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)
грудня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»
освітньо-професійної програми «Інформатика»
на тему: Інформаційна технологія аналізу даних візуального спостереження для навігації малогабаритного безпілотного апарату
здобувача групи ІН.м – 33 Тюльпа Сергій Олександрович

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

Тюльпа Сергій
(підпис)

Керівник
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-33 Тюльпа Сергія Олександровича

1. Тема роботи: ««Інформаційна технологія ……»
затверджую наказом по СумДУ від *«03» листопада 2024 року № 1257-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до 06 грудня 2024 року*

3. Вхідні дані до кваліфікаційної роботи
послідовність зображень отриманих із камери БПЛА

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд сучасних методів візуальної одометрії та технологій навігації. 3) Розробка алгоритму візуальної одометрії для БПЛА. 4) Проведення тестів із варіаціями параметрів алгоритму та оцінка впливу на точність. 5) Аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. 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*	20.08.24 – 27.08.24	
2	*Огляд технологій, що використовуються для побудови алгоритмів візуальної одометрії*	27.08.24 – 05.09.24	
3	*Розробка інтелектуальної системи візуальної одометрії*	06.09.24 – 28.10.24	
4	*Аналіз отриманих результатів*	28.10.24 – 30.10.24	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	01.11.24 – 30.11.24	

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

АНОТАЦІЯ

Записка: 121 с., 45 рис., 16 табл., 13 джерел

Обґрунтування актуальності теми роботи: Тема кваліфікаційної роботи є актуальною, оскільки присвячена вирішенню важливої практичної задачі підвищення точності навігаційних систем малогабаритних безпілотних апаратів за допомогою розробки та впровадження методів і алгоритмів візуальної одометрії.

Об’єкт дослідження: Процеси, пов’язані з навігацією малогабаритних безпілотних літальних апаратів, які здійснюють орієнтацію та переміщення у просторі за допомогою аналізу візуальної інформації, отриманої з камер.

Мета дослідження: Розробка та впровадження інформаційної технології аналізу даних візуального спостереження для забезпечення точного функціонування навігаційних систем малогабаритних безпілотних апаратів.

Методи дослідження: У роботі використано методи аналізу візуальних даних, алгоритми трекінгу ознак, оцінки руху та побудови траєкторій за допомогою візуальної одометрії.

Результати: Розроблено інформаційну систему для аналізу візуальних даних, яка дозволяє малогабаритним БПЛА орієнтуватися у просторі. Проведено тестування алгоритму візуальної одометрії, виконано налаштування параметрів. Отримано траєкторії руху БПЛА, які відображають результати аналізу зібраних візуальних даних.

Ключові слова: навігація, безпілотний літальний апарат, візуальна одометрія, інтелектуальний аналіз зображень, стерео бачення, монокулярне бачення, машинне навчання, картографування, автономна навігація.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АНАЛІЗУ ДАНИХ ВІЗУАЛЬНОГО СПОСТЕРЕЖЕННЯ ДЛЯ НАВІГАЦІЇ МАЛОГАБАРИТНОГО БЕЗПІЛОТНОГО АПАРАТУ

ЗМІСТ

Перелік умовних позначень, символів, скорочень і термінів

БПЛА – Безпілотний літальний апарат.

SLAM – Одночасна локалізація і картографування (The simultaneous localization and mapping).

GPS – Глобальна Система Позиціонування (Global Positioning System).

IMU – Інерційний вимірювальний блок (Inertial Measurement Unit).

GNSS – Глобальна Навігаційна Супутникова Система (Global Navigation Satellite System).

MAV – Мініатюрний літальний апарат (Miniature Aerial Vehicle).

VO – Візуальна одометрія (Visual odometry).

VIO – Візуально-інерційна одометрія (Visual-Inertial Odometry).

BA – Зв'язкова оптимізація (Bundle Adjustment).

CNN – Згорткова нейронна мережа (Convolutional Neural Network).

RNN – Рекурентна нейронна мережа (Recurrent Neural Network).

НС – Надзвичайна ситуація.

LSTM – Довгострокова короткочасна пам'ять (Long Short-Term Memory).

GRU – Вхідний рекурентний блок (Gated Recurrent Unit).

Інлайери — це точки, що добре узгоджуються з математичною моделлю, яка розроблена для руху чи трансформації.

MAE – Середня абсолютна помилка (Mean Absolute Error)

MSE – Середня квадратична помилка (Mean Squared Error)

RMSE – Квадратний корінь середньоквадратичної помилки (Root Mean Squared Error).

Ground truth – Реальне положення.

GUI – Графічний інтерфейс користувача (Graphical user interface).

FOE – Точка сходження FOE (Focus of Expansion).

ВСТУП

Актуальність: БПЛА відіграють ключову роль у різних сферах, таких як оборонна, аграрна, інфраструктурна, моніторинг та рятувальні операції. У складних умовах, де доступ до GPS-сигналу обмежений, наприклад у міських забудовах або закритих приміщеннях, навігація БПЛА стає викликом. Використання візуальної навігації на основі аналізу зображень та відео є перспективним підходом для забезпечення автономності й точності роботи БПЛА. Зростання потреб у компактних, енергоефективних та високоточних рішеннях робить дослідження цієї теми особливо актуальним.

Об’єкт дослідження: Процеси, пов’язані з навігацією малогабаритних безпілотних літальних апаратів, які здійснюють орієнтацію та переміщення у просторі за допомогою аналізу візуальної інформації, отриманої з камер.

Предмет дослідження: Методи та алгоритми аналізу візуальних даних, що використовуються для навігації малогабаритних безпілотних апаратів у контексті візуальної одометрії.

Новизна: Робота пропонує інтеграцію сучасних методів візуальної одометрії та алгоритмів обробки зображень у навігаційні системи малогабаритних БПЛА.

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

1. АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасний стан та тенденції розвитку технологій навігації малогабаритних безпілотних апаратів

БПЛА – це літальні апарати які не мають екіпажу на борту, вони мають різний ступінь автономності – від керованих дистанційно до повністю автоматичних, а також відрізняються за конструкцією, призначенням і безліччю інших параметрів. Типи дронів та їх порівняння можна побачити у таблиці 1.1.

Таблиця 1.1 – Типи дронів [1]

Типи дронів	Розміри	Вантажопідйомність	Застосування
Нано-дрони	До 250 грам	Декілька грам	Розвідка, спостереження, військові операції у вузьких просторах
Мікро-дрони	Від 250 грам до 2 кілограм	До 2 кілограм	Відеозйомка, спостереження, цивільні завдання в умовах міської забудови
Малі дрони	Від 2 до 25 кілограмів	До 5 кілограм	Аерофотозйомка, інспекції, картографування
Середні дрони	Від 25 до 150 кілограмів	Від 5 до 50 кілограм	Спостереження, логістика, сільське господарство
Великі дрони	Більше 150 кілограмів	Понад 50 кілограм	Вантажні перевезення, довготривала аерофотозйомка, військові місії

Основні аспекти від яких залежить розвиток технологій навігаційних систем для БПЛА:

- точність;
- складність;
- вартість;

– загальна адаптивність технологій.

Також можна відмітити такі технології як:

– GNSS;

– інерційні навігаційні системи;

– різноманітні візуальні навігаційні системи.

Такі технології зараз масово впроваджуються для покращення точності в умовах недостатнього або заглушеного сигналу, що особливо стало актуально після початку повномасштабної війни і величезного використання дронів на фронті. Технологічні розробки в цій сфері спричинені у першу чергу військовою необхідністю і це не дивно, бо під час війн ставалося багато технологічних проривів.

1.1.1 Актуальність навігаційних технологій для малогабаритних безпілотних апаратів

Технології навігації для БПЛА постійно вдосконалюються, щоб забезпечити їх ефективність і автономність. Важливо, щоб системи навігації використовували доступні технології без необхідності у власному апаратному забезпеченні, що забезпечує масштабне впровадження таких систем. Системи навігації, ймовірно, будуть впроваджені в широкому масштабі, оскільки ці технології не потребують специфічного апаратного забезпечення. Проте GPS-чіпи, які часто використовуються в безпілотниках, не забезпечують високої точності навігації і можуть демонструвати помилки до кількох метрів. Для забезпечення безперешкодного маршруту та зменшення витрат енергії необхідно створювати 3D-карти, проте це стає дедалі складнішим через динамічні та кінематичні обмеження БПЛА, що ускладнює розв'язання задачі локальних мінімумів у сучасних алгоритмах планування маршрутів [2].

Крім того, сучасні навігаційні системи повинні враховувати динамічні зміни середовища, такі як зміни погодних умов або взаємодія з іншими

об'єктами, що ще більше ускладнює їх реалізацію [2]. Це вимагає розробки нових підходів до інтеграції даних та адаптації алгоритмів до реальних умов.

1.1.2 Доступність та складність навігаційних технологій

Складність навігаційної системи є важливим аспектом при проектуванні систем зв'язку для дронів, оскільки вона часто пов'язана з підвищеними вимогами до енергетичних ресурсів, інфраструктури та обчислювальних потужностей. Складні системи можуть не підходити для компактних безпілотників, які мають обмежені можливості обробки даних та споживання енергії. Наприклад, навігація в реальному часі, особливо при обробці зображень, значно збільшує обчислювальну складність, що робить управління в межах обмеженого споживання батареї та обчислювальної потужності однією з основних проблем для БПЛА [3].

Сучасні системи навігації також потребують високої точності при обробці даних сенсорів. Це може вимагати спеціалізованого апаратного забезпечення для обробки даних у реальному часі, що ускладнює інтеграцію таких систем у малогабаритні апарати. Зокрема, необхідність обробки великої кількості даних може призвести до збільшення ваги та зниження тривалості польоту БПЛА [3].

1.1.3 Складність та вартість систем

Вартість, точність, узагальненість і масштабованість навігаційних систем залежать від їх складності. Високі вимоги до потужності та інфраструктури часто пов'язані з високою складністю системи. Системи з високою складністю зазвичай потребують більшої потужності, інфраструктури та обчислювальних ресурсів. Навігаційні системи для БПЛА потребують обробки великої кількості даних від сенсорів у реальному часі, що є викликом для малих апаратів з обмеженими ресурсами.

З іншого боку, спрощення навігаційних систем може знижувати їх точність або можливості, що також є важливим аспектом при виборі системи

для конкретного завдання. Вартість розробки та впровадження таких систем може варіюватися в залежності від їх складності та вимог до точності [3].

1.1.4 Методи Візуальної одометрії

Візуальна одометрія – процес визначення позиції і орієнтації робота шляхом аналізу послідовних зображень отриманих за допомогою камери.

Типовий алгоритм візуальної одометрії складається з таких операцій: отримання зображення з камери/стереокамери/панорамних камер, корекція зображення, детектування ключових точок зображення, перевірка векторів оптичного потоку на потенціальні похибки, визначення руху камери за оптичним потоком, періодичне оновлення набору ключових точок для відстеження.

Перевагою такого алгоритму є його універсальність, а недоліки є такі:

- алгоритм практично не працює з однотипними зображеннями;
- необхідність високої швидкості захоплення зображення (при використанні систем з частотою отримання кадрів порядку десятків кГц);
- високе обчислювальне навантаження та висока вартість камер [4].

Більшість ранніх підходів використовували стереосистему, але це не єдине рішення. Алгоритми VO класифікуються різними способами. Простий спосіб – звернутися до налаштування камери. Якщо використовуються дві камери, це називається стерео VO, а якщо одна камера використовується вона називається монокуляр VO. Деякі рішення використовують більше ніж дві камери.

Існують два основні підходи до оцінки руху в задачах візуальної одометрії: геометричний та заснований на методах машинного навчання (рис 1.1).

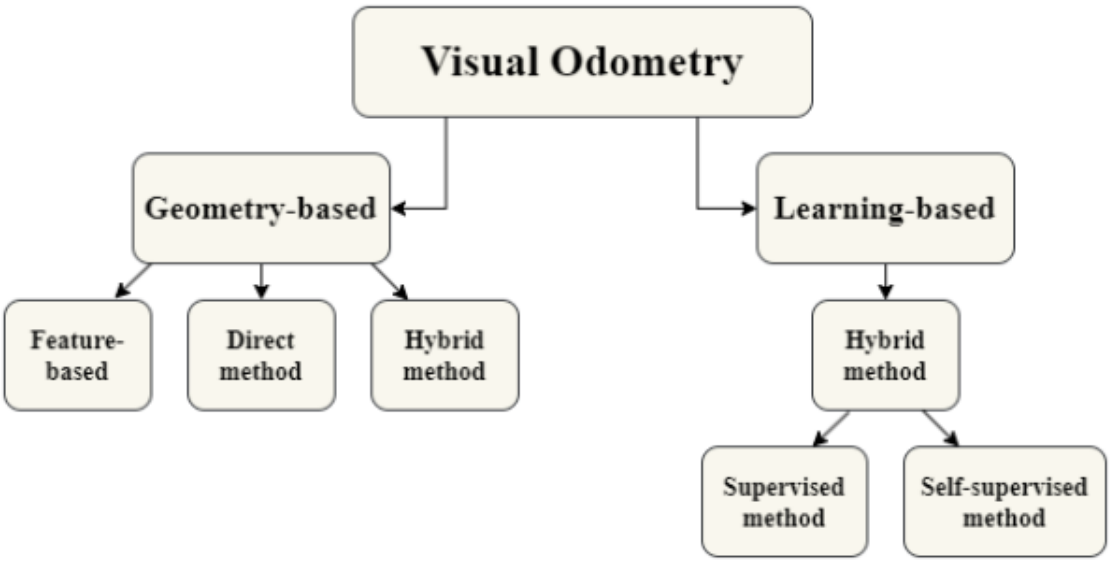

Рисунок 1.1 – Методи Візуальної одометрії

Стерео бачення – це метод оцінки руху робота або транспортного засобу, використовуючи дані від стереокамери, яка містить дві окремі камери, розташовані на певній відстані одна від одної (базова лінія). Завдяки цьому можна отримувати об'ємні 3D-зображення та інформацію про глибину сцени, що дозволяє точно відслідковувати рух у просторі.

