УДК 519.168:004.032.24

М. І. Шахов, О. Г. Байбуз

Дніпровський національний університет імені Олеся Гончара

АНАЛІЗ ТА МОЖЛИВІСТЬ МОДИФІКАЦІЇ НАЯВНИХ АЛГОРИТМІВ ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ НА КВАДРАТНІЙ СІТЦІ З ВИКОРИСТАННЯМ МЕТОДІВ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ

Сформована постановка задачі пошуку оптимального шляху на неперервній місцевості шляхом її дискретизації за допомогою квадратної сітки. Проведений огляд та аналіз найпоширеніших алгоритмів планування шляху на квадратній сітці, які є різновидами алгоритму A*. У якості алгоритму пошуку оптимального шляху на квадратній сітці, що враховує фізичні обмеження безпілотного літального апарату, був обраний алгоритм LIAN. Були виявлені недоліки алгоритму LIAN та можливі шляхи їх вирішення, а також запропоновані варіанти його модифікації з використанням методів паралельного програмування.

Ключові слова: *безпілотний літальний апарат, планування шляху, найкоротший шлях, квадратна сітка, кут повороту, паралельне програмування.*

Сформирована постановка задачи поиска оптимального пути на непрерывной местности путем её дискретизации с помощью квадратной сетки. Проведен обзор и анализ самых распространенных алгоритмов планирования пути на квадратной сетке, которые представляют собой разновидности алгоритма A*. В качестве алгоритма поиска оптимального пути на квадратной сетке, который учитывает физические ограничения беспилотного летательного аппарата, был выбран алгоритм LIAN. Были обнаружены недостатки алгоритма LIAN и возможные пути их решения, а также предложены варианты его модификации с использованием методов параллельного программирования.

Ключевые слова: *беспилотный летательный аппарат, планирование пути, кратчайший путь, квадратная сетка, угол поворота, параллельное программирование.*

The task about unmanned aerial vehicle optimal path planning between two points on continuous terrain, which has obstacles, was set. Advantages and disadvantages of existing methods of continuous terrain discretization, application of which required for solution the task using computer, were

considered. As the most optimal method of continuous terrain discretization the method of square grid was chosen. As selection criteria of the most optimal among existing path planning algorithms on square grid, the alteration angle was accepted. This criteria was suggested because of physical constraints of unmanned aerial vehicle during maneuvers on continuous terrain. According to the chosen criteria, review and analysis of the most widespread path planning algorithms on square grid, which are varieties of A* algorithm, was carrying out, including the short description of the principle of their work and data structures they use. As the most optimal path planning algorithm, which satisfies a given criteria, the LIAN algorithm was chosen. During testing the LIAN implementation in Delphi programming language were discovered disadvantages of this algorithm, and offered possible variants of their solution. Given that proposed and possible further modifications of LIAN will improve qualitative characteristics of founded path, and also will increase its execution time, LIAN algorithm was analyzed on the possibility of its modification using parallel programming methods. Was offer the scheme of work of parallel variant of LIAN algorithm, in which this algorithm will be divided into two parts: parallel part, which will perform integral subtask of the algorithm, and which can be implemented as an instance of one of the parallel threads, and synchronized part, which will be implemented as a main thread. In the context of reliability of software, which will be implement the parallel variant of LIAN algorithm, was determined, to which data structures, which use original LIAN algorithm, can access the synchronized part of new algorithm, and to which can access the parallel part. The specific variants of LIAN algorithm modifications, that use the parallel programming methods, which are planned in the future to implement and research on efficiency and reliability of execution, were offered.

Keywords: *unmanned aerial vehicle, path planning, shortest path, square grid, alteration angle, parallel programming.*

Вступ. У наш час безпілотні літальні апарати (БПЛА) широко використовуються в різних галузях народного господарства, наприклад, для вирішення задач екологічного моніторингу, спостереження за об’єктами транспортної інфраструктури, під час ліквідації наслідків надзвичайних ситуацій. При цьому рух безпілотного літального апарату відбувається у складних умовах рельєфу місцевості, наприклад, у міській місцевості серед будівель, в умовах гірського рельєфу, що унеможливлює реалізацію функції пошуку оптимального шляху безпілотного літального апарату безпосередньо оператором. Тому актуальним напрямком досліджень є задача повної автоматизації планування оптимального маршруту польоту безпілотного літального апарату для обльоту перешкод та досягнення цілі в умовах обмеженого часу та обчислювальних

ресурсів.

Із прогресом навігаційних технологій також зростає рівень автономності безпілотних літальних апаратів. У цьому напрямку вже проведено значну кількість досліджень, зокрема, у сфері пошуку оптимального шляху між двома заданими точками на місцевості. Для таких задач часто використовуються алгоритми планування шляху на основі дискретизації місцевості за допомогою квадратної сітки. Ці алгоритми є досить поширеними та потужними, оскільки вони дозволяють генерувати шлях з оптимальними обчислювальними затратами. Як правило, алгоритми планування шляху на основі квадратної сітки орієнтовані на знаходження найкоротшого шляху та лише розглядають задачу обходу плоских перешкод.

