References

1. ASHRAE, Commercial [Text] / Institutional Ground-Source Heat Pump Engineering Manual, American Society of Heating, Refrigerating, and Air-Conditioning Engineers, Inc. Atlanta. – GA, 2003.
2. Amis, T. Energy piles and other thermal foundations for GSHP [Text] / T. Amis, F. Loveridge // REHVA Journal. – 2014. – P. 32–35. – Available at: http://www.rehva.eu/fileadmin/REHVA_Journal/REHVA_Journal_2014/RJ_issue_1/P.32/32-35_Energy_piles_RJ1401_WEB.pdf
3. Design manual. Heat pumps [Text]. – Viessmann, 2012. – P. 65.
4. System using geothermal heat. Technical information [Text]. – REHAU, 2009. – P. 20.
5. Design manual. Heat pumps for heating and hot water supply [Text]. – Dimplex, 2009. – P. 77.
6. Suguang, X. Use of Geothermal Deep Foundations for Bridge Deicing [Text] / S. Xiao, M. Suleiman, C. Naito, S. Neti // Transportation Research Record: Journal of the Transportation Research Board. – 2013. – Vol. 2363. – P. 56–65. doi: 10.3141/2363-07
7. Rees, S. A Study of Geothermal Heat Pump and Standing Well Perfomance [Text] / S. Rees, J. Splintler, Z. Deng, C. Orio, C. Johnson // ASHRAE. – 2004. – Vol. 109. – P. 3–13.
8. Xiao, S. Use of Geothermal Deep Foundations for Bridge Deicing, Lehigh University [Text] / S. Xiao, M. T. Suleiman, C. J. Naito, S. Neti. – Report of TRB 2013 Annual Meeting, 2013.
9. Man, Y. Development of spiral heat source model for novel pile ground heat exchangers [Text] / Y. Man, H. Yang, N. Diao, P. Cui, L. Lu, Z. Fang // HVAC&R Research. – 2011. – Vol. 17, Issue 6. – P. 1075–1088. doi: 10.1080/10789669.2011.610281
10. Thermal Pile Design, Installation & Materials Standards [Text]. – Milton, UK., 2012. – 85 p. – Available at: http://www.gshp.org.uk/pdf/GSHPA_Thermal_Pile_Standard.pdf

References

1. ASHRAE, Commercial (2003). GA.
2. Amis, T., Loveridge, F. (2014). Energy piles and other thermal foundations for GSHP. REHVA Journal, 32–35. Available at: http://www.rehva.eu/fileadmin/REHVA_Journal/REHVA_Journal_2014/RJ_issue_1/P.32/32-35_Energy_piles_RJ1401_WEB.pdf
3. Design manual. Heat pumps (2012). Viessmann, 65.
4. System using geothermal heat. Technical information (2009). REHAU, 20.
5. Design manual. Heat pumps for heating and hot water supply (2009). Dimplex, 77.
6. Xiao, S., Suleiman, M., Naito, C., Neti, S. (2013). Use of Geothermal Deep Foundations for Bridge Deicing. Transportation Research Record: Journal of the Transportation Research Board, 2363, 56–65. doi: 10.3141/2363-07
7. Rees, S., Splintler, J., Deng, Z., Orio, C., Johnson, C. (2004). A Study of Geothermal Heat Pump and Standing Well Perfomance. ASHRAE, 109, 3–13.
8. Xiao, S., Suleiman, M. T., Naito, C. J., Neti, S. (2013). Use of Geothermal Deep Foundations for Bridge Deicing, Lehigh University. Report of TRB 2013 Annual Meeting.
9. Man, Y., Yang, H., Diao, N., Cui, P., Lu, L., Fang, Z. (2011). Development of spiral heat source model for novel pile ground heat exchangers. HVAC&R Research, 17 (6), 1075–1088. doi: 10.1080/10789669.2011.610281
10. Thermal Pile Design, Installation & Materials Standards (2012). Milton, UK., 85. Available at: http://www.gshp.org.uk/pdf/GSHPA_Thermal_Pile_Standard.pdf

Рекомендовано до публікації д-р техн. наук, професор Приймак О. В.
Дата надходження рукопису 08.06.2016

Kuzytskyi Ivan, Postgraduate student, Department of Heat Engineering, Kyiv National University of Construction and Architecture, Povitroflotskyi ave., 31, Kyiv, Ukraine, 03680
E-mail: Kuzytskyi@gmail.com

УДК 004.8, 004.94
DOI: 10.15587/2313-8416.2016.73625

ЕФЕКТИВНА ПОБУДОВА ШЛЯХУ ДЛЯ ЧОТИРИКОЛІСНИХ РОБОТІВ НА ОСНОВІ ПОЄДНАННЯ АЛГОРИТМІВ Theta* І ГІБРИДНОГО A*

© В. Г. Михалько, І. В. Круш

Запропоновано алгоритм ефективної побудови шляху для чотириколісних роботів на основі поєднання алгоритмів Theta і гібридного A*. Наведено і пояснено псевдокод для алгоритму . Реалізовано запропонований алгоритм і симулятор чотириколісного робота на мові програмування Java. Протестовано роботу алгоритму для U-подібних перешкод, складних карт і для вирішення задачі паркування*
***Ключові слова**: робототехніка, чотириколісні роботи, штучний інтелект, алгоритм пошуку шляху, Theta*, гібридний A**

Effective pathfinding algorithm based on Theta and Hybrid A* algorithms was developed for four-wheeled robot. Pseudocode for algorithm was showed and explained. Algorithm and simulator for four-wheeled robot were implemented using Java programming language. Algorithm was tested on U-obstacles, complex maps and for parking problem*
***Keywords**: robotics, four-wheeled robot, artificial intelligence, pathfinding algorithm, Theta*, Hybrid A**

1. Вступ

Одним із основних напрямків у сучасній робототехніці є проектування і побудова безпілотних автомобілів. Зважаючи на значну кількість автомобільних аварій, які є наслідком людської помилки, а також на те, що системи, засновані на використанні

машинного навчання і штучного інтелекту в багатьох галузях вже перевершують людей – цей напрям є дуже перспективним, адже у разі побудови розумних і надійних систем для автоматичного керування автомобілями можна буде значно підвищити безпеку на дорогах.

