НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Програмні засоби для автоматизованої побудови маршрутів БПЛА в аграрній сфері»

Виконав:
студент IV курсу, групи КП-52
Чеботарьов Кирило Сергійович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Т.М. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Рецензент:
Доцент кафедри ММСА, к.т.н., доцент,
Дідковська М.В. _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – 6.050103 «Програмна інженерія»

«ЗАТВЕРДЖУЮ»

Науковий керівник кафедри

_____ І.А. Дичка

«___»_____2018 р.

ЗАВДАННЯ
на дипломний проект студенту
Чеботарьову Кирилу Сергійовичу

1. **Тема проекту** «Програмні засоби для автоматизованої побудови маршрутів БПЛА в аграрній сфері» затверджена наказом по університету № 1331-С від «22» травня 2019 р.
2. **Термін подання** студентом завершеного проекту: «7» червня 2019 р.
3. **Вихідні дані для дипломного проектування:** див. Технічне завдання.
4. **Перелік задач, які мають бути вирішені:**
   – розробити загальну структуру програмного забезпечення;
   – розробити алгоритми роботи програми;
   – розробити графічного інтерфейсу;
   – виконати програмну реалізацію проекту відповідно до вимог технічного завдання;
   – виконати тестування програмного проекту.
5. **Перелік обов'язкового ілюстративного матеріалу:**
   – блок-схема алгоритму побудови маршруту БПЛА (креслення);
   – структура взаємодії представлень інтерфейсу користувача (креслення);
   – архітектура додатку для побудови маршрутів БПЛА (плакат);
   – діаграма варіантів використання (плакат).
6. **Консультанти розділів проекту**

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. **Дата видачі завдання: «**31» жовтня 2018 р.

КАЛЕНДАРНИЙ ПЛАН-ГРАФІК

№ з/п	Назва етапів роботи та питань, які мають бути розроблені відповідно до завдання	Термін виконання	Примітка
1.	Вивчення літератури за тематикою проекту	15.10.2018	
2.	Розроблення та узгодження технічного завдання	19.11.2018	
3.	Розроблення структури програмного забезпечення	03.12.2018	
4.	Підготовка матеріалів першого розділу дипломного проекту	17.12.2018	
5.	Розроблення дизайну системи та графічних елементів	01.02.2019	
6.	Підготовка матеріалів другого розділу дипломного проекту	25.02.2019	
7.	Програмна реалізація дипломного проекту	15.03.2019	
8.	Тестування програмного забезпечення	19.04.2019	
9.	Підготовка матеріалів третього розділу дипломного проекту	03.05.2019	
10.	Підготовка матеріалів четвертого розділу дипломного проекту	15.05.2019	
11.	Оформлення документації дипломного проекту	03.06.2019	

Керівник дипломного проекту _____ Т.М.Заболотня

Студент _____ К.С. Чеботарьов

АНОТАЦІЯ

Даний дипломний проект присвячений створенню програмних засобів для розв’язання актуальної виробничої задачі в аграрній сфері, а саме оптимізації довготривалого та монотонного процесу побудови маршруту для БПЛА. Тема роботи обумовлена необхідністю пришвидшення та автоматизації цього процесу, для запобігання можливих збитків при невчасному виконанні запланованих робіт.

Розроблені програмні засоби являють собою кросплатформенний десктоп-додаток, що має можливість комунікації з автопілотом БПЛА напряму через дротове підключення за протоколом MAVLink. Функціональність додатку забезпечує можливість побудови маршруту для БПЛА в заданій області з урахуванням відносної висоти рельєфу та природних перешкод за круговим способом обходу по спіралі зі зменшенням радіусу кожного наступного кола.

У даному дипломному проекті розроблено: архітектуру десктоп-додатку, алгоритм побудови маршруту, процедуру зчитування та записування маршруту в автопілот БПЛА, а також графічні елементи та дизайн дестоп-дотаку.

ABSTRACT

This diploma project is devoted to the creation of software tools for solving the actual production problem in the agrarian sector, namely optimization of the long-term and monotonous process of constructing a route for UAV. The theme of the work is due to the need to accelerate and automate this process, in order to prevent possible losses in the not timely execution of the planned work.

The developed software is a cross-platform desktop application that has the ability to communicate with UAV autopilot directly through the MAVLink wired connection. The functionality of the application provides the possibility of constructing a mission for the UAV in a given area, taking into account the relative elevation of the relief and natural obstacles in a circular way bypassing the spiral with a decrease in the radius of each next circle.

During this graduation project has developed: the architecture of the desktop application, the algorithm for constructing the route, the procedure for reading and writing the route in the UAV autopilot, as well as graphic elements and the design of the dashboard.

ДП.045490-01-90 Програмні засоби для автоматизованої побудови маршрутів БПЛА в аграрній сфері. Відомість проекту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проекту		
ДП.045490-02-91	Програмні засоби для	4	
	автоматизованої побудови		
	маршрутів БПЛА		
	в аграрній сфері.		
	Технічне завдання		
ДП.045490-03-81	Програмні засоби для	58	
	автоматизованої побудови		
	маршрутів БПЛА		
	в аграрній сфері.		
	Пояснювальна записка		
ДП.045490-04-51	Програмні засоби для	4	
	автоматизованої побудови		
	маршрутів БПЛА		
	в аграрній сфері.		
	Програма та методика		
	тестування		
ДП.045490-05-34	Програмні засоби для	6	
	автоматизованої побудови		
	маршрутів БПЛА		
	в аграрній сфері.		
	Керівництво користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045490-06-99	Програмні засоби для	1	
	автоматизованої побудови		
	маршрутів БПЛА		
	в аграрній сфері.		
	Побудова маршруту.		
	Блок-схема алгоритму		
ДП.045490-07-99	Програмні засоби для	1	
	автоматизованої побудови		
	маршрутів БПЛА		
	в аграрній сфері.		
	Інтерфейс користувача.		
	Структура взаємодії		
	представлень		
ДП.045490-08-98	Програмні засоби для	1	
	автоматизованої побудови		
	маршрутів БПЛА		
	в аграрній сфері.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

ПРОГРАМНІ ЗАСОБИ ДЛЯ АВТОМАТИЗОВАНОЇ ПОБУДОВИ МАРШРУТІВ БПЛА В АГРАРНІЙ СФЕРІ

Технічне завдання

ДП.045490-02-91

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ К.С. Чеботарьов

2018

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмні засоби для автоматизованої побудови маршрутів БПЛА в аграрній сфері.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання в якості засобу для автоматизації побудови маршрутів для БПЛА з метою економії часу операторами БПЛА.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Десктоп-додаток повинен забезпечувати такі основні функції:

1. можливість автоматично будувати маршрут для БПЛА в заданій опуклій області за заданими параметрами;
2. можливість автоматично закруглювати гострі кути маршруту. Величина кута, який вважається гострим, для певної моделі БПЛА має задаватись згідно до радіусу повороту БПЛА;
3. можливість побудови маршруту без доступу до мережі Інтернет;
4. можливість графічно переглянути перепади висоти в заданій області з/без можливості доступу до мережі Інтернет;

5. можливість швидко знайти своє місцезнаходження в редакторі на мапі;
6. можливість використання додатку на оперційних системах Windows, MacOS та Ubuntu;
7. можливість змінювати провайдер зображень географічної мапи;
8. можливість записувати маршрут в автопілот БПЛА;
9. можливість зчитувати поточний маршрут з автопілоту БПЛА;
10. можливість знаходження відносної висоти точки за вказаними географічними координатами без доступу до Інтернету.

Розробку виконати на платформі Electron.js.

Додаткові вимоги:

1. наявність адаптивного інтерфейсу;
2. наявність анімованих кнопок;
3. наявність “push” повідомлень;
4. дизайн інтерфейсу за вимогами Google Material Design.

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;

2) програма та методика тестування;

3) керівництво користувача;

4) креслення:

- «Побудова маршруту. Блок-схема алгоритму»;
- «Інтерфейс користувача. Структура взаємодії представлень».

6. ЕТАПИ ПРОЕКТУВАННЯ

Вивчення літератури за тематикою роботи.....14.11.2018

Розроблення та узгодження технічного завдання.....28.11.2018

Розроблення структури десктоп-додатку 15.12.2018

Розроблення дизайну сторінок та графічних елементів.....03.02.2019

Програмна реалізація десктоп-додатку 17.03.2019

Тестування десктоп-додатку.....03.04.2019

Підготовка матеріалів текстової частини проекту 28.04.2019

Підготовка матеріалів графічної частини проекту 12.05.2019

Оформлення технічної документації проекту.....25.05.2019

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

ПРОГРАМНІ ЗАСОБИ ДЛЯ АВТОМАТИЗОВАНОЇ ПОБУДОВИ МАРШРУТІВ БПЛА В АГРАРНІЙ СФЕРІ

Пояснювальна записка

ДП.045490-03-81

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ К.С. Чеботарьов

2019

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Автопілот БПЛА – інтегрована система, яка на борту БПЛА виконує такі функції:

- автоматичне керування по заданій траєкторії;
- стабілізація кутів орієнтації;
- визначення навігаційних параметрів;

БПЛА – безпілотний літальний апарат. На відміну від дистанційно керованої моделі не потребує фізичної присутності пілота на його борту, адже включає в себе автопілот;

ГІС-технології – технології отримання, оброблення, зберігання і розповсюдження інформації, які діють на засадах взаємозв'язку семантичних даних про об'єкти з їх просторовими характеристиками;

ЛЕП – лінія електропередачі;

Маршрут (місія) для БПЛА – сукупність команд для автопілоту БПЛА, при виконанні яких БПЛА здійснить політ по певній послідовності точок в просторі;

Оператор БПЛА – спеціаліст, який керує сучасними БПЛА дистанційно; Одними з головних обов’язків оператора БПЛА є:

- збірка БПЛА;
- підготовка БПЛА;
- побудова маршрутів для БПЛА;

ОС – операційна система;

Пай – частка, що вноситься учасником до спільної справи;

ПЗ – програмне забезпечення;

Термальна колона – висхідний потік повітря в атмосфері Землі, викликаний нерівномірним нагріванням приземного шару повітря під впливом сонячних променів [1];

Точне землеробство – концепт впровадження технологій у рільництво на основі ґрунтових картографічних одиниць, використання точних

дистанційних даних – знімків супутника чи дрона, використання технологій для оброблення цих даних [2];

Трихограма – корисна комаха-паразит, яка знищує шкідників технічних культур на стадії яйця та широко використовується в аграрній індустрії [3];

API – Application Programming Interface, набір визначень, протоколів та засобів для взаємодії з певним ПЗ;

GDEM, Global Digital Elevation Map – база даних що містить в собі інформацію щодо висоти рельєфу за певними географічними координатами;

GPS – Global Positioning System;

MAVLink – Micro Air Vehicle Link, протокол для комунікації БПЛА із наземною станцією [4];

MVC – Model-View-Controller;

SPA – Single Page Application, односторінковий інтерфейс;

PWA – Progressive Web App, веб-застосунок, який є гібридом звичайної веб-сторінки та мобільного додатку; створюється за допомогою можливостей, що надають сучасні веб-браузери, але при цьому його використання нагадує використання мобільного додатку [5]. Характеризується можливістю використання кешувати дані обсягом до 50 мб;

USB – Universal Serial Bus;

WGS – World Geodetic System, стандартизована система координат.

ВСТУП

Комерційне використання БПЛА є досить актуальною темою в наш час. Постійно з'являються нові способи їх застосування: від секретних військових операцій до рятувальних місій та точного землеробства. Сфери застосування БПЛА досить різноманітні, проте кожен оператор натрапляє на труднощі, що пов'язані з однаковими проблемами. Відмінність БПЛА від звичайної літаючої авіамоделі полягає в тому, що перший – повністю автоматизований. Тому напередодні кожного запуску перед оператором БПЛА постає задача проектування маршрутів польоту, що є доволі кропіткою та витратною (у часовому вимірі) процедурою. Потрібно розуміти, що побудова маршруту займає як мінімум втричі більше часу, ніж сам політ БПЛА. Чим більший маршрут, тим більше монотонних операцій слід виконувати оператору БПЛА над кожною точкою маршруту. Відповідно зростає можливість появи небажаних помилок та неточностей.

Побудова маршрутів для БПЛА в аграрній сфері різниться тим, що політ апарату здійснюється на ділянках зі змінним рельєфом на досить низькій висоті (до 15 м) та високій швидкості (до 100 км/год), порівнюючи з іншими галузями застосування (аерофотозйомка – до 100 м при 65 км/год). Ця відмінність вимагає від оператора БПЛА значніших затрат часу, більших умінь та вищої відповідальності при побудові маршруту.

При виконанні робіт із захисту рослин із використанням БПЛА, оператор повинен дотримуватися чітко визначених за договором на виконання робіт умов навколишнього середовища, таких як: швидкість вітру, температура та вологість повітря, час доби та інші. Зазвичай, у період сезону такі умови трапляються впродовж декількох годин на тиждень. Якщо сприятливий для вильоту час буде витрачено на побудову маршруту, то обидві сторони договору понесуть збитки.

Існує безліч наземних станцій та інших застосунків для роботи з автопілотом БПЛА, однією з функцій яких є зазначення та збереження

географічних координат точок маршруту в пам’яті автопілоту БПЛА. Лідерами серед них є: Mission Planer [6], QGroundControl [7] та DroidPlanner [8]. Наведені ПЗ мають здебільшого однакову функціональність, проте на сьогоднішній день жоден з них не передбачає таких можливостей:

- автоматичної побудови маршруту з витримкою постійної відносної висоти в кожній точці маршруту;
- автоматичної побудови маршруту у формі спіралі у вказаній області;
- автоматичної побудови маршруту з урахуванням перешкод;
- кешування мапи від різних провайдерів мап.

З огляду на недосконалості та непристосованість існуючих способів вирішення даних проблем вимогам сучасного аграрного бізнесу, метою даного дипломного проекту є створення першого в своєму роді ПЗ, яке дозволить автоматизувати та відчутно пришвидшити побудову маршруту для БПЛА.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Аналіз особливостей використання БПЛА в аграрній сфері

Нині аграрне виробництво – це високотехнологічна галузь, де використовуються найсучасніші досягнення науки й техніки, включаючи дрони [9]. ГІС-технології, дистанційний моніторинг посівів, точне землеробство, прогнозування врожаїв – це реалії сучасного сільського господарства. Наприклад, у США, за оцінками Міжнародної асоціації безпілотних систем (AUVSI), 80% цивільних дронів застосовується саме в сільськогосподарській галузі [10]. В Україні, за оцінкою експертів компанії Drone.UA [11], для великої вибірки полів сої та кукурудзи від використання БПЛА за рік додатковий прибуток становить близько 25–30 дол. США за 1 га, тоді як витрати на застосування для замовників агромоніторінгу були близько 2,5 дол. за 1 га на рік. При цьому розвиток ринку так званих «малих БПЛА» вагою до 25 кг на сьогодні є одним із найбільш зростаючих. За прогнозами ABI Research, до 2025 року він перевищить 30 млрд дол., із них 70% припадатиме на комерційний сектор і, в першу чергу, на сільське господарство [12].

Очевидно, що аграрний сектор нині є основним споживачем послуг БПЛА [9]. Також він вимагає найбільших випробувань від оператора БПЛА при побудові маршрутів. У наш час кожна взаємодія з БПЛА вимагає від оператора ручної роботи для планування маршруту. Дивно, що досі на якість виконаних робіт впливає людський фактор, що зовсім не характерно для точного землеробства. У середньому побудова маршруту для зони площею 100га займає від 45хв до 1,5 години, в той час, як дрон пролітає за цим маршрутом за 30хв. За добу одна команда оброблює площу 700-1000га. Якщо автоматизувати процес побудови маршруту, то можна збільшити продуктивність команди як мінімум вдвічі або спростити процес побудови маршруту.

Основні процеси, які виконуються в аграрній сфері за участі дронів, не обмежуються цим списком [9]:

- внесення різних видів ЗЗР а також отрутохімікатів, зокрема розселення трихограми;
- спостереження за станом рослин на різних етапах їх розвитку: починаючи від контролю за сходами і закінчуючи оцінкою стану озимих на початку весняного відновлення вегетації. Це включає оцінку забур'яненості, оцінку розвитку рослин у різні фенологічні фази, моніторинг захворюваності та пошкодження шкідниками, оцінку забезпеченості рослин елементами живлення, насамперед азотом, контроль щодо потреби рослин у зрошенні, перевірку дозрівання та якості збирання врожаю;
- контроль якості виконання технологічних операцій – обробітку ґрунту, сівби, зрошення, внесення добрив тощо;
- спостереження за станом ґрунту – дані високоточних зйомок використовують для побудови ґрунтових карт, оцінки неоднорідності ґрунтового покриву за рівнем гумусованості та забезпеченості елементами живлення, для моніторингу ґрунтових деградаційних процесів, що активно розвиваються (ерозія, підтоплення, осолонцювання, засолення).

Кожна з наведених вище операції потребує побудови маршруту обльоту оброблюваної території згідно з певною технологією. Ця процедура може бути повністю автоматизованою.

Кожен тип виконуваних робіт в аграрній промисловості має свою технологію виконання, якої оператор БПЛА має дотримуватись при плануванні маршруту. Технологія включає урахування умов навколишнього середовища, тільки за яких доцільно здійснювати оброблення:

- вологість повітря;
- температура повітря;

– швидкість та напрямок вітру. Не всі види процедур допускають наявність вітру. В будь-якому випадку при плануванні маршруту треба враховувати цей фактор;
– час доби. При різкому сході чи заході сонця утворюються термальні колони, що створюють рухи повітряних мас.

При невиконанні цих умов виліт забороняється. Досить часто сприятливий час для виконання робіт витрачається на планування маршруту, що призводить до значних втрат. Також технологія виконання аграрних робіт включає взяття до уваги технічних вимог до БПЛА:

– відносна висота польоту БПЛА при виконанні робіт;
– ємність акумулятору БПЛА, що впливає на максимальний час польоту;
– швидкість руху БПЛА відносно землі та відносно повітряних мас;
– критичний радіус розвороту БПЛА;
– робоча ширина захвату.

Це основний, проте абсолютно не повний список даних, на основі яких базується побудова маршруту БПЛА. Більшість технологій внесення ЗЗР потребують постійної відстані від БПЛА до земного покрову вздовж усього маршруту. В інших сферах застосування ця проблема вирішується апаратним шляхом. На БПЛА встановлюється лазерний дальномір. Проте в аграрній сфері через надвисоку швидкість польоту та зелений покрив, дальномір має значну похибку, що при польоті на високій швидкості та низькій висоті не є допустимими. Тож оператор БПЛА змушений вручну проставляти висоти в кожній точці маршруту. На даний момент жодне ПЗ не будує маршрут із врахуванням відносної висоти.

Слід зазначити, що при побудові маршруту для БПЛА опертатор дотримується певної послідовності кроків. Ця послідовність визначає спосіб руху БПЛА. Ідея полягає у тому, щоб покрити всю площу робочої ділянки слідуючи найкоротшим маршрутом. На даний момент використовуються лише 2 способи: човниковий або маятниковий (гоновий з петльовими

поворотами) та фігурний (круговий по спіралі зі зменшенням радіусу кожного наступного кола). Ці способи руху серед авіамоделістів дістали назву “змійка” та “спіраль”.

Планування маршрутів потребує доступу до мережі Інтернет для завантаження географічної та GDEM мап від різних провайдерів. Найбільш ймовірне місце розташування оператора БПЛА при виконанні робіт з побудови маршруту – поле в сільській місцевості. За даними Ukrtelecom: в Україні 8,3 млн. осіб які мешкають в 21,7 тис сіл, позбавлені можливості користуватися швидкісним інтернетом [13]. Щоб позбавитися цієї проблеми, всі необхідні дані необхідно завантажувати заздалегідь з місць де наявний доступ до мережі.

Досить часто сільська місцевість не має достатньої якості зображення при масштабі мапи більше 18. Цей масштаб необхідний для візуального розуміння границь поля та наявності на ньому перешкод (дерев, стовпів і т.д.). Ще частіше трапляється перекривання географічної мапи хмарами. Щоб уникнути даної проблеми, необхідно мати можливість змінювати провайдер мапи.

1.2. Існуючі програмні рішення

Планування польоту, орієнтованого на місію, є складним та трудомістким процесом, що потребує глибоких знань про автопілоти, їх внутрішні комунікаційні та реалізаційні деталі, динаміку польотів літальних апаратів, оптику тощо. Планувальник місії приховує всі ці технічні деталі за простим у використанні інтерактивним інтерфейсом, де потрібно лише вказати деталі, які пов’язані з місією, а всі інші завдання зі створення оптимального плану польоту виконуються повністю автоматично [14]. Отримані навігаційні точки та команди автопілоту генеруються відповідно до визначених вимог, пов’язаних з місією, орієнтуючись на оптимальні характеристики БПЛА.

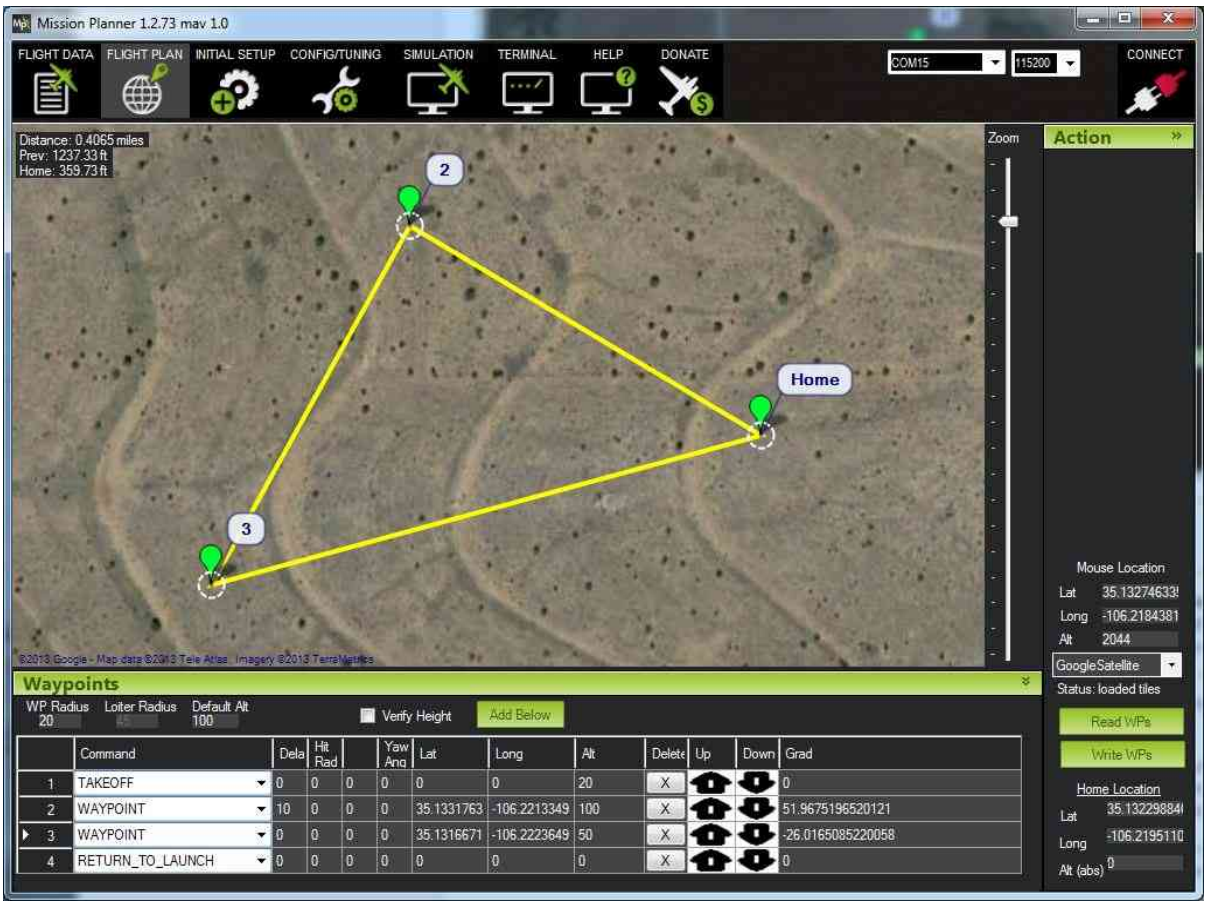


Рис. 1.1. Побудова маршруту за допомогою Mission Planner

1.2.1. *Mission Planer*

Mission Planner – це найпоширеніша повнофункціональна наземна станція для взаємодії з атопілотом ArduPilot з відкритим вихідним кодом. Mission Planner – це наземна станція управління для літака, вертольота і ровера (машини). Він сумісний лише з ОС Windows. Mission Planner може бути використаний як конфігураціна утиліта або як доповнення для динамічного контроль в реальному часі за БПЛА. Ось лише кілька речей, які можливо зробити з Mission Planner:

– завантажити прошивку (програмне забезпечення) в плату автопілоту (наприклад, серію Pixhawk), яка буде керувати БПЛА;
– налаштувати БПЛА для оптимальної роботи;
– планувати, зберігати в текстовому форматі і завантажувати автономні місії в автопілот, використовуючи ручну поточкову побудову на Google, Bing або іншій доступній мапі;

– завантажувати та аналізувати журнал місій, створений автопілотом;
– відстежувати стан свого БПЛА під час роботи.