Метою цієї статті є проведення аналітичного огляду наявних алгоритмів пошуку оптимального шляху на квадратній сітці, вибір найбільш оптимального з цих алгоритмів, який враховує фізичні обмеження літального апарату при здійсненні маневрів обльоту перешкод на місцевості, а також детальне дослідження цього алгоритму на предмет можливості його модифікації з використанням методів паралельного програмування.

Постановка задачі пошуку оптимального шляху на квадратній сітці. Неперервну місцевість необхідно дискретизувати для вирішення задачі планування шляху між двома заданими на ній точками за допомогою ЕОМ. Для вирішення задачі підбору оптимального методу дискретизації місцевості були введені наступні спрощення: місцевість представляється як квадратна сітка, а будь-яка точка на місцевості, зокрема ціль або літальний апарат, – як одна клітина в цій квадратній сітці. Клітина може бути або прохідною, тобто на ній може перебувати й через неї може рухатися будь-який об’єкт, або непрохідною, тобто вона взагалі не може бути частиною маршруту. Квадратні сітки як метод дискретизації місцевості широко використовуються та мають такі переваги [1]:

1) сітки є простими структурами даних, зручними для простих алгоритмів планування шляху;

2) місцевість може бути легко дискретизована у квадратну сітку простим накладенням сітки поверх місцевості й позначенням усіх клітин, які частково або повністю заблоковані;

3) сітки забезпечують всебічну картину всіх прохідних поверхонь у неперервній місцевості; цей аспект визначальний, коли алгоритм планування шляху використовується в динамічному середовищі й повинен взаємодіяти із системою навігації;

4) клітини сітки можуть зберігати додаткову інформацію щодо їхньої прохідності;

5) інформація, що зберігається у клітинах, характеризується швидким доступом, оскільки сітки є структурами даних із довільним доступом;

6) точність шляху та навігації може бути покращена збільшенням роздільної здатності сітки.

Існують також деякі альтернативні методи дискретизації місцевості, які припускають для спрощення задачі, що перешкоди на місцевості мають форму багатокутників:

1. Діаграми Вороного [2] дискретизують місцевість за допомогою зміщення шляхів від заблокованих багатокутників, тому результуючі шляхи можуть бути значно довшими, ніж справжні найкоротші шляхи.

2. Обрамлені квадрати [3] рекурсивно поділяють місцевість на чотири клітини однакового розміру доти, поки всі клітини не стануть повністю заблокованими, не заблокованими або досить малого розміру.

3. Ймовірнісні дорожні карти [4] або швидко досліджувані випадкові дерева, у яких вершини розташовуються непередбачено (на додаток до стартової та цільової вершин). Дві вершини з'єднуються прямою лінією, якщо мають пряму видимість. Випадкове розміщення вершин потрібно ретельно налаштувати, тому що це впливає на час виконання алгоритму планування шляху, ймовірність його виявлення та довжину.

4. Графи видимості [5] використовують кути кожного блокуючого багатокутника як вершини (на додаток до стартової та цільової вершин). Дві вершини з'єднуються за допомогою прямої лінії, якщо вони мають пряму видимість, що дозволяє алгоритму знайти істинні найкоротші шляхи. Час виконання алгоритму планування шляху може зростати нелінійно по відношенню до кількості вершин, так само й кількість ребер графа видимості може зростати квадратично відносно кількості вершин.

На сьогоднішній день найбільшого поширення набули два альтернативні типи нотацій квадратних сіток: 1) зосереджені на центрі клітини (місцезнаходження БПЛА прив'язане до центрів клітин сітки); 2) на кутках клітин (місцезнаходження БПЛА прив'язане до кутів клітин). Задача планування шляху формулюється наступним чином: знайти неблокований шлях із заданої стартової вершини в задану цільову вершину. Шлях називається неблокованим, якщо кожна

вершина, що йому належить, має пряму видимість із сусідніми вершинами шляху.

Наявні алгоритми планування шляху на квадратній сітці. У цьому розділі проводиться огляд та аналіз найпоширеніших алгоритмів планування шляху на квадратній сітці, які є різновидами популярного алгоритму A* [6]. Алгоритм A* використовує для своєї роботи три значення для кожної вершини *s*:

1) *g(s)* – довжина найкоротшого шляху від стартової вершини до поточної вершини *s*;

2) *h(s)* – оцінка довжини шляху до цільової вершини з вершини *s*. Алгоритм A* використовує її для обчислення величини *f(s) = g(s) + h(s)*, яка є оцінкою довжини найкоротшого шляху від стартової вершини через вершину *s* до цільової вершини;

3) *parent(s)* – вказівник на батьківську вершину, що використовується для отримання шляху від стартової вершини до цільової після завершення роботи алгоритму A*.

Алгоритм A* також обслуговує дві глобальні структури даних:

1) *OPEN* – список, що містить вершини, від яких будуть розгалужуватися інші вершини.

