# Лекція 9. РТтаМТ

Тема**: Автоматизоване планування оптимальних траєкторій переміщення мобільних мехатронних пристроїв**

## План

**9.1. Загальні положення.**

**9.2. Стисло щодо поширених алгоритмів безперешкодних (безколізійних) переміщень ММП.**

**9.3. Інформаційне забезпечення та формалізовані описи критеріїв вибору оптимальної траєкторії переміщення ММП.**

**9.4. Стуктурна схема розробленого програмного продукту “LSTr“ та приклади його працездатності.**

## 9.1. Загальні положення

На сьогодні автоматизація виробництва є одним з головних напрямів науково-технічного прогресу, головною метою якої є підвищення ефективності виробництва шляхом впровадження нових технологій, якості та надійності виготовлення продукції та створення умов для оптимального використання всіх виробничих ресурсів.

В зв’язку з цим особливу увагу в умовах виробництва приділяють використанню промислових роботів (ПР), що дозволяють забезпечити повний цикл технологічної дії схватів (Сх) ПР на об’єкт маніпулювання (ОМ) з високою точністю та продуктивністю.

На даному етапі розвитку робототехніки важливе місце посідають задачі оптимізації різних показників (геометричних, кінематичних, динамічних, траєкторних, економічних тощо), які є необхідним при проектуванні / синтезі роботизованих механоскладальних технологій (РМСТ), що реалізуються в гнучких виробничих комірках (ГВК) та в інших технологічних структурах гнучкого виробництва. Важливе місце серед них посідають траєкторні задачі, розв’язування яких неможливе без планування рухів ПР в ГВК та вибір оптимальних з них за попередньо обраним критерієм / критеріями.

Для реалізації планувальних задач система управління ПР повинна бути здатною будувати квазіоптимальну траєкторію, в ідеалі - оптимальну за попередньо прийнятим критерієм та вільну від зіткнень (колізій) з іншими елементами ГВК (так звану безколізійну траєкторію).

Побудова безперешкодного, близького до оптимального шляху необхідна для мінімізації часових та енергетичних затрат за рахунок бажаного зменшення відстані та збільшення швидкості переміщень ланок маніпуляційної системи (МС) та Сх ПР.

Задача пошуку найкоротшого шляху, який оминає перешкоди, в заданому просторі станів розглядається досить давно і включає багато алгоритмів їх вирішення.

Саме тому для ефективного розв'язування цієї задачі розглядається мобільний мехатронний пристрій (ММП) як деякий аналог Сх ПР, що переміщується чітко визначеними опорними точками з ОМ або без нього. І, знаючи закони переміщення саме цього кінцевого елемента, можна говорити про те, що в кінцевому етапі ці алгоритми і отримані дані можуть бути використані для подальшого розв'язку зворотних задач кінематики, тобто за конкретним положенням Сх ПР з/без ОМ, що є технологічним роботизованим комплектом (ТРК), визначати положення, яке займає вся МС ПР, щоб не тільки на рівні кінцевого елемента визначати перетини і колізії, але й на рівні всіх ланок МС ПР.

## 9.2. Стисло щодо поширених алгоритмів безперешкодних (безколізійних) переміщень ММП

На сьогодні існує ряд алгоритмів планування безперешкодних траєкторій, найпоширенішими з яких є наступні 9 алгоритмів (перелік далеко не повний), що умовно позначені як **$A_1$**, …, **$A_9$:**

- **A1** - KPIECE (**K**inodynamic Motion **P**lanning by **I**nterior-**E**xterior **C**ell **E**xploration);
- **A2** - Bi-directional KPIECE;
- **A3** - Lazy Bi-directional KPIECE;
- **A4** - PRM (probabilistic roadmap method);
- **A5** - SBL (A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking),
- **A6** - RRT (Rapidly exploring random tree);
- **A7** - RRT Connect;
- **A8** - Lazy RRT;
- **A9** - EST (Expansive Space Trees).

**A1** - KPIECE (**K**inodynamic Motion **P**lanning by **I**nterior-**E**xterior **C**ell **E**xploration) (Рис. 9.1).

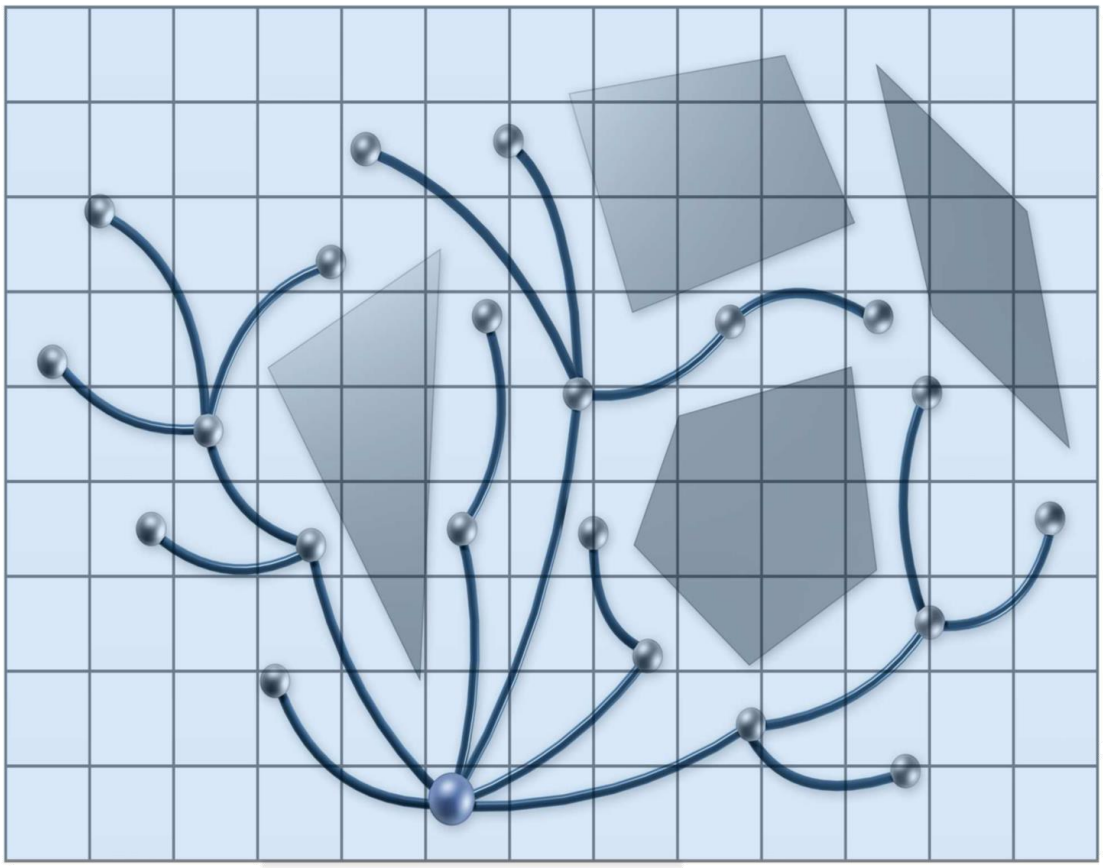


*Рис.9.1. Спрощен схема роботи алгоритму A1 - KPIECE*

Розшифровується як кінематично–динамічне планування переміщень шляхом дослідження внутрішніх і зовнішніх комірок. Це спеціально створений алгоритм зі складною динамікою. Дискретизація, яка базується на багаторівневій сітці, дозволяє оцінити покриття загального простору.

Вона складається з $k$ рівнів $L_1,\ldots,L_k$,. Кожен з цих рівнів являє собою сітку, де комірки є прототипами фіксованого розміру. KPIECE ітеративно будує дерево переміщень в загальній області робота. Кожне переміщення $м = (s; u; t)$ визначається станом s $\in Q$, керуючим впливом $u \in U$ і часовим проміжком $t$. Керуючий вплив u застосовується протягом часового проміжку $t$ із стану $s$ та ініціює переміщення.

*Постановка задачі* при роботі цього алгоритму зводиться до наступного.

Нехай:

$Q$ – загальний простір;

$U$ – простір керуючих впливів;

$I \subset Q$ – набір початкових станів;

$F \subset Q$ – набір кінцевих станів.

Динаміка описується розповсюдженням у прямому порядку

$f : Q\ X\ U \rightarrow TgQ,$

де $TgQ$ – це дотичний простір станів $Q$.

Необхідно зауважити, що перші чотири елементи визначають структуру системи, а останній елемент $f$ – динаміку поведінки системи.

Вирішення проблеми планування руху складається з послідовності керуючих впливів, що належать простору керуючих впливів $u_1,...,u_n \in U$, та часових проміжків $t_1,\ldots,t_n$, таких що $q_0 \in I$, $q_n \in F$ та $q_k$; $k = 1,..., n$, може бути отримана послідовно при інтеграції $f$.

*Основна ідея* алгоритму є наступною.

KPIECE ітеративно будує дерево переміщень в загальній області ПР. Кожне переміщення $м = (s; u; t)$ визначається станом $s \in Q$, керуючим впливом $u \in U$ і часовим проміжком $t$.

Керуючий вплив $u$ застосовується протягом часового проміжку $t$ із стану $s$ та ініціює переміщення. Можливе розбиття переміщення $м = (s; u; t)$ на $м1 = (s; u; t_\alpha)$ та $м2 = (\int_{t_0}^{t_0+t_a} f\ (s(\tau), u)\ d\tau, u, t_b)$, де $s(\tau)$ визначає час $\tau$ та $t_a + t_b = t$.

В процесі дослідження важливим є покриття якомога більшої частини загального простору за найменший час. Для досягнення цієї мети потрібні очікування покриття загального простору $Q$. Коли менш покрита ділянка із загального простору станів знайдена, дерево переміщень розповсюджується на цю ділянку. Цей процес виконується ітеративно поки не настане умова зупинки.

*Дискретизація* при роботі алгоритму KPIECE полягає в тому, що під час виконання процесу планування переміщень планувальник вирішує які області загального простору вимагають подальшого дослідження.

З ростом дерева можливих переміщень процес прийняття рішень стає складнішим. Існують різні способи вирішення цієї проблеми.

Одним з можливих підходів є побудова дискретизації, що дозволяє оцінювати покриття загального простору станів.

Ця дискретизація складається з $k$ рівнів $L_1,..., L_k$, як показано на рис.9.2. Кожен з цих рівнів являє собою сітку, де комірки є прототипами фіксованого розміру. Кількість рівнів та розміри комірки заздалегідь відомі. Проте комірки заповнюються значеннями тільки тоді, коли вони потрібні. Призначенням цих сіток є покриття частини загального простору, яка відповідає за площу, на яку розповсюджене дерево переміщень. Кожен з рівнів $L$ забезпечує різну роздільну здатність для оцінювання покриття. Сітка найнижчої роздільної здатності може

бути використана початково для наближеного виявлення найменш досліджених областей. На цих областях може бути використана наступна сітка з більш високою роздільною здатністю, щоб більш точно виявити найменш покриті області.

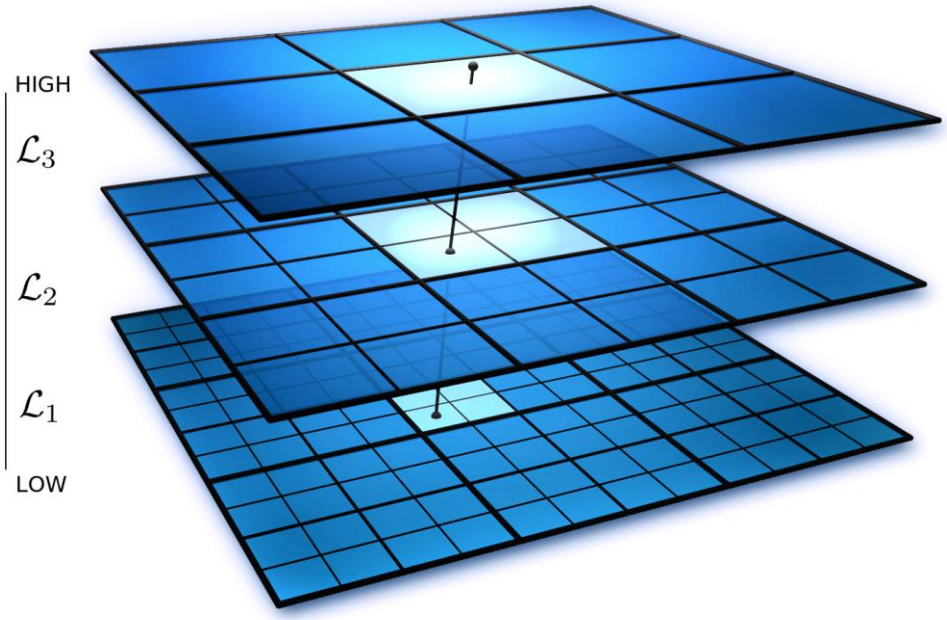



*Рис.9.2. Приклад 3-рівневої дискретизації за алгоритмом KPIECE*

Дерево переміщень існує в просторі станів *Q*, але оскільки розміри цього простору можуть буду дуже великими, використовується дискретизація, як проекція просторів стану *E(Q)*. Для будь-якого переміщення *м* кожен рівень дискретизації містить комірку, частиною якої є *м*. Переміщення *м* прийнято вважати частиною комірки *p*, якщо існує стан s вздовж переміщення μ таке, що проекція *E(s)* знаходиться всередині комірки *p*. Якщо переміщення займає більше однієї комірки одного і того самого рівня дискретизації, воно розбивається на дрібніші переміщення, так що жодне з переміщень не перетинає меж комірки.

