НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ

«На правах рукопису»
УДК _____

«До захисту допущено»
В.о. завідувача кафедрою
_____ М.М.Савчук
(підпис) (ініціали, прізвище)
“___” _____ 2019р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності: 113 Прикладна криптологія
(код і назва)

на тему: Модифікація та дослідження алгоритму пошуку шляху в динамічному середовищі на основі випадкових дерев

Виконав: студент 6 курсу, групи ФІ-83мп
(шифр групи)
Смірнов Антон Олександрович
(прізвище, ім’я, по батькові) (підпис)

Керівник: доцент, доктор фіз. мат. наук Олійник А.С.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант:
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент:
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»
Фізико-технічний інститут
Кафедра математичних методів захисту інформації

Рівень вищої освіти: другий (магістерський) за освітньо–професійною програмою

Спеціальність: 113 «Прикладна математика»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедрою
_____ М.М.Савчук
(підпис) (ініціали, прізвище)
«___»_____201_ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Смірнову Антону Олександровичу

(прізвище, ім'я, по батькові)

1.Тема дисертації

Модифікація та дослідження алгоритму пошуку шляху в динамічному середовищі на основі випадкових дерев

науковий керівник дисертації доцент, доктор фіз. мат. наук Олійник А.С.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від _____р. № _____

2. Термін подання студентом дисертації _____

3.Об'єкт дослідження процесс планування шляху в динамічних середовищах.

4. Предмет дослідження алгоритми планування шляху в динамічних середовищах на основі випадкових дерев.

5. Перелік завдань, які потрібно розробити

1. Вибір теми дослідження.
2. Ознайомлення з літературою за темою.
3. Ознайомлення з обраним алгоритмом.
4. Аналіз обраного алгоритму.
5. Модифікація та аналіз модифікованого алгоритму.
6. Практична реалізація обох алгоритмів.

6. Орієнтовний перелік ілюстративного матеріалу

52 сторінки, 10 рисунків , 2 таблиці, 14 джерел

7. Орієнтовний перелік публікацій

8. Консультанти розділів дисертації[1]*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Вибір теми магістерської дисертації	вересень - грудень 2018	
2.	Ознайомлення з літературою за темою	січень - квітень 2019	
3.	Ознайомлення з обраним алгоритмом	квітень - травень 2019	
4.	Аналіз обраного алгоритму	травень - червень 2019	
5.	Модифікація та аналіз алгоритму	червень - серпень 2019	
6.	Практична реалізація обох алгоритмів	серпень - жовтень 2019	
7.	Написання диплому	жовтень - грудень 2019	

Студент _____ Смірнов А.О.
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Олійник А.С.
(підпис) (ініціали, прізвище)

[1]* Консультантом не може бути зазначено наукового керівника магістерської дисертації.

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота містить 52 сторінки, 10 рисунків, 2 таблиці, 14 джерел.

Метою роботи був аналіз та модифікація алгоритму планування шляху на основі випадкових дерев. Об'єктом дослідження був процесс планування шляху в динамічних середовищах. Предметом дослідження були алгоритми планування шляху в динамічних середовищах на основі випадкових дерев.

В результаті виконання роботи було досліджено існуючі алгоритми пошуку шляху в режимі реального часу. Серед досліджених алгоритмів за основу було обрано алгоритм на основі випадкових дерев.

Для обраного алгоритму RT-RRT* запропоновано модифікацію, що замінює алгоритм розширення дерева та алгоритм індексації дерева. Новий алгоритм розширення дерева базується на побудові *області видимості* навколо вершин дерева та генерації точок всередині цієї області. Це забезпечує те, що будь-яка згенерована точка всередині області видимості може бути гарантовано приєднана принаймі до тих вершин, навколо яких будувалась область видимості. Модифікація алгоритму індексації дерева пропонує замінити grid-based індексацію на використання KD-дерев.

Порівняно складність обох алгоритмів та наведено доведення ймовірнісної повноти, що виконується для цих алгоритмів. Для модифікації алгоритму показано, що швидкість збіжності алгоритму є більшою ніж для оригінального алгоритму, відносно кількості ітерацій, завдяки запропонованим модифікаціям.

ПЛАНУВАННЯ ШЛЯХУ, ВИПАДКОВІ ДЕРЕВА, ТРАСУВАННЯ ПРОМЕНІВ

РЕФЕРАТ

Квалификационная работа содержит 52 страницы, 10 рисунков, 2 таблицы, 14 источников.

Целью работы был анализ и модификация алгоритма планирования пути на основе случайных деревьев. Объектом исследования был процесс планирования пути в динамичесской среде. Предметом исследования были алгоритмы планирования пути в динамичесской среде на основе случайных деревьев.

В результате выполнения работы было исследовано существующие алгоритмы поиска пути в режиме реального времени. Среди исследованых алгоритмов в качестве основного был обран алгоритм на основе случайных деревьев.

Для выбранного алгоритма RT-RRT* предложено модификацию, которая заменяет алгоритм расширения дерева и алгоритм индексации дерева. Новый алгоритм расширения дерева базируется на построении *области видимости* вокруг вершин дерева и генерации точек внутри этой области. Это гарантирует, что любая сгенерированная точка внутри области видимости гарантированно может быть присоединена хотя бы к тем вершинам, вокруг которых строилась область видимости. Модификация алгоритма индексации дерева предлагает заменить grid-based индексацию на использование KD-деревьев.

Проведено сравнение обоих алгоритмов и представленно доказательство вероятностной полноты, которое выполняется для этих алгоритмов. Для модификации алгоритма показано, что скорость сходимости алгоритма больше чем для оригинального алгоритма, относительно количества итераций, благодаря предложенным модификациям.

ПЛАНИРОВАНИЕ ПУТИ, СЛУЧАЙНЫЕ ДЕРЕВЬЯ, ТРАССИРОВКА ЛУЧЕЙ

ABSTRACT

Qualification work contains 52 pages, 10 figures, 2 tables, 14 sources.

The point of this work was analysis and modification of path-planning algorithm based on random trees. Object of the research was process of path-planning in dynamic environments. Subject of the research was path-planning algorithm in dynamic environments based on random trees.

As a result, path-planning algorithms in dynamic environments were studied. Among studied algorithms, one was selected as a base algorithm which is based on random trees.

For the selected algorithm RT-RRT* modification was suggested, that replaces expansion and tree-indexing algorithms. New expansion algorithm is based on building a *visibility region* around vertices of the tree and generating new points inside this region. This guarantess, that any samples point inside visibility region can be connected at least to those vertices around which visibility region was built. Modification of the tree-indexing algorithm suggest replacing grid-based indexing with KD-trees.

Compares both algorithms and presented a proof of their probabilistic completeness. For the modification was shown that convergence speed is higher than that of the original algorithm, in terms of number of iterations, due to the described modifications.

PATH PLANNING, RANDOM TREES, RAY CASTING

ВСТУП

Планування шляху є проблемою знаходження шляху між двома точками в середовищі, що зустрічається в таких областях, як робототехніка, self-driving cars, комп'ютерних іграх та інших.

Ця проблема є нетривіальною, так як вона потребує вирішення компромісу між оптимальністю шляху та часом що потрібен для пошуку. Завдання з планування шляху стає ще більш нетривіальним, якщо середовище, що використовується в алгоритмі, є динамічним, тобто перешкоди та цільова точка можуть змінювати свої позиції.

Реагування до змін в середовищі, одночасно шукаючи оптимальний шлях називається **плануванням шляху в режимі реального часу**.

Більшість алгоритмів, що використовуються в іграх базуються на A*[1] алгоритмі. Проте, так як A* вимагає дискретизації середовища, його роздільна здатність а також складність форми перешкод мають великий ефект на час необхідний для пошуку шляху.

Модифікації алгоритму A*, що працюють в режимі реального часу успадковують таку ж проблему, що робить їх малопридатними для такого режиму застосування.

Інший підхід до планування шляху, полягає в тому що конфігурацію агента розглядають як точку в потенційному полі разом з тяжінням до цільової точки та відштовхуванням від перешкод. В такому випадку кінцева траекторія і використовується як шлях. Такі алгоритми, що називаються **Штучні Потенційні Поля**[2], мають перевагу в тому, що для побудови траекторії потрібні невеликі обчислювання, що дозволяє алгоритмам працювати в режимі реального часу. Проте їх недоліком є те, що вони легко застрягають в локальному мінімумі, будучи не в змозі знайти шлях.

Алгоритми, що базуються на генерації вибірки, такі як **Rapidly-Exploring Random Trees**[3] спроектовані таким чином, щоб ефективно здійснювати пошук по багатовимірним просторам шляхом

побудови випадкового дерева, що заповнює цей простір. Дерево будується ітеративно, використовуючі випадково згенеровані точки з середовища, та є по суті упередженим відносно росту в напрямку великих недосліджених регіонів середовища. Вони легко вирішують проблему з перешкодами та були широко застосовні в робототехніці.

Проте їх модифікації для роботи в режимі реального часу або будують дерево спочатку, кожен раз як корінь дерева змінює свою позицію, або розблять обрізання гілок, що є обчислювально нездійсненними.

Алгоритм, що розглядається в цій роботі **Real-Time Rapidly-Exploring Random Trees***[4], запропонований Naderi та іншими, дозволяє використовувати дерево побудоване на попередній ітерації алгоритму, а також містить процедури для реагування до змін позицій перешкод та кореня дерева, що надає йому значну перевагу над існуючими алгоритмами.

В наступному розділі буде формально поставлена задача пошуку шляху, надано коротокий огляд алгоритму Real-Time Rapidly-Exploring Random Trees*, та розгялнено властивості ймовірнісної повноти алгоритму RRT*.

1 ПРОБЛЕМА ПЛАНУВАННЯ ШЛЯХУ

В цьому розділі буде надано формальне означення проблеми, наведено основні поняття, що використовуються при вирішенні проблеми планування шляху.

1.1 Постановка задачі

Нехай $\chi \subseteq \mathbb{R}^2$ — обмежений простір, в якому відбувається планування шляху. $\chi_{obs} \subset \chi$ — множина усіх перешкод в просторі, яка є відомою. Тоді $\chi_{free} = \chi \setminus \chi_{obs}$ визначає вільний від перешкод простір.

Визначення 1.1.1 ***Дерево*** *— скінченна множина $\mathcal{T}$ з однією або більше вершин (вузлів), яка задовольная вимогам: 1) існує один відокремлений вузол — корінь дерева; 2) інші вузли (за винятком кореня) розподілені серед $m \geq 0$ непересічних множин $\mathcal{T}_1, \ldots, \mathcal{T}_m$ (піддерев) і кожна з цих множин, в свою чергу, є деревом. Вузли з'єднані між собою ребрами. Послідовність вершин і ребер, що з'єднують вузол з нащадком, називається* ***шляхом****.*

Дерево позначається через $\mathcal{T}$, та кожна вершина в дереві позначається як $x_i \in \mathcal{T}$. Множина усіх вершин дерева як $\mathcal{V} = \{x_0, \ldots, x_n\}$, де x_0 — корінь дерева $\mathcal{T}$. Множина усіх гілок позначається як $\mathcal{E} = \{\langle x_p, x_c \rangle\}$, де $x_p, x_c \in \mathcal{V}$, та $Parent\,(x_c) = x_p$, а $Child\,(x_p) = x_c$. $\sigma_{curr} = (x_0, x_1, \ldots, x_k)$ позначає поточний запланований шлях на k кроків вперед, де k наперед задана константа.

Проблема планування шляху в динамічному середовищі в режимі реального часу визначається наступним чином.

