Міністерство освіти і науки України
Національний аерокосмічний університет
ім. М. Є. Жуковського «Харківський авіаційний інститут»

Кваліфікаційна наукова
праця на правах рукопису

ЛЕЙЧЕНКО КИРИЛО МИКОЛАЙОВИЧ

УДК 629.7.014-519.051.5:621.396.49

ДИСЕРТАЦІЯ

МЕТОДИ ТА ЗАСОБИ ПЛАНУВАННЯ РОЗГОРТАННЯ ЛІТАЮЧИХ МЕРЕЖ ДЛЯ ЗАБЕЗПЕЧЕННЯ ПЕРЕДАЧІ ДАНИХ В УМОВАХ РУЙНУВАНЬ

Спеціальність 123 Комп’ютерна інженерія
Галузь знань 12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ К. М. Лейченко
(підпис)

Науковий керівник Фесенко Герман Вікторович,
доктор технічних наук, професор

Харків – 2024

АНОТАЦІЯ

Лейченко Кирило Миколайович. Методи та засоби планування розгортання літаючих мереж для забезпечення передачі даних в умовах руйнувань. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 123 Комп'ютерна інженерія. – Національний аерокосмічний університет імені М.Є. Жуковського «Харківський авіаційний інститут», Харків, 2024.

Дисертаційна робота присвячена розробленню методів і програмних засобів підтримки розгортання безпілотних літаючих LiFi мереж для забезпечення передачі даних в умовах руйнувань з урахуванням вимог до часових і надійнісних характеристик.

Об'єктом дослідження є процеси розгортання літаючих мереж для забезпечення передачі даних в умовах руйнувань.

Вперше запропоновано методи планування розміщення безпілотних літальних апаратів (БПЛА) літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань, які на відміну від відомих враховують структурно-просторові параметри перешкод і базуються на комбінуванні алгоритмів прокладання маршрутів за різними стратегіями їх обходу, та надають змогу мінімізувати довжину маршруту та/або кількість необхідних БПЛА.

Удосконалено метод розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань шляхом формування та вибору маршрутів польоту у визначені точки у просторі для утворення мережі з урахуванням різних варіантів базування БПЛА, що зменшує сумарні часові та вартісні витрати на розгортання мережі.

Удосконалено метод підвищення надійності літаючої LiFi мережі шляхом розроблення моделі та алгоритмів випереджувальної заміни БПЛА з урахуванням вимог до ймовірності безвідмовної роботи мережі, показників безвідмовності та

автономності окремих БПЛА, що забезпечує гарантоване функціонування мережі впродовж заданого часу.

Ключові слова: безпілотний літальний апарат, моніторинг, надійність, безвідмовність, математична модель, відеопотік, бездротова сенсорна мережа, Інтернет речей, система підтримки прийняття рішень, LiFi технологія, передача даних, руйнування, обхід перешкод, розгортання, безпека.

Список публікацій здобувача:

1. Pevnev V., Plakhteev A., Tsuranov M., Zemlianko H., Leichenko K. “Smart City” Technology: Conception, Security Issues and Cases. *Lecture Notes in Networks and Systems*. 2022. Vol. 367. P. 207–218. DOI: 10.1007/978-3-030-94259-5_19 (закордонне періодичне наукове видання, Scopus, Q4).

2. Leichenko K., Fesenko H., Kharchenko V. Deploying the Reliable UAV Swarm for Providing P2P LiFi Communications Considering Physical Obstacles: Method of Rectangles, Algorithms, and Tool. *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS’2023)* : Proc. 12th IEEE Int. Conf., Dortmund, Germany, Sept. 07–09, 2023. P. 1011–1016. DOI: 10.1109/IDAACS58523.2023.10348819 (стаття у працях конференції, Scopus).

3. Leichenko K., Fesenko H., Borges J., Kharchenko V. Search for the Shortest Route Considering Physical Obstacles: Method of Controlled Waterfall, Tool, and Application. *Dependable Systems, Services and Technologies (DESSERT’2023)* : Proc. 13th IEEE Int. Conf., Athens, Greece, Oct. 13–15, 2023. P. 1–5. DOI: 10.1109/DESSERT61349.2023.10416479 (стаття у працях конференції, Scopus).

4. Лейченко К. М., Фесенко Г. В. Програмний засіб підтримки планування розгортання LiFi мережі на основі БПЛА для забезпечення передачі даних в умовах руйнувань. *Системи управління, навігації та зв’язку*. 2024. Вип. 1 (75). С. 193–200. DOI: 10.26906/SUNZ.2024.1.193 (наукове фахове видання категорії Б).

5. Leichenko K., Fesenko H., Kharchenko V., Illiashenko O. Deployment of a UAV swarm-based LiFi network in the obstacle-ridden environment: algorithms of finding the path for UAV placement. *Radioelectronic and Computer Systems*. 2023. No.

1(109). P. 176–195. DOI: 10.32620/reks.2024.1.14 (наукове фахове видання категорії A, Scopus, Q3).

6. Лейченко К., Фесенко Г., Харченко В. Стратегії розгортання та методи забезпечення надійності рою БПЛА для утворення LiFi мережі. *Вимірювальна та обчислювальна техніка в технологічних процесах*. 2024. №1. С. 21–31. DOI: 10.31891/2219-9365-2024-77-3 (наукове фахове видання категорії Б).

ANNOTATION

Leichenko Kyrylo. Methods and tools for planning the deployment of flying networks to ensure data transmission in conditions of devastation. – Manuscript copyright.

Thesis on competition of scientific degree of Doctor of Philosophy by specialty 123 Computer Engineering. – National Aerospace University "Kharkiv Aviation Institute", Kharkiv, 2024.

The thesis is devoted to the development of methods and software tools to support the deployment of unmanned flying LiFi networks to ensure data transmission in conditions of devastation, taking into account the requirements for time and reliability characteristics.

The object of the study is the processes of deploying flying networks to ensure data transmission in conditions of devastation.

For the first time, methods for planning the placement of unmanned aerial vehicles (UAVs) of a flying LiFi network to ensure data transmission in conditions of destruction are proposed, which, unlike the known ones, take into account the structural and spatial parameters of obstacles and are based on a combination of route planning algorithms using different strategies for their bypass, and make it possible to minimize the length of the route and/or the number of required UAVs.

The method of placing UAVs in a flying LiFi network to ensure data transmission in conditions of devastation has been improved by forming and selecting flight routes to specific points in space to form a network, taking into account various options for UAV basing, which reduces the total time and cost of network deployment.

The method of increasing the reliability of a flying LiFi network has been improved by developing a model and algorithms for proactive replacement of UAVs, taking into account the requirements for the probability of network uptime, reliability and autonomy of individual UAVs, which ensures guaranteed network operation for a given time.

Keywords: unmanned aerial vehicle, monitoring, reliability, dependability, mathematical model, video stream, wireless sensor network, Internet of Things, decision

support system, LiFi technology, data transmission, devastation, obstacle avoidance, deployment, safety.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БПЛА	–	Безпілотний літальний апарат
ЙБР	–	Ймовірність безвідмовної роботи
КІ	–	Критична інфраструктура
КЦ	–	Кризовий центр
CLI	–	Command Line Interface
FANET	–	Flying Ad-Hoc Network
GPS	–	Global Positioning System
GUI	–	Graphical User Interface

ВСТУП

Обґрунтування вибору теми дослідження. У разі аварій на об’єктах критичної інфраструктури персоналу кризового центру (КЦ) важливо отримувати інформацію як про основні параметри технологічного обладнання, так і про рівень забруднення робочого середовища виробничих приміщень. Однак внаслідок аварії штатне комунікаційне обладнання може бути пошкоджене і зазначена вище важлива інформація перестане надходити до КЦ, тим самим значно ускладнюючи прийняття його персоналом ефективних управлінських рішень щодо реагування на аварію. У цьому випадку для поновлення надходження інформації до КЦ може бути розгорнута літаюча LiFi мережа на основі безпілотних літальних апаратів (БПЛА).

Використання БПЛА для утворення LiFi мережі обумовлено тим, що вони забезпечують гнучкість LiFi мережі у сценаріях розгортання, особливо тих, що передбачають розгортання у різних середовищах, де традиційний радіочастотний зв’язок може бути проблематичним, наприклад, у виробничих приміщеннях з руйнуваннями. Така універсальність дозволяє цій мережі виконувати широкий спектр завдань у різноманітних операційних умовах.

Літаюча LiFi мережа є стійкою до електромагнітних перешкод від інших електронних пристроїв або радіочастотних сигналів і може запропонувати значно вищу швидкість безпечної передачі даних порівняно з традиційними радіочастотними технологіями зв’язку. Така властивість набуває особливого значення для сценаріїв, які передбачають швидке передавання великих обсягів даних (потокове відео високої чіткості, передача даних з датчиків в режимі реального часу) або широкомасштабне відеоспостереження.

Разом з тим, необхідно мати на увазі, що світлові сигнали можуть бути обмежені перешкодами у виробничих приміщеннях з руйнуваннями. Це потребуватиме прокладання раціональних маршрутів розповсюдження LiFi сигналу в обхід цих перешкод, визначення місць розташування на цих маршрутах

БПЛА і забезпечення гарантованого функціонування літаючої LiFi мережі впродовж заданого часу.

Таким чином, актуальною науково-прикладною задачею є розроблення методів і програмних засобів підтримки розгортання безпілотних літаючих LiFi мереж для забезпечення передачі даних в умовах руйнувань з урахуванням вимог до часових і надійнісних характеристик.

Об'єкт дослідження – процеси розгортання літаючих мереж для забезпечення передачі даних в умовах руйнувань.

Предмет дослідження – моделі, методи та засоби планування розгортання літаючої LiFi мережі на основі БПЛА.

Мета і завдання дослідження. Метою дослідження є забезпечення надійного функціонування та зменшення часу розгортання літаючої LiFi мережі шляхом розроблення і впровадження методів і програмних засобів для системи підтримки планування та розміщення БПЛА в умовах руйнувань.

Для досягнення мети дослідження необхідно вирішити наступні завдання:

– проаналізувати існуючі методи планування розгортання бездротових мереж в умовах руйнувань;

– розробити методи планування розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань;

– розробити методи розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань;

– розробити метод підвищення надійності літаючої LiFi мережі з урахуванням вимог до безвідмовності та автономності БПЛА;

– розробити програмні засоби для системи підтримки планування розгортання БПЛА літаючої LiFi мережі;

– впровадити розроблені методи і засоби в навчальному процесі, наукових проектах та індустрії.

Методи дослідження. У дисертаційній роботі використовувались методи системного аналізу, оптимізації, математичного моделювання, теорії надійності, теорії графів, теорії розкладів.

Наукова новизна отриманих результатів:

– **вперше запропоновано** *методи* планування розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань, які на відміну від відомих враховують структурно-просторові параметри перешкод і базуються на комбінуванні алгоритмів прокладання маршрутів за різними стратегіями їх обходу, та надають змогу мінімізувати довжину маршруту та/або кількість необхідних БПЛА;

– **удосконалено** *метод* розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань шляхом формування та вибору маршрутів польоту у визначені точки у просторі для утворення мережі з урахуванням різних варіантів базування БПЛА, що зменшує сумарні часові та вартісні витрати на розгортання мережі;

– **удосконалено** *метод* підвищення надійності літаючої LiFi мережі шляхом розроблення моделі та алгоритмів випереджувальної заміни БПЛА з урахуванням вимог до ймовірності безвідмовної роботи мережі, показників безвідмовності та автономності окремих БПЛА, що забезпечує гарантоване функціонування мережі впродовж заданого часу.

Особистий внесок здобувача полягає у розроблені методів і програмних засобів, які забезпечують вирішення поставлених задач, описаних вище. Всі основні результати отримані автором особисто та опубліковано у роботах [1]–[6].

У працях, які опубліковані у співавторстві, автору належать: результати аналізу основних підходів щодо передачі інформації між її джерелами та споживачами в межах «розумного міста» [1]; програмний засіб для забезпечення надійного LiFi зв’язку з використанням рою БПЛА для передачі даних між двома точками в умовах механічних перешкод, спричинених руйнуваннями, стратегії розробки алгоритмів обходу перешкод з урахуванням обмежень на дальність та надійність зв’язку [2]; метод та програмний засіб прокладання маршруту розповсюдження LiFi сигналу у приміщенні з механічними перешкодами з використанням алгоритму керованого водоспаду [3]; програмний засіб підтримки планування розгортання LiFi мережі на основі БПЛА для забезпечення передачі

даних в умовах руйнувань [4]; методи обходу перешкод з використанням алгоритму лівого та правих кутів, а також алгоритму керованого водоспаду, програмний засіб для пошуку раціонального розміщення БПЛА для забезпечення необхідних характеристик LiFi мережі [5]; стратегії розгортання рою БПЛА із стаціонарного депо для утворення літаючої LiFi мережі в приміщенні з перешкодами з використанням заздалегідь прокладеного маршруту розповсюдження LiFi сигналу, метод забезпечення безперебійного функціонування літаючої LiFi мережі із необхідним рівнем надійності протягом заданого часу у приміщенні з перешкодами шляхом використанням двох змін рою БПЛА [6].

Апробація матеріалів дисертації. Основні положення та ідеї дисертаційної роботи доповідалися та обговорювалися на: науково-технічному семінарі «Гарантоздатні Інформаційні Технології» (ГІТ) кафедри комп'ютерних систем, мереж і кібербезпеки Національного аерокосмічного університету ім. М. Є. Жуковського «ХАІ» (Харків, 04 жовтня 2023 р.); міжнародному науково-технічному семінарі «Критичні комп'ютерні технології та системи» (КриКТехС) (Харків, 02 квітня 2024 р.); міжнародній конференції «Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications» (IDAACS) (Дортмунд, Німеччина, 2023); міжнародній конференції «Dependable Systems, Services and Technologies» (DESSERT) (Афіни, Греція, 2023).

Зв'язок з науковими програмами, темами. Дисертаційна робота виконана у Національному аерокосмічному університеті ім. М. Є. Жуковського «Харківський авіаційний інститут» відповідно з державними програмами та планами НДР:

– НДР «Методи, програмно-апаратні засоби та інформаційні технології розроблення і модернізації гарантоздатних комп'ютерних систем, мереж та IT-інфраструктур» (ДР № 0117U005349, 2018-2020);

– НДР «Методологічні засади та технології оцінювання та забезпечення безпеки (захисту) критичних інформаційних інфраструктур» (ДР № 0119U100979, 2019-2021);

– НДР «Методи, моделі та інформаційні технології підвищення надійності та безпечності складних ІТ-систем на етапах розроблення та впровадження» (Д/Р № 0121U113842, 2021-2023);

– НДР «Наукові засади і методи забезпечення гарантоздатності флотів БПЛА інтелектуальних систем моніторингу потенційно небезпечних і військових об’єктів» (ДР № 0121U112172, 2021-2023).

Роль автора у зазначених НДР, в яких автор був безпосереднім виконавцем, полягає у розробці методів та засобів планування розгортання БПЛА літаючої LiFi мережі.

Практичне значення отриманих результатів. Практичні результати полягають у доведенні теоретичних положень дисертаційної роботи до конкретних алгоритмів та програмних засобів для підтримки планування розгортання літаючої LiFi мережі. Результати дисертаційної роботи впроваджено (додаток А):

– у товаристві з обмеженою відповідальністю «СІДІ ЛІНК» (акт впровадження від 29 березня 2024) під час розроблення проєктів комп’ютерних мереж різного призначення;

– у навчальному процесі Національного аерокосмічного університету ім. М. Є. Жуковського «Харківський авіаційний інститут» (акт впровадження від 01 квітня 2024);

– при виконанні науково-дослідних проєктів, що виконувались у Національному аерокосмічному університеті ім. М. Є. Жуковського «Харківський авіаційний інститут» (акт впровадження від 01 квітня 2024).

Структура та обсяг дисертації. Дисертація складається із вступу, чотирьох розділів, висновку, списку виконаних джерел і додатків. Загальний обсяг дисертації складає 183 сторінки, з яких анотація двома мовами на 5 сторінках, зміст на 3 сторінках, перелік умовних позначень на 1 сторінці, основний текст на 105 сторінках, список використаних джерел із 69 найменувань на 9 сторінках, додатки на 59 сторінках. Робота містить 9 таблиць та 54 рисунки.

Публікації. За темою дисертаційної роботи було опубліковано 6 наукових праць, серед яких: 2 статті у наукових фахових виданнях України категорії Б; 1

стаття у англомовному науковому фаховому виданні України категорії А, проіндексованому у базі даних Scopus (квартиль Q3); 1 стаття у закордонному періодичному науковому виданні, проіндексованому у базі даних Scopus (квартиль Q4); 2 публікації в працях міжнародних конференції, проіндексованих у базі даних Scopus.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ РОЗГОРТАННЯ БЕЗДРОТОВИХ МЕРЕЖ В УМОВАХ РУЙНУВАНЬ. ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз застосування БПЛА для передачі даних в приміщеннях

Останнім часом все більше досліджень пов’язано з використання БПЛА для передачі даних в приміщеннях. Наприклад, автори [7] розглядають можливість використання БПЛА в приміщеннях з перешкодами, фокусуючись на проблемах з навігацією БПЛА, обумовлених динамічністю БПЛА, обмеженим ресурсом батареї часом роботи тощо. Для виявлення перешкод та подальшого їх обходу ці автори пропонують використовувати сенсорні технології.

Робота [8] присвячена дослідженню можливості використання БПЛА для збору даних в приміщеннях з метою підвищення ефективності гасіння пожеж.

Робота [9] зосереджена на розробці та оптимізації багатороторного БПЛА, спеціально розробленого для проведення пошуково-рятувальних місій у приміщеннях, де сигнал Global Positioning System (GPS) недоступний і присутня велика кількість перешкод. Конструкція включає в себе легку структуру рами, щоб збільшити корисне навантаження БПЛА. Це необхідно, оскільки для успішного виконання цілей місії в середовищі, де немає GPS, БПЛА потребує численних датчиків розпізнавання та уникнення перешкод, а також присутність теплових та оптичні камер.

Метою дослідження [10] є розробка автоматичної системи моніторингу навколишнього середовища на основі БПЛА, оснащених спеціальними датчиками, для виявлення швидких змін в екологічних характеристиках рослин, що ростуть в теплицях.

У статті [11] запропоновано систему моніторингу та реагування на надзвичайні ситуації в будівлях за допомогою маячків і БПЛА і використання технологій Інтернету речей. Система моніторингу забезпечує безпечний зв’язок між БПЛА, інтелектуальними датчиками, сервером управління та додатком для

смартфонів для менеджерів з безпеки, а також забезпечує координацію між інтелектуальними датчиками та внутрішніми/зовнішніми БПЛА для розширення покриття моніторингом приміщень будівлі в реальному часі.

Автори [12] відзначають, що забезпечення польотної місії БПЛА у з гарантуванням відвідування кожного визначеного місця і дотриманням плани інспекцій у будівлях є доволі проблематичним внаслідок складних планувань будівель, жорстких вимог щодо безпеки польотів, і обмеженого ресурсу батареї БПЛА. Для розв'язання цієї проблеми автори запропонували чотиривимірні (4D) інформаційні моделі будівель для автоматичної розробки оптимальних планів польотів БПЛА. Вона автоматично визначає цілі інспекції на основі користувацького багатокритеріального опису цілей.

У статті [13] представлено вдосконалений алгоритм оптимізації на основі навчання для керування роєм БПЛА з метою моніторингу якості повітря в приміщенні в режимі реального часу та пошуку можливих джерел забруднення. Покращення алгоритму оптимізації на основі навчання автори бачать за рахунок таких дій: пропозиція змінного коефіцієнта навчання для підвищення швидкості і точності пошуку, створення моделі індивідуальних можливостей для інтеграції реального стану БПЛА з алгоритмом, збільшення кількості вчителів і надання оновлених формул для підвищення швидкості збіжності алгоритму, а також пропозиція стратегії групування для досягнення моніторингу якості повітря в приміщеннях.

У статті [14] ми представляємо концепцію моніторингу якості повітря в приміщеннях, наприклад, для охорони здоров'я та безпеки (промислових) робочих місць, яка передбачає застосування нано-БПЛА на базі квадрокоптера Crazyflie 2.0, і невеликих легких газових датчиків з оксиду металу для вимірювання загальної кількості летких органічних сполук у проміле і оцінки концентрації eCO2 (еквівалент розрахованого вуглекислого газу) в проміле. Для оцінки абсолютної 3D-позиції рою планується використовувати систему локалізації та позиціонування в приміщенні, подібну до GPS.

Робота [15] пропонує систему управління запасами на складах на основі БПЛА, яка складається з двох основних частин – інтерфейсу користувача та планувальника шляху. Використовуючи створений людино-машинний інтерфейс, оператор може визначити робочий простір і завдання перевірки. Модуль планування шляху використовує як вхідні дані визначення завдання перевірки, визначене оператором, і планує оптимізований шлях без зіткнень. Оператор може візуалізувати сформовану місію, запустити її, спостерігати за ходом виконання місії в реальному часі та отримувати результати місії. Функціональність системи перевірена моделюванням, а також тестовими польотами в середовищі, що імітує склад, за допомогою реального БПЛА під назвою AV Discovery, створеного компанією Airvolute s.r.o.

У статті [16] представлено алгоритм планування траєкторії руху БПЛА в реальному часі в приміщеннях під час виконання завдань контактної інспекції. Єдиними вхідними даними, які використовує цей алгоритм, є хмара точок будівлі, де планується застосовувати БПЛА буде застосуватись. Алгоритм поділяється на дві основні частини. Перша відповідає за попередню обробку хмари точок, а друга прокладає маршрут руху БПЛА.

Таким чином, проаналізовані роботи показують, що існує достатньо широкий спектр застосування БПЛА у приміщеннях:

- моніторинг якості повітря приміщень та пошук джерел забруднення;
- управління запасами на складах;
- проведення контактного та безконтактного інспектування приміщень;
- гасіння пожеж у приміщеннях;
- проведення пошуково-рятувальних робіт.

Разом з тим, автори відзначають проблеми, з якими доводиться стикатися під час застосування БПЛА у приміщеннях:

- наявність у приміщеннях середовищ з відсутністю GPS сигналів;
- наявність у приміщеннях механічних перешкод;
- специфічне планування окремих будівель, що ускладнює рух у них БПЛА;

– необхідність розгортання додаткової інфраструктури для забезпечення польотів всередині приміщень.

1.2 Аналіз існуючих методів, алгоритмів та програмних засобів розгортання літаючих бездротових мереж в умовах руйнувань

Автори [17] пропонують підходи по розгортанню системи БПЛА у складних умовах в приміщенні при відсутності можливості використання GPS для внутрішніх інспекцій трубопроводів та котлів. Запропоновано систему, яка складається з апаратного опису та інтеграції двох БПЛА, двоетапного методу одночасної локалізації та картографування для локалізації БПЛА та тривимірного картографування навколишнього середовища, алгоритму планування траєкторії з гарантованою безпекою для огляду та збору даних, а також методу генерації траєкторії з врахуванням перешкод.

У дослідженні [18] представлено підхід до локалізації БПЛА в приміщенні з використанням квазінатягнутого троса. Зазначається, що підхід може зменшити похибку сучасної локалізації БПЛА в приміщенні на основі тросів на 31,12%. Оскільки БПЛА локалізується відносно центру котушки з тросом, метод може бути використаний для локалізації БПЛА в рухомому кадрі. Тому він підходить для взаємної локалізації в сумчастих гетерогенних робототехнічних командах для пошуково-рятувальних робіт у міській забудові.

Робота [19] розглядає аналітичні моделі кооперативної системи зв’язку супутник-БПЛА, де БПЛА виступають у якості повітряних ретрансляторів з використанням оптичних технологій вільного простору.

Автори [20] присвячують дослідження проблемі функціонування фізичного рівня в системах зв’язку супутник-БПЛА з трьох точок зору: оцінка каналу, що змінюється в часі, високомобільний прийомопередавач і високомобільний множинний доступ. Представлено та розглянуто відповідні ключові технології, існуючі роботи, відкриті проблеми та подальші напрямки розвитку.

Робота [21] описує розробку та порівняння двох алгоритмів планування руху прив’язаного БПЛА з можливістю та неможливістю контакту з середовищем. Автори пропонують дві схеми планування руху, які зменшують досяжний простір конфігурації з урахуванням прив’язки і навмисно планують (і послаблюють) точки контакту прив’язки з навколишнім середовищем і забезпечують еквівалентний досяжний простір конфігурації, як і у неприв’язаного аналога. Автори наголошують, що завдяки їх підходам прив’язані літальні апарати можуть знайти своє застосування в обмеженому і захараченому середовищі з перешкодами на противагу ідеальному вільному простору, зберігаючи при цьому переваги від використання прив’язки.

Автори [22] пропонують набір алгоритмів планування траєкторії польоту БПЛА в приміщенні, оптимізованих для складних умов. По-перше, запропоновано алгоритм оптимізованого сіткового моделювання повітряного простору в приміщенні на основі сітки поділу географічних координат з одновимірним інтегральним кодуванням на 2^n-дереві в 3 вимірах для зменшення обчислювальної складності моделювання тривимірного повітряного простору в приміщенні за допомогою просторових підрозділів.

Автори роботи [23] надають огляд пропозицій щодо архітектури обходу перешкод та БПЛА. Зазвичай розглядаються різні підходи, включаючи підходи на основі сенсорів, планування та управління. Також обговорюються різні мережеві архітектури, такі як однорангові, централізовані та гібридні мережі. Автори доходять висновку, що уникнення перешкод і мережева архітектура є двома найважливішими аспектами розробки надійних і ефективних БПЛА. Вони зазначають, що існує багато різних підходів до обходу перешкод і мережевої архітектури, і вибір відповідного підходу залежить від конкретного застосування БПЛА.

Останнім часом все більше уваги приділяється вирішенню задач, пов’язаних з обходом перешкод і плануванням маршруту для наземних і повітряних роботів. Наприклад, автори роботи [24] застосовують алгоритм тріангуляції Делоне та вдосконалений алгоритм A* для аналізу складних перешкод та генерації точок

Вороного як пріоритетних вузлів пошуку шляху для підвищення ефективності планування траєкторії мобільних роботів. Перш за все, автори будують трикутну сітку на основі набору перешкод і вузлових точок. Потім кутові точки трансформуються у граф, який використовується в удосконаленому алгоритмі A* для пошуку найкоротшого шляху між початковою та кінцевою точками. Перш за все, слід зазначити, що алгоритм є стійким до змін і простим у реалізації. Однак, алгоритм може бути менш ефективним у простих середовищах, може втрачати ефективність у динамічних середовищах, де зміни відбуваються швидко, а також бути обчислювально дорогим для великих середовищ.

У роботах [25], [26] представлено новий підхід до динамічного планування шляху для мобільних роботів, заснований на гібридному рішенні мурашиної колонії та динамічних вікон. Алгоритм мурашиної колонії – це натхненний біологією алгоритм пошуку, який імітує поведінку мурах, що шукають їжу. Алгоритм динамічного вікна – це алгоритм планування шляху в реальному часі, який враховує поточний стан робота та динамічні зміни в навколишньому середовищі. Запропонований підхід поєднує переваги алгоритму мурашиної колонії та алгоритму динамічних вікон для створення більш ефективного та надійного алгоритму планування маршруту в обхід перешкод. Алгоритм мурашиної колонії використовується для пошуку глобального шляху до мети, а алгоритм динамічних вікон генерує локальний маршрут, яким робот може повітряний або наземний робот може рухатись в реальному часі. Автори пропонують можливі застосування алгоритму, такі як навігація на складах, автономне водіння, роботи-кур’єри та роботи-розвідники.

У [27] запропоновано метод, заснований на глибокому навчанні, алгоритмі трасування променів, правилі очікування та швидкодосліджуваному випадковому дереві. Запропонований підхід використовує нейронну мережу для навчання моделі, яка може передбачати безпечні маршрути для кімнатних роботів. Нейронна мережа навчається на наборі даних, що складається з зображень навколишнього середовища та інформації про перешкоди. Автори розглядають алгоритм для використання в офісній навігації, навігації в лікарнях тощо.

У статті [28] описано новий підхід до обходу перешкод за допомогою алгоритму диференціальної еволюції Флойда-Уоршалла. Він дозволяє здійснювати пошук найкоротших шляхів між усіма парами точок на графі. Граф представляє середовище, в якому БПЛА повинен уникати перешкод. Підхід працює наступним чином: спочатку будується граф середовища за допомогою датчиків БПЛА. Після цього використовується еволюційний алгоритм диференціальної еволюції Флойда-Уоршалла для пошуку найкоротшого маршруту від поточного положення БПЛА до цілі в обхід перешкод.

У дослідженні [29] розглядається питання проєктування траєкторії квадрокоптера за найкоротший час через серію визначених маршрутних точок, використовуючи всі переваги динаміки квадрокоптера. Спочатку будується граф оточення, використовуючи дані з датчиків квадрокоптера. Потім використовується алгоритм A* для пошуку найкоротшого маршруту від поточного положення квадрокоптера до цілі в обхід перешкод.

Автори роботи [30] пропонують підхід щодо планування маршруту польоту БПЛА, який з’єднує початкову точку з кінцевою в закритому просторі, в обхід перешкод. Автори пропонують створювати оптимальний маршрут за допомогою використання алгоритмів A* та підйому на пагорб із запізненням.

В роботі [31] розглянуті питання планування маршрутів БПЛА в приміщенні. Дослідження демонструють розробку вдосконаленого методу планування імовірнісних дорожніх карт для безпечних польотів у приміщеннях на основі припущення про квадрокоптерну модель БПЛА. Експериментальна частина показала, що метод забезпечує безпечні польоти БПЛА в приміщенні, значно підвищуючи при цьому ефективність обчислень.

Для максимізації інформації, зібраної мультикоптерами під час огляду великих будівель, в роботі [32] розглядається задача пошуку наближено оптимального шляху, що проходить через серію бажаних точок огляду в тривимірному середовищі з перешкодами. Запропонований метод складається з двох етапів: глобального та локального планування маршруту. При глобальному плануванні маршрут від початкової до кінцевої точки в обхід перешкод

прокладається приблизно. Це робиться за допомогою графового методу планування маршруту. На етапі локального планування планується більш детальний маршрут з урахуванням динамічних змін у навколишньому середовищі та обмежень мультикоптера. Для цього використовується метод планування маршруту на основі потенційного поля.

Набирають обертів дослідження щодо використання БПЛА для забезпечення LiFi зв'язку всередині і зовні приміщень. У дослідженні [33] представлено розробку інтерфейсного протоколу для протоколу маршрутизації, унікального для внутрішніх літаючих спеціальних мереж (FANET), які використовують LiFi як канал зв'язку. FANET (Flying Ad-Hoc Network) – це мережа БПЛА, які взаємодіють без використання базових станцій. FANET можна використовувати для різних цілей, таких як моніторинг навколишнього середовища, доставка посилок, пошук і порятунок.

В [34] розглядається система моніторингу погоди на основі БПЛА, де БПЛА передають зібрану інформацію на наземну станцію за допомогою технології LiFi.

Для захисту інформації та даних у FANET в [35] оцінюється можливість використання LiFi з декількома БПЛА для спільної роботи в приміщенні та кооперативних мережах.

У роботах [36]–[38] розглядається можливість використання LiFi, а також гібридної технології WiFi/LiFi в проблемі подолання обмежень трафіку. При описі теорії LiFi, представлені останні дослідження в галузі.

Автори [39] пропонують нову системну концепцію для LiFi в промислових бездротових додатках, де використовується розподілена багатокористувацька архітектура з багатьма входами і багатьма виходами, що забезпечує безперебійну мобільність, надійний зв'язок з низькою затримкою та інтеграцію з позиціонуванням і 5G.

Стаття [40] досліджує гіпотезу про те, що методи зв'язку LiFi, засновані на глибокому навчанні, можуть ефективно вивчати особливості внутрішнього середовища і поведінку користувачів, щоб забезпечити кращу продуктивність у

порівнянні зі звичайними методами оцінки каналу, особливо коли доступ до інформації про стан каналу в реальному часі обмежений.

Автори роботи [41] пропонують для внутрішніх систем LiFi нові реалістичні моделі каналів, засновані на вимірюваннях. Зокрема, отримано статистику коефіцієнта підсилення каналу для випадку випадково орієнтованих стаціонарних і мобільних приймачів LiFi. На основі отриманих моделей було досліджено вплив випадкової орієнтації та просторового розподілу користувачів LiFi, де показано, що вищезгадані фактори можуть сильно впливати на коефіцієнт підсилення каналу та продуктивність системи.

У стаття [42] досліджуються змінні в часі характеристики каналу мобільного LiFi на основі виміряних даних. Отриманий вираз швидкості показує, що мобільний зв'язок LiFi можливий при використанні щонайменше двох фотодіодів з різною орієнтацією. Автори наголошують, що запропоновані схеми оцінки та відстеження каналу є ефективними при проектуванні мобільних систем LiFi.

Автори [43] описують системну архітектуру для LiFi та 5G в інтелектуальному виробництві та розглядають переваги застосування LiFi та 5G в інтелектуальному виробництві.

Робота [44] розглядає технологію LiFi для використання у побудові мережі за допомогою рою БПЛА. Автори презентують аналітичне виведення середньої ймовірності помилки блоку у вигляді наближення Чебишева, нижньої та верхньої меж. Однак автори залишають поза увагою питання планування маршруту та власне розгортання літаючої БПЛА LiFi мережі в умовах перешкод.

Стаття [45] пропонує метод прогнозування трафіку для мереж БПЛА з урахування помилок підключення до точок доступу. В дослідженні використовується стохастична еволюційна теорія ігор для аналітичного розв'язання зміни пропускної здатності для кожної ймовірності помилки з'єднання.

Стаття [46] представляє два функціональних тестових стенди для перевірки точності передачі світла в реальному часі. Автори оцінюють використання технології LiFi на різних відстанях і при різному кутовому розміщенні освітлювальних пристроїв і кінцевих приймачів з локальним підключенням і

підключенням до Інтернету. Автори наголошують, що відстань і кутове положення не впливають на передачу даних LiFi як при завантаженні, так і при завантаженні на різні відстані.

Робота [47] розглядає питання надійної LiFi мережі в рамках просторових випадкових розташувань терміналів у трьохвимірному просторі. Проводяться дослідження аналізу продуктивності каналу LiFi, надаючи закриті вирази ймовірності відключення і середньої швидкості бітової помилки для вмикання-вимикання ключа за умови непрямої модуляції або прямого детектування на приймачі.

Динамічний розвиток БПЛА та технологій LiFi дає можливість детально описати основні характеристики БПЛА, необхідні для вирішення проблеми передачі та скорочення сигналу. Наразі існує достатньо матеріалів, які представляють використання БПЛА у зв'язку з технологіями LiFi. Наприклад, автори роботи [48] аналізують продуктивність уніфікованого фізичного рівня на основі нейронних мереж для гібридних LiFi/WiFi мереж з використанням БПЛА у внутрішніх приміщеннях. Використання літаючих БПЛА в якості мережевих вузлів в гібридних LiFi/WiFi мережах може забезпечити ряд переваг, таких як поліпшення і підвищення якості обслуговування, підвищення гнучкості і розгортання, а також зменшення перешкод і затримок. Однак автори також визначають можливі виклики, такі як динамічні зміни топології мережі та необхідність забезпечення безпеки і конфіденційності.

У свою чергу, автори роботи [49] представляють огляд використання програмно-визначених мереж та віртуалізації мережевих функцій для БПЛА. Програмно-визначені мережі та віртуалізації мережевих функцій – це технології, які дозволяють підвищити гнучкість, масштабованість та керованість мереж. Програмно-визначені мережі дозволяють відокремити площину управління від площини даних, що дозволяє централізовано керувати і програмувати поведінку мережі. Віртуалізації мережевих функцій робить ці функції більш гнучкими та масштабованими.

У статті [50] наведено огляд використання БПЛА для підтримки периферійних обчислень в мережах 6G Інтернету транспортних засобів. Автор зазначає, що використання БПЛА для підтримки 6G Інтернету для транспортних засобів має ряд переваг, таких як зменшення затримок, підвищення продуктивності, розширення покриття мережі, підвищення надійності зв'язку тощо.

Таким чином, можна зазначити, що використання БПЛА наразі є вигідним, і з'являється багато рішень на основі технологій БПЛА.

При огляді технології автори [51] розглядають застосування технології LiFi для зв'язку між транспортними засобами. Зазначається, що LiFi має ряд переваг перед традиційними радіочастотними технологіями, такими як Wi-Fi і Bluetooth, включаючи високу пропускну здатність, безпеку і завадостійкість. Ці переваги роблять LiFi ідеальною технологією для організації зв'язку між транспортними засобами, яка може бути використана для підвищення безпеки дорожнього руху та ефективності транспорту.

Автори [52] обговорюють прогрес, досягнутий у покращенні продуктивності систем LiFi, таких як збільшення пропускної здатності, дальності та безпеки. Вони також обговорюють перспективи LiFi, такі як його потенційне застосування в 5G і далі. Основна думка полягає в тому, що LiFi - це перспективна технологія бездротової передачі даних з багатьма перевагами над радіочастотними технологіями. LiFi може використовуватися в додатках 5G, IoT та кібербезпеки. Потенційно LiFi може зробити революцію в індустрії бездротового зв'язку, забезпечуючи вищу пропускну здатність, безпеку та енергоефективність, ніж традиційні радіочастотні технології. Однак технологія також має ряд обмежень, які необхідно враховувати при виборі технології для конкретного завдання, наприклад, обмежений радіус дії і потреба в спеціалізованому обладнанні.

Комбінування можливих технологій передачі даних також є відносно поширеним напрямком досліджень. Автори роботи [53] надають огляд сучасних та критичних напрямків досліджень у розробці гібридних оптичних бездротових мереж. Слід зазначити, що гібридні мережі поєднують переваги оптичного та

радіохвильового зв’язку, що дозволяє подолати обмеження кожної технології окремо і забезпечити вищу пропускну здатність, надійність та енергоефективність.

У цьому дослідженні розглядаються різні комбінації гібридних систем, включаючи радіочастотні/оптичні та оптичні/оптичні системи. У статті зазначається, що гібридні оптичні бездротові мережі є перспективним рішенням для задоволення зростаючого попиту на високошвидкісний і надійний бездротовий зв’язок.

У статті [54] розглядаються питання безпеки технології LiFi. Розглядаються наступні можливі атаки на канал зв’язку:

− атаки на конфіденційність: оскільки LiFi використовує видиме світло, можна перехоплювати дані, аналізуючи світлові флуктуації;

− атаки на цілісність даних: зловмисники можуть змінювати дані LiFi, маніпулюючи інтенсивністю або частотою світлових коливань;

− атаки на доступність: зловмисники можуть заблокувати або порушити передачу даних LiFi, блокуючи або змінюючи світловий сигнал.

Також обговорюються різні методи безпеки LiFi, такі як шифрування даних, автентифікація та виявлення атак. Загалом, LiFi є безпечною технологією, але для захисту даних і мереж LiFi від атак необхідно вжити певних заходів безпеки, а саме:

− шифрування даних може захистити дані LiFi від перехоплення та подальшої модифікації;

− аутентифікація може допомогти запобігти несанкціонованому доступу до LiFi мереж;

− розробка та впровадження систем виявлення атак, які можуть виявляти та блокувати атаки на LiFi мережі.

Автори [55] представляють систему позиціонування в приміщенні на основі часу польоту для LiFi на основі рекомендації ITU-T G.9991. Запропонований алгоритм позиціонування базується на грубому вимірюванні часу з використанням преамбули синхронізації кадру, подібно до визначення дальності, та точному вимірюванні часу з використанням преамбули оцінки каналу. Результати на місцевості показують, що позиціонування на основі G.9991 може досягати

середньої похибки відстані в кілька сантиметрів у тривимірному вимірі. З огляду на широке використання освітлення в приміщеннях і наявність розвиненої оптичної бездротової системи зв'язку з використанням G.9991, запропоноване позиціонування LiFi є багатообіцяючою новою функцією, яка може бути додана до існуючих протоколів і ще більше розширити можливості інтелектуальних систем освітлення на благо Індустрії 4.0.

Для планування розгортання дуже важливо мати підтримку у вигляді відповідних програмних засобів. Питання моделювання роботи БПЛА розглядаються у багатьох роботах. Наприклад, у [56] наголошується, що моделювання є важливим інструментом для розробки та тестування БПЛА. Моделювання дає змогу інженерам досліджувати поведінку БПЛА в різних умовах, що може допомогти їм уникнути дорогих і небезпечних випробувань на реальних БПЛА.

Автори [57] представили комп'ютерний симулятор для моделювання поведінки БПЛА в середовищі з перешкодами. При цьому вони використовували планування на основі графів, управління на основі зворотного зв'язку та різні методи машинного навчання.

У роботі [58] розглянуто реалізацію наземної станції управління для симулятора польоту БПЛА. Наземна станція управління дає змогу оператору керувати БПЛА в симуляторі та спостерігати за його поведінкою.

У [59] за допомогою обчислювального моделювання досліджується вплив запуску боєприпасів на стійкість шестироторних БПЛА. Автори аналізують вплив різних чинників, таких як кут запуску боєприпасів, час запуску, положення установки і маса, на стійкість БПЛА.

У статті [60] розглядається розробка наземного симулятора польоту з регульованою стійкістю для пілотної підготовки. Такий симулятор дає змогу імітувати поведінку літака в різних умовах польоту, включно зі змінами аеродинамічних характеристик і відмовами систем керування. Це допомагає пілотам тренувати свої навички пілотування і справлятися з несподіваними ситуаціями.

Автори [61] обговорюють ключові компоненти та функції програмного забезпечення, необхідні для безпечного та ефективного керування БПЛА.

Робота [62] пропонує структуру для планування ефективних експериментів з роями БПЛА на основі симулятору. Автори підкреслюють важливість вибору симулятору за такими параметрами, як масштабованість, достовірність та зручність використання.

У публікації [63] розглядається фреймворк UTSim, який призначений для інтеграції БПЛА до керування повітряним рухом.

У вже проаналізованих роботах з питань застосування технології LiFi, мова йшла в основному про стаціонарні рішення. Але останнім часом літаючі LiFi мережі, які певний час вважались принципово можливим, але доволі екзотичним технічним рішенням, стають все більш затребуваними під час вирішення задач з реалізації бездротового зв'язку як всередині приміщень, так і за їх межами.

Стаття [64] демонструє можливості застосування літаючої LiFi мережі для збору та передачі даних про метеоумови до наземної станції для подальшої їх обробки.

У роботі [65] обґрунтована доцільність застосування літаючої LiFi мережі всередині приміщення для підтримки рішень на основі технологій 6G.

Оскільки після розгортання літаючої мережі важливо забезпечити її надійне функціонування, варто торкнутися робіт, які є дотичними до вирішення цього питання.

Окремі роботи присвячені забезпеченню надійності роїв (флотів) БПЛА, задіяних у розгортанні літаючих бездротових мереж. У статті [66] представлено модель надійності рою БПЛА, який розгортає бездротову літаючу мережу в умовах інтенсивних бойових дій, на основі безперервного часового ланцюга Маркова.

Робота [67] пропонує новий метод оцінки надійності рою БПЛА з різними видами його резервування, який дозволяє з використанням міри важливості визначати БПЛА, несправність яких має найбільший вплив на відмову літаючої бездротової мережі.

Моделі оцінки надійності флоту БПЛА з централізованим і децентралізованим управлінням, які базується на представленні цього флоту як системи з бінарними станами, представлено у [68].

Авторами [69] розроблено та дослідження моделі надійності літаючої бездротової мережі на основі угруповань флотів БПЛА з централізованою, децентралізованою і змішаною схемою активації резервних БПЛА та з можливістю резервування пунктів управління, сформульовано рекомендації щодо вибору схеми активації резервних БПЛА.

Таким, чином проаналізовані роботи показують, що під час розгортання бездротових літаючих мереж у приміщеннях виникають наступні задачі:

– створення гібридних LiFi/WiFi мереж;

– прокладання маршрутів розповсюдження сигналу та руху БПЛА всередині приміщення в обхід перешкод з використанням різних алгоритмів обходу перешкод;

– забезпечення надійного функціонування літаючих бездротових мереж протягом заданого час;

– розроблення і застосування програмних засобів підтримки планування розгортання літаючих бездротових мереж.

Відзначено, що LiFi мережі стають все більш затребуваними під час вирішення задач з реалізації бездротового зв’язку всередині приміщень, але потрібують також питань щодо їх надійного та безпечного функціонування в приміщеннях з перешкодами.

1.3 Постановка задачі та завдань досліджень

В рамках дисертаційної роботи використовують принципи системного аналізу при розробці постановки задачі та вирішення часткових задач дослідження, а саме:

− при декомпозиції поставлених завдань на етапи та визначенні логіки їх виконання;

− обранні методів досліджень, математичного апарату та їх взаємозв’язку із результатами окремих етапів;

− побудові методів планування розгортання, власне розгортання та підвищення надійності утвореної літаючої LiFi мережі в приміщенні зі статичними перешкодами.

Для вирішенні сформульованих наукових та прикладних завдань була розроблена схема планування розгортання літаючої LiFi мережі на основі БПЛА у виробничому приміщенні з перешкодами (рис. 1.1), відповідно до якої таке планування відбувається у три етапи.

Етап 1. *Прокладання маршруту розповсюдження LiFi сигналу в умовах перешкод*. У найпростішому варіанті завдання може бути сформульовано так: потрібно прокласти маршрут (лінію LiFi зв’язку) з точки *A* (джерело інформації) в точку *B* (споживач інформації) у виробничому приміщенні з перешкодами в двовимірному (2D) просторі. Незважаючи на те, що використання 2D простору може обмежити гнучкість у виборі оптимальних маршрутів, особливо в умовах мінливого середовища або вимог до точності навігації, моделювання у 2D також є доцільним з таких міркувань:

– у 2D просторі завдання прокладання маршрутів розповсюдження LiFi сигналу в обхід перешкод може бути простішим порівняно з тривимірним (3D) простором і зменшити обчислювальне навантаження на систему підтримки планування розгортання літаючої LiFi мережі, прискорюючи тим самим процес ухвалення рішень;

– сценарії моделювання у 2D просторі можуть дозволити використання більш простих та дешевих датчиків і системи навігації, що також знижує вартість і складність обладнання.

Також необхідно прийняти припущення про те, що яскравість світла у приміщенні, а також рівень його задимлення (запиленості) дозволяє застосовувати LiFi технологію. На початку, якщо є можливість, необхідно оцінити наслідки руйнувань у виробничому приміщенні і скласти карту перешкод, причому для реалізації розглянутих далі методів кожну з перешкод представимо у вигляді

опуклого багатокутника. Далі обираємо один з методів обходу перешкод: метод прямокутників (коли обхід кожної перешкоди здійснюється виключно за правилом лівого або правого кута) або метод керованого водоспаду (більш детально зазначені методи буде розглянуто у розділі 2). При реалізації кожного з методів приймаємо припущення про те що точки *A* і *B*, а також перешкоди є статичними і не змінюються з часом. Застосування зазначених методів дозволяє сформувати множину маршрутів від точки *A* до точки *B* у 2D просторі виробничого приміщення, а поєднання всіх точок маршруту (*A*, *B* і точок зміни напрямку руху за маршрутом внаслідок обходу перешкод) – згенерувати граф маршрутів. Наявність такого графа на наступному кроці дозволяє застосувати алгоритм пошуку найкоротшого маршруту (наприклад, алгоритм Дейкстри) розповсюдження LiFi сигналу в умовах перешкод від джерела інформації (точка *A*) до її споживача (точка *B*). Заключним кроком першого етапу є оброблення отриманих результатів з метою їх подальшого використання на етапах 2 і 3. Під час виконання етапу 1 використовуються методи системного аналізу, оптимізації, математичного моделювання, теорії графів.

Етап 2. *Розміщення БПЛА у визначених точках прокладеного маршруту для розгортання LiFi мережі*. Першим кроком на цьому етапі є визначення переліку варіантів руху кожного БПЛА з місця свого базування до заданої точки маршруту для розгортання LiFi мережі. На наступному кроці відповідно із заданим критерієм (наприклад, часом розгортання) обирається найкращій варіант і здійснюється розміщення БПЛА у визначених точках прокладеного маршруту. Під час виконання етапу 2 використовуються методи системного аналізу, математичного моделювання, теорії графів, теорії розкладів.

Етап 3. *Забезпечення надійності розгорнутої LiFi мережі*. У разі отримання вимог від замовника стосовно мінімально необхідного значення ЙБР LiFi мережі визначається спосіб її резервування та кількість резервних БПЛА. У разі зміни вимог, здійснюється коригування способу резервування та/або кількості резервних БПЛА. Під час виконання етапу 2 використовуються методи системного аналізу, теорії надійності, математичного моделювання, теорії розкладів.

Результати виконання етапів у подальшому можуть бути використані системою планування розгортання БПЛА літаючої LiFi мережі, яка, в свою чергу є основним інструментом підтримки прийняття рішень особою, відповідальною за реагування на аварії на об’єктах критичної інфраструктури.