Важливою задачею при побудові безпілотних автомобілів є прокладання шляху з однієї точки в іншу. При цьому, для успішного вирішення цієї задачі до цільового алгоритму ставляться декілька основних вимог:

– шлях повинен бути близьким до оптимального;

– при плануванні шляху необхідно враховувати кінематику автомобіля;

– алгоритм прокладання шляху має бути доволі швидким і працювати за прийнятний час.

Більшість існуючих алгоритмів пошуку шляху поділяються на дві групи: одні з них є доволі швидкими, але не враховують кінематику руху платформи, а інші кінематику руху враховують, але є повільними у виконанні.

У цій роботі запропоновано метод пошуку шляху, оснований на поєднанні алгоритмів із описаних вище груп, а саме Theta* і гібридного A*. В результаті в комплексному алгоритмі, з одного боку враховується кінематика чотириколісної рухомої платформи, а з іншого – він є доволі ефективним з точки зору часу виконання. Окрім цього, запропонований алгоритм дозволяє будувати шляхи, які є близькими до оптимальних, тобто, іншими словами, він задовольняє всі три вимоги, наведені вище.

2. Аналіз літературних даних та попередніх досліджень

Існує значна кількість алгоритмів для пошуку і побудови шляху, серед них можна виділити такі:

– пошук в ширину і глибину на графі;

– алгоритм Дейкстри;

– A*;

– Theta*;

– RRT.

Найпростішими алгоритмами для пошуку є пошук в ширину і глибину. Більш складним є алгоритм Дейкстри [1, 2], який дозволяє знайти найкоротший шлях на зважених графах, але, в той же час, є доволі повільним, так як розглядає вершини у всіх напрямках. Алгоритми A* [3–5] і Theta* [6] використовують евристичну функцію, яка дає оптимістичний прогноз для правильного напрямку пошуку, що в свою чергу робить їх значно швидшими за алгоритм Дейкстри. Але основним недоліком описаних вище алгоритмів є те, що вони призначені для голономних платформ і роботів, тобто таких, які можуть миттєво рухатись в будь-якому напрямку. В свою чергу, чотириколісні рухомі платформи, які в тому числі є базою для автомобілів є неголономними платформами. Неголономні рухомі платформи – це такі платформи, які можуть рухатись тільки у деяких напрямках і по заданим траєкторіям, тобто в них є обмеження на кінематику руху. Чотириколісні роботи можуть рухатись або прямо, або по дузі кола, радіус якого залежить від кута повороту передніх коліс. Тому, в загальному випадку, такі алгоритми як A* і Theta* не можна напряму застосувати для автомобілів. Приклад цього зображено на рис. 1.

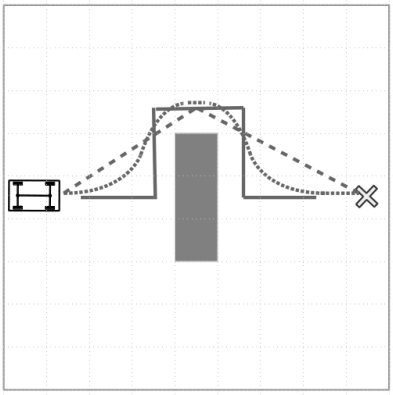


Рис. 1. Варіанти шляхів: суцільною лінією зображено траєкторія, побудована алгоритмом A*; штриховою – алгоритмом Theta*; точковою – можливу для виконання траєкторію

Ще одним підходом для вирішення задачі побудови шляху є використання алгоритму RRT (rapidly exploring random tree) [7], який дозволяє враховувати кінематику руху роботу і будувати можливі для виконання траєкторії. Але в ньому не передбачено використання евристичної функції, внаслідок чого час, за який виконується пошук, часто є неприйнятним.

Іншим варіантом вирішення проблеми є використання алгоритму “Гібридний A*”[8, 9]. Його основними відмінностями від звичайного A* є такі:

– в кожній дискретній клітинці поля зберігаються дійсні значення положення робота, а також його напрям;

– при плануванні наступного кроку розглядаються тільки такі рухи платформи, в яких враховується її кінематика.

На рис. 2 зображено приклад двох ітерацій для алгоритму “Гібридний A*”:

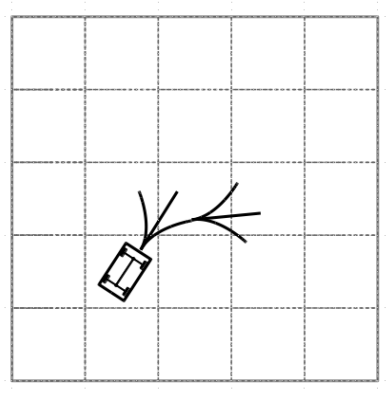


Рис. 2. Дерева руху, побудовані алгоритмом “Гібридний A*”

Гібридний A* враховує кінематику платформи, а також використовує евристичну функцію для збільшення ефективності виконання, але, тим не менш, він все одно є значно повільнішим за класичні A* і Theta*, і в деяких випадках, зокрема на картах з

великою кількістю перешкод, а також при наявності U-подібних перешкод (рис. 3), може працювати доволі повільно.