Визначення 1.1.2 (Проблема планування шляху) *Нехай задано* $(\chi, \chi_{free}, x_{init}, x_{goal})$, *де* $_{init} \in \chi_{free}$ *— початкова позиція з якої необхідно почати процедуру планування шляху (*x_{init} *також приймається за початкову позицію кореня* x_0 *дерева* $\mathcal{T}$*);* $x_{goal} \in \chi_{free}$ *— цільова позиція, до якої необхідно знайти шлях.*

Шлях *— це функція* $\sigma : \mathbb{Z}^+ \to \chi_{free}$, *така що* $\sigma(0) = x_0$, *а* $\sigma(k) = x_{goal}$, *де* $k < \infty$ *довжина шляху.*

Проблема планування шляху *тоді полягає в тому аби знайти шлях* σ, *між позицією агента* x_0 *та цільовою точкою* x_{goal} *шляхом розширення дерева* $\mathcal{T}$. *Знайденний шлях повинен мати мінімальну довжину, що визначається як сума евклідових відстаней між точками в шляху:*

$$c_\sigma = \sum_{j=0}^{k-1} dist\left(\sigma(j), \sigma(j+1)\right).$$

Окрім того, алгоритм повинен працювати в ***режимі реального часу****, що означає що час на розширення дерева та час на планування шляху — обмежений, множини* χ_{obs}, χ_{free} *можуть змінюватися з часом, а* x_0, x_{goal} *змінювати свої позиції в межах* χ_{free}.

1.2 Обмеження на рух перешкод

Для можливості аналізу ефективності алгоритму та порівняння з ориганільним алгоритмом RT-RRT* необхідно ввести наступні обмеження на рух перешкод в середовищі:

1) Середовища в яких цей алгоритм може використовуватись, ігрові рівні (комп'ютерні ігри), дороги (self-driving cars), мають дуже низьку ймовірність того, що рух перешкод буде повністю хаотичним, так як зазвичай перешкоди рухаються в таких середовищах за певними законами та з якоюсь ціллю. Також не розглядається випадок, коли існує зловмисник,

який керує перешкодами таким чином, аби запобігти знаходження шляху алгоритмом, через те, що це вже інша задача.

2) Відстань між позицією перешкоди на i-й ітерації та позицією на $i+1$-й ітерації не є більшою за деяку $\Delta << 1$, тобто для перешкоди b виконується $dist\left(p_{b,i}, p_{b,i+1}\right) < \Delta$, де $p_{b,i}$ та $p_{b,i+1}$ — позиції перекоди b на i-й та $i+1$-й ітерації алгоритму відповідно.

3) Проблему планування шляху вважається *надійно здійсненною* 4.1.3, тобто в будь-який момент часу в середовищі χ для деякого значення > 0 знайдеться такий шлях $\sigma \in \chi_{free}$, що для кожного $\sigma\left(i\right) \in \chi_{free}, i \in 0, \ldots, k$, відстань до найближчої перешкоди буде на меншою за δ.

1.3 Висновки до Розділу 1

В цьому розділі було наведено означення проблеми пошуку шляху в режимі реального часу в динамічних середовищах, надано основні поняття, що будуть необхідні для подальшої роботи. Введено обмеження на рух перешкод в середовищах, для можливості аналізу ефективності оригінального алгоритму та його модифікації.

В наступному розділі буде розглянуто оригінальний алгоритм RT-RRT*, а його модифікацію наведено в Розділі 3.

2 ОГЛЯД АЛГОРИТМУ RT-RRT*

В цьому розділі буде наведено алгоритм Real-Time Rapidly-Exploring Random Trees* (RT-RRT*), який використовується за основу для модифікації.

2.1 Алгоритм RT-RRT*

Алгоритм 1 чередує процедури розширення та перев'язування дерева з процедурою планування шляху та переміщення.

Алгоритм 1: Алгоритм RT-RRT*

Вхід: $x_a, \chi_{obs}, x_{goal}$

Ініціалізація дерева з x_a в якості кореня, Q_r, Q_s
while $dist\,(x_a, x_{goal}) > \epsilon$ **do**
  Оновити $x_{goal}, \chi_{free}, \chi_{obs}$
  while *час для розширення та перев'язки дерева не вичерпано* **do**
    Провести процедуру розширення 2.1.1 та перев'язки дерева $\mathcal{T}$ 2.1.5
  Побудова шляху $(x_0, x_1, \ldots, x_k)$ 2.1.6
  if $dist\,(x_a, x_0) \leq \epsilon$ **then**
    $x_0 \leftarrow x_1$
  Пересування x_a до x_0 доки є час

Дерево ініціалізується з корнем в точці x_a, що є поточною позицією агента. На кожній ітерації, дерево розширюється та перев'язується певний наперед заданий час. Потім відбувається планування шляху на k кроків вперед, де шлях починається з кореня дерева. На кожній ітерації агент пересувається в напрямку кореня дерева, таким чином корень дерева задає поточну ціль, до якої пересувається агент на кожній ітерації. Алгоритм завершує свою роботу, коли корень дерева та агент знаходяться близько до цільової точки (евклідова відстань менше наперед заданої константи ϵ).

Алгоритм для пересування агента може бути будь-яким.

Q_r, Q_s — черги, що ініціалізуються пустими та використовуються в алгоритмі перев'язки дерева $\mathcal{T}$.

2.1.1 Розширення дерева

Розширення дерева наведена в Алгоритмі 2. Розширення дерева відбувається шляхом випадкового генерування точок x_{rand} до тих пір, поки дерево не покриє середовище повністю.

Алгоритм 2: Алгоритм розширення дерева

Вхід: $\mathcal{T}, Q_r, Q_s, k_{max}, r_s$

Генерація випадкової точки x_{rand} 2.1.2
$x_{closest} = argmin_{x \in \chi_{SI}} \|x, x_{rand}\|$
if $line\,(x_{closest}, x_{rand}) \subset \chi_{free}$ **then**
    $\chi_{near} = FindNodesNear\,(x_{rand}, \chi_{SI})$
    if $|\chi_{near}| < k_{\max}$ *or* $|x_{closest} - x_{rand}| > r_s$ **then**
        $AddNodeToTree\,(\mathcal{T}, x_{rand}, x_{closest}, \chi_{near})$
        Додати x_{rand} в початок черги Q_r
    else
        Додати $x_{closest}$ в початок черги Q_r
    Перев'язка дерева починаючи з випадкової вершини $RewireRandomNode\,(Q_r, \mathcal{T})$
Перев'язка дерева починаючи з кореня $RewireFromRoot\,(Q_s, \mathcal{T})$

Згенеровані точки x_{rand} завжди використовуються для перев'язки випадкових частин дерева навколо себе або навколо найближчої вершини $x_{closest}$. Це необхідно для врахування змін в середовищі та зміни положення кореня дерева. k_{max} — задає максимальну кількість сусідів навколо вершини x_{rand}, а r_s — задає максимальну дозволену відстань між вершинами в дереві. Ці дві константи дозволяють котролювати щільність дерева в середовищі. χ_{near} — множина сусідніх вершин до x_{rand}. Вираз

$line\,(x_{closest}, x_{rand}) \subset \chi_{free}$ перевіряє чи шлях між $x_{closest}$ та x_{rand} не перетинає жодну з перешкод. Якщо умова виконується, то до дерева $\mathcal{T}$ додається нова вершина x_{rand}. Алгоритм додавання вершини до дерева $AddNodeToTree$ описано в Підрозділі 2.1.4.

В кінці відбувається перев'язка дерева, для урахування змін в середовищі та можливої зміни позиції кореня дерева. $RewireRandomNode$ та $RewireFromRoot$ виконують перев'язку дерева, та описані більш детально в Підрозділі 2.1.5.

2.1.2 Генерація точок

Генерації точок x_{rand} в Алгоритмі 2 виконується наступним чином:

$$x_{rand} = \begin{cases} LineTo(\mathbf{x}_{goal}), & \text{якщо } P_r > 1 - \alpha \\ Uniform(\chi), & \text{якщо } P_r \leq \frac{1-\alpha}{\beta} \text{ або } \nexists\, path(\mathbf{x}_0, \mathbf{x}_{goal}) \\ Ellipse(\mathbf{x}_0, \mathbf{x}_{goal}), & \text{в усіх інших випадках.} \end{cases}$$

P_r — випадкове число в $[0, 1]$ проміжку, α — мала наперед задана константа (наприклад 0.1). Константа $\beta \in \mathbb{R}$ використовується для розрізнення між $Uniform$ та $Ellipsis$ генерацією.

$LineTo\,(x_{goal})$ — рівномірна генерація точок вздовж лінії між x_{goal} та найближчою до x_{goal} вершиною дерева $\mathcal{T}$. $Uniform\,(\chi)$ — рівномірна генерація точок в χ. $Ellipsis\,(x_0, x_{goal})$ — рівномірна генерація точок в еліпсі так, що шлях між x_0 та x_{goal} знаходиться всередині.

Для генерації в еліпсі x_0 та x_{goal} використовуються в якості фокальних точок, а його поперечний та продольні діаметри дорівнюють c_{best} та $\sqrt{c_{best}^2 - c_{min}^2}$ відповідно. c_{best} — довжина шляху між x_0 та x_{goal}, а $c_{min} = \|x_0 - x_{goal}\|_2$. Орієнтація єліпса змінюється на кожній ітерації алгоритму, через зміну положення кореня дерева та зміни в середовищі.

2.1.3 Індексація дерева

Через те що дерево $\mathcal{T}$, побудоване на попередній ітерації, використовується на наступній ітерації, з часом воно стане занадто великим аби працювати в режимі реального часу. Тому Naderi та інші, в своїй роботі використовують grid-based індексацію дерева та працюють з $\mathcal{T}_{SI} \subset \mathcal{T}$ замість $\mathcal{T}$. Для того щоб для вершини x_i знайти $\mathcal{T}_{SI}$, спочатку середовище χ розбивається на однакового розміру квадратні регіони g_u. Для вершини x_i знаходиться відповідний їй регіон g_i до якого додаються усі сусідні регіони. Вершини що належать цим регіонам і формують $\mathcal{T}_{SI}$.

2.1.4 Додавання нової вершини до дерева

Алгоритм 3: Алгоритм додавання нової вершини

Вхід: $\mathcal{T}, x_{new}, x_{closest}, \chi_{near}$

$x_{min} = x_{closest}$
$c_{min} = cost\left(x_{closest}\right) + dist\left(x_{closest}, x_{new}\right)$
for $x_{near} \in \chi_{near}$ **do**
    $c_{new} = cost\left(x_{near}\right) + dist\left(x_{near}, x_{new}\right)$
    if $c_{new} < c_{min}$ $ma\ line\left(x_{near}, x_{new}\right) \in \chi_{free}$ **then**
        $c_{min} = c_{new}$
        $x_{min} = x_{near}$
$\mathcal{V} \leftarrow \mathcal{V} \cup \{x_{new}\}$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{x_{min}, x_{new}\}$

Коли необхідно додати нову вершину x_{new} до дерева $\mathcal{T}$, Алгоритм 3 знаходить вершину з наймешним значенням функції штрафу c_i серед χ_{near}. Так як значення c_i залежить від довжини шляху від x_0 до x_i, то необхідно обчислювати c_i кожен раз, як будь-яка з вершин в шляху $x_0 \rightarrow x_i$ змінилася. Якщо для якоїсь вершини x_i хоча б один з її батьків

заблокований перешкодою, то значення функції штрафу стає $c_i = \infty$ і ця вершина також стає заблокованою.

$\mathcal{V}, \mathcal{E}$ — множини вершин та гілок дерева $\mathcal{T}$ відповідно. Коли вершина додана до дерева 1) вона розширює дерево; 2) якщо вершина всередині χ_{goal}, шлях до x_{goal} знайдено; 3) необхідно оновити суміжні регіони для пошуку по дереву 2.1.3. Структура дерево використовується для побудови самого дерева і кожна вершина має доступ до свого нащадка та батька. Тобто $\mathcal{E}$ використовуються тільки для визначення.

2.1.5 Перев'язка дерева

Перев'язування відбувається тоді, коли для вершини x_i має менше значення функції штрафу c_i, якщо провести шлях від x_0 до x_i через іншу вершину x_j, а не через її поточного батька, тобто виконується:

$$c_j + dist\,(x_j, x_i) < c_i.$$

Тому перев'язка повинна відбуватися як для нових вершин, так і для вершин навколо нових вершин, через те, що корінь дерева постійно змінює свою позицію та через зміни в середовищі.

Алгоритми 4 та 5 перев'язують вершини навколо x_r та x_s відповідно. Одже перев'язування для вершини x_{near} відбувається, якщо при зміні батька $Parent\,(x_{near})$ на x_r (x_s), значення функції штрафу для x_{near} зменшується. Різниця між Алгоритмами 4 та 5 в точках перев'язування.