Для кожного переміщення *м* існуватиме лише одна комірка на кожному рівні дискретизації, частиною якої є *м*. Цей набір комірок формує кортеж $c = < p_1, \ldots, p_k >$, $p_i \subset p_{i+1}$; $p_i \in L_i$ , який матиме назву ланцюжок комірок для *м*. Оскільки комірки в $L_1$ будуть визначати чи є переміщення розбитим, наведемо визначання дискретизації.

$\forall\, p \in L_1$, $M_p$ = {*min*|*min* довжини переміщень, що належать $p$}, де $M_p$ – це довжина всіх переміщень, що належать комірці *p* на рівні дискретизації $L_1$.

Для всіх $p \in L_i$ вважаємо *p* таким, що містить $M_p$ і для всіх $p \in L_i$ , *i* >1, *p* містить $D_p$. Оскільки дискретизація потенційно покриває дуже велику проєкцію станів *E(Q)*, комірки ініціалізуються лише тоді, коли переміщення, що являється

їх частиною, знайдено. Це дозволяє планувальнику переміщень значно зменшити об’єм використовуваної пам’яті.

Відміною рисою алгоритму KPIECE є визначення внутрішньої та зовнішньої комірки.

Комірка вважається зовнішньою, якщо вона має менше, ніж *2n* ініціалізовані сусідні клітинки (діагональні сусідні клітинки не враховуються) на тому самому рівні дискретизації, де *n* є розмірністю *E(Q)*. Комірка з *2n* ініціалізованих сусідніх комірок вважаються внутрішньою (при цьому не може бути більше *2n* недіагональних сусідніх комірок у *n* вимірному просторі).

В процесі роботи алгоритму KPIECE ініціалізуються нові комірки і деякі зовнішні комірки стають внутрішніми. Коли більша частина просторів станів досліджена, більшість комірок буде внутрішніми. Проте для багатомірних просторів для уникнення ситуацій, коли всі комірки будуть зовнішніми, визначення внутрішньою комірки може бути спрощене і комірка буде вважатися внутрішньою до того, як *2n* сусідніх комірок ініціалізуються.

## A2 - Алгоритм Bi-directional KPIECE

Розшифровується як двонаправлене кінематично-динамічне планування переміщень шляхом дослідження внутрішніх і зовнішніх комірок. Головна ідея роботи така сама, як і в алгоритмі KPIECE.

Відмінність Bi-directional KPIECE від KPIECE в тому, що Bi-directional KPIECE є ускладнений алгоритм пошуку, в основі якого лежить така ідея, що одночасно будується два дерева $T_{init}$ та $T_{goal}$ (в прямому напрямку, від початкового стану $q_{init}$, та у зворотному напрямку, від цілі $q_{goal}$), зупиняючись після того, як два дерева з’єднаються.

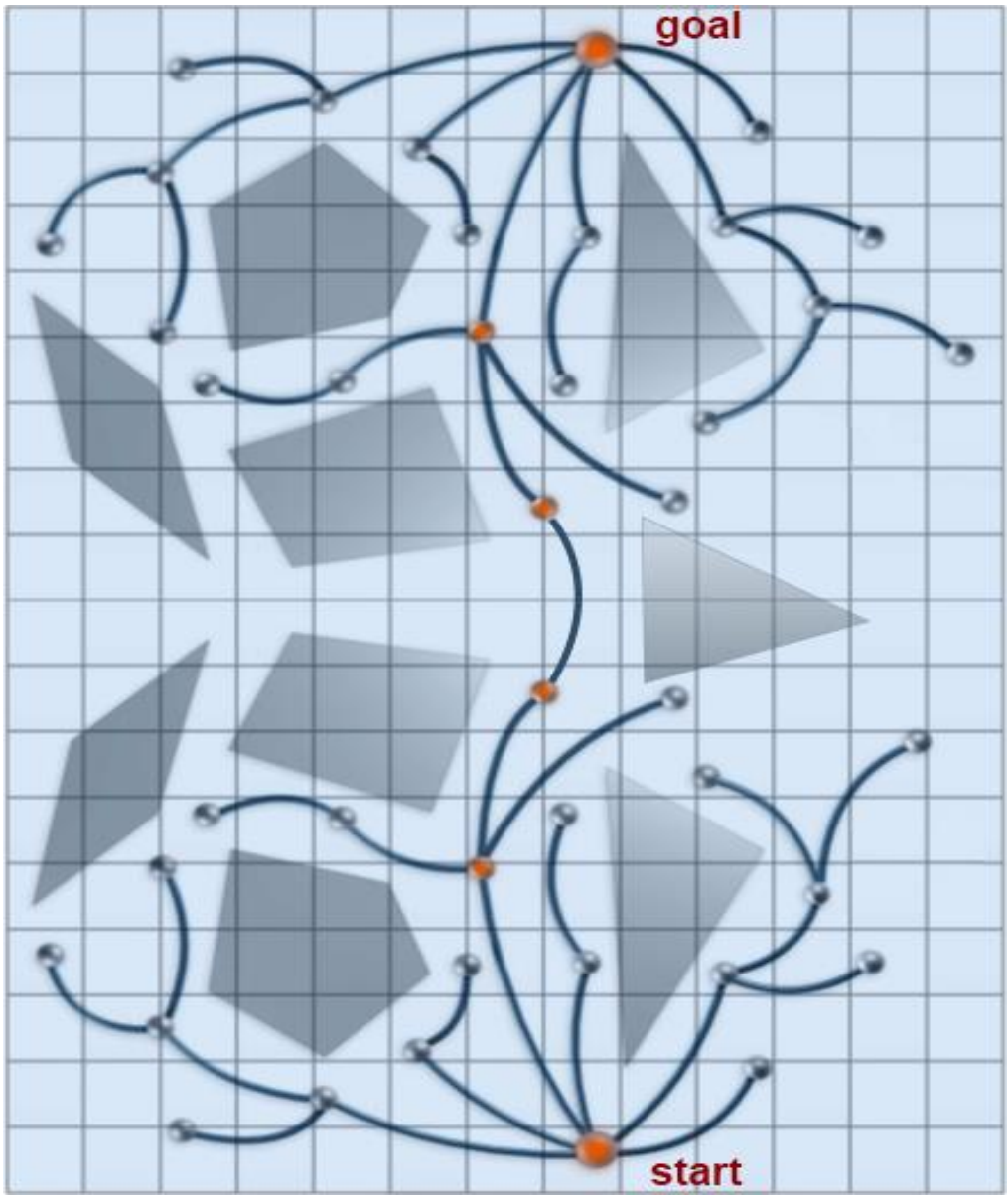
Візуальне представлення роботи алгоритму Bi-directional KPIECE представлено на рис. 9.3.

*Переваги* алгоритму Bi-directional KPIECE:

- більш швидко будує траєкторію, що не містить перешкод;
- потребує менше обчислювальних ресурсів порівняно з іншими алгоритмами.

*Недоліки* алгоритму Bi-directional KPIECE:

- в умовах обмеженого часу алгоритм може не встигнути знайти рішення;
- знайдена траєкторія може бути не найкращою із можливих;
- при однакових вхідних параметрах алгоритм може кожного разу знаходити різні рішення.




- *Рис.9.3. Графічне представлення роботи алгоритму*

- *Bi-directional KPIECE*

**А3 - Алгоритм Lazy Bi-directional KPIECE**

Розшифровується як двонаправлене кінематично-динамічне планування переміщень шляхом дослідження внутрішніх і зовнішніх комірок з мінімальною перевіркою на колізії.

Головна ідея роботи така сама, як і в алгоритмі Bi-directional KPIECE. Відрізняється тим, що він не перевіряє шлях на наявність колізій на кожній з ітерацій. Натомість Lazy Bi-directional KPIECE намагається якнайшвидше знайти потрібну траєкторію, що з'єднувала б $q_{init}$ та $q_{goal}$ і лише після цього знайдена траєкторія перевіряється на колізії. У випадку наявності колізії, сегмент, що містить її, відкидається та процес пошуку продовжується.

*Переваги* алгоритму Lazy Bi-directional KPIECE :

- відкладання перевірки наявності колізій на останній момент, коли траєкторія вже знайдена, значно підвищує продуктивність роботи алгоритму ;
- алгоритм потребує менше обчислювальних ресурсів порівняно з іншими алгоритмами.

*Недоліки* алгоритму Lazy Bi-directional KPIECE:

- в умовах обмеженого часу алгоритм може не встигнути знайти рішення;
- більш складніший у реалізації;
- знайдена траєкторія може бути не найкращою із можливих;
- при однакових вхідних параметрах алгоритм може кожного разу знаходити різні рішення.

## А4 - Алгоритм PRM (probabilistic roadmap method)

Розшифровується як *імовірнісний метод пошуку маршруту*.

Складається з двох фаз.

*Перша фаза* (стило) – фаза побудови:

1) задається початкова і кінцева точки, простір стану та параметри перешкод;

2) у просторі, вільному від перешкод, створюється довільні конфігурації;

3) використовуючи локальний планувальник, PRM алгоритм з'єднує сусідні, типово будь-які k найближчі сусідні конфігурації. Конфігурації та з'єднання додаються до графу.

*Друга фаза* (стисло) – фаза з'ясування:

1) початкове і кінцеве положення додаються до графа;

2) за допомогою алгоритму Дейкстри будується шлях від початкової до кінцевої точки

*Головна ідея* роботи алгоритму PRM полягає в наступному. Як було зазначено раніше, алгоритм складається з двох фаз.

Перша фаза – фаза побудови, яка реалізується наступною послідовністю дій (рис.9.4. – 9.7):

1) задається початкова і кінцева точки, простір стану та параметри перешкод;

2) у просторі, вільному від перешкод створюється довільна конфігурація;

3) використовуючи локальний планувальник, PRM алгоритм з'єднує сусідні, типово будь-які *k* найближчі сусідні конфігурації. Конфігурації та з'єднання додаються до графу.