2) *CLOSED* – список, що містить вершини, від яких вже були розгалужені інші вершини. Він гарантує, що A* розгалужує кожну вершину лише один раз.

Коли алгоритм A* вперше зустрічає вершину, він встановлює так зване *g*-значення цієї вершини рівним додатній нескінченності та вказівник на батьківську клітину рівним NULL. Алгоритм A* встановлює *g*-значення стартової вершини рівним нулю та вказівник на батьківську клітину рівним самій стартовій вершині. Списки *OPEN* та *CLOSED* спочатку встановлюються пустими, а потім у список *OPEN* вставляється стартова вершина з *f*-значенням у якості її ключа. Потім алгоритм A* повторно виконує наступну процедуру: якщо список *OPEN* пустий, то він сповіщає, що шляху не існує. Інакше він ідентифікує вершину *s* з найменшим *f*-значенням у списку *OPEN*. Якщо ця вершина є цільовою, то алгоритм A* сповіщає про те, що він знайшов шлях. Формування шляху здійснюється завдяки слідуванню вказівникам на батьківські клітини від цільової вершини до стартової. Інакше алгоритм A* вилучає вершину зі списку *OPEN*, вставляє у список *CLOSED*, а потім генерує кожного з її видимих сусідів наступним чином. Алгоритм A* перевіряє, чи є *g*-значення вершини *s* плюс довжина прямої лінії від *s* до вершини *s′* меншим, ніж *g*-значення вершини *s′*. Якщо так, тоді він встановлює *g*-значення вершини *s′*

рівним сумі *g*-значення вершини *s* та довжини прямої лінії з *s* до *s′*, встановлює вказівник на батьківську вершину *s′*,рівним *s*, і нарешті вставляє вершину *s′* у список *OPEN* у разі, якщо вона раніше не була вставлена в цей список. Тобто A* оновлює *g*-значення та вказівник на батьківську вершину для вершини *s′*, якщо ця модифікація призводить до коротшого шляху від стартової вершини до *s′*, ніж той шлях до *s′*, який був знайдений до цього моменту. Потім алгоритм повторює цю процедуру.

Існує модифікація алгоритму A* для згладжування знайдених шляхів – A* PS (A* post-smoothing) [7]. Алгоритм A* PS запускає A* на квадратній сітці, а потім згладжує результуючий шлях. Приймемо, що алгоритм A* знайшов шлях $[s_0, s_1, \ldots, s_n]$, де $s_0 = s_{start}$ та $s_n = s_{goal}$. Алгоритм A* PS використовує першу вершину шляху у якості поточної вершини. Потім він перевіряє, чи має поточна вершина s_0 пряму видимість із наслідником s_2 її наслідника s_1. Якщо так, то алгоритм A* PS вилучає проміжну вершину s_1 із шляху, таким чином скорочуючи його. Потім A* PS повторює цю процедуру повторною перевіркою того, чи має поточна вершина s_0 пряму видимість з наслідником s_3 її наслідника s_2, і так далі. Як тільки пряма видимість зникає, алгоритм A* PS змінює поточну вершину на її наслідника та повторює цю процедуру доти, доки вона не досягне кінця шляху.

У статті [1] описується концепція планування шляху з будь-яким кутом повороту та пропонуються два нових коректних алгоритми, що реалізують цю концепцію – Basic Theta* та Angle-Propagation Theta*. Вони є варіантами алгоритму A*, що поширюють інформацію вздовж ребер сітки (для досягнення меншого часу виконання) без прив'язки шляхів до ребер сітки. На відміну від варіанту алгоритму A* з графами видимості, вони не гарантують знаходження істинних найкоротших шляхів.

Алгоритми Theta* поєднують ідеї алгоритму A* для графів видимості та алгоритму A* на квадратній сітці. Шляхи, знайдені цими алгоритмами, лише трохи довші, ніж істинні найкоротші шляхи, що знаходяться алгоритмом A* для графів видимості, його час виконання дещо повільніший, ніж час виконання алгоритму A* на квадратних сітках. Ключова різниця між алгоритмами Theta* та A* на сітках полягає в тому, що у Theta* батьківською вершиною для даної вершини може бути будь-яка, тоді як у A* батьківською вершиною повинна бути лише сусідня.

Basic Theta* є ідентичний алгоритму A*, за винятком того, що коли він оновлює *g*-значення та вказівник на батьківську клітину нового

видимого сусіда s' вершини s, він розглядає два шляхи замість одного у A*:

• Шлях 1: Basic Theta* розглядає шлях зі стартової вершини до вершини s, та від s до вершини s', (як у A*).

• Шлях 2: Basic Theta* також розглядає шлях із стартової вершини до вершини, що є батьківською для вершини s, та від цієї батьківської вершини до вершини s'. Шлях 2 дозволяє алгоритму Basic Theta* знаходити шляхи з будь-яким кутом повороту.

