Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного модулю навігації з використанням нейронних мереж»

Виконав: студент 4 курсу
групи 2ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Бігас О.С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Рейда О.М.
(прізвище та ініціали)
Рецензент: к.т.н.,доц.каф. ОТ Кожем'яко А.В.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ Романюк О.Н.
«09» 06 2025 р.

ВНТУ– 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
Романюк О. Н.
« 21 » березня 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бігасу Олександру Сергійовичу

1. Тема роботи – Розробка програмного модулю навігації з використанням нейронних мереж.

Керівник роботи: Рейда Олександр Миколайович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 13 червня 2025 р.

3. Вихідні дані до роботи: модель розробки – клієнтський модуль для здійснення обрахунків координат зображень місцевості; Вхідні дані: середовище розробки Visual Studio Code; мова програмування Python; бібліотека розробки інтерфейсів Tkinter; нейронна модель Dino.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз існуючих підходів навігації та постановка задач дослідження; розробка структури та алгоритмів роботи програмного продукту; розробка програмних модулів фільтрації однорідних зображень та навігації; тестування програми; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд, актуальність, мета і завдання дослідження, предмет та об'єкт дослідження, задачі дослідження, наукова новизна, практичне значення отриманих результатів, GPS аналог, аналог інерціальний модуль, аналог стереокамера, аналог лідарний датчик, методи та засоби розробки, діаграма діяльності, алгоритм, висновки, публікації й апробації, завершальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Рейда О.М., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання і вибір методу вирішення поставленої задачі дослідження	25.03.25 - 05.04.25	Вик.
2	Розробка алгоритму фільтрації однорідних зображень	06.04.25 - 18.04.25	Вик.
3	Розробка алгоритму обчислення координат зображення місцевості	19.04.25 - 21.04.25	Вик.
4	Аналіз і вибір мови програмування та середовища розробки	22.04.25 - 10.05.25	Вик.
5	Програмна реалізація застосунку	11.05.25 - 18.05.25	Вик.
6	Тестування розробленого застосунку	19.05.25 - 25.05.25	Вик.
7	Оформлення матеріалів до захисту БКР	25.06.25 - 30.05.25	Вик.

Студент _____ (підпис) О.С. Бігас (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) О.М. Рейда (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.032.26:629.735

Бігас О.С. Розробка програмного модулю навігації з використанням нейронних мереж : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 95 с.

У рамках бакалаврської кваліфікаційної роботи було розроблено програмні засоби навігації БПЛА з використанням нейронних мереж. Дослідження включає розробку модулів визначення координат вхідних зображень шляхом їх порівняння з завантаженими картами із попередньою фільтрацією однорідних зображень. Призначення програмного модулю включає в себе використання на безпілотних літальних апаратах як основне або допоміжне джерело координат. Розроблене рішення може бути застосоване в різних сферах, зокрема в аграрному секторі для моніторингу посівних площ, у рятувальних операціях для пошуку людей у важкодоступних місцевостях, у логістиці для оптимізації маршрутів, а також у наукових дослідженнях, що потребують точного визначення місцеположення.

Програмний модуль розроблено з використанням мови програмування Python у середовищі Visual Studio Code. Для здійснення порівняльних операцій над зображеннями було використано нейронну модель Dino.

ABSTRACT

UDC 004.032.26:629.735

Development of a Navigation Software Module Using Neural Networks: Bachelor's Qualification Thesis in the specialty 121 Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2025. 95 p.

As part of the bachelor' thesis, software tools for UAV navigation using neural networks were developed. The research includes the development of modules for determining the coordinates of input images by comparing them with preloaded maps, with prior filtering of homogeneous images. The purpose of the software module includes its use on unmanned aerial vehicles as a primary or auxiliary source of coordinates. The developed solution can be applied in various fields, including the agricultural sector for monitoring crop areas, search and rescue operations for locating people in hard-to-reach areas, logistics for route optimization, as well as scientific research requiring precise location determination.

The software module was developed using the Python programming language in the Visual Studio Code environment. The neural model Dino was used to perform comparative operations on images.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі безпілотні літальні апарати (БПЛА) набули широкого розповсюдження та застосування у різних сферах, зокрема у військовій справі, наукових дослідженнях, промисловості, сільському господарстві, екологічному моніторингу та логістиці [1]. Завдяки стрімкому розвитку технологій, існує велике різноманіття безпілотних платформ – від компактних квадрокоптерів до великих літальних апаратів з фіксованим крилом.

Однією з ключових переваг БПЛА є можливість їх використання у автономному або віддаленому режимі керування. Це дозволяє оператору виконувати складні завдання, знижуючи ризики для людини та підвищуючи ефективність виконання операцій [2]. Важливим аспектом реалізації таких можливостей є точне визначення положення літального апарата у просторі.

Традиційні навігаційні алгоритми вразливі до шуму, незнайомого оточення та непередбачуваних перешкод, не здатні навчатись на нових даних та адаптуватись до змін у середовищі, що необхідно для успішного виконання поставлених задач у процесі використання БПЛА.

Актуальність розробки зумовлена необхідністю самонавчання та генералізації досвіду в процесі навігації системи у складних умовах, можливістю інтеграції з комп’ютерним зором (розпізнавання об'єктів, карт, знаків), підвищення швидкодії прийняття рішень.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Метою бакалаврської кваліфікаційної роботи є підвищення надійності і точності позиціювання БПЛА у складних умовах навігації з використанням нейронних мереж.

Дослідження передбачає розробку ряду модулів, що будуть інтегровані у кінцеве програмне рішення. Розробка включає модулі аналізу вхідних зображень на їх однорідність, тобто визначення можливості навігації з їх застосуванням та аналізу коректних зображень для обчислення координат. У розробку також включається модуль графічного інтерфейсу, що надасть змогу налаштування параметрів моделі та тестування обчислювального модулю.

Основними задачами дослідження є:

- розробити алгоритми аналізу зображень місцевості;
- модифікувати метод навігації для підвищення ефективності позиціювання;
- -розробити модуль обчислення координат;
- розробити модуль фільтрації зображень, що не містять опорних точок;
- розробити модуль графічного інтерфейсу;
- інтегрувати розроблені модулі у єдине програмне рішення.

Об'єктом дослідження є: процес розробки програмного модулю навігації з використанням нейронних мереж для аналізу зображень місцевості та обчислення координат.

Предметом дослідження є: методи та засоби розробки програмного модулю навігації з використанням нейронних мереж аналізу зображень.

Методи дослідження. У процесі проведення досліджень використовувались: теорія ймовірностей та нечітких множин, методи математичного аналізу, комплексний аналіз існуючих методів навігації; метод ітеративного пошуку відповідностей; методи аналізу на однорідність та наявність контурів; засоби нейронних мереж для обчислення координат місцевості шляхом порівняння з референсними зображеннями карти, теорія динамічних систем, формалізація вхідних та вихідних параметрів навігаційного модуля, методи побудови та навчання нейронних мереж.

Наукова новизна отриманих результатів.

Модифіковано метод навігації, що на відміну від існуючого, використовує фільтрацію однорідних зображень місцевості для підвищення ефективності позиціювання та точності обчислення координат.

Практична цінність отриманих результатів.

Практична цінність одержаних результатів полягає у можливості використання нейронних мереж для виконання обчислень координат вхідних зображень місцевості. Розроблені програмні модулі можуть бути використані в сфері аграрних технологій, наукових досліджень, у цілях виконання рятувальних операцій та у сферах, де є потреба у стабільних та точних рішеннях навігації безпілотних літальних засобах. Програмний модуль має на меті імплементувати можливість інтеграції у різноманітні апаратні системи.

Особистий внесок здобувача. Усі наукові результати, що викладені у бакалаврській кваліфікаційній роботі, отримані автором самостійно. У наукових працях автору належать такі результати: використання нейронної моделі Dino для обробки зображень місцевості, розробка алгоритму фільтрації однорідних зображень, розробка модулю обчислення координат місцевості з використанням нейронних мереж [3].

Апробація результатів роботи. Основні положення бакалаврської кваліфікаційної роботи доповідалися всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії у 2025 році і опубліковані в матеріалі конференції [3];

Публікації. За тематикою дослідження опубліковано 1 наукову працю праці у збірниках матеріалів конференцій.

1 АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ НАВІГАЦІЇ БПЛА ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану технологій навігації БПЛА

Принцип роботи навігації БПЛА базується на зборі та обробці даних із різних датчиків, які визначають поточне місцеположення літального апарату, його висоту, орієнтацію та швидкість [4]. Дані можуть надходити з GPS-модулів, інерціальних сенсорів (акселерометрів, гіроскопів, магнітометрів), камер, Lidar-систем, ультразвукових та радіолокаційних датчиків. Деякі системи додатково використовують карти місцевості або еталонні орієнтири для підвищення точності.

Сучасний стан технологій навігації БПЛА включає широкий спектр методів та систем, що постійно вдосконалюються. GPS-навігація залишається основним засобом визначення координат, але має свої обмеження, зокрема в умовах поганого сигналу або навмисного глушіння. Альтернативні методи, такі як візуальна навігація (комп'ютерний зір) [5], лазерні системи (Lidar) та інерціальні навігаційні системи (INS), набувають дедалі більшого значення. Застосування штучного інтелекту та нейромереж для аналізу візуальних даних дозволяє значно покращити ефективність виконання поставлених задач.

Навігаційні системи можна класифікувати за типом використовуваних технологій:

1. Супутникова навігація (GPS, ГЛОНАСС, Galileo, BeiDou).

2. Інерціальна навігація (INS).

3. Оптична навігація (візуальна одометрія, аналіз орієнтирів, нейронні мережі).

4. Лазерна та радіолокаційна навігація (Lidar, SLAM).

5. Гібридні системи, що поєднують кілька методів.

Таким чином, розвиток технологій навігації БПЛА спрямований на підвищення автономності, адаптацію до складних умов та використання штучного інтелекту для покращення точності.

1.2 Порівняльний аналіз аналогів

Сучасні безпілотні літальні апарати (БПЛА) потребують надійних та високоточних навігаційних систем, що забезпечують стабільний політ і точне позиціонування навіть у складних умовах. Для цього використовуються різні технології, кожна з яких має свої особливості та переваги. До найбільш відомих рішень відносять:

- GPS модуль u-blox ZED-F9P;
- інерціальний модуль VectorNav VN-100;
- модуль візуальної одометрії Intel RealSense T265;
- Lidar-сенсор Ouster OS1-64;
- гібридна система Septentrio AsteRx-i V.

Одним із основних методів є супутникова навігація, яка реалізується за допомогою GNSS-приймачів, таких як u-blox ZED-F9P [6]. Цей модуль підтримує глобальні навігаційні супутникові системи, включаючи GPS, ГЛОНАСС, Galileo та BeiDou, і забезпечує високу точність визначення координат. Завдяки підтримці технології RTK (Real-Time Kinematic) приймач може досягати сантиметрової точності, що є критично важливим для автономних систем та дронів, які виконують завдання у реальному часі.

Рисунок 1.1 – GNSS приймач u-blox ZED-F9P.

Широко застосовуються інерціальні навігаційні системи (INS), одним із прикладів яких є VectorNav VN-100 [7]. Цей компактний модуль поєднує в собі трьохосьові гіроскопи, акселерометри та магнітометри, що дозволяє визначати положення та орієнтацію БПЛА без необхідності використання зовнішніх навігаційних сигналів. INS-системи особливо ефективні в середовищах, де GPS недоступний, однак вони мають накопичувальну похибку, яка з часом зростає.

Рисунок 1.2 – Інерціальний модуль VectorNav VN-100.

Ще одним перспективним підходом є оптична навігація, що базується на аналізі візуальної інформації для визначення положення літального апарата. Один із сучасних модулів, що реалізує цей метод, – Intel RealSense T265, який використовує технологію візуальної одометрії [8]. Завдяки подвійним камерам та вбудованому інерціальному вимірювальному блоку (IMU), модуль здатний визначати зміну положення БПЛА у просторі, аналізуючи зміщення ключових точок на зображеннях. Такий підхід дозволяє працювати без використання

супутникових сигналів, що робить його ідеальним для застосування у приміщеннях або в умовах поганого освітлення.

Рисунок 1.3 – Модуль оптичної навігації Intel RealSense T265.

Окрім оптичної навігації, активно розвивається напрямок використання лазерних та радіолокаційних систем, які застосовуються для створення тривимірних карт місцевості та навігації у реальному часі. Одним із передових рішень у цій сфері є Ouster OS1-64 – високоточний Lidar-сенсор, що забезпечує детальне сканування оточення [9]. Завдяки технології SLAM (Simultaneous Localization and Mapping) цей модуль дозволяє БПЛА будувати карти місцевості та визначати своє місцезнаходження навіть у складних умовах, таких як лісові масиви або урбанізовані території.

Рисунок 1.4 – Lidar-сенсор Ouster OS1-64.

Для досягнення максимальної точності та надійності в навігації найчастіше застосовуються гібридні системи, які об'єднують кілька технологій. Одним із прикладів такого рішення є Septentrio AsteRx-i V, що комбінує GNSS-приймач із інерціальною системою [10]. Поєднання супутникової навігації з INS дозволяє отримати високу стійкість до зовнішніх перешкод, а також забезпечує стабільну навігацію у випадку короткочасної втрати супутникового сигналу. Цей модуль широко використовується в безпілотних системах, що працюють у складних умовах, таких як автономні літальні апарати, роботи та наземні транспортні засоби.

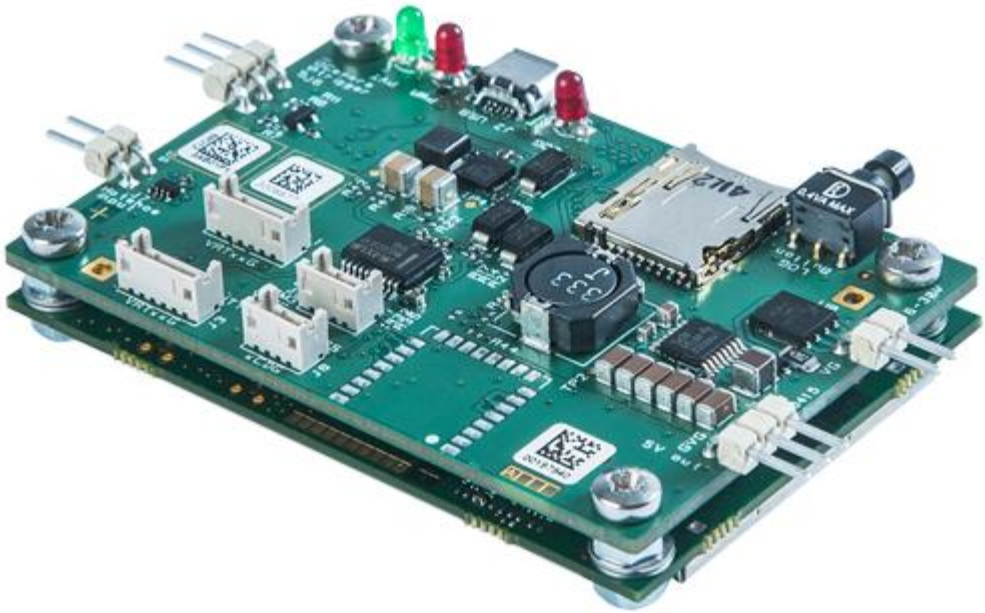


Рисунок 1.5 – Комбінований модуль навігації Septentrio AsteRx-i V

Розробка програмного модулю навігації з використанням нейронних мереж вимагає досвіду роботи з відповідними системами, їх тренуванням та конфігурацією. Це передбачає створення модульної системи комбінованих програмних підходів, підбір коректного датасету для тренування та оптимізацію для досягнення швидкодії. Складова інтерфейсу користувача дозволить взаємодію з розробленими алгоритмами для їх тестування та кінцевого налаштування.

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	u-blox ZED-F9P	VectorNav VN-100	Intel RealSense T265	Ouster OS1-64	Septentrio AsteRx-i V	Власний модуль
Точність вимірювань	1	0	0	1	1	1
Швидкодія	1	1	1	0	1	1
Стабільність до змін середовища	0	0	0	0	1	1
Вартість комплектуючих	1	1	1	0	0	1
Енергоефектив-ність	1	1	0	1	0	1
Загальна оцінка	4	3	3	2	3	5

Відповідно до таблиці порівняльних характеристик, розробка власного програмного модулю навігації з використанням нейронних мереж є доцільною. Отриманий продукт зможе покрити недоліки існуючих рішень та забезпечити стабільність потоку навігаційних даних.

1.3 Аналіз методів розв’язання задачі

Одним із найпоширеніших методів визначення місцеположення безпілотних літальних апаратів є використання глобальних навігаційних супутникових систем (GNSS), зокрема GPS. Ця технологія забезпечує високу точність визначення координат шляхом обробки сигналів від кількох супутників, що робить її основним інструментом для навігації. Основними перевагами GPS-модулів є їхня відносно низька вартість, невелике енергоспоживання та проста інтеграція у навігаційні системи БПЛА. Однак серед суттєвих недоліків варто відзначити можливість

втрати сигналу у складних умовах, таких як міська забудова, густі лісові масиви чи зони радіоперешкод.

Альтернативним підходом до навігації є використання технології Lidar (Light Detection and Ranging). Lidar-системи сканують навколишній простір за допомогою лазерного випромінювання, створюючи тривимірну карту місцевості [11]. Основними перевагами цього підходу є висока точність вимірювань, здатність проникати крізь тонкі об’єкти, такі як листя дерев, та можливість використання у середовищах, де супутникова навігація є недоступною. Водночас, значними недоліками є висока вартість обладнання та складність обробки отриманих даних, що потребує потужних обчислювальних ресурсів для аналізу хмар точок (Point Cloud).

Ще одним методом визначення місцеположення є використання оптичних стереокамер. Цей підхід базується на аналізі зображень з двох або більше камер, що дозволяє оцінити відстань до об'єктів шляхом порівняння різниці у перспективах. Основними перевагами цього методу є його відносно низька вартість та відсутність необхідності у високопродуктивних обчислювальних системах. Однак точність такого способу навігації є порівняно низькою, особливо у складних умовах освітлення або на місцевостях зі слабко вираженим рельєфом.

Інерціальні навігаційні системи (INS) є ще одним поширеним варіантом визначення положення БПЛА. Вони базуються на аналізі показників акселерометрів і гіроскопів, що дозволяє оцінювати переміщення апарата у просторі [12]. Перевагою такого методу є його незалежність від зовнішніх сигналів, що забезпечує стабільну роботу навіть за відсутності GPS. Більшість сучасних польотних контролерів уже мають вбудовані інерціальні датчики. Однак суттєвим недоліком є накопичення похибок у вимірюваннях, що призводить до поступового зниження точності через неможливість врахування впливу зовнішніх факторів, таких як пориви вітру. Існують високоточні INS-рішення, що компенсують цю проблему, але їхня вартість може значно перевищувати ціну самого БПЛА.

Метод візуальної одометрії є ще одним підходом до навігації. Він передбачає обробку послідовних кадрів із камери для оцінки зміни положення апарата у просторі. Такий метод може бути ефективним у контрольованих середовищах, але його точність може значно знижуватися в умовах недостатнього освітлення або при відсутності чітких орієнтирів у полі зору.

Оптична навігація передбачає порівняння отриманих зображень місцевості з попередньо створеними картами [13]. Цей метод може бути ефективним у стабільних умовах, однак він чутливий до змін рельєфу, сезонних змін ландшафту та атмосферних умов. Для забезпечення його ефективності необхідне регулярне оновлення бази картографічних даних.

Комбіновані підходи до навігації є найефективнішими, оскільки вони поєднують переваги кількох методів і мінімізують їхні недоліки. Найчастіше GPS-дані доповнюються інформацією з інерціальних датчиків, камер або Lidar-систем, що дозволяє забезпечити більш стабільну навігацію навіть за умов втрати супутникового сигналу. Однак під час вибору комбінованої системи слід враховувати баланс між точністю, вартістю та обчислювальною складністю, щоб не ускладнювати конструкцію та програмну реалізацію навігаційного модуля БПЛА.

В результаті аналізу порівняння було прийнято рішення розробити програмний модуль, що комбінуватиме можливості порівняння еталонних зображень за допомогою традиційних алгоритмів, нейронних мереж та попередньої оцінки зображень для фільтрації таких, обробка яких не є доцільною. Саме така комбінація надасть змогу компенсувати недоліки взаємні недоліки та у комбінації з GPS модулями забезпечуватиме стабільність роботи усієї системи.

1.4 Постановка задач

Проаналізувавши переваги та недоліки існуючих підходів для навігації БПЛА, було визначено наступні завдання, які необхідно виконати для успішної розробки програмного модулю:

- розробка алгоритмів аналізу зображень місцевості для обчислення координат;
- розробка модуля для фільтрації однорідних зображень, аналіз яких не є доцільним;
- розробка модуля графічного інтерфейсу, що надасть змогу налаштування параметрів нейронної моделі та тестування програмного модулю;
- інтеграція розроблюваних модуль у єдине програмне рішення.

1.5 Висновки

У першому розділі було розглянуто стан розвитку технологій навігації БПЛА, їх типи та аспекти програмних реалізацій. Було розглянуто такі технології, як GPS, Lidar, інерціальні системи, візуальну одометрію та оптичну навігацію.

Проаналізувавши переваги та недоліки існуючих засобів навігації, було сформульовано завдання для подальшої розробки власного програмного модулю для навігації літальних апаратів. Ці завдання включають розробку алгоритмів оптичної навігації, фільтрації однорідних зображень та модулю графічного інтерфейсу для конфігурації та тестування. Таким чином було доведено доцільність розробки програмного модулю. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власних програмних засобів.