Алгоритм 4 перев'язує випадкові частини дерева починаючи з вершин навколо x_{rand} або $x_{closest}$, що були додані до $\mathcal{Q}_r$ в Алгоритмі 2. Якщо перев'язування відбувається для будь-якої вершини навколо x_{near}, то Алгоритм 4 додає x_{near} до $\mathcal{Q}_r$, так як вершини навколо x_{near} можуть бути перев'язані.

Через додавання вершин, з можливою необхідністю в перев'язці їх

сусідів, до $\mathcal{Q}_r$, Алгоритм 4 підсилює ефект випадкової генерації точок, що використовується в Алгоритмі 2.

Алгоритм 4: Перев'язування дерева починаючи з випадкової вершини

Вхід: $Q_r, \mathcal{T}$

while *час для перев'язування не вичерпано або* Q_r *не пуста* **do**
  $x_r = pop\,(Q_r)$
  $\chi_{near} = FindNodesNear\,(x_r, \chi_{SI})$
  for $x_{near} \in \chi_{near}$ **do**
    $c_{old} = cost\,(x_{near})$
    $c_{new} = cost\,(x_r) + dist\,(x_r, x_{near})$
    if $c_{new} < c_{old}$ *та* $line\,(x_r, x_{near}) \in \chi_{free}$ **then**
      $\mathcal{E}_\mathcal{T} = (\mathcal{E}_\mathcal{T} \setminus \{Parent\,(x_{near}), x_{near}\}) \cup \{x_r, x_{near}\}$
      Додати x_{near} в кінець Q_r

Алгоритм 5: Перев'язування дерева починаючи з кореня

Вхід: $Q_s, \mathcal{T}$

if Q_s *порожня* **then**
  додати корінь x_0 до Q_s
while *час для перев'язування не вичерпано або* Q_s *не пуста* **do**
  $x_s = pop\,(Q_s)$
  $\chi_{near} = FindNodesNear\,(x_s, \chi_{SI})$
  for $x_{near} \in \chi_{near}$ **do**
    $c_{old} = cost\,(x_{near})$
    $c_{new} = cost\,(x_s) + dist\,(x_s, x_{near})$
    if $c_{new} < c_{old}$ *та* $line\,(x_s, x_{near}) \in \chi_{free}$ **then**
      $\mathcal{E}_\mathcal{T} = (\mathcal{E}_\mathcal{T} \setminus \{Parent\,(x_{near}), x_{near}\}) \cup \{x_s, x_{near}\}$
    if x_{near} *не була в* Q_s **then**
      Додати x_{near} в кінець Q_s

Проте Алгоритм 5 починає перев'язування навколо x_0, а одже і навколо позиції агента. Одже перев'язування починається з x_0 та всі сусідні вершини до x_0 додаються до $\mathcal{Q}_s$. Потім алгоритм перев'язує вершини, що знаходяться

все далі від x_0, та додає x_{near} навколо x_s до $\mathcal{Q}_s$, доти не досягнута умова виходу.

Аби запобігти циклічному додаванню однієї і тієї ж вершини в чергу, перед додаванням перевіряється, чи ця вершина вже була в черзі. Якщо x_0 змінює свою позицію, то $\mathcal{Q}_s$ дорівню пустій черзі, таким чином перев'язування починається знову з кореня дерева.

2.1.6 Планування шляху

Алгоритм 6 планування шляху на кожній ітерації будує шлях $(x_0, x_1, \ldots, x_k)$ на k кроків вперед. Де k наперед визначена константа.

Якщо дерево досягло при розширені цільової точки x_{goal}, то шлях від x_0 до x_{goal} вже знаходиться в дереві. В цьому випадку все що потрібно це оновити шлях після перев'язування дерева.

У випадку, коли дерево не досягло цільової точки x_{goal}, алгоритм знаходить такий шлях довжини k, який приведе якомога ближче до x_{goal}. Для оцінювання оптимальності шляху використовується функція витрат

$$f_i = c_i + h_i, h_i = \begin{cases} \|x_i - x_{goal}\|, & line\,(x_i, x_{goal}) \in \chi_{free} \\ \infty, & line\,(x_i, x_{goal}) \notin \chi_{free} \end{cases}$$

Проте використання такої функції витрат може призвести до того, що алгоритм застрягне в локальному мінімумі. Для цього на кожній ітерації після планування шляху, вершини, що потрапили до шляху блокуються від повторного розглядання. Знайдений шлях використовується як новий, якщо він приводить ближче до x_{goal} ніж поточний.

Заблоковані вершини розблоковуються тоді, коли до них приєднуються

нові незаблоковані вершини в результаті перев'язки дерева.

Алгоритм 6: Планування шляху

Вхід: $\mathcal{T}, x_{goal}$

Вихід: $(x_0, \ldots, x_k)$

if *дерево* $\mathcal{T}$ *досягло* x_{goal} **then**
  Оновити шлях від x_{goal} до x_0, якщо дерево було перев'язано.
  Повернути $(x_0, \ldots, x_{goal})$
for $x_i \in (x_1, \ldots, x_k)$ **do**
  $x_i = \underset{x_c \in child(x_{i-1})}{\operatorname{argmin}} \; cost\,(x_c) + \|x_c - x_{goal}\|$
  if x_i *листок, або його діти заблоковані* **then**
    $(x_0^*, \ldots, x_k^*) \leftarrow (x_0, \ldots, x_i)$
    Заблокувати x_i та вийти з циклу
Оновити шлях $(x_0, \ldots, x_k)$, якщо $(x_0^*, \ldots, x_k^*)$ кращий шлях ніж поточний кращий шлях.
Повернути $(x_0, \ldots, x_k)$

2.2 Висновки до Розділу 2

В цьому розділі було наведено детальний опис алгоритму Real-Time Rapidly-Exploring Random Trees*, для пошуку шляху в режимі реального часу в динамічих середовищах.

В наступному розділі буде наведено модифікацію алгоритму RT-RRT*.

Теоретичний апарат, необхідний для аналізу алгоритмів та сам аналіз алгоритму RT-RRT* та його модифікації наведено в Розділі 4.

3 МОДИФІКАЦІЯ АЛГОРИТМУ RT-RRT*

В цьому розділі буде описано техніки що використовувалися для модифікації оригінального алгоритму RT-RRT* та наведено самий алгоритм.

Основною відмінністю від оригінального алгоритму RT-RRT* є алгоритм генерації нових точок для розширення дерева. В RT-RRT* одним з видів генерації є рівномірна генерація точок в χ 2.1.2. Проблемою такої генерації є те, що вона відбувається без урахування позицій перешкод, що може призвести до того, що умова $line\,(x_{closest}, x_{rand}) \subset \chi_{free}$ не буде виконуватись і алгоритм буде витрачати час тільки на генерацію точок, без розширення самого дерева.

Для запобігання цього, необхідно генерувати точки в χ з урахуванням позицій перешкод. Тобто генерувати точки в $\chi_{vis} \subseteq \chi_{free} \subseteq \chi$, де χ_{vis} є "областю видимості" певної вершини дерева $x_{vis} \in \mathcal{T}$. χ_{vis} будується таким чином аби будь-яка точка x_{rand}, згенерована всередині χ_{vis}, гарантовано могла бути приєднана до дерева $\mathcal{T}$. Алгоритм генерації точок наведено в Розділі 3.1.2, а сам алгоритм розширення в Розділі 3.1.1.

3.1 Модифікований алгоритм RT-RRT*

Головна частина алгоритму залишається незмінною як і в оригінальному алгоритмі RT-RRT* 1.

Алгоритм 7: Алгоритм RT-RRT*

Вхід: $x_a, \chi_{obs}, x_{goal}$

Ініціалізація дерева з x_a в якості кореня, Q_r, Q_s
while $dist\,(x_a, x_{goal}) > \epsilon$ **do**
  Оновити $x_{goal}, \chi_{free}, \chi_{obs}$
  while *час для розширення та перев'язки дерева не вичерпано* **do**
    Провести процедуру розширення 3.1.1 та перев'язки дерева $\mathcal{T}$ 3.1.5
  Побудова шляху $(x_0, x_1, \ldots, x_k)$ 2.1.6
  if $dist\,(x_a, x_0) \leq \epsilon$ **then**
    $x_0 \leftarrow x_1$
  Пересування x_a до x_0 доки є час

Функція відстані задана наступним чином $dist\,(x, y) = \|x - y\|_2$. Q_r, Q_s — черги, що ініціалізуються пустими та використовуються в алгоритмі перев'язки дерева $\mathcal{T}$.

3.1.1 Розширення дерева

Алгоритм 8 спочатку перевіряє, чи знаходиться x_{goal} в межах досяжності дерева, тобто чи можна від k найближчих до x_{goal} вершин провести лінію не перетинаючи перешкоди. Якщо можна, то x_{goal}

приєднується до дерева і на цьому розширення дерева завершується.

Алгоритм 8: Алгоритм розширення дерева

Вхід: $\chi_{free}, x_{goal}, \mathcal{T}$

if $TargetInReach\left(\mathcal{T}, x_{goal}\right)$ **then**
    Приєднати x_{goal} до дерева $\mathcal{T}$, та завершити роботу.
$x_{vis}, x_{rand} \leftarrow$ генерація нової точки з використанням Алгоритму 9
$nodesNear = FindNodesNear\left(\mathcal{T}, x_{rand}\right)$
if $|nodesNear| < k_{max}$ *або* $dist\left(x_{vis}, x_{rand}\right) > r_s$ **then**
    Приєднати x_{rand} до x_{vis}
    Додати x_{rand} в початок черги $\mathcal{Q}_r$
else
    Додати x_{vis} в початок черги $\mathcal{Q}_r$

Інакше, використовуючи Алгоритм 9, генерується нова точка, яка приєднується до x_{vis}, у випадку, якщо виконується умова $|nodesNear| < k_{max}$ або $dist\left(x_{vis}, x_{rand}\right) > r_s$, де k_{max} — максимальная кількість сусідів навколо вершини x_{rand}, а r_s — максимальна відстань між вершинами в дереві, де x_{vis} — вершина дерева, з якої будувалася область видимості.

Вершина x_{rand}, додається в початок черги $\mathcal{Q}_r$, що використовується для перев'язування дерева. Якщо умова не виконується, то x_{vis} додається в початок черги $\mathcal{Q}_r$.

3.1.2 Генерація точок

Однією з головних відмінностей від оригінального алгоритму RT-RRT* являється алгоритм генерації нових точок x_{rand} для дерева. В RT-RRT* для кожної випадково згенерованої точки на кожному кроці робиться перевірка чи можна провести лінію від нової точки до найближчої вершини дерева не перетинаючи перешкоди: $line\left(x_{near}, x_{new}\right) \in \chi_{free}$. Якщо $line\left(x_{near}, x_{new}\right) \notin \chi_{free}$, то алгоритм перевіряє наступну точку $x_{near} \in \chi_{near}$.

Якщо для всіх точок з χ_{near} не вдалося знайти ту, до якої можна приєднати нову точку x_{new}, то генерація починається спочатку.

Очевидно що цей етап може тривати довгий час у випадку, якщо в середовищі багато перешкод, так як генерація відбувається рівномірно в середовищі (або в еліпсі), без урахування позицій перешкод.

Для того аби прибрати необхідність в перевірці $line\,(x_{near}, x_{new}) \in \chi_{free}$ в запропонованій в цій роботі модифікації RT-RRT* використовується алгоритм генерації точок, що будує для обраної вершини дерева "область видимості" цієї вершини, та генерує нову точку випадково в цій області. Таким чином, нова згенерована точка гарантовано може бути приєднана до дерева $\mathcal{T}$.

Алгоритм побудови "області видимості" наведено в Підрозділі 3.1.3.

Нижче наведено сам алгоритм генерації точок.