Принцип роботи стерео бачення полягає в тому, що дві камери одночасно отримують зображення однієї й тієї ж сцени з різних ракурсів. Далі здійснюється пошук відповідностей між точками на зображеннях, отриманих з обох камер. Цей процес називається кореляцією або пошуком відповідностей. Після цього, виходячи з відмінностей у положенні відповідних точок на двох зображеннях, обчислюється відстань до кожного об'єкта на сцені, що дозволяє створити тривимірну карту середовища. На основі змін у положенні об'єктів на послідовних кадрах можна визначити, як змінювався просторовий стан камери, що дозволяє обчислити її траєкторію руху.

ORB-SLAM2 це один з найпопулярніших методів для стерео бачення. Цей метод дозволяє системам, оснащеним камерами, відстежувати своє місцезнаходження і створювати карту оточення у режимі реального часу.

Основні етапи цього методу включають відстеження, де на кожному кадрі алгоритм виділяє ключові точки та порівнює ORB-ознаки з попередніми кадрами для визначення руху камери, зберігаючи лише ключові кадри для зменшення обчислювальних витрат. Далі алгоритм будує 3D-карту на основі ключових кадрів, де кожна точка відповідає певній ORB-ознаці. Останнім етапом є закриття петлі: коли система повертається до раніше відвіданого місця, вона розпізнає це за збігами в ключових кадрах і коригує карту для усунення накопичених помилок.

ELAS метод базується на створенні набору підтримуючих точок, розташованих по всій площині зображення, що забезпечують базові орієнтири для оцінки глибини.

Основні етапи цього методу включають побудову підтримуючих точок, де на зображенні визначаються ключові точки, стійкі до змін положення та освітлення, які слугують опорними орієнтирами для інтерполяції глибини інших точок сцени. Далі за допомогою інтерполяції від підтримуючих точок обчислюється глибина для всіх пікселів, що дозволяє створити 3D-карту сцени, оцінюючи відстань до кожної точки. Для збереження точності навіть у зонах зі слабко вираженими ознаками використовується метод регуляризації, що дозволяє отримати безперервну і надійну глибинну карту.

Stereo VIO - цей метод комбінує стерео камеру з IMU для досягнення точності в умовах низької освітленості або при відсутності чітких текстур. Stereo VIO обробляє як візуальні дані із стереокамери, так і IMU, зокрема інформацію про прискорення та кутову швидкість, що забезпечує додаткову стабільність та точність у складних умовах.

Основні етапи цього методу включають злиття даних стерео-камери та IMU, що дозволяє отримувати точніші результати, особливо при швидких рухах, де візуальної інформації може бути недостатньо. Для зменшення впливу шуму та помилок використовуються методи фільтрації, такі як фільтр Калмана, що підвищує стабільність оцінки траєкторії.

Монокулярне бачення це метод оцінки руху та положення транспортного засобу або робота, використовуючи зображення, отримані від однієї камери (на відміну від стереовізії, де використовуються дві камери). Монокулярна камера надає лише двовимірні зображення сцени, і головним викликом у цій технології є відсутність прямої інформації про глибину.

Принцип роботи монокулярного бачення полягає в тому, що камера знімає послідовність зображень сцени під час руху. Алгоритм виділяє ключові точки, такі як кути або грані, на кожному зображенні. Далі ці точки відстежуються на наступних кадрах, і визначається, як змінилося їхнє положення. Зміна положення ключових точок використовується для оцінки переміщення камери. Оскільки камера забезпечує лише двовимірну інформацію, для відновлення тривимірної структури використовуються математичні моделі. Одна з основних проблем монокулярної одометрії — це відсутність абсолютного масштабу, оскільки глибина сцени невідома. Наприклад, на зображенні видно, як зміна масштабу об'єктів (їх зменшення або збільшення в міру наближення чи віддалення від камери) може використовуватися для оцінки глибини (рис. 1.2). Для вирішення цієї проблеми можуть застосовуватися додаткові припущення або використовуватися інші сенсори, такі як інерційні системи.

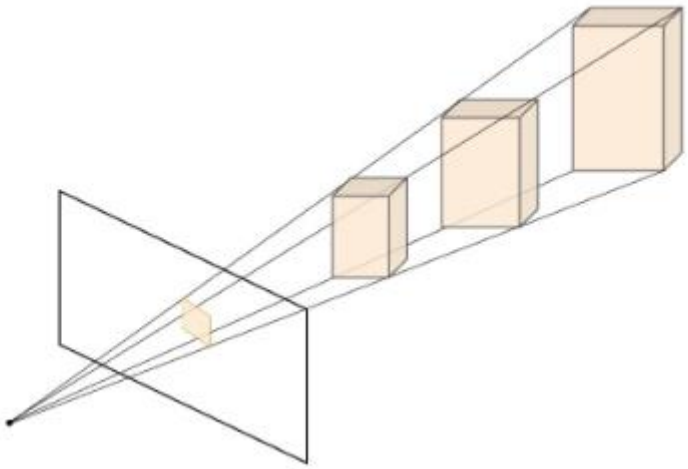

Рисунок 1.2 – Ілюстрація монокулярного сприйняття глибини

Об'єкти з різними розмірами проектуватимуться на однакові пікселі зображення, якщо відстань до них відповідає фіксованому співвідношенню. Це означає, що неможливо обчислити точну оцінку розміру об'єкта без інших посилань, таких як порівняння масштабу або оцінка дальності.

SIVO є методом оцінки руху, який використовує лише зображення з однієї камери, при цьому він не залежить від фіксованого масштабу. Метод дозволяє відновити траєкторію камери, навіть коли сцена має варіативний масштаб.

Основні етапи цього методу включають виділення та відстеження ключових точок, де алгоритм знаходить та відслідковує особливі точки, такі як кути або точки з високою контрастністю, на кожному зображенні. Далі рух обчислюється шляхом порівняння положення ключових точок між кадрами, що дозволяє оцінити переміщення камери відносно сцени. Для корекції масштабу використовуються додаткові алгоритми, які враховують зміни у вигляді об'єктів на зображеннях, що дозволяє точно визначати відстань і переміщення в умовах змінного масштабу.

Optical flow – це метод аналізу руху в сцені, що базується на вимірюванні зміни положення пікселів між двома кадрами. За допомогою цього методу можна оцінити напрямок та швидкість руху об'єктів, а також рух камери.

Основні етапи цього методу включають обчислення потоку пікселів за допомогою методів, таких як метод Лукаса-Канаде, для визначення зміщення пікселів між двома зображеннями, що дозволяє оцінити рух кожної точки сцени. На основі отриманих змін пікселів визначається загальна траєкторія руху об'єкта або камери. Крім того, проводиться корекція руху для врахування можливих помилок через неідеальне збереження інтенсивності пікселів, шуму чи складних умов руху.

Геометричні методи базуються на типових задачах комп'ютерного зору, таких як виявлення країв, реєстрація зображень та оцінка гомографії.

Зазвичай геометричні методи поділяють на дві основні категорії: прямий і на основі функцій методи.

Метод на основі ознак – це метод, який використовує ключові ознаки на зображеннях для оцінки переміщення камери. Цей метод ґрунтується на аналізі та відстеженні певних важливих елементів сцени, таких як кути, грані або текстуровані області, які є добре видимими та стабільними на різних зображеннях.

Метод на основі ознак працює шляхом виділення характерних точок на зображеннях за допомогою таких алгоритмів, як SIFT, ORB або SURF. Ці точки є унікальними і стійкими до змін ракурсу або руху камери, що дозволяє їх відстежувати на кількох послідовних кадрах. Після того, як ті самі ознаки ідентифіковані на нових кадрах, можна обчислити зміни їх положення відносно камери. На основі цих змін розраховується переміщення камери в просторі. Якщо додатково доступна інформація про масштаб або дані з інерційних сенсорів, можна також реконструювати тривимірну карту середовища.

SIFT - Один з перших алгоритмів для виділення стійких ознак, який добре працює на зображеннях із значними змінами масштабу або освітлення.

Основні етапи цього методу включають виявлення ключових точок шляхом пошуку локальних максимумів і мінімумів на різних масштабах зображення за допомогою піраміди Гаусса, що дозволяє знайти точки, стійкі

до змін масштабу і орієнтації. Кожній ключовій точці присвоюється орієнтація на основі локальних градієнтів інтенсивності в її околі, що забезпечує інваріантність до обертів зображення. Опис кожної ключової точки будується на основі інтенсивності пікселів в її околі, утворюючи вектор, який характеризує локальні градієнти в різних напрямках. Після цього описи точок на двох зображеннях порівнюються для знаходження відповідностей, використовуючи метрики відстані, такі як евклідова відстань. Для фільтрації хибних відповідностей застосовуються методи найближчого сусіда і другого найближчого сусіда, що допомагає покращити точність визначення руху.

SURF - це алгоритм, що використовується для виявлення характерних точок на зображеннях. Ці точки є стабільними до змін масштабу, обертання та освітлення, що робить SURF незамінним інструментом для визначення руху камери та створення карт оточення.

Основні етапи включають виявлення ключових точок за допомогою фільтрів Хаара, які оцінюють контраст на різних масштабах зображення, що забезпечує швидкість обробки завдяки апаратно-оптимізованим операціям. Кожній знайденій ключовій точці присвоюється орієнтація на основі локальних градієнтів інтенсивності, що забезпечує інваріантність до обертів. Опис кожної ключової точки створюється на основі фільтрів Хаара і складається з бінарних значень, що характеризують локальні градієнти навколо точки. Після цього описи точок порівнюються для пошуку відповідностей, використовуючи методи пошуку найближчих сусідів. Для фільтрації хибних відповідностей застосовуються різні методи, що допомагають покращити точність і зменшити кількість помилкових збігів.

Прямі методи – це підхід, який використовує всю доступну інформацію зображення для оцінки руху камери, замість того, щоб обмежуватися вибором окремих ключових точок (як у методах на основі ознак). Прямі методи працюють безпосередньо з пікселями інтенсивності зображення, що дозволяє максимально використовувати всі доступні дані та досягати більшої точності у відстеженні руху.

Прямі методи працюють на основі аналізу інтенсивностей пікселів (градієнта яскравості) на зображеннях, замість того щоб виділяти ключові точки, як у методах на основі ознак. Ці методи оцінюють зміни між послідовними кадрами за допомогою оптимізації фотометричних помилок. Вони мінімізують різницю в інтенсивності пікселів між кадрами, виходячи з того, що при переміщенні камери зміни на зображеннях мають відображати цей рух. Зміни у значеннях інтенсивності пікселів використовуються для оцінки переміщення камери в просторі, що дозволяє отримати інформацію про її траєкторію.

Існують три основні види прямих методів: щільні, напівщільні та рідкі методи.

Щільні методи використовують інформацію від усіх пікселів на зображенні, що робить їх найбільш точними, але водночас вимогливими до обчислювальних ресурсів. Основні етапи включають обчислення градієнтів інтенсивності в кожному пікселі, що дозволяє відстежувати зміни інтенсивності та знаходити відповідності між пікселями в різних кадрах. Для визначення параметрів руху камери, які мінімізують різницю між інтенсивностями відповідних пікселів, застосовується нелінійна оптимізація, часто за допомогою алгоритму Sparse Bundle Adjustment.

Напівщільні методи використовують тільки пікселі, де інформація про інтенсивність добре визначена, такі як краї або текстуровані ділянки зображення, що дозволяє зменшити кількість обчислень, зберігаючи високу точність. Основні етапи включають відбір пікселів, які найбільше підходять для відстеження, на основі критеріїв, таких як величина градієнта або контраст. Потім застосовується оптимізація, аналогічна щільним методам, але з меншою кількістю пікселів, що дозволяє значно зменшити обчислювальні витрати.

Рідкі методи використовують окремі ділянки зображення, не виділяючи явних ключових точок, як у методах на основі ознак. Основні етапи включають виділення характерних точок, таких як кути та краї, на зображенні,

відстеження цих ознак між різними кадрами, а також оптимізацію, де застосовуються геометричні обмеження для оцінки параметрів руху камери.