2 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ФІЛЬТРАЦІЇ ОДНОРІДНИХ ЗОБРАЖЕНЬ МІСЦЕВОСТІ ТА НАВІГАЦІЇ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

2.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

У процесі розробки програмного модуля навігації для безпілотного літального апарата було обґрунтовано обрано архітектуру, засоби та методи для виконання розробки програмних модулів оптичної навігації з використанням нейронних мереж та фільтрації однорідних зображень місцевості. Для виконання розробки обчислювальних модулів та модулю інтерфейсу було обрано мову програмування Python, з використанням технологій бібліотеки OpenCV, нейронної моделі Dino та бібліотеку Tkinter для імплементації інтерфейсу кінцевого застосунку. Середовищем розробки було обрано редактор коду Microsoft Visual Studio Code.

Python – це високорівнева, мультипарадигменна мова програмування загального призначення, яка підтримує як об’єктно-орієнтоване, так і процедурне програмування, а також функціональний стиль [14]. Однією з головних переваг цієї мови програмування є її високий рівень абстракції, що дозволяє зменшити кількість вихідного коду та покращити його читабельність і підтримуваність. Завдяки цьому Python активно використовується в широкому спектрі галузей, зокрема у сфері наукових досліджень, чисельного моделювання, обробки великих обсягів даних, автоматизації процесів, розробки веб-застосунків, машинного навчання, а також при створенні систем штучного інтелекту.

Python є інтерпретованою мовою: при виконанні програма транслюється в проміжний байт-код, який потім обробляється віртуальною машиною інтерпретатора. Такий підхід дозволяє забезпечити кросплатформність та швидке прототипування, однак має й певні недоліки, зокрема, нижчу продуктивність у порівнянні з компільованими мовами. Для компенсації цих недоліків існують

спеціалізовані інструменти оптимізації, які дозволяють досягати значного приросту продуктивності. Наприклад, реалізація CPython дозволяє інтегрувати низькорівневий код на мові C, що дає змогу оптимізувати критичні ділянки програми [15]. Крім того, застосування таких засобів як Numba або PyPy дозволяє компілювати окремі фрагменти Python-коду у нативний машинний код, що суттєво підвищує ефективність виконання.

Окрім використання в галузях математичного аналізу, автоматизації задач і створення серверної логіки, мова програмування Python набула широкого поширення у сфері штучного інтелекту, зокрема при розробці та реалізації нейронних мереж. Це зумовлено наявністю великого обсягу спеціалізованих бібліотек, таких як TensorFlow [16], PyTorch, Keras, Theano, які значно спрощують як процес створення моделей, так і їх навчання та подальшу експлуатацію.

Нейронна мережа є математичною моделлю, натхненною принципами функціонування біологічного мозку. Вона складається з набору штучних нейронів, які об’єднані між собою зваженими зв’язками та організовані в шари (вхідний, приховані, вихідний). Кожен нейрон виконує просту операцію: обчислює зважену суму своїх входів, до якої застосовується певна активаційна функція.

Процес навчання нейронної мережі полягає в ітеративному налаштуванні ваг зв’язків між нейронами з метою мінімізації похибки між реальним виходом моделі та очікуваним результатом [17]. Цей процес реалізується за допомогою алгоритмів оптимізації, таких як градієнтний спуск, у поєднанні з функціями втрат. У випадку попередньо навченої моделі (претренування) – модель спочатку навчається на великому універсальному датасеті, після чого її можна донавчити на конкретному наборі даних для вирішення певної прикладної задачі.

Зважаючи на високу обчислювальну складність навчання нейронних мереж, особливо глибоких, велика частина таких обчислень виконується не на центральному процесорі (CPU), а на графічному процесорі (GPU). Це пов’язано з особливостями архітектури GPU, яка, на відміну від CPU, містить не декілька

потужних універсальних ядер, а сотні або тисячі менш потужних, вузькоспеціалізованих CUDA-ядер (Compute Unified Device Architecture).

CUDA – це апаратна і програмна архітектура, розроблена компанією NVIDIA, яка дозволяє розробникам писати високопродуктивні обчислювальні алгоритми, які виконуються на GPU. CUDA-ядра здатні одночасно виконувати тисячі простих обчислювальних операцій, що є особливо ефективним у випадках обробки великих масивів даних, типових для задач нейронних мереж.

У процесі реалізації порівняльного алгоритму оптичної навігації в межах кваліфікаціної дипломної роботи було використано сучасну нейронну модель DINO (Distillation with No Labels). Модель DINO побудована на архітектурі трансформерів, що забезпечує кращу узагальнюваність і здатність до виявлення семантичних зв’язків між об’єктами на зображенні. Однією з ключових переваг цієї моделі є можливість навчання у безнаглядовий спосіб, що дозволяє використовувати великі масиви немаркованих зображень. Це значно знижує витрати на підготовку даних та робить модель придатною до застосування в реальних умовах, де розмічені датасети часто відсутні. DINO також відома високою продуктивністю в задачах виділення ознак, кластеризації зображень та порівняння візуальних шаблонів, що ідеально підходить для задач оптичної навігації, де критично важливо точно визначати відповідності між кадрами.

Для реалізації методів обробки зображень, зокрема в частині фільтрації однорідних зображень, попередньої обробки кадрів, нормалізації, а також пошуку ключових точок, було застосовано потужну бібліотеку OpenCV (Open Source Computer Vision Library). OpenCV – це відкрита кросплатформна бібліотека комп’ютерного бачення та цифрової обробки зображень, що підтримує мову Python і дозволяє реалізовувати широкий спектр операцій: від базових трансформацій (масштабування, поворот, зміна яскравості, гістограма тощо) до високорівневих алгоритмів детекції та відстеження об'єктів, розпізнавання облич, аналізу руху та побудови тривимірних сцен [18]. Завдяки високій продуктивності та оптимізації

під апаратні ресурси, OpenCV є стандартом де-факто в галузі прикладного комп'ютерного бачення.

Для забезпечення однорідності структури коду, зручності налаштування параметрів алгоритмів, а також для побудови простого графічного інтерфейсу користувача, у даній роботі використано бібліотеку Tkinter. Tkinter є стандартним інтерфейсом до графічного тулкіту Tcl/Tk у Python та забезпечує можливість створення кросплатформних GUI-додатків [19].

Також для реалізації було обрано середовище розробки Microsoft Visual Studio Code – кросплатформний редактор коду з відкритим вихідним кодом, розроблений компанією Microsoft [20]. Visual Studio Code підтримує широкий спектр мов програмування та має розширену інфраструктуру плагінів, яка дозволяє адаптувати середовище під специфіку конкретного проєкту.

Таким чином, використання моделі Dino у поєднанні з зазначеним програмним інструментарієм формує ефективну, модульну та масштабовану систему навігації, яка здатна адаптуватися до складних умов експлуатації та забезпечити високу точність просторової орієнтації БПЛА.

2.2 Розробка модуля обчислення координат зображення місцевості

У межах системи оптичної навігації для БПЛА було створено програмний модуль аналізу зображень, основною функцією якого є зіставлення вхідного візуального потоку з попередньо збереженими еталонними шаблонами місцевості. Це дозволяє визначати місцезнаходження апарата за допомогою візуальної інформації, отриманої з бортових камер, без залучення зовнішніх супутникових систем. Модуль інтегрується до загальної архітектури програмного забезпечення та взаємодіє з іншими підсистемами, такими як обчислювальний блок, система фільтрації однорідних зображень та інтерфейс користувача.

З метою формалізації структури та принципів роботи програмного програмного модулю було побудовано UML-діаграму класу (рис. 2.1).

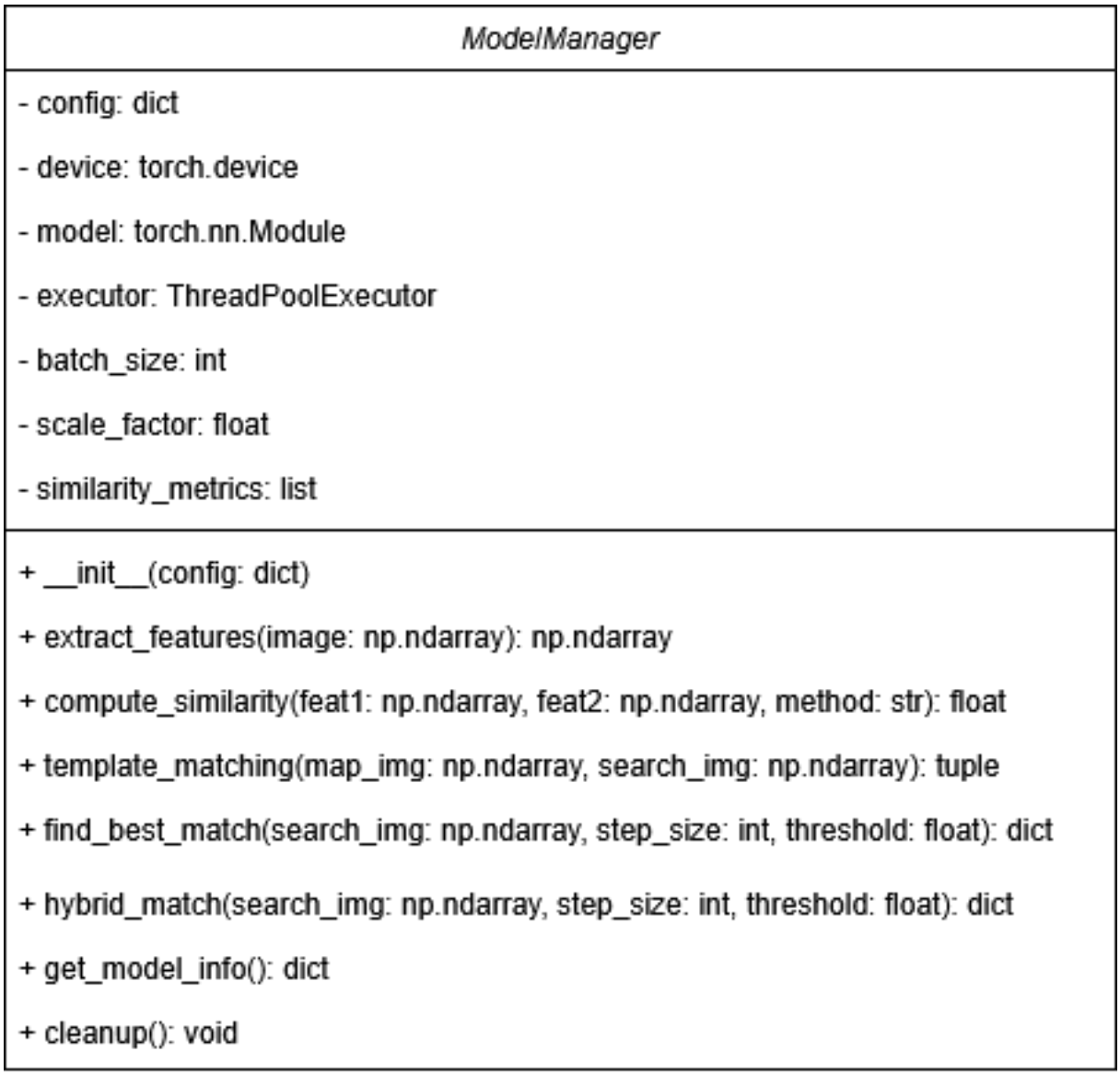


Рисунок 2.1 – Діаграма класу ModelManager.

Для забезпечення коректної та стабільної роботи алгоритму аналізу зображень, вхідні дані проходять попередню обробку, яка включає стандартну процедуру нормалізації. Зокрема, перед подачею зображення на вхід нейронної мережі, воно трансформується за допомогою типової послідовності перетворень, яка охоплює зміну розміру до фіксованих параметрів, приведення зображення до формату тензора, а також нормалізацію значень пікселів у каналах RGB відповідно до середніх значень та стандартних відхилень, прийнятих для попередньо натренованих моделей. Метод extract_features виконує це перетворення, а також гарантує перетворення зображення у формат, сумісний з моделлю (рис. 2.2).

```
try:
    # Pre-process images - handle different color channels
    processed_images = []
    for img in images:
        if img is None or img.size == 0:
            continue

        # Ensure correct color channels
        if len(img.shape) == 2:
            img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
        elif img.shape[2] == 4:  # Handle RGBA
            img = cv2.cvtColor(img, cv2.COLOR_RGBA2RGB)

        processed_images.append(img)

    if not processed_images:
        return []

    # Create batch tensor
    batch_tensors = torch.stack([self.transform(img) for img in processed_images]).to(self.device)

    # Extract features
    with torch.no_grad():
        if hasattr(self.model, 'forward_features'):
            features = self.model.forward_features(batch_tensors)
            # Global average pooling
            if len(features.shape) == 4:  # If features are [B, C, H, W]
                features = torch.mean(features, dim=[2, 3])
        else:
            features = self.model(batch_tensors)

    # Normalize feature vectors
    features = torch.nn.functional.normalize(features, p=2, dim=1)

    # Return as list of numpy arrays
    return [feat.cpu().numpy() for feat in features]
except Exception as e:
    print(f"Error in batch feature extraction: {str(e)}")
    return [np.zeros(1000) for _ in range(len(images))]
```

Рисунок 2.2 – Виконання нормалізації зображення.

Для зіставлення ознак між зображеннями використовується метод compare_features. Він підтримує різні метрики, зокрема косинусну подібність, евклідову відстань та скалярний добуток. У якості додаткового етапу аналізу використовується класичний підхід до шаблонного зіставлення зображень на основі OpenCV. Метод match_template застосовує нормалізовану кореляцію до пар сірорадієнтних зображень (рис. 2.3).

```
# Initial template matching for a baseline comparison
template_score, template_loc = self.template_matching(map_img, search_img)
if template_score > threshold:
    best_score = template_score
    best_loc = template_loc

# Use a multi-threaded approach for deep feature extraction
for scale in scales:
    # Skip invalid scales
    if int(h*scale) >= map_h or int(w*scale) >= map_w or int(h*scale) <= 0 or int(w*scale) <= 0:
        continue

    # Resize search image for this scale
    scaled_search = cv2.resize(search_img, (int(w*scale), int(h*scale)))
    scaled_h, scaled_w = scaled_search.shape[:2]

    # Extract features for scaled search image
    scaled_search_features = self.extract_features(scaled_search)

    # Create batches queue
    batch_queue = queue.Queue()
    result_queue = queue.Queue()

    # Fill queue with window positions
    for y in range(0, map_h - scaled_h, step_size):
        for x in range(0, map_w - scaled_w, step_size):
            batch_queue.put((x, y, scale))
```

Рисунок 2.3 – Виконання зіставлення ознак.

Основна логіка роботи модуля аналізу зображень зосереджена в методі find_best_match, який відповідає за виявлення найбільш ймовірної відповідності між поточним зображенням, отриманим з камери, та одним із шаблонів еталонної місцевості. Метод реалізує ітеративний підхід до порівняння, що передбачає перевірку кількох варіантів масштабування та просторового розташування шаблону в межах аналізованого кадру. Це дозволяє адаптуватися до змін у відстані до об'єкта, кута огляду та часткового перекриття зображення.

Пошук виконується паралельно в окремих вікнах, а для кожного вікна обчислюється схожість за допомогою ознак, отриманих від нейромережі (рис. 2.4).

```
# Start worker threads
num_workers = min(4, (torch.cuda.device_count() or 1) * 2)
threads = []
for _ in range(num_workers):
    thread = threading.Thread(target=process_windows)
    thread.daemon = True
    thread.start()
    threads.append(thread)

# Process results
scale_windows = ((map_h - scaled_h) // step_size + 1) * ((map_w - scaled_w) // step_size + 1)
scale_processed = 0

while scale_processed < scale_windows:
    try:
        score, (x, y, s) = result_queue.get(timeout=0.1)

        if score > best_score:
            best_score = score
            best_loc = (x, y)
            best_scale = s

        scale_processed += 1
        windows_processed += 1

        # Update progress
        if progress_callback:
            progress_callback(windows_processed, total_windows)

    except queue.Empty:
        if all(not t.is_alive() for t in threads):
            break

# Wait for threads
for thread in threads:
    thread.join()
```

Рисунок 2.4 – Виконання ітеративного пошуку відповідностей.

Таким чином, розроблений модуль відіграє ключову роль у забезпеченні високоточної відповідності між вхідним зображенням і заздалегідь підготовленим шаблоном. Завдяки поєднанню ефективних алгоритмів обробки зображень і адаптивних методів пошуку збігів, система здатна з високою надійністю і швидкістю визначати позицію апарата в просторі. Це, у свою чергу, створює передумови для стабільної та безперервної локалізації навіть в умовах обмеженої доступності традиційних навігаційних джерел, таких як GPS.

2.3 Розробка модуля фільтрації однорідних зображень місцевості

Загальна швидкодія системи навігації є критично важливою складовою для ефективного виконання завдань безпілотним літальним апаратом, оскільки саме безперервне надходження достовірних координат забезпечує стабільну й надійну роботу всієї системи під час польоту.

На першому етапі здійснюється перевірка на наявність порожнього або повністю чорного зображення. Після цього зображення конвертується у відтінки сірого з метою уніфікації подальшого аналізу. Наступним кроком є обчислення глобальної дисперсії яскравості пікселів: якщо її значення менше за встановлений поріг, зображення розцінюється як надмірно однорідне. Після цього здійснюється аналіз контурів із застосуванням методу Кенні – низька щільність виявлених контурів свідчить про брак виразних структур або текстур у кадрі (рис. 2.5).

```
# Check for empty images
if image is None or image.size == 0 or not np.any(image):
    return True, {'reason': 'Empty image'}

# Convert image to grayscale for analysis
if len(image.shape) > 2:
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
else:
    gray = image.copy()

# Calculate original metrics
mean_val = np.mean(gray)
std_val = np.std(gray)
global_variance = std_val ** 2

metrics = {
    'global_variance': global_variance,
    'mean_value': mean_val,
    'std_dev': std_val
}
```

Рисунок 2.5 – Аналіз на однорідність та наявність контурів.

Додатково застосовується аналіз локальної дисперсії. Для цього зображення розбивається на блоки фіксованого розміру, у кожному з яких окремо розраховується дисперсія яскравості. Якщо середнє значення дисперсії по всіх блоках не перевищує визначений поріг, кадр визнається однорідним також на локальному рівні, що вказує на відсутність значущих особливостей у різних ділянках зображення (рис. 2.6).

```
# If we have enough lines, run additional checks to confirm
# Calculate edge density
edge_density = np.sum(edges_medium > 0) / (gray.shape[0] * gray.shape[1])
metrics['edge_density'] = edge_density

# Even with sufficient lines, if edge density is still very low
if edge_density < edge_threshold and line_count < threshold * 1.5:
    metrics['reason'] = 'Low edge density despite some lines'
    return True, metrics

# Check if the lines are all in one direction (like field rows)
if lines is not None and len(lines) > 0:
    try:
        # Calculate angles of lines
        angles = []
        for line in lines:
            x1, y1, x2, y2 = line[0]
            if x2 - x1 != 0:  # Avoid division by zero
                angle = np.arctan((y2 - y1) / (x2 - x1)) * 180 / np.pi
                angles.append(angle)

        # Check if angles are clustered (parallel lines)
        if angles:
            angles = np.array(angles) % 180
            hist, _ = np.histogram(angles, bins=18, range=(0, 180))
            dominant_direction = np.max(hist) / len(angles) if len(angles) > 0 else 0

            metrics['dominant_direction_ratio'] = dominant_direction

            # If many lines point in the same direction (like field patterns)
            if dominant_direction > 0.6 and line_count < threshold * 2:
                metrics['reason'] = 'Lines primarily in one direction (field-like pattern)'
                return True, metrics
    except Exception:
        pass
```

Рисунок 2.6 – Обчислення локальної дисперсії.

У подальшому проводиться текстурний аналіз із використанням функціоналу бібліотеки skimage.feature. В межах цього аналізу розраховуються такі характеристики, як контрастність, однорідність і несхожість [21]. Високі показники однорідності в поєднанні з низьким контрастом є ознаками слабко вираженої текстури, що вказує на низьку придатність зображення до навігаційного аналізу (рис. 2.7).

```
# Pre-process images - handle different color channels
processed_images = []
for img in images:
    if img is None or img.size == 0:
        continue

    # Ensure correct color channels
    if len(img.shape) == 2:
        img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
    elif img.shape[2] == 4:  # Handle RGBA
        img = cv2.cvtColor(img, cv2.COLOR_RGBA2RGB)

    processed_images.append(img)

if not processed_images:
    return []

# Create batch tensor
batch_tensors = torch.stack([self.transform(img) for img in processed_images]).to(self.device)

# Extract features
with torch.no_grad():
    if hasattr(self.model, 'forward_features'):
        features = self.model.forward_features(batch_tensors)
        # Global average pooling
        if len(features.shape) == 4:  # If features are [B, C, H, W]
            features = torch.mean(features, dim=[2, 3])
    else:
        features = self.model(batch_tensors)
```

Рисунок 2.7 – Виконання текстурного аналізу.

Завершальним етапом аналізу є проведення частотного розкладу зображення, що здійснюється за допомогою швидкого перетворення Фур’є (ШПФ) – одного з

найпотужніших інструментів для виявлення просторових частот у цифровому зображенні. Отриманий спектр дозволяє оцінити розподіл енергії між різними частотними компонентами сцени. З метою зосередження уваги на високочастотних елементах, які відповідають за дрібні деталі та текстуру, з центральної зони спектра, де переважають низькочастотні складові, вилучається інформація, що несе загальну структуру та великомасштабні елементи (рис. 2.8).

```
h, w = search_img.shape[:2]
x, y = template_loc

# Expand ROI by 25% in all directions for verification
roi_x = max(0, x - w//4)
roi_y = max(0, y - h//4)
roi_w = min(map_img.shape[1] - roi_x, w * 1.5)
roi_h = min(map_img.shape[0] - roi_y, h * 1.5)

roi = map_img[roi_y:int(roi_y+roi_h), roi_x:int(roi_x+roi_w)]

# Use deep features on ROI with finer step size
roi_result = self.find_best_match(roi, search_img, step_size=step_size//2, threshold=threshold,
                                  progress_callback=progress_callback)

if roi_result['found']:
    rx, ry = roi_result['location']
    # Adjust coordinates to global image space
    global_x = roi_x + rx
    global_y = roi_y + ry

    result_img = map_img.copy()
    s_h, s_w = search_img.shape[:2]
    cv2.rectangle(result_img, (global_x, global_y),
                  (global_x + s_w, global_y + s_h), (0, 255, 0), 3)
    cv2.putText(result_img, f"Score: {roi_result['score']:.4f}",
                (global_x, global_y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
```

Рисунок 2.8 – Виконання швидкого перетворення Фур’є.