Алгоритм 9: Алгоритм генерації точок

Вхід: $\chi_{free}, x_{goal}, \mathcal{T}, \alpha, \beta$

$d \leftarrow_{rand} [0, 1]$

$x_{vis}, c_{x_{vis}} \leftarrow$ Алгоритм 10

if $d < \beta$ **then**

  $x_{rand} \leftarrow_{rand} \mathcal{U}_{c_{x_{vis}, goal}}$

else

  $x_{rand} \leftarrow_{rand} \mathcal{U}_{c_{x_{vis}}}$

Вихід: x_{vis}, x_{rand}

Спочатку для випадково обраної вершини дерева $\mathcal{T}$ будується область видимості $c_{x_{vis}}$ використовуючи Алгоритм 10. Потім для наперед визначеної константи β перевіряється чи випадково згенероване число $d \leftarrow_{rand} [0; 1]$ менше ніж β. У випадку $d \geq \beta$, випадкова точка генерується випадково всередині $c_{x_{vis}}$ з рівномірного розподілу над $c_{x_{vis}}$ $x_{rand} \leftarrow_{rand} \mathcal{U}_{c_{x_{vis}}}$. Інакше, область видимості розділяється на дві частини прямою, що перпендикулярна прямій (x_{vis}, x_{goal}), та x_{rand} генерується в тій частині, що знаходиться ближче до x_{goal}, тобто $x_{rand} \leftarrow_{rand} \mathcal{U}_{c_{x_{vis}, goal}}$.

Таким чином константа β розрізняє генерацію в цілій області видимості, та "направленій" області видимості, що є певним аналогом

рівномірної генерації в цілому середовищі, та рівномірної генерації в еліпсі відповідно.

3.1.3 Область видимості точки

Для побудови області видимості з дерева $\mathcal{T}$ випадково обирається вершина x_{vis}. Параметр α дозволяє контролювати вибір точки з цілого дерева чи з k найближчих точок до x_{goal}.

Алгоритм 10: Алгоритм побудови області видимості

Вхід: $\chi_{free}, x_{goal}, \mathcal{T}, \alpha$

$w \leftarrow_{rand} [0, 1]$

if $w < \alpha$ **then**

  $x_{vis} \leftarrow_{rand} \mathcal{U}_{\mathcal{T}}$

else

  $\mathcal{T}_{near} \leftarrow FindNodesNear\,(x_{goal}, \mathcal{T})$

  $x_{vis} \leftarrow_{rand} \mathcal{U}_{\mathcal{T}_{near}}$

Знайти точки перетину променів з початком в x_{vis}.

$P = CastRays\,(\chi_{obs}, x_{vis})$

Побудувати з точок перетину замкнутий контур $c_{x_{vis}}$.

Вихід: $\{x_{vis}, c_{x_{vis}}\}$

Функція $CastRays$ — повертає точки перетену променів з перешкодами в χ_{obs}, випущених з x_{vis}. Обчислення функції $CastRays$ є фактично вирішенням проблеми трасування променів [5], яка є добре вивченною та для якої існує багато ефективних алгоритмів, наприклад на основі просторових дерев (KD-дерев).

Далі знаходяться точки перетину P з перешкодами в χ променів з початком в x_{vis}. Якщо випадково згенероване число $r < \beta$, де β наперед заданий параметр, то з точок перетину обираються тільки ті, що знаходяться в напрямку x_{goal}.

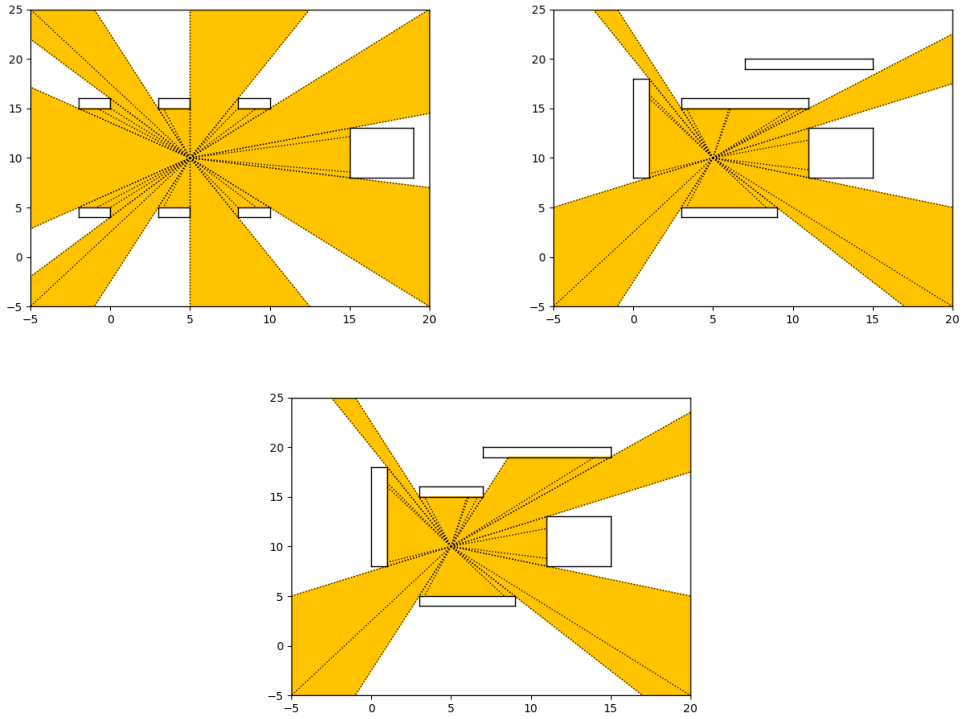

Рисунок 3.1 – Приклад області видимості точки в різних середовищах.

З відібраних точок перетину будується замкнутий контур $c_{x_{vis}}$, який і є областю видимості точки x_{vis}.

На Рисунку 3.1 наведено приклад того, як виглядає область видимості в середовищі з різними перешкодами. Жовтий регіон це і є область видимості, що була побудована з точки з якою починаються промені, а прямокутні регіони це перешкоди.

3.1.4 Індексація дерева

Для знаходження x_{near} та χ_{near} для x_{rand}, Naderi et.al. замість того аби працювати з цілим деревом $\mathcal{T}$, покривають простір сіткою з однакового розміру участками та працюють з піддеревом $\chi_{SI} \subseteq \mathcal{T}$.

Для того, аби знайти χ_{SI} навколо даної вершини x_u знаходиться відповідний участок сітки $g_u \ni x_u$. Тоді χ_{SI} складається з усіх тих вершин $\mathcal{T}$, що знаходяться в g_u та його сусідніх участках.

Проте, замість того аби розбивати простір на рівномірні участки, можна використовувати KD-дерево, що значно прискорює пошук.

KD-дерево [6] — це структура даних з поділом простору для упорядкування точок в k-мірному просторі. Кожна вершина в дереві є k-мірною точкою в просторі. Вершина дерева, що не являється листком дерева, поділяє простір на дві частини.

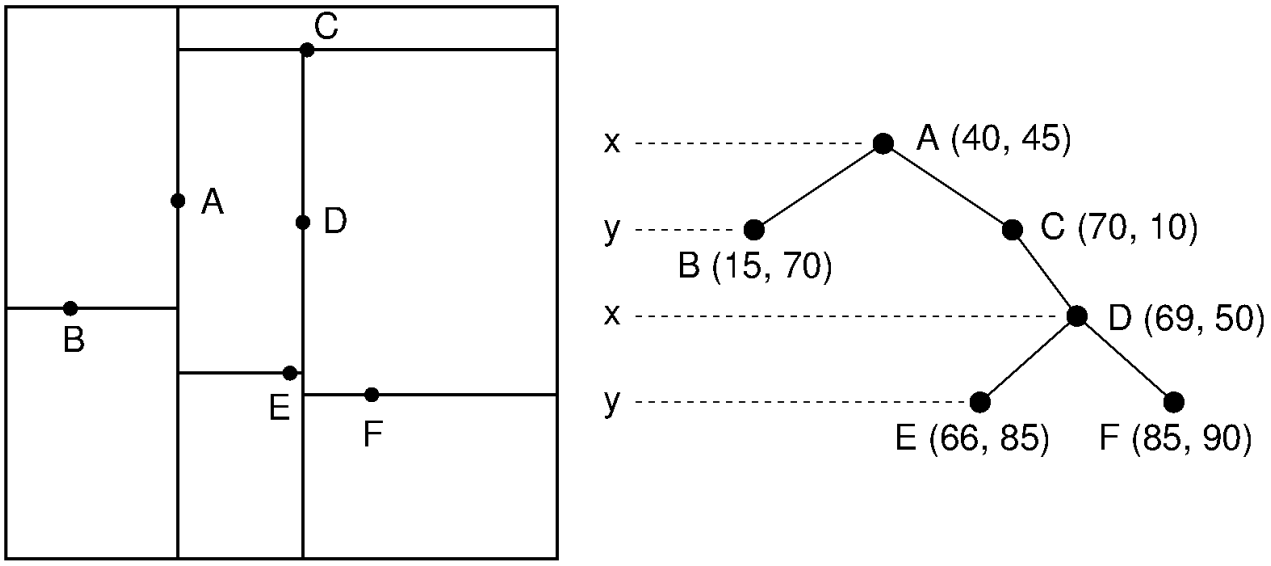

Рисунок 3.2 – Приклад KD-дерева, $K = 2$

Всі точки, що знаходяться в лівій частині простору, представлені в лівому піддереві, а точки в правій частині — правим піддеревом. Вісь, в якій робиться поділ простору для k-мірної точки x на глибині d, визначається наступним чином $a = d \mod k$, де $a \in [0, k)$ — номер вісі. Тоді ліве піддерево для цієї точки, будується з тих точок x_j для яких виконується $x_{j_a} < x_a$, а праве піддерево з $x_{j_a} \geq x_a$, де x_{j_a} дорівнює a-й координаті точки x_j.

Складність пошуку найближчого сусіда в збалансованому KD-дереві з випадково розподіленими вершинами $\mathcal{O}(\log n)$.

3.1.5 Перев'язка дерева

Перев'язка дерева відбувається так само, як і в оригінальному алгоритмі 2.1.5.

3.1.6 Пристосування дерева

Алгоритм перев'язки дерева 2.1.5 перев'язує ті вершини x_i , для яких значення функції штрафу c_i менше, якщо провести шлях від x_0 до x_i не через її поточного батька $Parent\,(x_i)$, а через іншу вершину x_j. Перев'язування починається або з випадкової вершини або з кореня дерева.

Через те, що вершини обираються випадково та через обмежений час на проведення процедури, деякі вершини дерева можуть не перев'язуватись. Через зміни в положені кореня дерева та змін в середовищі, це може привести до того, що ці вершини стануть заблокованими перешкодами. Одже і окремі гілки дерева $\mathcal{T}$ ставатимуть заблокованими і не будуть використовуватися в процедурі пошуку шляху. Це може привести до необхідності генерації нових точок та подальшого розширення дерева.

Для запобігання цьому, пропонується ввести процедуру "пристосування" (fitting) дерева до змін в середовищі. Ця процедура забезпечує те, що в будь-який момент часу всі шляхи $\sigma_i \forall i$ дерева $\mathcal{T}$ є не заблокованими, тобто дерево покриває максимально можливу частину середовища (якщо заблоковані шляхи також прийняти за непокриту частину).

Алгоритм 11 починає свою роботу з кореня дерева. Алгоритм працює доти, доки множина $\mathcal{X}_{chi}$ не стане пустою, де $\mathcal{X}_{chi}$ — множина, що на i-й

ітерації складається з нащадків вершин з $\mathcal{X}_{chi}$ на попередній ітерації.

Алгоритм 11: Алгоритм пристосування дерева

Вхід: $\mathcal{T}, \chi_{obs}$

$\mathcal{X} \leftarrow \{x_0\}$

$\mathcal{X}_{chi} \leftarrow \{Child(x_i) \mid \forall x_i \in \mathcal{X} : \exists Child(x_i)\}$

while $\mathcal{X}_{chi} \neq \emptyset$ **do**

  $\mathcal{X}_{ad} \leftarrow \{x_i \mid \forall x_i \in \mathcal{X}_{chi} : c_i = \infty\}$

  for $x_i \in \mathcal{X}_{ad}$ **do**

    $\mathcal{C} \leftarrow \{\langle x, cost(x)\rangle \mid \forall x \in FindNodesNear(x_i, \mathcal{T})\}$

    $x_a, c_a \leftarrow \arg\min_{\langle \_, c\rangle} \mathcal{C}$

    if $c_a \neq \infty$ **then**

      $\mathcal{E} \leftarrow (\mathcal{E} \setminus \{Parent(x_i), x_i\}) \cup \{x_a, x_i\}$

  $\mathcal{X}_{chi} \leftarrow \{Child(x_i) \mid \forall x_i \in \mathcal{X}_{chi} : \exists Child(x_i)\}$

Вихід: $\mathcal{T}$

Для кожної вершини в $\mathcal{X}_{chi}$ перевіряється її поточне значення функції штрафу. Якщо для якогось $x_i \in \mathcal{X}_{chi}$ значення $c_i = \infty$, то для цієї вершини знаходиться вершина $x_a \in FindNodesNear(x_i, \mathcal{T})$ з мінімальним значенням функції штрафу c_a, яка заміняє поточного батька (x_i) для вершини x_i. Якщо $c_a = \infty$, то вершина залишається заблокованою, через неможливість знайти нового батька.

