Хмельницький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерно-інтегрованих технологій та робототехніки

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Система керування рухом мобільного робота
Назва теми

КвРАКІТ.2021053.01.06 ПЗ

Галузь знань 15 «Автоматизація та приладобудування»
Шифр, назва

Спеціальність 151 «Автоматизація та комп'ютерно-інтегровані технології»
Шифр, назва

Освітня програма «Автоматизація та комп'ютерно-інтегровані технології»
Назва

Виконав:

студент 3 курсу, група АКІТс-21-1 _____ Владислав ЛІТВІНОВ
Підпис, Ім'я, ПРІЗВИЩЕ

Керівник _____ Денис МАКАРИШКІН
Підпис, дата, Ім'я, ПРІЗВИЩЕ

Нормоконтролер _____ Людмила КОРЕЦЬКА
Підпис, дата, Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
зав. кафедри автоматизації,
комп'ютерно-інтегрованих
технологій та робототехніки _____ Валерій МАРТИНЮК
Підпис, дата, Ім'я, ПРІЗВИЩЕ

«17» червня 2024 р.

Хмельницький 2024

Хмельницький національний університет

Факультет *інформаційних технологій*
Кафедра *автоматизації, комп'ютерно-інтегрованих технологій та робототехніки*
Освітній рівень *перший (бакалаврський)*
Галузь знань *15 – Автоматизація та приладобудування*
Спеціальність *151 – Автоматизація та комп'ютерно-інтегровані технології*
Освітня-професійна програма *Автоматизація та комп'ютерно-інтегровані технології*

ЗАТВЕРДЖУЮ
Зав. кафедрою АКІТтаР
Валерій МАРТИНЮК
«10» 01 2024р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Літвінов Владислав Олександрович

1 Тема роботи: *Розробка модуля керування рухом мобільного робота*
керівник роботи Макаришкін Д.А., к.т.н, доцент
Затверджено наказом по університету від «15» 02 2024 р. № 8.
2 Строк подання студентом роботи на кафедру: 01.06.24р.
3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування
4 Зміст пояснювальної записки (перелік питань, які потрібно розробити) Вступ. Огляд та аналіз особливостей алгоритмів планування руху. Основна частина. Розробка проєкту модулю переміщення мобільного роботу. Розробка модуля переміщення мобільного роботу. Висновки.
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали.

Завдання отримав

Науковий керівник

Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Антиплагіат	Федула М.В., доцент кафедри АКІТтаР		
Нормоконтроль	Корецька Л.О., доцент кафедри АКІТтаР		

7. Дата видачі завдання «10» 01 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів (розділів) дипломної роботи	Строк виконання етапів дипломної роботи	Примітка
1	Вступ	15.02.2024р.	Виконано
2	Огляд та аналіз особливостей алгоритмів планування руху	15.03.2024р.	Виконано
3	Основна частина Розробка проєкту модулю переміщення мобільного роботу	10.04.2024р.	Виконано
4	Розробка модуля переміщення мобільного роботу.	10.05.2024р.	Виконано
5	Висновки	15.05.2024р.	Виконано
6	Оформлення пояснювальної записки до КРБ	25.05.2024р.	Виконано
7	Оформлення презентаційних матеріалів	1.06.2024р.	Виконано

Студент Підпис В.О. Літвінов Ініціали, прізвище

Керівник роботи Підпис Д.А. Макаришкін Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Система керування рухом мобільного робота».
Автор роботи: Літвінов Владислав Олександрович
Керівник роботи: Макаришкін Д.А. к.т.н., доцент
Пояснювальна записка: 55 с., 30 рис., 1 табл., 2 дод., 37 джерел.
Графічна частина: Презентаційні матеріали.

МОБІЛЬНИЙ РОБОТ, MULTI AGENT ROBOTIC SYSTEM, МОДУЛЬ КЕРУВАННЯ, ISTRATEGY, АЛГОРИТМ TANGENT BUG, JAVA.

Метою роботи є розробка модуля керування рухом мобільного робота, призначеного для використання в системі Multi Agent Robotic System. Проведено огляд та аналіз існуючих алгоритмів планування руху мобільного робота, встановлено вимоги до модулю керування мобільного робота IStrategy, що розробляється.

Для реалізації алгоритму локального пошуку корисно виконати невелику корецкцію – дані із давачів збиратимуться не постійно, а на певних кроках, у режимі руху вздовж стіни не будуватиметься локальний граф дотичних, що дасть змогу суттєво заощадити ресурси. Коректність роботи алгоритму локального пошуку суттєво залежить від точності давача дальності.

Через особливості реалізації розроблюваного модуля, а саме використання емулятора замість реального мобільного робота обхід фізичних перешкоди в режимі руху уздовж стіни займає більшу кількість часу і призводить до накопичення помилки, оскільки проводиться поворот мобільного робота разом із давачем дальності під час кожного контролю відстані до фізичної перешкоди. Загалом розроблений алгоритм локального пошуку Tangent Bug показує прийнятні результати і надалі може бути використаний при роботі на реальному мобільного робота.

01.06.2023р.
дата

. Підпис

ЗМІСТ

ВСТУП

Тема керування рухом мобільного робота (МР) вивчається досить давно, але не втрачає своєї актуальності і сьогодні. Недорогі МР мають широке використання в ситуаціях, де виконання такого виду роботи людиною неможливе або сильно ускладнене – заражена або замінована місцевість, важкодоступні місця.

Мета даної кваліфікаційної роботи - розробка модуля керування рухом МР, призначеного для використання в системі MARS.

Модуль керування рухом буде відповідати за поведінку МР в ситуації, коли МР зустрів несподівану перешкоду, наприклад не нанесену на вже створену карту. В даній кваліфікаційній роботі розглянуто алгоритми глобального та локального планування, реалізовано алгоритм руху та обходу перешкод Tangent Bug, який використовує ультразвуковий давач дальності. Під час розроблення даного модуля враховували можливість подальшої його адаптації на роботах видів NXT та EV3 на базі елементів MINDSTORM та елементів LEGO.

1 ОГЛЯД ТА АНАЛІЗ ОСОБЛИВОСТЕЙ АЛГОРИТМІВ ПЛАНУВАННЯ РУХУ

1.1 Коротка характеристика програмної платформи MARS [1 - 3]

Multi Agent Robotic System або MARS [1-3] - це програмна платформа керування експериментальною роботизованою системою. Дана керована роботизована система (КРС) може вирішувати різні завдання, наприклад пошук об'єктів, що повинні відповідати певним встановленим критеріям. До системи можуть входити різні типи пристроїв і МР із кардинально різними технічними характеристиками. КРС може складатися лише з центрів керування та МР. На рисунку 1.1 представлена спрощена діаграма програмної платформи MARS [1-3].

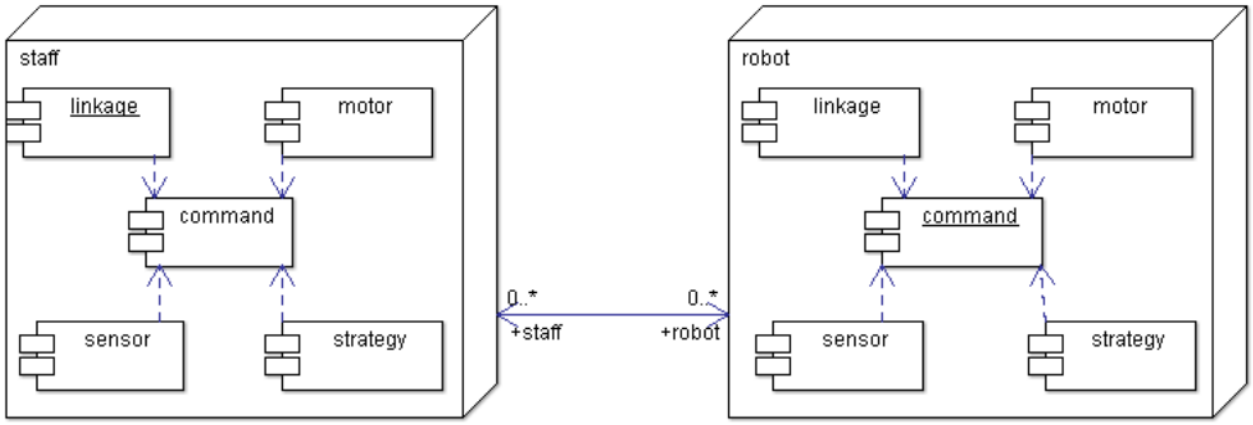


Рисунок 1.1 - Спрощена діаграма програмної платформи MARS [1-3]

1.1.1 Центри керування програмної платформи MARS [1-3]

Основні функції центрів керування (ЦК) програмної платформи MARS наступні:

- формулювання завдань, які повинна вміти вирішувати система;
- формування стратегії для вирішення поставлених завдань;
- розбиття завдань на підзадачі та розподіл підзадач між МР;
- контроль виконання поставлених завдань;

- надання ресурсів для обчислення МР (за необхідності);

- керування загальними даними системи та надання доступу до них (за необхідності).

ЦК можуть бути як повністю автоматизовані, так і керувантися через HMI людиною-оператором. Наразі ЦК являють собою портативні комп'ютери під керування операційної системи типу Android.

1.1.2 Мобільні роботи [4-6]

МР будуть виконувати наступні функції:

- виконання поставлених завдань, як самостійно, там і в групі з іншими МР;

- виконання руху відповідно до заданого маршруту, за необхідності огинання виявлених перешкод;

- передача отриманої інформації, зібраної давачами, в ЦК або іншим МР;

- передача результатів виконання поставленого завдання;

- передача у ЦК інформації про свій стан.

На даний момент програмна платформа MARS [1-3] представлена наступними апаратно-програмними засобами:

- ЦК: ОС видів - Linux, MacOS, Windows, Android;

- МР типів - Lego NXT, Raspberry PI, Lego EV3;

- програмні емулятори роботів.

1.1.3 Реалізація поставленого завдання

До складу реалізації вузлів керування та МР будуть входити наступні програмні пакети:

- robot.linkage - зв'язок з іншими компонентами, обмін повідомленнями за допомогою інтерфейсу ILinkage;

- robot.motor - керування моторами робота, надання інтерфейсу IMotor;

- robot.command - керування роботою вузла, взаємодія з іншими об'єктами

через інтерфейс IControl;

- robot.strategy - керування роботом у нестандартних ситуаціях, надання інтерфейсу IStrategy;

- robot.sensor - керування роботою давачів, надання доступу до даних давачів через інтерфейс ISensor.

1.2 Огляд та аналіз існуючих алгоритмів планування руху МР

Алгоритми, що дають змогу планувати рух МР, можна класифікувати як методи глобального та локального планування. Глобальне планування дає змогу отримати оптимальну траєкторію руху, за якою повинен рухатися МР, щоб дістатися з початкового пункт в кінцевий пункт призначення. Для роботи таких алгоритмів глобального планування необхідна повна інформація про місцевість, а саме, про форму і розташування перешкод. Багато відомих алгоритмів планування маршруту, таких як алгоритм пошуку A* [7], алгоритм Дейкстри [8], алгоритм потенційного поля [9], алгоритми, що базуються на методі Монте-Карло [10, 11], потребують досить точної карти, із зазначенням на ній усіх наявних перешкод. Методи глобального планування також іноді носять назву методів, що засновані на картах, а методи локального планування - реактивних методів.

Для роботи локальних методів планування необхідне лише часткове знання оточуючого середовища. Беручи до уваги, що навколишнє середовище може змінюватися й інформація про оточуюче середовище може бути недостатньою, МР повинен досягати поставленої мети, володіючи обмеженою кількістю інформації про навколишнє середовище – як мінімум володіти інформацією про власне поточне місце розташування і кінцевий пункт призначення. Локальні методи необхідно використовувати в тому випадку, коли потрібно скорегувати траєкторію руху МР, отриману під час глобального планування [4-6].

1.2.1 Огляд та аналіз алгоритмів глобального планування маршруту

Відомі типи алгоритмів глобального планування маршруту можливо умовно розділити на три базові типи:

1. Алгоритми руху по карті – такі, що дають можливість визначити набір маршрутів у вільному просторі.

Такий тип алгоритмів розглядає простір, у якому перебуває МР, у вигляді зваженого графа і для пошуку оптимального шляху деяким чином обходить даний граф.

Відомо алгоритм Дейкстри [8] в якості вхідних даних даний алгоритм отримує зважений граф із невід'ємною вагою ребер. У результаті роботи алгоритму Дейкстри [8] виходить найкоротший шлях у даному графі. Принцип роботи алгоритму Дейкстри [8] полягає в тому, що вузлу графа зіставляється мітка, яка є вартістю шляху в цей вузол із початкового вузла. Алгоритм Дейкстри [8] працює покроково - на кожному кроці він розглядає один вузол і намагається зменшити значення мітки суміжних вузлів. Робота алгоритму Дейкстри [8] завершується, коли всі вузли графу відвідані.

Алгоритм пошуку маршруту типу A* [7] - працює аналогічно алгоритму Дейкстри [8]. Але, на відміну від нього, як значення міток алгоритм пошуку маршруту A* [7] використовує евристичну функцію для оцінки якості шляху. У найпростіших реалізаціях алгоритму пошуку маршруту A* використовується наступна функція - сума мінімального знайденого шляху до цієї вершини графа і довжини прямого шляху в евклідовій площині до точки фінішу. У результаті роботи евристичної функції алгоритм пошуку маршруту A* [7] вибере ті вузли, які ближче до кінцевого вузла. Час роботи алгоритму пошуку маршруту A* залежить від використовуваної евристики. У гіршому випадку часова складність збігається зі складністю алгоритму Дейкстри [8]. У разі використання ефективних евристик час роботи може значно зменшуватися. Отриманий шлях потім може бути згладжений для отримання кращого результату.

Проте споживання ресурсів для таких алгоритмів пошуку маршруту у

гіршому випадку зростає експоненціально залежно від кількості вершин у зваженому графі.

Алгоритми пошуку маршруту, засновані на використанні методу Монте-Карло, називають алгоритмами пошуку на основі вибірки. Один із перших алгоритмів пошуку, заснованих на методі Монте-Карло - планувальник шляху на основі вибірки [12]. Ця робота вплинула на алгоритми, і з'явилися такі відгалуження - імовірнісні карти (або Probabilistic Road Map - PRM) [10] і швидкозростаючі дерева (або RRT) [13, 14].

Основна ідея даних методів: замість перебору всіх можливих варіантів зробити випадкову вибірку деякої множини вершин графу і вибрати відповідний варіант із отриманої множини. Алгоритмам пошуку маршруту такого типу необхідні координати старту і фінішу, а також карта наявних перешкод.

Розглянемо алгоритм пошуку маршруту PRM. Уперше алгоритми пошуку даного типу були описані в [10, 15]. Алгоритм пошуку маршруту складається з двох фаз - фаза вивчення наявної карти і фаза запиту. У першій фазі відбувається зваженого створення графа, на якому буде виконуватися пошук шляху. Граф будується шляхом вибору випадкових точок на карті та вершин, які можна задавати у вигляді перемінних параметрів.

У другій фазі виконується прокладання оптимального шляху. Потрібно зауважити, що, оскільки всі вершини даного графа відповідають випадковим станам, то отриманий граф не містить станів старту та фінішу МР. Тому спочатку необхідно знайти вершини графа, близькі до станів старту і фінішу, і додати ці два стани в граф, проклавши відповідні шляхи і додавши ребра.

Тобто, отримуємо ациклічний граф, у якому існує єдиний шлях від старту до фінішу, який є результатом роботи алгоритму пошуку маршруту. Крім того, відсутність циклів створює проблему можливості появи невиправдано довгих шляхів, що вирішується авторами за допомогою згладжування. Наприклад, у літературі [17] запропоновано варіант PRM-алгоритму пошуку, де на етапі додавання нової вершини в граф, випадкові стани будуть вибиратимуться не в

усьому вільному просторі, а так би мовити, на «середній лінії» вільного простору. Недоліком методу PRM-алгоритму пошуку є ускладнення пошуку за невеликих відстаней між існуючими перешкодами, оскільки ймовірність випадково вибрати точки, шлях між якими лежить у такому проході, дуже низька. У таких ситуаціях найімовірніше оптимальний шлях не буде знайдено.

2. Алгоритми розбиття карти - розподіл карти, що дає змогу розрізняти вільні та зайняті комірки на існуючій карті.

Розбиття існуючої карти використовується під час пошуку шляху в повністю відомій місцевості. В алгоритмах пошуку маршруту такого типу відомий простір розбивається на безліч прямокутних клітин різного розміру, такий спосіб дає змогу уникнути дискретизації простору і домогтися розбиття простору з різною точністю в різних ділянках карти перешкод і побудувати більш реалістичну траєкторію руху, що актуально для робота. Наприклад, в літературі [17] представлено алгоритм в якому клітини, на які розбивається простір, класифікують як вільні, непрохідні та змішані, одна частина яких зайнята перешкодою, а інша частина містить вільний простір. На початку роботи алгоритму пошуку маршруту весь простір карти оцінюється як одна клітина. Потім у циклі поетапно виконуються наступні дії [17]:

- виконується прокладання шляху по вільних і змішаних клітинах від початкового пункту до кінцевого;

- у випадку якщо знайдено шлях, що повністю проходить через вільні клітини, то алгоритм пошуку маршруту завершується;

- у випадку якщо в знайденому шляху є змішані клітини, то вони дробляться на більш дрібні клітини;

- у випадку якщо шлях до кінцевої мети не знайдено, отже, шуканого шляху не існує і алгоритм пошуку маршруту завершує роботу.

Алгоритм розбиття карти є лише модифікацією для пошуку шляху в певних практичних ситуаціях, і початковий пошук шляху між двома точками за допомогою розбиття карти виконується шляхом одного з вищеописаних

алгоритмів, як правило, алгоритмом пошуку типу A* [8].

Обчислювальна ефективність методу розбиття карти залежить від розміру комірок, на які розбивається карта, що часто робить алгоритм пошуку маршруту неефективним для реалізацій, що працюють у режимі реального часу.

3. Алгоритми потенціального поля - ґрунтуються на простому, але при цьому досить ефективному принципі – кінцева мета ніби притягує МР, а перешкоди відштовхують.

У літературі розробником методу вважають - О. Хатіба [9]. Метод цікавий тим, що він поєднує в собі принципи як глобального так і локального планування.

Суть методу потенційних полів полягає в тому, що кожна перешкода ніби має навколо себе відштовхувальне потенційне поле, сила якого обернено пропорційна відстані до неї; також існує однорідна сила тяжіння до кінцевої мети. Через короткі постійні інтервали часу обчислюється сума векторів, що притягують і відштовхують, і МР переміщається в потрібному напрямку.