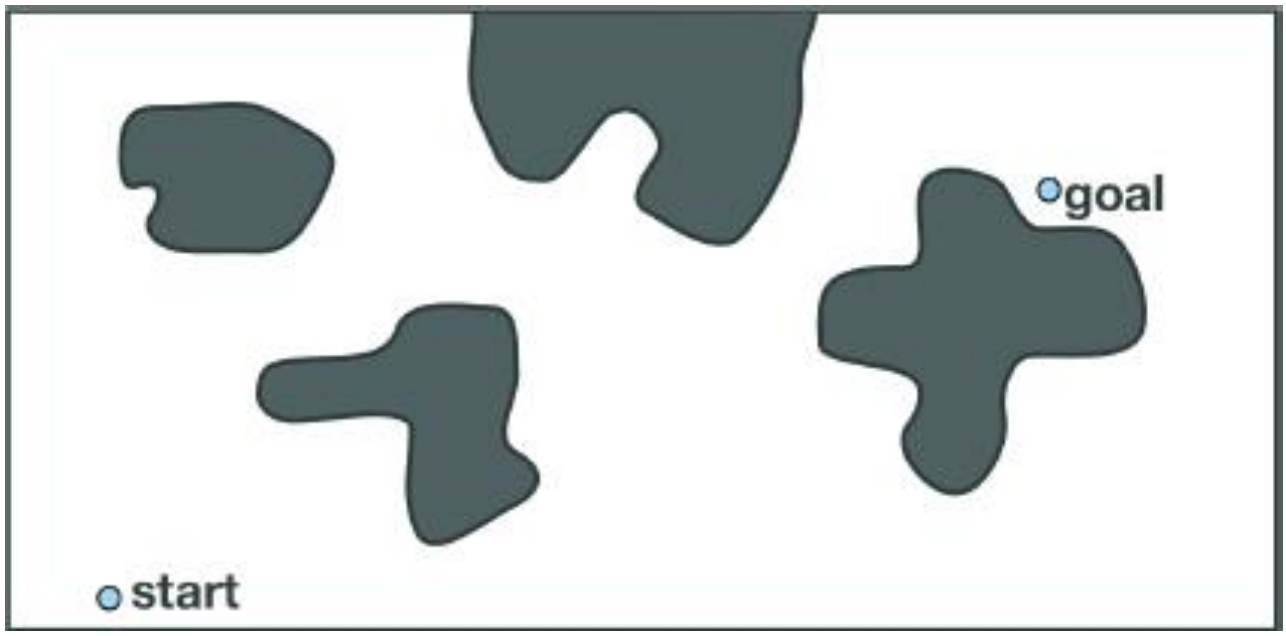



*Рис.9.4. Заданий простір станів та параметри перешкод*

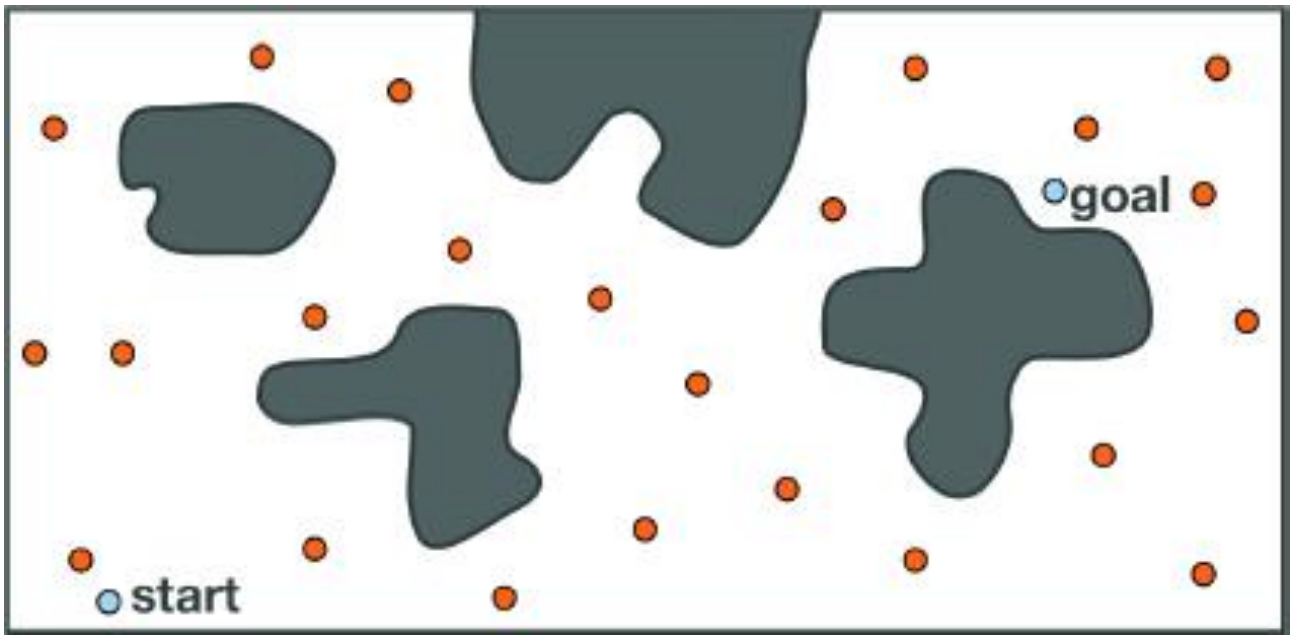



*Рис.9.5. Створення PRM алгоритмом довільних конфігурацій*

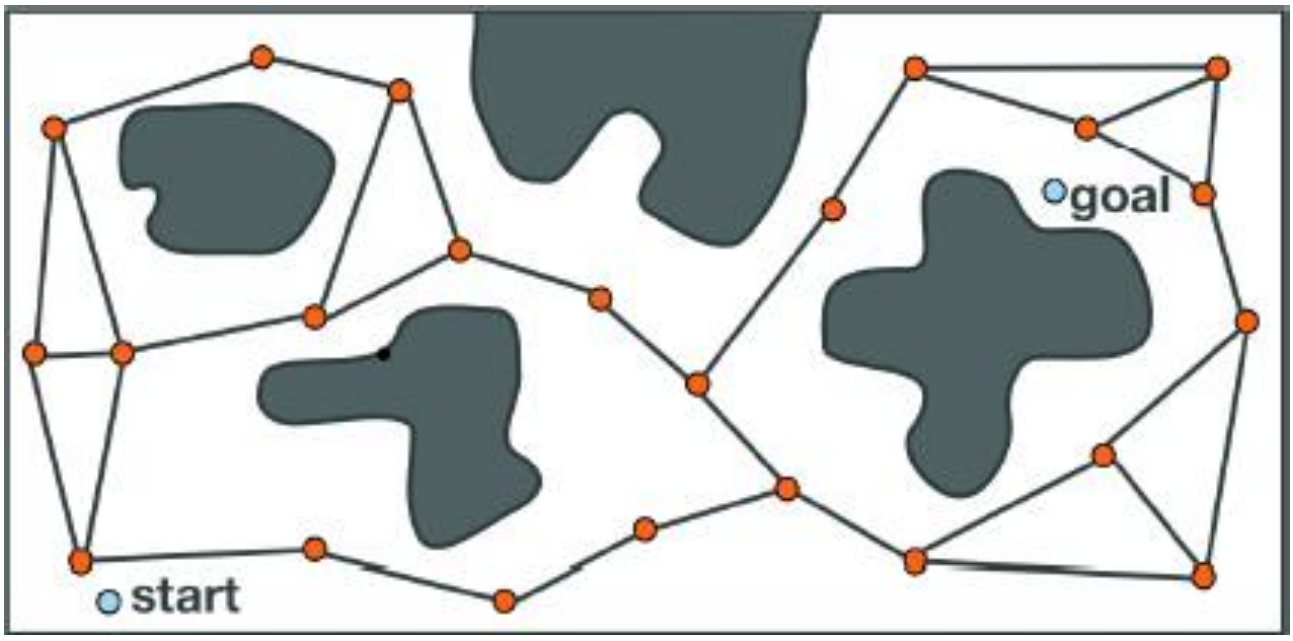



*Рис.9.6. Схема з'єднаних PRM алгоритмом сусідніх конфігурацій з формуванням графу*

Друга фаза – фаза з'ясування:

1) початкове і кінцеве положення додаються до графа;

2) за допомогою алгоритму Дейкстри будується шлях від початкової до кінцевої точки, як показано на рис. 9.7

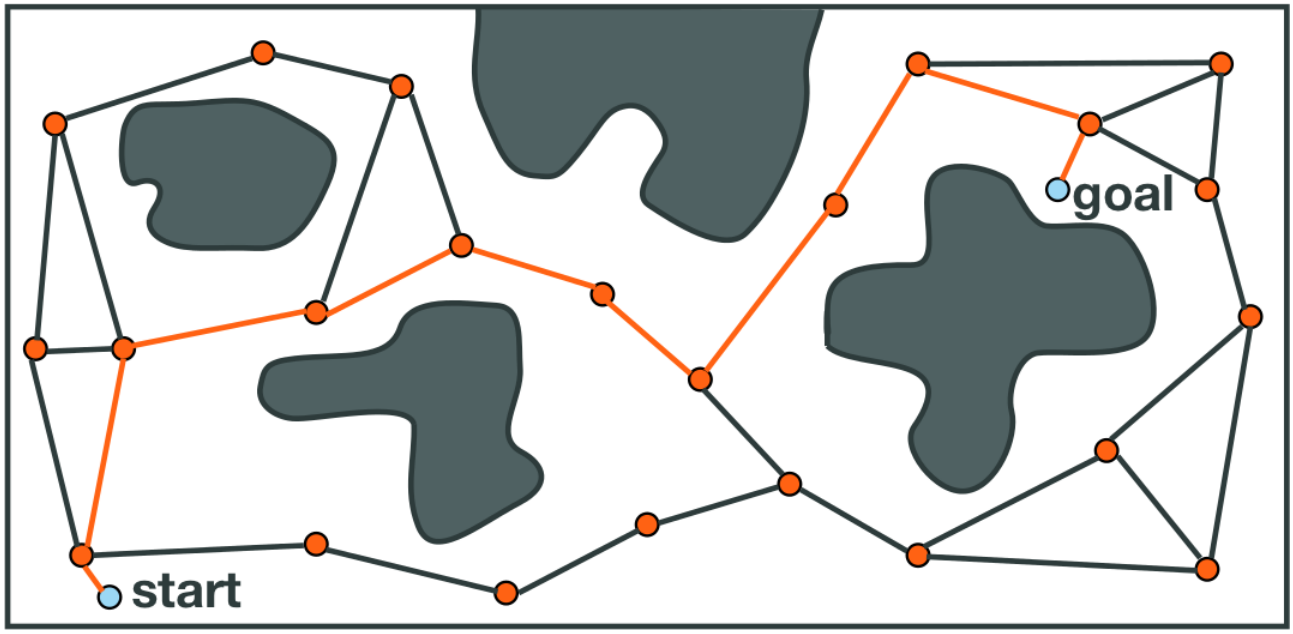



*Рис. 9.7. Приклад побудови траєкторії PRM алгоритмом. Початкова і кінцева точка з'єднані та найкоротший шлях в графі знайдений*

*Переваги* PRM алгоритму:

- простота реалізації.

*Недоліки* PRM алгоритму:

- напрям дослідження простору станів обирається довільно, що може призвести до марнування часу на дослідження хибних напрямків;
- потребує багато оперативної пам’яті та обчислювальної потужності;
- потребує багато часу на знаходження шляху.

## А 5 - Алгоритм SBL (A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking)

Розшифровується як *однозапитовий, двонаправлений ймовірнісний планувальник траєкторій з мінімальною кількість перевірок колізій*.

Алгоритм SBL входить до складу PRM алгоритму.

Він випадковим чином розбиває простір на множину точок, вільних від колізії, які називаються milestones – “контрольні точки”. Після чого з’єднує пари контрольних точок прямими відрізками і ті з них, що не містять колізій, стають локальними шляхами. Відмінною рисою алгоритму є те, що відсутність колізій між контрольними точками перевіряється лише при побудові кінцевого маршруту. По замовчуванню вважається, що всі точки та з’єднання між ними вільні від колізій. Якщо під час побудови найкоротшого шляху виникає колізія між деякими точками, то цей сегмент відкидається і починає будуватися новий шлях. Відкладання процесу перевірки колізій на більш пізній час є перспективним підходом, адже це допомагає значно скоротити кількість обрахунків та прискорити час виконання алгоритмів.

Алгоритм SBL поступово будує мережу із контрольних точок, які складають дерева, одне з яких будується з початкової точки, а інше – з кінцевої точки, назустріч одне одному. Алгоритм регулює довжину гілок дерева, роблячи ширші кроки у вільних частинах простору, та коротші – у вузьких місцях. Відрізки між контрольними точками перевіряється на наявність колізій не одразу, а лише після того, як два дерева, що будуються назустріч одне одному, з’єднаються. При цьому перевіряються лише ті відрізки, що є частиною потенційного маршруту. Таким чином, не витрачається час на перевірку відрізків, що лежать поза потенційним маршрутом. Порівняно зі схожими планувальниками, що використовують традиційну стратегію перевірки колізій, SBL показує в 4 – 40 разів кращий час розв’язання задачі в одному і тому ж середовищі.

*Визначення алгоритму SBL* є наступним.

Нехай $C$ визначає простір конфігурацій робота і $F \subseteq C$ є вільним простором. Нормалізуємо межі кожної з степенів свободи в межах $[0,1]$, тоді $C$ можна задати як $C\ [0,1]^n$, де $n$- є кількістю степенів рухомості робота. Визначимо $d$ в просторі $C$. Для будь- якого $q \in C$ сусідньою областю радіусу $r$ буде підмножина $B(q,r) = \{q' \in C \mid d\ (q, q') < r\}$. При $d=L_\infty$, метрика, що використовується SBL алгоритмом, являє собою $n - D$ куб.

Задача планування маршруту визначається двома конфігураціями $q_{init}$ та $q_{goal}$. Якщо обидві конфігурації лежать в єдиному просторі $F$, то планувальник повинен знайти вільний від колізій шлях між $q_{init}$ та $q_{goal}$. В іншому разі він повинен повідомити, що такого маршруту не існує.

Для роботи алгоритму SBL потрібно визначити два базових параметри:

$s$ – максимальна кількість контрольних точок, що може бути згенерована,

$p$ – поріг дальності.

Дві конфігурації вважаються близькими одна до одної, якщо їхнє $L_\infty$ є меншим за $p$. Зазвичай $p$ береться в межах 0,1 – 0,3.

SBL алгоритм складається із кінцевої множини складових (алгоритмів), а саме: PLANNER ($q_{init}$, $q_{goal}$), EXTAND – TREE, CONNECT – TREE, TEST – SEGMENT ($u$) та TAST – PATH ($\tau$).