Таким чином, даний алгоритм намагається максимально заповнити середовище деревом "виправляючі" заблоковані шляхи, таким чином зменшуючи необхідну кількість вершин в дереві $\mathcal{T}$ для пошуку шляху.

На Рисунку 3.3 наведено візуалізацію пристосування дерева до рухомих перешкод (чорні прямокутники).

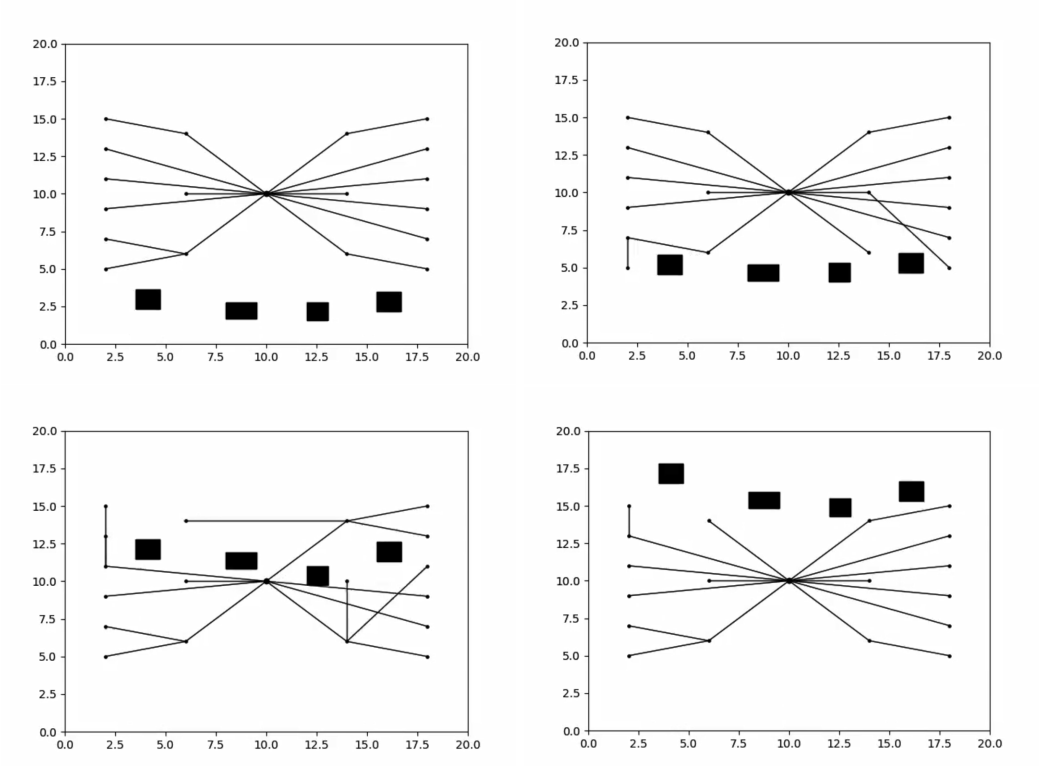

Рисунок 3.3 – Приклад пристосування дерева в середовищі з рухомими перешкодами.

3.2 Висновки до Розділу 3

В цьому розділі було наведено модифікацію алгоритму Real-Time Rapidly-Exploring Random Trees*. Головною відмінністю від оригінального алгоритму є метод генерації точок в середовищі, що відбувається за допомогою побудови "області видимості"навколо обраної випадковим чином вершини дерева $\mathcal{T}$.

Також запропоновано алгоритм "пристосування"дерева до середовища, який запезпечує максимальне покриття середовища деревом шляхом перев'язування заблокованих перешкодами вершин таким чином аби вони стали незаблокованими. Алгоритм намагається зменшити необхідну кількість вершин дерева для пошуку шляху.

В наступному розділі буде наведено порівняльний аналіз оригінального алгоритму RT-RRT* та його модифікації, наведеної в цьому розділі.

4 АНАЛІЗ АЛГОРИТМУ RT-RRT* ТА ЙОГО МОДИФІКАЦІЇ

В цьому розділі буде наведено необхідний теоретичний матеріал для аналізу складності та оптимальності алгоритму RT-RRT* та його модифікації, запропонованої в Розділі 3.

Розглянуто теореми які показують, що алгоритм RRT*, на основі якого базується RT-RRT* та його модифікація є ймовірнісно та асимптотично оптимальним.

4.1 Ймовірнісна повнота алгоритму

Ймовірнісна повнота алгоритму ALG забезпечує гарантію того, що з ростом розміру дерева $\mathcal{T}$, що використовується для знаходження шляху, ймовірність того, що вершина дерева потрапить в область цільової точки, прямує до 1.

Для надання формального означення ймовірнісної повноти необхідно спочатку ввести деякі допоміжні означення.

Визначення 4.1.1 *Нехай $\delta > 0 \in \mathbb{R}$. Точка $x \in \chi_{free}$, називається* ***δ-внутрішньою точкою*** *χ_{free}, якщо замкнена куля радіуса δ з центром в x $\mathcal{B}_{x,\delta}$ знаходиться повністю в χ_{free}.*

Множина всіх δ-внутрішніх точок χ_{free} позначається як $int_\delta(\chi_{free}) = \{x \in \chi_{free} | \, \mathcal{B}_{x,\delta} \subseteq \chi_{free}\}$.

Визначення 4.1.2 *Шлях без перешкод $\sigma \subset \chi_{free}$, називається* ***шляхом з сильним δ-зазором****, якщо $\sigma \subset int_\delta(\chi_{free})$.*

Іншими словами, шлях σ називається шляхом з сильним δ-зазором, якщо навколо кожної точки $x_i \in \sigma$, замкнена куля з центром в цій точці $\mathcal{B}_{x_i,\delta}$ буде повністю знаходитися в χ_{free}. Таким чином для кожної точки шляху σ відстань до найближчої перешкоди буде щонайменше $\delta > 0$.

Визначення 4.1.3 *Проблема планування шляху* $(\chi, \chi_{free}, x_{init}, x_{goal})$, *називається* ***надійно здійсненною****, якщо для деякого* $\delta > 0$ *існує шлях* σ *з сильним* δ*-зазором, який вирішує цю проблему.*

З означення *надійної здійсненності* проблеми планування шляху випливає, що відстань між перешкодами в будь-який момент часу не повинна бути меншою за деякую $\delta > 0$.

Тоді ймовірнісна повнота алгоритму ALG визначається наступним чином.

Визначення 4.1.4 (Ймовірнісна повнота) *Алгоритм* ALG *називається ймовірнісно повним, якщо для будь-якої надійно здійсненної проблеми* $(\chi, \chi_{free}, x_{init}, \chi_{goal})$ *виконується*

$$\lim_{n\to\infty} \inf \mathbb{P}\left(\left\{\exists\ x_{goal} \in \mathcal{V}_n^{ALG} \cap \chi_{goal} : (x_{init}, \ldots, x_{goal}) \subseteq \mathcal{T}^{ALG}\right\}\right) = 1$$

Якщо алгоритм $\mathbb{A}$ ймовірнісно повний та проблема планування шляху є надійно здійсненною, то гарантовано виконується

$$\lim_{n\to\infty} \mathbb{P}\left(\left\{\exists\ x_{goal} \in \mathcal{V}_n^{ALG} \cap \chi_{goal} : (x_{init}, \ldots, x_{goal}) \subseteq \mathcal{T}^{ALG}\right\}\right) = 1.$$

Ймовірнісна повнота алгоритму ALG для надійно здійсненної проблеми планування шляху забезпечує те, що алгоритм ALG при $n \to \infty$ з ймовірністю 1 вирішить цю проблему, де n — кількість вершин в дереві $\mathcal{T}$, що використовується для пошуку шляху,

Проте, для будь-якого алгоритму на основі випадкового генерування (включаючи ймовірнісно повні алгоритми), цей ліміт дорівнює нулю, якщо проблема планування шляху не є найдійно здійсненною.

Тому в цій роботі будуть розглядатися лише ті середовища χ, для яких буде виконуватись умова надійної здійсненності проблеми планування шляху.

З літератури[7][8] відомо, що алгоритм RRT є ймовірнісно повним. Також, якщо для проблеми планування шляху $(\chi, \chi_{free}, x_{init}, \chi_{goal})$ існує рішення, то ймовірність того, що воно буде знайдено експоненційно наближається до 1, при $n \to \infty$, це показано в наступних теоремах.

Теорема 4.1.1 (Ймовірнісна повнота RRT[7]) *Розглянемо надійно здійсненну проблему планування шляху* $(\chi, \chi_{free}, x_{init}, \chi_{goal})$. *Існує така константа* $a > 0$ *та* $n_0 \in \mathbb{N}$, *що залежать тільки від* χ_{free} *та* χ_{goal}, *такі що*

$$\mathbb{P}\left(\left\{\mathcal{V}_n^{RRT} \cap \chi_{goal} \neq \emptyset\right\}\right) > 1 - e^{-an}, \forall n > n_0,$$

де $\mathcal{V}_n^{RRT}$ *— множина вершин дерева* $\mathcal{T}$ *побудованого алгоритмом RRT на* n*-й ітерації.*

Тобто, для *надійно здійсненної проблеми* планування шляху та алгоритму RRT, ймовірність того, що множина вершин $\mathcal{V}_n^{RRT}$ дерева $\mathcal{T}_n^{RRT}$ на n-й ітерації, буде містити точку, яка знаходиться в цільовій області χ_{goal}, експоненційно наближається до 1, починаючи з деякого n_0.

Теорема 4.1.2 (Ймовірнісна повнота RRT*[9]) *Через те що* $\mathcal{V}_n^{RRT} = \mathcal{V}_n^{RRT*}$, *ймовірнісна повнота для RRT* є прямим наслідком ймовірнісної повноти RRT. Більш того, якщо RRT знайде шлях на* n*-му кроці, то і RRT*, *якщо генерація відбувається в одному й тому ж порядку.*

Далі буде наведено означення *асимптотичної оптимальності* алгоритму *ALG*.

4.2 Асимптотична оптимальність

З попереднього підрозділу алгоритм ALG називається ймовірнісно повним, якщо алгоритм вирішує проблему планування шляху з великою ймовірністю, для проблеми, яка є надійно здійсненною, тобто для якої існує шлях з *сильним δ-зазором*.

Для визначення асимптотичної оптимальності використовується схожий підхід, який полягається на означенні *слабкого δ-зазору* шляху та неперервності функції витрат шляху, що буде розглянуто далі.

Нехай $\sigma_1, \sigma_2 \subset \chi_{free}$ два шляхи без перешкод з однаковими кінцевими точками, тобто $\sigma_1(0) = \sigma_2(0) = x_0 \in \chi_{free}$ та $\sigma_1(k) = \sigma_2(k) = x_k \in \chi_{free}$. Шлях σ_1 називається гомотопічним до σ_2, якщо існує неперервна функція $\phi : [0,1] \to \chi_{free}$, *гомотопія*, така що $\phi(0) = \sigma_1, \phi(1) = \sigma_2$ і $\phi(\tau) \in \chi_{free}, \tau \in [0,1]$. Шлях що є гомотопічним до σ_2 може бути неперервно переведений в σ_2 через χ_{free}.