У разі, якщо жоден із зазначених критеріїв не підтверджує одноманітність зображення, функція повертає негативне рішення щодо його виключення, що означає наявність достатнього рівня текстурної, структурної та частотної інформативності для подальшої обробки. Запропонований підхід виступає ефективним багаторівневим фільтром, орієнтованим на виключення малозначущих

або порожніх кадрів, що у підсумку дозволяє суттєво оптимізувати використання обчислювальних ресурсів системи навігації.

2.4 Висновки

У третьому розділі було проведено обґрунтування обраних технологій та засобів розробки для імплементації програмних модулів. Було обрано мову програмування Python для реалізації обчислювальних модулів та модулю інтерфейсу як найбільш доцільну. Для розробки інтерфейсу було використано бібліотеку Tkinter. Для виконання обчислень координат було використано нейронну модель Dino. У розділі також було детально описано імплементацію алгоритмів фільтрації зображень однорідної місцевості та обчислення координат.

3 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

3.1 Аналіз вхідних даних системи

Програмний модуль навігації ґрунтується на використанні нейронних мереж для аналізу зображень і визначення координат об’єктів. Основним засобом для реалізації є мова програмування Python, яка має широкий спектр бібліотек для роботи з нейромережами та комп’ютерним зором. У процесі розробки використовується бібліотека PyTorch, яка забезпечує навчання та застосування нейронних мереж, а також модель DINO для сегментації та виявлення ключових точок на зображеннях. Вибір моделі DINO обґрунтований її здатністю ефективно працювати у безнаглядному режимі, що дозволяє використовувати її для аналізу зображень без попередньо підготовлених розмічених даних [22].

Система включає кілька основних компонентів, кожен з яких виконує свою функцію. Перший модуль – це інтерфейс користувача, що дозволяє завантажувати тестові зображення та переглядати результати обробки у зручному форматі. Для його реалізації використовується бібліотека Tkinter, яка забезпечує створення графічного інтерфейсу без необхідності застосування сторонніх фреймворків. Важливою особливістю є запуск інтерфейсу в окремому потоці, що запобігає зависанню програми під час інтенсивних обчислень, пов’язаних з обробкою зображень. Завдяки цьому користувач зможе взаємодіяти з інтерфейсом у реальному часі навіть під час виконання складних обчислень.

Другий ключовий компонент – це модуль обробки зображень, який базується на використанні нейронної моделі DINO. Основне завдання цього модуля полягає у виявленні ключових точок на зображенні та аналізі їхнього розташування. Виявлені опорні точки служать основою для подальших розрахунків координат цілі. Аналіз зображення здійснюється шляхом розпізнавання характерних особливостей об’єктів та їхнього просторового розташування, що дозволяє визначати положення камери відносно навколишнього середовища.

Якщо вхідне зображення не містить достатньої кількості орієнтирів, тобто є надто однорідним, система виконує попередній аналіз. У такому випадку спеціальний алгоритм оцінює структуру зображення та ухвалює рішення про доцільність подальшої обробки. Це дозволяє уникнути зайвих обчислень та покращити швидкодію системи, що критично важливо при використанні в режимах реального часу.

Швидкодія модуля обчислення координат є одним із ключових аспектів розробки. Для забезпечення ефективної роботи система оптимізована таким чином, щоб мінімізувати часові витрати на обробку кожного кадру. Це особливо важливо у випадках використання в безпілотних літальних апаратах, де кожна затримка в обробці може впливати на точність навігації. Оптимізація досягається завдяки використанню апаратного прискорення на графічних процесорах, що підтримують технологію CUDA [23].

Розробка програмного забезпечення ведеться у середовищі Visual Studio Code, яке забезпечує зручне написання, тестування та налагодження коду. Проектування модулів здійснюється таким чином, щоб вони були максимально гнучкими та могли бути інтегровані в інші програмні системи без значних змін у коді.

Тестування програмного модуля включає кілька етапів. Спочатку проводиться перевірка роботи кожного окремого компонента, щоб впевнитися у його коректному функціонуванні. Далі здійснюється комплексне тестування всієї системи, під час якого моделюються різні сценарії роботи, включаючи використання різних типів зображень та умов освітлення. Основна мета тестування – забезпечити стабільну та безвідмовну роботу системи у будь-яких умовах.

Таким чином, розробка програмного модуля навігації базується на використанні сучасних методів обробки зображень та нейронних мереж, що дозволяє досягти високої точності визначення координат. Вибір Python як основної мови програмування забезпечує гнучкість та широкий вибір бібліотек, а

використання PyTorch та моделі DINO надає можливість працювати з зображеннями ефективно та без потреби у попередньому маркуванні даних.

3.2 Розробка інтерфейсу програмного додатку

Ключовим аспектом під час розробки модуля інтерфейсу стало поєднання інтуїтивної зрозумілості для користувача з наявністю розширеного функціоналу. Такий підхід забезпечує одночасно гнучкість системи налаштувань та зручність взаємодії зі складними параметрами [24]. Основним кольором графічного інтерфейсу було обрано темно-сірий відтінок, який є візуально нейтральним і не викликає перевантаження уваги користувача. Після запуску програма відкривається у стандартному стані із типовими параметрами для пошуку та обробки зображень, як показано на рисунку 2.1.

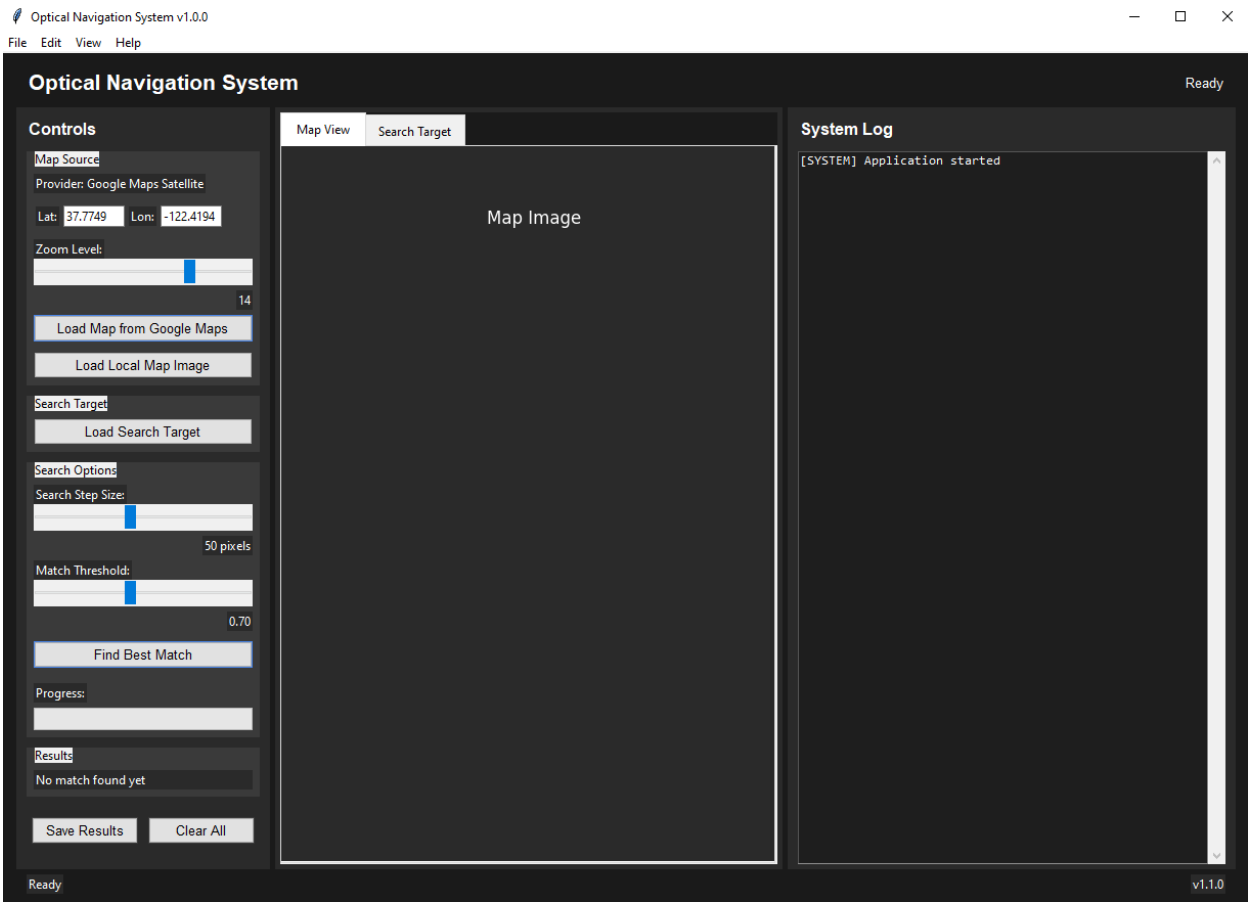


Рисунок 3.1 – Початковий стан роботи застосунку.

Для покращення взаємодії з програмою було реалізовано графічне меню, яке дублює основні функції застосунку. Зокрема, через нього можна завантажити карту, додати цільове зображення місцевості або зберегти результати виконаних операцій. Інтерфейс меню відображено на рисунку 3.2.

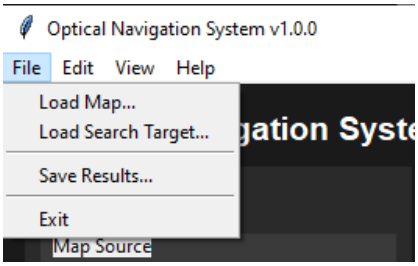


Рисунок 3.2 – Меню програмного застосунку.

Завантаження фрагмента карти може відбуватися як з локального пристрою, так і за допомогою підключення до супутникового сервісу Google Maps через API. У разі вибору другого варіанту користувач має вказати центрову координату області інтересу. Після ініціювання завантаження відбувається формування відповідного зображення карти шляхом надсилання ряду запитів до серверу в межах заданої області.

Для подальших обчислень користувач повинен завантажити локальний файл із зображенням місцевості, на основі якого здійснюється визначення координат. У разі виконання обох попередніх умов – завантаження карти та додавання зображення місцевості – користувач може розпочати процедуру пошуку збігу натисканням відповідної кнопки. В інтерфейсі реалізовано індикатор виконання, що візуалізує прогрес обробки зображень. У результаті виконаного обчислення відображається фрагмент карти з виділеною знайденою ділянкою, а також відсоткове значення рівня збігу, як представлено на рисунку 3.3.

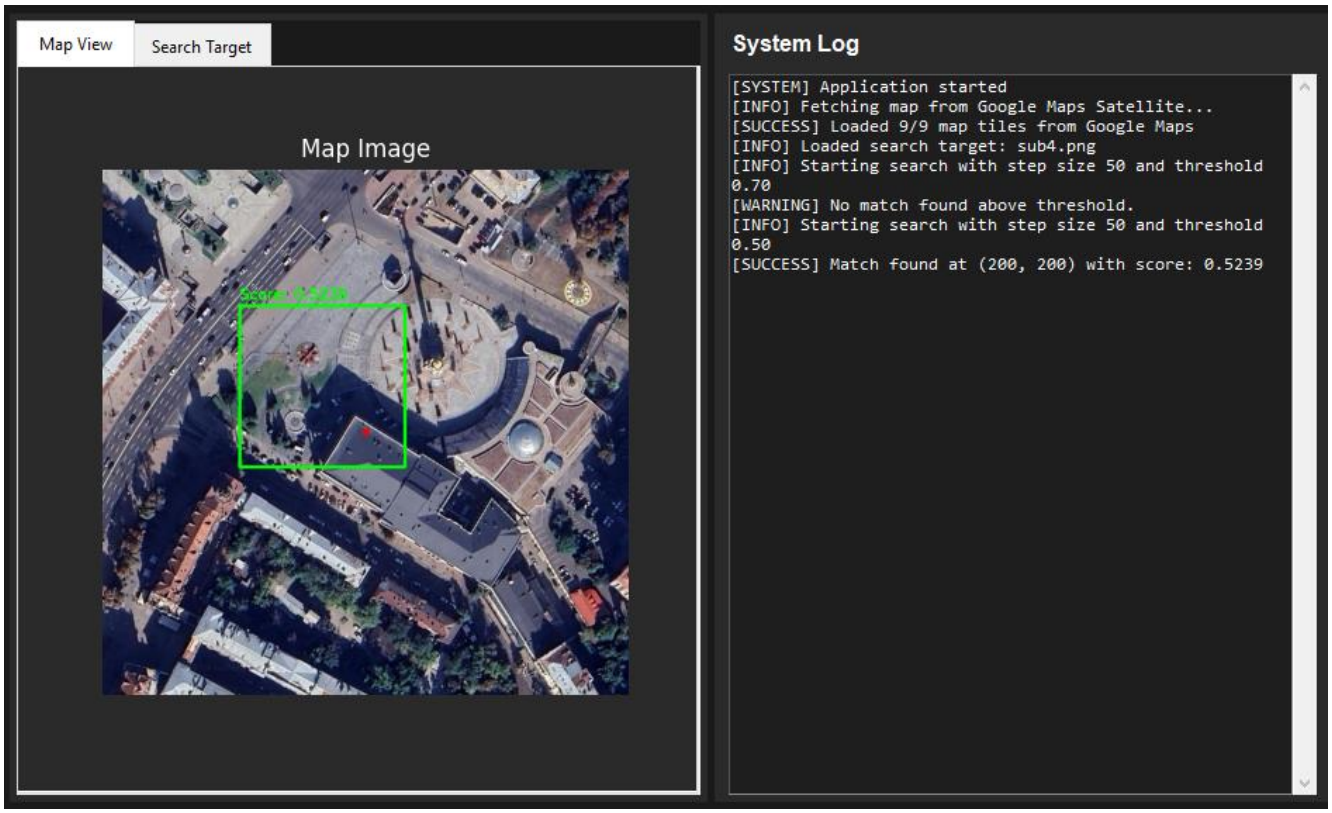


Рисунок 3.3 – Результат виконання обчислень координат вхідного зображення.

Інтерфейс модуля також передбачає можливість гнучкого налаштування параметрів, що впливають як на процес завантаження карти, так і на сам алгоритм пошуку. Зокрема, користувач може змінювати масштаб відображення місцевості, що впливає на деталізацію завантажених зображень. У процесі пошуку координат дозволено налаштування розміру обчислювального блоку у пікселях: при зменшенні цього параметра зростає точність обчислення, однак знижується швидкодія системи; при збільшенні, навпаки, підвищується продуктивність, але зменшується точність результату. Крім того, користувач має змогу встановити мінімальний поріг впевненості у відсотках, що дозволяє фільтрувати малоточні результати. Інтерфейс конфігураційного меню подано на рисунку 2.4.

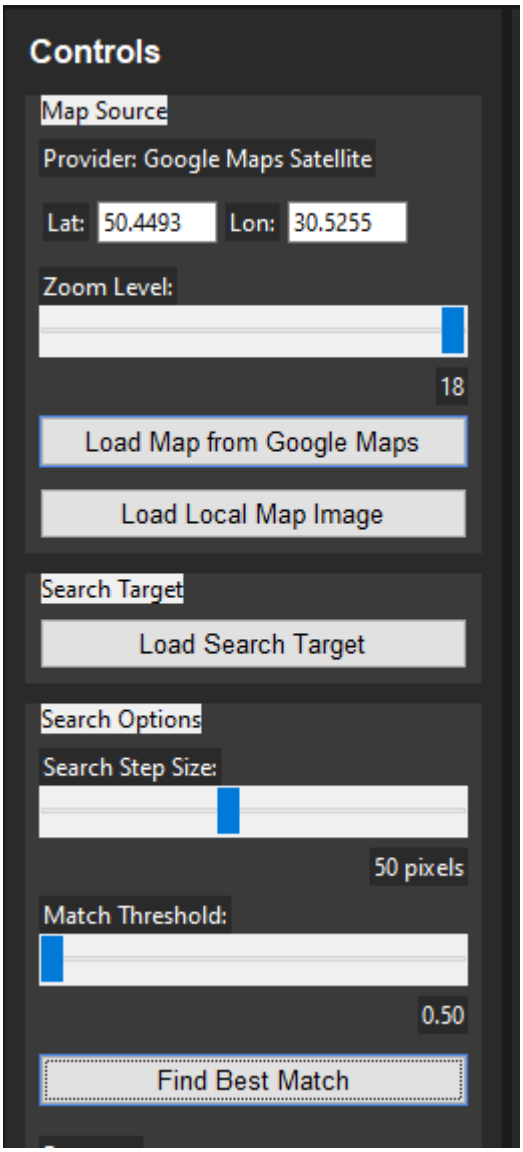


Рисунок 3.4 – Конфігураційна панель застосунку.

Для забезпечення прозорості процесів, які відбуваються у застосунку, було реалізовано панель логування. Ця панель інформує користувача про загальний стан роботи застосунку, прогрес завантаження карти, а також повідомляє про успішність або невдачу виконаних дій. У разі виникнення помилок система відображає відповідне повідомлення з поясненням причин, що полегшує процес діагностики та усунення можливих проблем, таких як неправильне зображення або некоректне значення порогу впевненості. Візуальне відображення інформаційної панелі наведено на рисунку 3.5.

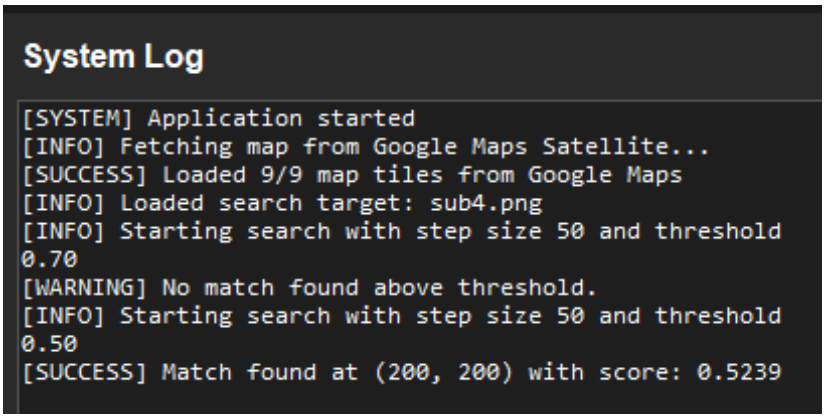


Рисунок 3.5 – Панель інформаційних повідомлень застосунку.

Додатково було реалізовано інформаційні вікна, що містять довідкові дані про застосунок, опис функціоналу, а також інструкції з користування. Одне з таких вікон містить відомості про розробника, версію програмного забезпечення та інші загальні дані, що доступні для кінцевого користувача. Вікно з інформацією про розробника показано на рисунку 3.6.

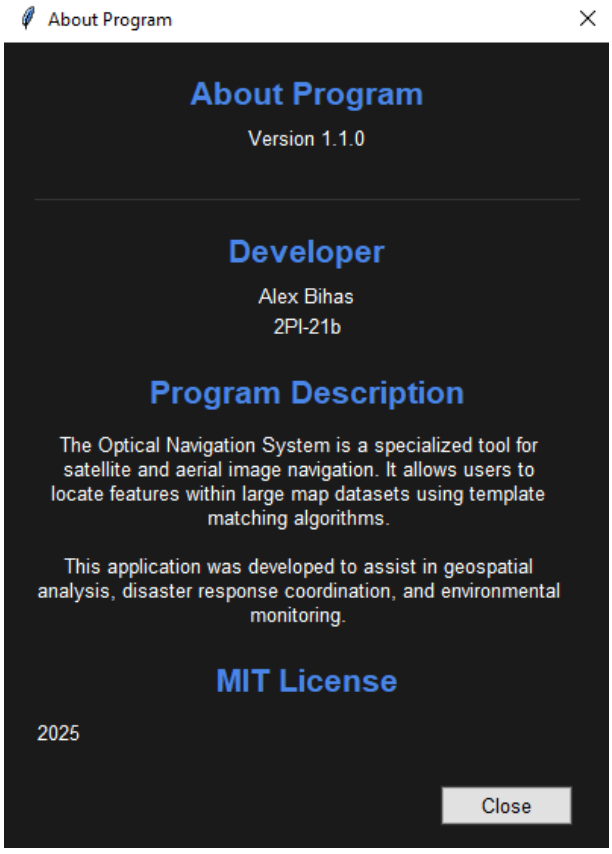


Рисунок 3.6 – Інформаційне вікно про розробника.

Інформаційне вікно інструкції користувача містить три підсекції. Підсекції включають в себе кроки для початку роботи з застосунком, опис джерел фрагментів карт та пояснення окремих технічних аспектів роботи застосунку.

У підсекції про кроки для початку роботи з застосунком описано загальний потік роботи застосунку, опис процесів вибору фрагментів карт та вихідних результатів. У секції про фрагменти карт детальніше описано варіанти використання карт та параметри, що враховуються при обробці. У секції про технічні аспекти описано, як параметри розміру кроку ітерації та мінімального порогу впевненості впливають на роботу застосунку. Також надано додатковий перелік кроків, що можуть покращити результати обробки. Інформаційне вікно документації представлено на рисунку 3.7.

