НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Кооперативний пошук шляху у середовищі ROS**»**

Виконав:
студент IV курсу, групи КА-75
Баськов Владислав Сергійович _____

Керівник:
Професор, д. т. н., Данилов Валерій Якович _____

Консультант з нормоконтролю:
Доцент, к. т. н., Коваленко Анатолій Єпіфанович _____

Консультант з економічного розділу:
Доцент, к. е. н., Рощина Надія Василівна _____

Рецензент:
Професор ФТІ, д. т. н., Новіков Олексій Миколайович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітня програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«26» травня 2021 р.

ЗАВДАННЯ
на дипломну роботу студенту
Баськова Владислава Сергійовича

1. Тема роботи «Кооперативний пошук шляху у середовищі ROS», керівник роботи Данилов Валерій Якович, професор, д. т. н., затверджені наказом по університету від «26» травня 2021 р. № 1344-с

2. Термін подання студентом роботи 08.06.2021.

3. Вихідні дані до роботи: стандарти та бібліотеки системи ROS.

4. Зміст роботи: дослідження предметної області пошуку шляху, аналіз алгоритмів кооперативного пошуку, модель системи пошуку шляху у середовищі ROS.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація – етапи вирішення задачі пошуку, класифікація алгоритмів, візуалізація роботи програмного пакету, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н. В.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми БДР	15.04.2021 - 21.04.2021	Виконано
2.	Ознайомлення зі структурою БДР згідно з Положенням про державну атестацію студентів НТУУ «КПІ»	15.04.2021 - 28.04.2021	Виконано
3.	Ознайомлення з ДСТУ 3008-95 та стандарти ЄСПД	21.04.2021 - 28.04.2021	Виконано
4.	Проведення дослідження за темою БДР під керівництвом керівника	28.04.2021 - 16.05.2021	Виконано
5.	Завершення роботи над першим варіантом частини БДР	05.05.2021 - 16.05.2021	Виконано
6.	Проведення роботи над експериментальною частиною БДР	05.05.2021 – 26.05.2021	Виконано
7.	Проведення роботи над програмним продуктом	19.05.2021 – 02.06.2021	Виконано
9.	Оформлення БДР та аналіз отриманих результатів	19.05.2021 – 03.06.2021	Виконано

Студент Владислав БАСЬКОВ

Керівник Валерій ДАНИЛОВ

РЕФЕРАТ

Дипломна робота: 110 с., 6 табл., 28 рис., 2 дод., 15 джерел.

ПОШУК ШЛЯХУ, ROS, МУЛЬТИАГЕНТНИЙ ПОШУК, АЛГОРИТМИ ПОШУКУ.

Об’єкт дослідження – методи кооперативного пошуку.

Мета роботи – проаналізувати існуючі методи, виконати порівняння та власну реалізацію кооперативного пошуку.

Алгоритми пошуку шляху мають безліч застосувань на сьогоднішній день. Галуззю що активно розвивається, де використання алгоритмів пошуку найбільш нагально, є робототехніка. Однією з окремих задач пошуку є кооперативною задачею пошуку. Ця задача полягає у вирішення задачі пошуку у мульти-агентному середовищі.

У цій роботі було досліджено та проаналізовано існуючі алгоритми пошуку та, зокрема, кооперативного пошуку. Також було реалізовано програмний пакет робототехнічного середовища ROS, що реалізує задачу пошуку.

ABSTRACT

Bachelor thesis: 110 p., 6 tabl., 28 fig., 2 appendixes, 15 references.

PATHFINDING, ROS, MULTI-AGENT PATH PLANNING, PATH SEARCH ALGORITHMS

The object of research - methods of cooperative search.

The purpose of research - analyze the existing methods, perform comparisons and implementation of cooperative search.

Pathfinding algorithms have many applications today. An actively developing industry, where the use of search algorithms is most urgent, is robotics. One of the special pathfinding tasks is the cooperative search task. This task includes pathfinding in a multi-agent environment.

In this paper, the existing search algorithms and, in particular, cooperative search were investigated and analyzed. The software package for the ROS environment, which applies the pathfinding task, was also implemented.

ЗМІСТ

ВСТУП

У широкому сенсі, пошук шляху можна визначити як пошук найкоротшого маршруту з початкового стану до цільового стану. Керуючись цим визначенням, складно не помітити як часто ми зустрічаємося з проблемами, що можуть бути охарактеризовані як проблеми пошуку шляху. Через розгляд станів та переходів між ними можливо описати велику кількість повсякденних задач, від побудови маршруту подорожі до поетапної підготовки певної події.

Як класичну проблему теорії графів, пошук шляху вперше був вирішений за допомогою алгоритму Дейкстри. Однак з швидким розвитком апаратних засобів та інформаційних технологій також зростала потреба у нових рішеннях для пошуку шляху. Поява нових алгоритмів спричинена доданням нових сценаріїв, обмежень та специфічних вимог, що мають враховуватися при пошуку шляху.

Однією з можливих вимог може буди вимога до здатності уникнення перешкод, причому, якщо ці перешкоди можуть змінюватися з плином часу, постає питання динамічного пошуку шляху. Найбільш актуальним на даний момент є рішення задачі динамічного пошуку шляху для автономного транспорту. При цьому виникає велика кількість супутніх проблем пов’язаних з великою кількістю сфер, як технічних, наприклад, проблема розпізнання образів, так і філософських.

Кооперативним пошуком шляху називають задачу мультиагентного пошуку шляху, де кожен з агентів повинен знайти маршрут до мети, що не конфліктує з маршрутами інших агентів, враховуючи інформацію про них. Відповідно, агентів можна розглядати як динамічні перешкоди по відношенню один до одного, проте, на відміну від ситуації, коли майбутнє положення перешкод необхідно прогнозувати, агенти мають можливість обмінюватися даними про планування своїх майбутніх позицій.

Метою даної роботи є огляд та аналіз існуючих методів кооперативного пошуку шляху, експериментальне моделювання пошуку шляху у середовищі для робототехнічної розробки ROS та порівняльний аналіз результатів, отриманих у ході роботи.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність поставленої задачі

Протягом усієї історії людства ми ставимо перед собою задачі, пов’язані з пошуком шляху. Від вирішення логістичної проблеми постачання перших міст провізією та прокладанням перших торгівельних маршрутів, до побудови шляхів для дослідницьких роверів чи персонажів комп’ютерних ігор. Саме тому потреба у алгоритмах, що вирішували би задачу пошуку шляху за своїх, особливих, умов, та покращення цих алгоритмів, завжди була та залишається актуальною.

Задачу пошуку шляху у просторі можна поділити на задачі глобального та локального пошуку. Локальний пошук будує маршрут з урахуванням інформації що находить з сенсорів пристрою, у той час як глобальний пошук полягає у створенні мапи, та, на основі усієї наявної інформації, побудові загального маршруту до мети. Розглянемо деякі сфери де активно використовуються алгоритми вирішення задачі пошуку.

Напевно найбільш очевидним місцем де вирішується проблема пошуку шляху є застосунки що будують маршрут на основі мапи. Оскільки при використанні подібних застосунків немає необхідності у врахуванні малих динамічних перешкод, справа йде саме про проблему глобального пошуку шляху. При чому, застосунок може містити додаткові умови, як наприклад пошук найкоротшого за часом шляху з урахуванням завантаженості доріг, чи побудова маршруту що має проходити через певну кількість додаткових точок. (Рис. 1.1) [1]

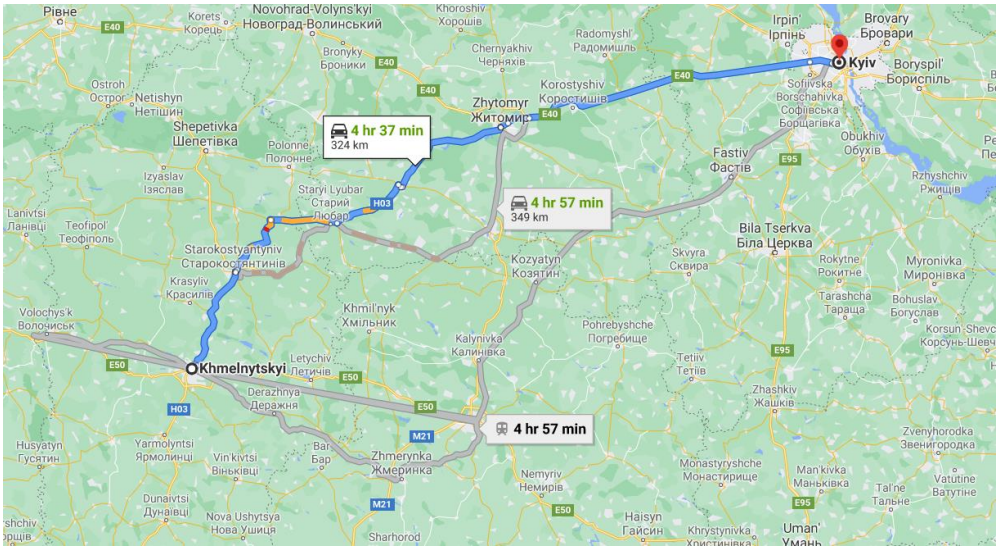


Рисунок 1.1 – Приклад побудованого маршруту, за допомогою www.google.com/maps

Автономний транспорт зазвичай покладається на створені подібними застосунками маршрути для вирішення проблеми глобального пошуку. Саме тому у роботі з автономним транспортом на передній план виходить проблема локального пошуку шляху. При цьому, у випадку коли у якості динамічних перешкод можуть виступати інші люди також виникають моральні дилеми, надання відповідей на які необхідно для прийняття рішень в умовах побудови шляху. За прогнозами поява транспортних засобів з п’ятим рівнем автономності, тобто здатного до роботи за повної відсутності водія, можлива вже у цьому десятиріччі. (Рис. 1.2) [2]

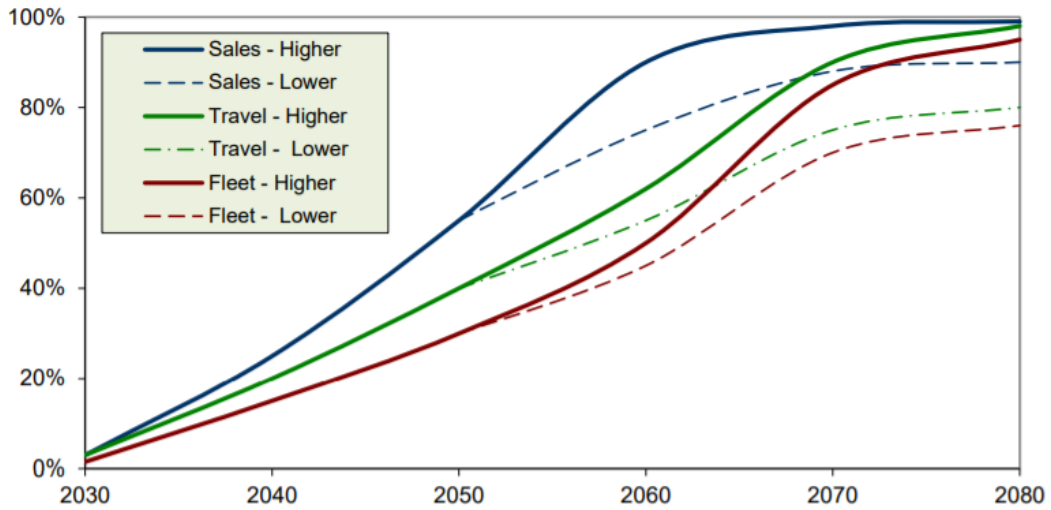


Рисунок 1.2 – Прогнози по відсотку продажу, експлуатації та автопарку електромобілів

Існує потреба у використанні роботів у індустрії, що займаються переміщенням вантажу на фабриках чи виробництві. Основним фактором, що мотивує перехід від людської праці до автоматизації є оптимізація процесів за мінімізації витрат. Саме тому подібні системи мають легко масштабуватися та витрачати мінімум коштів на підтримання.

Також, є досить велика кількість більш специфічних задач виробництва та повсякденного вжитку, що опосередковано також змушені вирішувати задачу пошуку шляху. Як приклад можна привести роботів для побудови мапи чи відслідковування прогресу забудови (Рис. 1.3), або роботів для 3-друку будівель. Звичні для багатьох робо-пилососи також займаються плануванням руху з урахуванням умови покриття усієї доступної площі.

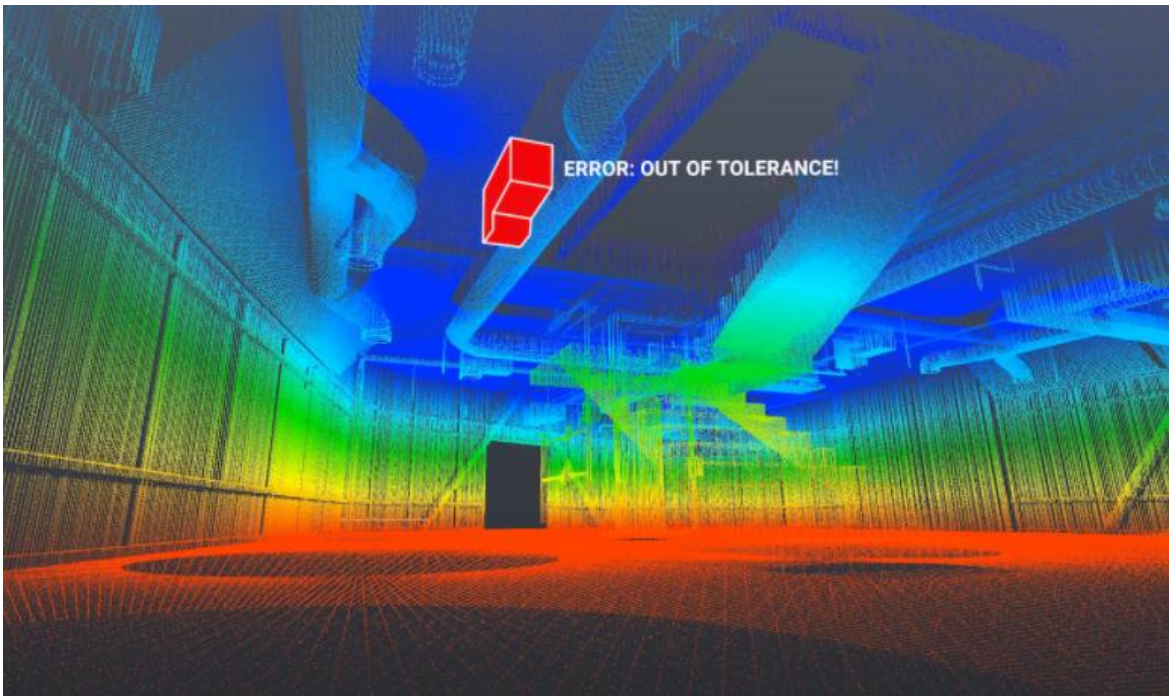


Рисунок 1.3 – Приклад відслідковування стану забудови [3]

Планування шляху для роботів також може використовуватися у дослідницьких цілях, а саме у дослідженнях важкодоступних та небезпечних місць. Такими місцями можуть бути будь-які місця де дослідження людиною є неможливим чи надзвичайно складним, і хоча печери, пустині чи глибини океану також використовують робототехніку для досліджень, найбільш помітними є робототехнічні дослідження за межами землі, а саме марсіанські

чи місячні ровери. (Рис. 1.4)

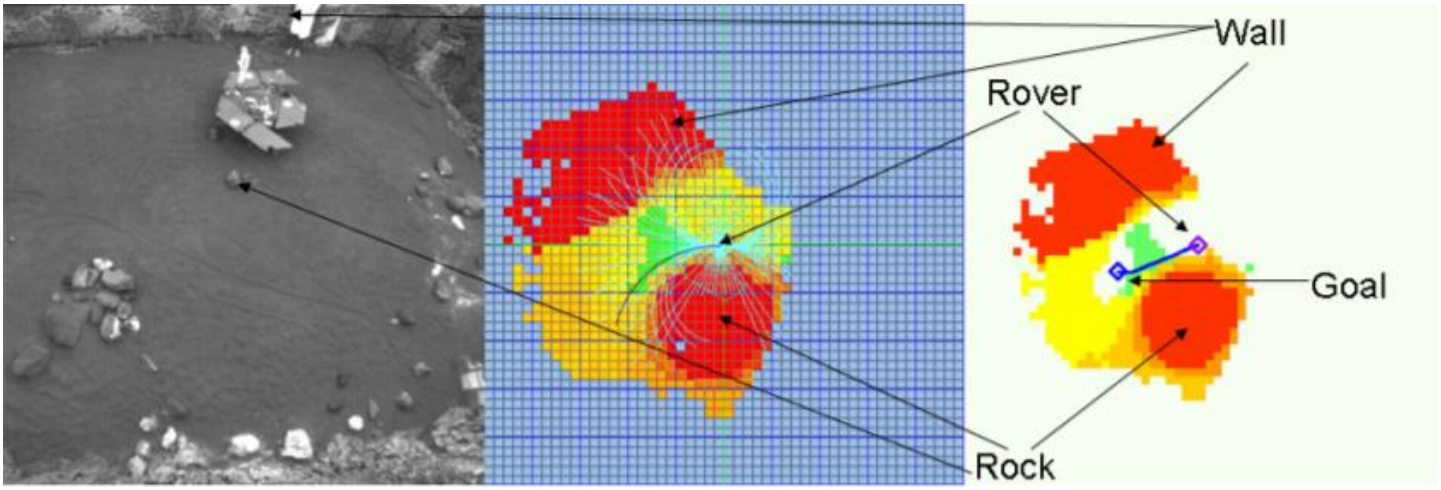


Рисунок 1.4 – Пошук шляху марсіанським ровером [10]

Іншими актуальними сферами є комп'ютерне моделювання систем та комп'ютерні ігри. До моделювання систем можна віднести задачі моделювання поведінки натовпів людей, як, наприклад, для оцінки планування екстрених виходів у екстремальних ситуаціях.

Комп'ютерні ігри є сферою що активно розвивається та залучає значного фінансування. За прогнозами, очікується що ігровий ринок досягне 287,1 мільярда доларів США до 2026 року, демонструючи CAGR у 9,24% протягом 2021-2026. [4]

Одним з останніх трендів також можна відмітити все частіше використання дронів для найрізноманітніших завдань. Таким чином, вирішення проблеми пошуку шляху необхідне як для дронів що займаються доставкою, так і для дронів військового призначення.

Щодо сфери яка займається виключно проблемою кооперативного пошуку шляху, зараз активно розвивається групова робототехніка (swarm robotics). Групова робототехніка зосереджена на дослідженнях емерджентної поведінки великої кількості достатньо простих агентів. До потенційних застосувань групової робототехніки можна віднести медичне використання нанороботів, рятувальні місії, 3д-друк, видобуток корисних копалин та багато інших.

1.2 Розвиток існуючих підходів до вирішення задачі пошуку шляху

У своїй основі, задачу пошуку шляху можна визначити як задачу пошуку на графі, де точка початку відповідає початковій вершині, а пошук відбувається по усіх інших вершинах доки мета не буде досягнута. Оскільки алгоритми пошуку на графах призначені для роботи з графами, виникає додаткова проблема побудови графу що відповідає “топології простору”.

Методи представлення неперервного простору у вигляді графу можна умовно розбити на дві техніки, скелетизацію та декомпозицію на комірки. Скелетизація створює “скелет” середовища, витягуючи характерну топологію простору задаючи граф G = (V, E), де V – набір вершин що відповідають певним координатам простору, а E – набір ребер, поєднуючих вершини що знаходяться в полі зору одна до іншої. Декомпозиція на комірки виконується за допомогою розбиття простору, по якому можливий рух, на окремі клітинки. Клітинки зазвичай визначаються колом чи опуклим многокутником, що дає можливість агенту пересуватись між будь-якими двома точками у середині клітини по прямій лінії. [5]

Після обробки простору, постає питання вибору алгоритму пошуку шляху. Вперше, пошук шляху на графі був вирішений за допомогою алгоритму Дейкстри у 1956 році. Ініціалізація алгоритму полягає у наданні початковій вершині мітки зі значенням 0, а іншим вершинам мітки зі значенням нескінченності. У якості шагу алгоритму береться нерозглянута вершина з мінімальною міткою. Для кожного сусіда розглядуваної вершини додаємо мітку розглядуваної зараз вершини до ваги ребра що веде до сусіду, та порівнюємо з наявним значенням мітки сусіду. Якщо нове значення мітки менше – присвоюємо його цій вершині. Коли розглянуто всі вершини алгоритм закінчує роботу. [6]

Проте, через значну витрату ресурсів для обчислень, алгоритм Дейкстри майже не використовується для пошуку шляху у просторі. Більш відомим алгоритмом для цієї задачі є алгоритм A*. Базуючись на ідеї алгоритму

Дейкстри, алгоритм A* оцінює вершини як

$$f(i) = g(i) + h(i) \quad (1.1)$$

При цьому, g(i) – вага шляху від початкової вершини до вершини i, а h(i) – деяка евристична функція, тобто функція що дає приблизну оцінку ваги шляху від вершини і до цільової вершини. Якщо евристична функція ніколи не переоцінює фактичну вагу мінімального шляху до цільової вершини, така евристика називається допустимою. У оригінальному варіанті алгоритму A* у якості евристичної функції використовується Мангеттенська метрика, за якою відстань між двома точками вираховується як сума модулів різниць їх координат. Додання евристичної функції необхідне для попередньої оцінки вершин, що допомагає розглядати лише ті вершини, що наближують нас до знаходження мети.

У випадках коли можливе виникнення непередбачуваних умов, постає питання знаходження не тільки оптимального шляху, але й шляхів що є наступними по оптимальності. Ця проблема називається проблемою k-коротших. Алгоритми, що її вирішують, можна поділити на загальні алгоритми, та алгоритми без циклів.