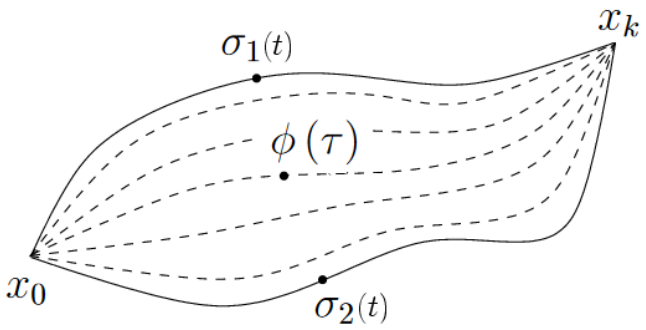

Рисунок 4.1 – Приклад гомотопії

Визначення 4.2.1 *Шлях без перешкод $\sigma : [0,s] \to \chi_{free}$ називається шляхом з* ***слабким*** *δ-**зазором**, якщо $\exists \sigma'$ з сильним δ-зазором, існує гомотопія $\phi : \phi(0) = \sigma, \phi(1) = \sigma'$ та $\forall \alpha \in (0,1]\ \exists \delta_\alpha > 0$ така, що $\phi(\alpha)$ має сильний δ_α зазор.*

Тобто якщо σ є шляхом зі *слабким δ-зазором*, то для нього необов'язково повинна виконуватись умова *сильного δ-зазору*, проте для

кожної його гомотопії $\phi(\alpha)$ — повинна.

На Рисунку 4.2 зліва наведено приклад шляху σ зі слабким δ-зазором. Шлях $\sigma' \in int_{delta}(\chi_{free})$ знаходиться в тому ж класі гомотопії, що й σ. Праворуч наведено приклад шляху σ, для якого не виконується умова слабкого δ-зазору. Для будь-якого значення $\delta > 0$ не існує шляху $\sigma' \in int(\chi_{free})$, що знаходиться в тому ж класі гомотопії що й σ.

Шлях $\sigma^* \subset \chi_{free}$, що вирішує проблему оптимальності називається *надійно оптимальним рішенням*, якщо він має слабкий δ-зазор, та для будь-якого набору шляхів без перешкод $\{\sigma_n \subset \chi_{free}\}_{n\in\mathrm{N}}$ виконується $\lim\limits_{n\to\infty}\sigma_n = \sigma^*$ та $\lim\limits_{n\to\infty} c(\sigma_n) = c(\sigma^*)$.

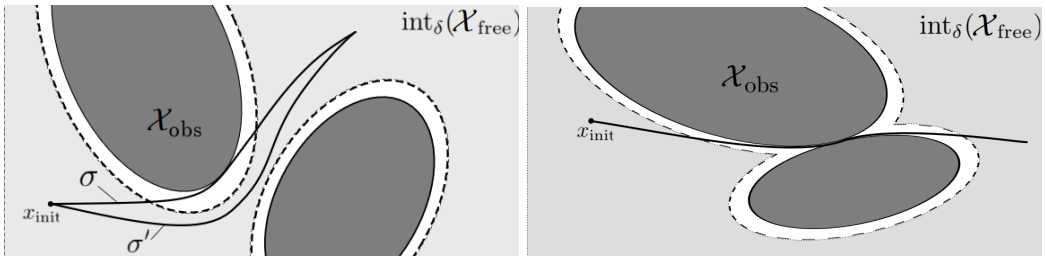

Рисунок 4.2 – Приклад шляху для якого виконується умова *слабкого δ-зазору* (лівий рисунок) та для якого не виконується (правий).

Нехай $c^* = c(\sigma^*)$, це функція витрат оптимального шляху та Y_n^{ALG} — випадкова величина, що відповідає значенню функції витрат шляху з мінімальним значенням функції витрат, що було повернено алгоритмом ALG на n-й ітерації. Тобто $Y_n^{ALG} \sim \min\limits_{\sigma\in\mathcal{T}} c(\sigma)$.

Визначення 4.2.2 *Алгоритм ALG називається* ***асимптотично оптимальним***, *якщо для будь-якої проблеми планування шляху $(\chi, \chi_{free}, x_{init}, \chi_{goal})$ та функції витрат $c : \Sigma \to \mathbb{R}_{\geq 0}$, що допускають наявність надійно оптимального рішення зі скінченним значенням функції втрат c^*,*

$$\mathrm{P}\left(\left\{\limsup_{n\to\infty} Y_n^{ALG} = c^*\right\}\right) = 1.$$

Так як $Y_n^{ALG} \geq c^*, \forall n \in \mathbb{N}$, асимптотична оптимальність алгоритму

ALG означає, що $\lim\limits_{n\to\infty} \sup Y_n^{ALG}$ існує і дорівнює c^*.

Таким чином асимптотична оптимальність алгоритму ALG гарантує, що при $n \to \infty$, значення функції втрат шляху поверненого алгоритмом ALG на n-й ітерації, буде наближатися до значення функції втрат оптимального шляху σ^*, що вирішує цю проблему планування шляху.

Очевидно, що для асимптотичної оптимальності необхідно аби виконувалась ймовірнісна оптимальність.

Далі розглянуто наведено теореми, що показують ймовірнісну та асимптотичну оптимальність алгоритму RRT*, на якому базується RT-RRT* та його модифікація.

Теорема 4.2.1 *Алгоритм RRT не є асимптотично оптимальним[9].*

Так як алгоритм RRT на кожній ітерації або додає нову вершину до дерева або залишає його незмінним, $\mathcal{V}_i^{RRT} \subseteq \mathcal{V}_{i+1}^{RRT}, i \in \mathbb{N}$, то границя $\lim\limits_{n\to\infty} Y_n^{RRT}$ існує і дорівнює деякій випадковій величині Y_∞^{RRT}. В теоремі 4.2.1 показується, що ця границя строго більша ніж c^* майже напевно, тобто $\mathbb{P}\left(\left\{\lim\limits_{n\to\infty} Y_n^{RRT} > c^*\right\}\right) = 1$. Тобто значення функції штрафу найкращого рішення поверненного алгоритмом RRT збігається до субоптимального значення з ймовірністю 1.

Теорема 4.2.2 *Алгоритм RRT* є асимптотично оптимальним[9].*

4.3 Обчислювальна складність

В цьому підрозділі буде порівняна обчислювальна складність алгоритму RT-RRT* та його модифікації, наведеної в попередньому Розділі 3.

4.3.1 Складність процедури *line*

Процедура $line\,(x_i, x_j)$ перевіряє, чи перетинає лінія між x_i та x_j будь-яку з перешкод, тобто чи $x_i, x_j \in \chi_{free}$.

Складність процедури $line$ в залежності від кількості перешкод в середовищі є добре дослідженною проблемою (Lin and Manocha 2004 [10] для огляду).

Основний результат складності процедури базується на роботі Six and Wood (1982) [11], де показано, що перевірка наявності колізії $line\,(x_i, x_j) \in \chi_{free}$ з m перешкодами може бути зроблена за $O\left(log^d m\right)$ час, де d — кількість вимірів ($d = 2$ в данному випадку), використовуючи структури даних на основі просторових дерев (наприклад KD-дерев).

4.3.2 Складність процедури побудови області видимості

Процедура побудови області видимості навколо вершини $x_{vis} \in \mathcal{V}$, що описана в Алгоритмі 10, знаходить точки перетину променів випущених з x_{vis} з перешкодами для побудови контуру $c_{x_{vis}} \subseteq \chi_{free}$, в середині якого буде відбуватися генерація точок.

Так як перешкоди представляються у вигляді опуклих 2D полігонів то, в найпростішому випадку, промені випускаються з x_{vis} в напрямку кожної з вершин $x \in \mathcal{V}_{obs}$ з яких складаються перешкоди, що дає $|\mathcal{V}_{obs}|$ променів.

Для зменшення кількості необхідних перевірок колізій, точки $x \in \mathcal{V}_{obs}$ можна фільтрувати використовуючи Quadtree [12] або Spatial Hashing [13] алгоритми, які залишають тільки ті точки, що знаходяться найближче до x_{vis} та перекриваючи інші точки, що знаходяться далі, для яких не потрібно робити перевірку колізій. Таким чином значно зменшуючи необхідну кількість перевірок. Проте для простоти аналізу припустимо, що беруться всі точки без фільтрації.

Так як згідно роботі Six and Wood (1982) [11], перевірка колізії з m перешкодами займає, в 2D випадку, $\mathcal{O}\left(log^2 m\right)$ часу. Тоді знаходження колізії для всіх променів може бути зроблено за $\mathcal{O}\left(|\mathcal{V}_{obs}| \log^2 m\right)$.

Таким чином процедура генерації точок хоч і є повільнішою для модифікації алгоритму RT-RRT* ніж для оригігнальної версії, проте гарантує те, що будь-яка згенерована всередині контуру $c_{x_{vis}}$ точка може бути приєднана принаймі до x_{vis} вершини дерева $\mathcal{T}$, що гарантує швидшу збіжність алгоритму відносно загальної кількості ітерацій необхідних для знаходження шляху, що буде наведено в TODO.

4.3.3 Складність процедури *FindNearestNodes*

Процедура $FindNodesNear\left(x, \chi\right)$ знаходить k найближчих вершин в χ для точки x.

Проблема пошуку найближчого сусіда є добре дослідженною, через широкий спектр можливих застосувань, таких як, комп'ютерна графіка, системи баз данних, обробка зображень, розпізнавання образів та інші.

Очевидно, що наївна реалізація алгоритму, що перевіряє кожну з вершин дерева виконується за $\mathcal{O}\left(n\right)$ час та потребує $\mathcal{O}\left(1\right)$ пам'яті. Проте в випадку алгоритмів, що працюють в режимі реального часу, однією з цілей є зменшення часу виконання кожної з ітерацій особливо для алгоритмів, що дають кращі рішення з ростом кількості ітерацій.

В оригінальному алгоритмі RT-RRT* для пошуку найближчого сусіда

точки x використовується рівномірне розбиття простору χ на однакового розміру участки g_i. Для побудови дерева $\mathcal{T}_{SI} \subseteq \mathcal{T}$ знаходиться відповідний їй участок g_x, тоді $\mathcal{T}_{SI}$ складається з усіх вершин, що належать цьому участку та прилеглим до g_x участкам. Найближчий сусід точки x тоді знаходиться шляхом перебору усіх вершин в $\mathcal{T}_{SI}$, з яких береться та вершина, що має найменшу відстань до x.

Нехай простір χ було розбито на M участків. Тоді пошук відповідного до x участку g_x виконується за $\mathcal{O}(M)$ час. А пошук найближчої до x вершини в $\mathcal{T}_{SI}$ — за $\mathcal{O}(|\mathcal{T}_{SI}|)$, шляхом перебору кожної з вершин, де $|\mathcal{T}_{SI}|$ — кількість вершин в $\mathcal{T}_{SI}$, яка збільшується з ростом кількості ітерацій. Таким чином загальна складність пошуку найближчої вершини $\mathcal{O}(M + |\mathcal{T}_{SI}|)$.

В модифікації алгоритму RT-RRT* для пошуку найближчої вершини дерева $\mathcal{T}$ для точки x використовується KD-дерево, що було описано в Підрозділі 3.1.4. Складність пошуку найближчої вершини дерева $\mathcal{T}$ дорівнює $\mathcal{O}(\log|\mathcal{T}|)$, де $|\mathcal{T}|$ — кількість вершин дерева $\mathcal{T}$.

Легко бачити, що $\mathcal{O}(M + |\mathcal{T}_{SI}|) > \mathcal{O}(\log|\mathcal{T}|)$, навіть з урахуванням того, що $\mathcal{T}_{SI} \subseteq \mathcal{T}$.

4.3.4 Складність пошуку шляху в дереві

Коли дерево $\mathcal{T}$ досягло x_{goal}, необхідно побудувати оптимальний шлях $\sigma = \{x_0, \ldots, x_{goal}\}$ використовуючи дерево $\mathcal{T}$.

Наступна Лемма 4.3.1 наводить асимптотичну складність обчислювання найкоротшого шляху. Нехай $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ — граф. $d : \mathcal{E} \to \mathbb{R}_{\geq 0}$ — функція відстані, що для кожної грані в $\mathcal{E}$ присвоює невід'ємну відстань. Для заданої вершини $v \in \mathcal{V}$, найкоротший шлях в $\mathcal{T}$ це граф $\mathcal{T}' = (\mathcal{V}, \mathcal{E}')$, де $\mathcal{E}' \subseteq \mathcal{E}$ такий що для будь-якого $v' \in \mathcal{V} \setminus \{v\}$ існує унікальний шлях в $\mathcal{T}$, що починається в v та досягає v' і окрім того цей шлях є оптимальним.