Метод потенційних полів відомий досить давно і досить добре досліджений. Це один із найпростіших і найефективніших алгоритмів навігації, в якості методики визначення векторів опору використовують досить прості формули з базового курсу фізики. Вони розраховуються тільки за показаннями давачів у реальний момент часу, тобто якщо МР не бачить перешкоду, то й опору вона не чинить. Навіть якщо в пам'яті МР (тобто на карті) дана перешкода фізично присутня.

Недоліки даного методу випливають із його переваг. Цей алгоритм пошуку маршруту хоча і дозволяє досягати мети є «реактивним», а не інтелектуальним. Локальність алгоритму пошуку маршруту часто робить його «короткозорим» і, відповідно, неефективним для досягнення складних цілей. Так, його часто критикують за високу ймовірність потрапляння МР в локальний мінімум. Однак, на практиці, така ймовірність досить мала і може бути значно «знижена» за рахунок використання додаткових, високорівневих поліпшень.

Існує декілька способів обходу круглих перешкод:

- закони руху рухомої точки;

- способи завдання функції відштовхування від перешкод.

Наприклад, нехай кінцева мета притягує рухому точку весь час з одиничною силою. Тоді силу відштовхування від кожної існуючої перешкоди бажано задати таким чином, щоб на границі вона приймала одиничне значення і була спрямована по можливості по нормалі до фізичної перешкоди, а поза фізичною перешкодою зменшувалась пропорційно відстані.

Сила відштовхування від фізичних перешкод формується на основі даних, отриманих від далекоміра. Фільтрація отриманих відстаней до фізичної перешкоди організована так, що всі відстані розташовані між "стрибками" (граничне значення «стрибка» задається і «стрибок» фіксується, якщо різниця між сусідніми відстанями в секторі огляду перевищує встановлений поріг) вважаються такими, що належать контуру однієї фізичної перешкоди. Далі виконується обрахунок відстані до видимої ділянки контуру за мінімальною дальності. Отримана мінімальна відстань до ділянки контуру є аргументом функції відштовхування [4-6].

1.2.2 Огляд та аналіз алгоритмів локального планування маршруту

Для розв'язання задачі локальної побудови маршруту використовуються суттєво різні види алгоритмів.

Наприклад, алгоритми пошуку маршруту сімейства Bug - найпростішими методами обходу фізичних перешкод [18]. Першим представником даного сімейства є алгоритм Bug1 [19, 20]. Під час роботи даного алгоритму пошуку маршруту МР слідує по границі перешкоди і безперервно вимірює реальну відстань від свого поточного положення до кінцевої точки. Потім МР, зробивши повний оберт навколо фізичної перешкоди, повертається до точки, в якій відстань до кінцевої цілі досягає мінімального значення. Очевидно, що головним недоліком алгоритму пошуку маршруту Bug1 є необхідність повного перебору

точок розташованих навколо перешкоди. Розвитком описаного раніше підходу є алгоритм пошуку маршруту Bug2 [19, 20]. Його принципова відмінність полягає в тому, що під час виявлення фізичної перешкоди МР запам'ятовує вектор, спрямований до кінцевої точки. Також як і в алгоритмі пошуку маршруту Bug1 МР рухається вздовж границі фізичної перешкоди, але при перетині вектора змінює траєкторію, прямуючи безпосередньо до кінцевої точки. Алгоритм DistBug [21] - являє собою інший варіант розвитку Bug алгоритмів пошуку маршруту. Суть даного методу полягає в постійному вимірюванні неузгодженості положення МР і кінцевої точки. У разі зменшення або збереженні величини неузгодженості - МР продовжує рух уздовж границі фізичної перешкоди, в іншому випадку напрямок змінюється на рух до кінцевої точки.

Алгоритм пошуку маршруту Tangent Bug, іншого представника алгоритмів пошуку групи Bug, було запропоновано 1995 року розробниками І. Камоном, Е. Раймоном та Е. Рівліном [22]. Даний алгоритм пошуку маршруту істотно відрізняється від попередніх алгоритмів пошуку цього сімейства, оскільки МР оснащується додатковими давачами-далекомірами, які визначають відстань до фізичної перешкоди.

Використовуючи даний алгоритм пошуку маршруту для руху й обходу перешкод МР не потрібна карта місцевості. МР зі своєї поточної позиції будує дотичні до країв фізичних перешкод, і представляє дані про відстань до фізичної перешкоди і кута повороту до них у вигляді локального графа дотичних. Локальний граф дотичних використовується алгоритмом пошуку маршруту для вибору оптимального напрямку в бік кінцевої точки та під час обходу фізичної перешкоди. Стратегію алгоритму пошуку маршруту Tangent Bug можливо описати таким чином:

- МР переміщається у напрямку до кінцевої мети, поки не виявляє на своєму шляху фізичну перешкоду. У разі виявлення МР визначає границі фізичної перешкоди за розривами функції дальності;

- МР переміщається до однієї з точок розриву (точки розриву знаходяться на границях фізичної перешкоди), для якої евристична оцінка мінімальна, наприклад до тієї, сума відстаней від якої до МР і до кінцевої точки мінімальна;

- у процесі переміщення отримуємо нові точки розриву і, відповідно, переміщуємось до них;

- у випадку потрапляння в локальний мінімум МР переходить у режим руху вздовж стіни, тобто переміщення вздовж границі фізичної перешкоди, зберігаючи головний напрямок;

- покидаємо границю, коли фізична перешкода не заважає проходу до кінцевої точки [23].

Follow the Gap (метод знаходження проміжків) був створений В. Сезер і М. Гокасан у 2012 році [24]. Даний метод пошуку маршруту веде МР через центр максимального прорізу між фізичними перешкодами. МР, зустрівшись із фізичною перешкодою на шляху до кінцевої точки, визначає відстань до фізичної перешкоди і записує її в масив. У масиві зберігаються відстані до видимих фізичних перешкод. Також створюється ще один масив для зберігання відстаней між фізичними перешкодами, так званих прорізів, з яких вибираються найширші. Якщо значення співпадають, тоді метод пошуку маршруту вибирає перше з них. Наступний етап це обчислення кута між найширшим прорізом і напрямком руху МР. Той напрямок, на якому відсутні фізичні перешкоди, є більш пріоритетним, ніж напрямок ближче до кінцевої точки. Метод пошуку маршруту вимагає наявності складних систем із давачів, але менш вимогливий до бортового ПК через меншу кількості розрахунків, завдяки чому він швидше визначає необхідну траєкторію руху МР, також може враховувати кінематичні обмеження МР. Проте даний метод не працює в ситуації з U-подібною фізичною перешкодою.

Гістограма векторних полів (або VFH-метод) [25-27] є використовує в роботі локальну карту місцевості, яка дає можливість зберігати та враховувати інформацію про визначені раніше найближчі об'єкти. Цей метод складається з

наступних етапів:

- створення сітки на двовимірній декартовій площині, що є детальним описом оточуючого середовища. Інформацію для формування такої сітки МР отримує за допомогою ультразвукових або лазерних давачів. Сітка постійно оновлюється в режимі реального часу;

- створення полярної гістограми - одновимірна полярна гістограма будується на основі даних, отриманих із сітки на двовимірній декартовій площині. По осі X відкладається кут між фізичними перешкодами та напрямком руху МР, по осі Y ймовірність наявності фізичної перешкоди. Загалом, така гістограма є статистичним представленням навколишнього середовища, у якому переміщується МР.

Алгоритм пошуку маршруту VFH було удосконалено в 1998 році і змінено назву на VFH+ [27]. Удосконалення включають збільшення гладкості розрахованої траєкторії руху, також у даній версії методу враховуються параметри МР, з'явилася функція прогнозування фізичної перешкоди - сектори, які заблоковані фізичними перешкодам відзначені особливим чином, для поліпшення роботи було додано вартісну функцію.

Алгоритм пошуку маршруту VFH* вважається модернізацією двох попередніх алгоритмів пошуку маршруту, він виконує перевірку ще раз обраного шляху. Плюсами цього алгоритму пошуку вважається те, що він враховує динаміку і форму МР. Вважається швидким і надійним у середовищі з великою кількістю фізичних перешкод. Метод підходить для роботи з неточною інформацією, завдяки використанню ймовірнісного підходу, але вимагає складних систем технічного оснащення, точних давачів. Незважаючи на очевидні переваги, такі як збіжність і ефективність, метод досить складний у реалізації, а також вимагає великої кількості ресурсів бортового комп'ютера МР - обчислювальної потужності та вбудованої пам'яті. Крім перерахованих також існують нерозв'язні ситуації за наявності U-образних та симетричних перешкод.

Метод динамічного вікна застосовний для МР, що рухаються із високою

швидкістю (наприклад, 75 м/с). За допомогою даного алгоритму пошуку маршруту, обравши оптимальний вектор швидкості з допустимої області простору швидкостей або «динамічного вікна», МР враховує розташування фізичних перешкод, справляється з їхнім обходом у небезпечному і тісному середовищі, також алгоритм пошуку маршруту враховує кінематику МР. Метод динамічного вікна висвітлюється в літературі [28] і, більш глобально, в роботі [4]. Алгоритм пошуку маршруту вибирає і будує відповідні траєкторії руху МР, а також контролює швидкість руху МР, виконується за рахунок вибору допустимої швидкості. Швидкість буде вважатись допустимою, якщо МР у змозі зупинитися перед найближчою фізичною перешкодою або поворотом. Під час наближення до фізичної перешкоди алгоритм пошуку маршруту знижує швидкість руху МР, якщо є можливість прискоритися, то алгоритм пошуку маршруту навпаки збільшує швидкість руху МР. Рішення будуть прийматись динамічно. В основі даного рішення лежить пошук функціоналу в динамічному вікні і враховується критерій оптимальності, що враховує ступінь збігу траєкторії руху МР зі спланованим маршрутом, швидкість власного руху та відстані до фізичних перешкод [29].

Варіація даного алгоритму пошуку маршруту дає змогу МР рухатися без планування маршруту, як для методу глобального динамічного вікна. Його основна відмінність від попереднього методу - це додавання навігаційної функції NF1 до критерію оптимальності, що дає змогу рухатися до кінцевої точки, оминаючи фізичні перешкоди [30]. Така функція виходить під час роботи хвильового алгоритму, з центром у вказаній точці. Функція NF1 у кожній точці являє собою відстань до кінцевої точки, з урахуванням обходу фізичних перешкод. Недоліком такого підходу є необхідність перераховувати навігаційну функцію за будь-якої зміни реальної карти по всій області визначення, що досить трудомістко. Зазначається, що обчислювальна складність алгоритму складає O(N3) [4-6], тому для реалізації необхідні високопродуктивні та дорогі програмовані логічні інтегральні схеми або програмовані логічні контролери.

Відомо, що оригінальний метод динамічного вікна було модифіковано для врахування впливу рухомих фізичних перешкод [31].

1.3 Вимоги до модулю керування IStrategy, що розробляється

Модуль повинен бути реалізований з можливістю адаптації його на МР типу NXT і EV3 на базі елементів Mindstorm та елементів Lego. Модель МР типу NXT оснащена порівняно невеликим обсягом пам'яті - оперативна пам'ять МР NXT 64 Кб, FLASH-пам'ять 256 Кб. Розмір оперативної пам'яті МР типу EV3 64 Мб, а FLASH-пам'ять становить 16Мб. Обидві моделі оснащені давачами дотику та спеціальними ультразвуковими далекомірами. З урахуванням технічних характеристик МР було встановлено наступні вимоги до модуля керування IStrategy:

1. Модуль керування повинен бути із можливістю вивантажування.

2. Модуль керування завантажується в ситуації, коли МР виявлено фізичну перешкоду, яка відсутня на карті.

3. Для орієнтації в навколишньому просторі МР користується такими засобами, як ультразвуковий далекомір і давач дотику.

4. Модуль керування повинен використовувати алгоритм пошуку маршруту, що дає змогу обійти фізичну перешкоду.

- алгоритм пошуку маршруту потребує мінімальної кількості обчислювальних ресурсів;

- алгоритм пошуку маршруту повинен володіти високою продуктивністю;

- алгоритм пошуку маршруту повинен забезпечувати швидке прийняття рішення.

5. Модуль керування здійснює перепланування частини маршруту.

- траєкторія маршруту повинна бути якомога оптимальнішим.

6. Модуль керування завершує роботу і повертає керування модулю IControl, коли вийшов на раніше визначену траєкторію після обходу фізичної

перешкоди.

- у випадку якщо фізичну перешкоду обійти неможливо, модуль керування передає керування іншому модулю IStrategy;

- якщо модуль IStrategy останній у ланцюжку виконання, то він повертає керування модулю IControl з помилковим кодом завершення.

1.4 Постановка завдань для розробки модулю керування рухом МР

Завданнями даної кваліфікаційної роботи є:

- вивчення структури та особливостей платформи MARS;

- вивчення наявних алгоритмів обходу перешкод МР в невизначених умовах;

- розробка алгоритму руху МР для обходу існуючих нерухомих перешкод;

- проєктування модуля керування рухом МР;

- реалізація модуля керування рухом МР на платформі JAVA із урахуванням існуючих особливостей і обмежень платформи MARS;

- перевірка працездатності модуля керування рухом.

1.5 Висновки до першого розділу

Виконано огляд та аналіз існуючих алгоритмів планування руху мобільного робота, встановлено вимоги до модулю керування МР IStrategy, що розробляється.

Виконано постановку завдань для розробки модулю керування рухом МР.

2 РОЗРОБКА ПРОЄКТУ МОДУЛЮ ПЕРЕМІЩЕННЯ МОБІЛЬНОГО РОБОТУ

2.1 Аналіз алгоритмів локального планування

У кожного з алгоритмів локального планування є свої сильні та слабкі сторони. Щоб зробити вибір потрібного алгоритму локального планування, який підходить для створення модуля IStrategy, необхідно проаналізувати і порівняти дані алгоритми локального планування виходячи з встановлених вимог.

Завдання модуля IStrategy - прийняти рішення в ситуації, наприклад, коли на шляху МР з'являється перешкода, не нанесена на створену карту. З таким завданням повинні впоратися алгоритми локального планування, оскільки для обходу перешкоди їм не потрібне чітке знання навколишнього середовища МР, достатньо координат місця розташування МР та координат встановленої цілі.

Порівняльну таблицю алгоритмів локального планування наведено в додатках.

Використовувані апаратні засоби, наприклад, контактний сенсор (bug1, bug2), ультразвуковий сенсор (УС) (DistBug) [21].

Швидкість роботи алгоритму локального планування: повільно. Алгоритм локального планування типу bug 1 - витрата великого проміжку часу на обхід навколо перешкоди, перш ніж отримати оптимальну траєкторію обходу з виходом до встановленої цілі, bug 2, distbug теж витрачають додатковий час на частковий обхід перешкоди, до того ж дані алгоритми завжди обходять перешкоду тільки в певному напрямку, що також негативно впливає на швидкість роботи алгоритму локального планування в цілому.

Обчислювальні ресурси: алгоритм Bug1 - МР повністю обходить перешкоду, і змушений зберігати всі відвідані точки, що затратно з точки зору обчислювальних ресурсів. Алгоритми Bug2, DistBug - порівняно з алгоритмом Bug 1 витрачають менше обчислювальних ресурсів [21].

Результат роботи алгоритму локального планування: МР гарантовано обходить перешкоди і потрапляє в встановлену кінцеву точку, але витративши при цьому більше часу.

Дані алгоритми не забезпечують швидкого прийняття рішення, а також втрачається енергія на непотрібний обхід навколо або частково навколо перешкоди, а траєкторія маршруту у підсумку виходить неоптимальною.

Алгоритм Tangent Bug [21]. Апаратні засоби – УС. Ефективність: ефективний у більшості випадках.

Швидкість роботи алгоритму, обчислювальні ресурси: для зберігання локального графа дотичних не потрібно витрачати значних ресурсів. Алгоритм локального планування працює досить швидко.

МР може потрапити в локальний мінімум, але алгоритм локального планування передбачає спосіб виходу з цієї ситуації. Порівняно з іншими розглянутими алгоритмами локального планування сімейства Bug пропонує оптимальну траєкторію маршруту.

Алгоритми типу VFH, VFH+, VFH*. Апаратні засоби – УС. Ефективність роботи – порівняно низька - розрахунки точні, але потрібен великий обсяг пам'яті та обчислювальних ресурсів.

Швидкість роботи алгоритму локального планування: VFH* працює довше, ніж VFH чи VFH+ за рахунок пошуку шляху за допомогою алгоритму пошуку A* (повний перебір).

Не працює, якщо МР потрапляє в локальний мінімум або при наявності досить вузького проходу між перешкодами.

Метод динамічного вікна. Апаратні засоби: потрібні кілька точних давачів.

Ефективність: ефективний алгоритм, який дає змогу обходити і статичні, і динамічні перешкоди. Швидкість роботи алгоритму: висока, рішення ухвалюються динамічно, при цьому допускається висока швидкість руху МР.

МР може потрапити в локальний мінімум, потрібні доповнення до алгоритму.

Метод знаходження проміжків. Апаратні засоби - УС, лідар сенсор або LIDAR англ. Light Identification Detection and Ranging - світлове виявлення і визначення дальності - технологія отримання та обробки інформації про віддалені об'єкти за допомогою активних оптичних систем, які використовують явища відбиття світла і його розсіювання в прозорих і напівпрозорих середовищах) [32].

Ефективність: досить ефективний, завжди пропонує найкоротший шлях, вміє оминати симетричні перешкоди та уникати локальних мінімумів.

Швидкість роботи алгоритму: працює досить швидко, бо ухвалює рішення, ґрунтуючись на невеликій кількості даних, які отримує динамічно.

МР повинен бути оснащений чутливим давачем, не виконує обхід U-подібних перешкод.

Алгоритми пошуку типу D*, D*Lite. Апаратні засоби - потрібні кілька точних давачів.

Ефективність: висока, алгоритми пропонують найкоротший шлях.

Швидкість роботи алгоритму - при наявності достатніх обчислювальних ресурсів працює досить швидко.

Витратний алгоритм у плані обчислювальних ресурсів, необхідно зберігати в пам'яті всі сегменти створеної карти.

Проаналізувавши алгоритми локального планування, такі як Bug1, Bug2, Tangent Bug, метод динамічного вікна, гістограми векторних полів, метод проміжків, алгоритм пошуку D* з точки зору використовуваних апаратних засобів, ефективності, швидкості та тривалості роботи алгоритму, вміння вирішувати ситуації локального мінімуму, було прийнято рішення використовувати алгоритм створений на основі алгоритму Tangent Bug [21].