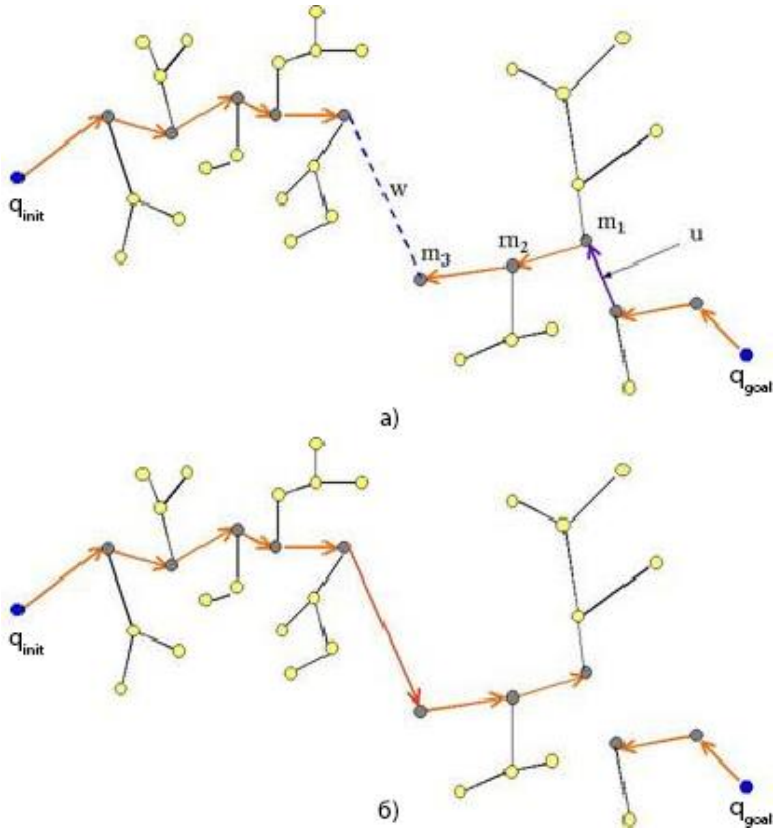



*Рис. 9.8. Процес перенесення точок з одного дерева в інше в алгоритмі SBL*

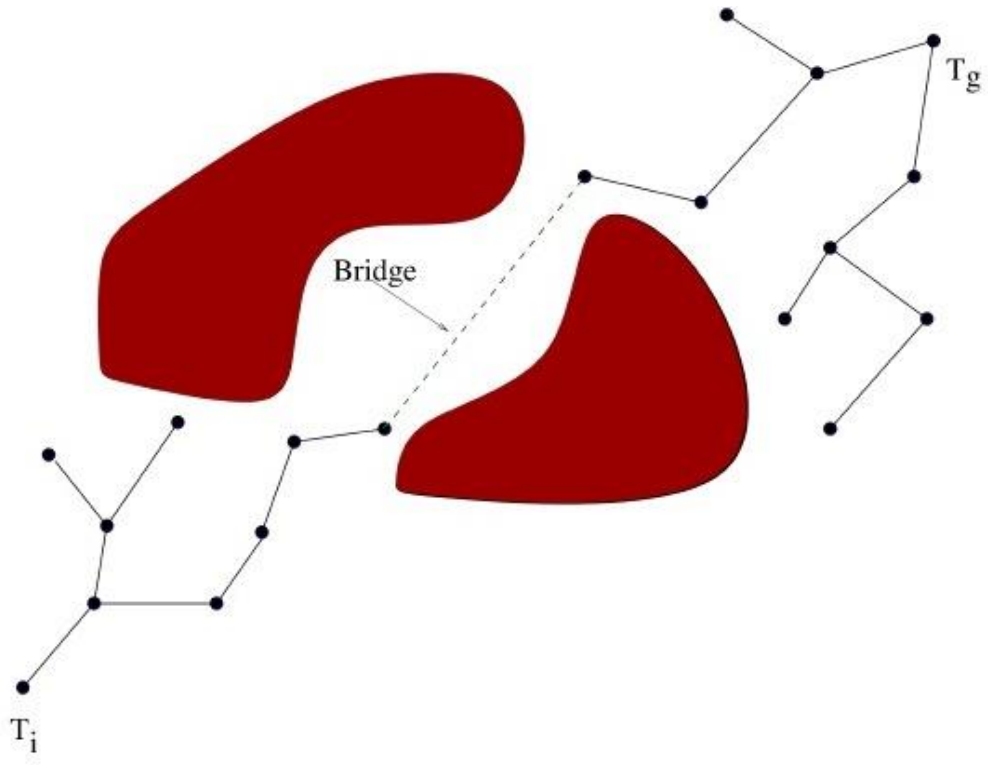



*Рис .9.9. Використання SBL алгоритму для побудови двох дерев* $T_{init}$ *та* $T_{goal}$

*Переваги* SBL алгоритму:

- знаходить розв'язок набагато швидше за інші алгоритми, бо перевірка колізій відкладається до останнього моменту і виконується лише при необхідності;
- простота реалізації.

Недоліки SBL алгоритму:

- отриманий результат може бути неоптимальний та кожного разу за однакових умов повертати зовсім інший результат, як і всі PRM алгоритми.
-

### А6 - Алгоритм RRT (Rapidly-Exploring Random Trees)

Розшифровується як швидко досліджувані випадкові дерева – це динамічна структура даних, що призначена для дослідження простору станів.

Нижче приведено зображення таких дерев у випадку двомірного простору станів.

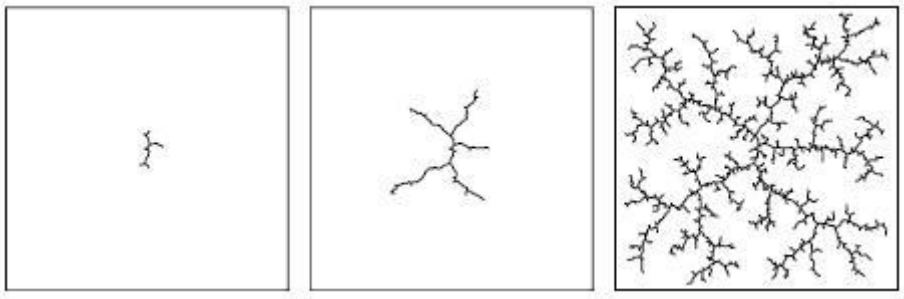


*Рис. 9.10. Зображення дерев в двомірному просторі станів*

*Головна ідея* роботи алгоритму RRT полягає в дослідженні просторів станів, починаючи з початкової конфігурації і до поки не досягне кінцевої цілі $q$. Містить дві базові конструкції: BUILD_RRT, де на кожній ітерації дерево розширяється, додаючи нові вершини, та EXTEND, що вибирає вершину,яка входить у склад дерева та є найближчою до кінцевої конфігурації. За допомогою функції NEW-CONFIG дерево розширюється на деяку фіксовану відстань $E$ до кінцевої цілі $q$ та при цьому відбувається перевірка на стан і можливість.

В результаті виконання функції NEW_CONFIG можлива поява трьох ситуацій:

1. *reached*, $q$ напряму добавлено в дерево, бо воно знаходиться не далі $E$ від вибраної вершини і є досяжним з неї;

2. *advanced*, новий стан $q_{new}$ не дорівнює $q$ , що додане до дерева;

3. trapped, запропонована вершина відкинута, так як вона не належить $C_{free}$ чи недосяжна із неї по найкоротшому шляху.

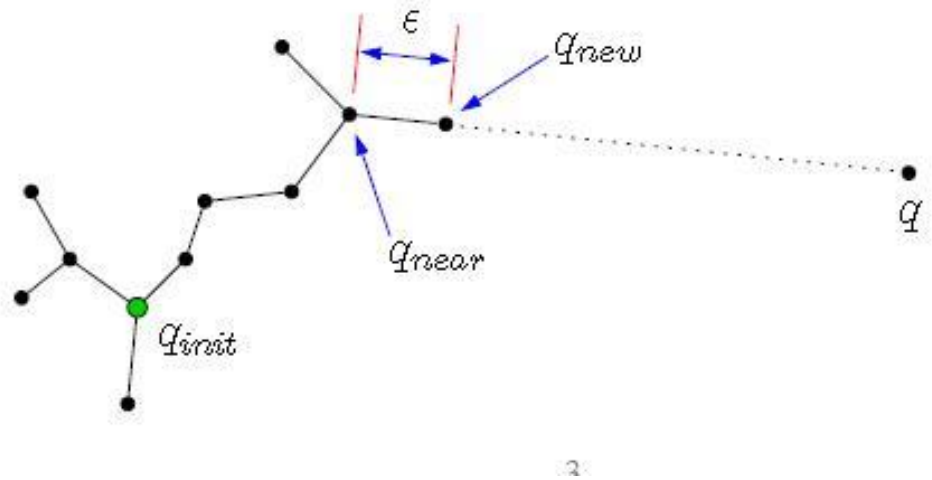



*Рис. 9.11.* Функція EXTEND алгоритму RRT

*Переваги* RRT алгоритму:

- простота та легкість реалізації;
- пріоритетним напрямом для розширення дерева є кінцева ціль $q$, що запобігає надмірного дослідження хибних напрямків.

*Недоліки* RRT алгоритму:

- при наявності великої кількості перешкод ефективність алгоритму значно зменшується.

## А7 - Алгоритм RRT Connect

Розшифровується як швидко досліджувані випадкові дерева, що розширюються назустріч. Алгоритм RRT-Connect був розроблений для планування траєкторій з врахуванням кінематики об'єктів. Алгоритм базується на двох ідеях:

1. використовується евристика, яка намагається досягнути цілі по прямій, роблячи мінімальну кількістю поворотів;
2. створюється два дерева, одне з яких починає пошук із початкового положення $q_{init}$, а друге – із кінцевого $q_{goal}$.

Евристика Connect виконує функцію EXTEND доти, доки не буде досягнуте цільове положення q або поки не зустрінеться перешкода. Вона дозволяє набагато швидше знаходити рішення. Нижче приведений алгоритм RRT-Connect. На кожній з ітерацій одне із дерев розширюється і намагається з'єднатися з найближчою вершиною іншого дерева. Після чого ролі дерев змінюються.

На рис. 9.12. приведений приклад поетапного планування траєкторій алгоритму RRT-Connect.

Алгоритм RTT-Connect будує неоптимальну траєкторію, тому її було б добре додатково оптимізувати. Для вибору оптимальної траєкторії може бути використаний алгоритм вирізання кутів. На траєкторії вибирається три суміжних вузла. Якщо переміщення по прямій із крайнього лівого вузла в крайній правий вузол можливо, то середній вузол відкидається. Процес повторюється ітеративно, доки не будуть видалені все можливі вузли.

*Основна ідея роботи алгоритму RTT-Connect* полягає в створенні двох дерев, одне з яких починає пошук траєкторії із початкового положення $q_{init}$, а друге – із кінцевого $q_{goal}$ поки не з'єднаються.

Одинична RTT-Connect ітерація представлена на рис.9.13.

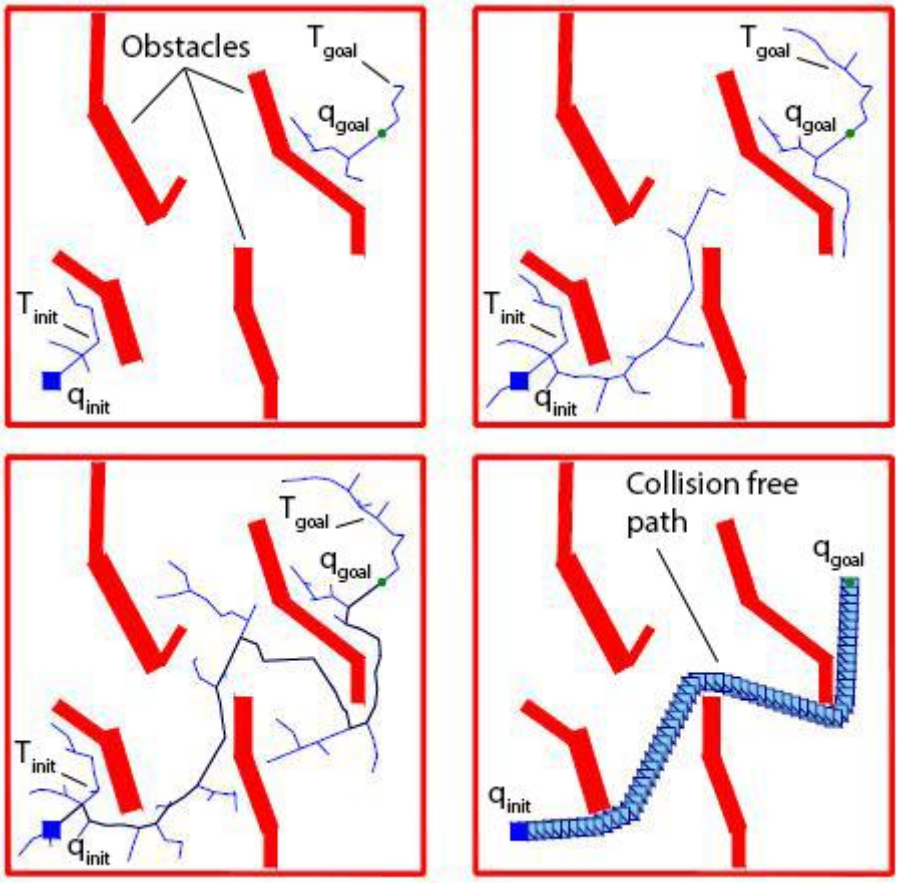



*Рис. 9.12. Етапи планування траєкторій алгоритмом RRT-Connect*

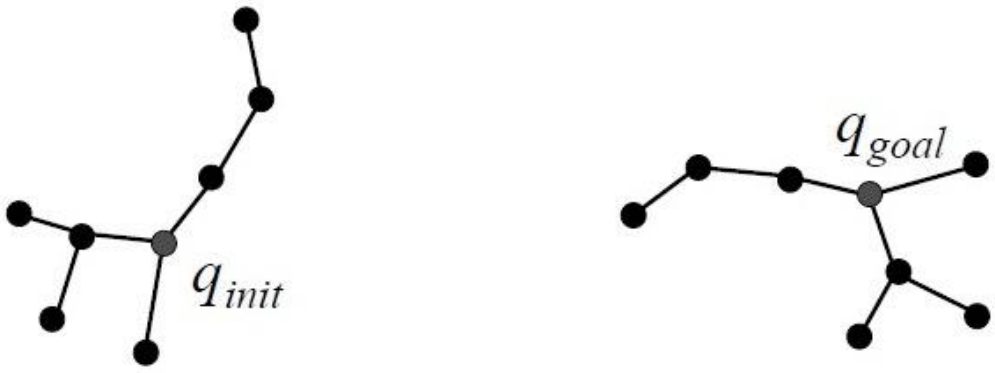



*Рис. 9.13. Одинична ітерація алгоритму RTT-Connect*

Після чого:

1) одне з дерев розширюється, використовуючи довільну ціль (див. рис. 1.17);

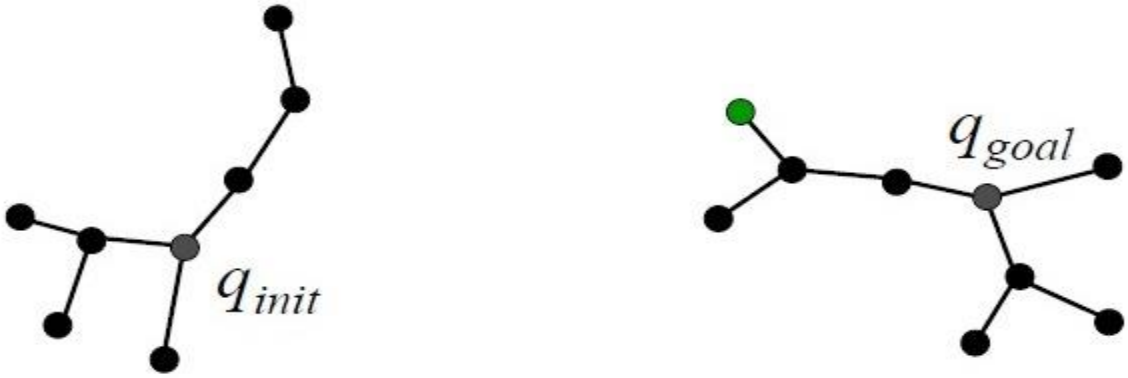



*Рис. 9.14. Довільне розширення одного із дерев*

2) нова точка $q_{target}$ стає ціллю для іншого дерева (рис.9.15);

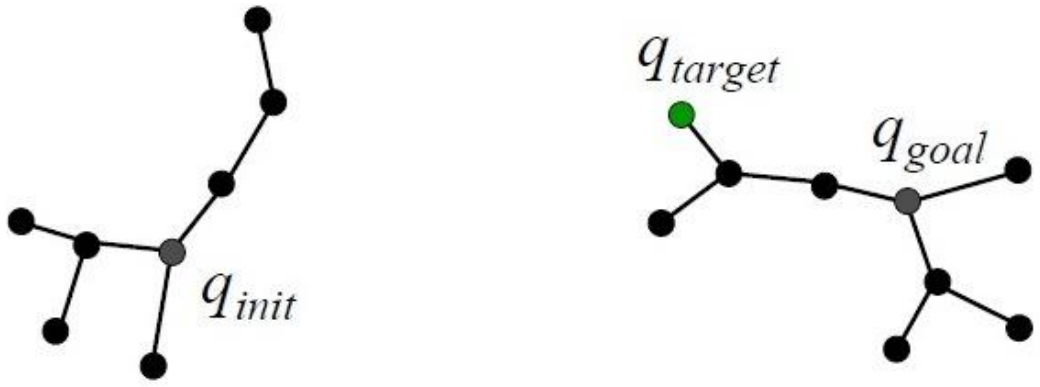



*Рис. 9.15. Точка $q_{target}$ стає ціллю для іншого дерева*

3) вираховується точка $q_{near}$, що лежить найближче до $q_{target}$ (рис .9.16);

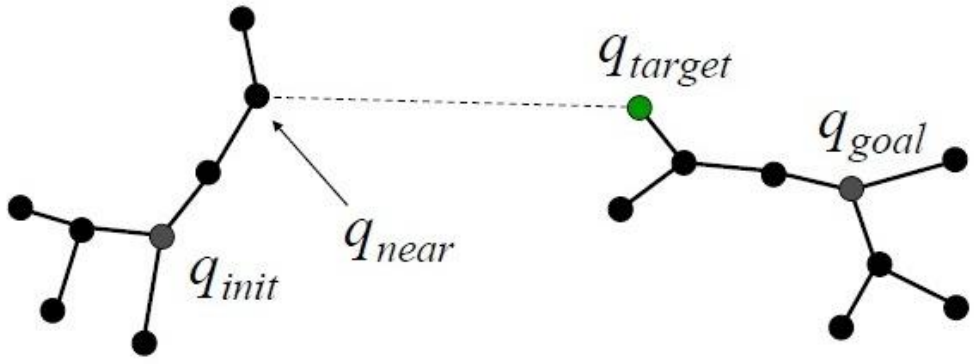



*Рис.11.16. Розрахонок точки $q_{near}$, що лежить найближче до $q_{target}$*

4) планувальник намагається додати нову, вільну від колізій вітку $q_{new}$ (рис.1.20);

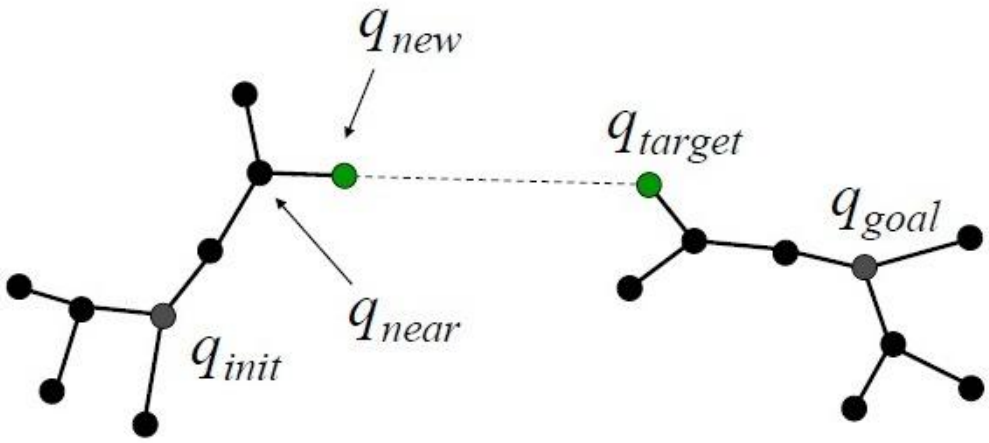



*Рис. 9.17. Додавання $q_{new}$*

5) якщо перша точка $q_{new}$ була додана успішно, то продовжується розширення вітки (рис.9.18);

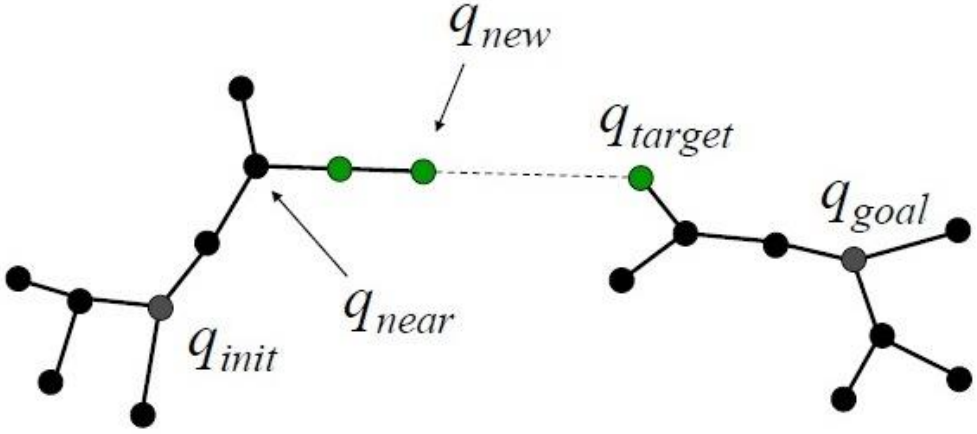



*Рис.9.18. Розширення вітки*

6) якщо успішно, то продовжуємо розширювати вітку, як показано на рис. 9.18 – рис. 9.23.

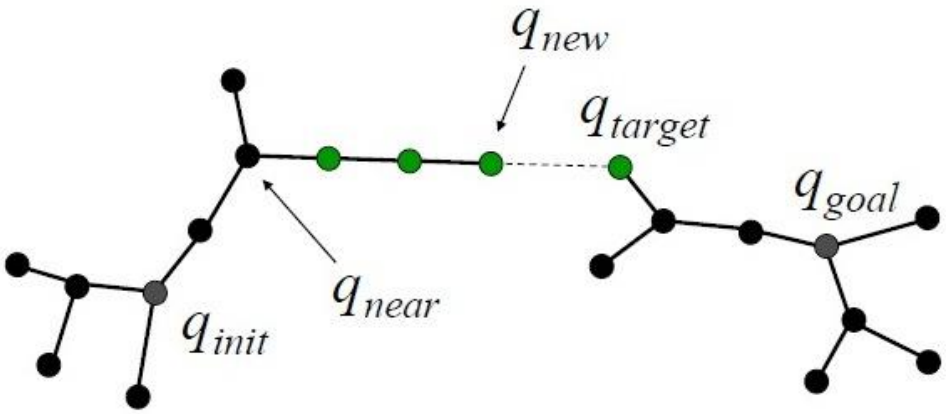



*Рис.9.18. Продовження розширення вітки $q_{new}$*

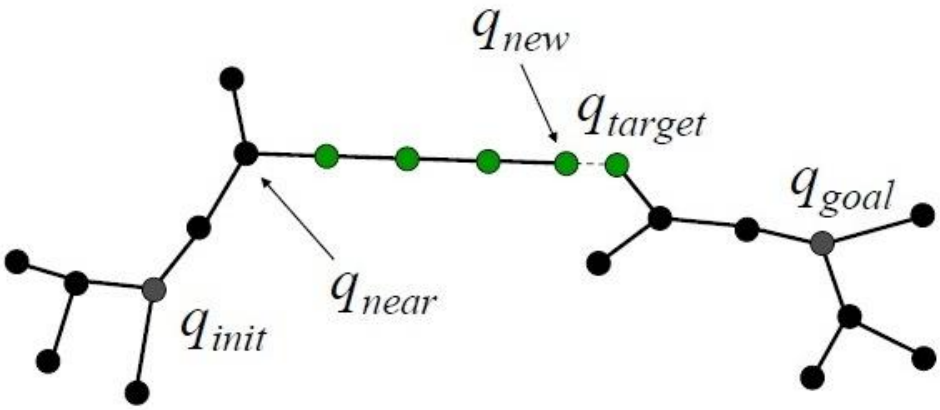



*Рис. 9.19. Продовження розширення вітки*

7) шлях знайдено, якщо вітка $q_{new}$ досягла кінцевої цілі $q_{target}$ (рис. 9.20);

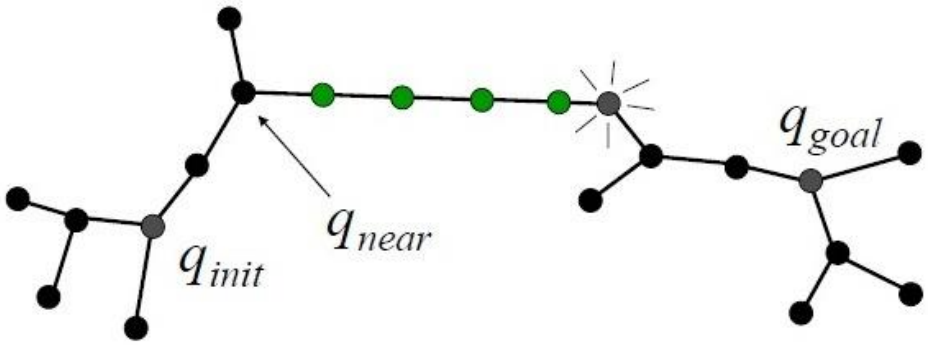



*Рис. 9.20. Досягнення цілі віткою $q_{new}$*

8) повертаємо шлях, що з’єднує початкову $q_{init}$ і кінцеву ціль $q_{goal}$ (рис. 9.21).