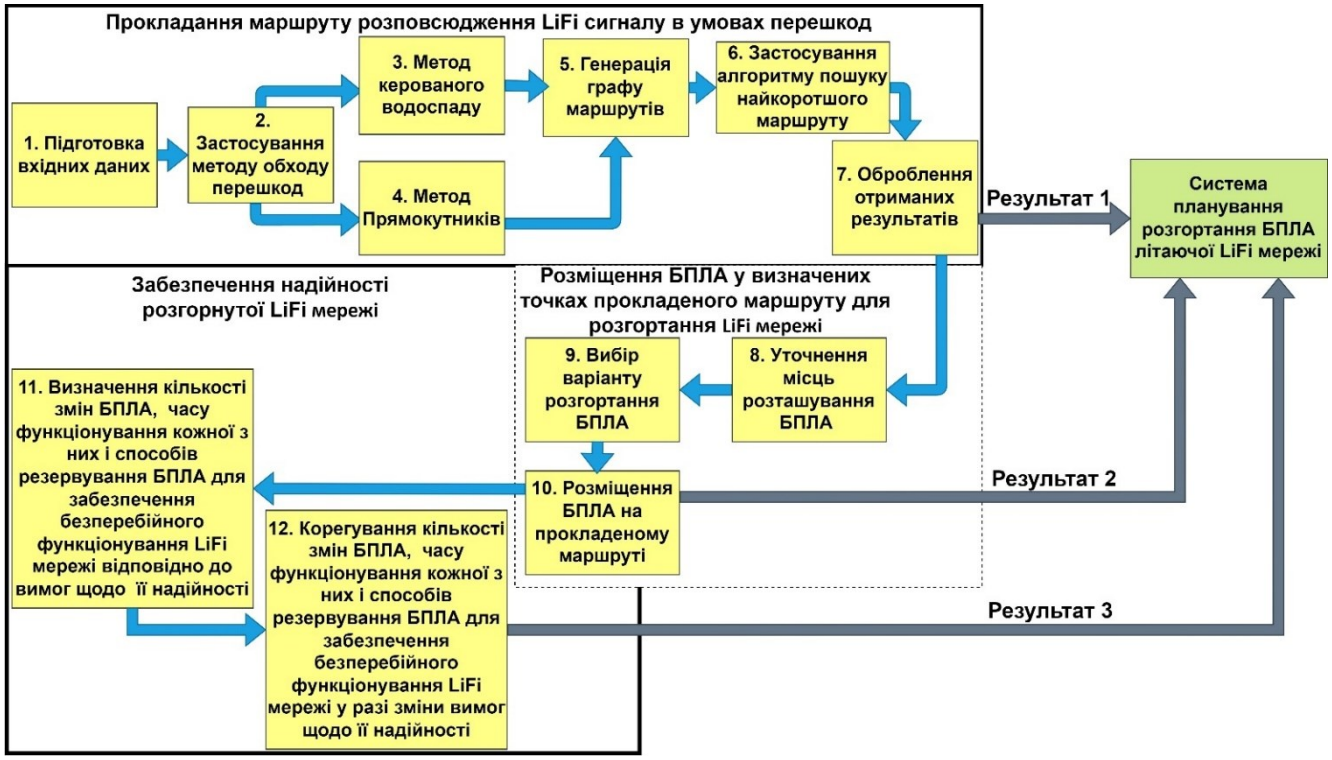

Рисунок 1.1 − Схема планування розгортання LiFi мережі на основі БПЛА у виробничому приміщенні з перешкодами

1.4 Висновки до першого розділу

1. Проведено аналіз способів застосування БПЛА у приміщеннях.

Проаналізовані роботи показують, що існує достатньо широкий спектр застосування БПЛА у приміщеннях:

– моніторинг якості повітря приміщень та пошук джерел забруднення;
– управління запасами на складах;
– проведення контактного та безконтактного інспектування приміщень;
– гасіння пожеж у приміщеннях;
– проведення пошуково-рятувальних робіт.

Разом з тим, автори відзначають проблеми, з якими доводиться стикатися під час застосування БПЛА у приміщеннях:

– наявність у приміщеннях середовищ з відсутністю GPS сигналів;

– наявність у приміщеннях механічних перешкод;

– специфічне планування окремих будівель, що ускладнює рух у них БПЛА;

– необхідність розгортання додаткової інфраструктури для забезпечення польотів всередині приміщень.

2. Проведено аналіз існуючих методів розгортання мереж в умовах руйнувань.

Проаналізовані роботи показують, що під час розгортання бездротових літаючих мереж у приміщеннях виникають наступні задачі:

– створення гібридних LiFi/WiFi мереж;

– прокладання маршрутів розповсюдження сигналу та руху БПЛА всередині приміщення в обхід перешкод з використанням різних алгоритмів обходу перешкод;

– забезпечення надійного функціонування літаючих бездротових мереж протягом заданого час;

– розроблення і застосування програмних засобів підтримки планування розгортання літаючих бездротових мереж.

Відзначено, що LiFi мережі стають все більш затребуваними під час вирішення задач з реалізації бездротового зв’язку всередині приміщень, але потребують також вирішення питань щодо їх надійного та безпечного функціонування в приміщеннях з перешкодами.

3. Відзначено що за рамками досліджень проаналізованих робіт залишаються або опосередковано розглядаються питання:

– комбінування різних методів обходу перешкод під час формування множин маршрутів; формування стратегій руху БПЛА до місць розміщення для утворення бездротової мережі;

– обмеженість у часі функціонування бездротової мережі на основі БПЛА внаслідок обмеженого ресурсу їх бортової батареї;

– організації забезпечення безперебійного функціонування бездротової мережі протягом заданого часу; застосування LiFi технології для розгортання літаючих мереж на основі БПЛА.

3. Показано переваги застосування LiFi технології для утворення літаючих мереж на основі БПЛА, які полягають у великій швидкості та безпеці передачі даних, меншій інтерференції та енергоефективності. Використання БПЛА-ретрансляторів дозволяє швидко розгорнути таку мережу у важкодоступних місцях і оперативно змінювати конфігурацію у разі корегування завдань щодо її застосування.

4. Розглянуто недоліки технології LiFi, які полягають у низькій якості зв'язку в умовах яскравого освітлення, задимленості та запиленості, а також у неможливості проникнення LiFi сигналу крізь механічні перешкоди.

5. На основі проведеного аналізу була сформульована загальна задача дисертаційного дослідження, яка була декомпозована на перелік часткових задач, виконання яких доцільно здійснювати у три етапи:

– прокладання маршруту розповсюдження LiFi сигналу в умовах перешкод;

– розміщення БПЛА у визначених точках прокладеного маршруту для розгортання LiFi мережі;

– забезпечення надійності розгорнутої LiFi мережі.

Результати виконання етапів у подальшому можуть бути використані системою підтримки планування розгортання БПЛА літаючої LiFi мережі.

РОЗДІЛ 2
РОЗРОБЛЕННЯ МЕТОДІВ ПЛАНУВАННЯ РОЗМІЩЕННЯ БПЛА ЛІТАЮЧОЇ LIFI МЕРЕЖІ ДЛЯ ЗАБЕЗПЕЧЕННЯ ПЕРЕДАЧІ ДАНИХ В УМОВАХ РУЙНУВАНЬ

2.1 Обґрунтування основних кроків проведення досліджень у розділі 2

Проведення досліджень у розділі 2 відбуваються у відповідності з вимогами, встановленими для реалізації Етапу 1. *Прокладання маршруту розповсюдження LiFi сигналу в умовах перешкод*, який є частиною *Схеми планування розгортання LiFi мережі на основі БПЛА у виробничому приміщенні з перешкодами* і був розглянутий у розділі 1. Типовий порядок дій за цим етапом включає такі кроки.

Крок 1. Підготовка вхідних даних, включаючи формулювання допущень і обмежень.

Крок 2. Застосування методів обходу перешкод: методу прямокутників і методу керованого водоспаду.

Крок 3. Генерація графу можливих LiFi маршрутів.

Крок 4. Застосування алгоритму пошуку найкоротшого LiFi маршруту на графі, згенерованому на кроці 3.

Крок 5. Оброблення отриманих результатів.

2.2 Розроблення методу прямокутників для планування розміщення БПЛА літаючої LiFi мережі

2.2.1 Обґрунтування основних положень методу прямокутників для планування розміщення БПЛА літаючої LiFi мережі

Перед розглядом основних положень методу прямокутників для планування розміщення БПЛА літаючої LiFi мережі приймемо ряд допущень та обмежень:

– планування здійснюється у 2D просторі, де місця розміщення джерела (споживача) інформації та БПЛА являють собою точки з координатами (x_i, y_i);

– джерело інформації і споживач інформації не змінюють своє положення з часом;

– перешкоди робочої області приміщення мають форму прямокутника і не змінюють свої розміри і положення з часом;

– щільність розташування перешкод дозволяє їх обходити;

– рівень яскравості (задимленості, запиленості) у приміщенні дозволяє застосовувати LiFi технологію.

Сутність планування розміщення БПЛА літаючої LiFi мережі за методом прямокутників полягає у наступному (рис. 2.1).

Формулюють задачу планування: прокласти маршрут розповсюдження LiFi сигналу (LiFi маршрут) від джерела інформації (точка *A*) до споживача інформації (точка *B*) в обхід перешкод робочої зони приміщення з позначенням на маршруті місць розміщення БПЛА для утворення літаючої LiFi мережі, використовуючи для обходу перешкод правило лівого (правого) кута. Сутність цих правил буде показана далі.

На наступному кроці проводять умовну пряму лінію між точками *A* і *B* і починають прокладати маршрут вздовж цієї лінії до точки зіткнення з перешкодою (у нашому випадку перешкода має вид прямокутника). Ця точка позначається як місце розміщення БПЛА. У разі, якщо довжина ділянки маршруту від точки *A* до точки зіткнення з перешкодою є більшою за встановлену для заданих умов приміщення дальність розповсюдження LiFi сигналу, на цій ділянці визначається додаткова точка (точки) розміщення БПЛА, щоб гарантувати надходження LiFi сигналу від точки *A* до точки зіткнення ділянки маршруту з перешкодою.

На наступному кроці прокладається наступна ділянка LiFi маршруту, яка являє собою пряму лінію між точкою зіткнення з перешкодою, що має вид прямокутника і вершиною верхнього правого (лівого) кута цього прямокутника. Ця вершина позначається як точка (місце) розміщення БПЛА.

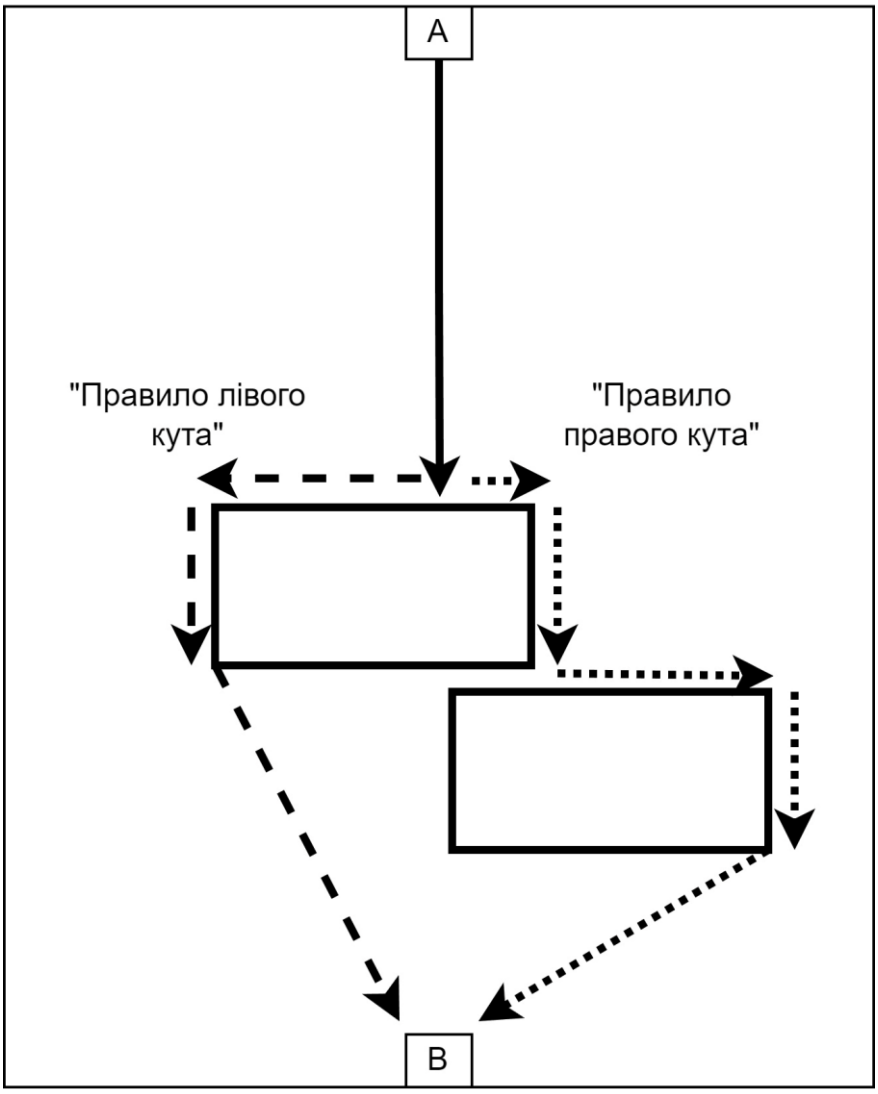

Рисунок 2.1 − Ілюстративний приклад, який пояснює особливості застосування для обходу перешкод алгоритмів лівого та правого кута

На наступному кроці прокладається наступна ділянка LiFi маршруту, яка являє собою пряму лінію між вершиною правого (лівого) верхнього кута і вершиною правого (лівого) нижнього кута прямокутника, яка позначається як точка (місце) розміщення БПЛА.

На наступному кроці проводять умовну пряму лінію між вершиною правого (лівого) нижнього кута прямокутника і точкою *B* і продовжують прокладати маршрут вздовж цієї лінії до точки зіткнення з наступною перешкодою. Ця точка позначається як чергова точка (місце) розміщення БПЛА. Всі дії щодо прокладання

LiFi маршруту в обхід цієї та наступних перешкод є аналогічні діям, що здійснювались під час обходу першої перешкоди.

Після обходу останньої перешкоди (коли визначено точку (місце) розміщення БПЛА у вершині правого (лівого) нижнього кута прямокутника) остання ділянка маршруту буде являти собою пряму лінію між вершиною правого (лівого) нижнього кута прямокутника і точкою *B*. У разі, якщо довжина ділянки маршруту між вершиною правого (лівого) нижнього кута прямокутника і точкою *B* є більшою за встановлену для заданих умов приміщення дальність розповсюдження LiFi сигналу, на цій ділянці LiFi маршруту визначається додаткова точка (точки) розміщення БПЛА, щоб гарантувати надходження LiFi сигналу від вершини правого (лівого) нижнього кута прямокутника до точки *B*.

В результаті виконання кроків, описаних вище, буде сформований LiFi маршрут між джерелом (точка *A*) і споживачем (точка *B*) інформації з нанесеними на ньому точками (місцями) розташування БПЛА для утворення літаючої LiFi мережі. Маршрут буде являти собою ламану лінію з вершинами в точках зміни напрямку LiFi маршруту. У якості цих точок виступають точки зіткнення ділянки маршруту з перешкодою, а також вершини правого (лівого) нижнього та верхнього кутів прямокутника, що символізує перешкоду. У більш загальному випадку перешкоди можуть мати довільну форму, однак кожна перешкода може бути вписана в опуклий багатокутник і бути представлена цією геометричною фігурою.

Кількість точок (місць) розміщення БПЛА на LiFi маршруті може або збігатися з кількістю вершин ламаної лінії (коли довжина кожної із ділянок LiFi маршруту не перевищує встановлену для заданих умов приміщення дальність розповсюдження LiFi сигналу) або бути більшою за кількість вершин ламаної лінії (коли існує хоча б одна ділянка LiFi маршруту з довжиною, більшою за встановлену для заданих умов приміщення дальність розповсюдження LiFi сигналу).

Таким чином, в залежності від правила обходу перешкод, метод прямокутників може включати реалізацію одного з двох алгоритмів:

– алгоритму правого кута, відповідно до якого кожен раз при зіткненні поточної ділянки LiFi маршруту з перешкодою, наступна його ділянка являє собою пряму лінію між точкою зіткнення з перешкодою і вершиною правого верхнього кута прямокутника, що символізує цю перешкоду;

– алгоритму лівого кута, відповідно до якого кожен раз при зіткненні поточної ділянки LiFi маршруту з перешкодою, наступна його ділянка являє собою пряму лінію між точкою зіткнення з перешкодою і вершиною лівого верхнього кута прямокутника, що символізує цю перешкоду.

2.2.2 Планування розміщення БПЛА літаючої LiFi мережі з використанням алгоритму правого кута

Розглянемо різні варіанти застосування алгоритму правого кута, сутність якого розглянута у п. 2.2.1.

З використанням програмного засобу “Simulation Way”, архітектура якого буде розглянута детально у розділі 4, проведемо моделювання процесу планування розміщення БПЛА LiFi мережі для забезпечення передачі даних від точки *A* (джерело даних) до точки *B* (споживач даних) в робочій зоні приміщення з руйнуваннями з використанням алгоритму правого кута для наступних вхідних даних:

- розмір робочої зони приміщення (довжина×ширина) = 20×20 м;
- кількість перешкод = 15;
- розмір кожної перешкоди = 3×2 м.

Результати моделювання показано на рис. 2.2.

На рис. 2.2 зеленою ламаною лінією показано прокладений LiFi маршрут в обхід перешкод, а точками на маршруті – місця розміщення БПЛА на цьому маршруті для утворення літаючої LiFi мережі. Як ми можемо бачити на рис. 2.2:

- алгоритм правого кута був застосований для обходу трьох перешкод під час прокладання LiFi маршруту;

– для розгортання LiFi мережі необхідно розмістити у визначених точках прокладеного LiFi маршруту 9 БПЛА.

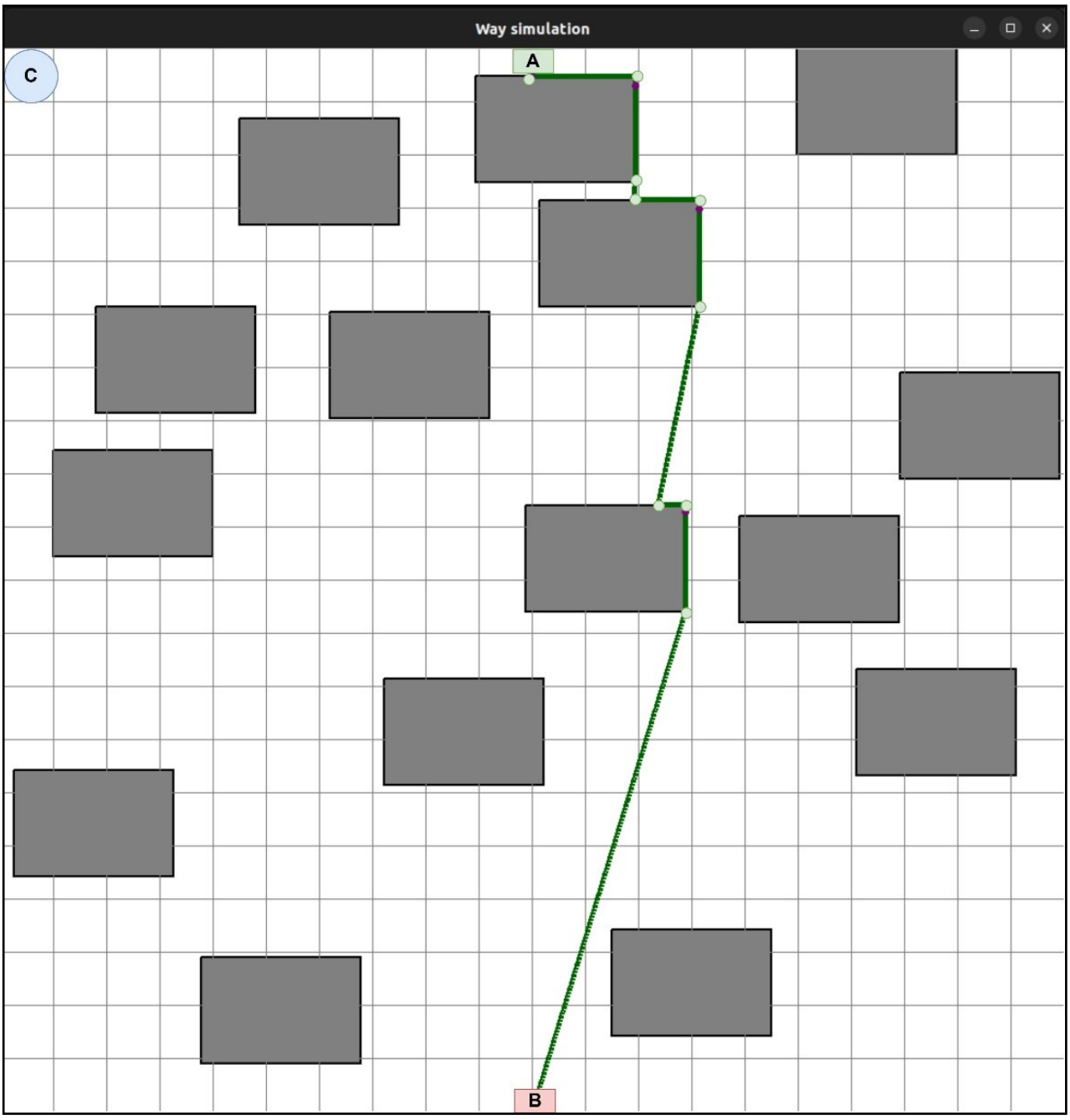

Рисунок 2.2 – Результат моделювання процесу планування розміщення БПЛА LiFi мережі для забезпечення передачі даних від точки *A* (джерело даних) до точки *B* (споживач даних) в робочій зоні приміщення з перешкодами з використанням алгоритму правого кута

Цей приклад є демонстраційним і показує особливості моделювання процесу планування розміщення БПЛА LiFi мережі для забезпечення передачі даних від точки *A* (джерело даних) до точки *B* (споживач даних) в робочій зоні приміщення з перешкодами з використанням алгоритму правого кута. Колом з літерою С всередині позначено місце базування (депо) БПЛА. Ці БПЛА у подальшому будуть використані під час опису стратегій їхнього розміщення на прокладеному LiFi маршруті у розділі 3.

У подальшому було проведено 29 експериментів з використанням програмного засобу “Simulation Way” (номер експерименту відповідає кількості згенерованих у ньому перешкод) і отримано залежності:

– довжини прокладеного LiFi маршруту в робочій зоні приміщення розмірами 20×20 м від кількості перешкод (рис. 2.3) при застосуванні алгоритму правого кута;

– кількості БПЛА (кількості точок на прокладеному LiFi маршруті), яка необхідна для утворення літаючої LiFi мережі, від кількості перешкод при застосуванні алгоритму правого кута (рис. 2.4).

Аналіз графіків, представлених на рис. 2.3, дозволяє зробити наступні висновки.

1) За наявності перешкод 2×2 м у 15-ти випадках вдається прокласти найкоротший у порівнянні з іншими перешкодами LiFi маршрут. Для перешкод 2×3 м та 3×2 м така кількість випадків становить 8 та 1 відповідно.

2) За наявності 20 перешкод, перешкоди 2×2 м у порівнянні з перешкодами 2×3 м та 3×2 м дозволяють прокласти маршрут довжиною, меншою відповідно на 4,95 м (20,25 м проти 25,2 м) та 13,31 м (20,25 м проти 33,56 м).

3) Найкоротший для всіх перешкод LiFi маршрут становить 20 м. Такий маршрут можливо прокласти у 10-ти, 4-х та 3-х випадках відповідно для перешкод 2×2 м, 2×3 м та 3×2 м.

4) Найдовший LiFi маршрут для перешкод 2×2 м, 2×3 м та 3×2 м становить 25,43 м (за наявності 28 перешкод), 29,83 м (за наявності 29 перешкод) та 39,93 м (за наявності 28 перешкод) відповідно.

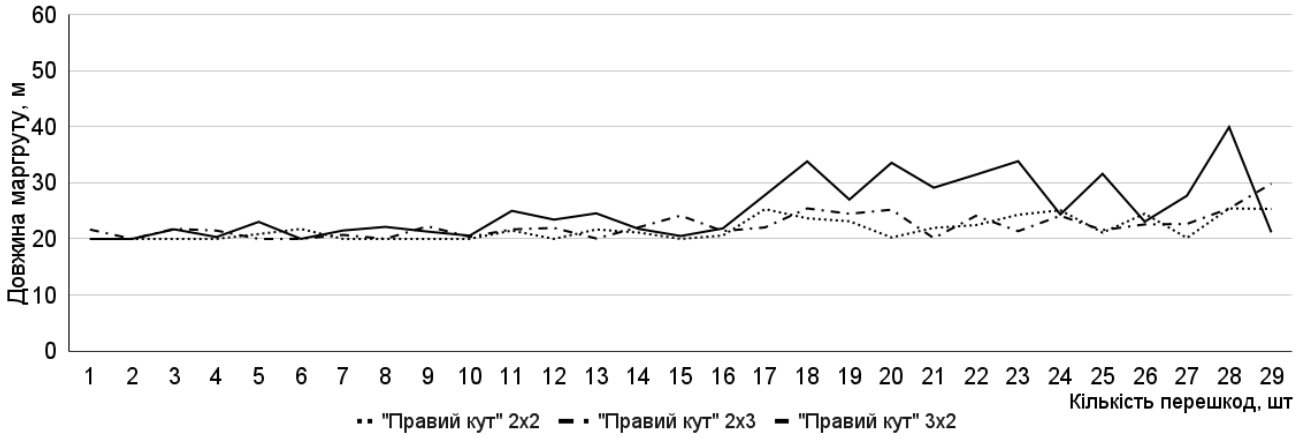

Рисунок 2.3 – Графік залежності довжини прокладеного LiFi маршруту в робочій зоні приміщення від кількості перешкод при застосуванні алгоритму правого кута

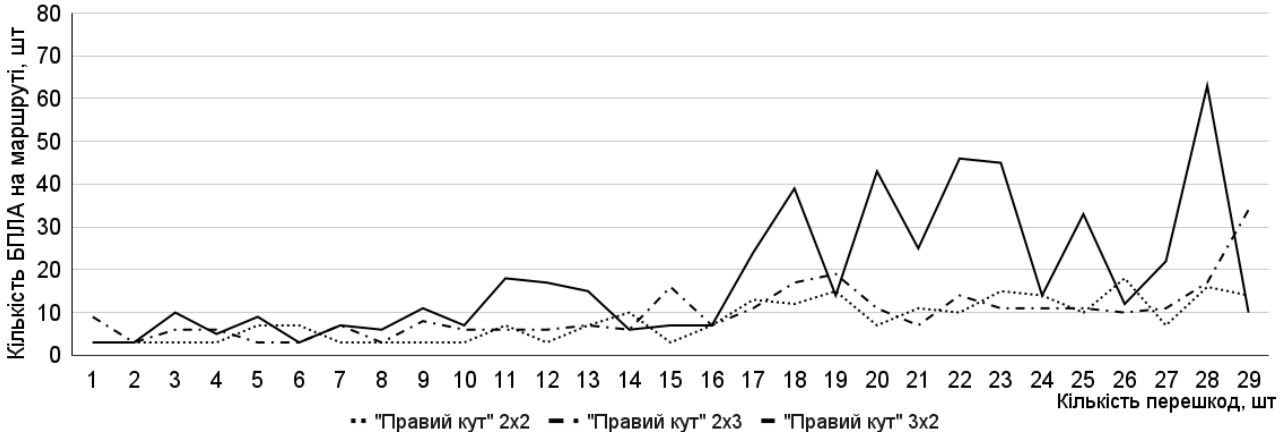

Рисунок 2.4 – Графік залежності кількості БПЛА (кількості точок на прокладеному LiFi маршруті), яка необхідна для утворення літаючої LiFi мережі, від кількості перешкод при застосуванні алгоритму правого кута

Аналіз графіків, представлених на рис. 2.4, дозволяє зробити наступні висновки.

1) За наявності перешкод 2×2 м у 13-ти випадках вдається досягти найменшої у порівнянні з іншими перешкодами кількості БПЛА, необхідної для утворення літаючої LiFi мережі. Для перешкод 2×3 м та 3×2 м така кількість випадків становить 7 та 2 відповідно.

2) За наявності 29 перешкод, перешкоди 2×2 м у порівнянні з перешкодами 2×3 м дозволяють використати для утворення літаючої LiFi мережі на 20 БПЛА (14 проти 34) менше.

3) За наявності 28 перешкод, перешкоди 2×2 м у порівнянні з перешкодами 3×2 м дозволяють використати для утворення літаючої LiFi мережі на 47 БПЛА (16 проти 63) менше.

4) Найменша для всіх перешкод кількість БПЛА, необхідна для утворення літаючої LiFi мережі, становить 3. Вона можлива у 10-ти, 4-х та 3-х випадках відповідно для перешкод 2×2 м, 2×3 м та 3×2 м.

5) Найбільша кількість БПЛА, необхідна для утворення літаючої LiFi мережі, для перешкод 2×2 м, 2×3 м та 3×2 м становить 18 (за наявності 26 перешкод), 34 (за наявності 29 перешкод) та 63 (за наявності 28 перешкод) відповідно.

2.2.3 Планування розміщення БПЛА літаючої LiFi мережі з використанням алгоритму лівого кута

Розглянемо різні варіанти застосування алгоритму лівого кута, сутність якого розглянута у п. 2.2.1.

З використанням програмного засобу “Simulation Way” проведемо моделювання процесу планування розміщення БПЛА LiFi мережі для забезпечення передачі даних від точки *A* (джерело даних) до точки *B* (споживач даних) в робочій зоні приміщення з руйнуваннями з використанням алгоритму лівого кута для тих же вхідних даних що і в п. 2.2.2.

Результати моделювання показано на рис. 2.5.

На рис. 2.5 зеленою ламаною лінією показано прокладений LiFi маршрут в обхід перешкод, а точками на маршруті – місця розміщення БПЛА на цьому маршруті для утворення літаючої LiFi мережі. Як ми можемо бачити на рис. 2.7:

– алгоритм лівого кута був застосований для обходу трьох перешкод під час прокладання LiFi маршруту;

– для розгортання LiFi мережі необхідно розмістити у визначених точках прокладеного LiFi маршруту 9 БПЛА.

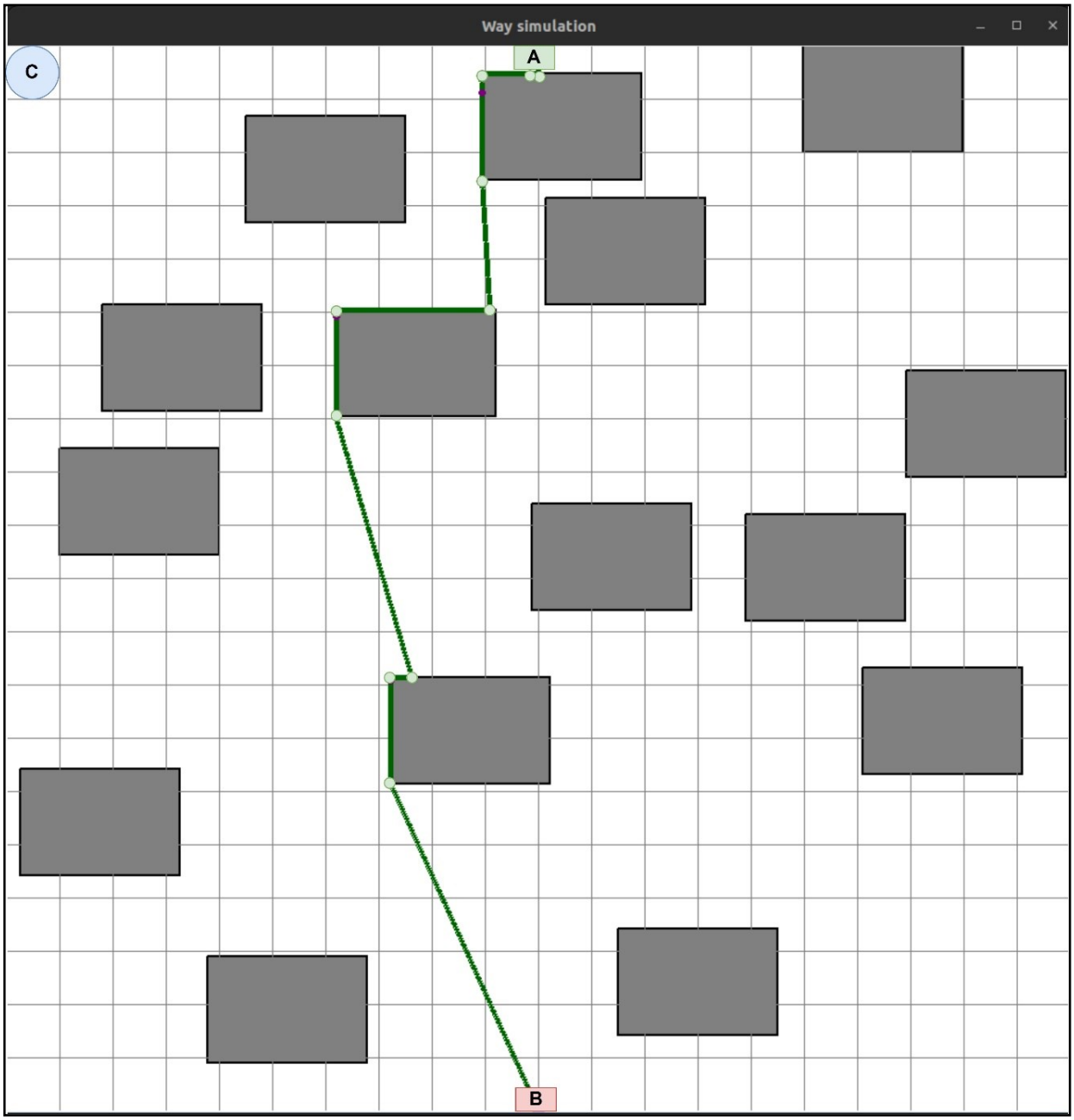

Рисунок 2.5 – Результат моделювання процесу планування розміщення БПЛА LiFi мережі для забезпечення передачі даних від точки *A* (джерело даних) до точки *B* (споживач даних) в робочій зоні приміщення з перешкодами з використанням алгоритму лівого кута

Цей приклад є демонстраційним і показує особливості моделювання процесу планування розміщення БПЛА LiFi мережі для забезпечення передачі даних від точки *A* (джерело даних) до точки *B* (споживач даних) в робочій зоні приміщення з перешкодами з використанням алгоритму лівого кута.

Як і для алгоритму правого кута у подальшому було проведено 29 експериментів з використанням програмного засобу “Simulation Way” (номер експерименту відповідає кількості згенерованих у ньому перешкод) і отримано залежності:

– довжини прокладеного LiFi маршруту в робочій зоні приміщення від кількості перешкод (рис. 2.6) при застосуванні алгоритму лівого кута;

– кількості БПЛА (кількості точок на прокладеному LiFi маршруті), яка необхідна для утворення літаючої LiFi мережі, від кількості перешкод при застосуванні алгоритму лівого кута (рис. 2.7).

Аналіз графіків, представлених на рис. 2.6, дозволяє зробити наступні висновки.

1) За наявності перешкод 2×2 м у 13-ти випадках вдається прокласти найкоротший відносно інших перешкод LiFi маршрут. Для перешкод 2×3 м та 3×2 м така кількість випадків становить 11 та 1 відповідно.

2) За наявності 15 перешкод, перешкоди 2×2 м у порівнянні з перешкодами 2×3 м дозволяють прокласти LiFi маршрут довжиною, меншою на 6,51 м (20 м проти 26,51 м).

3) За наявності 28 перешкод, перешкоди 2×2 м у порівнянні з перешкодами 3×2 м дозволяють прокласти LiFi маршрут довжиною, меншою на 11,15 м (22,77 м проти 23,92 м).

4) Найкоротший для всіх перешкод LiFi маршрут становить 20 м. Такий маршрут можливо прокласти у 10-ти, 4-х та 3-х випадках відповідно для перешкод 2×2 м, 2×3 м та 3×2 м.

5) Найдовший LiFi маршрут для перешкод 2×2 м, 2×3 м та 3×2 м становить 25,44 м (за наявності 14 перешкод), 27,5 м (за наявності 22 перешкод) та 33,92 м (за наявності 28 перешкод) відповідно.

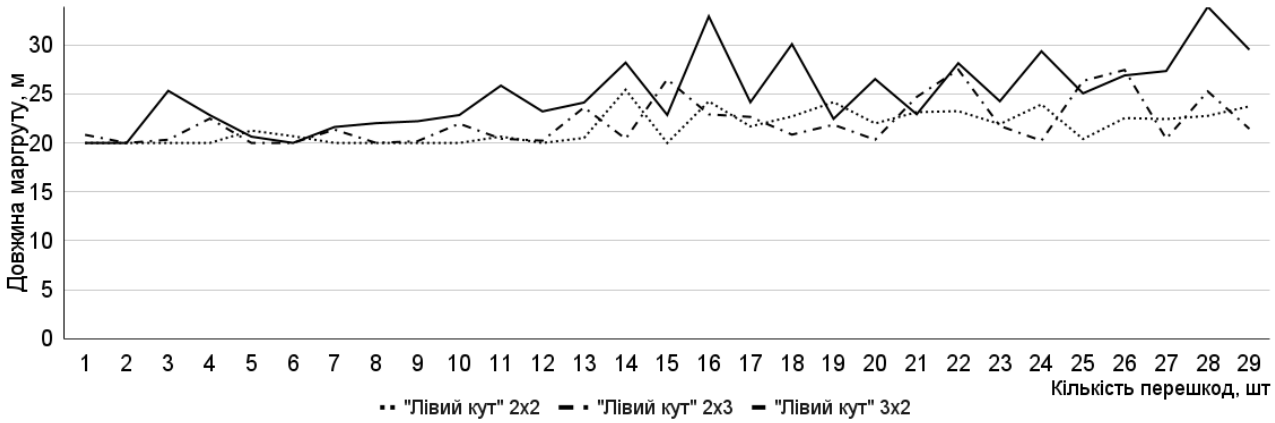

Рисунок 2.6 – Графік залежності довжини прокладеного LiFi маршруту в робочій зоні приміщення від кількості перешкод при застосуванні алгоритму лівого кута

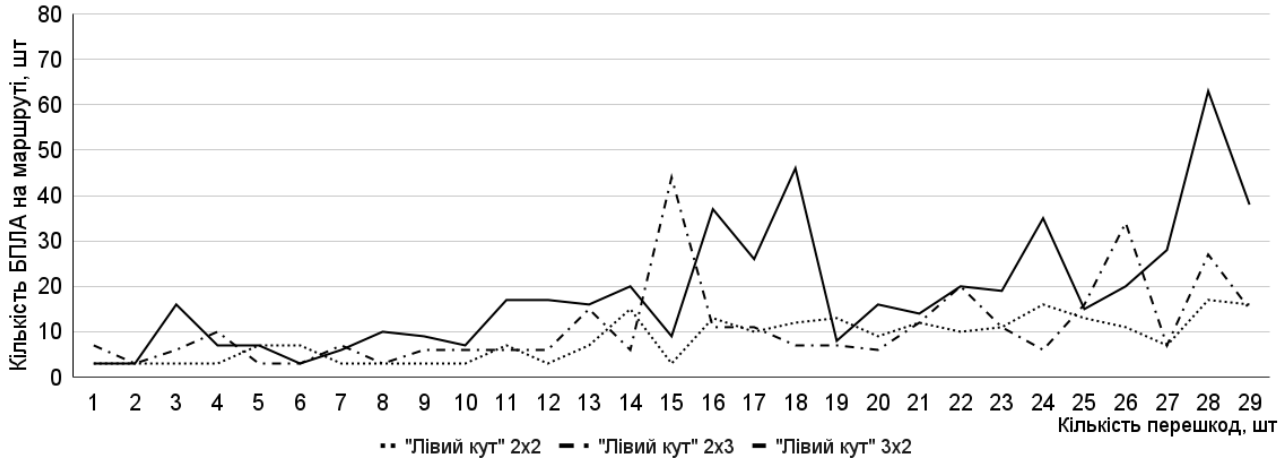

Рисунок 2.7 – Графік залежності кількості БПЛА (кількості точок на прокладеному LiFi маршруті), яка необхідна для утворення літаючої LiFi мережі, від кількості перешкод при застосуванні алгоритму лівого кута

Аналіз графіків, представлених на рис. 2.7, дозволяє зробити наступні висновки.

1) За наявності перешкод 2×2 м у 13-ти випадках вдається досягти найменшої у порівнянні з іншими перешкодами кількості БПЛА, необхідної для утворення літаючої LiFi мережі. Для перешкод 2×3 м та 3×2 м така кількість випадків становить 9 та 0 відповідно.

2) За наявності 17 перешкод, перешкоди 2×2 м у порівнянні з перешкодами 2×3 м дозволяють використати для утворення літаючої LiFi мережі на 16 БПЛА (20 проти 26) менше.

3) За наявності 28 перешкод, перешкоди 2×2 м у порівнянні з перешкодами 3×2 м дозволяють використати для утворення літаючої LiFi мережі на 46 БПЛА (17 проти 63) менше.

4) Найменша для всіх перешкод кількість БПЛА, необхідна для утворення літаючої LiFi мережі, становить 3. Вона можлива у 10-ти, 4-х та 3-х випадках відповідно для перешкод 2×2 м, 2×3 м та 3×2 м.

5) Найбільша кількість БПЛА, необхідна для утворення літаючої LiFi мережі, для перешкод 2×2 м, 2×3 м та 3×2 м становить 17 (за наявності 28 перешкод), 44 (за наявності 15 перешкод) та 63 (за наявності 28 перешкод) відповідно.

2.3 Розроблення методу керованого водоспаду для планування розміщення БПЛА літаючої LiFi мережі

2.3.1 Обґрунтування основних кроків реалізації методу керованого водоспаду для планування розміщення БПЛА літаючої LiFi мережі

Алгоритм керованого водоспаду передбачає можливість одночасного використання алгоритмів лівого і правого кутів, а також передбачає можливість поєднання під час прокладання LiFi маршруту вершин прямокутників, які перебувають у зоні видимості. Реалізація алгоритму відбувається у два кроки (рис. 2.8).

На першому кроці застосування алгоритму керованого водоспаду генерується множина можливих LiFi маршрутів від джерела даних (точки *A*) до їх споживача (точка *B*) з нанесеними на маршрутах точками (місцями) розміщення БПЛА.

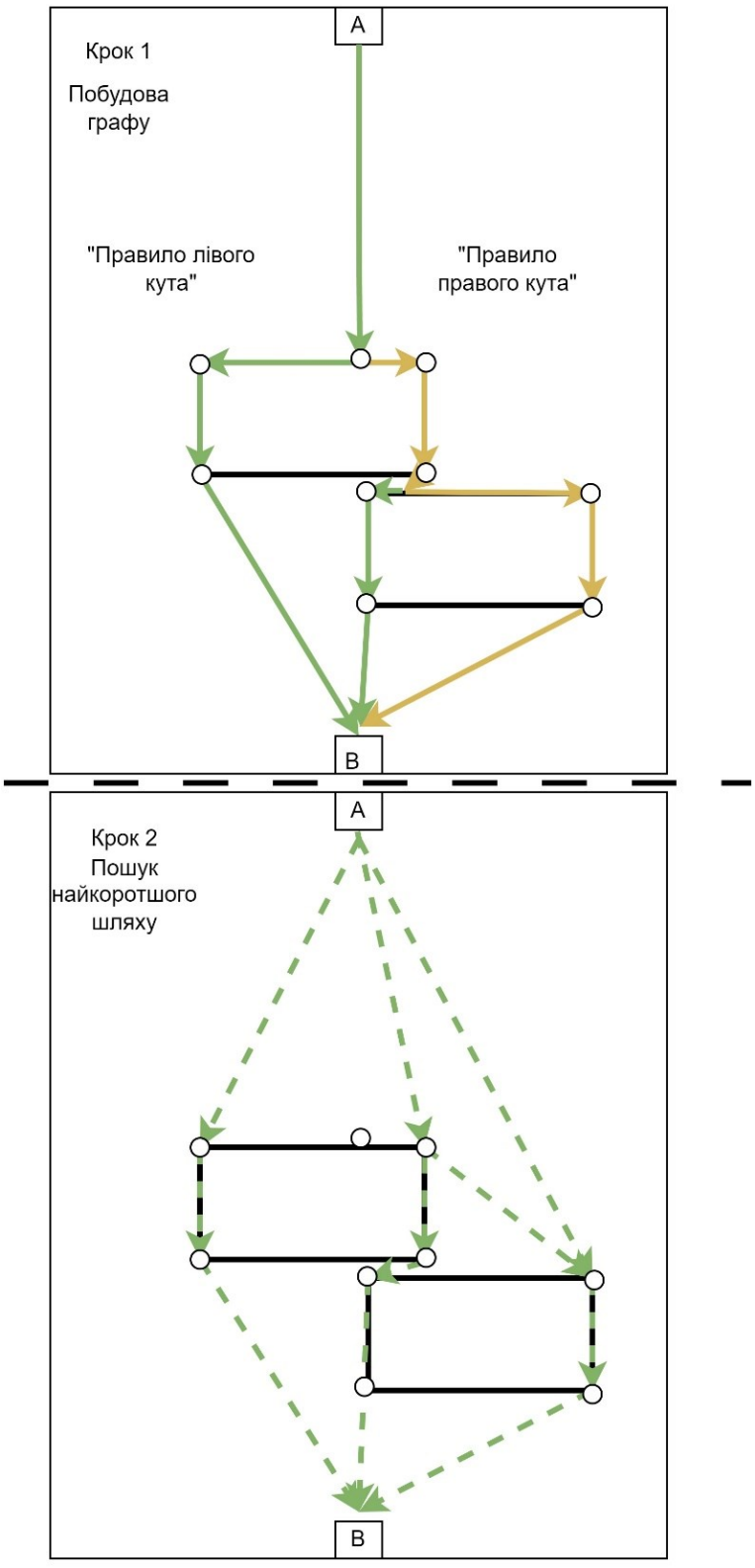

Рисунок 2.8 – Ілюстрація кроків реалізації алгоритму керованого водоспаду

На другому кроці формується граф можливих LiFi маршрутів, у якому присутні всі точки можливого розміщення БПЛА, і для пошуку найкоротшого LiFi маршруту застосовується алгоритм Дейкстри.

2.3.2 Планування розміщення БПЛА літаючої LiFi мережі з використанням алгоритму керованого водоспаду

Розглянемо різні варіанти застосування алгоритму керованого водоспаду, сутність якого розглянута у п. 2.3.1.

З використанням програмного засобу “Simulation Way” проведемо моделювання процесу планування розміщення БПЛА LiFi мережі для забезпечення передачі даних від точки *A* (джерело даних) до точки *B* (споживач даних) в робочій зоні приміщення з руйнуваннями з використанням алгоритму лівого кута для тих же вхідних даних що і в п. 2.2.2.

Результати моделювання показано на рис. 2.9.

Зеленим кольором показані маршрути, отримані при застосуванні алгоритму правого (лівого) кута, а червоним – додатковий маршрут, окремі ділянки якого представляють собою відрізки між вершинами прямокутників, що перебувають у зоні прямої видимості.

На наступному кроці був сформований граф можливих LiFi маршрутів, у якому присутні всі точки можливого розміщення БПЛА, і для пошуку найкоротшого LiFi маршруту застосований алгоритм Дейкстри. Результат роботи алгоритму показано на графі, представленому на рис. 2.10. Зеленим кольором на графі показані вершини, що відповідають початковій (*A*) та кінцевій (*B*) точкам найкоротшого прокладеного LiFi маршруту, а червоним – його проміжні точки – місця розміщення БПЛА для утворення літаючої LiFi мережі. Наявність на графі чотирьох червоних точок означає, що для розгортання літаючої LiFi мережі достатньо задіяти 4 БПЛА. Решта вершин, які враховувались, але не потрапили на найкоротший прокладений LiFi маршрут, показано синім кольором. Кожному ребру у відповідність поставлено його вагу, яка означає відстань між вершинами (точками можливого LiFi маршруту) у метрах.

