Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра автоматизації та комп'ютерно-інтегрованих технологій

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

АВТОМАТИЗАЦІЯ ПЛАНУВАННЯ РУХУ В СЕРЕДОВИЩАХ З ПЕРЕШКОДАМИ

AUTOMATION OF MOTION PLANNING IN ENVIRONMENTS WITH OBSTACLES

Спеціальність 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка

освітня програма «Автоматизація та комп'ютерно-інтегровані технології»

Виконав: здобувач вищої освіти
групи АВм - 21
Свирид Юрій Вячеславович

_____
(підпис)

Керівник:
к. т. н., доцент
Гуменюк Павло Олександрович

_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Гуменюк П. О.

_____
(підпис)

Луцьк – 2025

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій
Кафедра автоматизації та комп'ютерно-інтегрованих технологій
Ступінь вищої освіти: _*магістр*_
Галузь знань: *17 Електроніка, автоматизація та електронні комунікації*
Спеціальність: *174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка*
Освітня програма: *«Автоматизація та комп'ютерно-інтегровані технології»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О. Ю. Повстяной
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Свирида Юрія Вячеславовича
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *Автоматизація планування руху в середовищах з перешкодами*

Керівник роботи: *к.т.н., доцент Гуменюк Павло Олександрович*

затверджені наказом закладу вищої освіти від « 27 » 06 2025 року N 304/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: « 1 » 12 2025 року

3. Вихідні дані до роботи: дані з Internet

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

Планування шляху: простір конфігурацій, клітинна декомпозиція. Класифікація алгоритмів побудови шляху робота з перешкодами. Дослідження ефективності пошуку шляху в дискретному середовищі з перешкодами. Програмне та апаратне забезпечення для автономної навігації в середовищах з перешкодами. Автономне планування траєкторій у середовищах з перешкодами.

5. Перелік графічного матеріалу :

графічний матеріал виконано у вигляді презентації, яка складається з 15 слайдів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	*Гуменюк П. О.*		
Розділ 2	*Гуменюк П. О.*		
Розділ 3	*Гуменюк П. О.*		
Розділ 4	*Гуменюк П. О.*		
Розділ 5	*Гуменюк П. О.*		
Нормоконтроль	*Лапченко Ю. С.*		
Показник запозичень тексту			
Академічна доброчесність	*Федік Л. Ю.*		

7. Дата видачі завдання _____*27.06.2025 р.*_____

КАЛЕНДАРНИЙ ПЛАН

N з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми за темою роботи та постановка задач	01.09.2025 р.	
2	Аналіз і вибір напрямків дослідження	10.09.2025 р.	
3	Теоретичне дослідження та практична реалізація	20.09.2025 р.	
4	Опис засобів розробки об’єкта проєктування	01.10.2025 р.	
5	Загальні висновки та рекомендації	20.10.2025 р.	
6	Оформлення роботи	10.11.2025 р.	
7	Оформлення презентації	20.11.2025 р.	
8	Здача чистового варіанту кваліфкаційної роботи на	01.12.2025 р.	
	кафедру		

Здобувач вищої освіти _____ _____Свирид Ю. В._____
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи _____ _____Гуменюк П. О._____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Свирид Ю. В. Автоматизація планування руху в середовищах з перешкодами. Рукопис.

Кваліфікаційна робота магістра ОП «Автоматизація та комп'ютерно-інтегровані технології» спеціальності 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка. Луцький національний технічний університет, Луцьк 2025.

Кваліфікаційна робота магістра складається з вступу, п'яти розділів, загальних висновків та рекомендацій, переліку використаних джерел та додатків.

Головною метою кваліфікаційної роботи було створення системи автономного планування траєкторій для мобільних роботів у середовищах з перешкодами. Розроблене програмне забезпечення є ефективним інструментом для автономного планування траєкторій роботів. Використання інтегрального підходу дозволило створити систему, здатну швидко та точно оцінювати та будувати маршрути на основі заданих параметрів та змінних середовищ.

Розроблене програмне забезпечення може стати одним з компонентів у процесі проектування та експлуатації автономних роботів, забезпечуючи ефективність роботи.

Об'єм графічної частини магістерської роботи складає 15 слайдів. Обсяг пояснювальної записки становить 74 друковані сторінки.

Ключові слова: шлях робота, траєкторія, маршрутизація, рух з перешкодами, моделювання.

ANNOTATION

Svyryd Yu. Automation of motion planning in environments with obstacles. Manuscript.

Master's qualification work of OP "Automation and computer-integrated technologies" specialty 174 Automation, computer-integrated technologies and robotics. Lutsk National Technical University, Lutsk 2025.

The master's qualification work consists of an introduction, five chapters, general conclusions and recommendations, a list of references and appendices.

The main goal of the qualification work was to create a system for autonomous trajectory planning for mobile robots in environments with obstacles. The developed software is an effective tool for autonomous robot trajectory planning. The use of an integrated approach made it possible to create a system capable of quickly and accurately evaluating and constructing routes based on specified parameters and variable environments.

The developed software can become one of the components in the process of designing and operating autonomous robots, ensuring operational efficiency.

The volume of the graphic part of the master's thesis is 15 slides. The volume of the explanatory note is 74 printed pages.

Keywords: robot path, trajectory, routing, obstacle avoidance, simulation.

ЗМІСТ

ВСТУП

Управління рухом роботів та планування шляху є фундаментальними задачами робототехніки, які поєднують у собі механічні, алгоритмічні та інтелектуальні підходи. Зі зростанням попиту на автономні системи в різних галузях, таких як промисловість, логістика та сервісні послуги, виникає потреба у розробці ефективних та надійних методів навігації у складних та непередбачуваних умовах.

Особливу увагу привертає розробка методів та засобів забезпечення надійності систем, зокрема, ефективне планування шляху роботів у середовищах з перешкодами. Нездатність робота долати перешкоди або ефективно знаходити оптимальний шлях може призвести до збоїв у роботі, пошкоджень обладнання, а також значних економічних втрат. Тому забезпечення належного планування шляху є критично важливим для підвищення надійності та ефективності функціонування робототехнічних систем.

Актуальність теми зумовлена тим, що забезпечення надійності та ефективності функціонування робототехнічних систем у сучасних умовах є критично важливою задачею. Розробка автоматизованих методів планування шляху роботів є актуальною, оскільки це може сприяти підвищенню безпеки, зменшенню витрат на обслуговування та уникненню аварійних ситуацій.

Метою роботи є розробка та аналіз алгоритмів планування шляху роботів у середовищах з перешкодами для забезпечення їхньої надійності та ефективності.

Об’єктом дослідження є процеси планування шляху мобільних роботів у динамічних середовищах з перешкодами та їхній вплив на функціональність та безпеку.

Предметом дослідження є методи та алгоритми планування шляху роботів, їхня класифікація, програмна реалізація та автоматизація з метою підвищення надійності робототехнічних систем.

Наукова новизна полягає у порівняльному аналізі та розробці гібридних алгоритмів планування шляху, що можуть бути автоматизовані.

Практична новизна полягає у застосовуванні алгоритмів планування шляху для забезпечення надійності та ефективності роботи робототехнічних систем у різних галузях, де потрібна автономна навігація.

У процесі виконання роботи необхідно вирішити наступні задачі.

1. Проаналізувати теоретичні основи планування шляху, включаючи концепцію простору конфігурацій та клітинної декомпозиції.

2. Описати класифікацію та основні типи алгоритмів побудови шляху робота з перешкодами.

3. Розглянути сучасні підходи до оцінки ефективності пошуку шляху в дискретному середовищі.

4. Проаналізувати програмне та апаратне забезпечення для автономної навігації. Визначити необхідні бібліотеки та програмні засоби для реалізації автоматизованої системи планування шляху.

5. Розробити та реалізувати алгоритми автономного планування траєкторій з урахуванням визначених критеріїв. Провести порівняльний аналіз ефективності обраних алгоритмів на основі інтегральної оцінки.

6. Реалізувати комплексну програму для демонстрації автономної навігації, використовуючи інтегрований підхід.

Апробація результатів дослідження. Основні результати досліджень відображені у науковій статті Свирид Ю. В., Гуменюк П. О. Класифікація алгоритмів побудови шляху робота з перешкодами. *Технологічні комплекси*. Луцьк, 2025. Том 17, № 2. С. 64-75.

РОЗДІЛ 1
ПЛАНУВАННЯ ШЛЯХУ: ПРОСТІР КОНФІГУРАЦІЙ, КЛІТИННА ДЕКОМПОЗИЦІЯ

Управління роботами охоплює широкий спектр завдань, починаючи від фундаментальних механічних принципів, що керують фізичним рухом, і закінчуючи складними інтелектуальними алгоритмами для високорівневого прийняття рішень та вибору глобальних цілей.

Планування шляху є фундаментальною задачею в робототехніці, що дозволяє автономним системам ефективно та безпечно переміщатися з початкової точки до цільової в заданому середовищі. Особливо складним є планування шляху в середовищах, що містять перешкоди, які можуть бути статичними або динамічними, відомими заздалегідь або невідомими. Ефективне уникнення перешкод є критично важливим для успішної та безпечної роботи мобільних роботів у реальних умовах [1-3].

Центральною та найактуальнішою проблемою в галузі управління рухом та його планування є пошук оптимальних траєкторій. Траєкторія, на відміну від простого шляху, є послідовністю конфігурацій, яка враховує фізичні обмеження робота, включаючи швидкість, прискорення та можливості виконавчих механізмів [4-5]. Конкретне завдання побудови шляху на плоскій карті з дискретними перешкодами слугує важливим фундаментальним завданням, що забезпечує спрощений, але важливий контекст для розробки та тестування алгоритмів, застосовних до більш складних реальних сценаріїв.

Одночасне виконання цих складних завдань при точному врахуванні та управлінні динамікою роботів залишається значним викликом [6]. У практичних, реальних застосуваннях необхідність роботи в реальному часі з мінімальною попередньою обробкою часто має пріоритет над суворим дотриманням теоретичних гарантій, що призводить до компромісів у розробці алгоритмів [7]. Швидка еволюція промислової робототехніки від жорстких, одноцільових

систем до високоадаптивних та інтелектуальних технологій, зумовлена проривами в штучному інтелекті та машинному навчанні, додала нові рівні складності, пов’язані з масштабованістю, економічною ефективністю та ширшими соціальними наслідками [8].

Історично системи управління роботами були переважно детермінованими, покладаючись на точні, попередньо запрограмовані рухи та детальні моделі структурованих середовищ. Це було ефективно для повторюваних завдань у контрольованих заводських умовах. Однак, спостерігається чітка тенденція до розгортання роботів у складних, неструктурованих та динамічних реальних сценаріях, таких як автономні транспортні засоби, медична робототехніка та пошуково-рятувальні місії [2]. У цих середовищах повне попереднє знання неможливе, а умови змінюються непередбачувано. Це вимагає фундаментального переходу від суто детермінованого управління до адаптивної поведінки.

1.1 Концепція простору конфігурацій

Концепція простору конфігурацій (C-space) – це абстрактний математичний простір, у якому кожна точка відповідає певному положенню або стану системи (наприклад, робота чи робота-маніпулятора). Цей підхід дозволяє перетворити задачу планування руху з фізичного простору (де є перешкоди) в геометричну – у C-space, де перешкоди стають абстрактними областями, яких потрібно уникати [9].

Для робота конфігурація – це повний опис його просторового стану. Вона визначає положення кожної точки об’єкта у фіксованій системі координат. Конфігурація включає як координати, що задають позицію робота, так і його орієнтацію в просторі. У випадку складних систем, таких як маніпулятори, вона також охоплює всі внутрішні параметри, що впливають на положення окремих ланок.

Таким чином, конфігурація однозначно визначає геометричну форму і розташування робота у кожен момент часу [7, 9]. Отже, C-space робота – це всеосяжний простір, що містить усі такі можливі конфігурації.

Основна корисність С-простору полягає в його здатності абстрагувати самого робота в єдину точку в цьому спеціалізованому просторі, одночасно перетворюючи фізичні перешкоди, присутні в реальному робочому просторі, у відповідні області в С-просторі. Це елегантне перетворення спрощує складну проблему планування руху фізичного об’єкта з розмірами в більш кероване завдання планування руху безрозмірної точки [7].

Перешкоди, що зустрічаються в робочому просторі робота, відображаються в С-просторі, де вони проявляються як «С-перешкоди» [7]. Ці С-перешкоди окреслюють області, яких точка, що представляє робота, повинна ретельно уникати, щоб запобігти зіткненням [10]. «Вільний простір» у C-space, позначений як C_free, отже, визначається як доповнення до областей С-перешкод (C_obs). C_free представляє всі конфігурації, в яких робот може існувати та рухатися без будь-яких зіткнень [7, 10].

Основне значення конфігураційного простору полягає в тому, що він дозволяє звести задачу планування руху до геометричної проблеми пошуку шляху. Замість аналізу руху об’єкта у фізичному просторі, задача переноситься у простір вищої розмірності, де кожна точка відповідає певному стану системи.

У цьому просторі планування зводиться до побудови безперервної траєкторії між початковою та цільовою конфігурацією. Важливою умовою є те, що ця траєкторія повинна повністю перебувати в області допустимих конфігурацій, тобто не перетинати зону заборонених станів, які відповідають зіткненням з перешкодами. Така область називається вільним конфігураційним простором і позначається як C_free [10].

Для багатороботних систем створюється спеціалізований «безпечний простір конфігурацій» (SC) шляхом явного виключення будь-яких конфігурацій, де кілька роботів займали б одне й те саме положення, тим самим гарантуючи

уникнення зіткнень між ними [9]. Якщо цей безпечний простір конфігурацій є шляхово-зв’язним, це означає, що роботи «вільно транспортуються» у своєму спільному середовищі, що є критично важливою властивістю для кооперативної робототехніки.

C-простір не є просто теоретичною конструкцією, а фундаментальним концептуальним проривом, який лежить в основі практично всього сучасного планування руху. Він дозволяє систематичний, математичний підхід до уникнення зіткнень та генерації шляху, формуючи весь ландшафт досліджень та розробок у галузі планування руху роботів.

1.2 Клітинна декомпозиція для планування в дискретному просторі

Просторова декомпозиція, або клітинна декомпозиція, є базовим підходом у задачах планування шляху. Її суть полягає в поділі вільного конфігураційного простору (C_free) на обмежену кількість простих областей, які називаються комірками [8].

Цей підхід дозволяє представити безперервний простір C у вигляді дискретної структури. Кожна комірка отримує чітке призначення. Якщо вона перетинається з перешкодою (C_obs), вона вважається недоступною і позначається як BLOCKED. Якщо ж вона повністю лежить у допустимій області, її позначають як FREE [11].

Після цієї категоризації будується граф зв’язності. Цей граф будується на основі відносин суміжності, виявлених між комірками. У цьому графі вузли представляють FREE комірки, а зв’язки (ребра) з’єднують суміжні комірки, що означає можливість переміщення між ними [7, 11]. Проблема планування шляху потім переформулюється як задача пошуку на цьому щойно побудованому дискретному графі зв’язності. Метою стає знаходження послідовності суміжних FREE комірок, часто званої «каналом», яка з’єднує комірку початкової конфігурації з коміркою цільової конфігурації. Ця дискретизація є ключовою,

оскільки вона дозволяє застосовувати добре зрозумілі та ефективні алгоритми пошуку по графу (наприклад, A*) для вирішення проблеми планування руху. Без цього перетворення безпосередній пошук у безперервному, високорозмірному просторі був би обчислювально нерозв'язним.