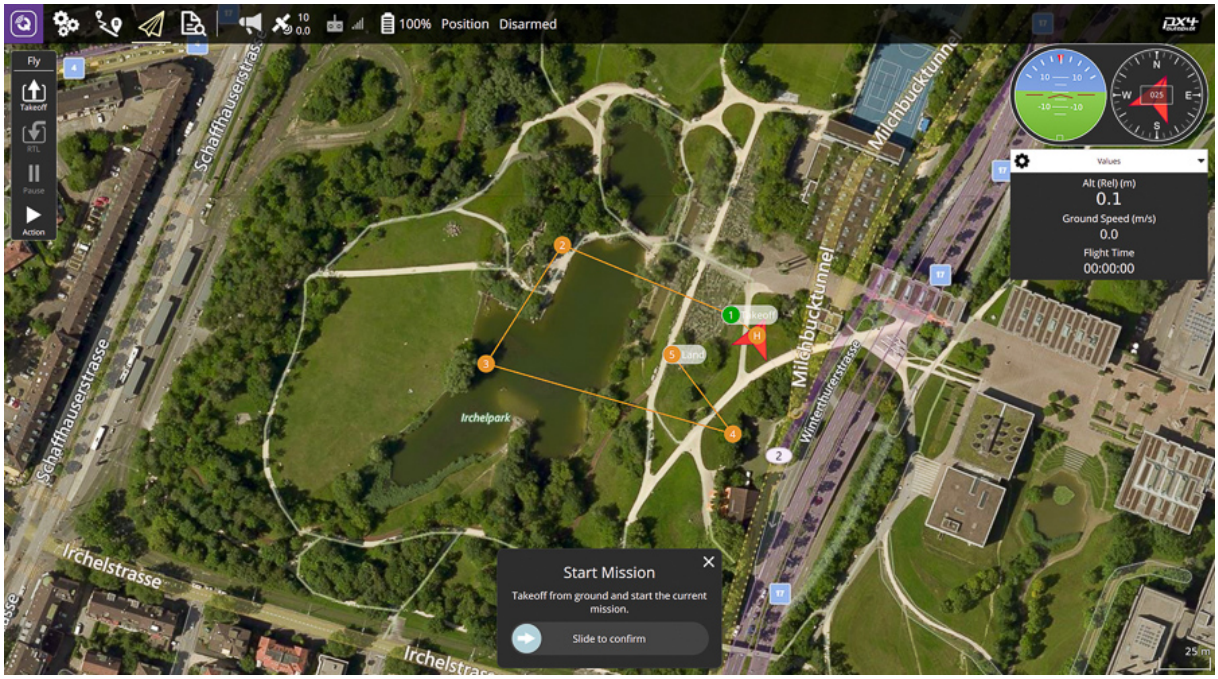


Рис. 1.2. Приклад побудови маршруту за допомогою QGroundControl

1.2.2. *Наземна станція QGroundControl*

QGroundControl забезпечує повний контроль над польотом та налаштуванням БПЛА, що працюють на PX4 або ArduPilot. Дана наземна станція забезпечує просте і прямолінійне використання для початківців, в той же час забезпечуючи високу підтримку функцій для досвідчених користувачів. Основні можливості:

– повна установка та конфігурація БПЛА автопілот яких базується на ArduPilot та PX4 Pro;
– підтримка управління декількома БПЛА одночасно;
– підтримка польотів для транспортних засобів, що використовують PX4 та ArduPilot (або будь-який інший автопілот, який встановлює зв'язок зі станцією за допомогою протоколу MAVLink);
– планування місії для автономного польоту;

– відображення карти польоту, що показують положення БПЛА, траєкторію польоту, шляхові точки та основні прибори;
– підтримка потокового відео з накладками на дисплеї;
– робота на платформах Windows, OS X, Linux, iOS і Android.

1.2.3. *Результати аналізу існуючих рішень*

Існує незліченна кількість аналогів наземних станцій. Для порівняння вибрано найпопулярніші. Здебільшого всі наземні станції мають подібні можливості, проте дещо можуть різнитись в залежності від типу автопілоту, виробника чи платформи, на яку націлена станція. У реальних умовах більша частина основних функцій перелічених рішень не використовується (або йде платною в комплекті з БПЛА), а користувацький інтерфейс досить складний для непідготовленого користувача. Сама ж побудова маршруту займає досить тривалий період часу. Кожен з розглянутих аналогів має можливість лише ручного додавання точок маршруту. Жодне з сучасних рішень не було націлене лише на побудову маршруту БПЛА.

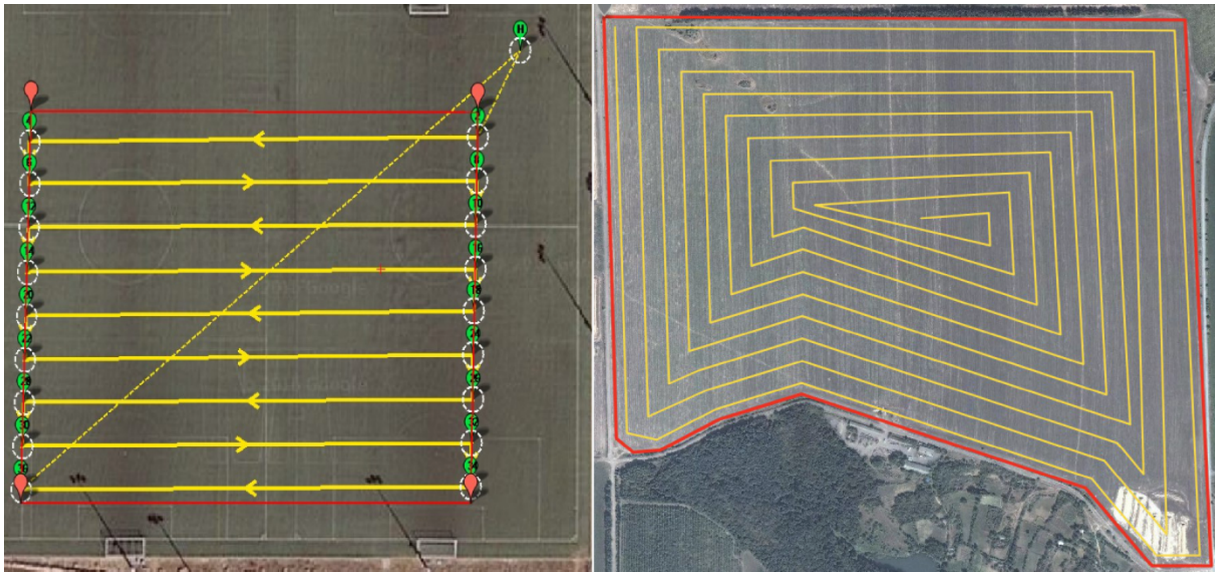


Рис. 1.3. Порівняння способів обходу гонами та круговий (спіраль)

Деякі наземні станції мають можливість автоматичної побудови маятникового маршруту гонами. Проте побудова відбувається без

врахування висот, тож оператор буде змушений переглянути кожну точку. Також не ведеться врахування природних перешкод (лісосмуг, ЛЕП чи стовпів). Через значну кількість поворотів (порівняно з круговим способом обходу) маршрут гонами завдає більшого навантаження на механізми БПЛА, що швидше виводить його зі строю. Певні технології аграрних робіт передбачають лише маршрут по спіралі.

Таким чином, результатом проведеного аналізу є виявлення невідповідності реальних рішень потребам аграрного бізнесу.

1.3. Постановка вимог до функціональності програмних засобів

Після проведення опитування зацікавлених сторін проекту (операторів БПЛА, які надають послуги по внесенню ЗЗР в Kernel [15], найбільшому агрохолдингу на території України) та вивчення всіх недоліків наявних рішень на практиці, було сформовано такі функціональні вимоги:

1. Застосунок має надавати можливість автоматично будувати маршрут БПЛА в заданій опуклій області за певними параметрами.
2. Застосунок має надавати можливість автоматично закруглювати гострі кути маршруту. Величина кута, який вважається гострим, для певної моделі БПЛА має задаватись згідно до радіусу повороту БПЛА.
3. Побудова маршруту має ефективно виконуватись без доступу до мережі Інтернет.
4. Застосунок має надавати можливість змінювати провайдер зображень географічної мапи.
5. Кожна точка побудованого маршруту повинна мати однакову висоту над рівнем землі відносно точки запуску.
6. Побудований маршрут має покрити всю площу робочої ділянки, слідуючи найкоротшим шляхом за чітко визначеним способом обходу “спіраль”.

7. Побудова маршруту має виконуватись не довше, ніж за хвилину при заданій площі робочої зони 500 гектар.
8. Застосунок має бути багатоплатформним.
9. Застосунок має надавати можливість графічно переглянути перепади висоти в заданій області з/без можливості доступу до мережі Інтернет.
10. Користувач повинен мати можливість швидко знайти своє місцезнаходження в редакторі на мапі.

Після проведення порівняльного аналізу з існуючими рішеннями, було виявлено, що при виконанні пунктів 1, 2, 5, 6, 7, проект набуде унікальності, адже станом на сьогодні ці можливості не реалізовані жодним аналогом.

2. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Вибір цільової платформи для реалізації проекту

Розроблюване ПЗ являє собою редактор для автоматичної побудови маршрутів БПЛА. В даному розділі описано причини вибору саме десктопної платформи та вибрані суміжні технології для реалізації цього дипломного проекту, що зображені на рис. 2.1.

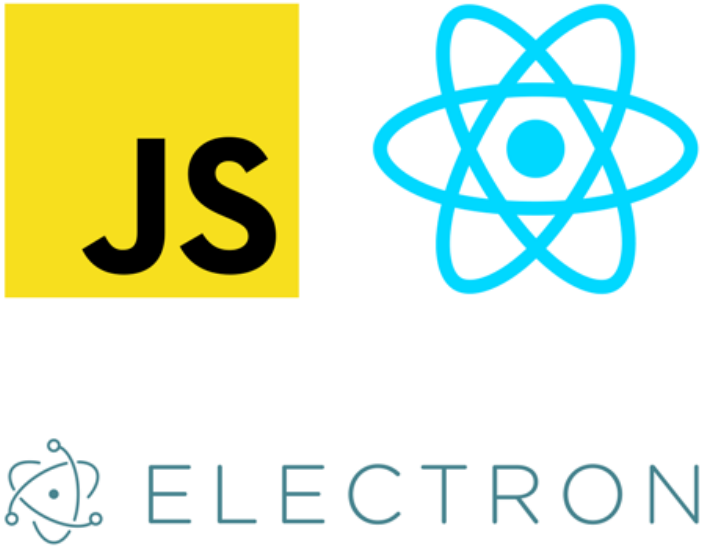


Рис. 2.1. Логотипи використаних технологій. (JavaScript, React.js, Electron.js)

2.1.1. *Настільні (десктопні) програми*

Настільні програми традиційно обмежені апаратним забезпеченням, на якому вони запускаються. Вони повинні бути розроблені та встановлені на певній операційній системі. При розробленні та встановленні такого типу ПЗ можуть виникати суворі апаратні вимоги, які повинні бути виконані для забезпечення правильної роботи. Оновлення програм повинне бути виконане користувачем безпосередньо до їх інсталяції і може вимагати оновлення обладнання (драйверів) або інших змін для роботи. Ця апаратна

залежність, а також застарілі застосунки, зазвичай обмежують рівень складності в інтерфейсах користувача настільних додатків. Проте в інших аспектах десктоп-застосунки значно переважають над іншими рішеннями.

2.1.2. *Порівняння десктоп-додатку з веб-додатком*

Порівняння виконане у табл. 2.1.

Таблиця 2.1

Порівняння десктоп- та веб-додатків

Критерій	Порівняння
Технічне обслуговування	Як вже згадувалося раніше, на відміну від веб-додатків, які встановлюються лише один раз, настільні програми повинні бути встановлені на кожному ПК незалежно. Коли йдеться про оновлення, речі залишаються незмінними. Необхідно передавати оновлення всім користувачам незалежно від того, скільки їх – кілька осіб або кілька тисяч. Більш того, додатки для настільних ПК потрібно оновлювати вручну кожного разу на кожному ПК, що також потребує деякого часу.
Масштабування	Настільні програми обмежені фізичним розташуванням, отже мають обмеження щодо використання. Розроблення веб-додатків, з іншого боку, робить його зручним для користувачів для доступу до програми з будь-якого місця за допомогою Інтернету.
Безпека	Зазвичай веб-програми стикаються з більшими ризиками для безпеки, ніж ті, що використовуються на ПК. Всім відомо, що Інтернет - не найбезпечніше місце. Отже, коли всі дані, що стосуються веб-додатків, зберігаються в хмарі, користувач ризикуєте втратити особисті дані.

Продовження табл. 2.1

Швидкість та продуктивність	Веб-додатки також звинувачують у тому, що вони повільніші за ті, що працюють на ПК. Розроблення веб-додатків значною мірою залежить від підключення до Інтернету та швидкості. Відсутність Інтернету або його погане підключення можуть викликати проблеми з продуктивністю веб-додатків. В свою чергу, десктоп-додатки є автономними за своєю природою, а отже, не стикаються з жодними перешкодами, що виникають в результаті підключення до Інтернету.
Вартість пропускної здатності	Оскільки веб-програми залежать від Інтернету, вони вимагають більшого використання мережі на стороні серверу, ніж прикладні програми на ПК.
Інтерфейс користувача	Інтерфейс користувача загалом не зазнає сильних змін залежно від платформи, протое десктоп-додатки мають набагато продуктивніші та більш оптимальні рішення. Наприклад, оброблення ‘hover’ подій.
Використання файлової системи	На відміну від веб-програм, які використовують сховище в хмарі, ПЗ для ПК потребує певного місця на самому пристрої. Іноді місця на жорсткому диску може бути недостатньо для коректної роботи програми, а покращення цієї проблеми може призвести до незручностей.

Хоча і настільні, і веб-програми мають свої плюси і недоліки, вони в кінцевому рахунку є лише інструментами, які люди використовують для вирішення проблем. Наприклад, можна працювати з електронними таблицями на своєму ноутбуці за допомогою настільної програми Microsoft Excel або веб-програми, як Документи Google. Обидві дозволяють здійснювати базове редагування електронних таблиць, але для роботи Google потрібне підключення до Інтернету. Щоб усунути цей недолік, деякі веб-програми розробили автономні можливості, які дозволяють почати роботу в Інтернеті, а потім продовжувати працювати пізніше, навіть якщо

користувач відключений від Інтернету. Деякі програми для настільних ПК також використовують технології, які спочатку були створені для створення веб-додатків. Наприклад, розробники програмного забезпечення можуть використовувати HTML і JavaScript для розроблення настільних додатків, а також веб-додатків. Це слугує гарним прикладом того, що додатки для ПК і веб-програми можуть запозичувати функції один одного.

2.1.3. *Результати порівняння*

Десктоп-додатки зазвичай мають більший контроль над комп'ютером користувачів порівняно з веб-додатком. Крім того, десктоп-додатоки часто перебувають в автономному режимі та не потребують підключення до Інтернету для функціонування в порівнянні з веб-додатками. Ця особливість підходить для типових місць роботи оператора БПЛА.

Порівнюючи переваги та недоліки кожного типу застосування, було визначено що для даного проекту краще використати десктопне рішення. Ключовим фактором, який вплинув на те, що дане програмне забезпечення має бути реалізованим у вигляді десктоп-додатку було те, що програмі необхідний доступ до файлової системи у великому обсязі (більшому ніж можуть надати сучасні PWA технології). В застосунку буде використовуватись кешування великих об’ємів даних про висоти та зображень географічних мап в певній місцевості. Також програмі необхідний доступ до USB порту для приєднання до телеметрії за протоколом MAVLink, що дасть змогу записувати маршрут безпосередньо на БПЛА без використання стороннього ПЗ.

2.2. Вибір технології для розроблення десктоп-додатку

Відповідно до функціональних вимог, даний проект повинен надавати багатоплатформене рішення, тому для реалізації десктоп-додатку вибрано бібліотеку Electron.js [16]. В цьому розділі буде обґрунтовано всі причини такого вибору.

2.2.1. *Аналіз сучасних способів розроблення десктоп-додатків*

Перш ніж говорити про Electron.js, потрібно досконало розібратися, як створюються десктоп-додатки для ПК. Також слід проаналізувати проблеми, що виникають при різних способах розроблення додатків для ПК, описати головні етапи створення повністю функціонального back-end сервісу, а також клієнтську частину, при розробці десктоп-додатків з використанням Electron.js

Основні ОС, для яких створюються десктоп-додатки:

– Windows. Загалом корпорація Майкрософт надає спільноті розробників розширений інструментарій для створення потужної системи керування та блискучого інтерфейсу для розроблення додатків Windows. Проте більшість розробників переважно використовують VB (Visual Basic), C ++, C# для розроблення додатків орієнтованих на цю систему. Розроблення може включати використання та повторне використання деяких бібліотек з відкритим вихідним кодом .NET для створення додатків у всіх екосистемах Windows. Найпоширеніші з них: Windows Store, Windows Phone, windows desktop application.

– MacOS. Розроблення ПЗ для MacOS має власну унікальну екосистмеу фреймворків. При розробці ПЗ для даної платформи, розробники мають чітку абстракцію технологій, які використовуються для створення додатків. Компанія Apple створила об'єктно-орієнтований програмний інтерфейс для MacOS під назвою "Cocoa" [17]. Він включає всі необхідні технології для створення інтерфейсу користувача додатку. Від розділений на певні шари (рівні абстракції). У шарі "Media" є всі інструменти та технології, необхідні для роботи з медіафайлами, які включають 2D та 3D анімацію, редагування фото та відео. Існує також рівень абстракції "Core Services", де можливо виконувати всі низькорівневі операції, взаємодіяти з мережею, маніпулювати з

рядками та даними. "Core OS" надає всі функціональні можливості для всіх процесорів і GPU для виконання високопродуктивних завдань. "Kernel & Device Drivers" надають підтримку роботи з файловими системи, мережею, взаємодіяти між процесами, драйверамив пристроїв і розширень ядра.

– Linux. Linux є однією з найпопулярніших ОС для розробників. На відміну від розроблення застосунків для інших ОС, розроблення під цю платформу вимагає більших зусиль. Linux зазвичай використовує Python або інші застосунки для розроблення інтерфейсів користувача. Для розроблення під платформу Linux, якщо потрібні можливості використання, такі як 3D і 2D рендеринг, розробники використовують OpenGL [18]. Здебільшого ядро Linux надає низькорівневі функціональні можливості для інших потреб. Проте більшість рішень, які вже є готові для інших платформ, при розробці під цю платформу треба буде реалізовувати самостійно.

2.2.2. *Обґрунтування вибору Electron.js*

Однією з функціональних вимог до цього проекту було забезпечення багатоплатформності. Це означає, що проект повинен бути більш простим і доступним для всіх платформ, а також з однією кодовою базою, підтримкою більшої кількості функцій і великим UI/UX. Жодне з перелічених вище напрямків розроблення не забезпечить цього. Неможна ігнорувати останні тенденції розроблення ПЗ. Жоден розробник не хоче писати 5 різних версій одного й того ж додатка. На допомогу приходить використання крос-платформних фреймворків. Одним з найкращих прикладом таких фреймворків є Electron.js.

Electron.js допомагає нам розвивати крос-платформні програми за допомогою існуючих веб-технологій. Не потрібні спеціальні навички, для більшості випадків, для розроблення додатків за допомогою Electron. В загальному ж випадку якщо постала задача, щоб програма була доступна

для всіх платформ, які були описані вище, потрібно розгортати розроблення за допомогою різних технологій та мов програмування. Це дуже рутинний і трудомісткий процес.

Якщо говорити про Electron.js, очевидно, що це фреймворк на основі JavaScript. Оскільки всі перелічені ОС підтримують веб-технології, Electron.js створений для розроблення крос-платформних додатків, базується на цьому принципі. Можливості, які надає Electron.js:

- Безпека. Розробнику не потрібно багато думати про безпеку додатку при міграції існуючих ПЗ на Electron.js, оскільки програма, яку отримають в кінці, є настільною програмою, а дані залишаються локально в системі. Завдяки цьому можна забезпечити безпеку даних. Якщо розробнику потрібно зберігати дані на віддалених серверах, то все, що треба зробити, це перевірити, чи достатньо захищена хмарна мережа, щоб уникнути небажаних наслідків.
- Доступність апаратного рівня. Electron.js забезпечує достатній контроль, щоб мати розширені інтерактивні функції у десктоп-додатку (такі як клавіатурні переривання та інші). Electron.js також забезпечує певний рівень доступності компонентів ПЗ до апаратних можливостей ОС.
- Продуктивність. Electron.js процвітає у цьому аспекті. Кожна нова версія фреймворку приносить кращі показники продуктивності. Якщо при розробці розробник звертає достатньо уваги цьому питанню, Electron.js може проявити великі успіхи в умовах продуктивності в порівнянні з програмами які були зроблені лише під певну ОС. Electron.js економить багато часу і надає більше можливостей. Єдина проблема, яку досі Electron.js вирішує недостатньо якісно – менеджмент використаної пам'яті.
- Керування кодовою базою. Electron.js не вимагає підтримки різних команд професіоналів для розроблення кодової бази для кожної

платформи (ОС). При розробці з Electron.js, за допомогою єдиної кодової бази реалізовується додаток одразу для всіх перелічених ОС. Це також зменшить витрати часу на те, щоб переконатися, що продукт має однакову функціональність для всіх цільових платформ.

– Повторне використання. Кодова база Electron.js повністю сумісна для використання в якості веб-додатку.
– Вартість та час. Розробник заощаджує багато часу і грошей на розроблення, тому, що для технологічного стека, який використовується, існує значно більша кількість розробників, які можуть зробити це за менші витрати і досягти не гірших результатів, порівняно з розробленням під одну платформу.