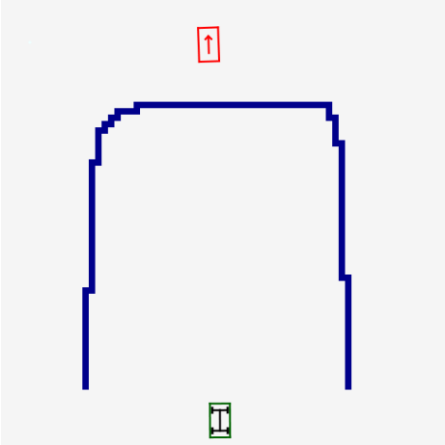


Рис. 3. U-подібна перешкода. Знизу зображено рухому платформу, а зверху – цільову позицію і напрям

Виходячи з цього, можна зробити висновок, що більшість існуючих алгоритмів можна поділити на дві основні групи: швидкі алгоритми, в яких не враховується кінематика платформи і повільні алгоритми, які враховують кінематику. Тобто, вони лише частково вирішують поставлену задачу.

3. Мета та задачі дослідження

Основною метою даного дослідження є розробка швидкого алгоритму пошуку і прокладання шляху на основі поєднання алгоритмів Theta* і гібридного A*. Розроблений алгоритм враховує кінематику руху чотириколісної рухомої платформи і будує шлях, який є близьким до оптимального.

Для досягнення поставленої мети були вирішені наступні задачі:

– реалізовано алгоритм Theta*;

– реалізовано алгоритм «Гібридний A*»;

– розроблено і реалізовано комплексний алгоритм, на основі поєднання Theta* і гібридного A*;

– реалізовано стимулятор чотириколісної рухомої платформи для тестування алгоритму.

4. Симулятор рухомої платформи

Для тестування роботи алгоритму було реалізовано симулятор чотириколісної рухомої платформи з паралельним рульовим керуванням, в якої центр кола повороту знаходиться на перетині ліній передньої і задньої осей. Схематично її зображено на рис. 4.

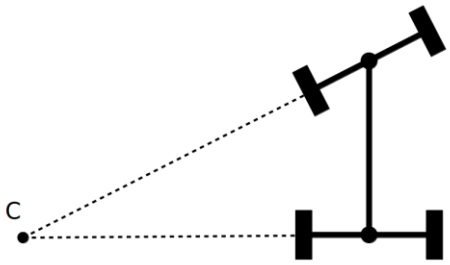


Рис. 4. Рухома платформа з паралельним рульовим керуванням. Точка C – центр кола повороту

В реальних автомобілях використовується схема рульового керування, побудована за принципом Акермана [10], в якій передні колеса при повороті відхилені на різні кути (рис. 5), але її використання не впливає на роботу алгоритму, тому для простоти було обрано варіант з паралельним керуванням.

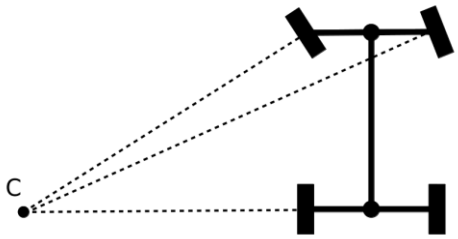


Рис. 5. Рульове керування за принципом Акермана. Точка C – центр кола повороту

Окрім цього, для спрощення також було припущено, що у платформи є лише три режими для повороту коліс: прямо, ліворуч на заданий кут і праворуч на заданий кут. Але доволі легко розширити роботу алгоритму і на випадок, коли значення кута повороту задається довільним з заданого проміжку.

5. Алгоритм прокладання шляху, оснований на поєднанні Theta* і гібридного A*

Порівнюючи алгоритми A* і Theta* треба зазначити, що алгоритм A* при роботі використовує манхеттенську відстань [11], тому будує "прямокутні траєкторії", які часто не є близькими до оптимальних в реальному світі. Цього недоліку немає в алгоритмі Theta*, в якому використовується евклідова відстань, і лінії, з яких складається траєкторія, можуть лежати під будь-яким кутом, це показано на рис. 1.

Як було зазначено вище, алгоритми A* і Theta* є доволі швидкими, проте вони в загальному випадку будують траєкторії, які є неможливими для виконання чотириколісним роботом.

З іншого боку, алгоритм "Гібридний A*" враховує кінематику руху робота, але є більш складним з точки зору обчислень, і в деяких ситуаціях працює доволі повільно.

Виходячи з цього, можна спробувати поєднати алгоритми Theta* і гібридний A*, щоб, з одного боку, пошук шляху відбувався доволі швидко, а з іншого – побудований шлях був можливий для виконання чотириколісною рухомою платформою.

Це поєднання можна реалізувати таким чином:

1. Спочатку побудувати шлях за допомогою алгоритму Theta* (в результаті отримаємо набір сполучених відрізків, які ведуть від початкової точки до кінцевої, оминаючи перешкоди).

2. Послідовно для кожного відрізку із шляху, побудованого за допомогою алгоритму Theta*, запускати алгоритм "Гібридний A*". Кожен з цих відрізків не перетинає перешкоди, тому гібридний A* має працювати доволі швидко.