Одним з основних алгоритмів для вирішення проблеми k-коротших без циклів є алгоритм Йена. Алгоритм можна розділити на етап знаходження першого за оптимальністю шляху та на етап знаходження наступних за оптимальністю шляхів, за допомогою варіювання вершин вже знайдених оптимальних шляхів. Для варіювання, для усіх вершин окрім кінцевої знаходимо найкоротший шлях від них до мети, без урахування ребер на яких лежить вже розглянутий оптимальний шлях. Знайдені у результаті варіювання шляхи порівнюються, і найкоротший зберігається для подальшого використання.

До загальних алгоритмів вирішення проблеми k-коротших можна віднести алгоритм маркування та алгоритм видалення шляху. Ідеєю

алгоритмів маркування є виконання послідовних ітерацій прямого та зворотного розгортання до тих пір, поки ваги маркування не перестануть змінюватися. Ідеєю алгоритмів видалення шляху є виключення з графу G оптимального шляху, тим самим створюючи новий граф G`, та знаходження нового оптимальному шляху на ньому.

Як один з алгоритмів, що може працювати з від'ємними вагами, будучи корисним для пошуку шляху при декількох цілях, можна привести приклад алгоритму Беллмана-Форда. Суть алгоритму полягає в розгляді усіх ребер графу та спробі поліпшення ваг відповідних вершин за допомогою ваг сусідніх вершин та ваг ребер що їх поєднують. Розгляд повторюється n – 1 разів, де n – кількість вершин. Також, у якості покращення алгоритму Беллмана-Форда, було створено швидкий алгоритм коротшого шляху SPFA, що відрізняється наявністю черги з вершин, що розглядаються для покращення.

Для ситуації, коли необхідно знаходження шляхів від декількох початкових точок, до декількох відповідних кінцевих точок, можна використовувати алгоритми Флойда-Воршелла чи алгоритм Джонсона. Ідея алгоритму Флойда-Воршелла є доволі примітивною, вона полягає у порівнянні всіх можливих шляхів між кожною парою вершин за допомогою матриць. Саме тому алгоритм є простим у реалізації, проте сильно витратним у підрахунках. Алгоритм Джонсона спочатку корегує ваги ребер, задля балансування негативних ваг. Потім створюється новий граф з доданням вершини, що є сусідньою до усіх інших, та пов'язана з ними за допомогою ребер з нульовою вагою. Далі алгоритм Беллмана-Форда використовується, починаючи з доданої вершини, для знаходження для кожної вершини v мінімальної ваги шляху до доданої вершини – h(v). Після цього ваги ребер оригінального графу корегуються за допомогою знайдених значень як

$$w`(u, v) = w(u, v) + h(u) – h(v) \qquad (1.2)$$

Далі алгоритм Дейкстри використовується для знаходження

найкоротших шляхів між усіма вершинами.

Підсумовуючи, можемо розбити розглянуті алгоритми за класом виконуваних задач:

Пошук з однієї точки старту до однієї кінцевої точки одного маршруту:

- Алгоритм Дейкстри
- A* з варіаціями

Пошук з однієї точки старту до однієї кінцевої точки декількох маршрутів:

- Алгоритм Йена
- алгоритм маркування
- алгоритм видалення шляху

Пошук з однієї точки старту до декількох кінцевих точок:

- алгоритм Беллмана-Форда
- SPFA

Пошук з декількох початкових точок до декількох кінцевих точок:

- алгоритм Флойда-Воршелла
- алгоритм Джонсона

Одним з актуальних направлень досліджень рішень пошуку шляху пов’язане з бурхливим розвитком технологій нейронних мереж. У випадку з нейронними мережами, ми зводимо проблему пошуку шляху не до пошуку шляху у графі, а до задачі машинного навчання, де у якості вхідних параметрів ми використовуємо дані з сенсорів, та на виході отримуємо рішення про рух засобу. Проте, складність обчислень та необхідність тренування нейронних мереж є значним недоліком даного підходу. Також можливе поєднання підходів, коли, наприклад, алгоритми пошуку на графі використовуються для глобального пошуку, але для локального ми використовуємо алгоритми машинного навчання.

1.3 Опис середовища ROS

Robot Operating System (ROS) – це відкрите програмне забезпечення для роботи з робототехнікою, що являє собою набір програмних фреймворків та сервісів для полегшення проектування, розробки та тестування робототехнічних проектів. Сервіси ROS забезпечують інструменти для абстрагування апаратного забезпечення, низькорівневого керування пристроями, реалізації часто використовуваного функціоналу та керування пакетами.

Історію створення ROS можна розпочати у Стенфордському університеті, коли двоє аспірантів, Ерік Бергер та Кінан Виробек, працюючи над Проектом Персональної Робототехніки, помітили складність з якою змушені були боротися їхні колеги. Робототехніка вимагає серйозних міждисциплінарних знань, але часто трапляються ситуації, коли чудовий розробник програмного забезпечення може не володіти достатніми апаратними знаннями, а хтось, хто розробляє найсучасніші шляхи планування, може не знати, як забезпечити механізм комп'ютерного зору. Саме це стало мотивацією для створення перших прототипів системи, що стала попередником ROS.

У ході пошуку фінансування для розробки свого проекту, Ерік Бергер та Кінан Виробек зустрілися зі Скотом Хассаном, засновником робототехнічної лабораторії Willow Garage. Хассан розділив їхнє бачення проекту системи для Linux, та запросив їх у свою компанію на роботу. Саме тоді, у 2007 році, було зроблено перший комміт до коду системи ROS. Протягом 2007-2013 років система ROS активно розвивалася, причому свій внесок зробила велика кількість установ. Перший офіційний випуск ROS-дистрибутива: ROS Box Turtle був випущений 2 березня 2010 року, що визначило момент, коли ROS вперше було офіційно розповсюджено із набором пакетів для загального користування. Це призвело до створення першого безпілотника, що працював на основі ROS, першого автономного автомобіля на основі ROS, та адаптації

ROS для Lego Mindstorms. [7]

Willow Garage почали роботу над створенням некомерційної організації Open Source Robotics Foundation (OSRF) у 2012 році. Після чого, у 2013 році основна підтримка проекту ROS перейшла до OSRF. Одним з найважливіших напрямків розвитку OSRF можна відзначити пропозицію та проектування система ROS2, що вносить значні зміни до ROS API, додаючи можливості програмування у режимі реального часу, більший набір інструментів та покращення використовуваних технологій.

Якщо ми намагатимемося описати загальну ситуацію з розробки робототехнічних проектів, можна відмітити що великі корпорації намагаються створювати свої системи, аналогічні ROS, для внутрішнього використання. Це надає їм можливість контролю над бібліотеками, специфічними умовами використання та інтелектуальною власністю. Наприклад, подібна система є у Microsoft - Microsoft Robotics Developer Studio.

Проте, що надає системі ROS значної переваги, це те що ROS є проектом з відкритим програмним забезпеченням та значною користувацькою базою, що забезпечує ROS великою кількістю пакетів і рішень, значно спрощуючи розробку.

Дистрибутивом у ROS називається набір пакетів ROS певної версії, що аналогічно до дистрибутивів Linux. Відповідно, при виборі дистрибутиву необхідно брати до уваги платформу що його підтримує, специфічних інструментів на які розрахований дистрибутив, а також дистрибутиви які підтримують пакети, що будуть використані у ході роботи над проектом. Зазвичай, нові дистрибутиви виходять щорічно, основною платформою для підтримки дистрибутиву є останній, на момент виходу дистрибутиву, дистрибутив Ubuntu.

Розглянемо більш детально структуру проекту ROS.

Структуру роботи ROS можна представити у вигляді графу обчислень, що складається з мережі процесів ROS, які сумісно оброблюють дані. Основні концепції обчислювальних графів ROS, це: вузли, Майстер, Сервер

Параметрів, повідомлення, сервіси, топіки та пакети. Відповідно кожна з цих концепцій працює з даними своїм, особливим, чином. (Рис. 1.5) [8]

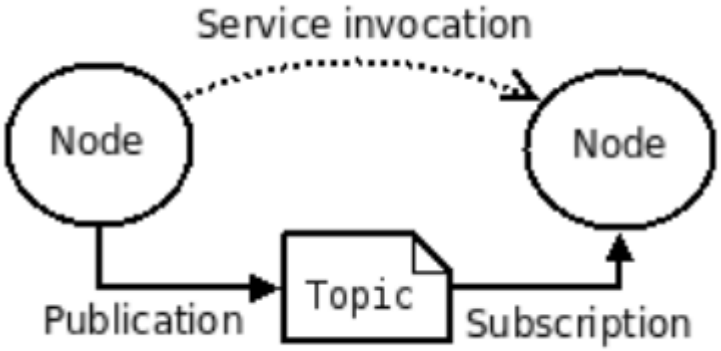


Рисунок 1.5 – Схематичне представлення графу обчислень системи ROS

- Вузлами у системі ROS, називаються процеси що відповідають за підрахунки. Оскільки дизайн системи ROS передбачає модульність, зазвичай система контролю робота включає у себе велику кількість вузлів. Як приклад, система може мати окремі вузли для контролю за рухом, для планування шляху, для розпізнання образів, обробки інформації з сенсорів чи виведення зображення з камер. Вузли у ROS пишуться за допомогою клієнтської бібліотеки ROS, що включає у себе пакети roscpp або rospy.
- Майстер у системі ROS відповідальний за реєстрацію імен та загальну підтримку графу обчислень. Без Майстра вершини не мали б можливості знайти один-одного, обмінюватися повідомленнями та викликати сервіси.
- Сервер Параметрів дозволяє зберігати дані за ключем у централізованому місці. В останніх версіях функціонал Серверу Параметрів виконується Майстром.
- Повідомлення є засобом комунікації між вузлами. Повідомлення реалізовані як структура даних, що містить визначені поля. Поля можуть містити стандартні типи даних, такі як цілі, булеві, символьні, та інші, а також масиви зі стандартних типів даних. Є

можливість створення своєї комбінації полів, проте більшість застосувань має реалізований варіант повідомлень, як, наприклад, повідомлення для відправки зображень чи повідомлення для відправки одометрії.

- Топіками називаються канали для передачі повідомлень. Обмін відбувається за схемою підписників / публікуючих. Вершина відправляє повідомлення шляхом публікації його у заданий топік. Кожен топік працює лише з повідомленнями одного типу. Вершина, що зацікавлена в отриманні повідомлень певного змісту, підписується на відповідний топік. Для кожного топіку можливе існування довільного числа підписників та публікуючих. Також, кожна вершина може підписуватись на довільну кількість топіків та публікувати повідомлення у довільну кількість топіків. Логічним є погляд на топіки, як на шини зі строго заданим типом повідомлень що приймаються.
- Сервіси використовуються для реалізації відношення запитів / відповідей. Оскільки відношення багато-до-багатьох, що використовують топіки, не є зручним для реалізації запитів, сервіси вирішують цю проблему. Сервіс забезпечується вершиною що його піднімає, у той час як клієнт може використовувати піднятий сервіс шляхом відправки запиту до нього, та очкувати на відповідь.
- Пакети, у контексті графу обчислень, являють собою формат для збереження інформації з повідомлень ROS, що може бути важливо при тестуванні алгоритмів.

Файлова система ROS складається з:

- Пакетів, що є основним елементом структури проекту. Пакети файлової системи ROS можуть складатися з записаних вершин, бібліотек, наборів даних, конфігураційних файлів, тощо.
- Метапакетів, що являють собою спеціалізовані пакети ROS,

призначені для представлення групи пов’язаних між собою звичайних пакетів ROS.

- Маніфестів пакетів (зберігаються у package.xml), що містять дані про відповідний пакет, включаючи його назву, версію, опис, інформацію про ліцензію, залежності та іншу мета інформацію, як, наприклад, інформацію про додаткові пакети що експортуються.
- Репозиторіїв, що є місцем зберігання для пакетів зі спільною системою контрою версій.
- Файлів для визначення типів структур, як наприклад файли повідомлень (msg) та файли визначення структури запитів і відповідей сервісів (srv).

Одним з інструментів, що інтегрований у систему ROS, є 3D-симулятор для робототехніки Gazebo. Gazebo має широкий функціонал для тестування алгоритмів, проектування роботів, та тренування систем штучного інтелекту. (Рис. 1.6)

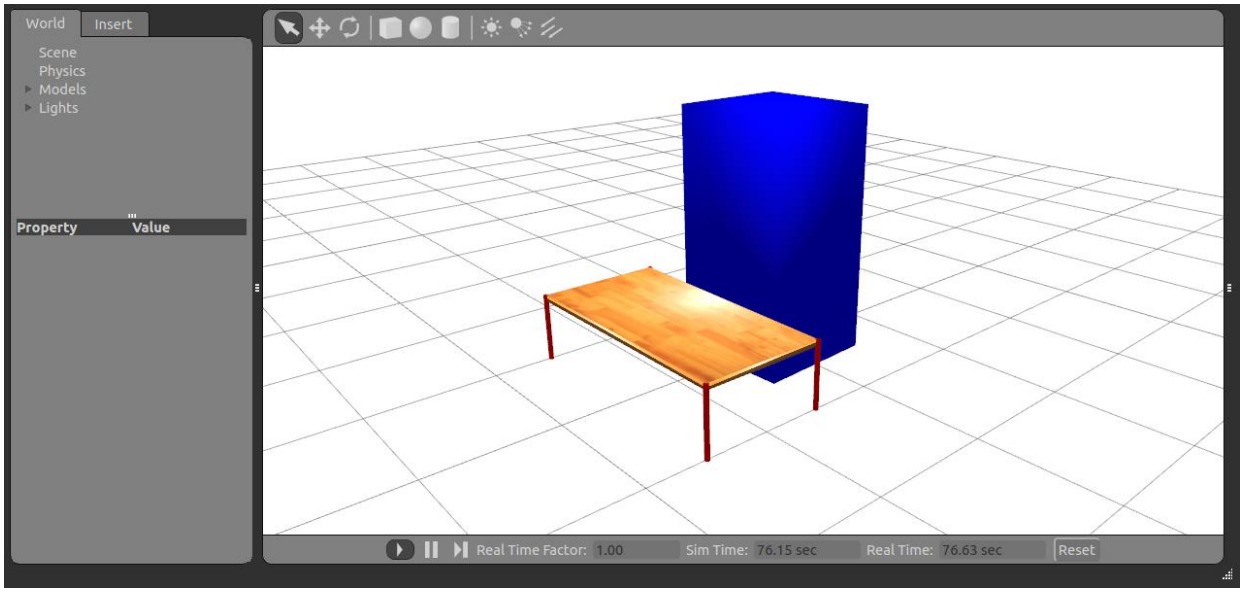


Рисунок 1.6 – Інтерфейс Gazebo

Іншим потужним інструментом, що використовується у ROS для тестування та виправлення помилок, є система RVIZ. RVIZ може відображати дані з камери, лазерів, 3D та 2D пристроїв, включаючи зображення та точкові сітки. (Рис. 1.7)

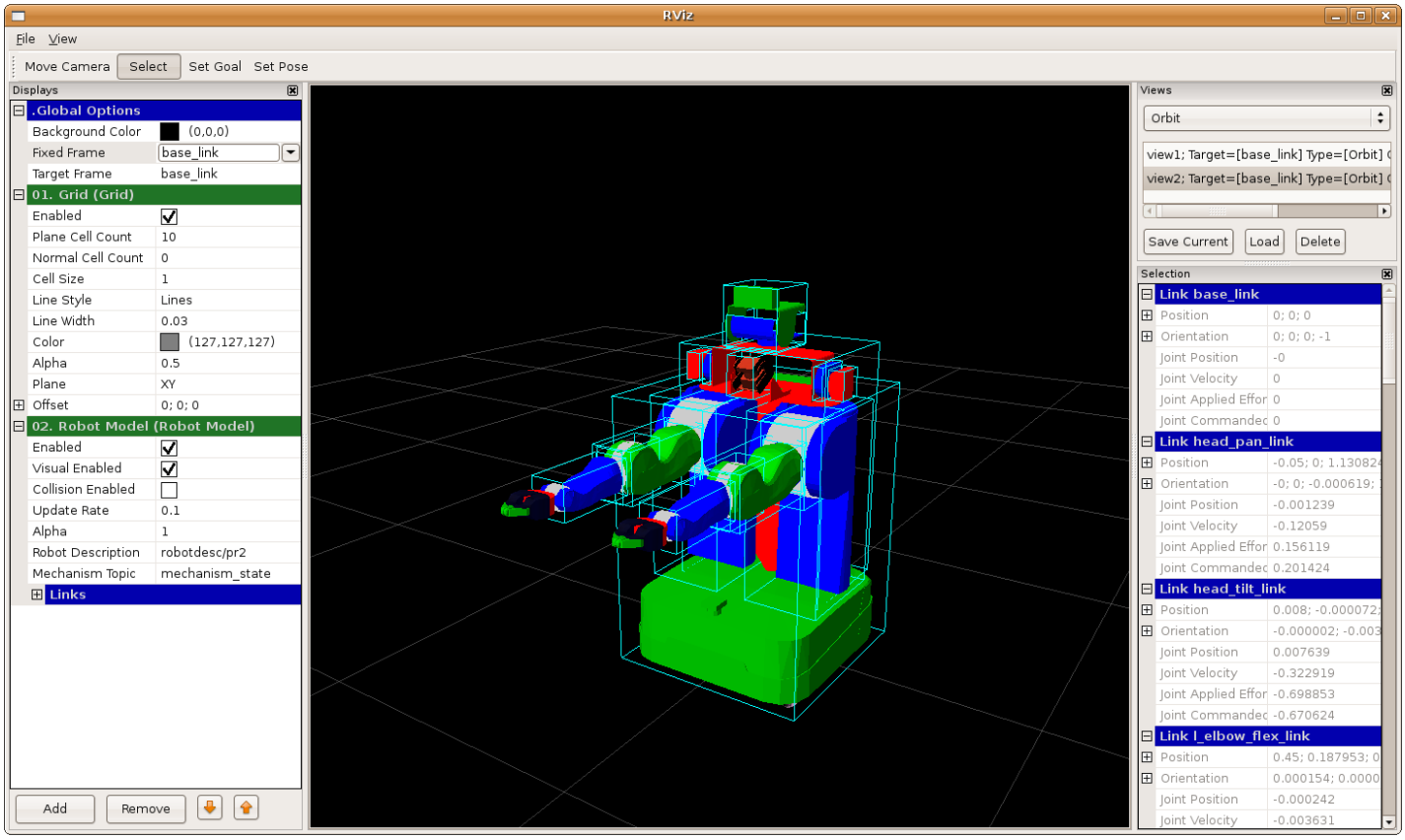


Рисунок 1.7 – Інтерфейс RVIZ

1.4 Постановка задачі дослідження

Проаналізувавши предметну область, можемо визначити напрямок подальшого дослідження, а саме, можемо виділити наступні етапи подальшої роботи:

- Проведення детального аналізу існуючих методів кооперативного пошуку шляху;
- Визначення критеріїв якості для задачі, а також вибір та формалізація власного методу вирішення задачі, для подальшої реалізації у системі ROS.
- Реалізація методу кооперативного пошуку шляху у системі ROS. За можливості оптимальним є реалізація декількох методів для їх подальшого порівняння.

- Аналіз отриманих результатів та пошук шляхів для покращення методу.

1.5 Висновки до Розділу 1

У даному розділі було виконано аналіз предметної області поставленої задачі.

Було розглянуто сфери у яких використовуються методи пошуку шляху для оцінки актуальності задачі.

Було розглянуто основні алгоритми, що використовуються для пошуку шляху.

Також було розглянуто особливості середовища ROS, історію його виникнення та структуру.

У результаті проведеного огляду, було формалізовано задачу для подальшої роботи над дослідженням.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

2.1 Поетапне дослідження існуючих методів для вирішення задачі пошуку шляху

2.1.1 Побудова мапи

При вирішенні задачі пошуку шляху питанням, що постає першим, є пошук способу представлення простору для подальшої роботи. Можливі два випадки – топологія простору нам може бути як відома, так і невідома. У залежності від цього маємо два кардинально різних підходи до вирішення проблеми пошуку руху.

У ситуації, коли нам не відома мапа, ми не маємо можливості для побудови глобального шляху. Саме тому, ситуація, у якій робот на має у наявності мапу, зазвичай пов’язана з дослідницькими задачами. Причому, задача може полягати як у, насамперед самому дослідженню, як, наприклад, побудова мапи для відслідковування прогресу на виробництві, лабораторні дослідження, військова розвідка, дослідження труднодоступних місць, тощо, так і використовувати методи побудови мапи для супутніх задач, як, наприклад, очистка приміщення робопилососом, видобуток корисних копалин, тощо.

Існує велика кількість алгоритмів побудови мапи. Як приклад одного з алгоритмів, що часто вживаються, можна привести алгоритм на основі кордонів. Суть методу полягає у визначенні меж між дослідженими областями навколишнього середовища, та областями, де інформація ще не зібрана. Робот шукає прохідні області у доступності, близькі до невідомих областей на вже дослідженій частині мапи. Стратегія найближчого кордону полягає в обчисленні відстаней від поточного положення робота до усіх прикордонних областей, та направленні робота до найближчої з них. Також можлива стратегія з обору довільного кордону для спостереження. Цей метод є доволі простим у використанні, а також ефективним по відношенню до часу покриття всієї мапи. У результаті роботи, найбільш підходящою репрезентацією для

створеної мапи є сіткова мапа.