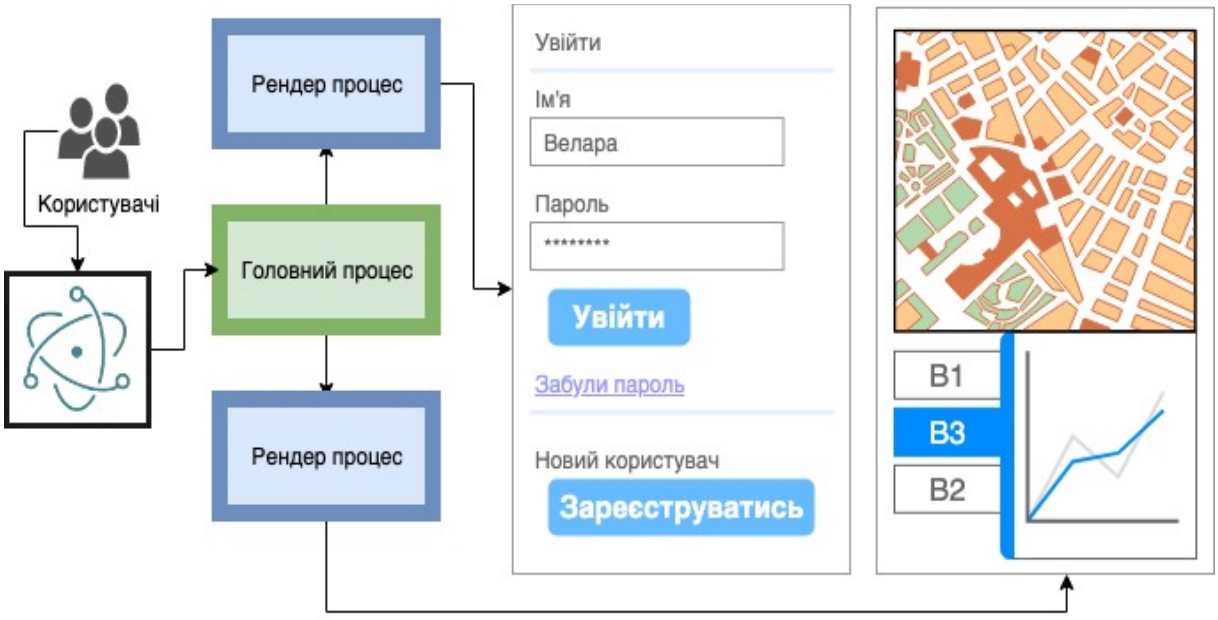


Рис. 2.2. Ілюстрація принципу роботи Electron.js

2.2.3. Принцип роботи Electron.js

Типова програма Electron.js використовує HTML, CSS, JavaScript і NodeJS поверх Chromium [19] двигуна. Розробник може використовувати інші фреймворки поверх базових налаштувань, такі як Vue.js [20] та React.js [21]. Це полегшить розроблення завдяки тому, що програми будуть

заздалегідь модульовані, що в свою чергу полегшує налагодження та розуміння програми.

Розглянемо найпростіший сценарій використання Electron.js (рис. 2.2):

1. Користувач запускає програму Electron.js з бажаної платформи, як Windows / MacOS / Linux на базі Ubuntu.
2. ПЗ робить запит до головного вікна, використовуючи головний процес. Головний процес виконується в середовищі Node.js. Рендер- процеси виконують JavaScript. Electron.js надає можливість комунікації цих процесів в рамках однієї програми, що робить Electron.js спроможним вирішити задачі, які не могли бути виконані лише в клієнтському чи лише в серверному середовищі.
3. Головний процес відповідає за те, щоб відправити і отримати запит від користувача та передачу даних між різними вікнами. Це як центральний медіатор для обміну даних для всіх вікон. Тут же обробляється все управління пам'яттю, створення і знищення вікон.
4. Головний процес робить запит на запуск вікна для реєстрації (рендер-процес).
5. Вікно для реєстрації буде запущене і приєднане до основного процесу, будучи доступним для взаємодії з користувачем.
6. Користувач вводить реєстраційну інформацію, після чого Electron.js буде обробляти натискання кнопок та інші події. В процесі візуалізації, рендер-процес буде подавати запити на головний процес.
7. Основний процес отримає відповідну інформацію через слухач подій і запустить вікно з редактором мапи, використовуючи інший рендер-процес.
8. Цей цикл продовжується для всієї програми доти, доки не буде зачинене головне вікно програми.

2.3. Вибір технології для розроблення клієнтської частини

Для розроблення клієнтської частини рендер-процесу Electron.js було вибрано сучасний фреймворк React.js. React.js в основному являє собою JavaScript бібліотеку з відкритим кодом, яка використовується для побудови інтерфейсів користувача спеціально для односторінкових веб-інтерфейсів (SPA). Він використовується для оброблення шару view з шаблону програмування MVC для веб та мобільних програм. React також дозволяє створювати повторно використовувані компоненти інтерфейсу. React дозволяє розробникам створювати великі веб-програми, які можуть змінювати дані без перезавантаження сторінки. Основна мета React - бути швидким, масштабованим та простим. Його можна використовувати з комбінацією інших бібліотек JavaScript або фреймворків.

Тепер, головне питання, яке виникає, це те, чому слід використовувати саме React.js. Існує дуже багато платформ з відкритим кодом для полегшення розроблення веб-додатків, таких як Angular. Розглянемо переваги React.js у порівнянні з іншими конкурентними технологіями або фреймворками. Основні переваги, які надає React.js:

1. Простота. React.js просто зрозуміти відразу. Компонентний підхід, чітко визначений життєвий цикл і використання лише простого JavaScript роблять React дуже простим для вивчення, навчання, створення професійних веб-інтерфейсів та їх підтримки.
2. Синтаксис. React.js використовує спеціальний синтаксис під назвою JSX, який дозволяє змішувати HTML з JavaScript. Розробник все ще може писати на чистому JavaScript, але JSX набагато простіше використовувати. Останні версії React.js включають так звані ‘hooks’, що дозволяють писати компоненти інтерфейсу простими функціями. Хуки повністю підлягають повторному використанню. Жоден рядок коду не буде повторюватись.

3. Прив'язка даних. React.js використовує односторонню прив'язку даних і архітектуру додатків, що називається Flux [22], що керує потоком даних та станом кожного компоненту через одну контрольну точку – диспетчер. Легше налагоджувати автономні компоненти великих програм.
4. Здатність до тестування. Клієнтські інтерфейси написані на React.js дуже легко тестувати. Компоненти створені за допомогою React.js можуть розглядатися як результати виконання функцій від певних даних (стану), тому ми можемо маніпулювати та підставляти необхідний стан як параметри функції. Це дає можливість спостерігати за виводом, спрацюванням певних дій, подій, функції тощо, тощо.

2.4. Вибір бібліотеки для взаємодії з мапою

Рис. 2.3. Приклад використання бібліотек Turf.js та Leaflet.js

Leaflet.js [25] – це JavaScript бібліотека з відкритим кодом для додавання інтерактивності до карт. Вона включає безліч можливостей та плагінів для можливості реалізації будь-якої взаємодії з географічною

мапою у веб-інтерфейсі. Leaflet.js не є повноцінним рішенням для додавання мапи, бібліотека лише відповідає за способи відображення мапи отриманої від інших провайдерів. Цю особливість дуже зручно використовувати, адже розробник може написати один код, який буде відповідати за відображення мапи одразу всіх провайдерів, таких як Google Maps [23], MapBox [24], Bing Maps [25], OpenStreetMaps [26] та багато інших.

Leaflet.js підтримує взаємодію з багатьма ГІС форматами даних, проте в даному проекті буде використовуватись GeoJSON [27]. Leaflet.js надає можливості розробнику створювати власні унікальні UI/UX, надаючи основні елементи мапи. На відміну від інших платформ (Google Maps), кожен з цих елементів є повністю налаштовуваним. Також розробник може створювати власні елементи.

Елементи діляться за типом:

- Растровий тип. Включає TileLayer та ImageOverlay, що дають змогу накладати будь-яке тло мапи, яке надав провайдер або власні зображення. Використовуються для відображення даних самої мапи.
- Векторний тип. Включає Path, Polygon, Polyline, Marker, Circle та багато інших примітивів. Використовуються для відображення даних на мапі.
- Груповий тип. Включає LayerGroup, FeatureGroup та GeoJSON. Надає змогу групувати примітиви та застосовувати спільні параметри до всіх елементів групи. GeoJSON підтримує перетворення даних з JSON формату до примітивів мати.
- Елементи управління. Включає Zoom, Scale та багато інших. Надають розробнику можливість створювати власні елементи управління мапою.

2.5. Вибір бібліотеки для проведення обчислень та маніпуляцій з географічними координатами

Turf.js – це JavaScript бібліотека для просторового аналізу. Він допомагає аналізувати, агрегувати та трансформувати дані для того, щоб візуалізувати його сучасними способами [28]. Turf.js використовує GeoJSON для всіх географічних даних. Turf.js передбачає, що координати довготи та широти будуть задані згідно зі стандартом WGS8. Більшість функцій Turf.js підтримують роботу з функціями GeoJSON. Це формат даних, що представляє собою збереження географічний об'єктів та їх набору властивостей (населення, висота, поштовий індекс тощо) разом з геометрією.

Turf.js використовується для просторового аналізу. Великий спектр завдань, таких як "обчислення площі та відстані" та "точки приєднання до багатокутників", які дозволяють людям бачити нові аспекти своїх даних. Просторовий аналіз використовується в кожній галузі: знайти найближчу кав'ярню, розрахувати час подорожі і показати регіональну статистику по використанню чого-небудь. Мова йде також про велику частину ГІС, де багато проблем вирішуються за допомогою просторового аналізу.

Варто зазначити, що виконуючи даний дипломний проект, всі обчислення, пов’язані з географічними даними, виконанні за допомогою цієї бібліотеки у форматі GeoJSON.

Аналогічні бібліотеки не розглядались тому, що не були знайдені.

3. ОПИС РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

3.1. Архітектура програмних засобів

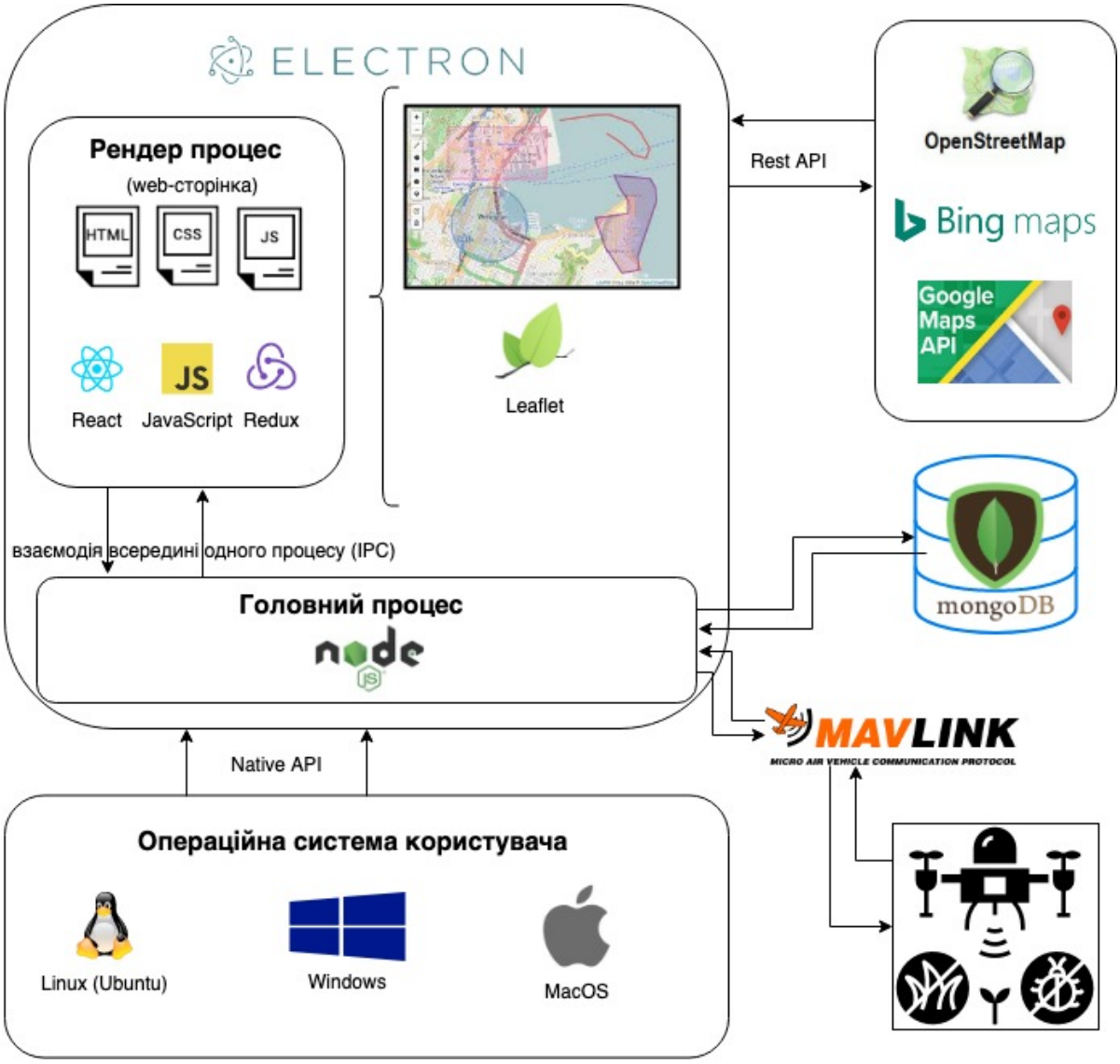


Рис. 3.1. Структурна схема ПЗ

Відповідно до функціональних вимог, програмні засоби реалізовані у вигляді багатоплатформного десктоп-додатку. Структурна схема програмних засобів зображена на рис. 3.1. На схемі наведено шляхи взаємодії ключових елементів ПЗ, а саме: взаємодія ОС користувача з десктоп-додатком Electron.js, міжпроцесорну взаємодію головного та рендер-процесів Electron.js, взаємодія головного процесу Electron.js з

автопілотом БПЛА за протоколом MAVLink, отримання даних від провайдерів географічних мап за REST API, взаємодія з базою даних MongoDB.

Основні функції, що виконує головний процес:

- Кешує зображення географічної мапи;
- Надає сервіс для знаходження відносної висоти за географічними координатами у двох випадках:
  1. при доступі до мережі висоти знаходяться за доступними API;
  2. при відсутності доступу до Інтернету висоти знаходяться із закешованих GDEM мап;
- Відповідає за з'єднання з БПЛА за USB.
- Надає можливість зчитувати за записувати повний маршрут на автопілот БПЛА за протоколом MAVLink.
- Відповідає за побудову маршруту БПЛА в заданій області.
- Відповідає за побудову рельєфної мапи в заданій області.
- Зберігає побудовані маршрути в різних форматах в базі даних та локально.

Рендер процес відповідає за взаємодію користувача з елементами інтерфейсу, а саме за:

- відображення географічної мапи та взаємодію з нею;
- роботу панелі налаштувань;
- відображення карти рельєфу в заданій області.

3.2. Структури даних

3.2.1. *Менеджер стану Redux*

Для передбачуваного контролю стану додатку використано шаблон проектування Redux [29]. Redux — це інструмент управління станом даних в JavaScript-додатках. З використанням Redux стан застосунку зберігається в одному сховищі, яке доступне з будь-якої точки програми. Дані

зберігаються в певних іменованих контейнерах. Унікальність підходу полягає в тому, що контейнери з даними доступні лише для читання. Для мутацій та оновлення даних існують action-функції, які задають тип мутації. Саму ж мутацію для кожного контейнеру окремо визначає спеціальна reducer-функція. Потік даних продемонстровано на рис. 3.2.

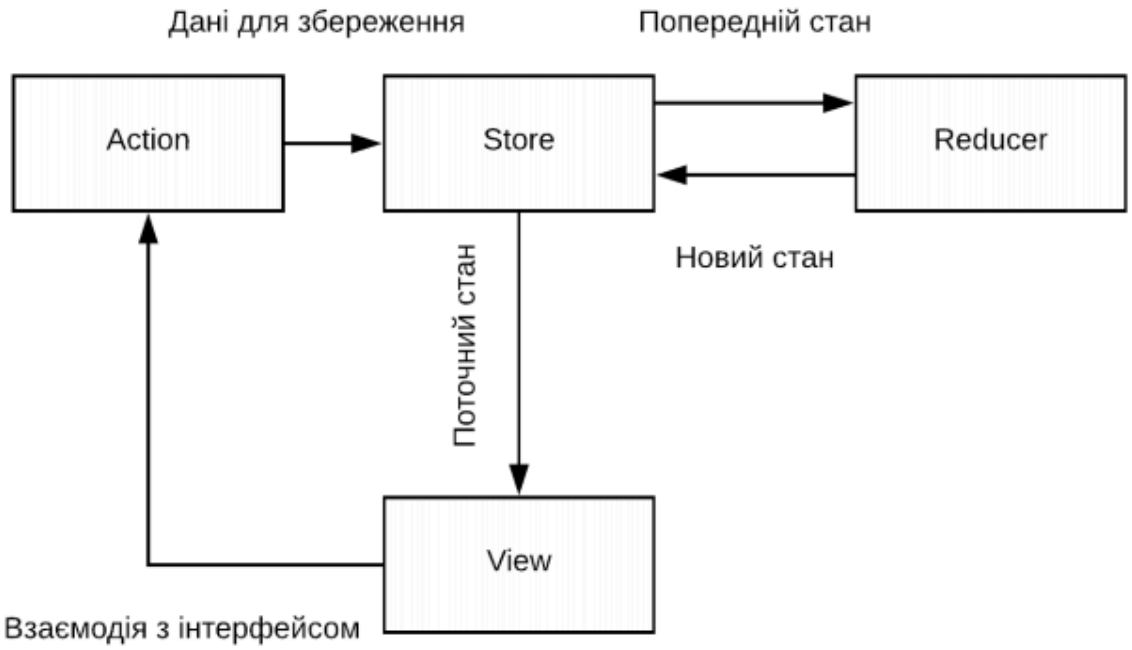


Рис. 3.2. Схема потоку даних з використанням Redux

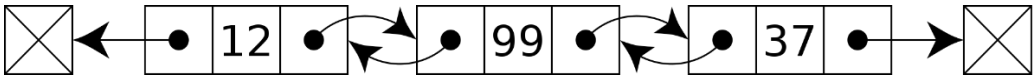


Рис. 3.3. Точки маршруту збережені як двобічно зв’язаний список

В даному проекті передбачені такі контейнери для даних:

1. MissionPoints. Контейнер для збереження поточного стану точок маршруту. Використовується для відображення точок маршруту на мапі. Ці дані є тимчасовими, адже після кожної мутації точки оновлюються. Мутації можуть спричинятись користувачем при редагуванні чи додаванні нових точок до місії, або при автоматичній генерації маршруту. Контейнер має структуру двобічно зв’язаного лінійного списку. Кожна точка маршруту знає

про сусідні точки (рис. 3.3). Детальніше про структуру точок маршруту буде описано у пункті 3.2.2.

2. Elevation. Контейнер для збереження даних з в висотами в шуканих географічний координатах. Використовується для відображення географічного рельєфу в шуканій області. Контейнер представлений у вигляді масиву. Кожний елемент масиву має таку структуру:
   - box – набір географічних координат, що описує квадрат;
   - center – центральна точка квадрату;
   - alt – середня відносна висота рельєфу в поточному квадраті.
3. AreaMarkup. Контейнер для зберігання усіх тимчасових розміток та маркерів на мапі. Включає в себе такі поля:
   - missionAreaPolygon – полігон, що описує область, в якій буде автоматично побудований маршрут;
   - elevationAreaPolygon – полігон, що описує область, в якій будуть графічно зображені перепади висоти в рельєфі;
   - trees – масив з координатами, за якими знаходиться перешкода типу “дерево”. Кожне дерево має свою висоту та радіус;
   - homePosition – описує географічні координати та висоту точки, яку БПЛА буде вважати за дім. При втраті сигналу, БПЛА автоматично повертається в цю точку. Також висота польоту рахується відносно цієї точки, а не відносно рівня моря;
   - powerLines – масив з координатами, за якими знаходиться перешкода типу “ЛЕП”. Кожен стовп має свою висоту та бажані радіус та висоту обльоту.
4. GeoPosition. Контейнер, що зберігає дані про географічне положення наземної станції. Використовується для знаходження місцезнаходження оператора БПЛА на мапі. GPS дані зчитуються з автопілоту БПЛА, якщо він приєднаний, або з пристрою, на якому запущений додаток.

5. MapContainer. Контейнер, що зберігає поточний стан елементів управління мапою. Має такі поля:
   - center – географічні координати центральної області мапи, щоб відновити мапу в тому ж місці;
   - zoom – значення поточного масштабу мапи;
   - layer – значення поточного шару мапу, який надходить від певного провайдеру;
   - isLoading – булеве значення, що використовується для блокування мапи при виконанні над нею операції
   - drawMode – значення, яке визначає режим взаємодії з мапою. Існує три головних режими:
     1. Редагування та перегляд маршруту.
     2. Редагувння зони для автоматичної побудови маршруту.
     3. Редагування зони, для графічного відображення рельєфу.
6. Notifications. Контейнер, що зберігає чергу повідомлень, які будуть відображатися після певних подій. Коли спрацьовує подія – на головному екрані з’являється повідомлення.

Варто зазначити, що після кожного виходу з програми всі контейнери з даними кешуються. Під час наступного запуску програми всі дані будуть відновлені. Це зроблене, щоб запобігти втраті даних побудови при випадковому закриванні програми.

3.2.2. *Структура та формат збереження точок маршруту*

Вся інформація про маршрут в додатку зберігається в форматі GeoJSON. GeoJSON – це формат для кодування різних географічних структур даних. Цей формат синтаксисом не відрізняється від звичайного JSON, але існує специфікація (GeoJSON RFC 7946 [30]), що описує правила описання геометрії фігур. GeoJSON підтримує такі типи геометрії: Point, LineString, Polygon, MultiPoint, MultiLineString і MultiPolygon. GeoJSON

передбачає зберігання мета даних про описану фігуру. Геометричні об'єкти з додатковими властивостями (метаданими) називаються Feature.

GeoJSON дуже зручно використовувати, адже бібліотека Turf.js всі обчислення виконує у цьому форматі. Також GeoJSON підтримується базою даних MongoDB при вибірках.


```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "alt": 157,
        "command": 16,
        "prev": null,
        "sequenceNumber": 0,
        "id": "205318fc-7fa3-11e9-bc42-526af7764f64",
        "next": "20531b68-7fa3-11e9-bc42-526af7764f64"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          31.63676261901855,
          49.02125228804479
        ]
      }
    },
    {
      "type": "Feature",
      "properties": {
        "alt": 159,
        "command": 16,
        "prev": "205318fc-7fa3-11e9-bc42-526af7764f64",
        "sequenceNumber": 1,
        "id": "20531b68-7fa3-11e9-bc42-526af7764f64",
        "next": null
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          31.63549661636353,
          49.01667884311484
        ]
      }
    }
  ]
}
```


Рис. 3.4. Приклад збереження маршруту БПЛА у форматі GeoJSON

Географічні координати зберігаються за стандартом WGS84. Згідно з цим стандартом висота кожної точки відраховується від початкового положення (точки запуску). Перше значення (x) – задає широту точки, друге

значення (y) – довготу, третє значення (alt) – висоту відносно точки включення автопілоту. Кожна точка маршруту містить такі дані:

- id – згенерований універсальний унікальний ключ (16-бітний номер, 32 символи);
- sequenceNumber – порядковий номер. Починається з нуля. Збільшується монотонно для шляхової точки маршруту, немає прогалин в послідовності (0, 1, 2, 3, 4). Використовується для завантаження точки в автопілот;
- x – географічна широта;
- y – географічна довгота;
- alt – висота, відраховуючи від точки старту;
- command – команда для автопілоту БПЛА, яку він має виконати в заданій точці маршруту.