Як і для алгоритмів правого та лівого кутів у подальшому було проведено 29 експериментів з використанням програмного засобу “Simulation Way” (номер експерименту відповідає кількості згенерованих у ньому перешкод) і отримано залежності:

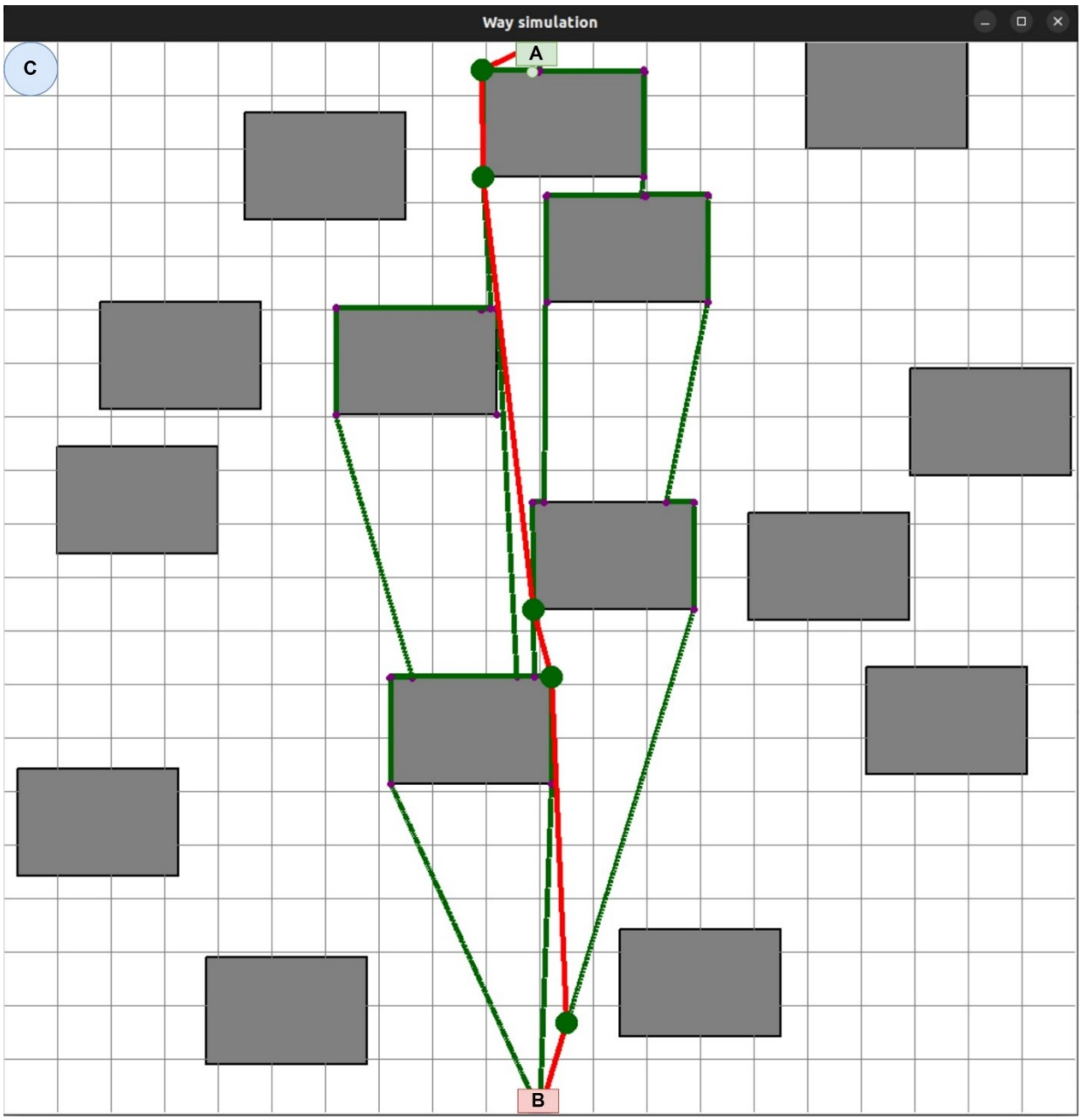

Рисунок 2.9 – Результат моделювання процесу планування розміщення БПЛА LiFi мережі для забезпечення передачі даних від точки *A* (джерело даних) до точки *B* (споживач даних) в робочій зоні приміщення з перешкодами з використанням алгоритму керованого водоспаду

– довжини прокладеного LiFi маршруту в робочій зоні приміщення від кількості перешкод (рис. 2.11) при застосуванні алгоритму керованого водоспаду;

– кількості БПЛА (кількості точок на прокладеному LiFi маршруті), яка необхідна для утворення літаючої LiFi мережі, від кількості перешкод при застосуванні алгоритму керованого водоспаду (рис. 2.12).

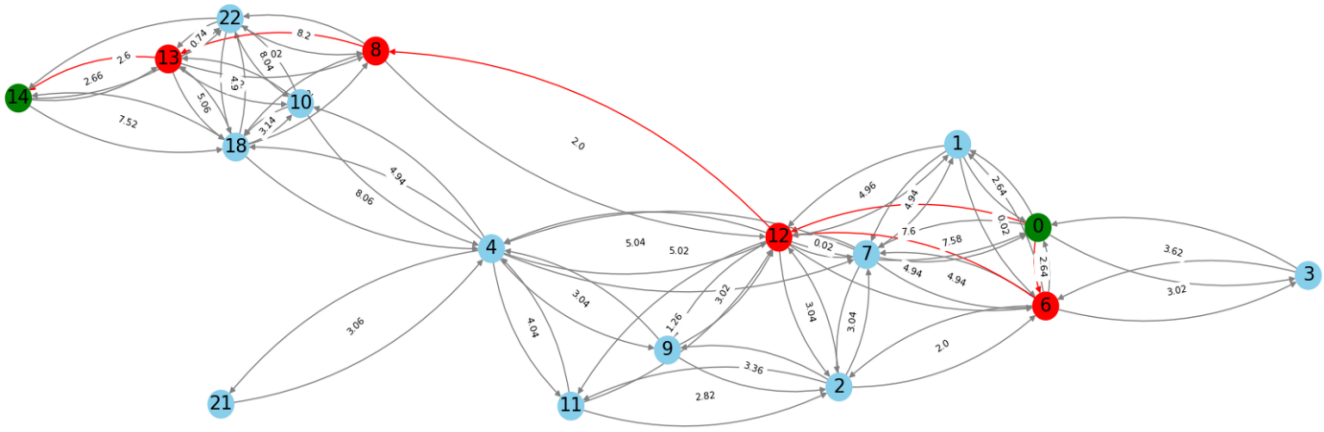

Рисунок 2.10 – Граф можливих LiFi маршрутів з визначеним на ньому шляхом застосування алгоритму Дейкстри найкоротшим маршрутом

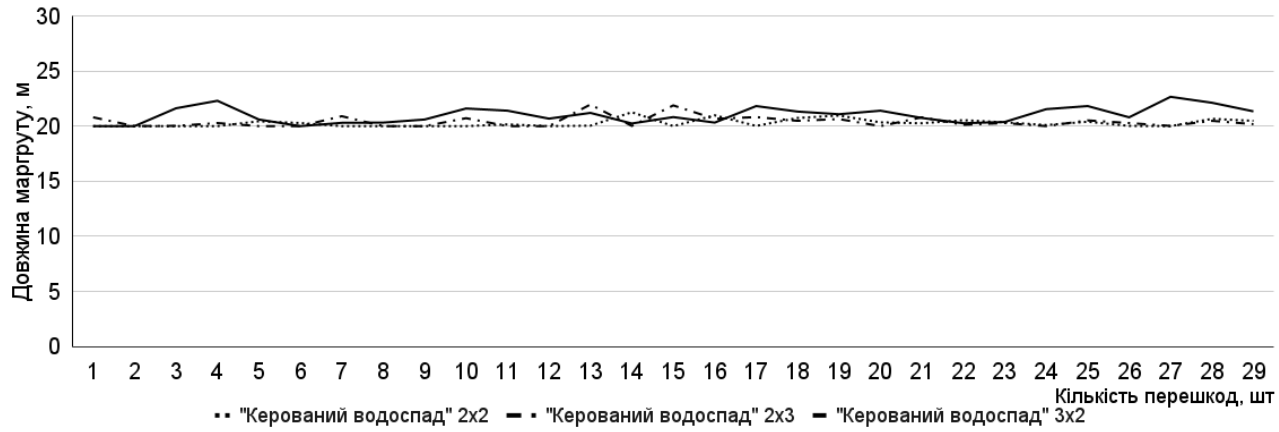

Рисунок 2.11 – Графік залежності довжини прокладеного LiFi маршруту в робочій зоні приміщення від кількості перешкод при застосуванні алгоритму керованого водоспаду

Аналіз графіків, представлених на рис. 2.11, дозволяє зробити наступні висновки.

1) За наявності перешкод 2×2 м у 11-ти випадках вдається прокласти найкоротший відносно інших перешкод LiFi маршрут. Для перешкод 2×3 м та 3×2 м така кількість випадків становить 11 та 1 відповідно.

2) За наявності 13 перешкод, перешкоди 2×2 м у порівнянні з перешкодами 2×3 м дозволяють прокласти LiFi маршрут довжиною, меншою на 1,92 м (20,04 м проти 21,2 м).

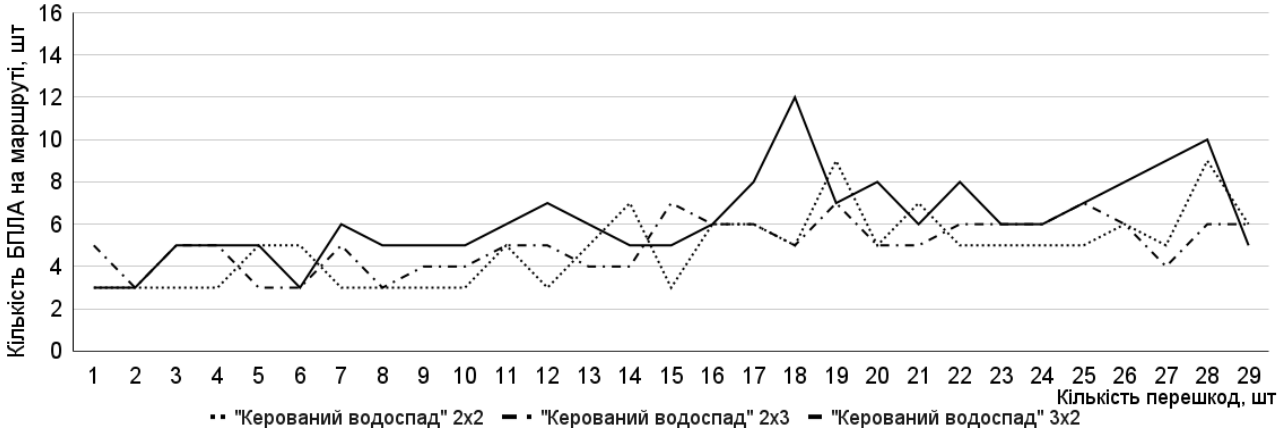

Рисунок 2.12 – Графік залежності кількості БПЛА (кількості точок на прокладеному LiFi маршруті), яка необхідна для утворення літаючої LiFi мережі, від кількості перешкод при застосуванні алгоритму керованого водоспаду

3) За наявності 27 перешкод, перешкоди 2×2 м у порівнянні з перешкодами 3×2 м дозволяють прокласти LiFi маршрут довжиною, меншою на 2,66 м (20 м проти 22,66 м).

4) Найкоротший для всіх перешкод LiFi маршрут становить 20 м. Такий маршрут можливо прокласти у 11-ти, 9-ти та 3-х випадках відповідно для перешкод 2×2 м, 2×3 м та 3×2 м.

5) Найдовший LiFi маршрут для перешкод 2×2 м, 2×3 м та 3×2 м становить 21,26 м (за наявності 14 перешкод), 21,96 м (за наявності 13 перешкод) та 22,66 м (за наявності 27 перешкод) відповідно.

Аналіз графіків, представлених на рис. 2.12, дозволяє зробити наступні висновки.

1) За наявності перешкод 2×2 м у 11-ти випадках вдається досягти найменшої відносно інших перешкод кількості БПЛА, необхідної для утворення літаючої LiFi мережі. Для перешкод 2×3 м та 3×2 м така кількість випадків становить 6 та 1 відповідно.

2) За наявності 15 перешкод, перешкоди 2×2 м у порівнянні з перешкодами 2×3 м дозволяють використати для утворення літаючої LiFi мережі на 4 БПЛА (3 проти 7) менше.

3) За наявності 18 перешкод, перешкоди 2×2 м у порівнянні з перешкодами 3×2 м дозволяють використати для утворення літаючої LiFi мережі на 7 БПЛА (5 проти 12) менше.

4) Найменша для всіх перешкод кількість БПЛА, необхідна для утворення літаючої LiFi мережі, становить 3. Вона можлива у 11-ти, 4-х та 3-х випадках відповідно для перешкод 2×2 м, 2×3 м та 3×2 м.

5) Найбільша кількість БПЛА, необхідна для утворення літаючої LiFi мережі, для перешкод 2×2 м, 2×3 м та 3×2 м становить 9 (за наявності 19 та 28 перешкод), 7 (за наявності 15, 19 та 25 перешкод) та 12 (за наявності 18 перешкод) відповідно.

2.4 Порівняльний аналіз розроблених алгоритмів

Для проведення порівняльного аналізу розроблених алгоритмів були використані раніше отримані з їх використанням залежності:

– довжини прокладеного LiFi маршруту в робочій зоні приміщення від кількості перешкод;

– кількості БПЛА (кількості точок на прокладеному LiFi маршруті), яка необхідна для утворення літаючої LiFi мережі, від кількості перешкод.

Графіки залежностей, що ілюструють результати проведеного порівняльного аналізу у разі наявності перешкод 2×2 м, представлено на рис. 2.13 та 2.14.

Аналіз графіків, представлених на рис. 2.13, дозволяє зробити наступні висновки.

1) При застосуванні алгоритму керованого водоспаду у 15-ти випадках вдається прокласти найкоротший LiFi маршрут у порівнянні з алгоритмами лівого та правого кутів. У 3-х випадках найкоротший LiFi маршрут отримуємо при застосування алгоритму правого кута. При застосуванні алгоритму лівого кута

жодного разу не вдалося прокласти найкоротший LiFi маршрут у порівнянні з алгоритмом правого кута чи алгоритмом керованого водоспаду.

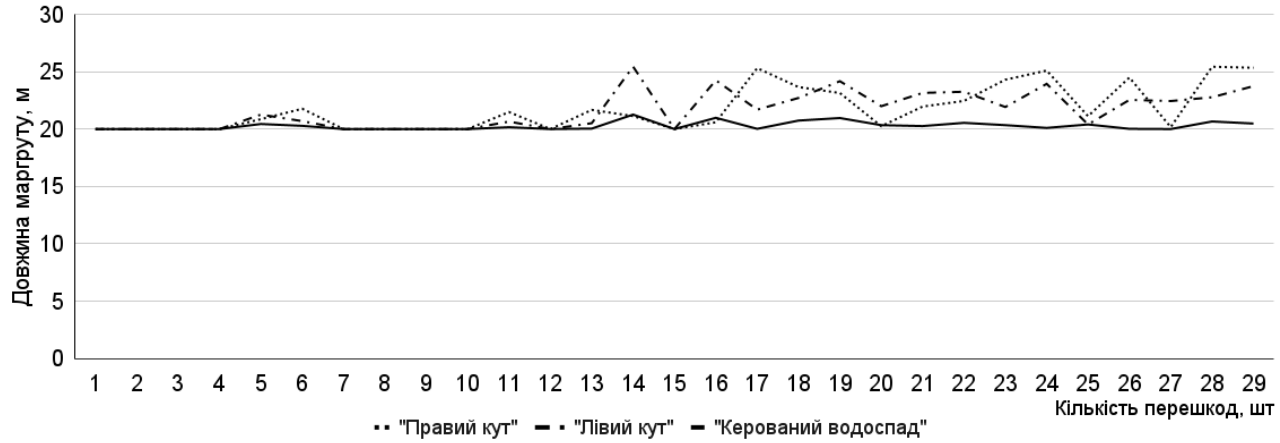

Рисунок 2.13 – Графік залежності довжини прокладеного LiFi маршруту в робочій зоні приміщення від кількості перешкод розмірами 2×2 м при застосуванні алгоритмів правого кута, лівого кута та керованого водоспаду

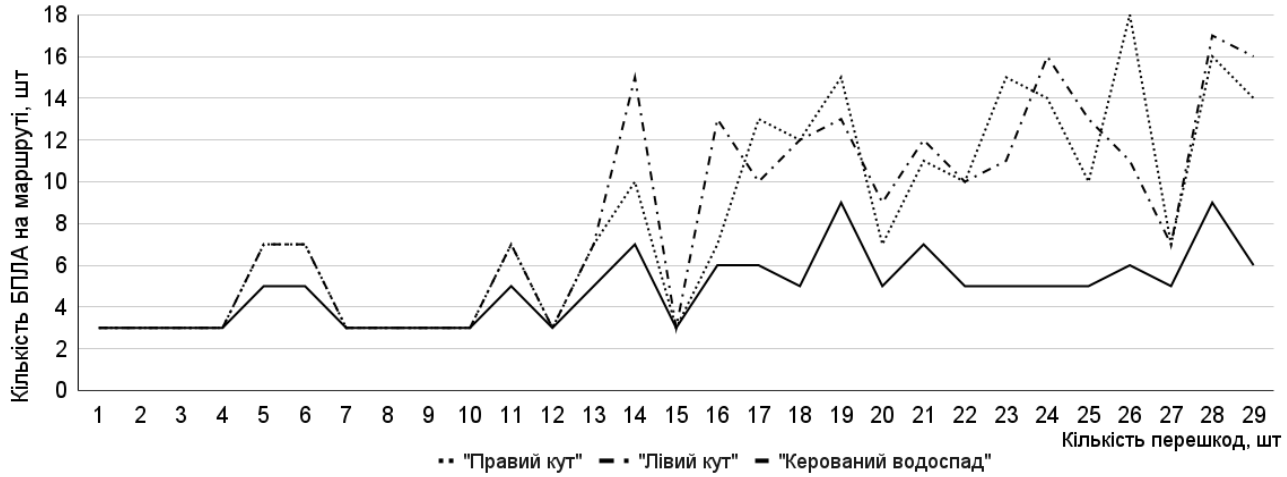

Рисунок 2.14 – Графік залежності кількості БПЛА (кількості точок на прокладеному LiFi маршруті), яка необхідна для утворення літаючої LiFi мережі, від кількості перешкод розмірами 2×3 м при застосуванні алгоритмів правого кута, лівого кута та керованого водоспаду

2) За наявності 17 перешкод, алгоритм керованого водоспаду у порівнянні з алгоритмом правого кута дозволяє прокласти LiFi маршрут довжиною, меншою на 5,29 м (20,02 м проти 25,31 м).

3) За наявності 14 перешкод, алгоритм керованого водоспаду у порівнянні з алгоритмом лівого кута дозволяє прокласти LiFi маршрут довжиною, меншою на 4,18 м (21,26 м проти 25,44 м).

4) Найкоротший LiFi маршрут для всіх алгоритмів становить 20 м. Такий маршрут можливо прокласти у 10-ти випадках при застосуванні алгоритмів правого кута або лівого кута та у 11-ти випадках при застосуванні алгоритму керованого водоспаду.

5) Найдовший LiFi маршрут для алгоритмів керованого водоспаду, правого кута та лівого кута становить 21,26 м (за наявності 14 перешкод), 25,43 м (за наявності 28 перешкод) та 25,44 м (за наявності 14 перешкод) відповідно.

Аналіз графіків, представлених на рис. 2.14, дозволяє зробити наступні висновки.

1) При застосуванні алгоритму керованого водоспаду у 19-ти випадках вдається досягти найменшої у порівняння з іншими алгоритмами кількості БПЛА, необхідної для утворення літаючої LiFi мережі. Ні застосування алгоритму правого кута, ні застосування алгоритму лівого кута жодного разу не дозволило отримати найменшу у порівнянні з іншими двома алгоритмами кількість БПЛА, необхідну для утворення літаючої LiFi мережі.

2) За наявності 26 перешкод, алгоритм керованого водоспаду у порівнянні з алгоритмом правого кута дозволяє використати для утворення літаючої LiFi мережі на 12 БПЛА (6 проти 18) менше.

3) За наявності 24 перешкод, алгоритм керованого водоспаду у порівнянні з алгоритмом лівого кута дозволяє використати для утворення літаючої LiFi мережі на 11 БПЛА (5 проти 16) менше.

4) Найменша розрахована кількість БПЛА, необхідна для утворення літаючої LiFi мережі, для всіх алгоритмів становить 3. Вона можлива у 10-х випадках при застосуванні кожного із алгоритмів.

5) Найбільша розрахована кількість БПЛА, необхідна для утворення літаючої LiFi мережі, при застосуванні алгоритмів керованого водоспаду, правого

кута і лівого кута становить 9 (за наявності 19 та 28 перешкод), 18 (за наявності 26 перешкод) та 17 (за наявності 28 перешкод) відповідно.

Графіки залежностей, що ілюструють результати проведеного порівняльного аналізу у разі наявності перешкод 2×3 м, представлено на рис. 2.15 та 2.16.

Аналіз графіків, представлених на рис. 2.15, дозволяє зробити наступні висновки.

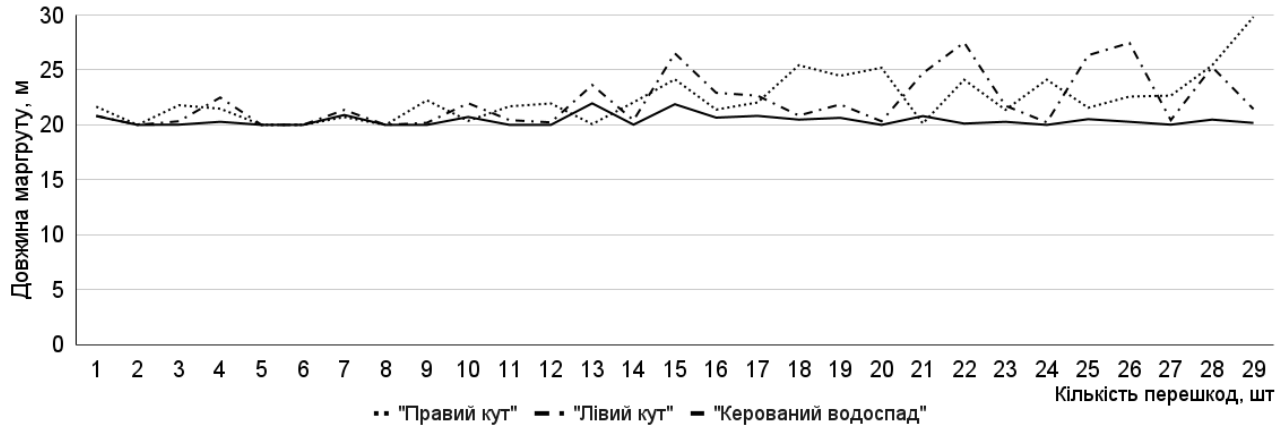

Рисунок 2.15 – Графік залежності довжини прокладеного LiFi маршруту в робочій зоні приміщення від кількості перешкод розмірами 2×3 м при застосуванні алгоритмів правого кута, лівого кута та керованого водоспаду

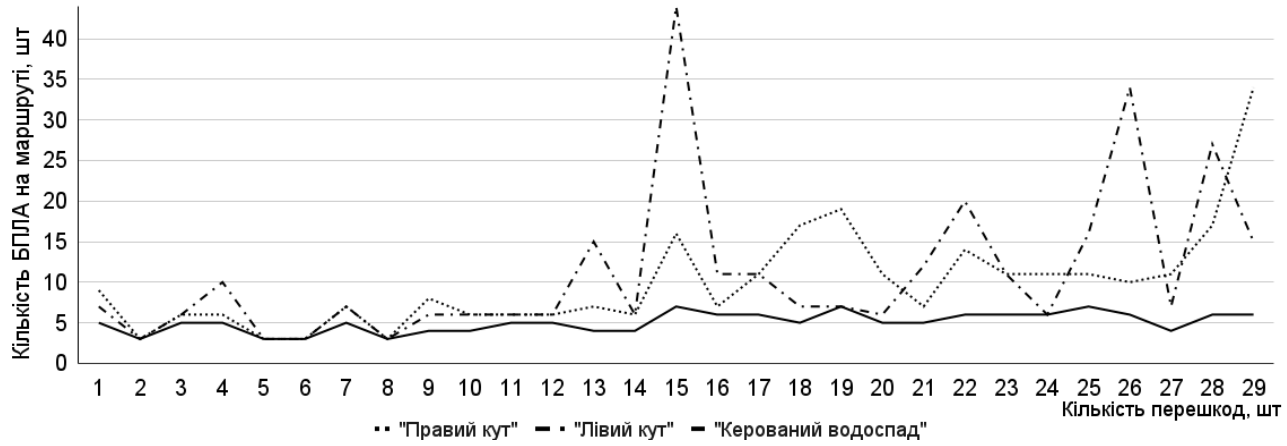

Рисунок 2.16 – Графік залежності кількості БПЛА (кількості точок на прокладеному LiFi маршруті), яка необхідна для утворення літаючої LiFi мережі, від кількості перешкод розмірами 2×3 м при застосуванні алгоритмів правого кута, лівого кута та керованого водоспаду

1) При застосуванні алгоритму керованого водоспаду у 21-му випадку вдається прокласти найкоротший LiFi маршрут у порівнянні з алгоритмами лівого та правого кутів. У 4-х випадках найкоротший LiFi маршрут отримуємо при застосування алгоритму правого кута. При застосуванні алгоритму лівого кута жодного разу не вдалося прокласти найкоротший LiFi маршрут у порівнянні з алгоритмом правого кута чи алгоритмом керованого водоспаду.

2) За наявності 29 перешкод, алгоритм керованого водоспаду у порівнянні з алгоритмом правого кута дозволяє прокласти LiFi маршрут довжиною, меншою на 9,65 м (20,18 м проти 29,83 м).

3) За наявності 22 перешкод, алгоритм керованого водоспаду у порівнянні з алгоритмом лівого кута дозволяє прокласти LiFi маршрут довжиною, меншою на 7,38 м (20,12 м проти 27,5 м).

4) Найкоротший LiFi маршрут для всіх алгоритмів становить 20 м. Такий маршрут можливо прокласти у 4-х випадках при застосуванні алгоритмів правого кута або лівого кута та у 9-ти випадках при застосуванні алгоритму керованого водоспаду.

5) Найдовший LiFi маршрут для алгоритмів керованого водоспаду, правого кута та лівого кута становить 21,96 м (за наявності 13 перешкод), 29,83 м (за наявності 29 перешкод) та 27,5 м (за наявності 22 перешкод) відповідно.

Аналіз графіків, представлених на рис. 2.16, дозволяє зробити наступні висновки.

1) При застосуванні алгоритму керованого водоспаду у 23-х випадках вдається досягти найменшої у порівняння з іншими алгоритмами кількості БПЛА, необхідної для утворення літаючої LiFi мережі. Ні застосування алгоритму правого кута, ні застосування алгоритму лівого кута жодного разу не дозволило отримати найменшу у порівнянні з іншими двома алгоритмами кількість БПЛА, необхідну для утворення літаючої LiFi мережі.

2) За наявності 29 перешкод, алгоритм керованого водоспаду у порівнянні з алгоритмом правого кута дозволяє використати для утворення літаючої LiFi мережі на 28 БПЛА (6 проти 34) менше.

3) За наявності 18 перешкод, алгоритм керованого водоспаду у порівнянні з алгоритмом лівого кута дозволяє використати для утворення літаючої LiFi мережі на 37 БПЛА (7 проти 44) менше.

4) Найменша розрахована кількість БПЛА, необхідна для утворення літаючої LiFi мережі, для всіх алгоритмів становить 3. Вона можлива у 4-х випадках при застосуванні кожного із алгоритмів.

5) Найбільша розрахована кількість БПЛА, необхідна для утворення літаючої LiFi мережі, при застосуванні алгоритмів керованого водоспаду, правого кута і лівого кута становить 7 (за наявності 15, 19 та 25 перешкод), 34 (за наявності 29 перешкод) та 44 (за наявності 15 перешкод) відповідно.

Графіки залежностей, що ілюструють результати проведеного порівняльного аналізу у разі наявності перешкод 3×2 м, представлено на рис. 2.17 та 2.18.

Аналіз графіків, представлених на рис. 2.17, дозволяє зробити наступні висновки.

1) При застосуванні алгоритму керованого водоспаду у 21-му випадку вдається прокласти найкоротший LiFi маршрут у порівнянні з алгоритмами лівого та правого кутів. У 3-х випадках найкоротший LiFi маршрут отримуємо при застосування алгоритму правого кута. При застосуванні алгоритму лівого кута жодного разу не вдалося прокласти найкоротший LiFi маршрут у порівнянні з алгоритмом правого кута чи алгоритмом керованого водоспаду.

2) За наявності 28 перешкод, алгоритм керованого водоспаду у порівнянні з алгоритмом правого кута дозволяє прокласти LiFi маршрут довжиною, меншою на 17,81 м (22,12 м проти 39,3 м).

3) За наявності 16 перешкод, алгоритм керованого водоспаду у порівнянні з алгоритмом лівого кута дозволяє прокласти LiFi маршрут довжиною, меншою на 12,62 м (20,32 м проти 32,94 м).

4) Найкоротший LiFi маршрут для всіх алгоритмів становить 20 м. Такий маршрут можливо прокласти у 3-х випадках при застосуванні кожного з алгоритмів.

5) Найдовший LiFi маршрут для алгоритмів керованого водоспаду, правого кута та лівого кута становить 22,66 м (за наявності 27 перешкод), 29,93 м (за наявності 28 перешкод) та 23,92 м (за наявності 28 перешкод) відповідно.

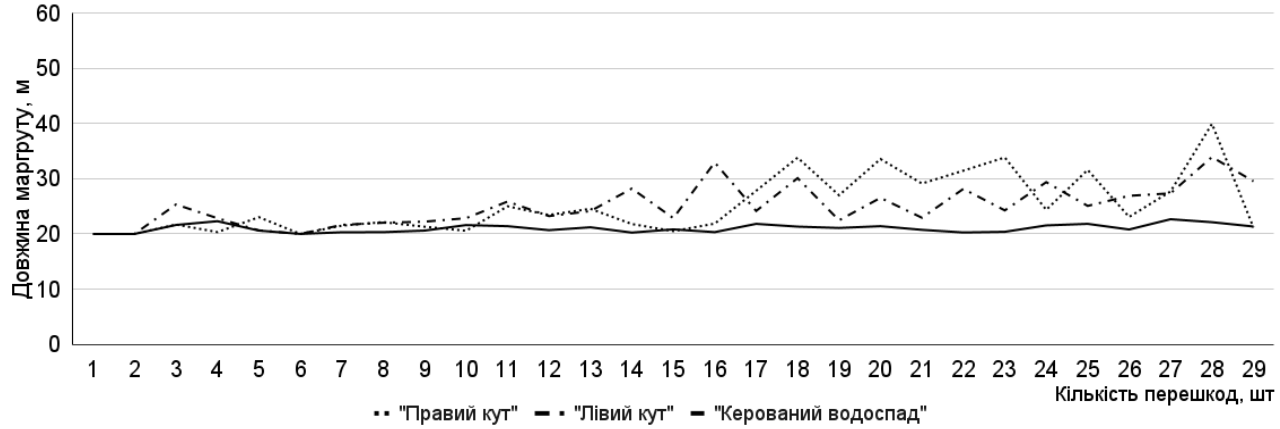

Рисунок 2.17 – Графік залежності довжини прокладеного LiFi маршруту в робочій зоні приміщення від кількості перешкод розмірами 3×2 м при застосуванні алгоритмів правого кута, лівого кута та керованого водоспаду

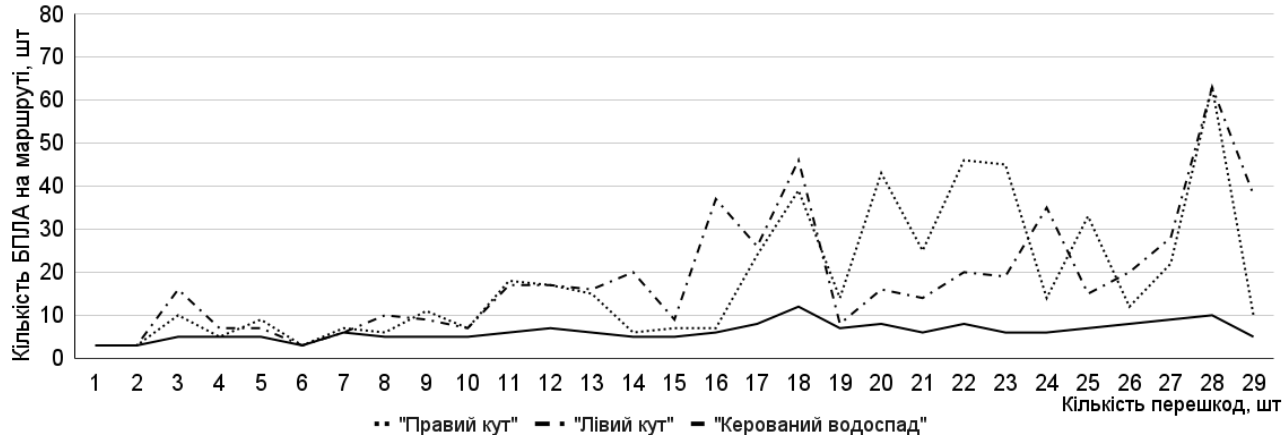

Рисунок 2.18 – Графік залежності кількості БПЛА (кількості точок на прокладеному LiFi маршруті), яка необхідна для утворення літаючої LiFi мережі, від кількості перешкод розмірами 3×2 м при застосуванні алгоритмів правого кута, лівого кута та керованого водоспаду

Аналіз графіків, представлених на рис. 2.18, дозволяє зробити наступні висновки.

1) При застосуванні алгоритму керованого водоспаду у 24-х випадках вдається досягти найменшої у порівняння з іншими алгоритмами кількості БПЛА, необхідної для утворення літаючої LiFi мережі. Ні застосування алгоритму правого кута, ні застосування алгоритму лівого кута жодного разу не дозволило отримати найменшу у порівнянні з іншими двома алгоритмами кількість БПЛА, необхідну для утворення літаючої LiFi мережі.

2) За наявності 28 перешкод, алгоритм керованого водоспаду у порівнянні з алгоритмами правого та лівого кутів дозволяє використати для утворення літаючої LiFi мережі на 53 БПЛА (10 проти 63) менше.

3) Найменша розрахована кількість БПЛА, необхідна для утворення літаючої LiFi мережі, для всіх алгоритмів становить 3. Вона можлива у 3-х випадках при застосуванні кожного із алгоритмів.

4) Найбільша розрахована кількість БПЛА, необхідна для утворення літаючої LiFi мережі, при застосуванні алгоритмів керованого водоспаду, правого кута і лівого кута становить 12 (за наявності 18 перешкод), 63 (за наявності 28 перешкод) та 63 (за наявності 28 перешкод) відповідно.

2.5 Висновки до другого розділу

1. Були розроблені і для заданих вхідних даних застосовані для обходу перешкод під час прокладання LiFi маршрутів такі алгоритми:

- алгоритм правого кута;
- алгоритм лівого кута;
- алгоритм керованого водоспаду.

2. Метою розроблення алгоритмів було забезпечити реалізацію наступних кроків, передбачених першим етапом схеми планування розгортання LiFi мережі на основі БПЛА у виробничому приміщенні з перешкодами:

крок 1. Підготовка вхідних даних, включаючи формулювання допущень і обмежень;

крок 2. Застосування методів обходу перешкод: методу прямокутників і методу керованого водоспаду;

крок 3. Генерація графу можливих LiFi маршрутів;

крок 4. Застосування алгоритму пошуку найкоротшого LiFi маршруту на графі, згенерованому на кроці 3;

крок 5. Оброблення отриманих результатів.

3. Було проведено порівняльний аналіз розроблених алгоритмів, в результаті якого встановлено:

– при застосуванні алгоритму керованого водоспаду у 15-ти випадках для перешкод 2×2 м та у 21 випадку для перешкод 2×3 м і 3×2 м вдається прокласти найкоротший LiFi маршрут у порівнянні з алгоритмами лівого та правого кутів. У 3-х випадках для перешкод 2×2 м і 3×2 м і 4-х випадках для перешкод 2×3 м у найкоротший LiFi маршрут отримуємо при застосування алгоритму правого кута;

– при застосуванні алгоритму лівого кута жодного разу не вдалося прокласти найкоротший LiFi маршрут у порівнянні з алгоритмом правого кута чи алгоритмом керованого водоспаду для всіх розмірів перешкод;

– при застосуванні алгоритму керованого водоспаду у 19-ти, 23-х та 24-х випадках для перешкод 2×2 м, 2×3 м та 3×2 м вдається досягти найменшої у порівняння з іншими алгоритмами кількості БПЛА, необхідної для утворення літаючої LiFi мережі;

– ні застосування алгоритму правого кута, ні застосування алгоритму лівого кута жодного разу не дозволило отримати найменшу у порівнянні з іншими двома алгоритмами кількість БПЛА, необхідну для утворення літаючої LiFi мережі.

РОЗДІЛ 3
РОЗРОБЛЕННЯ МЕТОДІВ РОЗМІЩЕННЯ БПЛА ЛІТАЮЧОЇ LIFI МЕРЕЖІ ТА ПІДВИЩЕННЯ ЇЇ НАДІЙНОСТІ ДЛЯ ЗАБЕЗПЕЧЕННЯ ПЕРЕДАЧІ ДАНИХ В УМОВАХ РУЙНУВАНЬ

3.1 Стратегії розгортання БПЛА із стаціонарного депо для утворення літаючої LiFi мережі в приміщенні з перешкодами

Метою даного розділу є розроблення стратегій розгортання БПЛА для утворення літаючої LiFi мережі у приміщеннях з перешкодами та забезпечення її безперебійного функціонування із визначеним рівнем надійності протягом заданого часу. Для реалізації заявленої мети необхідно виконати наступні завдання:

– проаналізувати різні варіанти розміщення БПЛА на прокладеному заздалегідь маршруті розповсюдження LiFi сигналу від джерела інформації до її споживча у приміщенні з перешкодами;

– запропонувати стратегії розгортання БПЛА із стаціонарного депо для утворення літаючої LiFi мережі;

– запропонувати методи забезпечення безперебійного функціонування літаючої LiFi мережі із необхідним рівнем надійності протягом заданого часу.

Під час опису стратегій приймемо наступні припущення:

– маршрут розповсюдження LiFi сигналу (LiFi маршрут) з точки *A* (джерело інформації) до точки *B* (споживач інформації), кількість БПЛА для утворення літаючої LiFi мережі та точки їх розміщення на маршруті вважаються визначеними заздалегідь (шляхом реалізації алгоритмів, описаних у розділі 2);

– місце базування БПЛА (депо), яке позначається точкою *C*, не співпадає з точками *A* і *B*, а також із жодною із точок розміщення БПЛА на LiFi маршруті;

– депо не змінює своє розташування з часом (є стаціонарним);

– кількість БПЛА є достатньою для покриття ними всіх визначених точок розміщення БПЛА на LiFi маршруті;

– використовуються однотипні БПЛА (під поняттям однотипні мається на увазі з однаковими характеристиками щодо автономності, швидкості, інтенсивності відмов тощо);

– час, необхідний для розгортання літаючої LiFi мережі визначається як сума часу прибуття останнього БПЛА із депо на визначене місце на LiFi маршруті і часу налаштування літаючої LiFi мережі.

Стратегії розгортання БПЛА для утворення літаючої LiFi мережі можна поділити на 2 групи.

1) *Індивідуальні*. Ця група стратегій передбачає рух БПЛА з депо до точок їх розміщення на LiFi маршруті за індивідуальними маршрутами відповідно до встановлених правил, включаючи правила обходу перешкод. Такі стратегії суттєво залежать від ємності бортової батареї кожного БПЛА, оскільки її ресурс, окрім забезпечення роботи літаючої LiFi мережі, додатково витрачається як на переміщення до визначеної точки, так і на повернення до депо.

2) *Колективні*. Ця група стратегій передбачає доставлення у визначену точку LiFi маршруту групи БПЛА за допомогою БПЛА-носія. Точка прибуття групи БПЛА може визначатися за критерієм часу розгортання мережі. Ці стратегії є більш складними, оскільки у набір правил руху та алгоритмів обходу перешкод вводиться додаткова сутність – БПЛА-носій. Однак його використання дозволяє зекономити ресурс батареї кожного БПЛА і забезпечити більш тривале функціонування LiFi мережі за одне розгортання.

Далі буде розглядатися лише група індивідуальних стратегій, а саме такі: стратегія першої точки маршруту, стратегія радіального руху, стратегія середньої точки маршруту.

3.1.1 Стратегія першої точки маршруту

Стратегія першої точки маршруту передбачає рух кожного БПЛА до першої точки LiFi маршруту і подальше розгортання мережі в межах нього (рис. 3.1). Ця стратегія може бути затребуваною у разі відсутності перешкод на

маршруті руху БПЛА від депо (точка *C*) до першої точки LiFi маршруту та наявності значної кількості перешкод на маршрутах руху БПЛА до інших його точок (необхідно прокладати додаткові маршрути в обхід цих перешкод).

Прибуваючи до першої точки, далі кожен БПЛА здійснює рух за LiFi маршрутом до точки призначення (визначеного для нього місця розміщення у складі LiFi мережі, що розгортається).

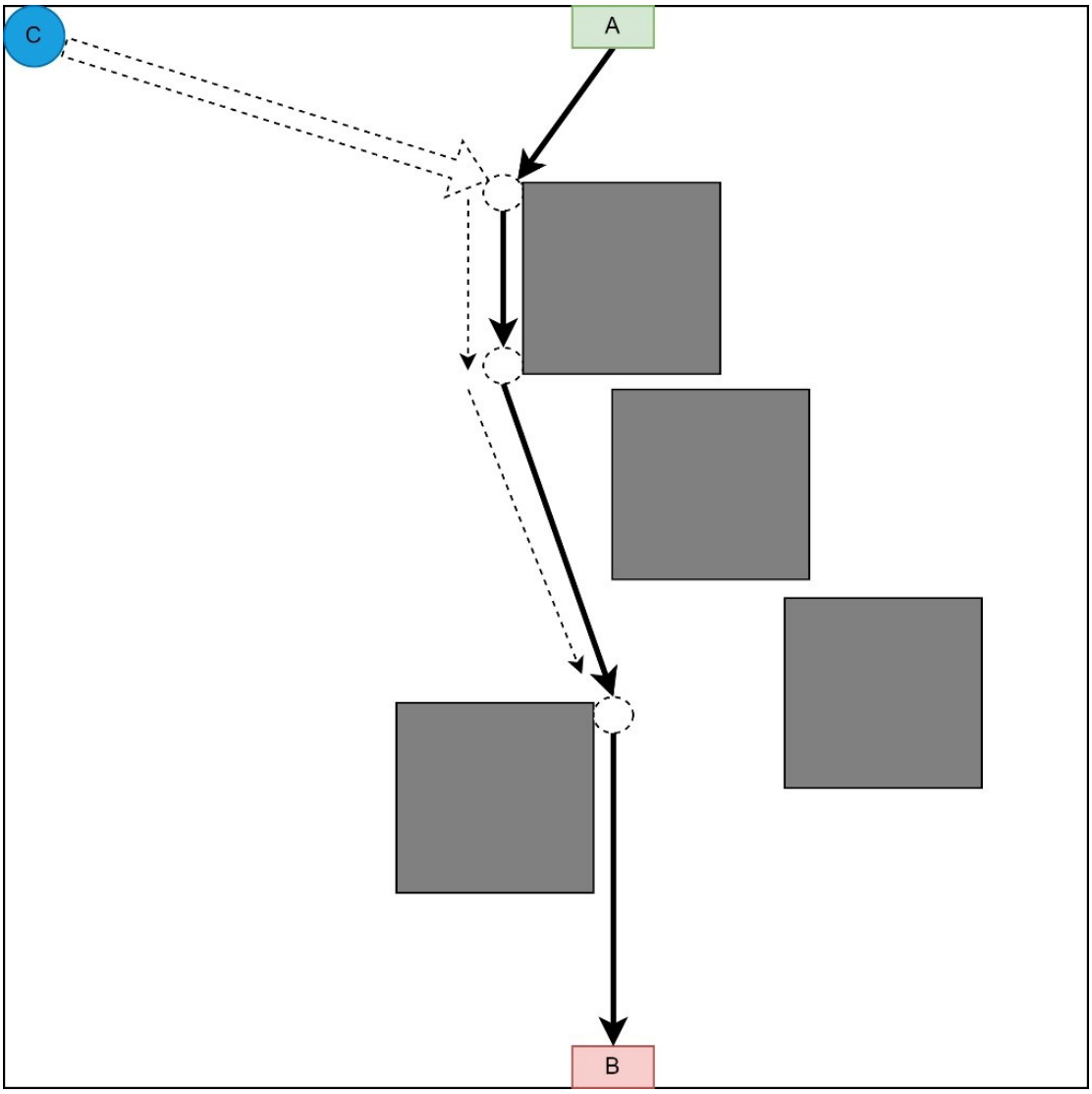

Рисунок 3.1 − Стратегія першої точки маршруту

Черговість вильотів БПЛА із депо (точка *C*) встановлюється у порядку зменшення відстані між депо і точкою призначення: першим вилітає найбільш віддалений від своєї точки призначення БПЛА і так далі. Ця стратегія

характеризуються значними витратами ресурсу бортової батареї БПЛА, найбільш віддаленого від свої точки призначення.

3.1.2 Стратегія радіального руху

Стратегія радіального руху передбачає рух кожного БПЛА одразу до точки призначення на LiFi маршруті (рис. 3.2).

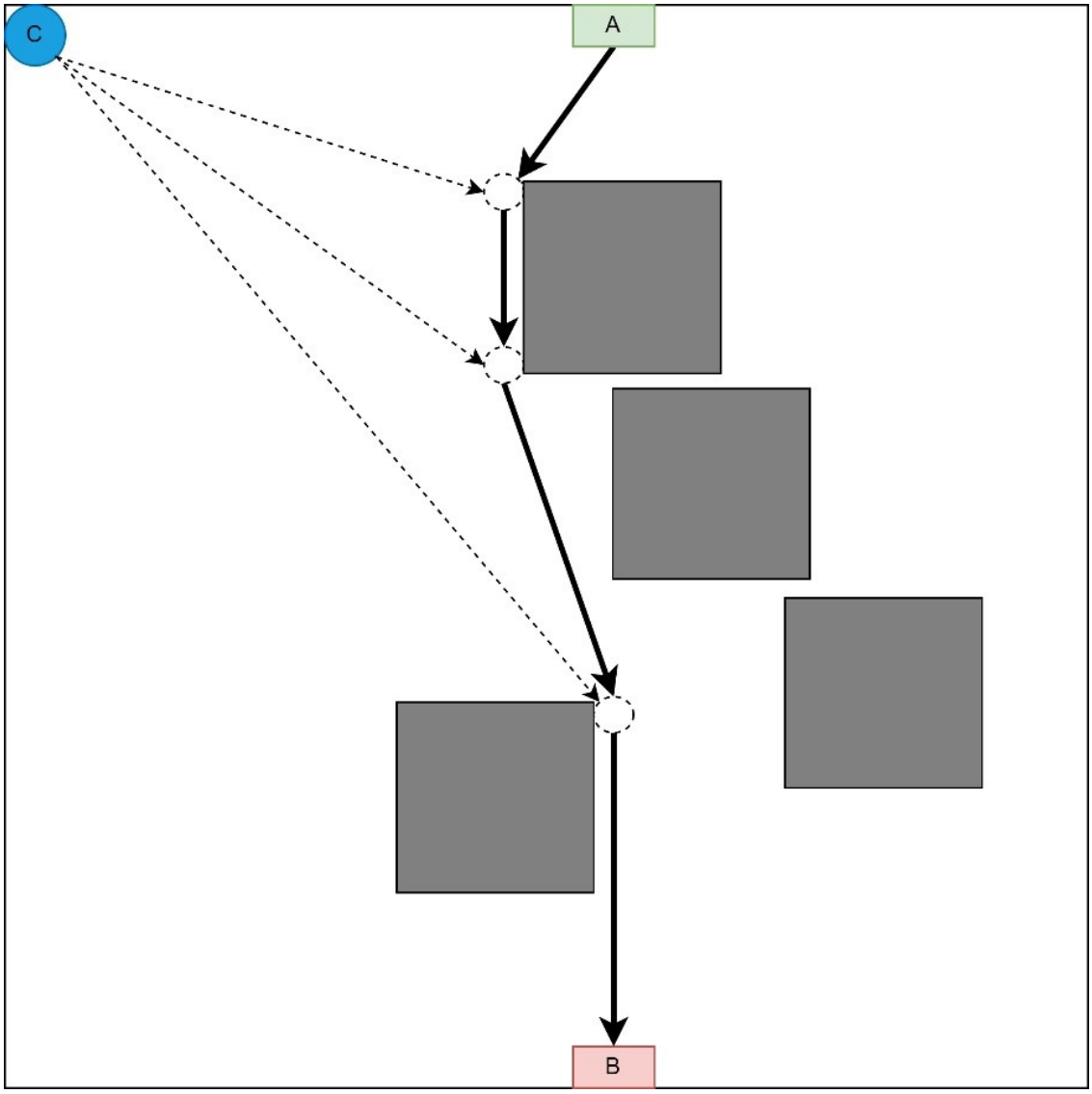

Рисунок 3.2 − Стратегія радіального руху

Цю стратегію доцільно застосовувати у випадку, коли на маршрутах руху більшості БПЛА з депо до точки призначення відсутні перешкоди. Ця стратегія

унеможливлює утворення черг у точках LiFi маршруту, оскільки кожен БПЛА рухається до своєї точки призначення за власним маршрутом. Черговість вильоту БПЛА із депо також відбувається у порядку зменшення відстані між депо і точкою призначення.