Гібридні методи – це методи візуальної навігації, що об'єднують підходи на основі ознак та прямі методи для досягнення кращої точності та надійності при відстеженні руху камери. Гібридні методи використовують переваги обох підходів: стійкість до змін ракурсу та освітлення від методів на основі ознак та точність від прямих методів. Це дозволяє забезпечити надійне відстеження навіть у складних умовах.

LSD-SLAM є гібридним методом, який комбінує прямі методи з відбором точок для відстеження, використовуючи щільну інформацію про інтенсивність зображень для створення карти та оцінки переміщення, але виділяє лише важливі пікселі, щоб уникнути надлишкових обчислень. Основні етапи включають напівщільне відстеження, де вибираються пікселі з високим градієнтом інтенсивності, що дозволяє знизити обчислювальні витрати, фотометричну оптимізацію для кращого відстеження дрібних змін у положенні камери, а також побудову карти, яка створюється поступово на основі зображень і інформації про переміщення, зберігаючи просторові зв’язки між об’єктами для повної тривимірної реконструкції.

Методи на основі навчання у візуальній одометрії використовують машинне навчання та глибокі нейронні мережі для оцінки руху камери, що дозволяє автоматично витягувати складні характеристики із зображень. На відміну від традиційних методів, де застосовуються чітко визначені математичні моделі, методи на основі навчання спираються на великі обсяги даних для навчання моделей, що здатні робити точні прогнози про рух.

Методи на основі навчання у візуальній одометрії спираються на великі набори даних з позначками для навчання моделей. На першому етапі, збір і підготовка даних, потрібні великі набори зображень з мітками про переміщення камери, які можуть бути як синтетичними, так і реальними. На другому етапі, навчання моделі, використовуються різні типи нейронних мереж, такі як CNN або RNN, для вивчення зв'язку між послідовними

зображеннями та відповідним переміщенням камери. Модель навчається прогнозувати зміни ракурсу або положення камери на основі вхідних зображень. На заключному етапі, оцінка руху, натренована модель може обробляти нові зображення та прогнозувати, як камера рухалася між кадрами.

Існують три основні види методів на основі навчання, які застосовуються для оцінки руху камери: методи із супроводжуваним навчанням (supervised learning), безсупроводжуваним навчанням (unsupervised learning) та навчанням з підкріпленням (reinforcement learning).

Supervised Learning: Модель навчається на позначених даних, де кожному набору зображень відповідають відомі зміщення та обертання камери. Цей підхід вимагає великих, точно позначених наборів даних для ефективного навчання.

Unsupervised Learning: Модель намагається навчитися оцінювати рух камери без явних міток. Використовуються методи відновлення геометрії сцени та мінімізації фотометричних помилок для навчання. Це дозволяє навчати моделі на необмежених наборах даних, не потребуючи дорогого маркування.

Reinforcement Learning: Модель навчається через спроби й помилки, отримуючи винагороду за правильне передбачення руху. Використовується в складніших середовищах, таких як робототехніка [5].

1.1.5 SLAM

Візуально-орієнтовані навігаційні системи для БПЛА включають дві основні перспективи: методи локалізації на основі карт і виявлення та уникання об'єктів. Методи на основі карт поділяються на незалежні від карт, залежні від карт та побудова карт. Для виявлення об'єктів використовуються різні методи, такі як оптичний потік і SLAM. Методи виявлення об'єктів включають різні підходи, такі як оптичний потік і SLAM, а для уникання об'єктів використовуються як глобальне, так і локальне планування маршрутів.

SLAM – Проблема одночасної локалізації та картографування. Це обчислювальна проблема побудови або оновлення карти невідомого середовища з одночасним відстеженням. На теоретичному та концептуальному рівні SLAM можна вважати вирішеною проблемою. Однак залишаються суттєві проблеми в практичній реалізації більш загальних рішень SLAM.

За останні два десятиліття SLAM стала гарячою областю досліджень у робототехніці та комп'ютерному зорі. Це фундаментальний модуль для багатьох додатків, таких як мобільна робототехніка, MAV, автономне водіння та доповнена реальність, оскільки всі вони потребують локалізації в реальному часі. Методи SLAM використовують інформацію, отриману датчиком, для створення карти невідомого середовища та локалізації датчика на карті. Візуальний SLAM широко вивчався через низьку ціну камери та її здатність отримувати повну інформацію про навколишнє середовище.

Більшість систем Visual SLAM, які зараз використовуються, покладаються на створені вручну візуальні функції. Наприклад, ORB-SLAM виконує зіставлення ознак і виявлення замикання циклу, витягуючи функції ORB у кожному кадрі зображення, а потім оцінює позицію камери та створює карту середовища. Однак створені вручну функції можуть не забезпечувати послідовне виявлення функцій і точні результати відповідності в складних середовищах.

Попри те, що візуальні функції відіграють важливу роль у багатьох сучасних системах SLAM, вони можуть стикатися з обмеженнями в складних середовищах. Зокрема, проблеми виникають через необхідність одночасної оцінки місцезнаходження робота та орієнтирів. Як показано на рисунку 1.3, справжнє місцезнаходження робота ніколи не буває безпосередньо виміряним, і для оцінки використовується спостереження між справжніми положеннями робота та орієнтирів.

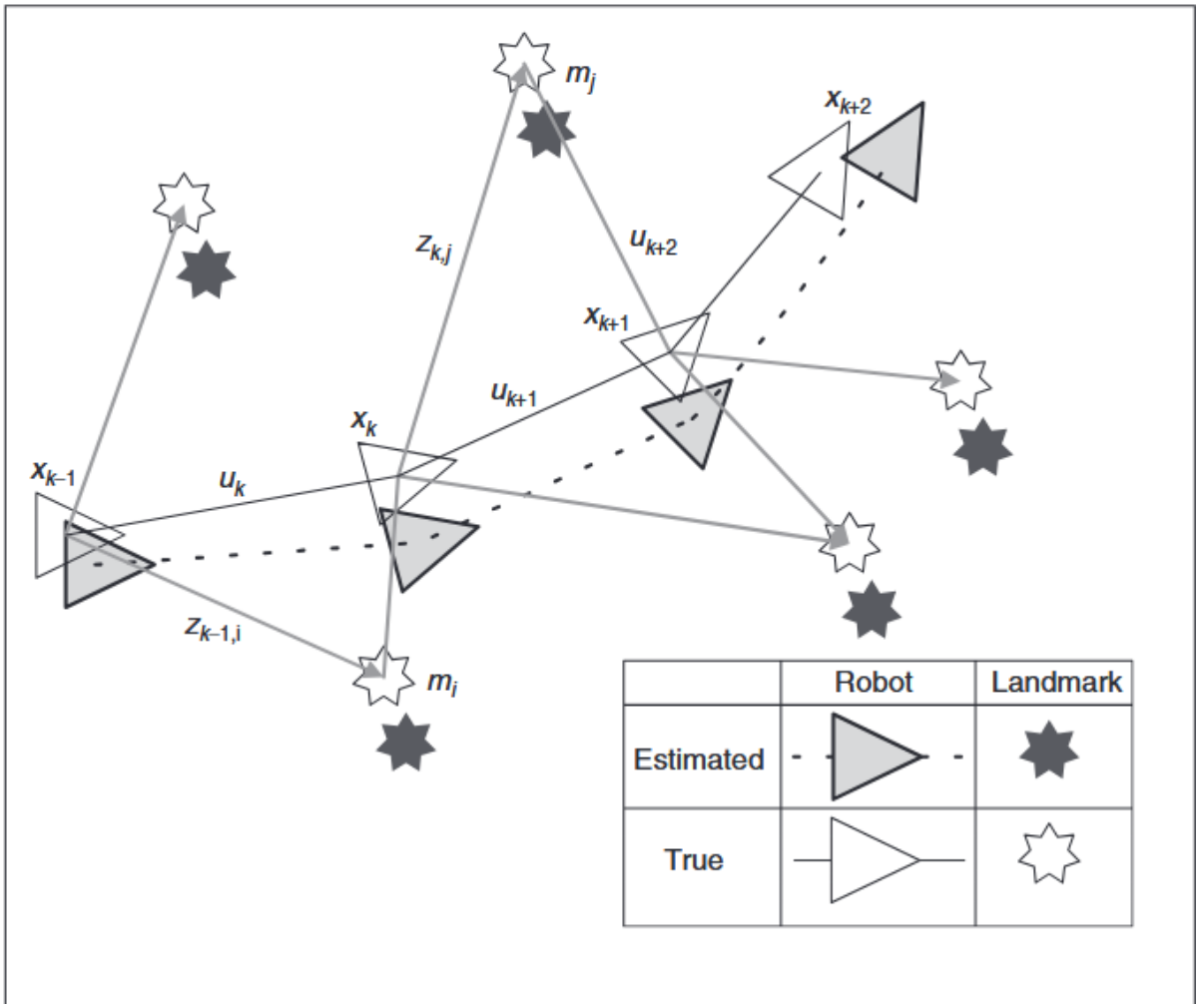

Рисунок 1.3 – Основна проблема SLAM [6]

Потрібна одночасна оцінка місцезнаходження робота та орієнтирів. Справжнє місцезнаходження ніколи не буває відомим або виміряним безпосередньо. Спостереження проводяться між справжніми положеннями робота та орієнтирів.

Робочий процес сучасного візуального алгоритму SLAM на основі ключових кадрів, ми коротко представимо його робочий процес, який складається з трьох основних модулів:

– Одометрія: це базовий модуль алгоритму SLAM. Його основна функція полягає в обробці останнього отриманого зображення методами на основі ознак або прямими методами, знаходячи відповідність між поточним зображенням і еталонним зображенням;

– Бек-енд: цей модуль підтримує глобальну наполегливу карту, виконуючи коригування пакетів для більшості сучасних візуальних алгоритмів SLAM. З одного боку, використовується стратегія ковзного вікна, яка зберігає фіксовану кількість ключових кадрів шляхом маргіналізації старого кадру для контролю вартості BA в реальному часі. З іншого боку, будується розріджена карта для оптимізації руху і структури для отримання більш точних результатів, оскільки спільна оптимізація з рухом і щільною картою не може працювати в режимі реального часу;
– Закриття петлі: цей модуль усуває накопичену помилку, спричинену великомасштабним і тривалим оцінюванням. Для цього виконується процедура виявлення петлі для виявлення потенційної петлі. Після виявлення петлі легка оптимізація графа пози співвідноситься з траєкторією, що значно підвищує точність алгоритму SLAM. Слід зазначити, що точність виявлення петель є критично важливою і повинна бути забезпечена. В іншому випадку, неправильне виявлення може безпосередньо призвести до збою алгоритму (рис.1.4) [7].

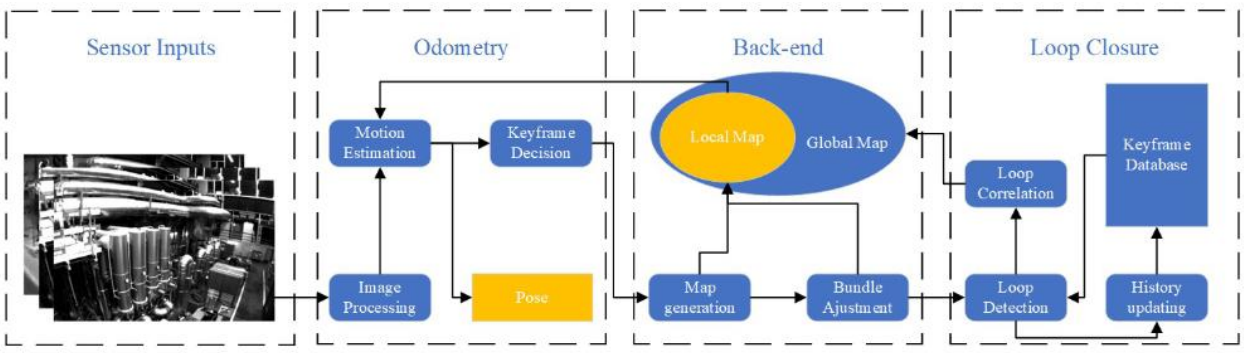

Рисунок 1.4 – Загальний конвеєр алгоритму візуального SLAM на основі ключових кадрів [7]

Постановка задачі: Агент створює зображення I_i. Ці зображення зроблені за допомогою жорсткого кріплення камери в певний час i. Після n кроків ми

маємо послідовність $I_{0:n} = \{I_0, I_1, ..., I_k, ..., I_n\}$ зроблені на камеру $C_{0:n} = \{C_0, C_1, ..., C_k, ..., C_n\}$. Два послідовних положення камери C_k і C_{k+1} з відповідними зображеннями I_k і I_{k+1} пов'язані через тверде тіло перетворення $T_{k+1,k} \in R^{4\times4}$.

$$T_{k+1,k} = \begin{pmatrix} R_{k+1,k} & t_{k+1,k} \\ 0 & 1 \end{pmatrix} \quad (1.1)$$

Тут $R_{k+1,k} \in SO(3)$ є матрицею обертання і $t_{k+1,k} \in \mathbf{R}^3$ є вектором переміщення відповідно. Трансформація $T_{k+1,k}$ описує відносний рух між позиціями C_k і C_{k+1} і є трансформацією, яка вирівнює зображення I_k із зображенням I_{k+1}.