Маршрут складається з окремих точок. Кожна точка передбачає команду для автопілоту БПЛА. Серед всіх можливих команд, які надає протокол MAVLink, були вибрані ті, які можуть бути безпечно автоматизовані. Вибрані команди не потребують додаткових параметрів і обмежуються лише визначенням точки в просторі та способом її досягнення. Кожній команді відповідає ціле число, яке буде опрацьоване автопілотом.

Таблиця 3.1

Перелік використаних команд маршруту БПЛА

№	Назва команди	Опис
1	MAV_CMD_NAV_WAYPOINT(16)	Слідувати до вказаної точки у просторі

Продовження табл. 3.1

2	MAV_CMD_NAV_LOITER_UNLIM(17)	Кружляти навколо вказаної точки маршруту необмежену кількість часу
3	MAV_CMD_NAV_RETURN_TO_LAUNCH (20)	Повернутися до місця запуску
4	MAV_CMD_NAV_LAND(21)	Здійснити посадку БПЛА у заданій точці
5	MAV_CMD_NAV_TAKEOFF(22)	Здійснити зліт БПЛА з заданої точки

Системи з використанням MAVLink часто повинні бути в змозі зберігати, обмінювати або відновлювати інформацію MAVLink, включаючи заплановані маршрути. Часто інформація визначається в одній системі і використовується на іншому пристрої (наприклад, місії створюються за допомогою певних інструментів планування і запускаються з комп'ютера-компаньйона). У багатьох системах для зберігання інформації про маршрут використовується формат звичайного тексту. Формат збереження маршруту для БПЛА зображений нижче.

Як показано на рис. 3.5, перший рядок маршруту містить інформацію про формат файлу та версію, а наступні рядки – це точки маршруту БПЛА. Відступи між номерами та полями протавляються символом tab, так, як на більшості мов програмування.

```
QGC WPL <VERSION>
<INDEX> <CURRENT WP> <COORD FRAME> <COMMAND> <PARAM1> <PARAM2> <PARAM3> <PARAM4>
<PARAM5/X/LONGITUDE> <PARAM6/Y/LATITUDE> <PARAM7/Z/ALTITUDE> <AUTOCONTINUE>
```

```
QGC WPL 110
0   1   0   16  0.149999999999999994    0   0   0   8.54800000000000004 47.3759999999999977 550 1
1   0   0   16  0.149999999999999994    0   0   0   8.54800000000000004 47.3759999999999977 550 1
2   0   0   16  0.149999999999999994    0   0   0   8.54800000000000004 47.3759999999999977 550 1
```

Рис. 3.5. Формат збереження та приклад збереження маршруту для БПЛА

3.3. Спосіб комунікації додатку з БПЛА

3.3.1. *Протокол MAVLink*

Для всіх комунікацій з БПЛА використовується MAVLink, легкий протокол зв'язку між безпілотними літаками і наземними станціями керування (GCS). Протокол визначає набір двонаправлених повідомлень, що обмінюються між БПЛА і наземною станцією. Повідомлення кодують інформацію про стан БПЛА і команди керування, відправлені з наземної станції.

Всі можливі повідомлення визначаються у файлах XML. Кожен файл XML визначає набір повідомлень, що підтримується певною системою MAVLink, також називається "діалект". Набір посилальних повідомлень, який реалізується більшістю наземних станцій управління і автопілотів, визначається в common.xml (більшість діалектів будуються поверх цього визначення). В цьому проекті використано саме common.xml діалект, адже він буде підтримуватись для всіх автопілотів БПЛА.

Для обміну повідомленнями MAVLink використовує сучасні шаблони проектування: гібридний шаблон publish-subscribe та point-to-point. Потоки даних про поточний стан БПЛА (положення БПЛА в просторі, технічний стан БПЛА, швидкість руху БПЛА, процент заряду акумулятору, режим польоту, heartbeat дані, кількість супутників, що направлять GPS, час польоту) надсилаються як publish-subscribe, тоді як збереження маршруту або зміна параметрів надсилаються за шаблоном point-to-point з повторною

передачею. Розглянемо основні сценарії комунікації, які реалізовані в цьому проекті.

3.3.2. *Завантаження та зчитування маршруту з БПЛА*

На діаграмі нижче показана послідовність комунікацій для завантаження місії на БПЛА. Розглянуто ідеальні умови, тобто всі операції будуть виконані без помилок.

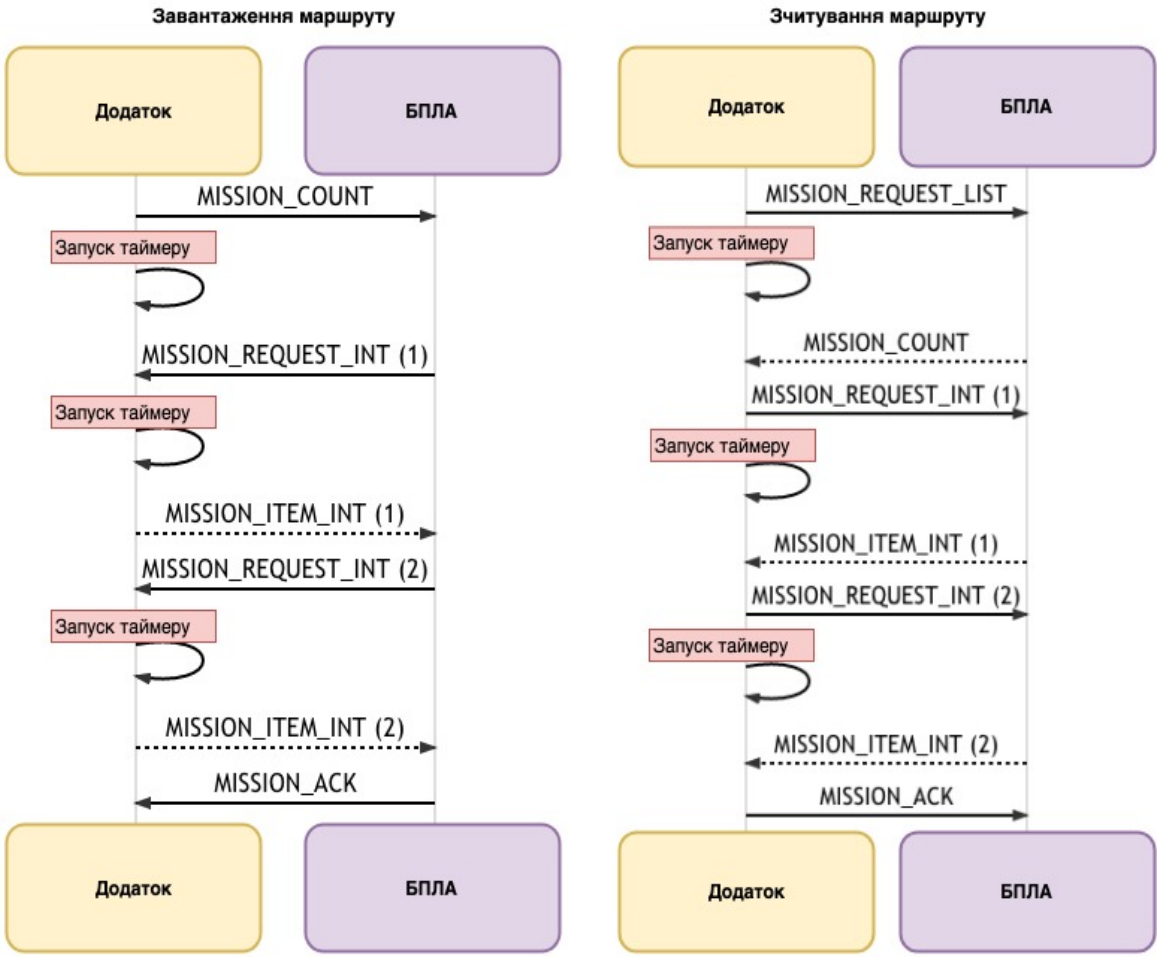


Рис. 3.5. Послідовність команд при завантаженні та зчитуванні маршруту БПЛА

Більш детально про послідовність операцій:

1. Додаток надсилає MISSION_COUNT команду, включаючи поле count, що задає кількість точок маршруту для завантаження.

2. Щоб додаток чекав на відповідь від БПЛА (MISSION_REQUEST_INT), потрібно запустити тайм-аут.
3. БПЛА отримує повідомлення і готується до завантаження елементів місії.
4. БПЛА відповідає з повідомлення MISSION_REQUEST_INT, запитуючи перший елемент місії. БПЛА передає поле seq, що дорівнює 0.
5. Додаток отримує повідомлення MISSION_REQUEST_INT і відповідає запитаним елементом місії в повідомленні MISSION_ITEM_INT.
6. БПЛА і додаток повторюють етапи 4 та 5, обмінюючись командами MISSION_REQUEST_INT та MISSION_ITEM_INT, збільшуючи поле seq, поки всі елементи маршруту не будуть завантажені.
7. Після отримання останнього елемента маршруту БПЛА надсилає команду MISSION_ACK з результатом операції завантаження в полі type. Якщо type дорівнює 0, то місія завантажена успішно. В іншому випадку type містить код помилки.
8. При успішному виконанні тип має бути встановлено на MAV_MISSION_ACCEPTED (0). У разі невдачі, тип повинен мати значення MAV_MISSION_ERROR або інший код помилки.
9. Якщо трапилась помилка, то транзакція не вдалася, але її можна повторити.

Варто звернути увагу на те, що додаток встановлює тайм-аут після кожного повідомлення і повторно надсилає повідомлення, якщо не буде відповіді від транспортного засобу. Також додаток буде повторно запитувати відсутні елементи місії, якщо вони отримані поза послідовністю.

При зчитуванні маршруту з БПЛА, послідовність команд залишається майже незмінною. Основна відмінність полягає в тому, що додаток надсилає спершу команду MISSION_REQUEST_LIST, яка запускає автопілот для відповіді з поточним числом завантажених елементів маршруту.

Починається цикл, коли додаток запитує кожну точку маршруту, а БПЛА постачає їх.

3.3.3. *Видалення маршруту з пам'яті БПЛА*

Перед тим, як завантажити новий маршрут БПЛА в пам'ять автопілоту, необхідно видалити той, який був збережений. Розглянемо алгоритм видалення маршруту при ідеальних умовах, тобто коли всі операції будуть виконані без помилок:

1. Додаток надсилає команду MISSION_CLEAR_ALL.
2. Починається тайм-аут, який чекає на MISSION_ACK відповіді від БПЛА.
3. БПЛА отримує повідомлення і очищує місію.
4. Місія вважається очищеною, якщо наступні запити на підрахунок місій або поточний пункт місії вказують на те, що не було завантажено місію.
5. БПЛА відповідає з командою MISSION_ACK, яка включає тип результату MAV_MISSION_RESULT. Після успішного видалення, це поле має бути встановлено на MAV_MISSION_ACCEPTED. У разі збою тип відповіді повинен мати значення MAV_MISSION_ERROR або інший код помилки.
6. Додаток отримує MISSION_ACK. Якщо відповідь має значення MAV_MISSION_ACCEPTED, БПЛА видаляє свою власну збережену інформацію про місію.
7. Якщо трапилась помилка, транзакція виходить з ладу, і запис про місію (якщо така є) зберігається.
8. Якщо MISSION_ACK не отримано, операція, зрештою, закінчиться тайм-аутом і може бути повторена.

3.4. Опис алгоритму побудови маршруту

Основний алгоритм застосунку вираховує положення точок в просторі, що в сукупності формують маршрут для БПЛА в формі спіралі. Блок-схема запропонованого алгоритму зображена на рис. 3.6.

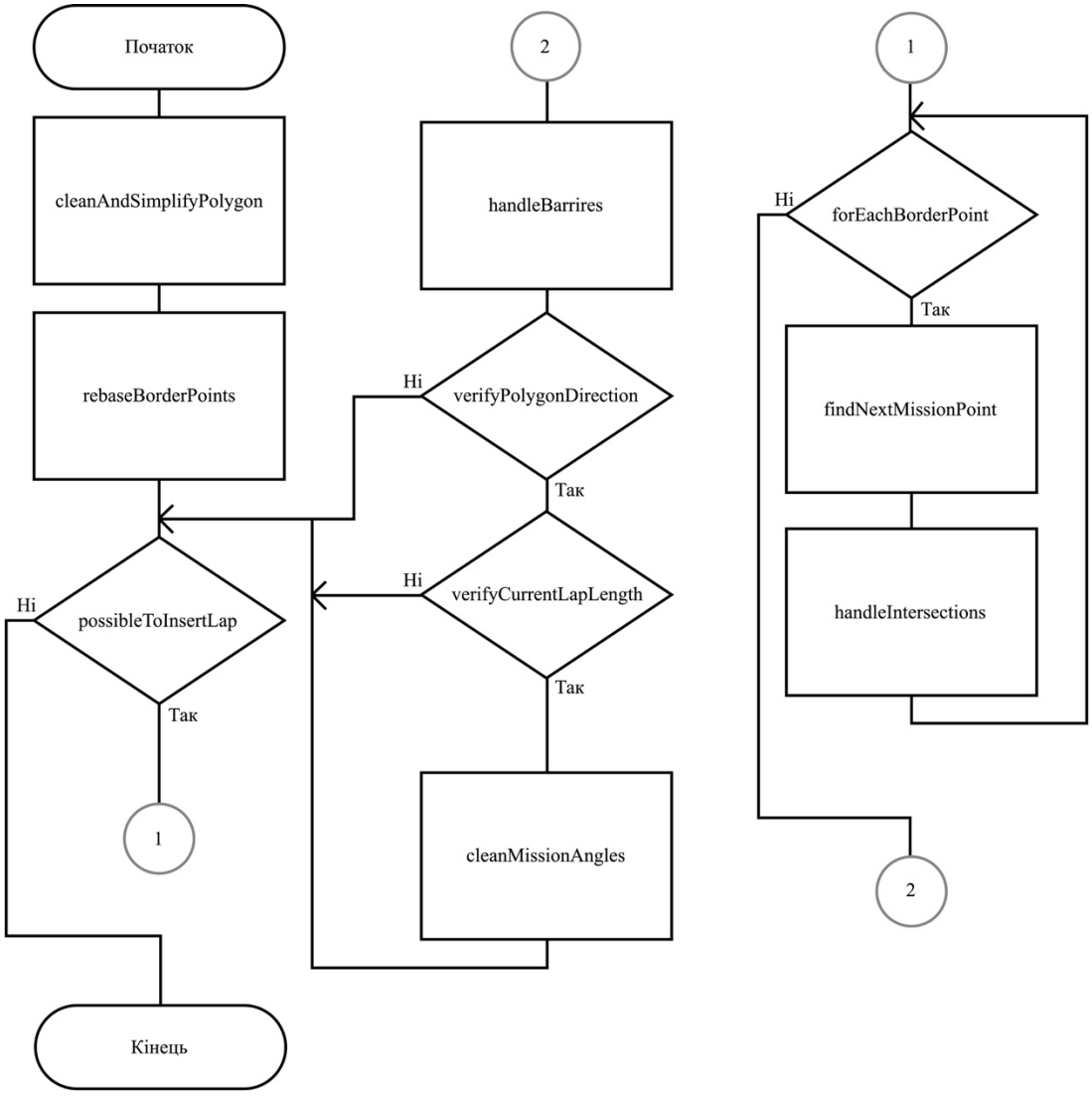


Рис. 3.6. Блок-схема алгоритму побудови маршруту

Вхідними даними для побудови є аргументи:

1. Географічні координати та висота над рівнем моря точки запуску БПЛА (home point).
2. Радіус точки маршруту (ширина відступу між проходами).
3. Полігон, що описує область, в якій треба побудувати маршрут.

4. Критичний кут повороту, при якому БПЛА безпечно здійснює поворот без звалювання.
5. Хеш-мапа, що описує перешкоди в заданій області.

Деталізуємо кожний крок алгоритму:

1. Здійснюється валідація вхідних аргументів.
2. cleanAndSimplifyPolygon. Здійснюється перевірка, чи полігон не має само-перетинів. Якщо такі існують, алгоритм завершується помилкою. Якщо форма полігону опукла та не перетинається, відбувається спрощення точок полігону. Якщо декілька точок полігону знаходяться поруч, буде використано першу з них. Оскільки полігон описує замкнене коло, кожну точку цього кола алгоритм розташовує проти годинникової стрілки. Це необхідно для подальших обчислень.
3. rebaseBorderPoints. Здійснюється перестановка точок полігону таким чином, щоб на першому місці стояла точка, яка найближче розташована від точки запуску БПЛА.
4. Далі відбувається перевірка, чи можливо в заданий полігон вписати коло спіралі.
5. Якщо вписати коло можливо, відбувається обхід кожної точки полігону.
6. findNextMissionPoint. Обраховуються азимути бічних сторін полігону, що при перетині формують дану точку полігону. Від цієї точки, за теоремою синусів, знаходиться точка, в середині полігону, що має однакову відстань від бічних сторін. Ця відстань дорівнює радіусу точки маршруту, яка була передана як аргумент. Ця точка додається в масив всіх отриманих точок.
7. handleIntersections. З отриманих до цього часу точок маршруту, формується лінія. Якщо лінія перетинається, то петля, яку вона утворює, вирізається, або ділиться на два полігони, що будуть оброблені окремо.

8. Кроки 5-6 повторюються для всіх точок полігону.
9. Результат 7 кроку формує новий полігон, в який можна буде вписати наступне коло.
10. handleBarrires. Новосформований полігон редагується, в залежності від наявності перешкод на його шляху, що були передані в якості аргументів. Різні типи перешкод мають різні стратегії обльоту.
11. verifyNextPolygonDirection. Відбувається перевірка, чи новосформований полігон зберіг положення точок проти годинникової стрілки. Якщо положення не збережене, алгоритм завершується.
12. verifyCurrentLapLength. Якщо площа новосформованого полігону більша, за площу полігону, на основі якого будувався новий, алгоритм завершується.
13. cleanMissionAngles. Кожен кут новосформованого полігону порівнюється з критичним, відповідно до заданих параметрів БПЛА. Якщо кут гостріший, сторони полігону редагуються.
14. Точки новосформованого полігону зберігаються.
15. Алгоритм повертається до перевірки умови на 3 кроці алгоритму.
16. Якщо умова 3 не виконується, значить маршрут побудовано.
17. Для кожної точки маршруту знаходиться відповідна відносна висота від точки запуску, що передана, як параметр на початку алгоритму. Вся лінія, яку описує маршрут, кадрується за вказаним кроком певної довжини та перевіряється на наявність перепадів висоти. Якщо такі є, додаються додаткові точки маршруту. Якщо дві сусідні точки маршруту мають значну різницю у висоті, додаються проміжні точки.
18. Отримані точки конвертуються до формату GeoJSON.

4. АНАЛІЗ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

4.1. Сценарії використання додатку

Основні сценарії використання додатку розглянуто у табл. 4.1-4.4.

Таблиця 4.1

Сценарій використання 1

Номер	1
Назва	Визначення точки старту на мапі
Опис	Перед тим як почати роботу з додатком, користувач має відшукати своє географічне положення на мапі.
Актори	Оператор БПЛА
Частота користування	Після кожного запуску додатку.
Передумови	1. Користувач встановив та запустив додаток. 2. Завантажилась географічна мапа з центром у точці перетину Гринвіцького меридіану та екватору. 3. Додаток отримав доступ до GPS модуля: а) користувач приєднав БПЛА до ПК через USB; б) ПК на якому запущений додаток містить GPS модуль.
Постумови	Вибраний спосіб визначення місцезнаходження збережеться при подальшому використанні
Передбачувана послідовність дій	1. Користувач переходить у панель налаштування 2. Користувач обирає спосіб визначення місцезнаходження: 2.1. Якщо виконана передумова 3а – місцезнаходження задає БПЛА; 2.2. Якщо виконана передумова 3б – місцезнаходження задає ПК на якому запущений додаток. 3. Якщо жодна з передумов з пункту 3 не виконана – користувач має можливість ввести координати точки старту вручну в панелі налаштувань. Доки це не буде зроблено, пошук місцезнаходження буде заблокованим.

Продовження табл. 4.1

Передбачувана послідовність дій	4. Користувач переходить у панель редагування маршруту 5. Користувач натискає на кнопку визначення місцезнаходження 6. Центр географічної мапи в редакторі тепер вказує на задане місцезнаходження
Альтернативні розвитки	Користувач може вручну перетягнути точку старту на мапі в бажане місце

Таблиця 4.2

Сценарій використання 2

Номер	2
Назва	Автоматична побудова маршруту
Опис	Побудова маршруту для БПЛА за вказаними параметрами
Актори	Оператор БПЛА
Частота користування	За потребою користувача
Передумови	1. Користувач встановив та запустив додаток. 2. Користувач визначив місцезнаходження точки старту маршруту.
Постумови	Побудований маршрут відновиться після виходу з додатку
Передбачувана послідовність дій	1. Користувач задає всі необхідні параметри побудови маршруту. 2. Користувач вказує полігон, що описує бажану зону, в якій необхідно побудувати маршрут. 3. Користувач задає всі перешкоди, які знаходяться у вказаному полігоні. 4. Користувач натискає на кнопку “Build”. 5. В заданій області з’являються точки маршруту доступні для редагування.
Альтернативні розвитки	Якщо вказано невалідні дані, побудова маршруту відміняється.

Таблиця 4.3

Сценарій використання 3

Номер	3
Назва	Географічне відображення рельєфу
Опис	Відображення зміни висоти ландшафту в заданій області відносно висоти точки старту
Актори	Оператор БПЛА
Частота користування	За потребою користувача
Передумови	1. Користувач встановив та запустив додаток. 2. Користувач визначив місцезнаходження точки старту маршруту.
Постумови	Відсутні
Передбачувана послідовність дій	1. Користувач вказує полігон, що описує бажану зону, в якій необхідно визначити рельєф 2. Користувач натискає на кнопку “Elevation” 3. Задана область зафарбовується маленькими квадратами. Якщо висота зони більша за висоту точки старту – колір квадрату буде ближчим до червоного, якщо менша – колір буде ближче до зеленого.
Альтернативні розвитки	Якщо вказано невалідні дані, визначення рельєфу відміняється.

Таблиця 4.4

Сценарій використання 4

Номер	4
Назва	Завантаження маршруту БПЛА
Опис	Завантаження маршруту в пам’ять автопілоту БПЛА за протоколом MAVLink.
Актори	Оператор БПЛА

Продовження табл. 4.4

Частота користування	За потребою користувача
Передумови	1. Користувач встановив та запустив додаток. 2. Користувач побудував маршрут 3. Користувач приєднав БПЛА до ПК через USB
Постумови	Щоб записати наступний маршрут, необхідно видалити попередній.
Передбачувана послідовність дій	1. Користувач переходить у панель налаштування 2. Користувач вибирає USB порт за яким приєднаний БПЛА. 3. Користувач натискає кнопку “Connect” 4. Користувач очікує з’єднання. 5. Користувач переходить на панель редагування маршруту 6. Користувач натискає кнопку “Upload Mission” 7. Маршрут завантажений в БПЛА
Альтернативні розвитки	При втраті зв’язку з БПЛА під час будь-якого кроку запис маршруту відміняється.

4.2. Дизайн інтерфейсу додатку

Інтерфейс являє собою вікно десктоп-додатку. Розміри вікна за замовчуванням мають ширину 800 та висоту 600. Розміри можуть змінюватись користувачем. Всі компоненти інтерфейсу виконані в дизайні Google Material Design. Зовнішній вигляд головного екрану додатку можна побачити на рис. 4.1.