Іншими прикладами типів представлення мапи на комірки можна назвати полігональні мапи, топологічні мапи та мапи на основі ліній, проте їх використання несе додаткові ускладнення, як, наприклад складність представлення просторів з великою кількістю об’єктів, недостатня точність представлення чи складність визначення ще не досліджених територій. (Рис. 2.1) [9]

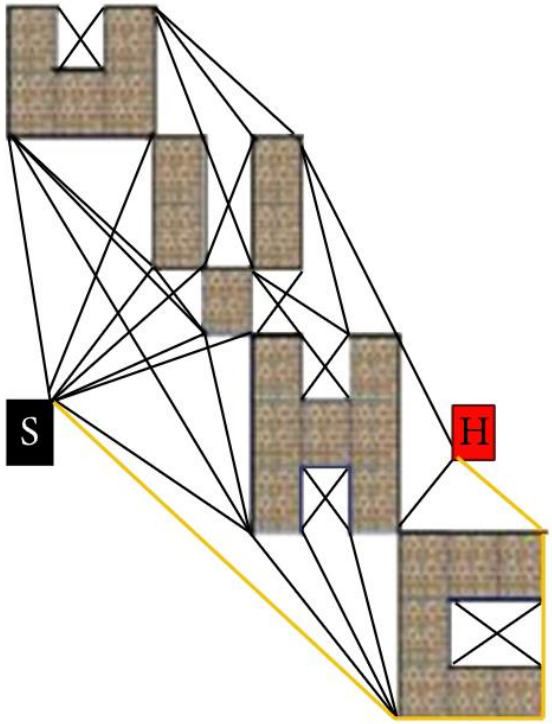


Рисунок 2.1 – Приклад нетипічного способу представлення графу у вигляді лінійний граф видимості.

У ситуації, коли мапа відома наперед, ми можемо використовувати класичні алгоритми пошуку глобального пошуку шляху. Насамперед, це включає у себе пошук шляху за допомогою систем GPS, розробку систем навігації, комп’ютерне моделювання, транспорт вантажів, тощо. На основі відомої мапи створюється представлення відповідно до заданої задачі. Часто, у ролі такого представлення виступає сіткова мапа, проте і при її створенні можливі нюанси. Так, наприклад, клітини мапи можуть поділятися за похідністю строго, тобто визначаючи кожну клітину як прохідну / не прохідну, або не строго, визначаючи кожній клітині певну вагу, у залежності від

складності її проходження.

Прикладом використання ваг у сітковій мапі є реалізація NASA глобального пошуку для марсіанських роверів. (Рис. 2.2) [10]

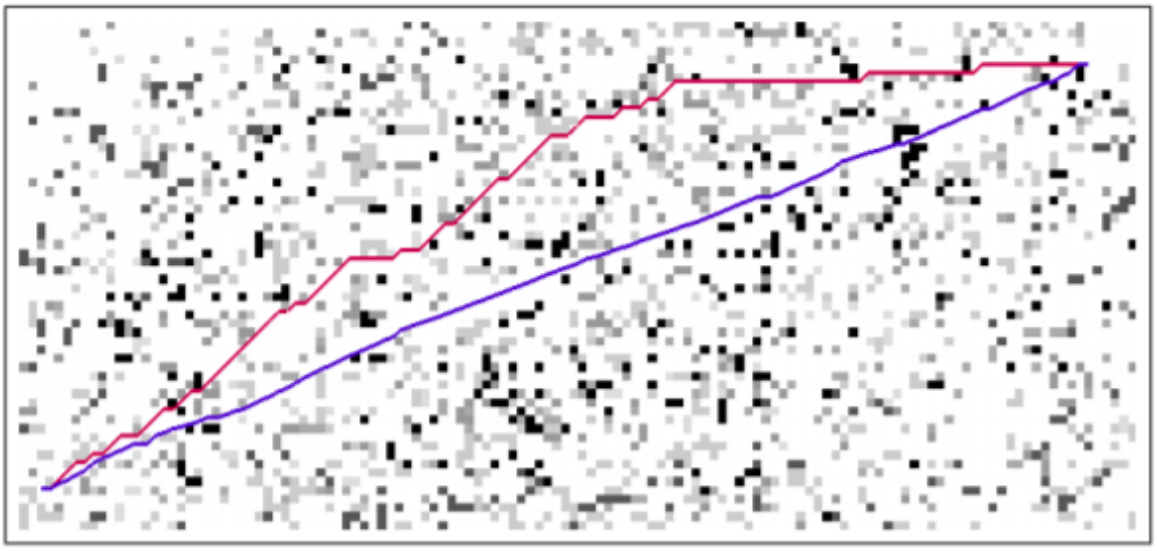


Рисунок 2.2 – Візуалізація маршрутів, прокладених класичним алгоритмом (червоний) та алгоритмом D* (синій) на сітковій мапі розміром 150×60. Темніші клітини позначають більшу вагу проходження цієї клітини ровером.

У ході виконання даної роботи доречно використання заданої сіткової мапи зі строго визначеною прохідністю клітин.

2.1.2 Глобальний пошук шляху

Наступним етапом при побудові методу пошуку шляху є вибір алгоритму для глобального пошуку шляху. Суть глобального пошуку полягає у знаходженні загального шляху до мети, що може вираховуватися на більш високому рівні абстрагування та ігнорувати можливу наявність динамічних перешкод.

Одним з найбільш вживаних алгоритмів для вирішення задачі глобального пошуку шляху є алгоритм A*. Як вже було зазначено, основною ідеєю алгоритму A* є поєднання вартості досягнення певного вузла n - g(n) та евристичної оцінки вартості проходження від цього вузла до кінцевої мети -

h(n):

$$f(n) = g(n) + h(n) \quad (2.1)$$

Це можна інтерпретувати як оцінку вартості найменш витратного шляху який проходить через певну вершину n. Функція h(n), що використовується для оцінки шляху до мети, є допустимою, якщо h(n) ніколи не переоцінює фактичну вартість шляху до мети з вершини n. Прикладом допустимої евристичної функції можна вважати відстань від вершини n до мети по прямій, оскільки така оцінка завжди буде давати найкоротший результат. Іншою властивістю евристичної функції є спадкоємність (або монотонність), що означає що для будь-якого вузла h(n) і для будь-якої дії a, оцінка вартості досягнення мети з вузла n не є більшим суми вартості етапу досягнення вузла n` та оцінки вартості досягнення мети з вузла n`:

$$h(n) \leq c(n, a, n`) + h(n`) \quad (2.2)$$

Одним з важливих наслідків визначення спадкоємності евристичної функції є висновок про те, що якщо функція h(n) спадкоємна, то значення функції f(n) вздовж будь-якого шляху є неспадним. Це легко продемонструвати порівнянням оцінки вузлів наступника та попередника.

Нехай оцінка вузла наступника:

$$f(n) = g(n) + h(n) \quad (2.3)$$

Оцінка вузла попередника:

$$f(n`) = g(n`) + h(n`) \quad (2.4)$$

Тоді, враховуючи умову спадкоємності:

$$h(n) \leq c(n, a, n`) + h(n`) \quad (2.5)$$

Маємо:

$$f(n) = g(n) + h(n) \leq g(n) + c(n, a, n`) + h(n`) = g(n`) + h(n`) = f(n`) \quad (2.6)$$

Сам алгоритм пошуку на графі можна описати досить просто. Спочатку ми записуємо стартову вершину до периферії. На кожній ітерації алгоритму ми беремо вершину з найменшою оцінкою f(n) з периферії, та розгортаємо її, додаючи її сусідів до периферії, причому якщо хтось з її сусідів вже міститься у периферії, але з більшою оцінкою, оновлюємо її. Якщо вершина що розглядається є цільовою, або досягнуто обмеження по кількості ітерацій – закінчуємо роботу. [11]

Не дивно, що завдяки популярності алгоритму A*, була створена велика кількість альтернативних алгоритмів що мають у своїй основі ідею алгоритму A*. Так, наприклад, ієрархічний алгоритм A* (HPA*) над працює шляхом розглядаючи варіанти окремо для різних рівнів абстрагування. Варіантом такого абстрагування може бути першочергове знаходження шляху між містами при подорожі, після чого уточнення шляху на рівні усередині міста. Також багато підходів полягає у виборі спеціальної евристичної функції. Якщо евристична функція дорівнює нулю для всіх вершин, алгоритм A* стає еквівалентним до алгоритму Дейкстри. Використання відстані до мети по прямій у якості евристичної функції призводить для пріоритетного розгляду вершин, що є ближчими до мети. Причому, певна переоцінка евристичної функції може пришвидшити роботу алгоритму шляхом відсікання більш віддалених вершин від мети. (Рис. 2.3) [12]

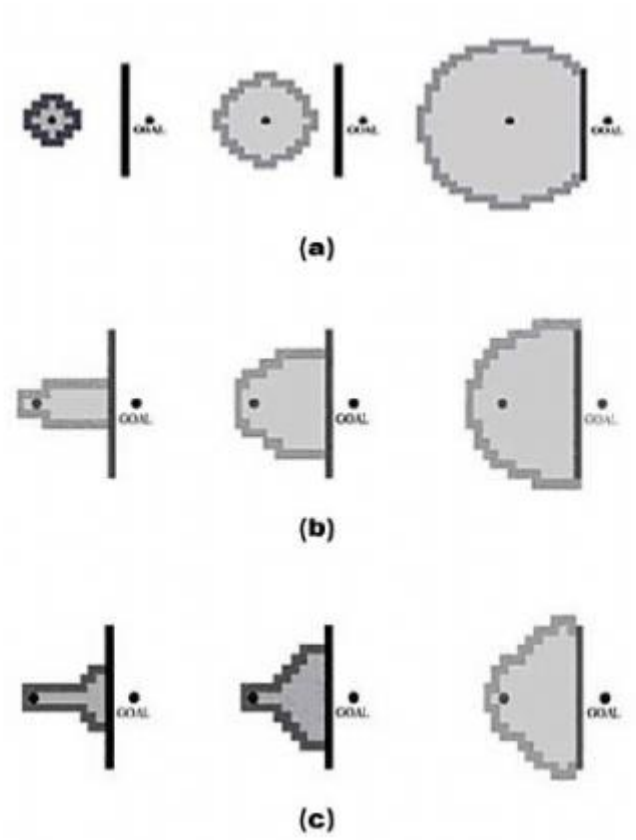


Рисунок 2.3 – Порівняння різних функцій евристики алгоритму A*

Також, у випадку якщо мапа являє собою сітку з вагами, доречно використовувати алгоритм D*, що використовує інтерполяцію для побудови маршруту з мінімальною вагою.

2.1.3 Локальний пошук шляху

Суть етапу локального пошуку шляху полягає у уточненні глобального маршруту з урахуванням можливих динамічних перешкод чи перетворення отриманого за допомогою глобального пошуку маршруту у відповідні точки шляху. Для цього, частина мапи що розглядається алгоритмом обмежується найближчим оточенням транспортного засобу, і оновлюється воно по мірі його руху.

Задля переоцінки мапи у головній мірі використовується інформація отримана за допомогою сенсорів транспортного засобу, через що обмеження

на частину мапи, якою оперує алгоритм, залежить від можливостей цих сенсорів.

Оперуючи даними про глобальний маршрут та про оновлену мапу, алгоритми локального пошуку генерують стратегії уникнення для динамічних перешкод і намагаються максимально узгодити результуючі траєкторії з наданими глобальним маршрутом точками.

Спочатку розглянемо задачу виявлення перешкод та оновлення мапи. З позиції сенсорів існують декілька головних типів вирішення цієї задачі.

Оцінка довкілля можлива за допомогою лазерів, що забезпечують достатньо точний вимір фізичних перешкод, проте у більшості випадків, не надають інформації, необхідної для точної класифікації об’єктів довкілля.

Іншим варіантом оцінки довкілля є комп’ютерне стереобачення, що реалізоване за рахунок зчитування цифрових зображень з камер. Цей метод надає можливість сприйняття трьох вимірної інформації шляхом порівняння зображень з декількох сусідніх камер, як це реалізовано у випадку біологічного зору. Хоча цей метод є більш інформаційно багатим, досягнення необхідної точності може бути доволі складним завданням. Саме тому, окрім алгоритмів стереобачення, інформацію з камер доречно використовувати для класифікації об’єктів методами комп’ютерного зору. (Рис. 2.4) [13]

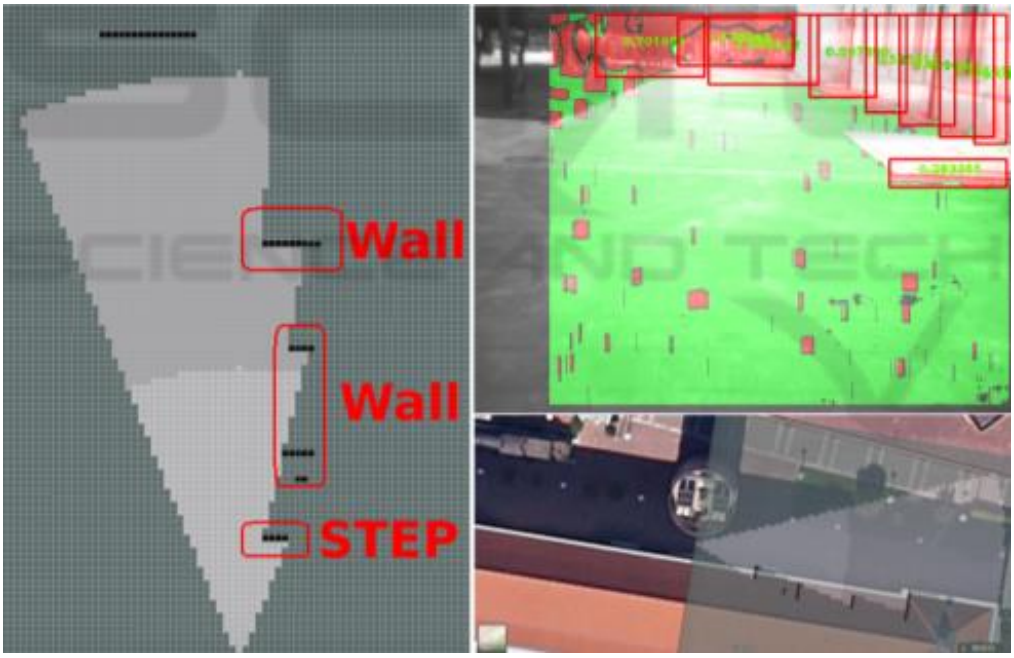


Рисунок 2.4 – Приклад роботи комп’ютерного стереобачення

Щодо задачі корегування траєкторії, технічно є можливим використання алгоритмів, аналогічних звичайним алгоритмам пошуку. Проте, необхідність постійного оновлення алгоритму у реальному часі вимагає пошуку більш оптимізованих підходів до вирішення задачі.

Прикладом алгоритму, призначеного для корегування траєкторії є алгоритм еластичних смужок час (Time Elastic Bands) або TEB, що займається деформування початкового глобального шляху з урахуванням кінематичної моделі транспортного засобу та оновленні локального шляху на основі мапи що враховує динамічні перешкоди чи інші відхилення від шляху. Алгоритм полягає у побудові череди проміжних позицій транспортного засобу. Вимогами алгоритму є дані про діапазон швидкостей та прискорення транспортного засобу, його кінематичні та динамічні обмеження та задання дистанції, що вважається безпечною при уникненні перешкод. Результатом виконання є генерація команд для задання швидкості (v) та кута повороту (δ), необхідних для досягнення визначених проміжних позицій. (Рис. 2.5) [14]

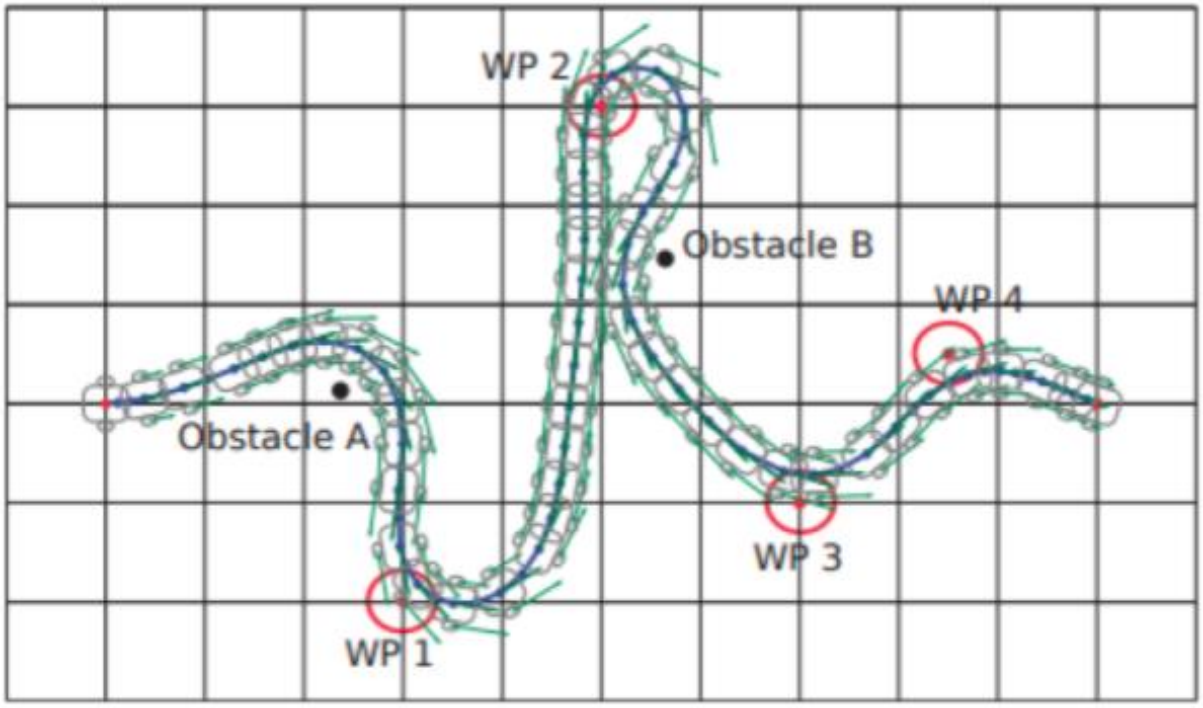


Рисунок 2.5 – Приклад роботи алгоритму TEB

Як не дивно, для локального пошуку шляху у реальному часі також розглядається використання методів машинного навчання. Проте, на практиці, використання машинного навчання для пошуку шляху рідко буває ефективним, оскільки узагальнення до якого приходить натренована нейронна мережа ймовірніше за все, стає аналогічним евристиці A*. Доречним є використання нейронних мереж для пошуку шляху у випадках, коли проектується рух механізмів з високим степенем свободи, через що керування класичними методами не є можливим. Прикладом такої проблеми може бути подолання шляху з перешкодами роботом-манекеном чи виконання складних маніпуляцій роботизованою кінцівкою. Однак, подібне навчання є доволі витратним зі сторони обчислень.

2.1.4 Методи кооперативного пошуку

Основною особливістю кооперативного пошуку шляху, що відрізняє його від класичної проблеми пошуку шляху, є можливість обміну інформацією між агентами. Задача, при цьому, ускладнюється можливістю створення

нескінченних циклів, коли агенти заважають руху один одного, та необхідністю забезпечення оптимального шляху для кожного з агентів.

Розглянемо алгоритми, що активно використовуються у вирішенні проблеми кооперативного пошуку руху.

Алгоритми локального відновлення (LRA*) полягає у використанні кожним агентом алгоритму A* без урахування інших агентів, окрім, безпосередньо, сусідніх до них. Агенти рухаються по своїм маршрутам доки не виникає можливість зіткнення. Коли така можливість виникає, агент, що намагається пересунутися у зайняту позицію, оновлює свій маршрут алгоритмом A*. Даний алгоритм є простим у реалізації, проте він має схильність до створення нескінченних циклів, що може бути частково вирішено доданням шуму до евристика алгоритму A*, проте працює не завжди і значною мірою погіршує ефективність пошуку оптимальних маршрутів для руху.

Кооперативний алгоритм A* (CA*) враховує заплановані шляхи кожного з агентів при пошуку шляху. Також до дій агенту додається можливість очікування. Після підрахунку маршруту він записуються в таблицю для бронювання, що репрезентує спільні знання агентів про маршрути один одного. Маршрути, при цьому, вираховуються не тільки у просторі, але й у часі. Найпростішою реалізацією таблиці для бронювання є представлення її у вигляді тривимірної сітки, що в свою чергу може бути представлена у вигляді хеш-таблиці. Важливо відмітити, що даний алгоритм, будучи жадібним, є неспроможним для вирішення деяких типів задач. (Рис. 2.6)

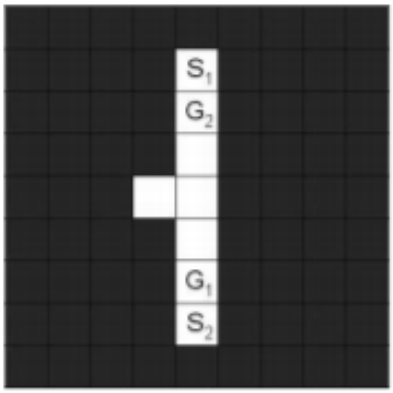


Рисунок 2.6 – Приклад задачі що не може бути вирішена у контексті жадібного пошуку.

Ієрархічний кооперативний алгоритм A* (HCA*) полягає у абстрагуванні задачі, до форми де ігнорується таблиця для бронювання та інші агенти, а пошук виконується у класичному двовимірному просторі. Оцінка шляху у даному випадку, виступає у якості допустимої та консистентної евристики. За своєю сутністю, алгоритм HCA* є варіацією алгоритму CA* проте з більш спеціалізованою евристикою.