З математики твердого тіла відомо, що

$$\begin{pmatrix} C_{k+1} \\ 1 \end{pmatrix} = T_{k+1,k} \begin{pmatrix} C_k \\ 1 \end{pmatrix} \quad (1.2)$$

Якщо ми тепер помножимо обидві частини рівняння вище зліва на $T_{k+2,k+1}$

$$T_{k+2,k+1} \begin{pmatrix} C_{k+1} \\ 1 \end{pmatrix} = T_{k+2,k+1} \left(T_{k+1,k} \begin{pmatrix} C_k \\ 1 \end{pmatrix}\right) \quad (1.3)$$

Потім використаємо рівняння (1.2) і (1.3),

$$\Rightarrow \begin{pmatrix} C_{k+2} \\ 1 \end{pmatrix} = T_{k+2,k+1} \begin{pmatrix} C_{k+1} \\ 1 \end{pmatrix} \quad (1.4)$$

З рівнянь (1.2) – (1.4) можна вивести, що якщо набір усіх наступних перетворення $T0{:}n = \{Tn,n-1, ..., T1,0\}$ відомо, ми можемо відновити всю траєкторію. Це мета VO, оцінити всі наступні перетворення. Цей процес можна виконувати багаторазово [6].

Переваги SLAM включають використання візуальних датчиків, які є дешевими та малопотужними, що важливо для БПЛА з нестабільним керуванням, обмеженою вантажопідйомністю та низьким енергоспоживанням. Такі датчики допомагають знизити вагу і енергоспоживання. Крім того, візуальні датчики здатні знімати зображення з високою частотою кадрів, що дозволяє алгоритмам точніше визначати локалізацію та забезпечувати необхідну точність керування для БПЛА. Вони також здатні захоплювати багату текстурну інформацію, що покращує

розуміння навколишнього середовища. Візуально-орієнтовані системи можуть забезпечувати високу точність навігації завдяки використанню камер та інших сенсорів для аналізу оточуючого середовища. Однак ці системи також можуть бути чутливими до змін освітлення і можуть вимагати значних обчислювальних ресурсів для обробки зображень в реальному часі [2].

1.1.6 Узагальнення технологій

Візуальні навігаційні системи базуються на використанні камер та алгоритмів комп'ютерного зору, таких як SLAM, що дозволяє БПЛА одночасно будувати карту навколишнього середовища та визначати своє положення в ньому. Це особливо важливо в умовах, де GNSS сигнал не доступний, наприклад, в приміщеннях або міських каньйонах.

Основними викликами є точність та адаптивність, оскільки ці системи можуть бути точнішими в умовах з низькою точністю GNSS, але вимагають високих обчислювальних ресурсів для обробки відео в реальному часі, що може бути проблемою для малогабаритних апаратів з обмеженою потужністю. Крім того, складність та вартість таких систем зумовлені потребою в високоякісних камерах та потужних обчислювальних ресурсах, хоча завдяки розвитку технологій і зниженню вартості апаратних засобів ці системи стають більш доступними.

Оцінка застосовності технологій для навігації також повинна враховувати ступінь узагальнення. Ідеально було б використовувати одні й ті ж апаратні засоби та алгоритми для всіх завдань навігації, проте різні проблеми вимагають різних характеристик, таких як розмір, вага, вартість, точність та умови експлуатації. Незважаючи на те, що датчики стають меншими і точнішими, інтеграція даних з різних сенсорів може бути складною через варіації в шумі та синхронізації.

Крім того, технології повинні бути адаптовані до специфічних умов експлуатації БПЛА. Це може включати адаптацію алгоритмів для різних типів

середовища або специфічних завдань, таких як пошуково-рятувальні операції або комерційні доставки [2].

Сучасні тенденції в розвитку навігаційних технологій для малогабаритних БПЛА включають інтеграцію методів машинного навчання, зокрема глибинного навчання, що дозволяє покращити якість обробки зображень та точність навігації.

Тенденції у навігації для БПЛА включають застосування глибинного навчання для покращення обробки візуальних даних, що дозволяє точніше визначати положення БПЛА у просторі завдяки моделям, що навчаються на великих наборах даних і адаптуються до нових середовищ. Крім того, зростаючий інтерес до автономних БПЛА стимулює розробку інтегрованих систем, здатних працювати без втручання оператора, що включає алгоритми для самостійного планування маршрутів і ухилення від перешкод.

Основні ризики для України та потенційні надзвичайні ситуації: Аналіз небезпек і загроз техногенного й природного характеру та виникнення НС в Україні свідчить про відносно стабільний стан техногенної та природної безпеки, за останні 10 років тенденцій до збільшення чи зменшення НС не спостерігається. Разом із тим рівні ризиків виникнення НС та збитків від них залишаються досить високими для більшості регіонів України.

Нині Україна є найбільш критичним регіоном Європи з техногенного навантаження, що у п’ять разів перевищує середньоєвропейський рівень і це не враховуючи військові дії. Ризик виникнення в Україні НС техногенного характеру є досить високим, що насамперед пов’язано з високим рівнем техногенного навантаження на регіони, наявністю комплексу енергетичних, хімічних, гірничодобувних об’єктів, значною кількістю промислово-міських агломерацій і високою щільністю населення у промислово розвинутих регіонах держави [8].

Ризики виникнення НС природного і техногенного характеру є фактором, що визначає якість життя населення будь-якої країни. На сьогодні масштаби НС та їх наслідки змушують розглядати їх як загрозу безпеці

суспільству, довкілля, розвитку економіки країни та національній безпеці загалом.

Високі ризики виникнення НС в Україні, необхідність збереження здоров'я й життя населення зумовлюють необхідність пошуку найбільш ефективних шляхів поліпшення роботи щодо попередження, виявлення, локалізації НС і ліквідації їх наслідків, в першу чергу за рахунок використання новітніх технологій, комплексного застосування сил і засобів, застосування методів, спрямованих на попередження, виявлення й локалізацію НС на ранніх стадіях їх виникнення й поширення.

Треба ще враховувати, що антропогенні зміни природного середовища відбуваються на два – три порядки швидше, ніж природні, і встежити за ними вже неможливо. У наш час ефективно вирішити настільки складну задачу можна лише єдиним способом: регулярною зйомкою земної поверхні з літаків і супутників, тобто аерокосмічним методом моніторингу.

Сучасна стратегія протидії природним небезпекам і техногенним катастрофам полягає в розвитку системи моніторингу і прогнозування НС, раціональному розміщенні продуктивних сил і розселенні населення з урахуванням можливих небезпек, будівництві будинків та споруджень підвищеної міцності, удосконаленні систем оповіщення й інформування населення про небезпеки тощо.

Завдання щодо зменшення ризику виникнення НС має три етапи:

- попередження можливої НС;
- моніторинг (розвідка) НС, що сталася;
- ліквідація НС.

Відповідно до цього моніторинг НС також має три складові, зокрема:

- моніторинг потенційно небезпечних об'єктів і територій;
- моніторинг стану НС;
- контроль її розвитку і ліквідації [8].

Розробка та вдосконалення навігаційних систем для БПЛА відкривають нові можливості в таких галузях, як логістика, сільське господарство, рятувальні операції та інспекція інфраструктури. Використання візуальних систем дозволяє вирішувати завдання в умовах складних або змінних середовищ, забезпечуючи автономну навігацію та аналіз об'єктів на місцевості.

1.1.7 Військове використання

Війна в Україні розкрила потенціал дронів у військовій сфері. Ефективне використання дронів привело до того, що в українській армії був створений новий тип військ.

Українські військові використовують дрони для розвідки, виявлення цілей і точкових ударів по ворогу. Різноманітність моделей, від комерційних безпілотників до потужних бойових дронів, дозволяє ефективно застосовувати їх в умовах бойових дій. Наприклад, дрони Bayraktar TB2 на початкових етапах конфлікту завдавали значної шкоди російським силам, зокрема завдяки своїй здатності нести ракети та залишатися в повітрі тривалий час. Але після покращення радіоелектронної боротьби ворога акцент був переміщений на менші дрони наприклад FPV.

Ця війна продемонстрував бойові переваги безпілотників, які стали меншими, смертоноснішими, легшими в експлуатації та доступними майже кожному. Вони стискають так званий ланцюг убивств, скорочуючи час від моменту виявлення цілі до моменту її знищення, і вони можуть посилити здатність військових розвідувати передній край поля бою. Безпілотники з довшими профілями витривалості можуть ефективно проводити години розвідки, дозволяючи іншим, більш досконалим безпілотникам виконувати точні удари глибоко всередині ворожої території. Інші моделі дозволяють окремим солдатам стежити за рухом супротивника, не ризикуючи життям і не поступаючись позицією солдата.

Безпілотники також можуть відігравати важливу міжнародну гуманітарну роль, наприклад, проводячи оцінку бойових і супутніх збитків або викриваючи військові злочини. Американський виробник безпілотників Skydio нещодавно подарував дев’ять безпілотників , які з камерами високої роздільної здатності будуть використані, щоб допомогти Україні документувати потенційні військові злочини Росії. Через Агентство США з міжнародного розвитку зроблені зображення будуть використані для допомоги Генеральній прокуратурі в документуванні багатьох випадків порушення прав людини [9].

Основні аспекти використання дронів включають розвідку та спостереження, де дрони дозволяють збирати розвідувальну інформацію в реальному часі, надаючи точні дані про розташування і рухи противника без необхідності залучати пілотовану авіацію. Вони також здатні завдавати точкових ударів по важливих цілях, зокрема бойові дрони-камікадзе або озброєні дрони, що мінімізує втрати серед цивільного населення та знижує ризики для військових.

Сучасні дрони можуть діяти автономно, виконуючи завдання без постійного контролю оператора, що є особливо важливим в умовах, коли зв'язок може бути ускладнений або недоступний. Проте використання дронів супроводжується новими викликами, такими як загроза від засобів радіоелектронної боротьби, що можуть порушити їх роботу, а також етичні питання, пов'язані з автономними ударами.

Загалом, дрони продовжують удосконалюватися, стаючи меншими, дешевшими і більш смертоносними. Вони перетворюються на ключовий елемент сучасних військових дій, і боротьба з ними стає все складнішою. Досвід війни в Україні демонструє, що дрони можуть значно змінити тактику ведення війни, зокрема шляхом швидкого реагування та точкових ударів по цілях.

Ця війна також показала, що ефективність дронів залежить від здатності адаптуватися до змінних умов на полі бою та використовувати доступні технології для досягнення переваги над ворогом.

1.2 Моделі і методи інтелектуального аналізу зображень та відео

У дослідженні візуального спостереження для навігації малогабаритних БПЛА використовуються різноманітні методи та алгоритми, які дозволяють ефективно аналізувати відео дані для виявлення аномальних подій та поведінки об'єктів. Основними напрямками є методи на основі ознак, трекінгу, просторово-часових підходів і методи глибокого навчання.

Методи на основі ознак фокусуються на виявленні подій, виходячи з домінуючих поведінкових патернів, де аномальні події визначаються через менш домінуючі поведінки. Наприклад, система, реалізована в Matlab 2014, використовує моделі Gaussian Mixture Model для моделювання фону і додаткові зовнішні правила для розпізнавання об'єктів. Основні етапи включають попередню обробку, витягування ознак, трекінг об'єктів і розуміння поведінки [10]. Однак, такі системи можуть мати обмеження в ситуаціях з перекриттям об'єктів або у випадках сумнівних активностей.

Трекінгові методи, що використовують виявлення та кореляції, також грають важливу роль у відеоаналізі. Методи, засновані на виявленні, таких як мультифазний каскад, забезпечують ідентифікацію об'єктів у кожному кадрі та їх групування в послідовності . У той же час, методи кореляційного трекінгу забезпечують швидкість завдяки використанню FFT для обробки відео. Проте ці підходи можуть мати труднощі з відстеженням об'єктів при складних умовах освітлення або при перекриттях [11]. Трекінг є критичним аспектом для безперервного відстеження об'єктів на відео. Використання методів, таких як Kalman Filter та Mean-Shift, дозволяє визначати місцезнаходження об'єктів у кожному кадрі відео, забезпечуючи плавний рух треку.

Kalman Filter використовується для прогнозування положення об'єкта на основі попередніх даних, забезпечуючи більш точний трекінг у реальному часі.

Mean-Shift використовується для відстеження об'єктів за кольором та яскравістю, що робить його ефективним для трекінгу в складних умовах.

Обробка зображень є основою для будь-якого аналізу відео та зображень. Цей процес включає попередню обробку, яка допомагає покращити якість зображень, видалити шум і виділити важливі елементи. Екстракція ознак здійснюється за допомогою алгоритмів, таких як гістограма орієнтованих градієнтів, що використовується для виявлення об'єктів. Ці методи є основою для побудови більш складних алгоритмів комп'ютерного зору.