Сітковий метод (Grid Method), вперше запропонований у 1968 році, є прикладом поширеного підходу до приблизної клітинної декомпозиції [11]. Він передбачає поділ робочого простору (або C-простору) робота на рівномірні сіткові комірки. Кожна комірка зазвичай містить двійкову інформацію, що вказує, чи вона вільна, чи зайнята перешкодою. У цьому підході дискретна сітка, що складається з комірок заздалегідь визначеної, узгодженої форми (наприклад, квадратів), накладається на C-простір. Будь-яка комірка, яка перетинається з C_obs, позначається як BLOCKED. Для спрощення задачі планування робот часто моделюється як рухома точкова маса, при цьому його положення обмежується центрами цих сіткових комірок, що ефективно зменшує складність задачі.

Приблизна клітинна декомпозиція відрізняється легкістю побудови, з обчислювальною складністю, лінійною відносно кількості комірок. Ці методи широко застосовуються на практиці завдяки їхній простоті реалізації та наявності ефективних процедур пошуку.

Значним недоліком є те, що ця декомпозиція є «втратною». Вона надає наближення C_free, яке може ненавмисно виключити або «видалити» потенційно доступні частини вільного простору. Це наближення призводить до явища «ефективного збільшення перешкод», де перешкоди здаються більшими через дискретний характер сітки. Важливо, що приблизна декомпозиція, як правило, не є повною. Навіть якщо дійсний шлях існує в безперервному просторі, алгоритм може не знайти його, особливо у вузьких проходах, залежно від обраного розміру сітки.

Збільшення роздільної здатності сітки для пом'якшення цієї проблеми (тобто використання менших комірок) різко збільшує кількість комірок і,

відповідно, розмір графу зв’язності, що призводить до вищих обчислювальних витрат.

Ітеративні методи приблизної декомпозиції, такі як Quad-trees (або N-trees), можуть динамічно вирішувати проблеми роздільної здатності шляхом рекурсивного поділу «ЗМІШАНИХ» комірок (тих, що частково перетинаються з перешкодами) до досягнення бажаної роздільної здатності або чіткої категоризації.

Точна клітинна декомпозиція точно розділяє C_free на кінцеву сукупність неперекриваючихся комірок, зазвичай опуклих багатокутників або трапецій [7, 11]. Ключовою характеристикою є те, що ця декомпозиція є «безвтратною», тобто вона зберігає кожну частину C_free без будь-якого наближення або видалення доступних областей. Одним з поширених способів реалізації є проєктування вертикальних променів від кожної вершини багатокутника в межах C_free, що ефективно розкладає C_free на набір трапецій. Вершини для графу зв’язності потім стратегічно розміщуються всередині цих трапецій (наприклад, у їхніх центроїдах) та вздовж вертикальних сегментів, які потім з’єднуються для формування графу.

Основною перевагою точної клітинної декомпозиції є її гарантія повноти: якщо в C_free існує шлях без зіткнень, цей метод гарантовано його знайде. Крім того, він забезпечує, що вільний простір у кожній комірці та вздовж шляхів, що з’єднують суміжні комірки, є справді вільним від зіткнень. Цей підхід теоретично може бути розширений на вищі виміри та обробляти середовища з неполігональними межами [11].

Незважаючи на свою теоретичну повноту, точна клітинна декомпозиція стає обчислювально дорогою та значно складнішою в реалізації в середовищах, що перевищують три виміри. Отже, вона «не дуже популярна» в практичних застосуваннях порівняно з приблизними методами через її внутрішню складність.Крім того, вона зазвичай вимагає допоміжного локального

контролера навігації для управління рухом робота всередині окремих комірок та переходами між ними.

1.3 Класифікація підходів до планування шляху

Алгоритми планування шляху широко класифікуються за їхньою сферою застосування та знаннями про навколишнє середовище на глобальне планування шляху та локальне планування шляху [12-14].

Глобальне планування шляху передбачає повне та попереднє знання про навколишнє середовище [12-13]. Його мета полягає в обчисленні оптимального шляху без зіткнень від початкової до цільової позиції в усьому відомому середовищі. До відомих прикладів належать алгоритм Дейкстри, алгоритм A*, алгоритм D* та алгоритм швидко досліджуваних випадкових дерев (RRT) [12, 14].

На противагу цьому, локальні алгоритми планування розроблені для уникнення перешкод у реальному часі та навігації в невідомих або динамічних середовищах [13]. Вони реагують на негайні дані сенсорів, що робить їх критично важливими для адаптації до непередбачених змін або рухомих перешкод. Прикладами є метод штучного потенційного поля (APF) та підхід динамічного вікна (DWA) [12-14].

Інша класифікація розрізняє класичні алгоритми та інтелектуальні алгоритми [12]. Традиційні методи зазвичай базуються на математичних моделях або заздалегідь визначених правилах для генерації шляху [13]. Інтелектуальні алгоритми, однак, інтегрують принципи штучного інтелекту та машинного навчання, пропонуючи розширені можливості навчання та адаптивності, особливо у високорозмірних та складних середовищах. Це супроводжується підвищеною складністю реалізації та вимогами до обчислювальних ресурсів.

Висновок до розділу 1

Управління рухом роботів та планування шляху є фундаментальними задачами робототехніки, що поєднують механічні, алгоритмічні та інтелектуальні підходи. Концепція простору конфігурацій надає потужний математичний інструмент для перетворення фізичних перешкод у геометричні області, що значно спрощує пошук безпечних траєкторій.

Клітинна декомпозиція, як метод дискретизації простору, забезпечує практичні алгоритми планування шляху, проте має компроміси між точністю і обчислювальною складністю. Класифікація алгоритмів на глобальні та локальні відображає різні підходи до роботи в статичних та динамічних середовищах.

Сучасні тенденції розвитку орієнтовані на інтеграцію адаптивних інтелектуальних методів, що дозволяють роботам ефективно працювати у непередбачуваних умовах. Ці зміни вимагають балансу між теоретичною повнотою алгоритмів та їхньою практичною ефективністю.

РОЗДІЛ 2
КЛАСИФІКАЦІЯ АЛГОРИТМІВ ПОБУДОВИ ШЛЯХУ РОБОТА З ПЕРЕШКОДАМИ

Класифікація методів планування шляху за різними критеріями, такими як використання інтелектуальних технологій, тип навколишнього середовища та повнота інформації про нього, не є лише описовою. Вона відображає основні рушійні сили для розвитку алгоритмів. Наприклад, якщо алгоритм демонструє низьку продуктивність у динамічному середовищі, це створює потребу в розробці нових алгоритмів або модифікацій, які явно враховують динаміку. Аналогічно, обмежена інформація про середовище вимагає переходу від глобального планування до локального або до постійного перепланування. Це вказує на те, що обмеження алгоритмів в одній категорії (наприклад, традиційних методів для статичних середовищ з повною інформацією) безпосередньо стимулюють інновації та створення рішень в інших категоріях (наприклад, евристичних методів для динамічних середовищ з неповною інформацією) або гібридних підходів. Такий взаємозв’язок демонструє безперервний цикл досліджень та розробок у робототехніці, де нові виклики в реальних умовах експлуатації (наприклад, невідомі, рухомі перешкоди, вимоги до роботи в реальному часі) призводять до вдосконалення та винаходу більш надійних та адаптивних рішень для планування шляху.

2.1 Класичні алгоритми пошуку на графах

Класичні алгоритми пошуку є основою для багатьох методів планування шляху. Вони ефективні в середовищах, де інформація про простір є повністю відомою та статичною, а середовище може бути представлене у вигляді графа або сітки.

2.1.1 Алгоритм Дейкстри

Алгоритм Дейкстри призначений для знаходження найкоротшого шляху від однієї початкової вершини до всіх інших вершин у графі з невід'ємними вагами ребер. Його діяльність ґрунтується на ітеративному процесі: на кожному кроці обирається невідвідана вершина з найменшою поточною відстанню від початку, після чого оновлюються відстані до її сусідів. Цей процес триває доти, доки всі вершини не будуть відвідані або поки не буде знайдено шлях до цільової вершини [15].

У контексті планування шляху робота, робоче поле попередньо розбивається на клітини або вузли графа, а ребрам присвоюються значення, що відображають відстань або вартість переміщення. Перешкоди моделюються як недоступні вузли або ребра з нескінченною вагою, що ефективно виключає їх з розгляду під час пошуку шляху. Головною перевагою алгоритму Дейкстри є гарантоване знаходження найкоротшого шляху у статичних середовищах без негативних ваг ребер. Однак, його основним недоліком є висока обчислювальна вартість для великих графів, оскільки він обчислює шлях до всіх вершин, навіть якщо потрібен шлях лише до однієї цілі.

2.1.2 Алгоритм A*

Алгоритм A* є розширенням алгоритму Дейкстри, що поєднує його переваги з евристичною функцією для прискорення пошуку [15, 16]. Він обчислює вартість шляху від початкової точки до кінцевої, використовуючи евристичну функцію h(n), яка оцінює відстань від поточної вершини n до цільової. Загальна функція вартості f(n) визначається як сума фактичної вартості g(n) (від початкової точки до n) та евристичної оцінки h(n). Алгоритм обирає наступну вершину для розширення, яка має найменше значення f(n).

Подібно до Дейкстри, A* працює на дискретизованій карті (сітці або графі), де перешкоди позначаються як непрохідні. Евристична функція, наприклад, манхеттенська або евклідова відстань, направляє пошук до цілі, уникаючи розширення вузлів у напрямку перешкод, оскільки їх вартість буде

значно вищою або нескінченною. Перевагою A* є значно вища швидкість порівняно з Дейкстрою у багатьох випадках завдяки евристиці, при цьому зберігаючи гарантію оптимальності, якщо евристика є допустимою. Проте, A* може мати проблеми з продуктивністю в реальному часі та великим обсягом обчислень на вузол, особливо в складних середовищах.

2.1.3 Алгоритм Флойда-Уоршелла

Алгоритм Флойда-Уоршелла призначений для знаходження найкоротших шляхів між усіма парами вузлів у зваженому орієнтованому графі [17]. Його робота ґрунтується на принципі динамічного програмування, де оцінки найкоротших шляхів ітеративно покращуються шляхом розгляду можливості використання проміжних вершин.

Хоча алгоритми Дейкстри та A* знаходять шлях від однієї точки до багатьох або однієї цілі, Флойд-Уоршелл є цінним, коли необхідно швидко визначити найкоротший шлях між будь-якими двома точками в мережі. Це особливо корисно в сценаріях з кількома роботами або кількома можливими початковими/кінцевими точками, де потрібно вибрати оптимальну пару (наприклад, який БПЛА найшвидше дістанеться до цілі). Перешкоди інтегруються шляхом встановлення нескінченної ваги для ребер, що перетинають перешкоди, або шляхом початкового побудови графа, який включає лише безперешкодні зв’язки. Алгоритм Флойда-Уоршелла ефективний для задач «всі пари найкоротших шляхів» і може бути використаний для динамічного призначення ваг у моделях глибоких графів для оптимізації шляхів. Однак, його основним недоліком є висока обчислювальна складність, що робить його менш придатним для дуже великих графів або динамічних середовищ, де граф часто змінюється.

2.1.4 Хвильовий алгоритм (BFS)

Хвильовий алгоритм (Breadth-First Search, BFS) є алгоритмом пошуку на графі, який досліджує всі вершини на поточному рівні глибини, перш ніж переходити до вершин наступного рівня. У контексті планування шляху,

особливо в сіткових картах, його можна розуміти як фізичне розповсюдження хвильового фронту [15]. Від початкової клітини хвиля розповсюджується на сусідні клітини, позначаючи їх відстанню від початку.

Перешкоди в сітковій карті позначаються як непрохідні клітини, через які хвильовий фронт не може розповсюджуватися. Це забезпечує, що знайдений шлях буде обходити перешкоди. Після розповсюдження хвилі до цільової клітини, шлях відновлюється зворотним ходом від цілі до початку, слідуючи за градієнтом найменших значень. Перевагою хвильового алгоритму є гарантоване знаходження найкоротшого шляху в незважених графах (якщо всі ребра мають однакову вагу) та простота реалізації. Проте, він не враховує ваги ребер (якщо вони різні) і може бути неефективним для великих просторів, оскільки досліджує всі можливі шляхи на кожному рівні.

2.2 Евристичні алгоритми

Евристичні алгоритми, часто базуючись на класичних методах, впроваджують оптимізації та адаптації для підвищення ефективності, особливо у складних або динамічних середовищах.

2.2.1 Модифікований A*

Модифікований A* спрямований на подолання обмежень традиційного A*, таких як низька продуктивність у реальному часі, великий обсяг обчислень на вузол, тривалий час обчислення та низька ефективність пошуку. Покращення включають оптимізацію евристичних функцій та коригування відстаней до перешкод на карті [18].

Одним з ключових методів є розширення перешкод (expansion distance), що суттєво змінює діапазон чутливості мобільного робота. Це дозволяє планувати шлях з більшою відстанню до перешкод, забезпечуючи достатній простір для маневрування та підвищуючи безпеку, особливо під час поворотів. Розмір радіуса розширення зазвичай визначається радіусом самого робота. Перевагами

модифікованого A* є покращена точність планування шляху, зменшення ризику зіткнень та здатність генерувати більш плавні та реалістичні траєкторії, особливо у випадку гібридного A*. Однак, це може призвести до збільшення часу виконання та довжини шляху.

2.2.2 D* Lite

Алгоритм D* Lite є ефективним алгоритмом пошуку найкоротшого шляху у зваженому орієнтованому графі, основною особливістю якого є здатність працювати в динамічних середовищах, де структура графа не відома заздалегідь або постійно змінюється [15]. Він є ефективною модифікацією A*, що вимагає менше часу обчислень та скорочує кількість розширених вузлів.

D* Lite дозволяє роботу вільно переміщатися в невідомому та швидко змінюваному середовищі. При виявленні змін (наприклад, появи нових перешкод або зникнення старих), алгоритм ефективно перепланує шлях, адаптуючись до нової обстановки без необхідності повного перерахунку з нуля. Це робить його ідеальним для динамічних середовищ зі змінними перешкодами. Перевагами D* Lite є його ефективність у динамічних та невідомих середовищах, швидке перепланування та менші обчислювальні витрати порівняно з повним перерахунком A*. Проте, його реалізація є складнішою порівняно з A*.

2.2.3 Theta*

Алгоритм Theta* є алгоритмом планування шляху на основі пошуку на графі, який забезпечує оптимальний шлях з більшою гнучкістю, ніж алгоритм A*, з точки зору маршрутів. Ключова відмінність від A* полягає в тому, що батьківський вузол вершини в Theta* не обов'язково має бути сусіднім, якщо між ними існує пряма видимість (line-of-sight) [19]. Це дозволяє генерувати «будь-які кути» (any-angle) шляхів, що є більш реалістичним для руху роботів, ніж шляхи, обмежені сіткою.

