МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Оксана ШОВКОПЛЯС
(підпис)
17 грудня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»
освітньо-професійної програми «Інформатика»
на тему: Інтелектуальна технологія позиціонування безпілотного літального апарату на основі машинного зору
здобувача групи ІН.м - 34 Кислощаєва Ігоря Андрійовича

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

Ігор КИСЛОЩАЄВ
(підпис)

Керівник
доцент кафедри
комп'ютерних наук, к.т.н.,
доцент
В'ячеслав МОСКАЛЕНКО
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Оксана ШОВКОПЛЯС
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 «Комп'ютерні науки», освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-34 Кислощаєва Ігоря Андрійовича

1. Тема роботи: Інтелектуальна технологія позиціонування безпілотного літального апарату на основі машинного зору
затверджую наказом по СумДУ від *«13» грудня 2024 року № 1257-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 06 грудня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для машинного зору 3) Розробка інтелектуальної системи позиціонування БПЛА 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	14.09.24	
2	*Огляд технологій, що використовуються для машинного зору*	21.09.24	
3	*Розробка інтелектуальної системи позиціонування БПЛА*	10.11.24	
4	*Аналіз отриманих результатів*	22.11.24	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	27.11.24	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 54 стр., 29 рис., 1 додаток, 17 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню проблем автономної навігації безпілотного літального апарату в умовах радіоелектронного впливу противника. Противник активно застосовує засоби радіоелектронної боротьби, спрямовані на глушіння GPS, що ускладнює виконання завдань безпілотниками. Використання технології машинного зору за відсутності GPS сигналу забезпечує ефективну роботу дронів, підвищує їхню функціональність у бойових умовах, сприяє успішному виконанню завдань і зміцнює обороноздатність країни.

Об'єкт дослідження — процес автономного позиціонування безпілотних літальних апаратів засобами машинного зору.

Мета роботи — розробка технології позиціонування безпілотного літального апарату на основі зображень з камер та попередньо завантаженої мапи місцевості.

Методи дослідження — алгоритм машинного зору та інструменти роботи з зображеннями.

Результати — розроблено інтелектуальну систему яка на основі зображень з камери безпілотного літального апарату та перед завантаженої супутникової мапи місцевості визначає місцезнаходження безпілотного літального апарату. Проведено тестування розробки на даних наближених до реальних.

ВІЗУАЛЬНА ОДОМЕТРІЯ, БЕЗПІЛОТНИЙ ЛІТАЛЬНИЙ АПАРАТ, БПЛА, МАШИННЕ НАВЧАННЯ, PYTHON

ЗМІСТ

ВСТУП

Обґрунтування вибору теми роботи. На сьогоднішній день для безпеки держави важливо мати сильну та сучасну армію, одним з ключових аспектів якої є оперативне забезпечення якісними розвідданими. Розвідувальні місії, особливо на передовій, потребують новітніх технологій, і безпілотні літальні апарати (БПЛА) стали важливою складовою такої розвідки. Проте ефективність БПЛА напряму залежить від їхньої здатності орієнтуватися в просторі, особливо коли традиційні навігаційні системи, як-от GPS, стають недоступними. Здатність дронів до автономного позиціонування без використання зовнішніх навігаційних сигналів може суттєво підвищити їхню ефективність, зберігаючи незалежність від зовнішніх систем.

Одним із сучасних підходів до вирішення цієї задачі є використання машинного зору. Завдяки йому безпілотники здатні «бачити» і розпізнавати елементи навколишнього середовища, орієнтуючись за даними отриманими з камер. Використання машинного зору для позиціонування відкриває можливості для автономного функціонування дронів у складних умовах, де інші методи можуть не спрацювати.

Актуальність теми. У сучасних реаліях війни в Україні, яка ведеться з застосуванням високотехнологічних засобів і включає інтенсивне використання безпілотникв, важливість розробки інтелектуальних технологій для БПЛА значно зросла. Безпілотники виконують низку важливих завдань: вони забезпечують розвідку, моніторинг, коригування вогню, збір оперативних даних та інші функції, критично важливі для успіху бойових операцій. Водночас противник активно застосовує засоби радіоелектронної боротьби (РЕБ), зокрема, для глушіння сигналів GPS, що призводить до втрати навігації дронів і їхньої здатності виконувати поставлені завдання.

Технологія позиціонування на основі машинного зору дозволяє БПЛА орієнтуватися, використовуючи зіставлення візуальних зображень та супутникової мапи, паралельно або навіть замість GPS, що забезпечує

можливість автономної навігації навіть в умовах інтенсивного радіоелектронного впливу з боку противника. Це значно підвищує ефективність роботи дронів у бойових умовах, дозволяє зберегти їхню функціональність та сприяє успішному виконанню завдань. Завдяки чому знижується залежність від супутникової навігації, і БПЛА можуть продовжувати виконувати розвідувальні операції, незважаючи на наявність перешкод з боку РЕБ. Тому дослідження в цьому напрямі є критично важливим для обороноздатності країни, ефективності військових операцій та підвищення безпеки особового складу, що покладається на дані, отримані за допомогою безпілотних літальних апаратів.

Об’єкт дослідження. Об'єктом дослідження кваліфікаційної роботи магістра є процес автономного позиціонування безпілотних літальних апаратів засобами машинного зору

Предмет дослідження. Предметом дослідження є застосування алгоритмів машинного зору для позиціонування БПЛА в умовах відсутності GPS-сигналу.

Новизна. Новизна роботи полягає в тому, що запропонований підхід до позиціонування безпілотних літальних апаратів базується на відкритих технологіях машинного зору і не має комерційних обмежень. На відміну від комерційних рішень, які часто супроводжуються високими цінами та експортними обмеженнями, розроблений метод є доступним для широкого кола користувачів, зокрема для тих, хто працює в умовах обмежених ресурсів. Такий підхід дозволяє значно знизити вартість розробки та забезпечити доступність технології для використання в різних сферах, включаючи військову та цивільну розвідку, без необхідності використання дорогих або обмежених в способах застосування ліцензійних рішень.

Структура. Дана робота складається зі вступу, аналітичного огляду, постановки задачі, вибору методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків та списку використаних джерел.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасний стан дослідження проблеми автономного навігації безпілотних літальних апаратів у середовищах без GPS

На сьогоднішній день автономне навігаційне керування безпілотними літальними апаратами в умовах, де відсутня можливість використання GPS, стає дедалі актуальнішим завданням. Така потреба виникає через численні обмеження GPS-сигналів, особливо в умовах радіоелектронної боротьби та в місцях, де GPS-сигнал може бути перекритий або відсутній. Зокрема, ця проблема стоїть перед військовими та розвідувальними безпілотниками, яким необхідно точно орієнтуватися в просторі та виконувати завдання за відсутності GPS-сигналу, наприклад, під час виконання польотів у зонах з високим рівнем радіоелектронного впливу.

Одним із основних підходів до вирішення цієї проблеми є використання машинного зору, що дозволяє здійснювати навігацію без необхідності використання GPS. Це підвищує автономність безпілотних апаратів і дає змогу ефективно виконувати завдання в складних умовах. Зокрема, використання камер та оптичних сенсорів для збирання зображень навколишнього середовища дозволяє БПЛА аналізувати візуальні особливості місцевості, що дає змогу оцінювати рухи апарата та орієнтувати його щодо стаціонарних об'єктів навколо. Дослідження показують, що для таких систем використовують різні підходи до обробки зображень та оцінки позиції апарата.

Наприклад, в огляді автономної навігації БПЛА в середовищах без GPS [3] було продемонстровано, як методи глибокого навчання, можуть значно підвищити точність локалізації безпілотних літальних апаратів. Дослідження з використанням методів глибокого навчання для обробки візуальних даних дозволяють досягти хороших результатів у виявленні особливостей місцевості, що особливо важливо для навігації в умовах відсутності, або ускладненого функціонування GPS. Також зазначено, що використання таких підходів дає можливість покращити точність локалізації за рахунок кращого

поєднання різних сенсорних даних [12].

Для успішної автономної навігації БПЛА без GPS важливо враховувати різні фактори, зокрема умови освітлення. У дослідженні систем візуальної навігації для БПЛА за різних умов освітлення [4] було запропоновано метод підвищення контрасту зображень для навігації в умовах низького освітлення (рис. 1.1). Це дозволяє значно покращити точність визначення положення БПЛА в нічний час або в умовах поганої видимості. Однак такий підхід вимагає вдосконалення технологій обробки зображень, щоб уникнути розмитості та недостатньої освітленості, які можуть виникати в реальних умовах.

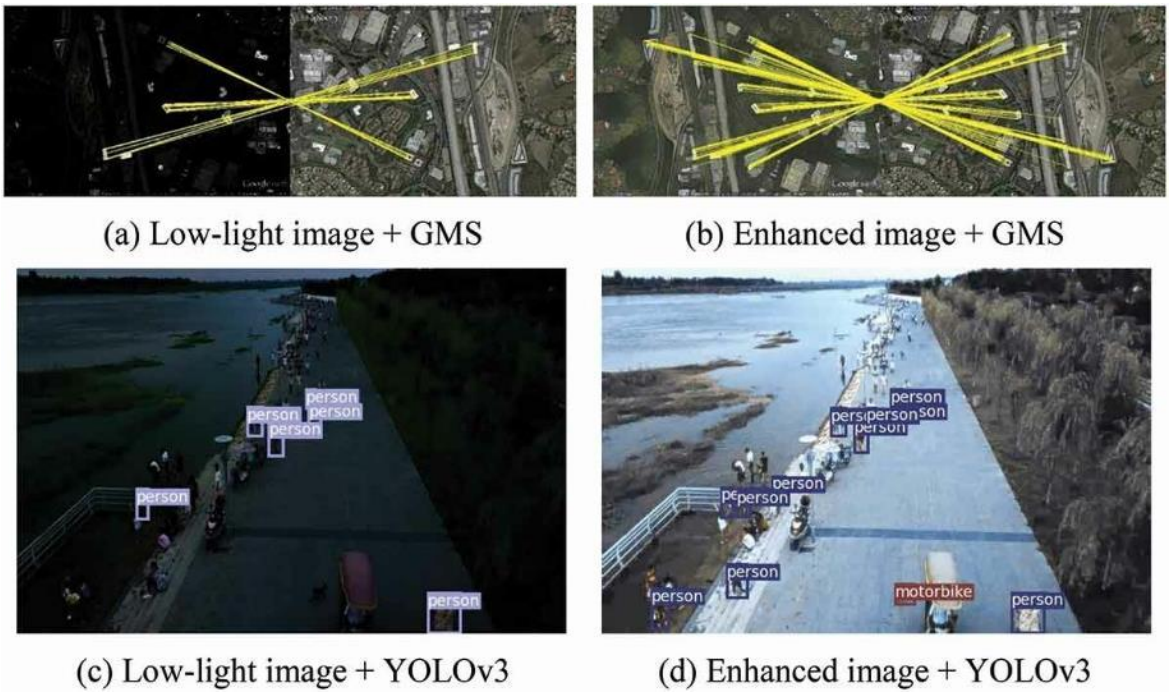

Рисунок 1.1 – Порівняння результатів відповідності між зображенням із слабким освітленням і покращеним зображенням зазначеним методом

Водночас існує певна складність у використанні лише камер для навігації, оскільки таке рішення є дешевим і доступним, але може потребувати додаткової оптимізації для роботи в складних умовах. Тому важливо розробити ефективні методи для усунення можливих помилок у визначенні позиції, зокрема за допомогою покращення алгоритмів для обробки візуальних даних. Для цієї мети активно розвиваються методи злиття даних з

різних сенсорів [16], однак в рамках цієї роботи розглядається лише використання камери, що дозволяє знизити вартість системи і зберегти її доступною для широкого використання в різних умовах.

У підсумку, сучасні дослідження у цій галузі показують, що для ефективної автономної навігації безпілотників без GPS можна досягти хороших результатів з використанням лише камери та алгоритмів обробки зображень. Проте, на практиці, система повинна бути здатна працювати в несприятливих умовах, що виключає втручання людини та отримання інформації з зовнішніх джерел.

1.2 Аналіз аналогічних проєктів

1.2.1 Intel RealSense T265

Загальні відомості: Intel RealSense T265 (рис 1.2) — це інноваційна компактна камера, розроблена компанією Intel для забезпечення автономного позиціонування у просторі [7]. Вона оснащена інтегрованою системою візуальної інерційної одометрії, яка дозволяє точно визначати положення та орієнтацію пристрою без зовнішніх навігаційних сигналів. Завдяки поєднанню даних із двох ширококутних риб'ячооких камер та інерційного вимірювального блоку, камера здатна обчислювати траєкторію руху в режимі реального часу. Особливістю RealSense T265 є те, що всі обчислення виконуються на борту пристрою за допомогою процесора Intel Movidius Myriad 2, що дозволяє знизити навантаження на основну систему та забезпечити мінімальні затримки у передачі даних.

Рисунок 1.2 – Intel RealSense T2

Ця технологія орієнтована на використання в широкому спектрі застосувань, зокрема в робототехніці, автономних транспортних засобах, а також у безпілотних літальних апаратах, які працюють в умовах, де GPS є недоступним або ненадійним. Камера також знаходить застосування в сферах, що вимагають високої мобільності та точності, таких як дрони для картографії, пристрої доповненої реальності, а також у рішеннях для індустрії логістики.

Переваги Intel RealSense T265: Однією з найбільш значущих переваг Intel RealSense T265 є її висока інтеграція апаратного та програмного забезпечення, що дозволяє забезпечити ефективну та швидку обробку даних. Пристрій поєднує в собі компактний розмір, легку вагу і низьке енергоспоживання, що робить його ідеальним вибором для мобільних платформ, які мають обмеження за вагою та доступним об’ємом. Завдяки цьому камера відмінно підходить для використання у БПЛА, особливо в сценаріях, де необхідна висока автономність та тривала робота без підзарядки.

Ще однією ключовою перевагою RealSense T265 є її здатність ефективно

працювати в складних умовах. Вбудовані ширококутні риб'ячоокі камери дозволяють захоплювати більше візуальної інформації навіть у середовищах зі слабким освітленням або невеликою кількістю унікальних візуальних ознак. Завдяки наявності інерційного вимірювального блоку, камера здатна компенсувати тимчасові втрати візуальних орієнтирів, забезпечуючи плавну та надійну навігацію навіть у динамічних сценаріях. Крім того, автономна обробка даних дозволяє використовувати пристрій у системах із низькою обчислювальною потужністю, таких як портативні роботи чи дрони.

Недоліки Intel RealSense T265: Попри численні переваги, Intel RealSense T265 має низку важливих обмежень, які слід враховувати при її використанні. Одним із ключових недоліків є обмеження, пов’язані з використанням риб'ячооких камер. Через меншу роздільну здатність порівняно з традиційними камерами точність розпізнавання дрібних деталей може бути значно нижчою. Це особливо помітно у завданнях, які вимагають високої точності, таких як орієнтація БПЛА на основі висотних аерознімків. Через низьку якість зображення ускладнюється ідентифікація деталей ландшафту, що може призводити до значних похибок позиціонування.

Ще одним важливим обмеженням є потужність пристрою. Реалізований на борту процесор має обмежену продуктивність, що робить обробку великих обсягів даних повільнішою в складних локаціях з великою кількістю подібних деталей. Це може бути особливо критично на відкритих одноманітних просторах із недостатньою кількістю унікальних візуальних ознак, таких як поля, лісопосадки чи озера . У таких випадках камера може втрачати точність, що ускладнює її використання в реальних умовах із високими вимогами до позиціонування.

Камера також є закритою системою, що унеможливлює доступ до її внутрішніх алгоритмів або параметрів для тонкого налаштування. Це обмеження створює додаткові труднощі при інтеграції RealSense T265 у складніші системи, які потребують адаптивності або налаштувань під специфічні завдання. Крім того, система не завжди забезпечує належний

рівень точності, якщо використовується без додаткових сенсорів, таких як лідар.

Окремо варто зазначити вартість Intel RealSense T265, яка стартує від 25 000 гривень. Ціна пристрою може здатися невисокою для професійних рішень, але є значною перешкодою для масового впровадження, особливо у випадках, коли потрібно обладнати велику кількість дронів або інших мобільних платформ. Для багатьох проєктів це може стати критичним фактором, що робить економічно недоцільним масштабне застосування цієї камери.

Крім того, суттєвим недоліком Intel RealSense T265 є низька ремонтоздатність. Камера є монолітним пристроєм, де всі компоненти, включаючи обчислювальний модуль, камери та IMU, інтегровані в єдиному корпусі. Це робить практично неможливим ремонт або заміну окремих частин у разі виходу з ладу одного з компонентів. У польових умовах така ситуація є особливо критичною, оскільки пристрій не підлягає швидкому відновленню або обслуговуванню. У разі пошкодження єдиним рішенням часто стає повна заміна камери, що збільшує експлуатаційні витрати і знижує загальну ефективність систем, які на неї спираються. Для застосувань, де надійність та швидка ремонтопридатність мають вирішальне значення, цей фактор є серйозним обмеженням.