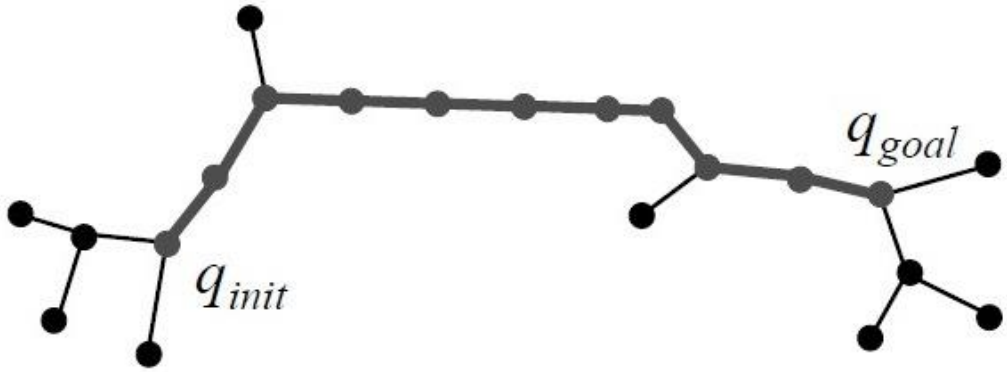



*Рис. 9.21.Схема шляху, що не містить колізій та з’єднує старт $q_{init}$ і кінцеву ціль $q_{goal}$*

*Переваги* алгоритму RTT-Connect :

- швидше знаходить розв’язок;
- простий та легкий в реалізації.

*Недоліки* алгоритму RTT-Connect:

- при наявності великої кількості перешкод ефективність алгоритму значно зменшується.

## А8 - Алгоритм Lazy RRT

Використовується евристика, яка намагається досягнути цілі по прямій, роблячи мінімальну кількість поворотів. Сворюється два дерева, одне з яких починає пошук із початкового положення $q_{init}$, а друге – із кінцевого $q_{goal}$. На кожній з ітерацій одне з дерев розширюється і намагається з’єднатися з найближчою вершиною іншого дерева. Після чого ролі дерев змінюються

Lazy RRT не перевіряє шлях на наявність колізій. Натомість він намагається знайти потрібну траєкторію, що з’єднувала б початкову та кінцеву точки якомога швидше. Коли траєкторія знайдена, вона перевіряється на колізії. У випадку виявлення колізії, сегмент, що її містить, видаляється, після чого процес пошуку продовжується.

## А9 - Алгоритм EST (Expansive Space Trees)

Розшифровується як просторові дерева, що розширюються. Це деревоподібний алгоритм планування траєкторій, що ефективно вирішує

кінематично – динамічні задачі планування траєкторій. Алгоритм EST будує два дерева $T_{init}$ та $T_{goal}$ назустріч один одному з точок $q_{init}$ та $q_{goal}$, поки два дерева не з'єднаються в одне. Також можливо будувати тільки одне дерево з $q_{init}$ до $q_{goal}$, але зазвичай це менш ефективно. На етапі конструювання нові конфігурації створюються у просторі $Q_{free}$ поблизу зовнішніх кордонів вже існуючих дерев. Конфігурація додається до дерева тільки тоді, коли вона може бути з'єднана локальним планувальником із однією з вже існуючих конфігурацій дерева. На етапі з'єднання локальний планувальник намагається з'єднати пару конфігурацій, що належать різним деревам. Якщо це вдається, то два дерева стають одним компонентом та повертається шлях, що з'єднує $q_{init}$ та $q_{goal}$.

Також в EST алгоритмі існує стратегія розширення, яка направляє процес розширення на недосліджену частину простору $Q_{free}$. Процес генерації нових конфігурацій залежить від початкової та кінцевої конфігурації, а також від усіх конфігурацій, що обули побудовані раніше. Планувальник стикається з наступною необхідністю вибору: з одного боку йому потрібно проводити пошук у частині $Q_{free}$ поблизу обраної конфігурації дерева, а з іншого боку планувальник повинен демонструвати можливість дослідити весь простір $Q_{free}$. EST розширює дерево у напрямку, протилежному від областей з найбільшим скупченням конфігурацій з гарантією, що за певних умов уся $Q_{free}$ буде досліджена.

На рис. 9.22 представлений простий приклад алгоритму EST у двомірному просторі. Нехай обрано $q$ і створено $q'_{rand}$ в певному радіусі від $q$. Як видно з рисунку, $q$ з $q'_{rand}$ успішно з'єднано. Конфігурація $q'_{rand}$ і вершина (q, $q'_{rand}$) додані до дерева *T*. Хоча $q''_{rand}$ теж було створена, ні точок, ні вершин не було додано до дерева *T*, оскільки локальний планувальник не зміг з'єднати $q$ та $q''_{rand}$.

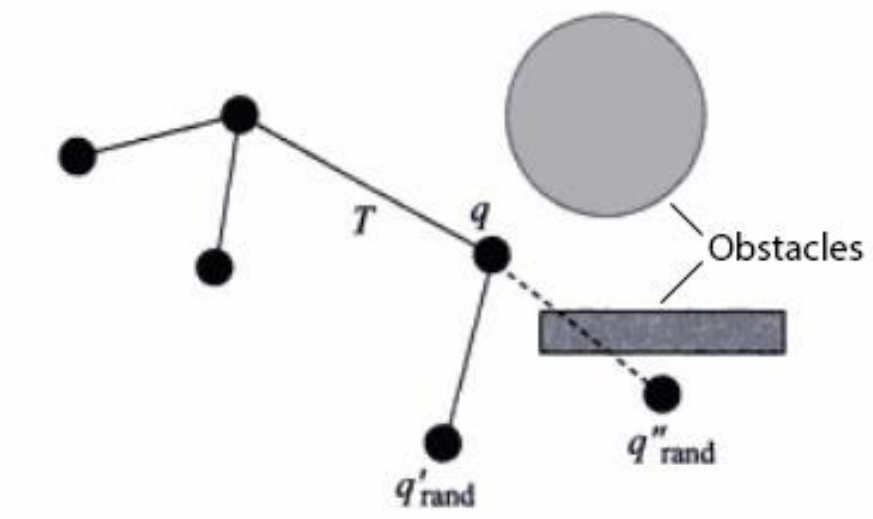



*Рис. 9.22. Приклад додавання нової конфігурації до дерева T*

Змістовний *аналіз визначених алгоритмів* вказує наступне:

- дані алгоритми досліджують простір станів, починаючи з початкової конфігурації $q_{init}$ і до поки не досягнуть кінцевої конфігурації $q_{goal}$.
- дослідження простору проходить шляхом розповсюдження дерев, що є графічною ілюстрацією можливих переміщень Сх, що розростаються від початкової точки до кінцевої, оминаючи при цьому наявні перешкоди;
- в деяких із наведених вище алгоритмів одночасно будується два дерева $T_{init}$ та $T_{goal}$ (в прямому напрямку, від початкового стану $q_{init}$, та у зворотному напрямку, тобто від цілі $q_{goal}$), зупиняючись після того, як два дерева з'єднаються;
- перевірка на колізії (зіткнення, перетини) в кожному із наведених алгоритмів проходить на різних стадіях їх роботи, але спільним є те, що у випадку виявлення перешкоди, сегмент, що її містить, видаляється, після чого процес пошуку продовжується.

За результатами роботи вказаних алгоритмів будуються безперешкодні (безколізійні) траєкторії.

Для оптимізації траєкторій використовується *алгоритм вирізання кутів*. Його стислий *зміст* зводиться до наступного;

- на траєкторії вибирається три суміжних вузла;

- якщо переміщення по прямій із крайнього лівого вузла в крайній правий вузол є можливим, то середній вузол відкидається;

- процес повторюється ітеративно, доки не будуть видалені всі можливі вузли;

- тому для побудови траєкторії, близької до оптимальної, бажано використовувати одразу декілька алгоритмів, співставляти отримані результати та оптимізувати отримані рішення.

## 9.3. Інформаційне забезпечення та формалізовані описи критеріїв вибору оптимальної траєкторії переміщення ММП

Для успішної реалізації процесу планування траєкторій ММТ вхідна інформація умовно поділена на постійну, змінну та умовно-змінну (рис. 9.23).

Постійна інформація містить основні відомості про аналізовані алгоритми, а саме - це існуючі алгоритми планування безперешкодних траєкторій. Джерелами постійної інформаціями є ресурси зі змістом та сутністю алгоритмів, а також їх програмна реалізація, що знаходиться у вільному доступі (бібліотека OMPL).

Змінна інформація містить дані про:

- сцену, що являє собою певної форми обмежений 3D-простір з наявними в ньому перешкодами;

- кінцеві та початкові координати об’єкта маніпулювання (ОМ), форма та геометричні розміри ОМ;
- кінцеві та початкові координати ММП, його форма та геометричні розміри.

*Джерелами змінної інформації* є технічне завдання, що необхідно реалізувати для досягнення поставленої мети.

В залежності від поставленого завдання, ММП пристрій може відноситись також і до умовно-змінної інформації. Це можливо в тому випадку, коли при одному ММП використовується різні ОМ.

*Джерелами умовно-змінної інформації* є конкретні виробничі умови, в яких аналізуються / проектуються відповідні траєкторії.

Результатом переміщення ММП по знайдених траєкторіях при розв’язуванні поставленої задачі є вибір оптимальних з них за попередньо прийнятими критеріями гладкості (**S**) або довжини (**L**) на 3-ох ділянках сцени (див. далі).


*Рис. 9.23. Структура пропонованого інформаційного забезпечення*

Основними вимогами, які мають бути при такому інформаційному забезпечені, є:

- можливість безперешкодного (безколізійного) переміщення ММП за найкоротшою із можливих траєкторій;

- реалізація принципу безрозривності та плавності самих траєкторій.

Формальна постановка розв'зуваних завдань передбачає визначення запропонованої системи критеріїв, що логічно випливають із змістовної сутності аналізованих алгоритмів $\mathbf{A_1, \dots, A_9}$.

В зв'язку з цим критерії, що використовуються в подальшому, є наступними:

- *локальні*:

  - мінімальна довжина $\mathbf{\min L_{A_j}}$ локальних траєкторій переміщення ММП на ділянках **1-2**, **2-3** та **3-1**, кожу з яких забезпечує відповідний $\mathbf{A_j}$-ий алгоритм із їх кінцевої аналізованої множини $\mathbf{A = (A_j | \, j = \overline{1, 9})}$:

$$\boldsymbol{F_{L_{A_j}}(z_i) = \min L_{A_j}, i = \overline{1,3};\ z \in (1-2,\ 2-3,\ 3-1); j = \overline{1,9}} \qquad (9.1)$$

де $\mathbf{A_1}$ – алгоритм **KPIECE**;
$\mathbf{A_2}$ – алгоритм **Bi – directional KPIECE** ;
$\mathbf{A_3}$ – алгоритм **Lazy Bi – directional KPIECE**;
$\boldsymbol{A_4}$ – алгоритм **PRM**;
$\mathbf{A_5}$ – алгоритм ***SBL*** ;
$\mathbf{A_6}$ – алгоритм **RRT** ;
$\mathbf{A_7}$ – алгоритм **Lazy RRT** ;
$\mathbf{A_8}$ – алгоритм **RRT Connect** ;
$\mathbf{A_9}$ – алгоритм **EST**;
$\mathbf{z_1}$ – **1**-ша ділянка;
$\mathbf{z_2}$ – **2**-га ділянка;
$\mathbf{z_3}$ – **3**-тя ділянка (про ці ділянки див. далі);

  - найменша гладкість відпрацювання локальних траєкторій переміщення ММП на ділянках **1-2**, **2-3** та **3-1**:

$$\mathbf{F_{S_{A_j}}(z_i) = \min S_{A_j}, i = \overline{1,3};\ z \in (1-2,\ 2-3,\ 3-1); j = \overline{1,9};\ S_{A_j} \in [0,1]} \qquad (9.2)$$

де $\mathbf{A_1, \dots, A_9}$ – наведені вище алгоритми планування траєкторій;
$\mathbf{z_1, z_2, z_3}$ – перша, друга та третя ділянка траєкторій відповідно;

- *<u>глобальний:</u>*

• сумарна мінімальна довжина переміщення ММП з врахуванням мінімальної довжини переміщення ММП на кожній із ділянок **1-2**, **2-3** та **3-1**одним із аналізованих алгоритмів:

$$\mathbf{F_{L_{min}}} = \mathbf{min}\, \mathbf{L_{A_j(1-2)}} + \mathbf{min}\, \mathbf{L_{A_j(2-3)}} + \mathbf{min}\, \mathbf{L_{A_j(3-1)}}, \qquad (9.3)$$

де $\mathbf{L_{A_j(1-2)}}$ – довжина траєкторії на ділянці **1-2**;

$\mathbf{L_{A_j(2-3)}}$ – довжина траєкторії на ділянці **2-3**;

$\mathbf{L_{A_j(3-1)}}$ – довжина траєкторії на ділянці **3-1**.