3.1.3 Стратегія середньої точки маршруту

Стратегія середньої точки маршруту передбачає рух кожного БПЛА до точки, максимально наближеної до середини LiFi маршруту, і подальше розгортання мережі в межах нього (рис. 3.3).

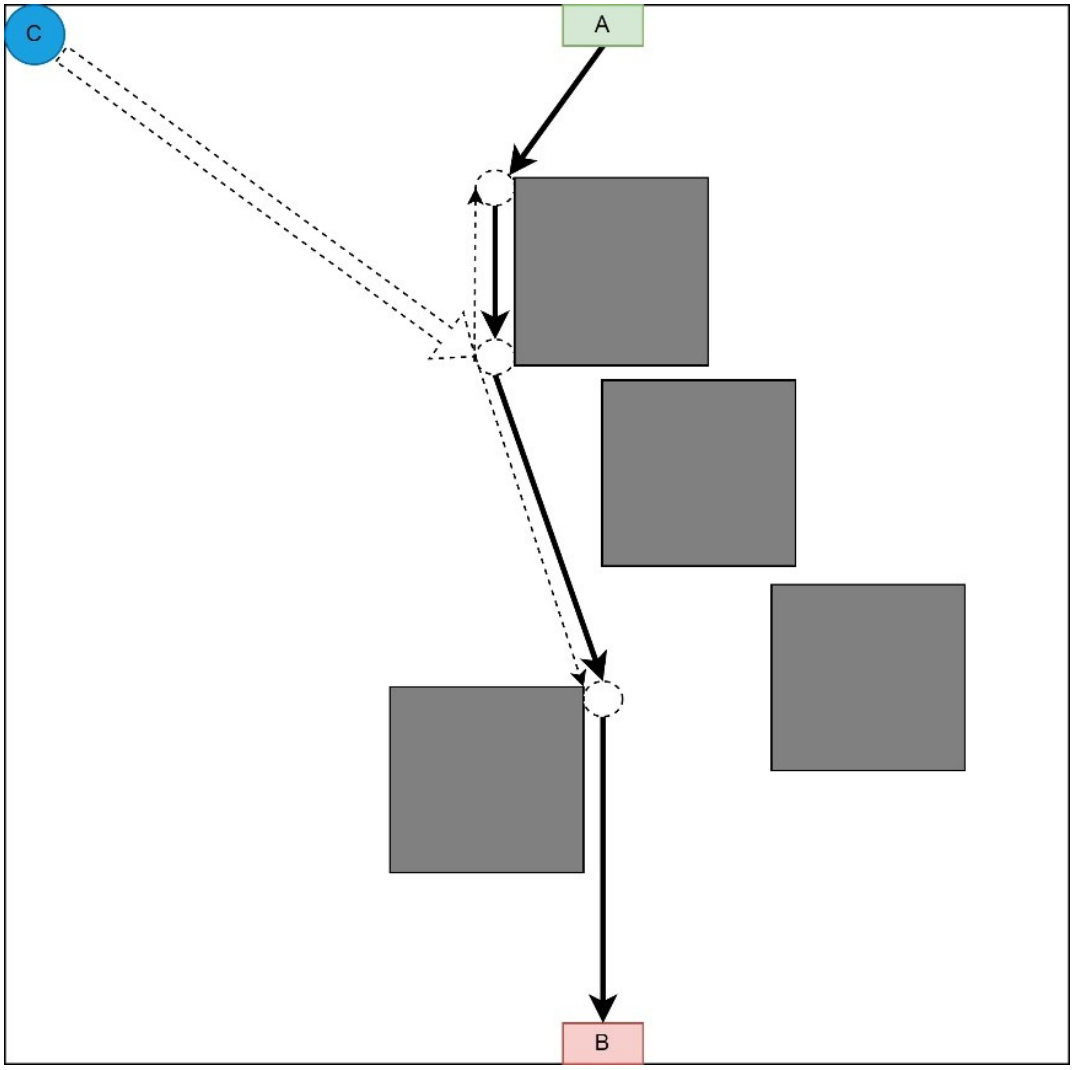

Рисунок 3.3 − Стратегія середньої точки маршруту

Ця стратегія є доцільною у разі відсутності перешкод на маршруті руху БПЛА від депо до цієї точки LiFi маршруту та наявності значної кількості перешкод на маршрутах руху БПЛА до інших його точок (необхідно прокладати додаткові маршрути в обхід цих перешкод). Прибуваючи до точки, максимально наближеної до середини LiFi маршруту, далі кожен БПЛА здійснює рух за LiFi маршрутом у напрямку точки призначення – або в напрямку точки *A* або в напрямку точки *B*. Черговість вильотів БПЛА із депо відбувається у порядку зменшення відстані між депо (точка *C*) і точкою призначення. Рух в обидві напрямки LiFi маршруту зменшить ймовірність утворення черг у цій точці.

3.2 Розроблення методів розгортання БПЛА із стаціонарного депо для утворення літаючої LiFi мережі в приміщенні з перешкодами

3.2.1 Розроблення та дослідження методу розгортання БПЛА відповідно до стратегії першої точки маршруту

Розробленню методу розгортання БПЛА згідно стратегії першої точки маршруту передувало прокладання LiFi маршруту і позначення на ньому точок розміщення БПЛА для утворення літаючої LiFi мережі з використанням для обходу перешкод алгоритму керованого водоспаду. Моделювання відбувалося з використанням програмного засобу “Simulation Way”. Результат моделювання проілюстровано на рис. 3.4, де показано:

− робочу площу виробничого приміщення з 20 прямокутними перешкодами розмірами 2×2 метри кожна;

− прокладений в обхід перешкод LiFi маршрут (показаний червоним кольором);

− точки розміщення БПЛА на LiFi маршруті для утворення літаючої LiFi мережі (показані зеленими точками). Ці точки отримувались з урахуванням обмеження на максимально можливу відстань між БПЛА (її збільшення може перевищити встановлену для заданих умов максимально можливу дальність LiFi сигналу);

– джерело інформації (позначене зеленим прямокутником з літерою *A* всередині);

– споживач інформації (позначений червоним прямокутником з літерою *B* всередині);

– депо для БПЛА (позначене синім колом з літерою *C* всередині).

Далі для розгортання літаючої LiFi мережі була застосована стратегія першої точки маршруту. Результати моделювання з використанням програмного засобу “Simulation Way” проілюстровані також на рис. 3.4, де синім кольором показані індивідуальні маршрути руху БПЛА до своїх точок призначення на LiFi маршруті.

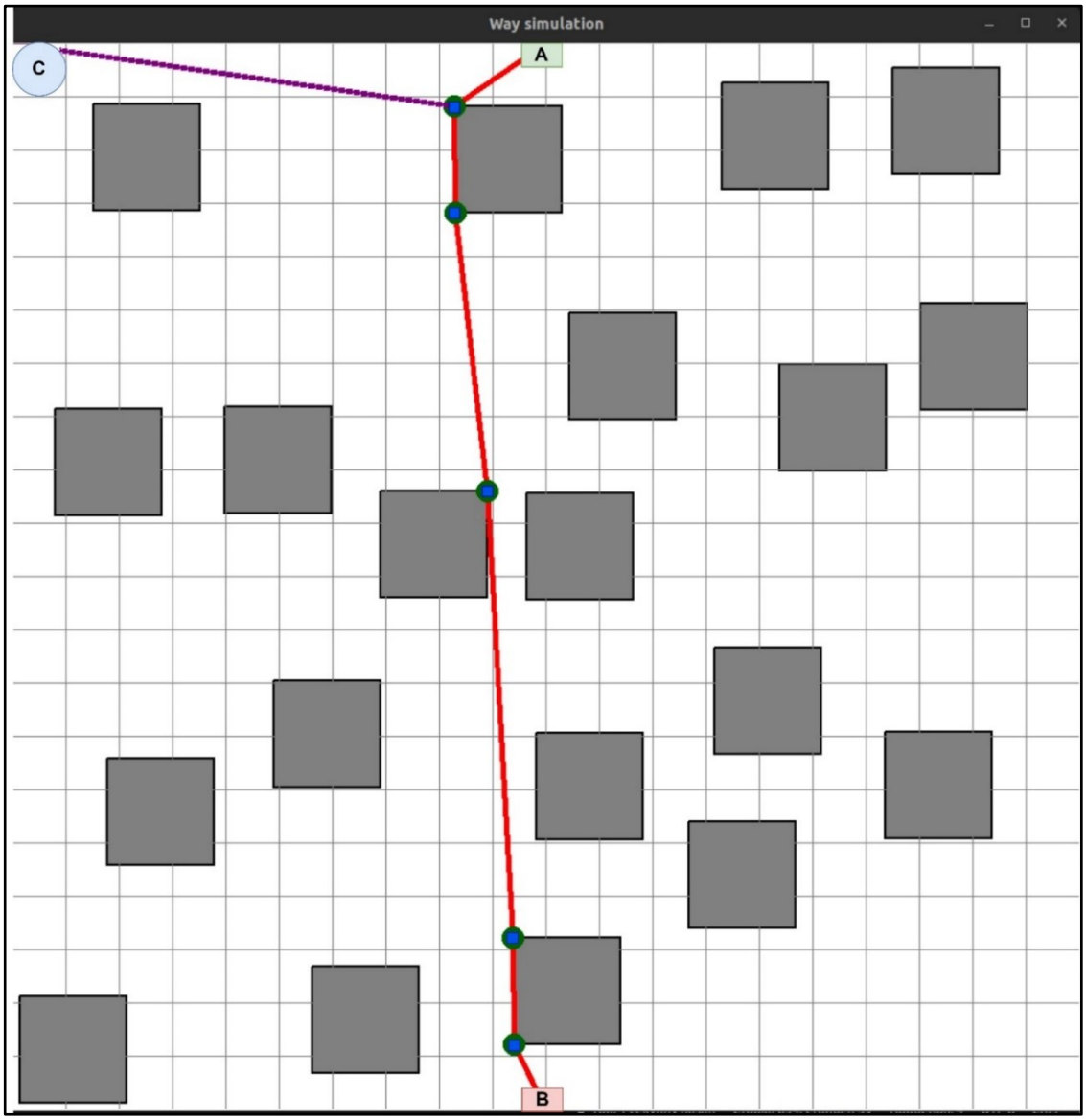

Рисунок 3.4 – Розгортання літаючої LiFi мережі відповідно до стратегії першої точки маршруту

Дані, які характеризують процес розгортання літаючої LiFi мережі за стратегією першої точки маршруту при швидкостях БПЛА 2 м/с, 3 м/с та 4 м/с відображені відповідно у таблицях 3.1, 3.2 та 3.3.

Таблиця 3.1 – Дані, які характеризують процес розгортання літаючої LiFi мережі за стратегією першої точки маршруту при швидкості БПЛА у 2 м/с

Номер БПЛА у черзі	Відстань до визначеної точки на LiFi маршруті, м	Час розміщення БПЛА на LiFi маршруті, с	Час налаштування літаючої LiFi мережі, с	Час розгортання літаючої LiFi мережі, с
1 БПЛА	25,99	13,00	30	43,00
2 БПЛА	23,99	12,00		
3 БПЛА	15,59	7,80		
4 БПЛА	10,34	5,17		
5 БПЛА	8,34	4,17		

Таким чином, за стратегією першої точки маршруту літаюча LiFi мережа із 5 БПЛА при швидкості БПЛА у 2 м/с може бути розгорнута за 43,00 с.

Таблиця 3.2 – Дані, які характеризують процес розгортання літаючої LiFi мережі за стратегією першої точки маршруту при швидкості БПЛА у 3 м/с

Номер БПЛА у черзі	Відстань до визначеної точки на LiFi маршруті, м	Час розміщення БПЛА на LiFi маршруті, с	Час налаштування літаючої LiFi мережі, с	Час розгортання літаючої LiFi мережі, с
1 БПЛА	25,99	8,66	30	38
2 БПЛА	23,99	8,00		
3 БПЛА	15,59	5,20		
4 БПЛА	10,34	3,45		
5 БПЛА	8,34	2,78		

Таким чином, за стратегією першої точки маршруту літаюча LiFi мережа із 5 БПЛА при швидкості БПЛА у 3 м/с може бути розгорнута за 38 с.

Таблиця 3.3 – Дані, які характеризують процес розгортання літаючої LiFi мережі за стратегією першої точки маршруту при швидкості БПЛА у 4 м/с

Номер БПЛА у черзі	Відстань до визначеної точки на LiFi маршруті, м	Час розміщення БПЛА на LiFi маршруті, с	Час налаштування літаючої LiFi мережі, с	Час розгортання літаючої LiFi мережі, с
1 БПЛА	25,99	6,50	30	36,5
2 БПЛА	23,99	6,00		
3 БПЛА	15,59	3,90		
4 БПЛА	10,34	2,59		
5 БПЛА	8,34	2,09		

Таким чином, за стратегією першої точки маршруту літаюча LiFi мережа із 5 БПЛА при швидкості БПЛА у 4 м/с може бути розгорнута за 36,5 с.

3.2.2 Розроблення та дослідження методу розгортання БПЛА відповідно до стратегії радіального руху

Для розгортання БПЛА відповідно до стратегії радіального руху був використаний той самий прокладений маршрут LiFi маршрут, що і в п. 3.2.1.

Далі для розгортання літаючої LiFi мережі була застосована стратегія радіального руху. Результати моделювання з використанням програмного засобу “Simulation Way” проілюстровані на рис. 3.5, де синім кольором показані індивідуальні маршрути руху БПЛА до своїх точок призначення на LiFi маршруті. Обхід перешкод під час прокладання маршрутів руху БПЛА з депо до точок призначення здійснювався або за алгоритмом керованого водоспаду.

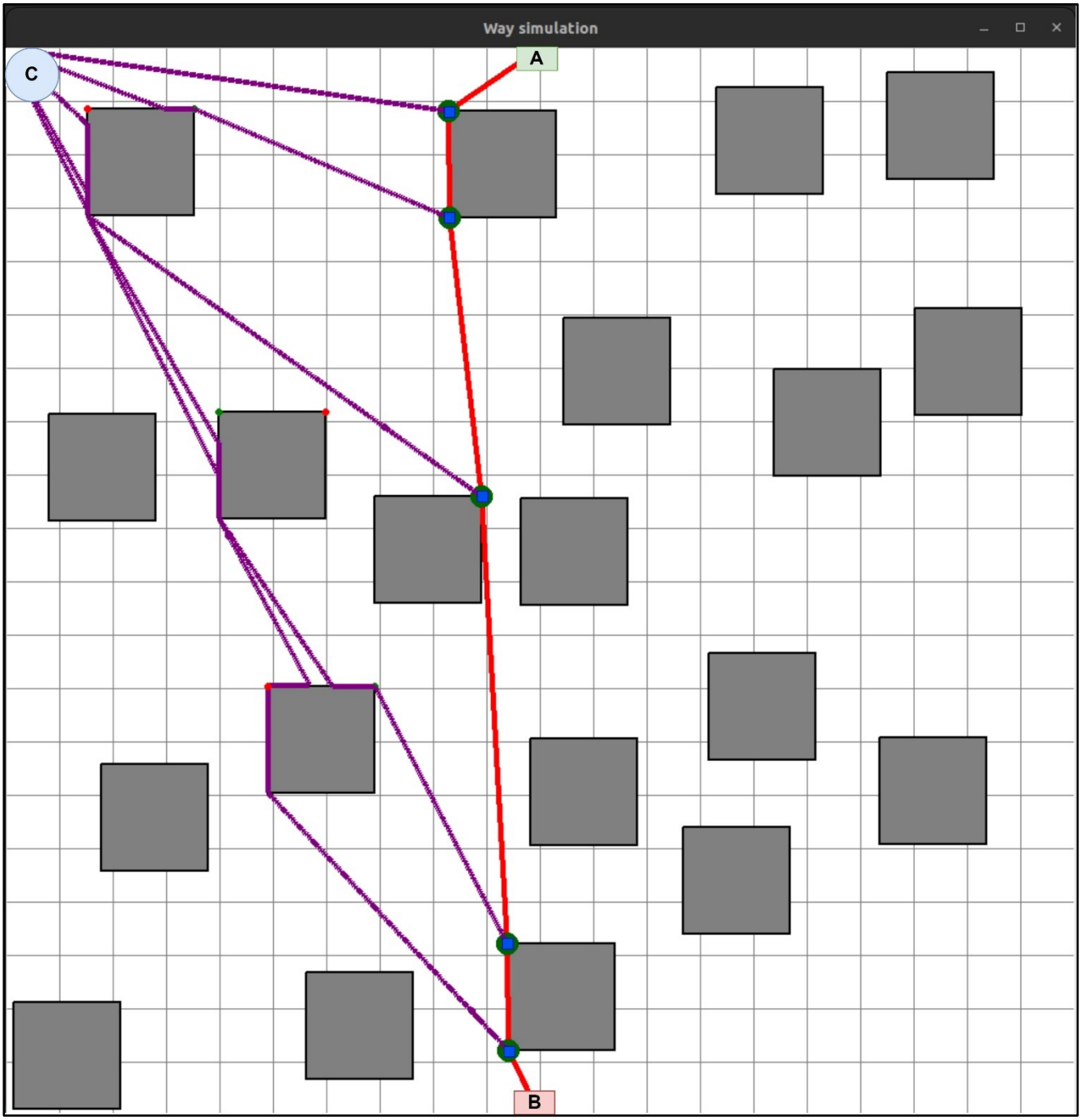

Рисунок 3.5 − Розгортання літаючої LiFi мережі відповідно до стратегії радіального руху

Дані, які характеризують процес розгортання літаючої LiFi мережі за стратегією радіального руху при швидкостях БПЛА 2 м/с, 3 м/с та 4 м/с відображені відповідно у таблицях 3.4, 3.5 та 3.6

Таблиця 3.4 – Дані, які характеризують процес розгортання літаючої LiFi мережі за стратегією радіального руху при швидкості БПЛА у 2 м/с

Номер БПЛА у черзі	Відстань до визначеної точки на LiFi маршруті, м	Час розміщення БПЛА на LiFi маршруті, с	Час налаштування літаючої LiFi мережі, с	Час розгортання літаючої LiFi мережі, с
1 БПЛА	22,76	11,38	30	41,38
2 БПЛА	19,86	9,93		
3 БПЛА	12,82	6,41		
4 БПЛА	8,89	4,45		
5 БПЛА	8,34	4,17		

Таким чином, за стратегією радіального руху літаюча LiFi мережа із 5 БПЛА при швидкості БПЛА у 2 м/с може бути розгорнута за 41,38 с.

Таблиця 3.5 – Дані, які характеризують процес розгортання літаючої LiFi мережі за стратегією радіального руху при швидкості БПЛА у 3 м/с

Номер БПЛА у черзі	Відстань до визначеної точки на LiFi маршруті, м	Час розміщення БПЛА на LiFi маршруті, с	Час налаштування літаючої LiFi мережі, с	Час розгортання літаючої LiFi мережі, с
1 БПЛА	22,76	7,59	30	37,59
2 БПЛА	19,86	6,62		
3 БПЛА	12,82	4,27		
4 БПЛА	8,89	2,96		
5 БПЛА	8,34	2,78		

Таким чином, за стратегією радіального руху літаюча LiFi мережа із 5 БПЛА при швидкості БПЛА у 3 м/с може бути розгорнута за 37,59 с.

Таблиця 3.6 – Дані, які характеризують процес розгортання літаючої LiFi мережі за стратегією радіального руху при швидкості БПЛА у 4 м/с

Номер БПЛА у черзі	Відстань до визначеної точки на LiFi маршруті, м	Час розміщення БПЛА на LiFi маршруті, с	Час налаштування літаючої LiFi мережі, с	Час розгортання літаючої LiFi мережі, с
1 БПЛА	22,76	5,69	30	35,69
2 БПЛА	19,86	4,97		
3 БПЛА	12,82	3,21		
4 БПЛА	8,89	2,22		
5 БПЛА	8,34	2,09		

Таким чином, за стратегією радіального руху літаюча LiFi мережа із 5 БПЛА при швидкості БПЛА у 4 м/с може бути розгорнута за 35,69 с.

3.2.3 Розроблення та дослідження методу розгортання БПЛА відповідно до стратегії середньої точки маршруту

Для розгортання БПЛА відповідно до стратегії середньої точки маршруту був використаний той самий прокладений маршрут LiFi маршрут, що і в п. 3.2.1.

Далі для розгортання літаючої LiFi мережі була застосована стратегія середньої точки маршруту. Результати моделювання з використанням програмного засобу “Simulation Way” проілюстровані на рис. 3.6, де синім кольором показані індивідуальні маршрути руху БПЛА до своїх точок призначення на LiFi маршруті. Обхід перешкод під час прокладання маршрутів руху БПЛА з депо до точок призначення здійснювався за алгоритмом керованого водоспаду.

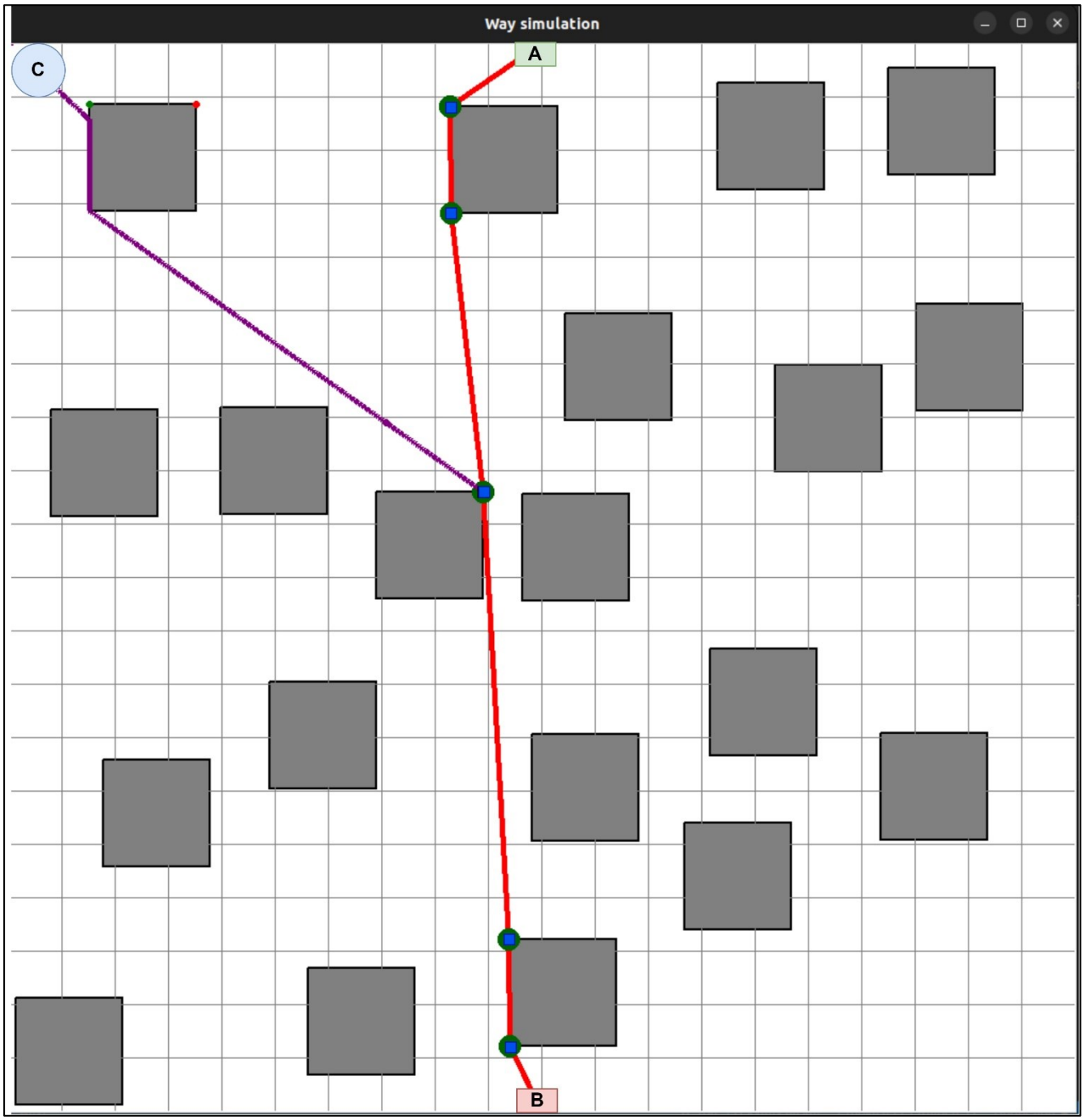

Рисунок 3.6 − Розгортання літаючої LiFi мережі відповідно до стратегії середньої точки маршруту

Дані, які характеризують процес розгортання літаючої LiFi мережі за стратегією середньої точки маршруту при швидкостях БПЛА 2 м/с, 3 м/с та 4 м/с відображені відповідно у таблицях 3.7, 3.8 та 3.9.

Таблиця 3.7 – Дані, які характеризують процес розгортання літаючої LiFi мережі за стратегією середньої точки маршруту при швидкості БПЛА у 2 м/с

Номер БПЛА у черзі	Відстань до визначеної точки на LiFi маршруті, м	Час розміщення БПЛА на LiFi маршруті, с	Час налаштування літаючої LiFi мережі, с	Час розгортання літаючої LiFi мережі, с
1 БПЛА	23,22	11,61	30	41,61
2 БПЛА	21,22	10,61		
3 БПЛА	26,47	13,24		
4 БПЛА	21,22	10,61		
5 БПЛА	12,82	6,41		

Таким чином, за стратегією середньої точки маршруту літаюча LiFi мережа із 5 БПЛА при швидкості БПЛА у 2 м/с може бути розгорнута за 41,61 с.

Таблиця 3.8 – Дані, які характеризують процес розгортання літаючої LiFi мережі за стратегією середньої точки маршруту при швидкості БПЛА у 3 м/с

Номер БПЛА у черзі	Відстань до визначеної точки на LiFi маршруті, м	Час розміщення БПЛА на LiFi маршруті, с	Час налаштування літаючої LiFi мережі, с	Час розгортання літаючої LiFi мережі, с
1 БПЛА	23,22	7,74	30	37,74
2 БПЛА	21,22	7,07		
3 БПЛА	26,47	8,82		
4 БПЛА	21,22	7,07		
5 БПЛА	12,82	4,27		

Таким чином, за стратегією середньої точки маршруту літаюча LiFi мережа із 5 БПЛА при швидкості БПЛА у 3 м/с може бути розгорнута за 37,74 с.

Таблиця 3.9 – Дані, які характеризують процес розгортання літаючої LiFi мережі за стратегією середньої точки маршруту при швидкості БПЛА у 4 м/с

Номер БПЛА у черзі	Відстань до визначеної точки на LiFi маршруті, м	Час розміщення БПЛА на LiFi маршруті, с	Час налаштування літаючої LiFi мережі, с	Час розгортання літаючої LiFi мережі, с
1 БПЛА	23,22	5,81	30	35,81
2 БПЛА	21,22	5,31		
3 БПЛА	26,47	6,62		
4 БПЛА	21,22	5,31		
5 БПЛА	12,82	3,21		

Таким чином, за стратегією середньої точки маршруту літаюча LiFi мережа із 5 БПЛА при швидкості БПЛА у 4 м/с може бути розгорнута за 35,81 с.

3.2.4 Порівняльний аналіз стратегій розгортання БПЛА для утворення літаючої LiFi мережі

Для порівняльного аналізу було побудовані графік залежності часу розгортання літаючої LiFi мережі від швидкості БПЛА для кожної із розглянутих стратегій, який представлено на рис. 3.7.

Аналіз графіку, представленого на рис. 3.7. дозволяє зробити наступні висновки.

1) Найбільше зменшення часу розгортання літаючої LiFi мережі внаслідок збільшення швидкості БПЛА з 2 м/с до 4 м/с спостерігається для стратегії першої точки маршруту і складає 6,5 с. Для стратегій радіального руху і середньої точки маршруту воно складає відповідно 5,69 с та 5,81 с відповідно. Якщо зробити ранжування стратегій у порядку збільшення виграшу у часі від збільшення

швидкості БПЛА, то результатом буде така послідовність: стратегія радіального руху, стратегія середньої точки маршруту, стратегія першої точки маршруту.

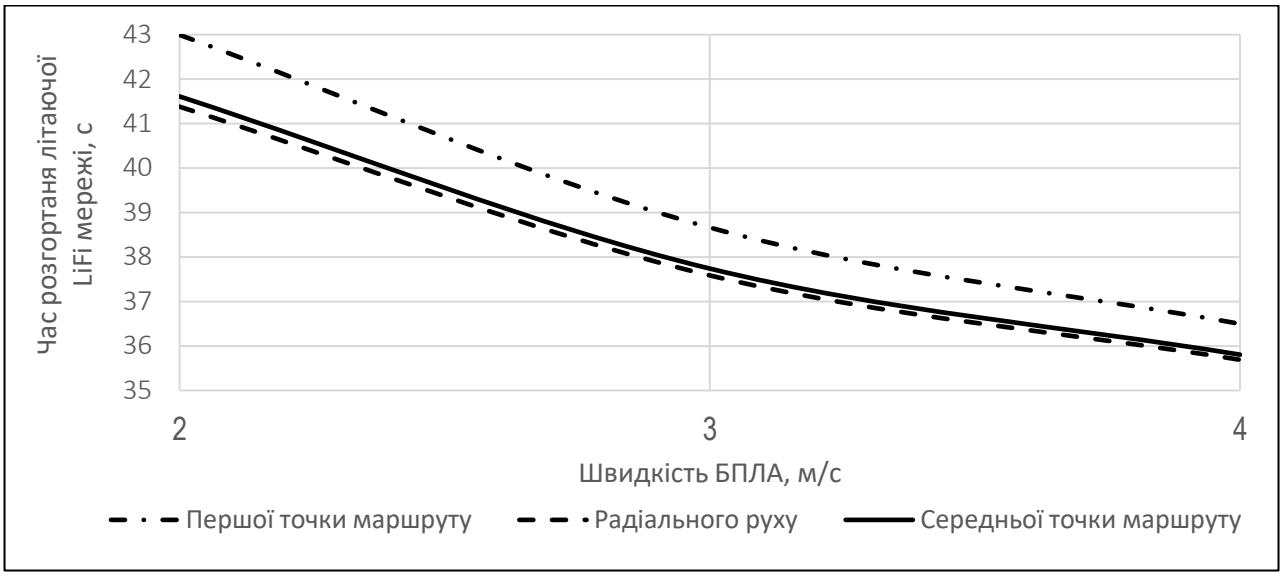

Рисунок 3.7 – Графік залежності часу розгортання літаючої LiFi мережі від швидкості БПЛА для стратегій першої точки маршруту, радіального руху і середньої точки маршруту

2) Для всіх розглянутих швидкостей БПЛА стратегія радіального руху забезпечує найменший час розгортання літаючої LiFi мережі. Так, наприклад, для швидкості 2 м/с цей час є на 1,39 с меншим за час розгортання літаючої LiFi мережі при застосуванні стратегії першої точки маршруту. Якщо зробити ранжування стратегій у порядку зменшення часу розгортання літаючої LiFi мережі, то результатом буде така послідовність: стратегія першої точки маршруту, стратегія середньої точки маршруту, стратегія радіального руху.

3.3 Розроблення та дослідження методу забезпечення необхідного рівня надійності літаючої LiFi мережі протягом заданого часу

Однією з вимог до розгорнутої літаючої LiFi мережі може бути така: забезпечити безперервну роботи мережі з ймовірністю безвідмовної роботи (ЙБР) $P(t)$ не нижче мінімально припустимого значення P_{min} протягом заданого часу. Це,

наприклад, може бути час передачі до споживача інформації даних з датчиків про стан обладнання, рівень забрудненості атмосфери в приміщенні, відеопотоку, який дає уявлення про ступінь пошкоджень у приміщенні, або наявність у ньому постраждалих осіб.

Під час розгляду методів забезпечення надійності літаючої LiFi мережі відповідно до вказаної вимоги приймемо наступні припущення:

– всі БПЛА є однотипними і характеризуються однаковими інтенсивностями відмов λ, 1/год;

– для забезпечення безперебійної роботи літаючої LiFi мережі використовуються дві зміни БПЛА з однаковою кількістю БПЛА в кожній з них n, причому ця кількість дорівнює кількості точок їх розміщення на прокладеному LiFi маршруті;

– працююча зміна БПЛА повинна бути замінена іншою зміною не пізніше часу t_{crit} досягнення працюючою зміною мінімально припустимого значення ЙБР P_{min} навіть за наявності ресурсу батареї БПЛА t_{batt}, достатнього для продовження їх функціонування у складі літаючої LiFi мережі;

– кожна зі змін перед початком першого циклу своєї роботи характеризується ЙБР $P_0 = 1$;

– з кожного наступного циклу роботи зміна починає свою роботу з ЙБР, якої вона досягла на момент повернення до депо;

– після повернення до депо ресурс батареї поновлюється.

Планування вильотів змін для утворення літаючої LiFi мережі відбувається з урахуванням того, що кожна з цих змін може перебувати у двох станах – стані функціонування у складі LiFi мережі, тривалість перебування у якому розраховується за формулою (3.1), і стані очікування, у якому зміна перебуває протягом часу t_{wait}.

$$t_{funct} = t_{set} + t_{transm}, \tag{3.1}$$

де t_{set} – час налаштування літаючої LiFi мережі;

t_{transm} – час функціонування літаючої LiFi мережі в режимі передачі даних.

Перехід із стану очікування у стан функціонування триває протягом часу перельоту БПЛА із депо до своїх точок на LiFi маршруті t_{arriv}, а із стану функціонування у стан очікування – протягом часу повернення БПЛА з LiFi маршруту до депо t_{return}. Таким чином, повний цикл роботи зміни може мати вигляд, представлений на рис. 3.8, а його тривалість буде розраховуватись за формулою:

$$t_{shift} = t_{arriv} + t_{set} + t_{transm} + t_{return}; t_{batt} > t_{shift}. \quad (3.2)$$

ЙБР кожної зміни на момент повернення до депо у першому циклі роботи може бути розрахована за формулою (3.2), а у другому та подальшому циклах ($k = 2, \dots, m$) – за формулою (3.3);

$$P_1(t_{shift}) = P_0 \times e^{-n\lambda t_{shift}};\ P_1(t_{shift}) > P_{min}. \quad (3.3)$$

$$P_k(t_{shift}) = P_{k-1} \times e^{-n\lambda t_{shift}};\ P_k(t_{shift}) > P_{min}. \quad (3.4)$$

Таким чином, кожна зміна може працювати не більше часу t_{crit_1} (формула (3.5)) у першому циклі і часу t_{crit_k} (формула (3.6)) – у другому і подальших циклах.

$$t_{crit\_1} = \frac{\ln(\frac{P_0}{P_{min}})}{n\lambda}. \quad (3.5)$$

$$t_{crit\_k} = \frac{\ln(\frac{P_{k-1}}{P_{min}})}{n\lambda}. \quad (3.6)$$

Для забезпечення безперервності передачі даних необхідно забезпечити виліт наступної зміни на $(t_{arriv} + t_{set})$ раніше часу $(t_{shift} - t_{return})$, що проілюстровано на рис. 3.9.

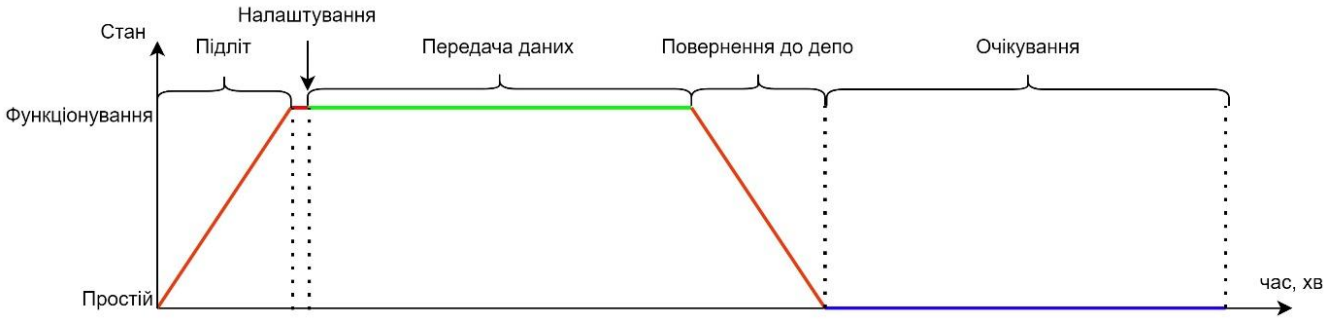

Рисунок 3.8 − Цикл роботи однієї зміни для розгортання і забезпечення функціонування LiFi

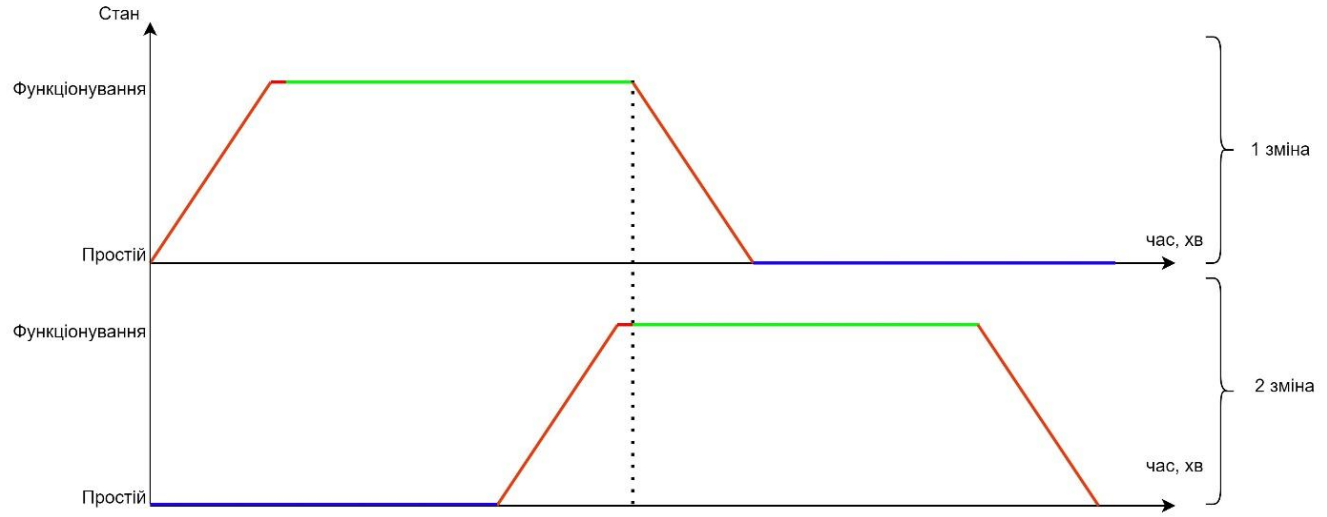

Рисунок 3.9 − Почергова робота двох змін для розгортання і забезпечення функціонування літаючої LiFi мережі

Подальшим кроком досліджень стало розроблення програмного засобу і моделювання з його використанням процесу забезпечення протягом заданого часу безперервної роботи мережі з ЙБР не нижче мінімально припустимого значення шляхом використання двох змін по n БПЛА у кожній.

Розглянемо випадок, коли кількість БПЛА у кожній зміні, яка відповідно до прийнятих раніше припущень дорівнює кількості точок на LiFi маршруті, складає 6 (n = 6 БПЛА). Нехай задано наступні вхідні дані: інтенсивність відмов БПЛА λ = 0,0005 1/год; P_0 = 1; P_{min} = 0,99875.

Використовуючи програмний засіб “Reliability Level” (більш детально буде розглянутий у розділі 4), зокрема інтегровану у нього бібліотеку *Gnuplot*, були побудовані графіки зменшення ЙБР літаючої LiFi мережі до мінімально допустимого значення при почерговому забезпеченні функціонування літаючої LiFi мережі для першої (рис. 3.10), другої (рис. 3.11) і двох (рис. 3.12) змін одночасно.

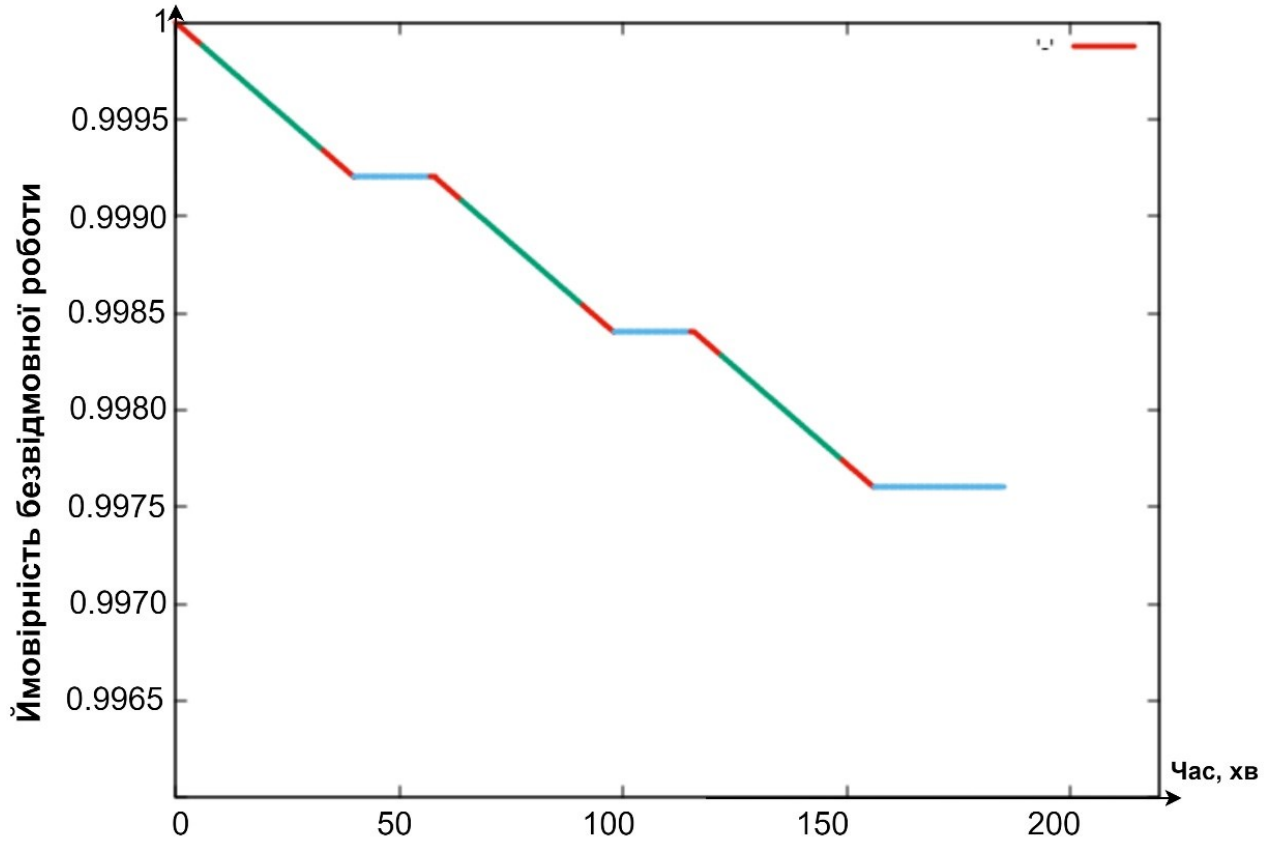

Рисунок 3.10 − Графік залежності ймовірності безвідмовної роботи першої зміни від часу

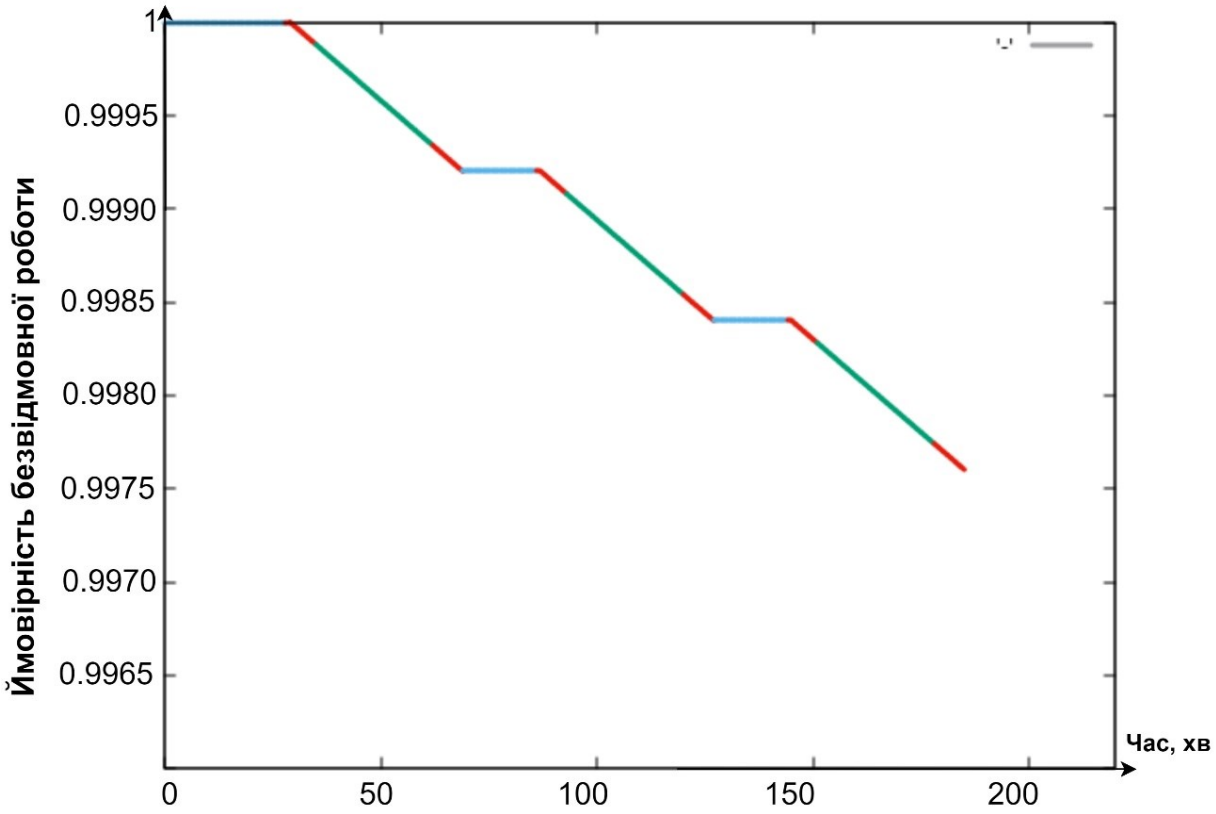

Рисунок 3.11 − Графік залежності ймовірності безвідмовної роботи другої зміни від часу

Кольорові відрізки графіків мають наступний зміст:

– відрізки червоного кольору демонструють зниження ЙБР як за час перельоту БПЛА до своїх точок на LiFi маршруті і налаштування літаючої LiFi мережі (верхні червоні відрізки), так і за час повернення БПЛА до депо (ніжні червоні відрізки);

– відрізки зеленого кольору ілюструють зниження ЙБР за час передачі даних від джерела до споживача даних;

– відрізки голубого кольору (тільки для рис. 3.10 і 3.11) показують ЙБР під час під час очікування зміною БПЛА на свій наступний цикл роботи.

Аналіз графіків, представлених на рис. 3.10−3.12, дозволяє зробити такі висновки:

– найбільше зниження ЙБР спостерігається під час передачі даних;

– під час очікування зміною БПЛА на свій наступний цикл роботи її ЙБР залишається незмінною;

– безперебійне функціонування літаючої LiFi мережі із необхідним рівнем надійності протягом 180 хвилин можна забезпечити двома змінами, кожна з яких складається із 6-ти БПЛА і виконує по три цикли роботи;

– останньою забезпечує функціонування літаючої LiFi мережі друга зміна у третьому циклі своєї роботи.