Ієрархічний кооперативний алгоритм A* з вікном (WHCA*) є алгоритмом на основі абстрагування алгоритмом HCA*. Прикладом конфліктної ситуації, що не вирішують наведені алгоритми з таблицями для бронювання, є ситуація, коли агент перебуває на місці свого призначення, але заважає при цьому руху інших агентів. Також проблемами можуть бути впорядкування агентів та складність підрахунків шляху у тривимірному просторі. Ці проблеми можливо вирішити за допомогою введення вікна для підрахунків кооперативного пошуку, що вираховуватиме частковий шлях та періодично оновлюватиметься. При цьому вікно охоплює лише кооперативний аспект пошуку, у той час як абстрактний пошук HCA* виконується за усією мапою. [15]

Отже, розглянуті алгоритми можна класифікувати наступним чином (Рис. 2.7):

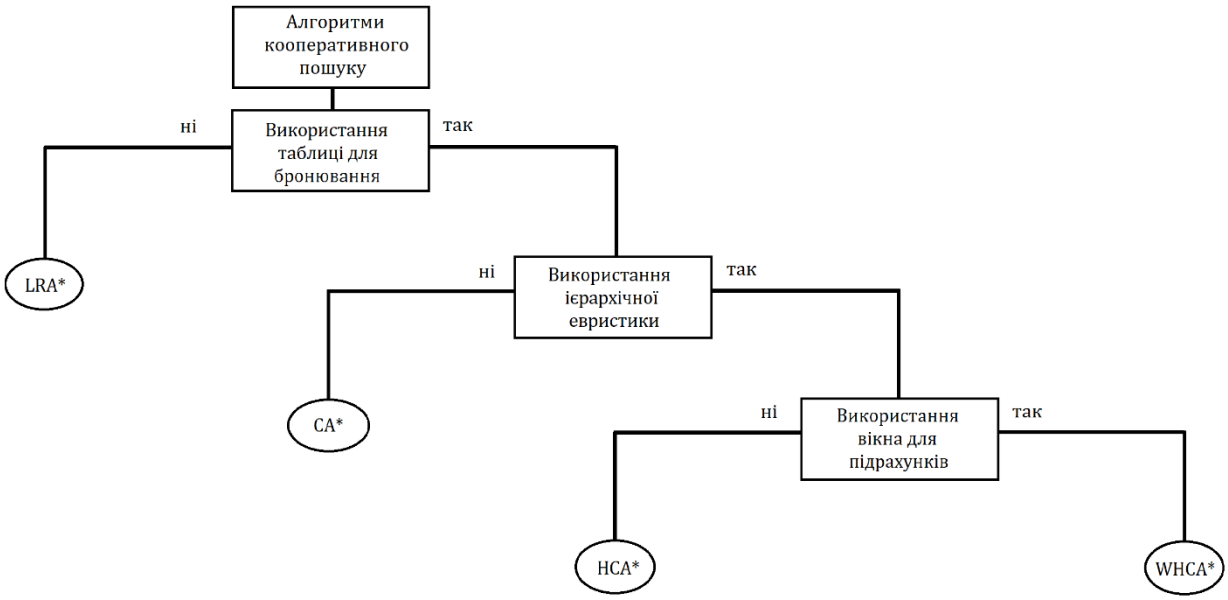


Рисунок 2.7 – Класифікація алгоритмів кооперативного пошуку

2.2 Критерії якості рішення задачі

Єдину метрику, що повністю описувала б поведінку агентів як інтелектуальну, підібрати складно. У рамках даної роботи розглядатимемо наступні метрики:

- Час, витрачений на виконання завдання усіма контрольованими агентами
- Сумарна довжина маршруту, що була подолана кожним з контрольованих агентів
- Наявність та кількість зіткнень, що відбулися як між агентами, так й між агентом і об’єктом

Також важливим для порівняння методів є урахування особливостей цих методів. Прикладами таких особливостей можуть бути необхідність у навчанні, або обмеження на клас задач, які надає можливість вирішити той чи

інший метод.

2.3 Алгоритм розв’язку задачі

Розглянувши основні підходи до вирішення поставленої задачі кооперативного пошуку шляху, та проаналізувавши їх особливості, маємо можливість формалізувати задачу, що буде реалізована у ході подальшої роботи.

По-перше, більш детально опишемо середовище, у якому проходитиме розробка програмного засобу. У якості цього середовища виступає система ROS, а саме її дистрибутив Noetic. Вибір дистрибутиву мотивований тим, що він є найбільш актуальним за датою випуску. Для тестування використовуватимемо інструмент RVIZ. Для моделювання світу, тестування, та демонстрації роботи програмного забезпечення використовуватимемо Gazebo.

Можемо розділити алгоритм розв’язку задачі створення програмного забезпечення для кооперативного пошуку шляху на наступні етапи:

1) Підготовча робота. Створення моделей роботів здатних до контрольованого руху та з наявними сенсорами. Створення мап для тестування або ж механізму для автоматичної генерації мап, та створення відповідних до них сіткових мап.
2) Реалізація класичних алгоритмів пошуку з використанням одного агенту. Задля тестування системи доречним є реалізація класичного алгоритму A*, з ігноруванням появи можливих динамічних перешкод на шляху.
3) Реалізація алгоритмів локального пошуку з доданням невідображених на сітковій мапі перешкод. Для оновлення мапи використовуватимемо дані з лазеру. Реалізація алгоритму TEB. Пошук та реалізація можливих альтернатив для корегування траєкторії.
4) Додання додаткового агенту та тестування системи. Оцінка частоти

зіткнень та конфліктних ситуацій при роботі агентів що керуються класичними алгоритмами пошуку руху.

5) Реалізація алгоритмів кооперативного пошуку руху. У якості можливих претендентів було обрано алгоритми LRA* та WHCA*, хоча реалізація додаткових алгоритмів не виключається. Пошук та реалізація можливих альтернатив чи покращень до зазначених алгоритмів. Збір метрик про роботу кожного з реалізованих алгоритмів.
6) Порівняння зібраних метрик. Детальний аналіз отриманих результатів з поясненням переваг та недоліків кожного з методів. Визначення напрямків для подальших досліджень у зазначеній області.

2.4 Висновки до Розділу 2

У даному розділі було описано та проаналізовано принципи роботи методів що активно вживаються при вирішенні поставленої задачі на сьогоднішній день.

Визначення методу розв’язку задачі пошуку руху містить наступні етапи:

1) Визначення чи є доступною мапа простору.
2) Визначення алгоритму глобального пошуку.
3) Визначення механізму локального пошуку.
4) За необхідності визначення механізму взаємодії між агентами.

Також було формалізовано алгоритм реалізації програмного продукту що вирішував би задачу кооперативного пошуку шляху.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСОБУ

3.1 Структура реалізованого проекту

У даній роботі було реалізовано програмний пакет ROS, що включає у собі модулі для тестування та аналізу поставленої задачі. Реалізація пакету була поділена на наступні етапи:

1) Створення моделі світу
2) Створення моделі роверу
3) Забезпечення руху
4) Створення сервісу побудови шляху

Розглянемо більш детально кожен із них.

3.1.1 Створення моделі світу

У якості середовища для тестування використовуємо Gazebo. Модель світу у середовищі Gazebo описується у файлі з розширенням .world за допомогою розмітки xml. Відповідний файл може включати у себе моделі, що використовуються у представленні світу. У нашому випадку, найважливішим при створення моделі світу є моделювання мапи. Також, у файлі .world, ми моделюємо освітлення для наочності при тестуванні.

Моделювання мапи відбувається шляхом підключення відповідної моделі карти висотності у файлі .world. Карта висотності являє собою sdf модель, що також описується за допомогою xml. У нашому випадку доречно створення декількох мап для тестування різноманітних ситуацій. Файл, що описує карту висот складається з фізичної та візуальної репрезентації. Фізична репрезентація потребує вхідного файлу, що візуально, у формі відтінків сірого, репрезентував би модельовану мапу. При виконанні даної роботи у якості подібного файлу використовуємо .pgm зображення. (Рис. 3.1)

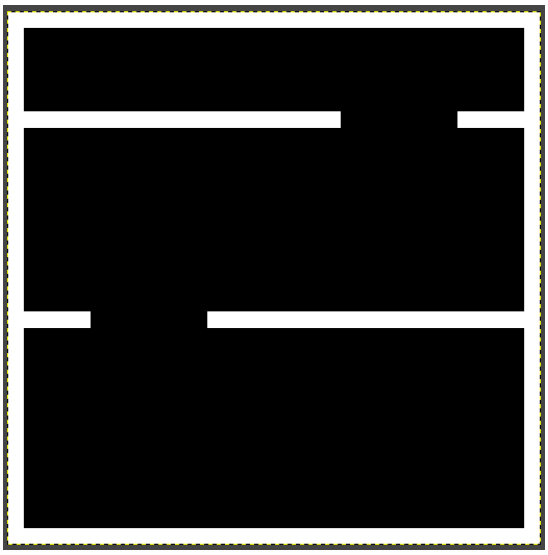


Рисунок 3.1 – Приклад .pgm зображення карти висот

Для візуального зображення карти висот використовується як файл з репрезентацією самої карти висот, так і файл текстури, що накладається на геометрію цієї карти. Gazebo має бібліотеку з доступними для роботи текстурами, проте для наочності, у даній роботі використовуватимемо власний файл текстури, у форматі зображення з розширенням .jpg. (Рис. 3.2)

Рисунок 3.2 – Приклад текстури

Також, для подальшої роботи алгоритмів даного проекту, необхідне представлення модельованої мапи у форматі, що сприймався би системою ROS. Для даного представлення використовується формат .yaml. У файлі формату .yaml ми зазначаємо наступні параметри мапи:

- image – шлях до вхідного файлу мапи, що у нашому випадку являє собою файл .pgm.

- resolution – масштаб мапи у відношенні метрів на піксель зображення.

- origin – початкова координата мапи

- occupied_thresh – поріг, після якого клітина зображення вважається

окупованою

- free_thresh – поріг, після якого клітина зображення вважається повністю вільною

- negate – параметр для кольоровою інварсії. Оскільки модель sdf сприймає темніші ділянки як глибші, а система сприймає темніші ділянки як перешкоди, у нашому випадку значення цього параметру дорівнює 1.

Дане представлення мапи надає нам можливість підняти сервіс, що містив би дані про мапу та метадані до неї, у файлі запуску пакету.

Отже, маємо наступне представлення середовища (Рис. 3.3):

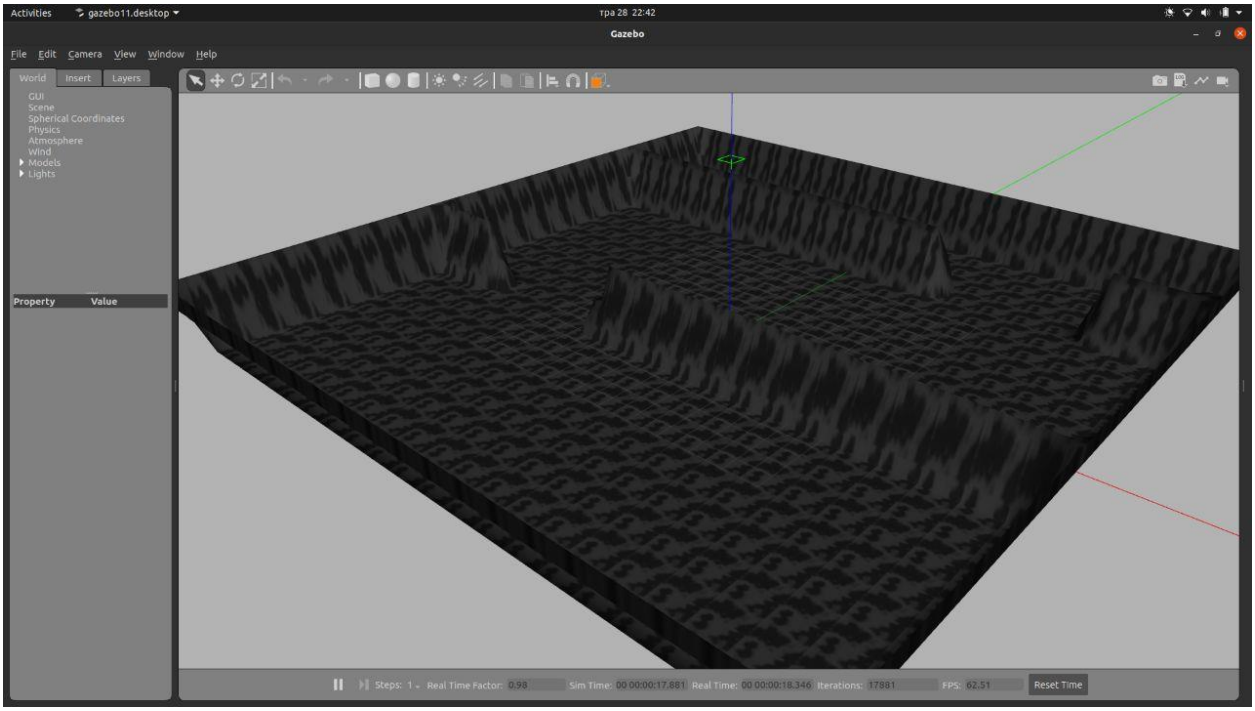


Рисунок 3.3 – Зображення середовища у Gazebo.

А також відповідний сервіс для мапи (Рис. 3.4):

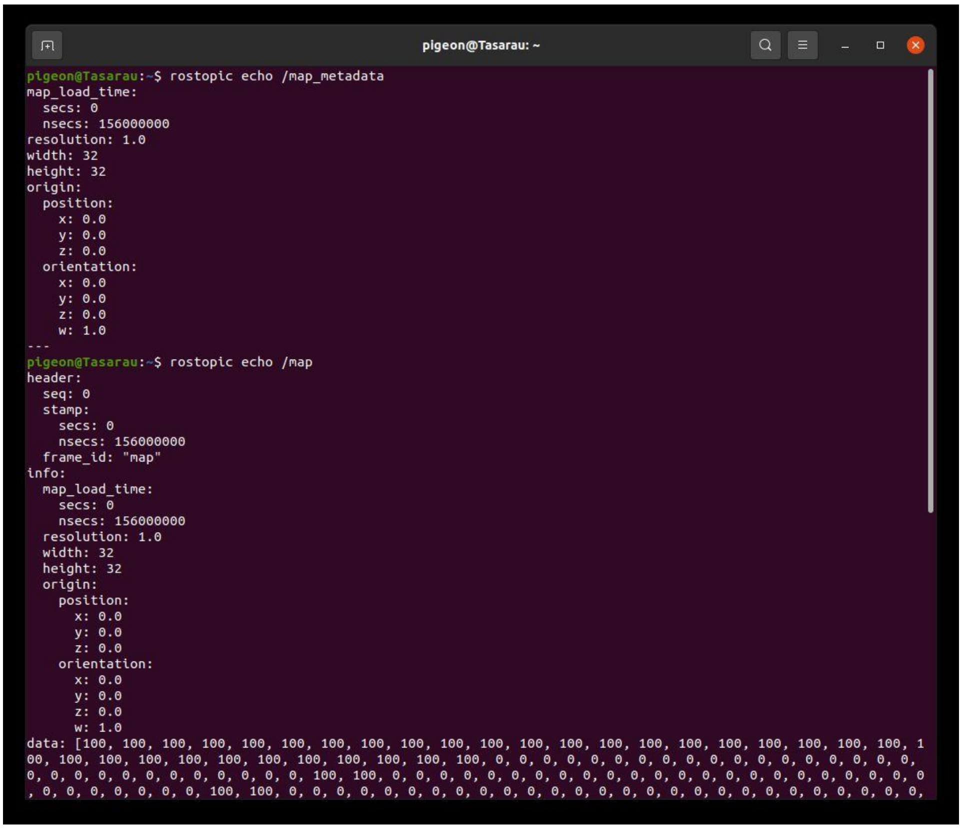


Рисунок 3.4 – Виклик сервісу мапи.

3.1.2 Створення моделі роверу

За визначенням ровер являє собою апарат, що призначений для руху по місцевості. Для роверів, як і для інших довільних робототехнічних моделей, у середовищі ROS використовують urdf моделі. Моделі urdf використовують для розмітки xml, та необхідні для опису робототехнічних апаратів з усіма супутніми модулями.

Найголовнішими елементами у файлі urdf є модулі link, що описують фізичні деталі з власною інерцією, властивостями та візуальним відображенням. У якості механізму поєднання link елементів використовуються елементи joint, що описують характер з’єднань елементів link між собою.

Таким чином, можемо описати найпростішу модель двухколісного роверу з лазерним сенсором та коліщатком для балансу. Можемо відобразити створену модель за допомогою Rviz (Рис. 3.5):

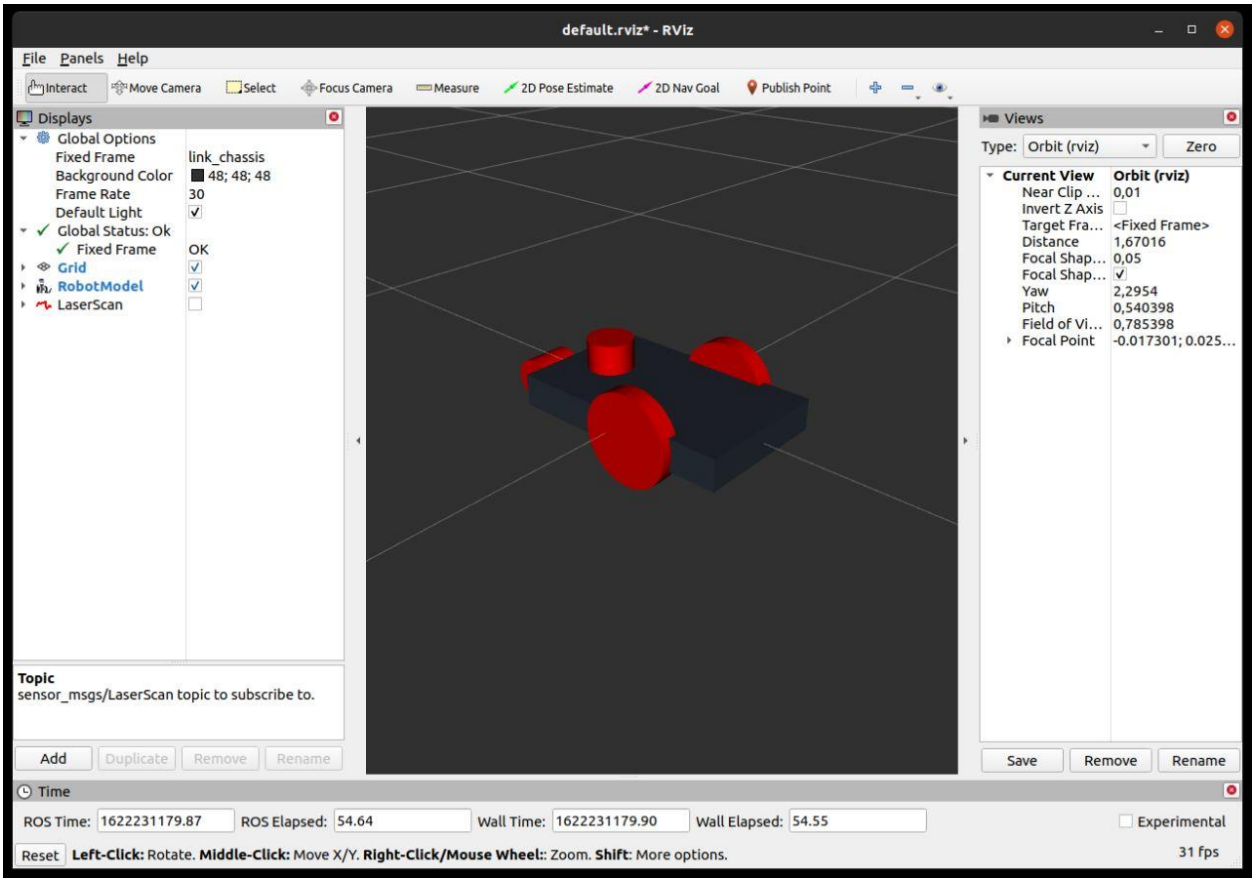


Рисунок 3.5 – Відображення моделі робота у Rviz

Rviz надає можливість відслідковувати детальний стан апарату та стан його елементів, а також інформацію, що апарат публікує за допомогою сенсорів.

Також, у моделюванні urdf файли можливе підключення окремих модулів .gazebo, що можуть використовуватись при забезпеченні руху апарату чи симуляції роботи лазеру. Так, для моделювання наших роверів, задля контролю руху ми використовуємо модуль differential_drive_controller, що публікує одометрію апарату та читає топіки з командами для руху коліс. Для моделюваня сенсору використовуємо gazebo_ros_head_hokuyo_controller, що моделює роботу лазеру.

Також, у моделюванні urdf файли можливе винесення окремих, повторюваних частин у якості макросів. Це важливо для нас, оскільки ми маємо багато повторюваних моделей для симуляції мульти-агентної задачі.

Отже результатом моделювання є модель роверу, що можемо помістити у наше середовище за допомогою файлу запуску пакету (Рис. 3.6).

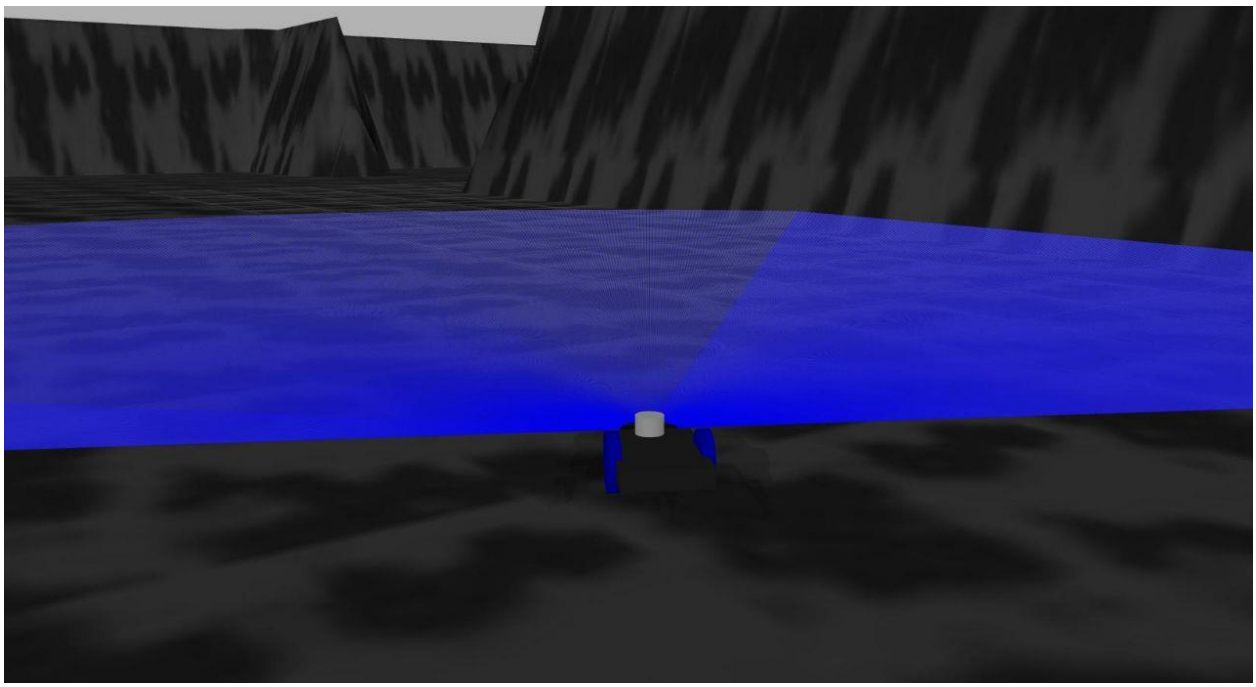


Рисунок 3.6 – Зображення роверу у середовищі

3.1.3 Забезпечення руху

Для забезпечення руху існують два основних підходи до розробки.

