, що призводить до невдачі;

– робот успішно переміщується вздовж стіни на низьких швидкостях (до 2-5 одиниць v);

– збільшення швидкості понад 5-10 одиниць v різко знижує успішність проходження, що вказує на потребу у більш точному налаштуванні параметрів для динамічніших режимів;

– занадто великі зміни параметрів (великі dv або v_sharp відносно v) можуть призводити до нестабільності руху.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методичні вказівки з підготовки та захисту кваліфікаційної роботи здобувачами другого (магістерського) рівня вищої освіти спеціальності 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка, освітньо-професійних програм: «Комп'ютерно-інтегровані технологічні процеси і виробництва», «Комп'ютеризовані та робототехнічні системи» / Упоряд. І. Ш. Невлюдов, Р. В. Артюх, В. В. Безкоровайний, Н. П. Демська, В. В. Євсєєв, О. І. Филипенко, О. М. Цимбал. Харків: ХНУРЕ, 2024. 57 с.

2. ДСТУ 3008:2015 Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. ДП «УкрНДНЦ», 2016. – 31 с.

3. Кривчун Р.В. Комп'ютерне моделювання та його роль у сучасному роботизованому виробництві / Р.В. Кривчун // Автоматизація та Приладобудування («Automation and Development of Electronic Devices» ADED-2025) [Електронний ресурс] : збірник студентських наукових статей / Харківський національний університет радіоелектроніки ; [редкол.: І.Ш. Невлюдов та ін.]. – Харків : ХНУРЕ, 2025. – Вип. 2., с. 81-87.

4. Adenubi, Adeola. Robotics and Automation in the 21st Century: Innovations and Applications / Adeola Adenubi, Ayorinde Oduroye // Online Journal of Robotics & Automation Technology, 2025, vol. 4, № 9, 9 с. https://doi.org/10.33552/OJRAT.
2025.04.000580.

5. Borcard, Daniel. State of the Art and Research Directions for Visual Conceptual Modeling in Robotics / Daniel Borcard, Hans-Georg Fill // International Conference on Evaluation and Modeling Methods for Systems Analysis and Development, 2025, pp 415–430. https://doi.org/10.1007/978-3-031-95397-2_26.

6. Embley-Riches, Jonathan. Unreal Robotics Lab: A High-Fidelity Robotics Simulator with Advanced Physics and Rendering / Jonathan Embley-Riches, Jianwei Liu, Simon Julier, Dimitrios Kanoulas // arXiv:2504.14135, 2025, 8 p. https://doi.org/10.48550/arXiv.2504.14135.

7. Munoz, Albert. Computer simulation and hands-on labs: A case study of teaching robotics and AI. / Alberto Munoz, Alexander Amigud, Ekaterina Sirazitdinova, // International Journal of Mechanical Engineering Education. 2024. Vol. 53, 10 p. https://doi.org/10.1177/03064190241240416.

8. Celikel, Resat. Design and Simulation of a Robotic System Integrated With Flywheel Energy Storage for Power Outage Resilience / Resat Celikel, Omur Aydogmus, Musa Yilmaz // Energy Science & Engineering, 2025, 13 p. https://doi.org/10.1002/ese3.70203.

9. Застосування цифрових двійників технічних засобів автоматизації для розроблення програмно-технічних комплексів АСУ ТП: Навчальний посібник / І. Ш. Невлюдов, С. П. Новоселов, О. В. Сичова. Харків: Видавництво Іванченка І. С., 2023. 267 с. ISBN 978-617-8059-95-8, DOI: 10.30837/978-617-8059-95-8.

10 Ghatge, Dipali. Interactive Robotic Arm Simulation / Dipali Ghatge, Pratham Patil, Atharva Algude, Shubhangi Chikane, Atharv Dhotre // International Research Journal on Advanced Engineering Hub (IRJAEH), 2024, Vol 2, pp. 1665-1668. 10.47392/IRJAEH.2024.0229.