Шлях 2 не може бути довшим, ніж Шлях 1 через трикутну нерівність – довжина будь-якої сторони трикутника не може бути довшою, ніж сума довжин інших двох сторін. Шлях 1 гарантовано є неблокованим, але Шлях 2 цього не гарантує. Тому Basic Theta* обирає Шлях 2, якщо вершина s' має пряму видимість із батьківською вершиною вершини s, інакше Basic Theta* обирає Шлях 1. Basic Theta* оновлює g-значення та вказівник на батьківську вершину вершини s', якщо обраний шлях є коротшим, ніж найкоротший шлях зі стартової вершини до вершини s', який був знайдений до цього моменту.

Basic Theta* не є оптимальним (не гарантує знаходження істинних найкоротших шляхів), тому що батьківська вершина поточної вершини повинна бути або видимим сусідом цієї вершини, або батьківською вершиною видимого сусіда, що не є завжди у випадку істинних найкоротших шляхів. Час виконання Basic Theta* під час генерації нових видимих сусідів може бути лінійно залежним від кількості клітин сітки, оскільки час виконання кожної перевірки прямої видимості може лінійно залежати від кількості клітин.

Angle-Propagation Theta* (AP Theta*) скорочує час виконання Basic Theta* за такт розширення вершини з лінійного до постійного. Ключовою різницею між AP Theta* та Basic Theta* є те, що AP Theta* поширює діапазон кута, та використовує його для визначення прямої видимості між двома вершинами. Якщо існує джерело світла біля вершини s і світло не може пройти через заблоковані клітини, то клітини в тіні не мають прямої видимості з вершиною s, тоді як усі інші клітини мають пряму видимість із s. Кожний суміжний регіон точок, що має пряму видимість із вершиноюs, може характеризуватися двома променями, що випромінюються з s, тобто кутовим діапазоном, що визначається двома кутовими межами. AP Theta* обчислює кутовий діапазон вершини s при її розширенні, а потім поширює його вздовж ребер сітки, що дає в результаті постійний час виконання за такт розширення вершини s, оскільки кутові діапазони можуть поширюватися за постійний час, а перевірки прямої видимості також

можуть виконуватися за постійний час.

Алгоритми для планування шляху з обмеженням кута повороту. Програмне забезпечення більшості систем керування враховує обмеження на кут повороту при плануванні шляху літального апарату. Кут повороту – це фізична величина, що враховує динамічні характеристики БПЛА як об'єкта керування та характеризує поворот тіла навколо нерухомої відносно тіла осі, або поворот променя, який виходить із центра обертання тіла відносно іншого променя, що вважається нерухомим. Оскільки проаналізовані вище алгоритми планування шляху на квадратній сітці зазвичай не розглядають це обмеження, вони можуть не лише створювати нереалістичні шляхи, але також наражати літальний апарат на небезпеку. Якщо обмеження на кут повороту не розглядається, алгоритм обльоту перешкод може не функціонувати коректно.

У статті [8] представлена модифікація алгоритму Basic Theta*, пристосована для вирішення проблеми планування шляху на квадратній сітці для об'єкта керування з обмеженнями на кут повороту. Автори цієї статті не розглядали безпосередньо обмеження на максимальний кут повороту. Замість цього вони досліджували випадки, коли задані швидкість і радіус повороту об'єкта керування та обчислювали обмеження на кут повороту в реальному часі, беручи до уваги довжину відрізків пройденого шляху. Але якщо замінити оригінальну процедуру обчислення обмеження кута повороту тією, яка завжди повертає α_m (максимальний кут повороту), Basic Theta* стає придатним для вирішення проблем пошуку шляху з обмеженням кута повороту. Автори статті [9] назвали такий алгоритм Theta*-LA (LA – скорочення від "limited angle").

Коли алгоритм Theta* намагається з'єднати клітину зі своєю прабатьківською клітиною (для того, щоб вилучити проміжний елемент, тобто батьківську клітину, зі шляху) він перевіряє лише обмеження прямої видимості між клітинами, тоді як Theta*-LA перевіряє також обмеження на кут повороту, і якщо кут між відрізками, що сполучають три послідовні клітини – прабатьківську клітину, батьківську клітину та поточну клітину – більший, ніж задана величина α_m, то батьківська клітина залишається в послідовності. Але така модифікація призводить до наступної проблеми: якщо обмеження на кут повороту менше, ніж 45° (що є найбільш реалістичним перебігом подій), то алгоритм дає збій при обході великих перешкод і через це не знаходить шлях з обмеженим кутом повороту.

Головна причина того, що алгоритм Theta*-LA не в змозі знайти

шлях у більшості випадків полягає в тому, що він не зберігає проміжні елементи шляху, а намагається зробити відрізки шляху якомога довшими. Автори статті [8] пропонують часткове вирішення цієї проблеми – зважування сітки, тобто присвоєння кожній клітині сітки невід'ємного значення вагового коефіцієнта, та беруть до уваги значення ваг клітин при обчисленні довжини відрізка. Використання ваг для штрафування клітин, що знаходяться поблизу перешкод, покращує загальну продуктивність алгоритму Theta*-LA.