1.2.2 ZED Mini

Загальні відомості: ZED Mini (рис1.3) — це стереоскопічна камера, розроблена компанією Stereolabs, яка поєднує візуальну одометрію та функції просторового сприйняття для навігації і позиціонування у тривимірному просторі[5]. Камера оснащена двома високошвидкісними 2K сенсорами, які забезпечують захоплення стереоскопічного зображення з кутом огляду 110° і можливістю визначати глибину на відстані від 0,1 до 12 метрів. ZED Mini також має інтегрований інерційний вимірювальний блок, який дозволяє використовувати методи візуально-інерційної одометрії для точного відстеження руху. Завдяки підтримці графічного процесора камера може

виконувати обробку стереозображень у реальному часі, забезпечуючи стабільність і точність роботи навіть у динамічному середовищі.

Рисунок 1.3 – ZED Mini

ZED Mini була розроблена для широкого спектра застосувань, зокрема для робототехніки, доповненої реальності, дронів та автономних транспортних засобів. Камера орієнтована на виконання складних завдань, таких як створення тривимірних моделей, навігація у приміщеннях та зовнішніх середовищах, а також моніторинг об'єктів у русі.

Переваги ZED Mini: Однією з головних переваг ZED Mini є її здатність до високоточної обробки стереозображень. Використання двох камер забезпечує значно кращу деталізацію та точність визначення глибини у порівнянні з монокулярними системами. Це дозволяє камері ефективно працювати у середовищах з великою кількістю перешкод, таких як густі міські забудови або складні внутрішні простори. Завдяки підтримці інерційних даних камера здатна компенсувати втрати візуальних орієнтирів, забезпечуючи плавну навігацію навіть у складних умовах.

Крім того, ZED Mini може використовувати потужності графічних

процесорів для обробки даних, що значно підвищує її продуктивність. Це робить її чудовим вибором для платформ, які можуть працювати з обчислювально потужними бортовими системами, такими як Nvidia Jetson. Камера також підтримує високу частоту кадрів і низьку затримку, що критично для застосувань у реальному часі, зокрема для дронів або доповненої реальності.

Недоліки ZED Mini: Попри свої переваги, ZED Mini має низку недоліків, аналогічних тим, що спостерігаються у Intel RealSense T265. Наприклад, через закриту архітектуру пристрою користувачі не мають доступу до внутрішніх алгоритмів і параметрів для налаштування. Це ускладнює інтеграцію камери в складніші системи, які потребують високого рівня гнучкості.

Як і в попередньому випадку, ZED Mini має обмежену ремонтопридатність. Всі компоненти інтегровані в єдиний корпус, що робить заміну окремих блоків у разі їх несправності практично неможливою, особливо у польових умовах. У випадку пошкодження пристрій зазвичай потребує повної заміни, що збільшує витрати на експлуатацію.

Хоча використання GPU підвищує продуктивність, обробка великих обсягів даних може бути проблемою в умовах складних середовищ або довготривалих місій. Для завдань, які вимагають синхронізації з глобальними картами або багаторівневою навігацією, камера може демонструвати затримки або втрачати точність.

Ще одним суттєвим обмеженням є вартість пристрою. ZED Mini продається за ціною близько 20 000 гривень. Така висока ціна унеможливлює її масове застосування, особливо в проєктах, які передбачають масштабування. Для бюджетних систем це може стати визначальним фактором, що обмежує використання цієї технології.

1.2.3 ORB-SLAM

Загальні відомості: ORB-SLAM (рис 1.4) — це відкрита програмна реалізація системи одночасної локалізації та картографування (SLAM), яка підтримує монокулярні, стереоскопічні та RGB-D камери [18]. Ця система

була розроблена для забезпечення високої точності та стабільності в різних середовищах і вважається однією з найбільш популярних і поширених SLAM-систем у науковій спільноті. ORB-SLAM використовує алгоритми обробки зображень для визначення ключових точок (ORB — Oriented FAST and Rotated BRIEF) та їх подальшого відстеження для побудови траєкторій руху та карт середовища.

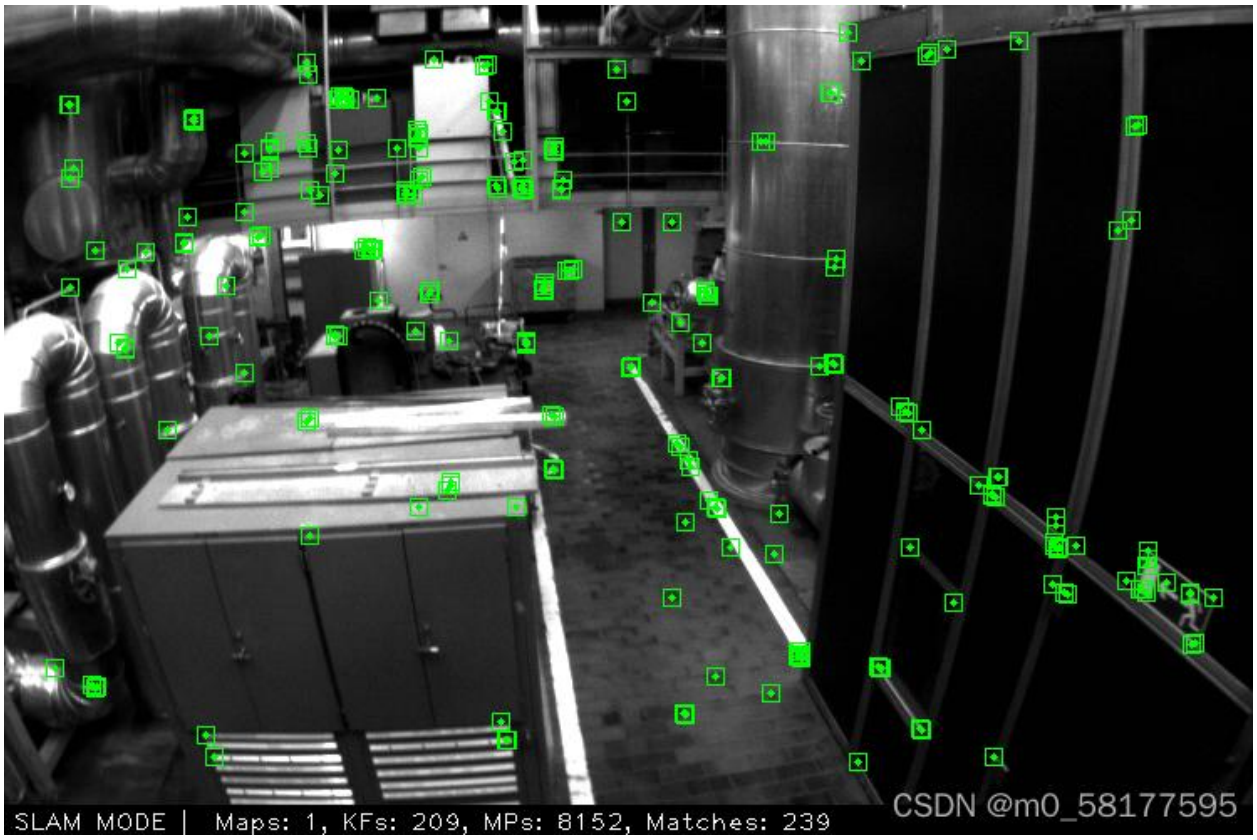

Рисунок 1.3 – ORB-SLAM

Система також підтримує такі функції, як локальна та глобальна оптимізація траєкторії, розпізнавання місць і повторне визначення положення за допомогою методу "закриття циклу". Завдяки своїй відкритій архітектурі та модульному дизайну ORB-SLAM може бути налаштована для роботи в різних сценаріях, від автономних дронів і роботів до систем доповненої реальності.

Переваги ORB-SLAM: Головною перевагою ORB-SLAM є її відкритий вихідний код, що дозволяє користувачам адаптувати систему під свої потреби. Завдяки цьому ORB-SLAM є ідеальним вибором для наукових досліджень і розробок, де потрібна гнучкість і можливість інтеграції з іншими системами. Крім того, підтримка різних типів камер (монокулярних, стерео та RGB-D) дає

змогу використовувати ORB-SLAM у широкому спектрі застосувань.

Ще однією перевагою є висока точність, яку забезпечує ORB-SLAM, особливо у середовищах із великою кількістю візуальних ознак. Завдяки механізму закриття циклу система здатна коригувати помилки, що накопичуються під час довготривалої навігації, і відновлювати точне положення навіть після втрати відстеження. У складних умовах, таких як лабіринтоподібні структури чи приміщення з повторюваними візуальними шаблонами, це стає значною перевагою.

Недоліки ORB-SLAM: Попри численні переваги, ORB-SLAM має низку недоліків, які обмежують її використання в деяких сценаріях. По-перше, система є досить обчислювально важкою. Використання монокулярного режиму на платформах із обмеженими апаратними ресурсами, таких як БПЛА призводить до зменшення точності локалізації.

Так само, як і в Intel RealSense T265 та ZED Mini, масштабованість ORB-SLAM є значною проблемою. На великих площах або в складних середовищах, де присутня велика кількість орієнтирів і об'єктів для обробки, система часто стикається із перевантаженням даних. Це призводить до втрати точності локалізації. Особливо це стосується завдань, які потребують аналізу великих обсягів даних, таких як аерознімки або астрознімки місцевості, де ORB-SLAM не може ефективно опрацювати всю доступну інформацію.

Крім того, система демонструє складність у використанні для пошуку конкретної локації в масштабних географічних даних. Без інтеграції з додатковими джерелами інформації, такими як GPS або спеціалізовані маркери, ORB-SLAM може зіткнутися з труднощами у визначенні точного положення в середовищах із низьким рівнем унікальних візуальних ознак. Це обмежує її можливості у завданнях глобального позиціонування чи співставлення знімків з картографічними даними.

Тож, попри свою відкритість і точність у сприятливих умовах, ORB-SLAM має обмеження, які значно ускладнюють її застосування у сценаріях, що потребують аналізу великих даних або високої масштабованості.

Висновок:

Жодна з розглянутих систем — Intel RealSense T265, ZED Mini чи ORB-SLAM — не є достатньо ефективною для вирішення завдання геолокації безпілотного літального апарату за візуальними даними у широкому спектрі сценаріїв. Усі ці системи мають суттєві обмеження, які ускладнюють їх використання в реальних умовах. Що продемонстровано в таблиці 1.1

Таблиця 1.1 Порівняння систем позиціонування

Характеристика	**Intel RealSense T265**	**ZED Mini**	**ORB-SLAM**
Тип камери	Риб'ячооке (ширококутна)	Стереоскопічна	Монокулярна, стерео, RGB-D
Роздільна здатність	Низька (риби'ячоокі камери)	Висока (2K камери)	Висока (залежно від камери)
Обробка даних	Обмежена продуктивність для великих даних	Підтримує GPU, але може бути недостатньо для великих даних	Обчислювально важка, обмежена для великих даних
Масштабованість	Проблеми при великих та складних середовищах	Проблеми при великих та довготривалих місіях	Проблеми з великою кількістю даних
Ціна	~25 000 грн	~20 000 грн	Безкоштовне ПЗ з відкритим кодом. Ціна залежть від комплектуючих

Intel RealSense T265 має низьку роздільну здатність риб'ячооких камер, що знижує точність у завданнях, які потребують ідентифікації дрібних деталей, таких як висотні аерознімки. Вона також обмежена в обробці великих обсягів даних через обмежену продуктивність і є закритою системою, що ускладнює інтеграцію в більш складні рішення.

ZED Mini має схожі проблеми з масштабованістю та вартістю. Хоча вона використовує стереоскопічну обробку, її продуктивність може бути

недостатньою для обробки великих обсягів даних або довготривалих місій. Закрита архітектура та обмежена ремонтоздатність також обмежують її ефективність.

ORB-SLAM показує обмеження у роботі з великими обсягами даних. Складність адаптації до великих географічних середовищ обмежує її можливості у глобальному позиціонуванні.

Для вирішення задачі геолокації БПЛА необхідне комплексне рішення, котре не матиме вищеназваних обмежень та зможе швидко та якісно обробляти знімки та знаходити їх локацію без зовнішніх джерел інформації.

1.3 Постановка задачі

Перед польотом оператор завантажує в БПЛА файли мапи з координатами. Джерелом файлів можуть виступати як сервіси що надають супутникові зображення , наприклад Maxar, Goole, Bing, так і знімки іншого БПЛА в достатній якості та кількості.

Метою роботи є розробка технології позиціонування безпілотного літального апарату на основі зображень з камер та попередньо завантаженої мапи місцевості.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) Визначення основних точок зображення;
2) Зіставлення точок з зображення та мапи;
3) Визначення координат потрібного місця;

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Вибір мови програмування

Вибір мови програмування є одним із ключових аспектів при розробці будь якого програмного забезпечення. Саме мова визначає, наскільки швидко та ефективно буде реалізовано алгоритм, які ресурси будуть потрібні для виконання програми, а також наскільки легко його можна буде адаптувати або інтегрувати з іншими компонентами системи.

Мови програмування мають різні характеристики, що впливають на вибір залежно від специфіки задачі. Зокрема, для задач локалізації БПЛА важливими факторами є:

- продуктивність виконання коду;
- доступність бібліотек для обробки зображень і машинного навчання;
- простота написання та тестування коду;
- можливість роботи на обмежених апаратних ресурсах;
- масштабованість рішення для реальних умов експлуатації.

2.1.1 Порівняння мов програмування

Для глибшого розуміння їхніх особливостей розглянемо найбільш популярні мови, які застосовуються для задач машинного зору, і порівняємо їх за основними критеріями. Порівняння наведено в табличці 2.1

Таблиця 2.1 Порівняння мов програмування

Характеристика	**C++**	**Java**	**MATLAB**	**Python**
Продуктивність	Висока	Середня	Середня	Середня
Зручність розробки	Низька	Середня	Висока	Висока
Наявність бібліотек	Середня	Середня	Висока	Дуже висока
Вимоги до апаратного забезпечення	Низькі	Середні	Високі	Середні
Придатність до швидкого прототипування	Низька	Низька	Висока	Дуже висока

C++ залишається вибором номер один для задач, де необхідна максимальна продуктивність. Її низькорівнева природа дозволяє програмісту точно контролювати використання ресурсів, що робить C++ ідеальною для систем реального часу та обчислень із жорсткими обмеженнями на швидкість. У той же час, розробка на C++ є складним і тривалим процесом. Синтаксис є важким для засвоєння, а управління пам’яттю вручну підвищує ризик помилок. Для задач швидкого прототипування цей підхід є занадто складним, особливо в порівнянні з іншими мовами. Також для задач машинного зору важлива наявність готових бібліотек котрих у C++ хоча і не мало, але не настільки щоб покрити усі можливі задачі.

Java вирізняється стабільністю, платформонезалежністю та потужною підтримкою багатопотоковості. Це робить її популярною у багатьох сферах, включаючи створення розподілених систем. Проте для обробки зображень і роботи з великими даними Java менш зручна через обмежену кількість бібліотек. У випадку з задачами локалізації, що потребують швидкого прототипування та інтеграції з бібліотеками нейронних мереж, Java не забезпечує такого рівня зручності, як Python.

MATLAB надає зручне середовище для моделювання, роботи з матрицями та прототипування алгоритмів. Завдяки інтуїтивному інтерфейсу і великій кількості готових функцій MATLAB є вибором багатьох науковців для розробки експериментальних алгоритмів. Проте MATLAB не є повноцінною мовою програмування для створення програмного забезпечення: його складно інтегрувати з апаратною частиною, а вартість ліцензії робить його недоступним для багатьох практичних застосувань.

Python вирізняється своєю простотою та універсальністю. Це мова з багатою екосистемою бібліотек для задач машинного навчання (TensorFlow, PyTorch), обробки зображень (OpenCV) та роботи з даними (NumPy, Pandas). Python також пропонує можливість працювати у кросплатформному середовищі, що робить його ідеальним для інтеграції з БПЛА, які часто використовують Linux. Простий синтаксис дозволяє скоротити час на

розробку, а також полегшує розуміння коду іншим програмістом. А потужні інструменти дозволяють швидко реалізувати та протестувати складні алгоритми.

2.1.2 Аналіз результатів порівняння

Після розгляду можливостей кожної мови програмування стає очевидним, що Python є найбільш збалансованим вибором для задач локалізації БПЛА на основі машинного зору. У порівнянні з C++, Python забезпечує простіший синтаксис і легкість розробки, хоча поступається в продуктивності. MATLAB підходить для наукових експериментів, але не для реальних умов експлуатації. Java ж має менше інструментів для машинного навчання, що є критичним у задачах позиціонування безпілотних літальних апаратів.

2.1.3 Висновки

Python поєднує в собі простоту, широкий набір бібліотек та гнучкість, що робить його ідеальним інструментом для розробки систем локалізації на основі машинного зору. З його допомогою можна швидко створювати прототипи, інтегрувати алгоритми в бортові системи дрона та працювати з великими обсягами даних, що робить його оптимальним вибором для поставленої задачі.

2.3 Вибір технології зіставлення зображень

2.3.1 Загальні відомості

Важливою частиною розробки системи є якісна технологія зіставлення зображень. Для вирішення цього завдання необхідно вибрати таку технологію, яка б забезпечувала високу точність і швидкість, а також була б здатна працювати в складних умовах. У цьому контексті було вирішено протестувати три популярні методи зіставлення зображень: ORB, LoFTR та SuperGlue. Кожен з цих методів має свої сильні та слабкі сторони, які будуть детально розглянуті у цьому розділі.

Завдяки високій популярності і розповсюдженості алгоритмів для зіставлення зображень, було вирішено порівняти три найбільш відомі методи, щоб вибрати найоптимальніший для реалізації автономної навігації БПЛА. Ці методи розрізняються по підходах до обробки зображень, вимогам до ресурсів, часу виконання та точності.