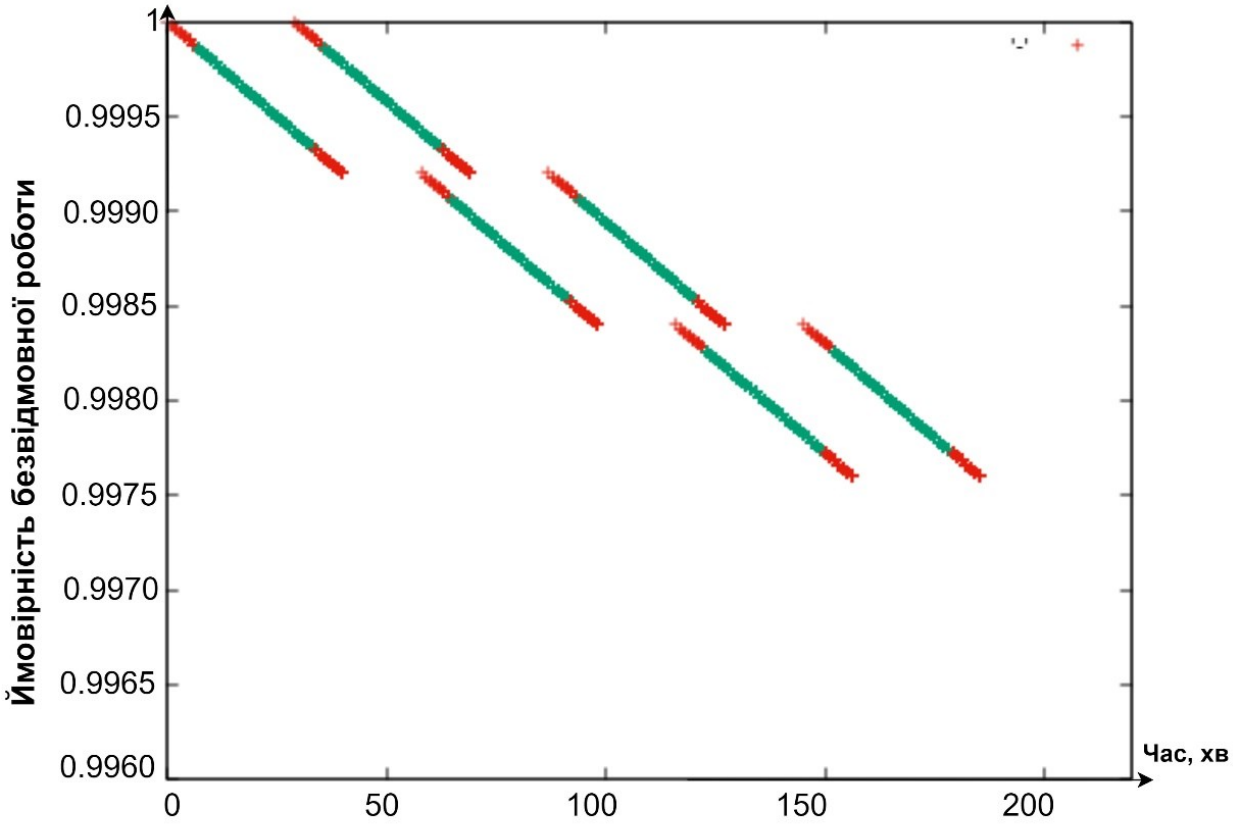

Рисунок 3.12 − Узагальнюючий графік залежності ймовірності безвідмовної роботи від часу для обох змін

3.4 Висновки до третього розділу

1. Було запропоновано три стратегія розгортання БПЛА для утворення літаючої мережі: стратегія першої точки маршруту, яка передбачає рух кожного БПЛА до першої точки LiFi маршруту і подальше розгортання мережі в межах нього; стратегія радіального руху, яка передбачає рух кожного БПЛА одразу до точки призначення на LiFi маршруті; стратегія середньої точки маршруту, яка

передбачає рух кожного БПЛА до точки, максимально наближеної до середини LiFi маршруту, і подальше розгортання мережі в межах нього.

2. Проведено порівняльний аналіз стратегій, в результаті якого встановлено:

– найбільше зменшення часу розгортання літаючої LiFi мережі внаслідок збільшення швидкості БПЛА з 2 м/с до 4 м/с спостерігається для стратегії першої точки маршруту і складає 6,5 с. Для стратегій радіального руху і середньої точки маршруту воно складає відповідно 5,69 с та 5,81 с відповідно. Якщо зробити ранжування стратегій у порядку збільшення виграшу у часі від збільшення швидкості БПЛА, то результатом буде така послідовність: стратегія радіального руху, стратегія середньої точки маршруту, стратегія першої точки маршруту;

– для всіх розглянутих швидкостей БПЛА стратегія радіального руху забезпечує найменший час розгортання літаючої LiFi мережі. Так, наприклад, для швидкості 2 м/с цей час є на 1,39 с меншим за час розгортання літаючої LiFi мережі при застосуванні стратегії першої точки маршруту. Якщо зробити ранжування стратегій у порядку зменшення часу розгортання літаючої LiFi мережі, то результатом буде така послідовність: стратегія першої точки маршруту, стратегія середньої точки маршруту, стратегія радіального руху.

3. Показано, що цикл роботи однієї зміни рою БПЛА для розгортання і забезпечення функціонування LiFi мережі складається з наступних етапів: виліт з депо та підліт до визначених точок на прокладеному LiFi маршруті; налаштування мережі; передача даних; повернення до депо; очікування на наступний цикл роботи.

4. Надана графічна інтерпретація почергової роботи двох змін БПЛА для розгортання і забезпечення безперебійного функціонування літаючої LiFi мережі, відповідно до якої БПЛА наступної зміни прилітають і налаштовують утворюваними ними мережу до початку руху БПЛА поточної зміни до депо.

5. З використанням розробленого програмного засобу було розраховано, що безперебійне функціонування літаючої LiFi мережі із ЙБР не нижче P_{min} = 0,99875 протягом 180 хвилин можна забезпечити двома змінами, кожна з яких виконує по три цикли роботи і складається із 6 БПЛА з інтенсивністю відмов λ = 0,0005 1/год.

РОЗДІЛ 4
РОЗРОБЛЕННЯ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ПІДТРИМКИ ПЛАНУВАННЯ РОЗГОРТАННЯ ЛІТАЮЧОЇ LIFI МЕРЕЖІ

4.1 Програмний засіб для підтримки планування розгортання БПЛА для утворення LiFi мережі

4.1.1 Архітектура та варіанти використання програмного засобу “Simulation Way”

Програмний засіб, що пропонується і має назву “Simulation Way”, може бути використаний для реалізації етапів планування розгортання БПЛА літаючої LiFi мережі, зокрема для:

– прокладання LiFi маршрутів від джерела до споживача інформації у приміщенні з перешкодами з позначенням на маршрутах точок (місць) розміщення БПЛА для утворення літаючої LiFi мережі з використанням для обходу перешкод алгоритмів лівого та правого кутів, а також керованого водоспаду;

– формування графу можливих LiFi маршрутів і визначення найкоротшого з них шляхом застосування алгоритму Дейкстри;

– розміщення БПЛА у визначених точках (місцях) на прокладеному найкоротшому LiFi маршруті з використанням різних стратегій розгортання: першої точки маршруту, радіального руху і середньої точки маршруту.

Архітектура програмного засобу зображена на рис. 4.1. Програмний засіб “Simulation Way” має трьохрівневу архітектуру та складається з наступних рівнів.

Рівень *графічного інтерфейсу користувача (Graphical User Interface (GUI))*. Цей рівень являє собою інтерфейс програмного засобу, який реалізований за допомогою бібліотеки Python з назвою tkinter.

Рівень *Business logic*. Цей рівень надає логіку взаємодії між сховищем даних (Storage data) та графічним інтерфейсом (GUI). Рівень містить модулі генерування звітів (результатів), модулі розрахункового ядра та модулі зовнішніх алгоритмів

(наприклад алгоритм Дейкстри для пошуку найкоротшого маршруту), які можуть взаємодіяти між собою.

Рівень *Storage data*. Цей рівень відповідає за отримання і зберігання у вигляді файлів даних результатів розрахунків та звітів щодо процесу роботи програмного засобу.

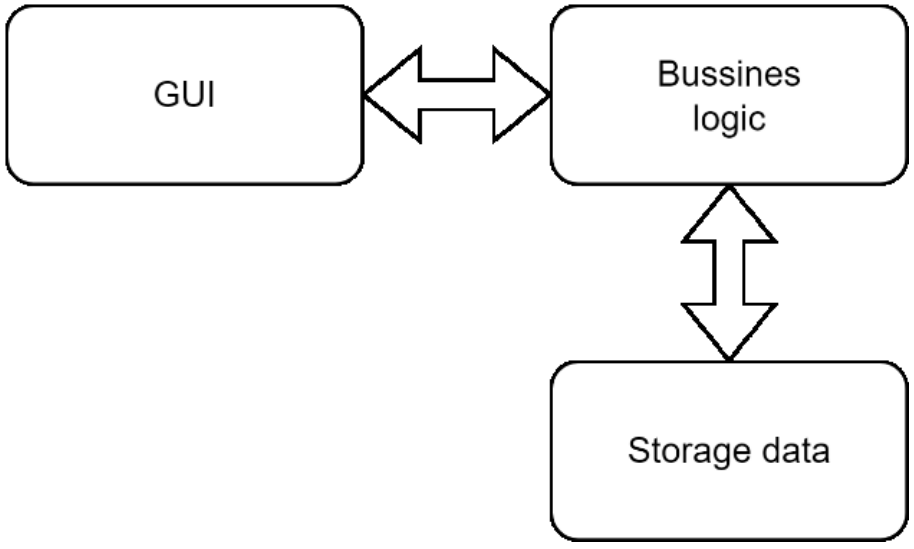

Рисунок 4.1 − Архітектура програмного засобу “Simulation Way”

Програмний засіб може бути використаний операторами КЦ під час моделювання розгортання LiFi мережі на основі БПЛА у виробничому приміщенні з перешкодами. Варіанти використання програмного засобу оператором КЦ продемонстровано на рис. 4.2.

Взаємодія оператора КЦ з програмним засобом здійснюється за допомогою графічного інтерфейсу.

Оператору КЦ доступна велика кількість налаштувань: встановлювати розміри робочої площі виробничого приміщення, генерувати необхідну кількість перешкод з заданими характеристиками та обирати методи (правила) обходу цих перешкод. Крім того, сформовані вхідні дані під час однієї ітерації моделювання можуть використовуватися у наступних ітераціях.

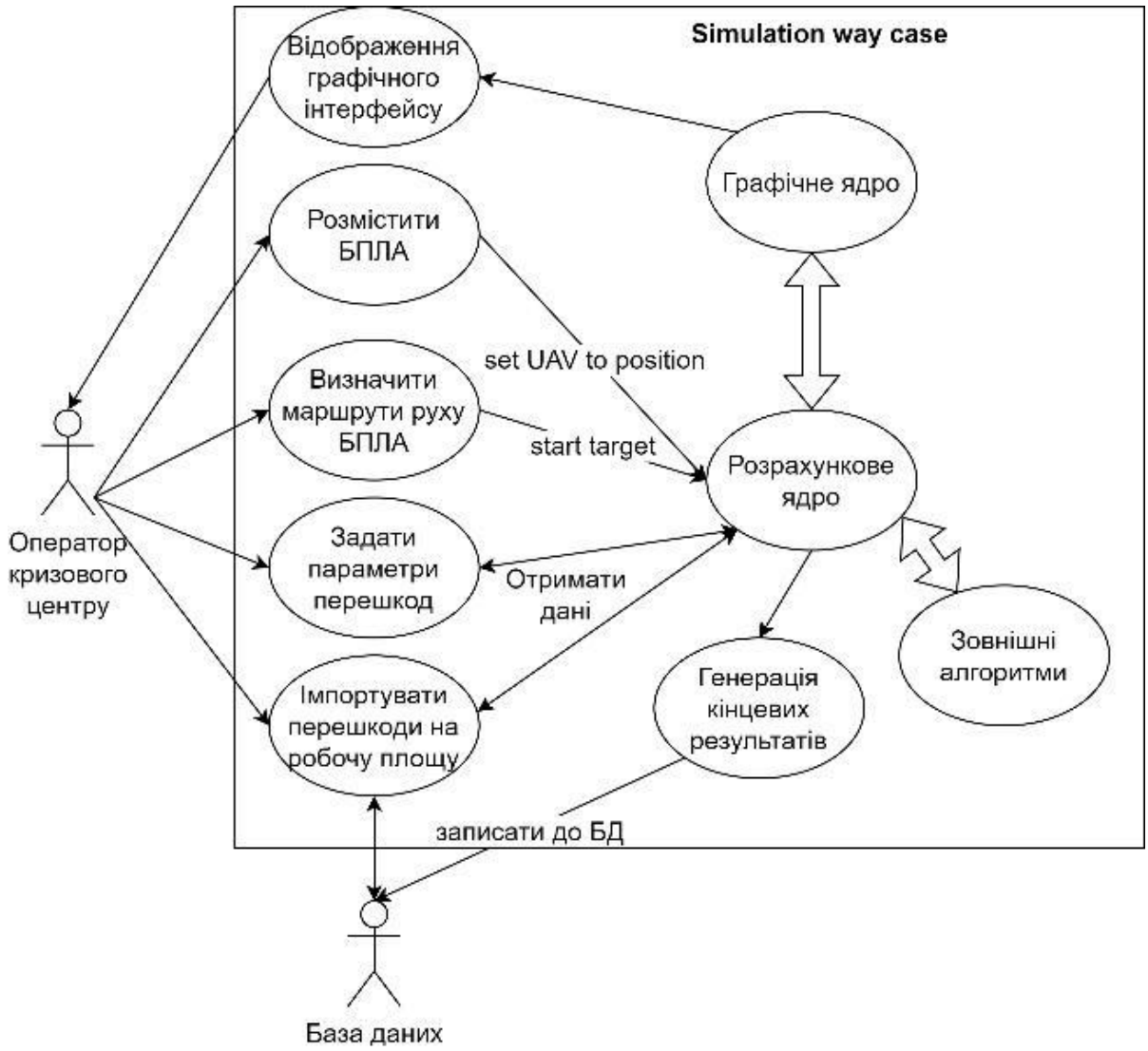

Рисунок 4.2 − Діаграма варіантів використання програмного засобу “Simulation Way”

За проведення розрахунків відповідає *розрахункове ядро*, яке поєднує у собі математичні та алгоритмічні модулі, а також модулі взаємодії. *Графічне ядро* інформує оператора КЦ про хід процесу моделювання, виводить результати розрахунків та дозволяє керувати правилами моделювання. Зокрема, оператор КЦ візуально може бачити:

– номер ітерації;

– назва алгоритму, відповідно до якого здійснюється обхід перешкод;

– координати точок (місць) розміщення БПЛА на прокладеному LiFi маршруті;

– довжину LiFi маршруту (довжину заданої ділянки LiFi маршруту);

– кількість БПЛА, яких потрібно розмістити у визначених точках LiFi маршруту для розгортання LiFi мережі.

Послідовність взаємодії оператора КЦ з програмним засобом “Simulation Way” показана на рис. 4.3.

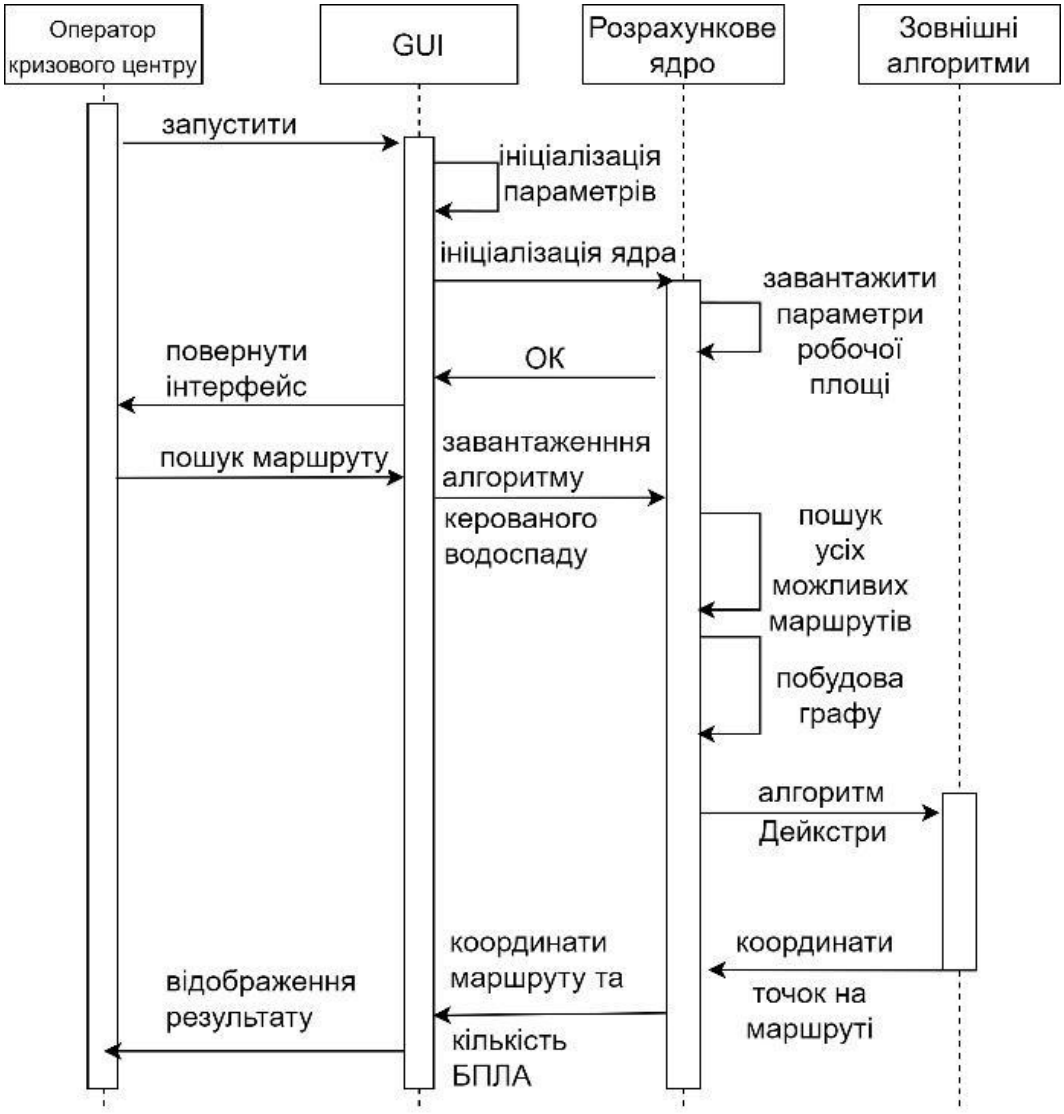

Рисунок 4.3 − Діаграма, що демонструє послідовність взаємодії оператора КЦ з програмним засобом “Simulation Way”

Під час активації програмного засобу “Simulation Way” ініціалізуються необхідні бібліотеки та модулі. Оператор КЦ напряму не взаємодіє з алгоритмами або розрахунковим ядром. Для цього оператор КЦ має у своєму розпорядженні

графічний інтерфейс. Функції оператора КЦ під час користування програмним засобом “Simulation Way” зводяться до введення (корегування) необхідних для моделювання параметрів. На підставі згенерованих під час моделювання множини LiFi маршрутів, програмний засіб “Simulation Way” формує граф можливих LiFi маршрутів і визначає найкоротший з них шляхом застосування алгоритму Дейкстри. Отримання результатів розрахунків можливе як у вигляді візуальної інформації на панелі керування графічного інтерфейсу, так і у вигляді підготовленого файлу звіту.

4.1.2 Структура програмного засобу “Simulation Way”

Програмний засіб “Simulation Way” має модульну структуру, де кожен модуль реалізує API для взаємодії, відокремлюючи окремі функціональні блоки. На рис. 4.4 зображена модульна діаграма, яка демонструє логіку взаємодії компонентів програмного засобу між собою.

Оператор КЦ взаємодіє з програмним засобом за допомогою модуля *draw.py*. Графічний інтерфейс користувача (GUI)) реалізовано за допомогою *tkinter* (бібліотеки Python), яка є одним із найпоширеніших рішень для відображення графічного інтерфейсу.

Блок *algo.py* реалізує міжмодульну взаємодію (обхід перешкод, рух БПЛА тощо) та надає базовий функціонал.

Модуль *math_core* є математичним ядром програмного засобу. У цьому модулі реалізоване API для розрахунку довжин, конвертації та корекції чисел з плаваючою крапкою тощо.

External zone представляє собою зону зовнішніх модулів, які використовуються для взаємодії з ресурсами операційної системи та базовими інтегрованими алгоритмами. Саме завдяки External zone стає можливим:

– генерувати зображення графів при використанні різних алгоритмів обходу перешкод;

– генерувати дані про: параметри перешкод, кількість ітерацій, довжину маршруту (ділянки маршруту), кількість БПЛА, необхідних для розгортання LiFi мережі, координати точки розміщення кожного БПЛА на прокладеному маршруті;

– вести журнал роботи програмного засобу “Simulation Way”.

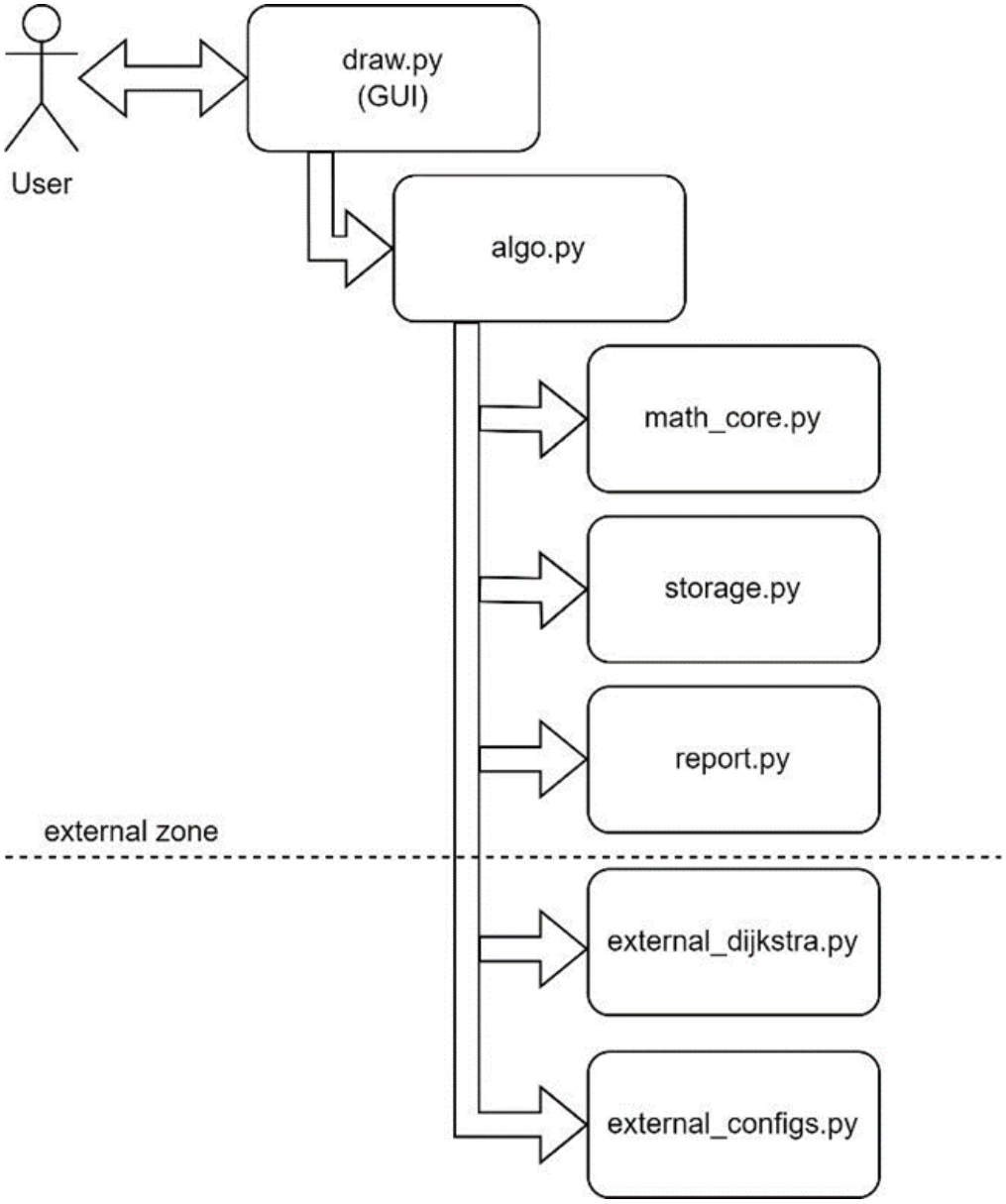

Рисунок 4.4 − Модульна діаграма, яка демонструє логіку взаємодії компонентів програмного засобу “Simulation Way” між собою

Модуль *storage.py* представляє собою сховище для змінних, які необхідні для міжмодульної взаємодії. Цей модуль зберігає характеристики БПЛА, координати

їх місць базування та точок подальшого розміщення на LiFi маршруті, координати розташування перешкод у приміщенні тощо.

Дані, які формуються у процесі використання програмного засобу “Simulation Way”, першочергово зберігаються саме у цьому модулі (координати точок маршруту для конкретної ітерації, назва алгоритму обходу перешкод тощо).

4.1.3 Опис базового функціоналу програмного засобу “Simulation Way”

Графічний інтерфейс програмного засобу “Simulation Way” розташований в окремих графічних вікнах: *Control* (рис. 4.5, 4.6), яке є центром керування параметрами та правилами моделювання (рис. 4.6), та *Way simulation* (рис. 4.7), яке відображає хід процесу моделювання у реальному часі.

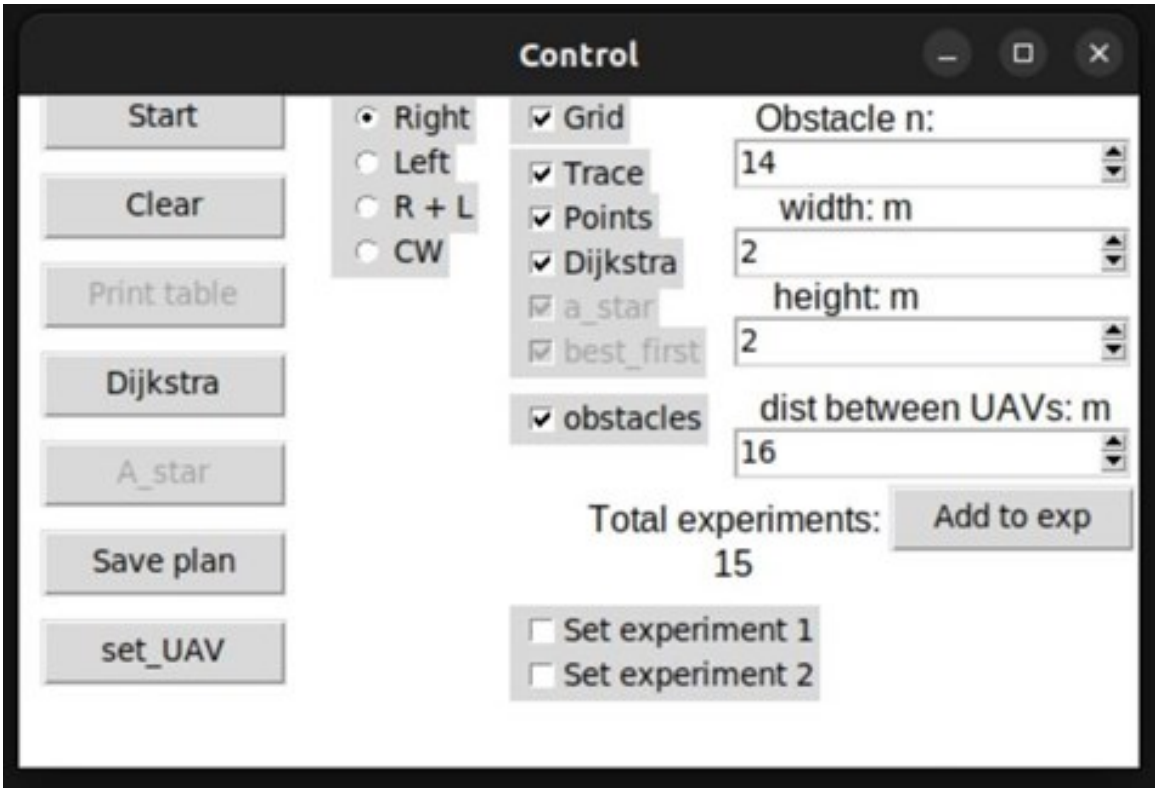

Рисунок 4.5 – Вид панелі Control

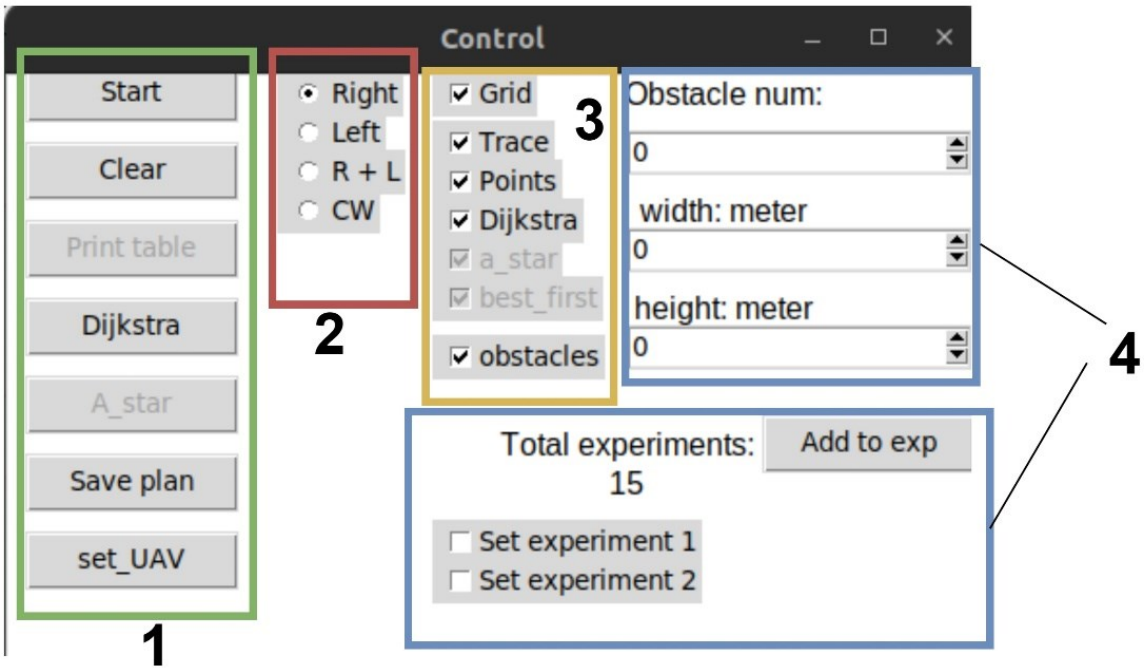

Рисунок 4.6 – Вид панелі Control з функціональними зонами: 1 – зона запуску процесів; 2 – зона реалізації методу (правила) обходу перешкод; 3 – зона графічного відображення шарів; 4 – зона налаштування параметрів моделювання

Вікно інтерфейсу, представлене на рис. 4.7, умовно можна поділити на чотири зони.

1) *Зона запуску процесів*. Кнопка Start відповідає за запуск процесу пошуку маршруту між початковою та кінцевою точками за заданими параметрами. Кнопка Clear дозволяє очистити дані з попередніх ітерацій. Кнопка Dijkstra дозволяє використати алгоритм Дейкстри для пошуку найкоротшого маршруту за поточним графом. Кнопка Start у поєднанні з пунктом CW із зони 2 також використовує цей алгоритм, проте граф формується автоматично та не може бути змінений протягом усіх ітерацій моделювання. Кнопка Save plan відповідає за збереження поточного плану приміщення разом з перешкодами. Дані будуть збережені у xml-файл та можуть бути повторно використані у майбутньому. Кнопка set_UAV відповідає за розміщення БПЛА в заданих точках прокладеного маршруту.

2) *Зона реалізації алгоритму обходу перешкод*. Ця зона відповідає за реалізацію алгоритму обходу перешкод у виробничому приміщенні. У разі вибору Right обхід перешкод буде відповідно до алгоритму правого кута, а Left – лівого

кута. Вибір R+L дозволить одночасно використовувати алгоритми лівого та правого кутів. Вибір CW дозволить застосувати алгоритм керованого водоспаду.

3) *Зона графічного відображення шарів*. Ця зона дозволяє включити та виключити кожен шар без втрати даних та обмежень.

4) *Зона налаштування параметрів моделювання*. Ця зона дозволяє встановити кількість перешкод, їх форму та автоматизувати процес створення статистичних даних у звіті, який формується для оператора КЦ.

Панель *Way simulation* (рис. 4.7) відображає робочу площу виробничого приміщення у 2D просторі.

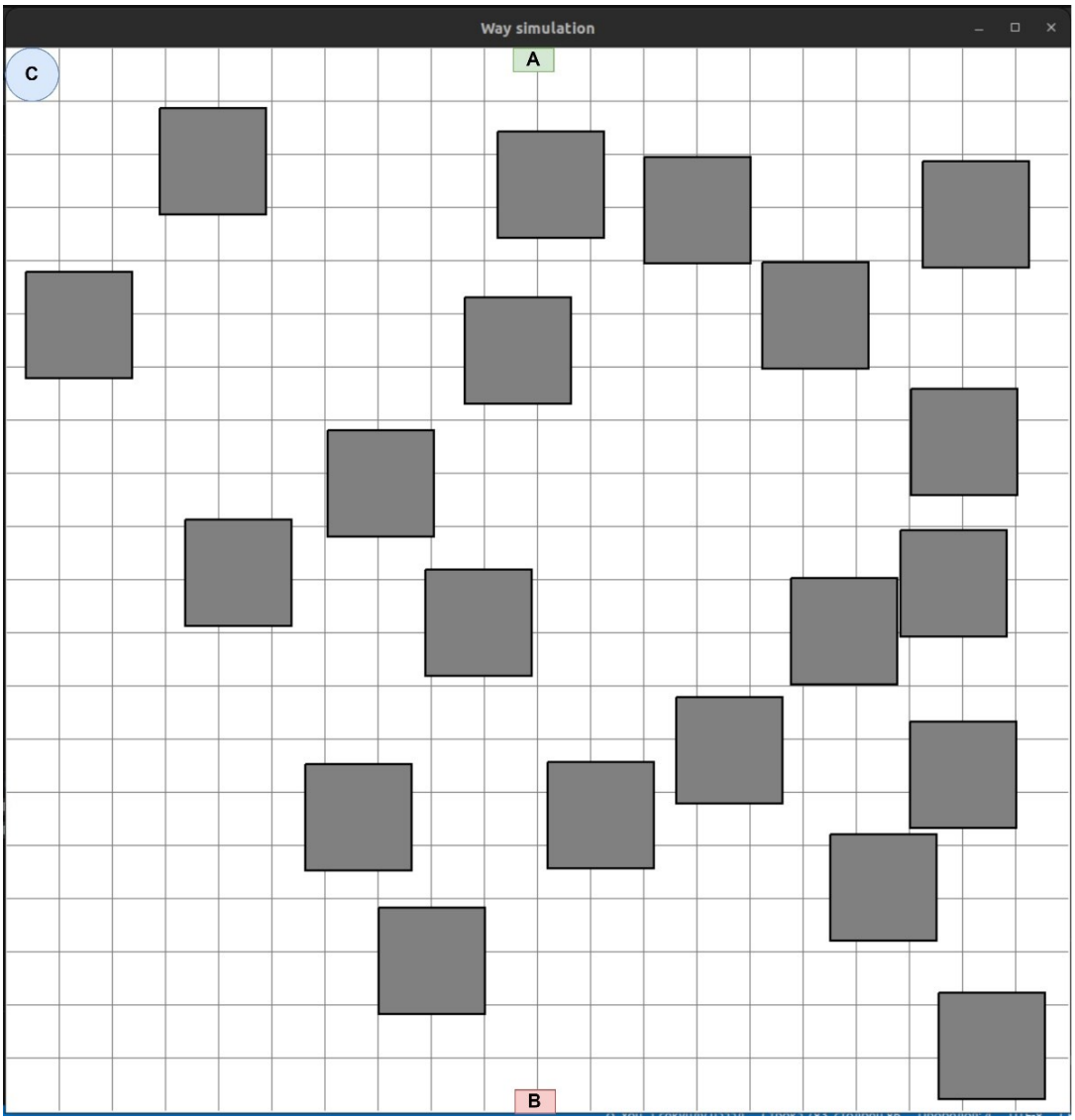

Рисунок 4.7 – Приклад відображення робочої площі виробничого приміщення з перешкодами у 2D просторі

Маленькі зелений (позначений літерою *A*) і червоний (позначений літерою *A*) прямокутники на рис. 4.7 є відповідно початковою (джерело інформації) та кінцевою (споживач інформації) точками маршруту. Сине коло (позначено літерою C) показує стаціонарне депо, де перебувають БПЛА, які потім буде необхідно розмістити на прокладеному LiFi маршруті. На робочій площі знаходяться згенеровані перешкоди, які зображені у вигляді прямокутника (за наявності більш складних форм перешкод їх проекція може бути вписана у випуклий багатокутник з довільною кількістю кутів).

Якщо обрати для обходу перешкод метод керованого водоспаду, то програмний засіб згенерує граф можливих LiFi маршрутів, представлений на рис. 4.8. Далі цей граф може бути використаний для реалізації зовнішнього алгоритму – алгоритму Дейкстри (*Dijkstra*).

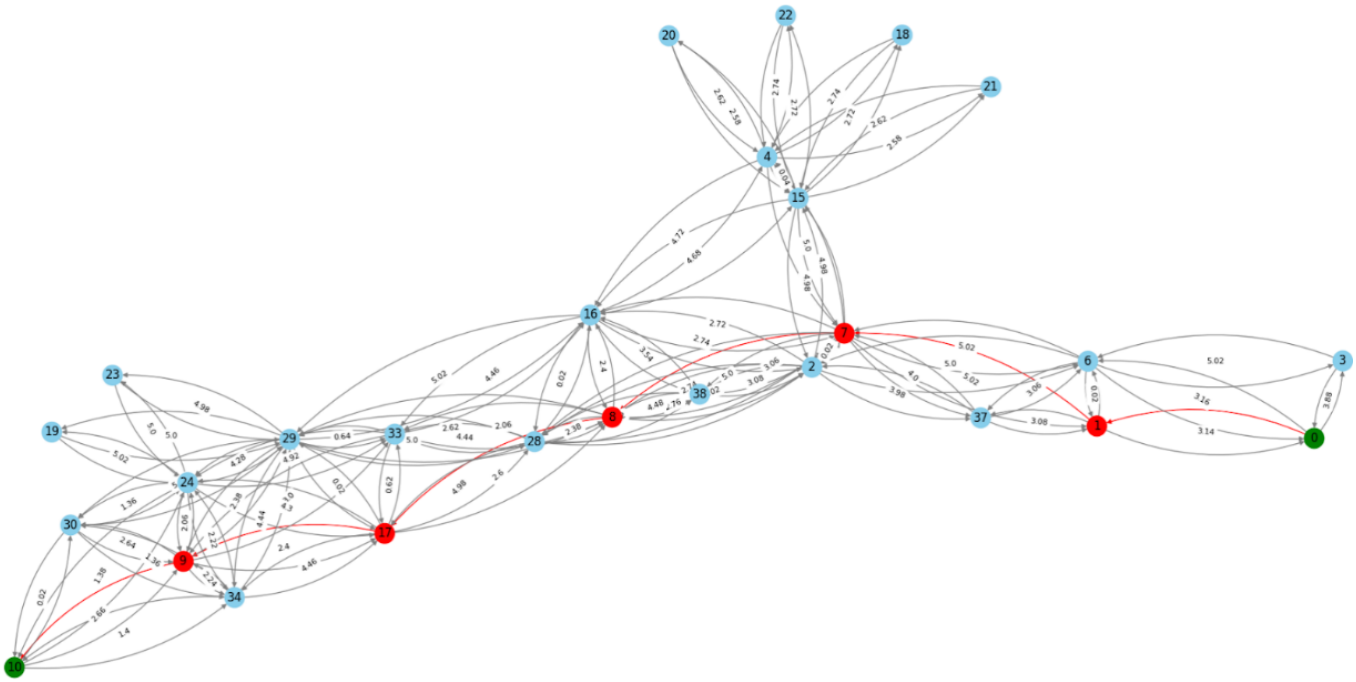

Рисунок 4.8 – Згенерований граф можливих LiFi маршрутів для методу керованого водоспаду

Зеленим кольором на графі показані вершини, що відповідають початковій та кінцевій точкам найкоротшого прокладеного LiFi маршруту, а червоним – його проміжні точки. Решта вершин показано синім кольором. Кожному ребру у

відповідність поставлено його вагу, яка означає відстань між вершинами (точками маршруту) у метрах.

4.1.4 Приклади застосування програмного засобу Simulation Way

Послідовно розглянемо приклади використання програмного засобу “Simulation Way” для прокладання маршруту розповсюдження LiFi сигналу з використанням для обходу перешкод алгоритмів лівого та правого кута, а також алгоритму керованого водоспаду.

Для використання алгоритму правого кута для обходу перешкод необхідно на панелі Control виставити перемикач на позицію Right та натиснути Start (рис. 4.9).

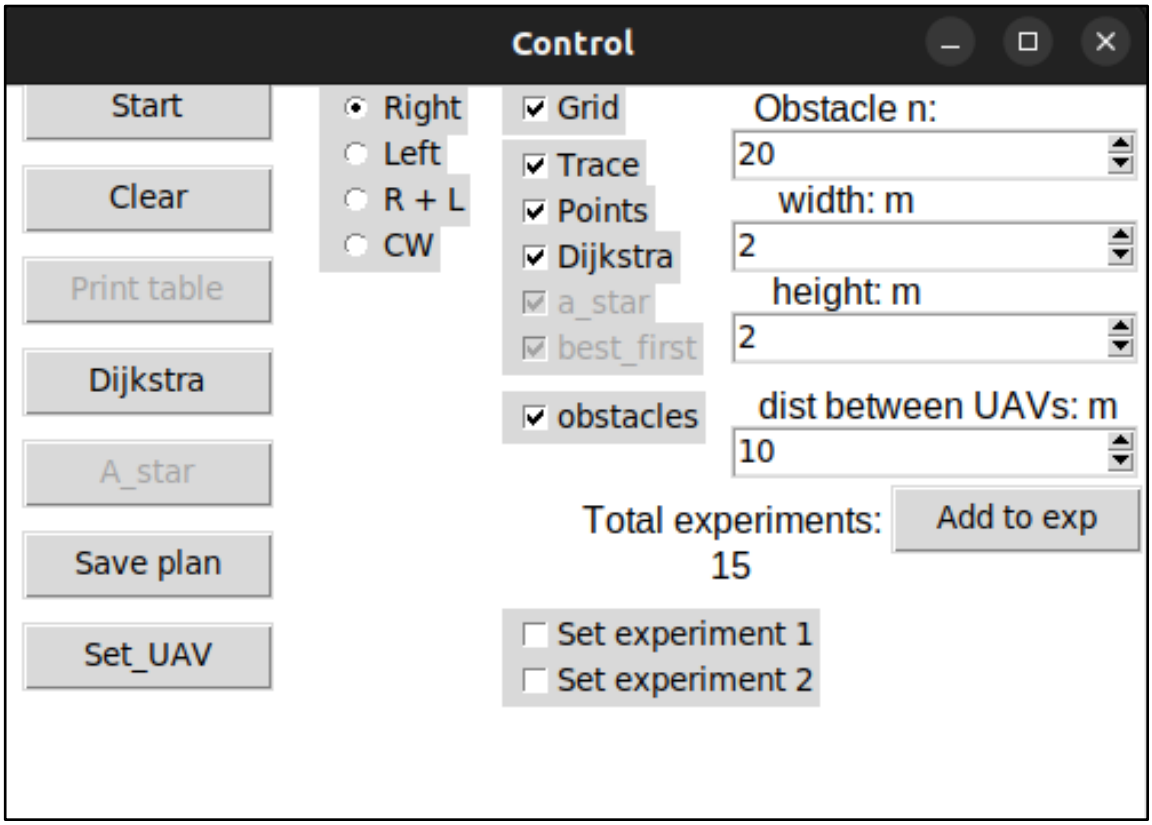

Рисунок 4.9 – Панель Control з параметрами для прокладання LiFi маршруту з використанням для обходу перешкод алгоритму правого кута

Результат моделювання представлено на рис. 4.10. Зелена ламана лінія, показна на цьому рисунку, і є прокладений маршрут. Крім того, програмний засіб “Simulation Way” визначає та зберігає координати точок, які у майбутньому будуть використані як точки розміщення БПЛА у складі утвореної ними LiFi мережі. У якості таких точок розглядають усі вершини ламаної (маршруту) на рис. 4.10, окрім початкової *A* (джерело інформації) і кінцевої *B* (споживач інформації).

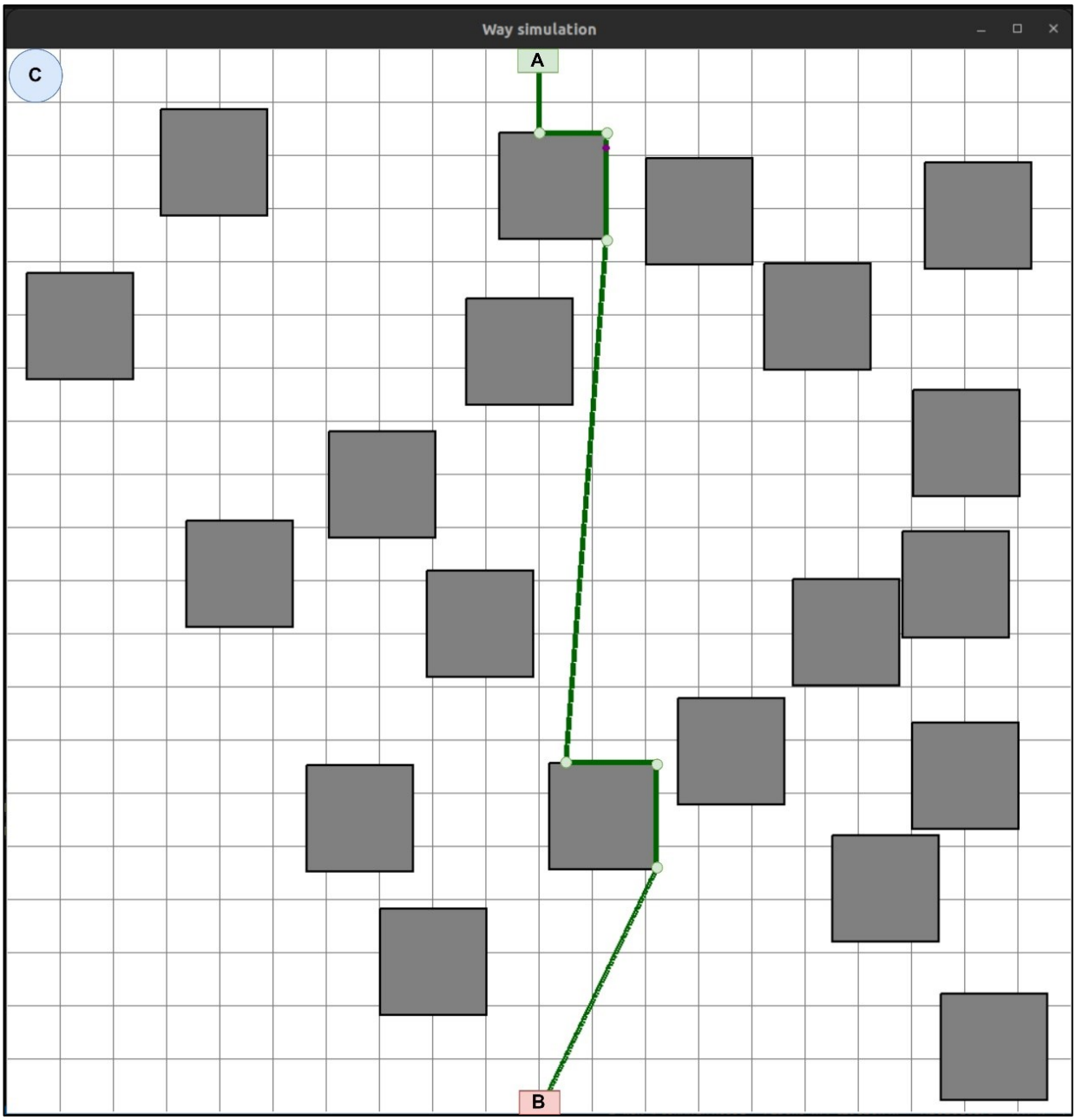

Рисунок 4.10 – Панель із зображенням прокладеного LiFi маршруту з використанням для обходу перешкод алгоритму правого кута

Для активації процесу моделювання з використанням для обходу перешкод алгоритму лівого кута необхідно на панелі Control виставити перемикач на позицію Left та натиснути Start (рис. 4.11).