Додаток має три панелі, що формують основне меню:

1) панель редактору маршруту (1a);
2) панель налаштувань (1b);
3) панель профілю користувача (1c).

Панель профілю користувача включає в себе налаштування особистих даних користувача, налаштування синхронізації збережених маршрутів.

Рис. 4.1. Головний екран додатку

Панель налаштувань включає системні налаштування додатку. Сюди входять:

– параметри зовнішнього виду мапи;
– параметри елементів управління мапою;
– налаштування провайдерів географічної мапи;
– параметри підключення до автопілоту БПЛА;
– параметри потоку даних отриманих з БПЛА;
– параметри автоматичної побудови маршруту;
– параметри стратегій обльоту перешкод.

Рис. 4.2. Приклад побудови маршруту з перешкодою “дерево”

Панель редактору маршруту є найбільш функціональною. На тлі всього додатку розміщена географічна мапа.

В зоні 2, відповідно до рис. 4.1, розташовані елементи управління мапою. Кнопка 2a відповідає за вибір провайдеру географічної мапи. При зміні провайдеру ресурси мапи змінюються. Елемент управління 2b відповідає за масштаб мапи. Користувач може збільшувати чи зменшувати його. Елемент управління 3c відповідає за перенесення центру мапи до місця фактичного місцезнаходження БПЛА чи оператора БПЛА відповідно до налаштувань. В зоні 3 розташована панель, що сповіщає користувача про фактичні географічні координати та висоту точку, на яку вказує курсор:

– поле lat – географічна широта;
– поле lng – географічна довгота;
– поле alt – висота точки відносно рівня моря.

В зоні 4 знаходяться інструменти для взаємодії з мапою. Зона 4a визначає режим взаємодії з доступних:

– визначення рельєфу;
– автоматична побудова маршруту;
– ручне редагування маршруту.

Кожен режим взаємодії має власні інструменти. Зона 4b визначає активний інструмент взаємодії з мапою. При виборі якогось інструменту, вікно ділиться навпіл, та з’являється додаткова інформація та параметри використання інструменту. За номером 6 на мапі розташована поточна точка старту для БПЛА. Відносно цієї точки автопілот БПЛА вираховує свою висоту. За номером 7 на мапі розташоване фактичне місце розташування оператора БПЛА. За номером 8 на мапі позначено приклад побудованого маршруту.

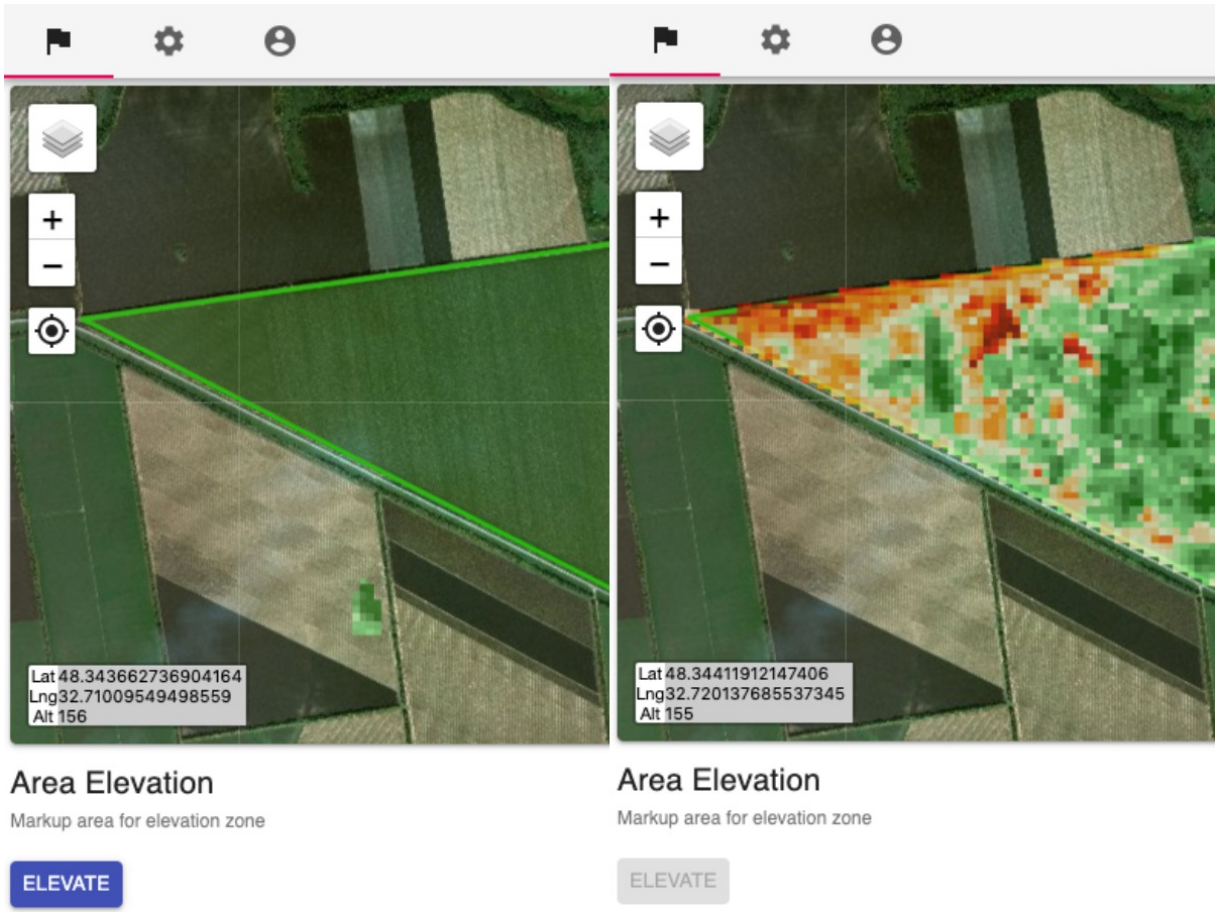
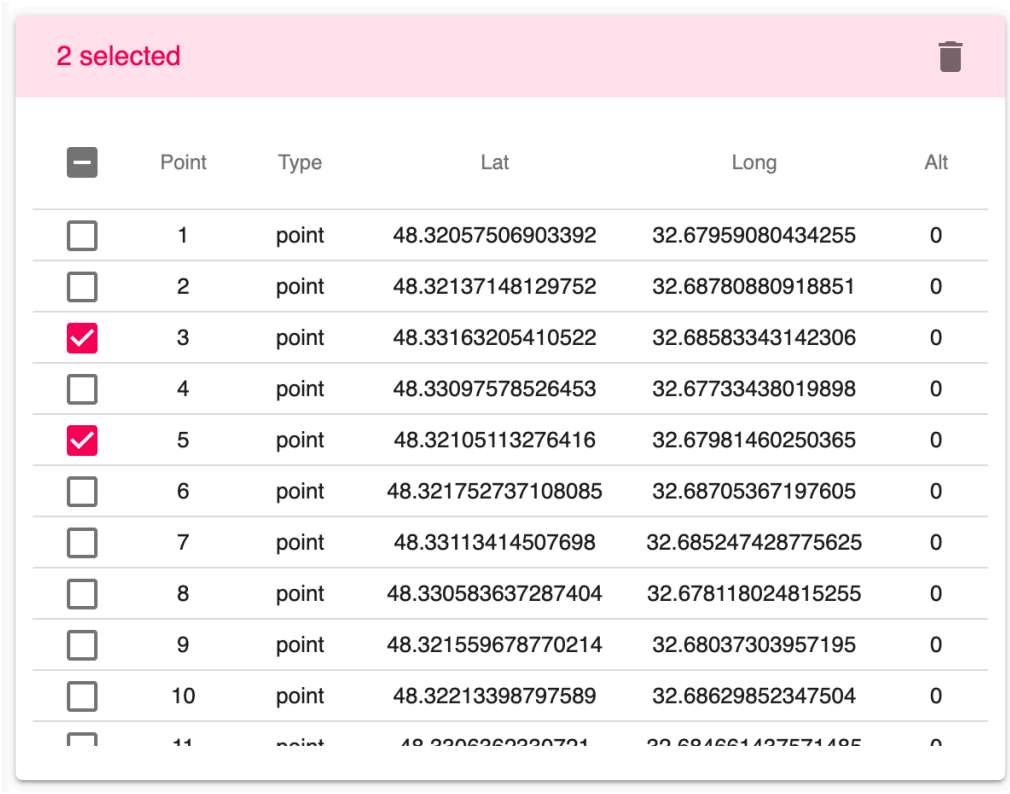


Рис. 4.3. Приклад графічного зображення рельєфу

На рис. 4.2 та рис. 4.3 показано, як виглядає результат роботи основних алгоритмів. На рис. 4.2 зображено результат побудови маршруту з обходом перешкоди типу “дерево”. Зліва показано виділену зону, справа – результуючий маршрут. На рис. 4.3 зображено графічне зображення рельєфу. ”. Зліва показано виділену зону, справа – результуючу зону.

В результаті автоматичної побудови маршруту БПЛА, кожна точка маршруту заноситься в таблицю (рис 4.5). В цій таблиці користувач може змінити параметри точок вручну.



Point	Type	Lat	Long	Alt
1	point	48.32057506903392	32.67959080434255	0
2	point	48.32137148129752	32.68780880918851	0
3	point	48.33163205410522	32.68583343142306	0
4	point	48.33097578526453	32.67733438019898	0
5	point	48.32105113276416	32.67981460250365	0
6	point	48.321752737108085	32.68705367197605	0
7	point	48.33113414507698	32.685247428775625	0
8	point	48.330583637287404	32.678118024815255	0
9	point	48.321559678770214	32.68037303957195	0
10	point	48.32213398797589	32.68629852347504	0

Рис. 4.5. Таблиця з параметрами точок маршруту БПЛА

4.3. Рекомендації щодо подальшого вдосконалення

Тема автоматизації побудови маршрутів для БПЛА в аграрній сфері досить комплексна. Багато компонентів реалізованого ПЗ можуть бути вдосконаленими, що в свою чергу зможе пришвидшити процес побудови маршруту, або зробити побудову більш універсальною. Проаналізувавши реалізовані компоненти ПЗ після тестового використання прототипу

додатку в реальних умовах та опитування зацікавлених сторін, було зібрано рекомендації щодо вдосконалення пропонованого рішення:

1. В прототипі для комунікації системи з БПЛА використовується протокол MAVLink в чистому вигляді. Протокол допускає вразливості щодо безпеки. Вразливий протокол MAVLink може бути захищений за допомогою механізму шифрування, що має реалізуватися у вже існуючому коді протоколу. Проте в наш час немає офіційних рішень, а це значить що для забезпечення безпеки, необхідно змінювати код автопілоту. Для певного сегменту користувачів додатку, ця опція може стати ключовою.
2. Для побудови маршруту в заданій області, додаток вимагає від користувача вручну задати зону в якій необхідно побудувати маршрут та інші параметри побудови. Так як у 95 % випадків, ця зона описує периметр (габарити) всього поля, що на географічній мапі мають значний контраст. Так як сусідні поля з різними засадженими культурами відрізняються за кольором, або периметр поля збігається з лісосмугою, то цілком можливо реалізувати комп’ютерне бачення, що дало б змогу автоматично знаходити габарити поля. Тоді побудова маршруту зведеться до одного кліку.
3. Здебільшого поля, які підлягають обробці, поділені на паї. Цей нюанс завдає певних труднощів при визначенні зони побудови маршруту. Якщо шматок поля не повинен підлягати обробці, при запропонованій реалізації, неможливо побудувати маршрут з врахуванням цього фактору.
4. Один з користувачів прототипу реалізованого додатку зауважив, що дуже корисною була б реалізація можливості колективного редагування маршруту. Ця можливість передбачає синхронізацію одного й того ж маршруту одразу на декількох пристроях в режимі реального часу. Застосування цього режиму було максимально

доцільним при колективному редагуванні маршруту оператором БПЛА та агрономом.

5. Кожен автопілот має власну кількість пам’яті в запам’ятовуючому пристрої. Це означає, що кожен автопілот має граничну кількість точок маршруту, які він може записати. Зазвичай це значення в діапазоні від 200 до 256 точок. Після записування наступної точки автопілот повертає помилку MAV_MISSION_NO_SPACE. Рекомендовано враховувати цей показник, щоб запобігти помилок на етапі завантаження маршруту в БПЛА.

ВИСНОВКИ

Головною метою реалізації цього дипломного проекту було вирішення проблеми реального бізнесу, а саме автоматизувати, пришвидшити та оптимізувати процес побудови маршруту для БПЛА в аграрній сфері, яка виникає при точному землеробстві для економії сприятливого для польотів часу та збільшення прибутків.

Перед тим, як розпочати виконання проекту, було проведене опитування кінцевих користувачів продукту – професійних операторів БПЛА, які надають послуги внесення ЗЗР на території України. В результаті опитування було виявлено, що оператори БПЛА часто нездатні виконати замовлення через низку основних проблем:

- важкість та монотонність проектування складних маршрутів;
- значний вплив умов навколишнього середовища на процес побудови;
- втрата сприятливого для польотів часу;
- складність та надлишковість наявних рішень;
- відсутність автоматизації рутинних процесів;
- неможливість зручно зберігати маршрути та ділитися ними;
- відсутність кросплатформенних рішень.

Реалізований додаток надає рішення для всіх вище описаних проблем. Використання розробленого додатку дозволить значно пришвидшити процес побудови маршруту для БПЛА та збільшити ефективність роботи користувачів.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Термальна колона [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BA%D0%BE%D0%BB%D0%BE%D0%BD%D0%B0
2. Точне землеробство [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A2%D0%BE%D1%87%D0%BD%D0%B5_%D0%B7%D0%B5%D0%BC%D0%BB%D0%B5%D1%80%D0%BE%D0%B1%D1%81%D1%82%D0%B2%D0%BE
3. Trichogramma [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/trichogramma
4. Mavlink [Електронний ресурс] – Режим доступу: https://mavlink.io/en/
5. Поступовий веб-застосунок [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D1%82%D1%83%D0%BF%D0%BE%D0%B2%D0%B8%D0%B9_%D0%B2%D0%B5%D0%B1-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA
6. Mission Planer [Електронний ресурс] – Режим доступу: http://ardupilot.org/planner/
7. QGroundControl [Електронний ресурс] – Режим доступу до ресурсу: http://ardupilot.org/planner/QGroundControl
8. DroidPlanner – [Електронний ресурс] – Режим доступу: https://github.com/DroidPlanner/Tower
9. Безпілотники – сучасний інструмент для аграрія [Електронний ресурс] – Режим доступу: https://agropro.club/articles/bezpilotniki-suchasnij-instrument-dlya-agrariya/.
10. The economic impact of unmanned aircraft systems integration in the United States [Електронний ресурс] – Режим доступу:

https://higherlogicdownload.s3.amazonaws.com/AUVSI/958c920a-7f9b-4ad2-9807-f9a4e95d1ef1/UploadedImages/New_Economic%20Report%202013%20Full.pdf

11. Drone.ua [Електронний ресурс] – Режим доступу: http://drone.ua/resheniya-dlya-apk/
12. Defense and Consumer Drone Makers Set Their Line of Sight on the Commercial sUAS Market as Growth Soars [Електронний ресурс] – Режим доступу: https://www.abiresearch.com/press/defense-and-consumer-drone-makers-set-their-line-s/
13. Цифровий розрив: чому українські села залишаються без інтернету [Електронний ресурс] – Режим доступу: https://biz.liga.net/all/telekom/article/tsifrovoy-razryv-pochemu-ukrainski e-sela-ostayutsya-bez-interneta
14. Безпілотний авіаційний комплекс для вирішення задач біологічного захисту росли [Електронний ресурс] – Режим доступу: http://mgsys.kpi.ua/article/download/158278/161455
15. Kernel – лідер українського аграрного експорту [Електронний ресурс] – Режим доступу: https://www.kernel.ua/ua/.
16. Electron.js [Електронний ресурс] – Режим доступу: https://electronjs.org/
17. Cocoa API [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Cocoa_(API)
18. OpenGL [Електронний ресурс] – Режим доступу: https://www.opengl.org/
19. Chromium [Електронний ресурс] – Режим доступу: https://www.chromium.org/Home
20. Vue.js [Електронний ресурс] – Режим доступу: https://vuejs.org/
21. React.js [Електронний ресурс] – Режим доступу: https://reactjs.org/
22. Flux [Електронний ресурс] – Режим доступу: https://facebook.github.io/flux/docs/in-depth-overview.html
23. Google Maps API [Електронний ресурс] – Режим доступу: https://developers.google.com/maps/documentation/

24. MapBox API [Електронний ресурс] – Режим доступу: https://docs.mapbox.com/mapbox-gl-js/api/
25. Bing Maps Dev Center [Електронний ресурс] – Режим доступу: https://www.bingmapsportal.com/
26. OpenStreetMaps [Електронний ресурс] – Режим доступу: https://www.openstreetmap.org/#map=5/51.500/-0.100
27. GeoJSON [Електронний ресурс] – Режим доступу: https://geojson.org/
28. Turf.js [Електронний ресурс] – Режим доступу: https://turfjs.org/
29. Redux.js [Електронний ресурс] – Режим доступу: https://redux.js.org/
30. The GeoJSON Specification [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc7946