ORB — це один із найшвидших та найбільш доступних методів зіставлення зображень. Його основною перевагою є швидкість роботи, що робить його ідеальним для пристроїв з обмеженими обчислювальними ресурсами, таких як безпілотні літальні апарати. Алгоритм ORB поєднує два основних компоненти: FAST для виявлення ключових точок і BRIEF для їх опису. Це дозволяє знижувати вимоги до апаратних ресурсів та забезпечувати швидку обробку зображень у реальному часі.

Однією з головних переваг ORB є його простота у використанні. У порівнянні з іншими методами, такими як LoFTR та SuperGlue. ORB є значно простішим у налаштуванні та інтеграції. Для роботи з ним не потрібно складних налаштувань або високих обчислювальних потужностей, що робить його привабливим для швидких прототипів та базових застосувань. Це особливо важливо в умовах, де час на розробку є обмеженим.

Переваги:

- Висока швидкість обробки.
- Простота у використанні та налаштуванні.
- Підходить для пристроїв з обмеженими ресурсами.

Недоліки:

- Обмежена точність у складних сценах.
- Погана стабільність у випадку сильних змін освітлення чи перспективи.
- Чутливість до шуму на зображеннях.

LoFTR є методом на основі нейронних мереж, який забезпечує високу точність і стійкість в умовах зашумленості зображень і поганого освітлення. Застосування глибоких нейронних мереж дозволяє ефективно справлятися з завданнями зіставлення зображень навіть при наявності значних змін в

перспективі, освітленні та деформаціях. LoFTR є чудовим вибором для вирішення задач, де точність є важливішою за швидкість, оскільки цей метод забезпечує високу точність навіть у складних умовах.

Однак, як і інші методи, LoFTR має деякі недоліки, зокрема значний час обробки і великі вимоги до обчислювальних ресурсів. Це може стати проблемою при використанні на пристроях з обмеженими апаратними можливостями або коли час виконання завдання є критичним.

Переваги:

- Висока точність у складних умовах.
- Стійкість до змін освітлення і шумів.
- Чудова здатність до роботи з великими зображеннями.

Недоліки:

- Повільний час обробки.
- Великі вимоги до обчислювальних потужностей.
- Потребує більше часу на налаштування та адаптацію.

SuperGlue є інноваційним методом зіставлення зображень, що має в основі глибокі нейронні мережі для точного зіставлення точок на зображеннях. Цей метод був розроблений для того, щоб забезпечити точність подібну до LoFTR при прийнятному часі обробки, що робить його чудовим вибором для реальних умов. SuperGlue адаптується до змін освітлення та інших спотворень, що робить його потужним інструментом для автономної навігації.

Переваги:

- Висока точність і стабільність.
- Прийнятний час обробки.
- Добре працює в умовах складних змін освітлення та деформацій.
- Найменша кількість похибок та чітке виділення збігів.

Недоліки:

- Помірна вимогливість до обчислювальних ресурсів.
- Складний в експлуатації.

2.3.2 Практичне тестування

Для оцінки ефективності кожного з методів було проведено тестування на супутниковій мапі місцевості (рис 2.1) , поділеній на дванадцять частини (рис 3.2), а також на зображеннях з різних джерел, які частково перебували на одному з тайлів мапи, а частково виходило за межі мапи (рис 2.3). Що додатково мало ускладнити задачу звставлення а також наблизити тест до реальних умов. Референсні зображення були зібрані з таких джерел, як Google, Bing і HERE(рис 2.4), а сама мапа була отримана з Максар. Метою тестування було перевірити здатність кожного методу точно зіставляти зображення навіть при частковому співпадінні, та виході за межі кордонів.

Рисунок 2.1 – Мапа місцевості

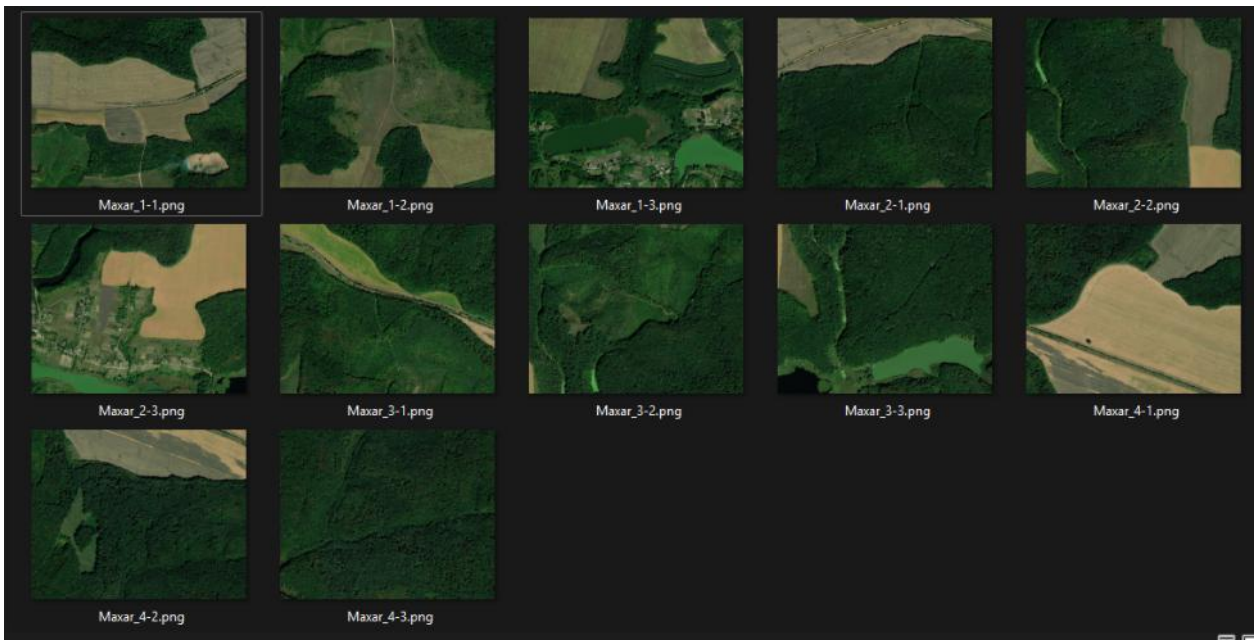

Рисунок 2.2 – Структура файлів мапи місцевості

Рисунок 2.3 – Локація зйомки відносно мапи місцевості

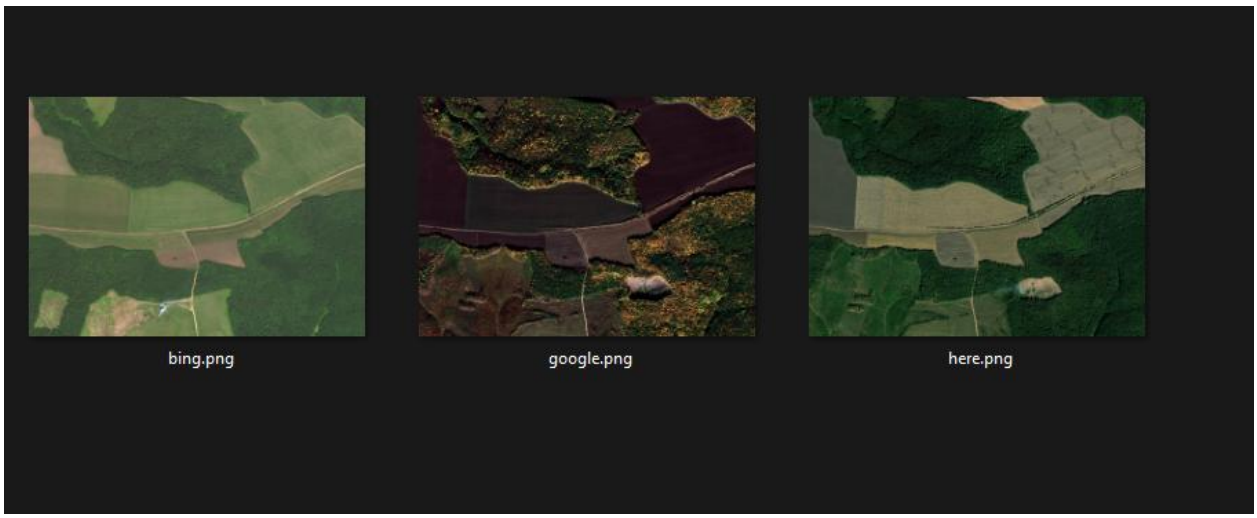

Рисунок 2.4 – Файли тестових зображень

Тестування показало, що ORB є неймовірно простим у використанні. Для його налаштування не потрібно великих обчислювальних потужностей, і з ним можна швидко працювати, проте точність цього методу залишала бажати кращого. Зокрема, ORB успішно зіставив лише 1 з 3 тестових зразків (рис 2.5), демонструючи високу швидкість, але обмежену точність.

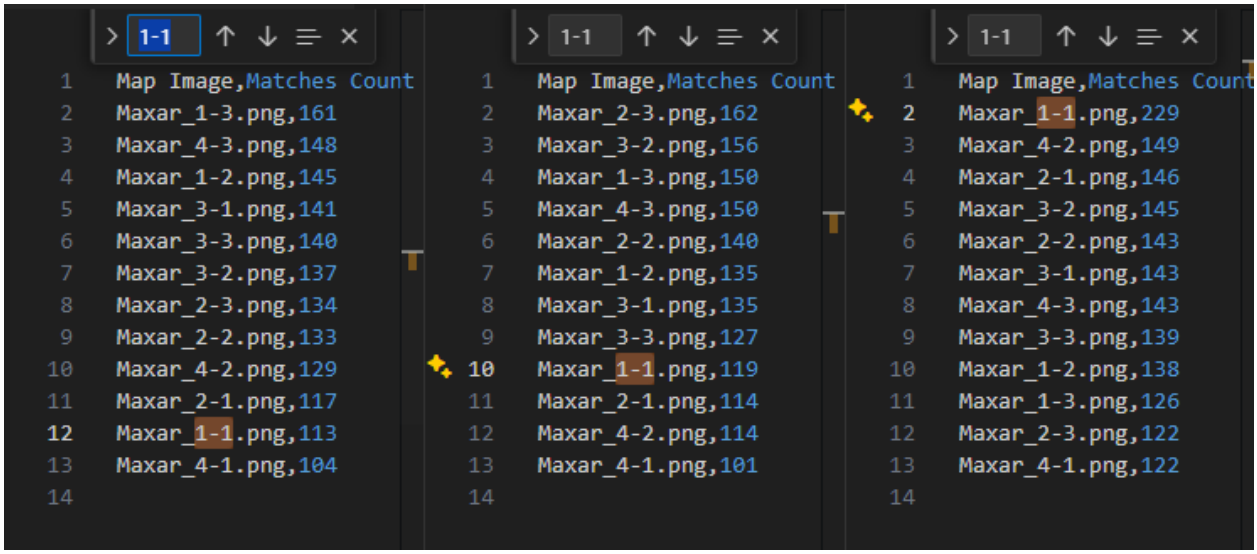

Рисунок 2.5 – Результати зіставлення зображень ORB (правильна відповідь підсвічена)

LoFTR продемонстрував високу точність, успішно зіставивши всі три зразки(рис 2.6), але час обробки був надмірно великий, що робить цей метод менш придатним для реального використання на пристроях з обмеженими ресурсами або в умовах реального часу.

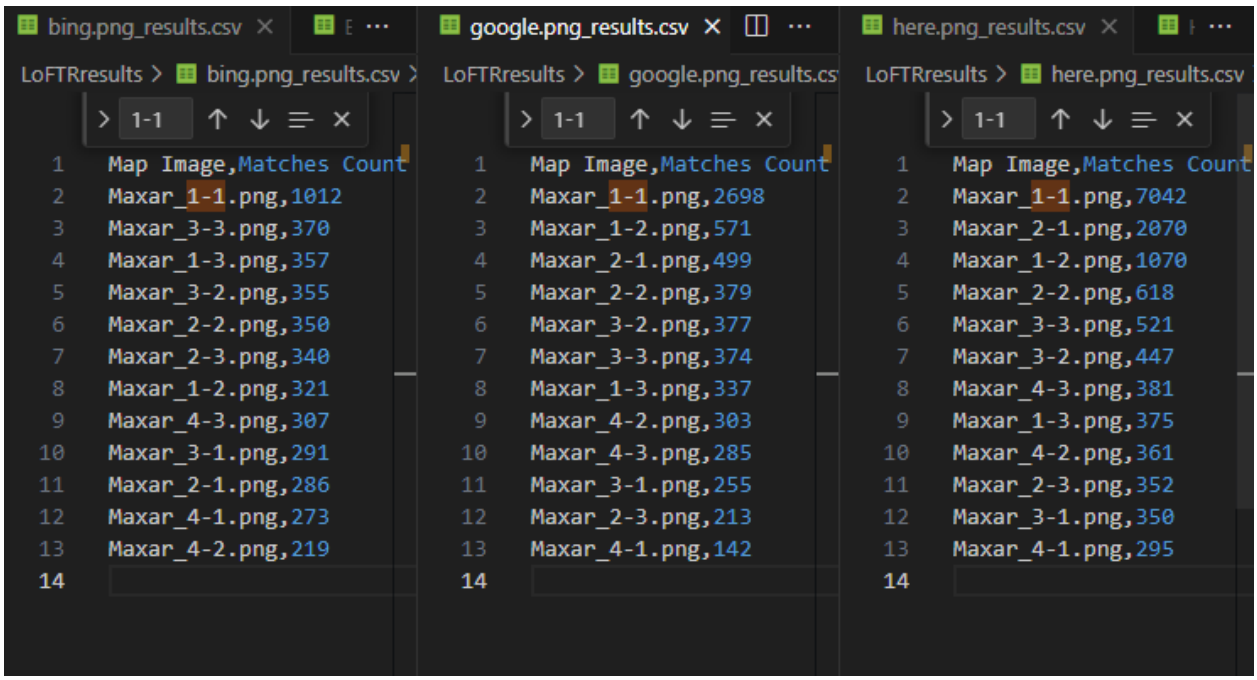

Рисунок 2.6 – Результати зіставлення зображень LoFTR (правильна відповідь підсвічена)

SuperGlue забезпечив високу точність та прийнятний час обробки, що робить його найкращим вибором серед протестованих методів для автономної навігації БПЛА. Цей метод продемонстрував баланс між точністю та швидкістю, що робить його оптимальним для застосування в реальних умовах. Він також дав найменше значення похибок (рис 2.7), чітко виділяючи лише один або два збіги і показав інші значення, близькі до нуля. Важливою особливістю є також те, що SuperGlue дає значення похибок у відсотках що дозволяє легше спрацювати з даними , чітко виділяючи один або кілька конкретних збігів.

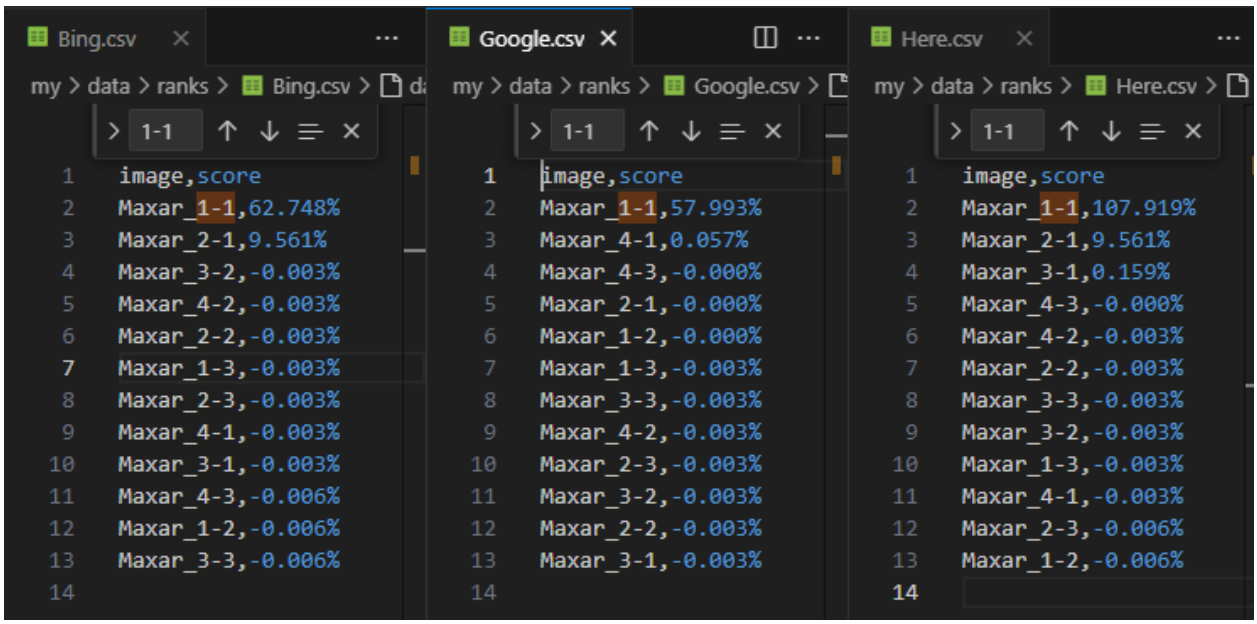

Рисунок 2.7 – Результати зіставлення зображень SuperGlue (правильна відповідь підсвічена)

За результатами тестів було створено таблицю 2.2. В якій порівнюоиться всі три технології за наступними критеріями. точність, час роботи, складність використання та якість порівння визначена на основі того яка кількість хибних тайлів набрали результат достатньо великий щоб вирізнятись серед інших на рівні з основним.