Theta* використовує механізм перевірки видимості, щоб «зламати» обмеження растрового середовища. Це дозволяє роботу планувати шлях, який не проходить через вузли сітки, а замість цього може «зрізати кути» навколо

перешкод, якщо пряма лінія до батьківського вузла є вільною. Це допомагає уникнути надмірної кількості точок повороту та гострих кутів, які є проблемою для традиційних сіткових алгоритмів. Перевагами Theta* є генерація плавніших та коротших шляхів порівняно з A* на сіткових картах та більш реалістичні траєкторії для мобільних роботів. Однак, традиційний Theta* може бути складним для одночасного врахування глобальних та детальних аспектів планування, а також може обходити більше вузлів, що призводить до значного обсягу обчислень, особливо зі збільшенням розміру карти [20].

Еволюція від A* до модифікованого A* та Theta* демонструє критичну зміну від чистої оптимальності найкоротшого шляху до якості шляху (плавності, запасу безпеки) як першочергової мети в плануванні шляху роботів. Для фізичних роботів математично «найкоротший» шлях (наприклад, зигзагоподібний на сітці) може бути непрактичним, енергетично неефективним або навіть небезпечним через кінематичні обмеження, граничні можливості приводів або шуми сенсорів. Акцент зміщується від чисто геометричного найкоротшого шляху до «здійсненного та безпечного» шляху, який мінімізує втрати енергії, знос та ризик зіткнень.

2.3 Метаевристичні алгоритми

Метаевристичні алгоритми – це оптимізаційні підходи, які не гарантують знаходження глобального оптимуму, але є дуже ефективними для пошуку «достатньо хороших» рішень у складних, багатовимірних або нелінійних просторах, де традиційні методи можуть бути неефективними.

2.3.1 Генетичні алгоритми (ГА)

Генетичні алгоритми (ГА) є метаевристикою, натхненною процесом природного відбору, що належить до ширшого класу еволюційних алгоритмів [21]. Вони працюють шляхом ітеративної модифікації популяції індивідуальних рішень, де кожне рішення (хромосома) представляє можливий шлях для робота.

Шляхи оцінюються за допомогою функції вартості (фітнесу), яка враховує такі параметри, як відстань, кількість змін напрямку та зіткнення з перешкодами [22]. Найкращі рішення (з найменшою вартістю) вибираються для «розмноження» за допомогою генетичних операторів, таких як схрещування та мутація, для створення нового покоління [21, 22].

Обробка перешкод є критичною частиною функції вартості. Якщо будь-яка точка запропонованого шляху збігається з позицією перешкоди, вартість цієї хромосоми різко збільшується, що значно зменшує її ймовірність бути обраною для наступного покоління. ГА можуть ефективно планувати шлях як зі статичними, так і з динамічними перешкодами. Перевагами ГА є здатність знаходити оптимальні шляхи у складних, динамічних середовищах з непередбачуваними перешкодами та гнучкість у визначенні функції фітнесу для врахування різних критеріїв (довжина, плавність, безпека). Проте, їх недоліками є повільна збіжність, схильність до локальних оптимумів, невизначена спрямованість мутації. Стохастична природа може призвести до непослідовних результатів, що вимагає багаторазових запусків.

2.3.2 Алгоритм заливного поля (потенційні поля)

Метод потенційних полів передбачає рух мобільного пристрою вздовж ліній векторного поля [15]. Ціль створює притягуюче потенційне поле, а перешкоди – відштовхуюче. Сила відштовхування зменшується зі збільшенням відстані до перешкоди. Сума цих векторів (притягування до цілі та відштовхування від перешкод) визначає напрямок руху робота.

Перешкоди моделюються як джерела відштовхувальних сил, які «виштовхують» робота з небезпечних зон. Для функціонування в динамічному середовищі, сума векторів перераховується через короткі проміжки часу на основі показань сенсорів, що дозволяє роботу реагувати на рухомі перешкоди. Перевагами алгоритму заливного поля є простота реалізації та ефективність у реальному часі для уникнення перешкод. Однак, його основним недоліком є локальність підходу, що часто робить його непридатним для досягнення

комплексних цілей через схильність до потрапляння в локальні мінімуми або зависання в «пастках».

2.3.3 RRT (Rapidly-exploring Random Tree)

Метод RRT (Rapidly-exploring Random Tree) полягає в швидкому дослідженні простору конфігурацій шляхом побудови дерева опорних точок, яке послідовно розширюється від початкової до кінцевої точки [23]. Процес починається зі стартового дерева, що складається лише з початкового пункту. На кожному кроці генерується випадкова точка в області, і до дерева додається новий вузол, з'єднаний з найближчим існуючим вузлом дерева, якщо шлях між ними вільний від перешкод.

Кожен новий вузол та ребро, що додаються до дерева, проходять перевірку на зіткнення з перешкодами. Якщо виявляється зіткнення, точка відкидається або ребро замінюється. RRT ефективний для навігації в складних, багатовимірних просторах з перешкодами, оскільки він не вимагає попереднього обчислення всього простору конфігурацій. Він може бути використаний для перепланування в динамічних середовищах, коли перешкоди блокують бажаний шлях. Перевагами RRT є висока ефективність для задач з великою кількістю вимірів, нелінійними динаміками або диференціальними обмеженнями. Він не потребує попереднього обчислення простору конфігурацій та здатний швидко знаходити шлях у складних середовищах. Проте, він не гарантує оптимальності шляху і може бути неефективним у складних середовищах, таких як сади, через рівномірну випадкову вибірку, яка може сповільнювати збіжність.

2.4 Біонатхненні алгоритми

Біонатхненні алгоритми черпають натхнення з природних процесів та колективної поведінки організмів для вирішення оптимізаційних задач, включаючи планування шляху.

2.4.1 Алгоритми ройового інтелекту

2.4.1.1 Мурашиний алгоритм (ACO)

Мурашиний алгоритм (ACO) імітує поведінку мурах, що шукають їжу. Мурахи залишають феромонні сліди на шляхах, які вони проходять, і інші мурахи з більшою ймовірністю слідують шляхами з вищою концентрацією феромонів [24]. З часом феромони випаровуються, дозволяючи алгоритму адаптуватися до змін у середовищі [25].

У плануванні шляху, «мурахи» досліджують простір, залишаючи «феромони» на пройдених шляхах. Шляхи, що уникають перешкод і є коротшими, отримують більше феромонів, приваблюючи більше «мурах». Покращені версії ACO використовують такі модифікації, як покращена евристична функція (включення штучного потенційного поля для притягання до цілі), адаптивні правила оновлення феромонів та метод обрізки трикутників для усунення зайвих точок повороту, що робить шляхи плавнішими та ефективнішими. Перевагами ACO є ефективність для пошуку оптимальних шляхів у складних графах та здатність адаптуватися до змін у середовищі. Проте, його недоліками є повільна швидкість збіжності, схильність до потрапляння в локальні оптимуми та надмірна кількість точок повороту в традиційній реалізації.

2.4.1.2 Алгоритм рою часток (PSO)

Алгоритм рою часток (PSO) – це оптимізаційний метод, натхненний соціальною поведінкою рою птахів або косяків риб [26]. Кожна «частка» (потенційне рішення) переміщується в просторі пошуку, коригуючи свою швидкість та позицію на основі власного найкращого досвіду (pbest) та найкращого досвіду всього рою (gbest).

У плануванні шляху, частки представляють можливі траєкторії. Якщо позиція частки знаходиться всередині перешкоди, цей шлях відкидається. Перешкоди можуть бути змодельовані як розширені області (збільшені на радіус робота), щоб забезпечити безпечний прохід.PSO використовує функцію фітнесу,

яка мінімізує довжину шляху та час подорожі, уникаючи зіткнень. Перевагами PSO є простота, легкість реалізації, надійність, швидка збіжність. Він ефективний для пошуку глобальних оптимумів у нелінійних задачах. Проте, може застрягти в локальних оптимумах у дуже складних ландшафтах фітнесу.

2.4.1.3 Алгоритм бджолиної колонії (ABC)

Алгоритм бджолиної колонії (ABC) – це потужна техніка оптимізації, натхненна поведінкою медоносних бджіл під час пошуку їжі [27]. Він використовує популяційний підхід, де штучні бджоли (розвідники, робітники, спостерігачі) співпрацюють для пошуку оптимальних рішень у просторі пошуку, балансуючи між дослідженням (exploration) та експлуатацією (exploitation).

У плануванні шляху, бджоли шукають «джерела їжі» (потенційні шляхи), оцінюючи їх «якість». Шляхи, що уникають перешкод та є коротшими/ефективнішими, розглядаються як кращі джерела їжі. ABC може оптимізувати траєкторії роботів у складних середовищах з перешкодами, знаходячи найкоротші або найефективніші шляхи, уникаючи зіткнень. Він також може працювати в динамічних середовищах з рухомими перешкодами. Перевагами ABC є ефективність для складних оптимізаційних задач та здатність балансувати дослідження та експлуатацію. Він застосовується для багатоцільового планування шляху. Однак, може мати слабку здатність до експлуатації та низьку точність рішення в деяких випадках, хоча модифікації, такі як FOABC, спрямовані на їх подолання.

2.4.1.4 Алгоритм сірих вовків (GWO)

Алгоритм сірих вовків (GWO) – це алгоритм ройового інтелекту, що імітує механізм полювання сірих вовків [28]. Він поділяє вовків на ієрархію: альфа (лідер), бета (помічник), дельта (розвідник) та омега (решта зграї). Полювання включає три етапи: оточення, переслідування та напад, де позиції вовків оновлюються на основі позицій альфа, бета та дельта вовків.

У плануванні шляху, «вовки» шукають оптимальну траєкторію. Перешкоди можуть бути розширені для забезпечення безпечної відстані.

Покращені версії GWO, такі як Golden Sine Grey Wolf Optimizer (GSGWO), вирішують проблеми повільної збіжності та потрапляння в локальні оптимуми. GSGWO може адаптуватися до функції перетину перешкод, дозволяючи роботу робити вибір між перетином перешкоди (з певною вартістю) та її уникненням. Перевагами GWO є сильна продуктивність збіжності та відносно проста структура алгоритму. Він здатний оптимізувати довжину шляху, кількість точок повороту та плавність. Проте, схильний до потрапляння в локальні оптимуми, має повільну збіжність на пізніх етапах та проблему з єдиною початковою популяцією в оригінальному GWO.

2.4.2 Інші біомоделі

2.4.2.1 Алгоритм кажана

Алгоритм кажана (BA) – це біонатхненний алгоритм, заснований на ехолокаційних (біо-сонарних) здібностях кажанів. Кажани випромінюють звукові імпульси та слухають ехо, що повертається від перешкод або здобичі, використовуючи цю інформацію для навігації [29]. У алгоритмі «штучні кажани» (потенційні позиції робота) переміщуються в просторі пошуку, оновлюючи свою частоту, швидкість та позицію на основі глобального найкращого рішення.

У плануванні шляху, алгоритм кажана шукає найкоротший шлях, уникаючи динамічних перешкод. Модифіковані версії, такі як Modified Frequency Bat (MFB) algorithm, балансують дослідження та експлуатацію. Обробка перешкод відбувається за допомогою віртуальних сенсорів, які виявляють перешкоди, та процедури уникнення перешкод, яка використовує концепцію «вектора проміжку» (gap vector) для вибору вільного шляху. Алгоритм може оцінювати траєкторію рухомих перешкод для їх уникнення. Перевагами є ефективність для пошуку найкоротшого шляху в динамічних середовищах з рухомими перешкодами та здатність адаптуватися до змін у середовищі. Проте, оригінальний BA може застрягти в локальних оптимумах.

2.4.2.2 Алгоритм зозулі (CS)

Алгоритм зозулі (CS) – це метаевристичний алгоритм, натхненний стратегією гніздового паразитизму зозуль [30]. Кожна «зозуля» (потенційне рішення) відкладає яйце (пропонує рішення) у випадково вибране гніздо господаря. Рішення покращуються за допомогою механізму Levy flight для глобального пошуку та випадкового блукання для локального дослідження.

У плануванні шляху, початкова популяція позицій генерується випадковим чином. Функція оцінки (фітнесу) враховує довжину шляху та його плавність. Перешкоди моделюються у сітковому середовищі як непрохідні області. Хоча прямих механізмів активного уникнення перешкод у базовому CS не описано, функція фітнесу неявно відштовхує алгоритм від перешкод, штрафуючи шляхи, які їх перетинають. Перевагами є проста структура, мінімальна кількість параметрів, легкість реалізації та надійність. Він ефективний для складних оптимізаційних задач. Проте, має недостатнє покриття простору пошуку, передчасну збіжність, низьку точність пошуку та повільну швидкість пошуку в традиційному CS.

2.5 Ймовірнісні алгоритми

Імовірнісні алгоритми, зокрема методи на основі вибірки, ефективні для планування шляху у високовимірних просторах конфігурацій, де побудова явного графа є непрактичною.

2.5.1 PRM (Probabilistic Roadmap)

PRM (Probabilistic Roadmap) – це метод, що описує зв’язність вільного простору для переміщення за допомогою графів [15]. Він працює шляхом випадкової вибірки точок у просторі конфігурацій робота та з’єднання їх ребрами, якщо пряма лінія між ними вільна від перешкод. Таким чином, будується «дорожня карта» (граф) доступних шляхів.

Перешкоди враховуються на етапі побудови дорожньої карти, коли ребра додаються до графа лише в тому випадку, якщо вони не перетинаються з перешкодами. Це забезпечує, що всі шляхи, знайдені на дорожній карті, є безперешкодними. Метод покладається на повноту інформації про навколишнє середовище, що створює труднощі при його застосуванні в умовах динамічного середовища. Перевагами PRM є ефективність для планування шляху у високовимірних просторах, де явне представлення перешкод є складним, та здатність знаходити шлях у складних середовищах. Проте, він може бути неефективним у динамічних середовищах, оскільки дорожня карта повинна бути перебудована при зміні перешкод, і не гарантує оптимальності шляху.

2.5.2 RRT* (Optimal RRT)

RRT* (Optimal RRT) є покращеною версією алгоритму RRT, яка, на відміну від базового RRT, гарантує оптимальність шляху при достатній кількості зразків [31, 32]. RRT* працює за принципом RRT, розширюючи дерево від початкової точки шляхом випадкової вибірки точок. Однак, RRT* додає два ключові кроки: перепідключення (rewiring) та вибір найкращого батька (best parent selection). При додаванні нового вузла, RRT* перевіряє, чи можна досягти сусідніх вузлів через новий вузол з меншою вартістю, і якщо так, перепідключає їх.

Як і RRT, RRT* виконує перевірку на зіткнення для кожного нового вузла та ребра, забезпечуючи, що побудований шлях є вільним від перешкод. Перепідключення також враховує перешкоди, забезпечуючи, що нові, оптимізовані зв’язки також є безперешкодними. Перевагами є гарантована оптимальність шляху при достатній кількості зразків, а також зменшення довжини шляху та покращення плавності порівняно з RRT. Проте, RRT* має вищі обчислювальні витрати порівняно з базовим RRT через додаткові кроки оптимізації і може бути неефективним у дуже складних середовищах, де випадкова вибірка може сповільнювати збіжність.