ДОДАТКИ

Додаток 1

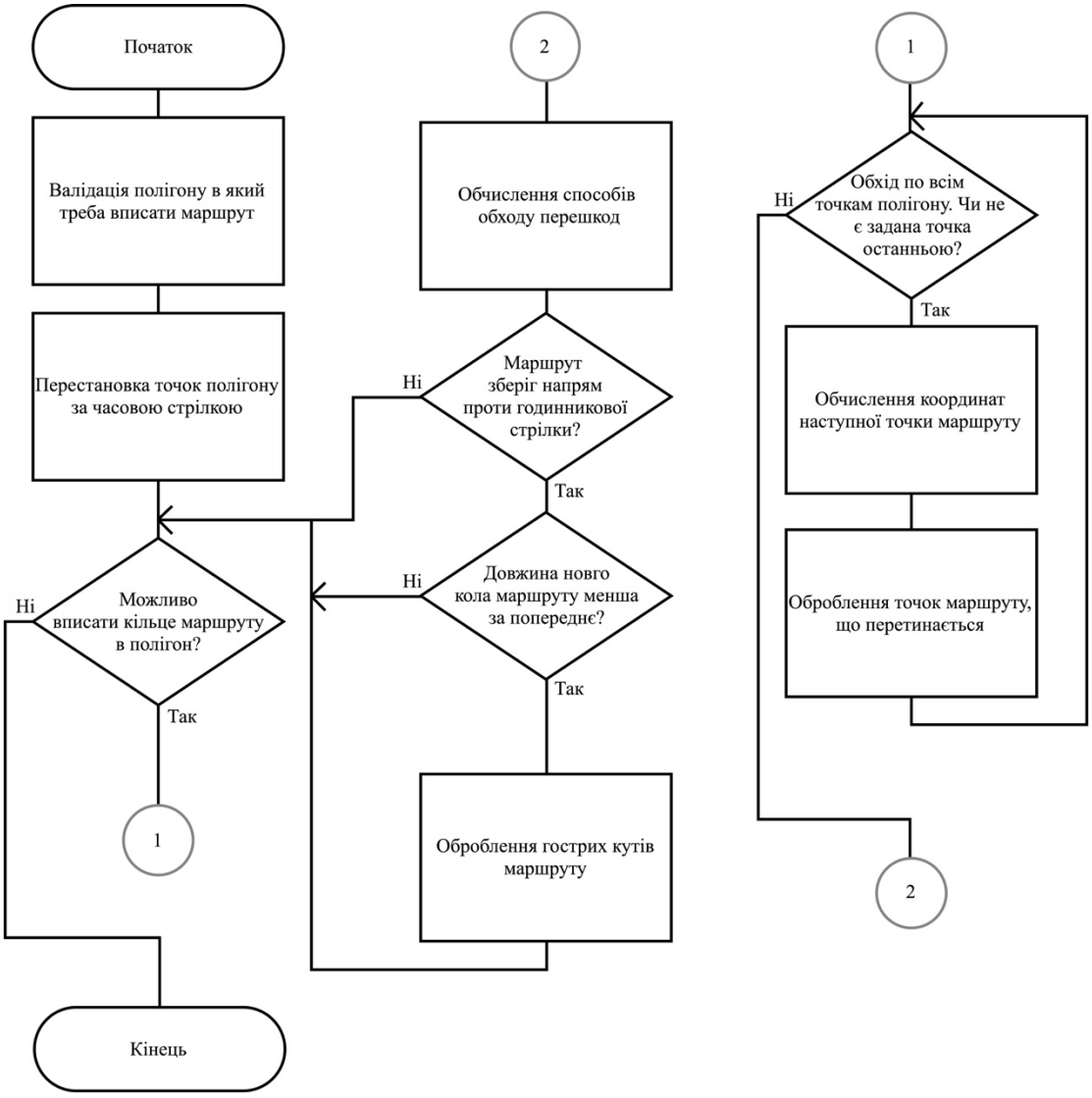
Копії графічних матеріалів

Додаток 2

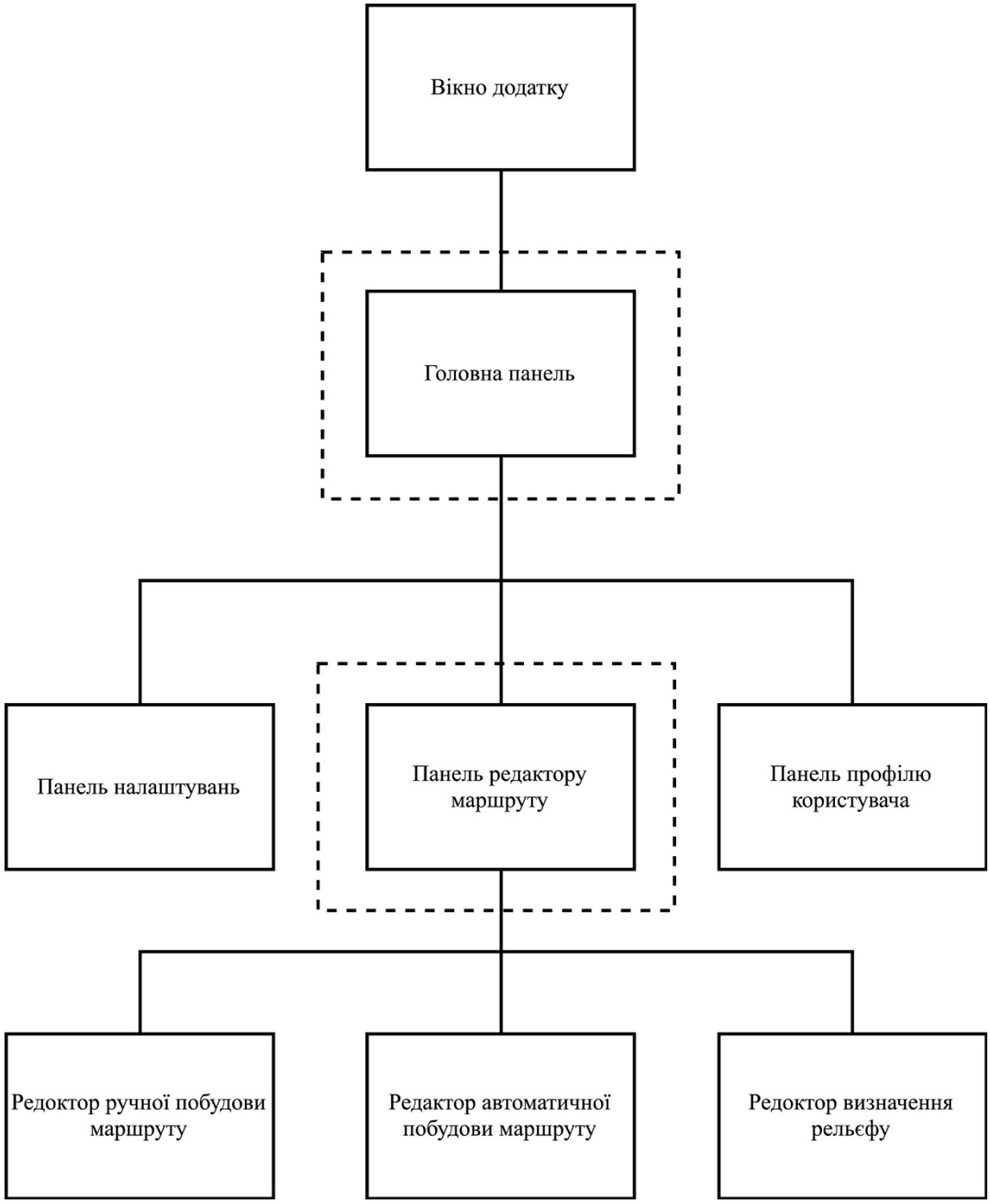
Лістинг модуля взаємодії з БПЛА

Додаток 3

Копія презентації

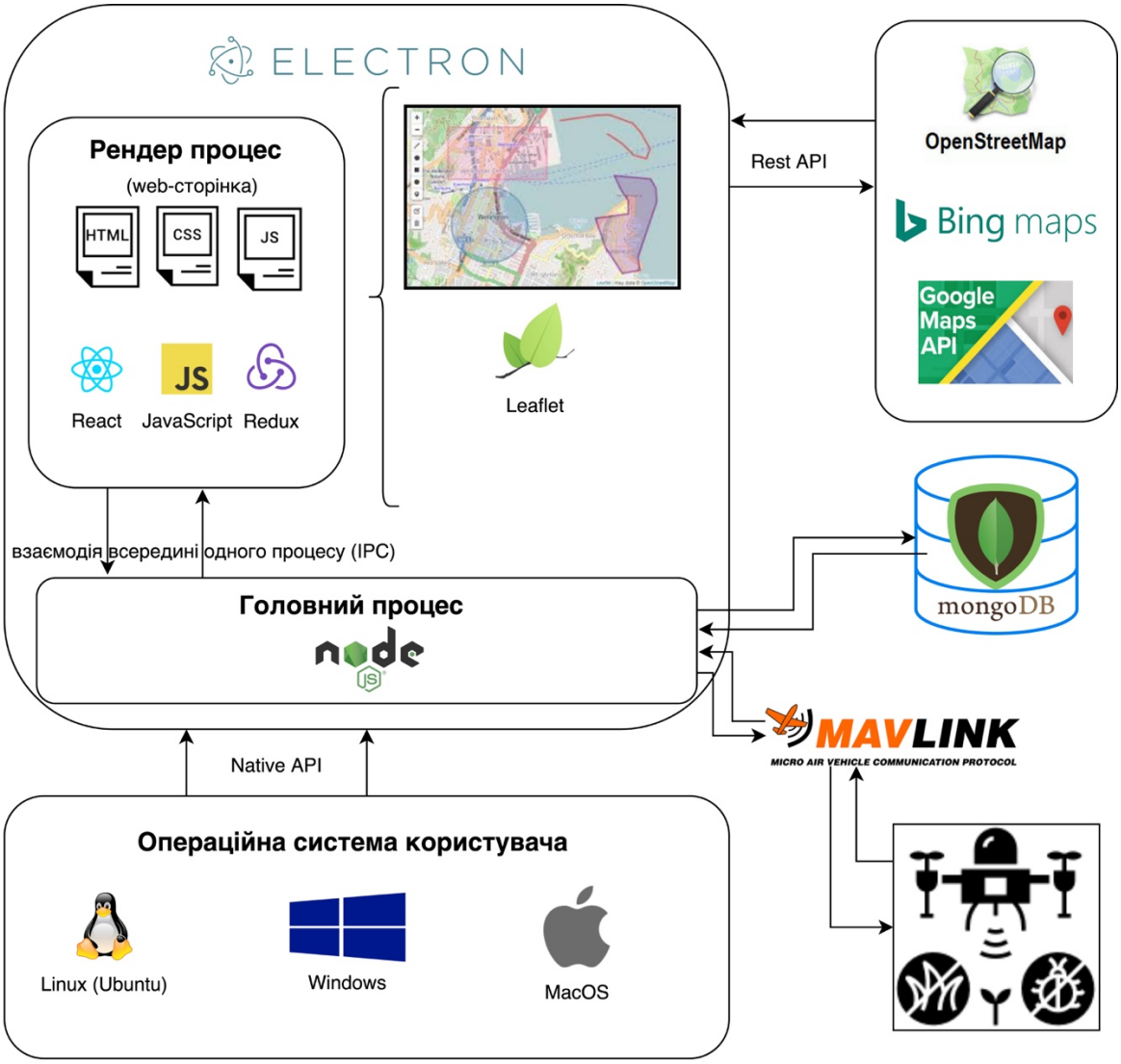


ДП.045490-06-99
Програмні засоби для автоматизованої побудови маршрутів БПЛА в аграрній сфері. Побудова маршруту. Блок-схема алгоритму



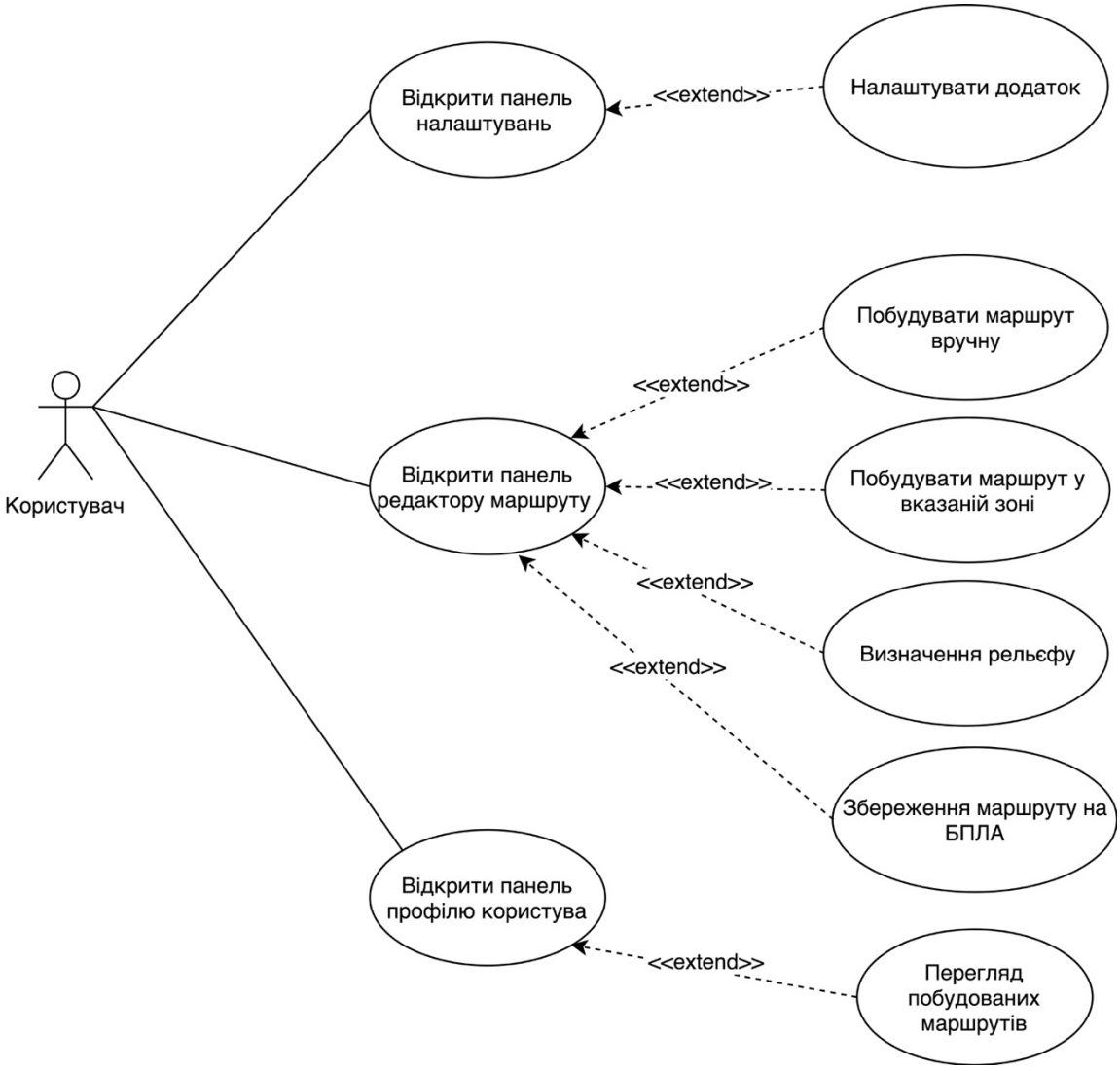
ДП.045490-07-99
Програмні засоби для автоматизованої побудови маршрутів БПЛА в аграрній сфері. Інтерфейс користувача. Структура взаємодії представлень

Архітектура додатку для побудови маршрутів БПЛА



Чеботарьов К.С., група КП-52

Діаграма варіантів використання додатку для побудови маршрутів БПЛА



Чеботарьов К.С., група КП-52

```
const SerialPort = require('serialport');
const mavLink = require('mavlink');
const { EventEmitter } = require('events');

const { APIEvents, MAV_DATA_STREAM, APM_MODE_STR } =
require('./constants');
const mavHeader = require('./mavHeader.js');

const SYSTEM_ID = 1;
const COMPONENT_ID = 1;
const USB_BAUD_RATE = 115200;
const MISSION_TRANS_STATE_IDLE = 0;
const MISSION_TRANS_STATE_TRANS = 1;
const DOWNLOADING_NUM_AT_A_TIME = 2;

class WayPoint {
    constructor(lat, lng, num, yaw, altitude, spd, hld_time, do_action) {
        this.lat = lat;
        this.lng = lng;
        this.yaw = yaw;
        this.index = num;
        this.alt = altitude;
        this.speed = spd;
        this.hold_time = hld_time;
        this.action = do_action;
    }
}

function getMAVLinkConnectionApi() {
    const emitter = new EventEmitter();
    const emitterOriginal = emitter.emit;
    emitter.emit = function(a, b) {
        console.error(`${a}, ${JSON.stringify(b, '',
2)}`.replace(/\s{1,}/g, ' '));
        emitterOriginal.apply(emitter, arguments);
    };

    const state = {
        attitude: {
            time_boot_ms: 0,
            roll: 0,
            pitch: 0,
            yaw: 0,
            rollspeed: 0,
            pitchspeed: 0,
            yawspeed: 0,
        },
        baseStatus: {
            autopilot: 0,
            base_mode: 0,
            custom_mode: 0,
            mavlink_version: undefined,
            system_is_armed: false,
            system_status: 0,
            type: 0,
        },
        connection: {
            lostTime: 0,
            lost: true,
            path: '',
        },
        dataStream: {
            period: 20000,
            rates_extended_status: 5,
```

```
        rates_extra1: 5,
        rates_extra2: 0,
        rates_extra3: 0,
        rates_position: 2,
        rates_raw_controller: 1,
        rates_raw_sensors: 1,
        rates_rc_channels: 2,
    },
    globalPosition: {
        lat: 0,
        lon: 0,
        alt: 0,
        relative_alt: 0,
        vx: 0,
        xy: 0,
        vz: 0,
        hdg: 0,
    },
    gps: {
        time_usec: 0,
        lat: 0,
        lon: 0,
        alt: 0,
        eph: 9999,
        epv: 65535,
        vel: 0,
        cog: 0,
        fix_type: 0,
        satellites_visible: 0,
    },
    sysStatus: {},
    heartbeat: {
        grace_interval: 5000,
        lastPingTimeStamp: 0,
    },
    map: {
        maplist: [],
    },
    mission: {
        uploadingMission: [],
        state: MISSION_TRANS_STATE_IDLE,
        downloadingLength: 0,
        downloadingMission: {},
    },
    // mission: {
    //     saved_gcs_mission_list: [],
    //     trans_state: ,
    //     length: 0,
    //     list: [],
    //     current_seq: 0,
    //     current_trans_seq: -1,
    //     trans_start_time: 0,
    //     trans_end_time: 0,
    // },
};

const timers = {
    updateHeartbeat: null,
    heartbeat: null,
    dataStream: null,
};

let mav_port = null;
```

```
    let mav_parser = null;

    let mag_cal_datastream = false;

    const _connectUSBSerialPort = (connection_path) => {
        if (mav_port) {
            emitter.emit(
                APIEvents.STATUS_TEXT,
                `Warning: USB port ${
                    state.connection.path
                } is already connected or is connecting now`,
            );
            return;
        }

        state.connection.path = connection_path;

        mav_port = new SerialPort(connection_path, {
            baudRate: USB_BAUD_RATE,
        });

        mav_port.open(_openSerialPort);
        mav_port.on('error', disconnectSerialPort);
        mav_port.on('close', disconnectSerialPort);
    };

    const _heartbeatCb = (message, fields) => {
        state.heartbeat.lastPingTimeStamp = Date.now();

        const currentlyArmed = Boolean(
            fields.base_mode &
mavHeader.MAV_MODE_FLAG_DECODE_POSITION_SAFETY,
        );

        if (currentlyArmed !== state.baseStatus.system_is_armed) {
            this.emitter.emit(APIEvents.STATUS_TEXT, currentlyArmed ?
'arming' : 'disarming');
        }

        state.baseStatus = {
            system_is_armed: currentlyArmed,
            ...fields,
        };

        emitter.emit(APIEvents.HEARTBEAT, state.baseStatus);
    };

    const _initMissionUploading = (missionLength) => {
        if (!isFinite(missionLength)) {
            throw new Error('Invalid mission length');
        }
        mav_parser.createMessage(
            'MISSION_COUNT',
            {
                count: missionLength,
                target_system: SYSTEM_ID,
                target_component: 1,
            },
            _sendPortMessage,
        );
    };

    const _initUpdateData = () => {
        // timers.heartbeat = setInterval(_sendHeartbeat, 1000);
```

```
        // timers.dataStream = setInterval(_sendDefaultDataStream,
state.dataStream.period);
        // timers.updateHeartbeat = setInterval(_updateHeartbeat, 500);

        state.heartbeat.lastPingTimeStamp = 0;
        state.connection.lost = true;
        state.connection.lostTime = 0;
        // state.mission.current_seq = 0;
        // state.mission.list = [];
        // state.mission.length = 0;
        // state.mission.trans_state = MISSION_TRANS_STATE_IDLE;
        // state.mission.current_trans_seq = -1;
        // state.mission.trans_start_time = 0;
        // state.mission.trans_end_time = 0;
    };

    const _missionUploadWaypoint = (fields) => {
        if (!mav_parser) {
            emitter.emit(APIEvents.STATUS_TEXT, `ERROR: MAV_PARSER is not
set`);
            return;
        }

        mav_parser.createMessage(
            'MISSION_ITEM',
            {
                autocontinue: 1,
                /*'command': mavheader.MAV_CMD_NAV_WAYPOINT,*/
                command: fields.command,
                current: 0,
                // Global (WGS84) coordinate frame + altitude relative to
the home position.
                // First value / x: latitude,
                // second value / y: longitude,
                // third value / z: positive altitude with 0 being at the
altitude of the home location.
                frame: 3,
                param1: fields.param1, //hold_time
                param2: fields.param2, //radius
                param3: fields.param3, //param3
                param4: fields.param4, //yaw
                seq: fields.seq,
                target_component: COMPONENT_ID,
                target_system: SYSTEM_ID,
                x: fields.x, //lat
                y: fields.y, //lon
                z: fields.z, //alt
            },
            _sendPortMessage,
        );
    };

    const _missionDownloadWaypoint = (seq) => {
        if (!mav_parser) {
            emitter.emit(APIEvents.STATUS_TEXT, `ERROR: MAV_PARSER is not
set`);
            return;
        }

        mav_parser.createMessage(
            'MISSION_REQUEST',
            {
                seq: seq,
                target_system: SYSTEM_ID,
```

```
                target_component: COMPONENT_ID,
            },
            _sendPortMessage,
        );
    };

    const _openSerialPort = () => {
        if (mav_parser) {
            emitter.emit(APIEvents.STATUS_TEXT, `Warning: mav_parser is
already exist`);
            return;
        }

        mav_parser = new mavLink(SYSTEM_ID, COMPONENT_ID);

        // When mav link is ready, assign some listeners
        mav_parser.on('ready', () => {
            mav_port.on('data', (data) => {
                mav_parser.parse(data);
            });

            mav_port.on('open', () => {
                setTimeout(() => {
                    emitter.emit(APIEvents.CONNECTED,
state.connection.path);
                }, 1000);
            });

            mav_parser.on('message', function(message) {
                if (message.id === 26) {
                    emitter.emit(APIEvents.STATUS_TEXT, message);
                }
            });

            mav_parser.on('HEARTBEAT', _heartbeatCb);

            mav_parser.on('SYS_STATUS', (message, fields) => {
                state.sysStatus = fields;
                emitter.emit(APIEvents.SYS_STATUS, state.sysStatus);
            });

            mav_parser.on('ATTITUDE', (message, fields) => {
                state.attitude = fields;
                emitter.emit(APIEvents.ATTITUDE, fields);
            });

            mav_parser.on('LOCAL_POSITION', (message, fields) => {
                emitter.emit(APIEvents.STATUS_TEXT, fields);
            });

            mav_parser.on('GLOBAL_POSITION_INT', (message, fields) => {
                state.globalPosition = fields;
                emitter.emit(APIEvents.GLOBAL_POSITION, fields);
            });

            mav_parser.on('GPS_RAW_INT', (message, fields) => {
                state.gps = fields;
                emitter.emit(APIEvents.GPS_RAW_INT, fields);
            });

            mav_parser.on('COMMAND_ACK', (message, fields) => {
                emitter.emit(APIEvents.COMMAND_ACK, fields);
            });
```

```
            mav_parser.on('RADIO', (message, fields) => {
                emitter.emit(APIEvents.RADIO, fields);
            });

            mav_parser.on('RADIO_STATUS', (message, fields) => {
                emitter.emit(APIEvents.STATUS_TEXT, fields);
            });

            mav_parser.on('PARAM_VALUE', (message, fields) => {
                emitter.emit(APIEvents.STATUS_TEXT, fields);
            });

            mav_parser.on('STATUSTEXT', (message, fields) => {
                emitter.emit(APIEvents.STATUS_TEXT, fields.text);
            });

            // mav_parser.on('MISSION_CURRENT', function(message, fields) {
            //     if (
            //         state.mission.current_seq !== fields.seq &&
            //         fields.seq < state.mission.list.length
            //     ) {
            //         if (state.mission.current_seq < fields.seq) {
            //             emitter.emit('current_cmd',
state.mission.list[fields.seq]);
            //         }
            //     }
            //     state.mission.current_seq = fields.seq;
            //     const gcs_seq = get_gcs_seq(fields.seq);
            //     const ret_fields = {
            //         seq: gcs_seq,
            //     };
            //
            //     emitter.emit(APIEvents.MISSION_CURRENT, ret_fields);
            //
            //     if (fields.seq === state.mission.list.length - 1) {
            //         emitter.emit('mission_finished');
            //     }
            // });

            mav_parser.on('MISSION_ITEM', (message, fields) => {
                const mission_index = fields.seq;
                const missionLength = state.mission.downloadingLength;

                if (mission_index < 0 || mission_index >= missionLength) {
                    emitter.emit(APIEvents.STATUS_TEXT, 'ERROR: Receive seq
not in regular range');
                    return;
                }

                emitter.emit('reading_next', mission_index);

                state.mission.downloadingMission[mission_index] = fields;

                if (mission_index + DOWNLOADING_NUM_AT_A_TIME <
missionLength) {
                    _missionDownloadWaypoint(mission_index +
DOWNLOADING_NUM_AT_A_TIME);
                }

                if (mission_index === missionLength - 1) {
                    emitter.emit(APIEvents.STATUS_TEXT, 'INFO: Mission
reading finished');
                }
            });
```

```
            mav_parser.on('MISSION_COUNT', (message, fields) => {
                state.mission.downloadingLength = fields.count;
                emitter.emit(APIEvents.STATUS_TEXT, 'Requesting all
mission');

                if (state.mission.downloadingLength <= 0) {
                    emitter.emit(APIEvents.STATUS_TEXT, 'Warning: No
mission');
                    emitter.emit('reading_completed', []);
                    return;
                }

                for (
                    let i = 0;
                    i < state.mission.downloadingLength && i <
DOWNLOADING_NUM_AT_A_TIME;
                    i++
                ) {
                    setTimeout(() => {
                        _missionDownloadWaypoint(i);
                    }, 200);
                }
            });

            mav_parser.on('MISSION_ACK', function(message, fields) {
                if (state.mission.state !== MISSION_TRANS_STATE_TRANS) {
                    emitter.emit(APIEvents.STATUS_TEXT, 'Unexpected
MISSION_ACK');
                    return;
                }

                state.mission.state = MISSION_TRANS_STATE_IDLE;
                state.gcsDate = new Date();

                if (fields.type !== 0) {
                    emitter.emit(
                        APIEvents.STATUS_TEXT,
                        `Mission uploading failed. MAV_MISSION_RESULT:
${fields.type}`,
                    );
                    state.mission.uploadingMission = [];
                } else {
                    emitter.emit(APIEvents.STATUS_TEXT, 'Mission uploading
finished successful');
                }

                emitter.emit(APIEvents.MISSION_UPLOADED,
state.mission.uploadingMission);
            });

            mav_parser.on('MISSION_REQUEST', (message, fields) => {
                emitter.emit(APIEvents.MISSION_REQUEST, fields);
                const mission_index = fields.seq;

                if (mission_index < 0 && mission_index >
state.mission.uploadingMission.length) {
                    emitter.emit(APIEvents.STATUS_TEXT, 'ERROR: Receive seq
not in regular range');
                    return;
                }

                if (state.mission.state === MISSION_TRANS_STATE_TRANS) {
```

```
_missionUploadWaypoint(state.mission.uploadingMission[mission_index]);
                }
            });
        });
    };

    const _sendDataStreamRate = (id, rate) => {
        if (!mav_parser) {
            emitter.emit(APIEvents.STATUS_TEXT, `ERROR: MAV_PARSER is not
set`);
            return;
        }

        mav_parser.createMessage(
            'REQUEST_DATA_STREAM',
            {
                target_system: SYSTEM_ID,
                target_component: COMPONENT_ID,
                req_stream_id: id,
                req_message_rate: rate,
                start_stop: 1,
            },
            _sendPortMessage,
        );
    };

    const _sendDefaultDataStream = () => {
        if (!mav_parser) {
            emitter.emit(APIEvents.STATUS_TEXT, `ERROR: MAV_PARSER is not
set`);
            return;
        }

        if (mag_cal_datastream) {
            const dataStream = [
                [MAV_DATA_STREAM.MAV_DATA_STREAM_RAW_SENSORS, 50],
                [MAV_DATA_STREAM.MAV_DATA_STREAM_EXTENDED_STATUS, 0],
                [MAV_DATA_STREAM.MAV_DATA_STREAM_RC_CHANNELS, 0],
                [MAV_DATA_STREAM.MAV_DATA_STREAM_RAW_CONTROLLER, 0],
                [MAV_DATA_STREAM.MAV_DATA_STREAM_POSITION, 0],
                [MAV_DATA_STREAM.MAV_DATA_STREAM_EXTRA1, 0],
                [MAV_DATA_STREAM.MAV_DATA_STREAM_EXTRA2, 0],
                [MAV_DATA_STREAM.MAV_DATA_STREAM_EXTRA3, 0],
            ];

            dataStream.forEach(([param, value]) =>
_sendDataStreamRate(param, value));
        } else {
            const dataStream = [
                // Enable IMU_RAW, GPS_RAW, GPS_STATUS packets 5
                [MAV_DATA_STREAM.MAV_DATA_STREAM_RAW_SENSORS,
state.dataStream.rates_raw_sensors],
                // Enable GPS_STATUS, CONTROL_STATUS, AUX_STATUS 5
                [
                    MAV_DATA_STREAM.MAV_DATA_STREAM_EXTENDED_STATUS,
                    state.dataStream.rates_extended_status,
                ],
                // Enable RC_CHANNELS_SCALED, RC_CHANNELS_RAW,
SERVO_OUTPUT_RAW 5
                [MAV_DATA_STREAM.MAV_DATA_STREAM_RC_CHANNELS,
state.dataStream.rates_rc_channels],
                // Enable ATTITUDE_CONTROLLER_OUTPUT,
POSITION_CONTROLLER_OUTPUT, NAV_CONTROLLER_OUTPUT 2
                [
```

```
                    MAV_DATA_STREAM.MAV_DATA_STREAM_RAW_CONTROLLER,
                    state.dataStream.rates_raw_controller,
                ],
                //     Enable LOCAL_POSITION,
GLOBAL_POSITION/GLOBAL_POSITION_INT messages 3
                [MAV_DATA_STREAM.MAV_DATA_STREAM_POSITION,
state.dataStream.rates_position],
                [MAV_DATA_STREAM.MAV_DATA_STREAM_EXTRA1,
state.dataStream.rates_extra1],
                [MAV_DATA_STREAM.MAV_DATA_STREAM_EXTRA2,
state.dataStream.rates_extra2],
                [MAV_DATA_STREAM.MAV_DATA_STREAM_EXTRA3,
state.dataStream.rates_extra3],
            ];

            dataStream.forEach(([param, value]) =>
                setTimeout(() => _sendDataStreamRate(param, value), 200),
            );
        }
    };

    const _sendPortMessage = (message) => {
        if (!mav_port) {
            emitter.emit(APIEvents.STATUS_TEXT, 'ERROR: port is not
initializded!');
            return;
        }

        if (mav_port.isOpen) {
            mav_port.write(message.buffer);
        } else {
            emitter.emit(
                APIEvents.STATUS_TEXT,
                `ERROR: Port is not open! On msg id: ${message.id}`,
            );
        }
    };

    const _sendHeartbeat = () => {
        if (!mav_parser) {
            emitter.emit(APIEvents.STATUS_TEXT, `ERROR: MAV_PARSER is not
set`);
            return;
        }

        mav_parser.createMessage(
            'HEARTBEAT',
            {
                custom_mode: 0,
                type: mavHeader.MAV_TYPE_GCS,
                autopilot: mavHeader.MAV_AUTOPILOT_INVALID,
                base_mode: mavHeader.MAV_MODE_MANUAL_ARMED,
                system_status: mavHeader.MAV_STATE_ACTIVE,
                mavlink_version: 3,
            },
            _sendPortMessage,
        );
    };

    const _updateHeartbeat = () => {
        const heartbeat_interval = Date.now() -
state.heartbeat.lastPingTimeStamp;

        // check if heartbeat time out
```

```
        if (!state.connection.lost && heartbeat_interval >
state.heartbeat.grace_interval) {
            state.connection.lost = true;

            emitter.emit(APIEvents.STATUS_TEXT, 'Connection lost');
            emitter.emit(APIEvents.CONNECTION_LOST);
        }

        if (state.connection.lost) {
            if (heartbeat_interval > state.heartbeat.grace_interval) {
                // update connection loss time
                state.connection.lostTime = heartbeat_interval;
            } else {
                // connection regained
                state.connection.lost = false;
                state.connection.lostTime = 0;
            }
        }
    };

    const connectSerialPort = (connection_path) => {
        try {
            _initUpdateData();
            _connectUSBSerialPort(connection_path);
            return true;
        } catch (e) {
            return false;
        }
    };

    const disconnectSerialPort = (error) => {
        mav_port = null;
        mav_parser = null;

        emitter.emit(
            APIEvents.SERIAL_PORT_DISCONNECTED,
            `Connection closed on ${state.connection.path} ${error ? error
: ''}`,
        );
        clearInterval(timers.heartbeat);
        clearInterval(timers.dataStream);
        clearInterval(timers.updateHeartbeat);
    };

    const getModeStr = () => {
        return APM_MODE_STR[state.baseStatus.custom_mode];
    };

    const missionInitUploadingProcess = (missionPoints) => {
        const missionLength = missionPoints.length;

        if (!missionLength) {
            emitter.emit(
                APIEvents.STATUS_TEXT,
                'ERROR: Start mission transmiting, mission length not
legal',
            );
            return;
        }

        if (state.mission.state === MISSION_TRANS_STATE_IDLE) {
            state.mission.state = MISSION_TRANS_STATE_TRANS;
        } else {
            emitter.emit(
```

```
                APIEvents.STATUS_TEXT,
                'Impossible to start transmision during another transmision
procedure',
            );
            return;
        }

        state.mission.uploadingMission = missionPoints;
        _initMissionUploading(missionLength);

        setTimeout(() => {
            if (state.mission.state !== MISSION_TRANS_STATE_IDLE) {
                emitter.emit(APIEvents.STATUS_TEXT, 'ERROR: Uploading
mission not finish');
            }
            state.mission.state = MISSION_TRANS_STATE_IDLE;
        }, 1000 * missionLength);
    };

    const missionInitDownloadingProcess = () => {
        mav_parser.createMessage(
            'MISSION_REQUEST_LIST',
            {
                target_system: SYSTEM_ID,
                target_component: 1,
            },
            _sendPortMessage,
        );

        setTimeout(() => {
            if (state.mission.downloadingLength < 0) {
                emitter.emit('status_text', 'ERROR: Mission request list
get count not legal');
                state.mission.downloadingLength = 0;
            }
        }, 30000);
    };

    const pauseSerialPort = () => {
        clearInterval(timers.heartbeat);
        clearInterval(timers.dataStream);
        clearInterval(timers.updateHeartbeat);
        mav_port.pause();
        emitter.emit(APIEvents.PAUSE);
    };

    return {
        connectSerialPort,
        disconnectSerialPort,
        emitter,
        getModeStr,
        missionInitUploadingProcess,
        missionInitDownloadingProcess,
        pauseSerialPort,
        resumeSerialPort,
        state,
    };
}
```