Просторово-часові методи, такі як методи на основі гістограм орієнтованих градієнтів або оптичного потоку, забезпечують комбінацію просторової і рухової інформації. Метод Bag of Visual Words є одним з класичних підходів у цьому напрямку, що дозволяє обробляти відео, фокусуючи увагу на важливих частинах зображення і обробляючи ці частини незалежно [11]. Глибоке навчання також відіграє важливу роль у розпізнаванні дій, завдяки використанню нейронних мереж, таких як CNN, RNN або «трансформери» [10][11]. Зокрема, новітні трансформери, як Visual Transformers, дозволяють ефективніше обробляти відео, паралелізуючи операції та поліпшуючи захоплення глобальних зв’язків між кадрами [11].

Сучасні методи глибинного навчання включають використання CNN для обробки зображень, таких як архітектури AlexNet і VGG, які ефективно застосовуються для виявлення та класифікації об'єктів на кадрах відео.

CNNs є найпоширенішим типом нейронних мереж для обробки зображень і відео. Їхня структура спеціально розроблена для ефективного розпізнавання просторових і патернових зв'язків у зображеннях, що робить їх основним інструментом в таких завданнях, як класифікація зображень, сегментація, детекція об’єктів та відеоаналіз.

Основні принципи роботи CNN полягають у кількох ключових етапах. По-перше, CNN використовують операцію згортки, яка є основною особливістю цієї архітектури. Замість того, щоб кожен нейрон обробляв всі пікселі зображення, згортковий шар використовує невеликі фільтри або ядра згортки, які проходять через зображення та витягують локальні ознаки, такі як краї, кути або текстури. Це дозволяє мережі формувати складні уявлення про об'єкти через кілька шарів.

По-друге, під час навчання CNN автоматично навчаються оптимальних значень для своїх фільтрів, що дозволяє їм ефективно розпізнавати ознаки на різних рівнях складності. Перші шари мережі витягують прості ознаки, такі як краї і текстури, тоді як наступні шари фокусуються на більш складних патернах, наприклад, частинах об'єктів або цілих об'єктах.

Після кожної операції згортки зазвичай застосовується операція пулінгу, яка зменшує розмір просторового представлення. Це допомагає знизити обчислювальну складність і робить мережу більш стійкою до зсувів і незначних змін в зображенні. Найпоширеніший тип пулінгу — це Max Pooling, де вибирається максимальне значення з певної області, що дозволяє зберігати найважливішу інформацію.

Після кожної згортки результат проходить через нелінійну функцію активації, таку як Rectified Linear, що допомагає мережі моделювати складні взаємозв'язки між вхідними даними і результатами. Цей процес додає нелінійність і дозволяє мережі виявляти складніші патерни в зображеннях.

Останні шари CNN зазвичай складаються з повністю зв'язних шарів, які поєднують всі витягнуті ознаки з попередніх шарів і передають їх на класифікацію. Ці шари використовуються для прийняття остаточного рішення про те, до якого класу належить об'єкт на зображенні.

RNNs відрізняються від традиційних нейронних мереж тим, що вони мають зворотні зв’язки між шарами, що дозволяє їм зберігати стан на кожному кроці часу. Це робить їх придатними для обробки послідовних даних, таких як

відео, текст чи часоряди. У випадку відеоаналізу це допомагає моделювати залежності між послідовними кадрами.

У традиційній RNN на кожному кроці часу вихід не лише залежить від поточного входу, але й від попереднього стану. Проте, звичайні RNN можуть мати труднощі з моделюванням довготривалих залежностей через проблему зникнення градієнтів.

RNN призначені для обробки послідовних даних, де кожен елемент, як-от кадр у відео, залежить від попереднього. Це дозволяє моделі вивчати залежності у часі завдяки зворотним зв'язкам, які дозволяють враховувати інформацію про попередні стани на кожному кроці. RNN можуть зберігати важливу інформацію на довготривалих часових відрізках, а для цього використовуються більш розвинуті варіанти, такі як LSTM або GRU, які краще управляють інформацією завдяки механізмам пам'яті та вентилям.

Для відеоаналізу RNN часто комбінують з CNN. Спочатку CNN витягують просторові ознаки з кожного кадру відео, як-от об’єкти або текстури, а потім RNN аналізує ці ознаки, щоб вивчити часові залежності. Головна перевага RNN полягає у здатності враховувати залежності між кадрами відео, що важливо для розпізнавання дій і поведінки об’єктів з часом.

Стандартні RNN можуть мати труднощі з обробкою довгих послідовностей через проблему "зникнення градієнтів". Для вирішення цієї проблеми використовують LSTM і GRU, які краще зберігають важливу інформацію на тривалих відрізках часу завдяки своїм специфічним структурам. RNN навчаються обробляти відео, враховуючи часовий контекст, де кожен елемент послідовності впливає на ваги моделі, що забезпечує здатність передбачати наступні стани на основі минулих.

RNN забезпечують ефективне прогнозування та класифікацію подій у відео, таких як розпізнавання дій і прогнозування руху. У сучасних моделях для відеоаналізу часто використовують механізм уваги, який дозволяє фокусуватися на ключових моментах у послідовності кадрів, покращуючи точність аналізу довгих відеопослідовностей.

Нейронні мережі на основі капсул – це новий підхід до обробки зображень, запропонований Джеффрі Гінтоном у 2017 році як альтернатива традиційним CNN. Вони здатні краще вловлювати просторові взаємозв’язки між об'єктами на зображеннях та мають значний потенціал для підвищення точності і надійності розпізнавання, зокрема для систем малогабаритних БПЛА, які обмежені в ресурсах.

Принципи роботи Capsule Networks полягає у наступному: Капсули виступають як базові елементи мережі, являючи собою невеликі групи нейронів, які здатні не лише кодувати ймовірність присутності об'єкта на зображенні, але й його властивості, такі як орієнтація, масштаб та кут нахилу. Це дозволяє капсулам зберігати більше контексту про об'єкт та його просторове положення. В Capsule Networks використовується механізм динамічної маршрутизації, де кожна капсула визначає, до яких капсул у наступному шарі передавати інформацію на основі подібності між векторами вихідних сигналів. Цей підхід допомагає мережі краще відстежувати складні просторові зв'язки між об'єктами та уникати типових помилок згорткових мереж, таких як невірне розпізнавання зміщених об'єктів. Capsule Networks також виявляють високу стійкість до деформацій і трансформацій, таких як повороти, зміна масштабу або перспективи, що допомагає покращити точність при значних змінах у вигляді об'єктів.

Важливою перевагою є менша потреба в даних та обчислювальних ресурсах, оскільки ці мережі краще кодують просторові відносини, що є корисним для БПЛА, де доступ до великих навчальних даних може бути обмеженим, а потужності для обробки – недостатніми.

Методи обробки на основі подій є новим підходом до комп'ютерного зору, який відрізняється від традиційних методів, орієнтованих на обробку кадрів. Цей підхід зосереджений на аналізі змін у сцені, представлених як події, що дозволяє знижувати затримки та споживання ресурсів.

Методи обробки на основі подій використовують принципи, що відрізняють їх від традиційних камер. Подієві сенсори реєструють лише зміни

в сцені, замість того, щоб захоплювати повні кадри з фіксованою частотою. Кожен піксель у подієвій камері має здатність незалежно виявляти зміни яскравості та передавати ці зміни у вигляді подій, які містять координати пікселя, час зміни і знак зміни (зростання чи зменшення яскравості). Оскільки подієві камери фіксують зміни без фіксованої частоти кадрів, вони забезпечують дуже низьку затримку, що робить їх ідеальними для застосувань, де важлива швидкість реакції, таких як автономні системи чи реальний час. Крім того, подієві камери мають високий динамічний діапазон, що дозволяє їм ефективно працювати в умовах різкого контрасту освітлення, де традиційні камери можуть мати труднощі.

Зменшення залежності від великих наборів даних: Використання методів reinforcement learning або transfer learning може значно зменшити необхідність у великих наборах даних для навчання моделей. Це дозволяє розвивати системи, які можуть ефективно працювати в реальному часі з обмеженою кількістю навчальних даних.

Аналіз аномалій у відео є критично важливим аспектом для систем візуальної одометрії БПЛА, особливо в контексті моніторингу та безпеки. Виявлення аномальних подій в такому контексті може значно підвищити ефективність використання БПЛА, дозволяючи вчасно реагувати на небезпечні або непередбачені ситуації.

Аномалії в відео можуть бути визначені шляхом аналізу поведінки об'єктів і їхніх траєкторій. Це включає в себе виявлення неочікуваних змін у русі об'єктів, їхніх взаємодіях, або навіть у самій сцені. Наприклад, раптові зміни швидкості або напрямку об'єктів, поява нових об'єктів в зоні моніторингу, або аномалії в шаблонах поведінки можуть бути ознаками значущих подій, що потребують подальшого аналізу або втручання.

Для досягнення цієї мети використовуються різні підходи та алгоритми обробки відео. Один з них включає використання методів машинного навчання для моделювання нормального поведінкового профілю об'єктів і виявлення відхилень від цього профілю. Інший підхід передбачає

застосування алгоритмів візуальної одометрії для відстеження змін у траєкторіях об'єктів та порівняння їх з очікуваними або попередніми даними. Також можуть використовуватися техніки аналізу змін у зображеннях, такі як детекція країв або аналіз векторів руху, щоб виявити аномальні ситуації.

Ці алгоритми повинні бути здатні працювати в реальному часі, адже в ситуаціях, що потребують негайного реагування, затримки в обробці можуть бути критичними. Отже, важливо забезпечити ефективність та точність системи виявлення аномалій, щоб вона могла швидко і надійно реагувати на потенційні загрози або зміни в середовищі.

Завдання виявлення аномалій в відео в рамках візуальної одометрії також включає розробку і оптимізацію алгоритмів, що дозволяють зменшити кількість хибних спрацьовувань та покращити загальну точність системи. Це може включати в себе інтеграцію різних джерел даних, таких як сенсори і навігаційні системи, для покращення контекстуального розуміння ситуації.

Таким чином, аналіз аномалій у відео є важливою частиною систем візуальної одометрії для БПЛА, яка дозволяє забезпечити високу ступінь надійності і ефективності в управлінні і моніторингу об'єктів у реальному часі.

Сучасні методи глибокого навчання все ще стикаються з певними проблемами, такими як висока обчислювальна складність і потреба в великих наборах даних. Це створює виклики для розробки практичних систем, які могли б працювати в реальному часі і за різних умов навколишнього середовища. Разом з тим, поєднання різних підходів, таких як трекінг, аналіз просторово-часових ознак і глибоке навчання, може забезпечити більш комплексне рішення для задач візуального спостереження в умовах навігації малогабаритних БПЛА.

1.3 Формалізована постановка задачі

Основна задача полягає в оцінці положення та орієнтації малогабаритного безпілотного апарату P(x,y,z) та R(ϕ,θ,ψ) у просторі на основі аналізу послідовності візуальних даних.

Ця проблема може бути формалізована через систему рівнянь руху:

$$P_{t+1} = P_t + \Delta P \tag{1.5}$$

де ΔP — зміщення між кадрами, обчислене на основі аналізу оптичного потоку.

Орієнтація безпілотного апарату визначається за допомогою трансформаційної матриці:

$$R_{t+1} = R_t + R_\Delta \tag{1.6}$$

де R_Δ отримується через розв’язання задачі Perspective-n-Point з використанням ключових точок.

Ключові компоненти розв’язання задачі:

1) Оптичний потік: визначення зміщень пікселів між кадрами;
2) Фільтрація викидів: видалення хибних відповідностей між кадрами;
3) Аналіз послідовності кадрів: визначення кумулятивного руху.

Основні завдання, які необхідно вирішити:

- Виділення ключових візуальних ознак із послідовних кадрів та їх відстеження у часі;
- Оцінка тривимірного переміщення БПЛА на основі двовимірних зображень;
- Забезпечення стабільного функціонування алгоритмів в умовах змінного освітлення та інших факторів навколишнього середовища;
- Інтеграція інформації з візуальних сенсорів з іншими сенсорними системами для надійного і безперервного позиціонування.

2. МОДЕЛІ І МЕТОДИ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ВІЗУАЛЬНОЇ НАВІГАЦІЇ БЕЗПІЛОТНОГО АПАРАТУ

2.1 Математична модель

Нехай I_1 і I_2 — два зображення, отримані з камери, що фіксують зміну сцени в часі. Метою є визначення векторів швидкості для кожного пікселя, що відображають його переміщення між двома кадрами.

Для кожного пікселя p=(x,y) на зображенні, припускається, що зміщення зображення між двома близькими кадрами є малим і постійним в межах локального вікна. Таким чином, для кожного пікселя виконується рівняння оптичного потоку:

$$I_x(p)V_x + I_y(p)V_y = -I_t(p) \tag{2.1}$$

де:

- $I_x(p)$, $I_y(p)$ – часткові похідні інтенсивності зображення по координатах x та y;
- V_x, V_y – компоненти вектора швидкості пікселя p;
- $I_t(p)$ – часткова похідна інтенсивності зображення по часу.