Алгоритм Tangent Bug [21] використовує УС, що відповідає встановленим вимогам і пропонує ефективне рішення в ситуаціях, коли потрібно виконати обхід U-подібної перешкоди, що не вміють робити багато алгоритмів локального планування, при цьому відрізняється прийнятно високою швидкістю роботи і не

вимагає значних обчислювальних ресурсів.

2.2 Характеристика алгоритму Tangent Bug

Алгоритм обходу перешкоди повинен бути створений із урахуванням можливості адаптації його до систем роботів типу NXT чи EV3, тому цей алгоритм обходу перешкод повинен реалізувати інтерфейс модуля IStrategy.

На рисунку 2.1 представлено узагальнену схему програмного модуля вузла платформи MARS [1-3].

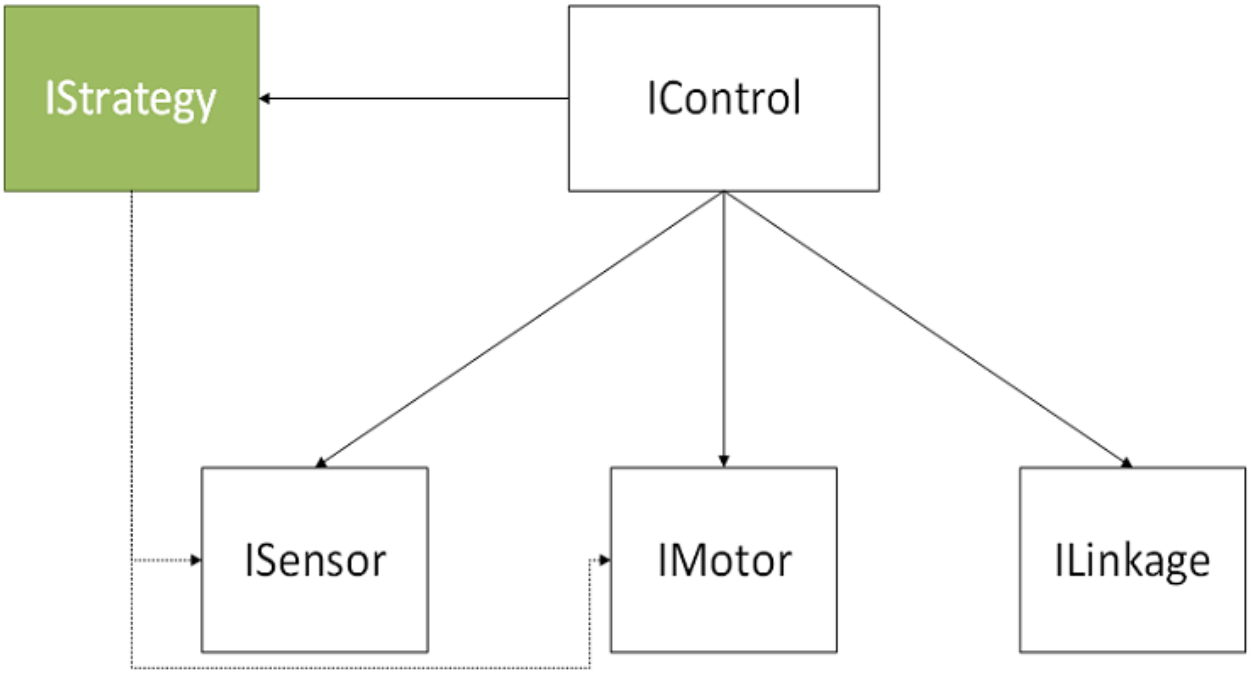


Рисунок 2.1 – Спрощена схема програмного модулю вузла платформи MARS

Модуль IStrategy відповідає за стратегію поведінки МР і викликається модулем керування IControl. Модуль IStrategy пропонує рішення, використовуючи алгоритм обходу перешкоди і руху до встановленої цілі (ми будемо використовувати алгоритм Tangent Bug [21]).

Компоненти програмного модуля повинні виконувати наступні функції:

- модуль IControl - модуль керування МР, виконує сполучну функцію між

усіма модулями керування;

- ILinkage – модуль для комунікації;

- IMotor - модуль, що керує рухом, містить набір команд керування МР;

- ISensor - здійснює отримання даних із давачів та відповідає за первинну оброблення.

За допомогою алгоритму локального пошуку Tangent Bug [21] МР здатний переміщуватись в частково знайомому середовищі, в якому знаходяться нерухомі перешкоди. Алгоритм локального пошуку Tangent bug [21] - локальний алгоритм, який використовує дані, отримані за допомогою давачів, для побудови, так званого, локального графа дотичних. Локальний граф дотичних необхідний для вибору траєкторії руху МР [33].

Граф дотичних є підграфом графа видимості. Граф видимості - граф, вершинами якого є вершини перешкод-багатокутників, а ребрами з'єднані тільки вершини, які взаємно видні.

Граф видимості складається з великої кількості ребер, зокрема тих, які навіть не входять до складу найкоротшого шляху.

Як і граф видимості, граф дотичних містить у собі найкоротший шлях у даному середовищі, але порівняно з графом видимості він мінімальний і оптимальний, оскільки містить набагато меншу кількість ребер та вершин. Таким чином, пошук найкоротшого шляху буде більш ефективним на графі дотичних. На рисунку 2.2 представлено загальний вигляд графів видимості та дотичних.

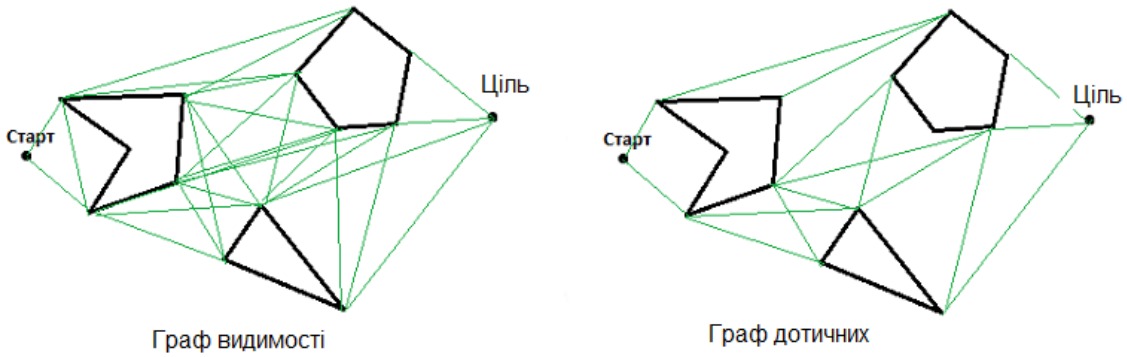


Рисунок 2.2 – Загальний вигляд графів дотичних та видимості

Локальний граф дотичних

Методу локального пошуку Tangent Bug не потрібне глобальне знання про навколишнє середовище, брак цього знання компенсується тим, що алгоритм генерує локальний граф дотичних. Локальний граф дотичних - це граф дотичних, який містить лише ті перешкоди, що знаходяться

у межах видимості робота, що спрощує обробку даних. Локальний граф дотичних зображений на малюнку 2.3.

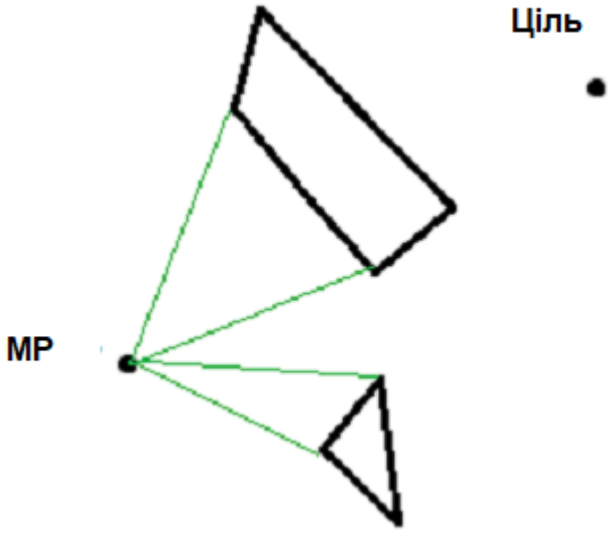


Рисунок 2.3 – Загальний вигляд локального графу дотичних

Інформація про відстань може бути представлена як функція $r(\Theta)$, де Θ - кут, що відповідає певному напрямку, і $r(\Theta)$ - відстань у напрямку Θ від місця розташування MP, наприклад, x до найближчої точки на досяжній межі перешкоди. Використовуючи дані вимірювань, можна виділити окремі перешкоди, ґрунтуючись на тому, що відстань змінюється безперервно вздовж межі видимої перешкоди. Функція $r(\Theta)$ має розриви, які знаходяться на дотичних до країв перешкод. Інтервал $(r(\alpha), r(\beta))$, функція дальності на якому змінюється безперервно, і межі якого є точками розриву функції $r(\Theta)$ або точки, у яких $r(\Theta) = R$ (де R - максимальна відстань, що може бути виміряна давачем), вважається виявленою перешкодою [34].

Вузлами локального графа дотичних є наступні точки: поточне положення МР, кінцеві точки виявлених перешкод і (опціонально) додатковий вузол на напрямку до встановленої цілі - T_{node}. Якщо на відрізку від місця положення МР до встановленої цілі в межах видимості МР відсутні перешкоди, то T_{node} це найдальший вузол. Ребра графа дотичних з'єднують поточне положення МР з усіма іншими вузлами локального графа. Локальний граф дотичних також містить ребра, які з'єднують вершини, які є взаємовидимі, але алгоритм пошуку не використовує дані ребра.

Локальний граф дотичних генерується наступним чином.

Збираються дані необхідні для функції $r(\Theta)$ або вільний простір у напрямку Θ. Це відстань між МР і першою видимою перешкодою, Θ - це напрямок до встановленої цілі та F - вільний простір у напрямку до встановленої цілі. Необхідно враховувати, що $F = r(\Theta)$.

Функція $r(\Theta)$ повертає відстань до першої видимої перешкоди в заданому напрямку Θ. Потім, функція $r(\Theta)$ обробляється за наступними правилами:

- якщо встановлена ціль не видима, але на траєкторії до неї немає видимих перешкод, створюється вузол на напрямку до встановленої цілі;

- якщо встановлена ціль видима, то створюється вузол на точці, де розташований фініш;

- функція $r(\Theta)$ перевіряється на порушення безперервності. Якщо виявлено розрив у точці Θ, то в цьому напрямку створюється вузол.

- якщо $r(\Theta) = R$ (де R - максимальна відстань, що вимірюється за допомогою давача) та $r(\Theta)$ послідовно спадає, створюється вузол на напрямку Θ. Також, якщо $r(\Theta) \neq r$ та $r(\Theta)$ послідовно зростає, так що $r(\Theta) = r$, створюється вузол у напрямку Θ.

Метод локального пошуку Tangent Bug [21] використовує два основні типи поведінки - рух до встановленої цілі та рух уздовж границі перешкоди. На кожному кроці МР будує спеціальний локальний граф дотичних, заснований на

поточних даних про відстань, отриманих за допомогою давачів. Під час руху до встановленої цілі МР рухається в локальному оптимальному напрямку, тобто по траєкторії найкоротшого шляху згідно з підграфом локального графа дотичних.

Якщо МР потрапляє в локальний мінімум, то він перемикається на режим руху вздовж границі перешкоди і рухається в даному режимі, використовуючи локальний граф дотичних для знаходження локальної траєкторії найкоротшого шляху. МР не покидає границі перешкоди доти, доки не буде виконано умову виходу. МР рухається уздовж кордону, фіксуючи мінімальну відстань до встановленої цілі - $d_{\min}(T)$. МР залишає границю перешкоди, коли в локальному графі дотичних з'являється вузол V_{leave}, який відповідає умові $d(V_{leave},T) \le d_{\min}(T)$.

Покинувши границю перешкоди, МР потрапляє у фазу переходу, перш ніж він знову перейде в режим руху до встановленої цілі. У фазі переходу МР рухається до вузла V_{leave}, поки не досягне встановленої точки Z, у якій виконується умова $d(Z,T) \le d_{\min}(T)$. У вказаній точці МР знову переходить у режим руху до встановленої цілі.

Підіб'ємо підсумок в описі алгоритму [34]:

Крок 1. Виконувати рух в бік встановленої цілі, дотримуючись локального оптимального маршруту на поточному підграфі локального графа дотичних, доки не відбудеться одна з наступних подій:

- встановлена ціль досягнута. Стоп.

- виявлено локальний мінімум. Виконання Кроку 2.

Крок 2. Перехід у режим руху вздовж границі перешкоди. Рухатися вздовж границі, використовуючи локальний граф дотичних, одночасно фіксуючи $d_{\min}(T)$, поки не відбудеться одна з наступних подій:

- встановлена ціль досягнута. Стоп;

- виконується умова виходу: існує вузол V_{leave}, що належить локальному графу дотичних, такий що $d(V_{leave},T) \le d_{\min}(T)$. Виконання Кроку 3;

- МР обійшов навколо перешкоди, встановлена ціль недосяжна. Стоп.

Крок 3. Виконати фазу переходу. Рухатися у напрямку до V_{leave}, поки не буде досягнута встановлена точка Z, в якій виконується умова $d(Z,T) \le d_{\min}(T)$.

2.3 Коригування в алгоритмі локального пошуку Tangent Bug [21]

В оригінальному алгоритмі локального пошуку локальний граф дотичних генерується постійно в процесі руху МР. У відкоригованій реалізації локальний граф дотичних генерується, коли МР досягає певних вузлів. Таке спрощення дає змогу трохи заощадити ресурси.

Також у даній роботі адаптовано частину алгоритму локального пошуку [21], яка відповідає за рух МР вздовж границі перешкоди. У даній частині алгоритму МР не будує граф дотичних, рухаючись уздовж стіни, а порівнює дві перемінні Dfollowed та Dreach. Перемінна Dreach ініціалізується перед входом у частину алгоритму локального пошуку, що реалізує рух уздовж стіни. Потім, рухаючись уздовж стіни, МР порівнює значення перемінної Dreach зі перемінною Dfollowed.

Перемінна Dfollowed зберігає в собі відстань до перешкоди від точки, що знаходиться на границі перешкоди. Умовою виходу з алгоритму локального пошуку є: значення перемінної Dfollowed менше Dreach. На рисунку 2.4 представлено відстані, які задаються перемінними Dreach і Dfollowed.

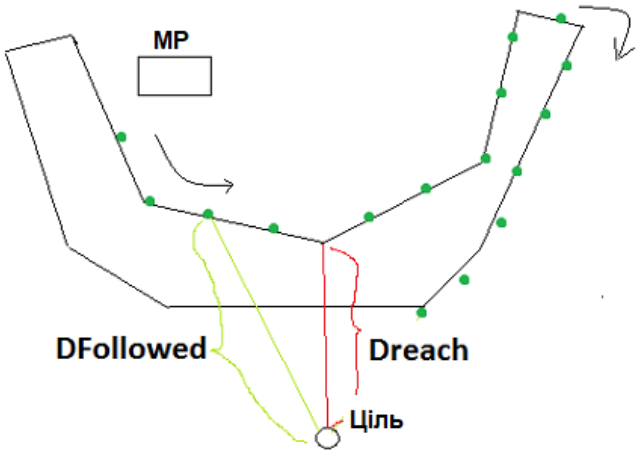


Рисунок 2.4 – Загальний вигляд перемінних Dreach та Dfollowed

Описана вище адаптація дає змогу заощадити ресурси на побудові локального графа дотичних.

2.4 Висновки до другого розділу

Алгоритм локального пошуку Tangent Bug [21] дає змогу MP не тільки обійти ненанесену на створену карту перешкоду, а й дійти до встановленої цілі. Тому, за необхідності, після обходу перешкоди можна не віддавати дію модулю IControl, а продовжувати виконання руху до встановленої мети.

Для реалізації алгоритму локального пошуку буде корисна невелика корекція – дані із давачів збиратимуться не постійно, а на певних кроках, у режимі руху вздовж стіни не будуватиметься локальний граф дотичних, що дасть змогу суттєво заощадити ресурси.

3 РОЗРОБКА МОДУЛЯ ПЕРЕМІЩЕННЯ МОБІЛЬНОГО РОБОТУ

Модуль буде реалізовано на мові програмування JAVA [35]. В якості алгоритму обходу перешкоди буде використовуватись алгоритм, заснований на алгоритмі локального пошуку Tangent Bug [21].

На рисунку 3.1 наведено скорочену діаграму класів модуля IStrategy. Повна діаграма класів представлена на рисунках 3.2-3.3.