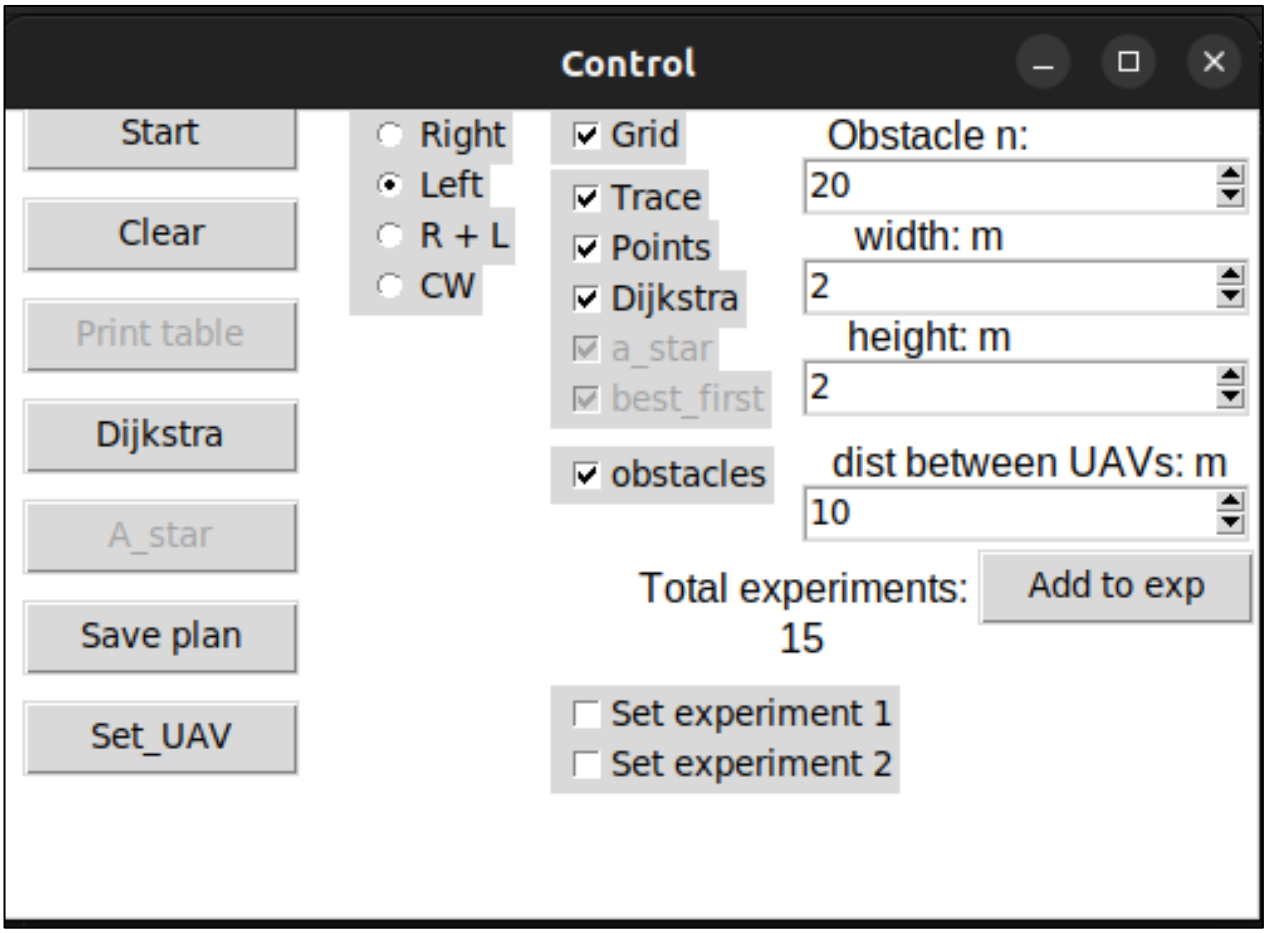

Рисунок 4.11 – Панель Control з параметрами для прокладання LiFi маршруту з використанням для обходу перешкод алгоритму лівого кута

Як і у попередньому випадку результатом моделювання буде згенерований LiFi маршрут у вигляді зеленої ламаної лінії (рис. 4.12).

Для активації процесу моделювання з використанням для обходу перешкод алгоритму керованого водоспаду, необхідно на панелі Control виставити перемикач на позицію CW та натиснути Start (рис. 4.13).

Покладений LiFi маршрут буде відображатися червоною ламаною лінією з зеленими точками, які позначають місця розміщення БПЛА на прокладеному LiFi маршруті (рис. 4.14).

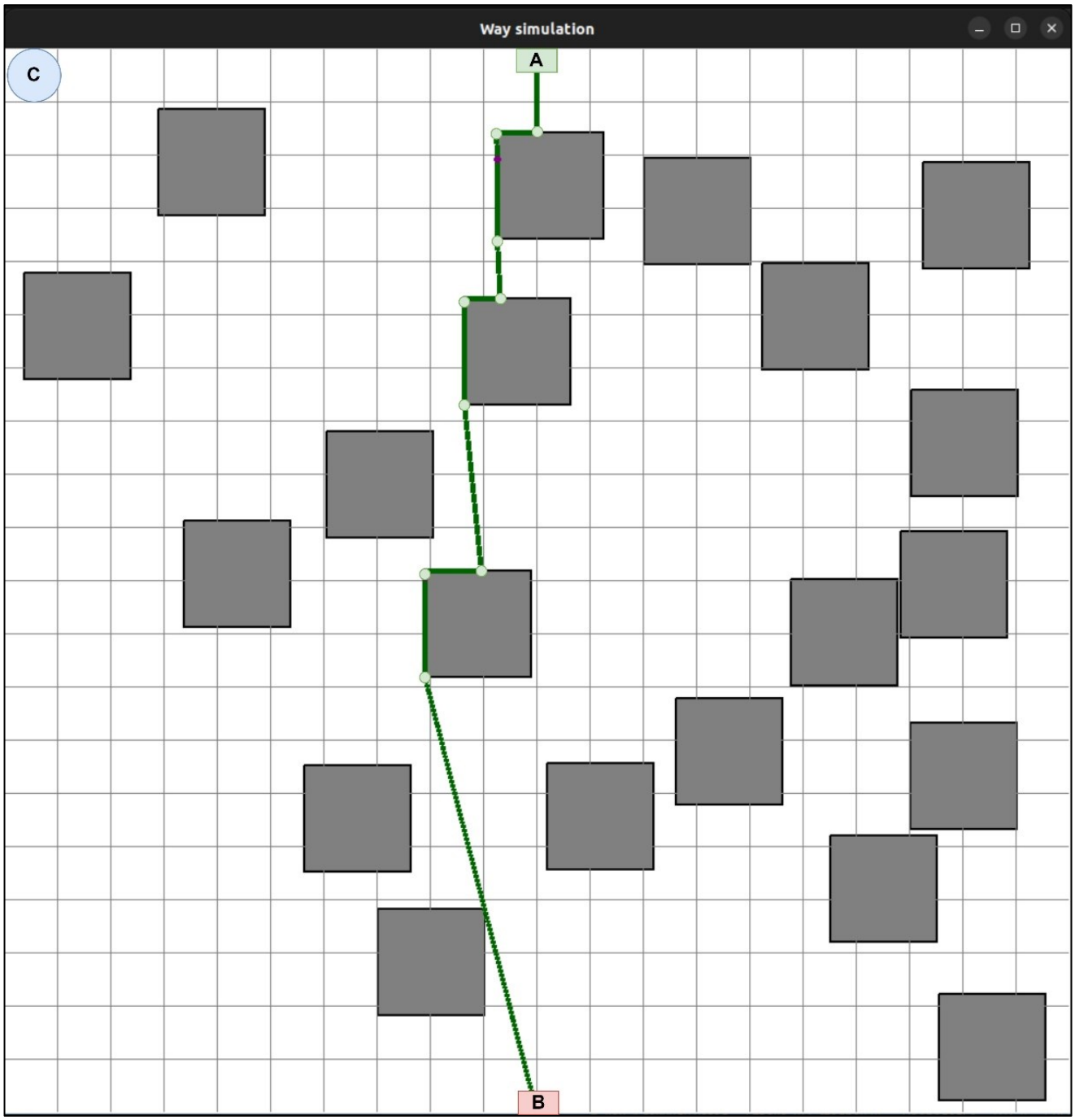

Рисунок 4.12 – Панель із зображенням прокладеного LiFi маршруту з використанням для обходу перешкод алгоритму лівого кута

Як ми можемо бачити, тут точками майбутнього розміщення БПЛА для утворення мережі будуть не тільки вершини ламаної лінії, а й певна кількість додаткових точок.

Необхідність їх введення обумовлена тим, що в умовах підвищеної яскравості (запиленості, задимленості) виробничого приміщення відстань між

вершинами ламаної лінії може перевищувати встановлену для заданих умов дальність LiFi сигналу.

Таким чином, між БПЛА, які розміщуються на точках сусідніх вершин ламаної ліній, необхідно буде розміщувати додатковий (додаткові) БПЛА.

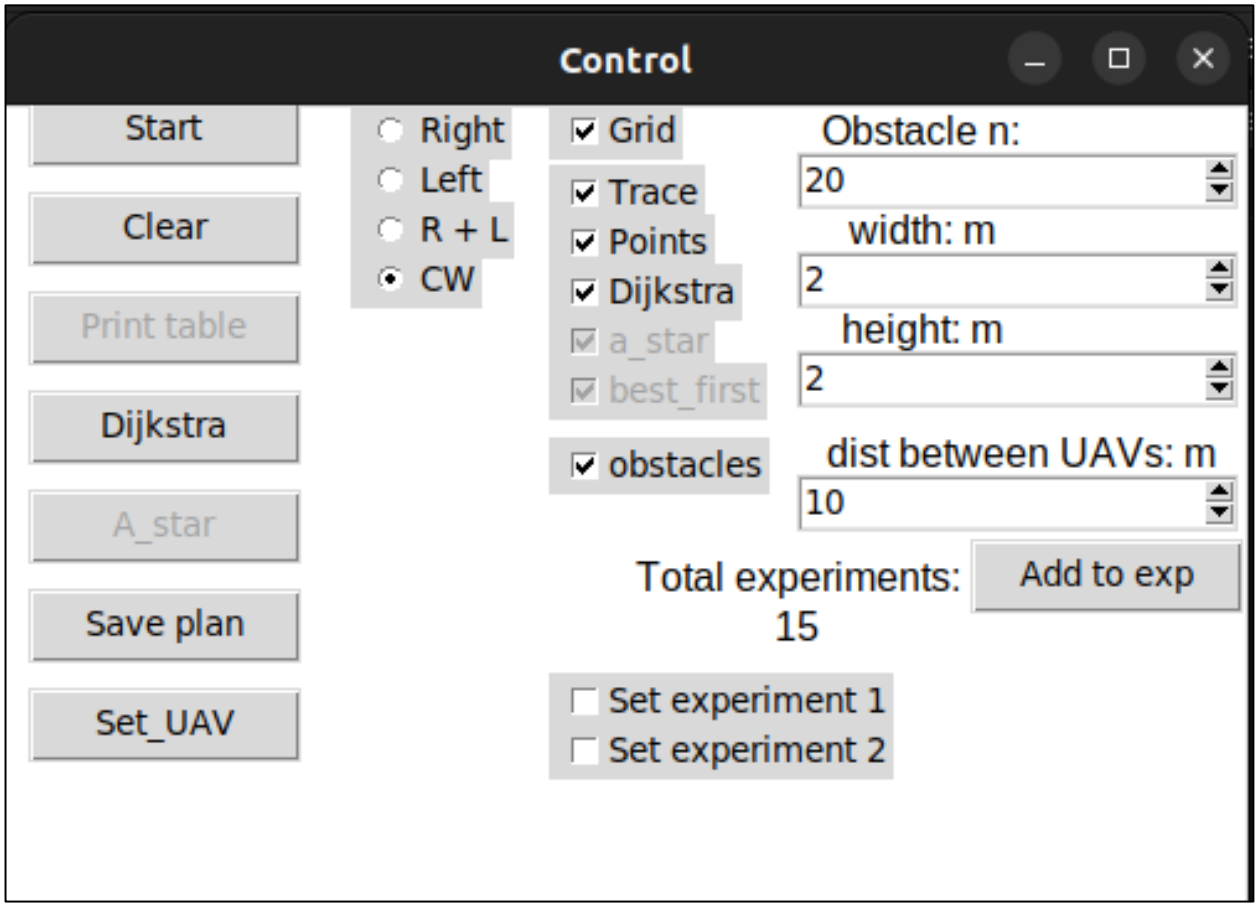

Рисунок 4.13 – Панель Control з параметрами для прокладання LiFi маршруту з використанням для обходу перешкод алгоритму керованого водоспаду

Для генерації маршрутів руху БПЛА з місць базування (депо) до точок розміщення на прокладеному необхідно на панелі Control натиснути кнопку set_UAV. Для прокладання маршрутів руху БПЛА від депо до місць свого розміщення на прокладеному LiFi маршруті можуть використовуватися ті ж самі алгоритми, що і для прокладання LiFi маршруту.

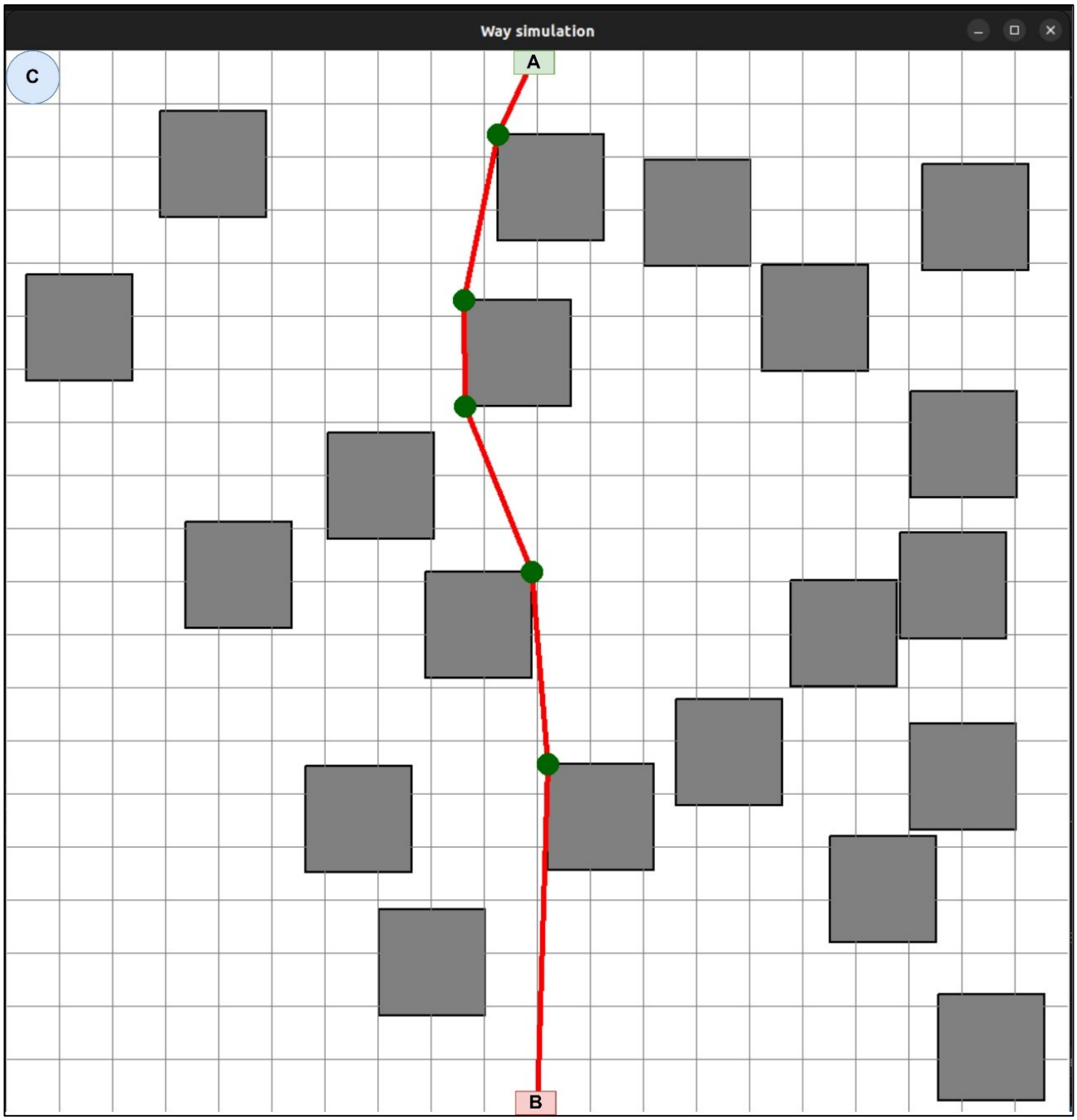

Рисунок 4.14 – Панель із зображенням прокладеного LiFi маршруту з використанням для обходу перешкод алгоритму керованого водоспаду

У прикладі, представленому на рис. 4.15, для обходу перешкод використовується одночасно алгоритми лівого та правого кутів (перемикач поставлено у положення R+L).

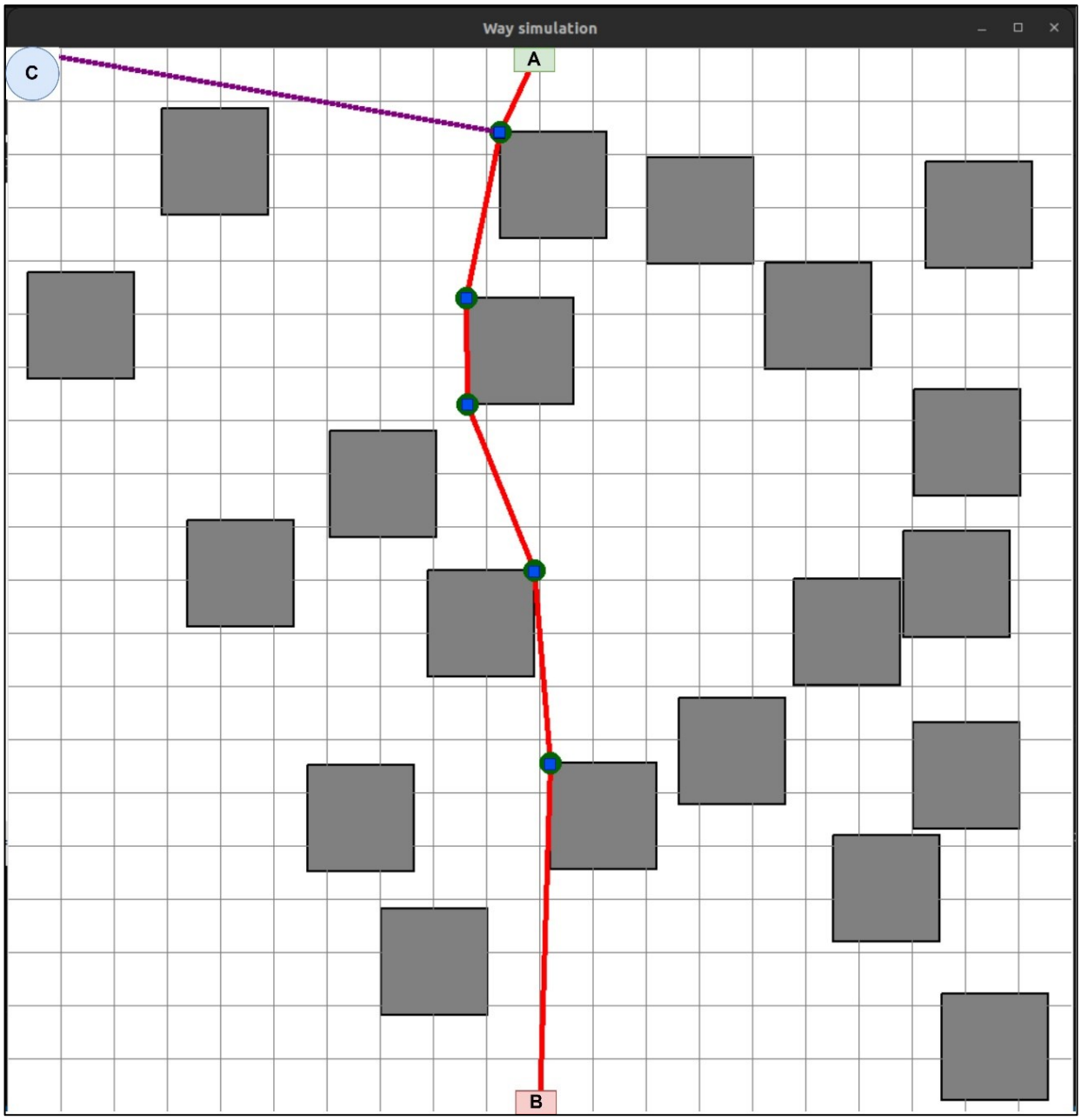

Рисунок 4.15 – Панель із зображенням маршрутів руху БПЛА з депо до точок (місць) розміщення на прокладеному LiFi маршруті з використанням стратегії першої точки маршруту

На рис. 4.15, 4.16 та 4.17 фіолетовими лініями позначено маршрути руху кожного БПЛА до місця свого розміщення на прокладеному LiFi маршруті за стратегіями першої точки маршруту, радіального руху і середньої точки маршруту відповідно.

4.2 Програмний засіб для розрахунку необхідного рівня надійності літаючої LiFi мережі

4.2.1 Архітектура та варіанти використання програмного засобу “Reliability Level”

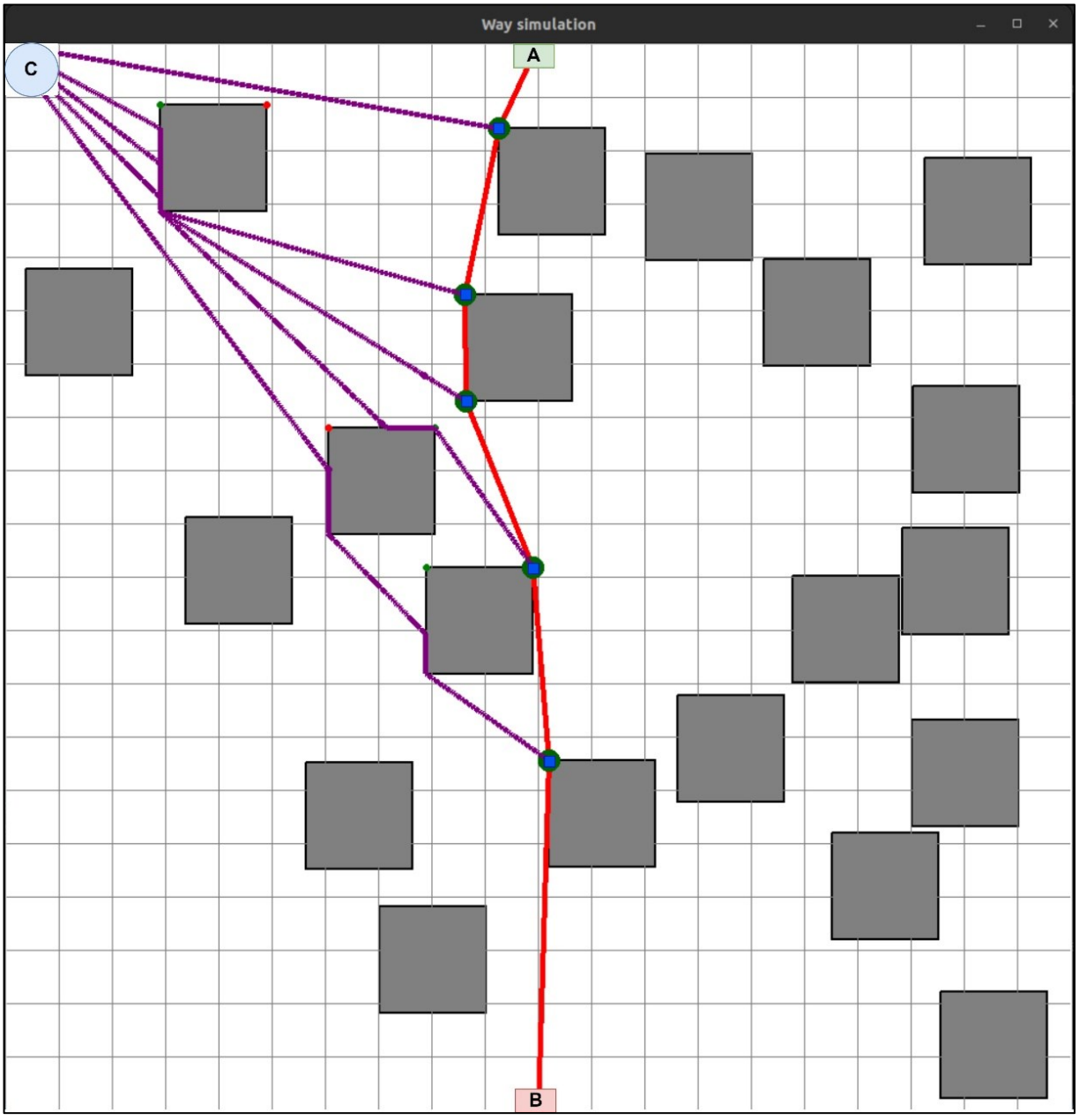

Рисунок 4.16 – Панель із зображенням маршрутів руху БПЛА з депо до точок (місць) розміщення на прокладеному LiFi маршруті з використанням стратегії радіального руху

Програмний засіб, що пропонується і має назву “Reliability Level”, може бути використаний для реалізації етапу 3. *Забезпечення надійності розгорнутої LiFi мережі* схеми планування розгортання на основі БПЛА у виробничому приміщенні з перешкодами, представленої на рис. 1.1 під час таких кроків виконання етапу.

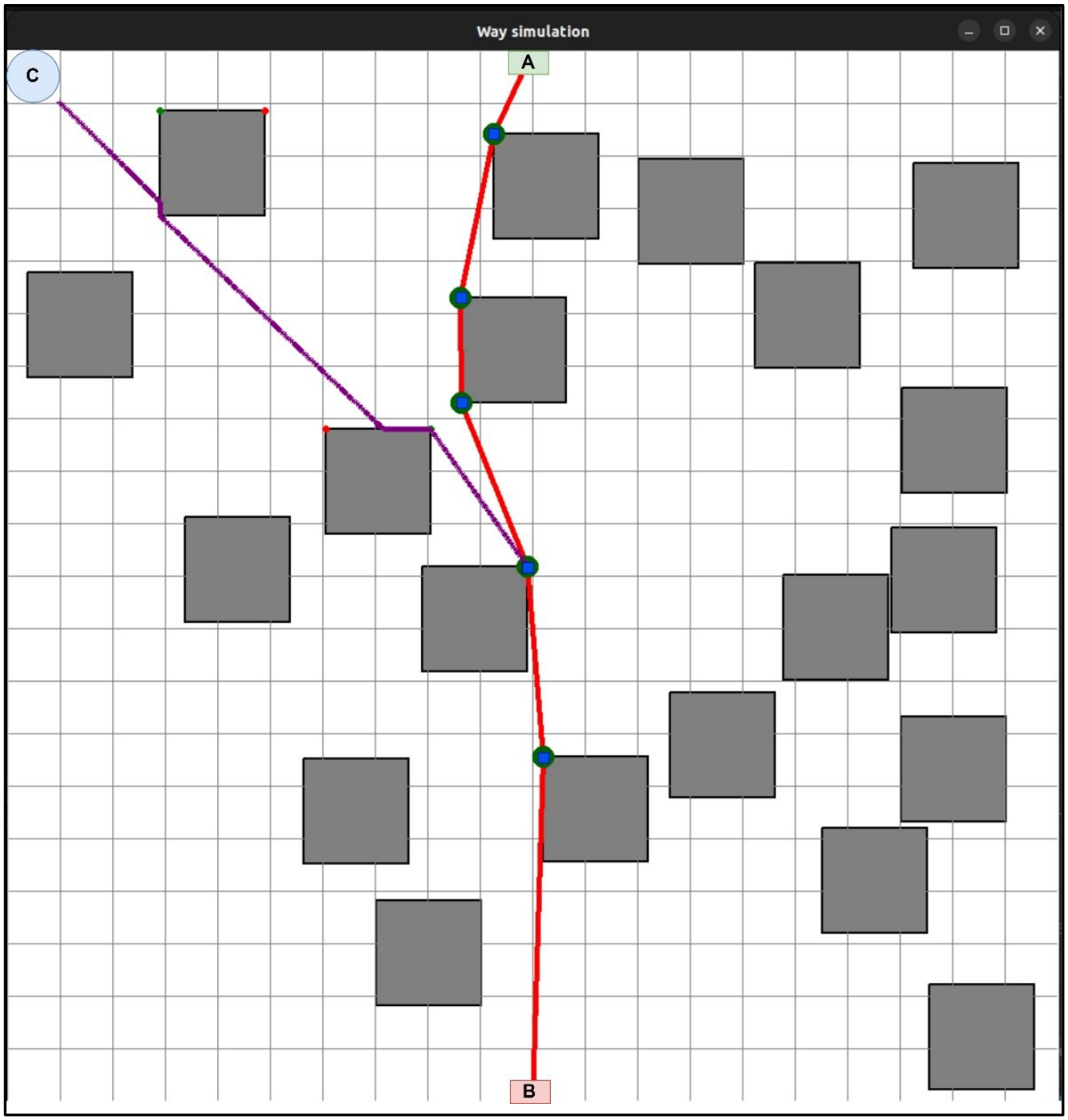

Рисунок 4.17 – Панель із зображенням маршрутів руху БПЛА з депо до точок (місць) розміщення на прокладеному LiFi маршруті з використанням стратегії середньої точки маршруту

Крок 1. Визначення кількості змін БПЛА, часу функціонування кожної з них і способів резервування БПЛА для забезпечення безперебійного функціонування LiFi мережі відповідно до вимог щодо її надійності.

Крок 2. Корегування кількості змін БПЛА, часу функціонування кожної з них і способів резервування БПЛА для забезпечення безперебійного функціонування у разі зміни вимог щодо її надійності.

Програмний засіб має дворівневу структуру та інтерфейс взаємодії у вигляді консольного терміналу (рис. 4.18).

1) Рівень *GUI*. Цей рівень являє собою інтерфейс програмного засобу, який реалізований за допомогою інтерфейсу командного рядка (Command Line Interface (CLI) операційної системи.

2) Рівень *Business logic*. Цей рівень надає логіку взаємодії між логікою програмного засобу “Reliability Level” та CLI. Рівень містить логіку розрахунку ймовірності безвідмовної роботи розгорнутої LiFi мережі (зміни БПЛА, що забезпечує функціонування LiFi мережі протягом заданого часу)

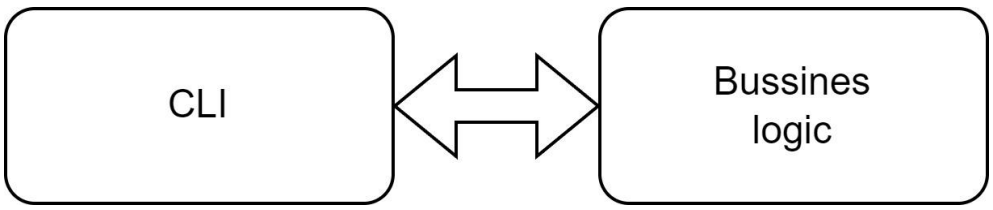

Рисунок 4.18 – Архітектура програмного засобу “Reliability Level”

4.2.2 Структура програмного засобу “Reliability Level”

Програмний засіб “Reliability Level” має однорівневу структуру та складається з монолітного модуля, який поєднує в собі всю логіку. Реалізований у програмному засобі “Reliability Level” алгоритм у вигляді блок-схеми представлено на рис. 4.19.

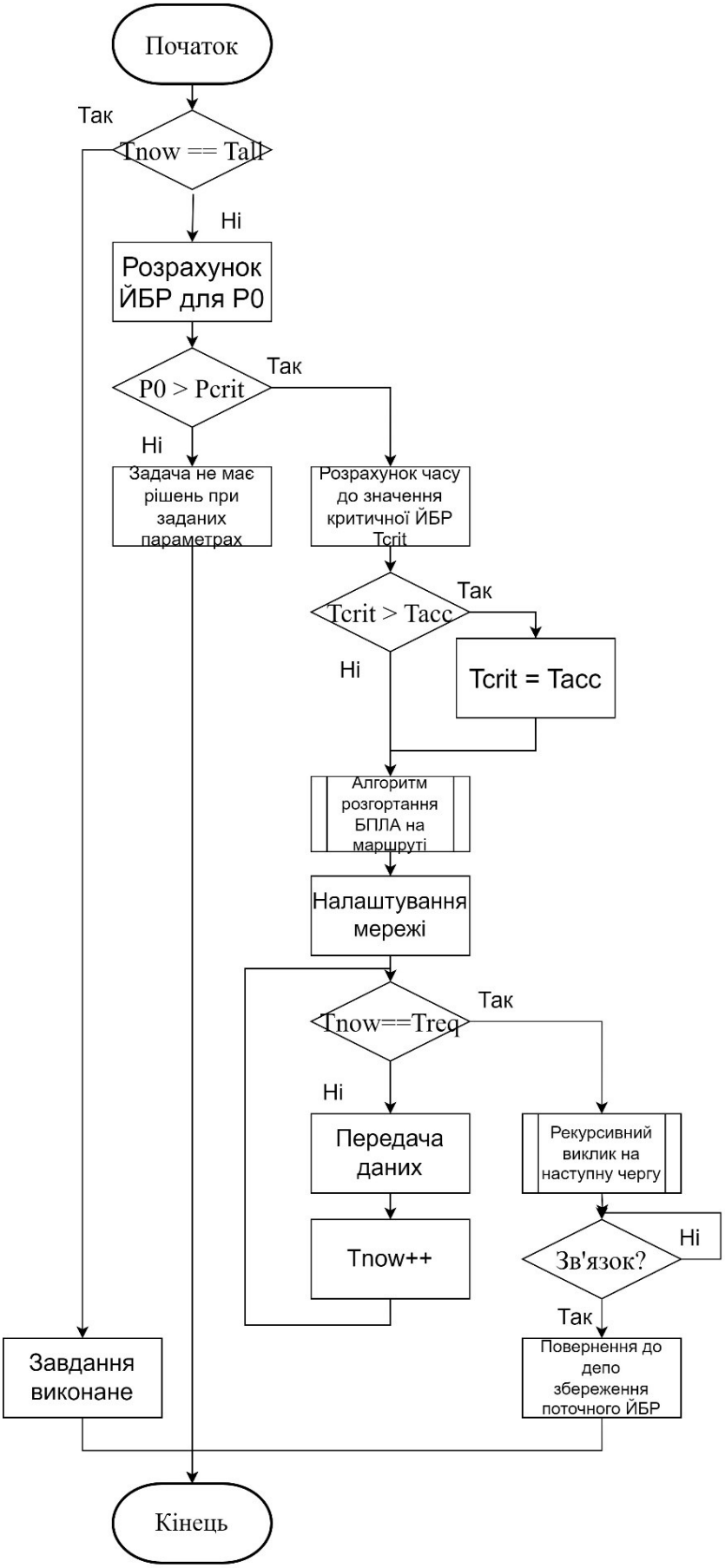

Рисунок 4.19 – Алгоритм, реалізований у програмному засобі “Reliability Level”

4.2.3 Опис базового функціоналу “Reliability Level”

Програмний засіб “Reliability Level” може бути використаний операторами КЦ для визначення часу функціонування літаючої LiFi мережі з рівнем надійності не менше мінімально допустимого протягом заданого часу. Варіанти використання програмного засобу “Reliability Level” оператором КЦ продемонстровано на рис. 4.20.

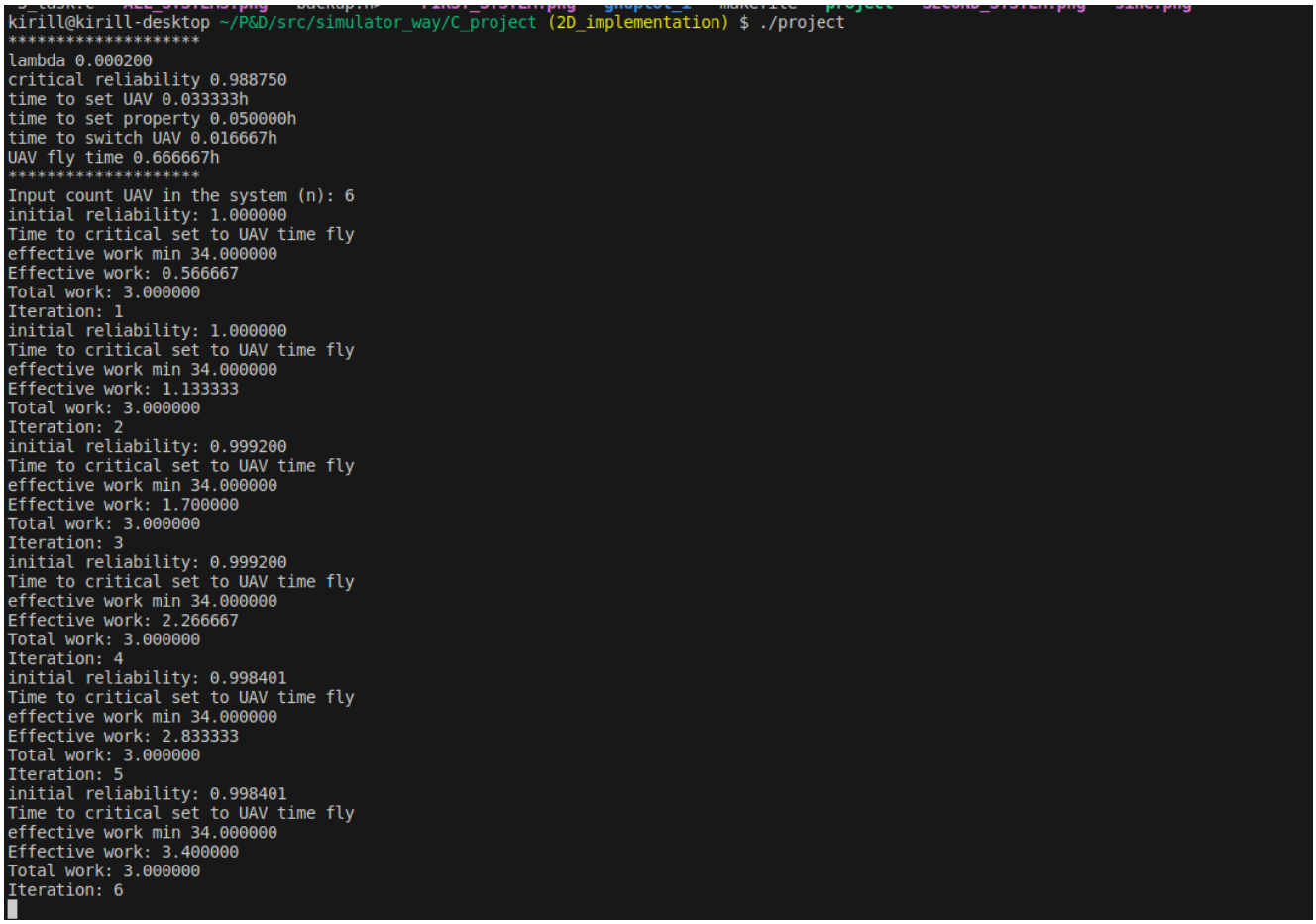

Рисунок 4.20 – Інтерфейс програмного засобу “Reliability Level”

Взаємодія оператора КЦ з програмним засобом “Reliability Level” здійснюється за допомогою консольного інтерфейсу.

Оператору КЦ доступна можливість введення наступних параметрів:

– інтенсивність відмов БПЛА;

– час налаштування літаючої LiFi мережі;

– час функціонування літаючої LiFi мережі в режимі передачі даних;

– часовий ресурс бортової батареї БПЛА;

– мінімально припустимий рівень надійності, з яким може функціонувати літаюча LiFi мережа.

4.2.4 Приклади застосування програмного засобу “Reliability Level”

Усі дані задаються через конфігураційний файл. Результатом роботи є графіки, представлені на рисунках 4.21 (графік залежності ЙБР від часу для першої зміни) 4.22 (графік залежності ЙБР від часу для другої зміни) та 4.23 (графік залежності ЙБР від часу для двох змін одночасно).

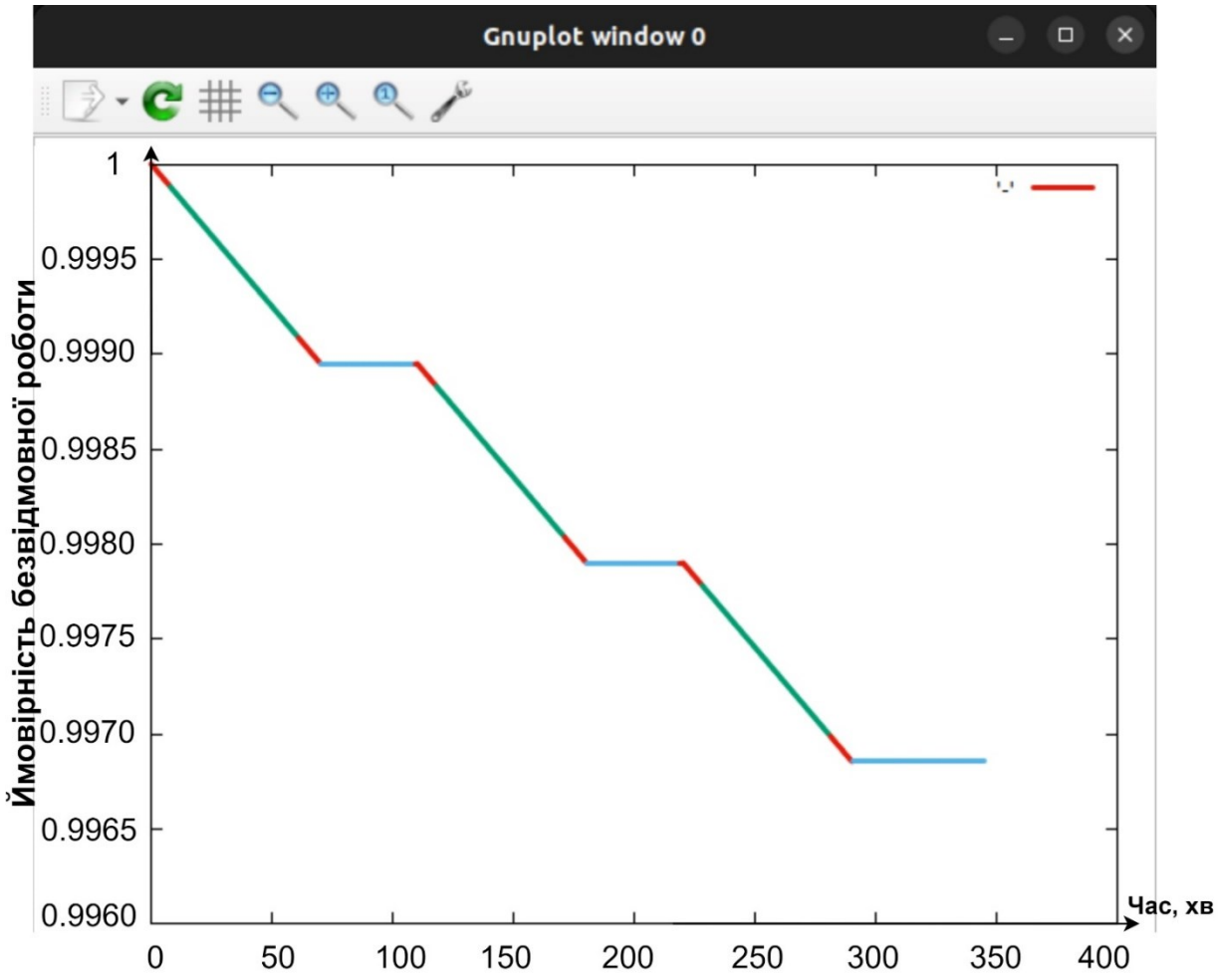

Рисунок 4.21 − Графік залежності ймовірності безвідмовної роботи першої зміни від часу

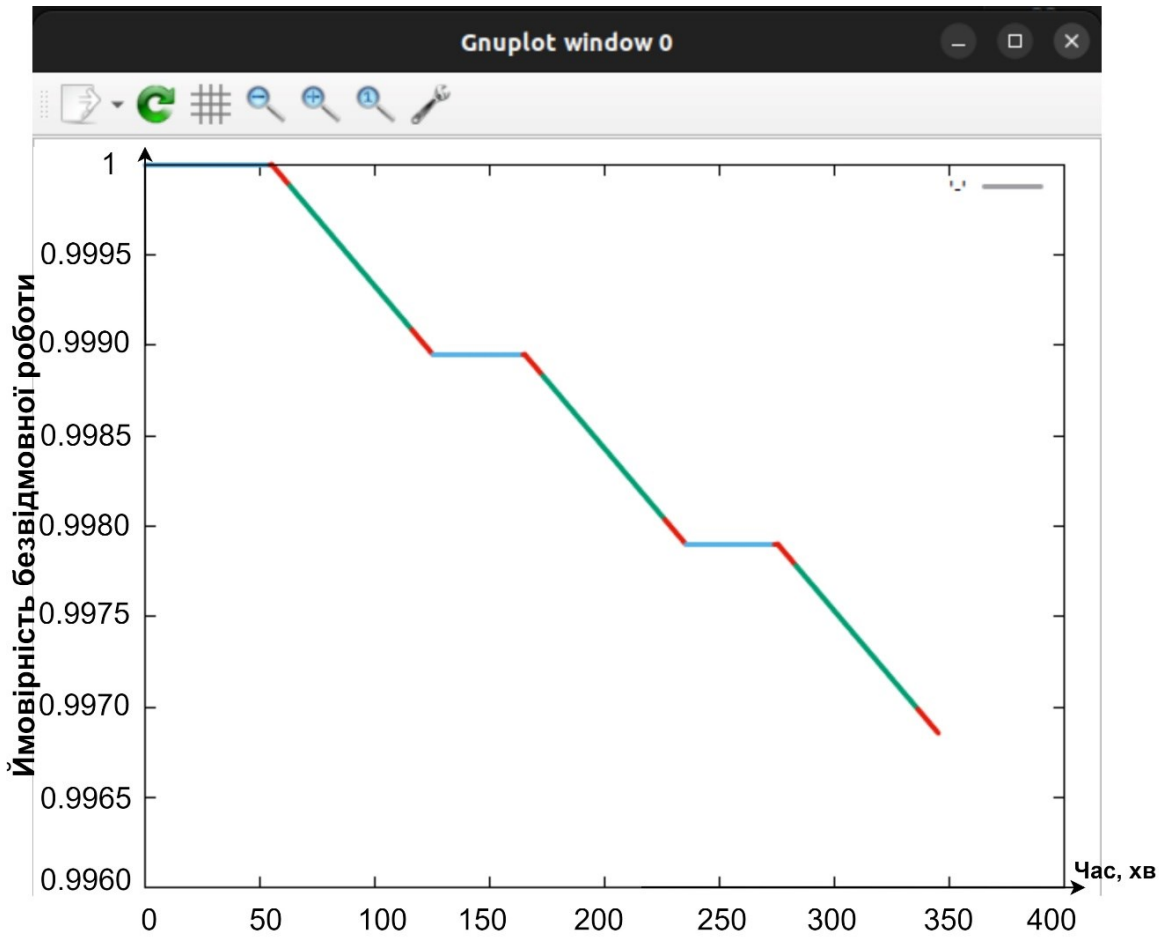

Рисунок 4.22 − Графік залежності ймовірності безвідмовної роботи другої зміни від часу

Кольорові відрізки графіків мають наступний зміст:

– відрізки червоного кольору демонструють зниження ЙБР як за час перельоту БПЛА до своїх точок на LiFi маршруті і налаштування літаючої LiFi мережі (верхні червоні відрізки), так і за час повернення БПЛА до депо (ніжні червоні відрізки);

– відрізки зеленого кольору ілюструють зниження ЙБР за час передачі даних від джерела до споживача даних;

– відрізки голубого кольору показують ЙБР під час під час очікування зміною БПЛА на свій наступний цикл роботи.