Сила RRT полягає в його «швидкому дослідженні» та здатності швидко знаходити будь-який здійсненний шлях у високовимірних просторах, де інші

методи зазнають невдачі. Це вирішує проблему «повноти» (знаходження шляху). Після того, як шлях знайдено, наступним логічним кроком є його оптимізація. RRT* додає крок «перепідключення», який ітеративно покращує оптимальність шляху. Це передбачає двоетапний підхід до вирішення проблеми: спочатку знайти рішення (здійсненність), потім його оптимізувати.

2.6. Гібридні алгоритми

Гібридні алгоритми поєднують сильні сторони декількох підходів для подолання їх індивідуальних обмежень, створюючи більш надійні та ефективні рішення.

A* + потенційні поля. Цей гібридний підхід поєднує глобальні пошукові можливості алгоритму A* з реактивними можливостями уникнення перешкод, які надають потенційні поля [16, 33, 34]. A* використовується для глобального планування шляху, забезпечуючи оптимальний або майже оптимальний маршрут від початку до кінця. Потенційні поля, зі своїми силами притягання до цілі та відштовхування від перешкод, застосовуються для локального уникнення перешкод у реальному часі та згладжування траєкторії.

Гібридні алгоритми, такі як MAAPF (Modified A* and Artificial Potential Field), можуть ефективно працювати в динамічних середовищах. A* планує глобальний шлях, а коли робот виявляє динамічні перешкоди, система переходить до локального планування, використовуючи потенційні поля та прогнозування траєкторії перешкод. Це дозволяє роботу адаптуватися до непередбачених змін у середовищі та уникати зіткнень у реальному часі. Перевагами є подолання обмежень окремих алгоритмів: A* допомагає уникнути локальних мінімумів потенційних полів, а потенційні поля забезпечують плавність та реактивність, яких бракує традиційному A*. Це генерує безпечніші, плавніші та більш реалістичні шляхи. Проте, недоліками є складність інтеграції та налаштування, оскільки необхідно правильно збалансувати вплив

глобального планувальника та локального контролера. Гібридний A* може мати довший час виконання у порівнянні з базовим A*.

Поширеність гібридних алгоритмів, особливо тих, що поєднують глобальний пошук (як A*) з локальними реактивними методами (як потенційні поля або DWA), свідчить про перехід до ієрархічних та надійних архітектур планування шляху, які враховують як довгострокову оптимальність, так і адаптивність у реальному часі.

Потенційні поля або підхід динамічного вікна (DWA) відмінно підходять для локального, реактивного уникнення в реальному часі та згладжування шляху, але страждають від локальних мінімумів та відсутності глобальної обізнаності. Їхнє поєднання створює ієрархічну систему, де A* надає «макро» план, а реактивний метод обробляє «мікро» коригування та уникнення перешкод. Ця архітектурна тенденція є вирішальною для автономних систем, що працюють у складних, непередбачуваних реальних середовищах (наприклад, безпілотні автомобілі, сільськогосподарські роботи, сервісні роботи). Вона свідчить про те, що майбутні рішення для планування шляху все частіше включатимуть складні багатошарові системи, які інтегрують різноманітні алгоритмічні парадигми.

Висновок до розділу 2

Для подальшого дослідження обрано одинадцять алгоритмів з шести категорій, що забезпечує повне охоплення сучасних підходів до планування шляху в середовищах з перешкодами. Кожен алгоритм був включений з урахуванням специфічних переваг, які мають значення для подальшої програмної реалізації та порівняльного аналізу ефективності.

A* і алгоритм Дейкстри представляють класичні підходи на графах. A* обрано через його здатність працювати з евристиками, що дає змогу зменшити обчислювальні витрати, зберігаючи оптимальність. Дейкстру використано як еталонний метод для гарантії точності в умовах фіксованих ваг. Обидва

алгоритми є добре формалізованими, що полегшує їх програмну реалізацію в системах зі статичним середовищем.

D* Lite і Theta* демонструють евристичний підхід до задач у динамічних середовищах. D* Lite дозволяє адаптувати маршрут у реальному часі без повного перерахунку, що є ключовим для роботів, які взаємодіють із рухомими об’єктами. Theta* формує більш гладкі траєкторії, що важливо для фізично реалізованих роботів. Обидва методи придатні до інтеграції в робототехнічні системи, де потрібна швидка реакція на зміну оточення.

Генетичний алгоритм і RRT репрезентують метаевристичні стратегії, де ціль – обхід складних або великорозмірних просторових конфігурацій. ГА дозволяє досліджувати простори з багатьма локальними мінімумами, а RRT забезпечує швидке охоплення великих областей. Їхня гнучка структура робить їх придатними до модифікацій для конкретних задач і середовищ.

ACO і GWO як біонатхненні методи показують ефективність у задачах з множинними маршрутами та змінними умовами. ACO моделює розподілену оптимізацію через феромони, а GWO – ієрархічну стратегію пошуку, яка спрощує реалізацію. Обидва алгоритми добре масштабуються і можуть адаптуватися до змін топології середовища.

PRM і RRT* – ймовірнісні алгоритми, що дозволяють працювати у високовимірних просторах. PRM будує граф можливих шляхів, ефективний для багаторазового використання в одному середовищі. RRT* покращує шлях з часом, що цінно при обмеженнях на ресурси. Ці алгоритми легко комбінуються з іншими методами для досягнення балансу між швидкістю і якістю.

Гібридний алгоритм, який поєднує A* з методом потенційних полів, забезпечує як глобальне планування, так і локальне уникнення перешкод. Така комбінація дозволяє створювати адаптивні маршрути з урахуванням нових перешкод у режимі реального часу.

Кожен із розглянутих алгоритмів має окремі властивості, які важливі для різних класів задач планування.

РОЗДІЛ 3
ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПОШУКУ ШЛЯХУ В ДИСКРЕТНОМУ СЕРЕДОВИЩІ З ПЕРЕШКОДАМИ

3.1 Побудова сіткової карти з перешкодами для задач маршрутизації

У задачах автоматизованого планування руху, навігації автономних агентів та пошуку оптимального маршруту ключову роль відіграє адекватне моделювання навколишнього середовища. Одним із найбільш поширених підходів є представлення простору у вигляді регулярної сітки, де кожна клітина описує окрему ділянку простору, що може бути або вільною, або зайнятою перешкодою. Така дискретизація дозволяє ефективно застосовувати алгоритми пошуку шляху та контролю пересування. Сіткова модель дозволяє формалізувати задачу маршрутизації .

Моделі сітки призначена для створення двовимірного представлення середовища з випадково згенерованими перешкодами та подальшої її візуалізації. Вона є базовим інструментом для досліджень у сфері мобільної робототехніки, штучного інтелекту в ігрових середовищах, а також при розробці систем планування траєкторій у складних умовах.

Даний код реалізує побудову прямокутної сітки з випадковими перешкодами та її візуалізацію із позначенням початкової та кінцевої точок. Код складається з двох основних функціональних компонентів: генерація сітки з перешкодами та графічне представлення цієї сітки.

Функція create_grid_map відповідає за створення двовимірної матриці, яка моделює карту. Вхідними параметрами є розмір сітки, відсоток клітин, які повинні містити перешкоди, та початкове значення для генератора випадкових чисел. У межах функції ініціалізується сітка як матриця нулів, після чого обчислюється кількість перешкод, що повинні бути розміщені. Позиції для перешкод вибираються випадковим чином без повторень. Обрані індекси трансформуються у двовимірні координати, відповідно до розміру сітки, та

відповідні елементи масиву встановлюються в одиницю, що позначає перешкоду.

Функція visualize_map реалізує графічне представлення створеної сітки з використанням бібліотеки matplotlib. Вхідним параметром є сітка, а також, опціонально, маршрут у вигляді послідовності координат, початкова точка та точка призначення. Візуалізація здійснюється шляхом відображення значень елементів матриці у градаціях сірого кольору, де перешкоди мають чорне забарвлення. Якщо передано маршрут, відповідні клітини маршруту позначаються як проміжні значення (наприклад, 0,5), а також на графіку відображається лінія, що з'єднує ці координати. Початкова та кінцева точки позначаються маркерами відповідно зеленого та червоного кольору. Заголовок, сітка координат і легенда додаються для зручності інтерпретації результату. Після побудови графік виводиться на екран.

У завершальній частині коду здійснюється виклик функції create_grid_map без передачі параметрів, що призводить до створення сітки розміром 20×20 з 20 % заповненістю перешкодами та фіксованим генератором випадкових чисел. Задаються початкова та кінцева координати, після чого функція visualize_map будує графічне зображення сітки з відповідними маркерами (рис. 3.1).

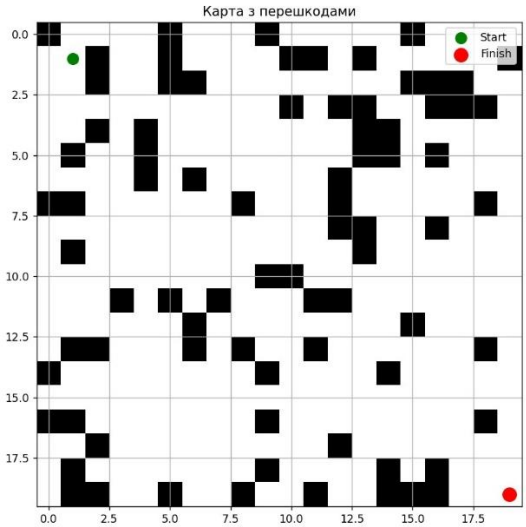

Рисунок 3.1 – Графічне представлення створеної сітки

Повний код генерації та візуалізації сітки з перешкодами подано в Додатку А.

Для проведення досліджень алгоритмів пошуку шляху модифіковано код створення сітки з перешкодами.

Код реалізує більш контрольований підхід до генерації перешкод. Якщо початкове значення генератора не задано, воно створюється автоматично на основі поточного часу. Це забезпечує унікальність карти при кожному запуску. Перед розміщенням перешкод виключаються індекси, які відповідають початковій та кінцевій точкам руху робота. Таким чином, ці клітини завжди залишаються вільними. Також змінено підхід до вибору індексів: усі клітини представлені у вигляді послідовних індексів, з яких вилучаються заборонені значення, після чого відбирається необхідна кількість випадкових позицій.

Основна відмінність полягає у тому, що другий варіант гарантує придатність згенерованої карти для побудови маршруту між фіксованими точками та підтримує варіативність під час кожного запуску.

Скрипт представлено в Додатку Б.

3.2 Дослідження ефективності алгоритмів пошуку шляху в дискретному середовищі з перешкодами

Це дослідження зосереджено на порівняльному аналізі ефективності різних алгоритмів пошуку шляху в дискретних середовищах, що містять перешкоди. Обрані алгоритми представляють різноманітні підходи до розв’язання цієї задачі в галузі робототехніки. Серед них – A* та алгоритм Дейкстри, D* Lite та Theta*, генетичний алгоритм та RRT (Rapidly-exploring Random Tree), ACO (Ant Colony Optimization) та GWO (Grey Wolf Optimizer), PRM (Probabilistic RoadMap) та RRT*, а також A* з методом потенційних полів.

Дослідження проводиться за стандартизованим підходом. Перший етап полягає у генерації двовимірних сіток заданого розміру. У цих сітках випадковим

чином розміщуються перешкоди. При цьому забезпечується контроль над співвідношенням перешкод до загальної площі сітки. Важливою умовою є те, що стартова (0,0) та цільова (n-1,n-1) клітини завжди залишаються вільними, гарантуючи можливість існування шляху.

Отримані дані, тобто згенерована карта з перешкодами, стартова та цільова точки, є вхідними для кожного з реалізованих алгоритмів. Після запуску кожного алгоритму розраховуються три ключові метрики для оцінки якості знайденого шляху та ефективності самого алгоритму. Ці метрики включають час виконання, який відображає обчислювальні витрати алгоритму; довжину шляху, що вказує на оптимальність знайденої траєкторії; та гладкість шляху, яка оцінює кількість поворотів або різких змін напрямку вздовж знайденого шляху.

Для забезпечення статистично значущих даних кожен алгоритм протестовано 100 разів на різних випадково згенерованих картах з однаковими параметрами. Такий підхід дозволяє зібрати репрезентативну статистику ефективності алгоритмів в різних умовах. Після завершення всіх прогонів обчислено середні значення всіх визначених метрик, але лише для тих випадків, коли алгоритм успішно знайшов шлях. Це забезпечує надійне порівняння продуктивності алгоритмів, що дозволяє зробити висновки про їхню ефективність в дискретних середовищах з перешкодами.

Приклад програмної реалізації приведений в Додатку В.

Оптимальний вибір алгоритму маршрутизації часто вимагає врахування одночасно кількох критеріїв (час виконання, довжина та плавність траєкторії). У цьому дослідженні порівнюються одинадцять алгоритмів (A*, алгоритм Дейкстри, Theta*, D* Lite, A* з потенційними полями, PRM, RRT*, ACO, GWO, GA, RTT) за трьома показниками: час виконання (мс), довжина шляху та гладкість (кількість поворотів). Основна увага приділяється швидкості та плавності траєкторії.

Результати робити алгоритмів Дейкстри; A*; D* Lite; Theta*; GA; RTT; ACO; GWO; PRM; RRT*; алгоритму A* + потенційні поля наведені в Додатку

Г1-Г11. Зведені результати за часом виконання алгоритмів; за довжиною отриманого шляху і за гладкістю (кількість поворотів) наведені в Додатку Г12-Г14.

Для порівняння критеріїв спочатку нормалізовано значення часу виконання та показника гладкості (повороти) за принципом min-max. Цей підхід забезпечує перетворення вихідних даних у єдину шкалу. Далі для кожного алгоритму обчислено зважені суми нормалізованих (60 % час + 40 % гладкість). Спочатку проведена нормалізація приведення значень часу та гладкості до інтервалу [0,1] за формулою max-min (формула 3.1).

$$score_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}, \tag{3.1}$$

де x – вихідне значення показника (час або гладкість),

$x_{\min}$, $x_{\max}$ – відповідно мінімальне і максимальне значення серед усіх експериментів.

Далі зважения – розрахунок інтегрального балу. Інтегральна оцінка ефективності маршруту обчислюється за формулою 3.2:

$$score = \omega_{time} \cdot score_{time} + \omega_{smooth} \cdot score_{smooth}, \tag{3.2}$$

де $score_{time}$ – нормалізоване значення часу,

$score_{smooth}$ – нормалізоване значення гладкості,

ω_{time}, ω_{smooth} – вагові коефіцієнти, що задавольняють умову:

$$\omega_{time} + \omega_{smooth} = 1.$$

В кінці порівняння результатів – ранжування алгоритмів за отриманим інтегральним балом у кожному випадку ваг.

Вибір вагових коефіцієнтів із пріоритетом 60 % на час виконання та 40 % на гладкість у контексті автоматизації планування руху в середовищах з перешкодами обґрунтовано з кількох аспектів.

По-перше, час виконання алгоритму безпосередньо впливає на оперативність прийняття рішень у системах навігації та планування траєкторії. У динамічних та змінних середовищах, де перешкоди можуть з’являтися або змінюватися в реальному часі, швидкість обчислень є критичною для запобігання зіткненням і забезпечення безпеки руху. Відповідно, більша вага часу дозволяє віддавати перевагу алгоритмам, що здатні генерувати рішення максимально швидко.