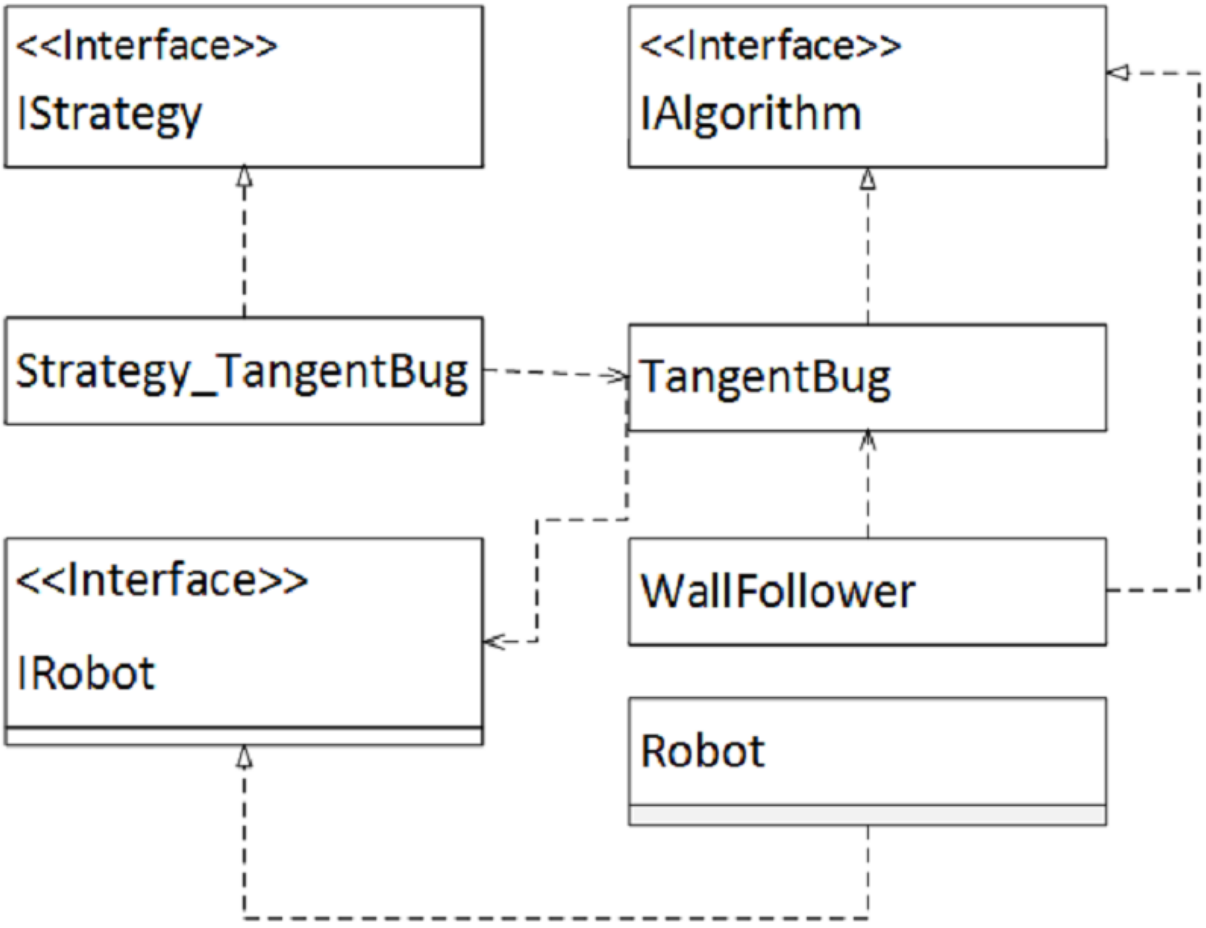


Рисунок 3.1 – Скорочена діаграма класів модуля IStrategy

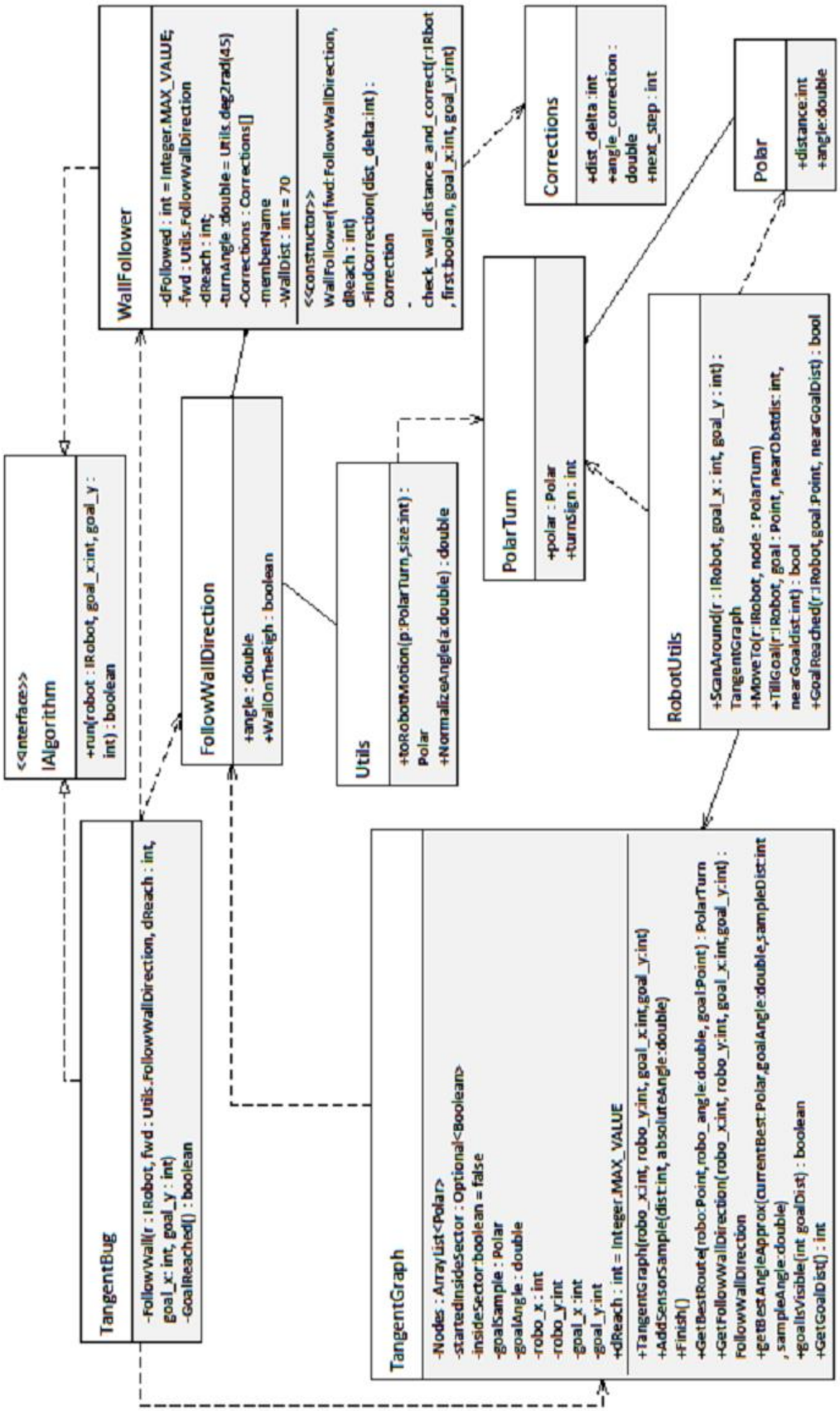


Рисунок 3.2 – Діаграма класів модуля IStrategy

Рисунок 3.3 – Діаграма класів модуля IStrategy

3.1 Розробка інтерфейсів та класів модулю IStrategy

Інтерфейси та класи модуля IStrategy

Інтерфейс IStrategy. Даний інтерфейс використовується для запуску модуля IStrategy. Функція запуску цього модуля run повертає логічне значення, що сигналізує про успішність виконання (див. рис. 3.1).

Клас Strategy_TangentBug (рис. 3.1-3.3). Клас являє собою реалізацію інтерфейсу IStrategy, яка використовує алгоритм локального пошуку Tangent Bug в якості алгоритму обходу перешкоди.

Інтерфейс IAlgorithm (рис. 3.1-3.3). Інтерфейс дає змогу запускати будь-які алгоритми, які як аргумент приймають об'єкти, що реалізують інтерфейс IRobot, і координати встановленої цілі.

Інтерфейс IRobot (рис. 3.1-3.3). Інтерфейс містить у собі основні функції керування МР, такі як рух уперед на вказану відстань, обертання на вказаний кут, зупинка руху та обертання, отримання координат МР, отримання даних про відстань до перешкоди і кута повороту МР відносно координатної осі.

Клас Robot (рис. 3.1-3.3). Реалізує інтерфейс IRobot для програмної симуляції руху МР в обраному середовищі симулятору, наприклад RobotSim [36].

Клас TangentBug (рис. 3.1-3.3). Даний клас виконує інтерфейс IAlgorithm. В початку роботи алгоритму локального пошуку Tangent Bug [21] запускається функція TillGoal. Дана функція запускає рух МР в напрямку встановленої цілі. Якщо встановлена ціль досягнута, або МР зустрів перешкоду, функція TillGoal завершується з відповідним кодом повернення. Якщо МР зустрів перешкоду, він будує локальний граф дотичних за допомогою функції ScanAround (рис. 3.2-3.3).

Побудова графа дотичних за допомогою функції ScanAround виконується наступним чином: МР виконує повний оберт на місці навколо своєї осі, водночас безперервно близько 100 разів за секунду знімає показники давача дальності й у такий спосіб визначає напрямки на границі перешкод, аналізуючи розриви в

отриманих значеннях функції дальності.

Результат роботи функції ScanAround розміщується в масиві (Nodes) у вигляді пар полярних координат, перша з яких вказує на початок перешкоди, друга - на кінець перешкоди. Потім за допомогою спеціальної функції GetBestRoute виконується пошук найкращого шляху в цьому графі. Якщо функція GetBestRoute повертає null, це означає, що МР перебуває в локальному мінімумі, у такому разі вмикається алгоритм руху вздовж стіни WallFollower (рис. 3.3).

В іншому випадку функція GetBestRoute повертає найкращий з точки зору алгоритму локального пошуку Tangent Bug напрямок, за яким МР виконує рух, запускаючи функцію MoveTo. При цьому обраний напрямок руху і відстань трохи коригуються з урахуванням габаритних розмірів МР таким чином, щоб МР проходив трохи осторонь від границі перешкоди, а не впирався в неї.

Клас TangentGraph (рис. 3.1-3.3). У даному класі реалізовано локальний граф дотичних. Локальний граф дотичних представлений за допомогою структури ArrayList, з полярними координатами, що зберігаються в ній, - об'єктами класу Polar. Вибір полярних координат зумовлений високою зручністю, оскільки під час сканування перешкод виходять готові полярні координати для МР.

Клас WallFollower (рис. 3.2-3.3). Даний клас реалізує частину алгоритму локального пошуку, яка працює тільки в тому випадку, якщо МР потрапляє в так званий, локальний мінімум. За допомогою даного класу виконується рух уздовж стіни. МР виконує обрахунок необхідного кута повороту і повертається вздовж перешкоди. Давач дальності повертається під кутом 45 градусів до перешкоди. МР починає виконувати рух, і, періодично контролюючи показники давача дальності, намагається підтримувати постійну відстань від встановленої перешкоди: якщо виміряна відстань вища за необхідну, МР трохи повертає до стіни, якщо ж вона нижча, МР трохи повертає від стіни. Величина кута повороту при цьому залежить від різниці підтримуваної та виміряної відстаней: чим

більша різниця, тим більший кут на який повертається МР.

Оскільки в процесі моделювання немає можливості повертати давач незалежно від МР, то під час руху вздовж границі перешкоди для контролю відстані до стіни МР періодично буде повертатись разом із давачем і після визначення відстані повертається назад з урахуванням корекції кута.

Класи Utils і RobotUtils (рис. 3.3). Клас Utils допоміжний клас, містить різні допоміжні функції. Клас RobotUtils - різні допоміжні функції, що стосуються руху МР й отримання даних про наявні перешкоди [36].

Математичний опис алгоритму локального пошуку Tangent Bug [21] не враховує лінійні габаритні розміри МР та похибку у вимірах відстаней і координат, тому дана реалізація передбачає методи, що дають змогу врахувати вказані вимоги. За допомогою наступних методів коригуються:

- точка встановленої цілі, точка початку руху вздовж стіни - під час порівняння координат МР з координатами встановленої цілі та з координатами початку руху вздовж стіни допускається не зовсім точна рівність вказаних координат;

- рух МР до вузлів графа дотичних - МР виконує рух не прямо до вузла, а з невеликим відхиленням від нього, щоб не зіткнутися з відповідною перешкодою, при цьому МР повинен просунутися трохи далі від вузла, щоб поліпшити огляд навколишнього середовища;

- визначення розривів функції дальності - у випадку, коли в межах видимості МР знаходяться дві окремих перешкоди, що накладаються одна на одну, під час аналізу розриву функції дальності r(Ѳ) враховуються тільки ті розриви, перепад дальності яких більший за габаритні розміри МР з деяким запасом. Такий випадок наведено на рисунку 3.4.

У реалізації, крім інтерфейсу IStrategy, використовуються додаткові інтерфейси IRobot і IAlgorithm, що дозволить в свою чергу легко додавати інші реалізації алгоритмів локального пошуку і змінювати реалізацію інтерфейсу IRobot, наприклад, для запуску алгоритму локального пошуку на реальному МР,

а не в режимі моделювання [36].

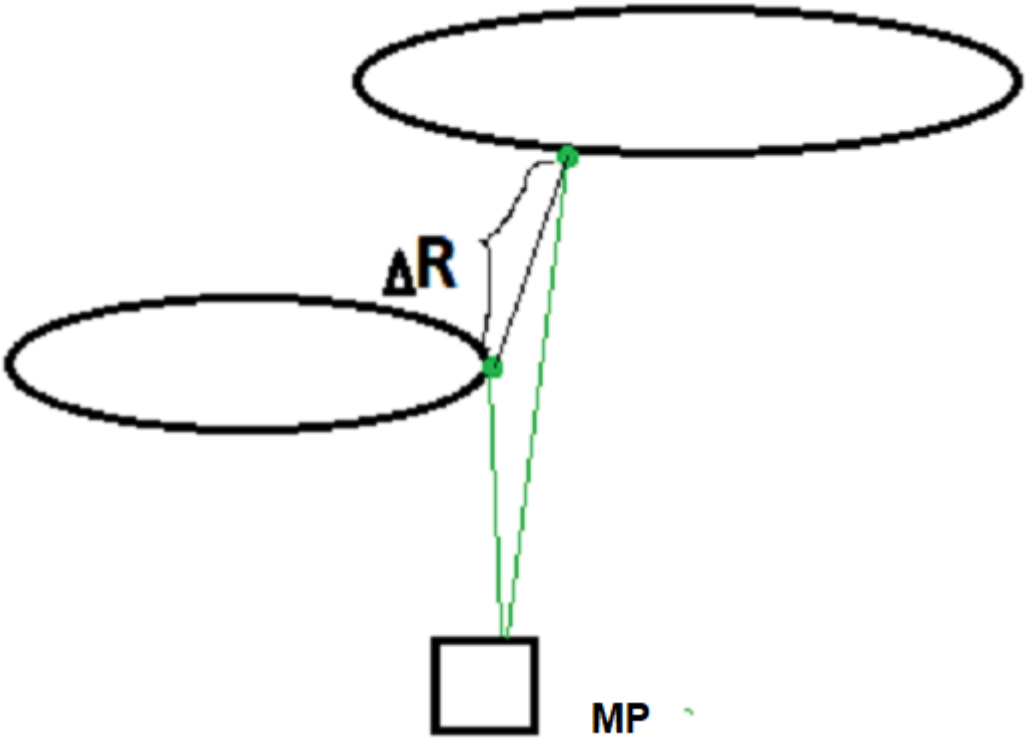


Рисунок 3.4 – Приклад визначення розриву функції дальності у випадку з перешкодами, що накладаються

Оскільки МР має ненульові габаритні розміри, які необхідно враховувати під час виконання руху, проводиться корекція обчислених алгоритмом локального пошуку Tangent Bug значень з урахуванням габаритних розмірів МР. Обчислення даних корекцій становить істотну частину даної реалізації алгоритму локального пошуку.

3.2 Реалізація розробленого алгоритму переміщення мобільного робота

Модульна реалізація

Для реалізації основних функцій алгоритму локального пошуку Tangent Bug були написані модульні тести з використанням спеціальної бібліотеки JUnit. Основна увага була приділена найбільш важливій та алгоритмічно складній частині створеної програми - обробці результатів сканування, побудові

локального графа дотичних для різних варіантів розташування фізичних перешкод.

Функціональна реалізація була виконана в умовно безкоштовному емуляторі типу RobotSim [36].

```
package robo;
public class Interfaces {
 public interface IStrategy {
 boolean run();
 }
 public interface IAlgorithm {
 boolean run(IRobot r, int goal_x, int goal_y);
 }
 public interface IRobot {
 static public class Params {
 int RobotSize; // лінійні розміри МР. Враховуються при виконанні корекції руху.
 int OneTurnTime; // час повного оберту навколо власної осі. Виконується для обертання на заданий кут
 }
  Params get_params();
 // рух вперед на вказану відстань та зупинка
 void forward(int dist);
 // почати рух вперед
 void forward();
// виконати поворот на заданий кут і зупинитись
 void rotate(double angle);
 // почати обертання, до тих пір поки не буде викликана функція
stop()
 void rotate();
 // зупинити обертання та рух
 void stop();
 // отримати відстань до перешкоди
 // повертати -1, у випадку відсутності фізичної перешкоди
 int get_distance();
 // координати МР
 int get_x();
 int get_y();
 // поточний кут розташування МР відносно координатної осі
 double get_angle();
 }
}
```

Рисунок 3.5 – Приклад коду для керування МР

Для функціональної реалізації було змодельовано різні типи фізичних перешкод: горизонтальна перешкода (рис. 3.7), хрестоподібна перешкода (рис. 3.8), U-подібна перешкода (рис. 3.10), Т-подібна перешкода (рис. 3.11), лабіринт (рис. 3.13), равлик (рис. 3.15) та моделювання ситуації недосяжності мети (рис. 3.17).

```
import robo.Interfaces.IRobot;
public class Main {
 // Реалізація IStrategy, використовує алгоритм локального пошуку Tangent Bug
 static class Strategy_TangentBug implements Interfaces.IStrategy {
 final IRobot robot;
 final int goal_x;
 final int goal_y;
 Strategy_TangentBug(IRobot r, int goal_x, int goal_y) {
 robot = r;
 this.goal_x = goal_x;
 this.goal_y = goal_y;
 }
 // Запуск алгоритму локального пошуку Tangent Bug
 @Override
 public boolean run() {
 final Interfaces.IAlgorithm tb = new TangentBug();
 final boolean tbSuccess = tb.run(robot, goal_x, goal_y);
 if (tbSuccess) {
System.out.println("TangentBug SUCCESS");
 } else {
 System.out.println("TangentBug FAILURE");
 }
 return tbSuccess;
 }
 }
 public static void main(String[] args) {
 try {
 //final IRobot r = new Simulator. Robot(250, 20); // U
 //final IRobot r = new Simulator.Robot(170, 20); // Plus
 final IRobot r = new Simulator.Robot(170, 20); // Square
 //final IRobot r = new Simulator.Robot(170,20); // T
 //final IRobot r = new Simulator.Robot(50,50); // Lab
 //final IRobot r = new Simulator.Robot(250,300); // Snail
 Tools.delay(1000);
 final Date startDate = new Date();
 Interfaces.IStrategy strategy = new Strategy_TangentBug(r,
goal_x, goal_y);
 boolean success = strategy.run();
 if (success) {
 System.out.println("IStrategy SUCCESS");
 } else {
 System.out.println("IStrategy FAILURE");
 }
 final Date endDate = new Date();
// розраховуємо час для проходження фізичних перешкод
 final long diff = (endDate.getTime() - startDate.getTime()) /
1000;
 System.out.println("Time: "+diff+" seconds");
 } catch (Exception ex) {
 System.out.println("Exception: " + ex);
 }
 }
```

Рисунок 3.6 – Приклад коду для запуску алгоритму локального пошуку [36]