Набір рівнянь для n пікселів в локальному вікні можна записати у вигляді матричної системи:

$$A \cdot v = b \tag{2.2}$$

де:

- A – матриця розміру n×2, що містить компоненти похідних інтенсивності для кожного пікселя;
- v – вектор швидкості;
- b — вектор, що містить значення похідних інтенсивності по часу для кожного пікселя.

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \cdots & \cdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \qquad (2.3\text{-}2.4)$$

Оскільки система рівнянь $A \cdot v = b$ є перенавантаженою, для її розв'язку застосовується метод найменших квадратів. Це дає можливість знайти вектор швидкості v через вирішення рівняння:

$$V = (A^T A)^{-1} A^T b \qquad (2.5)$$

де:

- $A^T A$ – матриця «структурний тензор», що описує зміни інтенсивності пікселів в межах вікна;
- $(A^T A)^{-1}$ — обернена матриця до $A^T A$.

Після вирішення системи рівнянь, отримуються компоненти вектора швидкості V_x і V_y, що визначають оптичний потік для кожного пікселя зображення[12].

Узагальнена математична форма виглядає наступним чином:

$$v = (\sum_i I_x(q_i)^2 \sum \; I_x(q_i) I_y(q_i))^{-1} \begin{bmatrix} -\sum_i I_x(q_i) I_t(q_i) \\ -\sum_i I_y(q_i) I_t(q_i) \end{bmatrix} \qquad (2.6)$$

2.2 Піраміда зображень

Для покращення стійкості до змін масштабу та великих зсувів застосовується піраміда зображень. Цей метод передбачає побудову декількох рівнів зображення, де на кожному рівні зображення проходять обробку за допомогою фільтрації та зменшення розміру, що дозволяє враховувати різні масштаби та деталі.

Фільтрація зображення здійснюється за допомогою Гаусового фільтра для згладжування зображення і зменшення високочастотних компонент:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (2.7)$$

де σ — стандартне відхилення, що контролює ступінь згладжування [13].

Після цього зображення зменшують вдвічі для отримання нового рівня піраміди.

2.3 Фільтрація викидів

Для видалення хибних відповідностей між кадрами використовується метод RANSAC, який генерує модель руху на основі випадкової підмножини точок. Потім визначаються інлайери, тобто точки, які відповідають моделі, за певним порогом:

$$\left\| \Delta P_{predicted} - \Delta P_{computed} \right\| < threshold \quad (2.8)$$

де:

- $\Delta P_{predicted}$ – передбачене зміщення на основі моделі;
- $\Delta P_{computed}$ – обчислене зміщення виміряне за даними;
- threshold – поріг допустимої похибки.

Модель із найбільшою кількістю інлайерів приймається як оптимальна.

2.4 Кумулятивне обчислення руху

Кумулятивне обчислення руху етапом у задачах, пов'язаних з відстеженням і оцінкою руху об'єктів на основі візуальних даних, що дозволяє визначити, де і як рухався об'єкт за певний період часу на основі вимірювань, що надходять від датчиків або алгоритмів обробки зображень.

1. Кумулятивне обчислення позиції полягає в акумулюванні зміщення позицій об'єкта відносно початкової точки;

$$P_{t+1} = P_t + \sum_{i=1}^{t} \Delta P_i \quad (2.9)$$

де:

- P_t – позиція об'єкта на момент часу t;
- ΔP_i – зміщення між двома послідовними кадрами;
- P_{t+1} – позиція об'єкта на момент часу t+1.

2. Кумулятивне обчислення орієнтації полягає в поступовому оновленні орієнтації об'єкта в тривимірному просторі на основі змін, що відбуваються між кадрами.

$$R_{t+1} = R_t + \prod_{I=1}^{T} R_{\Delta_i} \quad (2.10)$$

де:

- R_t – орієнтація об'єкта на момент часу t;
- R_{Δ_i} – зміна орієнтації між двома послідовними кадрами;
- R_{t+1} – орієнтація об'єкта на момент часу t+1.

2.5 Критерії валідації якості навчених моделей візуальної навігації

Основні критерії, які використовуються для оцінки таких моделей, включають:

1. MAE оцінює середню абсолютну різницю між передбаченими значеннями та істинними значеннями. Для візуальної навігації це може бути використано для оцінки помилок у позиціонуванні або орієнтації.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - x_i| \quad (2.11)$$

де:

- N - кількість точок;

- x - прогнозоване значення;
- y - істинне значення.

2. MSE дає більший штраф за великі помилки. Вона є важливою метрикою, коли великі відхилення від істинних значень критичні для задачі.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i)^2 \quad (2.12)$$

де:

- N – кількість точок;
- x – прогнозоване значення;
- y – істинне значення.

3. RMSE використовується для оцінки точності моделей регресії, де важливим є вимірювання помилок між передбаченими значеннями та реальними значеннями. RMSE дозволяє зрозуміти, наскільки сильно відрізняються передбачення моделі від реальних результатів, при цьому враховуючи квадратичну помилку.

$$RMSE = \sqrt{\frac{1}{N}\ \sum_{i=1}^{N}(P_i^{true} - P_i^{pred})^2} \quad (2.13)$$

де:

- N – кількість точок;
- P_i^{true} – істинна позиція;
- P_i^{pred} – передбачене значення.

4. Рівень інлайерів (Inlier Ratio) — це метрика, яка оцінює ефективність фільтрації за допомогою методу RANSAC. Вона відображає кількість точок, які відповідають математичній моделі (інлайери), відносно загальної кількості точок, що були використані для оцінки руху чи трансформації.

$$Inlier\ Ratio = \frac{Number\ of\ Inliers}{Total\ Number\ of\ Correspondences} \quad (2.14)$$

де:

- Number of Inliers – кількість точок, що задовольняють умови точності моделі;
- Total Number of Correspondences – загальна кількість точок, які були використані для оцінки руху чи трансформації.

3. ПРОГРАМНА РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ВІЗУАЛЬНОЇ НАВІГАЦІЇ МАЛОГАБАРИТНОГО БЕЗПІЛОТНОГО АПАРАТУ

3.1 Формування навчальних та тестових даних

Для реалізації задачі візуальної одометрії використовується набір зображень, що відображають послідовність кадрів, отриманих в процесі руху безпілотного літального апарату. Ці дані є основою для аналізу просторового переміщення апарату за допомогою алгоритмів комп'ютерного зору. У наборі даних програми містить 211 кадрів у форматі .png, ці дані зберігаються у папці images. Кожен кадр відображає окремий момент у часі, що дозволяє відстежувати зміни положення апарату та визначати його траєкторію в просторі.

Для коректної роботи алгоритмів візуальної одометрії необхідно використовувати дані внутрішньої та зовнішньої калібрування камери, яка застосовується для зйомки. Ці дані зберігаються у файлі calib.txt (рис 3.1). Файл містить матриці калібрування камери, які визначають основні параметри, необхідні для перетворення двовимірних зображень у тривимірну просторову модель. А саме:

- Внутрішню матрицю: містить фокусну відстань, координати головної точки та інші параметри;
- Зовнішню матрицю: описує трансформацію між системою координат камери та світовою системою координат.

```
9.126492738800e+02 0.000000000000e+00 4.8932738527000e+02 0.000000000000e+00
0.000000000000e+00 9.142160340600e+02 3.695963400300e+02 0.000000000000e+00
0.000000000000e+00 0.000000000000e+00 1.000000000000e+00 0.000000000000e+00

7.070912000000e+02 0.000000000000e+00 6.01887300000e+02 -3.79814500000e+02
0.000000000000e+00 7.070912000000e+02 1.831104000000e+02 0.000000000000e+00
0.000000000000e+00 0.000000000000e+00 1.000000000000e+00 0.000000000000e+00
```

Рисунок 3.1 – Матриці калібрування камери

Для аналізу руху малогабаритного безпілотного апарату використовується файл imu.csv (рис 3.2), який містить дані, зібрані IMU. Цей модуль є важливим джерелом інформації для оцінки положення, швидкості та орієнтації апарату.

Файл має наступну структуру:

- Image – ідентифікатор кадру, пов'язаний з певним часом запису даних;
- Time – маркер часу у секундах для кожного вимірювання;
- Time difference – різниця часу між поточним та попереднім вимірюваннями;
- roll, pitch, yaw – кути орієнтації апарату у трьох вимірах;
- v_x, v_y, v_z – лінійні швидкості вздовж осей x, y та z відповідно;
- a_x, a_y, a_z – лінійні прискорення вздовж осей x, y та z;
- h – висота над рівнем землі.

Ці дані можна використовувати для доповнення візуальної інформації та підвищення точності алгоритмів навігації. Наприклад:

- Лінійні прискорення (ax, ay, az) дозволяють оцінити зміну швидкості та напрямку руху;

- Кути орієнтації (roll, pitch, yaw) допомагають визначити нахил апарату;
- Швидкості (vx, vy, vz) разом із даними часових інтервалів використовуються для розрахунку пройденої відстані.

1	Image,Time,Time difference,roll,pitch,yaw,vx,vy,vz,ax,ay,az,h
2	/rec_0096,0.09000229835510254,0.09000229835510254,0,0,0,0,0,0,-20.0,28.0,-1005.0,0
3	/rec_0097,0.18000483512878418,0.09000253677368164,0,0,0,0,0,0,5.0,26.0,-1007.0,0
4	/rec_0098,0.2700059413909912,0.09000110626220703,0,0,0,0,0,0,-12.0,19.0,-988.0,0
5	/rec_0099,0.36000728607177734,0.09000134468078613,0,0,0,0,0,0,-8.0,20.0,-1034.0,0
6	/rec_0100,0.4500124454498291,0.09000515937805176,0,0,0,0,0,0,-14.0,20.0,-1238.0,0
7	/rec_0101,0.5400142669677734,0.09000182151794434,0,0,0,0,0,-30,-24.0,13.0,-1344.0,0
8	/rec_0102,0.6300156116485596,0.09000134468078613,0,-1,0,0,0,-50,-13.0,16.0,-1187.0,0
9	/rec_0103,0.7200181484222412,0.09000253677368164,0,-1,0,0,0,-60,-10.0,13.0,-1082.0,0
10	/rec_0104,0.8100192546844482,0.09000110626220703,-1,-1,0,0,0,-70,-7.0,26.0,-1054.0,0
11	/rec_0105,0.9000205993652344,0.09000134468078613,-1,-1,0,0,0,-70,-7.0,26.0,-1054.0,0
12	/rec_0106,0.9900219440460205,0.09000134468078613,-1,0,0,0,0,-70,-30.0,21.0,-974.0,10
13	/rec_0107,1.0800299644470215,0.09000802040100098,-1,0,0,0,0,-60,-5.0,6.0,-947.0,10
14	/rec_0108,1.1700315475463867,0.09000158309936523,-1,0,0,0,0,-50,-9.0,6.0,-919.0,20
15	/rec_0109,1.2600336074829102,0.09000205993652344,0,0,0,0,0,-50,-5.0,-5.0,-884.0,20
16	/rec_0110,1.3500380516052246,0.09000444412231445,0,0,0,0,0,-40,-22.0,44.0,-897.0,20
17	/rec_0111,1.4400391578674316,0.09000110626220703,-1,-1,0,0,0,-30,-8.0,13.0,-879.0,30
18	/rec_0112,1.5300405025482178,0.09000134468078613,-1,-1,0,0,0,-20,-14.0,7.0,-918.0,30
19	/rec_0113,1.620042324066162,0.09000182151794434,-1,-1,0,0,0,-20,-14.0,7.0,-918.0,30
20	/rec_0114,1.7100434303283691,0.09000110626220703,-1,-1,0,0,0,-10,-10.0,3.0,-912.0,40

Рисунок 3.2 – Дані IMU

Для відображення ground truth малогабаритного безпілотного апарату використовується файл ground_truth.csv (). Дані реального положення використовуються для порівняння з результатами алгоритму навігації, що дозволяє оцінити точність та ефективність алгоритму.

Файл містить такі дані:

- Frame – номер кадру, з яким асоційовано значення координат;
- Time – час, коли було знято зображення у секундах;

– X, Y, Z — кілька колонок з координатами, що представляють кілька точок, кути обертання та якістьоцінки маркера на одному кадрі.

Рисунок 3.3 – Дані ground truth

3.2 Вибір засобів програмної реалізації

Для розробки алгоритмів візуальної одометрії часто використовуються мови програмування Python і C++. Python обраний через зручність для швидкої розробки, тестування та використання потужних бібліотек, таких як OpenCV і NumPy, які значно спрощують реалізацію алгоритмів. Python дозволяє легко працювати з великими обсягами даних, що важливо для візуальної одометрії, а також має велику кількість готових рішень, які можна адаптувати під конкретні задачі.