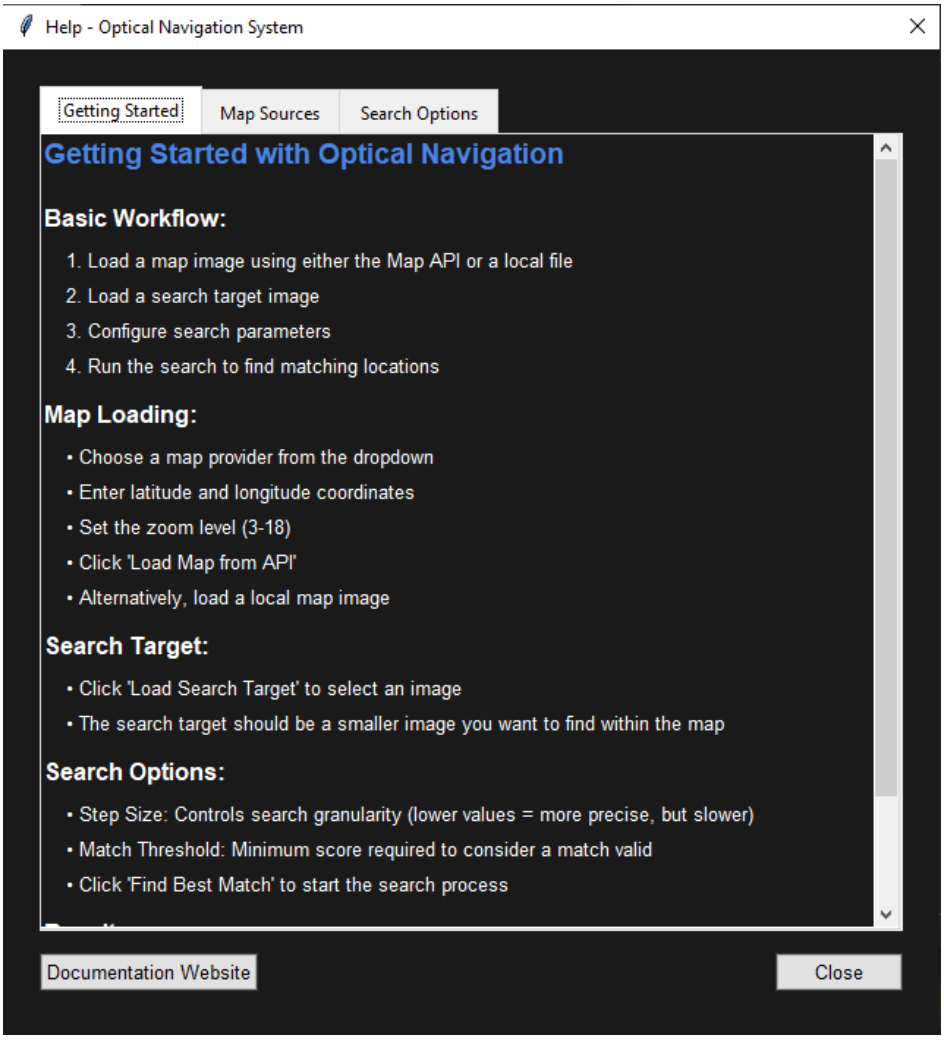


Рисунок 3.7 – Інформаційне вікно документації застосунку.

Розробка інтерфейсного модуля здійснювалася в середовищі Visual Studio Code з використанням мови програмування Python та графічної бібліотеки Tkinter. Для досягнення модульності та зручності супроводу проєкту, інтерфейсну частину було винесено в окремий набір файлів, що забезпечує можливість автономного функціонування обчислювального модуля та подальшої інтеграції.

3.3 Розробка моделі системи

Для забезпечення повноти розуміння функціонування системи програмних модулів навігаційного забезпечення БПЛА було розроблено UML-діаграму діяльності (рис. 3.7), яка ілюструє послідовність дій користувача та внутрішню логіку роботи програмного забезпечення.

Процес взаємодії користувача із застосунком починається з ініціалізації, що передбачає обов’язкове завантаження карти місцевості. Зображення може бути як завантажене з Google Satellite API, так і імпортована з пристрою користувача. Для завантаження карти з зовнішнього API необхідно вказати центральні координати зони, в якій буде відбуватись пошук. За такого сценарію використання, алгоритм здатний прорахувати абсолютні значення координат зображення місцевості. У випадку використання фрагменту карти з пристрою, обчислювальний алгоритм, у випадку успішного виконання обчислень, поверне значення у пікселях, відносні до самого зображення карти.

Фрагмент карти слугує еталонним зображенням, яке буде використовуватись як основа для зіставлення з вхідними зображеннями. Одразу після завантаження карти користувач має імпортувати вхідне зображення місцевості. Це зображення проходить через первинний етап перевірки, зосереджений на оцінці його інформативності. На цьому етапі здійснюється аналіз однорідності, що передбачає розрахунок статистичних характеристик яскравості або контрастності зображення. Якщо зображення визначається як надто однорідне, тобто не містить достатньої кількості візуально-інформативних ознак, система виводить повідомлення про

неможливість здійснення подальшого аналізу і повертається до початкового стану очікування нового зображення.

У випадку успішного проходження фільтрації зображення передається до модуля обробки, який виконує серію послідовних операцій для пошуку відповідної області на карті. На цьому етапі здійснюється нормалізація кольорового простору вхідного зображення, його масштабування до уніфікованих розмірів, а також обчислення векторів ознак за допомогою попередньо натренованої моделі глибокого навчання. Після цього система виконує зіставлення ознак вхідного зображення з попередньо обчисленими ознаками ділянок еталонної карти. У процесі зіставлення застосовуються евристичні або нейронні методи оцінки подібності, наприклад, косинусна метрика, евклідова відстань або інші функції втрат.

Якщо в результаті зіставлення виявлено область, яка має достатній рівень подібності до вхідного зображення, система виводить відповідне зображення карти із виділеною прямокутною рамкою області збігу. До рамки додається числовий показник впевненості, що демонструє, наскільки сильно модель переконана у правильності знайденої відповідності. Це дозволяє оператору або користувачу самостійно інтерпретувати достовірність результатів.

Після завершення процесу обробки користувачу надається можливість експортувати результати у вигляді зображення із нанесеною рамкою відповідності. Збереження здійснюється у стандартному графічному форматі (наприклад, PNG або JPEG), що забезпечує зручність подальшого аналізу або звітування. Завершальним етапом є очищення попередньо використаних даних: карта та вхідне зображення автоматично видаляються з оперативної пам’яті, а інтерфейс переходить у режим очікування нових даних. Це забезпечує готовність системи до обробки наступної серії зображень і запобігає накопиченню залишкових даних, які можуть впливати на продуктивність та інформаційну перевантаженість застосунку. Панель логування зберігає попередні повідомлення для можливості подальшого аналізу попередніх результатів чи помилок.

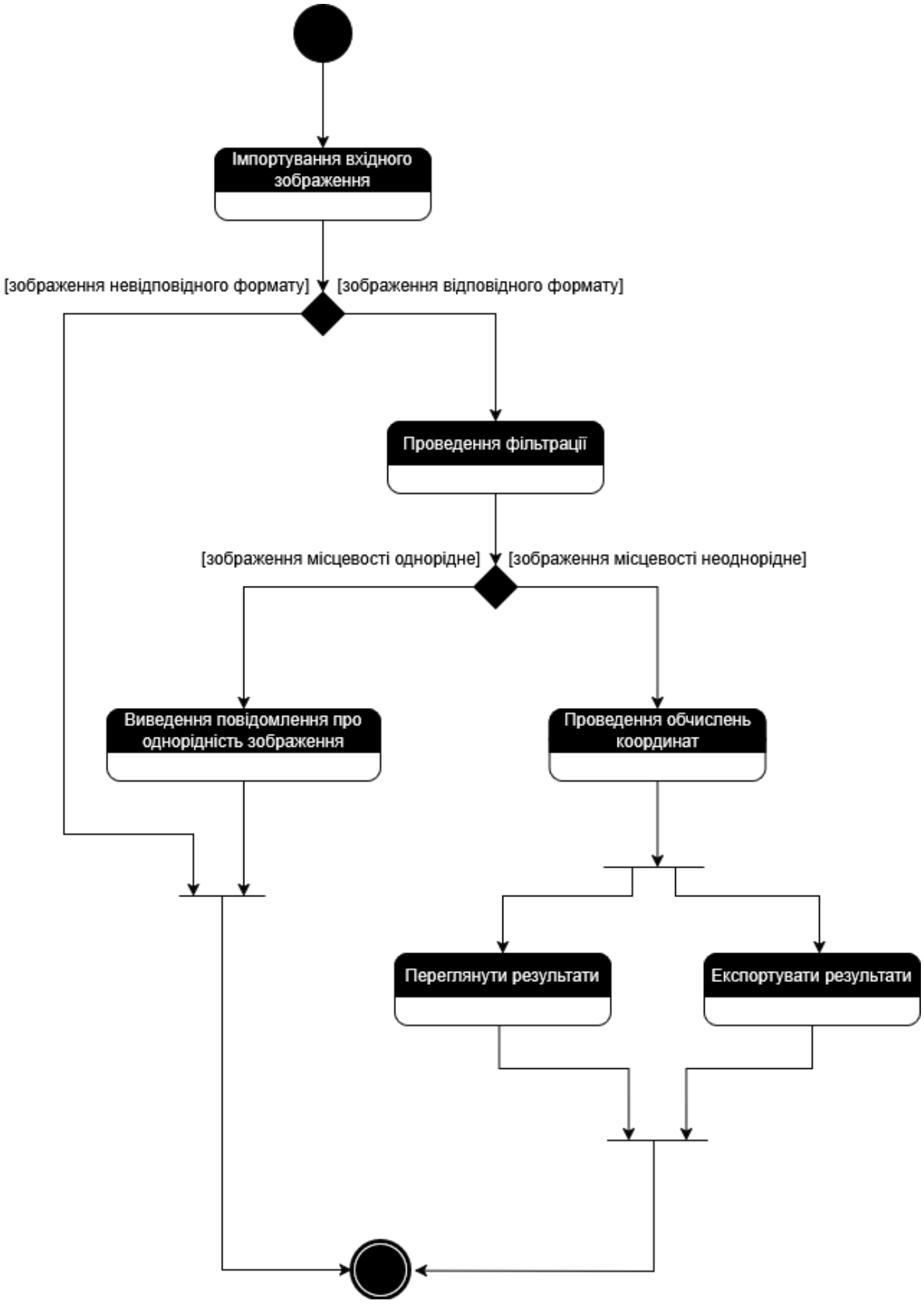


Рисунок 3.7 – Діаграма діяльності застосунку.

Діаграму діяльності було розроблено в середовищі diagrams.net.

3.4 Розробка алгоритмів роботи системи

Загальний алгоритм роботи програмного модуля навігації безпілотного літального апарата (БПЛА) передбачає послідовне виконання низки етапів, що забезпечують визначення координат за допомогою аналізу зображень місцевості (рис. 3.8). На початку роботи системи ініціалізує нейронну мережу, зчитує конфігураційні параметри та виконує запуск модулю інтерфейсу. У подальшому користувач здійснює завантаження карти місцевості, яка слугує еталонною основою для подальших порівнянь. Карта повинна бути попередньо обробленою та мати просторову прив’язку, що дозволяє зіставляти зображення, отримані в реальному часі, з її фрагментами. Просторова прив’язка відбувається у випадку завантаження карти із зовнішнього API шляхом вказування користувачем центральних координат області, в якій буде відбуватись пошук. У випадку використання зображення з локального девайсу, просторової прив’язки не відбувається та результатом будуть відносні координати. Після завантаження карти користувач імпортує зображення місцевості, яке було отримане під час польоту. Це зображення буде проаналізоване для визначення його розташування на завантаженій карті.

Далі зображення проходить етап попередньої обробки, що включає фільтрацію шумів, вирівнювання яскравості та контрасту, а також зменшення роздільності з метою підвищення швидкодії подальших обчислень. Після цього виконується аналіз однорідності зображення. Основним критерієм оцінки є локальна дисперсія – для кожної ділянки зображення розраховується дисперсія яскравості, яка дозволяє оцінити, наскільки зображення насичене деталями. У разі якщо дисперсія виявляється нижчою за встановлене порогове значення, вхідне зображення вважається надто однорідним, а отже – непридатним для подальшого аналізу. Додатково виконується текстурний аналіз, що дозволяє оцінити ступінь структурованості зображення, а також здійснюється швидке перетворення Фур’є для вивчення частотних характеристик. За результатами цих обчислень система приймає рішення про доцільність подальшої обробки зображення.

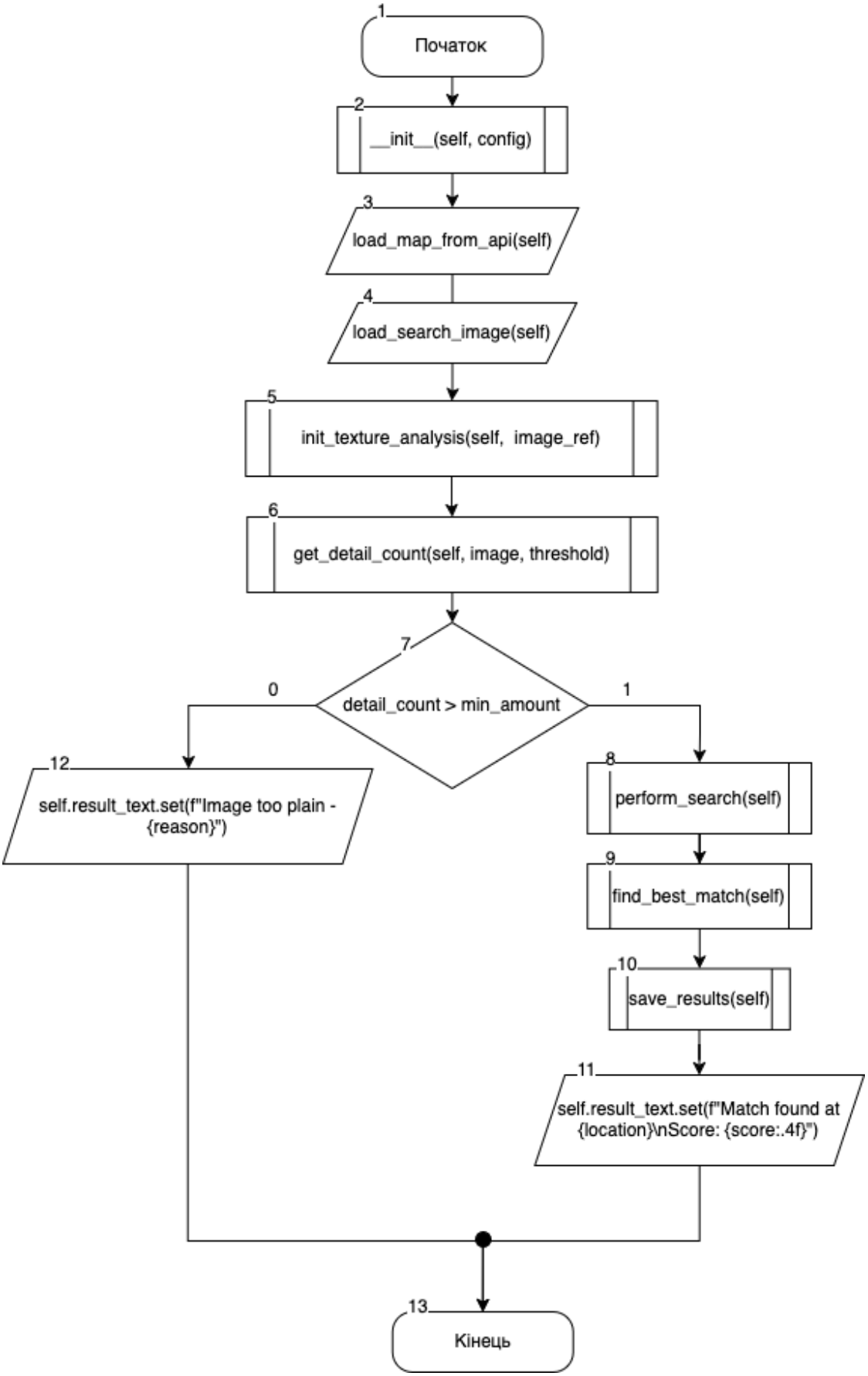


Рисунок 3.8 – Блок-схема роботи застосунку.

У разі, якщо вхідне зображення містить достатню кількість інформативних елементів, тобто чітко виражених структур, текстур та характерних ознак, які можуть бути використані для подальшого аналізу, ініціюється процес запуску модуля визначення координат. На початковому етапі зображення проходить процедуру попередньої обробки, зокрема нормалізації. Цей етап є критично важливим, оскільки дозволяє мінімізувати вплив варіацій, які можуть виникати внаслідок змін масштабу, варіацій яскравості, освітлення або орієнтації об'єкта в полі зору камери. Завдяки нормалізації забезпечується уніфікованість вхідних даних, що, своєю чергою, підвищує достовірність і точність подальшого аналізу.

Наступним етапом є процес зіставлення ознак між вхідним зображенням та еталонною картою місцевості. Для цього застосовується ітеративний підхід до пошуку відповідностей, що базується на використанні алгоритмів різномасштабного аналізу. Такий підхід дозволяє ідентифікувати стабільні відповідності навіть у разі часткових змін вигляду сцени або спотворень, зумовлених перспективою чи кутом зйомки. Отримані в результаті аналізу відповідності між окремими ознаками зображення та карти дають змогу з достатньою точністю визначити положення зображення в межах геоприв’язаного простору.

Після встановлення геометричної відповідності система переходить до етапу обчислення просторових координат. З огляду на прив’язку карти до географічної системи координат, результатом цього етапу є отримання точного географічного положення апарата, яке може бути використане як для навігації, так і для інших цілей автономного управління.

Таким чином, запропонований підхід забезпечує високу надійність і точність визначення координат безпілотного літального апарата на основі візуальної інформації.

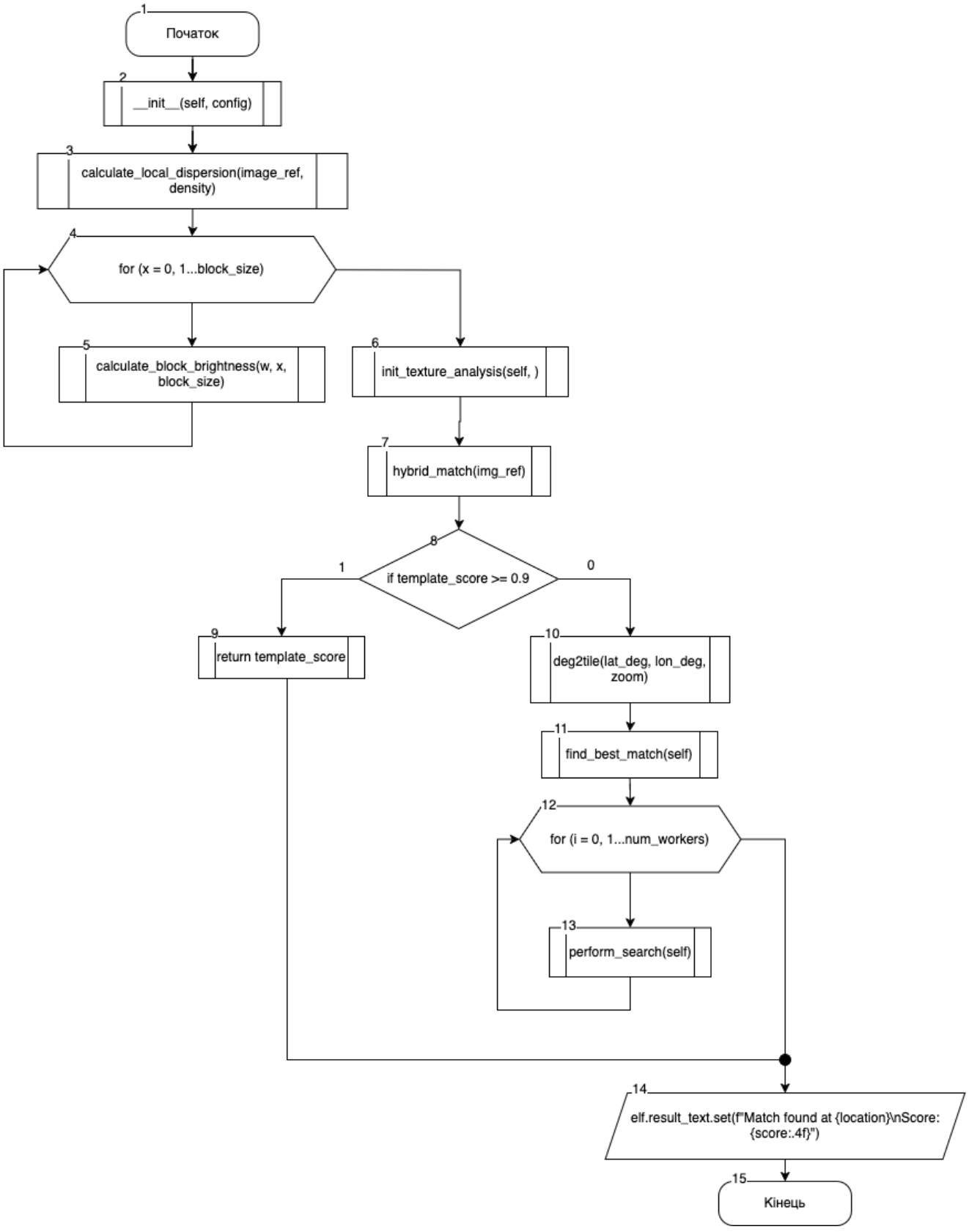


Рисунок 3.9 – Блок-схема роботи модуля обчислення координат.