Таблиця 2.2 Порівняння технологій зіставлення зображень

Метод	**Точність**	**Час обробки**	**Складність використання**	**Якість порівняння**
ORB	Низька 1 з 3	Швидкий. < 1 с	Простиа	Низька
LoFTR	Висока 3 з 3	Повільний 3 хв	Помірна	Середня
SuperGlue	Висока 3 з 3	Прийнятний 13 с	Помірна	Висока

2.3.3 Висновок

Підсумовуючи результати тестування, можна зробити висновок, що для вирішення задачі навігації БПЛА кращим варіантом є **SuperGlue** як найбільш точний і достатньо швидкий метод зіставлення зображень

2.3 Підбір бібліотек

У рамках розробки алгоритму локалізації БПЛА на основі машинного зору необхідно вирішити три ключові завдання:

1. Робота з вхідними зображеннями, що передбачає їх обробку та попередню підготовку.
2. Порівняння зображень з дрону та мапи за допомогою методів машинного навчання.
3. Обробка та аналіз вихідних даних, які включають координати та результати порівняння для подальшої інтерпретації.

Для кожного із цих завдань було обрано відповідну бібліотеку: OpenCV для роботи з вхідними зображеннями, PyTorch для реалізації моделей машинного навчання, та Pandas для обробки та аналізу вихідних даних.

2.3.1 Бібліотека для роботи з вхідними зображеннями

Для обробки зображень було обрано бібліотеку OpenCV, яка є лідером у галузі комп’ютерного зору. OpenCV підтримує широкий набір інструментів для зчитування, попередньої обробки та аналізу зображень, що включає фільтрацію, корекцію освітлення, зміни масштабу та виділення контурів. Серед альернатив можна назвати наступні:

- Pillow: Використовується для базової обробки зображень, наприклад, зміни формату чи розміру. Однак вона не має достатнього функціоналу для складних задач комп’ютерного зору.
- scikit-image: Пропонує широкий набір функцій для аналізу зображень, але поступається OpenCV у продуктивності.

Переваги OpenCV

OpenCV забезпечує високу продуктивність та має оптимізовані алгоритми, які працюють швидше за альтернативи. Вона також підтримує роботу з апаратним прискоренням, що є важливим для обробки великих обсягів вхідних даних у реальному часі.

2.3.2. Бібліотека дляорівняння зображень методами машинного навчання

Для реалізації моделей машинного навчання було обрано бібліотеку PyTorch, яка забезпечує гнучкість і зручність розробки алгоритмів глибокого навчання. PyTorch дозволяє будувати нейронні мережі для порівняння зображень, навчати їх і використовувати для порівняння зображень. Серед альтернатив можна назвати наступні:

- TensorFlow: Потужна бібліотека з широкими можливостями, але її обчислювальні графи статичні, що робить її менш гнучкою під час досліджень.
- Keras: Простий інструмент для побудови нейронних мереж, але він не дозволяє створювати кастомізовані рішення так само легко, як PyTorch.

Переваги PyTorch

PyTorch надає динамічні обчислювальні графи. Крім того, PyTorch підтримує GPU-прискорення, що дозволяє працювати з великими обсягами даних ефективно.

2.3.3 Бібліотека для робота з вихідними даними

Результати порівняння зображень потребують аналізу та збереження у зручному форматі для подальшої роботи. Для цього було обрано бібліотеку Pandas, яка є стандартом для роботи з табличними даними у Python. Pandas дозволяє легко зберігати, фільтрувати, групувати та аналізувати вихідні дані, зокрема результати порівнянь окремих зображень.

Альтернативи:

- NumPy: Забезпечує високопродуктивну роботу з масивами, але не підтримує структури даних, такі як DataFrame, які спрощують аналіз.
- Dask: Підходить для обробки великих даних у розподілених системах, але є занадто складним для задач середнього масштабу.

Переваги Pandas

Pandas забезпечує інтуїтивний інтерфейс для роботи з даними, зокрема для виконання операцій фільтрації та об’єднання. Це дозволяє зручно

аналізувати результати алгоритмів, генерувати звіти та інтегрувати дані з іншими частинами системи.

2.3.4 Висновки

Поєднання OpenCV, PyTorch та Pandas забезпечує повний цикл обробки даних: від підготовки зображень, через порівняння їх методами машинного навчання, до обробки вихідних результатів. Ці бібліотеки є оптимальним вибором для розробки алгоритму локалізації БПЛА завдяки своїй продуктивності, простоті використання та широким можливостям.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Формування вхідних даних

Формування вхідних даних є ключовим етапом у створенні системи локалізації БПЛА, що базується на аналізі зображень. Відбір даних для навчання та тестування алгоритму впливає на його точність, надійність та здатність адаптуватися до реальних умов експлуатації. У рамках цієї роботи було сформовано набір тестових даних, що включає картографічні тайли та аерознімки, отримані за допомогою сервісу SAS.Planet.

3.1.1 Джерела даних

Для забезпечення реалістичності тестових умов використовувалися два основних джерела даних:

1. Картографічні тайли, що представляють собою фрагменти карт обраного регіону з географічною прив’язкою. Ці дані були завантажені з сервісу Maxar.
2. Аерофотознімки, що імітують зображення, отримані дронами під час польоту. Вони були створені на основі супутникових карт від Google, Bing та HERE.

Картографічні тайли та аерознімки мають різну природу, оскільки супутникові карти часто оновлюються з різною частотою і зняті за інших умов. Це дозволяє створити набір даних, який враховує потенційні розбіжності між знімками БПЛА та передзавантаженою мапою місцевості.

3.1.2 Локалізація мапи

Окрім зображень картографічних тайлів, було завантажено метадані, що відповідають кожному окремому тайлу, вони містять точні географічні координати кутів кожного тайлу (рис 3.1-3.2). Ці координати є критично важливими для забезпечення коректної роботи алгоритму локалізації, оскільки дозволяють однозначно співвіднести

знайдений тайл із його розташуванням у реальному просторі. Використання координат також забезпечує можливість точного зіставлення отриманих знімків із відповідними тайлами мапи під час тестування.

Рисунок 3.1 – Структура мапи

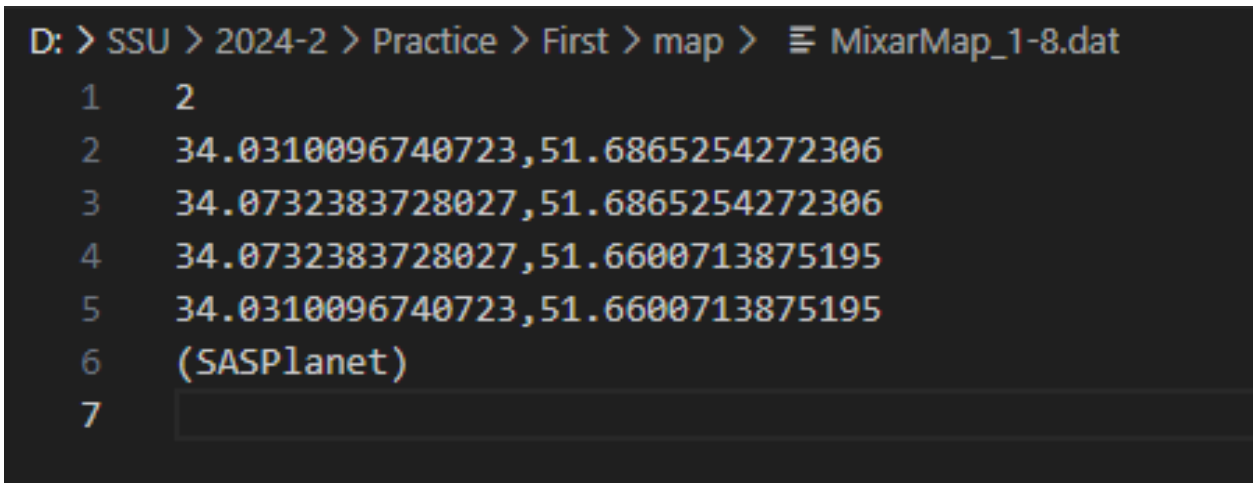

Рисунок 3.1 – Структура файлу

3.1.3 Особливості формування даних

Під час формування вхідних даних було враховано кілька важливих аспектів:

По-перше сучасність картографічних даних. У реальних умовах оператори БПЛА часто мають доступ до актуальних супутникових мап або до результатів попередніх обльотів тієї ж місцевості. Це дозволяє значно підвищити точність локалізації, адже "свіжість" мапи зменшує ймовірність помилок через меншу кількість змін відносно поточної ситуації. Наприклад, вирубки лісів, нові забудови або знищення старих, поява техніки, воронок чи зміни дорожньої інфраструктури можуть ускладнити зіставлення старих даних із новими знімками.

По-друге обмеження доступу до актуальних мап. На сьогодні цивільним особам важко отримати доступ до сучасних супутникових карт, а тим більше до аерофотознімків необхідної місцевості. Це ускладнює створення наборів даних, які повністю відповідали б реальним умовам. Тому для перевірки роботи алгоритму були обрані загальнодоступні дані, отримані за допомогою SAS.Planet, що моделюють можливу ситуацію.

По-третє оскільки супутникові мапи та реальні знімки дрона суттєво відрізняються за такими параметрами якість, кольорокорекція,

освітлення та деталізація, для наближення до бойових умов було вирішено використовувати дані з різних джерел. Для цієї роботи мапа одного сервісу (Maxar) була порівняна з аерознімками, отриманими з трьох інших джерел (Google, Bing, HERE). Ці дані мають значну різницю у часі зйомки (іноді кілька років), якості та інших характеристиках (рис 3.3). Такий підхід дозволяє врахувати можливі труднощі алгоритму під час роботи у складних умовах.

Рисунок 3.3 – Порівняння одного знімку з різних джерел

3.1.4 Тестовий сценарій

Тестовий сценарій передбачає моделювання польоту дрона уздовж заданого маршруту, який охоплює територію кордону між Сумською областю України та Курською областю Росії (рис 3.4) . Дрон летить із півночі на південь, здійснюючи аерознімки території. Кожне зображення передається на алгоритм локалізації, який порівнює його з

тайлами карти, створеної раніше, і повертає координати тайлу, який найкраще відповідає вхідному зображенню.

Рисунок 3.4 – Тестовий маршрут

3.1.5 Реалістичність моделі

Важливо зазначити, що хоча набір даних був сформований із загальнодоступних джерел, він дозволяє адекватно моделювати умови роботи алгоритму в реальних бойових сценаріях. Використання різних джерел для карт та знімків і різниці у їх характеристиках сприяє підготовці алгоритму до роботи у складних, динамічних умовах, де оновлення оточення може суттєво вплинути на точність локалізації.

3.1.6 Висновок

Підбір та формування вхідних даних для розробленої системи здійснювався з урахуванням реальних умов експлуатації, а також із використанням доступних інструментів і сервісів. Врахування різниць у джерелах даних, часі їх оновлення та якості дозволило створити набір, який моделює можливі труднощі під час локалізації. Такий

підхід дозволяє не лише оцінити ефективність алгоритму, але й підготувати його до роботи у складних сценаріях, включаючи умови бойових дій.

3.2 Опис програмної реалізації

3.2.1 Загальний принцип роботи

Програмна реалізація алгоритму включає кілька ключових етапів. На першому, перед польотному, етапі оператор завантажує мапу з координатами та обчислює ключові точки кожного тайлу мапи відповідного файлу, на основі цих точок в подальшому буде проводитись порівняння. Після отримання ключових точок візуальна мапа втрачає свою цінність і може бути видалена.

На другому етапі коли дрон вже піднятий у небо і має необхідність перевірити наявні або визначити фактичні геодані він робить знімок території під собою та надсилає його на обробку алгоритму, котрий розділяє його на декілька окремих з перекриттям для більш точного пошуку збігів у випадку коли зображення припадає на межу кількох тайлів. Оскільки отримані зображення можуть бути зашумленими або мати нерівномірну освітленість, вони проходять попередню обробку за допомогою методів із бібліотеки OpenCV. До цієї обробки входять фільтрація шуму, корекція яскравості та контрастності, що покращує якість вхідного сигналу та забезпечує високу точність подальшого аналізу.

Підготовлені зображення обробляються для виділення ключових точок, які слугують основою для їх зіставлення з базою точок картографічних тайлів. Ці точки аналізуються з урахуванням просторових і текстурних ознак, що дозволяє ідентифікувати найбільш відповідний тайл із бази. Результати зіставлень кожної пари зображень фото з дрону – тайли мапи, Зберігаються в окремому файлі.

Фінальним етапом іде визначення координат цільового тайлу,

котрі і є координатами дрону. Для цього проводиться пошук кращого збігу для всіх зображень та отримуються координати тайлук з котрим збіг був найбільшим. Таким чином подавши на вхід зображення з камери дрон отримує локацію знімку, відповідно досхеми зображеної на (рис 3.6).

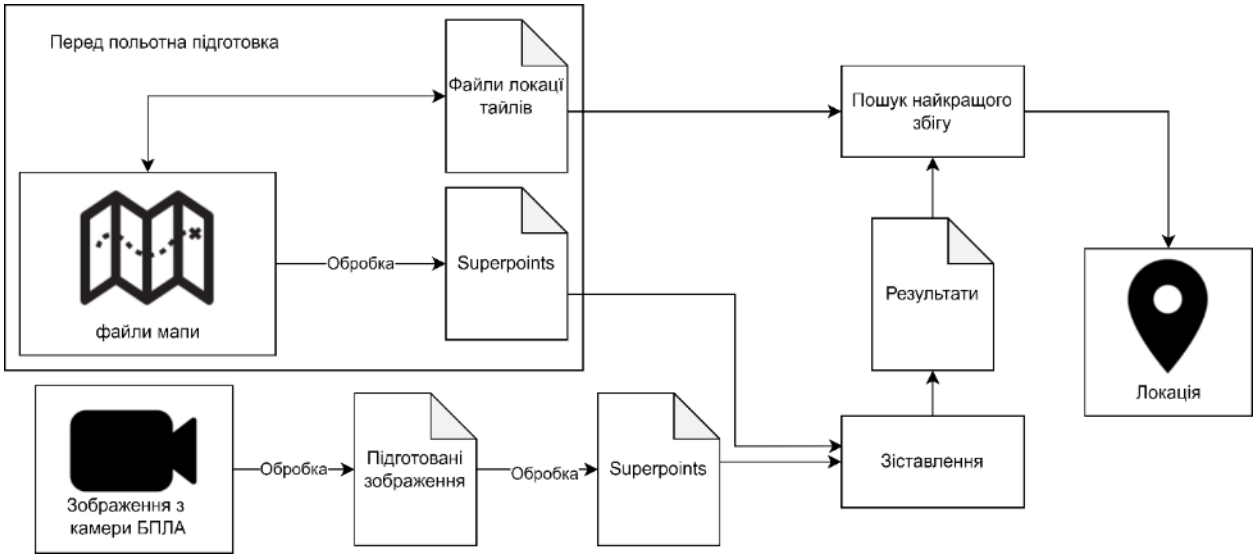

Рисунок 3.6 – Алгоритм пошуку локації

3.2.2 Практична реалізація

Практична реалізація включає в себе декілька окремих файлів в кожному з яких набір функцій та класів зводиться до єдиної котра має вирішити поставлену задачу. А саме:

- crop.py;
- filter_map.py;
- img2superpoint.py;
- rank2coords.py;
- summary4csv.py;
- superpoints2rank.py.