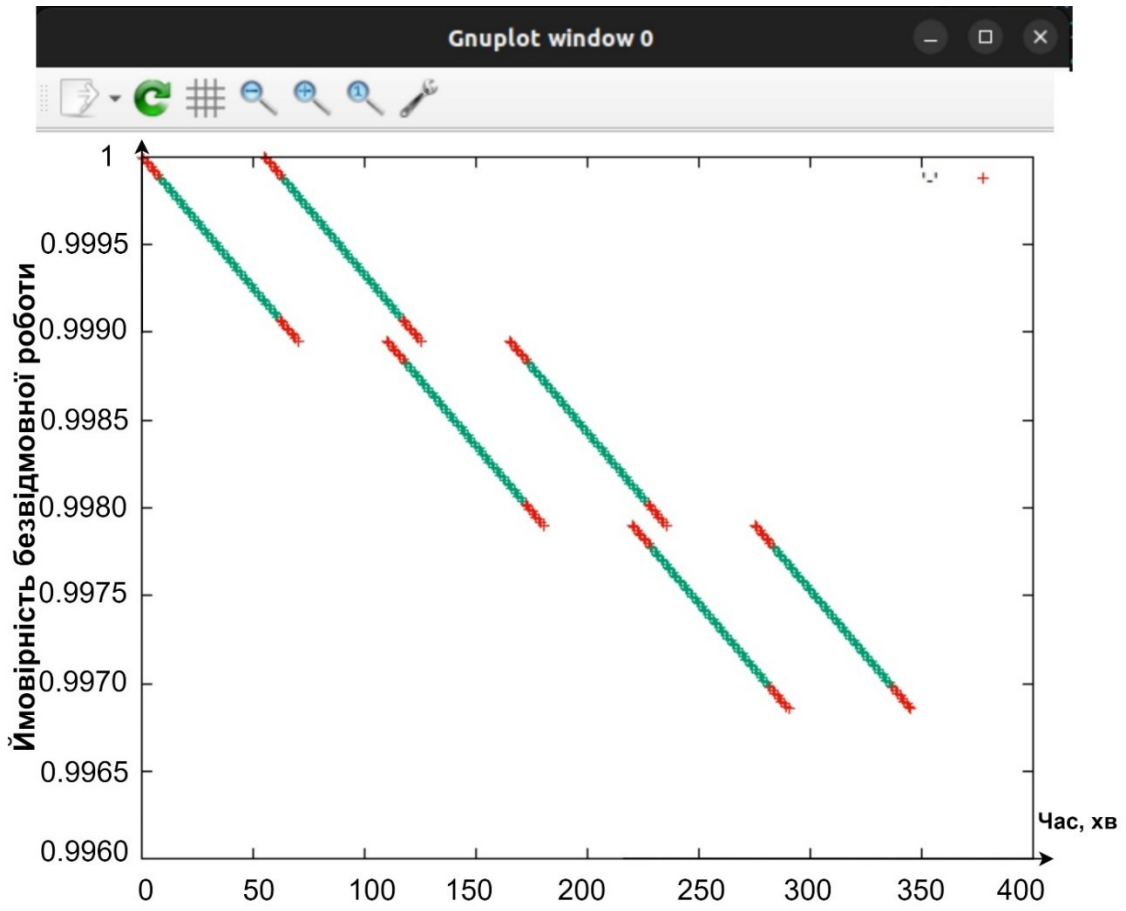

Рисунок 4.23 − Узагальнюючий графік залежності ймовірності безвідмовної роботи від часу для обох змін

4.3 Висновки до четвертого розділу

1. Розроблено програмний засіб “Simulation Way” для підтримки планування розгортання БПЛА для утворення LiFi мережі, який має трьохрівневу архітектуру та складається з наступних рівнів:

– рівень графічного інтерфейсу користувача (GUI), що являє собою інтерфейс програмного засобу і реалізований за допомогою бібліотеки Python з назвою tkinter;

– рівень Business logic, який надає логіку взаємодії між сховищем даних (Storage data) та графічним інтерфейсом (GUI) та містить модулі генерування звітів (результатів), модулі розрахункового ядра та модулі зовнішніх алгоритмів, які можуть взаємодіяти між собою;

– рівень Storage data, який відповідає за отримання і зберігання у вигляді файлів даних результатів розрахунків та звітів щодо процесу роботи програмного засобу.

2. Оператору КЦ, що використовує програмний засіб “Simulation Way”, доступна велика кількість налаштувань: встановлювати розміри робочої площі виробничого приміщення, генерувати необхідну кількість перешкод з заданими характеристиками та обирати методи (правила) обходу цих перешкод. Крім того, сформовані вхідні дані під час однієї ітерації моделювання можуть використовуватися у наступних ітераціях.

3. Розроблено програмний засіб “Reliability Level”, який може бути використаний операторами КЦ для визначення часу функціонування літаючої LiFi мережі з рівнем надійності не менше мінімально допустимого протягом заданого часу.

4. Програмний засіб “Reliability Level” має однорівневу структуру та складається з монолітного модуля, який поєднує в собі всю логіку.

5. Взаємодія оператора КЦ з програмним засобом “Reliability Level” здійснюється за допомогою консольного інтерфейсу. Оператору КЦ доступна можливість введення наступних параметрів:

– інтенсивність відмов БПЛА;

– час налаштування літаючої LiFi мережі;

– час функціонування літаючої LiFi мережі в режимі передачі даних;

– часовий ресурс бортової батареї БПЛА;

– мінімально припустимий рівень надійності, з яким може функціонувати літаюча LiFi мережа.

ВИСНОВКИ

1. У дисертації розроблено методи і програмні засоби підтримки розгортання безпілотних літаючих LiFi мереж для забезпечення передачі даних в умовах руйнувань з урахуванням вимог до часових і надійнісних характеристик.

2. Вперше запропоновано методи планування розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань, які на відміну від відомих враховують структурно-просторові параметри перешкод і базуються на комбінуванні алгоритмів прокладання маршрутів за різними стратегіями їх обходу, та надають змогу мінімізувати довжину маршруту та/або кількість необхідних БПЛА.

3. Удосконалено метод розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань шляхом формування та вибору маршрутів польоту у визначені точки у просторі для утворення мережі з урахуванням різних варіантів базування БПЛА, що зменшує сумарні часові та вартісні витрати на розгортання мережі. Зокрема запропоновано такі стратегії розгортання БПЛА для утворення літаючої LiFi мережі:

– стратегія першої точки маршруту, яка передбачає рух кожного БПЛА до першої точки LiFi маршруту і подальше розгортання мережі в межах нього;

– стратегія радіального руху, яка передбачає рух кожного БПЛА одразу до точки призначення на LiFi маршруті;

– стратегія середньої точки маршруту, яка передбачає рух кожного БПЛА до точки, максимально наближеної до середини LiFi маршруту, і подальше розгортання мережі в межах нього.

4. Удосконалено метод підвищення надійності літаючої LiFi мережі шляхом розроблення моделі та алгоритмів випереджувальної заміни БПЛА з урахуванням вимог до ймовірності безвідмовної роботи мережі, показників безвідмовності та автономності окремих БПЛА, що забезпечує гарантоване функціонування мережі впродовж заданого часу.

5. Практичні результати полягають у доведенні теоретичних положень дисертаційної роботи до конкретних алгоритмів та програмних засобів для підтримки планування розгортання літаючої LiFi мережі. Зокрема розроблено:

– програмний засіб “Simulation Way”, який забезпечує: прокладання LiFi маршрутів від джерела до споживача інформації у приміщенні з перешкодами з позначенням на маршрутах точок (місць) розміщення БПЛА для утворення літаючої LiFi мережі з використанням для обходу перешкод алгоритмів лівого та правого кутів, а також керованого водоспаду; формування графу можливих LiFi маршрутів і визначення найкоротшого з них шляхом застосування алгоритму Дейкстри; розміщення БПЛА у визначених точках (місцях) на прокладеному найкоротшому LiFi маршруті з використанням різних стратегій розгортання;

– програмний засіб “Reliability Level”, який розраховує кількість БПЛА та їх змін для забезпечення безперебійного функціонування літаючої LiFi мережі із визначеним рівнем надійності впродовж заданого часу та дозволяє: візуалізувати цикл роботи однієї зміни рою БПЛА для розгортання і забезпечення функціонування LiFi мережі (виліт з депо та підліт до визначених точок на прокладеному LiFi маршруті, налаштування мережі, передача даних, повернення до депо, очікування на наступний цикл роботи); надавати графічну інтерпретацію почергової роботи змін БПЛА для розгортання і забезпечення безперебійного функціонування літаючої LiFi мережі, відповідно до якої БПЛА наступної зміни прилітають і налаштовують утворюваними ними мережу до початку руху БПЛА поточної зміни до депо.

6. Результати дисертаційної роботи впроваджено:

– у товаристві з обмеженою відповідальністю «СІДІ ЛІНК» (акт впровадження від 29 березня 2024) під час розроблення проєктів комп’ютерних мереж різного призначення;

– у навчальному процесі Національного аерокосмічного університету ім. М. Є. Жуковського «Харківський авіаційний інститут» (акт впровадження від 01 квітня 2024);

– при виконанні науково-дослідних проєктів, що виконувались у Національному аерокосмічному університеті ім. М. Є. Жуковського «Харківський авіаційний інститут» (акт впровадження від 01 квітня 2024).

7. Подальші дослідження доцільно зосередити на розробленні:

– алгоритмів прокладання LiFi маршрутів в обхід перешкод у тривимірному просторі;

– розробленні стратегій розгортання літаючих LiFi мереж з використанням БПЛА-носія та методів забезпечення надійності таких мереж з використанням трьох і більше змін.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pevnev V., Plakhteev A., Tsuranov M., Zemlianko H., Leichenko K. “Smart City” Technology: Conception, Security Issues and Cases. *Lecture Notes in Networks and Systems*. 2022. Vol. 367. P. 207–218. DOI: 10.1007/978-3-030-94259-5_19.

2. Leichenko K., Fesenko H., Kharchenko V. Deploying the Reliable UAV Swarm for Providing P2P LiFi Communications Considering Physical Obstacles: Method of Rectangles, Algorithms, and Tool. *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS’2023)* : Proc. 12th IEEE Int. Conf., Dortmund, Germany, Sept. 07–09, 2023. P. 1011–1016. DOI: 10.1109/IDAACS58523.2023.10348819.

3. Leichenko K., Fesenko H., Borges J., Kharchenko V. Search for the Shortest Route Considering Physical Obstacles: Method of Controlled Waterfall, Tool, and Application. *Dependable Systems, Services and Technologies (DESSERT’2023)* : Proc. 13th IEEE Int. Conf., Athens, Greece, Oct. 13–15, 2023. P. 1–5. DOI: 10.1109/DESSERT61349.2023.10416479.

4. Лейченко К. М., Фесенко Г. В. Програмний засіб підтримки планування розгортання LiFi мережі на основі БПЛА для забезпечення передачі даних в умовах руйнувань. *Системи управління, навігації та зв’язку*. 2024. Вип. 1 (75). С. 193–200. DOI: 10.26906/SUNZ.2024.1.193.

5. Leichenko K., Fesenko H., Kharchenko V., Illiashenko O. Deployment of a UAV swarm-based LiFi network in the obstacle-ridden environment: algorithms of finding the path for UAV placement. *Radioelectronic and Computer Systems*. 2023. No. 1(109). P. 176–195. DOI: 10.32620/reks.2024.1.14.

6. Лейченко К., Фесенко Г., Харченко В. Стратегії розгортання та методи забезпечення надійності рою БПЛА для утворення LiFi мережі. *Вимірювальна та обчислювальна техніка в технологічних процесах*. 2024. №1. С. 21–31. DOI: 10.31891/2219-9365-2024-77-3.

7. Li J., Xiong X., Yan Y., Yang Y. A Survey of Indoor UAV Obstacle Avoidance Research. *IEEE Access*. 2023. Vol. 11. P. 51861–51891. DOI: 10.1109/ACCESS.2023.3262668.

8. Unlu H. U., Chaikalis D., Tsoukalas A., Tzes A. UAV Indoor Exploration for Fire-Target Detection and Extinguishing. *Journal of Intelligent & Robotic Systems*. 2023. Vol. 108, article no. 54. DOI: 10.1007/s10846-023-01835-0.

9. Pliakos C., Vlachos S., Bliamis C., Yakinthos K. Preliminary design of a multirotor UAV for indoor search and rescue applications. *Journal of Physics: Conference Series*. 2024. Vol. 2716, no. 1, article no. 012067. DOI: 10.1088/1742-6596/2716/1/012067.

10. Solis J., Karlsson C., Johansson S., Richardsson K. Towards the development of an automatic UAV-based indoor environmental monitoring system: Distributed off-board control system for a micro aerial vehicle. *Applied Sciences*. 2021. Vol. 11, no. 5. P. 1–16. DOI: 10.3390/app11052347.

11. Seo S.-H., Choi J.-I., Song J. Secure Utilization of Beacons and UAVs in Emergency Response Systems for Building Fire Hazard. *Sensors*. 2017. Vol. 17, article no. 2200. DOI: 10.3390/s17102200.

12. Hamledari H., Sajedi S., McCabe B., Fischer M. Automation of Inspection Mission Planning Using 4D BIMs and in Support of Unmanned Aerial Vehicle–Based Data Collection. *Journal of Construction Engineering and Management*. 2021. Vol. 147, no. 3. DOI: 10.1061/(asce)co.1943-7862.0001995.

13. Yang J., Ding T., Deng Q., Li Z., Wang Y., Wu J., Shi M. A Multi-UAV Indoor Air Real-Time Detection and Gas Source Localization Method Based on Improved Teaching–Learning-Based Optimization. *Atmospheric Environment*. 2024. Vol. 318, article no. 120200. DOI: 10.1016/j.atmosenv.2023.120200.

14. Neumann P. P., Hüllmann D., Bartholmai M. Concept of a gas-sensitive nano aerial robot swarm for indoor air quality monitoring. *Materials Today: Proceedings*. 2019. Vol. 12. P. 470–473. DOI: 10.1016/j.matpr.2019.03.151.

15. Stanko J., Stec F., Rodina J. Process automation of warehouse inspection using an autonomous unmanned aerial vehicle. *MM Science Journal*. 2022. P. 5864–5869. DOI: 10.17973/MMSJ.2022_10_2022063.

16. González de Santos L. M., Frías Nores E., Martínez Sánchez J., González Jorge H. Indoor Path-Planning Algorithm for UAV-Based Contact Inspection. *Sensors*. 2021. Vol. 21, no. 2, article no. 642. DOI: 10.3390/s21020642.

17. Gao C., Wang X., Wang R., Zhao Z., Zhai Y., Chen X., Chen B. M. A UAV-based explore-then-exploit system for autonomous indoor facility inspection and scene reconstruction. *Automation in Construction*. 2023. Vol. 148, article no. 104753. DOI: 10.1016/j.autcon.2023.104753.

18. Xiao X., Fan Y., Dufek J., Murphy R. Indoor UAV Localization Using a Tether. *Safety, Security, and Rescue Robotics (SSRR)* : Proc. IEEE Int. Symp., Philadelphia, PA, USA, Aug. 06–08, 2018. P. 1–6. DOI: 10.1109/SSRR.2018.8468627.

19. Yu T., Jiliang Z., Gaofeng P., Mohamed-Slim A. Satellite-UAV Communications. *UAV Communications: Modeling and Analyses* / ed. by G. Pan, Xi. Miao, X. Yang, Z. Yang. Cham, Switzerland : Springer, 2024. P. 113–171. DOI: 10.1007/978-981-97-0383-8_4.

20. Wang J., Jiang C., Kuang L. High-Mobility Satellite-UAV Communications: Challenges, Solutions, and Future Research Trends. *IEEE Communications Magazine*. 2022. Vol. 60, no 5. P. 38–43. DOI: 10.1109/MCOM.001.2100850.

21. Xiao X., Dufek J., Suhail M., Murphy R. Motion Planning for a UAV with a Straight or Kinked Tether. *Intelligent Robots and Systems (IROS)* : Proc. IEEE/RSJ Int. Conf., Oct. 01–05, 2018. P. 8486–8492. DOI: 10.1109/IROS.2018.8594461.

22. Han B., Qu T., Tong X., Jiang J., Zlatanova S., Wang H., Cheng C. Grid-optimized UAV indoor path planning algorithms in a complex environment. *International Journal of Applied Earth Observation and Geoinformation*. 2022. Vol. 111, article no. 102857. DOI: 10.1016/j.jag.2022.102857.

23. Sawalmeh A., Noor O. An Overview of Collision Avoidance Approaches and Network Architecture of Unmanned Aerial Vehicles (UAVs). *International Journal of Engineering & Technology*. 2018. Vol. 7, no 35. DOI: 10.14419/ijet.v7i4.35.27395.

24. Liu Z., Liu H., Lu Z., Zeng Q. A Dynamic Fusion Pathfinding Algorithm Using Delaunay Triangulation and Improved A-Star for Mobile Robots. *IEEE Access*. 2021. Vol. 9. P. 20602–20621. DOI: 10.1109/ACCESS.2021.3055231.

25. Yang L., Fu L., Li P., Mao J., Guo, N. An Effective Dynamic Path Planning Approach for Mobile Robots Based on Ant Colony Fusion Dynamic Windows. *Machines*. 2022. Vol. 10, no. 1. DOI: 10.3390/machines10010050.

26. He Y., Zeng Q., Liu J., Xu G., Deng X. Path planning for indoor UAV based on Ant Colony Optimization. *Proc. 25th Chinese Control and Decision Conference (CCDC)*, Guiyang, China, May 25–27, 2013. P. 2919–2923. DOI: 10.1109/CCDC.2013.6561444.

27. Zhang L., Zhang Y., Li Y. Path planning for indoor Mobile robot based on deep learning. *Optik*. 2020. Vol. 219, article no. 16509. DOI: 10.1016/j.ijleo.2020.165096.

28. Guruprasad Y. K., Nageswara Guptha M. Autonomous UAV Object Avoidance with Floyd–Warshall Differential Evolution (FWDE) approach. *Inteligencia Artificial*. 2022. Vol. 25. P. 77–94. DOI: 10.4114/intartif.vol25iss70pp77-94.

29. Penicka R., Scaramuzza D. Minimum-Time Quadrotor Waypoint Flight in Cluttered Environments. *IEEE Robotics and Automation Letters*. 2022. Vol. 7. P. 5719–5726. DOI: 10.1109/LRA.2022.3154013.

30. Jacob H., Forrest J., Alaa S., Hamza T., Kar D. Path Planning for Indoor UAV Using A* and Late Acceptance Hill Climbing Algorithms Utilizing Probabilistic Roadmap. *International Journal of Engineering & Technology*. 2020. Vol. 9, no. 4, article no. 857. DOI: 10.14419/ijet.v9i4.31033.

31. Bolognini M., Fagiano L. A. Scalable Hierarchical Path Planning technique for Autonomous Inspections with multicopter drones. *Proc. European Control Conference (ECC)*, Delft, Netherlands, Jun. 29 – Jul. 02, 2021. P. 787–792. DOI: 10.23919/ECC54610.2021.9655086.

32. Jin Q., Hu Q., Zhao P., Wang S., Ai M. An Improved Probabilistic Roadmap Planning Method for Safe Indoor Flights of Unmanned Aerial Vehicles. *Drones*. 2023. Vol. 7, no. 2, article no. 92. DOI: 10.3390/drones7020092.

33. Sadiq B. O., Salawudeen A. T., Sha'aban Y. A., Adedokun E. A., Mu'azu M. B. Interface protocol design: A communication guide for indoor FANET. *Telkomnika*. 2019. Vol. 17. P. 3175–3182. DOI: 10.12928/TELKOMNIKA.v17i6.13296.

34. Jayashree T. R., Priyadarshini K., Sri Harini K. LiFi & Wi-Fi based drone for weather monitoring with data storage in the cloud using IoT. *Advances in Parallel Computing*. 2021. Vol. 38. P. 434–438. DOI: 0.3233/APC210079.

35. Sadiq B. O. A Feasibility Study of Using Light Fidelity with Multiple Unmanned Aerial Vehicles for Indoor Collaborative and Cooperative Networking. URL: https://arxiv.org/abs/1707.08627 (date of access: 08.01.2024).

36. Ghaderi M. R. LiFi and Hybrid WiFi/LiFi indoor networking: From theory to practice. *Optical Switching and Networking*. 2022. Vol. 47, no. 100699. DOI: 10.1016/j.osn.2022.100699.

37. Sharma R., Gurjar D. S., Rahman E., Raghav A., Shukla P., Mishra V. LiFi Technology: A Breakthrough for Massive Data Rates in Indoor Applications. *Intelligent Systems for Social Good*. 2022. P. 63–79. DOI: 10.1007/978-981-19-0770-8_6.

38. Haas H., Yin L., Wang Y., Chen C. What is LiFi? *Journal of Lightwave Technology*. 2016. Vol. 34, no 6. P. 1533–1544. DOI: 10.1109/JLT.2015.2510021.

39. Jungnickel V., Berenguer P. W., Mana S. M., Hinrichs M., Kouhini S. M., Bober K. L., Kottke C. LiFi for Industrial Wireless Applications. *Proc. Optical Fiber Communication Conference (OFC)*, San Diego, CA, USA, Mar. 8–12, 2020. P. 1–3. 10.1364/OFC.2020.M3I.1.

40. Amran N. A., Soltani M. D., Yaghoobi M., Safari M. Learning Indoor Environment for Effective LiFi Communications: Signal Detection and Resource Allocation. *IEEE Access*. 2022. Vol. 10. P. 17400–17416. DOI: 10.1109/ACCESS.2022.3150919.

41. Arfaoui M. A., Soltani M. D., Tavakkolnia I., Ghrayeb A., Assi C. M., Safari M., Haas H. Measurements-Based Channel Models for Indoor LiFi Systems. *IEEE Transactions on Wireless Communications.* 2020. Vol. 20, no. 2. P. 827–842. DOI: 10.1109/TWC.2020.3028456.

42. Ma S., Sheng H., Sun J., Li H., Liu X., Qiu C., Safari M., Al-Dhahir N., Li S. Feasibility Conditions for Mobile LiFi. *IEEE Transactions on Wireless Communications*. 2023. DOI: 10.1109/TWC.2023.3346056.

43. Muller M., Behnke D., Bok P. B., Kottke C., Bober K. L., Jungnickel V. Leverage LiFi in Smart Manufacturing. *Proc. IEEE Globecom Workshops (GC Wkshps)*, Taipei, Taiwan, Dec. 07–11, 2020. P. 1–6. DOI: 10.1109/GCWkshps50303.2020.9367446.

44. Ozyurt A. B., Tinnirello I., Popoola W. O. LiFi-enabled UAV swarm networks. *Applied Optics*. 2024. Vol. 63, no. 6. P. 1471–1480. DOI: 10.1364/AO.506515.

45. Leon T., Sumiko M. A UAV traffic prediction considering AP connection errors and UAV altitude. *Nonlinear Theory and Its Applications, IEICE*. 2024. Vol. 15, no. 2. P. 365–375. DOI: 10.1587/nolta.15.365.

46. Gordon V., Danquah P. An Experimental Assessment of LiFi Data Communication. *Ghana Journal of Science*. 2020. Vol. 61, no. 1. P. 73–87. DOI: 10.4314/gjs.v61i1.6.

47. Vaiopoulos N., Vavoulas A., Sandalidis H. G. Impact of a Randomly Placed Terminal on LiFi Performance. *IEEE Transactions on Communications*. 2022. Vol. 70, no. 3. P. 1875–1885. DOI:10.1109/TCOMM.2021.3133930.

48. Dil A., Rizwana A., Salameh B., Hany E., Moussa A. Performance analysis of neural network-based unified physical layer for indoor hybrid LiFi–WiFi flying networks. *Neural Computing and Applications*. 2023. Vol. 35, no. 34. P. 1–11. DOI: 10.1007/s00521-023-09017-7.

49. Oubbati O., Atiquzzaman M., Ahanger T., Atef I. Softwarization of UAV Networks: A Survey of Applications and Future Trends. *IEEE Access*. 2020. Vol. 8. P. 98073–98125. DOI: 10.1109/ACCESS.2020.2994494.

50. Hu J., Cai L., Khosravi M., Pei Q., Wan S. UAV-Assisted Vehicular Edge Computing for the 6G Internet of Vehicles: Architecture, Intelligence, and Challenges. *IEEE Communications Standards Magazine*. 2021. Vol. 5, no. 2. P. 12–18. DOI: 10.1109/MCOMSTD.001.2000017.

51. Rahul G., Srikumar V., Amandeep R., Deepa K. LiFi for Vehicle to Vehicle Communication – A Review. *Procedia Computer Science*. 2020. Vol. 165. P. 25–31. DOI: 10.1016/j.procs.2020.01.066.

52. Devi G., Jayanthi N., Rahul S., Karthick M., Srinivasan G., Anand M. A critical review on LiFi technology and its future applications. *AIP Conference Proceedings*. 2023. Vol. 2690, no. 1, article no. 020061. P. 1–7. DOI: 10.1063/5.0121967.

53. Chowdhury M., Hasan M. K., Shahjalal M., Hossan M., Jang Y. Optical Wireless Hybrid Networks: Trends, Opportunities, Challenges, and Research Directions. *IEEE Communications Surveys & Tutorials*. 2020. Vol. 22, no. 2. P. 930–966. DOI: 10.1109/COMST.2020.2966855.

54. Ramadhani E. A Mini Review of LiFi Technology: Security Issue. *International Journal of Computer and Information System (IJCIS)*. 2020. Vol. 3, no. 3. P. 90–93. DOI: 10.29040/ijcis.v3i3.74.

55. Kouhini S. M., Kottke C., Ma Z., Freund R., Jungnickel V., Muller M., Behnke D., Vazquez M. M., Linnartz J.-P. M. G. LiFi Positioning for Industry 4.0. *IEEE Journal of Selected Topics in Quantum Electronics.* 2021. Vol. 27, no. 6. P. 1–15. DOI: 10.1109/JSTQE.2021.3095364.

56. Muraleedharan N., Cohen D. S. Modelling and simulation of UAV systems. *Imaging and Sensing for Unmanned Aircraft Systems: Control and Performance*. 2020. P. 101–121. DOI:10.1049/PBCE120F_ch5.

57. Udvardy P., Beszedes B., Toth B., Foldi A., Botos A. Simulation of obstacle avoidance of an UAV. *New Trends in Aviation Development (NTinAD'2020)* : Proc. 15th IEEE Int. Conf., Starý Smokovec, Slovakia, Sept. 17–18, 2020. P. 245–249. DOI: 10.1109/NTAD51447.2020.9379113.

58. Romaniuk S., Gosiewski Z., Ambroziak L. A ground control station for the UAV flight simulator. *Acta Mechanica et Automatica*. 2016. Vol 10, no. 1. P. 28–32. DOI: 10.1515/ama-2016-0005.

59. Zhenxiong W., Kai K., Xueying H., Huanming H., Jianghai L., Lang C. Computational simulation study on disturbance of six-rotor UAVs due to ammunition

launch. *Journal of Physics: Conference Series*. 2023. Vol. 2478, article no. 102022. P. 1–19. DOI: 10.1088/1742-6596/2478/10/102022.

60. Chandrasekaran K., Theningaledathil V., Hebbar A. Ground based variable stability flight simulator. *Aviation*. 2021. Vol. 25, no. 1. P. 22–34. DOI: 10.3846/aviation.2021.13564.

61. Kampf R., Soviar J., Bartuška L., Kubina M. Creation of SW for Controlling Unmanned Aerial Systems. *LOGI - Scientific Journal on Transport and Logistics*. 2022. Vol. 13, no. 1. P. 198–209. DOI: 10.2478/logi-2022-0018.

62. Phadke A., Medrano F. A., Sekharan C. N., Chu T. Designing UAV Swarm Experiments: A Simulator Selection and Experiment Design Process. *Sensors*. 2023. Vol. 23, no. 17, article no. 7359. P. 1–26. DOI: 10.3390/s23177359.

63. Al-Mousa A., Sababha B. H., Al-Madi N., Barghouthi A., Younisse R. UTSim: A framework and simulator for UAV air traffic integration, control, and communication. *International Journal of Advanced Robotic Systems*. 2019. Vol. 16, no. 5. P. 1–19. DOI: 10.1177/1729881419870937.

64. Jayashree T. R., Priyadarshini K., Sri Harini K. Li-Fi & Wi-Fi based drone for weather monitoring with data storage in cloud using IoT. *Advances in Parallel Computing*. 2021. Vol. 38. P. 434–438. DOI: 10.3233/APC210079.

65. Ahmad R., Ayyash M., Salameh H. B., El-Khazali R., Eigala H. Indoor Flying Networks for 6G: Concepts, Challenges, Enabling Technologies, and Opportunities. *IEEE Communications Magazine*. 2023. Vol. 61, no. 10. P. 156–162. DOI: 10.1109/MCOM.009.2200559.

66. Qi X., Zhou Y., Liu L. Evaluation of the reliability of UAV swarm for ground combat missions. *Systems Engineering and Electronics*. 2023. Vol. 45, no. 9. P. 2971–2978. DOI: 10.12305/j.issn.1001-506X.2023.09.38.

67. Zaitseva E., Levashenko V., Mukhamediev R., Brinzei N., Kovalenko A., Symagulov A. Review of Reliability Assessment Methods of Drone Swarm (Fleet) and a New Importance Evaluation Based Method of Drone Swarm Structure Analysis. *Mathematics*. 2023. Vol. 11, no. 11, article no. 2551. P. 1–26. DOI: 10.3390/math11112551.

68. Zaitseva E., Levashenko V., Brinzei N., Kovalenko A., Yelis M., Gopejenko V., Mukhamediev R. Reliability Assessment of UAV Fleets. *Lecture Notes in Electrical Engineering*. 2023. Vol. 965. P. 335–357. DOI: 10.1007/978-3-031-24963-1_19.

69. Фесенко Г. В., Харченко В. С. Моделі надійності угруповань флотів БПЛА з ковзним резервуванням для моніторингу потенційно небезпечних об’єктів. *Радіоелектронні і комп’ютерні системи*. 2019. № 2 (90). С. 147–156. DOI: 10.32620/reks.2019.2.14.

ДОДАТОК А.
АКТИ ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОЇ РОБОТИ

Затверджую
Проректор з науково-педагогічної роботи
Національного аерокосмічного університету
ім. М. Є. Жуковського
«Харківський авіаційний інститут»
к.т.н., доцент
_____ Андрій ГУМЕННИЙ
« 01 » квітня 2024 року

АКТ ВПРОВАДЖЕННЯ

наукових результатів дисертаційної роботи
Лейченка Кирила Миколайовича,
виконаної на здобуття наукового ступеня доктора філософії,
у навчальному процесі кафедри комп'ютерних систем, мереж і кібербезпеки
Національного аерокосмічного університету
ім. М. Є. Жуковського «Харківський авіаційний інститут»

Комісія у складі: голови комісії – декана факультету радіоелектроніки, комп'ютерних систем та інфокомунікацій к.т.н., доцента Олексія ОДОКІЄНКА, і членів – завідувача кафедри комп'ютерних систем, мереж і кібербезпеки д.т.н., професора Вячеслава ХАРЧЕНКА, професора кафедри комп'ютерних систем, мереж і кібербезпеки к.т.н., професора Клайда ФУРМАНОВА, доцента кафедри комп'ютерних систем, мереж і кібербезпеки к.т.н., старшого наукового співробітника Ігоря КЛЮШНІКОВА встановила, що наукові результати, а саме:

– методи планування розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань;

– метод розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань;

– метод підвищення надійності літаючої LiFi мережі;

реалізовані у навчальному процесі кафедри комп'ютерних систем, мереж і кібербезпеки у вигляді:

лекційного матеріалу і практичних занять з використанням алгоритмів і програмних засобів для системи підтримки планування розгортання БПЛА для утворення літаючої LiFi мережі та забезпечення її надійного функціонування у навчальних дисциплінах «Мобільне програмування» (4 години), «Надійність та відмовостійкість комп'ютерних систем» (4 години) і «Програмування систем IoT» (4 години).

Це дозволило покращити наочність і практичну спрямованість викладання матеріалу із застосування сучасних бездротових технологій, інтернету речей, розгортання та забезпечення надійності літаючих сенсорних мереж, підвищити якість підготовки фахівців за спеціальністю 123 Комп'ютерна інженерія.

Голова комісії Олексій ОДОКІЄНКО

Члени комісії Вячеслав ХАРЧЕНКО

Клайд ФУРМАНОВ

Ігор КЛЮШНІКОВ

Затверджую
Проректор з наукової роботи
Національного аерокосмічного університету
ім. М. Є. Жуковського
«Харківський авіаційний інститут»
д.т.н., професор
_____ Володимир ПАВЛІКОВ
« 01 » квітня 2024 року

АКТ ВПРОВАДЖЕННЯ

наукових результатів дисертаційної роботи
Лейченка Кирила Миколайовича,
виконаної на здобуття наукового ступеня доктора філософії,
у науково-дослідних проєктах Національного аерокосмічного університету
ім. М. Є. Жуковського «Харківський авіаційний інститут»

Комісія у складі: голови – декана факультету радіоелектроніки, комп'ютерних систем та інфокомунікацій к.т.н., доцента Олексія ОДОКІЄНКА і членів – завідувача кафедри комп'ютерних систем, мереж і кібербезпеки д.т.н., професора Вячеслава ХАРЧЕНКА, професора кафедри комп'ютерних систем, мереж і кібербезпеки д.т.н, професора Ольги МОРОЗОВОЇ, доцента кафедри комп'ютерних систем, мереж і кібербезпеки к.т.н., старшого наукового співробітника Ігоря КЛЮШНІКОВА, встановила, що наукові результати, а саме:

– методи планування розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань;

– метод розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань;

– метод підвищення надійності літаючої LiFi мережі;

реалізовані у вигляді наукових положень і розробок, використаних при виконанні науково-дослідних проєктів за замовленням Міністерства освіти та науки України:

– Методи, програмно-апаратні засоби та інформаційні технології розроблення і модернізації гарантоздатних комп'ютерних систем, мереж та IT-інфраструктур (Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», ДР № 0117U005349, 2018-2020);

– Методологічні засади та технології оцінювання та забезпечення безпеки (захисту) критичних інформаційних інфраструктур (Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», ДР № 0119U100979, 2019-2021);

– Методи, моделі та інформаційні технології підвищення надійності та безпечності складних IT-систем на етапах розроблення та впровадження (Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», (ДР № 0121U113842, 2021-2023);

– Наукові засади і методи забезпечення гарантоздатності флотів БПЛА інтелектуальних систем моніторингу потенційно небезпечних і військових об'єктів (Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», ДР № 0121U112172, 2021-2023).

Це дозволило підвищити зменшує сумарні часові та вартісні витрати на розгортання бездротових літаючих мереж систем моніторингу критичних інфраструктур, які досліджувалися в рамках виконання НДР впродовж 2019-2023 рр.

Голова комісії Олексій ОДОКІЄНКО

Члени комісії Вячеслав ХАРЧЕНКО

Ольга МОРОЗОВА

Ігор КЛЮШНІКОВ

ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ «СІДІ ЛІНК»

61166, м. Харків
вул. Серпова, 4
р/р UA143515330000026000052147747
в ХГРУ АТ КБ "ПРИВАТБАНК»
МФО 339500
Код ЄДРПОУ 37575641
ІПН 375756420301
cdlink1977@gmail.com
тел. +38(050)-915-24-32

ЗАТВЕРДЖУЮ
Директор ТОВ «СІДІ ЛІНК»
_____Дмитро КОЧКАР
«29» березня 2024 р.

АКТ ВПРОВАДЖЕННЯ

наукових результатів дисертаційної роботи
Лейченка Кирила Миколайовича,
виконаної на здобуття наукового ступеня доктора філософії,
у ТОВ «СІДІ ЛІНК»

Комісія у складі Голови комісії – Генерального директора Дмитра КОЧКАРЯ та членів комісії – фінансового директора Юлії КОЧКАР, бухгалтера Галини ГОЛОВИНОЇ склала цей акт про те, що наукові результати, а саме:

– методи планування розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань;

– метод розміщення БПЛА літаючої LiFi мережі для забезпечення передачі даних в умовах руйнувань;

– метод підвищення надійності літаючої LiFi мережі,

впроваджені в ТОВ «СІДІ ЛІНК».

Зазначені результати було використано під час розроблення проєктів комп'ютерних мереж і мобільних систем різного призначення, що надало змогу покращити часові та надійнісні показники варіантів при прийнятті відповідних проєктних рішень.

УКРАЇНА ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ «СІДІ ЛІНК» №37575641 МІСТО ХАРКІВ

Голова комісії: Дмитро КОЧКАР

Члени комісії: Юлія КОЧКАР

Галина ГОЛОВИНА

ДОДАТОК Б.
ЛІСТИНГИ КОДІВ ПРОГРАМНИХ ЗАСОБІВ

Б.1 Лістинг коду програмного засобу “Simulation Way”

Way_core.py

```
import math

RIGHT_WAY = 0
LEFT_WAY  = 1
MIN_WAY   = 2
SPLIT     = 3

coef = 1

def calculate_distance(x1, y1, x2, y2):
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
    return distance

def correction_coordinates(coord_1, coord_2, coef):
    if int(coord_1) == int(coord_2):
        return True
    if int(coord_1) + coef == int(coord_2):
        return True
    if int(coord_1) == int(coord_2) + coef:
        return True

    return False
#
#        A
#   ----------
#   |        |
# D |        | B
```

```
#  |      |
#   ----------
#       C
#
#
def coorect_avoid_obstacle(dx, dy, x, y, way, obstacle):
    minx, miny, maxx,maxy = obstacle.bounds
    new_dx = 0
    new_dy = 0
    if correction_coordinates(y, miny, coef): #int(miny) == int(y):
        if way == RIGHT_WAY:
            new_dx = dy
            new_dy = 0
        if way == LEFT_WAY:
            new_dx = -dy
            new_dy = 0
        # print("A")
    elif correction_coordinates(x, maxx, coef): #int(maxx) == int(x):
        if way == RIGHT_WAY:
            new_dx = 0
            new_dy = dy
        if way == LEFT_WAY:
            new_dx = 0
            new_dy = -dy
    elif correction_coordinates(y, maxy, coef): #int(maxy) == int(y):
        if way == RIGHT_WAY:
            new_dx = dy
            new_dy = 0
        if way == LEFT_WAY:
            new_dx = -dy
            new_dy = 0
    elif correction_coordinates(x, minx, coef): #int(minx) == int(x):
        if way == RIGHT_WAY:
            new_dx = 0
            new_dy = dx
        if way == LEFT_WAY:
```

```
            new_dx = 0
            new_dy = dy
    else:
                print("x " + str(x))
                print("minx " + str(minx))
                print("maxx " + str(maxx))
                print("y " + str(y))
                print("miny " + str(miny))
                print("maxy " + str(maxy))
                print("ERROR")

    return new_dx, new_dy

def calculate_total_distance(coord_list):
    total_distance = 0

    for i in range(1, len(coord_list)):
        tmp1, x1, y1 = coord_list[i - 1]
        tmp2, x2, y2 = coord_list[i]
        distance = calculate_distance(x1, y1, x2, y2)
        total_distance += distance

    return total_distance

def prepare_coordinates(graph_points, obstacles):
    coordinates = []
    f = open('coordinates.txt', 'w' )

    for graph_point in graph_points:
        for i in range(len(graph_point)):
                id, x, y = graph_point[i]
                if [x,y] in coordinates:
                    continue
                else:
                    coordinates.append([int(x),int(y)])
```

```
    for item in coordinates:
        f.write("%s\n" % item)

    return coordinates

def prepare_obstacles(obstacles):
    obstacles_coordinates = []
    f = open('obstacles_coordinates.txt', 'w' )

    for obstacle in obstacles:
        minx, miny, maxx,maxy = obstacle.bounds

        obstacles_coordinates.append([int(minx), int(miny), int(maxx), int(maxy)])

    for item in obstacles_coordinates:
        f.write("%s\n" % item)

    return obstacles_coordinates
```

storage.py

```
CANVAS_WIDTH = 1000
CANVAS_HEIGHT = 1000
UAV_SIZE = 5

X_INDEX      = 0
Y_INDEX      = 1

X_END_INDEX   = 2
Y_END_INDEX   = 3

WAY_INDEX     = 4

MAX_DISTANCE = 500 #10 meter
NUM_OBSTACLES = 11
```

```
UAV_COUNT = 1

DBG_VALUE = 0

dijkstra_coord = []
a_star_coord = []
best_first_coord = []

def get_dijkstra_coord():
    return dijkstra_coord

def set_dijkstra_coord(value):
    global dijkstra_coord

    dijkstra_coord = value

def get_a_star_coord():
    return a_star_coord

def set_a_star_coord(value):
    global a_star_coord

    a_star_coord = value

def get_best_first_coord():
    return best_first_coord

def set_best_first_coord(value):
    global best_first_coord

    best_first_coord = value

def set_debug_value(value):
    global DBG_VALUE
    DBG_VALUE = value
```

```
def get_debug_value():
    return DBG_VALUE

def get_width_height():
    return CANVAS_WIDTH, CANVAS_HEIGHT

def get_uav_size():
    return UAV_SIZE

def get_X_INDEX_Y_INDEX():
    return X_INDEX, Y_INDEX

def get_end_X_INDEX_Y_INDEX():
    return X_END_INDEX, Y_END_INDEX

def get_WAY_INDEX():
    return WAY_INDEX

def get_MAX_DISTANCE():
    return MAX_DISTANCE

def get_NUM_OBSTACLES():
    return NUM_OBSTACLES

def get_UAV_COUNT():
    return UAV_COUNT
```

reports.py

```
file_path = "report.txt"

def set_path(path):
    file_path = path

def write_to_file(data):
    with open(file_path, "a", encoding="utf-8") as file:
```

```
        file.write(data)

    print("Log added")

def write_to_file_with_path(data, path):
    with open(path, "a", encoding="utf-8") as file:
        file.write(str(data))

    print("Log added")

def read_experimental_num():
    with open("property.txt", "r", encoding="utf-8") as file:
        content = file.read()
        try:
            number = int(content)
            print(f"Number: {number}")
        except ValueError:
            print("Not a number")

    return number
```

min_dist.py

```
import way_core
import draw

def process_min_way(canvas, current_x, current_y, obstacle):
    way = 1
    color_left = "black"
    color_right = "black"
    minx, miny, maxx,maxy = obstacle.bounds

    dist1 = way_core.calculate_distance(current_x, current_y, minx, miny)
    dist2 = way_core.calculate_distance(current_x, current_y, maxx, miny)

    if dist1 > dist2:
```

```
        way = 0
        color_left = "red"
        color_right = "green"
    elif dist1 <= dist2:
        color_left = "green"
        color_right = "red"
        way = 1

    draw.draw_point(canvas, minx,miny, 3, color_left, "points")
    draw.draw_point(canvas, maxx,miny, 3, color_right, "points")

    return way
```

graph_GUI.py

```
import networkx as nx
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
from storage import CANVAS_HEIGHT, CANVAS_WIDTH

def graph_init():
    G = nx.DiGraph()

    return G

def rename_node_in_graph(G, old_name, new_name):
    if G.has_node(old_name):
        attributes = G.nodes[old_name]

        G.remove_node(old_name)

        G.add_node(new_name, **attributes)

def add_node_to_graph(G, name, coord):
    x, y = coord
```

```
    existing_nodes = [n for n, attrs in G.nodes(data=True) if "pos" in attrs and
attrs["pos"] == (x, y)]

    if not G.has_node(name) and not existing_nodes:
        G.add_node(name, pos=(x, y))

def remove_node_from_graph(G, name):
    G.remove_node(name)

def add_edge_to_graph(G, name1, name2, w):
    value = w/50
    G.add_edge(name1, name2, weight=value)

def show_graph(G, highlighted_nodes):
    pos = nx.spring_layout(G)
    edge_labels = {(u, v): d['weight'] for u, v, d in G.edges(data=True)}

    node_colors = ['red' if node in highlighted_nodes else 'skyblue' for node in G.nodes()]

    for node, node_coords in nx.get_node_attributes(G, 'pos').items():
        if list(node_coords) == [500, 0] or list(node_coords) == [500, 1000]:
            node_colors[list(G.nodes()).index(node)] = 'green'

    sorted_edges = [(u, v) for u, v in G.edges() if u in highlighted_nodes and v in
highlighted_nodes]

    edge_directions = {(u, v): edge for u, v, edge in G.edges(data=True)}

    same_direction_edges = []
    for u, v in sorted_edges:
        if (u, v) in same_direction_edges:
            continue
        if (v, u) not in same_direction_edges:
            same_direction_edges.append((u, v))

    edge_colors = ['red' if (u, v) in same_direction_edges else 'gray' for u, v in G.edges()]
```

```
    plt.figure(figsize=(6, 6))
    nx.draw(G, pos, with_labels=True, node_size=500, node_color=node_colors,
font_size=16, font_color='black', edge_color=edge_colors,
connectionstyle="arc3,rad=0.2")
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=8)
    plt.show(block=False)
    print(matplotlib.get_backend())
    # plt.savefig("output_plot.png")

def highlight_nodes_by_coords(G, coords_list):
    highlighted_nodes = []
    for node, node_coords in nx.get_node_attributes(G, 'pos').items():
        if list(node_coords) in coords_list:
            highlighted_nodes.append(node)

    return highlighted_nodes
```

draw.py

```
import tkinter as tk
from tkinter import Button, W
from threading import Thread
import queue

import storage

colors = [
    "gray", "black", "blue", "red", "purple", "dark blue", "yellow", "dark red"
]

##########################################################################
#########
def create_window(name, width, height):
    window = tk.Tk()
    window.title(name)
```

```
    canvas = tk.Canvas(window, width=width, height=height, bg="white")
    canvas.pack()

    return window, canvas

def change_window_size(window, canvas, new_width, new_height):
    window.geometry(f"{new_width}x{new_height}")
    canvas.config(width=new_width, height=new_height)

def create_button(canvas, name, width, height, callback, button_state="normal"):
    button = Button(canvas, text=name, command=callback, state=button_state)
    canvas.create_window(width, height, window=button, width=110, height=30,
anchor='w')

    return button

def create_spinbox(canvas, min_value, max_value, width, height, callback):
    spinbox = tk.Spinbox(canvas, from_=min_value, to=max_value, increment=1)
    canvas.create_window(width, height, window=spinbox, anchor='w')

    spinbox.bind("<FocusOut>", lambda event: callback())

    return spinbox

def create_label(canvas, x, y, text):
    label = canvas.create_text(x, y, text=text, font=("Arial", 12))

    return label

def update_label(canvas, label, new_text):
    canvas.itemconfig(label, text=new_text)

def create_radio(canvas, name, width, height, var, init_val, callback):
    radio = tk.Radiobutton(canvas, text=name, variable=var, value=init_val,
command=callback)
    canvas.create_window(width, height, window=radio, anchor='w')
```

```
def create_checkbox(canvas, name, width, height, is_enabled, callback,
checkbox_state="normal"):
    frame = tk.Frame(canvas)  # Создаем рамку
    checkbox = tk.Checkbutton(frame, text=name, variable=is_enabled,
command=callback, state=checkbox_state)
    checkbox.pack()  # Размещаем Checkbutton внутри рамки
    canvas.create_window(width, height, window=frame, anchor='w')  # Создаем рамку
на холсте
##########################################################################
#########

def draw_point(canvas, x, y, size, color, tag):
    x1, y1 = x - size, y - size
    x2, y2 = x + size, y + size
    canvas.create_oval(x1, y1, x2, y2, fill=color, outline=color,  tags=tag)

def draw_graph_points(canvas, ID, graph_points):
    color = "purple"

    for i in range(len(graph_points) - 1):
        id, x, y = graph_points[i]
        if ID != id:
            continue

        draw_point(canvas, x,y, 3, color, "points")

def draw_trace_way(canvas, list_way, color, tag):

    for i in range(len(list_way) - 1):
        x1, y1 = list_way[i]
        x2, y2 = list_way[i+1]
        canvas.create_line(x1, y1, x2, y2, fill=color, width=5, tags=tag)

def create_dbg_line(canvas, x1, y1, x2, y2):
    draw_point(canvas, x1,y1, 3, "green", "DBG")
```

```
    draw_point(canvas, x2,y2, 3, "green", "DBG")
    canvas.create_line(x1, y1, x2, y2, fill="black", width=5, tags="trace")

def draw_trace(canvas, ID, list_trace, color):
    trace = []
    p = 0

    for item in list_trace:
        if item[0] == ID:
            trace.append(item)
    # canvas.delete("trace")
    for i in range(len(trace) - 1):
        p, x1, y1 = trace[i]
        p, x2, y2 = trace[i+1]
        canvas.create_line(x1, y1, x2, y2, fill=color, width=5, tags="trace")

def clear_robot(canvas, ID_UAV):
    canvas.delete("robot"+ str(ID_UAV) )

def draw_robot(canvas, coordinates, ID_UAV):
    X_INDEX, Y_INDEX = storage.get_X_INDEX_Y_INDEX()
    UAV_SIZE = storage.get_uav_size()

    clear_robot(canvas, ID_UAV)
    canvas.create_rectangle(coordinates[ID_UAV][X_INDEX] - UAV_SIZE,
                            coordinates[ID_UAV][Y_INDEX] - UAV_SIZE,
                            coordinates[ID_UAV][X_INDEX] + UAV_SIZE,
                            coordinates[ID_UAV][Y_INDEX] + UAV_SIZE,
                            fill="blue", tags="robot" + str(ID_UAV))

def draw_target(canvas, coordinates, ID_UAV):
    X_INDEX, Y_INDEX = storage.get_X_INDEX_Y_INDEX()
    X_END_INDEX, Y_END_INDEX = storage.get_end_X_INDEX_Y_INDEX()
    UAV_SIZE = storage.get_uav_size()

    canvas.delete("target")
```

```
    canvas.create_rectangle(coordinates[ID_UAV][X_END_INDEX] - UAV_SIZE,
coordinates[ID_UAV][Y_END_INDEX] - UAV_SIZE,
                            coordinates[ID_UAV][X_END_INDEX] + UAV_SIZE,
coordinates[ID_UAV][Y_END_INDEX] + UAV_SIZE, fill="red", tags="target")
    canvas.create_rectangle(coordinates[ID_UAV][X_INDEX] - UAV_SIZE,
                            coordinates[ID_UAV][Y_INDEX] - UAV_SIZE,
                            coordinates[ID_UAV][X_INDEX] + UAV_SIZE,
                            coordinates[ID_UAV][Y_INDEX] + UAV_SIZE,
                            fill="green", tags="start")

def draw_obstacles(canvas, obstacles):
    canvas.delete("obstacle")
    for obstacle in obstacles:
        x, y = obstacle.exterior.coords.xy
        canvas.create_polygon(*zip(x, y), fill='gray', outline=colors[1], tags="obstacle",
                              width=2)

def draw_grid(canvas, step):
    CANVAS_WIDTH, CANVAS_HEIGHT = storage.get_width_height()

    for x in range(0, CANVAS_WIDTH, step):
        canvas.create_line(x, 0, x, CANVAS_HEIGHT, fill="gray",  tags="grid")

    for y in range(0, CANVAS_HEIGHT, step):
        canvas.create_line(0, y, CANVAS_WIDTH, y, fill="gray",  tags="grid")

def clear_grid(canvas):
        clear_canvas(canvas, "grid")

def clear_trace(canvas):
    clear_canvas(canvas, "trace")

def clear_obstacles(canvas):
    clear_canvas(canvas, "obstacle")

def clear_graph_points(canvas):
```

```
    clear_canvas(canvas, "points")

def clear_all_canvas(canvas):
    clear_canvas(canvas, "robot")
    clear_canvas(canvas, "trace")
    clear_canvas(canvas, "points")
    clear_canvas(canvas, "drones")
    clear_canvas(canvas, "list_way")
    clear_canvas(canvas, "points")

    clear_canvas(canvas, "dijkstra")

def clear_canvas(canvas, tag):
    canvas.delete(tag)

def prepare_int_var(canvas, init):
    return tk.IntVar(canvas, init)

def prepare_boolean_var(canvas, init):
    return tk.BooleanVar(canvas, init)

def draw_drones(canvas, drones, tag):
    for drone in drones:
        x, y = drone
        # draw_drone(canvas, x, y)
        draw_point(canvas, x, y, 10, "dark green", tag)

def update_interface(canvas, queue):
    if queue.empty() == True:
        pass
    else:
        result = queue.get_nowait()
        ID, coordinates = result
        draw_robot(canvas, coordinates, ID)
    canvas.after(1, update_interface, canvas, queue)
```