3.5 Висновки

У другому розділі було проведено аналіз вхідних та вихідних даних системи. Також було розглянуто процес розробки модулю інтерфейсу застосунку. Було розроблено алгоритми роботи обчислювального модулю та модулю фільтрації однорідних зображень. Також було описано роботу програмного застосунку з використанням UML діаграм.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування програмного забезпечення є важливим етапом життєвого циклу розробки, метою якого є виявлення помилок, перевірка коректності функціонування всіх компонентів та забезпечення стабільної і надійної роботи системи. Це процес, який дозволяє оцінити якість програмного продукту та впевнитися у його відповідності до поставлених функціональних та нефункціональних вимог [25]. Проведення тестування дає змогу вчасно виявити потенційні проблеми, які можуть вплинути на досвід кінцевого користувача або призвести до некоректної обробки даних.

Методика тестування програмного забезпечення – це систематизований підхід до організації й проведення тестування, який включає планування тестових робіт, розробку тест-кейсів, визначення критеріїв успішності проходження тестів, безпосереднє виконання тестів, фіксацію результатів, а також аналіз виявлених дефектів [26].

В рамках реалізації цієї методики застосовуються різноманітні види тестування, серед яких можна виділити модульне, функціональне, інтеграційне, системне, регресійне, а також приймальне тестування. Кожен тип тестування має свою специфіку та цільову спрямованість, що дозволяє охопити всі ключові аспекти функціонування програмного продукту.

Крім того, методика може включати як ручне тестування (мануальне), так і автоматизоване, залежно від складності системи, доступного інструментарію та обсягу робіт. При виборі підходів до тестування також беруться до уваги відповідні техніки, серед яких найбільш поширеними є тестування «чорного ящика» (black-box testing) та тестування «білого ящика» (white-box testing). В межах проведеного дослідження було обрано методику тестування типу «білого ящика», яка дозволяє перевірити внутрішню логіку роботи програмного коду.

Для забезпечення повноцінного та обґрунтованого тестування було розроблено низку тест-кейсів, які охоплюють основні функціональні можливості розробленого програмного забезпечення. Кожен тест-кейс містить опис початкових умов, вхідні дані, очікуваний результат, а також фактичний результат виконання тесту. Всі тест-кейси було систематизовано та подано у вигляді таблиці, що дозволяє зручно фіксувати результати перевірок і формувати загальну оцінку якості реалізованого програмного модуля.

Таблиця з тест-кейсами представлена у таблиці 4.1.

Таблиця 4.1 – Тест-кейси.

№	Назва	Методика тестування	Очікуваний результат	Результат
1	Відсутність вхідних зображень	Запустити пошук збігів без попереднього завантаження зображення	Повідомлення про помилку	Виконано (рис. 4.1)
2	Завантаження карти через API з правильними координатами	Ввести дійсні координати, задати масштаб, натиснути "Завантажити"	Карта завантажується відповідно до параметрів	Виконано (рис. 4.2)
3	Введення некоректних координат	Ввести текст або неправильний формат координат	З’являється повідомлення про помилку. Завантаження не відбувається	Виконано

Продовження таблиці 4.1

№	Назва	Методика тестування	Очікуваний результат	Результат
4	Відсутність вхідних зображень	Запустити пошук збігів без попереднього завантаження зображення	Повідомлення про помилку	Виконано (рис. 4.1)
5	Пошук збігу з релевантним зображенням	Завантажити фрагмент карти та відповідне зображення місцевості	Збіг знайдено, область виділено, виведено рівень впевненості	Виконано (рис. 4.4)
6	Пошук збігу з нерелевантним зображенням	Завантажити зображення місцевості, що не відповідає фрагменту карти	Відображено повідомлення про відсутність збігу	Виконано (рис. 4.5)
7	Аналіз однорідного зображення	Завантажити однорідне (неінформативне) зображення	Система повідомляє про непридатність зображення, обробка не виконується	Виконано (рис. 4.6)

Продовження таблиці 4.1

№	Назва	Методика тестування	Очікуваний результат	Результат
8	Кросплатформенність: запуск на MacOS	Запустити застосунок на MacOS	Застосунок працює стабільно	Виконано (рис. 4.7)
9	Асинхронність інтерфейсу під час обробки	Під час інтенсивної обробки взаємодіяти з інтерфейсом	Інтерфейс залишається чутливим, не зависає	Виконано
10	Вплив параметрів threshold і step size на точність і час обробки	Встановити різні значення threshold та step size, виконати обчислення координат	При зменшенні step size — точність зростає, час обробки збільшується.	Виконано

У рамках розробки програмного застосунку модулю оптичної навігації було проведено комплексне тестування, яке включало базову перевірку працездатності інтерфейсу, коректність обробки вхідних даних, а також стабільність взаємодії між інтерфейсом та обчислювальним модулем. Окрема увага приділялася роботі із зовнішніми джерелами, зокрема завантаженню карт через API Google Maps, що потребувало перевірки обробки координат, масштабування, а також ситуацій із відсутністю підключення до мережі або помилками сервера.

Процес тестування також охоплював перевірку логіки завантаження зображень місцевості, правильності ініціалізації процесу пошуку збігів та коректності виведення результатів. Особлива увага приділялася візуальному компоненту — відповідність відображення карти, правильність візуалізації

виділеної області, адекватна робота прогрес-бара та повідомлень системи. Оскільки застосунок передбачає взаємодію з користувачем, важливо було перевірити, наскільки логічно побудований інтерфейс, чи всі дії користувача мають очікувану реакцію з боку системи.

У процесі тестування було імітовано різні сценарії використання, включаючи як типові, так і граничні випадки, наприклад: спробу пошуку без попереднього завантаження зображення, введення некоректних координат, або встановлення надмірно низького порогу впевненості. Такі перевірки дозволили виявити критичні та неочевидні помилки, які могли б залишитися непоміченими під час звичайної експлуатації.

Під час першого етапу тестування функціоналу програмного забезпечення було здійснено перевірку коректності роботи застосунку у випадках, коли користувач не надає необхідних вхідних даних, зокрема — відсутні попередньо завантажені вирізи карти або вхідне зображення місцевості. У таких сценаріях програма повинна інформувати користувача про помилку, не допускаючи подальшого виконання процесів. У результаті тестування було підтверджено, що у разі пропущеного етапу підготовки даних, інтерфейс застосунку виводить відповідне повідомлення про помилку в інформаційному вікні (рис. 4.1), що забезпечує базову захищеність від некоректного використання.

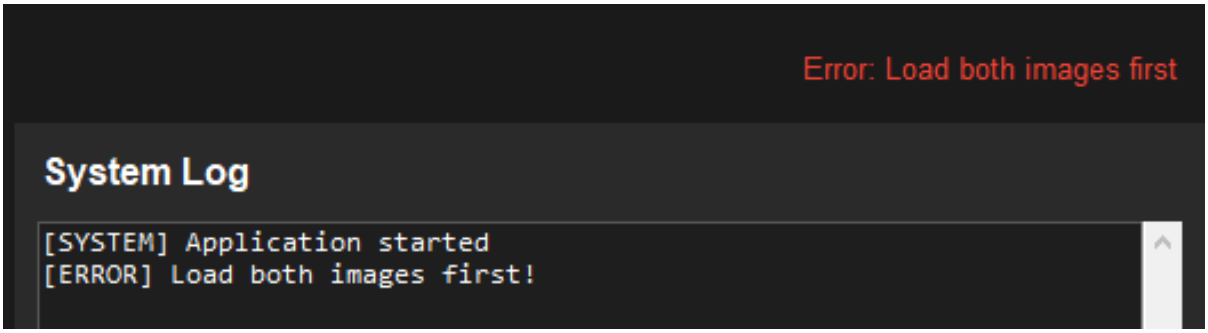


Рисунок 4.1 – Повідомлення про помилку за відсутніх зображень карти та місцевості.

У подальшому етапі тестування була проаналізована робота модуля завантаження зображення карти за допомогою інтерфейсу Google Satellite API. Завантаження здійснюється на основі параметрів конфігурації, до яких входять географічні координати та рівень масштабування (zoom level), який визначає деталізацію зображення. В процесі випробування були протестовані різні значення масштабу і координат, що дозволило перевірити стабільність функціонування механізму завантаження при змінних вхідних параметрах. Результати тестування засвідчили, що програмний продукт здатен отримувати фрагменти карти із зовнішнього джерела із заданою точністю та відповідно до введених координат.

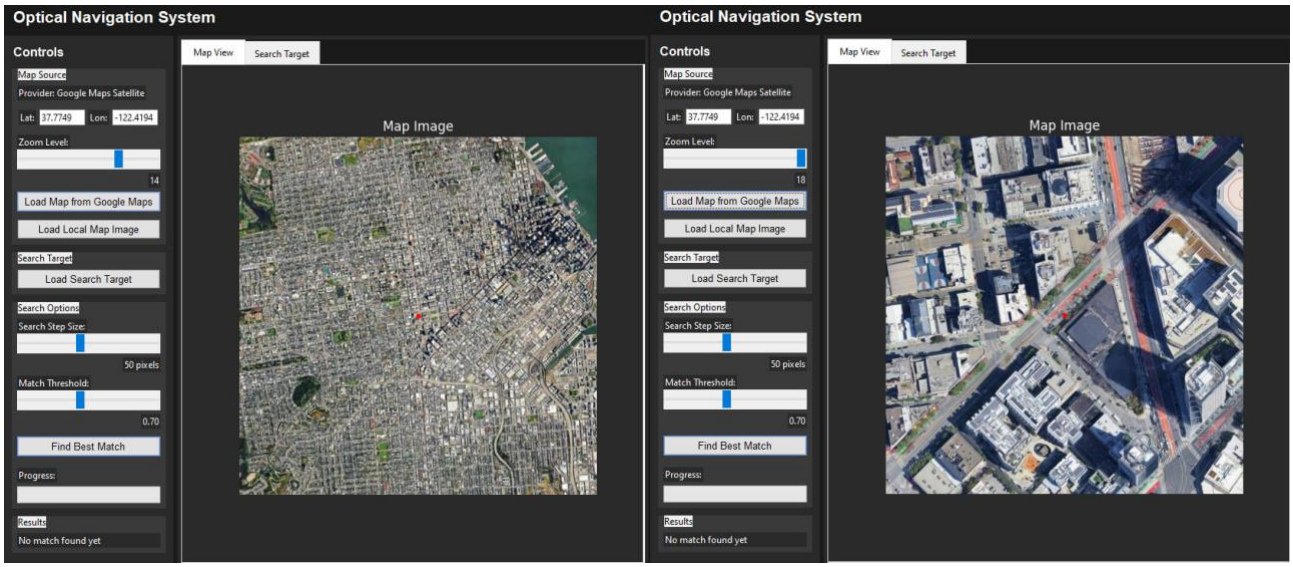


Рисунок 4.2 – Завантажені фрагменти карти різних масштабів.

Окремо було виконано тестування функціоналу, що дозволяє використовувати зображення карти, збережені локально на пристрої користувача. Такий режим роботи забезпечує автономне функціонування застосунку без необхідності підключення до мережі Інтернет. Під час тестування було підтверджено, що програма коректно обробляє локальні зображення, здійснюючи побудову координат на основі референсних фрагментів карти, що зберігаються на диску (рисунок 4.3).

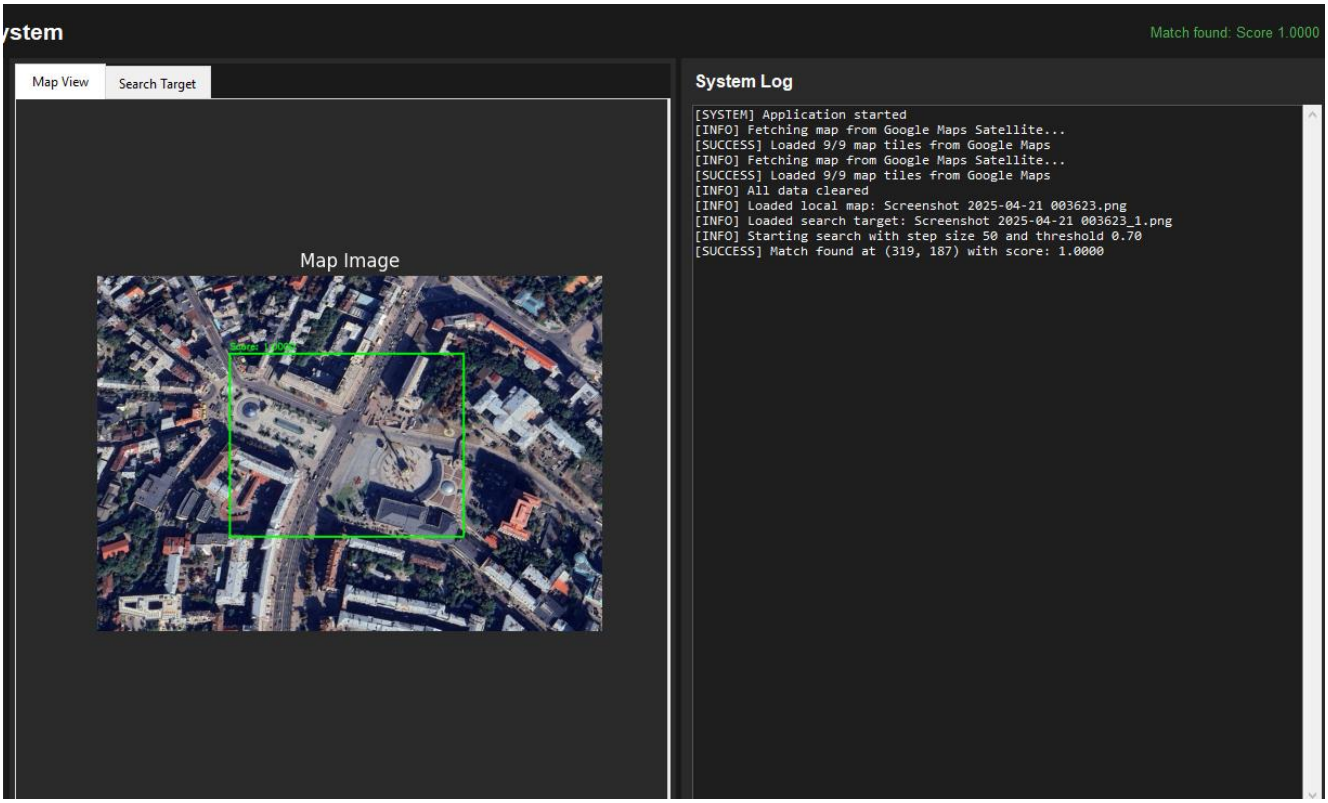


Рисунок 4.3 – Результат обрахунку координат з використанням імпортованого фрагменту карти

З метою оцінки працездатності та коректності функціонування розробленого застосунку було проведено серію тестувань із використанням фрагментів картографічних зображень, отриманих із зовнішнього API. Такі фрагменти імітують реальні умови експлуатації системи, коли дані карти надходять із зовнішніх джерел у динамічному режимі, що є типовим для сучасних автономних навігаційних рішень.

У ході тестування перевірялося, наскільки точно модуль обчислення координат здатен здійснювати виявлення відповідної ділянки на карті на основі вхідного зображення місцевості. У випадку, коли вхідне зображення відповідає певному фрагменту карти, система повинна не лише визначити його положення, а й здійснити виділення знайденої області на карті з візуальним позначенням. Додатково система надає числову оцінку рівня впевненості у відповідності, що є важливим елементом для прийняття рішень у подальшій навігації (рис. 4.4).

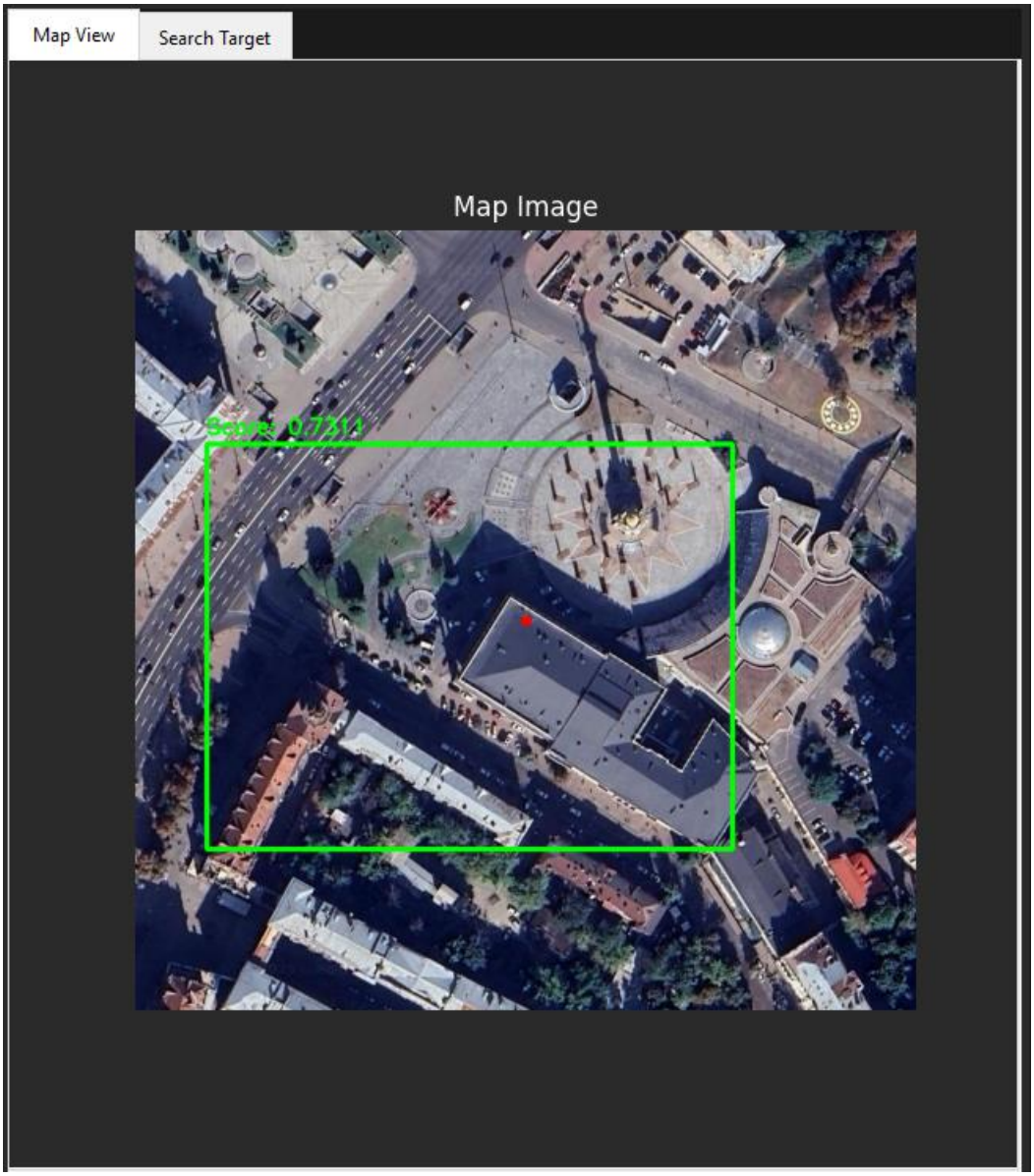


Рисунок 4.4 – Результат обчислень координат з фрагментом карти зовнішнього API.

Також було перевірено сценарій, при якому зображення місцевості не відповідає фрагменту карти. У випадку, коли вхідне зображення помилкове та не визначається як однорідне, коректний сценарій обробки не повинен знайти жодного збігу. (рис. 4.5).

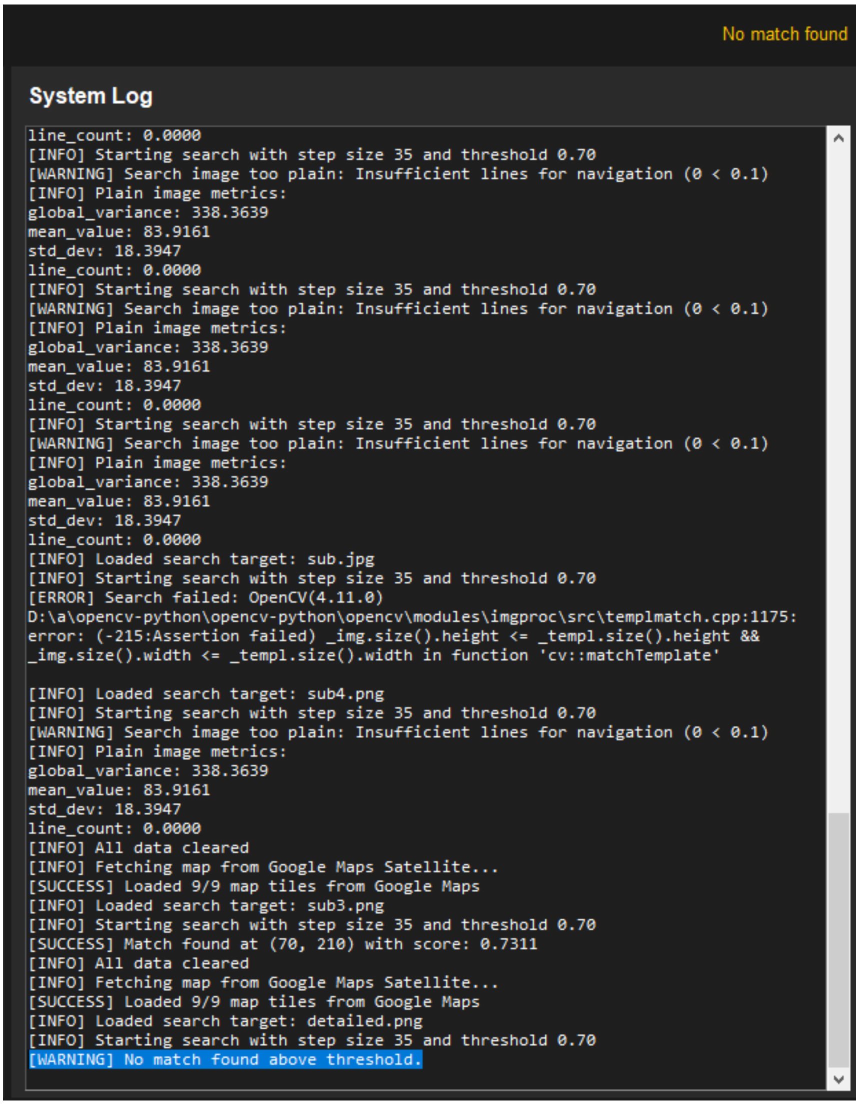


Рисунок 4.5 – Повідомлення про відсутній збіг.