5. 1. Алгоритм Theta*

Першою задачею є реалізація алгоритму Theta*. В цьому алгоритмі на кожній ітерації потріб-

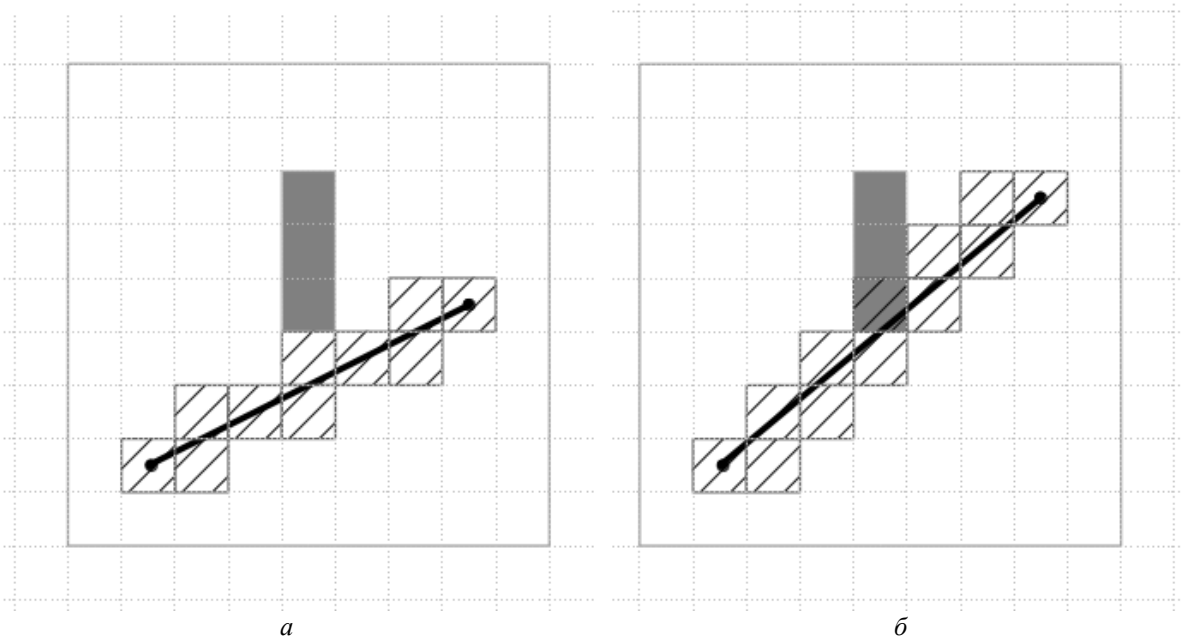
но визначити список наступних потенційних вершин і їх вагу, використовуючи евристичну функцію. При цьому, для кожної нової вершини, що додається в список потенційних вершин, батьківською вершиною може бути будь-яка з попередніх. Після побудови списку потенційних вершин, необхідно вибрати ту, в якої найменша вага. Псевдокод для алгоритму Theta* наведено нижче:

```
function Theta*(start, finish, grid):
    open_nodes = new Set()
    closed_nodes = new Set()
    current_node = create_node(null, start, heuristic(start, finish))
    open_nodes.add(current_node)
    while open_nodes is not empty {
        for each neighbor of current_node.cell:
            if neighbor is not in open_nodes and closed_nodes:
                heuristic = compute_heuristic(neighbor)
                node = create_node(current_node, neighbor, heuristic)
                open_nodes.add(node)
        closed_nodes.add(current_node)
        current_node = pop node with min(gValue + hValue) from open_nodes
        if (current_node.cell == finish):
            return reconstruct_path(current_node)
    return failure

function create_node(previous_node, cell, heuristic):
    chosen_parent = previous_node
    previous_node_parent = previous_node.previous
    if previous_node_parent != null:
        if line_of_sight(cell, previous_node_parent.cell):
            chosen_parent = previous_node_parent
    gValue = chosen_parent.gValue + distance(chosen_parent.cell, cell)
    hValue = heuristic
    return new Node(chosen_parent, cell, gValue, hValue)
```

При цьому, важливим моментом є функція line_of_sight(), яка визначає чи лінія між двома клітинками перетинає перешкоду. Ця функція викликається на кожній ітерації, тому від її ефективності залежить ефективність всього алгоритму Theta*.

Легко помітити, що функція line_of_sight() подібна до малювання лінії на растровому дисплеї. Тобто спочатку можна визначити всі клітинки, через які проходить лінія, а потім для кожної з них перевірити, чи є на ній перешкода. Це схематично зображено на рис. 6.



а *б*

Рис. 6. Визначення існування прямого шляху між двома точками: *а* – між двома точками існує прямий шлях, бо лінія не перетинає перешкоду; *б* – лінія перетинає перешкоду, тому прямого шляху не існує

Також потрібно враховувати, що розмір робота може бути більшим за одну клітинку. В такому випадку необхідно будувати і лінію відповідної товщини.

5. 2. Алгоритм "Гібридний A*"

Далі потрібно реалізувати алгоритм "Гібридний A*" враховуючи те, що від цього алгоритму в більшій частині випадків вимагається пошук шляху між двома точками, які можна сполучити прямою лінією, що не перетинає перешкоди. Ще однією важливою умовою для алгоритму "Гібридний A*" є те, що побудований шлях повинен враховувати не тільки потрібні кінцеві координати робота, а і правильний кут орієнтації.

Однією з головних проблем для реалізації гібридного A* є розробка правильної функції для оцінки ваги стану робота у вузлі, що впливає на вибір наступного вузла і, відповідно необхідних рухів. Простим варіантом цієї функції є такий:

$$cost=g\_value+heuristic+cell\_weight, \quad (1)$$

де g_value – це відстань, яка вже пройдена для досягнення поточного стану; heuristic – це оптимістичний прогноз відстані до цілі (зазвичай це евклідова відстань); cell_weight – вага клітинки, в якій знаходиться робот. Вага клітинки є великою коли ця клітинка вже розглядалась і робот в ній знаходився зі схожим кутом орієнтації.