Для формування алгоритму планування шляху як частини системи керування БПЛА за основу був обраний алгоритм LIAN, оскільки він дозволяє знаходити шлях з обмеженням на кут повороту, що враховує фізичні обмеження літального апарату при здійсненні маневрів обльоту перешкод на місцевості.

LIAN (з англ. «Limited angle») – це евристичний алгоритм пошуку найкоротшого шляху між двома точками на деякій квадратній сітці, що містить перешкоди, який враховує обмеження на кут повороту [9]. Шлях у цьому алгоритмі являє собою послідовність клітин квадратної сітки, не обов'язково сусідніх. Кожна з клітин такого шляху характеризується значенням g – довжиною шляху (що має обмеження на кут повороту) від стартової клітини s до даної клітини a. Також кожна клітина має вказівник на батьківську клітину $bp(a)$, що вказує на клітину в сітці, яка є попередником a у послідовності клітин шляху.

Алгоритм LIAN протягом своєї роботи зберігає в оперативній пам'яті два списки клітин: *OPEN* та *CLOSED*. Список *OPEN* – це набір клітин, що являють собою потенційних кандидатів для подальшої обробки, який спочатку містить клітину s. Список *CLOSED* є набором клітин, які вже були оброблені. На кожному кроці клітина a з мінімальним значенням $f(a)$ вилучається зі списку *OPEN*, де $f(a) = g(a) + h(a)$, $h(a)$ – евристична оцінка довжини шляху від поточної клітини a до цільової клітини g. Потім формуються потенційні наступники клітини a у вигляді списку *SUCC*. До цього списку входять клітини, що розташовані на відстані Δ від поточної клітини a. Для знаходження таких клітин використовується алгоритм Midpoint. Якщо відстань від a до g менша ніж Δ, то цільова клітина також входить до списку *SUCC*. Потім зі списку *SUCC* видаляються клітини, які не є прохідними, порушують обмеження на кут повороту або вже містяться у списку *CLOSED*. Далі для клітин зі списку *SUCC* обчислюються значення $g(succ_i) = g(a) + dist(a, succ_i)$, де $dist(c_1, c_2)$ – відстань між клітинами c_1 іc_2, після чого ці клітини додаються до списку *OPEN*, а клітина a – до списку *CLOSED*.

Алгоритм LIAN завершує своє виконання, коли цільова клітина вилучається зі списку *OPEN*, але якщо список *OPEN* стає пустим під час пошуку, алгоритм завершує своє виконання оповіщенням про те, що шлях, який відповідає заданим обмеженням на кут повороту, не був знайдений.

Алгоритм LIAN має наступні властивості:

1) LIAN завжди завершує свою роботу;

2) якщо рішення задачі планування шляху з обмеженням кута повороту існує, то LIAN знайде це рішення, а якщо ні, LIAN сповістить про невдалу спробу знайти шлях;

3) якщо існують різні рішення задачі планування шляху з обмеженням кута повороту, LIAN поверне найкоротший шлях.

У статті [9] алгоритм LIAN був досліджений експериментально в різних моделях вуличних навігаційний сценаріїв, і він значно перевершував наявні аналоги: вирішував більше задач планування шляху з обмеженням кута повороту, ніж інші алгоритми, при цьому використовував менше оперативної пам’яті та процесорного часу.

Можливості модифікації алгоритму LIAN з використанням методів паралельного програмування. У ході тестування реалізації алгоритму LIAN мовою програмування Delphi 7 були виявлені наступні недоліки:

1) після завершення роботи LIAN знайдений шлях потребує згладжування (рис. 1), оскільки в ході роботи алгоритму не перевіряється можливість з’єднання поточної клітини з прабатьківською для вилучення проміжної батьківської клітини; таке вилучення проміжної клітини можливе лише за дотримання умов прямої видимості між сусідніми клітинами в ланцюжку клітин шляху та обмеження на кут повороту;

2) отриманий шлях може проходити дуже близько біля перешкод, що становить небезпеку для польоту БПЛА, оскільки через низку причин, наприклад, помилки в роботі навігаційної системи або пориви вітру літальний апарат може зіштовхнутися з перешкодою (рис. 2).

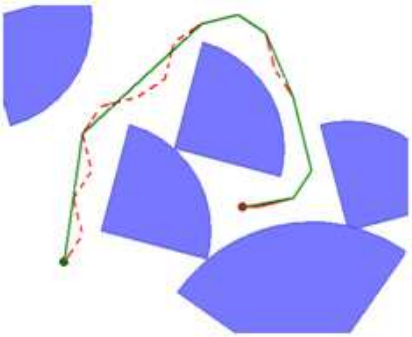

Рисунок 1 – Шлях, знайдений за допомогою алгоритму LIAN (пунктир), та його згладжена модифікація (суцільна лінія)