Першою має виконуватись функція filter_map. Вона прйимає в себе директорію розташування файлів мап та видаляє ті тайли котрі мають більше половини чпрозорих пікселів, або чорного кольору. Це пов’язано з принципом

роботи програми SAS.Planet. В випадку коли вона намагається завантажити мапу з не прямим кутами вона робить мапу у форма квадрата в котрий можна вписати цю неправильну фігуру. Та розбиває його на тайли. Таким чином в випадку коли тайл припадає на край виділеної мапи він може виявитись повністю прозорим, крім невеличкої зони біля краю. А в випадку коли тайл поза обраною зоною програма зберігає його але просто чорним (рис 3.7). Тож такі проблеми треба виправляти, для економії пам’яті та ресурсів. Також функція filter_map проводить нормалізацію зображень (рис 3.8)

Рисунок 3.7 – Приклад необробленої мапи

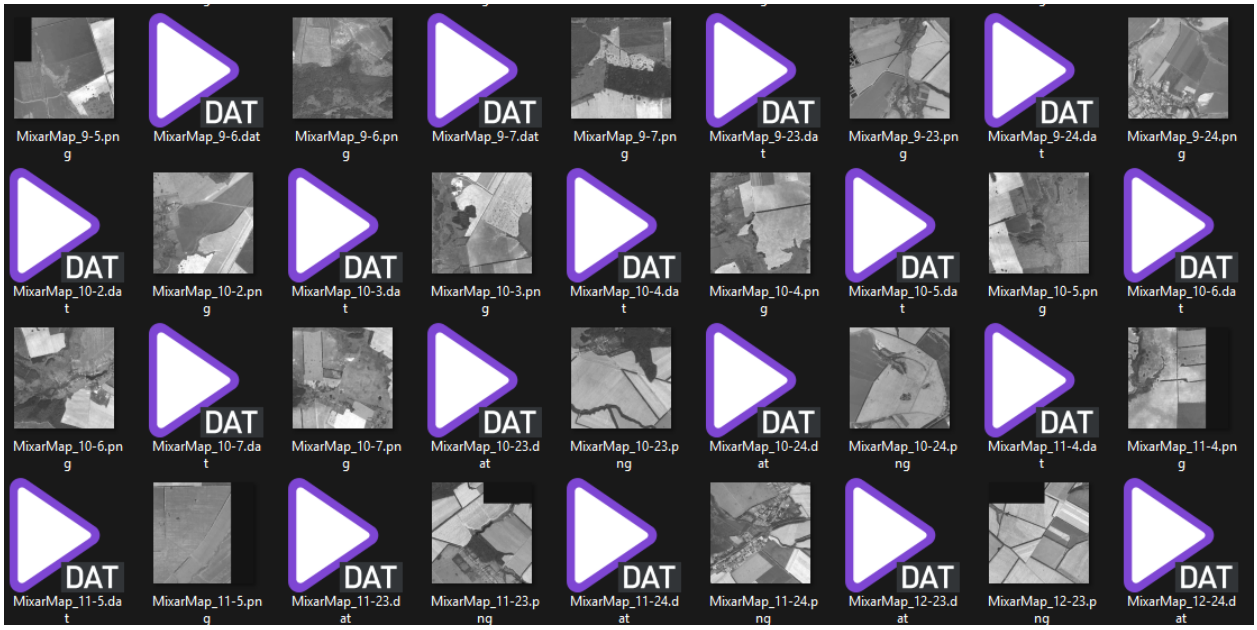

Рисунок 3.8 – Приклад мапи що пройшла обробку

Наступним етапом є визначення основних точок кожного зображення superpoints. Для цього призначена однойменна функція img2superpoint. Однойменного файлу. Ця функція приймає два параметри, директорію з зображеннями та вихідну для зберігання файлів ключових точок. Вона на основі навчених ваг знаходить усі ключові місця зображень. Такі як кути полів, прямі лінії доріг, лісові масиви неправильної форми та зберігає набори цих точко в окремий файл розширення pickle, що є зручним засобом зберігання складних об'єктів Python. Функція проходиться по кожному файлу зображень вхідної директорії і створює його відповідник з ключовими точками в вихідній директорії.

Так завершується передпольотна підготовка. За бажання оператор може видалити зображення мапи для економії пам’яті. Наступний етап відбувається у небі. Коли безпілотник має потребу визначити свої координати або перевірити вірність свого GPS, він робить фото ділянки під собою. Та запускає функцію crop однойменного файлу. Ця функція подібно до попередньої приймає дві змінних. Шлях до файлу знімку ділянки та вихідну директорію. Ця функція розбиває зображення на декілька з пеерекриттям. Таким чином щоб у випадку якщо зображення припадає на границю тайлу, одне з них точно мало

збіг з цим тайлом. Також функція проводить нормалізацію та очистку зображення від шумів (рис 3.9)

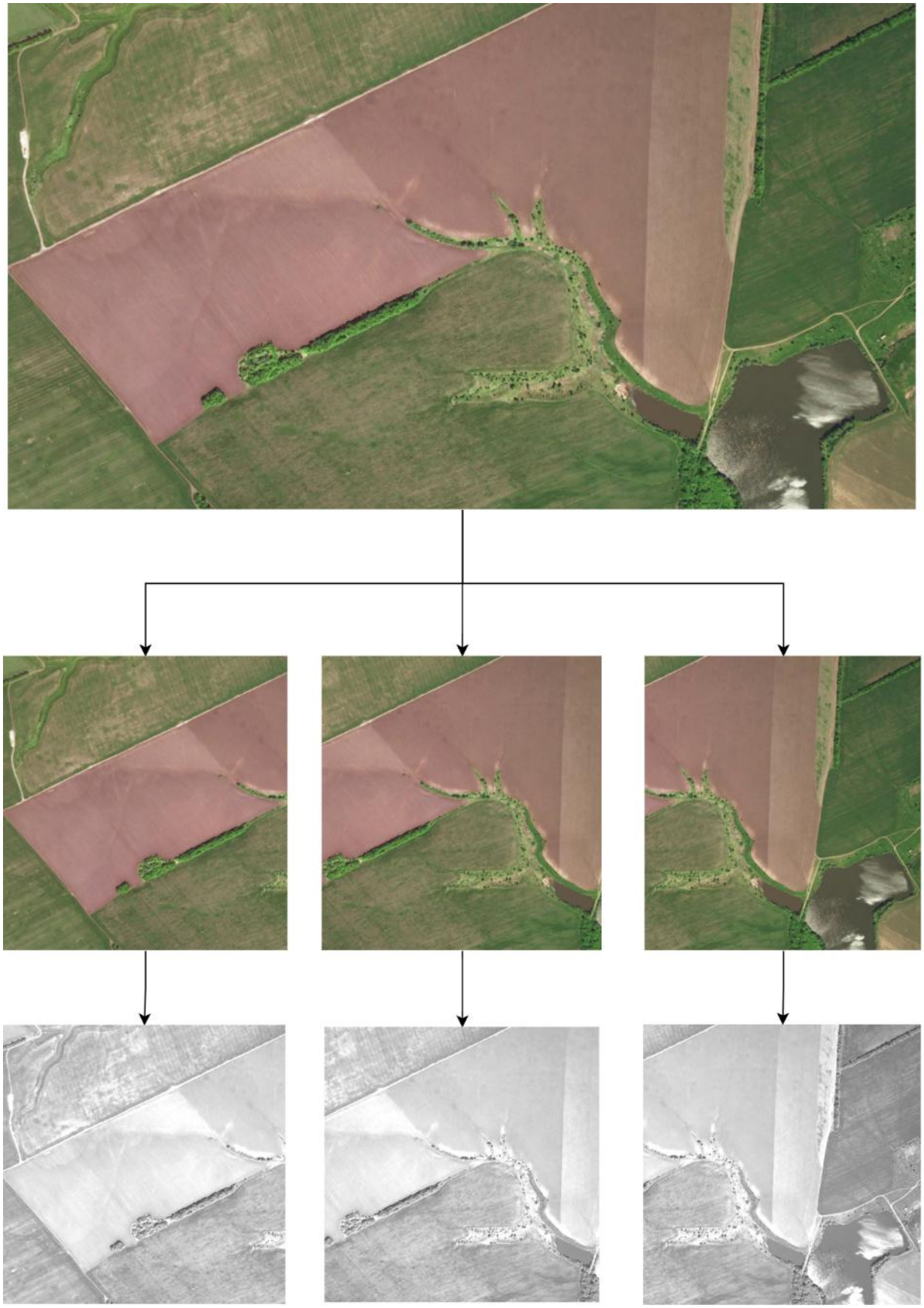

Рисунок 3.9 – Приклад мапи що пройшла обробку

Кожен розбитий кадр отримує свій файл ключових точок за допомогою вже згаданої функції img2superpoint. Наступним етапом іде порівняння файлів ключових точок для кожного зображення та всіх тайлів мапи. За це відповідає функція superpoints2rank відповідного файлу. Ця функція приймає два параметри, а саме вхідну директорію з ключовими точками, та цільову директорію де будуть розміщені результати її роботи. Результатами роботи стають папки, по кількості оброблених зображень. В кожній з яких результати індивідуальних порівнянь кадру з кожним тайлом мапи в форматі .npz, зручному для обробки засобами бібліотеки NumPY, та файл ranking_score.csv. В котрому в зрозумілому форматі знаходяться відсотки співпадінь зображень між собою, відсортовані за зменшенням для зручності сприйняття (рис 3.10).

data > ranks > 2 > ranking_score.csv > data

```
,image,score
0,2,100.000%
1,MixarMap_6-23,14.172%
2,MixarMap_30-30,0.043%
3,MixarMap_29-32,0.039%
4,MixarMap_7-18,0.037%
5,MixarMap_21-24,0.034%
6,MixarMap_7-6,0.006%
7,MixarMap_5-11,-0.000%
8,MixarMap_4-12,-0.000%
9,MixarMap_11-5,-0.000%
10,MixarMap_22-27,-0.000%
11,MixarMap_5-17,-0.000%
12,MixarMap_3-9,-0.000%
13,MixarMap_9-4,-0.000%
14,MixarMap_19-27,-0.000%
15,MixarMap_5-12,-0.000%
16,MixarMap_7-14,-0.000%
17,MixarMap_7-24,-0.000%
```

Рисунок 3.10 – Приклад файлу ranking_score.csv

Тепер маючи декілька файлів ranking_score.csv, ми можемо знайти найбільш вдалий збіг тайлу з зображення. Для цього застосовуємо функцію

summary4csv відповідного файлу. Вона знаходить усі файли ranking_score.csv, та проводить пошук найбільшого відсотку в усіх трьох. Відповідний файл і є нашим тайлом. Ця функція приймає два параметрии. Вхідну директорію та назву файлу для виводу результатів. Після закінчення розрахунків функція видаляє усі папки та залишає на їх місці файл csv аналогічний за структурою до ranking_score.csv з підрахованим спільним співпадінням для всіх трьох кадрів.

Останнім етапом іде отримання координат. Цю задачу виконує функція rank2coords котра повертає значення широти та довготи в форматі DMS котрий використовується переважною більшою сервісів включно з Google maps, та іншими навігаційними програмами. Ця функція приймає шлях до csv отриманого на попередньому етапі та папку де зберігаються координати тайлів. Отримавши ці дані функція визначає файл з найбільшим відсотком збігу та знаходить відповідний йому файл координат. В цьому файлі знаходяться координати усіх отирьох кутів тайлу, ці координати приводяться до середнього переводяться в формат DMS та повертаються назад. Таким чином БПЛА отримує власні координати з точністю до половини тайлу. Чого більш ніж достатньо для навігації розвідувального БПЛА.

3.3 Тестування

Для перевірки працездатності та ефективності розробленого алгоритму навігації безпілотного літального апарата було проведено тестування з використанням реальних картографічних даних. У якості основи для бази картографічних тайлів використовувалися супутникові зображення курсько-сумського кордону з сервісу Maxar, які є високоякісною і деталізованою комерційною розробкою котра має у відкритому доступі застарілі мапи. Що ідеально підходить для тестування, оскільки різниця з більш акутальними мапами сервісу Google та інших сервісів буде разючою, що забезпечить надійну перевірку алгоритму.

Зображення, що імітують кадри з камери дрона, були вибрані з різних джерел, щоб забезпечити різноманітність у тестових умовах. Було використано шість зображень (рис 3.11), два з яких належали до карт Bing Maps, два до Google Maps і два до HERE Maps. Ці зображення спеціально обиралися таким чином, щоб охопити різні типи місцевості та умови освітлення, а також щоб протестувати адаптивність алгоритму до джерел даних із відмінними характеристиками. Розміщення зображень відносно мапи ви можете побачити на рисунку 3.12, на якому червоним кольором позначенно зображення Bing, зеленим – Google, а жовтим – HERE.

Рисунок 3.11 – Зображення що імітують фото з дрона

Рисунок 3.12 – Розташування зображеннь що імітують фото з дрона

У ході тестування кожне із зображень порівнювалося з базою тайлів, створеною на основі мап Maxar, для визначення відповідного тайлу і встановлення координат. Результати тестування показали, що алгоритм успішно впорався із завданням і коректно ідентифікував місцезнаходження на всіх шести тестових локаціях (рис 3.13-3.19). Незалежно від джерела зображення, алгоритм забезпечував точне співставлення ключових точок із базою тайлів, демонструючи високу адаптивність та універсальність.