Лемма 4.3.1 *A. Schrijver, 2003 [14] Нехай дано граф $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, функцію відстані $d : \mathcal{E} \to \mathbb{R}_{\geq 0}$ та вершину $v \in \mathcal{V}$, найкоротший шлях для $\mathcal{T}, d$, та v може бути знайдений за $\mathcal{O}\left(|\mathcal{V}| \log\left(|\mathcal{V}|\right) + |\mathcal{E}|\right)$.*

Згідно [9], $|\mathcal{E}_n| \in \mathcal{O}\left(n\right)$.

4.3.5 Просторова складність

Просторова складність алгоритму визначається кількістю пам'яті, необхідної для обчислювання графу $\mathcal{T}_n = (\mathcal{V}_n, \mathcal{E}_n)$. Очевидно, що просторова складність як для RT-RRT* так і для його модифікації дорівнює розміру $\mathcal{T}_n$, тобто $\mathcal{V}_n + \mathcal{E}_n$.

4.4 Швидкість збіжності

В цьому розділі буде наведено доведення ймовірнісної повноти для для алгоритму RT-RRT* та його модифікації і показано, що швидкість збіжності модифікації більша ніж швидкість оригінального алгоритму.

Нехай χ — середовище, в якому відбувається вирішення проблеми пошуку шляху. Без обмежень загальності, будемо вважати $\chi = [0,1]^d$ — d-вимірний гіперкуб зі стандартною Евклідовою метрикою. Простір без перешкод позначається як $\chi_{free} \subseteq \chi$, а через $|\chi_{free}|$ позначається його міра Лебега. $\mathcal{B}_r(x)$ позначає кулю радіуса r з центром в точці $x \in \mathbb{R}^d$. $x_{init} \in \chi_{free}$ — початкова позиція, а $\chi_{goal} \subset \chi_{free}$ — цільова область, де $\chi_{goal} = \mathcal{B}_{\delta_{goal}}(x_{goal})$.

На середовище χ накладаються умови *надійної здійсненності* 4.1.3, що були наведені раніше. З умов надійної здійсненності слідує, що існує такий шлях $\sigma : [0,t] \to \chi_{free}$, для якого виконуються умови *сильного* δ_{clear}*-зазору*. Та нехай $\sigma(t) = x_{goal}$, тобто шлях закінчується в центрі цільової області χ_{goal}. Через L — позначимо довжину шляху σ, та нехай $\delta = \min\{\delta_{clear}, \delta_{goal}\}$.

Покладемо $m = \frac{5L}{v}, v = \min(\delta, r_s)$, де r_s 2 — задає максимальну дозволену відстань між вершинами в дереві.

Задамо послідовність $m+1$ точок $x_0 = x_{init}, \ldots, x_m = x_{goal}$ вздовж σ, так що відстань між кожними двома послідовними точками x_i, x_{i+1} дорівнює $\frac{v}{5}$. Далі визначимо множину $m+1$ шарів з радіусом $\frac{v}{5}$ з центром в цих точках та доведемо, що RT-RRT* з великою ймовірністю згенерує шлях, який проходить через ці шари.

Так як на середовище накладена умова *надійної здійсненності* і враховуючи те, що позиції перешкод між i-ю та $i+1$-ю ітераціями не можуть відрізнятися більше ніж на деяку константу Δ, в кожен момент часу i, знайдеться така множина точок $\{x_0, \ldots, x_{m+1}\}$ та шарів $\left\{\mathcal{B}_{\frac{v}{5}}(x_0), \ldots, \mathcal{B}_{\frac{v}{5}}(x_{m+1})\right\}$, що будуть прокладати шлях між x_{init} та x_{goal}.

Для початку доведемо Лемму 4.4.1, що використовується для доведення

Теореми 4.4.1 та задає умову для вдалого розширення дерева до цільової точки.

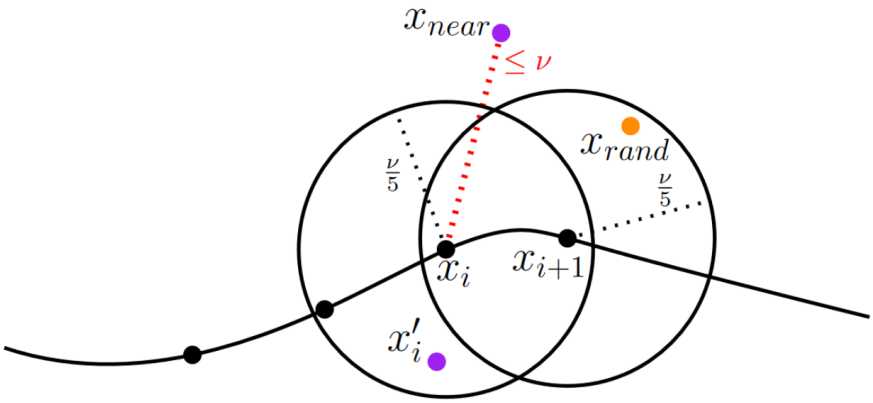

Рисунок 4.3 – Візуалізація Лемми 4.4.1

Лемма 4.4.1 *[8] Нехай алгоритм RT-RRT* досяг* $\mathcal{B}_{\frac{v}{5}}(x_i)$*, тобто дерево* $\mathcal{T}$ *містить вершину* $x' \in \mathcal{B}_{\frac{v}{5}}(x_i)$*. Якщо нова згенерована точка* x_{rand} *знаходиться в* $\mathcal{B}_{\frac{v}{5}}(x_{i+1})$*, тоді відрізок прямої між* x_{rand} *та його найближчим сусідом* x_{near} *в дереві* $\mathcal{T}$ *повністю знаходиться в* χ_{free}*.*

Доведення Нехай $x_{near} \in \mathcal{V}$ — найближчий сусід до x_{rand} в дереві $\mathcal{T}$. Тоді з означення x_{near} виконується $\|x_{near} - x_{rand}\| \leq \|x'_i - x_{rand}\|, \forall x'_i \in \mathcal{V}$.

Покажемо, що x_{near} повинен лежати в $\mathcal{B}_v(x_i)$, з чого слідує $line(x_{near}, x_{rand}) \subset \chi_{free}$, так як $x_{rand} \in \mathcal{B}_{\frac{v}{5}}(x_{i+1}) \subset \mathcal{B}_v(x_i)$. З нерівності $\|x_{near} - x_{rand}\| \leq \|x'_i - x_{rand}\|, \forall x'_i \in \mathcal{V}$ та з нерівності трикутника маємо, що:

$$\|x_{near} - x_i\| \leq \|x_{near} - x_{rand}\| + \|x_{rand} - x_i\| \leq \|x'_i - x_{rand}\| + \|x_{rand} - x_i\|$$

З нерівності трикутника маємо:

$$\|x_{rand} - x_i\| \leq \|x_{rand} - x_{i+1}\| + \|x_{i+1} - x_i\|,$$

$$\|x'_i - x_{rand}\| \leq \|x'_i - x_i\| + \|x_i - x_{i+1}\| + \|x_{i+1} - x_{rand}\|.$$

А отже:

$$\begin{aligned}\|x_{near} - x_i\| &\leq \|x'_i - x_i\| + 2 \cdot \|x_{i+1} - x_{rand}\| + 2 \cdot \|x_{i+1} - x_i\| \\ &\leq 5 \cdot \frac{v}{5} = v.\end{aligned}$$

Таким чином $x_{near} \in \mathcal{B}_v(x_i) \subseteq \chi_{free}$ і $line(x_{near}, x_{rand}) \subset \chi_{free}$
Необхідно зауважити, що $\|x_{near} - x_{rand}\| \leq r_s$, так як:

$$\begin{aligned}\|x_{rand} - x_{near}\| &\leq \|x_{rand} - x'_i\| \\ &\leq \|x'_i - x_i\| + \|x_i - x_{i+1}\| + \|x_{i+1} - x_{rand}\| \\ &\leq 3 \cdot \frac{v}{5} < v \leq r_s.\end{aligned}$$

Те що $\|x_{near} - x_{rand}\| \leq r_s$ означає, що $x_{new} = x_{rand}$. □

Тепер наведемо доведення для основної теореми.

Теорема 4.4.1 *[8] Ймовірність того, що алгоритм RT-RRT* (або його модифікація) не зможе досягти χ_{goal} з початкової позиції x_{init} після k ітерацій дорівнює щонайбільше $a \cdot e^{-b \cdot k}$, для деяких констант $a, b \in \mathbb{R}_{>0}$.*

Доведення Нехай $\mathcal{B}_{\frac{v}{5}}(x_i)$ вже містить вершину дерева $\mathcal{T}$. Позначимо через p — ймовірність того, що в наступній ітерації алгоритму нова вершина буде додана до $\mathcal{T}$, що буде міститися в $\mathcal{B}_{\frac{v}{5}}(x_{i+1})$. Згідно Леми 4.4.1, $x_{rand} \in \mathcal{B}_{\frac{v}{5}}(x_{i+1})$ забезпечує те, що алгоритм досягне $\mathcal{B}_{\frac{v}{5}}(x_{i+1})$.

Так як на кожній ітерації i алгоритму x_{rand} генерується рівномірно з $[0,1]^d$, ймовірність того, що $x_{rand} \in \mathcal{B}_{\frac{v}{5}}(x_{i+1})$ дорівнює $\frac{\left|\mathcal{B}_{\frac{v}{5}}\right|}{|[0,1]^d|} = \left|\mathcal{B}_{\frac{v}{5}}\right|$.

Для того, аби алгоритм досягнув χ_{goal} з x_{init}, необхідно повторити цей крок m разів з x_i до $x_{i+1}, 0 \leq i < m$. Цей стохастичний процес може бути визначений, як ланцюг Маркова (Рисунок 4.4), або як k послідовних випробувань з ймовірністю успіху p.

Проблема планування може бути вирішена через m успішних випробувань, де i-та успішна спроба додає вершину $x \in \mathcal{B}_{\frac{v}{5}}(x_i)$ до $\mathcal{T}$. Процес може завершитися і раніше ніж після m успішних випробувань, де m в такому випадку задає верхню границю ймовірності невдачі.

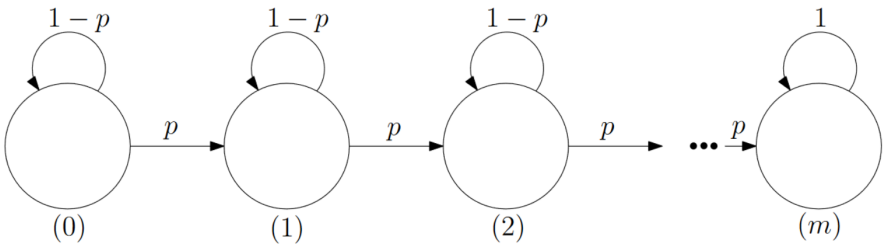

Рисунок 4.4 – Ланцюг Маркова, де ймовірність успіху $p = \left|\mathcal{B}_{\frac{v}{5}}\right|$ — ймовірність рівномірно згенерувати точку всередині кулі радіусу $\frac{v}{5}$. Стан m — є кінцевим станом, а m успішних випробувань означають, що алгоритм знайшов шлях від початкового стану до цільового.

Нехай X_k — кількість успіхів після k випробувань, тоді:

$$\Pr[X_k < m] = \sum_{i=0}^{m-1} C_k^i \cdot p^i \cdot (1-p)^{k-i} \tag{4.1}$$

$$\leq \sum_{i=0}^{m-1} C_k^{m-1} \cdot p^i \cdot (1-p)^{k-i} \tag{4.2}$$

$$\leq C_k^{m-1} \sum_{i=0}^{m-1} (1-p)^k \tag{4.3}$$

$$\leq C_k^{m-1} \sum_{i=0}^{m-1} \left(e^{-p}\right)^k \tag{4.4}$$

$$= C_k^{m-1} \cdot m e^{-pk} \tag{4.5}$$

Де (4.2) виконується через те, що $m \ll k$, (4.3) використовує те, що $p < \frac{1}{2}$, а (4.4) полягається на $(1-p) \leq e^{-p}$. Так як p, m не залежать від k, вираз $C_k^{m-1} \cdot m e^{-pk}$ прямує до 0 з експоненційною швидкістю.