На наступному етапі увага була зосереджена на перевірці роботи алгоритму фільтрації зображень із низьким рівнем інформативності, а саме — однорідних фрагментів місцевості, які не містять виражених особливостей або візуальних маркерів. У разі завантаження такого зображення користувачем, застосунок повинен виявити його недостатню придатність для обчислення координат і не запускати надмірно ресурсоємні процедури подальшої обробки. Проведене тестування підтвердило, що система успішно визначає однорідні зображення та припиняє обчислення, про що свідчить результат, представлений на рис. 4.4.

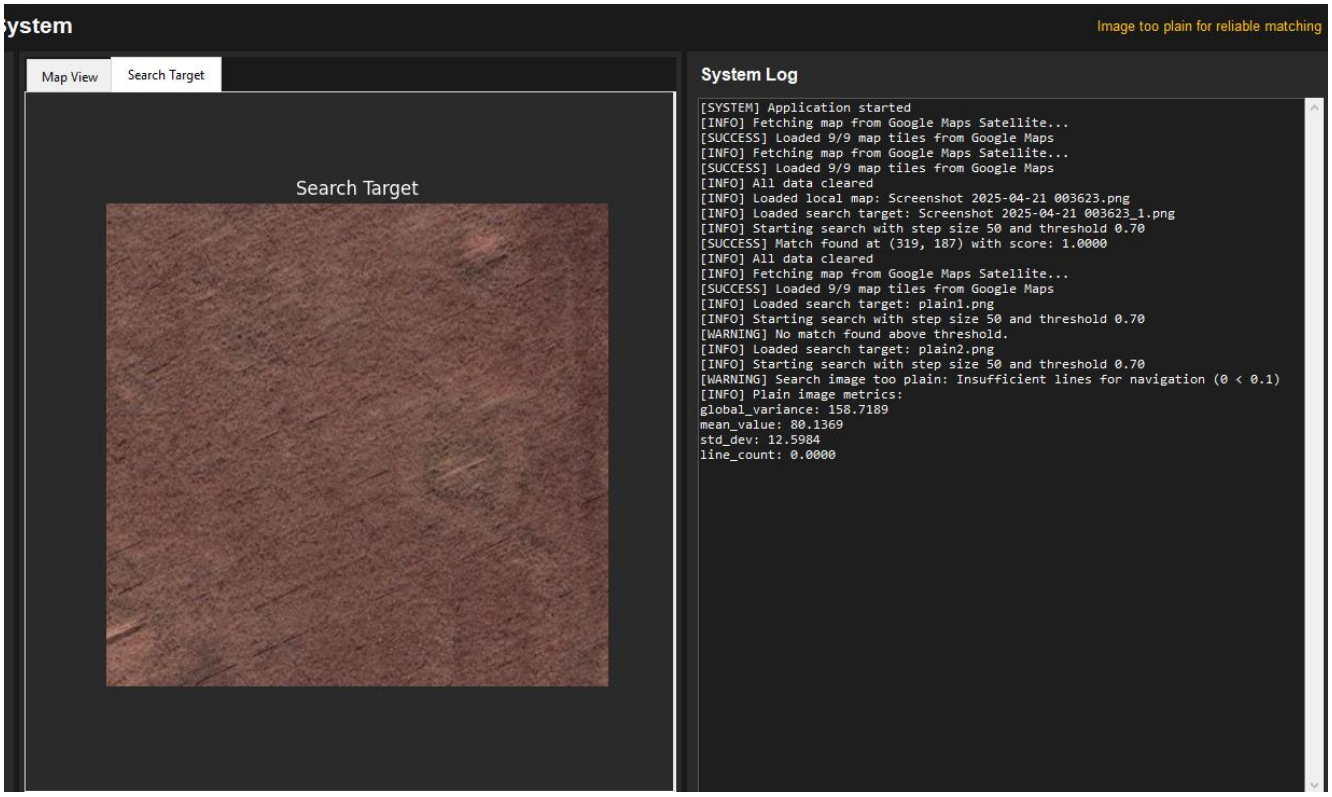


Рисунок 4.6 – Результат обробки однорідного зображення місцевості.

Окремий етап було присвячено перевірці кросплатформенності програмного забезпечення. Було здійснено тестування на трьох основних операційних системах: Windows, macOS та Ubuntu. Завдяки використанню мови програмування Python, яка підтримується всіма зазначеними платформами, програмне забезпечення продемонструвало стабільну роботу без необхідності адаптації коду під кожну систему окремо. Це підтверджує його придатність до експлуатації в різних середовищах без втрати функціональності (рис. 4.7).

При використанні застосунку на платформі macOS обчислювальний модуль здійснює операції з використанням центрального процесора системи [26]. Натомість, при розгортанні на операційній системі Linux, для забезпечення оптимальної швидкодії система вимагає наявності графічного прискорювача з підтримкою технології NVIDIA CUDA, що дозволяє суттєво підвищити ефективність паралельних обчислень та скоротити час виконання ресурсоємних операцій.

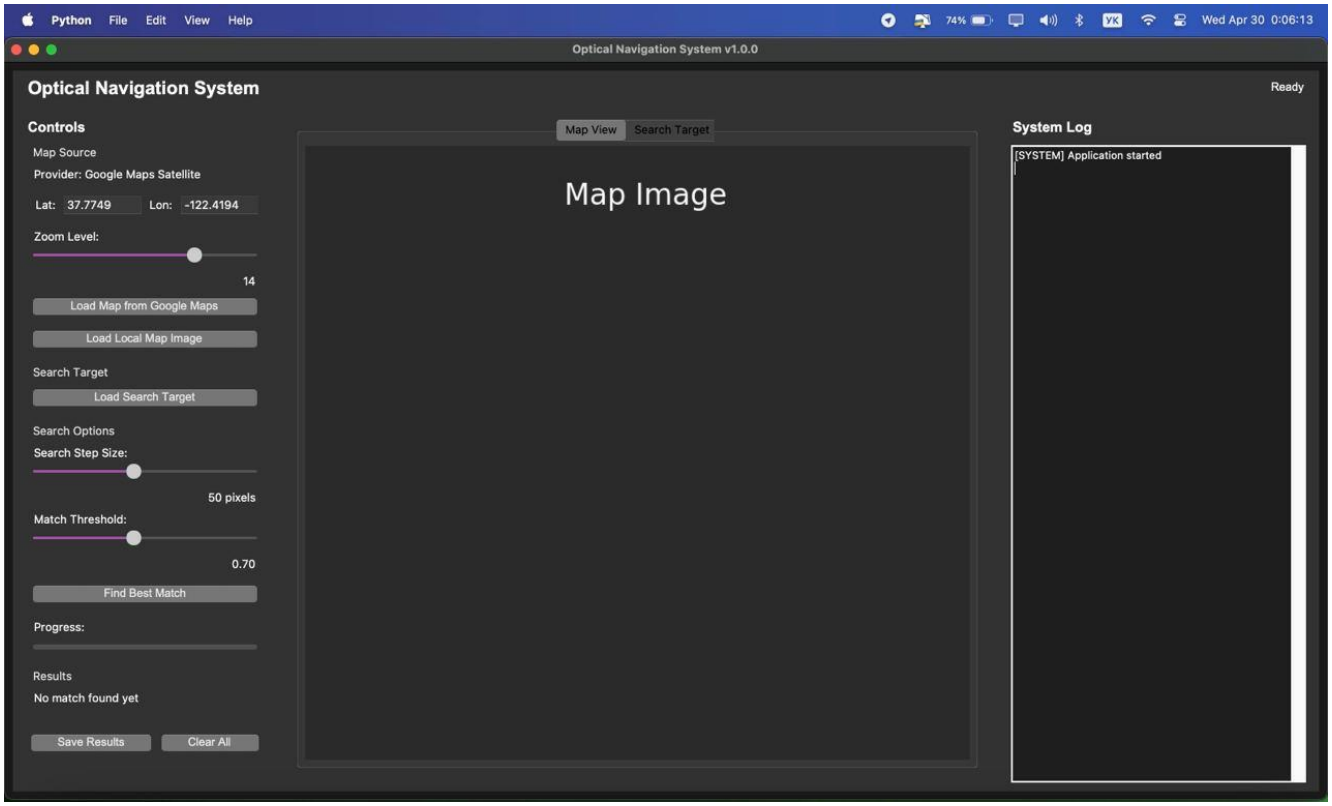


Рисунок 4.7 – Робоче середовище застосунку в операційній системі MacOS.

Крім того, проведено аналіз швидкодії застосунку у сценаріях з високим навантаженням, зокрема під час виконання асинхронних операцій, таких як фільтрація зображень і обчислення координат за допомогою нейронної моделі. Тестування дозволило переконатись, що винесення графічного інтерфейсу користувача в окремий потік виконання забезпечує збереження швидкого та стабільного відгуку на дії користувача навіть під час інтенсивної обробки зображень.

Завершальний етап тестування був присвячений перевірці чутливості та ефективності обчислювального модуля до змін параметрів точності, зокрема таких як нижній поріг прийнятної відповідності та розмір кроку зсуву при аналізі зображення, відомий як step size. Зменшення значення розміру кроку призводить до збільшення щільності аналізу зображення, що, своєю чергою, позитивно впливає на точність визначення координат.

Тестування підтвердило, що зміна зазначених параметрів має передбачуваний вплив на продуктивність і точність системи, що дозволяє адаптувати її поведінку під потреби конкретного сценарію використання.

4.2 Розробка інструкції користувача

Інструкція користувача програмного забезпечення модуля оптичної навігації передбачає наявність попередньо визначених технічних вимог для забезпечення стабільної та ефективної роботи системи. Для ефективної роботи застосунку користувачу необхідно мати відеокарту, що містить CUDA ядра. У випадку, коли система користувача не передбачає наявності відповідної відеокарти, алгоритм використовує потужності центрального процесору, що знижує ефективність роботи обчислювальних алгоритмів. Деталізовану інформацію щодо мінімальної та рекомендованої конфігурації наведено в таблиці 4.2.

Таблиця 4.2 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i5 або AMD Ryzen 5	Intel Core i3 або AMD Ryzen 3
Оперативна пам'ять	8 ГБ RAM	4 ГБ RAM
Вільний простір на диску	5 ГБ	2 ГБ
Відеокарта	NVIDIA GTX 1650 або інтегрована з підтримкою OpenGL 3.3	Інтегрована графіка з OpenGL 2.1
Операційна система	Windows 10 або новіша, Linux Ubuntu 20.04+	Windows 7 або Linux Ubuntu 18.04

Продовження таблиці 4.2

Вимоги до конфігурування	Рекомендовано	Мінімально
Дисплей	Роздільна здатність 1920x1080 пікселів або вище	Роздільна здатність 1366x768 пікселів

Після встановлення та запуску програмного забезпечення користувач переходить до початкового вікна застосунку, у якому автоматично ініціалізується інтерфейс із типовими параметрами. В межах цього вікна доступні основні елементи керування, серед яких панель меню, панель інструментів, інформаційне поле та віджет візуалізації зображення місцевості. На цьому етапі користувач може одразу розпочати завантаження карти або локального зображення місцевості.

Першим етапом взаємодії з програмою є вибір джерела карти. Користувач може завантажити карту з локального сховища або скористатися завантаженням з сервісу Google Maps. У разі вибору зовнішнього джерела необхідно ввести координати центру регіону, після чого карта буде скомпонована та відображена у вікні перегляду. Далі необхідно завантажити зображення місцевості, на основі якого виконуватиметься пошук координат.

Після завантаження обох зображень користувач переходить до налаштування параметрів пошуку, де він може змінити масштаб, розмір обчислювального блоку, а також нижній поріг довіри. Після натискання кнопки «Знайти збіг» активується обчислювальний модуль, який аналізує зображення та виводить знайдену область збігу разом із відсотковим значенням точності. У ході обчислення користувач спостерігає за прогрес-баром, який відображає поточний стан виконання операції.

Після завершення пошуку результати відображаються на карті, де відповідна область буде виділена кольоровою рамкою. У правій частині вікна користувач отримає числове значення відсотка збігу, а також повідомлення про успішність завершення операції. У разі виникнення помилки буде відображене відповідне інформаційне повідомлення в панелі логів.

На завершення користувач може зберегти результати у файл, імпортувати нові дані для повторного аналізу або ознайомитися з довідковою інформацією через меню «Про програму» або «Інструкція користувача». Такий підхід забезпечує простий, логічно структурований процес взаємодії з інструментом для точного виявлення координат на основі зображень місцевості.

4.3 Висновки

У четвертому розділі було проведено тестування функціоналу модулів інтерфейсу, обчислення координат місцевості та фільтрування зображень однорідних місцевостей. Було перевірено стійкість застосунку до помилкових сценаріїв та на використання різних типів зображень. Було розроблено інструкцію користувача по роботі із програмним застосунком.

ВИСНОВКИ

У рамках дослідження були розроблені модулі програмного забезпечення навігації БПЛА із застосуванням методів фільтрації однорідних зображень місцевості та оптичної навігації. Для розробки використовувалося середовище редактора коду Visual Studio Code. Реалізація бакалаврської кваліфікаційної роботи відповідає методичним вказівкам [27, 28].

Під час виконання дослідження проаналізовано проблематику предметної області. Були розглянуті аналогічні продукти, виявлені їхні недоліки та порівняно з власним розробленим застосунком. На основі цього порівняння були сформульовані основні завдання бакалаврської кваліфікаційної роботи.

Під час аналізу технологій розробки було обґрунтовано вибір мов програмування Python, а також бібліотек Tkinter та нейронної моделі Dino.

У дослідженні було зроблено наступне:

- розроблено алгоритми аналізу зображень місцевості для обчислення координат;

- розроблень модуль для фільтрації однорідних зображень, аналіз яких не є доцільним;

- розроблено модуль графічного інтерфейсу, що надає змогу налаштування параметрів нейронної моделі та тестування програмного модулю;

- інтегровано розроблені модулі у єдине програмне рішення;

- проведено тестування програмного забезпечення згідно поставлених задач.

Було розроблено схеми загального алгоритму роботи програмного застосунку та модуля обчислення координат місцевості. Також було розроблено модель системи з використанням UML діаграм.

Тестування програми довело повну працездатність розробленого програмного продукту та відповідність поставленому технічному завданню. Також було розроблено інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дрони (БПЛА), їх роль у сучасному житті та екології [Електронний ресурс] – Режим доступу до ресурсу: https://ela.kpi.ua/server/api/core/bitstreams/e4abf288-2dc6-477e-9d7e-7067b340eaf8/content (дата звернення: 18.03.2025)

2. Top 10 Commercial Uses For Drones [Електронний ресурс] – Режим доступу до ресурсу: https://www.inspiredflight.com/news/top-10-commercial-uses-for-drones.php (дата звернення: 18.03.2025)

3. Bihas O.S. (2025). Analysis of UAV Navigation Methods. Materials of LIV All-Ukrainian Scientific and Technical Conference of Divisions of Vinnytsia National Technical University (2025).

4. Vision-based navigation and guidance for agricultural autonomous vehicles and robots: A review [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S0168169922008924 (дата звернення: 02.04.2025)

5. Vision-based navigation and guidance for agricultural autonomous vehicles and robots: A review [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S0168169922008924 (дата звернення: 03.04.2025)

6. RTK GPS Heading with Dual u-blox F9P [Електронний ресурс] – Режим доступу до ресурсу: https://docs.px4.io/main/en/gps_compass/ u-blox_f9p_heading.html (дата звернення: 22.03.2025)

7. VN-100 IMU / AHRS Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.vectornav.com/products/detail/vn-100 (дата звернення: 25.03.2025)

8. Intel RealSense T265 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://ardupilot.org/copter/docs/common-vio-tracking-camera.html (дата звернення: 25.03.2025)

9. Ouster OS1-64 Specification [Електронний ресурс] – Режим доступу до ресурсу: https://ouster.com/products/hardware/os1-lidar-sensor (дата звернення: 25.03.2025)

10. Septentrio AsteRx UAS GPS Family [Електронний ресурс] – Режим доступу до ресурсу: https://ardupilot.org/copter/docs/common-gps-septentrio.html (дата звернення: 27.03.2025)

11. How is LiDAR used in Robotic Navigation [Електронний ресурс] – Режим доступу до ресурсу: https://www.leorover.tech/post/how-is-lidar-used-in-robotic-navigation-pros-and-cons (дата звернення: 22.03.2025)

12. The Role of Inertial Navigation Systems on Unmanned Aerial Vehicles [Електронний ресурс] – Режим доступу до ресурсу: https://inertiallabs.com/role-of-inertial-navigation-systems-on-unmanned-aerial-vehicles-uav/ (дата звернення: 27.03.2025)

13. Optical Navigation Systems [Електронний ресурс] – Режим доступу до ресурсу: https://engineering.purdue.edu/~andrisan/Publications/Papers_Under_ Submission/ aiaa03.pdf (дата звернення: 27.03.2025)

14. Learn Python the Hard Way (Zed Shaw's Hard Way Series) 5th Edition, Zed Shaw, Addison-Wesley Professional, 2024 – 348 с.

15. Інструментування CPython за допомогою DTrace і SystemTap [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/uk/3.12/howto/instrumentation.html (дата звернення: 07.04.2025)

16. What Is TensorFlow? | NVIDIA Glossary [Електронний ресурс] – Режим доступу до ресурсу: https://www.nvidia.com/en-us/glossary/tensorflow/ (дата звернення: 08.04.2025)

17. Neural Networks and Deep Learning: A Textbook Second Edition, Charu C. Aggarwal, Springer, 2023 – 553 с.

18. Learning OpenCV 5 Computer Vision with Python, Fourth Edition: Tackle computer vision and machine learning with the newest tools, techniques and algorithms 4th Edition, Joseph Howse, Packt Publishing, 2025 – 202 с.

19. Graphical User Interfaces with Tk [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/library/tk.html (дата звернення: 02.04.2025)

20. Visual Studio Code documentation [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/docs (дата звернення: 02.04.2025)

21. Image texture analysis: methods and comparisons [Електронний ресурс] – Режим доступу до ресурсу: https://reu.dimacs.rutgers.edu/~xgui/texture.pdf (дата звернення: 01.04.2025)

22. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent, Aurélien Géron, O'Reilly Media, 2022 – 861 с.

23. Знайомство з технологією CUDA [Електронний ресурс] – Режим доступу до ресурсу: https://devzone.org.ua/post/znayomstvo-z-tekhnolohiyeiu-cuda (дата звернення: 02.04.2025)

24. 10 Usability Heuristics for User Interface Design [Електронний ресурс] – Режим доступу до ресурсу: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 05.04.2025)

25. Software Testing Strategies: A testing guide for the 2020s, Matthew Heusser, Packt Publishing, 2023 – 378 с.

26. Training a Neural Network with Metal Performance Shaders [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/metalperformanceshaders/training_a_neural_network_with_metal_performance_shaders (дата звернення: 02.04.2025)

27. Методичні вказівки до виконання курсових проєктів з дисципліни "Проєктний практикум" для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. В.В. Войтко, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 44 с.

28. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 51 с.

ДОДАТКИ

65

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедрою ПЗ

д.т.н., проф. О. Н. Романюк

"25" березня 2025 р.

Технічне завдання

на бакалаврську кваліфікаційну роботу

«Розробка програмного модулю навігації з використанням нейронних мереж»

за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:

к.т.н., доцент Рейда О.М.

"24" березня 2025 р.

Виконав: студент гр. 2ПІ-21б

Бігас О.С.

"24" березня 2025 р.

Вінниця – 2025

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмного модулю навігації з використанням нейронних мереж».

Галузь застосування – навчальна галузь, навігаційне програмне забезпечення.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ № 97 від «20» березня 2025 р. ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою бакалаврської кваліфікаційної роботи є розробка програмного модулю навігації з використанням нейронних мереж.

Призначення роботи – розробка та програмна реалізація застосунку для визначення координат зображеної місцевості.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Neural Networks and Deep Learning: A Textbook Second Edition, Charu C. Aggarwal, Springer, 2023 – 553 с.
2. Learn Python the Hard Way (Zed Shaw's Hard Way Series) 5th Edition, Zed Shaw, Addison-Wesley Professional, 2024 – 348 с.
3. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent, Aurélien Géron, O'Reilly Media, 2022 – 861 с.
4. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 51 с.

67

5. Технічні вимоги

Вихідні дані до роботи: модель розробки – клієнтський модуль для виконання обчислень координат місцевості зображення; Вхідні дані: середовище розробки Visual Studio Code; каскадна модель розробки; мова програмування Python; графічна бібліотека Tkinter; нейронна модель Dino.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БКР;
2. Технічне завдання;
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання і вибір методу вирішення поставленої задачі дослідження	25.03.25 - 05.04.25	Вик.
2	Розробка алгоритму фільтрації однорідних зображень	06.04.25 - 18.04.25	Вик.
3	Розробка алгоритму обчислення координат зображення місцевості	19.04.25 - 21.04.25	Вик.
4	Аналіз і вибір мови програмування та середовища розробки	22.04.25 - 10.05.25	Вик.
5	Програмна реалізація застосунку	11.05.25 - 18.05.25	Вик.
6	Тестування розробленого застосунку	19.05.25 - 25.05.25	Вик.
7	Оформлення матеріалів до захисту БКР	25.06.25 - 30.05.25	Вик.