По-друге, гладкість траєкторії – це показник якості руху, що визначає кількість поворотів або різких змін напрямку. Занадто великий показник гладкості може призвести до різких маневрів, що негативно впливають на енергоспоживання, знос механізмів, а також безпеку руху автономних систем. Проте, на відміну від часу, гладкість не завжди є критичним фактором у найпершу чергу, особливо якщо забезпечується прийнятний рівень плавності руху.

Поєднання 60 % на час і 40 % на гладкість дозволяє збалансувати потребу в швидкості прийняття рішення із якістю траєкторії, що забезпечує ефективність та безпеку руху в складних умовах середовища з перешкодами. Цей баланс є ключовим для практичних застосувань, зокрема в автономних роботах та транспортних системах, де затримки у плануванні можуть мати критичні наслідки.

Отже, вибір таких вагових коефіцієнтів відображає реальні технічні вимоги задачі автоматизації планування руху, забезпечуючи пріоритетність оперативності без значного погіршення якості траєкторії.

Нормалізовані значення часу виконання та гладкості наведено в таблиці 3.1.

Таблиця 3.1 – Нормалізовані значення часу виконання та гладкості

Алгоритм	Час (норм.)	Гладкість (норм.)
A* + потенційні поля	0,00034	0,0979
PRM	0,0429	0,1117
RRT*	0,1442	0,0529
ACO	1,0000	0,0443
GWO	0,6831	0,0000
GA	0,1947	0,06879
RTT	0,00155	1,0000
D* Lite	0,00016	0,2383
Theta*	0,00003	0,7842
Дейкстри	0,00000	0,7414
A*	0,00000	0,7459

Результати розрахунку інтегральних балів наведено в таблиці 3.2.

Таблиця 3.2 – Результати розрахунку інтегральних балів

Алгоритм	60 % час + 40 % гладкість
A* + потенційні поля	0,0403
PRM	0,0721
RRT*	0,1115
ACO	0,6177
GWO	0,4098
GA	0,3947
RTT	0,4009
D* Lite	0,0956
Theta*	0,3137
Дейкстри	0,2965
A*	0,2983

Висновок до розділу 3

Проаналізовано ефективність 11 алгоритмів пошуку шляху (A, Дейкстри, Theta, D* Lite, A* з потенційними полями, PRM, RRT*, ACO, GWO, GA, RTT) на сіткових картах з перешкодами. Критерії оцінки: час виконання, довжина та гладкість шляху.

Для досягнення мети розроблена гнучка модель сіткової карти, що дозволяє генерувати середовища з випадково розподіленими перешкодами, при цьому гарантуючи прохідність початкової та кінцевої точок.

Проведені 100 ітерацій тестування для кожного алгоритму на унікальних, випадково згенерованих картах дозволили зібрати репрезентативну статистику. Аналіз цих даних показав значні відмінності у продуктивності алгоритмів за кожною з метрик. Зокрема, традиційні алгоритми пошуку шляху (Дейкстри, A*) продемонстрували високу швидкість виконання, але не завжди оптимальну гладкість шляху. Алгоритми, засновані на біологічних принципах та еволюційних обчисленнях (ACO, GWO, GA), хоча і здатні знаходити дуже короткі та гладкі шляхи, виявилися значно повільнішими за інші. Алгоритми на основі вибірки (RRT, RRT*, PRM) демонстрували компроміс між швидкістю та якістю шляху.

Основним аспектом дослідження стало впровадження інтегральної оцінки, що дозволила врахувати одночасно декілька критеріїв ефективності. Застосування нормалізації min-max та розрахунок зваженої суми нормалізованих значень (60 % на час виконання та 40 % на гладкість) дозволили отримати об’єктивну картину. Вибір саме таких вагових коефіцієнтів обґрунтований вимогами до практичних застосувань у робототехніці та автономних системах навігації. У цих сценаріях оперативність прийняття рішень (швидкість) є критично важливою для безпеки та запобігання зіткненням у динамічних середовищах. Водночас плавність траєкторії безпосередньо впливає на енергоспоживання, знос механізмів та загальну якість руху. Такий баланс

дозволяє віддавати перевагу алгоритмам, які забезпечують швидке генерування прийнятних за якістю траєкторій.

Згідно з інтегральною оцінкою за заданими ваговими коефіцієнтами, алгоритм A* з потенційними полями виявився найбільш збалансованим та ефективним. Він успішно поєднує відносно швидке виконання з високою якістю знайденого шляху, що робить його привабливим для використання в реальних системах. Це підтверджує, що модифікації класичних алгоритмів можуть значно підвищувати їхню практичну цінність.

РОЗДІЛ 4
ПРОГРАМНЕ ТА АПАРАТНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОНОМНОЇ НАВІГАЦІЇ В СЕРЕДОВИЩАХ З ПЕРЕШКОДАМИ

4.1. Програмне забезпечення для автономної навігації

Розробка програмного забезпечення для автономної навігації в невідомих середовищах є складним процесом, що вимагає інтеграції різноманітних компонентів. Цей процес охоплює вибір відповідних мов програмування, використання спеціалізованих бібліотек та фреймворків, а також створення продуманої архітектури системи.

4.1.1 Мови програмування та бібліотеки

Вибір мов програмування та бібліотек має важливе значення для забезпечення продуктивності та гнучкості системи автономної навігації.

Мова програмування C++ часто вважається основною мовою для робототехніки та залишається однією з провідних завдяки своїй максимальній продуктивності та високому рівню контролю. C++ компілюється безпосередньо в машинний код, що забезпечує високу швидкість виконання, яка є критично важливою, коли необхідно реагувати на мікросекундному рівні. Також C++ надає низькорівневе управління пам’яттю, що дозволяє ефективно оптимізувати розподіл ресурсів робота.

Python є дуже популярним завдяки своїй простоті та зручності використання, що робить його чудовим вибором у робототехніці. Ця мова має розвинену екосистему бібліотек та фреймворків для робототехніки та штучного інтелекту, зокрема NumPy та SciPy для наукових обчислень, а також OpenCV для комп’ютерного зору. Python добре підтримується в екосистемі ROS, де багато програмних пакетів написано саме цією мовою, і часто використовується для автоматизації та створення сценаріїв. Хоча Python не такий швидкий, як C++, він ідеально підходить для розробки проектів.

MATLAB є цінним інструментом для математичних обчислень, пропонуючи безліч вбудованих функцій для робототехніки, комп'ютерного зору та обробки сигналів . Його спеціалізовані набори інструментів (toolboxes) для робототехніки, обробки зображень та нейронних мереж надають перевірені функції, що дозволяє уникнути повторної розробки вже існуючих рішень. Однак, MATLAB має значну криву навчання, а його синтаксис відрізняється від Python або C++ .

Java забезпечує крос-платформну універсальність та широку підтримку. Хоча вона є більш багатослівною та дещо повільнішою, ніж C++, Java підходить для розробки великих проектів.

Серед ключових бібліотек та фреймворків варто виділити ROS (Robot Operating System). Це стандарт у спільноті робототехніки, що надає набір програмних бібліотек та інструментів, які спрощують створення складних та надійних поведінок роботів. ROS забезпечує модульність, що дозволяє легко налаштовувати та масштабувати його для різних застосувань. Він підтримує кілька мов програмування, включаючи C++ та Python.

OpenCV (Open Source Computer Vision Library) – це комплексна бібліотека для комп'ютерного зору та обробки зображень. Вона надає понад 2500 оптимізованих алгоритмів для завдань, що виконуються в реальному часі, таких як обробка зображень, виявлення об'єктів, розпізнавання облич та відстеження об'єктів. OpenCV підтримує C++, Python та Java. Вона може інтегруватися з фреймворками глибокого навчання, такими як TensorFlow та PyTorch, для складніших завдань, наприклад, для навчання моделей виявлення об'єктів, тоді як OpenCV виконує попередню обробку зображень та вилучення ознак.

PCL (Point Cloud Library) – це потужний відкритий фреймворк для обробки 2D/3D зображень та даних хмар точок. PCL містить понад тисячу функцій, розподілених за модулями для фільтрації (видалення шуму та викидів), сегментації (розділення хмари точок на окремі об'єкти), оцінки ознак (вилучення значущих характеристик), реєстрації (вирівнювання кількох хмар точок) та

візуалізації. Він є незамінним для автономних транспортних засобів та робототехніки, зокрема для SLAM.

Eigen – це популярна відкрита бібліотека шаблонів C++ для лінійної алгебри, що надає високоефективні реалізації загальних математичних операцій з векторами, матрицями та тензорами. Eigen використовується в ROS та TensorFlow, що підкреслює його важливість для кінематики, динаміки та керування роботами. Його оптимізації продуктивності включають використання шаблонних виразів та векторизацію для SIMD-рушіїв (SSE, AVX, ARM NEON).

TensorFlow та PyTorch є провідними фреймворками штучного інтелекту для розробки додатків комп'ютерного зору та глибокого навчання. TensorFlow, розроблений Google, відомий своєю масштабованістю та готовністю до виробництва, підходить для великомасштабних проектів ШІ. Він використовує графи потоку даних, де вузли представляють математичні операції, а зв'язки – тензори. PyTorch, створений Meta, зазвичай використовується для досліджень та прототипування завдяки своїй гнучкості та простоті використання. Він відрізняється динамічним обчислювальним графом, що спрощує експериментування та відлагодження. Обидва фреймворки є життєво важливими для побудови інтелектуальних моделей, що поєднують глибоке навчання та комп'ютерний зір для широкого спектру застосувань, включаючи автономну навігацію.

4.1.2 Архітектура програмного забезпечення та фреймворки

Типова архітектура програмного забезпечення для систем автономної навігації, особливо тих, що використовують ROS, інтегрує кілька ключових компонентів для забезпечення ефективної та надійної навігації роботів у динамічних та неструктурованих середовищах. Цей модульний підхід максимально використовує можливості ROS та вирішує конкретні виклики автономної навігації.

Моделювання робота є початковим етапом, що передбачає створення точного цифрового представлення робота. Це включає його фізичні розміри, а

також кінематичні (рух без урахування сил) та динамічні (рух з урахуванням сил) описи. URDF (Unified Robot Description Format) – це XML-формат, що використовується для стандартизованого опису структури робота в ROS. RViz є 3D-інструментом візуалізації в ROS, що використовується для перевірки розробленої моделі робота, забезпечуючи її точну відповідність фізичному прототипу.

Для належного руху робот повинен сприймати та розуміти своє оточення. Картографування середовища передбачає побудову карти робочого простору та одночасне визначення місцезнаходження робота на цій карті. SLAM (Simultaneous Localization and Mapping) є ключовою технікою, що дозволяє роботу створювати карту невідомого середовища, одночасно визначаючи своє місцезнаходження. Дані для картографування збираються з LiDAR-сенсорів, камер та датчиків глибини. Обробка даних сенсорів у реальному часі здійснюється за допомогою ROS-пакетів, таких як GMapping або Cartographer.

Планування шляху включає обчислення оптимального або найкоротшого можливого шляху руху робота від його поточного місцезнаходження до бажаного пункту призначення. Ці алгоритми враховують згенеровану карту, відомі перешкоди та бажані траєкторії для забезпечення плавного та безперешкодного руху. MoveIt! – це потужний ROS-пакет для планування руху, що інтегрує різні алгоритми планування шляху. OMPL (Open Motion Planning Library) часто інтегрується з MoveIt! і надає широкий вибір алгоритмів планування руху на основі вибірки. Поширені алгоритми планування шляху включають RRT, A* та D*.

Глобальні планувальники, такі як Dijkstra та A*, присутні в стеку навігації ROS, тоді як локальні планувальники, такі як DWA та TEB (Timed Elastic Band), оптимізують траєкторію, уникаючи перешкод.

Після планування траєкторії розробляються стратегії керування, щоб забезпечити ефективне, плавне та точне виконання роботом запланованого шляху. Вони також дозволяють роботу відхилятися від запланованої траєкторії

при виникненні динамічних перешкод або змін у середовищі. PID-контролери (Proportional-Integral-Derivative Controllers) широко використовуються для керування швидкістю та кутами повороту робота. ROS Control надає фреймворк для реалізації та керування механізмами керування, такими як PID-контролери, забезпечуючи чутливу та адаптивну навігацію.

Механізми виконання перетворюють високорівневі плани шляху в низькорівневі виконувані команди для двигунів та актуаторів робота. ROS Nodes керують зв’язком та синхронізацією даних сенсорів, команд керування та результатів планування. Gazebo – це віртуальне контрольоване середовище, що використовується для симуляції, де механізми виконання можуть бути ретельно протестовані перед розгортанням у реальному світі.

Весь фреймворк проходить інтенсивне симулювання та тестування в Gazebo. Цей етап передбачає тестування різних сценаріїв, включаючи різні типи перешкод та динамічні зміни середовища. Це дозволяє виявляти та усувати потенційні проблеми на етапі симуляції, значно підвищуючи надійність системи для реальних застосувань.

Останній і вирішальний крок передбачає оцінку в реальному світі. Тестування робота в реальних фізичних середовищах проводиться в різноманітних умовах, таких як промислові склади або відкриті простори. Це дозволяє визначити, чи може робот ефективно навігувати та реагувати на динамічні та складні умови реального світу.

4.2. Апаратна реалізація

Апаратна реалізація автономної навігації є не менш важливою, ніж програмна, оскільки вона забезпечує фізичні можливості для сприйняття, обробки та руху. Ефективна апаратна платформа є основою для успішної роботи алгоритмів у невідомих середовищах.

4.2.1 Сенсори для сприйняття середовища

Сенсори є «очима» та «вухами» автономної системи, надаючи дані, необхідні для розуміння навколишнього середовища. Їх можна розділити на екстероцептивні (зовнішні) та пропріоцептивні (внутрішні) сенсори.

Екстероцептивні сенсори (зовнішні).

LiDAR-сенсори (Light Detection and Ranging) використовують лазерні імпульси для вимірювання точної відстані до об’єктів, створюючи високоточні 3D-хмари точок [35, 36]. LiDAR чудово працює в умовах низького освітлення, туману або пилу, оскільки не залежить від зовнішнього освітлення. Існують механічні LiDAR-системи, які мають обертові дзеркала, забезпечують 360-градусний огляд, але є дорогими та громіздкими. Натомість, твердотільні LiDAR-системи не мають рухомих частин, є компактнішими, дешевшими та довговічнішими. Перевагами LiDAR є висока точність 3D-картографування та надійне виявлення та класифікація об’єктів. Обмеження включають високу вартість, вузьке поле зору для деяких типів, генерацію великих обсягів даних, чутливість до сильного дощу або снігу, які можуть створювати шум. LiDAR не може безпосередньо вимірювати швидкість об’єктів.