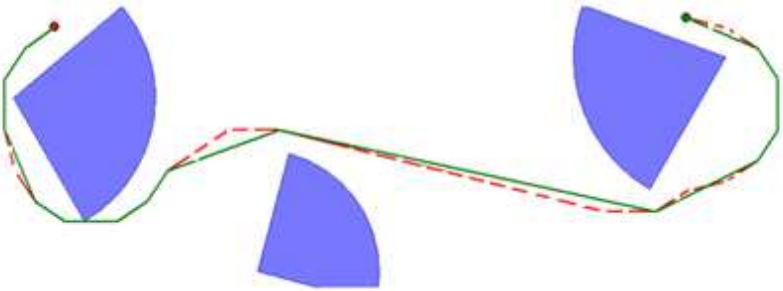

Рисунок 2 – Проходження шляху, знайденого за допомогою алгоритму LIAN, дуже близько біля перешкод

Для вирішення першого недоліку після роботи алгоритму LIAN можна провести згладжування знайденого шляху, наприклад, за допомогою алгоритму A* PS [7]. Причина другого недоліку алгоритму LIAN полягає в реалізації його складового алгоритму *LineOfSight* – перевірки ознаки прямої видимості між двома клітинами сітки. Ця перевірка проводиться таким чином: за допомогою алгоритму Брезенхема будується дискретний відрізок прямої з клітин, які лежать на цій прямій та з'єднують дві задані клітини. Потім кожна клітина відрізку перевіряється на прохідність, і якщо хоча б одна з цих клітин є непрохідною, то дві задані клітини, які є кінцями цього відрізка, не

мають прямої видимості. Дві клітини квадратної сітки мають пряму видимість між собою тоді й тільки тоді, коли дискретний відрізок, що сполучає їх, містить лише прохідні клітини. Недоліком такого алгоритму перевірки прямої видимості є те, що дві клітини можуть мати пряму видимість тоді, коли сам дискретний відрізок, що їх сполучає, може проходити дуже близько біля перешкод.

Модернізувати алгоритм LineOfSight можна наступним чином. Введемо в алгоритм LIAN новий вхідний параметр – *nearRadius*, який буде за величиною кратний лінійному розміру клітини сітки та позначатиме мінімальну відстань між будь-якою клітиною, що належить найкоротшому шляху, та будь-якою клітиною на сітці, яка є непрохідною. З'єднаємо дві клітини, що будуть перевірятися на пряму видимість, дискретним відрізком. Позначимо через *V* множину клітин, що належать цьому дискретному відрізку. Побудуємо за допомогою алгоритму *Midpoint* дискретні кола радіусом *nearRadius*, центри яких будуть належати клітинам із множини *V*. Позначимо через *U* множину клітин, що належать цим дискретним колам. Дві задані клітини будуть мати пряму видимість між собою, та можуть бути з'єднані відрізком, який належить найкоротшому шляху, кожна клітина якого знаходиться від перешкод на відстані, не меншій, ніж *nearRadius*, тільки за умови, що всі клітини з множини *U* будуть прохідними (рис. 3).

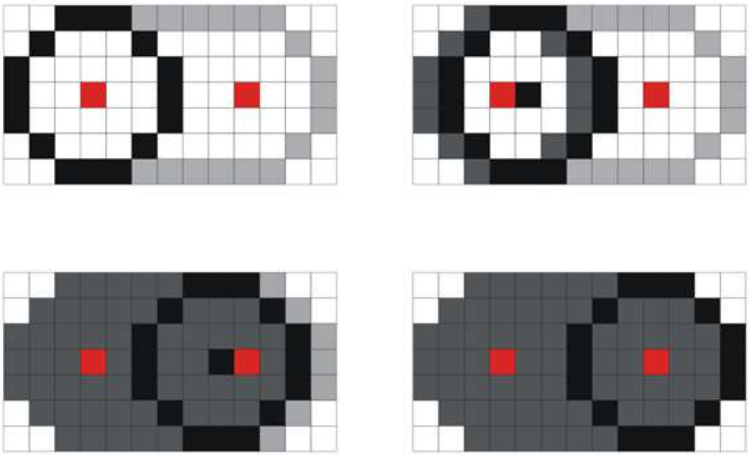

Рисунок 3 – Алгоритм роботи модернізованої процедури перевірки прямої видимості між двома клітинами

Позначимо через *count(U)* кількість клітин у множині *U*, а через *count(V)* – у множині *V*. З точки зору обчислювальної складності, модифікована версія алгоритму *LineOfSight* є більш обчислювально складнішою, ніж стара версія цього ж алгоритму, оскільки в ній необхідно перевірити у *count(U)/count(V)* разів більшу кількість клітин на прохідність, до того ж, сама процедура побудови дискретного кола радіусом *nearRadius* за допомогою алгоритму *Midpoint* у новій версії алгоритму викликається *count(V)* разів.