Доречно наголосити, що глобальний критерій щодо гладкості траєкторій $\mathbf{F_S}$ підлягає додатковому детальному дослідженню, яке саме тут і в такій постановці не розглядається. Аналогічне можна стверджувати і щодо комбінованого критерію $\mathbf{F_{LS}}$, що комплексно враховує довжини та гладкості траєкторій, тобто локальні критерії $\mathbf{F_{L_{A_j}}}$ та $\mathbf{F_{S_{A_j}}}$.

## 9.4. Стуктурна схема розробленого програмного продукту (ПП) “LSTr“ та приклади його працездатності

Нижче представлено опис розроблено програмного продукту (ПП) “**LSTr**” та наведені результати тестових прикладів його функціонування. Для програмної реалізації змісту поставлених задач вибрана мова програмування С++ з використанням:

- графічної бібліотеки *OpenGL* (*Open Graphics Library*);

- *QT*, що являє собою крос-платформовий інструментарій розробки програмного забезпечення та включає в себе всі основні класи, які потрібні при розробці елементів графічного інтерфейсу;

- бібліотеки *OMPL* (*Open Motion Planning Library*) для побудови самих траєкторій;

- бібліотека *Assimp* (*Open Asset Import Library*) для роботи з 3D-моделями;

- програма *SketchUp* для створення 3D-моделей сцен.

На рис. 9.24 зображена структурна схема функціонування розробленого ПП, що демонструє взаємозв’язок різних прикладних пакетів програмування для планування безперешкодних (безколізійних) траєкторії переміщення ММП.

На рис. 9.25 зображено інформаційне поле розробленого ПП “**LSTr**”. Згідно рис. 9.25 задача розбивається на три частини, розв’язок яких виконується на таких ділянках:

1-ша ділянка – переміщення ММП від початкового (**Initial position**) до ОМ (**Take object of manupulation**) (ділянка **1-2**);

**2**-га ділянка – ММП з ОМ переміщується до цілі, де залишає ОМ (ділянка **2-3**);

**3**-тя ділянка – "пустий", тобто без ОМ в Сх, ММП повертається у початкове положення (ділянка **3-1**).

Для вирішення тестових задач як приклад було створено три сцени, що включають в себе простір стану та параметри перешкод, ММП та ОМ, початкову та кінцеву ціль ММП, тобто початкові та кінцеві координати точок переміщення мехатронного мобільного пристрою.

На рис. 9.24 – 9.28 представлені результати розв'язання тестової задачі №1, що відповідає побудованій сцені 1.

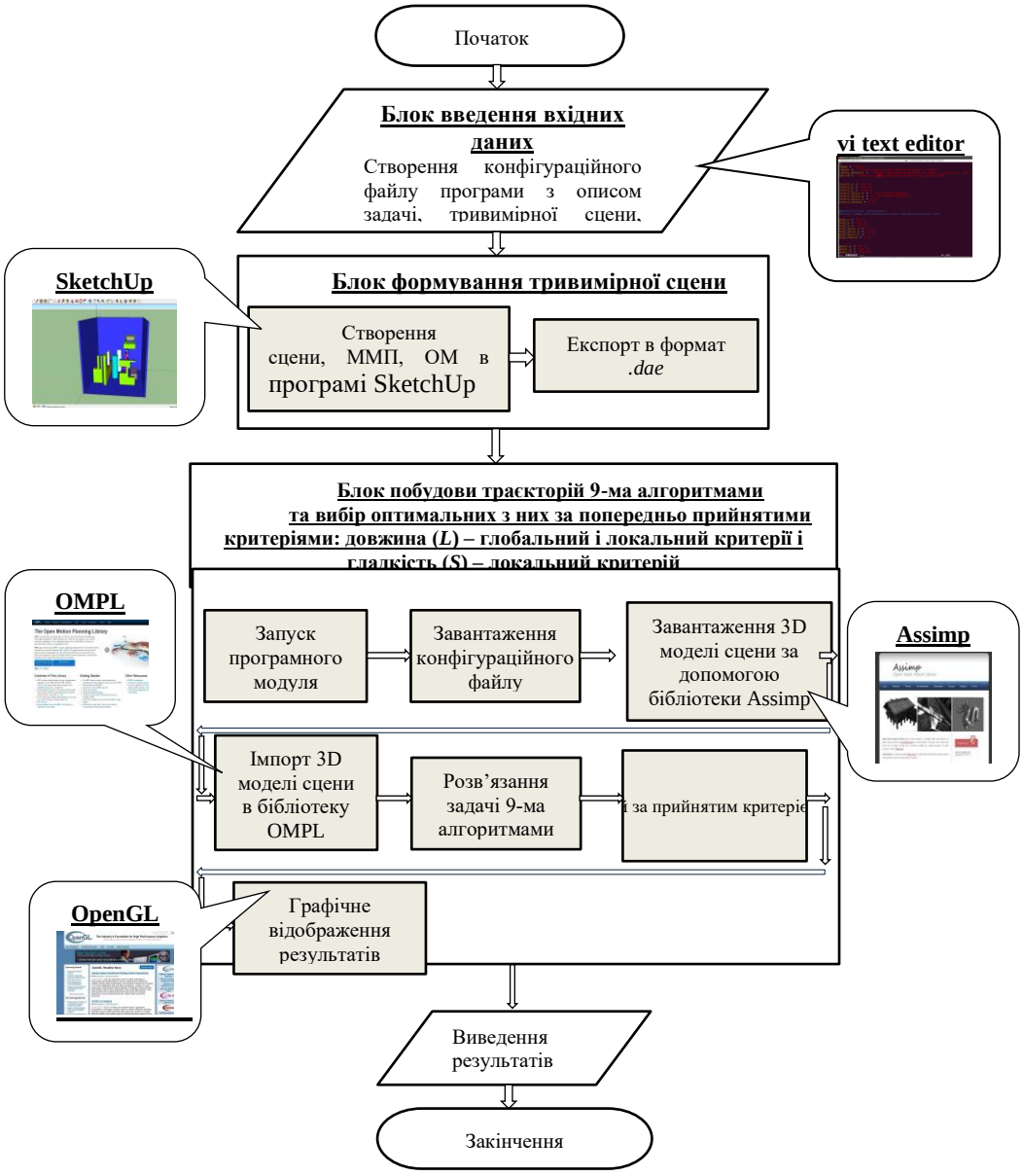



*Рис. 9.24. Структурна схема функціонування розробленого ПП* "**LSTr**"

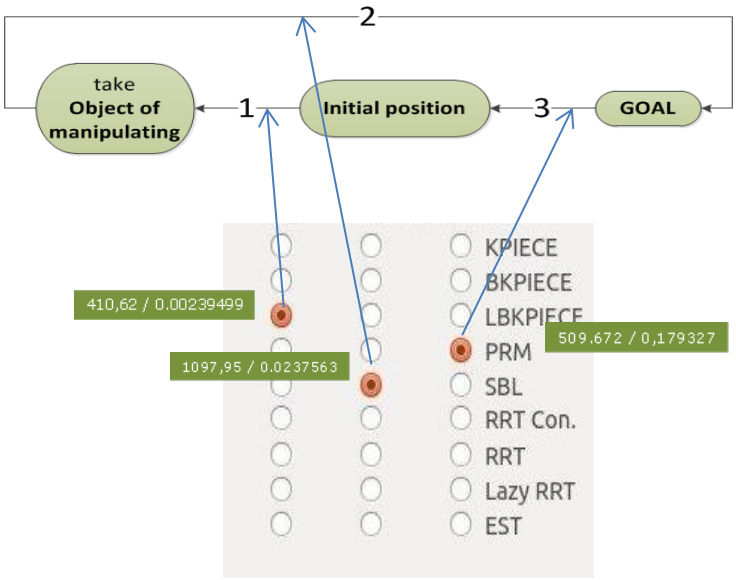



*Рис. 9.25. Інформаційне поле розробленого* ПП “**LSTr**”

На рис. 9.26 представлені результати оптимальної траєкторії за критерієм її довжини **L**, а на рис. 9.27. – за критерієм гладкості **S**.

Відповідні кінцеві параметри вказані нижче.

*За рис. 9.26:*

• на **1**-ій ділянці: найкращий результат забезпечив алгоритм **SBL** зі значенням (тут та далі величини двнф в умовних одиницях) 336,810;

• на **2**-ій ділянці: найкращий результат дав алгоритм **EST** зі значенням 710,749;

• на **3**-й ділянці: найкращий результат відпрацював алгоритм **EST** зі значенням 306,724.

Дані результати вказані на рис. 9.30.

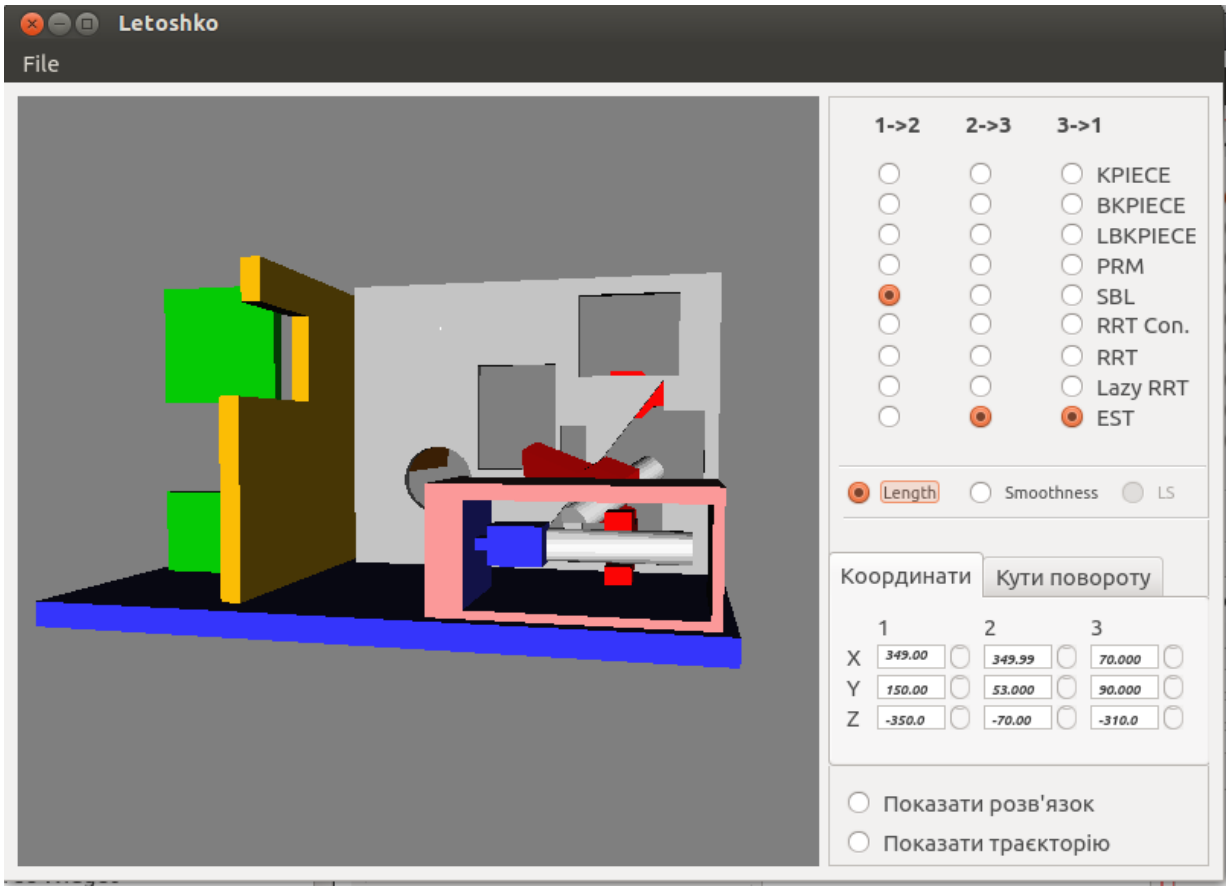



*Рис. 9.26. Результат роботи аналізованих алгоритмів на сцені №1 за критерієм довжини* **L**

*За рис. 9.27:*

- на 1-ій ділянці: найкращий результат забезпечив алгоритм **KPIECE** зі значенням $\mathbf{S_1}$=0,00614645;
- на 2-ій ділянці: найкращий результат дав алгоритм **Lazy RRT** зі значенням $\mathbf{S_2}$=0,00712410;
- на 3-ій ділянці: найкращий результат відпрацював алгоритм EST зі значенням $\mathbf{S_3}$=0,015405091.