Камери надають багату візуальну інформацію, подібну до людського зору, що є критично важливою для розпізнавання об’єктів, розуміння сцени та відстеження смуги руху [35, 37]. Існують різні типи камер. Монокулярні камери мають одне «око», є економічно ефективними, але мають обмежену точність оцінки глибини. Стереокамери, що мають дві камери, імітують бінокулярний зір, дозволяють безпосередньо реконструювати 3D-середовище та покращують оцінку відстані. RGB-камери збирають видиме світло для детальної інформації про колір. Інфрачервоні/термальні камери виявляють теплові сигнатури, працюють в умовах низького освітлення або в певних погодних умовах, наприклад, тумані. Важливими особливостями камер є поле зору (FOV) – ширококутні камери для всебічного огляду, телеоб’єктиви для далеких об’єктів; динамічний діапазон – HDR-сенсори для складних умов освітлення; та типи

затворів – rolling shutter може викликати спотворення при швидкому русі. Камери відіграють роль у сприйнятті (виявлення та розпізнавання об’єктів), локалізації (візуальні підказки доповнюють GPS/IMU) та 3D-реконструкції (SfM, щільний SLAM).

RADAR-сенсори (Radio Detection and Ranging) використовують радіохвилі для виявлення об’єктів та вимірювання їхньої швидкості. Вони не схильні до впливу погодних умов. Розрізняють далекобіжні (до 250 м) та короткобіжні (до 30 м) типи [35, 38]. Перевагами RADAR є надійність в екстремальних погодних умовах (дощ, туман) та здатність вимірювати відносну швидкість об’єктів. Однак їхня роздільна здатність є нижчою порівняно з LiDAR, що обмежує ефективність у складних середовищах.

Ультразвукові сенсори вимірюють відстань за часом прольоту звукової хвилі. Вони часто використовуються для короткодіапазонного виявлення статичних об’єктів.

Пропріоцептивні сенсори (внутрішні).

IMU (Inertial Measurement Unit) складається з акселерометрів (вимірюють лінійне прискорення), гіроскопів (вимірюють кутову швидкість) та іноді магнітометрів (вимірюють магнітне поле Землі для корекції дрейфу) [39, 40]. Принцип його роботи полягає в перетворенні виявленої інерції в вихідні дані, що описують рух об’єкта. Роль IMU полягає у відстеженні орієнтації, швидкості та гравітаційних сил. Він є критично важливим для систем керування польотом, автопілотів та інерційних навігаційних систем. Високоточні IMU можуть мати дрейф менше 0.001 °/год.

Одометри (Wheel Encoders) вимірюють оберти коліс для оцінки пройденої відстані, що є дуже корисним для локалізації та SLAM [35].

4.2.2 Обчислювальні платформи

Ефективність алгоритмів автономної навігації значною мірою залежить від потужності та архітектури обчислювальних платформ.

Вбудовані системи є основою автономних транспортних засобів та роботів, забезпечуючи обробку сенсорних даних, прийняття рішень у реальному часі та можливості керування [41, 42]. Вони обробляють дані з камер, LiDAR та ультразвукових сенсорів для створення карт у реальному часі, локалізації робота та планування оптимальних шляхів. Вбудовані системи реалізують алгоритми уникнення перешкод, планування шляху та керування рухом, дозволяючи роботам самостійно навігувати в складних середовищах. Крім того, ці системи забезпечують безпеку та відмовостійкість, контролюючи параметри системи та реалізуючи стратегії виявлення та відновлення після помилок.

GPU (Graphics Processing Units) є золотим стандартом для великомасштабного навчання ШІ та автономних систем завдяки їхнім можливостям паралельної обробки [43, 44]. Перевагами GPU є швидше навчання моделей та висновок у реальному часі, оскільки вони обробляють величезні набори даних паралельно, значно скорочуючи час навчання моделей (з тижнів до годин) та забезпечуючи миттєвий висновок для таких додатків, як самокеровані автомобілі. Також вони забезпечують економічно ефективну масштабованість, оскільки хмарні GPU усувають витрати на апаратне забезпечення та обслуговування, дозволяючи динамічно масштабувати ресурси за потребою. GPU надають миттєвий доступ до обчислювальних ресурсів корпоративного рівня, дозволяючи стартапам та дослідникам запускати експерименти з глибоким навчанням на потужних GPU без значних початкових інвестицій. Ключові характеристики GPU включають VRAM (Video RAM), що визначає обсяг даних, які GPU може обробляти одночасно (великі моделі вимагають великих обсягів VRAM, наприклад, 400+ ГБ для LLaMA 3 405B); Tensor Cores – спеціалізовані ядра, що прискорюють операції глибокого навчання, такі як множення матриць та згортки, значно підвищуючи швидкість та ефективність навчання; пропускну здатність пам’яті – швидкість передачі даних усередині GPU, де висока пропускна здатність запобігає вузьким місцям при обробці великомасштабних обчислень; та підтримку точності FP8 / FP16 / FP32, що дозволяє

використовувати змішану точність для прискорення та економії пам’яті при збереженні точності. Прикладами платформ є Neousys Nuvo-5095GC та Nuvo-6108GC – промислові GPU-комп’ютери, що використовуються в проектах автономного водіння, таких як Baidu Apollo, завдяки їхній надійності та здатності працювати в екстремальних умовах.

FPGA (Field-Programmable Gate Arrays) є потужною апаратною платформою для реалізації сенсорної фузії в автономній робототехніці завдяки їхнім можливостям паралельної обробки, низькій затримці та енергоефективності [45, 46]. Переваги FPGA включають паралельну обробку, що дозволяє одночасно обробляти великі обсяги даних з різних сенсорів (камер, LiDAR, IMU) у реальному часі. Низька затримка є критично важливою для систем керування в реальному часі, де роботи повинні реагувати миттєво на зміни в середовищі. Настроюване апаратне забезпечення дозволяє розробникам адаптувати апаратну архітектуру до конкретних завдань робототехніки, оптимізуючи продуктивність та енергоспоживання. Енергоефективність FPGA значно вища порівняно з CPU та GPU, що важливо для роботів з живленням від батарей. Вони також підтримують алгоритми ШІ/МН, ефективно обробляючи алгоритми ШІ/машинного навчання, особливо для завдань комп’ютерного зору, таких як розпізнавання об’єктів та планування шляху. Однак, викликом є складність проектування та розробки, що вимагає досвіду в апаратному проектуванні та паралельних обчисленнях.

4.2.3 Техніки злиття сенсорних даних (Sensor Fusion)

Злиття сенсорних даних є ключовою технікою в автономній навігації, що дозволяє об’єднувати інформацію з різних сенсорів для створення більш повного та точного розуміння навколишнього середовища [38, 47]. Це дозволяє компенсувати недоліки окремих сенсорів та підвищити надійність системи.

Призначення злиття сенсорних даних полягає в забезпеченні автономних транспортних засобів комплексним та точним розумінням їхнього оточення, подібно до того, як люди використовують кілька органів чуття. Це має

вирішальне значення для безпечної навігації, виявлення перешкод, визначення місцезнаходження та орієнтації, а також прийняття обґрунтованих рішень. Типи сенсорів, що залучаються до злиття, включають камери, LiDAR, RADAR, ультразвукові сенсори, GPS та IMU.

Розрізняють кілька рівнів злиття сенсорних даних, залежно від абстракції. Низькорівневе злиття безпосередньо об’єднує сирі дані сенсорів. Це мінімізує втрату інформації, але вимагає значних обчислювальних ресурсів і є менш адаптивним до додавання нових сенсорів. Середньорівневе (ознакове) злиття вилучає ключові ознаки з сирих даних сенсорів перед їх злиттям. Це зменшує вимоги до пропускної здатності та є більш адаптивним, зберігаючи при цьому ефективність вилучення релевантних даних. Високорівневе злиття передбачає, що кожен сенсор незалежно обробляє свої дані та виконує своє завдання, а потім відбувається злиття виявлених об’єктів або траєкторій. Цей підхід забезпечує високу модульність та простоту, але призводить до втрати деяких ключових даних нижчих рівнів.

Висновок до розділу 4

Автоматизація планування руху в середовищах з перешкодами потребує комплексного підходу, що поєднує сучасні програмні алгоритми та спеціалізоване апаратне забезпечення.

Програмна частина базується на мовах (C++, Python), бібліотеках (ROS, OpenCV, PCL) та алгоритмах SLAM, планування шляху (A*, RRT) і керування (PID). Апаратна реалізація включає сенсори (LiDAR, камери, RADAR, IMU) та обчислювальні платформи (GPU, FPGA) для обробки даних у реальному часі. Ключову роль відіграє злиття даних сенсорів, що підвищує точність та надійність системи. Сучасні технології дозволяють роботам ефективно навігувати в динамічних середовищах, але залишаються виклики, пов’язані з вартістю, енергоефективністю та адаптацією до складних умов.

Головними викликами залишаються забезпечення енергоефективності, обробка динамічних перешкод та підвищення надійності систем у неструктурованих середовищах. Впровадження таких систем відкриває нові можливості для автономних транспортних засобів, промислових роботів та сервісних платформ, де критично важливими є точність і швидкість реакції.

Найбільш перспективним напрямом є розробка гібридних алгоритмів, які поєднують переваги класичних методів планування шляху з нейромережевими підходами для ефективної роботи в умовах невизначеності. Це дозволить створити більш адаптивні та надійні системи автоматизованого планування руху.

РОЗДІЛ 5
АВТОНОМНЕ ПЛАНУВАННЯ ТРАЄКТОРІЙ У СЕРЕДОВИЩАХ З ПЕРЕШКОДАМИ

5.1 Базові алгоритми та їх програмна реалізація

5.1.1 Алгоритм та програмна реалізація алгоритму Frontier-Based Exploration

Frontier-Based Exploration є методом дослідження невідомого середовища для автономних роботів. Алгоритм використовує концепцію фронтирів – кордонів між дослідженими та недослідженими областями. Основна ідея полягає в ітеративному виборі та досягненні фронтирів для максимального охоплення карти.

Він поступово обирає доступні фронтири та будує маршрут до їх центроїдів, використовуючи алгоритм A* для планування шляху. Кожна клітинка карти може бути вільною, зайнятою перешкодою або вже дослідженою, і відповідно впливає на прийняття рішень. Основні допоміжні функції відповідають за перевірку меж карти, пошук сусідів, визначення фронтирів та кластеризацію їх для вибору цілей. Робот досліджує карту до тих пір, поки всі фронтири або досліджені, або недоступні, що забезпечує ефективне охоплення невідомого середовища.

Функція is_in_bounds перевіряє, чи знаходиться точка в межах карти. Вона використовується для визначення допустимих позицій. Функція get_neighbors повертає сусідні клітинки для заданої позиції, враховуючи межі карти. Ці функції є базовими для роботи з простором.

Функція is_frontier_cell визначає, чи є клітинка фронтиром. Фронтиром вважається вільна клітинка (0), яка межує з хоча б однією дослідженою клітинкою (2). Ця умова дозволяє ідентифікувати кордони невідомих областей. Функція find_frontiers знаходить усі кластери фронтирів на карті за допомогою

пошуку в ширину (BFS). Кожен кластер складається з суміжних клітинок, які відповідають умові фронтиру.

Функція heuristic реалізує манхеттенську відстань для оцінки вартості шляху. Вона використовується в алгоритмі A*. Функція a_star реалізує алгоритм A* для пошуку шляху від поточної позиції до цільової точки. Алгоритм враховує лише досліджені (2) та вільні (0) клітинки, ігноруючи перешкоди (1). Якщо шлях існує, функція повертає послідовність клітинок від старту до цілі.

Функція frontier_centroid обчислює центроїд кластера фронтирів. Вона використовує середнє арифметичне координат усіх клітинок у кластері. Центроїд слугує цільовою точкою для руху робота.

Функція explore є основним алгоритмом дослідження. Вона починає зі стартової позиції, позначає її як досліджену та ітеративно вибирає фронтири для подальшого дослідження. На кожній ітерації алгоритм знаходить усі фронтири, сортує їх за розміром та вибирає найбільший з них, до якого можна дістатися. Для цього використовується A*. Після знаходження шляху робот рухається по ньому, позначаючи відвідані клітинки як досліджені. Процес повторюється, поки не залишиться доступних фронтирів.

Функція create_grid_map генерує випадкову карту з перешкодами. Вона використовує NumPy для створення двовимірного масиву, де 0 позначає вільний простір, а 1 – перешкоду. Функція visualize_map візуалізує карту, шлях дослідження, стартову та кінцеву позиції за допомогою Matplotlib.

Алгоритм завершує роботу, коли всі фронтири досліджені або недосяжні. Результатом є повний шлях, який пройшов робот під час дослідження. Візуалізація дозволяє оцінити ефективність алгоритму та охоплення карти.

У коді, який використовує Frontier-Based Exploration**,** немає фіксованої кінцевої точки (Finish), такої як finish = (19, 19).

Принцип Frontier-Based Exploration полягає в тому, що робот досліджує невідоме середовище, рухаючись до фронтирів (меж між дослідженими та

невідомими областями), поки всі доступні фронтири не будуть досліджені або не залишиться шляху до них.

Саме тому у функції explore немає параметра goal. Замість цього, вона динамічно вибирає найближчий центроїд фронтиру як тимчасову ціль для A* на кожному кроці, щоб розширити досліджену область.

Таким чином, у даному випадку, finish = (19, 19) не задається, оскільки завдання алгоритму – дослідити карту, а не дістатися до конкретної кінцевої точки. Остання позиція робота у шляху (exploration_path[-1]) є кінцевою точкою дослідження, а не наперед визначеною ціллю. Результати роботи алгоритму Frontier Based Exploration наведено в Додатку Д1.

5.1.2 Алгоритм та програмна реалізація алгоритму Heuristic Cost Function

Алгоритм Heuristic_Cost_Function виконує локальне планування шляху, оцінюючи кожен можливий напрям руху за складною функцією вартості. Він враховує відстань до цілі, ризик зіткнення з перешкодами, плавність траєкторії та уникнення повторних відвідувань. Основна мета – обрати найменш затратний і безпечний наступний крок. Кожна компонента має свою вагу, що дозволяє налаштовувати поведінку планувальника. Алгоритм працює покроково, адаптуючись до змін у середовищі.

Клас HeuristicPlanner реалізує планувальник шляху, який використовує складну вартісну функцію для вибору оптимального маршруту. Основна мета – знайти шлях від стартової точки до цілі, уникнувши перешкод та мінімізуючи загальну вартість переміщення.

Функція calculate_j_pos обчислює вартість відстані до цілі за допомогою евклідової відстані. Ця складова забезпечує прагнення робота до цільової точки. Чим ближче кандидат до цілі, тим нижча вартість. Використання евклідової відстані дає більш точну оцінку порівняно з манхеттенською, особливо при діагональних рухах.

Функція calculate_j_risk оцінює ризик зіткнення з перешкодами. Вона аналізує 3x3 окіл навколо кандидата та нараховує штраф за кожну сусідню

перешкоду. Якщо кандидат знаходиться на перешкоді або поза межами карти, функція повертає нескінченну вартість, що робить таку позицію неприйнятною. Цей механізм запобігає вибору небезпечних маршрутів.

Функція calculate_j_vel_reflection враховує плавність руху, штрафуючи за різкі зміни напрямку. Вона обчислює кут між поточним рухом та потенційним напрямком. Нульовий кут відповідає руху вперед, тоді як великі кути вказують на різкі повороти. Ця складова допомагає зменшити непотрібні маневри, що особливо важливо для фізичних роботів.