З урахуванням того, що запропоновані та подальші модифікації алгоритму LIAN будуть покращувати якісні характеристики знайденого найкоротшого шляху, але також будуть збільшувати час його виконання, покращити продуктивність LIAN можна з використанням методології паралельного програмування. Для цього в алгоритмі LIAN необхідно виділити деякий тип підзадач, кожна з яких може виконуватися паралельно та незалежно від інших, та дві точки синхронізації цих підзадач, які будуть розміщуватися в головному потоці виконання. У першій точці буде відбуватися запуск на виконання всіх таких підзадач, а у другій точці головний потік буде чекати завершення всіх цих підзадач для того, щоб зібрати та обробити результати їхньої роботи та знову запустити на виконання в наступній ітерації алгоритму. Тобто алгоритм LIAN необхідно розділити на дві частини: синхронізувальну частину, у якій будуть відбуватися дії, що стосуються всієї задачі, та паралельну частину, у якій виконуватимуться складові підзадачі алгоритму та яка може бути реалізована у вигляді кількох потоків, що будуть виконуватися паралельно на відповідній кількості процесорів. Схема роботи такого варіанту алгоритму LIAN показана на рис. 4.

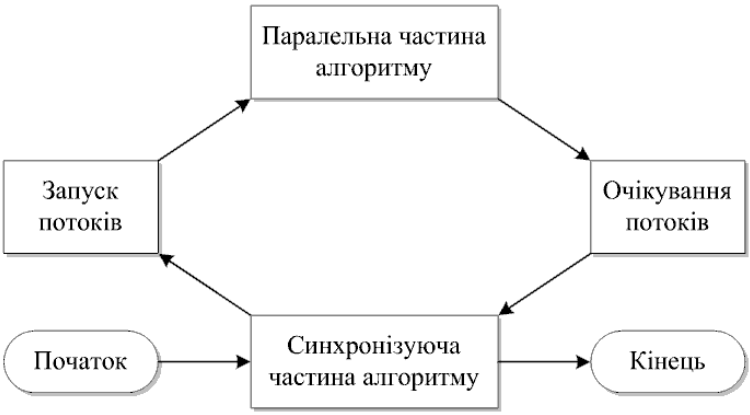

Рисунок 4 – Запропонована схема роботи паралельного варіанту алгоритму LIAN

Також необхідно визначити, до яких структур даних може отримувати доступ синхронізувальна частина алгоритму, а до яких – паралельна. Від якості та повноти вирішення задачі організації доступу кількох потоків виконання до однієї й тієї ж структури даних залежить надійність роботи програмного забезпечення, що буде реалізувати цей алгоритм, адже неконтрольовані та непослідовні операції доступу й модифікації даних, що сумісно використовуються кількома потоками, можуть призвести до збоїв у роботі алгоритму. Для забезпечення надійності роботи паралельного варіанту алгоритму LIAN, синхронізувальній частині можна надати доступ до всіх структур даних, у тому числі й до тих, які належать паралельній частині. У свою чергу, паралельній частині алгоритму повинні бути доступні тільки власні структури даних, причому кожен екземпляр паралельної задачі повинен мати доступ саме до власного набору даних, а не до наборів даних інших потоків – реалізацій паралельної частини. Таким чином, синхронізувальна частина буде організовувати роботу екземплярів паралельної частини та використовувати результати їхньої роботи, тобто буде синхронізувати роботу всіх дочірніх потоків, виконуючи операції, які потребують блокування роботи всіх паралельних потоків та модифікації глобальних структур даних, які існують у єдиному екземплярі й акумулюють результати роботи паралельної частини алгоритму.

У паралельному варіанті алгоритму LIAN списки *OPEN* та *CLOSED* повинні зберігатися в єдиному екземплярі, тому доступ до них повинна мати тільки синхронізувальна частина задачі. У свою чергу, списки *SUCC* належать до вершин, які генерують своїх наступників, тобто кожна з таких вершин може мати власний список *SUCC*. Тому цей список може виступати у якості структури даних, що належить екземпляру паралельної частини алгоритму, яка буде виконувати генерацію наступників вершин зі списку *OPEN* та відкидання тих вершин-наступників, що порушують умови формування шляху з обмеженням кута повороту. Головним джерелом конфліктів між різними екземплярами паралельної частини алгоритму є можливість потрапляння однієї й тієї самої клітини квадратної сітки до різних списків *SUCC* різних потоків, що може стати причиною неоднозначності визначення її вказівника на батьківську клітину. Тому головна вимога до паралельного використання списків *SUCC* формується таким чином: у будь-який момент часу множина клітин усіх списків *SUCC* повинна містити лише унікальні клітини сітки.

Ефективність програмної реалізації алгоритму LIAN, що використовуватиме методи паралельного програмування, залежить від співвідношення часу виконання паралельної та синхронізувальної частин алгоритму. Чим більша частка виконання паралельної частини алгоритму від загального часу виконання, тим більш ефективною буде паралельна реалізація алгоритму LIAN. Тому в подальшому планується реалізувати й дослідити на предмет ефективності та надійності виконання наступні варіанти організації паралельних обчислень в алгоритмі LIAN:

1. У синхронізувальній частині алгоритму: при досягненні деякого граничного значення кількості клітин N список *OPEN* ділиться на два окремих списки, кожен з яких, у свою чергу, також при досягненні того ж граничного значення кількості клітин N ділиться навпіл. Усі ці списки вершин $OPEN_i$ будуть належати різним потокам виконання. У кінці синхронізувальна частина алгоритму для кожного зі списків $OPEN_i$ обирає клітину a_i з мінімальним значенням $f(a_i)$, вилучає її з відповідного списку $OPEN_i$, формує наступників клітини a_i у вигляді списку $SUCC_i$, додає клітину a_i у список *CLOSED* та відкидає зі списку $SUCC_i$ ті клітини, що вже містяться у списку *CLOSED*. У паралельній частині алгоритму: видалення зі сформованих списків $SUCC_i$ клітин, які не є прохідними та порушують обмеження на кут повороту.