11. Stein, Gordon & Jean, Devin & Kittani, Saman & Deweese, Menton & Ledeczi, Akos. (2025). A Novice-Friendly and Accessible Networked Educational Robotics Simulation Platform. // Education Sciences, 2025, Vol. 15, iss. 198, 36 p. 10.3390/educsci15020198.

12. Rabley, Andrew. Robotics Training: Virtual Simulation / Andrew Rabley, Chandru Sundaram // The Comprehensive Atlas of Robotic Urologic Surgery, 2025, pp. 9-17. 10.1007/978-3-031-72305-6_2.

13. Проектування мобільних маніпуляційних роботів: монографія /І. Ш. Невлюдов, А.О. Андрусевич, В. В.Євсєєв, С. П. Новоселов, Н. П. Демська. Х. :, 2022. 427 с.

14. BEAM робототехніка [Електронний ресурс] : Навчальний посібник / І. Ш. Невлюдов, В. В. Євсєєв, С. С. Максимова ; Харків. нац. ун-т радіоелектроніки, кафедра комп’ютерно-інтегрованих технологій,

автоматизації та робототехніки (КІТАР). Харків : Видавець Чернявський Д. О., 2024. 276 с. ISBN 978-617-8045-79-1.

15. Abou-Chakra. Real-is-Sim: Bridging the Sim-to-Real Gap with a Dynamic Digital Twin for Real-World Robot Policy Evaluation / Jad Abou-Chakra, Sun Abou-Chakra, Rana Lingfeng, May Krishan, Brandon Karl Schmeckpeper, Maria Minniti, Laura Herlant, // arXiv:2504.03597, 2025, 16 p. 10.48550/arXiv.2504.03597.

15. Ismail, Andi. CAD for Robotics: Trends, Opportunities, Considerations, and Constraints / Andi Ismail, Azmi Shawkat, Ismail Panessai // International Journal of Artificial Intelligence, 2025, vol. 12, pp 53-67. 10.36079/lamintang.ijai-01201.849.

16. Zhang, Julie. Development and Assessment of an Introductory Robotics Course to Adapt Industry 4.0 to Manufacturing and Automation Engineering Technology Programs / Julie Zhang, Rukmini Revuru, Lisa Riedle, // The Journal of Technology, Management, and Applied Engineering vol. 1, 3 p. 10.31274/jtmae.17932.

17. Sophie, Emily. Automation-driven technical support: leveraging robotics for real-time quality monitoring in construction projects / Emily Sophie // ResearchGate Publishing, 2025, Vol. 11, 12 p.

18. Pestana, Larissa. A Rapid Review on Advanced Software-in-the-Loop Methods for Automotive Software Verification / Larissa Pestana, Breno Miranda // Anais do IX Simpósio Brasileiro de Testes de Software Sistemático e Automatizado, 2024, pp. 56-65. 10.5753/sast.2024.3883.

19. Bonetto, Elia. GRADE: Generating Realistic and Dynamic Environments for robotics research with Isaac Sim / Elia Bonetto, Chenghao Xu, Aamir Ahmad, // The International Journal of Robotics Research, 2025, 33 p. 10.1177/02783649251346211.

20. Baltabay, Dauren. Creation of an algorithm for the 3D modeling of manipulator motion and forward positional kinematics / Dauren Baltabay, Muratulla Utenov, Tarek Sobh, Sichen Yuan, Zhadyra Zhumasheva, // International Journal of

Innovative Research and Scientific Studies, 2025, vol 8, pp. 2454-2465. 10.53894/ijirss.v8i3.7026.

21. Lynch, Kevin. Wheeled Mobile Robots / Kevin Lynch, Frank Park // Modern Robotics Mechanics, Planning, and Control, 2024, pp. 445-489. 10.1017/9781316661239.016.