Отже RT-RRT* є ймовірнісно повним. □

Для модифікації алгоритму RT-RRT* головна різниця в доведенні полягає в значенні ймовірності p того, що в наступній ітерації алгоритму нова вершина буде додана до $\mathcal{T}$, що буде міститися в $\mathcal{B}_{\frac{v}{5}}(x_{i+1})$.

Нагадаємо, що в модифікації нова точка генерується рівномірно не з усього середовища $\chi = [0,1]^d$, а тільки з області видимості $c_{vis} \subseteq \chi$ 3.1.3, де

рівність досягається тоді, коли $\chi_{free} = \chi$, тобто в середовищі немає перешкод.

Таким чином, ймовірність для кожної з m куль дорівнює $p_i = \frac{\left|\mathcal{B}_{\frac{v}{5}}\right|}{\left|c^i_{vis}\right|}$, де c^i_{vis} — область видимості, побудована на i-му кроці. А так як $c^i_{vis} \subseteq \chi_{free} \subseteq \chi$, то з цього слідує, що $p_i \geq p, \forall i \in \{0, \ldots, m\}$. А отже $C_k^{m-1} \sum_{i=0}^{m-1} \left(e^{-p_i}\right)^k$ збігається до 0 швидше ніж $C_k^{m-1} \sum_{i=0}^{m-1} \left(e^{-p}\right)^k$.

Отже як оригінальний алгоритм RT-RRT* так і його модифікація є ймовірнісно повними, проте модифікація алгоритму має більшу швидкість збіжності.

4.5 Аналіз практичних результатів

В даному розділі буде наведено порівняння практичних результатів відносно кількості ітерації необхідних для знаходження рішення проблеми планування шляху.

Для порівняння кількості ітерацій оригінального алгоритму RT-RRT* та його модифікації було обрано декілька різних середовищ, як з рухомими перешкодами, так і повністю статичні. Для кожного середовища, проблема планування шляху вирішувалась 20 разів з однакових початкових конфігурацій та за остаточний результат для даного середовища бралося середнє арифметичне кількості ітерацій.

Для визуалізації кожного середовища червона точка — початкова позиція агента, чорний контур — цільова область, чорні прямокутники — перешкоди.

4.5.1 Динамічне середовище

В Таблиці 4.1 наведено порівняння середньої кількості ітерацій необхідних оригінальному алгоритму RT-RRT* та його модифікації отриманої шляхом описаним на початку розділу.

Таблиця 4.1 – Таблиця порівняння кількості ітерацій алгоритмів.

Середовище	**Оригінальний**	**Модифікація**
1	541	398

На Рисунку 4.5 наведено візуалізацію динамічного середовища, що використовувалось при порівнянні.

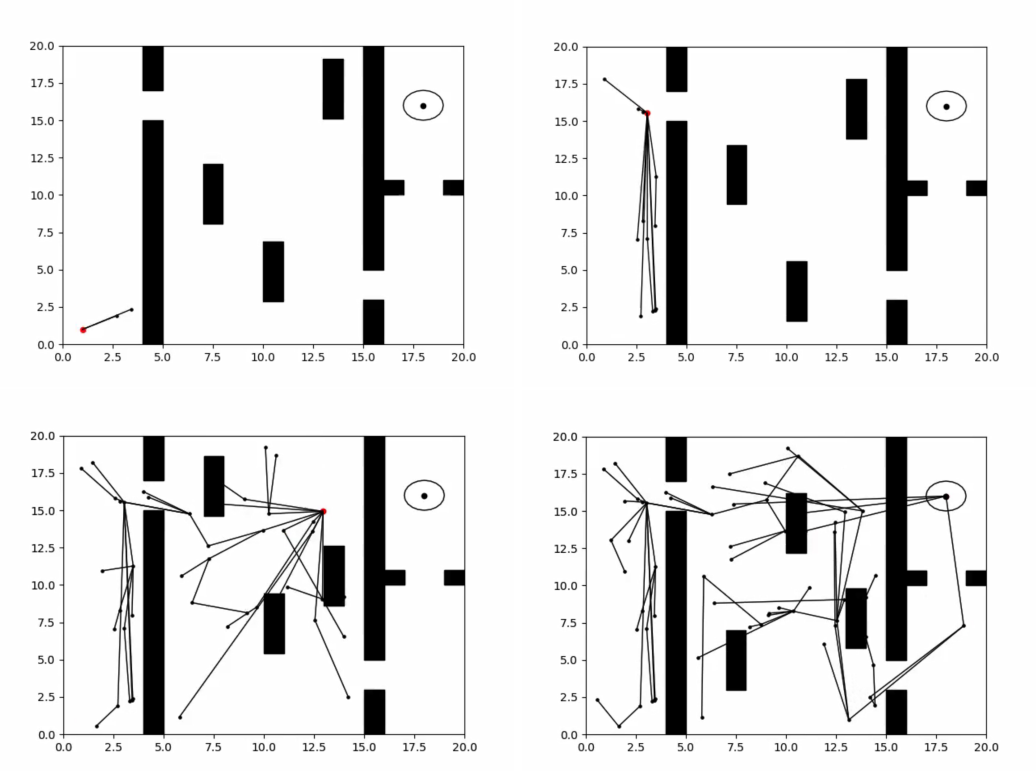

Рисунок 4.5 – Динамічне середовище

4.5.2 Статичне середовище

В Таблиці 4.2 наведено порівняння середньої кількості ітерацій необхідних оригінальному алгоритму RT-RRT* та його модифікації отриманої шляхом описаним на початку розділу.

Таблиця 4.2 – Таблиця порівняння кількості ітерацій алгоритмів.

Середовище	**Оригінальний**	**Модифікація**
1	414	213
2	683	327

На Рисунку 4.6 наведено візуалізацію статичних середовищ, що використовувались при порівнянні.

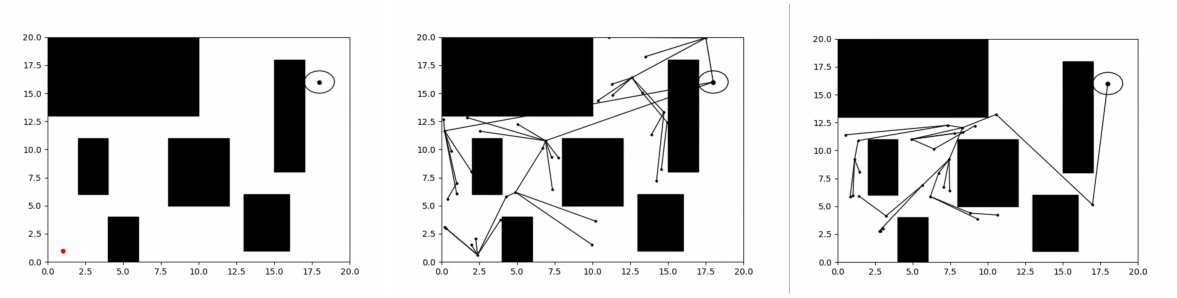

Рисунок 4.6 – Статичне середовище 1. Ліве зображення — початкова конфігурація, середнє — результат оригінального алгоритму, праве — результат модифікації.

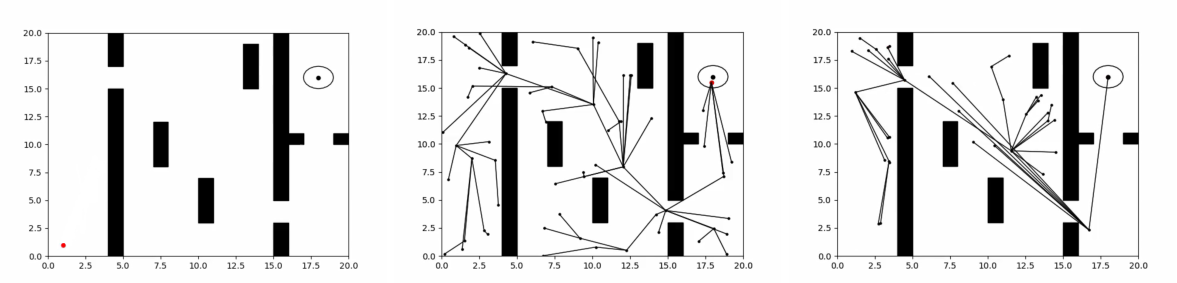

Рисунок 4.7 – Статичне середовище 2. Ліве зображення — початкова конфігурація, середнє — результат оригінального алгоритму, праве — результат модифікації.

4.6 Висновки до Розділу 4

В даному було наведено означення *ймовірнісної* та *асимптотичної* повноти алгоритму. Наведено означення *надійної здійсненності* проблеми планування шляху, виконання якої накладено на проблеми планування, що розглядаються в цій роботі.

Порівняно складність окремих операцій оригінального алгоритму RT-RRT* та його модифікації, наведеної в цій роботі.

Наведено доведення ймовірнісної повности алгоритмів, та для модифікації показано, що швидкість збіжності є більшою, ніж для оригінального алгоритму.

ВИСНОВКИ

В результаті виконання роботи було досліджено існуючі алгоритми пошуку шляху в режимі реального часу. Серед досліджених алгоритмів за основу було обрано алгоритм на основі випадкових дерев, тобто таких дерев, що будуються шляхом випадкової генерації їх вершин.

Для обраного алгоритму Real-Time Rapidly-Exploring Random Trees запропонованого Naderi et.al., запропоновано модифікацію, що замінює алгоритм розширення дерева $\mathcal{T}$ та алгоритм індексації дерева.

Новий алгоритм розширення дерева базується на побудові "області видимості"навколо вершин дерева та генерації точок всередині цієї області. Це забезпечує те, що будь-яка згенерована точка всередині області видимості може бути гарантовано приєднана принаймі до тих вершин, навколо яких будувалась область видимості, що пришвидшує збіжність алгоритму до розв'язку.

Модифікація алгоритму індексації дерева пропонує замінити grid-based індексацію, на використання KD-дерев для пошуку серед вершин $\mathcal{V}$.

Порівняно складність оригінального алгоритму так його модифікації, запропонованої в цій роботі та наведено доведення ймовірнісної повноти, що виконується для обох алгоритмів. Для модифікації алгоритму показано, що швидкість збіжності алгоритму до рішення задачі планування шляху є більшою ніж для оригінального алгоритму, відносно кількості ітерацій, завдяки запропонованим модифікаціям.

ПЕРЕЛІК ПОСИЛАНЬ

[1] N. Nilsson, "Principles of artificial intelligence." https://archive.org/details/principlesofarti00nils, 1980.

[2] Y. H. et.al., "A potential field approach to path planning." http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.389.4041&rep=rep1&type=pdf, FEBRUARY 1992.

[3] J. K. Steven LaValle, "Rapidly-exploring random trees: Progress and prospects." http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01.pdf.

[4] K. N. et.al., "Rt-rrt*: A real-time path planning algorithm based on rrt*." http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01.pdf.

[5] "Ray casting." https://en.wikipedia.org/wiki/Ray_casting.

[6] "K-d tree." https://en.wikipedia.org/wiki/K-d_tree.

[7] LaValle and Kuffner, "Randomized kinodynamic planning." http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01b.pdf, 2001.

[8] K. et.al., "Probabilistic completeness of rrt for geometric and kinodynamic planning with forward propagation." https://arxiv.org/pdf/1809.07051.pdf.

[9] Karaman and Frazzoli, "Sampling-based algorithms for optimal motion planning." https://core.ac.uk/download/pdf/18173530.pdf, 2011.

[10] D. M. Ming C. Lin and Y. J. Kim, "Collision and proximity queries." https://www.csun.edu/~ctoth/Handbook/chap39.pdf.

[11] W. D. Six H.W., "Counting and reporting intersections of d-ranges." https://www.infona.pl/resource/bwmeta1.element.ieee-art-000001675973.

[12] "Quadtree." https://en.wikipedia.org/wiki/Quadtree.

[13] "Spatial hashing." https://www.gamedev.net/articles/programming/general-and-gameplay-programming/spatial-hashing-r2697/.

[14] A. Schrijver., *Combinatorial Optimization, volume A.* Springer, 2003.