3.3 Короткий опис програмного забезпечення

Програма реалізує візуальну одометрію, яка оцінює траєкторію руху камери в просторі на основі аналізу послідовності зображень. Вона обробляє відеопотік, використовуючи метод Лукаса-Канаде для розрахунку оптичного потоку, що дає змогу відстежувати переміщення ключових точок між кадрами. Дані про обертання та трансляцію камери обчислюються шляхом аналізу

есенціальної матриці, яка формується з координат ключових точок. Для підвищення точності програма були інтегровані дані IMU. Результатом роботи є тривимірна траєкторія камери, яка дозволяє візуалізувати рух об'єкта у просторі.

Опис бібліотек використаних у програмі можете побачити у таблиці 3.1.

Таблиця 3.1 – Бібліотеки використані у програмі

Назва бібліотеки	Опис
tkinter	Бібліотека для створення GUI. Використовується у програмі для створення вікон, кнопок, текстових полів та інших елементів інтерфейсу програми.
cv2	Бібліотека для обробки зображень і відео. Використовується у програмі для роботи з відеопотоком, зчитування кадрів, виявлення ключових точок, обчислення оптичного потоку тощо.
scipy.spatial.transform.Rotation	Модуль Rotation є частиною бібліотеки SciPy і спеціалізується на роботі з обертаннями в тривимірному просторі. Він забезпечує інструменти для представлення, обчислення, комбінування та аналізу обертальних перетворень. Використовується у програмі для представлення, обчислення та перетворення матриць обертання.
numpy	Бібліотека для числових обчислень із використанням багатовимірних масивів. Використовується у програмі для обчислення матриць, векторів, обробки координат, обчислення геометричних перетворень, а також для роботи з точками в просторі.
threading	Бібліотека для роботи з багатопоточністю. Використовується у програмі для запуску паралельних потоків, таких як обробка відео та оновлення інтерфейсу користувача одночасно.
matplotlib.pyplot	Pyplot модуль у складі бібліотеки Matplotlib надає інструменти для створення статичних, інтерактивних та анімованих візуалізацій даних у Python. Використовується у програмі для створення 3D-графіків, наприклад, траєкторії руху камери чи ключових точок.
mpl_toolkits.mplot3d. Axes3D	Axes3D це модуль у складі Matplotlib, який додає підтримку для створення тривимірних графіків і візуалізацій. Використовується у програмі для візуалізації 3D-траєкторії камери.
time	Бібліотека для роботи з часом. Використовується у програмі для відстеження часу роботи програми.
os	Бібліотека для роботи з операційною системою. Використовується у програмі для взаємодії з файловою системою.
tqdm	Бібліотека для створення прогрес-барів у циклах. Використовується у програмі для відображення прогресу обробки даних.

Опис файлів програми:

1) gui.py. Цей файл реалізує GUI для візуальної одометрії. Додаток дозволяє налаштовувати параметри візуальної одометрії, запускати її в окремому потоці і візуалізувати результати. Список класів та методів можна побачити у таблиці 3.2.

Таблиця 3.2 – Список класів та методів файлу gui.py

Назва методу або класу	Опис
Клас VOApp	Клас VOApp є основним класом додатку, який відповідає за створення GUI та управління візуальною одометрією. Він ініціалізує параметри візуальної одометрії, такі як кількість ознак, кількість ітерацій, поріг вхідних точок та інші, а також створює елементи інтерфейсу, зокрема поля вводу і кнопки для налаштування цих параметрі.
Метод __init__	В методі __init__ задаються початкові значення параметрів, таких як кількість ознак, кількість ітерацій, поріг вхідних точок тощо. Потім викликається метод setup_ui для створення графічного інтерфейсу користувача.
Метод setup_ui	Метод setup_ui відповідає за створення елементів інтерфейсу, таких як поля вводу для параметрів (наприклад кількість ознак, кількість ітерацій, обрізання кадрів) і кнопки для запуску процесу візуальної одометрії. Інтерфейс дозволяє користувачеві налаштовувати параметри перед початком роботи.
Метод start_vo	Метод start_vo відповідає за перевірку введених значень та ініціалізацію нового потоку для запуску процесу візуальної одометрії. Всі параметри передаються в окремий потік для виконання основних обчислень, що допомагає уникнути блокування інтерфейсу користувача під час роботи програми.
Метод compute_mse	Метод compute_mse використовується для порівняння істинного шляху та оціненого шляху в 3D-просторі. Він обчислює MSE та RMSE, що дозволяє оцінити точність результатів візуальної одометрії.
Метод run_vo	Метод run_vo отримує параметри від користувача (кількість ознак, кількість ітерацій, розмір вікна оптичного потоку та інші налаштування) і передає їх до функцій модуля visual_odometry для виконання обчислень. Після цього результат візуалізується разом з часом виконання роботи, MSE та RMSE.

Детальна реалізація файлу міститься в Додатку А.

2) visual_odometry.py. Цей файл реалізує методи для виконання візуальної одометрії (Visual Odometry) за допомогою бібліотеки OpenCV, а також використовує оптичний потік (метод Лукаса-Канаде) для оцінки руху камери в тривимірному просторі. Основна мета цього коду — обчислення позиції та орієнтації камери, використовуючи зображення з послідовностей та дані IMU. Клас VisualOdometry є основним компонентом цього файлу, який інтегрує різні методи, включаючи обробку зображень та оцінку матриць трансформації. Список класів та методів можна побачити у таблиці 3.3.

Таблиця 3.3 – Список класів та методів файлу visual_odometry.py

Назва методу або класу	Опис
Клас VisualOdometry	Клас VisualOdometry реалізує алгоритм візуальної одометрії. Він інтегрує різні методи для завантаження калібрувальних даних, зображень, даних ground truth та IMU, а також обробляє ці дані для визначення позицій та руху камери в просторі.
Метод __init__	Метод __init__ ініціалізує об'єкт VisualOdometry, завантажуючи всі необхідні дані: калібрувальні матриці, матриці проекції, дані ground truth і IMU, а також послідовність зображень з вказаного каталогу. Цей метод викликається при створенні об'єкта класу і надає базову конфігурацію для подальшої роботи.
Метод load_calibration	Метод load_calibration завантажує калібрувальні параметри камери з вказаного файлу. Він відкриває файл, зчитує матрицю проекції та визначає внутрішню матрицю камери, що містить її фокусну відстань та інші параметри.
Метод load_ground_truth_and_imu	Метод load_ground_truth_and_imu завантажує дані ground truth та IMU за допомогою функції load_optitrak_data. Він приймає шляхи до файлів з даними про позу та час і повертає відповідні трансформаційні матриці та відстані.
Метод load_images	Метод load_images завантажує зображення з вказаного каталогу. Зображення обробляються за допомогою OpenCV, і повертається їх список для подальшої обробки. Це дає можливість працювати з великою кількістю зображень, що містять інформацію про рух камери.
Метод create_transformation_matrix	Метод create_transformation_matrix створює матрицю трансформації на основі матриці обертання та вектора переміщення. Ця матриця комбінує обертання та переміщення для зміщення кадрів у 3D просторі, що дозволяє відновити рух камери між кадрами.

Продовження таблиці 3.3

Назва методу або класу	Опис
Метод get_feature_matches	Метод get_feature_matches обчислює оптичний потік між двома зображеннями за допомогою методу Лукаса-Канаде. Він збирає відповідні точки між кадрами, візуалізує їх та повертає координати точок на обох зображеннях.
Метод compute_pose	Метод compute_pose обчислює позу камери, тобто матрицю обертання та вектор переміщення, між двома зображеннями на основі співпадаючих точок. Він використовує есеціальну матрицю для оцінки цих параметрів, що дозволяє отримати інформацію про позицію камери в просторі.
Метод compute_positive_z_count	Метод compute_positive_z_count обчислює кількість точок з позитивними координатами Z у двох наборах тривимірних точок, що отримуються через триангуляцію. Метод оцінює, скільки точок знаходяться перед камерою, що допомагає оцінити точність оцінки позиції.
Метод decompose_essential_matrix	Метод decompose_essential_matrix виконує декомпозицію есеціальної матриці, яка описує геометричні відносини між двома зображеннями, отриманими з різних точок зору, на можливі варіанти обертання та трансляції. Ця операція дає два можливих варіанти для кожного з параметрів, що дає змогу вибрати правильний шлях руху камери або встановити її орієнтацію в просторі. Результати декомпозиції використовуються для реконструкції 3D шляху, оцінки руху об'єкта чи камери, а також для обчислення відстані та орієнтації між точками в просторі.
Метод plot_optical_flow	Метод plot_optical_flow візуалізує оптичний потік, який показує, як змінюються положення точок зображення при їх переміщенні між двома послідовними кадрами. Для цього на зображенні малюються стрілки, що вказують напрямок та величину переміщення кожної точки. Така візуалізація дозволяє відстежувати рух об'єктів, оцінювати їх переміщення в просторі, а також аналізувати швидкість та траєкторії руху в реальному часі, що є корисним для задач з візуального одометрії та трекінгу.
Метод main	Метод main викликає основну функцію з модуля visual_odometry для виконання візуальної одометрії, збирає результат і візуалізує порівняння між оціненим шляхом та істинними даними в 3D просторі. Ще цей метод застосовує корекцію шляху за допомогою матриці обертання, щоб вирівняти результат візуальної одометрії з реальними даними.

Детальна реалізація файлу міститься в Додатку Б.

3) imu_integration.py. Цей файл обробляє та аналізує дані позицій та часових міток з файлів csv для створення трансформаційних матриць.

Він також обчислює відстані між точками на основі прискорень та часових міток за допомогою чисельного інтегрування. Файл містить функції для завантаження та обробки даних з файлів, а також для створення коригуючих матриць, що дозволяє вирівняти результат візуальної одометрії з реальними даними. Список методів можна побачити у таблиці 3.4.

Таблиця 3.4 – Список методів файлу imu_integration.py

Назва методу	Опис
Метод load_optitrak_data	Метод load_optitrak_data завантажує та обробляє дані з файлів CSV, що містять інформацію про позиції та орієнтації об'єкта. Використовуючи часові мітки, метод синхронізує дані позицій з відповідними вимірюваннями, що дозволяє точно співвіднести інформацію про рух з часом. Далі створюються трансформаційні матриці для кожної позиції, де орієнтація коригується за допомогою матриць обертання. Останнім етапом є обчислення відстаней між точками на основі прискорень, що дає змогу оцінити шлях об'єкта з високою точністю.
Метод Integrate	Метод Integrate обчислює відстань між точками за допомогою чисельного інтегрування. Використовуючи метод трапецидальних сум, він обчислює швидкість по кожній осі на основі виміряних прискорень, після чого інтегрує ці швидкості для отримання переміщення. Для оцінки загальної відстані між точками застосовується евклідова відстань, що дозволяє точно визначити пройдений шлях.

Детальна реалізація файлу міститься в Додатку В.

4) foe_ransac.py. Цей код реалізує методи для оцінки FOE за допомогою методу RANSAC, а також для фільтрації внутрішніх точок на основі відстані від середнього значення в статичних сценах. Він включає функції для знаходження найкращої точки сходження з векторів потоку, а також для фільтрації точок і коригування результатів з урахуванням розміру зображення. Ключовою частиною є використання RANSAC для підвищення точності оцінок точки сходження та зменшення впливу шуму. Список методів можна побачити у таблиці 3.5.

Таблиця 3.5 – Список методів файлу foe_ransac.py

Назва методу	Опис
Метод estimate_foe	Метод estimate_foe оцінює FOE за допомогою методу RANSAC для векторів потоку, що дозволяє знаходити точку, до якої сходяться всі потоки на зображенні. Для цього метод випадковим чином вибирає дві точки з масиву векторів потоку, щоб розрахувати потенційну точку сходження. Потім оцінює кількість внутрішніх точок. Метод використовує поріг для визначення, які точки вважаються внутрішніми, і припиняє ітерації, коли кількість внутрішніх точок перевищує 90% від загальної кількості точок.
Метод inlier_static	Метод inlier_static використовує функцію estimate_foe для оцінки точки сходження на статичній сцені. Він фільтрує точки та напрямки, базуючись на відстані точок від середнього значення, щоб виділити лише найбільш релевантні точки для оцінки. Метод також використовує RANSAC для фільтрації точок і напрямків, що дозволяє зменшити вплив шуму та помилок на кінцеву точку сходження. Після фільтрації метод повертає відфільтровані координати точок, напрямки та точку сходження, кориговану відповідно до розмірів зображення, що дозволяє отримати більш точну та стабільну оцінку FOE.

Детальна реалізація файлу міститься в Додатку Г.

5) lucas_kanade_flow.py. Цей файл реалізує алгоритм для обчислення оптичного потоку за методом Лукаса-Канаде, використовуючи піраміду зображень для зменшення розмірів зображень та прискорення обчислень. Він включає функції для обчислення індексів вікон, градієнтів зображень та різниці інтенсивностей, а також для обчислення швидкості точок за допомогою оптичного потоку. Ключовою частиною є ітеративне коригування швидкості точок для поліпшення точності результатів, а також застосування методів для фільтрації аномальних точок та обчислення остаточного руху точок. Окрім того, використовуються багаторівневі піраміди зображень для покращення точності виявлення та відстеження точок на різних рівнях деталізації зображень. Список методів можна побачити у таблиці 3.6.