10. Порядок контролю та прийняття

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

69

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного модулю навігації з використанням нейронних мереж

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 7.5 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) Рейда О.М, к.т.н., доцент кафедри ПЗ
(прізвище, ініціали, посада)

Здобувач (підпис) Бігас О.С.
(прізвище, ініціали)

ДОДАТОК В

Лістинг програми

```
import tkinter as tk
import sys
import os
import argparse
import logging

# Add path to modules
sys.path.append(os.path.join(os.path.dirname(__file__), 'src'))

# Import application modules
from config.config_manager import ConfigManager
from src.gui.main_window import OpticalNavigationUI
from src.models.model_manager import ModelManager
from src.utils.logger import setup_logger

def parse_arguments():
    """Parse command line arguments"""
    parser = argparse.ArgumentParser(description='Optical Navigation System')
    parser.add_argument('-c', '--config', help='Path to configuration file')
    parser.add_argument('-d', '--debug', action='store_true', help='Enable debug mode')
    parser.add_argument('-m', '--model', help='Specify model name')
    parser.add_argument('--cpu', action='store_true', help='Force CPU usage (disable GPU)')
    return parser.parse_args()

def main():
    """Main entry point for the application"""
    # Parse command line arguments
    args = parse_arguments()

    # Determine config path
    config_path = args.config
    if not config_path:
        config_path = os.path.join(os.path.dirname(__file__), 'config', 'config.yaml')

    # Setup logger
    logger = setup_logger(config_path)
    logger.info("Starting Optical Navigation System")

    # Enable debug logging if requested
    if args.debug:
        logger.setLevel(logging.DEBUG)
        logger.debug("Debug mode enabled")

    # Load configuration
    logger.info(f"Loading configuration from: {config_path}")
    config_manager = ConfigManager(config_path)
```

```
config = config_manager.load_config()

if not config:
    logger.error("Failed to load configuration. Exiting.")
    return 1

# Override configuration with command line arguments
if args.debug:
    config['development']['debug_mode'] = True

if args.model:
    logger.info(f"Overriding model: {args.model}")
    config['model']['model_name'] = args.model

if args.cpu:
    logger.info("Forcing CPU usage")
    config['model']['use_gpu'] = False

# Save updated configuration
config_manager.save_config(config)

# Initialize model manager
logger.info("Initializing model manager")
try:
    model_manager = ModelManager(config.get('model', {}))
    logger.info(f"Model initialized: {config['model']['model_name']} on {model_manager.device}")
except Exception as e:
    logger.error(f"Failed to initialize model: {str(e)}")
    return 1

# Initialize UI
logger.info("Setting up user interface")
try:
    root = tk.Tk()
    app = OpticalNavigationUI(root, model_manager)

    # Configure main window
    root.title(config['app']['name'])
    root.geometry(config['app']['window_size'])

    # Set icon if available
    icon_path = os.path.join(os.path.dirname(__file__), 'assets', 'icon.png')
    if os.path.exists(icon_path):
        icon = tk.PhotoImage(file=icon_path)
        root.iconphoto(True, icon)

    # Display app version in title
    if 'version' in config['app']:
        root.title(f"{config['app']['name']} v{config['app']['version']}")
except Exception as e:
```

```
        logger.error(f"Error setting up UI: {str(e)}")
        return 1

    # Run the application
    logger.info("Application ready")
    try:
        root.mainloop()
    except Exception as e:
        logger.error(f"Application error: {str(e)}")
        return 1
    finally:
        # Cleanup on exit
        logger.info("Cleaning up resources")
        model_manager.cleanup()

    logger.info("Application terminated normally")
    return 0

if __name__ == "__main__":
    sys.exit(main())

import cv2
import numpy as np
import torch
import torchvision.transforms as T
import timm
import threading
import queue
from concurrent.futures import ThreadPoolExecutor
from scipy.spatial.distance import cosine
from skimage.feature import match_template
from skimage.transform import pyramid_gaussian

class ModelManager:
    def __init__(self, config):
        """
        Initialize the model manager with configuration settings

        Args:
            config: Configuration dictionary
        """
        self.config = config
        self.device = torch.device("cuda" if torch.cuda.is_available() and config.get('use_gpu', True) else
"cpu")
        self.model = None
        self.transform = None
        self.executor = None
        self.batch_size = config.get('batch_size', 4)
        self.feature_cache = {}
```

```
        # Multi-scale search parameters
        self.scale_factor = config.get('scale_factor', 0.8)
        self.min_scale = config.get('min_scale', 0.5)
        self.max_scale = config.get('max_scale', 1.5)

        # Feature extraction technique
        self.feature_type = config.get('feature_type', 'deep')  # 'deep', 'template', 'hybrid'

        # Multiple confidence metrics
        self.similarity_metrics = config.get('similarity_metrics', ['cosine', 'l2'])

        # Initialize model and transforms
        self.initialize_model()

    def initialize_model(self):
        """Initialize the deep learning model based on configuration"""
        try:
            # Use a more robust model - DINOv2 or ConvNext
            model_name = self.config.get('model_name', 'convnext_base.fb_in22k_ft_in1k')

            # Load pre-trained model - use a layer that provides better feature representations
            self.model = timm.create_model(model_name, pretrained=True,
features_only=False).to(self.device)
            self.model.eval()

            # Improved image transformations with multi-scale normalization
            self.transform = T.Compose([
                T.ToPILImage(),
                T.Resize((224, 224)),
                T.ToTensor(),
                T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
            ])

            # Initialize thread pool for parallel processing
            max_workers = self.config.get('max_workers', min(8, (torch.cuda.device_count() or 1) * 2))
            self.executor = ThreadPoolExecutor(max_workers=max_workers)

            return True
        except Exception as e:
            print(f"Error initializing model: {str(e)}")
            return False

    def extract_features(self, image):
        """
        Extract deep features using the model

        Args:
            image: Input image (numpy array)

        Returns:
```

```
        numpy array: Feature vector
    """
    # Handle empty or corrupted images
    if image is None or image.size == 0 or not np.any(image):
        return np.zeros(1000)  # Default feature size

    # Ensure correct color channels (RGB)
    if len(image.shape) == 2:
        image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
    elif image.shape[2] == 4:  # Handle RGBA
        image = cv2.cvtColor(image, cv2.COLOR_RGBA2RGB)

    # Convert to tensor
    try:
        image_tensor = self.transform(image).unsqueeze(0).to(self.device)

        # Extract features - getting global pool features instead of raw features
        with torch.no_grad():
            if hasattr(self.model, 'forward_features'):
                features = self.model.forward_features(image_tensor)
                # Global average pooling
                if len(features.shape) == 4:  # If features are [B, C, H, W]
                    features = torch.mean(features, dim=[2, 3])
            else:
                # Get the last fully connected layer's input
                features = self.model(image_tensor)

        # Normalize feature vector to unit norm
        features = torch.nn.functional.normalize(features, p=2, dim=1)

        # Return as numpy array
        return features.flatten().cpu().numpy()
    except Exception as e:
        print(f"Error extracting features: {str(e)}")
        return np.zeros(1000)  # Return zeros as fallback

def extract_features_batch(self, images):
    """
    Extract features from a batch of images

    Args:
        images: List of images

    Returns:
        List of feature vectors
    """
    # Handle empty batches
    if not images:
        return []
```

```
        try:
            # Pre-process images - handle different color channels
            processed_images = []
            for img in images:
                if img is None or img.size == 0:
                    continue

                # Ensure correct color channels
                if len(img.shape) == 2:
                    img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
                elif img.shape[2] == 4:  # Handle RGBA
                    img = cv2.cvtColor(img, cv2.COLOR_RGBA2RGB)

                processed_images.append(img)

            if not processed_images:
                return []

            # Create batch tensor
            batch_tensors = torch.stack([self.transform(img) for img in processed_images]).to(self.device)

            # Extract features
            with torch.no_grad():
                if hasattr(self.model, 'forward_features'):
                    features = self.model.forward_features(batch_tensors)
                    # Global average pooling
                    if len(features.shape) == 4:  # If features are [B, C, H, W]
                        features = torch.mean(features, dim=[2, 3])
                else:
                    features = self.model(batch_tensors)

            # Normalize feature vectors
            features = torch.nn.functional.normalize(features, p=2, dim=1)

            # Return as list of numpy arrays
            return [feat.cpu().numpy() for feat in features]
        except Exception as e:
            print(f"Error in batch feature extraction: {str(e)}")
            return [np.zeros(1000) for _ in range(len(images))]

    def compute_similarity(self, feat1, feat2, method='cosine'):
        """
        Compute similarity between feature vectors using multiple metrics

        Args:
            feat1, feat2: Feature vectors
            method: Similarity method ('cosine', 'l2', etc.)

        Returns:
            float: Similarity score
```

```
        """
        if method == 'cosine':
            # Using 1 - cosine distance for similarity score (higher is better)
            return 1 - cosine(feat1, feat2) if np.any(feat1) and np.any(feat2) else 0
        elif method == 'l2':
            # L2 distance - convert to similarity (higher is better)
            dist = np.linalg.norm(feat1 - feat2)
            return 1 / (1 + dist)
        else:
            # Default to dot product
            return np.dot(feat1, feat2) / (np.linalg.norm(feat1) * np.linalg.norm(feat2) + 1e-8)

    def template_matching(self, map_img, search_img):
        """
        Perform template matching using traditional CV methods

        Args:
            map_img: The larger image to search within
            search_img: The target image to find

        Returns:
            tuple: (best_score, best_location)
        """
        # Convert to grayscale for template matching
        if len(map_img.shape) > 2:
            gray_map = cv2.cvtColor(map_img, cv2.COLOR_BGR2GRAY)
        else:
            gray_map = map_img

        if len(search_img.shape) > 2:
            gray_search = cv2.cvtColor(search_img, cv2.COLOR_BGR2GRAY)
        else:
            gray_search = search_img

        # Apply template matching
        result = cv2.matchTemplate(gray_map, gray_search, cv2.TM_CCOEFF_NORMED)

        # Find the best match
        min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)

        return max_val, max_loc

    def find_best_match(self, map_img, search_img, step_size=30, threshold=0.65,
progress_callback=None, plain_threshold=0.1):
        """
        Find the best match of search_img within map_img using multi-scale search
        and hybrid features approach, with plain image detection

        Args:
            map_img: The larger image to search within
```

```
        search_img: The target image to find
        step_size: Step size for sliding window (pixels)
        threshold: Minimum similarity threshold
        progress_callback: Function to call with progress updates
        plain_threshold: Threshold for plain image detection

    Returns:
        dict: Results including location, score, and annotated image
    """
    # Check for empty images
    if map_img is None or search_img is None:
        return {'found': False, 'error': 'Invalid input images'}

    # First, check if the search image is too plain/uniform
    is_plain, plain_metrics = self.is_plain_image(search_img, threshold=plain_threshold)

    if is_plain:
        return {
            'found': False,
            'error': 'Search image too plain',
            'plain_image': True,
            'plain_metrics': plain_metrics
        }

    # Initialize best match tracking
    best_score = -1
    best_loc = None
    best_scale = 1.0
    current_scale = 1.0

    # Multi-scale search
    scales = [self.scale_factor**i for i in range(-3, 4) if self.min_scale <= self.scale_factor**i <=
self.max_scale]

    # Get dimensions
    h, w = search_img.shape[:2]
    map_h, map_w = map_img.shape[:2]

    # Calculate total windows across all scales
    total_windows = sum(((map_h - int(h*s)) // step_size + 1) * ((map_w - int(w*s)) // step_size + 1)
                        for s in scales if int(h*s) < map_h and int(w*s) < map_w)
    windows_processed = 0

    # Extract features from search image once
    search_features = self.extract_features(search_img)

    # Initial template matching for a baseline comparison
    template_score, template_loc = self.template_matching(map_img, search_img)
    if template_score > threshold:
        best_score = template_score
```

```
    best_loc = template_loc

# Use a multi-threaded approach for deep feature extraction
for scale in scales:
    # Skip invalid scales
    if int(h*scale) >= map_h or int(w*scale) >= map_w or int(h*scale) <= 0 or int(w*scale) <= 0:
        continue

    # Resize search image for this scale
    scaled_search = cv2.resize(search_img, (int(w*scale), int(h*scale)))
    scaled_h, scaled_w = scaled_search.shape[:2]

    # Extract features for scaled search image
    scaled_search_features = self.extract_features(scaled_search)

    # Create batches queue
    batch_queue = queue.Queue()
    result_queue = queue.Queue()

    # Fill queue with window positions
    for y in range(0, map_h - scaled_h, step_size):
        for x in range(0, map_w - scaled_w, step_size):
            batch_queue.put((x, y, scale))

    # Worker function
    def process_windows():
        while True:
            try:
                batch_windows = []
                batch_coords = []

                # Create batch
                for _ in range(self.batch_size):
                    if batch_queue.empty():
                        break

                    x, y, s = batch_queue.get()
                    cropped = map_img[y:y+scaled_h, x:x+scaled_w]
                    batch_windows.append(cropped)
                    batch_coords.append((x, y, s))

                if not batch_windows:
                    break

                # Process batch
                batch_features = self.extract_features_batch(batch_windows)

                # Compute similarities
                for i, features in enumerate(batch_features):
                    x, y, s = batch_coords[i]
```

```
                    # Compute similarity using multiple metrics
                    scores = []
                    for metric in self.similarity_metrics:
                        score = self.compute_similarity(features.flatten(), scaled_search_features,
method=metric)
                        scores.append(score)

                    # Average the scores
                    avg_score = sum(scores) / len(scores)
                    result_queue.put((avg_score, (x, y, s)))

                except Exception as e:
                    print(f"Worker error: {str(e)}")
                    break

        # Start worker threads
        num_workers = min(4, (torch.cuda.device_count() or 1) * 2)
        threads = []
        for _ in range(num_workers):
            thread = threading.Thread(target=process_windows)
            thread.daemon = True
            thread.start()
            threads.append(thread)

        # Process results
        scale_windows = ((map_h - scaled_h) // step_size + 1) * ((map_w - scaled_w) // step_size + 1)
        scale_processed = 0

        while scale_processed < scale_windows:
            try:
                score, (x, y, s) = result_queue.get(timeout=0.1)

                if score > best_score:
                    best_score = score
                    best_loc = (x, y)
                    best_scale = s

                scale_processed += 1
                windows_processed += 1

                # Update progress
                if progress_callback:
                    progress_callback(windows_processed, total_windows)

            except queue.Empty:
                if all(not t.is_alive() for t in threads):
                    break

        # Wait for threads
```

```
            for thread in threads:
                thread.join()

        # Create result
        if best_score >= threshold and best_loc:
            x, y = best_loc
            scaled_h, scaled_w = int(h * best_scale), int(w * best_scale)

            # Create annotated image
            result_img = map_img.copy()
            cv2.rectangle(result_img, (x, y), (x + scaled_w, y + scaled_h), (0, 255, 0), 3)
            cv2.putText(result_img, f"Score: {best_score:.4f}", (x, y - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

            return {
                'found': True,
                'location': best_loc,
                'scale': best_scale,
                'score': best_score,
                'dimensions': (scaled_w, scaled_h),
                'annotated_image': result_img
            }
        else:
            return {
                'found': False,
                'best_score': best_score if best_score > -1 else None
            }

    def hybrid_match(self, map_img, search_img, step_size=30, threshold=0.65,
progress_callback=None):
        """
        Enhanced matching using both template matching and deep features

        Args:
            map_img: The larger image to search within
            search_img: The target image to find
            step_size: Step size for sliding window
            threshold: Minimum similarity threshold
            progress_callback: Function to call with progress updates

        Returns:
            dict: Results including location, score, and annotated image
        """
        # First, use template matching as a fast first pass
        template_score, template_loc = self.template_matching(map_img, search_img)

        # If template matching gives strong confidence, return that result
        if template_score >= 0.9:
            h, w = search_img.shape[:2]
            x, y = template_loc
```

```
    result_img = map_img.copy()
    cv2.rectangle(result_img, (x, y), (x + w, y + h), (0, 255, 0), 3)
    cv2.putText(result_img, f"Score: {template_score:.4f}", (x, y - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

    return {
        'found': True,
        'location': template_loc,
        'score': template_score,
        'method': 'template',
        'annotated_image': result_img
    }

# If template matching gives a reasonable score, use deep features on ROI
if template_score >= 0.7:
    h, w = search_img.shape[:2]
    x, y = template_loc

    # Expand ROI by 25% in all directions for verification
    roi_x = max(0, x - w//4)
    roi_y = max(0, y - h//4)
    roi_w = min(map_img.shape[1] - roi_x, w * 1.5)
    roi_h = min(map_img.shape[0] - roi_y, h * 1.5)

    roi = map_img[roi_y:int(roi_y+roi_h), roi_x:int(roi_x+roi_w)]

    # Use deep features on ROI with finer step size
    roi_result = self.find_best_match(roi, search_img, step_size=step_size//2, threshold=threshold,
                                      progress_callback=progress_callback)

    if roi_result['found']:
        rx, ry = roi_result['location']
        # Adjust coordinates to global image space
        global_x = roi_x + rx
        global_y = roi_y + ry

        result_img = map_img.copy()
        s_h, s_w = search_img.shape[:2]
        cv2.rectangle(result_img, (global_x, global_y),
                      (global_x + s_w, global_y + s_h), (0, 255, 0), 3)
        cv2.putText(result_img, f"Score: {roi_result['score']:.4f}",
                    (global_x, global_y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

        return {
            'found': True,
            'location': (global_x, global_y),
            'score': roi_result['score'],
            'method': 'hybrid',
            'annotated_image': result_img
```

```
            }

        # Fall back to full deep feature search
        return self.find_best_match(map_img, search_img, step_size=step_size,
                                    threshold=threshold, progress_callback=progress_callback)

    def precompute_map_features(self, map_img, search_img, step_size=50):
        """
        Precompute and cache features for the map

        Args:
            map_img: The map image
            search_img: The search target (for dimensions)
            step_size: Step size for sliding window
        """
        h, w = search_img.shape[:2]
        map_h, map_w = map_img.shape[:2]

        # Clear previous cache
        self.feature_cache = {}

        # Multi-scale precomputation
        scales = [self.scale_factor**i for i in range(-2, 3) if self.min_scale <= self.scale_factor**i <=
self.max_scale]

        for scale in scales:
            scaled_h, scaled_w = int(h*scale), int(w*scale)

            # Skip invalid scales
            if scaled_h >= map_h or scaled_w >= map_w or scaled_h <= 0 or scaled_w <= 0:
                continue

            # Process windows in batches
            batch_windows = []
            batch_coords = []

            for y in range(0, map_h - scaled_h, step_size):
                for x in range(0, map_w - scaled_w, step_size):
                    cropped = map_img[y:y+scaled_h, x:x+scaled_w]
                    batch_windows.append(cropped)
                    batch_coords.append((x, y, scale))

                    # Process batch when full
                    if len(batch_windows) >= self.batch_size:
                        batch_features = self.extract_features_batch(batch_windows)

                        # Add to cache
                        for i, coords in enumerate(batch_coords):
                            self.feature_cache[coords] = batch_features[i].flatten()
```

```
                # Clear batch
                batch_windows = []
                batch_coords = []

        # Process remaining windows
        if batch_windows:
            batch_features = self.extract_features_batch(batch_windows)
            for i, coords in enumerate(batch_coords):
                self.feature_cache[coords] = batch_features[i].flatten()

    return len(self.feature_cache)

def get_model_info(self):
    """Return information about the loaded model"""
    return {
        'model_name': self.config.get('model_name', 'vit_base_patch16_224.dino'),
        'device': str(self.device),
        'batch_size': self.batch_size,
        'cache_size': len(self.feature_cache),
        'max_workers': self.config.get('max_workers', min(8, (torch.cuda.device_count() or 1) * 2)),
        'feature_type': self.feature_type,
        'similarity_metrics': self.similarity_metrics,
        'multi_scale_search': {
            'min_scale': self.min_scale,
            'max_scale': self.max_scale,
            'scale_factor': self.scale_factor
        }
    }

def is_plain_image(self, image, threshold=10, visualization=False, block_size=8,
edge_threshold=0.03, texture_sensitivity=0.2):
    """
    Detect if an image is too plain/uniform to be effectively matched using line detection

    Args:
        image: Input image as numpy array
        threshold: Minimum number of lines required for navigation
        visualization: Whether to show visualization of analysis
        block_size: Size of blocks to divide the image for local variance calculation
        edge_threshold: Minimum edge density required for non-plain images
        texture_sensitivity: Sensitivity to textured patterns (higher = more likely to classify as plain)

    Returns:
        bool: True if image is too plain, False otherwise
        dict: Additional metrics about the image complexity
    """
    # Check for empty images
    if image is None or image.size == 0 or not np.any(image):
        return True, {'reason': 'Empty image'}
```

```
    # Convert image to grayscale for analysis
    if len(image.shape) > 2:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        gray = image.copy()

    # Calculate original metrics
    mean_val = np.mean(gray)
    std_val = np.std(gray)
    global_variance = std_val ** 2

    metrics = {
        'global_variance': global_variance,
        'mean_value': mean_val,
        'std_dev': std_val
    }

    # For very low global variance, image is definitely plain
    if global_variance < 0.025:
        metrics['reason'] = 'Extremely low global variance'
        return True, metrics

    # Line detection implementation (from filter_plain_images)
    # Preprocess image to reduce fine texture noise
    denoised = cv2.bilateralFilter(gray, 9, 75, 75)

    # Detect edges at medium scale (good balance for line detection)
    edges_medium = cv2.Canny(cv2.GaussianBlur(denoised, (9, 9), 0), 30, 100)

    # Detect lines using Hough transform
    lines = cv2.HoughLinesP(edges_medium, 1, np.pi/180,
                            threshold=50, minLineLength=30, maxLineGap=10)

    # Count lines
    line_count = len(lines) if lines is not None else 0
    metrics['line_count'] = line_count

    # Visualization
    if visualization:
        try:
            import matplotlib.pyplot as plt
            plt.figure(figsize=(12, 4))

            plt.subplot(1, 3, 1)
            plt.imshow(cv2.cvtColor(image if len(image.shape) > 2 else cv2.cvtColor(gray,
cv2.COLOR_GRAY2BGR),
                                    cv2.COLOR_BGR2RGB))
            plt.title('Original Image')

            plt.subplot(1, 3, 2)
```

```
            plt.imshow(edges_medium, cmap='gray')
            plt.title('Edge Detection')

            plt.subplot(1, 3, 3)
            line_img = image.copy() if len(image.shape) > 2 else cv2.cvtColor(gray,
cv2.COLOR_GRAY2BGR)
            if lines is not None:
                for line in lines:
                    x1, y1, x2, y2 = line[0]
                    cv2.line(line_img, (x1, y1), (x2, y2), (0, 255, 0), 2)
            plt.imshow(cv2.cvtColor(line_img, cv2.COLOR_BGR2RGB))
            plt.title(f'Detected Lines: {line_count}')

            plt.tight_layout()
            plt.show()
        except ImportError:
            print("Matplotlib is required for visualization")

    # Decision primarily based on line count
    if line_count < threshold:
        metrics['reason'] = f'Insufficient lines for navigation ({line_count} < {threshold})'
        return True, metrics

    # If we have enough lines, run additional checks to confirm
    # Calculate edge density
    edge_density = np.sum(edges_medium > 0) / (gray.shape[0] * gray.shape[1])
    metrics['edge_density'] = edge_density

    # Even with sufficient lines, if edge density is still very low
    if edge_density < edge_threshold and line_count < threshold * 1.5:
        metrics['reason'] = 'Low edge density despite some lines'
        return True, metrics

    # Check if the lines are all in one direction (like field rows)
    if lines is not None and len(lines) > 0:
        try:
            # Calculate angles of lines
            angles = []
            for line in lines:
                x1, y1, x2, y2 = line[0]
                if x2 - x1 != 0:  # Avoid division by zero
                    angle = np.arctan((y2 - y1) / (x2 - x1)) * 180 / np.pi
                    angles.append(angle)

            # Check if angles are clustered (parallel lines)
            if angles:
                angles = np.array(angles) % 180
                hist, _ = np.histogram(angles, bins=18, range=(0, 180))
                dominant_direction = np.max(hist) / len(angles) if len(angles) > 0 else 0
```

```
                metrics['dominant_direction_ratio'] = dominant_direction

                # If many lines point in the same direction (like field patterns)
                if dominant_direction > 0.6 and line_count < threshold * 2:
                    metrics['reason'] = 'Lines primarily in one direction (field-like pattern)'
                    return True, metrics
            except Exception:
                pass

        # Image has passed all tests - it has sufficient navigational features
        return False, metrics

    def cleanup(self):
        """Clean up resources"""
        if self.executor:
            self.executor.shutdown()
        # Clear cache
        self.feature_cache = {}

import tkinter as tk
from tkinter import filedialog, ttk, scrolledtext
import cv2
import numpy as np
from PIL import Image, ImageTk
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import threading
import os
import requests
import io
from urllib.parse import quote
import time
from .help_dialog import HelpDialog
from .about_dialog import AboutDialog

class OpticalNavigationUI:
    def __init__(self, root, model_manager):
        self.root = root
        self.model_manager = model_manager

        # Map API variables
        self.map_providers = {
            "Google Maps Satellite": "https://mt1.google.com/vt/lyrs=s&x={x}&y={y}&z={z}"
        }
        self.current_provider = "Google Maps Satellite"

        self.map_zoom = tk.IntVar(value=14)  # Default zoom for Google Maps Satellite
        self.map_lat = tk.DoubleVar(value=37.7749)
        self.map_lon = tk.DoubleVar(value=-122.4194)
```