В такому варіанті cost-функції є один недолік – траєкторії побудовані з її використанням можуть містити багато зайвих поворотів, що в загальному випадку є небажаним ефектом. Цього можна уникнути якщо додавати "штраф" за зміну напряму руху – тобто збільшувати значення cost-функції в такому випадку. Також варто зауважити, що при великій відстані між рухомою платформою і кінцевою точкою за умови відсутності перешкод оптимальною стратегією є рух вздовж прямої, що сполучає початкове і кінцеве положення для платформи. Для досягнення такого ефекту можна в cost-функцію ввести ще два доданки – один буде пропорційний відстані між платформою і прямою, а інший – пропорційний куту між платформою і прямою. При цьому, коли платформа вже знаходиться близько біля цілі, то ці доданки враховувати непотрібно, адже вони будуть заважити правильному паркуванню. Цього можна досягти, якщо зробити вагу зазначених вище доданків пропорційною відстані між поточним положенням платформи і кінцевою точкою.

Враховуючи зазначені вище модифікації для cost-функції, її псевдокод буде виглядати наступним чином:

```
function compute_state_cost(car, movement, previous_state,
                            start, target):
    distance_to_target = distance_between(car, target)
    cell_weight = compute_cell_weight(car)
    move_backward_weight = movement.is_backward() ?
                            BACKWARD_MOVE_WEIGHT : 0
    change_steering_weight = movement != previous_state.movement ?
                            CHANGE_STEERING_WEIGHT : 0
    shift_error = distance_between(car, line(start, target))
    shift_error_weight = SHIFT_ERROR_COEF * distance to target
    orientation_error = |angle_between(car, line(start, target))
    orientation_error_weight =
                ORIENTATION_ERROR_COEF * distance_to_target
    cost = distance_to_target + previous_distance + cell_weight
                + move_backward_weight + change_steering_weight
                + shift_error * shift_error_weight
                + orientation_error * orientation_error_weight
    return cost
```

5. 3. Поєднання алгоритмів Theta* і гібридного A*

Як було зазначено вище, для поєднання цих двох алгоритмів необхідно спочатку запустити Theta*, а потім для кожного відрізку із отриманого шляху запускати гібридний A*. При цьому треба врахувати, що для побудови більш плавної траєкторії, бажано, щоб при досягненні будь-якої з точок шляху Theta* вектор орієнтації робота був направлений у наступну точку. З іншого боку, для гібридного A* найлегшим для виконання варіантом є такий, при якому орієнтація робота співпадає з вектором, який сполучає попередню і поточну точку. Для балансування між цими двома випадками, можна вибрати цільову орієнтацію, яка є середньою між варіантами, зазначеними вище. Цього можна досягнути просто додавши два вектори, як зображено на рис. 7.

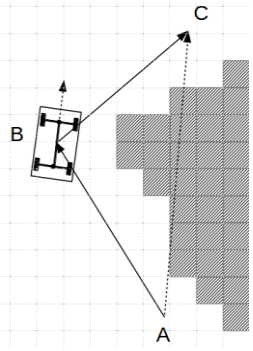


Рис. 7. Визначення необхідного напрямку орієнтації рухомої платформи у проміжній точці

Ще одним важливим моментом є те, що при великій довжині відстані, яку проходить платформа за один крок і високій точності для цільової позиції та орієнтації робота, алгоритм “Гібридний A*” може працювати неефективно, адже існує ймовірність, що платформа буде проїжджати кінцеву позицію, але зупинятись вже на неприйнятній відстані до неї. Ефект від цього можна зменшити, реалізувавши два додаткові кроки при поєднанні зазначених алгоритмів:

1) Для проміжних точок дозволяти більшу похибку для позиції і орієнтації платформи, адже в них цей критерій є значно менш важливим, ніж у кінцевій точці.

2) Накласти обмеження на максимальну кількість ітерацій в алгоритмі “Гібридний A*”. При досягненні максимальної кількості ітерацій потрібно зробити випадковий рух роботом і ще раз запустити “Гібридний A*”. Це дозволить збільшити ймовірність вдалого початкового положення робота.

На рис. 8 наведено високорівневу блок-схему для комплексного алгоритму.

Псевдокод для поєднання цих двох алгоритмів наведено нижче:

```
function resolve_path(start, target):
    movements = new List()
    attempts = 0
    loc_start = start;
    while attempts < MAX_ATTEMPTS_TO_RESTART:
        try:
            theta_star_path = theta_star_path(loc_start, target)
            for (i = 1; i < theta_star_path.size - 1; i++):
                previous = theta_star_path.get(i - 1)
                current = theta_star_path.get(i)
                next = theta_star_path.get(i + 1)
                target_direction = new Vector(previous, current)
                    + new Vector(current, next))
                target_position = current
                n_movements = hybrid_a_star(intermediate args)
                movements.add(n_movements)
                loc_start = update_local_start(movements)
            final_movements = hybrid_a_star(final args)
            movements.add(final_movements)
        catch TooManyIterationsException:
            loc_start = make_random_movement(loc_start)
            attempts++
    return movements
```