Додаток 3

Копія презентації

ПРОГРАМНІ ЗАСОБИ ДЛЯ АВТОМАТИЗОВАНОЇ ПОБУДОВИ МАРШРУТІВ БПЛА В АГРАРНІЙ СФЕРІ

студент Чеботарьов К.С.

ФПМ, КП-52

ОСОБЛИВОСТІ ВИКОРИСТАННЯ БПЛА В АГРАРНІЙ СФЕРІ

Умови, що ускладнюють виконання робіт:

- відсутність доступу до мережі Інтернет
- політ на низькій висоті
- політ на високій швидкості
- трав'яний покрив (неможливість використання дальномірів)
- змінний ланшафт

Умови, що впливають на можливість виконання робіт:

- час доби
- вологість повітря
- температура повітря
- швидкість та напрямок вітру

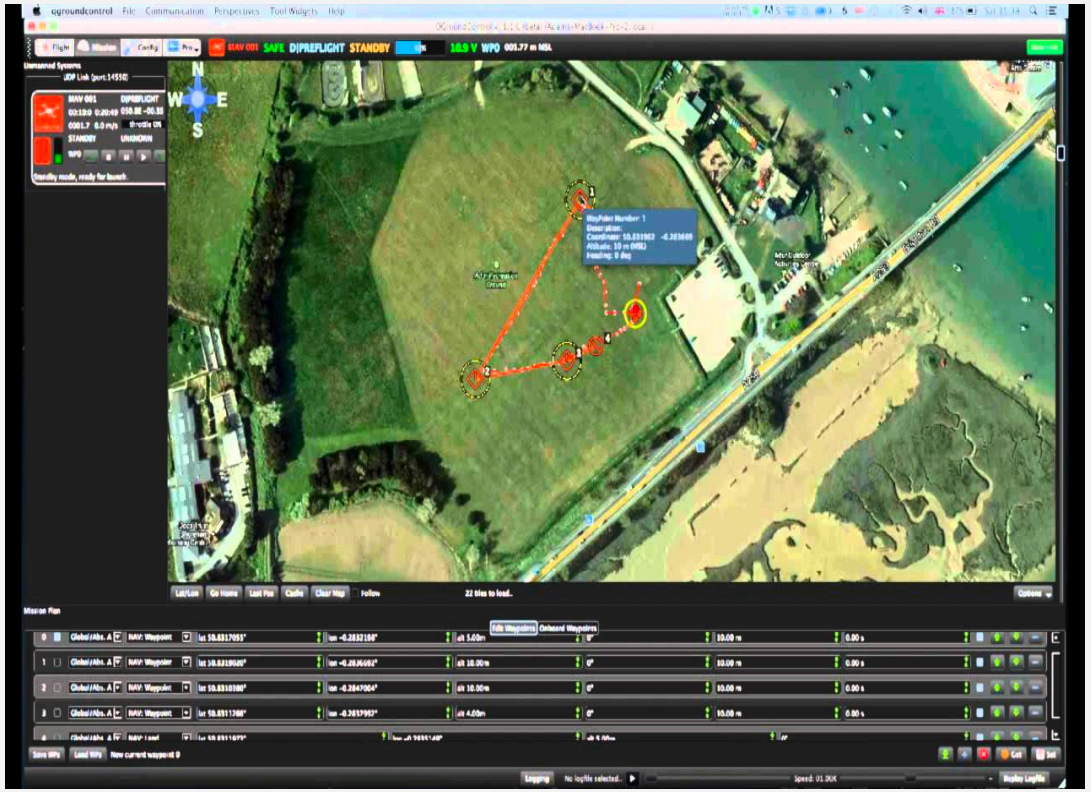
ПРОБЛЕМИ З ЯКИМИ СТИКАЮТЬСЯ ОПЕРАТОРИ БПЛА

У середньому побудова маршруту для зони площею 100 га займає від 45хв до 1,5 години через:

- відсутність автоматизації побудови
- відсутність кросплатформних рішень
- неможливість змінювати провайдер мапи
- неможливість відстеження рельєфу при побудові маршруту
- неможливість зручно зберігати маршрут

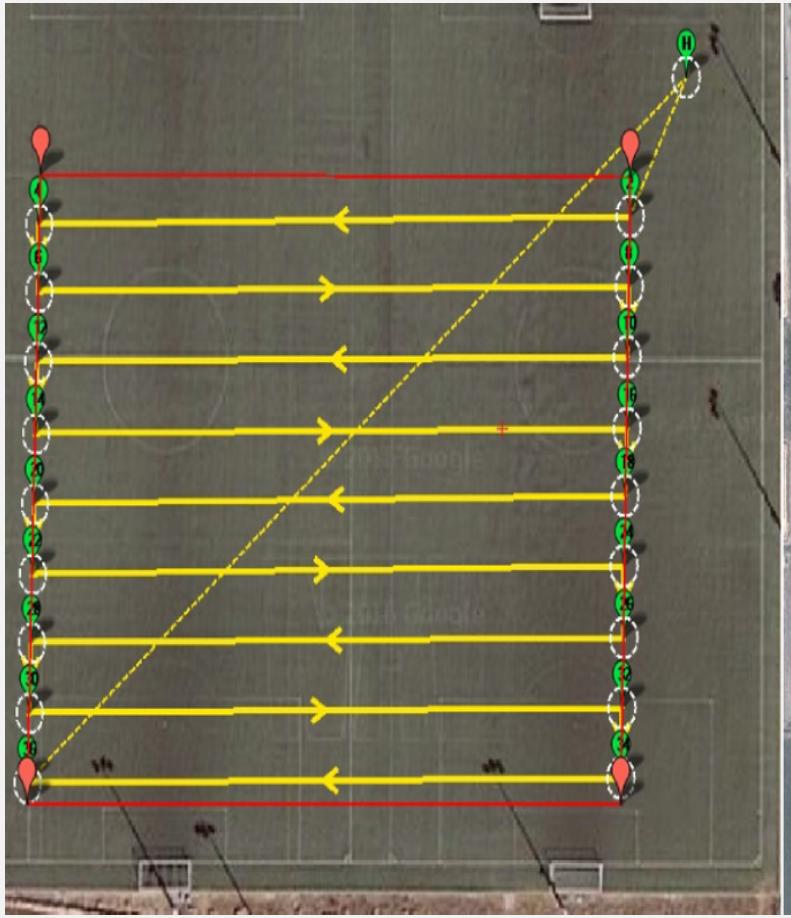
ІСНУЮЧІ РІШЕННЯ

Mission Planer



QGroundControl

ПОРІВНЯННЯ СПОСОБІВ ОБХОДУ “СПІРАЛЬ” ТА “ГОНАМИ”



МЕТА РОЗРОБЛЕННЯ

Метою даного дипломного проекту є вирішення комерційної проблеми в аграрній сфері, шляхом створення першого в своєму роді ПЗ, яке дозволить автоматизувати та відчутно пришвидшити побудову маршруту для БПЛА.

Унікальність ПЗ полягає в:

- автоматична побудова маршруту за способом обходу ‘спіраль’
- автоматична побудова маршруту з урахуванням висоти рельєфу

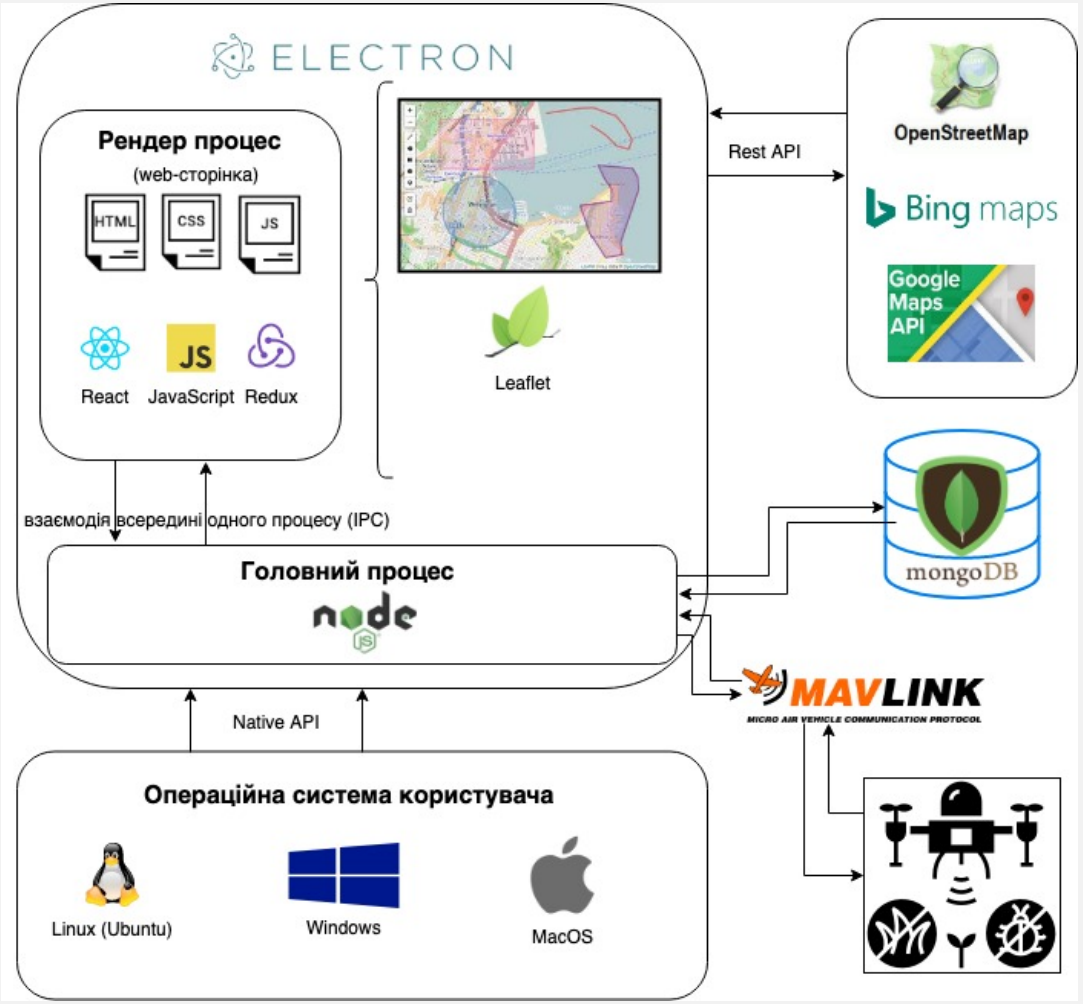
ВИМОГИ

Застосунок має надавати можливість:

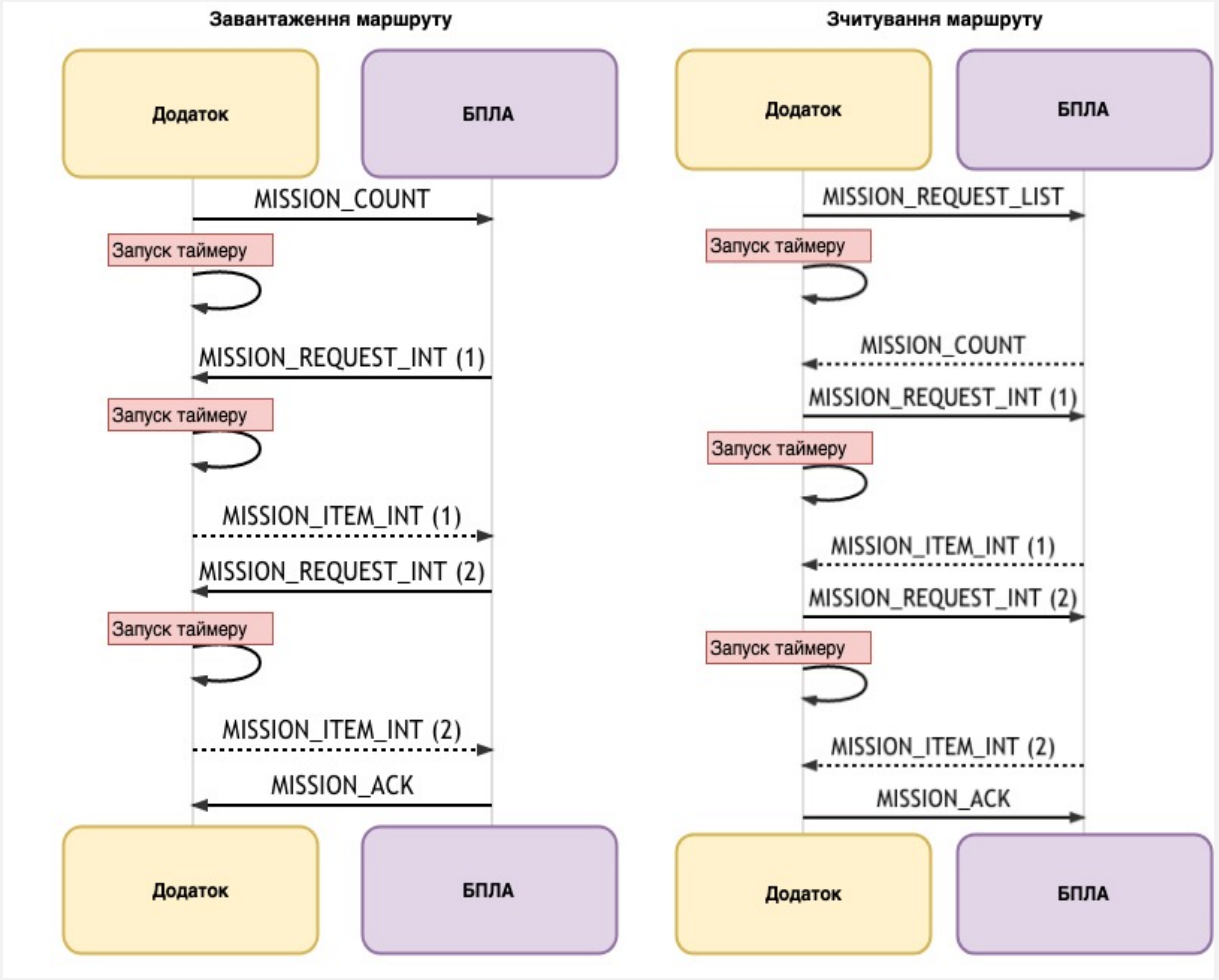
- автоматично будувати маршрут БПЛА в заданій області
- автоматично заокруглювати гострі кути маршруту
- будувати маршрут без доступу до мережі Інтернет
- змінювати провайдер зображень географічної мапи
- витримувати постійну висоту в кожній точці маршруту
- обійти всю задану зону способом обходу “спіраль”
- побудова має здійснюватись не довше, ніж за хвилину при заданій площі робочої зони 500 гектар
- графічно переглянути перепади висоти в заданій області
- знаходження місцезнаходження в редакторі мапи