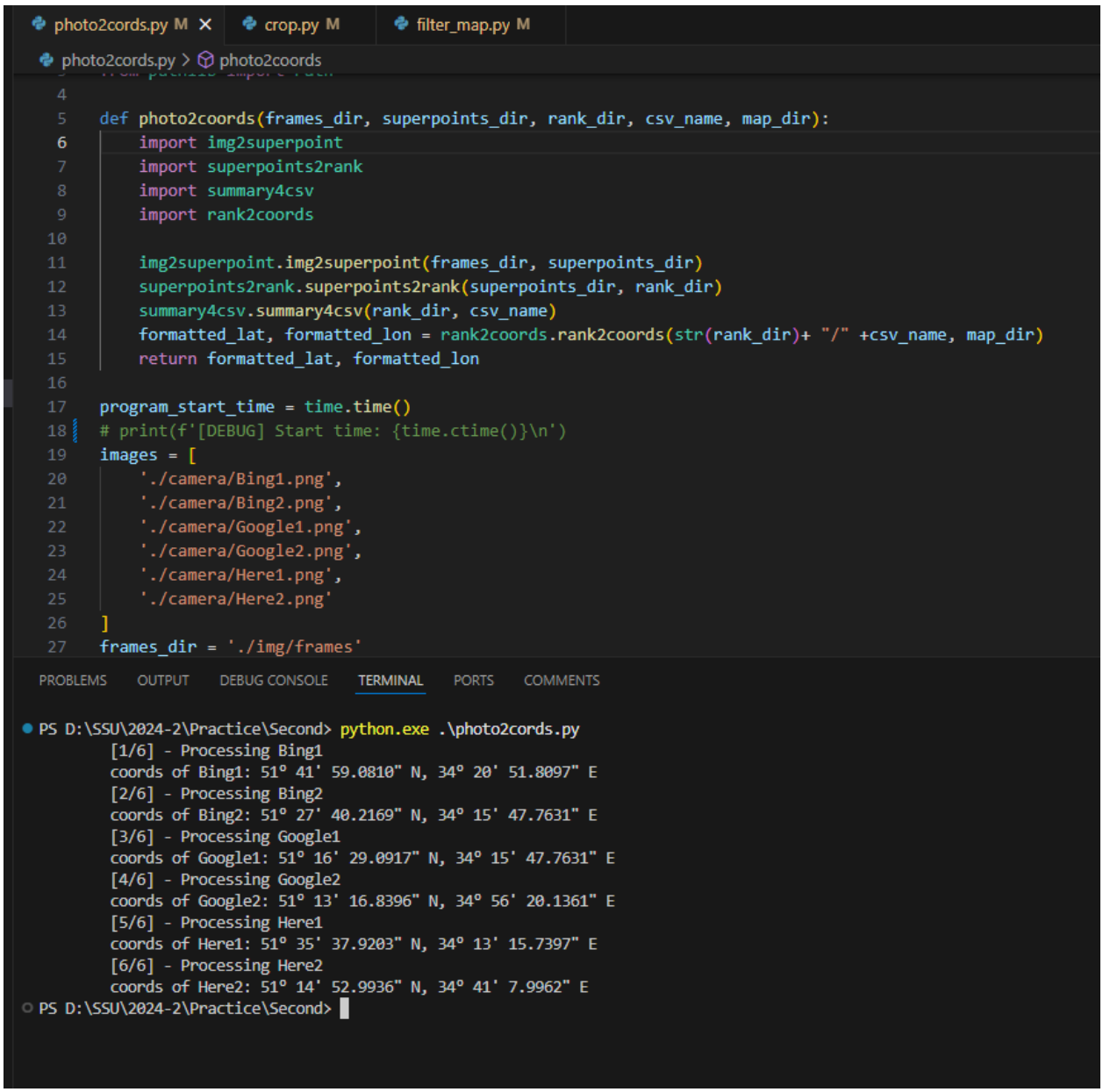

Рисунок 3.13 – Результат роботи коду

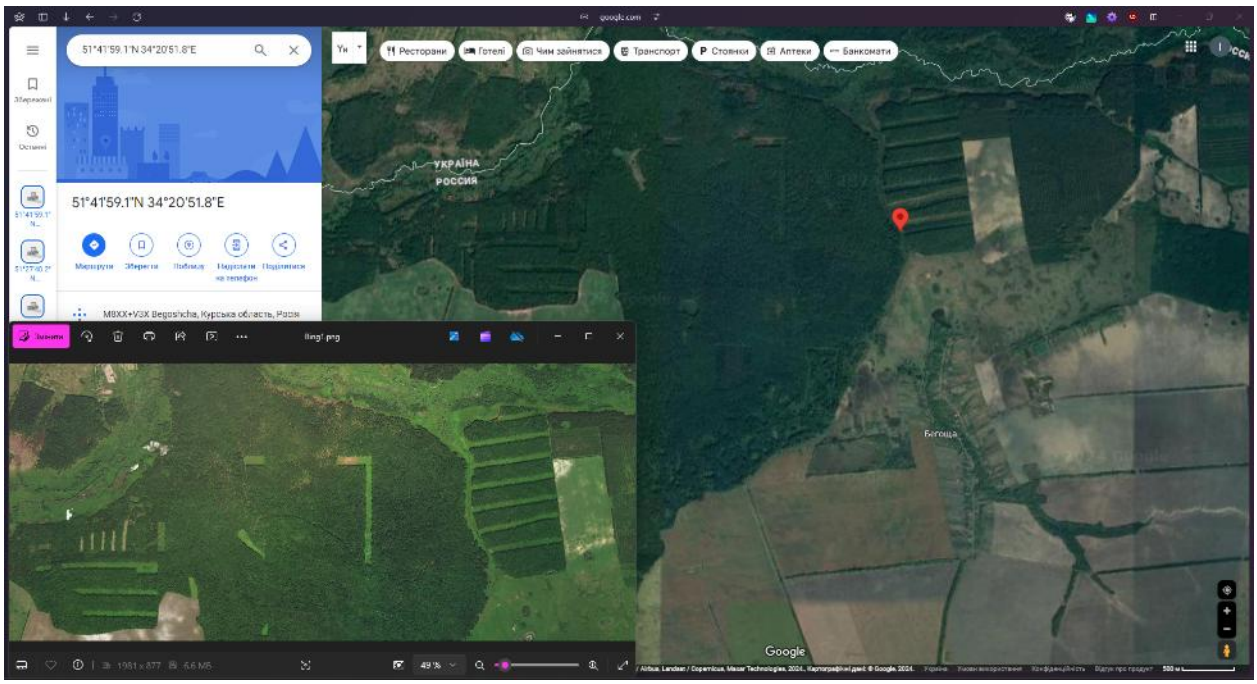

Рисунок 3.14 – Локація зображення bing1

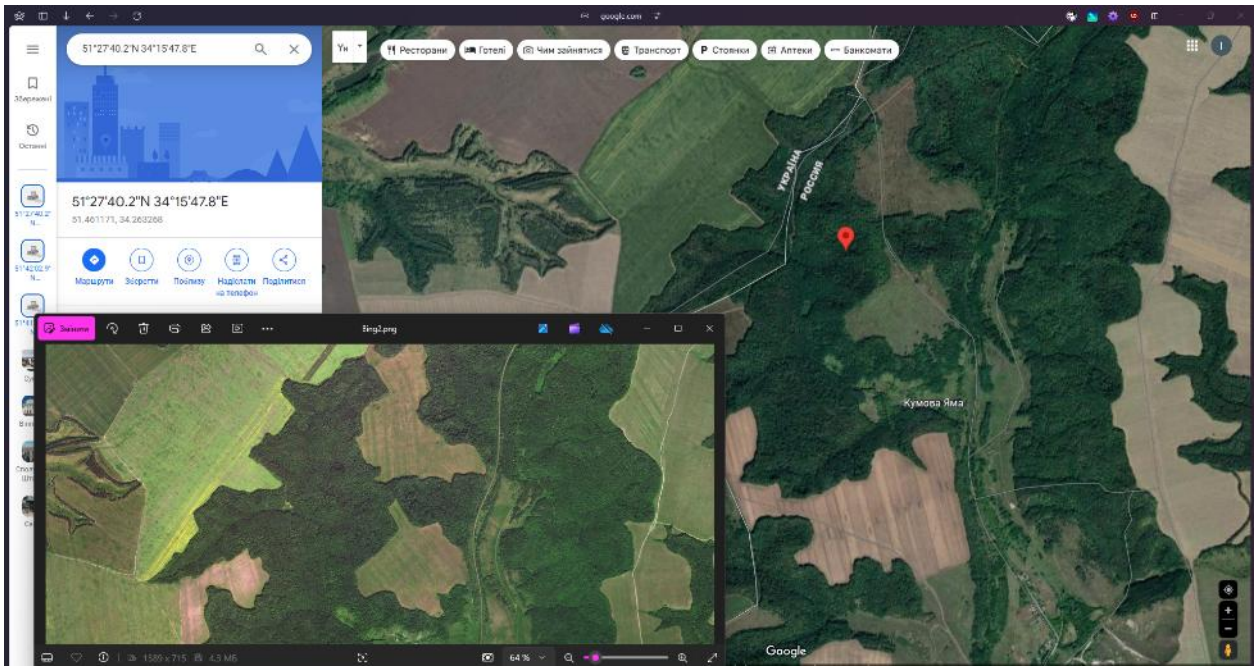

Рисунок 3.15 – Локація зображення bing2

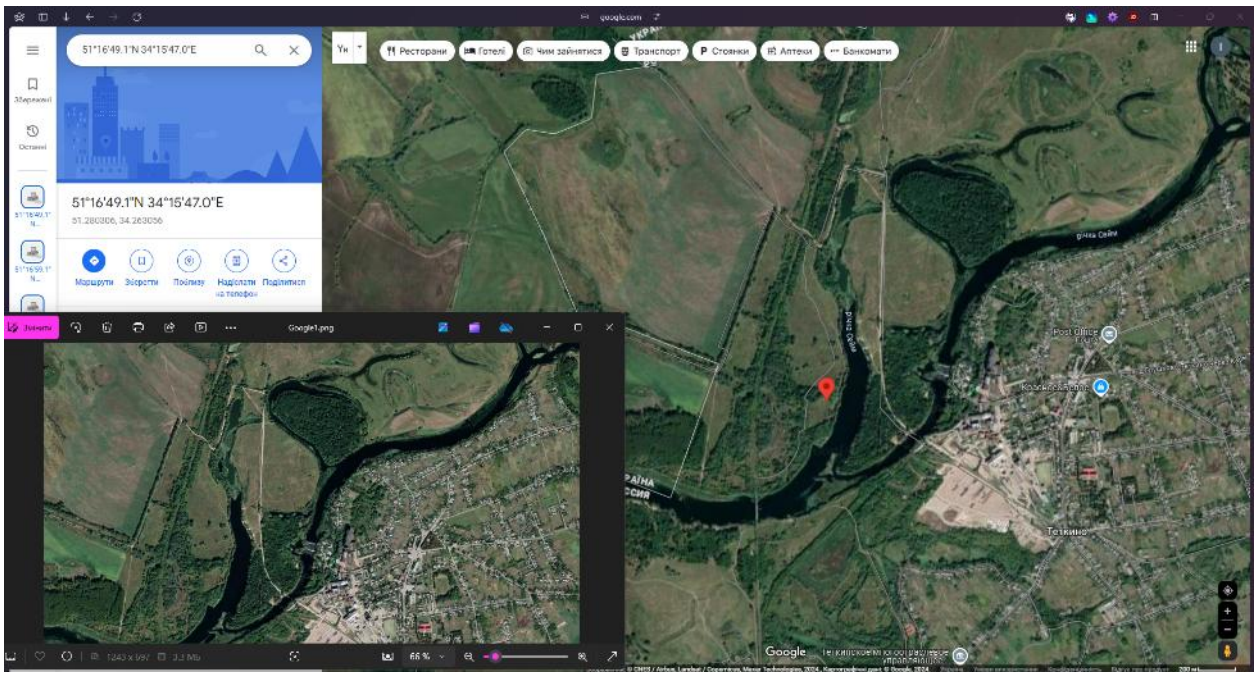

Рисунок 3.16 – Локація зображення google1

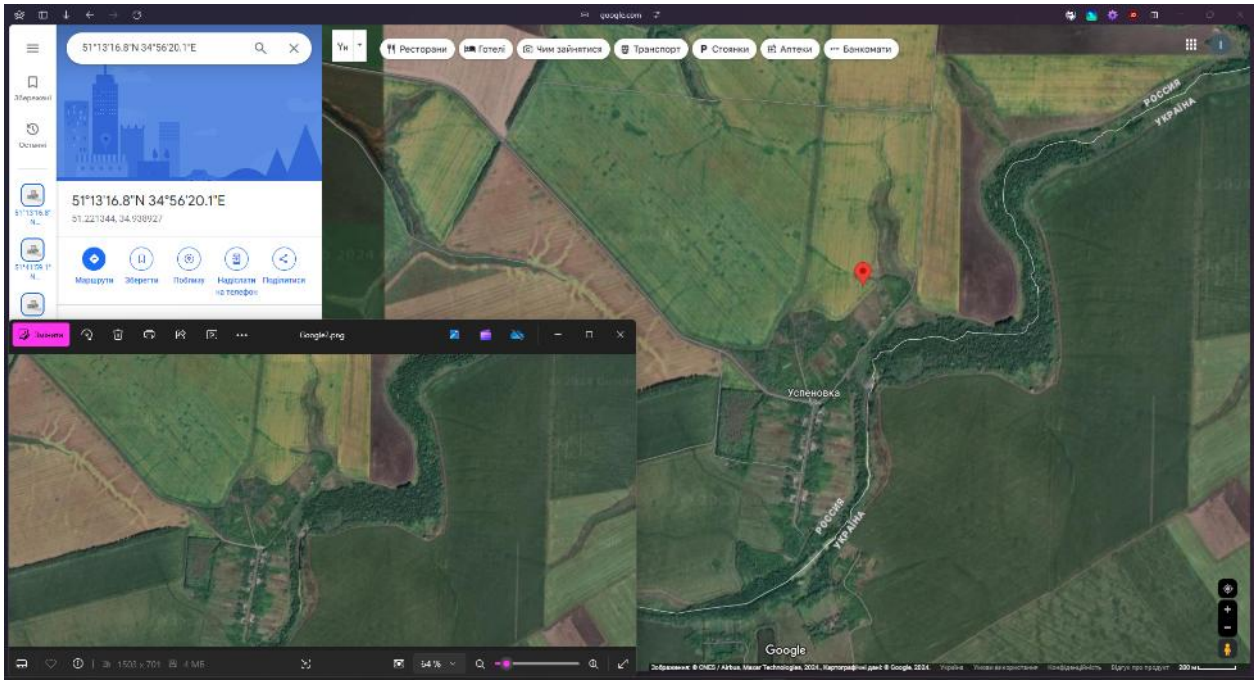

Рисунок 3.17 – Локація зображення google2

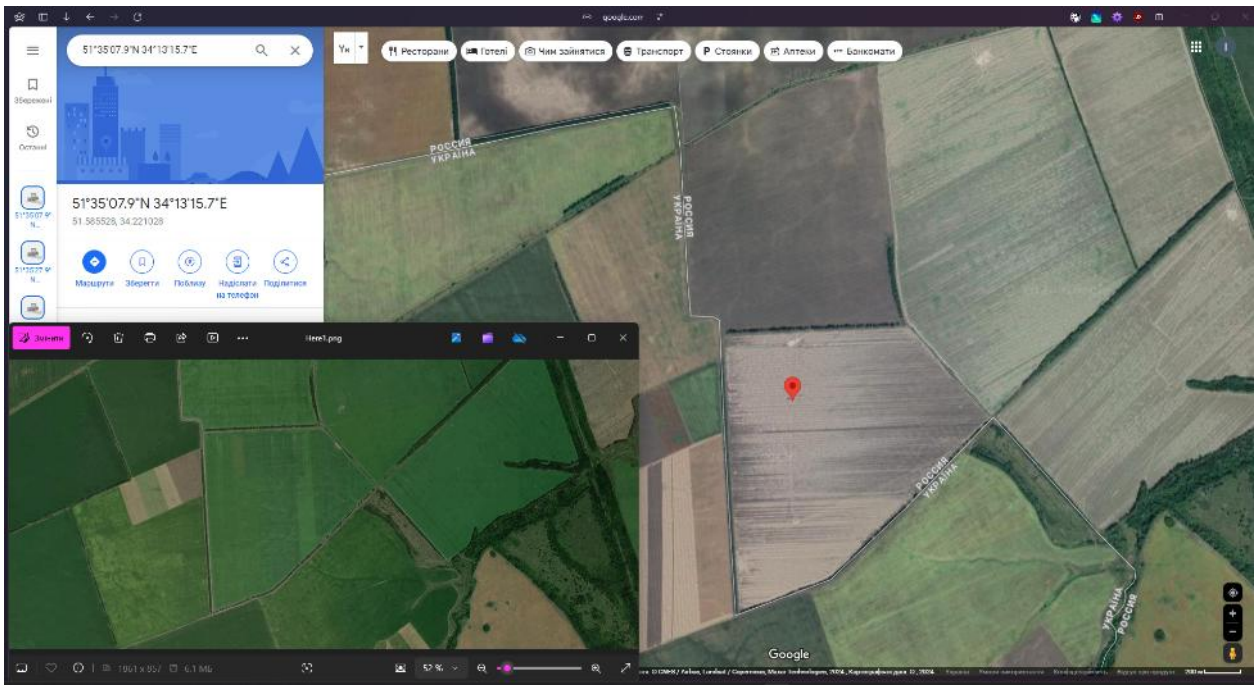

Рисунок 3.18 – Локація зображення here1

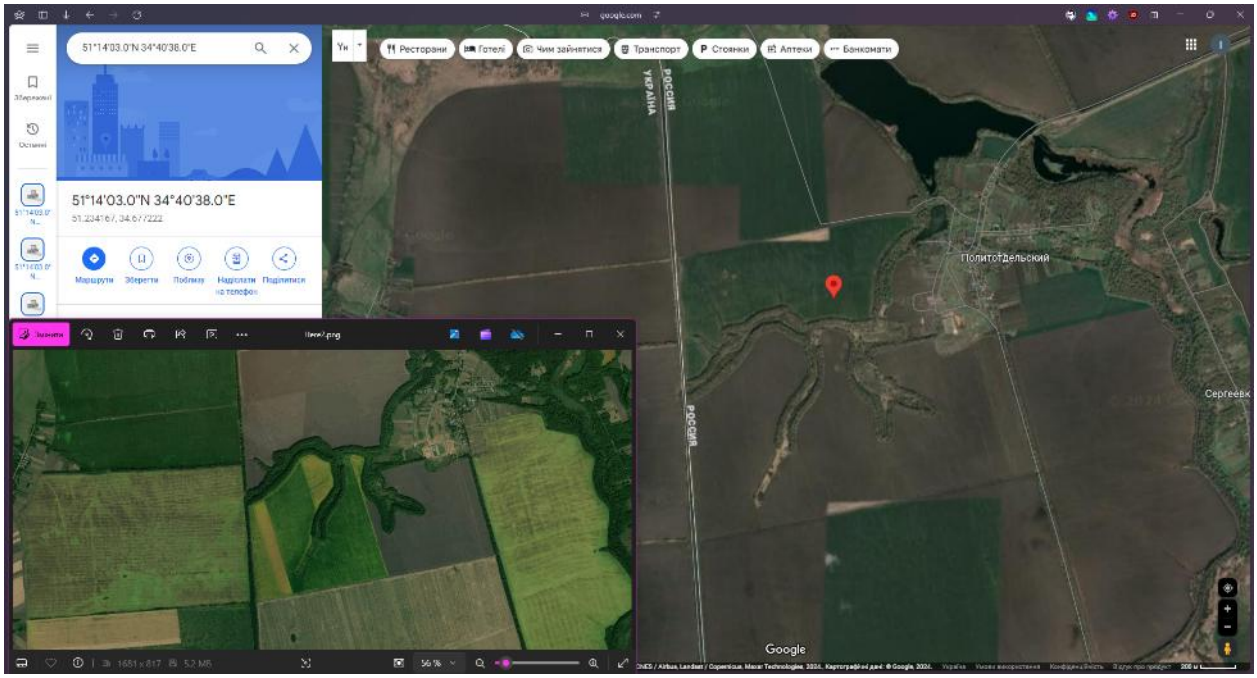

Рисунок 3.19 – Локація зображення here2

Цей успішний результат підкреслює ефективність обраного підходу до розробки алгоритму. Він підтверджує здатність системи працювати з різноманітними джерелами зображень і надає впевненість у можливості її подальшого застосування для реальних польових умов.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи магістра було розроблено інтелектуальну технологію позиціонування безпілотного літального апарату на основі машинного зору. Ця технологія демонструє високу ефективність і адаптивність до різних умов експлуатації. Основний принцип роботи алгоритму полягає у визначенні місцезнаходження дрона через аналіз зображень, отриманих із бортової камери, та їх зіставлення з попередньо завантаженою мапою місцевості.

У процесі виконання були досліджені існуючі методи позиціонування, що дозволило обрати оптимальний підхід для реалізації алгоритму. Алгоритм інтегрує сучасні рішення, такі як нейронна мережа SuperGlue, що забезпечує високу точність зіставлення зображень і продуктивність роботи системи. Його тестування на Даних з різних платформ, включаючи Maxar, Bing, Google і HERE, підтвердило універсальність і стабільність роботи. Алгоритм успішно обробляє всі тестові зображення, забезпечуючи точність визначення координат.