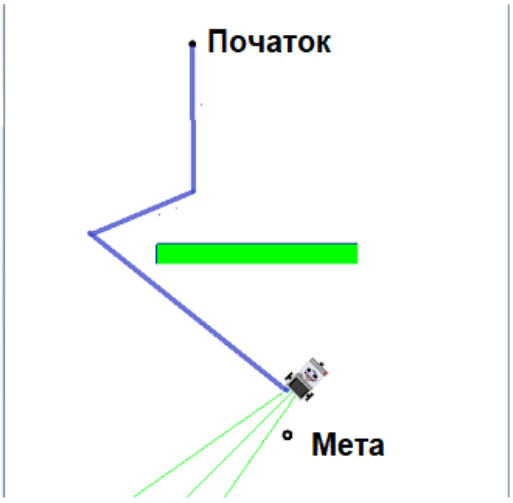


Рисунок 3.7 – Моделювання фізичної горизонтальної перешкоди

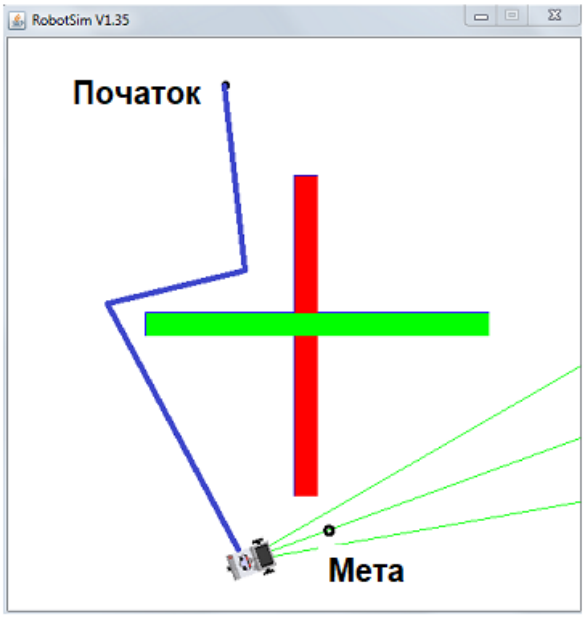


Рисунок 3.8 – Моделювання фізичної хрестоподібної перешкоди

```
// Хрестоподібна фізична перешкода
if (false) {
addObstacleBar(250, 120, 20, 280, Color.red);
addObstacleBar(120, 240, 300, 20, Color.green);
}
```

Рисунок 3.9 – Приклад коду для створення хрестоподібної перешкоди [36]

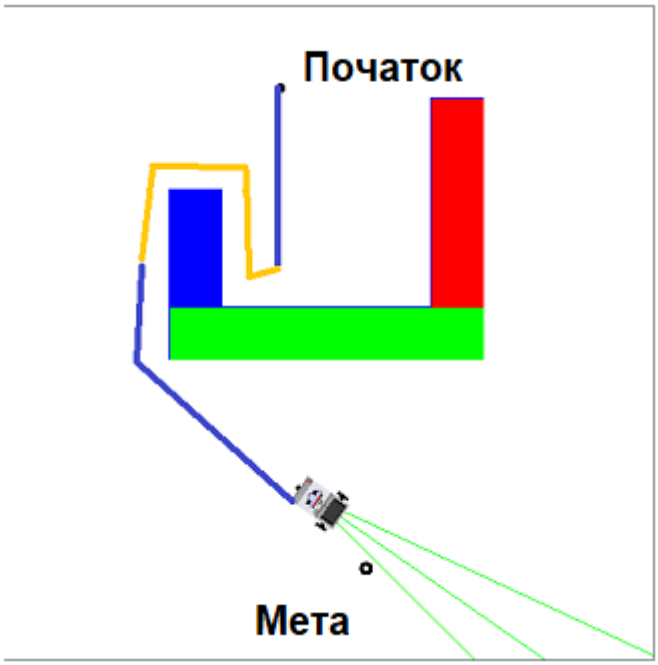


Рисунок 3.10 – Моделювання фізичної U-подібної перешкоди

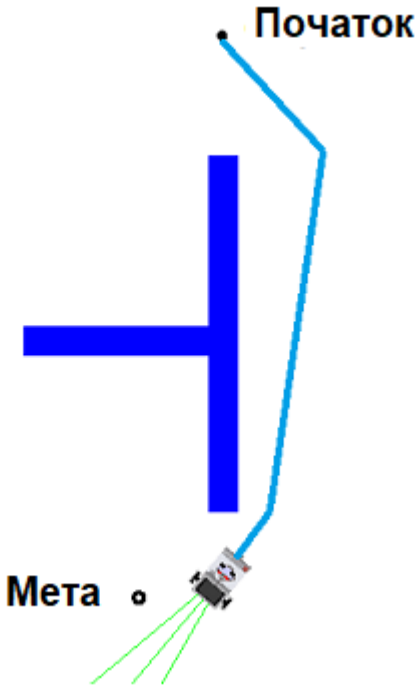


Рисунок 3.11 – Моделювання фізичної Т-подібної перешкоди

```
//Т-подібна фізична перешкода
 if (false) {
 addObstacleBar(250, 120, 20, 250, Color.blue);
 addObstacleBar(120, 240, 150, 20, Color.blue);
}
```

Рисунок 3.12 – Приклад коду для створення Т-подібної перешкоди [36]

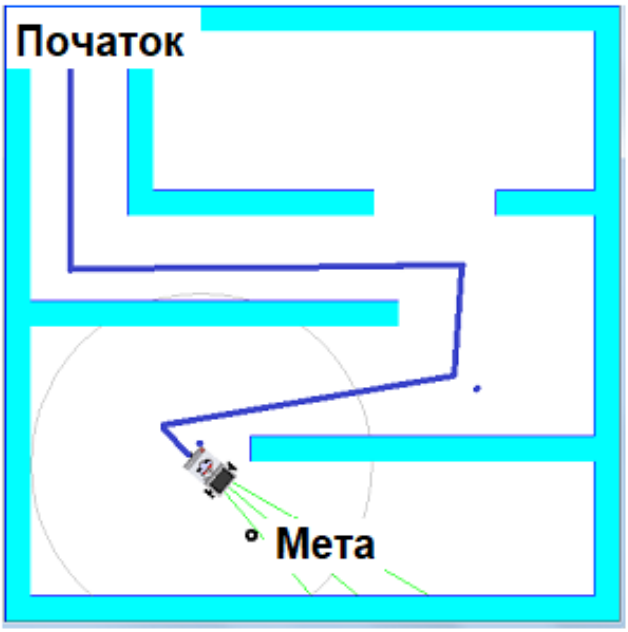


Рисунок 3.13 – Моделювання фізичної перешкоди типу лабіринт

```
//Перешкода типу Лабіринт
if (false) {
addObstacleBar(0, 0, 20, 600, Color.CYAN);
addObstacleBar(0, 0, 600, 20, Color.CYAN);
addObstacleBar(480, 0, 20, 600, Color.CYAN);
addObstacleBar(0, 480, 600, 20, Color.CYAN);
addObstacleBar(20, 240, 300, 20, Color.CYAN);
addObstacleBar(100, 0, 20, 150, Color.CYAN);
addObstacleBar(100, 150, 200, 20, Color.CYAN);
addObstacleBar(400, 150, 150, 20, Color.CYAN);
addObstacleBar(200, 350, 400, 20, Color.CYAN);
}
```

Рисунок 3.14 – Приклад коду для створення перешкоди типу лабіринт

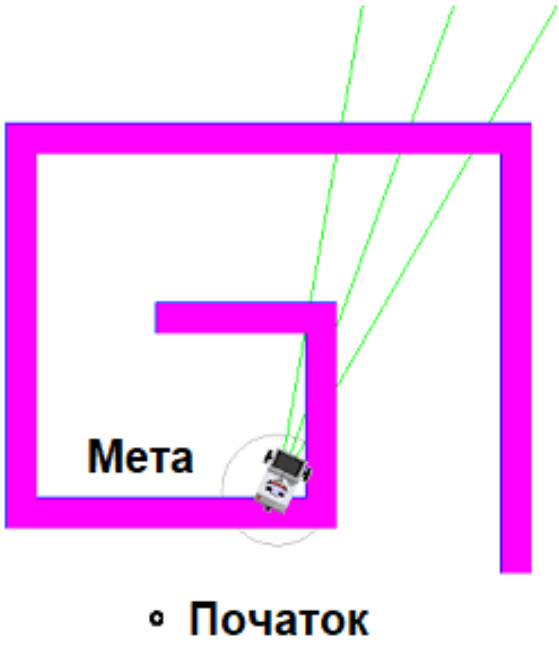


Рисунок 3.15 – Моделювання фізичної перешкоди типу равлик

```
// перешкода типу равлик
if (false) {
addObstacleBar(100, 100, 20, 250, Color.MAGENTA);
addObstacleBar(100, 350, 200, 20, Color.MAGENTA);
addObstacleBar(300, 220, 20, 150, Color.MAGENTA);
addObstacleBar(200, 220, 100, 20, Color.MAGENTA);
addObstacleBar(100, 100, 350, 20, Color.MAGENTA);
addObstacleBar(430, 100, 20, 300, Color.MAGENTA);
}
}
```

Рисунок 3.16 – Приклад коду для створення перешкоди типу равлик

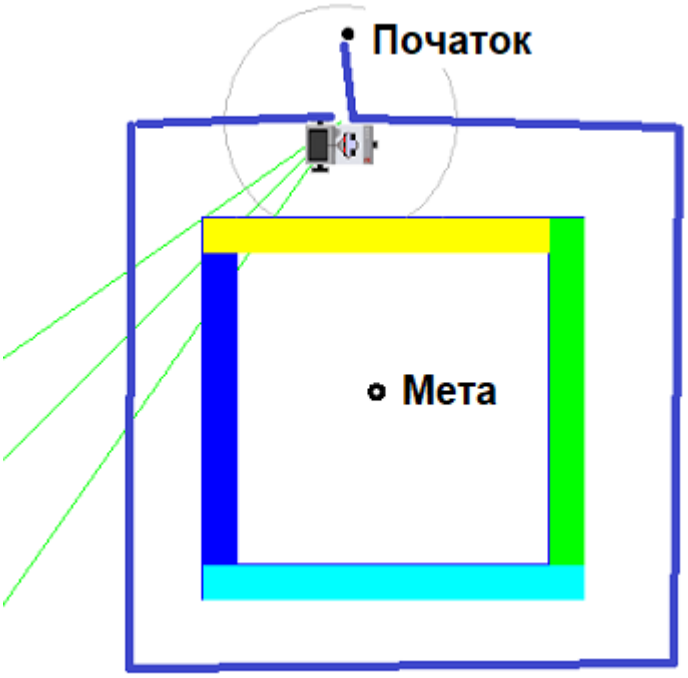


Рисунок 3.17 – Моделювання ситуації недосяжності мети

```
 // вказання координати мети
 static final int goal_x = 280;
 static final int goal_y = 430;
// Координати мети, для квадратної перешкоди
 //static final int goal_x = 250;
 //static final int goal_y = 250;
 static {
 addObstacles();
 RobotContext.useObstacle(circle(5, Color.black), goal_x, goal_y);
//draw a black small circle on a TARGET
 }
 private static GGBitmap circle(int radius, Color color) {
 GGBitmap bm = new GGBitmap(2 * radius, 2 * radius);
 bm.setPaintColor(color);
  bm.setLineWidth(3);
 bm.drawCircle(new Point(radius, radius), radius - 1);
 return bm;
 }

 // прямокутник
 private static GGBitmap bar(int width, int length, Color color) {
 GGBitmap bm = new GGBitmap(width, length);
 bm.setPaintColor(color);
 bm.fillRectangle(new Point(0, 0), new Point(width - 1, length -
1));
 return bm;
 }
}
```

Рисунок 3.18 – Приклад коду для задачі координат мети руху МР

```
package robo;
import ch.aplu.robotsim.Tools;
import java.awt.Point;
import robo.Interfaces.IRobot;
/*
Різні допоміжні функції для руху МР
*/
public class RobotUtils {
 // Відсканувати навколишнє середовище та створити граф дотичних
 // під час сканування зберігати відстань до мети
 static TangentGraph ScanAround(IRobot r, int goal_x, int goal_y) {
 System.out.println("ScanAround: start...");
 TangentGraph tg = new TangentGraph(r.get_x(), r.get_y(), goal_x,
goal_y);
 final double robot_angle = r.get_angle();
 tg.AddSensorSample(r.get_distance(), robot_angle);
 Tools.startTimer(); // для розрахунку поточного кута повороту МР
 r.rotate();
 while (Tools.getTime() < r.get_params().OneTurnTime) {
 int d = r.get_distance(); // відстань до найближчої перешкоди
 // поточний кут повороту МР
 final double robotPovAngle = 2 * Math.PI * Tools.getTime() /
r.get_params().OneTurnTime;
final double absoluteAngle = robotPovAngle + robot_angle;
 tg.AddSensorSample(d, absoluteAngle);
 Tools.delay(10); //100 раз в секунду зняття даних
 }
 r.stop();
 tg.Finish();
 System.out.println(" - ScanAround: end");
 Tools.delay(1000);
 return tg;
 }
 // виконання корекції з урахуванням розмірів МР
 static void MoveTo(IRobot r, Utils.PolarTurn node) {
 final Utils.Polar motion = Utils.ToRobotMotion(node,
r.get_params().RobotSize);
 final double deltaAngle = motion.angle - r.get_angle(); // перевод абсолютного кута в кут відносно положення МР
r.rotate(deltaAngle);
 r.forward(motion.distance);
 }
 // рух в бік мети або фізичної перешкоди
 // повертає true, якщо мета досяжна, false якщо на траєкторії є перешкода
 static boolean TillGoal(IRobot r, final Point goal, int
nearObstacleDist, int nearGoalDist)
 {
 System.out.println("TillGoal...");
 final double absolute_angle = Utils.ComputeAngle(r.get_x(),
r.get_y(), goal.x, goal.y); //знайти напрямок повороту. кут повороту МР відносно мети
 r.rotate(absolute_angle - r.get_angle()); // знаходимо кут відносно положення МР та робимо поворот
// починаємо рух до мети
 r.forward();
 while (true) {
 if (PointReached(r, goal, nearGoalDist)) {
 // мета досягнута
 r.stop();
 System.out.println(" - TillGoal: goal reached");
 return true;
 }
 // мета поки не досяжна, контролюємо відстань до перешкоди
final int d = r.get_distance();
 if (d >= 0 && d <= nearObstacleDist) { //досягли перешкоди
 r.stop();
 System.out.println(" - TillGoal: obstacle");
  return false;
 }
 Tools.delay(10);
 }
 }
 // чи знаходиться МР біля точки point, з урахування погрішності nearDist
 static boolean PointReached(IRobot r, final Point point, int
nearDist) {
 return Math.abs(r.get_x() - point.x) <= nearDist
 && Math.abs(r.get_y() - point.y) <= nearDist;
 }
}
```

Рисунок 3.19 – Приклад коду по введенню допоміжних функцій для МР

```
package robo;
import ch.aplu.robotsim.*;
import java.awt.Color;
import robo.Interfaces.IRobot;
/*
Simulator-specific implementations of some interfaces
*/
public class Simulator {
 static class Robot implements IRobot {
 private final IRobot.Params params;
 {
 params = new IRobot.Params();
 params.RobotSize = 10; // Інтервал від МР до перешкоди, для того, врахувати розміри МР
 //для виконання коректування функції руху
 params.OneTurnTime = 2180; //в емулятору є недолік - неможливо вказати кут повороту.
  //відомо, що МР виконує оберт за 2 секунди
 }
 @Override
 public IRobot.Params get_params() {
 return params;
 }
 final private static int Speed = 50;//-1; // одиниця виміру швидкості - юніти в секунду. Юніти - координати монітору X та Y.
private final Gear gear;
 private final UltrasonicSensor sensor;
 private final LegoRobot robot;
 private double angle = 0; //поточний кут повороту МР
 @Override
 public int get_x() {
 return gear.getX();
 }
 @Override
 public int get_y() {
 return gear.getY();
 }
 @Override
 public double get_angle() {
 return angle;
 }
 //вмикає поворот МР в напрямку збільшення кута
 @Override
 public void rotate() {
 gear.right();
 }
//функція отримує дані з УЗ-сенсору/ повертає -1 якщо мета не знаходиться в границях видимості
 @Override
 public int get_distance() {
 int d = sensor.getDistance();
 //return (d==255 ? -1 : d); //для МР Лего
  //return (d>100 ? -1 : d); // обмеження давача дальності
 return d;
 }
 // Функція повороту МР на заданий кут
 // знаючи час повного оберту (OneTurnTime), знаходимо час повороту на заданий кут
 // потім вмикаємо обертання на вирахуваний проміжок часу
 @Override
 public void rotate(double a) {
 a = Utils.NormalizeAngle(a); // кут на який потрібно зробити оберт
 angle = Utils.NormalizeAngle(angle + a);
 final int time = (int) ((double) params.OneTurnTime *
Math.abs(a) / (2 * Math.PI));
 if (a > 0) {
 gear.right(time); //time - на скільки необхідно вмикати функцію повороту
 } else {
 gear.left(time);
 }
 Tools.delay(time + 100);
 }
 @Override
public void stop() {
 gear.stop();
 }
 //Рух вперед на задану відстань (відстань в юнітах, відповідає координатам екрану)
 @Override
 public void forward(int dist) {
 assert Speed > 0;
 final int time = dist * 1000 / Speed;
 gear.forward(time);
 }
 @Override
 public void forward() {
 gear.forward();
 }
```

Рисунок 3.20 – Приклад коду для виконання повороту МР

```
 // Конструктор. МР створюється і потрапляє в частину екрану, вказану в координатах X та Y
 Robot(int x, int y) {
 RobotContext.setStartPosition(x, y); //Robot.Context – клас емулятору, з функціями для створення навколишнього середовища
 angle = 0;
 RobotContext.setStartDirection(0); //setStartDirect – куди виконувати орієнтування МР
 robot = new LegoRobot();
gear = new Gear();
 robot.addPart(gear); //Функція емулятору. Ініціалізація різних компонентів.
 sensor = new UltrasonicSensor(SensorPort.S1); //s1 - значить, що давач "дивиться" вперед
 sensor.setBeamAreaColor(Color.green);
 sensor.setMeshTriangleColor(Color.blue);
 sensor.setProximityCircleColor(Color.lightGray);
 robot.addPart(sensor);
 System.out.println("Calibrating speed...");
 System.out.println(" - Calibrated, speed = " + Speed + "
units/second");
 }
 }
}
```

Рисунок 3.21 – Приклад коду по створенню МР та розміщення його на екрані

Через специфіку роботи емулятора, а саме функції повороту МР, яка залежить від системного таймера, результати різних запусків можуть сильно відрізнятися. Для кожного типу фізичних перешкод було проведено мінімум 10 запусків алгоритму локального пошуку.