- Моделювання системи руху за допомогою ROS navigation stack, що включає у себе велику кількість модулів для полегшення симуляції руху апаратів. Даний варіант є очевидним, за необхідності швидкої реалізації певної задачі. Також, цей варіант можна вважати доцільно інтегрованим у систему середовища ROS, що може як значно спрощувати взаємодію з користувачем та іншими модулями, так і навпаки, викликати конфлікти з іншими модулями та обмежувати простір можливих дій з апаратом.

- Іншим підходом є створення оригінального коду для взаємодії з апаратом. Цей варіант надає більшої пластичності при розробці, проте є більш складним у реалізації.

У ході створення даного проекту більш доречним є використання

власного коду, оскільки не всі алгоритми що досліджуються мають реалізацію у ROS navigation stack. Також метою даної роботи є більш детальне ознайомлення зі структурою ROS, що також додає балів до реалізації саме власного коду.

Для реалізації коду використовуватимемо Python. Кожен файл з кодом являє собою окремий вузол, призначений для відокремленої задачі. Всі вузли піднімаються у файлі запуску пакету.

Для реалізації руху, створюємо файл mover.py, що відповідає за рух певного роверу. Для окремих роверів використовуємо окремі вузли зі схожим кодом, що у подальшому можливо оптимізувати за допомогою винесення роверів у окремий пакет чи створення загального вузла для роботи з усіма роверами.

Основною метою файлу mover.py є зчитування топіку одометрії апарату та публікації команд у відповідний топік апарату. Задля цього, у файлі mover.py використовуємо звернення до сервісу, що надає нам шлях до мети ровера, з урахуванням поточної одометрії робота. Отримавши відповідь від сервісу ми направляємо ровер по точкам наданого шляху. (Рис. 3.7)

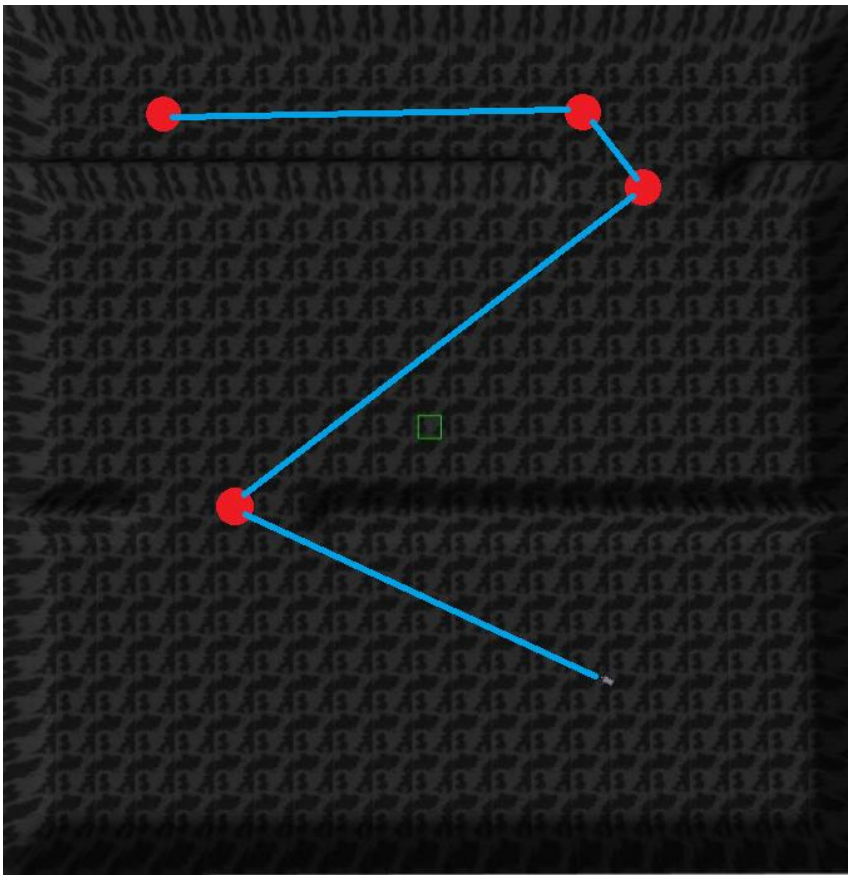


Рисунок 3.7 – Схематичне зображення шляху роверу. Червоним позначені опорні точки шляху, синім – маршрут роверу

3.1.4 Створення сервісу побудови шляху

Метою сервісу є створення шляху для апарату. Задля створення сервісу необхідно специфікувати формат звернень та відповідей даного сервісу. Це робиться за допомогою створення відповідного .srv файлу.

У нашому випадку, для звернення до сервісу використовується повідомлення, що складається з:

- Масиву що містить необроблену інформацію про карту середовища;
- Інформації про ширину мапи;
- Інформації про довжину мапи;
- Точку старту;
- Кінцеву точку мети.

Для відповіді використовується повідомлення, що складається з:

- Масиву з точок шляху.

Сервіс звертається до функцій, у яких реалізовані відповідні алгоритми пошуку шляху. Задля цього, він оброблює дані що подаються у запиті, у відповідний для роботи алгоритмів формат. Це перетворення включає у себе створення матриці мапи, на основі масиву з інформацією про карту середовища та інформації про ширину та довжину, а також перетворення точок у пари індексів, що відповідали би координатам на створеній матриці мапи.

Забезпечення кооперативної роботи апаратів можливе за допомогою створення окремих вузлів для мульти-агентної обробки руху, а також розширення інформації що надається сервісу.

3.1.5 Реалізація алгоритмів

Оскільки основою реалізації наших алгоритмів є алгоритм A*, розглянемо його реалізацію більш детально. Вхідними параметрами алгоритму є:

- costmap, матриця що описує вільні клітини середовища
- start_index, що відповідає парі стартових координат
- goal_index, що відповідає парі кінцевих координат
- resolution, що визначає відношення метрів на клітину

Опишемо роботу алгоритму. Спочатку, створимо пусті масиви з відкритими та закритими вузлами. Додаємо до масиву відкритих вузлів вузол з початковими координатами, нульовою вартістю шляху та нульовою оцінкою. Доки масив відкритих вузлів не пустий, ми обираємо з нього поточний вузол як вузол з найменшою оцінкою та переміщуємо його з масиву відкритих вузлів до масиву закритих. Якщо поточний вузол є кінцевим, закінчуємо роботу циклу. Якщо вузол не є кінцевим, розглядаємо його сусідів. У нашому випадку, сусідніми вважаємо 8 прилеглих клітинок, проте, клітинкам по

діагоналі надається вища оцінка. Вузол зберігається до масиву сусідів за умови що відповідна клітина у матриці середовища є вільною. Задля евристичної оцінки використовуємо евклідівську відстань. Отримавши масив сусідів ми перевіряємо наявність його вузлів у масивах відкритих та закритих вузлів. Якщо вузол є серед закритих – пропускаємо його. Якщо він є серед відкритих, однак з більшою оцінкою, оновлюємо його, разом із інформацією про попередній вузол. Якщо вузол відсутній у цих масивах, додаємо його до масиву відкритих вузлів.

Отримавши масив закритих вузлів що містить у собі кінцевий вузол, створюємо новий масив, у який почергово, починаючи з кінцевого вузла, додаємо координати попереднього до нього вузла, доки вузол з початковими координатами не буде досягнутий. Отриманий масив пар індексів повертаємо у якості результату роботи алгоритму.

Приклад результату роботи алгоритму (Рис. 3.8):

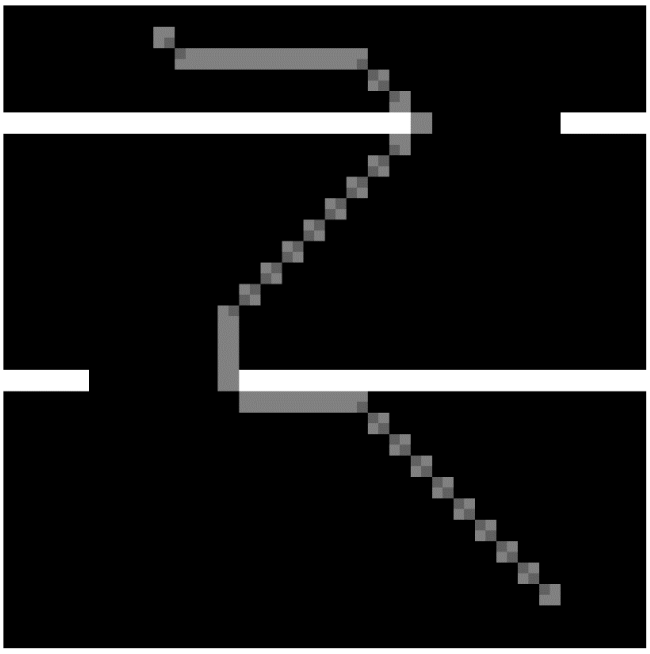


Рисунок 3.8 – Маршрут побудований алгоритмом A*

На основі даної реалізації розглянемо принципи роботи кооперативних алгоритмів пошуку.

Алгоритм LRA* реалізується за рахунок оновлення мапи у конфліктних ситуаціях. Це можливо за допомогою зчитування інформації з лазеру. Якщо

виявлено що певний об’єкт поблизу апарату заважає його руху, координати об’єкту подаються до сервісу, для чого додаємо додатковий параметр до структури вхідного повідомлення сервісу. Ми оновлюємо мапу позначивши відповідну клітину як зайняту, додавши до ней відмінну від 0 та 100 константу. Після цього ми подаємо оновлену мапу на сервіс побудови шляху і отримуємо новий маршрут для нашого апарату.

На відміну від попередніх алгоритмів, що будують шлях для кожного апарату окремо, алгоритми CA* та WHCA* потребують інформацію від декількох апаратів одночасно. Відповідно сервіс побудови шляху приймає масиви з початкових та цільових точок для кожного апарату і повертає відповідні маршрути.

Реалізація алгоритму CA* відрізняється від A* створенням таблиці бронювання. Кожен з апаратів почергово обирає найкращу клітину для руху на основі певної евристики. Відповідно збережені клітини позначаються як зайняті для руху наступних апаратів. HCA* відрізняється тим, що у якості евристики для оцінки клітин, використовується довжина шляху алгоритму A*.

Відповідно алгоритм WHCA* відрізняється тим, що таблиця бронювання використовується лише на певну кількість кроків, у той час як для подальшого шляху використовується A*. Варіантами реалізації є використання таблиці бронювання у випадках, коли координати двох апаратів є близькими один до одного, або у випадках, коли сервіс виявляє що шляхи декількох апаратів перетинаються, та час перетину співпадає. Складність відслідковування перетину маршрутів полягає у синхронізації роботи роверів, через що доречно розглядати інтервал часу. Також, якщо перетин відбувається не у точці, а протягом певної ділянки шляху, можливий варіант з очікуванням за межами цієї ділянки чи позначення її частини як зайняті.

3.2 Тестування

Для запуску проекту необхідне створення файлу запуску пакету, що має

включати у собі почерговий виклик змодельованого середовища Gazebo, підняття сервісу з інформацією про карту, виклик моделей роботів та відповідних вузлів з забезпеченням руху, а також підняття сервісу з пошуку руху.

Відповідне зображення проекту (Рис. 3.9):

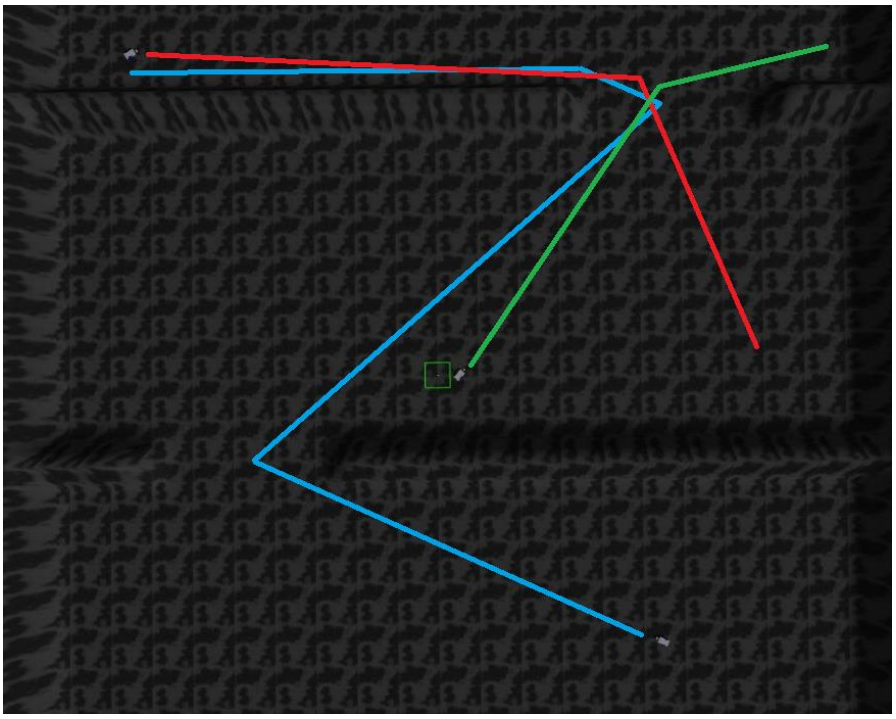


Рисунок 3.9 – Схематичне зображення одночасного руху декількох апаратів

3.3 Висновки до Розділу 3

У даному розділі було описано створення пакету у середовищі ROS, що забезпечує роботу симуляції з кооперативного пошуку руху. У ході даного розділу було згадано створення моделі світу, мапи, моделі апаратів, підключення сенсорів, забезпечення руху апаратів, створення сервісів.

У ході подальшої роботи доцільне порівняння результатів моделювання різних алгоритмів кооперативного пошуку шляху.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Постановка завдання

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для кооперативного пошуку руху. Даний продукт розроблений у середовищі ROS, та являє собою код написаний мовою Python та створений набір моделей для тестування. Програмний продукт придатний до використання у реальних робототехнічних засобах.

4.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує проблему кооперативного пошуку. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір середовища для моделювання;

F_2 – вибір мови програмування;

F_3 – вибір методу реалізації алгоритмів.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) середовище ROS; б) фізичне моделювання;

Функція F_2:

а) мова програмування C++; б) мова програмування Python;

Функція F_3:

а) ROS navigation stack; б) Оригінальна реалізація.

4.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

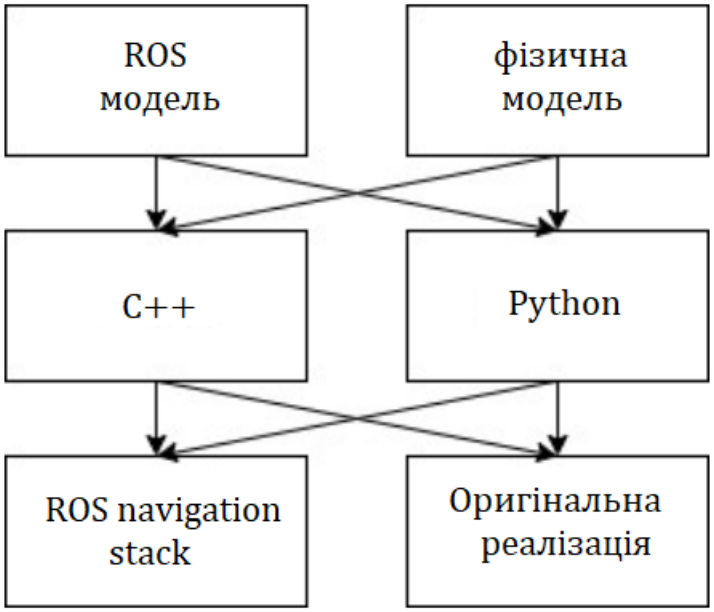


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F_1	a	Низька вартість розробки, наявність реалізованих методів, модульність	Необхідність вивчення середовища, помилки та неточності у середовищі

	б	Наближення до реального середовища	Значні витрати, відсутність пластичності моделей
F_2	*а*	Висока швидкодія	Висока вартість розробки
	б	Наявність великої кількості бібліотек, простіший програмний код	Низька швидкодія
F_3	*а*	Простота реалізації алгоритмів, підтримка середовищем	Можливість конфліктних ситуацій
	б	Висока пластичність у реалізації	Складність реалізації

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки метою даної роботи є тестування та аналіз алгоритмів, постає необхідність у пластичності реалізованих моделей. У поєднанні з нижчими витратами, використаємо варіант а) як єдиний можливий.

Функція *F2:*

Доцільно розглянути обидва варіанти.

Функція *F3:*

Оскільки метою даної роботи є аналіз алгоритмів, а тому важливим є пластичність реалізації та уникнення конфліктів, використаємо варіант б).

Таким чином, будемо розглядати такий варіант реалізації ПП:

1. F_1а – F_2а – F_3б

2. F_1а – F_2б – F_3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обґрунтування системи параметрів ПП

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – швидкодія мови програмування;
- *X2* – об’єм пам’яті для обчислень та збереження даних;
- *X3* – час навчання даних;
- *X4* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	10000	14000	19000
Об’єм пам’яті	X2	Мб	420	128	64
Час попередньої обробки даних	X3	мс	4	3	2
Потенційний об’єм програмного коду	X4	кількість рядків коду	4000	2500	1000