Результати роботи свідчать про значний потенціал запропонованої технології для застосування як в цивільних так і військових безпілотних літальних апаратах. Також розробка має потенціал з певними корективами дозволити визначати місцезнаходження не лише безпілотним літальним апаратам а також наземним та підводним дронам. Що дозволить більшій кількості безпілотних систем орієнтуватись на місцевості в умовах слабкого або відсутнього GPS сигналу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стасенко Д. В., Яковина В. С. Аналіз наявних методів і засобів удосконалення навігації БПЛА з використанням штучного інтелекту. Scientific Bulletin of UNFU. 2023. Т. 33, № 4. С. 78–83. URL: https://doi.org/10.36930/40330411

2. Arafat M. Y., Alam M. M., Moh S. Vision-Based navigation techniques for unmanned aerial vehicles: review and challenges. Drones. 2023. Т. 7, № 2. С. 89. URL: https://doi.org/10.3390/drones7020089 (дата звернення: 01.12.2024)

3. A review of UAV autonomous navigation in GPS-denied environments / Y. Chang та ін. Robotics and autonomous systems. 2023. С. 104533. URL: https://doi.org/10.1016/j.robot.2023.104533

4. A visual navigation system for UAV under diverse illumination conditions / J. Hai та ін. Applied artificial intelligence. 2021. Т. 35, № 15. С. 1529–1549. URL: https://doi.org/10.1080/08839514.2021.1985799

5. Comparison of three off-the-shelf visual odometry systems / A. Alapetite та ін. Robotics. 2020. Т. 9, № 3. С. 56. URL: https://doi.org/10.3390/robotics9030056

6. Computer vision for autonomous UAV flight safety: an overview and a vision-based safe landing pipeline example / E. Kakaletsis та ін. ACM computing surveys. 2022. Т. 54, № 9. С. 1–37. URL: https://doi.org/10.1145/3472288

7. Coupling of localization and depth data for mapping using Intel RealSense T265 and D435i cameras. arXiv.org. URL: https://arxiv.org/abs/2004.00269

8. Deep learning for vision-based navigation in autonomous drone racing / H. X. Pham та ін. Deep learning for robot perception and cognition. 2022. С. 371–406. URL: https://doi.org/10.1016/b978-0-32-385787-1.00020-8

9. Drone navigation using region and edge exploitation-based deep CNN / M. A. Arshad та ін. IEEE access. 2022. С. 1. URL:

https://doi.org/10.1109/access.2022.3204876

10. Flight in GPS-denied environment: autonomous navigation system for micro-aerial vehicle / H. Lu та ін. Aerospace science and technology. 2022. Т. 124. С. 107521. URL: https://doi.org/10.1016/j.ast.2022.107521

11. Google map aided visual navigation for UAVs in GPS-denied environment / M. Shan та ін. 2015 IEEE international conference on robotics and biomimetics (ROBIO), м. Zhuhai, 6–9 груд. 2015 р. 2015. URL: https://doi.org/10.1109/robio.2015.7418753

12. Hussien A. A literature review on visual-based UAV navigation in GPS-denied environments. Research Gate. 2024. URL: https://www.researchgate.net/publication/377235208_A_literature_review_on_visual-based_UAV_navigation_in_GPS-denied_environments

13. Mur-Artal R., Montiel J. M. M., Tardos J. D. ORB-SLAM: A versatile and accurate monocular SLAM system. IEEE transactions on robotics. 2015. Т. 31, № 5. С. 1147–1163. URL: https://doi.org/10.1109/tro.2015.2463671

14. ORB-SLAM map initialization improvement using depth / S. Fujimoto та ін. 2016 IEEE international conference on image processing (ICIP), м. Phoenix, AZ, USA, 25–28 верес. 2016 р. 2016. URL: https://doi.org/10.1109/icip.2016.7532359

15. SuperGlue: learning feature matching with graph neural networks. arXiv.org. URL: https://arxiv.org/abs/1911.11763

16. Turgeon T. GPS-Denied drones: navigating without signal. GNSS Jamming. URL: https://www.gnssjamming.com/post/gps-denied-drones (date of access: 01.12.2024).

17. Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction. arXiv.org. URL: https://doi.org/10.48550/arXiv.1803.03893