22. Ferigo, Diego. Robot Platforms and Simulators / Diego Ferigo, Alberto Parmiggiani, Elena Rampone, Vadim Tikhanoff, Silvio Traversaro, Daniele Pucci, Lorenzo Natale // Cognitive Robotics, 2022, pp 124-144. 10.7551/mitpress/13780.003.0012.

23. Laabousse, Ismail. Advanced Software-in-the-Loop Simulation Environment for UAV Control Algorithms / Ismail Laabousse, Mohamed Haidoury, Younes Idrissi // 5th International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET), 2025, pp. 1-5. 10.1109/IRASET64571.2025.11008351.

24. Chen, Jingyu. A Survey of Real-time Simulation and Hardware-in-the-loop Technology / Jingyu Chen // Transactions on Computer Science and Intelligent Systems Research, 2024, vol. 6, pp.464-470. 10.62051/z5rekr42.

25. Oyediran, Hafiz. Integration of 4D BIM and Robot Task Planning: Creation and Flow of Construction-Related Information for Action-Level Simulation of Indoor Wall Frame Installation / Hafiz Oyediran, & William Turner, Kyungki Kim, Matthew Barrows // Journal of Information Technology in Construction, 2025, vol 30, pp. 352-374. 10.36680/j.itcon.2025.015.

26. Bronnikov, A. Розроблення кінематичної моделі маніпулятора на базі Abb Robot Studio / A. Bronnikov, M. Bendeberia, // Сучасний стан наукових досліджень та технологій в промисловості, 2024, № 4(30), с. 5–18. doi: 10.30837/2522-9818.2024.4.005.

27. Jdidou, Aymane. Synergistic Integration of Gazebo and ROS: Advancing Robotic Simulations and Learning Scenarios in Research / Aymane Jdidou, Aammou Souhaib // Technological Tools for Innovative Teaching, 2023, 27 p. 10.4018/979-8-3693-3132-3.ch011.

28. Saini, Vaibhav. MATLAB and Simulink for Building Automation / Saini, Shah Vaibhav, Ravi Sekhar Pritesh // IEEE Bombay Section Signature Conference (IBSSC), 2022, pp. 1-6. 10.1109/IBSSC56953.2022.10037485.

29. Mumba, Emmanuel. Enhanced autonomous driving within Webots simulation for student experiments / Emmanuel Mumba, Abubakar Gezawa, Chibiao Liu // Intelligent Service Robotics, 2025, pp. 1-21. 10.1007/s11370-025-00608-y.

30. Yuan, Zekun. Dynamic Simulation Study of Artillery Firing Mechanism based on ADAMS / Zekun Yuan, Lizhou Li, Chen Wan, Jiajun Zhao // Journal of Physics: Conference Series, 2024, vol. 2891, 11p. 10.1088/1742-6596/2891/13/132014.

31. Sudandhira Veeran. CoppeliaSim: Adaptable modular robot and its different locomotions simulation framework / Veeran Sudandhira, Suraj Rooban, Vali Shaik, Dhanush Nagandla Shaik // Materials Today: Proceedings, 2021, 7 p. 10.1016/j.matpr.2021.01.055.

32. CoppeliaRobotics User Manual [Електронний ресурс]. – Режим доступу: https://manual.coppeliarobotics.com/en/apiFunctions.htm.

33. Методичні вказівки до лабораторних робіт з дисципліни «Теорія автоматичного управління» для студентів усіх форм навчання спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» освітня програма Автоматизація та комп’ютерно-інтегровані технології [Електронне видання] / Упоряд.: О.В. Токарєва.– Харків: ХНУРЕ, 2022. – 86 с.

34. Методичні вказівки до виконання розділу "Охорона праці" у випускних роботах ОКР "бакалавр" усіх форм навчання / упоряд.: В. А. Айвазов. Т. Є. Стиценко., Н. Л. Березуцька ; М-во освіти і науки України, ХНУРЕ. – Харків : ХНУРЕ, 2018. – 28 с. – 1,81.