Функція get_memory_penalty запобігає зацикленню, збільшуючи вартість клітинок, які вже відвідувалися. Кожен візит до клітинки збільшує її штраф, що спонукає робота досліджувати нові області замість постійного повернення до старих. Це важливо для уникнення локальних мінімумів у складних середовищах.

Функція plan_step інтегрує всі вартісні компоненти для вибору оптимального кроку. Вона аналізує всі можливі напрямки руху (включаючи діагональні), обчислює загальну вартість для кожного кандидата та вибирає варіант з мінімальною вартістю. Ваги окремих компонентів можна налаштовувати для зміни поведінки планувальника. Наприклад, збільшення ваги j_pos прискорить рух до цілі, але може збільшити ризик зіткнення. Результати роботи алгоритму Heuristic Cost Function наведено в Додатку Д2.

5.1.3 Алгоритм та програмна реалізація алгоритму Local Sampling

Алгоритм Local Sampling реалізує планування шляху за допомогою випадкової вибірки напрямків руху в межах локального простору. На кожному кроці генерується обмежена кількість кандидатних напрямків, для яких обчислюється вартість руху. Вартість базується на відстані до цілі, ризику зіткнення, зміні напряму та історії відвідувань. Серед кандидатів вибирається напрямок із найменшою вартістю, що дозволяє рухатись ефективно з низькими обчислювальними витратами. Алгоритм працює покроково до досягнення цілі або блокування руху.

Клас LocalSamplingPlanner реалізує метод планування шляху на основі локальної випадкової вибірки. Він використовує стохастичний підхід для дослідження простору навколо поточної позиції робота. Алгоритм працює шляхом оцінки обмеженої кількості випадкових напрямків на кожному кроці.

Функція calculate_cost обчислює загальну вартість переміщення до кандидатної позиції. Вона враховує чотири ключові фактори: відстань до цілі, ризик зіткнення з перешкодами, плавність зміни напрямку руху та історію відвідувань. Кожен фактор має свій ваговий коефіцієнт, що дозволяє налаштовувати поведінку алгоритму.

Функція plan_step є основним механізмом прийняття рішень. На кожній ітерації вона генерує набір потенційних напрямків руху. Кількість напрямків визначається параметром num_samples. Для кожного напрямку розраховується вартість переміщення, і вибирається варіант з мінімальними витратами. Цей підхід дозволяє зменшити обчислювальну складність порівняно з повним перебором усіх можливих напрямків.

Алгоритм враховує обмеження карти через перевірку меж і наявності перешкод. Кандидати, що виходять за межі карти або знаходяться на перешкодах, отримують нескінченну вартість, що робить їх неприйнятними для вибору. Це забезпечує безпеку руху робота.

Механізм visited_count запобігає зацикленню шляхом збільшення вартості клітинок, які вже відвідувалися. Однак у порівнянні з детермінованими алгоритмами вплив цього механізму менш виражений через стохастичну природу методу.

Алгоритм продовжує роботу до досягнення цільової позиції або до вичерпання максимальної кількості кроків. У разі застрявання (відсутності допустимих напрямків руху) він повідомляє про проблему та завершує роботу. Це важливо для уникнення нескінченних циклів у складних середовищах. Результати роботи алгоритму Local Sampling наведено в Додатку Д3.

5.1.4 Алгоритм та програмна реалізація алгоритму Greedy Reactive Planning

Алгоритм Greedy Reactive Planning використовує стратегію, яка обирає на кожному кроці найвигідніший напрямок без довгострокового планування. Він оцінює 8 можливих напрямків і вибирає той, що мінімізує вартість, враховуючи відстань до цілі, близькість до перешкод і частоту відвідувань. Прийняття рішень відбувається локально, що робить алгоритм швидким, але потенційно неефективним у складних конфігураціях. Обхід перешкод та меж карти забезпечується шляхом відкидання недопустимих кандидатів. Алгоритм працює, поки не досягнуто цілі або не вичерпано можливості руху.

Клас GreedyReactivePlanner реалізує реактивний підхід до планування шляху. Алгоритм орієнтований на миттєве прийняття рішень без глобального аналізу карти. На кожному кроці він обирає напрямок, який виглядає оптимальним з поточного стану.

Функція get_memory_penalty відстежує історію відвідувань клітинок. Вона повертає кількість попередніх візитів до заданої позиції. Цей механізм допомагає запобігти зацикленню, але має відносно невеликий вплив на загальну поведінку алгоритму через низьку вагу параметра.

Функція calculate_cost обчислює вартість переміщення до кожної потенційної позиції. Вона враховує три ключові фактори: відстань до цілі, близькість до перешкод та історію відвідувань. Кожен фактор має свій ваговий коефіцієнт, що дозволяє налаштувати баланс між різними аспектами руху.

Функція plan_step є основним механізмом прийняття рішень. На кожній ітерації вона аналізує всі можливі напрямки руху (8 сусідніх клітинок) і обирає варіант з мінімальною вартістю. Цей підхід гарантує локальну оптимальність на кожному кроці, але не забезпечує глобально оптимального маршруту.

Алгоритм перевіряє межі карти та наявність перешкод. Кандидати, що знаходяться за межами карти або на перешкодах, отримують нескінченну вартість, що робить їх неприйнятними для вибору. Це забезпечує безпеку руху робота в невідомому середовищі.

Параметр weight_pos відповідає за прагнення до цілі. Вища вага цього параметра прискорює рух у напрямку кінцевої точки, але може призвести до небезпечних маневрів поблизу перешкод. Параметр weight_risk контролює ступінь обережності при русі. Параметр weight_memory допомагає уникнути зациклення, але його вплив обмежений через невелике значення ваги.

Алгоритм продовжує роботу до досягнення цільової позиції або до вичерпання максимальної кількості кроків. У разі відсутності допустимих напрямків руху він повідомляє про застрягання та завершує роботу. Це важливо для уникнення нескінченних циклів у складних середовищах. Результати роботи алгоритму Greedy Reactive Planning наведено в Додатку Д4.

5.1.5 Використання алгоритмів планування шляху

Система планування траєкторії реалізує комплексну систему навігації, яка поєднує чотири основні алгоритми планування шляху. Кожен з них виконує певну роль у процесі дослідження середовища та пошуку оптимального маршруту.

Frontier-Based Exploration використовується у функції simulate_sensor для виявлення кордонів між дослідженими та недослідженими областями. Алгоритм сканує навколишнє середовище в радіусі SENSOR_RANGE та ідентифікує фронтири – вільні клітинки, що межують з недослідженими зонами. Ці точки стають потенційними цілями для подальшого руху. Функція також оновлює мапу known_grid, позначаючи нововідкриті ділянки.

Heuristic Cost Function реалізована у трьох основних функціях вартості. calculate_j_pos оцінює відстань до цілі та штрафує за різкі зміни напрямку руху. calculate_j_risk аналізує ризик зіткнення з перешкодами, враховуючи як безпосередню близькість, так і час до можливого зіткнення. calculate_j_vel_reflection додає механізм «користного відбиття», який імітує фізичну взаємодію з перешкодами. Ці функції комбінуються з ваговими коефіцієнтами для отримання загальної оцінки кожного потенційного кроку.

Local Sampling застосовується для генерації точок-кандидатів у функції simulate_sensor. Алгоритм створює набір точок навколо поточної позиції робота, зосереджуючись на фронтирах та вільних клітинках біля перешкод. Функція prune_samples видаляє надлишкові точки, що знаходяться надто близько одна до одної, оптимізуючи таким чином кількість обчислень.

Greedy/Reactive Planning є основним алгоритмом прийняття рішень у головному циклі програми. На кожній ітерації система обирає точку з найменшою вартістю (функція plan_step), розраховує оптимальний вектор швидкості та виконує крок за допомогою move_one_step. Реактивність забезпечується швидкою реакцією на зміни в оточенні, виявлені сенсорами, а жадібність проявляється у виборі локально оптимального рішення на кожному кроці.

Алгоритми тісно взаємодіють у єдиному циклі планування. Frontier-Based Exploration забезпечує актуальну інформацію про стан середовища. Local Sampling генерує набір точок-кандидатів для подальшого аналізу. Heuristic Cost Function оцінює кожен варіант за комплексом критеріїв. На основі цих оцінок Greedy/Reactive Planning приймає рішення про наступний крок. Така комбінація дозволяє ефективно досліджувати невідоме середовище, уникаючи перешкод і поступово наближаючись до цілі.

5.2 Алгоритм автономного планування траєкторій у середовищах з перешкодами

Система автономного планування траєкторій реалізована через комбінацію кількох взаємопов’язаних компонентів. Основний цикл роботи складається з шести ключових етапів, які повторюються на кожній ітерації.

Функція simulate_sensor виконує сканування навколишнього середовища в заданому радіусі SENSOR_RANGE. Вона оновлює мапу known_grid, позначаючи нововідкриті ділянки як вільні або перешкоди. Паралельно виявляються

фронтирні клітинки – межі між дослідженими та недослідженими областями. Ці точки стають потенційними цілями для подальшого руху.

Функція prune_samples оптимізує кількість точок-кандидатів шляхом видалення надлишкових семплів, що знаходяться надто близько один до одного. Алгоритм сортує точки за відстанню до цілі та залишає лише найперспективніші варіанти, що підвищує ефективність подальших обчислень. Це допомагає швидше і ефективніше шукати шлях до цілі.

Обчислення вартості кожної точки виконується через три ключові функції. calculate_j_pos оцінює відстань до цілі та враховує плавність траєкторії. calculate_j_risk аналізує ризик зіткнення з перешкодами, використовуючи концепцію часу до зіткнення. calculate_j_vel_reflection додає механізм корисного відбиття, що імітує фізичну взаємодію з перешкодами.

Функція move_one_step реалізує безпечний рух робота до обраної точки. Вона враховує максимальну швидкість ROBOT_MAX_SPEED та перевіряє чи немає перешкод або зіткнень на шляху. У разі виявлення перешкоди на шляху система автоматично коригує траєкторію, зменшуючи крок руху.

Візуалізаційний модуль plot_results забезпечує інтерактивне відображення процесу планування. Він показує поточний стан карти, історію руху робота, позиції семплів та обраний напрямок руху. Це дозволяє аналізувати роботу алгоритму в реальному часі.

5.2.1 Імітація роботи сенсорів

Функція simulate_sensor є ключовим компонентом, що імітує роботу сенсорної системи робота. Вона реалізує три основні функції: оновлення карти середовища, виявлення перешкод та визначення фронтирів.

Механізм оновлення карти працює за принципом кругового сканування. Алгоритм визначає квадратну область навколо робота зі сторонами $2 \times$ SENSOR_RANGE. Для кожної клітинки в цій області перевіряється відстань від центру клітинки до позиції робота. Якщо клітинка знаходиться в межах радіусу SENSOR_RANGE, її статус оновлюється в мапі known_grid.

Процес виявлення перешкод використовує глобальну карту global_grid як еталон. Коли робот виявляє перешкоду, відповідна клітинка в known_grid позначається значенням OBSTACLE. Вільний простір маркується як FREE. Недосліджені області зберігають значення UNKNOWN.

Функція визначення фронтирів аналізує межі між дослідженими та недослідженими областями. Для кожної вільної клітинки перевіряються її сусіди у восьми напрямках. Якщо серед сусідів є недосліджені клітинки, поточна клітинка вважається фронтиром. Центри таких клітинок додаються до списку потенційних цілей.

Додатково система імітує виявлення перешкод шляхом аналізу клітинок, що межують з відомими перешкодами. Це дозволяє створювати додаткові точки-кандидати біля кордонів перешкод, що підвищує безпеку траєкторії.

Функція повертає оновлений список фронтирних точок та прапорець, що вказує на зміни в мапі. Ця інформація використовується для подальшого планування руху та оновлення візуалізації.

5.2.2 Програмна реалізація системи

Система ініціалізується створенням глобальної карти за допомогою функції create_grid. Початкова позиція робота START_POS і ціль GOAL_POS задаються у вигляді координат з плаваючою точкою для підвищення точності. Відома карта known_grid спочатку містить лише недосліджені області.

Головний цикл планування виконується поки робот не досягне цільової позиції або не вичерпає максимальну кількість ітерацій MAX_ITERATIONS. На кожному кроці система:

1. оновлює інформацію про навколишнє середовище через simulate_sensor,

2. генерує та фільтрує набір точок-кандидатів за допомогою prune_samples,

3. оцінює вартість кожного семплу через функції calculate_j_pos, calculate_j_risk та calculate_j_vel_reflection,

4. обирає оптимальний напрямок руху на основі мінімальної сумарної вартості,

5. виконує крок у обраному напрямку через move_one_step,

6. візуалізує поточний стан за допомогою plot_results.

Коефіцієнти W_POS, W_RISK та W_VEL дозволяють налаштувати баланс між різними аспектами руху. Параметри RESTITUTION_NORMAL і RESTITUTION_TANGENTIAL контролюють поведінку системи при взаємодії з перешкодами.

Система включає механізми обробки особливих ситуацій. При відсутності недопустимих точок руху вона автоматично переходить у режим відновлення, намагаючись знайти найближчу недосліджену область. У разі застрягання алгоритм пробує альтернативні маршрути перед повною зупинкою.

Фінальна візуалізація показує повний шлях робота, відзначаючи успішність досягнення цілі. Система також надає статистику про витрачений час та кількість ітерацій, що дозволяє оцінити ефективність роботи алгоритму. Кроки роботи програмної реалізації планування руху в середовищах з перешкодами наведено в Додатку Д5.

Повний код програмної реалізації планування руху в середовищах з перешкодами представлено в Додатку Е.

Висновок до розділу 5

Розглянуто аспекти автономного планування траєкторій для мобільних роботів, що функціонують у середовищах із перешкодами. Основну увагу приділено аналізу та програмній реалізації ключових алгоритмів, які забезпечують ефективну навігацію та дослідження невідомих територій.

Описано чотири базові підходи. Frontier-Based Exploration показує ефективність у вивченні невідомих просторів завдяки руху до меж між дослідженими й недослідженими ділянками, використовуючи A* для навігації до центрів цих меж. Цей метод не потребує наперед визначеної цілі, а динамічно обирає цілі для розширення карти. Heuristic Cost Function виступає як локальний

планувальник, що обирає наступний крок на основі комплексної оцінки вартості з урахуванням відстані до цілі, ризику зіткнень, плавності руху та історії відвідувань, що дозволяє адаптивно реагувати на зміни середовища. Local Sampling демонструє стохастичний підхід, який через випадкову вибірку напрямків у локальному просторі знижує обчислювальні витрати, зберігаючи ефективність руху шляхом оцінки кандидатних точок за вартісною функцією. Greedy Reactive Planning реалізує швидке, реактивне прийняття локально оптимальних рішень на основі оцінки сусідніх позицій, що забезпечує швидку реакцію, але не гарантує глобальну оптимальність маршруту.

Продемонстровано інтегроване використання цих алгоритмів у комплексній системі планування. Frontier-Based Exploration використовується для ідентифікації потенційних цілей (фронтирів) при симуляції роботи сенсорів. Local Sampling генерує точки-кандидати, які потім оцінюються за допомогою Heuristic Cost Function. На основі цих оцінок Greedy/Reactive Planning приймає фінальне рішення про наступний крок робота.