Успішним ми вважали запуск, у результаті якого МР досяг мети руху або коректно визначив, що мета недосяжна. Результати моделювання представлено в табличному вигляді (табл. 3.1) [36].

Таблиця 3.1 - Результати функціонального моделювання в емуляторі типу RobotSim [36]

Тип фізичної перешкоди	Обмеження давача дальності	Кількість вдалих запусків	Середній час успішних запусків	Кількість невдалих запусків	Сумарна кількість запусків
Хрестоподібна	-	10	46 сек.	0	10
Хрестоподібна	100	10	39 сек	0	10
U-подібна	-	10	56 сек	0	10
U-подібна	100	2	1 хв 24 сек	8	10
T-подібна	-	10	30 сек	0	10
T-подібна	100	10	37 сек	0	10
Лабіринт	-	6	1 хв 9 сек	4	10
Лабіринт	100	0	-	10	10
Равлик	-	4	3 хв 10 сек	6	10
Равлик	100	3	3 хв 6 сек	7	10
Недосяжна мета	-	2	2 хв 11 сек	8	10
Недосяжна мета	100	2	2 хв 24 сек	8	10

```
package robo;
import java.awt.Point;
import robo.Interfaces.IAlgorithm;
import robo.Interfaces.IRobot;
import robo.Utils.PolarTurn;
public class TangentBug implements IAlgorithm {
 static private final int NearObstacle = 70; //якщо відстань до перешкоди менше 80, то МР зупиняється
 static private final int NearGoal = 50; //збільшує "вплив мету"
 @Override
 public boolean run(IRobot r, int goal_x, int goal_y) {
 final Point goal = new Point(goal_x, goal_y);
 boolean tryTillGoal = true;
 while (true) {
 if (tryTillGoal && RobotUtils.TillGoal(r, goal, NearObstacle, NearGoal)) {
 // МР досяг мети
 return true;
 }
 tryTillGoal = false;
 final TangentGraph tg = RobotUtils.ScanAround(r, goal_x, goal_y);
 final int goalDist = Utils.GetDist(r.get_x(), r.get_y(),
goal.x, goal.y);
 if (tg.goalIsVisible(goalDist)) {
// мету видно, виконуємо рух до неї до наступної ітерації циклу
 tryTillGoal = true;
 System.out.println("Goal is visible");
 continue;
 }
 System.out.println("Goal is NOT visible");
 final PolarTurn best = tg.GetBestRoute(new Point(r.get_x(),
r.get_y()), goal);
 if (best == null) {
 // немає кращого вузла, який розташовано ближче до мети. Переходимо в режим руху вздовж стіни
  final Utils.FollowWallDirection fwd =
tg.GetFollowWallDirection(
 r.get_x(),
 r.get_y(),
 goal_x,
 goal_y
 );
 final boolean followWallSuccess = FollowWall(r, fwd,
tg.dReach, goal_x, goal_y);
 if (!followWallSuccess) {
 System.out.println("FollowWall FAILURE (loop)");
 return false; // помічене зациклювання
 }
  System.out.println("FollowWall SUCCESS");
 } else {
 // рух до найкращого вузла
 System.out.println("Moving to best node: dist:" +
best.polar.distance + " angle:" + best.polar.angle + " deg:" +
Utils.rad2deg(best.polar.angle));
RobotUtils.MoveTo(r, best);
 System.out.println(" - Moved");
 }
 }
 }
```

Рисунок 3.22 – Приклад коду для виконання руху МР

МР загалом добре проходить окремо розташовані фізичні перешкоди, що розташовуються на, так би мовити, «відкритій місцевості», тобто коли інші фізичні перешкоди розташовуються на віддалі від МР і не чинять істотного

впливу на розрахунок траєкторії шляху.

```
package robo;
import java.awt.Point;
import java.util.ArrayList;
import java.util.Optional;
public class TangentGraph {
 private static int dDist = 20;
 private final ArrayList<Utils.Polar> Nodes = new ArrayList<>();
 private Optional<Boolean> startedInsideSector = Optional.empty();
 private boolean insideSector = false;
 private Utils.Polar goalSample = null; //для запису напрямку та відстані до мети
 private final double goalAngle;
 final int robo_x;
 final int robo_y;
 final int goal_x;
 final int goal_y;
 int dReach = Integer.MAX_VALUE;
 TangentGraph(int robo_x, int robo_y, int goal_x, int goal_y) {
 this.robo_x = robo_x;
 this.robo_y = robo_y;
 this.goal_x = goal_x;
 this.goal_y = goal_y;
 goalAngle = Utils.ComputeAngle(robo_x, robo_y, goal_x, goal_y);
 }
void AddSensorSample(int dist, double absoluteAngle) {
 goalSample = getBestAngleApprox(goalSample, goalAngle, dist,
absoluteAngle);
 if (!startedInsideSector.isPresent()) {
 assert Nodes.isEmpty();
 startedInsideSector = Optional.of(dist >= 0);
 }
 if (!insideSector) {
 assert (Utils.isEven(Nodes.size()));
 }
 if (dist < 0) {
 // кінець поточного сектору, якщо такий є
 if (!Utils.isEven(Nodes.size())) {
 // сектор почався, але не закінчився
 // дублюємо останній елемент
 assert insideSector;
 Nodes.add(Nodes.get(Nodes.size() - 1));
 }
 assert Utils.isEven(Nodes.size());
 insideSector = false;
 return;
 }
 final Utils.Polar polar = new Utils.Polar(dist, absoluteAngle);
 // розрахунок dReach
 dReach = Math.min(
dReach,
 Utils.GetDist(
 goal_x,
 goal_y,
 Utils.DecartFromPoint(robo_x, robo_y, polar)
 )
 );
 if (!insideSector || !Utils.isEven(Nodes.size())) {
 // початок нового сектору або друга точка останнього сектору
 // Просто додати точку
 //assert Utils.isEven(Nodes.size());
 Nodes.add(polar);
 insideSector = true;
 return;
 }
 // третя точка поточного сектору
 assert !Nodes.isEmpty();
 assert Utils.isEven(Nodes.size());
 final Utils.Polar last = Nodes.get(Nodes.size() - 1);
 if (Math.abs(last.distance - polar.distance) < dDist) {
 //замінити останню точку
 Nodes.set(Nodes.size() - 1, polar);
 } else {
 // знайдено "перешкоди, що накладаються"
 // для того щоб потім розпізнати перешкоди, що накладаються в семплі зберігається однакове значення кута.
```

Рисунок 3.23 – Приклад коду для керування рухом МР [36]

```
// Пошук по графу дотичних. Повертає null якщо є локальний мінімум
 Utils.PolarTurn GetBestRoute(Point robo, Point goal) {
 assert !Nodes.isEmpty();
 final int dRoboGoal = Utils.GetDist(robo, goal);
 boolean localMin = true;
 int bestIdx = -1;
 int bestDist = Integer.MAX_VALUE;
 for (int i = 0; i < Nodes.size(); ++i) {
 final Utils.Polar node = getRobotMotion(i).polar;
 final Point delta = Utils.PolarToDecart(node); // зміщення від поточного положення
 final Point obstacleNode = new Point(robo.x + delta.x, robo.y + delta.y);
 final int dObstacleNodeToGoal = Utils.GetDist(obstacleNode, goal);
 if (dObstacleNodeToGoal < dRoboGoal) {
 // існує вузол, кращий за поточне положення
 localMin = false;
 }
 final int dist = Utils.GetDist(robo, obstacleNode) + dObstacleNodeToGoal;
 if (dist < bestDist) {
 bestIdx = i;
 bestDist = dist;
 }
 }
 if (localMin) {
 //знайдено локальний мінімум
 return null;
}
 // Перевірити, чи перетинається краща точка із сусідньою
 final int nearIdx = (Utils.isEven(bestIdx) ? bestIdx - 1 : bestIdx + 1);
 if (nearIdx >= 0 && nearIdx < Nodes.size()) {
 final Utils.Polar nearPolar = Nodes.get(nearIdx);
 final Utils.Polar bestPolar = Nodes.get(bestIdx);
 final boolean overlap = (nearPolar.angle == bestPolar.angle);
 if (overlap && bestPolar.distance > nearPolar.distance) {
  // Якщо кращий вузол перетинається із сусіднім і знаходиться далі ніж сусідній,
// виконуємо рух до ближчого вузла (для ситуації з "накладанням" перешкод
 // так як не зрозуміло, чи там є прохід.
 bestIdx = nearIdx;
 }
 }
 return getRobotMotion(bestIdx);
 }
 private Utils.PolarTurn getRobotMotion(int idx) {
 // Використовується раніше для корекції кута. Див. ToRobotMotion().
 // Парний індекс означає початок сектору, корекція кута повинна бути в -
 // Не парний індекс означає кінець сектору, корекція кута в +
 final int turnSign = Utils.isEven(idx) ? -1 : 1;
 return new Utils.PolarTurn(Nodes.get(idx), turnSign);
 }

Utils.FollowWallDirection GetFollowWallDirection(int robo_x, int robo_y, int goal_x, int goal_y) {
 if (Nodes.isEmpty()) {
 return null;
 }
```

Рисунок 3.24 – Приклад коду для керування рухом МР по графу дотичних [36]

```
 assert Nodes.size() >= 2;
 final Utils.Polar p1 = Nodes.get(0);
 final Utils.Polar p2 = Nodes.get(1);
 final Point p1_dec = Utils.DecartFromPoint(robo_x, robo_y, p1);
 final Point p2_dec = Utils.DecartFromPoint(robo_x, robo_y, p2);
 final int dist_goal_1 = Utils.GetDist(goal_x, goal_y, p1_dec);
 final int dist_goal_2 = Utils.GetDist(goal_x, goal_y, p2_dec);
 final boolean wallOnTheRight = dist_goal_1 > dist_goal_2;
 double sector_angle = Utils.ComputeAngle(p1_dec.x, p1_dec.y, p2_dec.x, p2_dec.y);
 if (!wallOnTheRight) {
 sector_angle += Math.PI;
 }
 return new Utils.FollowWallDirection(wallOnTheRight, sector_angle);
 }
 private static Utils.Polar getBestAngleApprox(Utils.Polar currentBest, double goalAngle, int sampleDist, double sampleAngle) {
 if (currentBest == null || (Math.abs(goalAngle - sampleAngle) < Math.abs(currentBest.angle - goalAngle))) {
 return new Utils.Polar(sampleDist, sampleAngle);
 } else {
 return currentBest;
 }
 }

int GetGoalDist() {
 return goalSample.distance;
 }
 boolean goalIsVisible(int goalDist) {
 assert goalSample != null;
 assert Utils.isEven(Nodes.size());
 for (int i = 0; i < Nodes.size(); i += 2) {
 final Utils.Polar begin = Nodes.get(i);
 final Utils.Polar end = Nodes.get(i + 1);
 assert begin.angle <= end.angle;
 if (begin.angle <= goalSample.angle && goalSample.angle <=end.angle) {
 final boolean visible = (goalSample.distance >=goalDist);
 if (visible) System.out.println("visible: "+begin.angle+" <= "+goalSample.angle+" <= "+end.angle);
 return visible;
 }
 }
 return true;
 }
 private void PrintNodes() {
 System.out.println("Nodes: ");
 for (Utils.Polar p : Nodes) {
 System.out.println("Dist:" + p.distance + " angle:" + p.angle + " deg:" + Utils.rad2deg(p.angle));
 }
 }
```

Рисунок 3.25 – Закінчення коду для керування рухом МР по графу дотичних

У разі потрапляння до локального мінімуму МР коректно будує траєкторію руху у разі, якщо давач дальності вистачає на огляд усієї фізичної перешкоди, і відразу вмикається режим проходження вздовж границі. Якщо ж у точці локального мінімуму давача дальності не вистачає на огляд усієї фізичної перешкоди, МР робить кілька дрібних переміщень, накопичуючи при цьому помилку, що зменшує шанси на успішне проходження фізичної перешкоди.

Для перешкод закритого типу з близько розташованими стінами (типу лабіринт, равлик) похибка у визначенні кутів під час побудови графа дотичних суттєво впливає на шанс проходження перешкод, і загалом такі фізичні перешкоди МР проходить не дуже добре.

Основні причини накопичення помилки в використовуваному емуляторі -

відсутність вбудованої можливості повороту МР та окремо давача дальності на певний кут та використання таймера для виконання поворотів.

Для перенесення розробленої програми на реальну платформу очікуються наступні покращення:

- можливість повороту давача дальності, що стоїть на окремій платформі, без повороту всього МР, що дозволить в свою чергу покращити (і прискорити) режим руху вздовж стіни, оскільки дасть можливість безперервно контролювати відстань до стіни в процесі руху і виконувати більш дрібні та, відповідно, плавні корекції напрямку рух МР;

- можливість більш точного повороту МР на заданий кут;

- можливість виконувати сканування та побудову локального графа дотичних без повороту самого МР, що суттєво зменшить накопичення помилки у визначенні поточної величини кута повороту.

3.3 Висновки до третього розділу

Коректність роботи алгоритму локального пошуку суттєво залежить від точності давача дальності.

Через особливості реалізації розроблюваного модуля, а саме використання емулятора замість реального МР обхід фізичних перешкоди в режимі руху уздовж стіни займає більшу кількість часу і призводить до накопичення помилки, оскільки проводиться поворот МР разом із давачем дальності під час кожного контролю відстані до фізичної перешкоди.

Загалом розроблений алгоритм локального пошуку Tangent Bug показує прийнятні результати і надалі може бути використаний при роботі на реальному МР.

ВИСНОВКИ

Виконано огляд та аналіз існуючих алгоритмів планування руху мобільного робота, встановлено вимоги до модулю керування МР IStrategy, що розробляється. Виконано постановку завдань для розробки модулю керування рухом МР.

Алгоритм локального пошуку Tangent Bug дає змогу МР не тільки обійти ненанесену на створену карту перешкоду, а й дійти до встановленої цілі. Тому, за необхідності, після обходу перешкоди можна не віддавати дію модулю IControl, а продовжувати виконання руху до встановленої мети.

Для реалізації алгоритму локального пошуку буде корисна невелика корецкція – дані із давачів збиратимуться не постійно, а на певних кроках, у режимі руху вздовж стіни не будуватиметься локальний граф дотичних, що дасть змогу суттєво заощадити ресурси.

Коректність роботи алгоритму локального пошуку суттєво залежить від точності давача дальності.

Через особливості реалізації розроблюваного модуля, а саме використання емулятора замість реального МР обхід фізичних перешкоди в режимі руху уздовж стіни займає більшу кількість часу і призводить до накопичення помилки, оскільки проводиться поворот МР разом із давачем дальності під час кожного контролю відстані до фізичної перешкоди.

Загалом розроблений алгоритм локального пошуку Tangent Bug показує прийнятні результати і надалі може бути використаний при роботі на реальному МР.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Multi-agent robotics systems and artificial intelligence. [Електронний ресурс] – Режим доступу:

https://smprobotics.com/technology_autonomous_mobile_robot/multi-agent-robotics-systems/

2. Multi-Agent Systems in Robotics: Coordination and Communication using Machine Learning / Dr. Coenrad Adolph Groenewald, Gonesh Chandra Saha, Garima Mann, Bharat Bhushan, Eric Howard, Elma Sibonghanoy Groenewald - NATURALISTA CAMPANO ISSN: 1827-7160 Volume 28 Issue 1, 2024 – 882-897 p.

3. Kumar, V., & Kumar, V. (2010). Multi-robot systems: A classification focused review. Swarm and Evolutionary Computation, 1(2), 85-116 p.

4. Робототехнічні системи та комплекси: мобільні роботи довільної орієнтації: підруч. для студ. спец. «Інформаційні системи та технології» / М. М. Поліщук, М. М. Ткач; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 14,7 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2023. – 301 с. ISBN 978-966-990-076-0

5. Tin Lun Lam, Yangsheng Xu. Tree Climbing Robot: Design, Kinematics and Motion Planning / Tin Lun Lam, Yangsheng Xu. – Springer Heidelberg, New York, 2012. – 178 p.

6. Polishchuk M. M. Optimization of mobile robot parameters for surfaces of arbitrary y orientation / M. M. Polishchuk // Вчені записки Таврійського нац. ун-ту ім. В. І. Вернадського. Технічні науки. – 2020. – Т. 31 (70). – № 1. – С. 1 – 5.

7. Hart, P.E. A Fromal Basis for the Heuristic Determination of Minimum Cost Paths [Текст]/ P.E. Hart, N.J. Nilsson, B.A. Raphael// System Science and Cybernetics: IEEE Transactions, 1968. - Т. 4. - С. 100-107.

8. Dijkstra, E. W. A note on two problems in connection with graphs [Текст]/ E. W. Dijkstra// Numerische Mathematik. - 1959. - 1. - С. 269-271.

9. Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots [Текст]/O. Khatib// International Journal of Robotics Research. - 1986. - Т. 5. - С. 90-98.

10. Kavraki, L.E. Probabilistic roadmaps for path planning in high-dimensional configuration space [Текст]/ L.E. Kavraki, P. Svestka, J.C. Latombe и др.// IEEE Transactions on Robotics and Automation - 1996. - Т. 12. - С. 566-568.

11. Geraets, R. A Comparative Study of Probabilistic Roadmap Planners [Текст]/ R. Geraets, M.H. Overmars// Workshop on the Algorithmic Foundations of Robotics. - 2002. - С. 43-57.

12. Latombe, J. C Monte-Carlo algorithm for path planning with many degrees of freedom [Текст]/ J.C. Latombe, J. Barraquand// IEEE International Conference on Robotics and Automation. - 1990. - С. 1712–1717.

13. LaValle, S. M. Rapidly-exploring random trees: A new tool for path planning [Текст]/ S.M. LaValle.-Department of Computer Science., Iowa State University. - Iowa, 1998.

14. LaValle, S.M. Rapidly-exploring random trees: Progress and Prospects [Текст]/ S.M. LaValle, J. J. Kuffner// In Proceedings Workshop on the Algorithmic Foundations of Robotics. - 2000. - С. 293-380.

15. Svestka, P. A probabilistic approach to motion planning for car-like robots [Текст]/ P. Svetska, M. H. Overmars// The International Journal of Robotics Research. - 1997. - vol 16(2).- С. 119-143.