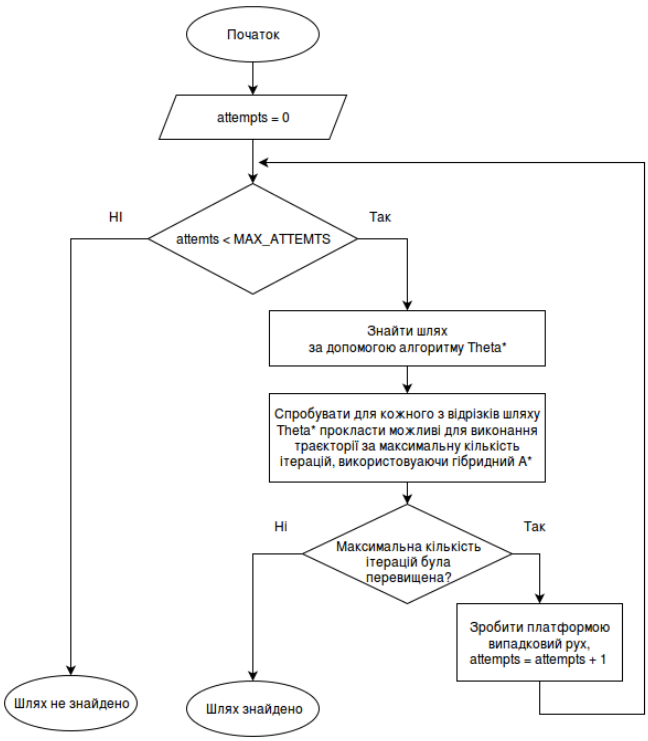


Рис. 8. Блок-схема для комплексного алгоритму.

6. Результати досліджень та їх обговорення

Запропонований вище алгоритм і симулятор чотириколісного робота для тестування роботи алгоритму було реалізовано на мові програмування Java і викладено в публічний доступ [12]. Тестування показало, що алгоритм швидко будує шлях, який є близьким до оптимального, навіть на картах з великою кількістю перешкод і при наявності U-подібних перешкод. Приклад роботи алгоритму для оминання U-подібної перешкоди показано на рис. 9.

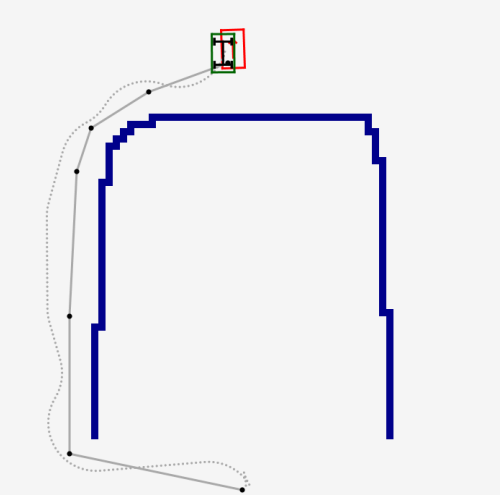


Рис. 9. Результати роботи алгоритму для оминання U-подібної перешкоди. Суцільною лінією зображено шлях, прокладений алгоритмом Theta*; пунктирною – результуючий шлях після поєднання алгоритмів

На рис. 10, 11 зображено роботу алгоритму на більш складній карті.

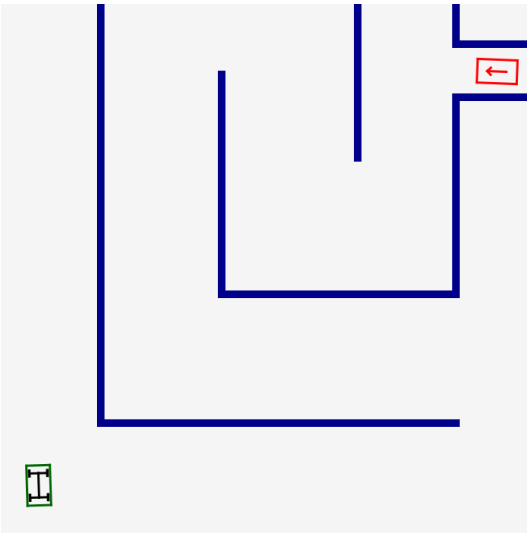


Рис. 10. Початкові умови задачі на більш складній карті

Окрім цього, варто зазначити, що розроблений алгоритм також успішно справляється з задачею паркування. Це зображено на рис. 12, 13.