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

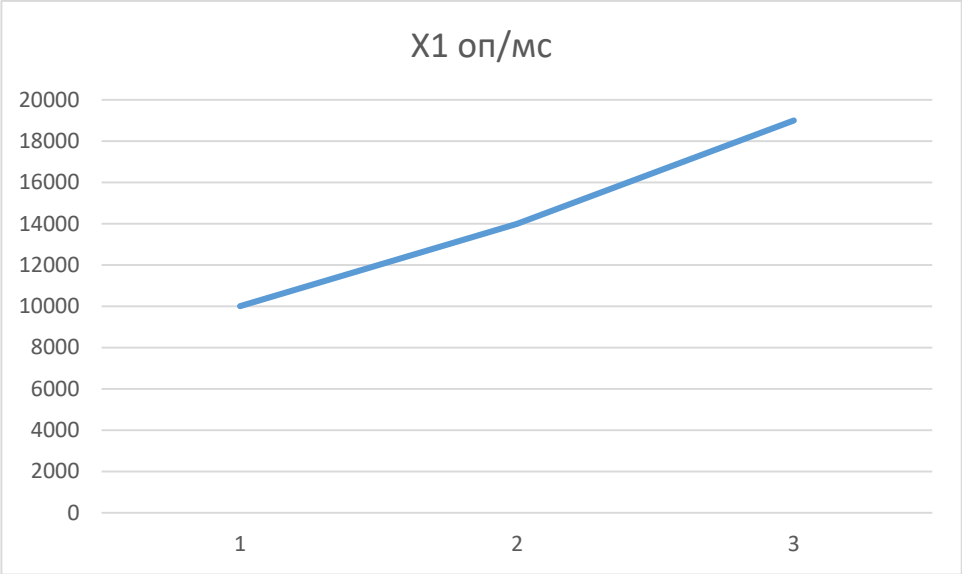


Рисунок 4.2 – X1, швидкодія мови програмування

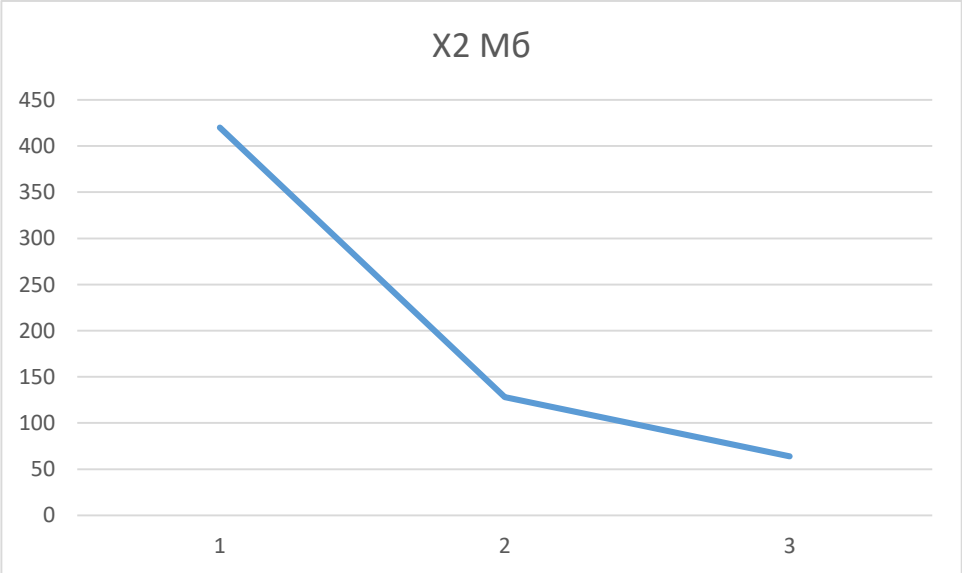


Рисунок 4.3 – X2, об’єм пам’яті

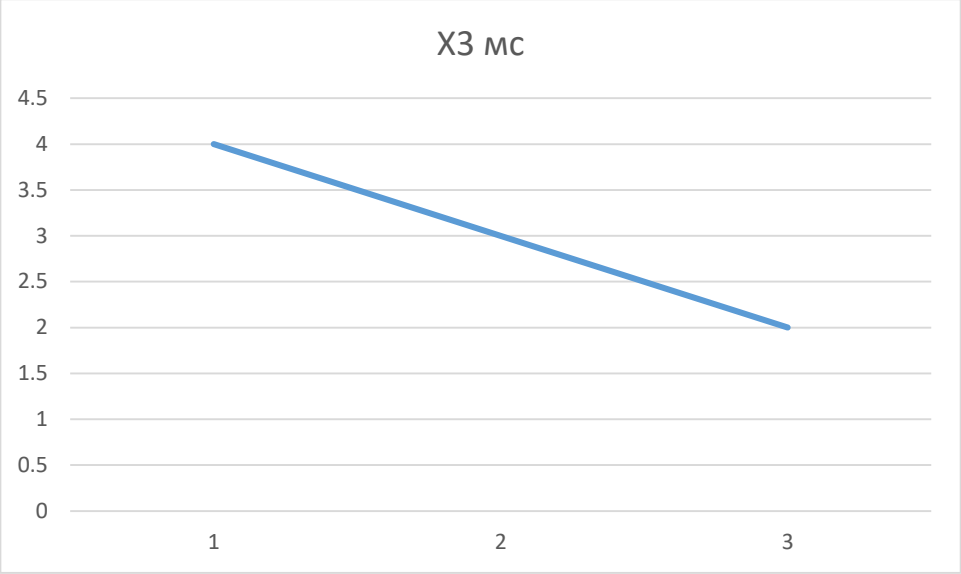


Рисунок 4.4 – X3, час попередньої обробки даних

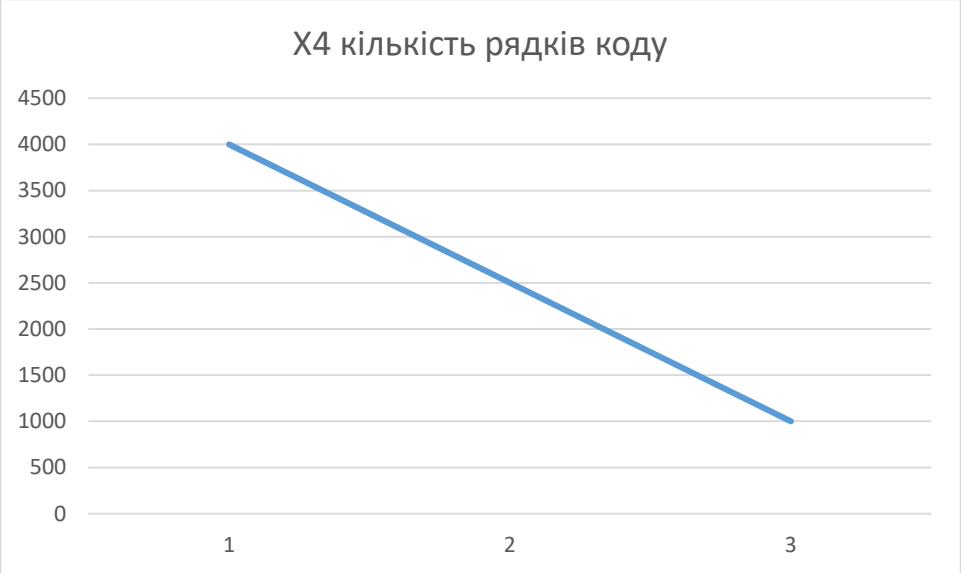


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні

параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	4	5	2	5	3	4	5	28	3,5	12,25
X2	Об’єм пам’яті	Mb	2	1	3	1	2	1	2	12	-12,5	156,25
X3	Час попередньої обробки даних	мс	5	3	5	5	4	5	3	30	5,5	30,25
X4	Потенційний об’єм	Кількість рядків коду	3	5	4	3	5	4	4	28	3,5	12,25

	програмного коду											
	Разом		14	14	14	14	14	14	14	98	0	211

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 98, \quad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24{,}5 \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 211. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 211}{7^2(4^3 - 4)} = 0{,}86 > W_k = 0{,}67. \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.5.

Таблиця 4.4 – Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	<	<	>	>	>	>	1,5
X1 і X3	<	>	<	=	<	<	>	<	0,5
X1 і X4	>	>	<	=	<	=	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	>	<	>	>	<	>	<	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

.Як видно з таблиці 4.6, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметриx_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	1,5	0,5	1,5	4,5	0,36	17,75	0,25	73,38	0,25
X2	0,5	1,0	1,5	0,5	3,5	0,28	15,75	0,22	65,63	0,23
X3	1,5	1,5	1,0	1,5	5,5	0,44	22,75	0,33	93,6	0,32
X4	0,5	1,5	0,5	1,0	3,5	0,28	13,75	0,2	57,63	0,2
Всього:					12,5	1	70	1	290,25	1

4.4 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об’єм пам’яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об’єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	10000	4	0,3	1,2
F2	А	X2	64	3	0,22	0,66
	Б	X2	128	8	0,23	1,84
F3	А	X3	1000	2	0,25	0,5

За даними з таблиці 5.7 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \qquad (5.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{Kl} = 1{,}2 + 0{,}66 + 0{,}5 = 2{,}36,$$

$$K_{K2} = 1{,}2 + 1{,}84 + 0{,}5 = 3{,}54.$$

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.5 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як

$$\mathrm{T_O} = \mathrm{T_P} \cdot \mathrm{K_П} \cdot \mathrm{K_{СК}} \cdot \mathrm{K_М} \cdot \mathrm{K_{СТ}} \cdot \mathrm{K_{СТ.М}}, \qquad (4.13)$$

де $\mathrm{T_P}$ – трудомісткість розробки ПП;

$\mathrm{K_П}$ – поправочний коефіцієнт;

$\mathrm{K_{СК}}$ – коефіцієнт на складність вхідної інформації;

$\mathrm{K_М}$ – коефіцієнт рівня мови програмування;

$\mathrm{K_{СТ}}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$\mathrm{K_{СТ.М}}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто T_P =27 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 = 1328{,}64 \text{ людино-годин.}$$

$$T_{II} = (122.4 + 19.44 + 6.91 + 19.44) \cdot 8 = 1345.52 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 16000 грн., один аналітик в області даних з окладом 17000. Визначимо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} грн., \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{16000 + 16000 + 17000}{3 \cdot 21 \cdot 8} = 97{,}22 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д, \quad (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 97.22 \cdot 1328.64 \cdot 1.2 = 155004{,}46$ грн.

II. $С_{ЗП} = 97.22 \cdot 1345.52 \cdot 1.2 = 156973{,}75$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 155004{,}46 \cdot 0.22 = 34100{,}98$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 156973,75 \cdot 0.22 = 34534,23$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 16000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 16000 \cdot 0,2 = 38400 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 38400 \cdot (1 + 0.2) = 46080 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 46080 \cdot 0,22 = 10137,6 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 30% та вартості ЕОМ – 32000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.3 \cdot 32000 = 11040 \text{ грн.,}$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 32000 \cdot 0.05 = 1840 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.9 =$$

$$= 1677.6 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1677.6 \cdot 0,3 \cdot 0.69 \cdot 3,51547 = 1220,79 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 32000 \cdot 0,67 = 21440 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_{А} + С_{Р} + С_{ЕЛ} + С_{Н}, \quad (4.17)$$

$С_{ЕКС}$ = 46080 + 10137,6 + 11040 + 1840 + 1220,79 + 21440 = 91758,39 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 91758,39 /1677.6 = 54,7 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_{М} = С_{М-Г} \cdot Т, \quad (4.18)$$

I. $С_{М}$ = 54,7 · 1328,64 = 72676,61 грн.

II. $С_{М}$ = 54,7 · 1345.52 = 73599,94 грн.

Накладні витрати складають 67% від заробітної плати:

$$С_{Н} = С_{ЗП} \cdot 0,67, \quad (4.19)$$

I. $С_{Н}$ = 155004,46 · 0,67 = 103852,99 грн.

II. $С_{Н}$ = 156973,75 · 0,67 = 105172,41 грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_{М} + С_{Н}, \quad (4.20)$$

I. $С_{ПП}$ = 155004,46 + 34100,98 + 72676,61 + 103852,99 = 365635,04 грн.

II. $С_{ПП}$ = 156973,75 + 34534,23 + 73599,94 + 105172,41 = 370280,33 грн.

4.6 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{Кj} / С_{Фj}, \quad (5.21)$$

$$К_{ТЕР1} = 2,36/ 365635,04 = 6,45 \cdot 10^{-6},$$

$$К_{ТЕР2} = 3,54/ 370280,33 = 9,56 \cdot 10^{-6}.$$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 9,56 \cdot 10^{-6}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 9,56 \cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – Python;

– Використання моделей з великою ємністю
– Використання стандартного інтерфейсу візуалізації, швидкість розробки

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

4.7 Висновки до розділу 4

Проведено повний функціонально-вартісний аналіз програмного продукту. Визначено та проведено оцінку основних функцій програмного продукту. Визначено параметри, які характеризують програмний продукт. Проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій.

Проведено економічний аналіз варіантів розробки – трудомісткість, витрати на заробітну плату та інші витрати.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У даній дипломній роботі проаналізовано існуючі методи пошуку шляху та імплементовано програмний пакет середовища ROS, що забезпечує пошук шляху у мульти-агентному середовищі.

У першому розділі було розглянуто актуальні сфери використання алгоритмів пошуку шляху. Було описано існуючі методи вирішення задачі пошуку шляху, а також історія цих методів. Також було детально описано історію, структуру та принципи середовища робототехнічної розробки ROS. Крім того була формалізована задача подальшого дослідження.

У другому розділі було проаналізовано існуючі методи пошуку шляху. Було описано етапи:

- побудови мапи середовища;
- глобального пошуку шляху;
- локального пошуку шляху;
- кооперативного пошуку.

Також було класифіковано існуючі алгоритми кооперативного пошуку та визначення задачі для реалізації у програмному пакеті.

У третьому розділі було описано послідовність розробки програмного пакету. Було описано розробку моделі середовища, моделі апарату, механізму руху, сервісу пошуку шляху та функцій алгоритмів пошуку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pozniak-Koszalka I., Koszalka L., Kasprzak A. Comparative Analysis of the Algorithms for Pathfinding in GPS Systems. 2015
2. Litman T. Autonomous Vehicle Implementation Predictions. 2021. URL: https://www.vtpi.org/avip.pdf
3. Doxel AI - Artificial Intelligence for Construction Productivity. Doxel AI. URL: https://www.doxel.ai
4. Markets R. A. Global Gaming Market (2021 to 2026) - Industry Trends, Share, Size, Growth, Opportunity and Forecasts. GlobeNewswire News Room. URL: https://www.globenewswire.com/news-release/2021/03/01/2184028/0/en/Global-Gaming-Market-2021-to-2026-Industry-Trends-Share-Size-Growth-Opportunity-and-Forecasts.html
5. Abd A. Z., Sunar M. S., Kolivand H. A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games. International Journal of Computer Games Technology. 2015. URL: https://doi.org/10.1155/2015/736138
6. Pan T., PUN-Cheng S. C. A Discussion on the Evolution of the Pathfinding Algorithms. 2020
7. Wizards of ROS: Willow Garage and the Making of the Robot Operating System. IEEE Spectrum. URL: https://spectrum.ieee.org/automaton/robotics/robotics-software/wizards-of-ros-willow-garage-and-the-making-of-the-robot-operating-system
8. Documentation - ROS Wiki. URL: http://wiki.ros.org/ROS/Concepts
9. HEREDIA S. N. Exploration Strategies for Robotic Vacuum Cleaners. 2018. URL: http://www.diva-portal.org/smash/get/diva2:1295924/FULLTEXT01.pdf
10. Carsten J., Rankin A., Ferguson D., Stentz A. Global Path Planning on Board the Mars Exploration Rovers. 2007. URL: http://www.cs.ait.ac.th/~mdailey/cvreadings/Carsten-MarsPath.pdf
11. Laparra D. M. PATHFINDING ALGORITHMS IN GRAPHS AND

APPLICATIONS. 2019. URL: http://diposit.ub.edu/dspace/bitstream/2445/140466/1/memoria.pdf

12. Xiao C., Hao S. A*-based Pathfinding in Modern Computer Games. 2010. URL: https://www.researchgate.net/publication/267809499_A-based_Pathfinding_in_Modern_Computer_Games
13. Martin-Plaza P., Beltran J., Hussein A., Musleh B., Martin D., Escalera A., Armingol J. M. Stereo Vision-based Local Occupancy Grid Map for Autonomous Navigation in ROS. 2016. URL: https://www.researchgate.net/profile/Ahmed-Hussein-146/publication/301125858_Stereo_Vision-based_Local_Occupancy_Grid_Map_for_Autonomous_Navigation_in_ROS/links/5752a15b08ae10d93370f02c/Stereo-Vision-based-Local-Occupancy-Grid-Map-for-Autonomous-Navigation-in-ROS.pdf
14. Marin-Plaza P., Hussein A., Martin D., Escalera A. Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles. 2018. URL: https://www.hindawi.com/journals/jat/2018/6392697/
15. Silver D. Cooperative Pathfinding. 2005. URL: https://www.aaai.org/Papers/AIIDE/2005/AIIDE05-020.pdf

Додаток А Презентація

Кооперативний пошук шляху у середовищі ROS

Баськов Владислав Сергійович, КА-75
Керівник: Данилов В. Я.

Мета та цілі

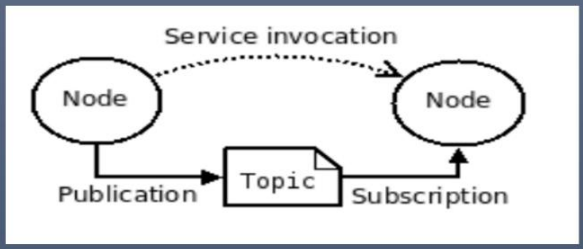
- Кооперативним пошуком шляху називають задачу мультиагентного пошуку шляху, де кожен з агентів повинен знайти маршрут до мети, що не конфліктує з маршрутами інших агентів, враховуючи інформацію про них.
- Метою даної роботи є огляд та аналіз існуючих методів кооперативного пошуку шляху, експериментальне моделювання пошуку шляху у середовищі для робототехнічної розробки ROS

Актуальність

- Пошук маршруту на мапі
- Автономний транспорт
- Робототехніка у індустрії
- Дослідницькі цілі
- Комп'ютерні симуляції та ігри
- Групова робототехніка

Середовище ROS

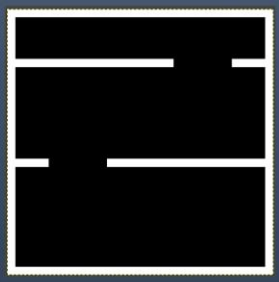
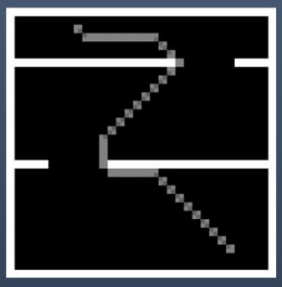
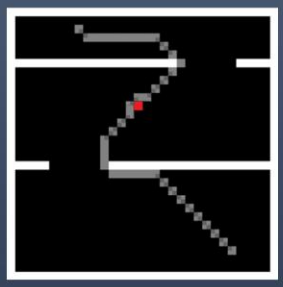
- Вузли
- Майстер та сервер параметрів
- Повідомлення
- Топіки
- Сервіси
- Пакети



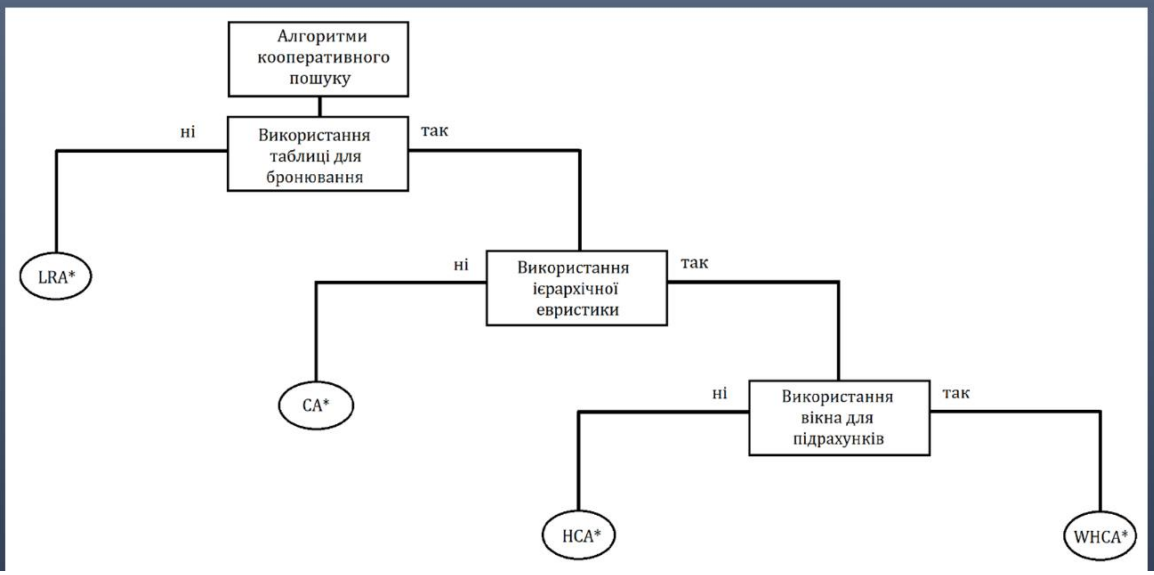
- Robot Operating System (ROS) – це відкрите програмне забезпечення для роботи з робототехнікою, що являє собою набір програмних фреймворків та сервісів для полегшення проектування, розробки та тестування робототехнічних проектів.

Етапи вирішення задачі пошуку руху

- Побудова чи підключення мапи простору
- Глобальний пошук шляху
- Локальний пошук шляху
- Визначення механізму взаємодії між агентами

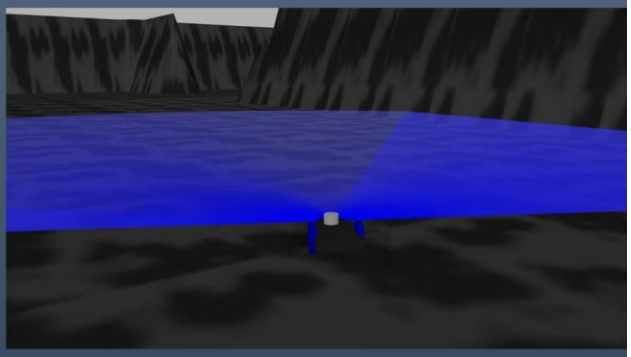




Алгоритми кооперативного пошуку

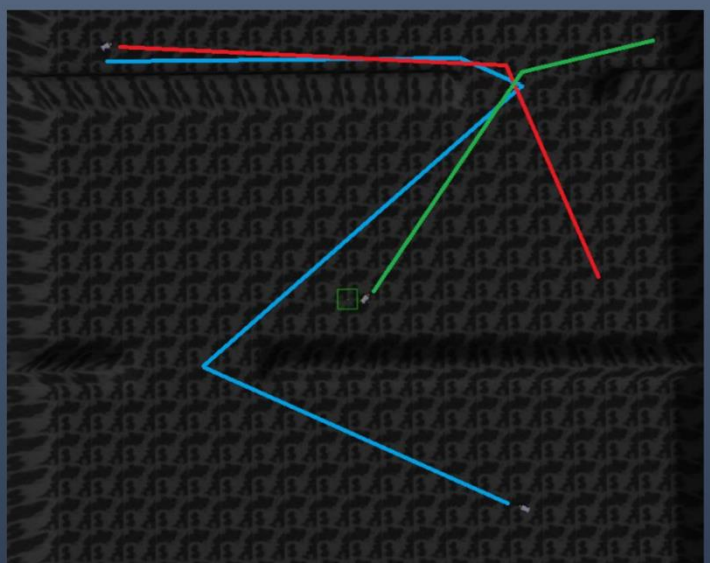


Реалізація програмного пакету

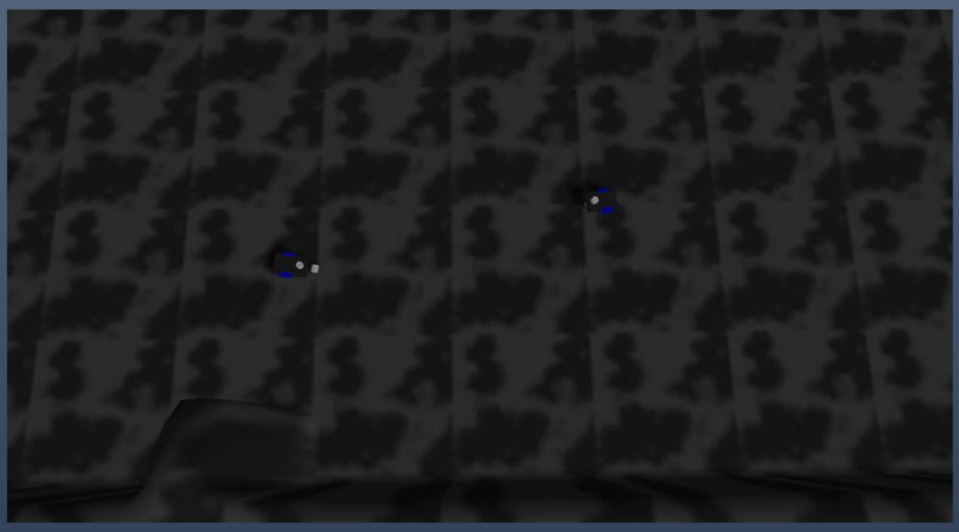
- Створення моделі світу
- Створення моделі апарату
- Забезпечення руху
- Створення сервісу побудови шляху
- Реалізація методів кооперативного пошуку



Результати роботи програмного пакету



Результати роботи програмного пакету



Висновки

- Досліджено існуючі алгоритми кооперативного пошуку шляху.
- Реалізовано програмний продукт для моделювання кооперативного пошуку шляху у мульти-агентному середовищі.