16. Wilmarth, S. A. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space [Текст]/ S. A. Wilmarth, N. M. Amato, P.F. Stiller// IEEE International Conference on Robotics and Automation. - 1999. - С. 1024-1031.

17. Lingelbach, L. Path planning using probabilistic cell decomposition [Текст]/L. Lingelblach. - Stockholm, Sweden: Royal Institute of Technology. - 2005. - С. 73.

18. Yufka A., Performance Comparison of Bug Algorithms for Mobile Robots

[Текст]/ A. Yufka, O. Parlaktuna// 5th International Advanced Technologies Symposium. - Karabuk, Turkey: - 2009. - C.416-421.

19. Lumelsky, V. Effect of Uncertainty on Continuous Path Planning for an Autonomous Vehicle [Текст]/ V. Lumelsky, P. Stepanov// Proceedings of the 23rd Conference on Decision and Control. - Las Vegas, Nevada : 1984. - C. 1616-1621.

20. Stepanov, A. Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape [Текст]/ A. Stepanov, E. Lumelsky.// Algorithmica. - New York : Springer-Verlag New York Inc., 1987. - T. 2. - C. 403-430.

21. Kamon, I. Sensory-Based Motion Planning with Global Proofs [Текст]/ I. Rivlin, E. Kamon// IEEE Transaction on Robotic and Automation - 1997. - T. 13. - C. 814- 822.

22. Kamon, I. A new Range-Sensor Based Globally Convergent Navigation Algorithm for Mobile Robots [Текст]/ I. Kamon, E. Rivlin, E. Rimon// Minneapolis: - Robotics and Automation. Proceedings on IEEE International Conference - 1995.- T1.- C. 429-435.

23. Ng James An Analysis of Mobile Robot Navigation Algorithms in Unknown Environments [Текст]/ James Ng.: Ph.D. thesis, School of Electrical, Electronic and Computer Engineering, Newcastle University, 2010. - P.203.

24. Sezer, V. A Novel Obstacle Avoidance Algorithm: Follow the Gap Method [Текст]/V. Sezer, M. Goksan // Robotics and Autonomous Systems. - 2012 . - vol. 60(9). - C. 1123-1134.

25. Borenstein, J. The Vector Field Histogram fast obstacle avoidance for mobile robots[Текст]/ J. Borenstein, Y. Koren// IEEE Transactions on Robotics and Automation . - 1991. - №3. - C. 278-288.

26. Koren, Y. High Speed Obstacle Avoidance for Mobile Robotics [Текст]/ Y. Koren, J. Borenstein// in Proceedings of the IEEE Symposium on Intelligent Control. - Arlington, VA : 1988. - C. 382-384.

27. Siegwart, R. Introduction to Autonomous Mobile Robots [Текст]/R.

Siegward, I.R. Nourbakhsh, D. Scaramuzza. - Massachusets: The MIT Press, 2004.

28. Fox, D. The Dynamic Window Approach to Collision Avoidance [Текст]/ D. Fox, W. Burgard, S.Thrun // IEEE Robotics and Automation Magazine. - 1997. - С. 23– 33.

29. Fox, D. Markov Localization for Reliable Robot Navigation and People Detection [Текст]/ D. Fox , W. Burgard, S. Thrun// In Selected Papers from the International Workshop on Sensor Based Intelligent Robots. - 1999. - С. 1-20.

30. Brock, O., High-Speed Navigation Using the Global Dynamic Window Approach [Текст]/ O. Brock, O. Khatib// in Proceeding of the IEEE International Conference on Robotics and Automation. - Detroit : 1999. - Т. 1. - С. 341–346.

31. Seder, M. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles[Текст]/M. Seder, I Petrovic// IEEE International Conference on Robotics and Automation. - 2007. - С. 1986-1991.

32. Zhuang Y. et al. A novel outdoor scene-understanding framework for unmanned ground vehicles with 3D laser scanners. Transactions of the Institute of Measurement and Control. 2015. Vol. 37, no. 4, pp. 435-445.

33. Kamon, I. A new Range-Sensor Based Globally Convergent Navigation Algorithm for Mobile Robots [Текст]/ I. Kamon, E. Rivlin, E. Rimon// Minneapolis: - Robotics and Automation. Proceedings on IEEE International Conference - 1995.- Т1.- С. 429-435.

34. Алгоритми та методи обчислень [Електронний ресурс] : навч. посіб. для студ. спеціальностей 121 «Інженерія програмного забезпечення», спеціалізації «Програмне забезпечення високопродуктивних комп'ютерних систем та мереж» та 123 «Комп'ютерна інженерія», спеціалізації «Комп'ютерні системи та мережі» / М. А. Новотарський; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 4648 Кбайт). – Київ : КПІ ім. Ігоря Сікорського, 2019. – 407 с.

35. Java. Теорія і практика: навчальний посібник для студентів природничих спеціальностей університетів / Кадомський К.К., Ніколюк П.К. –

Вінниця: Донну, 2019. – 197 с.

36. RobotSim. A Simulation Package for the Lego EV3/NXT Robot [Електронний ресурс] – Режим доступу:

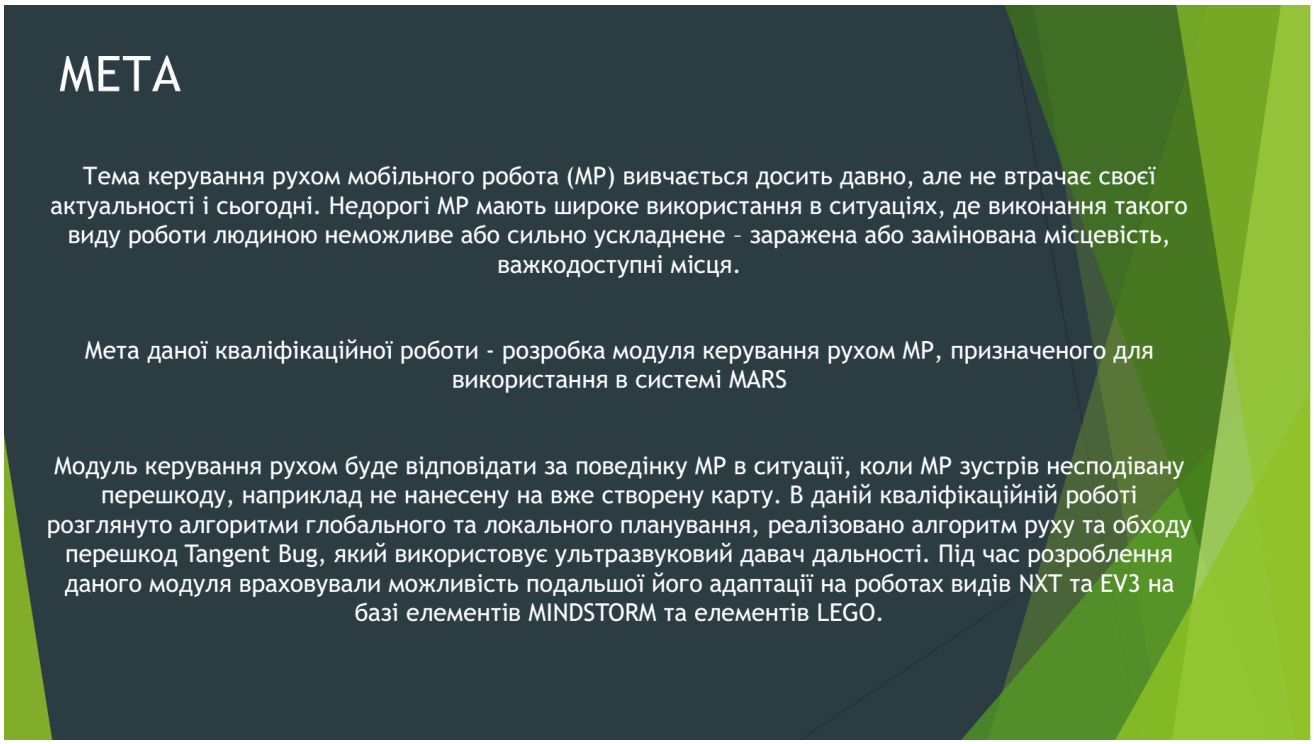
http://www.aplu.ch/home/apluhomex.jsp?site=75

37. Кваліфікаційна робота : методичні вказівки щодо її виконання для студентів спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» / уклад.: Ю.В. Форкун, Г.І. Радельчук, І.В. Форкун, А.С. Каштальян, В.В. Мартинюк. Хмельницький : ХНУ, 2020. – 50 с.

ДОДАТКИ

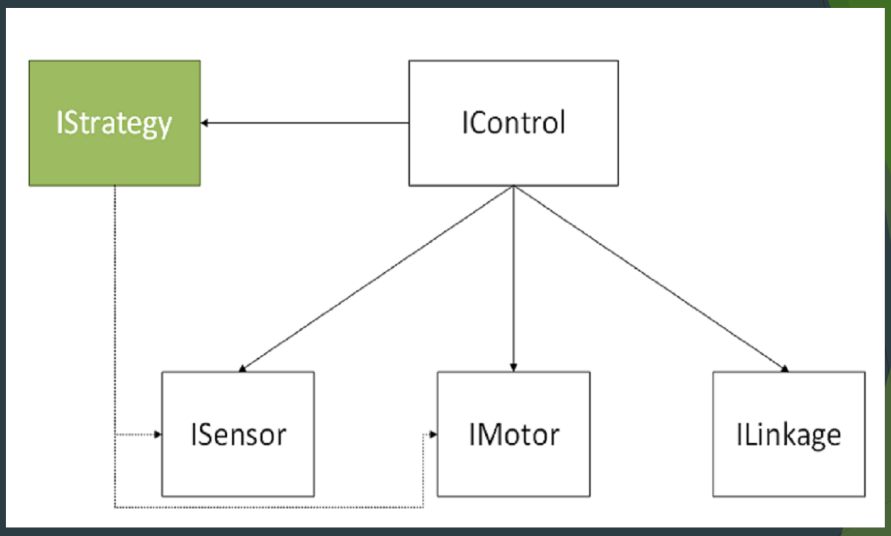
Додаток А

Презентаційні матеріали

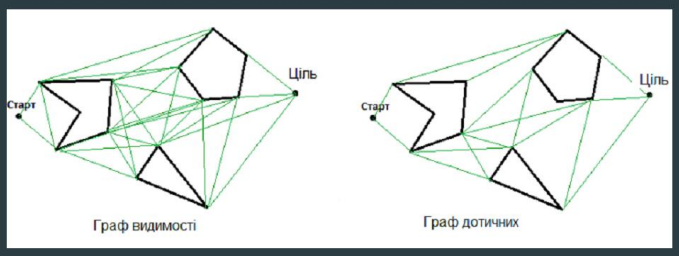


Для досягнення поставленої мети кваліфікаційної роботи бакалавра необхідно вирішити наступні завдання:

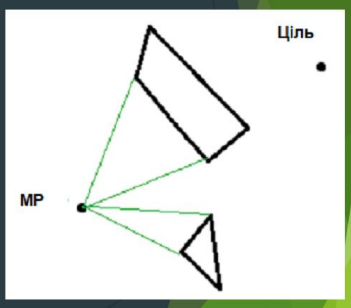
- - вивчення структури та особливостей платформи MARS;
- - вивчення наявних алгоритмів обходу перешкод МР в невизначених умовах;
- - розробка алгоритму руху МР для обходу існуючих нерухомих перешкод;
- - проєктування модуля керування рухом МР;
- - реалізація модуля керування рухом МР на платформі JAVA із урахуванням існуючих особливостей і обмежень платформи MARS;
- - перевірка працездатності модуля керування рухом.



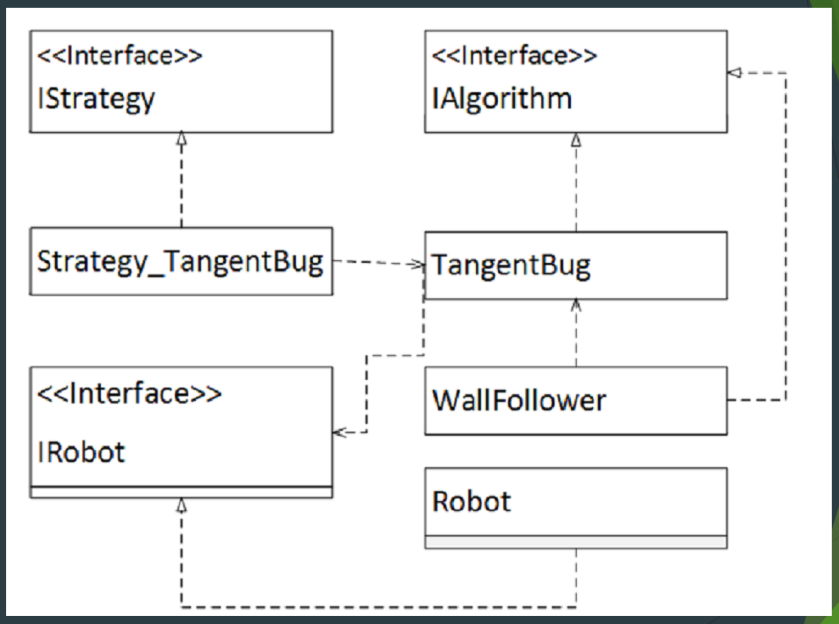
- Спрощена схема програмного модулю вузла платформи MARS



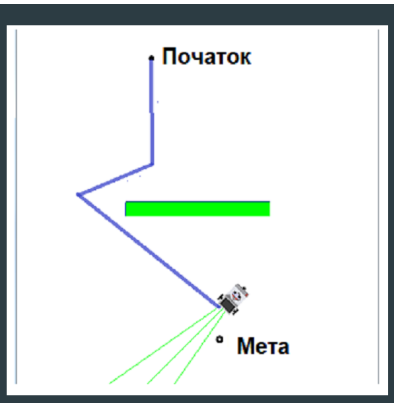
- Загальний вигляд графів дотичних та видимості



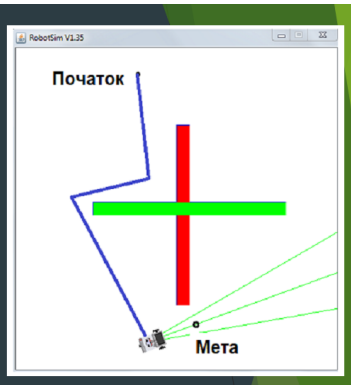
- Загальний вигляд локального графу дотичних



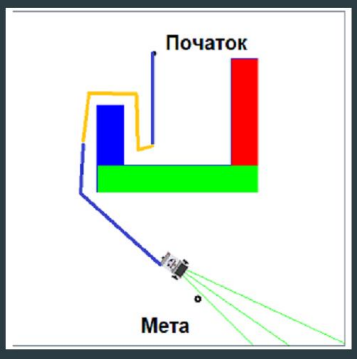
- Скорочена діаграма класів модуля IStrategy



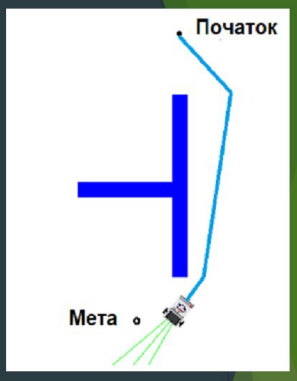
- Моделювання фізичної горизонтальної перешкоди



- Моделювання фізичної хрестоподібної перешкоди

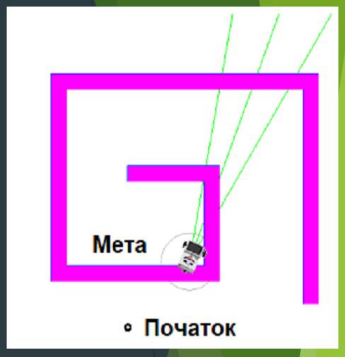


- Моделювання фізичної U-подібної перешкоди

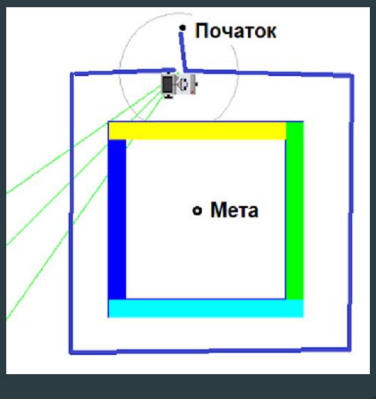


- Моделювання фізичної T-подібної перешкоди

- Моделювання фізичної перешкоди типу лабіринт



- Моделювання фізичної перешкоди типу равлик



- Моделювання ситуації недосяжності мети

Тип фізичної перешкоди	Обмеження давача дальності	Кількість вдалих запусків	Середній час успішних запусків	Кількість невдалих запусків	Сумарна кількість запусків
Хрестоподібна	-	10	46 сек.	0	10
Хрестоподібна	100	10	39 сек	0	10
U-подібна	-	10	56 сек	0	10
U-подібна	100	2	1 хв 24 сек	8	10
T-подібна	-	10	30 сек	0	10
T-подібна	100	10	37 сек	0	10
Лабіринт	-	6	1 хв 9 сек	4	10
Лабіринт	100	0	-	10	10
Равлик	-	4	3 хв 10 сек	6	10
Равлик	100	3	3 хв 6 сек	7	10
Недосяжна мета	-	2	2 хв 11 сек	8	10
Недосяжна мета	100	2	2 хв 24 сек	8	10