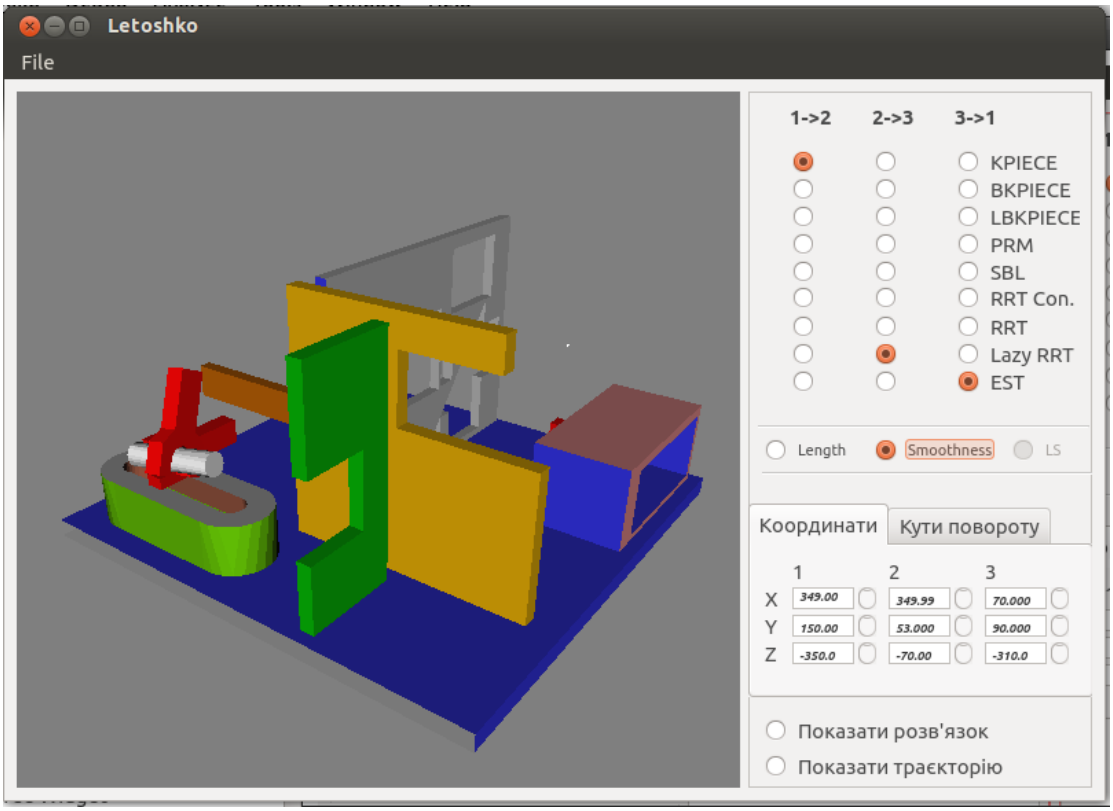



*Рис. 9.27. Результат роботи аналізованих алгоритмів на сцені №1 за критерієм гладкості* **S**

Графічне представлення отриманих траєкторій за критеріями довжини **L** та гладкості **S** зображені відповідно на рис. 9.28. та рис. 9.29.

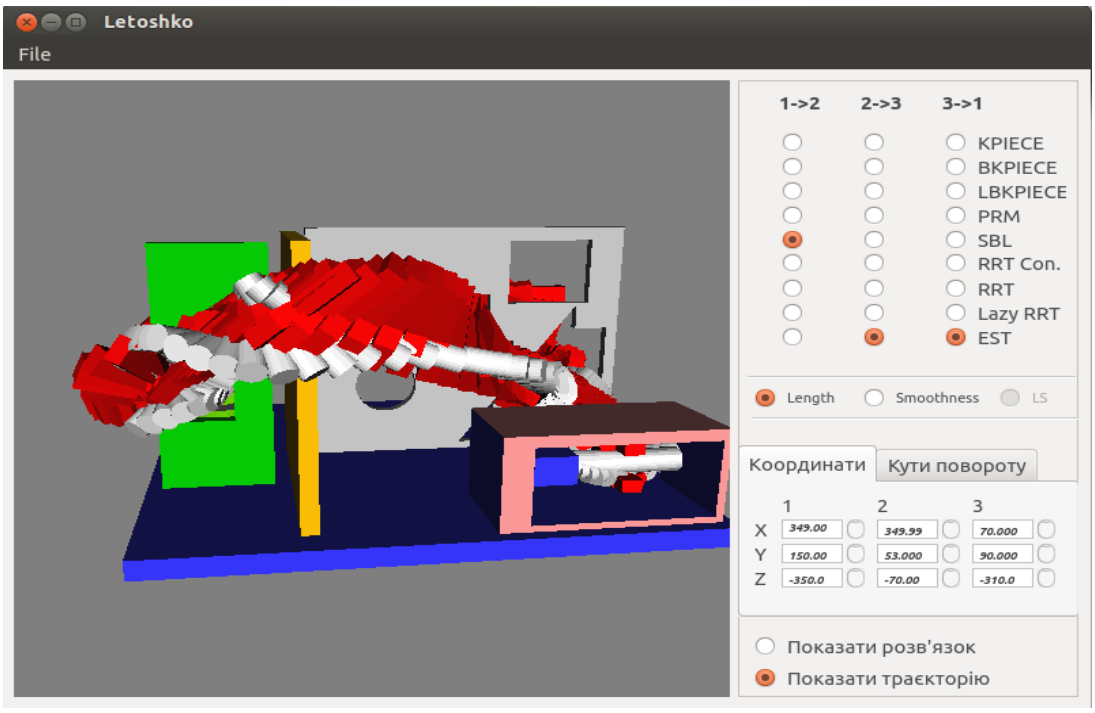



*Рис. 9.28. Графічне представлення траєкторії за даними рис. 9.4*

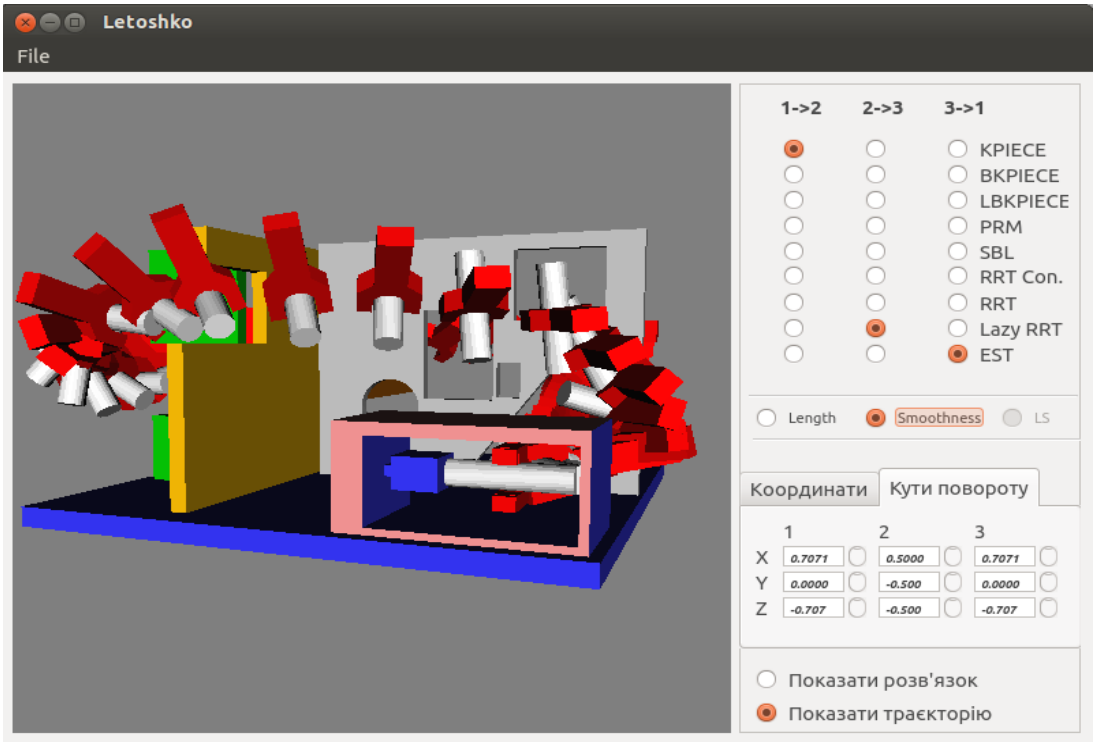



*Рис. 9.29. Графічне представлення траєкторії за даними рис.9.26 за критерієм* ***S***

Загальні результати траєкторій за всіма 9-ма алгоритмам за прийнятими критеріями довжини **L** та гладкості **S** приведені на рис. 9.30.

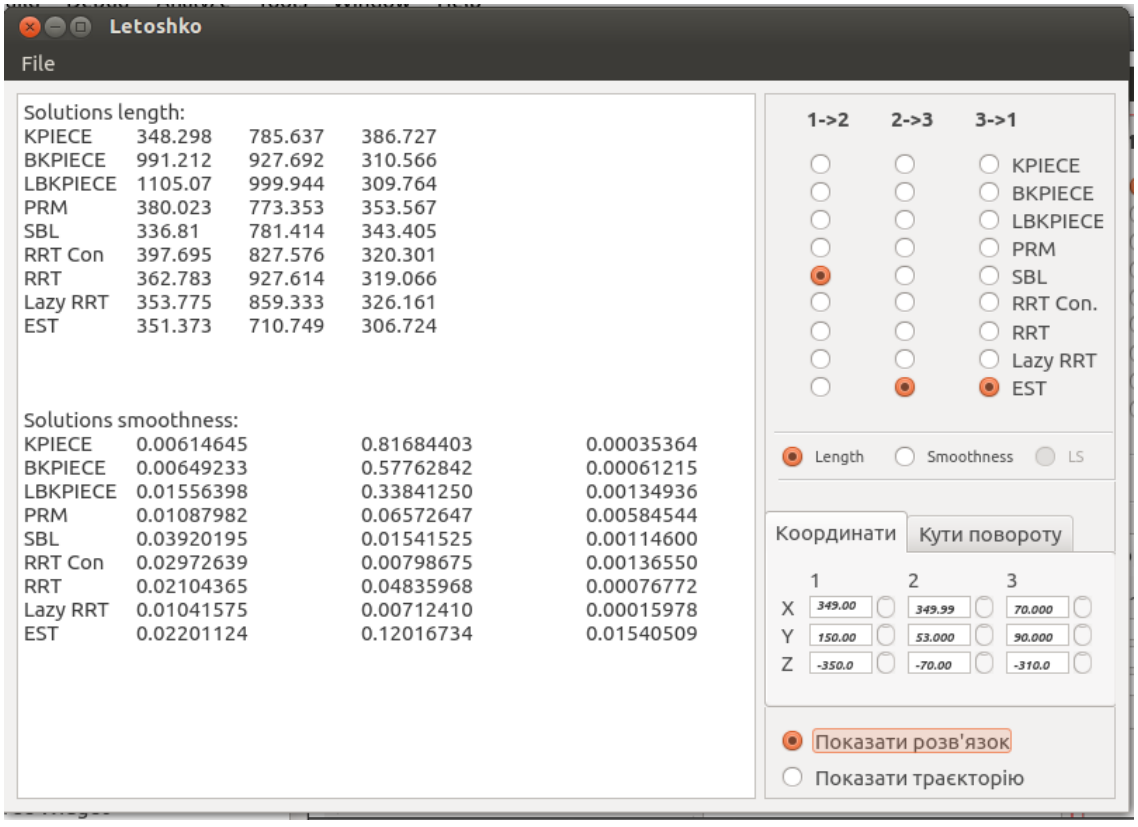



*Рис. 9.30. Загальні результати роботи ПП "***LSTr***" за 9-ма алгоритмами*

Як слідує із наведеного вище, створений ПП “**LSTr**” дозволяє розв’язувати задачу планування безперешкодного (безколізійного) переміщення ММП з ОМ та без нього в межах попередньо сформованих сцен.

Таким чином, представлений матеріал вказує на його **наукову новизну**, яку можна сформулювати наступним чином:

- *набув подальшого розвитку підхід* до визначення оптимального планування траєкторій за критеріями довжини та гладкості, що базується на основі існуючих алгоритмів безколізійного переміщення рухомих об’єктів (ММП) з врахуванням змодельованої сцени, конструктивно-геометричних параметрів ММП та ОМ, робота яких передбачає визначення локальних критеріїв довжини та гладкості, і дає можливість в подальшому розв’язувати пряму та зворотні задачі кінематики.

Виконані розробки і отримані результати розглядаються як **практичне** підґрунтя для подальшого розвитку підходу щодо планування траєкторій ПР та / або ММТ з розв’язанням прямої та зворотної задач кінематики, а в подальшому і прямої та зворотної задач динаміки. Остання є основою для розробки законів керування ланками МС ПР та його Сх.