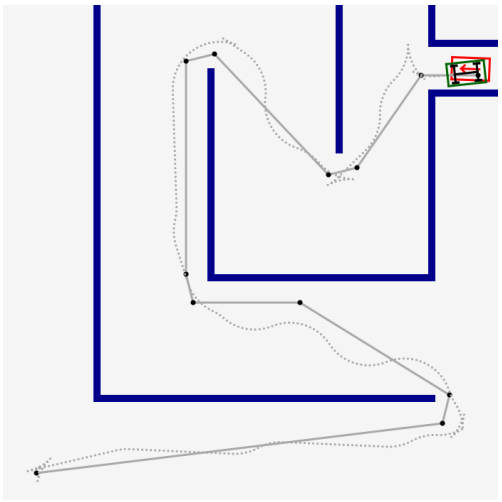


Рис. 11. Результат роботи алгоритму на більш складній карті

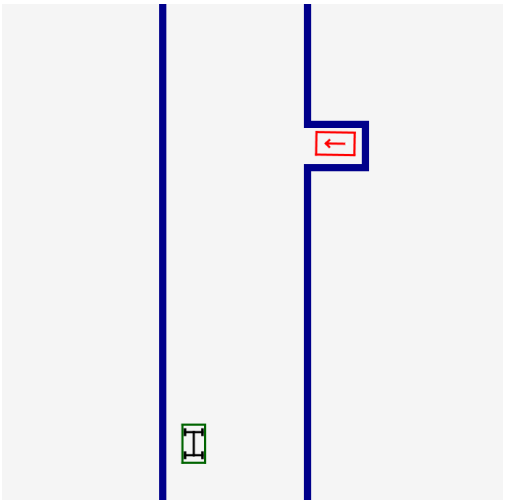


Рис. 12. Початкові умови для задачі паркування

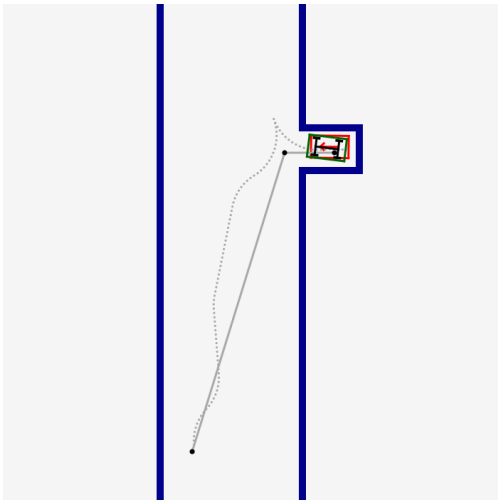


Рис. 13. Результат роботи алгоритму для вирішення задачі паркування

У прикладах були задані такі параметри карти і рухомої платформи (розмірність наведена у пікселях):

- розмір поля: 700×700;
- розмір клітинки (перешкоди): 10×10;
- розмір платформи 30×50;

• відстань, яку проходить платформа за один крок: 20;

• кут відхилення передньої осі при повороті: 25 градусів.

В табл. 1 зведено результати порівняння кількості ітерацій, які необхідні алгоритмам "Гібридний A*" і комплексному алгоритму для пошуку шляху до заданої цілі у наведених вище прикладах.

Таблиця 1

Порівняння ефективності алгоритмів

Назва прикладу	Кількість ітерацій, виконаних алгоритмом "Гібридний A*"	Кількість ітерацій, виконаних комплексним алгоритмом
U-подібна перешкода	285 523	2 270
Складна карта	183 781	5 591
Задача паркування	3 284	1 128

Аналізуючи результати порівняння, можна зробити висновок, що різниця в ефективності між гібридним A* і запропонованим комплексним алгоритмом особливо помітна при наявності великих U-подібних перешкод на шляху до цілі. Також різниця є доволі значною і на складних картах з великою кількістю перешкод.

У всіх трьох прикладах, наведених вище, час роботи комплексного алгоритму на сучасному комп'ютері тривав менше однієї секунди, тобто час його виконання є прийнятним.

Окрім цього, варто зазначити, що у запропонованому алгоритмі виконуються і дві інші поставлені умови для вирішення задачі. Як видно з прикладів, у траєкторії повністю враховується кінематика руху платформи, а побудовані шляхи є близькими до оптимальних.

Внаслідок виконання поставленої задачі, даний алгоритм можна використовувати як основу для прокладання шляху при розробці систем для автоматичного керування безпілотними автомобілями.

7. Висновки

В даній роботі було проаналізовано існуючі алгоритми для пошуку шляху, а також визначено, що їх не можна напряму застосувати для чотириколісних роботів. Тому було запропоновано комплексний алгоритм на основі поєднання Theta* і гібридного A*, який враховує кінематику руху робота, а також дозволяє швидко прокладати шляхи, які є близькими до оптимального. Запропонований алгоритм, а також симулятор чотириколісної рухомої платформи, було реалізовано на мові програмування Java і протестовано. Результати тестування показали, що алгоритм успішно виконує поставлені перед ним задачі, зокрема швидко прокладає шляхи, які є близькі до оптимального навіть при наявності U-подібних перешкод, а також на складних картах з великою кількістю перешкод. Також було показано, що він успішно справляється і з задачею паркування. Внаслідок цього його можна використовувати як основу при проектуванні систем пошуку шляху для безпілотних автомобілів.

Література

1. Dijkstra, E. W. A note on two problems in connexion with graphs [Text] / E. W. Dijkstra // Numerische Mathematik. – 1959. – Vol. 1, Issue 1. – P. 269–271. doi: 10.1007/bf01386390

2. Sniedovich, M. Dijkstra's algorithm revisited: the dynamic programming connexion [Text] / M. Sniedovich // Journal of Control and Cybernetics. – 2006. – Vol. 35, Issue 3. – P. 599–620.

3. Delling, D. Engineering route planning algorithms [Text] / D. Delling, P. Sanders, D. Schultes, D. Wagner // Lecture Notes in Computer Science, 2009. – P. 117–139. doi: 10.1007/978-3-642-02094-0_7

4. Zeng, W. Finding shortest paths on real road networks: the case for A* [Text] / W. Zeng, R. L. Church // International Journal of Geographical Information Science. – 2009. – Vol. 23, Issue 4. – P. 531–543. doi: 10.1080/13658810801949850

5. Russell, S. Artificial Intelligence: A Modern Approach; 3rd ed. [Text] / S. Russell, P. Norvig. – Prentice Hall, 2009. – 1152 p.