2. У синхронізувальній частині алгоритму: вибірка перших N клітин зі списку *OPEN* з мінімальними значеннями $f(a_i)$, які мають відстань між собою не меншу, ніж $2 \cdot \Delta + \varepsilon$, де ε – довжина сторін клітин сітки; подальше формування списків $SUCC_i$ цих клітин та відкидання з множини клітин цих списків тих, що вже містяться у списку *CLOSED*. У паралельній частині алгоритму: видалення зі сформованих списків $SUCC_i$ клітин, які не є прохідними та порушують обмеження на кут повороту.

3. У синхронізувальній частині алгоритму: поділ клітин списку *OPEN* на декілька списків $OPEN_i$, кожен із яких буде належати до свого паралельного потоку виконання, та які будуть представляти собою так звані «фронти» списку *OPEN*. Далі кожен із фронтів обирає клітину a_i з мінімальним значенням $f(a_i)$, вилучає її з відповідного списку $OPEN_i$, формує наступників клітини a_i у вигляді списків $SUCC_i$, кожен із яких містить унікальні клітини, додає клітину a_i у список *CLOSED* та відкидає з множини клітин списків $SUCC_i$ ті, що вже містяться у списку *CLOSED*. У паралельній частині алгоритму: видалення зі сформованих списків $SUCC_i$ клітин, які не є прохідними та порушують обмеження на кут повороту.

Висновки. У статті був проведений огляд та аналіз наявних алгоритмів планування шляху на місцевості як частини системи керування безпілотного літального апарату. У якості моделі дискретизації неперервної місцевості була обрана квадратна сітка.

Були розглянуті та проаналізовані найпоширеніші алгоритми планування шляху на квадратній сітці, що є похідними від алгоритму A*. Серед них був обраний алгоритм LIAN, оскільки він дозволяє генерувати згладжені шляхи з обмеженням кута повороту, що враховує фізичні обмеження літального апарату при здійсненні маневрів обльоту перешкод на місцевості. Були проаналізовані недоліки алгоритму LIAN та засоби їхнього усунення. Також LIAN був проаналізований на можливість його модифікації з використанням методів паралельного програмування та були запропоновані конкретні варіанти організації в ньому паралельних обчислень.

Модифікація алгоритму знаходження найкоротших шляхів на місцевості LIAN, яка використовує методи паралельного програмування, дозволить підвищити швидкодію та продуктивність роботи системи керування БПЛА за рахунок організації паралельних обчислень на багатоядерних процесорах.

Бібліографічні посилання

1. Nash A., Daniel K., Koenig S., Felner, A. Theta*: Any-Angle Path Planning on Grids // Journal of Artificial Intelligence Research. 2010. N 39. P. 533–579.

2. Aurenhammer, F. Voronoi diagrams – a survey of a fundamental geometric data structure // ACM Computing Surveys. 1991. Vol. 23, Issue 3. P. 345–405.

3. Yahja A., Stentz A., Singh S., Brumitt B. Framed-quadtree path planning for mobile robots operating in sparse environments // In Proceedings of the International Conference on Robotics and Automation. 20-20 May 1998. P. 650–655.

4. Choset H., Lynch K., Hutchinson S., Kantor G., Burgard W., Kavraki L., Thrun S. Principles of Robot Motion: Theory, Algorithms, and Implementations // MIT Press. 2005.

5. Lozano-Pérez T., Wesley M. An algorithm for planning collision-free paths among polyhedral obstacles // Communication of the ACM. 1979. Vol. 22., N.10. P. 560–570.

6. Hart P. E., Nilsson N. J., Raphael B. A formal basis for the heuristic determination of minimum cost paths // IEEE Transactions on Systems Science and Cybernetics. 1968. P. 100–107.

7. Thorpe C. Path relaxation: Path planning for a mobile robot // In Proceedings of the AAAI Conference on Artificial Intelligence. 1984. P. 318–321.

8. Kim H., Kim D., Shin J. U., Kim H., Myung H. Angular rate-constrained path planning algorithm for unmanned surface vehicles // Ocean Engineering. 2014. Vol. 84. P. 37–44.

9. Yakovlev K., Baskin E., Hramoin I. Grid-Based Angle-Constrained Path Planning // KI 2015: Advances in Artificial Intelligence. Lecture Notes in Computer Science. Springer International Publishing, Switzerland, 2015. P. 208–221.

Надійшла до редколегії 23.10.18