Особливу увагу приділено алгоритму автономного планування в цілому, що включає симуляцію сенсорної системи для оновлення карти відомого середовища та виявлення фронтирів, оптимізацію вибірки кандидатних точок, розрахунок їхньої вартості за допомогою комбінованих критеріїв та безпечне виконання руху. Програмна реалізація системи об’єднує ці компоненти в єдиний цикл, дозволяючи роботу автономно досліджувати простір, уникати перешкод та рухатися до заданої цілі, що підтверджується результатами візуалізації.

Таким чином, у розділі представлено як окремі ключові методики планування шляху, так і комплексний підхід до їх поєднання для створення ефективної системи автономної навігації робота в динамічних середовищах з перешкодами.

ВИСНОВКИ

Головною метою дослідження було створення системи автономного планування траєкторій для мобільних роботів у середовищах з перешкодами.

Для досягнення цієї мети проведено детальне вивчення та аналіз фундаментальних аспектів управління рухом роботів, зокрема концепції простору конфігурацій, яка перетворює фізичні перешкоди на геометричні області, спрощуючи пошук безпечних траєкторій. Розглянуто також метод клітинної декомпозиції, що дозволяє дискретизувати простір, зважуючи точність та обчислювальну складність. Визначено, що сучасні тенденції у робототехніці вимагають інтеграції адаптивних інтелектуальних методів для ефективної роботи в непередбачуваних умовах.

Складено огляд та класифікацію одинадцяти алгоритмів побудови шляху робота з перешкодами, поділених на шість категорій: класичні підходи на графах (A*, Дейкстри), евристичні підходи для динамічних середовищ (D* Lite, Theta*), метаевристичні стратегії (Генетичний алгоритм, RRT), біонатхненні методи (ACO, GWO), імовірнісні алгоритми (PRM, RRT*) та гібридні алгоритми (A* з потенційними полями). Визначено, що кожен із розглянутих алгоритмів має окремі властивості, які важливі для різних класів задач планування. Описано основні характеристики та переваги кожного типу алгоритмів, що є важливим для їх програмної реалізації та порівняльного аналізу ефективності.

Оцінено сучасні підходи до оцінки ефективності пошуку шляху в дискретному середовищі з перешкодами. Розроблено гнучку модель сіткової карти для генерації середовищ з випадковими перешкодами, що забезпечує прохідність. Проведено 100 ітерацій тестування для кожного алгоритму, оцінюючи час виконання, довжину та гладкість шляху. Виявлено значні відмінності у продуктивності алгоритмів за кожною з метрик.

Обрано критерії інтегральної оцінки ефективності алгоритмів, що включають час виконання (ваговий коефіцієнт 60 %) та гладкість шляху (ваговий

коецієнт 40 %). Цей вибір обґрунтований вимогами до практичних застосувань у робототехніці, де оперативність та плавність траєкторії є критично важливими.

Згідно з інтегральною оцінкою за заданими ваговими коефіцієнтами, алгоритм A* з потенційними полями виявився найбільш збалансованим та ефективним. Він успішно поєднує відносно швидке виконання з високою якістю знайденого шляху, що робить його привабливим для використання в реальних системах.

Проаналізовано програмне та апаратне забезпечення для автономної навігації. Програмна частина базується на мовах (C++, Python), бібліотеках (ROS, OpenCV, PCL) та алгоритмах SLAM, планування шляху (A*, RRT) і керування (PID). Апаратне забезпечення включає сенсори (LiDAR, камери, RADAR, IMU) та обчислювальні платформи (GPU, FPGA) для обробки даних у реальному часі. Ключову роль відіграє злиття даних сенсорів, що підвищує точність та надійність системи.

Розглянуто аспекти автономного планування траєкторій для мобільних роботів, що функціонують у середовищах із перешкодами. Проведено серію симуляцій з різними алгоритмами та продемонстровано інтегроване використання чотирьох базових підходів до автономного планування траєкторій: Frontier-Based Exploration, Heuristic Cost Function, Local Sampling та Greedy Reactive Planning. Порівняльний аналіз ефективності обраних алгоритмів на основі інтегральної оцінки підтвердив ефективність системи в автономному дослідженні простору, уникненні перешкод та русі до заданої цілі.

Продемонстровано інтегроване використання цих алгоритмів у комплексній системі планування. Frontier-Based Exploration використовується для ідентифікації потенційних цілей (фронтирів) при симуляції роботи сенсорів. Local Sampling генерує точки-кандидати, які потім оцінюються за допомогою Heuristic Cost Function. На основі цих оцінок Greedy/Reactive Planning приймає фінальне рішення про наступний крок робота.

Особливу увагу приділено алгоритму автономного планування в цілому, що включає симуляцію сенсорної системи для оновлення карти відомого середовища та виявлення фронтирів, оптимізацію вибірки кандидатних точок, розрахунок їхньої вартості за допомогою комбінованих критеріїв та безпечне виконання руху. Програмна реалізація системи об’єднує ці компоненти в єдиний цикл, дозволяючи роботу автономно досліджувати простір, уникати перешкод та рухатися до заданої цілі, що підтверджується результатами візуалізації.

Таким чином, результати дослідження показують, що розроблене програмне забезпечення є ефективним інструментом для автономного планування траєкторій роботів. Використання інтегрального підходу дозволило створити систему, здатну швидко та точно оцінювати та будувати маршрути на основі заданих параметрів та змінних середовищ. Це програмне забезпечення може стати одним з компонентів у процесі проектування та експлуатації автономних роботів, забезпечуючи ефективність роботи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Robot Path Planning and Motion Control: A Systematic Review. URL: https://www.espublisher.com/uploads/article_pdf/es1261_JustAccepted.pdf (дата звернення: 15.07.2025).

2. Path Planning for Robots: Methods, Challenges, and Applications. URL: https://thinkrobotics.com/blogs/learn/path-planning-for-robots-methods-challenges-and-applications (дата звернення: 15.07.2025).

3. An Overview and Comparison of Traditional Motion Planning Based on Rapidly Exploring Random Trees. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC11991108/ (дата звернення: 15.07.2025).

4. Robotics Part 32. URL: https://www.roboticsunveiled.com/robotics-trajectory-generation/ (дата звернення: 15.07.2025).

5. Trajectory generation and smoothing. URL: https://library.fiveable.me/robotics/unit-7/trajectory-generation-smoothing/study-guide/uc9CWa9RzOdzDOrF (дата звернення: 15.07.2025).

6. Humanoid Locomotion and Manipulation: Current Progress and Challenges in Control, Planning, and Learning. URL: https://arxiv.org/html/2501.02116v2 (дата звернення: 15.07.2025).

7. Mobile robot path planning using exact cell decomposition and potential field methods. URL: https://scispace.com/pdf/mobile-robot-path-planning-using-exact-cell-decomposition-4s2738owcm.pdf (дата звернення: 15.07.2025).

8. Recent Advances and Challenges in Industrial Robotics: A Systematic Review of Technological Trends and Emerging Applications. URL: https://www.mdpi.com/2227-9717/13/3/832 (дата звернення: 15.07.2025).

9. The Topology of Robotic Configuration and Motion Planning. URL: https://oniani.org/pdf/agv.pdf (дата звернення: 15.07.2025).

10. Motion planning and configuration spaces. URL: https://library.fiveable.me/computational-algebraic-geometry/unit-12/motion-

planning-configuration-spaces/study-guide/9OLocTFSks6smxWQ (дата звернення: 15.07.2025).

11. Introduction to robotics. URL: https://web2.qatar.cmu.edu/~gdicaro/16311-Fall17/slides/16311-27-PathPlanning-2.pdf (дата звернення: 15.07.2025).

12. Research on path planning of mobile robots based on improved A* algorithm. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC11888910/ (дата звернення: 15.07.2025).

13. Progress in Construction Robot Path-Planning Algorithms: Review. URL: https://www.mdpi.com/2076-3417/15/3/1165 (дата звернення: 15.07.2025).

14. Motion Planning for Robotics: A Review for Sampling-based Planners. URL: https://arxiv.org/html/2410.19414v1 (дата звернення: 15.07.2025).

15. Obstacle Avoidance and Path Planning Methods for Autonomous Navigation of Mobile Robot. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC11175283/ (дата звернення: 15.07.2025).

16. Path planning diagram. URL: https://www.researchgate.net/figure/Path-planning-diagram_fig1_366333455 (дата звернення: 15.07.2025).

17. How can i use particle swarm optimisation algorithm for to find optimal path interms of shortest distance between start and goal point to be followed by mobile robot? URL: https://www.mathworks.com/matlabcentral/answers/263094-how-can-i-use-particle-swarm-optimisation-algorithm-for-to-find-optimal-path-interms-of-shortest-dis (дата звернення: 15.07.2025).

18. Obstacle-Free Path Planning for Autonomous Drones Using Floyd's Algorithm. URL: https://www.arxiv.org/pdf/2409.13149v1 (дата звернення: 15.07.2025).

19. Theta* algorithm path planning schematic. URL: https://www.researchgate.net/figure/Theta-algorithm-path-planning-schematic_fig3_366333455 (дата звернення: 15.07.2025).

20. Modified A* algorithm for path smoothing and obstacle avoidance. URL: https://www.ewadirect.com/proceedings/ace/article/view/9971/pdf (дата звернення: 15.07.2025).

21. Genetic Algorithm-based Robot Path Planning. URL: https://repo.pens.ac.id/188/1/c3d14cd2e419cde877993555eeef.pdf (дата звернення: 15.07.2025).

22. The Path Planning of Mobile Robots Based on an Improved Genetic Algorithm. URL: https://www.researchgate.net/publication/390277090_The_Path_Planning_of_Mobile_Robots_Based_on_an_Improved_Genetic_Algorithm (дата звернення: 15.07.2025).

23. HPS-RRT*: An Improved Path Planning Algorithm for a Nonholonomic Orchard Robot in Unstructured Environments. URL: https://www.mdpi.com/2073-4395/15/3/712 (дата звернення: 15.07.2025).

24. Path-Planning Strategy: Adaptive Ant Colony Optimization Combined with an Enhanced Dynamic Window Approach. URL: https://www.mdpi.com/2079-9292/13/5/825 (дата звернення: 15.07.2025).

25. Optimizing Robot Path Planning with the Particle Swarm Optimization Algorithm. URL: https://geniusjournals.org/index.php/ejet/article/download/4938/4152/4825 (дата звернення: 15.07.2025).

26. Artificial bee colony algorithm. URL: https://library.fiveable.me/swarm-intelligence-and-robotics/unit-3/artificial-bee-colony-algorithm/study-guide/ZoVdCnZHGoSv73f5 (дата звернення: 15.07.2025).

27. Multi-objective path planning for mobile robot with an improved artificial bee colony algorithm. URL: https://pubmed.ncbi.nlm.nih.gov/36899544/ (дата звернення: 15.07.2025).

28. Path Planning and Obstacle Avoidance of a Mobile Robot based on GWO Algorithm. URL: https://pdfs.semanticscholar.org/bbfc/a56c289bd3310fd8103cef8bf1418effc76a.pdf (дата звернення: 15.07.2025).

29. Robot Motion Control using Dual Avoidance Scheme. URL: https://www.espublisher.com/journals/articlehtml/engineered-science/10.30919-es1261 (дата звернення: 15.07.2025).

30. Cuckoo Search Algorithm using Lèvy Flight: A Review. URL: https://www.researchgate.net/publication/269651786_Cuckoo_Search_Algorithm_using_Levy_Flight_A_Review (дата звернення: 15.07.2025).

31. Wang, H.; Wang, S.; Yu, T. Path Planning of Inspection Robot Based on Improved Ant Colony Algorithm. *Appl. Sci.* 2024, 14, 9511. URL: https://doi.org/10.3390/app14209511 (дата звернення: 15.07.2025).

32. Hybrid potential field based control of differential drive mobile robots. URL: https://chaoslab.unm.edu/MaterialDownload/Hybrid%20potential%20Field%20based%20control%20of%20differential%20drive%20mobile%20robots.pdf (дата звернення: 15.07.2025).

33. Optimal Path Planning using RRT for Dynamic Obstacles. URL: https://nopr.niscpr.res.in/bitstream/123456789/54909/1/JSIR%2079(6)%20513-516.pdf (дата звернення: 15.07.2025).

34. Path Tracking Hybrid A* For Autonomous Agricultural Vehicles. URL: https://arxiv.org/html/2411.14086v1 (дата звернення: 15.07.2025).

35. The main types of sensors in Robotics & Self-Driving Cars. URL: https://www.thinkautonomous.ai/blog/types-of-sensors/ (дата звернення: 15.07.2025).

36. LIDAR Autonomous Mapping System – Electrical & Computer Engineering. URL: https://my.ece.utah.edu/~kstevens/4710/reports/lidar-mapping.pdf (дата звернення: 15.07.2025).

37. Autonomous Vehicle Systems. Cameras. URL: https://library.fiveable.me/autonomous-vehicle-systems/unit-2/cameras/study-guide/o56z4IxJbGXmrZFI (дата звернення: 15.07.2025).

38. Sensor Fusion Software in Autonomous Vehicles. URL: https://binmile.com/blog/sensor-fusion-software-in-self-driving-cars/ (дата звернення: 15.07.2025).

39. A Complete Guide to Inertial Measurement Unit (IMU). URL: https://www.jouav.com/blog/inertial-measurement-unit.html (дата звернення: 15.07.2025).

40. Inertial Measurement Unit (IMU). An Introduction. URL: https://www.advancednavigation.com/tech-articles/inertial-measurement-unit-imu-an-introduction/ (дата звернення: 15.07.2025).

41. Embedded Systems and Robotics: A Match Made in Heaven. URL: https://skill-lync.com/blogs/embedded-systems-and-robotics-a-match-made-in-heaven (дата звернення: 15.07.2025).

42. Embedded Systems in Autonomous Vehicle Technology. URL: https://www.ngenium.io/post/embedded-systems-in-autonomous-vehicle-technology (дата звернення: 15.07.2025).

43. How Online GPUs for Deep Learning Can Supercharge Your AI Models. URL: https://blog.runpod.io/how-online-gpus-for-deep-learning-can-supercharge-your-ai-models/ (дата звернення: 15.07.2025).

44. The Ultimate Guide to GPUs for Machine Learning in 2025. URL: https://blog.spheron.network/the-ultimate-guide-to-gpus-for-machine-learning-in-2025 (дата звернення: 15.07.2025).

45. FPGAs in Robotics – Revolutionizing Automation. URL: https://fpgainsights.com/fpga/fpgas-in-robotics/ (дата звернення: 15.07.2025).

46. FPGA-Based Sensor Fusion for Autonomous Robotics. URL: https://www.researchgate.net/publication/389085264_FPGA-Based_Sensor_Fusion_for_Autonomous_Robotics (дата звернення: 15.07.2025).

47. Sensor-Fusion Based Navigation for Autonomous Mobile Robot. URL: https://www.mdpi.com/1424-8220/25/4/1248 (дата звернення: 15.07.2025).

48. Свирид Ю. В., Гуменюк П. О. Класифікація алгоритмів побудови шляху робота з перешкодами. *Технологічні комплекси*. Луцьк, 2025. Том 17, № 2. С. 64-75.