6. Theta*: Any-Angle Path Planning for Smoother Trajectories in Continuous Environments [Electronic resource]. – Available at: http://aigamedev.com/open/tutorials/theta-star-any-angle-paths/

7. LaValle, S. M. Rapidly-exploring random trees: A new tool for path planning [Electronic resource] / S. M. LaValle. – Available at: http://msl.cs.uiuc.edu/~lavalle/papers/Lav98c.pdf

8. Practical Search Techniques in Autonomous Driving [Electronic resource]. – Available at: http://ai.stanford.edu/~ddolgov/papers/dolgov_gpp_stair08.pdf

9. Junior: The Stanford Entry in the Urban Challenge [Electronic resource]. – Available at: http://robots.stanford.edu/papers/junior08.pdf

10. rcTek – Ackerman Steering Principle [Electronic resource]. – Available at: http://www.rctek.com/technical/handling/ackerman_steering_principle.html

11. Krause, E. F. Taxicab Geometry [Text] / E. F. Krause. – Dover, 1987. – 96 p.

12. Car simulator and pathfinding algorithm, source code [Electronic resource]. – Available at: https://github.com/vmykh/car-model

References

1. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1 (1), 269–271. doi: 10.1007/bf01386390

2. Sniedovich, M. (2006). Dijkstra's algorithm revisited: the dynamic programming connexion. Journal of Control and Cybernetics, 35 (3), 599–620.

3. Delling, D., Sanders, P., Schultes, D., Wagner, D. (2009). Engineering Route Planning Algorithms. Lecture Notes in Computer Science, 117–139. doi: 10.1007/978-3-642-02094-0_7

4. Zeng, W., Church, R. L. (2009). Finding shortest paths on real road networks: the case for A*. International Journal of Geographical Information Science, 23 (4), 531–543. doi: 10.1080/13658810801949850

5. Russell, S, Norvig, P. (2009). Artificial Intelligence: A Modern Approach. 3rd ed. Prentice Hall, 1152.

6. Theta*: Any-Angle Path Planning for Smoother Trajectories in Continuous Environments. Available at: http://aigamedev.com/open/tutorials/theta-star-any-angle-paths/

7. LaValle, S. M. Rapidly-exploring random trees: A new tool for path planning. Available at: http://msl.cs.uiuc.edu/~lavalle/papers/Lav98c.pdf

8. Practical Search Techniques in Autonomous Driving. Available at: http://ai.stanford.edu/~ddolgov/papers/dolgov_gpp_stair08.pdf

9. Junior: The Stanford Entry in the Urban Challenge. Available at: http://robots.stanford.edu/papers/junior08.pdf

10. rcTek – Ackerman Steering Principle. Available at: http://www.rctek.com/technical/handling/ackerman_steering_principle.html

11. Krause, E. F. (1987). Taxicab Geometry. Dover, 96.

12. Car simulator and path finding algorithm, source code. Available at: https://github.com/vmykh/car-model

Рекомендовано до публікації д-р техн. наук Петренко А. І.
Дата надходження рукопису 14.06.2016

Михалько Віталій Геннадійович, кафедра системного проектування, Національний технічний університет України "Київський політехнічний інститут", пр. Перемоги, 37, м. Київ, Україна, 03056
E-mail: mikhalko.ukr@gmail.com

Круш Ігор Володимирович, кафедра системного проектування, Національний технічний університет України "Київський політехнічний інститут", пр. Перемоги, 37, м. Київ, Україна, 03056
E-mail: ihor@kroosh.me

УДК 624.011
DOI: 10.15587/2313-8416.2016.74484

ВПЛИВ НЕРІВНОМІРНИХ ОСІДАНЬ ОПОР НА НАПРУЖЕНО-ДЕФОРМОВАНИЙ СТАН КАРНИЗНОГО ВУЗЛА ГНУТОКЛЕЄНИХ РАМ

© Д. В. Михайловський, Д. М. Матющенко, А. О. Смоленський

В статті наведено порівняльний аналіз напружень в карнизному вузлі тришарнірних гнутоклеєних рам, що виникають при нерівномірних осіданнях опор. Для дослідження розроблено просторову скінченно-елементну модель будівлі в програмному комплексі ЛІРА-САПР 2013. Осідання були визначені за методом пошарового підсумовування та на основі розрахунку системи «основа-фундамент-надземні конструкції» з використанням фізично-нелінійного багатошарового ґрунтового масиву
Ключові слова: *гнутоклеєна рама, клеєна деревина, карнизний вузол, напружено-деформований стан, метод скінченних елементів*

Comparison analysis of the cornice node's stress of three-hinged curved glulam frames that caused by uneven settlements of bearings are shown in the article. Three-dimensional finite elements model of the building using LIRA-SAPR 2013 was developed for research. Settlements were determined by the stratified method and by calculations of the system "substructures – foundations – constructions" using physically non-linear soil massive
Keywords: *curved glulam frame, glued wood, cornice node, stress-strain state, finite-elements method*

1. Вступ

Великопрольотні конструкції застосовуються в громадських та промислових будівлях. Несучими конструкціями яких є рами, арки, ферми, вантові покриття тощо. Однією з проблем великопрольотних, особливо каркасних будівель, є нерівномірне осідання опор, що пов'язане з неоднорідністю геологічного складу основи (ґрунтів) або наявності в ній прошарків слабких ґрунтів. Інженерно-геологічні вишукування для таких будівель виконуються згідно нормативного документу [1]. В залежності від категорії складності змінюється кількість гірничих виробок та відстань між ними, однак їхнє розташування в межах плями забудови та особливості конструктивного рішення, нормативним документом [1] не визначаються.

Враховуючи вище наведене та той факт, що дані вишукувань недостатньо повно описують дійсний геологічний склад основи майданчика забудови, найбільшою проблемою при проектуванні конструкцій є наявність прошарків слабких ґрунтів в зонах фундаментів під несучі конструкції. Наявність, навіть незначних, прошарків слабких ґрунтів може викликати нерівномірне осідання опор та зміну напружено-деформованого стану конструкцій.