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО МОДУЛЮ НАВІГАЦІЇ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

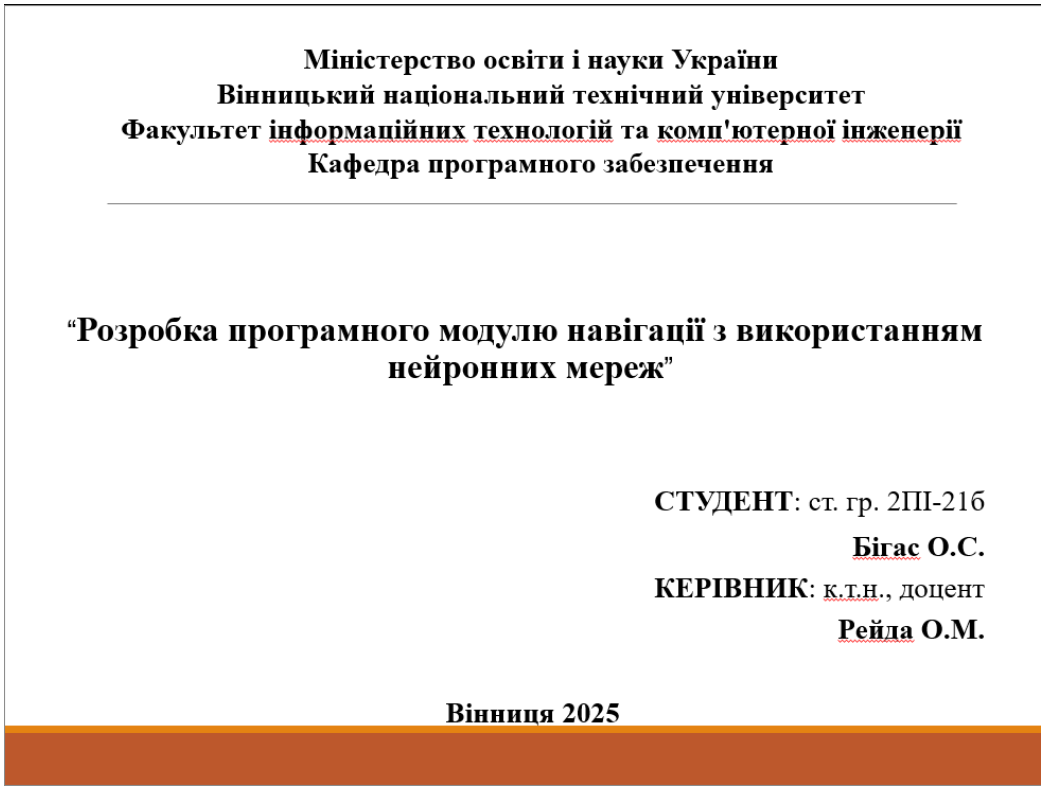


Рисунок Г.1 – Титульний слайд

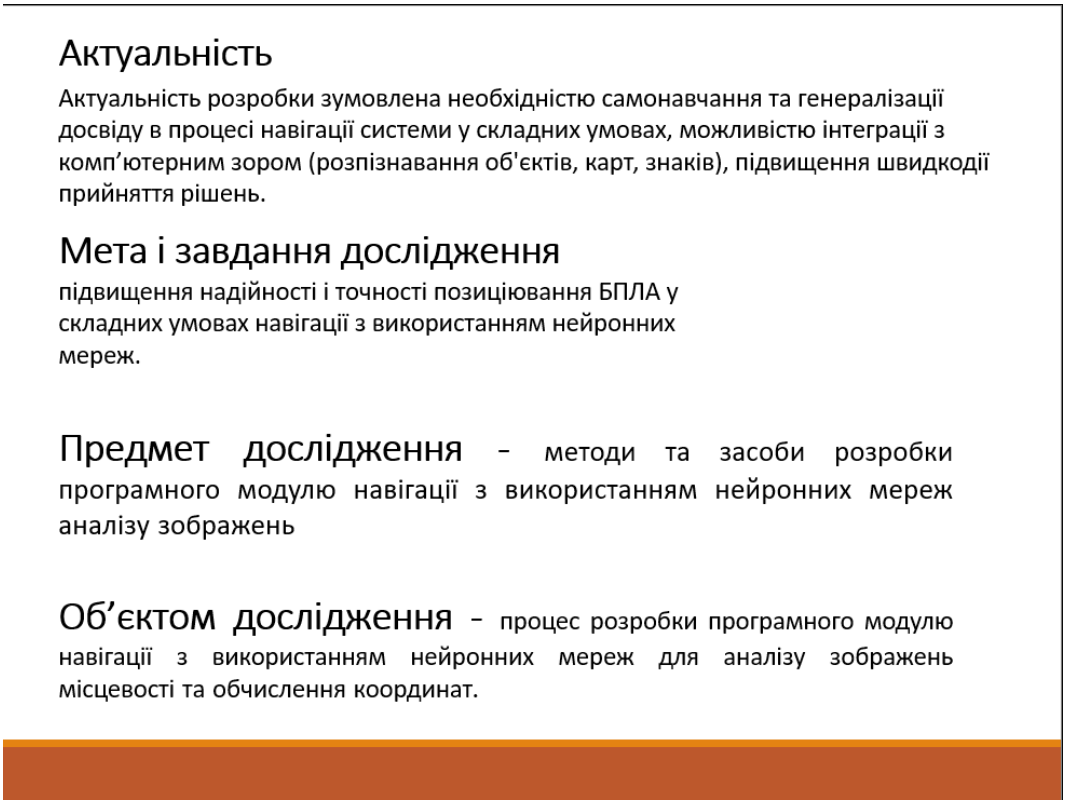
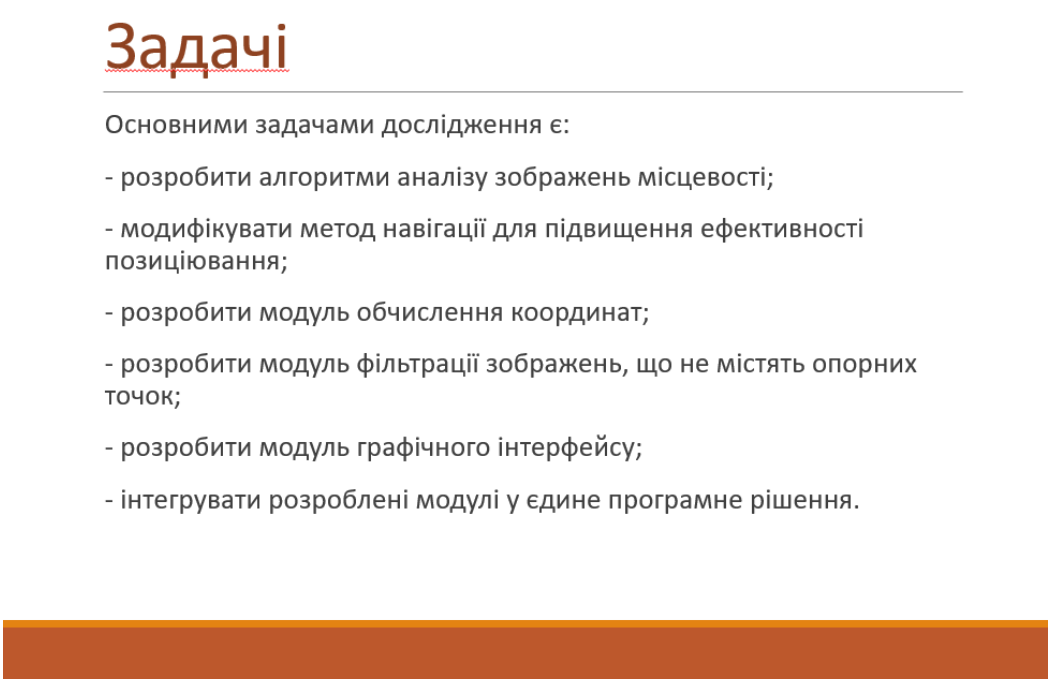


Рисунок Г.2 – Актуальність, мета і завдання дослідження, предмет та об’єкт дослідження



Рисуное Г.3 – Задачі дослідження

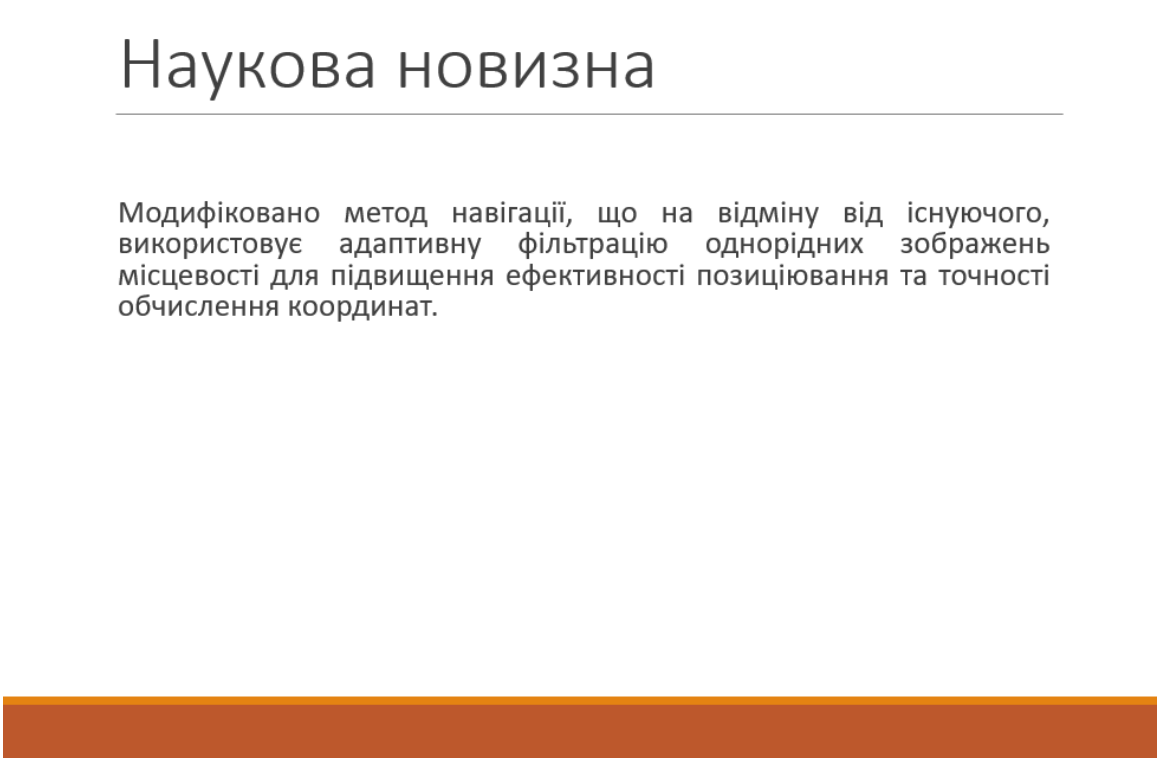


Рисунок Г.4 – Наукова новизна

Практичне значення отриманих результатів

1. Практична цінність одержаних результатів полягає у можливості використання нейронних мереж для виконання обчислень координат вхідних зображень місцевості.
2. Розроблені програмні модулі можуть бути використані в сфері аграрних технологій, наукових досліджень, у цілях виконання рятувальних операцій та у сферах, де є потреба у стабільних та точних рішеннях навігації безпілотних літальних засобах.
3. Програмний модуль має на меті імплементувати можливість інтеграції у різноманітні апаратні системи.

Рисунок Г.5 – Практичне значення отриманих результатів

Аналоги

GNSS приймач u-blox ZED-F9P

- **Метод**: Супутникова навігація (GNSS).
- **Функції:** Визначення точних координат у глобальному масштабі.
- **Особливість:** Підтримка **RTK (Real-Time Kinematic)** для сантиметрової точності.
- **Переваги:** Висока точність, критично важлива для автономних систем і БПЛА в реальному часі.

6

Рисунок Г.6 – GPS Аналог

Аналоги

Інерціальний модуль VectorNav VN-100

- **Склад:** 3-осьові гіроскопи, акселерометри, магнітометри.
- **Функції:** Визначення положення та орієнтації БПЛА без GPS.
- **Переваги:** Працює без зовнішніх сигналів, ефективна у GPS-недоступних умовах.
- **Недолік:** Накопичувальна похибка, що зростає з часом.

Рисунок Г.7 – Аналог інерціальний модуль

Аналоги

Intel RealSense T265

- **Склад:** 3-осьові гіроскопи, акселерометри, магнітометри.
- **Функції:** Визначення положення та орієнтації БПЛА без GPS.
- **Переваги:** Працює без зовнішніх сигналів, ефективна у GPS-недоступних умовах.
- **Недолік:** Накопичувальна похибка, що зростає з часом.

Рисунок Г.8 – Аналог стереокамера

Аналоги

Lidar-сенсор Ouster OS1-64

- **Функції:** Високоточне тривимірне сканування оточення.
- **Технологія:** SLAM (Simultaneous Localization and Mapping) для побудови карт і локалізації.
- **Переваги:** Робота в складних умовах — ліси, міські зони.

9

Рисунок Г.9 – Аналог лідарний датчик

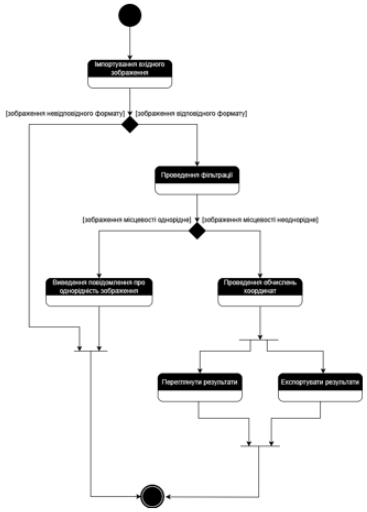
Методи та засоби розробки

- **Мова програмування:** Python
- **Нейронна модель:** Dino для аналізу зображень
- **Інтерфейс користувача:** Tkinter для графічного UI

10

Рисунок Г.10 – Методи та засоби розробки

Діаграма діяльності



1. Користувач запускає застосунок → ініціалізація нейромережі та параметрів.
2. Завантаження карти.
3. Первинна перевірка зображення: аналіз однорідності, недостатньо інформативне → завершення аналізу.
4. Якщо зображення придатне → передача до модуля обробки: нормалізація кольору, масштабування, обчислення векторів ознак (нейромодель).
5. Пошук збігів на карті
6. Вивід результату

11

Рисунок Г.11 – Діаграма діяльності

Алгоритм

1. Ініціалізація нейронної мережі та завантаження конфігурацій.
2. Запуск інтерфейсу та вибір карти місцевості.
3. Імпорт зображення з БПЛА для аналізу.
4. Попередня обробка зображення: фільтрація шумів, нормалізація, зменшення роздільності.
5. Оцінка однорідності:
6. Прийняття рішення щодо доцільності подальшої обробки.

12

Рисунок Г.12 – Алгоритм

Висновки

1. Розроблено програмні модулі навігації БПЛА з використанням фільтрації однорідних зображень та оптичної навігації.
2. Використано Python, бібліотеки Tkinter та модель Dino у середовищі Visual Studio Code.
3. Проаналізовано предметну область та існуючі рішення, визначено недоліки й сформульовано завдання проєкту.
4. Реалізовано: алгоритми аналізу зображень для визначення координат, модуль фільтрації недоцільних зображень, інтерфейс для налаштування та тестування.
5. Проведено тестування, яке підтвердило працездатність ПЗ.

Рисунок Г.13 – Висновки

Публікації й апробації

Результати роботи доповідалися на:

Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025 р., м. Вінниця)

За тематикою дослідження опубліковано 1 наукову працю у збірниках матеріалів конференцій.

Рисунок Г.14 – Публікації й апробації

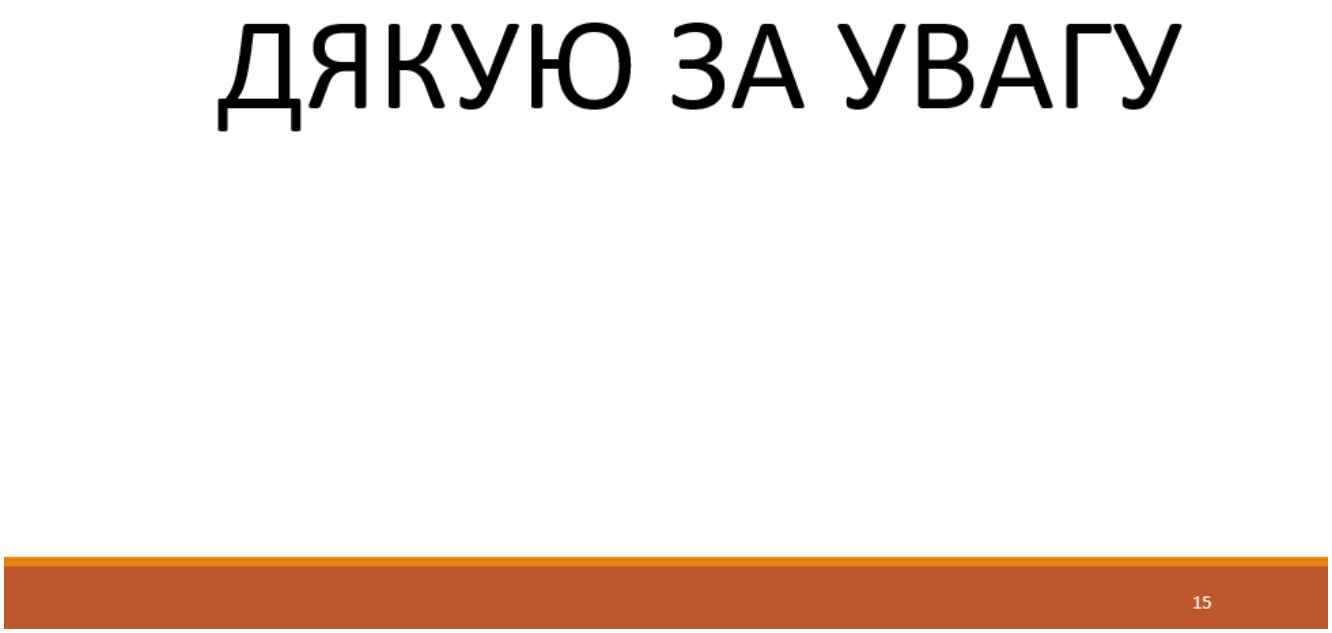


Рисунок Г.15 – Завершальний слайд