- Результати функціонального моделювання в емуляторі типу RobotSim

Додаток Б

Приклад коду програми для виконання симуляції

```
// для тесту
 ArrayList<Utils.Polar> GetNodes() {
 /*PrintNodes();*/
 return Nodes;
 }
}

package robo;
import java.util.ArrayList;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import robo.Utils.Polar;
public class TangentGraphTest {
 public TangentGraphTest() {
 }
 @BeforeClass
 public static void setUpClass() {
 }
 @AfterClass
 public static void tearDownClass() {
 }
 @Before
 public void setUp() {
 }
 @After
public void tearDown() {
 }
 static private TangentGraph CreateTangentGraph()
 {
 return new TangentGraph(0, 0, 1, 1);
 }
 static private TangentGraph CreateTangentGraph(double goal_angle)
 {
 final int goal_x = 1000000;
 final int goal_y = (int) ((double) goal_x *
Math.tan(goal_angle));
 return new TangentGraph(0, 0, goal_x, goal_y);
 }
 @Test
 public void testAddSensorSample1() {
 TangentGraph tg = CreateTangentGraph();
 tg.AddSensorSample(1, 0.1);
 tg.AddSensorSample(2, 0.2);
 tg.AddSensorSample(3, 0.3);
 tg.AddSensorSample(-1, 0.4);
 tg.AddSensorSample(4, 0.5);
 tg.Finish();
 ArrayList<Polar> nodes = tg.GetNodes();
 assertTrue(nodes.size()==2);
 assertTrue(nodes.get(0).distance==4);
 assertTrue(nodes.get(1).distance==3);
```

```
 }
@Test
 public void testAddSensorSample2() {
 TangentGraph tg = CreateTangentGraph();
 tg.AddSensorSample(1, 0.1);
 tg.AddSensorSample(2, 0.2);
 tg.AddSensorSample(3, 0.3);
 tg.AddSensorSample(-1, 0.4);
 tg.AddSensorSample(4, 0.6);
 tg.AddSensorSample(5, 0.7);
 tg.AddSensorSample(6, 0.8);
 tg.AddSensorSample(-1, 0.9);
 tg.Finish();
 ArrayList<Polar> nodes = tg.GetNodes();
 assertTrue(nodes.size()==4);
 assertTrue(nodes.get(0). distance==1);
 assertTrue(nodes.get(1).distance==3);
 assertTrue(nodes.get(2).distance==4);
 assertTrue(nodes.get(3).distance==6);
 }

@Test
 public void testAddSensorSample3() {
 TangentGraph tg = CreateTangentGraph();
 tg.AddSensorSample(1, 0.1); // end of sector
 tg.AddSensorSample(-1, 0.2);
 tg.AddSensorSample(-1, 0.3);
 tg.AddSensorSample(10, 0.4); // begin of sector
 tg.AddSensorSample(11, 0.5);
 tg.Finish();
ArrayList<Polar> nodes = tg.GetNodes();
 assertTrue(nodes.size()==2);
 assertTrue(nodes.get(0).distance==10);
 assertTrue(nodes.get(1).distance==1);
 }
 @Test
 public void testAddSensorSample4() {
 TangentGraph tg = CreateTangentGraph();
 tg.AddSensorSample(1, 0.1);
 tg.AddSensorSample(2, 0.15); // end of sector
 tg.AddSensorSample(-1, 0.2);
 tg.AddSensorSample(-1, 0.3);
 tg.AddSensorSample(10, 0.4); // begin of sector
 tg.AddSensorSample(11, 0.5);
 tg.Finish();
 ArrayList<Polar> nodes = tg.GetNodes();
 assertTrue(nodes.size()==2);
 assertTrue(nodes.get(0).distance==10);
 assertTrue(nodes.get(1).distance==2);
 }
 @Test
 public void testAddSensorSample5() {
 TangentGraph tg = CreateTangentGraph();
 tg.AddSensorSample(-1, 0.05);
 tg.AddSensorSample(1, 0.1);
 tg.AddSensorSample(2, 0.15);
 tg.AddSensorSample(-1, 0.2);
 tg.AddSensorSample(-1, 0.3);
tg.AddSensorSample(10, 0.4);
 tg.AddSensorSample(11, 0.5);
 tg.Finish();
```

```
ArrayList<Polar> nodes = tg.GetNodes();
assertTrue(nodes.size()==4);
assertTrue(nodes.get(0).distance==1);
assertTrue(nodes.get(1).distance==2);
assertTrue(nodes.get(2).distance==10);
assertTrue(nodes.get(3).distance==11);
}
@Test
public void testAddSensorSample_goalVisible1() {
TangentGraph tg = CreateTangentGraph(0.11);
tg.AddSensorSample(10, 0.1);
tg.AddSensorSample(20, 0.2);
tg.AddSensorSample(-1, 0.25);
tg.AddSensorSample(30, 0.3);
tg.AddSensorSample(40, 0.4);
tg.AddSensorSample(-1, 0.5);
tg.Finish();
ArrayList<Polar> nodes = tg.GetNodes();
assertTrue(nodes.size()==4);
assertTrue(nodes.get(0).distance==10);
assertTrue(nodes.get(1).distance==20);
assertTrue(nodes.get(2).distance==30);
assertTrue(nodes.get(3).distance==40);
assertTrue(tg.goalIsVisible(9));
assertFalse(tg.goalIsVisible(11));
}
@Test
public void testAddSensorSample_goalVisible2() {
TangentGraph tg = CreateTangentGraph(0.26);
tg.AddSensorSample(10, 0.1);
tg.AddSensorSample(20, 0.2);
tg.AddSensorSample(-1, 0.25);
tg.AddSensorSample(30, 0.3);
tg.AddSensorSample(40, 0.4);
tg.AddSensorSample(-1, 0.5);
tg.Finish();
ArrayList<Polar> nodes = tg.GetNodes();
assertTrue(nodes.size()==4);
assertTrue(nodes.get(0).distance==10);
assertTrue(nodes.get(1).distance==20);
assertTrue(nodes.get(2).distance==30);
assertTrue(nodes.get(3).distance==40);
assertTrue(tg.goalIsVisible(1));
assertTrue(tg.goalIsVisible(100));
}
// накладання перешкод
@Test
public void testAddSensorSample6() {
TangentGraph tg = CreateTangentGraph();
tg.AddSensorSample(1, 0.05);
tg.AddSensorSample(2, 0.1);
// накладання перешкод: ні від’ємного семпла, велика різниця дальності
tg.AddSensorSample(101, 0.15);
tg.AddSensorSample(102, 0.2);
tg.AddSensorSample(-1, 0.9);
tg.Finish();
ArrayList<Polar> nodes = tg.GetNodes();
assertTrue(nodes.size()==4);
assertTrue(nodes.get(0).distance==1);
assertTrue(nodes.get(1).distance==2);
assertTrue(nodes.get(1).angle==nodes.get(2).angle);
```

```
 assertTrue(nodes.get(2).distance==101);
 assertTrue(nodes.get(3).distance==102);
 }
}

package robo;
import java.awt.Point;
public class Utils {
 static class Polar {
 int distance;
 double angle;
 Polar(int d, double a) { distance=d; angle=a; }
 }
 //Полярні координати. З врахуванням того, де знаходиться перешкода справа чи зліва.
 static class PolarTurn {
 Polar polar;
 int turnSign; // { -1,+1}
 PolarTurn(Polar p, int sign) { polar=p; turnSign=sign; }
 }
 static class FollowWallDirection {
 boolean WallOnTheRight;
 double angle;
 FollowWallDirection(boolean wr, double a) {
 WallOnTheRight = wr;
 angle = a;
 }
 }
// отримані математичні обрахунки наблизити до реальних
 // - виконати коректування кута, щоб MP наблизився до точки не задіваючи її.
 // - трохи збільшити пройдений шлях, щоб покращити кут обзору для збирання наступних даних
 static Polar ToRobotMotion(PolarTurn p, int roboSize)
 {
 double angleCorrection =
Math.asin((double)(roboSize*2)/(double)p.polar.distance);
 if (p.turnSign < 0)
 angleCorrection = -angleCorrection;
 final int distCorrection = 0;//-2*roboSize;
 return new Polar(p.polar.distance + distCorrection, p.polar.angle
+ angleCorrection);
 }
 // Конвертувати кут від [ -pi, pi]. кути в радіанах
 // вліво більше як на 180 та вправо більше як на 180
 static double NormalizeAngle(double a) {
 while (a > Math.PI) {
 a -= 2 * Math.PI;
 }
 while (a < -Math.PI) {
 a += 2 * Math.PI;
 }
 return a;
 }
 static Point PolarToDecart(Polar p) {
 double dd = (double) p.distance;
 return new Point((int) (dd * Math.cos(p.angle)), (int) (dd *
Math.sin(p.angle)));
}
 static int GetDist(int x1, int y1, int x2, int y2) {
 return (int) Math.sqrt(Math.pow(x1 - x2, 2) + Math.pow(y1 - y2,
2));
 }
 static int GetDist(int x1, int y1, Point p2) {
```

```
 return GetDist(x1, y1, p2.x, p2.y);
 }
 static int GetDist(Point p1, Point p2) {
 return GetDist(p1.x, p1.y, p2.x, p2.y);
 }
 static Point DecartFromPoint(int x, int y, Polar pp) {
 final Point pp_dec = PolarToDecart(pp);
 return new Point(x + pp_dec.x, y + pp_dec.y);
 }
 static boolean isEven(int i) {
 return (i & 1) == 0;
 }
 // знайти кут напрямку
 static double ComputeAngle(int x1, int y1, int x2, int y2) {
 return Math.atan2((double)(y2 - y1), (double)(x2 - x1));
 }
 static double deg2rad(int deg)
{
 return Math.PI / 180 * deg;
 }
 static int rad2deg(double rad)
 {
 return (int) (rad / Math.PI * 180);
 }
}

package robo;
import java.awt.Point;
import robo.Interfaces.IAlgorithm;
import robo.Interfaces.IRobot;
/*
Алгоритм руху Алгоритм руху вздовж стіни:
Встаємо паралельно стіні в напрямку обходу.
1. Робимо поворот на 45 градусів у бік стіни, визначаємо відстань до неї.
2. Обчислюємо різницю між ідеальною відстанню і виміряною,
за цією різницею визначаємо величину корекції кута (пошук у масиві Corrections).
3. Повертаємося назад на 45 градусів ± корекція.
4. Робимо крок уперед.
5. goto 1.
*/
public class WallFollower implements IAlgorithm {
 private final Utils.FollowWallDirection Fwd;
 private int dFollowed = Integer.MAX_VALUE;
 private final int dReach;
 private Point Start = null; // точка, з якої починали рух вздовж стіни
 private final int start_diff = 20; // погрішність визначення координат при зациклюванні
 private final int loop_step = 5; // після скількох переміщень визначати зациклювання
 WallFollower(Utils.FollowWallDirection fwd, int dReach) {
 Fwd = fwd;
this.dReach = dReach;
 System.out.println("FollowWall:
wallOnTheRight?"+Fwd.WallOnTheRight+" angle:"+Utils.rad2deg(Fwd.angle)+"
dReach:"+this.dReach);
 }
 // кут повороту до стіни для визначення відстані до неї
 private static final double TurnAngle = Utils.deg2rad(45);
 // ідеальна відстань до стіни, підтримуємо за допомогою коректування
 private static final int WallDist = 70;
 // один запис коректування
 private static class Correction {
 int dist_delta; // різниця між ідеальною відстанню та фактичною
```

```
 double angle_correction; // кут на який необхідно виконати коректування
 int next_step; // довжина наступного кроку
 Correction(int dd, double a, int s) { dist_delta=dd;
angle_correction=a; next_step=s; }
 };
 private static final Correction[] Corrections = {
 new Correction(5, Utils.deg2rad(0), 30),
 new Correction(10, Utils.deg2rad(5), 20),
 new Correction(20, Utils.deg2rad(20), 20),
 new Correction(30, Utils.deg2rad(30), 10),
 new Correction(50, Utils.deg2rad(50), 15),
 new Correction(Integer.MAX_VALUE, Utils.deg2rad(70), 10),
 };
private Correction FindCorrection(int dist_delta) {
 for (final Correction c : Corrections) {
 if (c.dist_delta >= dist_delta) {
 return c;
 }
 }
 return null;
 }
 @Override
 public boolean run(IRobot r, int goal_x, int goal_y) {
 Start = new Point(r.get_x(), r.get_y());
 boolean first = true;
 int step_number = 0;
 while (true) {
 final int next_step_size = check_wall_distance_and_correct(r,
first, goal_x, goal_y);
 ++step_number;
 System.out.println("dFollowed:"+dFollowed+" dReach:"+dReach);
 if (dFollowed < dReach) {
 // успех
 return true;
 }
 if (step_number > loop_step && RobotUtils.PointReached(r,
Start, start_diff)) {
 // зацикливание
 return false;
 }
 r.forward(next_step_size);
 first = false;
 }
}
 // поворот до стіни, визначення відстані до неї
 // обрахунок корекції кута, поворот назад з врахуванням корекції
 private int check_wall_distance_and_correct(IRobot r, boolean first,
int goal_x, int goal_y) {
 System.out.println("StepAndCorrect");
 // повертаємось на місці до стіни, для визначення відстані до неї
 final double turn_angle = Fwd.WallOnTheRight ? -TurnAngle :
TurnAngle;
 // на першій ітерації ми стоїмо не паралельно стіні,
 // додаємо кут між напрямком MP і напрямком стіни
 final double rot_angle = first ? turn_angle + Fwd.angle :
turn_angle;
 r.rotate(rot_angle);
 // визначаємо відстань до стіни
 int current_wall_dist = r.get_distance();
 if (current_wall_dist < 0) {
 current_wall_dist = Integer.MAX_VALUE; // для виконання пошуку по масиву Corrections
```

```
 } else {
 final Point wall = Utils.DecartFromPoint(
 r.get_x(),
 r.get_y(),
 new Utils.Polar(current_wall_dist, r.get_angle())
 );
 final int wall_to_goal = Utils.GetDist(goal_x, goal_y, wall);
 if (wall_to_goal < dFollowed) {
 dFollowed = wall_to_goal;
}
 }
 // різниця між ідеальною відстанню та виміряною
 final int delta = Math.abs(WallDist - current_wall_dist);
 System.out.println(" - CurrentDist: " + current_wall_dist + "/" +
WallDist + " delta: " + delta);
 // шукаємо поправки
 final Correction correction = FindCorrection(delta);
 assert correction != null;
 assert correction.dist_delta >= delta;
 // якщо стіна справа і відстань до неї менша за ідеальну то ми повинні
 // відвернути від стіни, тобто повернути наліво
 // і т.п.
 final boolean correct_to_left = (Fwd.WallOnTheRight && current_wall_dist < WallDist)
 || (!Fwd.WallOnTheRight && current_wall_dist > WallDist);
 System.out.println(" - DeltaAngle:"+Utils.rad2deg(correction.angle_correction));
 final double correction_angle = correct_to_left
 ? correction.angle_correction
 : -correction.angle_correction;
 // кут повороту з врахуванням повороту назад паралельно стіні та корекції
 final double rotation_angle = correction_angle - turn_angle;
 System.out.println(" - correction_angle:
"+Utils.rad2deg(correction_angle));
// повертаємось
 r.rotate(rotation_angle);
 // повертаємо величину наступного кроку
 return correction.next_step;
 }
}
```

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Літвінов Владислав Олександрович

Тема: Система керування рухом мобільного робота

Спеціальність: 151 «Автоматизація та комп'ютерно-інтегровані технології»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 10 Кількість сторінок записки 55

1. Короткий зміст роботи та прийнятих рішень: створено модуль керування рухом мобільного робота, призначеного для використання в системі Multi Agent Robotic System

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі виконано огляд та аналіз існуючих алгоритмів планування руху мобільного робота, встановлено вимоги до модулю керування мобільним роботом IStrategy, що розробляється. Виконано постановку завдань для розробки модулю керування рухом мобільного роботу. У другому розділі обґрунтовано вибір алгоритму локального пошуку Tangent Bug, що дає змогу мобільному роботу не тільки обійти ненанесену на створену карту перешкоду, а й дійти до встановленої цілі. Після обходу перешкоди можна не віддавати дію модулю IControl, а продовжувати виконання руху до встановленої мети. Для реалізації алгоритму локального пошуку внесена корекція – дані із давачів збиратимуться не постійно, а на певних кроках, у режимі руху вздовж стіни не будуватиметься локальний граф дотичних, що дасть змогу суттєво заощадити ресурси.
У третьому розділі розроблено модуль, але через використання емулятора замість реального мобільного робота обхід фізичних перешкоди в режимі руху уздовж стіни займає більшу кількість часу і призводить до накопичення помилки, оскільки проводиться поворот мобільного робота разом із давачем дальності під час кожного контролю відстані до фізичної перешкоди. Загалом розроблений алгоритм локального пошуку Tangent Bug показує прийнятні результати і надалі може бути використаний при роботі на реальному мобільному роботі.

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: доцільно було б виконати експериментальні дослідження створеного модулю керування рухом мобільного робота, призначеного для використання в системі Multi Agent Robotic System

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. *Інші* зауваження: відсутні

9. Оцінка дипломної роботи: добре (B/4,25)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи)
Майдан Павло Сергійович, доцент каф. МАЕЕС ХНУ

"17" 06 2024 р. (підпис)

Завідувачу кафедри АКІТтаР
д-ру техн.наук, проф. Мартинюку В.В.

Літвінова В.О.
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи АКІТс-21-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність плагіату ознайомлений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

01.06.2024
дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ АВТОМАТИЗАЦІЇ, КОМП'ЮТЕРНО-ІНТЕГРОВАНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Система керування рухом мобільного робота

Автор: Владислав ЛІТВІНОВ

Спеціальність: 151 Автоматизація та комп'ютерно-інтегрованих технологій

Освітня програма: Освітньо-професійна програма «Автоматизація та комп'ютерно-інтегровані технології»

Науковий керівник: к.т.н., доц. Денис МАКАРИШКІН

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій у переліку джерел посилання;

2) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

3) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 2,87% і адресується до 84 джерел, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Завідувач кафедри — Валерій МАРТИНЮК

Гарант освітньої програми — Юрій ФОРКУН

Керівник кваліфікаційної роботи — Денис МАКАРИШКІН

Thu Jun 13 16:53:05 EEST 2024, Федула Микола Васильович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 7%**

ID: 130300 Назва: БКР Система керування рухом мобільного робота Додано в БД: 2024-06-13 Автора: Владислав ЛІТВІНОВ Керівники: Денис МАКАРІШКІН Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	50011	764	1125 (2%)	16 (2%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK
by Turnitin

User name:
Кафедра АКІТіТК

Check ID:
1016357837

Check date:
13.06.2024 21:40:40 EEST

Check type:
Doc vs Internet + Library

Report date:
13.06.2024 21:51:39 EEST

User ID:
100005862

File name: **Літвінов_антиплаг**

Page count: **56** Word count: **9134** Character count: **66260** File size: **1.28 MB** File ID: **1016162147**

1329 words are marked as "ignored" and excluded from word count

Text modifications detected (similarity score might be affected)

2.87% Matches

Highest match: **1.55%** with Library source (File ID: **1016162148**)

1.22% Internet sources	84	Page 58
1.77% Library sources	41	Page 58

0% Quotes

No quotes found

No references found

0.01% Exclusions

Some exclusions were automatic (exclusion filters: matched word count less than **8 words** and **0%**)

0.01% Internet exclusions	13	Page 59
0% Library exclusions	1	Page 59

Modifind

Text modifications detected. Find more details in the online report.

Replaced characters	10
Suspicious formatting	11 Pages