Dijkstra.py

```
import math
import heapq
from shapely.geometry import Point, LineString, Polygon

import graph_GUI

import way_core
import storage
import reports

def is_segment_touching_or_inside_polygon(start_point, end_point,
polygon_coordinates):
    start = Point(start_point)
    end = Point(end_point)

    segment = LineString([start, end])

    polygon = Polygon(polygon_coordinates)

    if segment.covered_by(polygon):
        return False

    if segment.crosses(polygon):
        return False

    if segment.touches(polygon):
        return True

    return True

def check_intersection(start, end, rectangles):
        path = True

        for rectangle in rectangles:
```

```
            x_min, y_min, x_max, y_max = rectangle

            figure = Polygon([
                (x_min, y_min),
                (x_max, y_min),
                (x_max, y_max),
                (x_min, y_max)])
            path = is_segment_touching_or_inside_polygon(start, end, figure)
            if path == False:
                break

        return path

def calculate_distance(coord1, coord2):
    return math.sqrt((coord1[0] - coord2[0])**2 + (coord1[1] - coord2[1])**2)

def build_graph_to_way(G, coord_list, distance_len, obstacles):
    graph = {}

    distance_len = distance_len + 2

    for i, coord in enumerate(coord_list):
        graph[i] = {}
        print("Progress: " + str(i) + "/" + str(len(coord_list)))

        graph_GUI.add_node_to_graph(G, i, coord)

        for j, neighbor_coord in enumerate(coord_list):
            distance = calculate_distance(coord, neighbor_coord)
            if i != j:
                if check_intersection(coord, neighbor_coord, obstacles):
                    graph[i][j] = int(distance)
                    graph_GUI.add_node_to_graph(G, j, neighbor_coord)
                    graph_GUI.add_edge_to_graph(G, i, j, int(distance))

    return graph
```

```
def build_graph(G, coord_list, distance_len, obstacles):
    graph = {}

    distance_len = distance_len + 2

    for i, coord in enumerate(coord_list):
        graph[i] = {}
        print("Progress: " + str(i) + "/" + str(len(coord_list)))

        graph_GUI.add_node_to_graph(G, i, coord)

        for j, neighbor_coord in enumerate(coord_list):
            distance = calculate_distance(coord, neighbor_coord)
            if i != j and distance <= distance_len:
                if check_intersection(coord, neighbor_coord, obstacles):
                    graph[i][j] = int(distance)
                    graph_GUI.add_node_to_graph(G, j, neighbor_coord)
                    graph_GUI.add_edge_to_graph(G, i, j, int(distance))

    return graph

def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    priority_queue = [(0, start)]
    previous_nodes = {}

    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)

        if current_distance > distances[current_node]:
            continue

        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight
```

```
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous_nodes[neighbor] = current_node
                heapq.heappush(priority_queue, (distance, neighbor))

    return distances, previous_nodes

def shortest_path(previous_nodes, end):
    path = []
    while end is not None:
        path.insert(0, end)
        end = previous_nodes.get(end)
    return path

def start_dijkstra(init, graph_points, obstacles):
    coord = []
    obstacles_coordinates = []

    G = graph_GUI.graph_init()

    MAX_DISTANCE = storage.get_MAX_DISTANCE()
    X_INDEX, Y_INDEX = storage.get_X_INDEX_Y_INDEX()
    X_END_INDEX, Y_END_INDEX = storage.get_end_X_INDEX_Y_INDEX()

    coord = way_core.prepare_coordinates(graph_points, obstacles)

    obstacles_coordinates = way_core.prepare_obstacles(obstacles)

    graph = build_graph(G, coord, MAX_DISTANCE, obstacles_coordinates)

    reports.write_to_file_with_path(graph, "graph.txt")

    start_coord = (init[0][X_INDEX], init[0][Y_INDEX])
    end_coord = (init[0][X_END_INDEX], init[0][Y_END_INDEX])
```

```
    start_node = min(range(len(coord)), key=lambda i: calculate_distance(coord[i],
start_coord))
    end_node = min(range(len(coord)), key=lambda i: calculate_distance(coord[i],
end_coord))

    shortest_distances, previous_nodes = dijkstra(graph, start_node)
    path_nodes = shortest_path(previous_nodes, end_node)

    path_coords = [coord[node] for node in path_nodes]

    total_distance = shortest_distances[end_node]

    print("Way:", total_distance)
    print("path_coord:", path_coords)

    highlighted_nodes = graph_GUI.highlight_nodes_by_coords(G, path_coords)

    graph_GUI.show_graph(G, highlighted_nodes)

    nodes = graph_GUI.highlight_nodes_by_coords(G, path_coords)
    graph_GUI.show_graph(G, nodes)

    return total_distance, path_coords
```

algo.py

```
from shapely.geometry import Point, Polygon
import random
import json
import os.path
import copy
from threading import Thread
import queue
import time

import way_core
```

```
from way_core import RIGHT_WAY, LEFT_WAY, MIN_WAY, SPLIT

from storage import CANVAS_HEIGHT, CANVAS_WIDTH

import dijkstra
import a_star
import graph_GUI
import best_first

import min_dist

import draw
import storage
import reports

#UAVs
SET_UAV_COUNTER = 0
SET_UAV_LOCKER = 0

RANDOM = 2
LOAD = 1

GLOBAL_WAY = 0

GLOBAL_COLOR = "dark green"

experiment_num = 0
method_name = "RIGHT"

EXPERIMENT_1_MODE = 0
EXPERIMENT_2_MODE = 0

GLOBAL_INDEX = 0
EXPERIMENTAL_INDEX = 0
END = True
```

```
SPLIT_COUNTER = 0

START_TIME = 0
END_TIME = 0

#UAVs start coordinates
start_coordinates = [
   [CANVAS_WIDTH/2, 0, CANVAS_WIDTH/2, CANVAS_HEIGHT, 1]
]

current_coordinates = copy.deepcopy(start_coordinates)
init = copy.deepcopy(start_coordinates)

obstacles = []

total_distance = []

graph_points = []
list_traces = []
#1 metr = 50 points

With = 125
Height = 100
table_counter = 0

reached_target = []
state_prew = []

def generate_random_obstacles():
   global NUM_OBSTACLES, With, Height
   obstacles.clear()

   NUM_OBSTACLES = int(spinbox.get())
   With = int(spinbox_w.get) * 50
   Height = int(spinbox_h.get) * 50
```

```
    for _ in range(NUM_OBSTACLES):
        width = With
        height = Height
        x = random.randint(0, CANVAS_WIDTH - width)
        y = random.randint(0, CANVAS_HEIGHT - height)

        obstacle = Polygon([
            (x, y),
            (x + width, y),
            (x + width, y + height),
            (x, y + height)
        ])

        if any(obstacle.intersects(o) for o in obstacles):
            continue

        obstacles.append(obstacle)

    draw.draw_obstacles(canvas, obstacles)

def load_obstacles_with_index(index):
    if os.path.isfile("obstacles" + str(index) + ".json"):
        with open("obstacles" + str(index) + ".json", "r") as file:
            data = json.load(file)

    obstacles.clear()
    for obstacle_data in data:
        obstacle = Polygon(obstacle_data)
        minx, miny, maxx,maxy = obstacle.bounds
        if maxx < CANVAS_WIDTH:
            obstacles.append(obstacle)

def save_obstacles_to_file_with_index(obstacles, index):
    with open("obstacles" + str(index) + ".json", "w") as file:
        data = [list(obstacle.exterior.coords) for obstacle in obstacles]
        json.dump(data, file)
```

```
def generate_obstacles(With, Height):
    obstacles.clear()

    NUM_OBSTACLES = int(spinbox.get())
    With = float(spinbox_w.get()) * 50
    Height = float(spinbox_h.get()) * 50

    for _ in range(NUM_OBSTACLES):
        while True:
            width = With
            height = Height
            x = random.randint(0, CANVAS_WIDTH - width)
            y = random.randint(0, CANVAS_HEIGHT - height)

            obstacle = Polygon([
                (x, y),
                (x + width, y),
                (x + width, y + height),
                (x, y + height)
            ])

            if not any(obstacle.intersects(o) for o in obstacles):
                break

        obstacles.append(obstacle)

    draw.draw_obstacles(canvas, obstacles)

def load_obstacles_from_file(obstacles):
    if os.path.isfile("obstacles.json"):
        with open("obstacles.json", "r") as file:
            data = json.load(file)

        obstacles.clear()
        for obstacle_data in data:
```

```
            obstacle = Polygon(obstacle_data)
            obstacles.append(obstacle)

def save_obstacles_to_file(obstacles):
    with open("obstacles.json", "w") as file:
        data = [list(obstacle.exterior.coords) for obstacle in obstacles]
        json.dump(data, file)

def check_obstacle(new_x, new_y):
    for obstacle in obstacles:
        if obstacle.contains(Point(new_x, new_y)):
            return True

def get_obstacle(new_x, new_y):
    for obstacle in obstacles:
        if obstacle.contains(Point(new_x, new_y)):
            return obstacle

def set_point_obstacle(ID, x, y, graph_points, obstacle):
    if obstacle != None:
        minx, miny, maxx,maxy = obstacle.bounds
        if graph_points.count([ID, minx, miny]) == True:
            return
        graph_points.append([ID, minx, miny])
        graph_points.append([ID, minx, maxy])
        graph_points.append([ID, maxx, miny])
        graph_points.append([ID, maxx, maxy])

def set_point(ID, x,y, state_now, graph_points, obstacle, set_obstacle):
    global state_prew
    if not graph_points:
        graph_points.append([ID, x, y])
        return

    if set_obstacle == True:
        set_point_obstacle(ID, x, y, graph_points, obstacle)
```

```
    tmp, last_x, last_y = graph_points[-1]
    distance = way_core.calculate_distance(last_x, last_y, x, y)

    if distance >= MAX_DISTANCE:
        graph_points.append([ID, x, y])

    if state_now != state_prew[ID]:
        graph_points.append([ID, x, y])

    state_prew[ID] = state_now

def clear_trace():
    canvas.delete("trace")

def clear_graph_points():
    canvas.delete("graph_points")

def clear_trace():
    canvas.delete("trace")

def get_method_name(way):
    if way == LEFT_WAY:
        return "LEFT_WAY"
    if way == RIGHT_WAY:
        return "RIGHT_WAY"
    if way == SPLIT:
        return "CONTROLLED_WATERFLOW"

    return "Unexpected way"

def draw_table(canvas, rows, cols, cell_width, cell_height, data):
    for row in range(rows + 1):
        y = row * cell_height
        canvas.create_line(0, y, cols * cell_width, y)
```

```
    for col in range(cols + 1):
        x = col * cell_width
        canvas.create_line(x, 0, x, rows * cell_height)

    for row in range(rows):
        for col in range(cols):
            x = col * cell_width
            y = row * cell_height
            value = data[row][col]
            canvas.create_text(x + cell_width/2, y + cell_height/2, text=str(value))

def get_dx_dy_distance(ID, target_x, target_y):
    dx = target_x - start_coordinates[ID][X_INDEX]
    dy = target_y - start_coordinates[ID][Y_INDEX]
    dist = (dx ** 2 + dy ** 2) ** 0.5

    return dx, dy, dist

def get_dx_dy_distance_way(ID, x,y, target_x, target_y):
    dx = target_x - x
    dy = target_y - y
    dist = (dx ** 2 + dy ** 2) ** 0.5

    return dx, dy, dist

def get_new_x_y(ID, dx, dy, dist):
    new_x = start_coordinates[ID][X_INDEX] + dx / dist
    new_y = start_coordinates[ID][Y_INDEX] + dy / dist

    return new_x, new_y

def check_exists_point(list_trace, item):
    counter = 0
    for list in list_traces:
        for i in list:
            # if i[0] == item[0]:
```

```
            #     break

            if way_core.correction_coordinates(i[1], item[1],1) or
way_core.correction_coordinates(item[1],i[1],1):
                counter = counter + 1

            if counter == 3:
                return True
    # if list_trace.count(item) == True:
    #     return True

    return False

def process_control_waterfall(ID, x, y, new_x, new_y, target_x, target_y, way,
graph_point, obstacle):
    minx, miny, maxx,maxy = obstacle.bounds
    if (way_core.correction_coordinates(new_y, miny, 1) == True and \
        way_core.correction_coordinates(miny, new_y, 1) == True) and \
        way_core.correction_coordinates(maxx, new_x, 1) == False and \
        way_core.correction_coordinates(minx, new_x, 1) == False and \
        way_core.correction_coordinates(new_x, maxx, 1) == False and \
        way_core.correction_coordinates(new_x, minx, 1) == False:
        # if int(new_y) == int(miny) and int(minx) != int(new_x) and int(maxx) !=
int(new_x): #split in only A zone
            set_point(ID, new_x, new_y, True, graph_point, obstacle, True)
            if parent == False and  check_exists_point(list_trace, [ID, x, y]) == False: #split
ony one step
                parent = True
                new = RIGHT_WAY
                if way == LEFT_WAY:
                    new = RIGHT_WAY
                else:
                    new = LEFT_WAY
                start_coordinates.append([new_x, new_y, target_x, target_y, new])
                UAV_COUNT = UAV_COUNT + 1
                SPLIT_COUNTER = SPLIT_COUNTER + 1
```

```
            prepare_robot()
            start_robot(queue, UAV_COUNT - 1, target_x, target_y, not way,
graph_points[-1], list_traces[-1], parent)
    # else:
        # set_point_obstacle(ID, new_x, new_y, graph_point, obstacle)

#UAVs = [
#    [start_x, start_y, end_x, end_y, not_used]
#]

#end_points = [ID, [X,Y]]

end_points = []
counters_point = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

def move_robot_to_way(queue, ID, UAVs, end_points):
    global counters_point

    UAV = UAVs[ID]
    x = UAV[X_INDEX]
    y = UAV[Y_INDEX]
    target_x = UAV[X_END_INDEX]
    target_y = UAV[Y_END_INDEX]
    points = end_points[ID]
    end_coords = points[counters_point[ID]]

    if counters_point[ID] == len(end_coords):
        print("Finished")
        return

    end_coord = points[0]
    # for end_coord in points:
    #     target_x, target_y = end_coord

    dx, dy, dist = get_dx_dy_distance_way(ID, x, y, target_x, target_y)
```

```
    if dist > 1:
        new_x, new_y = get_new_x_y(ID, dx, dy, dist)

        UAV[X_INDEX] = new_x
        UAV[Y_INDEX] = new_y
        UAVs[ID] = UAV

    result = ID, UAVs
    queue.put(result)

    if dist > 1:
        canvas.after(7, move_robot_to_way, queue, ID, UAVs, end_points)
    else:
        print("Finish middle point, go next")
        counters_point[ID] = counters_point[ID] + 1

        canvas.after(7, move_robot_to_way, queue, ID, UAVs, end_points)


def move_robot(queue, ID, target_x, target_y, way, graph_point, list_trace, parent):
    global start_coordinates, reached_target, UAV_COUNT, graph_points, list_traces,
experiment_num, END, SPLIT_COUNTER, SET_UAV_COUNTER,
SET_UAV_LOCKER
    x = start_coordinates[ID][X_INDEX]
    y = start_coordinates[ID][Y_INDEX]

    if GLOBAL_WAY != SPLIT:
        way = GLOBAL_WAY
        start_coordinates[ID][WAY_INDEX] = GLOBAL_WAY

    dx, dy, dist = get_dx_dy_distance(ID, target_x, target_y)

    if dist > 1:
        new_x, new_y = get_new_x_y(ID, dx, dy, dist)

        if check_obstacle(new_x, new_y):
```

```
        obstacle=get_obstacle(new_x, new_y)
        minx, miny, maxx,maxy = obstacle.bounds
        if GLOBAL_WAY == LEFT_WAY or GLOBAL_WAY == RIGHT_WAY:
            set_point(ID, new_x, new_y, True, graph_point, obstacle, False)
        elif GLOBAL_WAY == MIN_WAY:
            way = min_dist.process_min_way(canvas, new_x, new_y,
obstacle=get_obstacle(new_x, new_y))
        elif GLOBAL_WAY == SPLIT:
            if (way_core.correction_coordinates(new_y, miny, 1) == True and \
                way_core.correction_coordinates(miny, new_y, 1) == True) and \
                way_core.correction_coordinates(maxx, new_x, 1) == False and \
                way_core.correction_coordinates(minx, new_x, 1) == False and \
                way_core.correction_coordinates(new_x, maxx, 1) == False and \
                way_core.correction_coordinates(new_x, minx, 1) == False:
                # if int(new_y) == int(miny) and int(minx) != int(new_x) and int(maxx) !=
int(new_x): #split in only A zone
                set_point(ID, new_x, new_y, True, graph_point, obstacle, True)
                if parent == False and  check_exists_point(list_trace, [ID, x, y]) == False:
#split ony one step
                    parent = True
                    new = RIGHT_WAY
                    if way == LEFT_WAY:
                        new = RIGHT_WAY
                    else:
                        new = LEFT_WAY
                    start_coordinates.append([new_x, new_y, target_x, target_y, new])
                    UAV_COUNT = UAV_COUNT + 1
                    SPLIT_COUNTER = SPLIT_COUNTER + 1
                    prepare_robot()
                    start_robot(queue, UAV_COUNT - 1, target_x, target_y, not way,
graph_points[-1], list_traces[-1], parent)
            # else:
                # set_point_obstacle(ID, new_x, new_y, graph_point, obstacle)
        dx,dy = way_core.coorect_avoid_obstacle(dx, dy, x,y, way, obstacle)

        new_x = start_coordinates[ID][X_INDEX] + dx / dist
```

```
            new_y = start_coordinates[ID][Y_INDEX] + dy / dist
        else:
            set_point(ID, new_x, new_y, False, graph_point, None, True)
            parent = False

        start_coordinates[ID][X_INDEX] = new_x
        start_coordinates[ID][Y_INDEX] = new_y

        list_trace.append([ID, start_coordinates[ID][X_INDEX],
start_coordinates[ID][Y_INDEX]])

    result = ID, start_coordinates
    queue.put(result)

    if dist > 1 and not reached_target[ID]:
        canvas.after(7 + UAV_COUNT, move_robot, queue, ID, target_x, target_y, way,
graph_point, list_trace, parent)
    else:
        UAV_COUNT = UAV_COUNT - 1
        graph_point.append([ID, target_x, target_y])
        reached_target[ID] = True
        print("Finish: ID = {}, Total = {}".format(ID, UAV_COUNT))
        draw.draw_trace(canvas, ID, list_trace, GLOBAL_COLOR)
        draw.draw_graph_points(canvas,ID, graph_point)
        draw.clear_robot(canvas, UAV_COUNT)
        distance = way_core.calculate_total_distance(list_trace)

        if GLOBAL_WAY == LEFT_WAY or GLOBAL_WAY == RIGHT_WAY:
            method_name = get_method_name(GLOBAL_WAY)
            experiment_data = "\nExperiment: " + str(GLOBAL_INDEX) + "\nmethod: " +
method_name + "\ndistance: " + str(distance / 50) + " Number UAV: " +
str(len(graph_point) - 2);
            print(experiment_data)
            reports.write_to_file(experiment_data)
            END = True
```

```
        if GLOBAL_WAY == SPLIT:
            if SPLIT_COUNTER <= 0:
                button_dijkstra()
                SPLIT_COUNTER = 0
            else:
                SPLIT_COUNTER = SPLIT_COUNTER - 1

        total_distance.append([start_coordinates[ID][WAY_INDEX], distance])

        # if SET_UAV_LOCKER == 1:
            # SET_UAV_COUNTER = SET_UAV_COUNTER + 1
            # button_set_UAV()

def prepare_robot():
    global graph_points, list_traces, current_coordinates

    reached_target.append(False)
    state_prew.append(False)

    graph_points.append([])
    list_traces.append([])
    current_coordinates.append([start_coordinates[-1][X_INDEX], start_coordinates[-
1][Y_INDEX],start_coordinates[-1][X_END_INDEX], start_coordinates[-
1][Y_END_INDEX], start_coordinates[-1][WAY_INDEX]])

    return list_traces[-1], graph_points[-1]

def start_robot(queue, ID, x_end, y_end, way, graph_point, list_trace, parent):
    thread = Thread(target=move_robot, args=(queue, ID, x_end, y_end, way,
graph_point, list_trace, parent))
    thread.start()
    # thread.join()

def set_start_robot(queue, ID, x_end, y_end, way, graph_point, list_trace, parent):
        move_robot(queue, ID, x_end, y_end, way, graph_point, list_trace, parent)
```

```
def start_uavs(queue):
    global graph_points, list_trace
    for i in range(UAV_COUNT):

        reached_target.append(False)
        state_prew.append(False)

        for c in range(UAV_COUNT):
            graph_points.append([])
            list_traces.append([])

        graph_points[i].append([i,500,0])
        start_robot(queue,i, start_coordinates[i][X_END_INDEX],
start_coordinates[i][Y_END_INDEX], start_coordinates[i][WAY_INDEX],
graph_points[i], list_traces[i], False)

def clear_uavs():
    global start_coordinates, current_coordinates, UAV_COUNT, list_traces,
EXPERIMENT_1_MODE
    UAV_COUNT = 1
    print("Clear ")
    start_coordinates = copy.deepcopy(init)
    current_coordinates = copy.deepcopy(start_coordinates)
    reached_target.clear()
    graph_points.clear()
    list_traces.clear()

    draw.clear_all_canvas(canvas)

    if LOAD != 1 and EXPERIMENT_1_MODE == 0:
        obstacles.clear()
        draw.clear_obstacles(canvas)
        generate_obstacles(With, Height)
        draw.draw_obstacles(canvas, obstacles)

def check_end_variable():
```

```
    global END, EXPERIMENT_1_MODE, EXPERIMENTAL_INDEX,
GLOBAL_INDEX
    while True:
        if END == True and (EXPERIMENT_1_MODE == 1 or
EXPERIMENT_2_MODE == 1):
            # draw.update_label(control, number_experiment_label,
str(EXPERIMENTAL_INDEX) + "/" + str(GLOBAL_INDEX))
            END = False
            button_start()
        time.sleep(2)

def button_start():
    global GLOBAL_INDEX, CANVAS_WIDTH, CANVAS_HEIGHT,
start_coordinates
    if EXPERIMENT_1_MODE == 1:
        if EXPERIMENTAL_INDEX - 1 <= GLOBAL_INDEX:
            print("Experimental session done")
            return

        clear_uavs()
        load_obstacles_with_index(GLOBAL_INDEX)
        draw.draw_obstacles(canvas, obstacles)
        GLOBAL_INDEX = GLOBAL_INDEX + 1
        print("Experiment: " + str(GLOBAL_INDEX) + "/" +
str(EXPERIMENTAL_INDEX))

    if EXPERIMENT_2_MODE == 1:
        if EXPERIMENTAL_INDEX - 1 <= GLOBAL_INDEX:
            print("Experimental session done")
            return

        CANVAS_WIDTH = CANVAS_WIDTH - GLOBAL_INDEX *10
        start_coordinates.clear()
        current_coordinates.clear()
        init.clear()
```

```
        start_coordinates.append([CANVAS_WIDTH/2, 0, CANVAS_WIDTH/2,
CANVAS_HEIGHT, 1])
        current_coordinates.append([CANVAS_WIDTH/2, 0, CANVAS_WIDTH/2,
CANVAS_HEIGHT, 1])
        init.append([CANVAS_WIDTH/2, 0, CANVAS_WIDTH/2, CANVAS_HEIGHT,
1])

        draw.draw_robot(canvas, start_coordinates, 0)
        draw.draw_target(canvas, start_coordinates, 0)

        clear_uavs()
        draw.clear_grid(canvas)
        load_obstacles_with_index(0)
        draw.draw_obstacles(canvas, obstacles)
        draw.change_window_size(window, canvas, CANVAS_WIDTH,
CANVAS_HEIGHT)
        draw.draw_grid(canvas, 50)
        GLOBAL_INDEX = GLOBAL_INDEX + 1
        print("Experiment: " + str(GLOBAL_INDEX) + "/" +
str(EXPERIMENTAL_INDEX))
    thread = Thread(target=start_uavs(queue_variable))
    thread.start()

def button_dijkstra():
    global END
    total_distance, path_coords = dijkstra.start_dijkstra(init, graph_points, obstacles)

    storage.set_dijkstra_coord(path_coords)

    print("distance:" + str(total_distance / 50))
    draw.draw_trace_way(canvas, path_coords, "red", "dijkstra")
    draw.draw_drones(canvas, path_coords, "drones")

    method_name = get_method_name(GLOBAL_WAY)
```

```
    experiment_data = "\nExperiment: " + str(GLOBAL_INDEX) + "\nmethod: " +
method_name + "\ndistance: " + str(total_distance / 50) + " Number UAV: " +
str(len(path_coords) - 2);
    print(experiment_data)
    reports.write_to_file(experiment_data)
    END = True

def button_a_star():
    total_distance, path_coords = a_star.start_a_star(init, graph_points, obstacles)

    storage.set_a_star_coord(path_coords)

    draw.draw_trace_way(canvas, path_coords, "blue", "a_star")
    draw.draw_drones(canvas, path_coords)

def button_set_UAV():
    global graph_points, list_traces, current_coordinates, UAV_COUNT,
GLOBAL_COLOR, SET_UAV_COUNTER, SET_UAV_LOCKER

    start_point = [500, 0]
    end_point = [500, 1000]
    UAV = [ [0, 0, 500, 500]]
    end_points = [ [[100, 100], [200, 200], [300, 300], [500, 500]]]
    move_robot_to_way(queue_variable, 0, UAV, end_points)
    SET_UAV_LOCKER = 1

    path_coord = storage.get_dijkstra_coord()
    parent = False
    count = 0

    UAV_COUNT = 0
    GLOBAL_COLOR = "purple"
    print("Prepare to position ")
    start_coordinates.clear()
    current_coordinates.clear()
    reached_target.clear()
```

```
    graph_points.clear()
    list_traces.clear()

    sorted_list = sorted(path_coord, reverse=True, key=lambda x: x[1])

    if start_point in sorted_list:
        sorted_list.remove(start_point)

    if end_point in sorted_list:
        sorted_list.remove(end_point)

    # # for coord in sorted_list:
    draw.clear_robot(canvas, UAV_COUNT)
    # target_x, target_y = sorted_list[SET_UAV_COUNTER] #radial way
    # target_x, target_y = sorted_list[3] #first way
    target_x, target_y = sorted_list[1] #middle way
    start_coordinates.append([0, 0, target_x, target_y, MIN_WAY])
    UAV_COUNT = UAV_COUNT + 1
    prepare_robot()
    set_start_robot(queue_variable, UAV_COUNT - 1, target_x, target_y, MIN_WAY,
graph_points[-1], list_traces[-1], parent)
    count = count + 1
    print(count)

def button_save_obstacles():
    save_obstacles_to_file(obstacles)

def button_obstacles_experiment():
    global start_coordinates, current_coordinates, UAV_COUNT, list_traces,
GLOBAL_INDEX, With, Height
    UAV_COUNT = 1
    print("Clear old data ")
    start_coordinates = copy.deepcopy(init)
    current_coordinates = copy.deepcopy(start_coordinates)
    reached_target.clear()
    graph_points.clear()
```

```
    list_traces.clear()

    draw.clear_all_canvas(canvas)

    obstacles.clear()
    draw.clear_obstacles(canvas)
    generate_obstacles(With, Height)
    save_obstacles_to_file_with_index(obstacles, GLOBAL_INDEX)
    draw.draw_obstacles(canvas, obstacles)

    GLOBAL_INDEX = GLOBAL_INDEX + 1

def button_clear():
    clear_uavs()

def method_callback():
    global GLOBAL_WAY
    GLOBAL_WAY = var.get()
    index = 0

def visible_callback():
    trace = trace_visible.get()
    index = 0

    if (trace == False):
        draw.clear_trace(canvas)
        index = 0
    if(trace == True):
        for list_trace in list_traces:
            draw.draw_trace(canvas, index, list_trace, GLOBAL_COLOR)
            index = index + 1

def clear_obstacles_callback():
    global LOAD, RANDOM
    obst_clear = obstacles_clear.get()
```

```
    if obst_clear == True:
        LOAD = 0
        RANDOM = 2
    else:
        LOAD = 1
        RANDOM = 2

def experiment_callback():
    global EXPERIMENT_1_MODE, EXPERIMENT_2_MODE
    exp_1_var = experiment_1_var.get()
    exp_2_var = experiment_2_var.get()

    if exp_1_var == True:
        EXPERIMENT_1_MODE = 1
    else:
        EXPERIMENT_1_MODE = 0

    if exp_2_var == True:
        EXPERIMENT_2_MODE = 1
    else:
        EXPERIMENT_2_MODE = 0

def visible_grid_callback():
    grid = grid_visible.get()

    if (grid == False):
        draw.clear_grid(canvas)
    if(grid == True):
        draw.draw_grid(canvas, 50)


def visible_points_callback():
    point = point_visible.get()
    index = 0

    if (point == False):
```

```
        draw.clear_graph_points(canvas)
        index = 0
    if(point == True):
        for point in graph_points:
            draw.draw_graph_points(canvas, index, point)
            index = index + 1

def on_spinbox_change():
    global NUM_OBSTACLES
    NUM_OBSTACLES = int(spinbox.get())

def on_spinbox_w_change():
    global With
    With = float(spinbox_w.get()) * 50

def on_spinbox_h_change():
    global Height
    Height = float(spinbox_h.get()) * 50

def visible_dijkstra_callback():
    dijkstra = dijkstra_visible.get()
    dijkstra_coord = storage.get_dijkstra_coord()

    if(dijkstra == False):
        draw.clear_canvas(canvas, "dijkstra")
    if (dijkstra == True):
        draw.draw_trace_way(canvas, dijkstra_coord, "red", "dijkstra")

def visible_a_star_callback():
    a_star = a_star_visible.get()
    a_star_coord = storage.get_a_star_coord()

    if(a_star == False):
        draw.clear_canvas(canvas, "a_star")
    if (a_star == True):
        draw.draw_trace_way(canvas, a_star_coord, "blue", "a_star")
```

```
def visible_best_first_callback():
    best_first = best_first_visible.get()
    best_first_coord = storage.get_best_first_coord()

    if(best_first == False):
        draw.clear_canvas(canvas, "best_first")
    if (best_first == True):
        draw.draw_trace_way(canvas, best_first_coord, "green", "best_first")

def create_buttons():
    #Control buttons
    buttons_x = 10
    draw.create_button(control, "Start", buttons_x, 10, button_start)
    draw.create_button(control, "Clear", buttons_x, 50, button_clear)
    draw.create_button(control, "Print table", buttons_x, 90, button_table, "disabled")
    draw.create_button(control, "Add to exp", 390, 190, button_obstacles_experiment)
    # draw.create_button(control, "Force build graph", 50, 300,
button_obstacles_experiment)

    #Algo buttons
    draw.create_button(control, "Dijkstra", buttons_x, 130, button_dijkstra)
    draw.create_button(control, "A_star", buttons_x, 170, button_a_star, "disabled")
    draw.create_button(control, "Save plan", buttons_x, 210, button_save_obstacles)

    draw.create_button(control, "set_UAV", buttons_x, 250, button_set_UAV)

def create_radios():
    draw.create_radio(control, "Right", 140, 10, var, 0, method_callback)
    draw.create_radio(control, "Left", 140, 30, var, 1, method_callback)
    draw.create_radio(control, "R + L", 140, 50, var, 2, method_callback)
    draw.create_radio(control, "CW", 140, 70, var, 3, method_callback)

def create_checkboxes():
    draw.create_checkbox(control, "Grid", 220, 10, grid_visible, visible_grid_callback)
    draw.create_checkbox(control, "Trace", 220, 35, trace_visible, visible_callback)
```

```
    draw.create_checkbox(control, "Points", 220, 55, point_visible,
visible_points_callback)
    draw.create_checkbox(control, "Dijkstra", 220, 75, dijkstra_visible,
visible_dijkstra_callback)
    draw.create_checkbox(control, "a_star", 220, 95, a_star_visible,
visible_a_star_callback, "disabled")
    draw.create_checkbox(control, "best_first", 220, 115, a_star_visible,
visible_best_first_callback, "disabled")
    draw.create_checkbox(control, "obstacles", 220, 145, obstacles_clear,
clear_obstacles_callback)
    draw.create_checkbox(control, "Set experiment 1", 220, 240, experiment_1_var,
experiment_callback)
    draw.create_checkbox(control, "Set experiment 2", 220, 260, experiment_2_var,
experiment_callback)

def create_spinboxes():
    spinbox = draw.create_spinbox(control, 0, 50, 320, 30, on_spinbox_change) #
TODO: Add MAX_NUM_OBSTACLES in storage
    spinbox_w = draw.create_spinbox(control, 0, CANVAS_WIDTH, 320, 70,
on_spinbox_w_change)
    spinbox_h = draw.create_spinbox(control, 0, CANVAS_HEIGHT, 320, 110,
on_spinbox_h_change)
    spinbox_d = draw.create_spinbox(control, 0, CANVAS_HEIGHT, 320, 160,
on_spinbox_h_change)

    return spinbox, spinbox_w, spinbox_h

def button_table():

    tmp, table = draw.create_window("Table", 350, 350)

    rows = 5
    cols = 3
    cell_width = 110
    cell_height = 50
    left_total = 0
```

```
    right_total = 0
    left_counter = 0
    right_counter = 0
    simulation = 0
    coef_m = 50

    h1 = 150
    w1 = 100
    h2 = 100
    w2 = 100
    h3 = 125
    w3 = 100

    if simulation == 1:
        for i in range(len(total_distance)):
            way, dist = total_distance[i]

            if way == 1:
                left_total = left_total + dist
                left_counter = left_counter + 1
            else:
                right_total = right_total + dist
                right_counter = right_counter + 1

        right_total = right_total / right_counter
        left_total = left_total / left_counter

    data = [
        ["L*h C", "Da_Left", "Da_Right"],
    ]

    data.append([str(h1/coef_m) + "x" + str(w1/coef_m) + " {" +
str(NUM_OBSTACLES) + "}",  1858/coef_m, 1943/coef_m])
    data.append([str(h2/coef_m) + "x" + str(w2/coef_m) + " {" +
str(NUM_OBSTACLES) + "}", 1390/coef_m, 1225/coef_m])
```

```
    data.append([str(h3/coef_m) + "x" + str(w3/coef_m) + " {" +
str(NUM_OBSTACLES) + "}", 1588/coef_m, 1644/coef_m])
    data.append(["S:" + str(CANVAS_WIDTH/coef_m*CANVAS_HEIGHT/coef_m) ,
"WIDTH:" + str(CANVAS_WIDTH/coef_m), "HEIGHT:" +
str(CANVAS_HEIGHT/coef_m)])

    draw_table(table, rows, cols, cell_width, cell_height, data)
CANVAS_WIDTH, CANVAS_HEIGHT = storage.get_width_height()
UAV_SIZE = storage.get_uav_size()
X_INDEX, Y_INDEX = storage.get_X_INDEX_Y_INDEX()
X_END_INDEX, Y_END_INDEX = storage.get_end_X_INDEX_Y_INDEX()
WAY_INDEX = storage.get_WAY_INDEX()
MAX_DISTANCE = storage.get_MAX_DISTANCE()
NUM_OBSTACLES = storage.get_NUM_OBSTACLES()
UAV_COUNT = storage.get_UAV_COUNT()

window, canvas = draw.create_window("Way simulation", CANVAS_WIDTH,
CANVAS_HEIGHT)
tmp, control = draw.create_window("Control", 500, 300)

# storage.set_debug_value(canvas)

if LOAD == 1:
    load_obstacles_from_file(obstacles)
elif not obstacles and RANDOM == 1 and LOAD == 0:
    generate_random_obstacles()
else:
    generate_obstacles(With,Height)

draw.draw_obstacles(canvas, obstacles)

draw.draw_grid(canvas, 50)

for i in range(UAV_COUNT):
    draw.draw_robot(canvas, start_coordinates, i)
    draw.draw_target(canvas, start_coordinates, i)
```

```
create_buttons()

var = draw.prepare_int_var(control, 0)
grid_visible = draw.prepare_boolean_var(control, True)
trace_visible = draw.prepare_boolean_var(control, True)
point_visible = draw.prepare_boolean_var(control, True)
dijkstra_visible = draw.prepare_boolean_var(control, True)
a_star_visible = draw.prepare_boolean_var(control, True)
best_first_visible = draw.prepare_boolean_var(control, True)

obstacles_clear = draw.prepare_boolean_var(control, True)
experiment_1_var = draw.prepare_boolean_var(control, False)
experiment_2_var = draw.prepare_boolean_var(control, False)

create_radios()
create_checkboxes()

spinbox, spinbox_w, spinbox_h = create_spinboxes()

draw.create_label(control, 370,10, "Obstacle n:")
draw.create_label(control, 370,50, "width: m")
draw.create_label(control, 370,90, "height: m")
draw.create_label(control, 410,140, "dist between UAVs: m")

draw.create_label(control, 320, 190, "Total experiments:")
number_experiment_label = draw.create_label(control, 320, 210, "0/0")

queue_variable = queue.Queue()

draw.update_interface(canvas, queue_variable)

save_obstacles_to_file(obstacles)

EXPERIMENTAL_INDEX = reports.read_experimental_num()
```

```
draw.update_label(control, number_experiment_label, str(EXPERIMENTAL_INDEX))

thread = Thread(target=check_end_variable)
thread.daemon = True
thread.start()

window.mainloop()
#-------------------------------------------------------------------------------------------/
#-------------------------------------------------------------------------------------------/
```

Б.2 Лістинг коду програмного засобу “Reliability Level”

c_project.c

```
#include <stdio.h>
#include <math.h>

#include "gnuplot_i.h"

#define INITIAL_RELIABILITY  1
#define CRITICAL_RELIABILITY 0.98875
#define LAMBDA               0.0002

#define EXPECTED_WORKS_H     3

#define DEGRADATE            0.00001

#define MINUTES_TO_HOURS(minutes) ((minutes)/60U)

#define TIME_TO_SET_UAV_H  MINUTES_TO_HOURS(2.0)
#define TIME_TO_SET_PROP_H MINUTES_TO_HOURS(3.0)
#define TIME_TO_SWITCH_H   MINUTES_TO_HOURS(1.0)

#define UAV_MAX_FLY_TIME   MINUTES_TO_HOURS(40.0)

typedef enum {
```

```
    ALL_RESTORE = 0,
    PART_RESTORE = 1,
    NO_RESTORE = 2
} task_mode;

typedef enum {
    FIRST_SYSTEM = 0,
    SECOND_SYSTEM = 1,
    ALL_SYSTEMS = 2
} systems_print;

typedef struct data_systems {
    double value;
    int color;
} data_systems_st_t;

#define MODE        NO_RESTORE
#define SYSTEM_PRINT SECOND_SYSTEM

double reliabilityAtTime(double lambda, double initialReliability, double t, int n) {
    return initialReliability * exp(-lambda * t * n);
}

double timeToCriticalReliability(double lambda, double initialReliability, double
criticalReliability, int n) {

    return log(initialReliability / criticalReliability) / (lambda * n);
}

data_systems_st_t first_system[1024] = {0};
data_systems_st_t second_system[1024] = {0};

int main() {
    double lambda = LAMBDA; //intensive
    double critical_reliability = CRITICAL_RELIABILITY;
    double initial_reliability = INITIAL_RELIABILITY;
```

```
double degradate_coef = DEGRADATE;
double time_to_set_UAV = TIME_TO_SET_UAV_H;
double time_to_set_property = TIME_TO_SET_PROP_H;
double time_to_switch_UAV = TIME_TO_SWITCH_H;
double uav_time_fly = UAV_MAX_FLY_TIME;
double total_work = EXPECTED_WORKS_H;

task_mode switch_mode = MODE;
systems_print systems = SYSTEM_PRINT;

double effective_work = 0;
double timeToCritical = -1;
double timeToResetUAV = -1;
double timeToResetUAV_min = 0;
double ctn = 0;
int n = 0;

first_system[0].value = initial_reliability;
second_system[0].value = initial_reliability;

gnuplot_ctrl *g = gnuplot_init();
if (g == NULL) {
        printf("Failed to initialize gnuplot\n");
    return 1;
}



printf("********************\n");
printf("lambda %lf\n", lambda);
printf("critical reliability %lf\n", critical_reliability);
printf("time to set UAV %lfh\n", time_to_set_UAV);
printf("time to set property %lfh\n", time_to_set_property);
printf("time to switch UAV %lfh\n", time_to_switch_UAV);
printf("UAV fly time %lfh\n", uav_time_fly);
printf("********************\n");
```

```
    printf("Input count UAV in the system (n): ");
    scanf("%d", &n);

    FILE *gnuplotPipe = popen("gnuplot -persistent", "w");

    double total_work_min = 3 * 60;
    fprintf(gnuplotPipe, "set yrange [0.996:1]\n");
    fprintf(gnuplotPipe, "set xrange [0:%lf]\n", total_work_min + 40);
    if (systems == ALL_SYSTEMS) {
        fprintf(gnuplotPipe, "plot '-' with points lt 1 lw 1 lc variable\n");
    } else {
        fprintf(gnuplotPipe, "plot '-' with lines lt 3 lw 3 lc variable\n");
    }

    int effective_counter = 0;
    int non_effective_counter = 0;
    double reliability_value = 0;
    double reliability_value_for_1_system = 0;
    int iter = 0;
    int global_minute = 0;
    while (effective_work < total_work) {
            printf("initial reliability: %lf\n", initial_reliability);
            iter++;
            timeToResetUAV = time_to_set_UAV + time_to_set_property +
time_to_switch_UAV;
            timeToResetUAV_min = timeToResetUAV * 60;

            timeToCritical = timeToCriticalReliability(lambda, initial_reliability,
critical_reliability, n);

            if (uav_time_fly > 0) {

                if (timeToCritical > uav_time_fly) {
                    printf("Time to critical set to UAV time fly\n");
                    timeToCritical = uav_time_fly;
```

```
            }
        }

        double effective_work_min = (timeToCritical - timeToResetUAV) * 60;
        double total_work_to_1_UAVs_min = effective_work_min +
timeToResetUAV_min;
        double last_time_to_swap = total_work_to_1_UAVs_min -
effective_work_min;
        printf("effective work min %lf\n", effective_work_min);

        effective_work = effective_work + timeToCritical - timeToResetUAV;
        printf("Effective work: %lf\n", effective_work);
        printf("Total work: %lf\n", total_work);
        printf("Iteration: %d\n", iter);

        for (double j = ctn; j <= total_work_to_1_UAVs_min; j++) {
            if (ctn != 0) {
                ctn = 0;
            }
            double hour = MINUTES_TO_HOURS(j);

            reliability_value = reliabilityAtTime(lambda, initial_reliability, hour, n);
            if (reliability_value <= critical_reliability) {
                printf("Alarm!!!!\n");
            }

            double tmp = total_work_to_1_UAVs_min - j;
            int color = ((j <= timeToResetUAV_min) || (tmp <= timeToResetUAV_min))
? 7 : 2;

            if ((iter - 1) % 2 == 0) {
                first_system[global_minute].value = reliability_value;

                first_system[global_minute].color = color;
            } else {
                second_system[global_minute].value = reliability_value;
```

```
            second_system[global_minute].color = color;
        }
        global_minute++;
    }

        if (systems != ALL_SYSTEMS) {
            reliability_value_for_1_system = reliability_value;
        }

    if ((iter - 1) % 2 != 0 && switch_mode == NO_RESTORE) {
        initial_reliability = reliability_value;
    } else if ((iter - 1) % 2 != 0 && switch_mode == PART_RESTORE) {
        initial_reliability = initial_reliability - degradate_coef;
    }

    if (effective_work * 60 >= total_work_min) {
        continue;
    }

        global_minute = global_minute - timeToResetUAV_min * 2;

        for (double k = 0; k < timeToResetUAV_min; k++) {
            double reliability_value = 0;
            double hour = MINUTES_TO_HOURS(k);
            reliability_value = reliabilityAtTime(lambda, initial_reliability, hour, n);

        if ((iter - 1) % 2 != 0) {
            first_system[global_minute].value = reliability_value;
            first_system[global_minute].color = 7;
        } else {
            second_system[global_minute].value = reliability_value;
            second_system[global_minute].color = 7;
        }
```

```
                // fprintf(gnuplotPipe, "%d %.12f %d\n", global_minute, reliability_value,
7);
                ctn++;
                global_minute++;
            }
    }

    for (int i = 0; i < global_minute; i++) {

        if (systems == FIRST_SYSTEM) {
            if (first_system[i].value == 0) {
                first_system[i].value = first_system[i - 1].value;
                first_system[i].color = 3;
        }

        fprintf(gnuplotPipe, "%d %.12f %d\n", i, first_system[i].value,
first_system[i].color);
        } else if (systems == SECOND_SYSTEM) {
            if (second_system[i].value == 0) {
                second_system[i] = second_system[i - 1];
                second_system[i].color = 3;
        }

        fprintf(gnuplotPipe, "%d %.12f %d\n", i, second_system[i].value,
second_system[i].color);
        } else {
            int counter = i;
            while (first_system[counter].value != 0) {
                fprintf(gnuplotPipe, "%d %.12f %d\n", counter, first_system[counter].value,
first_system[counter].color);
                counter++;
            }

            counter = i;
            while (second_system[counter].value != 0) {
```

```
            fprintf(gnuplotPipe, "%d %.12f %d\n", counter,
second_system[counter].value, second_system[counter].color);
            counter++;
        }
    }

}
fprintf(gnuplotPipe, "e\n");
fclose(gnuplotPipe);

pause();

return 0;
}
```