Дякую за увагу

Додаток Б Лістинг програми

```
// launch/demo.launch
<?xml version="1.0" encoding="UTF-8" ?>
<launch>
  <arg name="gui" default="true"/>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="debug" default="false"/>
  <arg name="world_name" default="$(find test_world)/worlds/test_world.world"/>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="debug" value="$(arg debug)" />
    <arg name="world_name" default="$(arg world_name)"/>
  </include>
  <!-- Run the map server -->
  <arg name="map_file" default="$(find test_world)/models/map1/materials/map1.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />
  <!-- robot -->
 <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find test_world)/urdf/robot1/robot.urdf'" />
  <arg name="x" default="10"/>
  <arg name="y" default="-13"/>
  <arg name="z" default="1.5"/>
  <arg name="yaw" default="-3"/>
  <node name="my_robot_spawn" pkg="gazebo_ros" type="spawn_model" output="screen"
        args="-urdf -param robot_description -model my_robot -x $(arg x) -y $(arg y) -z $(arg z) -Y $(arg yaw)" />
  <!-- robot2 -->
```

```
    <param name="robot_description_2" command="$(find xacro)/xacro --inorder '$(find test_world)/urdf/robot2/robot.urdf'" />
    <arg name="x2" default="2"/>
    <arg name="y2" default="2"/>
    <arg name="z2" default="1.5"/>
    <arg name="yaw2" default="0.5"/>
    <node name="my_robot_spawn2" pkg="gazebo_ros" type="spawn_model" output="screen"
        args="-urdf -param robot_description_2 -model my_robot_2 -x $(arg x2) -y $(arg y2) -z $(arg z2) -Y $(arg yaw2)" />
    <node pkg="test_world" type="mover_c1.py" name="mover_c1" output="screen"/>
    <!-- <node pkg="test_world" type="mover.py" name="mover" output="screen"/> -->
    <!-- <node pkg="test_world" type="mover2.py" name="mover2" output="screen"/> -->
    <!-- Start service server that responds with a plan for global path planning -->
    <node pkg="test_world" name="path_plan_server" type="PathServer.py" output="screen"/>
    <node pkg="test_world" name="path_plan_server2" type="PathServer_c1.py" output="screen"/>
  </launch>
  // models/map1/ model.config
  <?xml version="1.0"?>
  <model>
    <name>testmap_1</name>
    <version>1.0</version>
    <sdf version="1.4">model.sdf</sdf>
    <description>
      test map for course project
    </description>
  </model>
  // models/map1/ model.sdf
  <?xml version="1.0" ?>
  <sdf version="1.4">
    <model name="testmap_1">
      <static>true</static>
      <link name="link">
```

```
    <collision name="collision">
      <geometry>
        <heightmap>
          <uri>model://map1/materials/map1.png</uri>
          <size>33 33 4</size>          <pos>0 0 0</pos>
        </heightmap>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <heightmap>
          <texture>
            <diffuse>file://map1/materials/textures/texture1.jpg</diffuse>
            <normal>file://media/materials/textures/flat_normal.png</normal>
            <size>2</size>
          </texture>
          <uri>model://map1/materials/map1.png</uri>
          <size>33 33 4</size>          <pos>0 0 0</pos>        </heightmap>
      </geometry>
    </visual>
  </link>
 </model>
</sdf>
// models/map1/materials/ map1.yaml
image: map1.png
resolution: 1
origin: [0.0, 0.0, 0.0]
occupied_thresh: 0.65
free_thresh: 0.2
negate: 1     # black & white
// msg/PathArray.msg
geometry_msgs/Point[] path
// scripts/ PathServer.py
#! /usr/bin/env python3
import rospy
```

```
from test_world.srv import *
from geometry_msgs.msg import Point
from A_star import *
def tr_x_to(x, dist):
    return int(round(x + dist/2 - 0.5))
def tr_y_to(y, dist):
    return int(round(-y + dist/2 - 0.5))
def tr_x_from(x, dist):
    return x - dist/2 + 0.5
def tr_y_from(y, dist):
    return -y + dist/2 - 0.5
def handle_path_plan(req):
    rospy.loginfo("handle_path_plan +++++++++++++++++")
    map_matr_tmp = [req.costmap_ros[i:i+req.width] for i in range(0, len(req.costmap_ros), req.width)]
    map_matr = []
    for i in range(len(map_matr_tmp)):
        map_matr.append(map_matr_tmp.pop())
    rospy.loginfo(map_matr)
    start_index = []
    start_index.append(tr_x_to(req.start.x, req.width))
    start_index.append(tr_y_to(req.start.y, req.height))
    rospy.loginfo(start_index)
    goal_index = []
    goal_index.append(tr_x_to(req.goal.x, req.width))
    goal_index.append(tr_y_to(req.goal.y, req.height))
    rospy.loginfo(goal_index)
    rospy.loginfo("handle_path_plan +++++++++++++++++")
    # side of each grid map square in meters
    resolution = 1
    # time statistics
    start_time = rospy.Time.now()
    # how TODO LRA* and WHCA* ?
    tmp_path = A_star(map_matr, start_index, goal_index, resolution)
    goal_point = req.goal
```

```
                path = []
                path.append(goal_point)
                for p_point in tmp_path:
                        tmp_desired_position = Point()
                        tmp_desired_position.x = tr_x_from(p_point[0], req.width)
                        tmp_desired_position.y = tr_y_from(p_point[1], req.height)
                        tmp_desired_position.z = 0
                        path.append(tmp_desired_position)
                if not path:
                        rospy.logwarn("No path returned by algorithm")
                        path = []
                else:
                        execution_time = rospy.Time.now() - start_time
                        print("\n")
                        rospy.loginfo('++++++++++++++++++ Metrics ++++++++++++++++++')
                        rospy.loginfo('Total execution time: %s seconds',
str(execution_time.to_sec()))
                rospy.loginfo('++++++++++++++++++++++++++++++++++++++++++++++++')
                        print("\n")
                resp = PathPlanResponse()
                resp.plan = path
                return resp
        def path_plan_server():
                rospy.init_node('path_plan_server')
                s = rospy.Service('path_plan', PathPlan, handle_path_plan)
                rospy.spin()
        if __name__ == "__main__":
                path_plan_server()
        // scripts/ A_star.py
        #! /usr/bin/env python3
        import rospy
        D2 = 1.41421
        D = 1
        def get_eur(node, goal):
                dx = abs(node[0] - goal[0])
```

```
        dy = abs(node[1]- goal[1])
        return D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)
# so costmap[j][i] or [row][column]
# child.g = currentNode.g + distance between child and current
# child.h = distance from child to end
# child.f = child.g + child.h
def generate_n(costmap, node_ind, node_g, goal_ind):
        n_list = []
        tmp_ind = [0, 0]
        tmp_ind[0] = node_ind[0]+1
        tmp_ind[1] = node_ind[1]
        if 0 <= tmp_ind[0] < len(costmap):
                if costmap[tmp_ind[1]][tmp_ind[0]] == 0:
                        h = get_eur(tmp_ind, goal_ind)
                        g = node_g+D
                        n_list.append([[tmp_ind[0], tmp_ind[1]], g+h, g, node_ind])
        tmp_ind[0] = node_ind[0]-1
        tmp_ind[1] = node_ind[1]
        if 0 <= tmp_ind[0] < len(costmap):
                if costmap[tmp_ind[1]][tmp_ind[0]] == 0:
                        h = get_eur(tmp_ind, goal_ind)
                        g = node_g+D
                        n_list.append([[tmp_ind[0], tmp_ind[1]], g+h, g, node_ind])
        tmp_ind[0] = node_ind[0]
        tmp_ind[1] = node_ind[1]+1
        if 0 <= tmp_ind[1] < len(costmap[0]):
                if costmap[tmp_ind[1]][tmp_ind[0]] == 0:
                        h = get_eur(tmp_ind, goal_ind)
                        g = node_g+D
                        n_list.append([[tmp_ind[0], tmp_ind[1]], g+h, g, node_ind])
        tmp_ind[0] = node_ind[0]
        tmp_ind[1] = node_ind[1]-1
        if 0 <= tmp_ind[1] < len(costmap[0]):
                if costmap[tmp_ind[1]][tmp_ind[0]] == 0:
                        h = get_eur(tmp_ind, goal_ind)
```

```
                g = node_g+D
                n_list.append([[tmp_ind[0], tmp_ind[1]], g+h, g, node_ind])
        tmp_ind[0] = node_ind[0]+1
        tmp_ind[1] = node_ind[1]+1
        if 0 <= tmp_ind[0] < len(costmap) and 0 <= tmp_ind[1] < len(costmap[0]):
            if costmap[tmp_ind[1]][tmp_ind[0]] == 0:
                h = get_eur(tmp_ind, goal_ind)
                g = node_g+D2
                n_list.append([[tmp_ind[0], tmp_ind[1]], g+h, g, node_ind])
        tmp_ind[0] = node_ind[0]+1
        tmp_ind[1] = node_ind[1]-1
        if 0 <= tmp_ind[0] < len(costmap) and 0 <= tmp_ind[1] < len(costmap[0]):
            if costmap[tmp_ind[1]][tmp_ind[0]] == 0:
                h = get_eur(tmp_ind, goal_ind)
                g = node_g+D2
                n_list.append([[tmp_ind[0], tmp_ind[1]], g+h, g, node_ind])
        tmp_ind[0] = node_ind[0]-1
        tmp_ind[1] = node_ind[1]-1
        if 0 <= tmp_ind[0] < len(costmap) and 0 <= tmp_ind[1] < len(costmap[0]):
            if costmap[tmp_ind[1]][tmp_ind[0]] == 0:
                h = get_eur(tmp_ind, goal_ind)
                g = node_g+D2
                n_list.append([[tmp_ind[0], tmp_ind[1]], g+h, g, node_ind])
        tmp_ind[0] = node_ind[0]-1
        tmp_ind[1] = node_ind[1]+1
        if 0 <= tmp_ind[0] < len(costmap) and 0 <= tmp_ind[1] < len(costmap[0]):
            if costmap[tmp_ind[1]][tmp_ind[0]] == 0:
                h = get_eur(tmp_ind, goal_ind)
                g = node_g+D2
                n_list.append([[tmp_ind[0], tmp_ind[1]], g+h, g, node_ind])
        return n_list
    def A_star(costmap, start_index, goal_index, resolution):
        D = resolution
        D2 = D*1.41421
        path = []
```

```
    openList = []
    closedList = []
    openList.append([start_index, 0, 0, []])
    while openList:
        ind = 0
        for i in range(1, len(openList)):
            if openList[i][1] < openList[ind][1]:
                ind = i
        currentNode = openList.pop(ind)
        closedList.append(currentNode)
        if currentNode[0]==goal_index:
            break
        neighbor_list = generate_n(costmap, currentNode[0], currentNode[2],
goal_index)
        for n_node in neighbor_list:
            if not any((c_node[0] == n_node[0] for c_node in closedList)):
                for o_node in openList:
                    if o_node[0]==n_node[0]:
                        if o_node[2]>n_node[2]:
                            # print(n_node[0])
                            o_node[2] = n_node[2]
                            o_node[1] = n_node[1]
                            o_node[3] = n_node[3]
                if not any((o_node[0] == n_node[0] for o_node in openList)):
                    openList.append(n_node)
    tmp = goal_index
    while tmp!=start_index:
        # print(tmp)
        for node in closedList:
            if node[0]==tmp:
                tmp=node[3]
                if tmp!=start_index:
                    path.append([tmp[0], tmp[1]])
                break
    return path
```

```
// scripts/PathServer_c1.py
#! /usr/bin/env python3
import rospy
from test_world.srv import *
from test_world.msg import *
from geometry_msgs.msg import Point
from A_star import *
def tr_x_to(x, dist):
    return int(round(x + dist/2 - 0.5))
def tr_y_to(y, dist):
    return int(round(-y + dist/2 - 0.5))
def tr_x_from(x, dist):
    return x - dist/2 + 0.5
def tr_y_from(y, dist):
    return -y + dist/2 - 0.5
def handle_path_plan(req):
    rospy.loginfo("handle_path_plan ++++++++++++++++++")
    map_matr_tmp = [req.costmap_ros[i:i+req.width] for i in range(0, len(req.costmap_ros), req.width)]
    map_matr = []
    for i in range(len(map_matr_tmp)):
        map_matr.append(map_matr_tmp.pop())
    rospy.loginfo(map_matr)
    start_index1 = []
    start_index1.append(tr_x_to(req.start[0].x, req.width))
    start_index1.append(tr_y_to(req.start[0].y, req.height))
    start_index2 = []
    start_index2.append(tr_x_to(req.start[1].x, req.width))
    start_index2.append(tr_y_to(req.start[1].y, req.height))
    goal_index1 = []
    goal_index1.append(tr_x_to(req.goal[0].x, req.width))
    goal_index1.append(tr_y_to(req.goal[0].y, req.height))
    goal_index2 = []
    goal_index2.append(tr_x_to(req.goal[1].x, req.width))
    goal_index2.append(tr_y_to(req.goal[1].y, req.height))
```

```
rospy.loginfo("handle_path_plan +++++++++++++++++")
resolution = 1
start_time = rospy.Time.now()
goal_point = req.goal[0]
goal_point2 = req.goal[1]
tmp_path1 = A_star(map_matr, start_index1, goal_index1, resolution)
path1 = PathArray()
path1.path.append(goal_point)
for p_point in tmp_path1:
        tmp_desired_position = Point()
        tmp_desired_position.x = tr_x_from(p_point[0], req.width)
        tmp_desired_position.y = tr_y_from(p_point[1], req.height)
        tmp_desired_position.z = 0
        path1.path.append(tmp_desired_position)
tmp_path2 = A_star(map_matr, start_index2, goal_index2, resolution)
path2 = PathArray()
path2.path.append(goal_point2)
for p_point in tmp_path2:
        tmp_desired_position = Point()
        tmp_desired_position.x = tr_x_from(p_point[0], req.width)
        tmp_desired_position.y = tr_y_from(p_point[1], req.height)
        tmp_desired_position.z = 0
        path2.path.append(tmp_desired_position)
tmp_i1 = 0
for r1_point in tmp_path1:
        tmp_i2 = 0
        for r2_point in tmp_path2:
                if r1_point==r2_point and tmp_i1==tmp_i2:
rospy.loginfo("AAAAAAAAAAAAAAAAAAAAAAAA")
                        rospy.loginfo(r1_point)
                        rospy.loginfo(tr_x_from(r1_point[0], req.width))
                        rospy.loginfo(tr_y_from(r1_point[1], req.height))
                        rospy.loginfo(tmp_i1)
                tmp_i2 = tmp_i2+1
        tmp_i1 = tmp_i1+1
```

```
                Plan_array = [path1, path2]
                if not path1:
                        rospy.logwarn("No path returned by algorithm")
                        path1 = []
                else:
                        execution_time = rospy.Time.now() - start_time
                        print("\n")
                        rospy.loginfo('+++++++++++++++++ Metrics +++++++++++++++++')
                        rospy.loginfo('Total execution time: %s seconds',
str(execution_time.to_sec()))
                rospy.loginfo('+++++++++++++++++++++++++++++++++++++++++++++')
                        print("\n")
                resp = PathPlan2Response()
                resp.plan_array = Plan_array
                return resp
        def path_plan_server2():
                rospy.init_node('path_plan_server2')
                s = rospy.Service('path_plan2', PathPlan2, handle_path_plan)
                rospy.spin()
        if __name__ == "__main__":
                path_plan_server2()
        // src/mover.py
        #! /usr/bin/env python3
        import rospy
        from sensor_msgs.msg import LaserScan
        from geometry_msgs.msg import Twist, Point
        from nav_msgs.msg import Odometry
        from tf import transformations
        from nav_msgs.msg import OccupancyGrid
        import math
        from test_world.srv import *
        # robot state variables
        position_ = Point()
        position_.z = 4
        desired_position_ = Point()
```

```
yaw_ = 0
# machine state
state_ = 0
# goal
desired_path = []
# map data
map_data = []
# parameters
yaw_precision_ = math.pi / 180 # +/- 1 degree allowed
dist_precision_ = 0.2
# publishers
pub = None
def change_state(state):
        global state_
        state_ = state
        # rospy.loginfo('State changed to [%s]' % state_)
def clbk_odom(msg):
        global position_
        global yaw_
        # position
        position_ = msg.pose.pose.position
        # yaw
        quaternion = (
                msg.pose.pose.orientation.x,
                msg.pose.pose.orientation.y,
                msg.pose.pose.orientation.z,
                msg.pose.pose.orientation.w)
        euler = transformations.euler_from_quaternion(quaternion)
        yaw_ = euler[2]
def get_tr_position():
        tmp_point = Point()
        tmp_point.x = position_.x
        tmp_point.y = position_.y
        tmp_point.z = 0
        return tmp_point
```

```
def callback_map(msg):
        global map_data
        map_data = msg.data
def clbk_laser(msg):
        # 720 / 5 = 144
        # regions = [
        #     min(min(msg.ranges[0:143]), 10),
        #     min(min(msg.ranges[144:287]), 10),
        #     min(min(msg.ranges[288:431]), 10),
        #     min(min(msg.ranges[432:575]), 10),
        #     min(min(msg.ranges[576:713]), 10),
        # ]
        # rospy.loginfo(regions)
        if min(min(msg.ranges[288:431]), 10) < 0.5:
                rospy.loginfo("----------------------")
                rospy.loginfo(min(min(msg.ranges[288:431]), 10))
                rospy.loginfo(position_)
                rospy.loginfo(yaw_)
def main():
        global pub, desired_path, desired_position_
        global yaw_, yaw_precision_, state_
        rospy.init_node('mover')
        pub = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
        sub_odom = rospy.Subscriber('/odom', Odometry, clbk_odom)
        sub_map = rospy.Subscriber("/map", OccupancyGrid, callback_map)
        sub = rospy.Subscriber('/robot/laser/scan', LaserScan, clbk_laser)
        rate = rospy.Rate(20)
        while not map_data or position_.z==4:
                # rospy.loginfo('kk')
                rate.sleep()
        pp_request = PathPlanRequest()
        pp_request.start = get_tr_position()
        goal_point = Point()
        goal_point.x = -8
        goal_point.y = 13
```

```
                goal_point.z = 0
                pp_request.goal = goal_point
                pp_request.costmap_ros = map_data
                pp_request.width = 33
                pp_request.height = 33
                rospy.wait_for_service('path_plan')
                try:
                        path_plan = rospy.ServiceProxy('path_plan', PathPlan)
                        resp = path_plan(pp_request)
                        desired_path = resp.plan
                        rospy.loginfo(desired_path)
                        desired_position_ = desired_path.pop()
                        rospy.loginfo(desired_position_)
                except rospy.ServiceException as e:
                        print("Service call failed: %s"%e)
                while not rospy.is_shutdown() or state_!=2:
                        desired_yaw = math.atan2(desired_position_.y - position_.y,
desired_position_.x - position_.x)
                        err_yaw = desired_yaw - yaw_
                        twist_msg = Twist()
                        if math.fabs(err_yaw) > yaw_precision_*2:
                                if err_yaw > math.pi:
                                        err_yaw = err_yaw - 2*math.pi
                                if err_yaw < -math.pi:
                                        err_yaw = err_yaw + 2*math.pi
                                twist_msg.linear.x = 0
                                # if math.fabs(err_yaw) > yaw_precision_:
                                if err_yaw > 0:
                                        twist_msg.angular.z = -1
                                else:
                                        twist_msg.angular.z = 1
                        else:
                                # rospy.loginfo("-------------------------")
                                err_pos = math.sqrt(pow(desired_position_.y - position_.y, 2) +
pow(desired_position_.x - position_.x, 2))
```

```
                        if err_pos > dist_precision_:
                                twist_msg.linear.x = 1
                                twist_msg.angular.z = 0
                        else:
                                if desired_path:
                                        desired_position_ = desired_path.pop()
                                        rospy.loginfo(desired_position_)
                                        # change_state(0)
                                else:
                                        # rospy.loginfo('Position error: [%s]' % err_pos)
                                        change_state(2)
                # rospy.loginfo(twist_msg)
                pub.publish(twist_msg)
                rate.sleep()
if __name__ == '__main__':
        main()
// src/mover_c1.py
#! /usr/bin/env python3
import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist, Point
from nav_msgs.msg import Odometry
from tf import transformations
from nav_msgs.msg import OccupancyGrid
import math
from test_world.srv import *
from test_world.msg import *
# robot state variables
position_ = Point()
position_.z = 4
position2_ = Point()
position2_.z = 4
desired_position_ = Point()
desired_position2_ = Point()
yaw_ = 0
```

```
yaw2_ = 0
# goal
desired_path = []
desired_path2 = []
# map data
map_data = []
# parameters
yaw_precision_ = math.pi / 180 # +/- 1 degree allowed
dist_precision_ = 0.8
# publishers
pub = None
pub2 = None
# def change_state(state):
#     global state_
#     state_ = state
    # rospy.loginfo('State changed to [%s]' % state_)
def clbk_odom(msg):
    global position_
    global yaw_
    # position
    position_ = msg.pose.pose.position
    # yaw
    quaternion = (
        msg.pose.pose.orientation.x,
        msg.pose.pose.orientation.y,
        msg.pose.pose.orientation.z,
        msg.pose.pose.orientation.w)
    euler = transformations.euler_from_quaternion(quaternion)
    yaw_ = euler[2]
def clbk_odom2(msg):
    global position2_
    global yaw2_
    # position
    position2_ = msg.pose.pose.position
    # yaw
```

```
    quaternion = (