ДОДАТОК А

```
#crop


import os
import cv2

def crop(input_path, output_dir):
    # Перевіряємо, чи існує вихідний каталог, якщо ні - створюємо його
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    else:
        # Якщо каталог існує, видаляємо всі файли в ньому
        for file in os.listdir(output_dir):
            file_path = os.path.join(output_dir, file)
            if os.path.isfile(file_path):
                os.remove(file_path)

    # Завантажуємо зображення у відтінках сірого
    img = cv2.imread(input_path, cv2.IMREAD_GRAYSCALE)
    height, width = img.shape
    smaller_side = min(width, height)  # Знаходимо меншу сторону зображення
    larger_side = max(width, height)  # Знаходимо більшу сторону зображення

    # Обчислюємо крок для обрізки
    step = (larger_side - smaller_side) // 2

    # Обрізаємо зображення на три частини
    for i, offset in enumerate([0, step, larger_side - smaller_side]):
        if width > height:
            left = offset
            upper = 0
        else:
            left = 0
            upper = offset

        right = left + smaller_side
        lower = upper + smaller_side

        # Обрізаємо зображення
        cropped_img = img[upper:lower, left:right]

        # Видаляємо шум за допомогою медіанного фільтра
        cropped_img = cv2.medianBlur(cropped_img, 3)

        # Регулюємо яскравість та контрастність
        cropped_img = cv2.convertScaleAbs(cropped_img, alpha=1.5, beta=20)

        # Зберігаємо обрізане зображення у вихідний каталог
        output_file = os.path.join(output_dir, f"{i + 1}.png")
        cv2.imwrite(output_file, cropped_img)

crop('./camera/600.png', './c')  # Виклик функції crop для тестування
```

```
# filter_map


import os
from PIL import Image
import numpy as np
from concurrent.futures import ThreadPoolExecutor
import cv2

def is_black_image(image_path):
    # Відкриваємо зображення та перетворюємо його у масив
    image = Image.open(image_path)
    image_array = np.array(image)
    # Перевіряємо, чи всі пікселі чорні
    return np.all(image_array == 0)

def has_transparent_pixels(image_path):
    # Відкриваємо зображення та перетворюємо його у формат RGBA
    image = Image.open(image_path).convert("RGBA")
    image_array = np.array(image)
    # Рахуємо кількість прозорих пікселів
    transparent_pixels = np.sum(image_array[:, :, 3] < 255)
    total_pixels = image_array.shape[0] * image_array.shape[1]
    # Повертаємо відсоток прозорих пікселів
    return transparent_pixels / total_pixels

def process_image(filename, directory):
    print(f"Checking: {filename}")
    if filename.endswith('.png'):
        png_path = os.path.join(directory, filename)
        dat_path = os.path.join(directory, filename.replace('.png', '.dat'))

        # Перевіряємо, чи зображення чорне або має більше половини прозорих
пікселів
        if is_black_image(png_path) or has_transparent_pixels(png_path) >
0.5:
            if os.path.exists(png_path):
                os.remove(png_path)
            if os.path.exists(dat_path):
                os.remove(dat_path)
            print(f"Deleted: {png_path} and {dat_path}")
        else:
            # Перевести зображення в ЧБ
            image = cv2.imread(png_path, cv2.IMREAD_GRAYSCALE)

            # Видалити шум
            image = cv2.medianBlur(image, 3)

            # Відрегулювати яскравість та контрастність
            image = cv2.convertScaleAbs(image, alpha=1.5, beta=20)

            # Зберегти результат
            cv2.imwrite(png_path, image)
            print(f"Processed and saved: {png_path}")

def filter_map(directory):
    # Використовуємо ThreadPoolExecutor для паралельної обробки зображень
    with ThreadPoolExecutor() as executor:
        filenames = os.listdir(directory)
        futures = [executor.submit(process_image, filename, directory) for
filename in filenames]
        for future in futures:
            future.result()
```

```
# img2superpoint

import cv2
import torch
import os
from pathlib import Path
import pickle
from torch import nn
import shutil

def simple_nms(scores, nms_radius: int):

    # Перевіряємо, що радіус NMS не від'ємний
    assert(nms_radius >= 0)

    # Функція для виконання max pooling
    def max_pool(x):
        return torch.nn.functional.max_pool2d(
            x, kernel_size=nms_radius*2+1, stride=1, padding=nms_radius)

    zeros = torch.zeros_like(scores)
    max_mask = scores == max_pool(scores)

    for _ in range(2):
        supp_mask = max_pool(max_mask.float()) > 0
        supp_scores = torch.where(supp_mask, zeros, scores)
        new_max_mask = supp_scores == max_pool(supp_scores)
        max_mask = max_mask | (new_max_mask & (~supp_mask))
    return torch.where(max_mask, scores, zeros)

def remove_borders(keypoints, scores, border: int, height: int, width: int):
    # Маска для видалення ключових точок біля меж зображення
    mask_h = (keypoints[:, 0] >= border) & (keypoints[:, 0] < (height -
border))
    mask_w = (keypoints[:, 1] >= border) & (keypoints[:, 1] < (width -
border))
    mask = mask_h & mask_w
    return keypoints[mask], scores[mask]

def top_k_keypoints(keypoints, scores, k: int):
    # Якщо кількість ключових точок менша за k, повертаємо всі точки
    if k >= len(keypoints):
        return keypoints, scores
    # Вибираємо k ключових точок з найвищими оцінками
    scores, indices = torch.topk(scores, k, dim=0)
    return keypoints[indices], scores

def sample_descriptors(keypoints, descriptors, s: int = 8):
    # Нормалізуємо координати ключових точок
    b, c, h, w = descriptors.shape
    keypoints = keypoints - s / 2 + 0.5
    keypoints /= torch.tensor([(w*s - s/2 - 0.5), (h*s - s/2 - 0.5)],
                              ).to(keypoints)[None]
    keypoints = keypoints*2 - 1  # нормалізуємо до (-1, 1)

    # Вибираємо дескриптори, які відповідають ключовим точкам
    args = {'align_corners': True} if int(torch.__version__[2]) > 2 else {}
    descriptors = torch.nn.functional.grid_sample(
    descriptors, keypoints.view(b, 1, -1, 2), mode='bilinear', **args)
    descriptors = torch.nn.functional.normalize(
    descriptors.reshape(b, c, -1), p=2, dim=1)
    return descriptors

class SuperPoint(nn.Module):
```

```
default_config = {
    'descriptor_dim': 256,
    'nms_radius': 4,
    'keypoint_threshold': 0.005,
    'max_keypoints': -1,
    'remove_borders': 4,
}

# Конструктор класу
def __init__(self, config):
    # Викликаємо конструктор батьківського класу
    super().__init__()
    self.config = {**self.default_config, **config}

    # Створюємо шари нейронної мережі
    self.relu = nn.ReLU(inplace=True)
    self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
    c1, c2, c3, c4, c5 = 64, 64, 128, 128, 256

    # Шари для обчислення ключових точок
    self.conv1a = nn.Conv2d(1, c1, kernel_size=3, stride=1, padding=1)
    self.conv1b = nn.Conv2d(c1, c1, kernel_size=3, stride=1, padding=1)
    self.conv2a = nn.Conv2d(c1, c2, kernel_size=3, stride=1, padding=1)
    self.conv2b = nn.Conv2d(c2, c2, kernel_size=3, stride=1, padding=1)
    self.conv3a = nn.Conv2d(c2, c3, kernel_size=3, stride=1, padding=1)
    self.conv3b = nn.Conv2d(c3, c3, kernel_size=3, stride=1, padding=1)
    self.conv4a = nn.Conv2d(c3, c4, kernel_size=3, stride=1, padding=1)
    self.conv4b = nn.Conv2d(c4, c4, kernel_size=3, stride=1, padding=1)
    self.convPa = nn.Conv2d(c4, c5, kernel_size=3, stride=1, padding=1)
    self.convPb = nn.Conv2d(c5, 65, kernel_size=1, stride=1, padding=0)

    # Шари для обчислення дескрипторів
    self.convDa = nn.Conv2d(c4, c5, kernel_size=3, stride=1, padding=1)
    self.convDb = nn.Conv2d(
        c5, self.config['descriptor_dim'],
        kernel_size=1, stride=1, padding=0)

    # Завантажуємо ваги моделі
    path = Path(__file__).parent / 'weights/superpoint_v1.pth'
    self.load_state_dict(torch.load(str(path), weights_only = True))

    # Перевіряємо, що параметри моделі задовільняють обмеження
    mk = self.config['max_keypoints']
    if mk == 0 or mk < -1:
        raise ValueError('\"max_keypoints\" must be positive or \"-1\"')

# Метод для визначення ключових точок та дескрипторів
def forward(self, data):
    # Обчислюємо ключові точки
    x = self.relu(self.conv1a(data['image']))
    x = self.relu(self.conv1b(x))
    x = self.pool(x)
    x = self.relu(self.conv2a(x))
    x = self.relu(self.conv2b(x))
    x = self.pool(x)
    x = self.relu(self.conv3a(x))
    x = self.relu(self.conv3b(x))
    x = self.pool(x)
    x = self.relu(self.conv4a(x))
    x = self.relu(self.conv4b(x))

    # Оцінюємо ключові точки
    cPa = self.relu(self.convPa(x))
    scores = self.convPb(cPa)
```

```
        scores = torch.nn.functional.softmax(scores, 1)[:, :-1]
        b, _, h, w = scores.shape
        scores = scores.permute(0, 2, 3, 1).reshape(b, h, w, 8, 8)
        scores = scores.permute(0, 1, 3, 2, 4).reshape(b, h*8, w*8)
        scores = simple_nms(scores, 4)  # nms_radius

        # Визначаємо ключові точки з оцінкою більше порогового значення
        keypoints = [
            torch.nonzero(s > 0.005)  # keypoint_threshold
            for s in scores]
        scores = [s[tuple(k.t())] for s, k in zip(scores, keypoints)]

        # Видаляємо ключові точки біля меж зображення
        keypoints, scores = list(zip(*[
            remove_borders(k, s, 4, h*8, w*8)
            for k, s in zip(keypoints, scores)]))

        # Вибираємо кращі ключові точки
        keypoints, scores = list(zip(*[
            top_k_keypoints(k, s, 1024)
            for k, s in zip(keypoints, scores)]))
        keypoints = [torch.flip(k, [1]).float() for k in keypoints]

        # Обчислюємо дескриптори
        cDa = self.relu(self.convDa(x))
        descriptors = self.convDb(cDa)
        descriptors = torch.nn.functional.normalize(descriptors, p=2, dim=1)

        # Вибираємо дескриптори, які відповідають ключовим точкам
        descriptors = [sample_descriptors(k[None], d[None], 8)[0]
                       for k, d in zip(keypoints, descriptors)]

        return {
            'keypoints': keypoints,
            'scores': scores,
            'descriptors': descriptors,
        }

def img2superpoint(input_dir, output_path):
    # Вимикаємо автоматичне обчислення градієнтів
    torch.set_grad_enabled(False)

    # Завантажуємо модель SuperPoint
    superpoint = SuperPoint({}).eval().to('cpu')

    # Завантажуємо всі зображення з вказаної директорії
    input_path = Path(input_dir)
    all_images_name = os.listdir(input_path)
    all_images_name = [image_name for image_name in all_images_name if
image_name.endswith('.jpg')
                       or image_name.endswith('.png') or
image_name.endswith('jpeg')]

    # Створюємо вихідну директорію
    if not os.path.exists(output_path):
        os.makedirs(output_path)
    else:
        # Якщо директорія існує видаляємо вміст вихідної директорії
        for item in os.listdir(output_path):
            item_path = os.path.join(output_path, item)
            if os.path.isfile(item_path):
                os.remove(item_path)  # Видаляє файл
            elif os.path.isdir(item_path):
                shutil.rmtree(item_path)  # Видаляє каталог з усім вмістом
```

```
    # Обробляємо кожне зображення директорії
    for i, image_name in enumerate(all_images_name):
        stem_image_name = Path(image_name).stem

        # Завантажуємо зображення та перетворюємо його у масив
        image = cv2.imread(str(input_path / image_name),
cv2.IMREAD_GRAYSCALE)
        image = cv2.resize(image, (640, 480))

        # Видаляємо шум за допомогою медіанного фільтра
        image = image.astype('float32')

        # Відрегулювати яскравість та контрастність
        tensor_image = torch.from_numpy(image / 255.).float()[None,
None].to('cpu')

        # Передаємо зображення у модель SuperPoint
        pred = superpoint({'image': tensor_image})

        # Зберігаємо результат у файл .pickle
        with open(f'{output_path}/{stem_image_name}.pickle', 'wb') as file:
            pickle.dump(pred, file)
```

```
# rank2coords


import os
import pandas as pd

def read_coordinates_from_file(filename):
    # Відкриваємо файл для читання
    with open(filename, 'r') as file:
        lines = file.readlines()
        coordinates = []
        # Пропускаємо перший рядок (заголовок) і обробляємо кожен наступний
рядок
        for line in lines[1:]:
            try:
                # Розділяємо рядок на довготу і широту та перетворюємо їх у
числа з плаваючою комою
                lon, lat = map(float, line.strip().split(','))
                # Додаємо координати до списку
                coordinates.append((lat, lon))
            except ValueError:
                # Пропускаємо рядки, які не вдалося перетворити
                continue
        return coordinates

def convert_to_dms(deg):
    # Розбиваємо значення на градуси, хвилини та секунди
    d = int(deg)
    md = abs(deg - d) * 60
    m = int(md)
    sd = (md - m) * 60
    return d, m, sd

def format_coordinates(lat, lon):
    # Форматуємо широту та довготу у формат DMS (градуси, хвилини, секунди)
    lat_d, lat_m, lat_s = convert_to_dms(lat)
    lon_d, lon_m, lon_s = convert_to_dms(lon)

    # Визначаємо напрямок (північ/південь для широти, схід/захід для довготи)
    lat_direction = 'N' if lat >= 0 else 'S'
    lon_direction = 'E' if lon >= 0 else 'W'

    # Форматуємо координати у рядки
    formatted_lat = f"{abs(lat_d)}° {lat_m}' {lat_s:.4f}\" {lat_direction}"
    formatted_lon = f"{abs(lon_d)}° {lon_m}' {lon_s:.4f}\" {lon_direction}"

    return formatted_lat, formatted_lon

def calculate_center(coordinates):
    # Обчислюємо середнє значення широти та довготи
    avg_lat = sum(lat for lat, lon in coordinates) / len(coordinates)
    avg_lon = sum(lon for lat, lon in coordinates) / len(coordinates)
    return avg_lat, avg_lon

def rank2coords(csv_path, img_folder_path):
    # Зчитуємо файл CSV
    df = pd.read_csv(csv_path)

    # Перетворюємо стовпець score у числовий формат (без %)
    df['score'] = df['score'].str.rstrip('%').astype(float)

    # Знаходимо рядок із максимальним значенням score
    max_row = df.loc[df['score'].idxmax()]
    max_image = max_row['image']
```

```
    dat_file_path = os.path.join(img_folder_path, f"{max_image}.dat")

    # Перевіряємо, чи існує файл
    if os.path.exists(dat_file_path):
        # Читаємо вміст файлу .dat
        coordinates = read_coordinates_from_file(dat_file_path)
        # Обчислюємо центр координат
        center_lat, center_lon = calculate_center(coordinates)
        # Форматуємо координати у формат DMS
        formatted_lat, formatted_lon = format_coordinates(center_lat,
center_lon)
        return formatted_lat, formatted_lon
    else:
        # Виводимо повідомлення, якщо файл не знайдено
        print("Файл не знайдено за вказаним шляхом.")
```

```
# summary4csv

import os
import pandas as pd
import shutil

def summary4csv(main_directory, output_csv):
    # Шлях до файлу
    output_csv_path = os.path.join(main_directory, output_csv)

    # Перевірка чи існує файл, якщо так то видаляємо
    if os.path.exists(output_csv_path):
        os.remove(output_csv_path)

    summary_data = {}

    # Перевірка кожної підпапки
    for subdir in os.listdir(main_directory):
        subdir_path = os.path.join(main_directory, subdir)
        if os.path.isdir(subdir_path):
            # Пошук CSV файлу у підпапці
            csv_files = [f for f in os.listdir(subdir_path) if
f.endswith('.csv')]
            if len(csv_files) != 1:
                continue

            # Читання CSV файлу
            csv_path = os.path.join(subdir_path, csv_files[0])
            data = pd.read_csv(csv_path)

            # Перетворення стовпця `score` у числовий формат
            data['score'] = data['score'].str.rstrip('%').astype(float)

            # Сумування значень стовпця `score` за іменами зображень
            for _, row in data.iterrows():
                image = row['image']
                score = row['score']
                if image in summary_data:
                    summary_data[image] += score
                else:
                    summary_data[image] = score

    # Перетворення результатів у DataFrame
    summary_df = pd.DataFrame(list(summary_data.items()), columns=['image',
'score'])

    # Відфільтрувати лише ті рядки, де `image` не є числом
    summary_df = summary_df[~summary_df['image'].str.isnumeric()]

    # Сортування за зменшенням значення `score`
    summary_df = summary_df.sort_values(by='score', ascending=False)

    # Додавання символу `%` до значень `score`
    summary_df['score'] = summary_df['score'].apply(lambda x: f'{x:.3f}%')

    # Збереження результатів у новий CSV файл
    summary_df.to_csv(output_csv_path, index=False)

    # Видалення усіх каталогів у папці main_directory
    for subdir in os.listdir(main_directory):
        subdir_path = os.path.join(main_directory, subdir)
        if os.path.isdir(subdir_path):
            shutil.rmtree(subdir_path)
```

```
# superpoints2rank


import numpy as np
import pickle
import os
import torch
from pathlib import Path
from torch import nn
from copy import deepcopy
import pandas as pd
import shutil

def MLP(channels: list, do_bn=True):
    # Створюємо послідовну модель для MLP
    n = len(channels)
    layers = []
    for i in range(1, n):
        layers.append(
            nn.Conv1d(channels[i - 1], channels[i], kernel_size=1,
bias=True))
        if i < (n-1):
            if do_bn:
                layers.append(nn.BatchNorm1d(channels[i]))
            layers.append(nn.ReLU())
    return nn.Sequential(*layers)

class KeypointEncoder(nn.Module):
    # Клас для кодування ключових точок

    # Конструктор класу
    def __init__(self, feature_dim, layers):
        super().__init__()
        self.encoder = MLP([3] + layers + [feature_dim])
        nn.init.constant_(self.encoder[-1].bias, 0.0)

    # Метод для передачі даних через модель
    def forward(self, kpts, scores):
        inputs = [kpts.transpose(1, 2), scores.unsqueeze(1)]
        return self.encoder(torch.cat(inputs, dim=1))

def normalize_keypoints(kpts, image_shape):
    # Нормалізуємо координати ключових точок
    _, _, height, width = image_shape
    one = kpts.new_tensor(1)
    size = torch.stack([one*width, one*height])[None]
    center = size / 2
    scaling = size.max(1, keepdim=True).values * 0.7
    return (kpts - center[:, None, :]) / scaling[:, None, :]

def attention(query, key, value):
    # Обчислюємо ймовірність та вагу для кожної пари ключ-значення
    dim = query.shape[1]
    scores = torch.einsum('bdhn,bdhm->bhnm', query, key) / dim**.5
    prob = torch.nn.functional.softmax(scores, dim=-1)
    return torch.einsum('bhnm,bdhm->bdhn', prob, value), prob

class MultiHeadedAttention(nn.Module):
    # Клас для обчислення уваги для кількох голов

    # Конструктор класу
    def __init__(self, num_heads: int, d_model: int):
        super().__init__()
        assert d_model % num_heads == 0
```

```
        self.dim = d_model // num_heads
        self.num_heads = num_heads
        self.merge = nn.Conv1d(d_model, d_model, kernel_size=1)
        self.proj = nn.ModuleList([deepcopy(self.merge) for _ in range(3)])

    # Метод для передачі даних через модель
    def forward(self, query, key, value):
        batch_dim = query.size(0)
        query, key, value = [l(x).view(batch_dim, self.dim, self.num_heads, -
1)
                             for l, x in zip(self.proj, (query, key, value))]
        x, _ = attention(query, key, value)
        return self.merge(x.contiguous().view(batch_dim,
self.dim*self.num_heads, -1))

class AttentionalPropagation(nn.Module):
    # Клас для обчислення якості співпадіння ключових точок

    # Конструктор класу
    def __init__(self, feature_dim: int, num_heads: int):
        super().__init__()
        self.attn = MultiHeadedAttention(num_heads, feature_dim)
        self.mlp = MLP([feature_dim*2, feature_dim*2, feature_dim])
        nn.init.constant_(self.mlp[-1].bias, 0.0)

    # Метод для передачі даних через модель
    def forward(self, x, source):
        message = self.attn(x, source, source)
        return self.mlp(torch.cat([x, message], dim=1))

class AttentionalGNN(nn.Module):
    # Клас для обчислення зваженої суми дескрипторів

    # Конструктор класу
    def __init__(self, feature_dim: int, layer_names: list):
        super().__init__()
        self.layers = nn.ModuleList([
            AttentionalPropagation(feature_dim, 4)
            for _ in range(len(layer_names))])
        self.names = layer_names

    # Метод для передачі даних через модель
    def forward(self, desc0, desc1):
        for layer, name in zip(self.layers, self.names):
            if name == 'cross':
                src0, src1 = desc1, desc0
            else:  # if name == 'self':
                src0, src1 = desc0, desc1
            delta0, delta1 = layer(desc0, src0), layer(desc1, src1)
            desc0, desc1 = (desc0 + delta0), (desc1 + delta1)
        return desc0, desc1

def log_sinkhorn_iterations(Z, log_mu, log_nu, iters: int):
    # Обчислюємо стабільну оптимальну транспортну функцію
    u, v = torch.zeros_like(log_mu), torch.zeros_like(log_nu)
    for _ in range(iters):
        u = log_mu - torch.logsumexp(Z + v.unsqueeze(1), dim=2)
        v = log_nu - torch.logsumexp(Z + u.unsqueeze(2), dim=1)
    return Z + u.unsqueeze(2) + v.unsqueeze(1)

def log_optimal_transport(scores, alpha, iters: int):
    # Обчислюємо оптимальну транспортну функцію

    # Розмірність матриці
```

```
    b, m, n = scores.shape
    one = scores.new_tensor(1)
    ms, ns = (m*one).to(scores), (n*one).to(scores)

    bins0 = alpha.expand(b, m, 1)
    bins1 = alpha.expand(b, 1, n)
    alpha = alpha.expand(b, 1, 1)

    # Об'єднуємо оцінки та бінні значення
    couplings = torch.cat([torch.cat([scores, bins0], -1),
                           torch.cat([bins1, alpha], -1)], 1)

    # Логарифмуємо оцінки та бінні значення
    norm = - (ms + ns).log()
    log_mu = torch.cat([norm.expand(m), ns.log()[None] + norm])
    log_nu = torch.cat([norm.expand(n), ms.log()[None] + norm])
    log_mu, log_nu = log_mu[None].expand(b, -1), log_nu[None].expand(b, -1)

    # Обчислюємо оптимальну транспортну функцію
    Z = log_sinkhorn_iterations(couplings, log_mu, log_nu, iters)
    Z = Z - norm
    return Z

def arange_like(x, dim: int):
    # Створюжємо тензор з послідовністю чисел
    return x.new_ones(x.shape[dim]).cumsum(0) - 1

class SuperGlue(nn.Module):
    # Клас для обчислення співпадіння ключових точок
    default_config = {
        'descriptor_dim': 256,
        'weights': 'indoor',
        'keypoint_encoder': [32, 64, 128, 256],
        'GNN_layers': ['self', 'cross'] * 9,
        'sinkhorn_iterations': 100,
        'match_threshold': 0.2,
    }

    # Конструктор класу
    def __init__(self, config):
        super().__init__()
        self.config = {**self.default_config, **config}

        self.kenc = KeypointEncoder(
            self.config['descriptor_dim'], self.config['keypoint_encoder'])

        self.gnn = AttentionalGNN(
            self.config['descriptor_dim'], self.config['GNN_layers'])

        self.final_proj = nn.Conv1d(
            self.config['descriptor_dim'], self.config['descriptor_dim'],
            kernel_size=1, bias=True)

        # Завантажуємо ваги для моделі
        bin_score = torch.nn.Parameter(torch.tensor(1.))
        self.register_parameter('bin_score', bin_score)
        assert self.config['weights'] in ['indoor', 'outdoor']
        path = Path(__file__).parent
        path = path /
'weights/superglue_{}.pth'.format(self.config['weights'])
        self.load_state_dict(torch.load(str(path), weights_only = True))

    def forward(self, data):
        # Передача даних через модель
```

```
        desc0, desc1 = data['descriptors0'], data['descriptors1']
        kpts0, kpts1 = data['keypoints0'], data['keypoints1']

        # Перевірка наявності ключових точок
        if kpts0.shape[1] == 0 or kpts1.shape[1] == 0:
            shape0, shape1 = kpts0.shape[:-1], kpts1.shape[:-1]
            return {
                'matches0': kpts0.new_full(shape0, -1, dtype=torch.int),
                'matches1': kpts1.new_full(shape1, -1, dtype=torch.int),
                'matching_scores0': kpts0.new_zeros(shape0),
                'matching_scores1': kpts1.new_zeros(shape1),
            }

        # Нормалізуємо координати ключових точок
        kpts0 = normalize_keypoints(kpts0, data['image0'].shape)
        kpts1 = normalize_keypoints(kpts1, data['image1'].shape)

        # Кодуємо ключові точки
        desc0 = desc0 + self.kenc(kpts0, data['scores0'])
        desc1 = desc1 + self.kenc(kpts1, data['scores1'])

        # Передача даних через графічну нейронну мережу
        desc0, desc1 = self.gnn(desc0, desc1)

        # Проектуємо дескриптори
        mdesc0, mdesc1 = self.final_proj(desc0), self.final_proj(desc1)

        # Обчислюємо оцінки співпадіння
        scores = torch.einsum('bdn,bdm->bnm', mdesc0, mdesc1)
        scores = scores / self.config['descriptor_dim']**.5

        # Обчислюємо оптимальну транспортну функцію
        scores = log_optimal_transport(
            scores, self.bin_score,
            iters=self.config['sinkhorn_iterations'])

        # Обчислюємо співпадіння
        max0, max1 = scores[:, :-1, :-1].max(2), scores[:, :-1, :-1].max(1)
        indices0, indices1 = max0.indices, max1.indices
        mutual0 = arange_like(indices0, 1)[None] == indices1.gather(1,
indices0)
        mutual1 = arange_like(indices1, 1)[None] == indices0.gather(1,
indices1)
        zero = scores.new_tensor(0)
        mscores0 = torch.where(mutual0, max0.values.exp(), zero)
        mscores1 = torch.where(mutual1, mscores0.gather(1, indices1), zero)
        valid0 = mutual0 & (mscores0 > self.config['match_threshold'])
        valid1 = mutual1 & valid0.gather(1, indices1)
        indices0 = torch.where(valid0, indices0, indices0.new_tensor(-1))
        indices1 = torch.where(valid1, indices1, indices1.new_tensor(-1))

        return {
            'matches0': indices0,
            'matches1': indices1,
            'matching_scores0': mscores0,
            'matching_scores1': mscores1,
        }

# Функція для обчислення оцінки співпадіння
def ranking_score(matches, match_confidence):
    return np.sum(np.multiply(matches, match_confidence)).astype(np.float32)

# Функція для завантаження файлу .pickle
def load_pickle(path):
```

```
    with open(path, 'rb') as file:
        loaded = pickle.load(file)
    return loaded

# Функція для обробки файлу .pickle
def process_superpoints(file, input_dir, output_dir):
    # Отримуємо ім'я файлу з його шляху
    main_file = os.path.basename(file)
    # Створюємо новий шлях для файлу в каталозі ./data
    new_file_path = os.path.join(input_dir, main_file)
    # Копіюємо файл
    shutil.copy(os.path.join(input_dir, file), new_file_path)

    # Створюємо словник для збереження оцінок
    score_dict = {}

    # ініціалізуємо шляхи
    input_dir = Path(input_dir)
    output_dir = Path(output_dir)
    output_dir.mkdir(exist_ok=True, parents=True)
    all_file_name = os.listdir(input_dir)

    # Знаходимо всі файли .pickle в input_dir
    pairs = [(main_file, file_name) for file_name in all_file_name if
file_name.endswith('.pickle')]

    config = {
        'superglue': {
            'weights': 'indoor',
            'sinkhorn_iterations': 20,
            'match_threshold': 0.2,
        }
    }

    # Завантажуємо модель SuperGlue
    superglue = SuperGlue(config).eval().to('cpu')

    # Обробляємо кожну пару файлів
    for i, pair in enumerate(pairs):
        name0, name1 = pair[:2]
        stem0, stem1 = Path(name0).stem, Path(name1).stem
        matches_path = output_dir / '{}_{}_matches.npz'.format(stem0, stem1)
        do_match = True

        # Завантажуємо файли .pickle
        superpoints_0 = load_pickle(str(input_dir / name0))
        superpoints_1 = load_pickle(str(input_dir / name1))

        # Перевіряємо наявність ключових точок
        superpoints_0 = {k + '0': v for k, v in superpoints_0.items()}
        superpoints_1 = {k + '1': v for k, v in superpoints_1.items()}

        # Перевіряємо наявність ключових точок
        if superpoints_0 is None or superpoints_1 is None:
            exit(1)

        # Передача даних через модель SuperGlue
        dummy_data = {'image0': np.zeros((1, 1, 1000, 1000)),
                      'image1': np.zeros((1, 1, 1000, 1000))}
        data = {**dummy_data, **superpoints_0, **superpoints_1}
        for k in data:
            if isinstance(data[k], (list, tuple)):
                data[k] = torch.stack(data[k])
        pred = superglue(data)
```

```
        pred = {k: v[0].cpu().numpy() for k, v in pred.items()}

        kpts0, kpts1 = superpoints_0['keypoints0'][0].cpu().numpy(),
superpoints_1['keypoints1'][0].cpu().numpy()
        matches, conf = pred['matches0'], pred['matching_scores0']

        # Зберігаємо результат у файл
        out_matches = {'keypoints0': kpts0, 'keypoints1': kpts1,
                       'matches': matches, 'match_confidence': conf}

        # Обчислюємо оцінку співпадіння
        score_dict[stem1] = ranking_score(matches, conf)

        # Перевіряємо чи ім'я файлу співпадає
        if name0 == name1:
            full_score = score_dict[stem1]

        # Зберігаємо результат у файл .npz
        np.savez(str(matches_path), **out_matches)

    # Сортуємо оцінки
    ranked_images = {k: v for k, v in sorted(score_dict.items(),
reverse=True, key=lambda x: x[1])}
    ranked_images_percentage = {k: f'{((v / full_score) * 100):.3f}%' for k,
v in ranked_images.items()}

    # Зберігаємо результат у файл .csv
    df = pd.DataFrame.from_dict(ranked_images_percentage, orient='index',
columns=['score'])
    df.reset_index(inplace=True)
    df.rename(columns={'index': 'image'}, inplace=True)
    df.to_csv(str(output_dir / 'ranking_score.csv'), index=True)

    # Після завершення роботи видаляємо копію файлу
    if os.path.exists(new_file_path):
        os.remove(new_file_path)

# Функція для обчислення оцінки співпадіння
def superpoints2rank (input_dir, output_base_dir):
    # Встановлюємо параметри для обчислення
    os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"

    # Вимикаємо автоматичне обчислення градієнтів
    torch.set_grad_enabled(False)

    # Знаходимо всі файли .pickle в input_dir
    pickle_files = [f for f in os.listdir(input_dir) if
f.endswith('.pickle')]

    input_dir = os.path.dirname(input_dir.rstrip('/'))

    # Запускаємо process_superpoints для кожного файлу
    for pickle_file in pickle_files:
        file_path = f'frame_superpoints/{pickle_file}'
        output_dir, _ = os.path.splitext(os.path.join(output_base_dir,
pickle_file))
        process_superpoints(file_path, input_dir, output_dir)
```