ТЕХНОЛОГІЇ РОЗРОБЛЕННЯ

Мова – JavaScript
Комунікація з БПЛА – MAVLink
База даних – MongoDB



КОМУНІКАЦІЯ З БПЛА

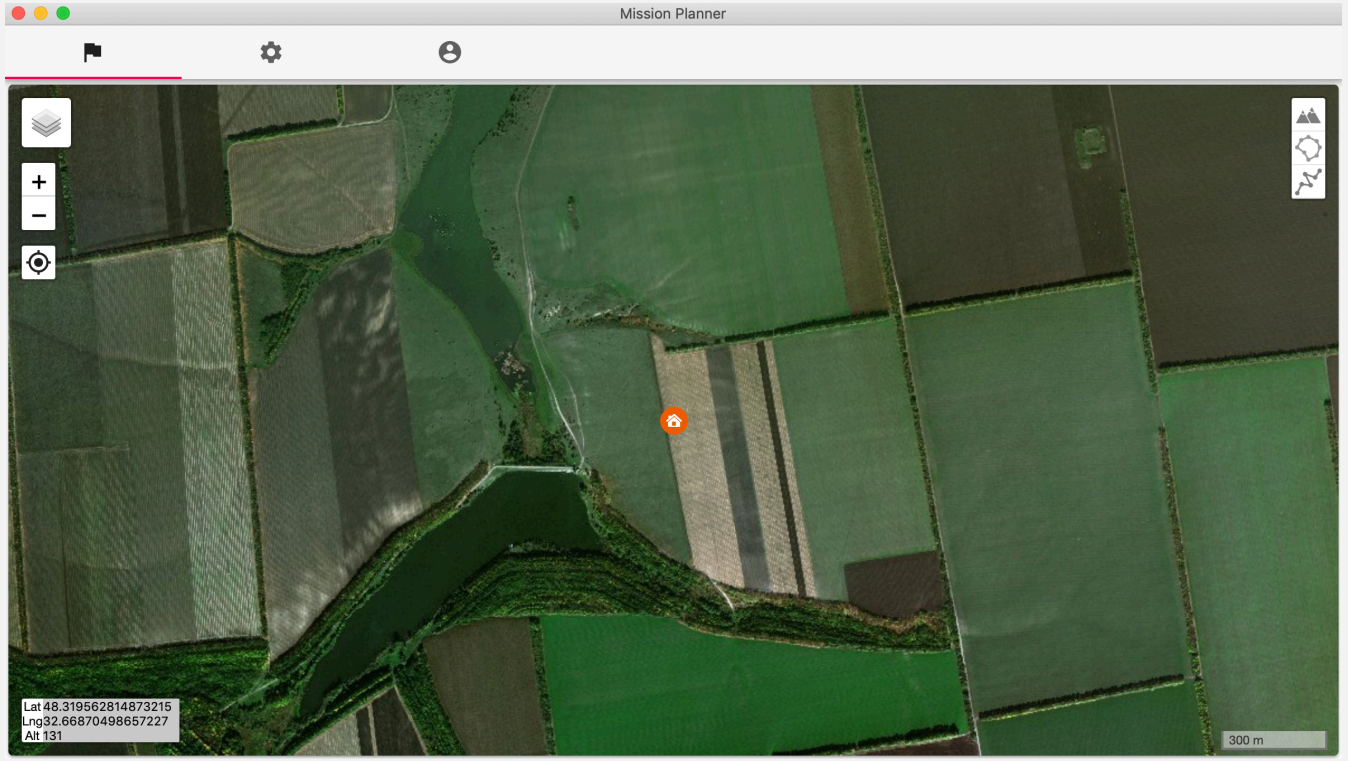


СТРУКТУРА ВЗАЄМОДІЇ ПРЕДСТАВЛЕНЬ ІНТЕРФЕЙСУ КОРИСТУВАЧА

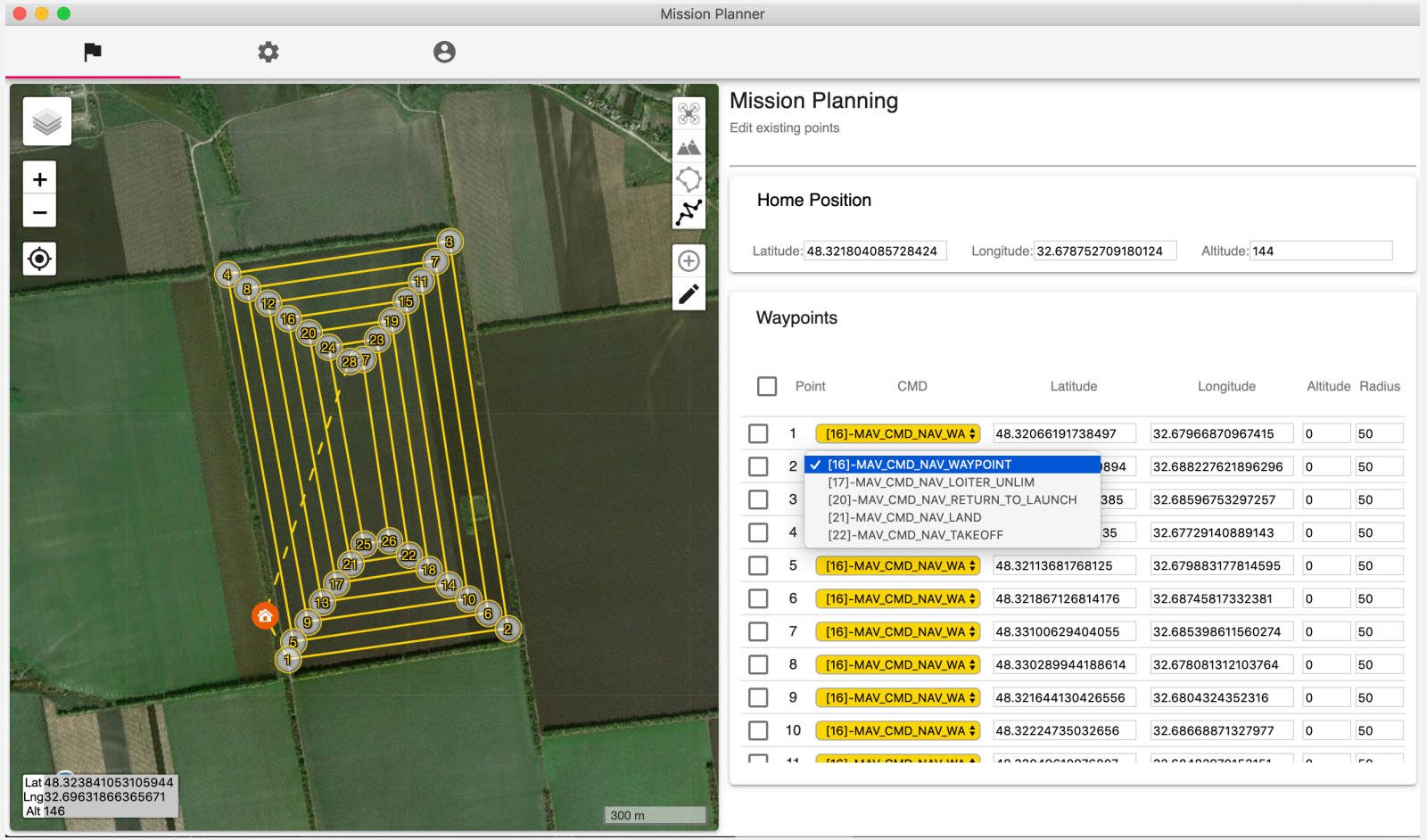
ОСНОВНИЙ АЛГОРИТМ. ПОБУДОВА МАРШРУТУ БПЛА

ДИЗАЙН ДОДАТКУ

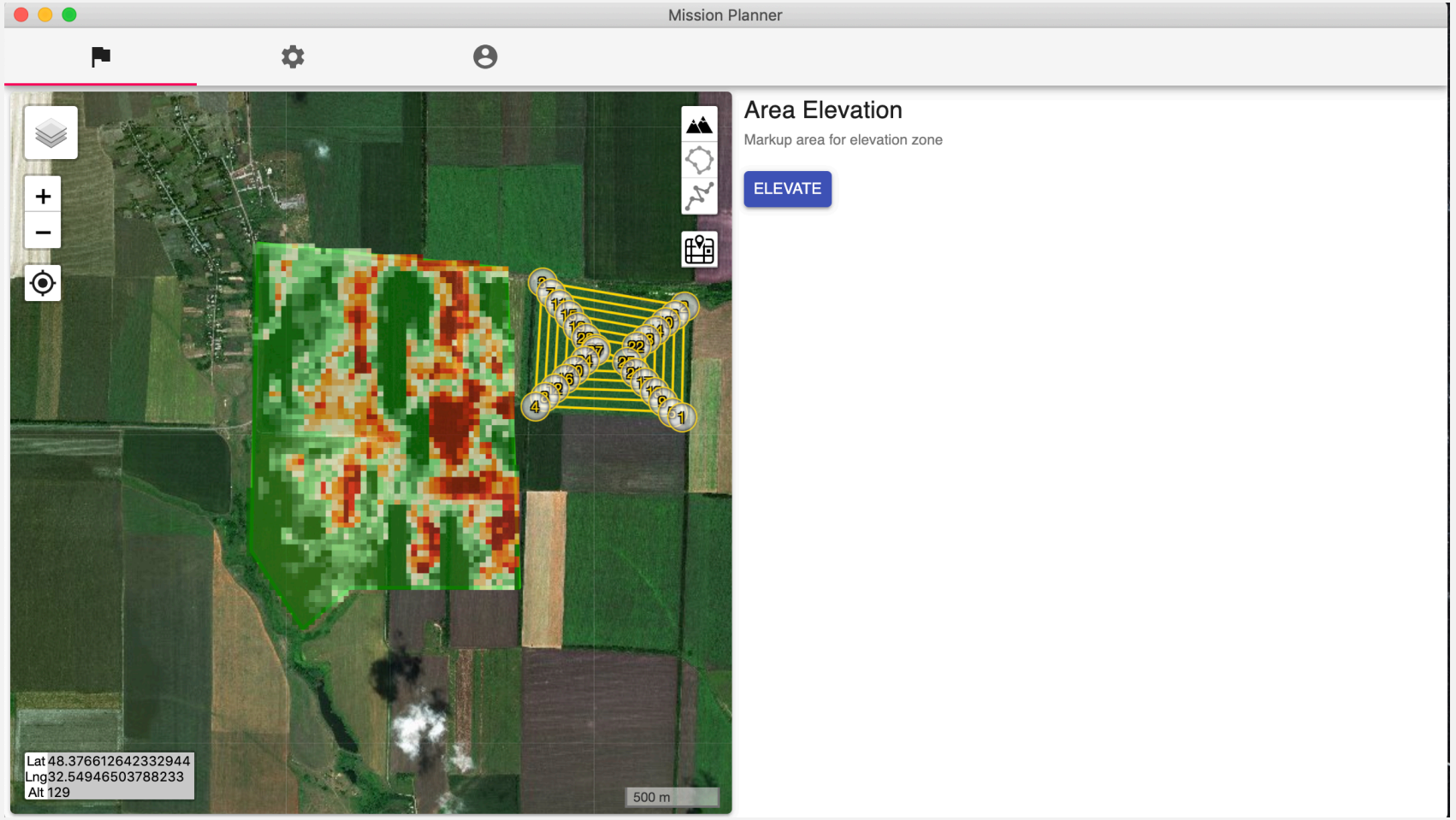
Дизайн виконано за вимогами Google Material Design



ПОБУДОВА МАРШРУТУ



ВИЗНАЧЕННЯ РЕЛЬЄФУ МІСЦЕВОСТІ



ВИСНОВКИ

Проведений аналіз наявних рішень виявив потребу створення даного програмного продукту.

Розроблений десктоп-додаток включає:

1. Автоматична побудова маршруту зі способом обходу “спіраль” з врахуванням висоти рельєфу та перешкод
2. Побудова 500 точок маршруту займає менше 1 хв
3. Можливість побудови маршруту без доступу до мережі Інтернет
4. Можливість записування/зчитування маршруту в/з автопілот(у) БПЛА

Розробку виконано в повному обсязі у відповідності до сформованих вимог.

ДЯКУЮ ЗА УВАГУ

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

ПРОГРАМНІ ЗАСОБИ ДЛЯ АВТОМАТИЗОВАНОЇ ПОБУДОВИ МАРШРУТІВ БПЛА В АГРАРНІЙ СФЕРІ

Програма та методика тестування

ДП.045490-04-51

«ПОГОДЖЕНО»

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ К.С. Чеботарьов

2018

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Програмне забезпечення для атоматизованої побудови маршрутів для БПЛА являє собою десктоп-додаток, написаний на мові програмування JavaScript, використовуючи технологію Electron.js.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) Побудова маршруту для БПЛА в заданих зонах різних опуклих форм (трикутної, прямокутної та шестикутної).
2) Завантаження маршруту в автопілот БПЛА за протоклом MAVLink.
3) Зчитування маршруту з автопілот БПЛА за протоклом MAVLink.
4) Очищення маршруту з автопілоту БПЛА
5) Зміна стану кожної reducr-функції.
6) Коректне відображення мапи.
7) Забезпечення коректного відображення кожного компоненту інтерфесу
8) Забезпечення коректного відображення додатку на платформах Windows, MacOS та Ubuntu.
9) Відповідність програмного забезпечення вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

1) Тестування відповідності, Тестування для перевірки відповідності функціональних можливостей додатку функціональним вимогам (acceptance testing).

2) Тестування сумісності, тестування для забезпечення сумісності додатку з різними ОС і апаратними платформами (compatibility testing).
3) Функціональне тестування, перевірка програми на відповідність до специфікацій і правильності виконання всіх необхідних функції (functional testing).
4) Тестування продуктивності програмного забезпечення, особливо тестування стабільності та масштабованості додатку (performance testing).
5) Навантажувальне тестування, особливо перевірка коректної роботи додатку з/без доступу до мережі Інтеренет (load testing).
6) Стрес-тестування, перевірка коректної роботи під час побудови великих навантажень на додаток (stress testing).

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність десктоп-додатку перевіряється шляхом:

1) Виконання юніт тестів для основних алгоритмів з використанням Mocha.js та Should.js
2) Виконання юніт тестів для компонентів інтерфейсу з використанням Jest.js
3) динамічного ручного тестування на відповідність функціональним вимогам;
4) тестування інтерфейсу.
5) статичного тестування коду;
6) тестування додатку на різних ОС та на різних пристроях;
7) тестування при максимальному навантаженні (побудова маршруту, який включає більше 1000 точок);
8) тестування з/без можливості доступу до мережі Інтеренет

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

ПРОГРАМНІ ЗАСОБИ ДЛЯ АВТОМАТИЗОВАНОЇ ПОБУДОВИ МАРШРУТІВ БПЛА В АГРАРНІЙ СФЕРІ

Керівництво користувача

ДП.045490-05-34

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ К.С. Чеботарьов

2019

ЗМІСТ

1. Опис структури інтерфейсу додатку

Інтерфейс являє собою вікно десктоп-додатку. Розміри вікна за замовчуванням мають ширину 800 та висоту 600. Розміри можуть змінюватись користувачем.

Додаток включає наступні панелі:

- «Панель редагування маршруту»;
- «Панель налаштувань»;
- «Прогнозування»;
- «Панель профілю користувача».

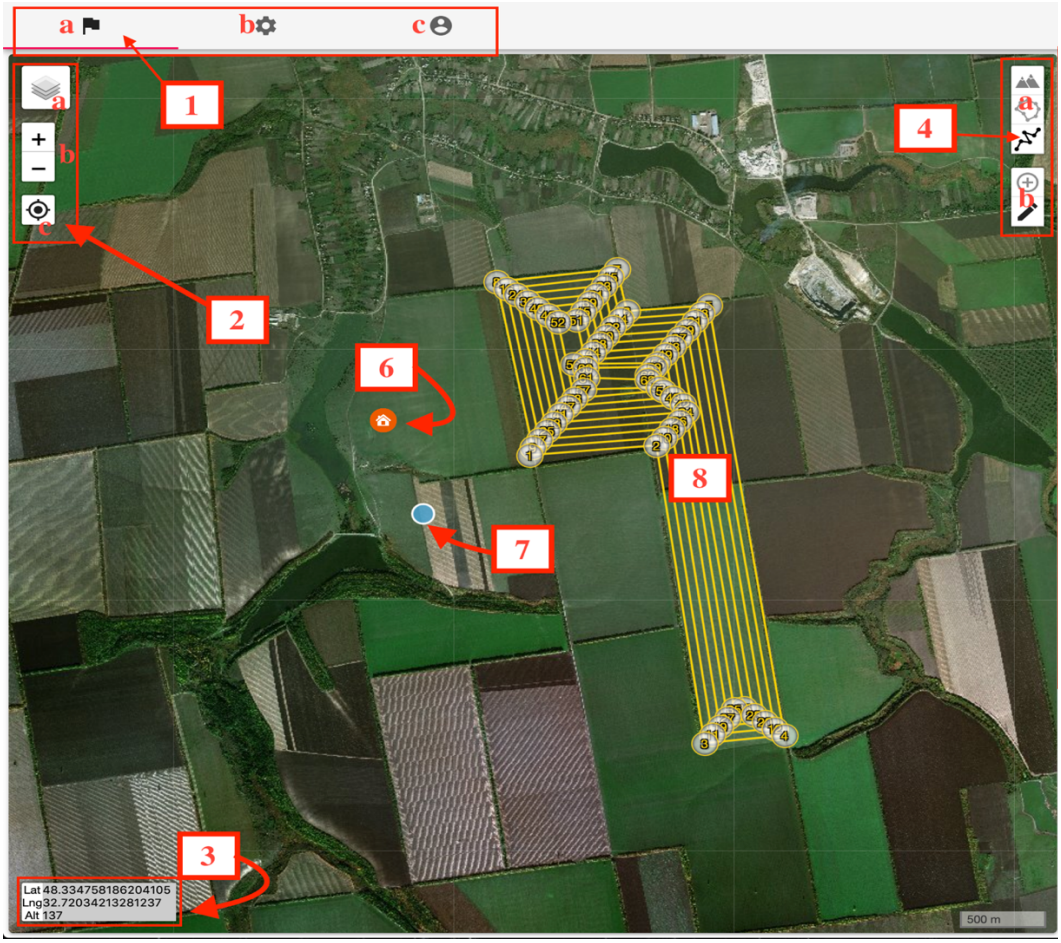


Рис. 1. «Панель редагування маршруту»

У верхній частині додатку завжди відображаються кнопки для переходу в іншу панель. Активна панель підсвічена рожевим кольором.

2. Опис вмісту додатку

Розглянемо кожну панель інтерфейсу користувача (рис. 1). Панель профілю користувача включає в себе налаштування особистих даних користувача, налаштування синхронізації збережених маршрутів. Панель налаштувань включає системні налаштування додатку. Сюди входять:

- параметри зовнішнього виду мапи;
- параметри елементів управління мапою;
- налаштування провайдерів географічної мапи;
- параметри підключення до автопілоту БПЛА;
- параметри потоку даних отриманих з БПЛА;
- параметри автоматичної побудови маршруту;
- параметри стратегій обльоту перешкод.

Панель редактору маршруту є найбільш функціональною. На тлі всього додатку розміщена географічна мапа. В зоні 2, відповідно до рис. 1, розташовані елементи управління мапою. Кнопка 2a відповідає за вибір провайдеру географічної мапи. При зміні провайдеру ресурси мапи змінюються. Елемент управління 2b відповідає за масштаб мапи. Користувач може збільшувати чи зменшувати його. Елемент управління 3c відповідає за перенесення центру мапи до місця фактичного місцезнаходження БПЛА чи оператора БПЛА відповідно до налаштувань. В зоні 3 розташована панель, що сповіщає користувача про фактичні географічні координати та висоту точку, на яку вказує курсор:

- поле lat – географічна широта;
- поле lng – географічна довгота;
- поле alt – висота точки відносно рівня моря.

В зоні 4 знаходяться інструменти для взаємодії з мапою. Зона 4a визначає режим взаємодії з доступних:

- визначення рельєфу;
- автоматична побудова маршруту;

– ручне редагування маршруту.

Кожен режим взаємодії має власні інструменти. Зона 4b визначає активний інструмент взаємодії з мапою. При виборі якогось інструменту, вікно ділиться навпіл, та з’являється додаткова інформація та параметри використання інструменту. За номером 6 на мапі розташована поточна точка старту для БПЛА. Відносно цієї точки автопілот БПЛА вираховує свою висоту. За номером 7 на мапі розташоване фактичне місце розташування оператора БПЛА. За номером 8 на мапі позначено приклад побудованого маршруту.

3. Процедура авторизації користувача

Авторизація користувача відбувається на панелі профілю користувача. Користувач має ввести свої логін і пароль (рис. 2) та натиснути кнопку «Go», після чого, користувач отримує інформацію про свої збережені маршрути.

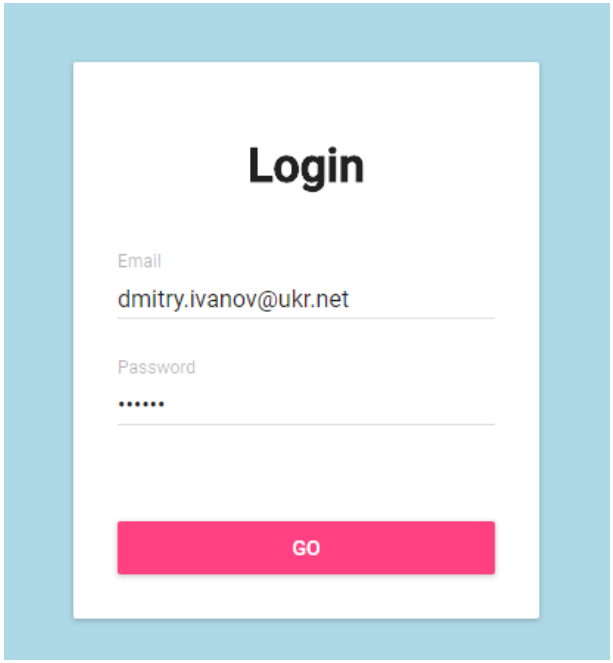


Рис. 2. Авторизація

4. Процедура побудови маршруту БПЛА

Для побудови маршруту користувач задає всі необхідні параметри побудови маршруту, потім вказує полігон, що описує бажану зону, в якій необхідно побудувати маршрут. Після цього користувач має можливість задати всі перешкоди, які знаходяться у вказаному полігоні. Потім користувач натискає на кнопку “Build”. В заданій області з’являються точки маршруту доступні для редагування (рис. 3).

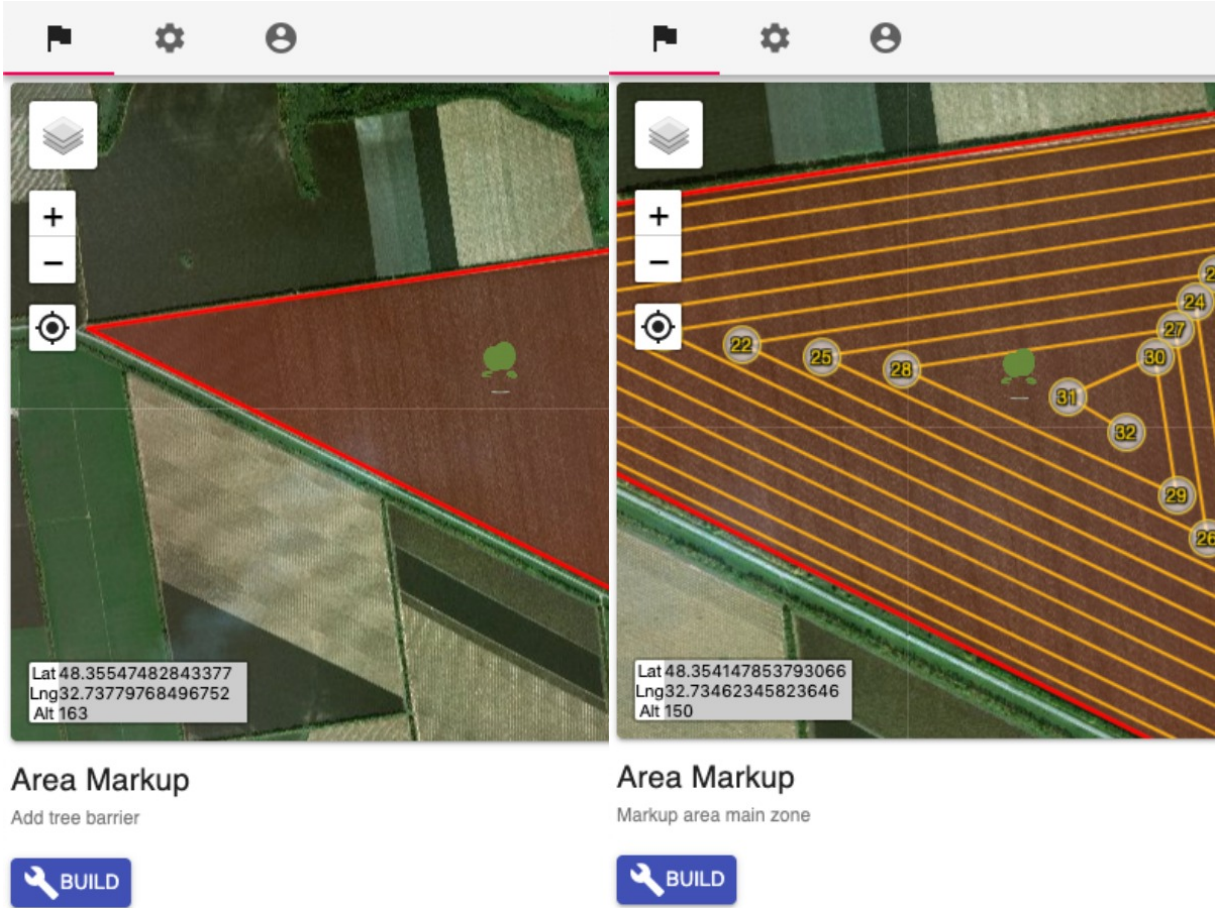


Рис. 3. Приклад побудови маршруту БПЛА