        msg.pose.pose.orientation.x,
        msg.pose.pose.orientation.y,
        msg.pose.pose.orientation.z,
        msg.pose.pose.orientation.w)
    euler = transformations.euler_from_quaternion(quaternion)
    yaw2_ = euler[2]
def get_tr_position():
    tmp_point = Point()
    tmp_point.x = position_.x
    tmp_point.y = position_.y
    tmp_point.z = 0
    return tmp_point
def get_tr_position2():
    tmp_point = Point()
    tmp_point.x = position2_.x
    tmp_point.y = position2_.y
    tmp_point.z = 0
    return tmp_point
def callback_map(msg):
    global map_data
    map_data = msg.data
def clbk_laser(msg):
     # 720 / 5 = 144
     if min(min(msg.ranges[288:431]), 10) < 0.5:
         rospy.loginfo("----------------------")
         rospy.loginfo(min(min(msg.ranges[288:431]), 10))
         rospy.loginfo(position_)
         rospy.loginfo(yaw_)
def main():
    global pub, pub2, desired_path, desired_path2, desired_position_, desired_position2_
    # global yaw_, yaw2_, yaw_precision_, dist_precision_
    rospy.init_node('mover_c1')
    sub_map = rospy.Subscriber("/map", OccupancyGrid, callback_map)
    pub = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
```

```
sub_odom = rospy.Subscriber('/odom', Odometry, clbk_odom)
sub = rospy.Subscriber('/robot/laser/scan', LaserScan, clbk_laser)
pub2 = rospy.Publisher('/cmd_vel2', Twist, queue_size=1)
sub_odom = rospy.Subscriber('/odom2', Odometry, clbk_odom2)
sub = rospy.Subscriber('/robot2/laser/scan', LaserScan, clbk_laser2)
rate = rospy.Rate(20)
while not map_data or position_.z==4 or position2_.z==4:
    # rospy.loginfo('kk')
    rate.sleep()
pp_request = PathPlan2Request()
# int8[] costmap_ros
# int32 width
# int32 height
# geometry_msgs/Point[] start
# geometry_msgs/Point[] goal
start = get_tr_position()
start2 = get_tr_position2()
pp_request.start = [start, start2] # tmp
goal_point = Point()
goal_point.x = -8
goal_point.y = 13
goal_point.z = 0
goal_point2 = Point()
goal_point2.x = 4.5
goal_point2.y = 9.5
goal_point2.z = 0
pp_request.goal = [goal_point, goal_point2] # tmp
pp_request.costmap_ros = map_data
pp_request.width = 33
pp_request.height = 33
rospy.wait_for_service('path_plan2')
try:
    path_plan = rospy.ServiceProxy('path_plan2', PathPlan2)
    resp = path_plan(pp_request)
    desired_path = resp.plan_array[0].path
```

```
            rospy.loginfo(desired_path)
            desired_position_ = desired_path.pop()
            desired_path2 = resp.plan_array[1].path
            rospy.loginfo(desired_path2)
            desired_position2_ = desired_path2.pop()
        except rospy.ServiceException as e:
            print("Service call failed: %s"%e)
        state_ = 0
        state2_ = 0
        while not rospy.is_shutdown() or not state_ or not state2_:
            desired_yaw = math.atan2(desired_position_.y - position_.y, desired_position_.x - position_.x)
            desired_yaw2 = math.atan2(desired_position2_.y - position2_.y, desired_position2_.x - position2_.x)
            err_yaw = desired_yaw - yaw_
            err_yaw2 = desired_yaw2 - yaw2_
            twist_msg = Twist()
            twist_msg2 = Twist()
            if not state_:
                if math.fabs(err_yaw) > yaw_precision_*2:
                    if err_yaw > math.pi:
                        err_yaw = err_yaw - 2*math.pi
                    if err_yaw < -math.pi:
                        err_yaw = err_yaw + 2*math.pi

                    twist_msg.linear.x = 0
                    if err_yaw > 0:
                        twist_msg.angular.z = -1
                    else:
                        twist_msg.angular.z = 1
                else:
                    err_pos = math.sqrt(pow(desired_position_.y - position_.y, 2) + pow(desired_position_.x - position_.x, 2))
                    if err_pos > dist_precision_:
                        twist_msg.linear.x = 1
```

```
                twist_msg.angular.z = 0
            else:
                if desired_path:
                    desired_position_ = desired_path.pop()
                    rospy.loginfo(desired_position_)
                else:
                    state_ = 1
        if not state2_:
            if math.fabs(err_yaw2) > yaw_precision_*2:
                if err_yaw2 > math.pi:
                    err_yaw2 = err_yaw2 - 2*math.pi
                if err_yaw2 < -math.pi:
                    err_yaw2 = err_yaw2 + 2*math.pi
                twist_msg2.linear.x = 0
                if err_yaw2 > 0:
                    twist_msg2.angular.z = -1
                else:
                    twist_msg2.angular.z = 1
            else:
                err_pos2 = math.sqrt(pow(desired_position2_.y - position2_.y, 2) +
pow(desired_position2_.x - position2_.x, 2))
                if err_pos2 > dist_precision_:
                    twist_msg2.linear.x = 1
                    twist_msg2.angular.z = 0
                else:
                    if desired_path2:
                        desired_position2_ = desired_path2.pop()
                        rospy.loginfo(desired_position2_)
                    else:
                        state2_ = 1
        pub.publish(twist_msg)
        pub2.publish(twist_msg2)
        rate.sleep()
if __name__ == '__main__':
    main()
```

```
// srv/PathPlan.srv
int8[] costmap_ros
int32 width
int32 height
geometry_msgs/Point start
geometry_msgs/Point goal
---
geometry_msgs/Point[] plan
// srv/PathPlan2.srv
int8[] costmap_ros
int32 width
int32 height
geometry_msgs/Point[] start
geometry_msgs/Point[] goal
---
PathArray[] plan_array
// urdf/macros.xacro
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
   <xacro:macro name="link_wheel" params="name">
      <link name="${name}">
         <inertial>
           <mass value="0.2"/>          <origin rpy="0 1.570796 1.570796" xyz="0 0 0"/>
           <xacro:cylinder_inertia mass="0.2" l="0.03" r="0.1" />
         </inertial>
         <collision name="${name}_collision">
           <origin rpy="0 1.570796 1.570796" xyz="0 0 0"/>
           <geometry>          <cylinder length="0.03" radius="0.1"/>          </geometry>
         </collision>
         <visual name="${name}_visual">
           <origin rpy="0 1.570796 1.570796" xyz="0 0 0"/>
           <geometry>          <cylinder length="0.03" radius="0.1"/>          </geometry>
         </visual>
      </link>
   </xacro:macro>
```

```
    <xacro:macro name="joint_wheel" params="name child origin_xyz parent">
     <joint name="${name}" type="continuous">
      <origin rpy="0 0 0" xyz="${origin_xyz}"/>
      <parent link="${parent}"/>
      <child link="${child}"/>
      <axis rpy="0 0 0" xyz="0 1 0"/>
      <limit effort="10000" velocity="1000"/>
      <joint_properties damping="1.0" friction="1.0"/>
     </joint>
    </xacro:macro>
    <xacro:macro name="cylinder_inertia" params="mass r l">
     <inertia  ixx="${mass*(3*r*r+l*l)/12}" ixy = "0" ixz = "0"
         iyy="${mass*(3*r*r+l*l)/12}" iyz = "0"
         izz="${mass*(r*r)/2}" />
    </xacro:macro>
  </robot>
  // urdf/robot1/robot.urdf
  <?xml version="1.0" ?>
  <robot name="m2wr" xmlns:xacro="http://www.ros.org/wiki/xacro">
   <xacro:include filename="$(find test_world)/urdf/robot1/robot.gazebo" />
   <xacro:include filename="$(find test_world)/urdf/macros.xacro" />
   <material name="blue">    <color rgba="0.203 0.238 0.285 1.0"/>  </material>
   <link name="link_chassis">
    <!-- pose and inertial -->
    <pose>0 0 0.1 0 0 0</pose>
    <inertial>      <mass value="5"/>      <origin rpy="0 0 0" xyz="0 0 0.1"/>
     <!-- TODO calculate? -->
     <inertia  ixx="0.0395416666667"  ixy="0"  ixz="0"  iyy="0.106208333333"  iyz="0"
izz="0.106208333333"/>
    </inertial>
    <!-- body -->
    <collision name="collision_chassis">
     <geometry>        <box size="0.5 0.3 0.07"/>      </geometry>
    </collision>
    <visual>      <origin rpy="0 0 0" xyz="0 0 0"/>
```

```
        <geometry>      <box size="0.5 0.3 0.07"/>     </geometry>
        <material name="blue"/>
      </visual>
      <!-- caster front -->
      <collision name="caster_front_collision">
        <origin rpy=" 0 0 0" xyz="0.35 0 -0.05"/>
        <geometry>      <sphere radius="0.05"/>     </geometry>
        <surface>      <friction>      <ode>      <mu>0</mu>      <mu2>0</mu2>
<slip1>0</slip1>      <slip2>0</slip2>      </ode>      </friction>     </surface>
      </collision>
      <visual name="caster_front_visual">
        <origin rpy=" 0 0 0" xyz="0.2 0 -0.05"/>
        <geometry>
          <sphere radius="0.05"/>
        </geometry>
      </visual>
    </link>
    <xacro:link_wheel name="link_right_wheel" />
    <xacro:joint_wheel name="joint_right_wheel" child="link_right_wheel" origin_xyz="-
0.05 0.15 0" parent="link_chassis" />
    <xacro:link_wheel name="link_left_wheel" />
    <xacro:joint_wheel name="joint_left_wheel" child="link_left_wheel" origin_xyz="-
0.05 -0.15 0" parent="link_chassis" />
    <link name="sensor_laser">
      <inertial>
        <origin xyz="0 0 0" rpy="0 0 0" />      <mass value="1" />
        <xacro:cylinder_inertia mass="1" r="0.05" l="0.1" />
      </inertial>
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <geometry>      <cylinder radius="0.05" length="0.1"/>      </geometry>
      </visual>
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>      <cylinder radius="0.05" length="0.1"/>      </geometry>
```

```
    </collision>
  </link>
  <joint name="joint_sensor_laser" type="fixed">
    <origin xyz="0.15 0 0.05" rpy="0 0 0"/>
    <parent link="link_chassis"/>    <child link="sensor_laser"/>
  </joint>
</robot>
// urdf/robot1/robot.gazebo
<?xml version="1.0" ?>
<robot>
  <gazebo reference="link_chassis">    <material>Gazebo/DarkGrey</material>
  </gazebo>
  <gazebo reference="link_left_wheel">    <material>Gazebo/Blue</material>
  </gazebo>
  <gazebo reference="link_right_wheel">    <material>Gazebo/Blue</material>
  </gazebo>
  <gazebo>
    <plugin                                    filename="libgazebo_ros_diff_drive.so"
name="differential_drive_controller">
      <!-- <legacyMode>false</legacyMode> -->
      <alwaysOn>true</alwaysOn>      <updateRate>20</updateRate>
      <leftJoint>joint_left_wheel</leftJoint>      <rightJoint>joint_right_wheel</rightJoint>
      <wheelSeparation>0.2</wheelSeparation>      <wheelDiameter>0.2</wheelDiameter>
      <torque>0.1</torque>
      <commandTopic>cmd_vel</commandTopic>
      <odometryTopic>odom</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <robotBaseFrame>link_chassis</robotBaseFrame>
    </plugin>
  </gazebo>
  <gazebo reference="sensor_laser">
    <sensor type="ray" name="head_hokuyo_sensor">
      <pose>0 0 0 0 0 0</pose>      <visualize>true</visualize>
      <update_rate>5</update_rate>
      <ray>
```

```
            <scan>
              <horizontal>
                <samples>720</samples>          <resolution>1</resolution>          <min_angle>-
1.570796</min_angle>          <max_angle>1.570796</max_angle>
              </horizontal>
            </scan>
            <range>
              <min>0.10</min>          <max>10.0</max>          <resolution>0.01</resolution>
            </range>
            <noise>
              <type>gaussian</type>          <mean>0.0</mean>          <stddev>0.01</stddev>
            </noise>
          </ray>
          <plugin                                    name="gazebo_ros_head_hokuyo_controller"
filename="libgazebo_ros_laser.so">
            <topicName>/robot/laser/scan</topicName>
            <frameName>sensor_laser</frameName>
          </plugin>
        </sensor>
      </gazebo>
    </robot>
    // worlds/test_world.world
    <?xml version="1.0" ?>
    <sdf version="1.4">
      <world name="default">
        <!-- A hightmap plane -->
        <include>      <uri>model://map1</uri>     </include>
        <light name="sun" type="directional">
          <cast_shadows>1</cast_shadows>      <pose>0 0 10 0 0 0</pose>      <diffuse>0.9 0.9
0.9 1</diffuse>      <specular>0.0 0.0 0.0 1</specular>
          <attenuation>      <range>100</range>      <constant>1</constant>      </attenuation>
          <direction>-1.0 1.0 -1.0</direction>
        </light>
      </world>
    </sdf>
```

```
// src/mover2.py
#! /usr/bin/env python3
import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist, Point
from nav_msgs.msg import Odometry
from tf import transformations
from nav_msgs.msg import OccupancyGrid
import math
from test_world.srv import *
position_ = Point()
position_.z = 4
desired_position_ = Point()
yaw_ = 0
state_ = 0
desired_path = []
map_data = []
yaw_precision_ = math.pi / 180 # +/- 1 degree allowed
dist_precision_ = 0.8
pub = None
def fix_yaw(des_pos):
        global yaw_, pub, yaw_precision_, state_
        desired_yaw = math.atan2(des_pos.y - position_.y, des_pos.x - position_.x)
        err_yaw = desired_yaw - yaw_
        if err_yaw > math.pi:
                err_yaw = err_yaw - 2*math.pi
        if err_yaw < -math.pi:
                err_yaw = err_yaw + 2*math.pi
        twist_msg = Twist()
        if math.fabs(err_yaw) > yaw_precision_:
                if err_yaw > 0:
                        twist_msg.angular.z = -1
                else:
                        twist_msg.angular.z = 1
        pub.publish(twist_msg)
```

```
            if math.fabs(err_yaw) <= yaw_precision_:
                change_state(1)
        def go_straight_ahead(des_pos):
            global yaw_, pub, yaw_precision_, state_, desired_position_, desired_path
            desired_yaw = math.atan2(des_pos.y - position_.y, des_pos.x - position_.x)
            err_yaw = desired_yaw - yaw_
            err_pos = math.sqrt(pow(des_pos.y - position_.y, 2) + pow(des_pos.x - position_.x,
2))
            if err_pos > dist_precision_:
                twist_msg = Twist()
                twist_msg.linear.x = 1
                pub.publish(twist_msg)
            else:
                if desired_path:
                    desired_position_ = desired_path.pop()
                    rospy.loginfo(desired_position_)
                    change_state(0)
                else:
                    change_state(2)
            if math.fabs(err_yaw) > yaw_precision_*20:
                change_state(0)
        def done():
            twist_msg = Twist()
            twist_msg.linear.x = 0
            twist_msg.angular.z = 0
            pub.publish(twist_msg)
        def change_state(state):
            global state_
            state_ = state
            # rospy.loginfo('State changed to [%s]' % state_)
        def clbk_odom(msg):
            global position_, yaw_
            position_ = msg.pose.pose.position
            quaternion = (
                msg.pose.pose.orientation.x,
```

```
                msg.pose.pose.orientation.y,
                msg.pose.pose.orientation.z,
                msg.pose.pose.orientation.w)
        euler = transformations.euler_from_quaternion(quaternion)
        yaw_ = euler[2]
def get_tr_position():
        tmp_point = Point()
        tmp_point.x = position_.x
        tmp_point.y = position_.y
        tmp_point.z = 0
        return tmp_point
def callback_map(msg):
        global map_data
        map_data = msg.data
def main():
        global pub, desired_path, desired_position_
        rospy.init_node('mover2')
        pub = rospy.Publisher('/cmd_vel2', Twist, queue_size=1)
        sub_odom = rospy.Subscriber('/odom2', Odometry, clbk_odom)
        sub_map = rospy.Subscriber("/map", OccupancyGrid, callback_map)
        rate = rospy.Rate(20)
        while not map_data or position_.z==4:
                rate.sleep()
        pp_request = PathPlanRequest()
        pp_request.start = get_tr_position()
        goal_point = Point()
        goal_point.x = 4.5
        goal_point.y = 9.5
        goal_point.z = 0
        pp_request.goal = goal_point
        pp_request.costmap_ros = map_data
        pp_request.width = 32
        pp_request.height = 32
        rospy.wait_for_service('path_plan')
        try:
```

```
                path_plan = rospy.ServiceProxy('path_plan', PathPlan)
                resp = path_plan(pp_request)
                desired_path = resp.plan
                rospy.loginfo(desired_path)
                desired_position_ = desired_path.pop()
                rospy.loginfo(desired_position_)
            except rospy.ServiceException as e:
                print("Service call failed: %s"%e)
            while not rospy.is_shutdown():
                if state_ == 0:
                    tmp_yaw = yaw_
                    fix_yaw(desired_position_)
                    rate.sleep()
                    if abs(yaw_ - tmp_yaw) < 0.001:
                        change_state(1)
                elif state_ == 1:                    go_straight_ahead(desired_position_)
                elif state_ == 2:                    done()                    pass
                else:
                    rospy.logerr('Unknown state!')
                    pass
                rate.sleep()
        if __name__ == '__main__':
            main()
        // urdf/robot2/robot.urdf
        <?xml version="1.0" ?>
        <robot name="m2wr2" xmlns:xacro="http://www.ros.org/wiki/xacro">
          <xacro:include filename="$(find test_world)/urdf/robot2/robot.gazebo" />
          <xacro:include filename="$(find test_world)/urdf/macros.xacro" />
          <material name="blue">    <color rgba="0.203 0.238 0.285 1.0"/>  </material>
          <link name="link_chassis2">
            <pose>0 0 0.1 0 0 0</pose>
            <inertial>
              <mass value="5"/>      <origin rpy="0 0 0" xyz="0 0 0.1"/>
              <inertia  ixx="0.0395416666667"  ixy="0"  ixz="0"  iyy="0.106208333333"  iyz="0"
izz="0.106208333333"/>
```

```
  </inertial>
  <collision name="collision_chassis">
    <geometry>  <box size="0.5 0.3 0.07"/>  </geometry>
  </collision>
  <visual>  <origin rpy="0 0 0" xyz="0 0 0"/>  <geometry>  <box size="0.5 0.3
0.07"/>  </geometry>  <material name="blue"/>  </visual>
 </link>
 <link name="caster">
  <inertial>
    <mass value="0.2"/>
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <xacro:cylinder_inertia mass="0.2" r="0.05" l="0.1" />
  </inertial>
  <collision name="caster_collision">
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <geometry>  <cylinder length="0.1" radius="0.05"/>  </geometry>
  </collision>
  <visual name="caster_visual">
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <geometry>  <cylinder length="0.1" radius="0.05"/>  </geometry>
  </visual>
 </link>
 <joint name="joint_caster" type="continuous">
  <origin rpy="0 0 0" xyz="0.35 0 -0.05"/>  <parent link="link_chassis2"/>
  <child link="caster"/>  <axis rpy="0 0 0" xyz="0 1 0"/>
  <limit effort="10000" velocity="1000"/>
  <joint_properties damping="1.0" friction="1.0"/>
 </joint>
 <xacro:link_wheel name="link_right_wheel" />
 <xacro:joint_wheel name="joint_right_wheel" child="link_right_wheel" origin_xyz="-
0.05 0.15 0" parent="link_chassis2" />
 <xacro:link_wheel name="link_left_wheel" />
 <xacro:joint_wheel name="joint_left_wheel" child="link_left_wheel" origin_xyz="-
0.05 -0.15 0" parent="link_chassis2" />
 <link name="sensor_laser">
```

```
    <inertial>
      <origin xyz="0 0 0" rpy="0 0 0" />       <mass value="1" />
      <xacro:cylinder_inertia mass="1" r="0.05" l="0.1" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>       <cylinder radius="0.05" length="0.1"/>       </geometry>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>       <cylinder radius="0.05" length="0.1"/>       </geometry>
    </collision>
  </link>
  <joint name="joint_sensor_laser" type="fixed">
   <origin xyz="0.15 0 0.05" rpy="0 0 0"/>
   <parent link="link_chassis2"/>    <child link="sensor_laser"/>
  </joint>
</robot>
// urdf/robot2/robot.gazebo
<?xml version="1.0" ?>
<robot>
  <gazebo reference="link_chassis2">    <material>Gazebo/DarkGrey</material>
  </gazebo>
  <gazebo reference="link_left_wheel">    <material>Gazebo/Blue</material>
  </gazebo>
  <gazebo reference="link_right_wheel">    <material>Gazebo/Blue</material>
  </gazebo>
  <gazebo>
   <plugin                                    filename="libgazebo_ros_diff_drive.so"
name="differential_drive_controller">
    <alwaysOn>true</alwaysOn>      <updateRate>20</updateRate>
    <leftJoint>joint_left_wheel</leftJoint>      <rightJoint>joint_right_wheel</rightJoint>
    <wheelSeparation>0.2</wheelSeparation>      <wheelDiameter>0.2</wheelDiameter>
    <torque>0.1</torque>
    <commandTopic>cmd_vel2</commandTopic>
```

```
      <odometryTopic>odom2</odometryTopic>
      <odometryFrame>odom2</odometryFrame>
      <robotBaseFrame>link_chassis2</robotBaseFrame>
     </plugin>
    </gazebo>
    <gazebo reference="sensor_laser">
     <sensor type="ray" name="head_hokuyo_sensor">
      <pose>0 0 0 0 0 0</pose> <visualize>true</visualize> <update_rate>5</update_rate>
      <ray>
       <scan>
        <horizontal>
         <samples>720</samples>          <resolution>1</resolution>
         <min_angle>-1.570796</min_angle>          <max_angle>1.570796</max_angle>
        </horizontal>
       </scan>
       <range>
        <min>0.10</min>        <max>10.0</max>        <resolution>0.01</resolution>
       </range>
       <noise>
        <type>gaussian</type>        <mean>0.0</mean>        <stddev>0.01</stddev>
       </noise>
      </ray>
      <plugin                                    name="gazebo_ros_head_hokuyo_controller"
filename="libgazebo_ros_laser.so">
       <topicName>/robot2/laser/scan</topicName>
       <frameName>sensor_laser2</frameName>
      </plugin>
     </sensor>
    </gazebo>
   </robot>
```