Таблиця 3.6 – Список методів файлу lucas_kanade_flow.py

Назва методу	Опис
Метод calculate_window	Метод calculate_window обчислює індекси пікселів для кожного вікна навколо точки на зображенні, при цьому враховується обмеження меж вікна, щоб уникнути виходу за межі зображення. Повертає індекси пікселів для поточної та попередньої координати в межах вікна для кожного пікселя. Це важливо для подальшого аналізу інтенсивності пікселів у вікні та виконання точних обчислень під час пошуку оптичного потоку.
Метод calculate_gradients	Метод calculate_gradients обчислює просторові градієнти інтенсивності пікселів зображення по осях X та Y для кожного пікселя в межах заданого вікна. Градієнти використовуються для подальшого обчислення оптичного потоку, дозволяючи ідентифікувати зміни інтенсивності в межах зображення, що є критичним для відстеження руху між кадрами.
Метод compute_intensity_difference	Метод compute_intensity_difference обчислює різницю інтенсивностей між двома зображеннями, використовуючи вектори швидкості та градієнти інтенсивності пікселів. Це дозволяє оцінити зміни інтенсивності зображення внаслідок переміщення об'єктів між двома кадрами, надаючи важливу інформацію для подальшого аналізу руху та оптичного потоку.
Метод lucas_kanade_optical_flow	Метод lucas_kanade_optical_flow реалізує алгоритм Лукаса-Канаде для обчислення оптичного потоку між двома зображеннями. Алгоритм створює піраміди зображень для підвищення ефективності виявлення і відстеження рис, визначає вектори руху точок і проводить ітерації для покращення точності результату. В кінці застосовує метод RANSAC для фільтрації викидів, забезпечуючи точніші результати обчислення оптичного потоку, навіть при наявності шуму або помилок у даних.

Детальна реалізація файлу міститься в Додатку Д.

3.4 Аналіз результатів експериментів

Було проведено серію тестів з варіативними значеннями ключових параметрів програми: кількість ознак, кількість ітерацій, поріг вхідних точок, обрізання початкових кадрів та обрізання кінцевих кадрів. Для кожного набору параметрів були оцінені MSE, RMSE та час виконання програми. MSE відображає середню величину квадратів різниць між обчисленими та істинними значеннями координат траєкторії, що дає загальне уявлення про

точність методу. RMSE, своєю чергою, є квадратним коренем з MSE, що дозволяє інтерпретувати похибку у тих самих одиницях вимірювання, що й координати. Порівняння цих метрик дозволяє краще зрозуміти як окремі помилки, так і їхній вплив на загальну точність. Час виконання програми показує ефективність алгоритму, що є критично важливим для реального використання, особливо у задачах навігації, де потрібна обробка даних у режимі реального часу. Аналіз цих показників дозволяє визначити оптимальні параметри, які забезпечують баланс між точністю, швидкістю та стабільністю роботи програми, що особливо важливо для її практичного застосування.

Початкові параметри були обрані такі:

- Кількість ознак – 500, була обрана як компроміс між якістю результатів і швидкістю обробки;
- Кількість ітерацій – 70, була вибрана для забезпечення стабільних результатів при розумному використанні часу виконання;
- Поріг вхідних точок – 5, відповідає мінімальній кількості точок для стабільного функціонування алгоритму, забезпечуючи точність при реалістичних даних, де шум може бути присутнім;
- Обрізання кадрів було обране 1 для початкових та 5 для кінцевих для зменшення впливу шуму.

1. Експеримент 1: Зміна кількості ознак:

Кількість ознак визначає кількість характерних точок, які алгоритм вибирає на кожному кадрі для відстеження. Ці точки є унікальними елементами зображення, які легко ідентифікувати на наступних кадрах, що дозволяє алгоритму порівнювати послідовні зображення та оцінювати рух.

Для першого тесту були взяті параметри, які були визначені як початкові (Рисунок 3.4).

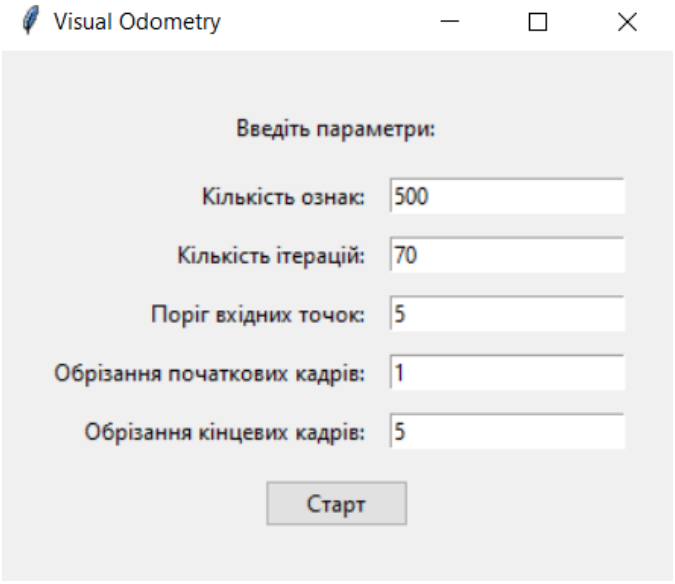

Рисунок 3.4 – Параметри для першого тесту

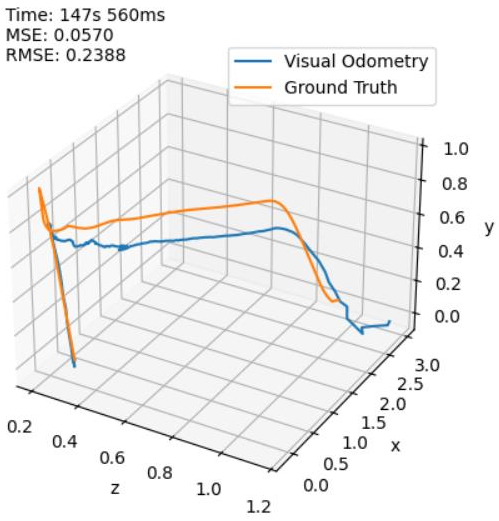

Рисунок 3.5 – Результат при кількості ознак 500

Аналіз рисунку 3.5 показує нам наші перші результати. MSE є 0.0570, що є невеликим що є добре. RMSE є 0.2388 що є помірним результатом. Обидва показники свідчать про помірну точність моделі, з помилками, які можуть бути прийнятними для даної системи.

Для другого тесту кількість вхідних точок була зменшена (Рисунок 3.6).

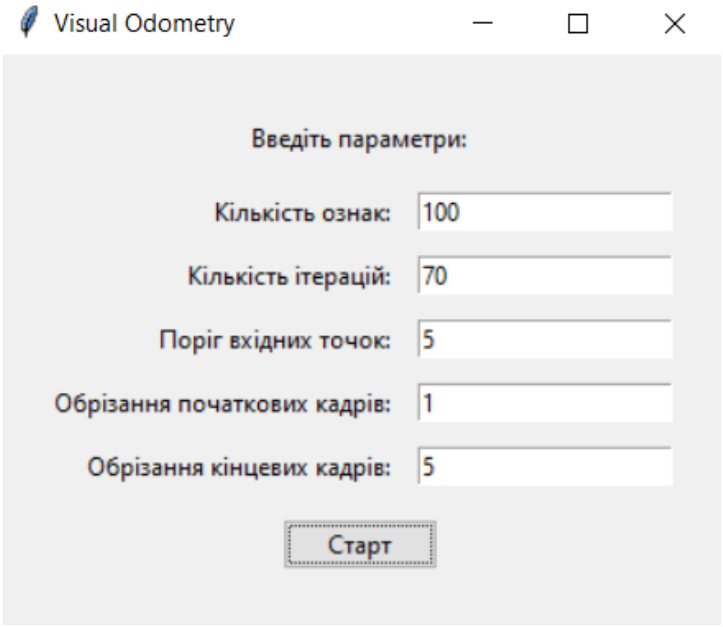

Рисунок 3.6 – Параметри для другого тесту

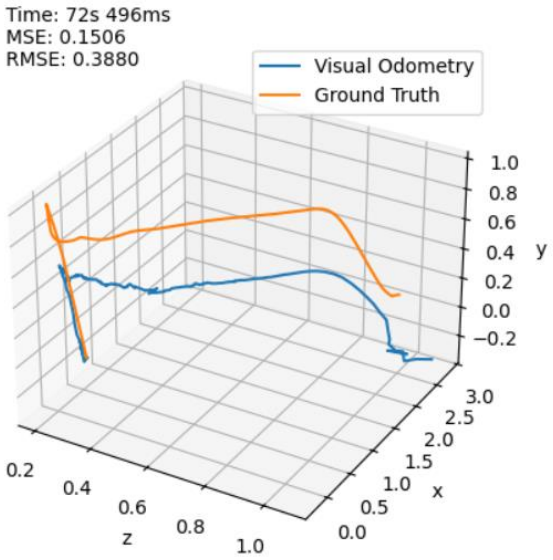

Рисунок 3.7 – Результат при кількості ознак 100

Аналіз рисунку 3.7 показує значне зменшення точності траєкторії при зменшенні точок приблизно у два рази, але при цьому швидкість роботи скоротилася також у два рази.

Для третього тесту кількість вхідних точок була збільшена(Рисунок 3.8).

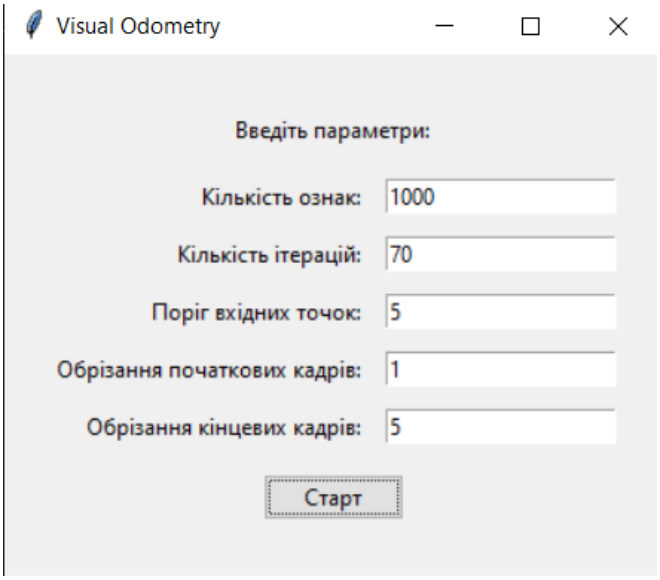

Рисунок 3.8 – Параметри для третього тесту

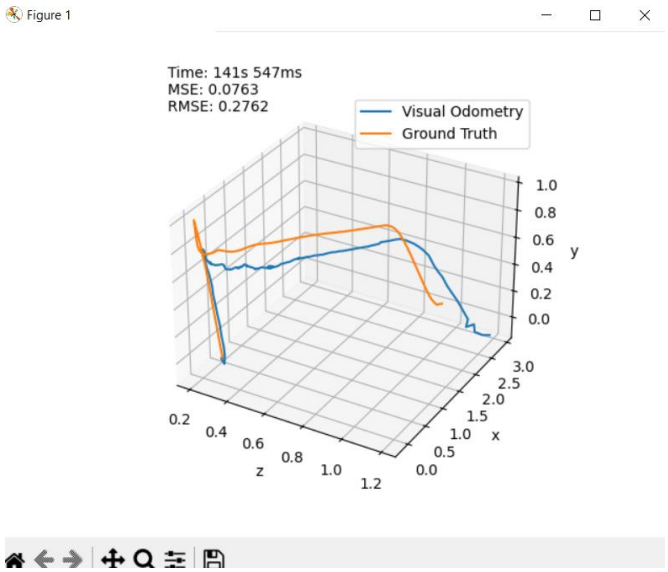

Рисунок 3.9 – Результат при кількості ознак 1000

Аналіз рисунку 3.9 показує незначне зменшення точності траєкторії при збільшенні точок, але при цьому швидкість роботи несуттєво скоротилася.

Таблиця 3.7 – Результати експерименту 1

Кількість ознак	Кількість ітерацій	Поріг вхідних точок	Обрізання початкових кадрів	Обрізання кінцевих кадрів	MSE	RMSE	Час роботи програми
500	70	5	1	5	0,057	0,2388	147,56 с.
100	70	5	1	5	0,1506	0,388	72,496 с.
1000	70	5	1	5	0,0763	0,2762	141,547 с.

Експеримент показав що при дуже малих значеннях цього параметру час роботи програми зменшкється, але значення MSE та RMSE є більшими чим наприклад при параметрі ключових точок 500 який показав дуже малий параметр MSE та RMSE але час виконання програми збільшився майже у двічі, якщо брати параметр ключових точок 1000 то через збільшення кількості точок збільшилася кількість шумів через що MSE та RMSE стали гіршими чим при параметрі ключових точок 500, але тут можна побачити цікаву аномалію, що при час роботи трохи зменшився, але з наступних тестів ми побачимо що збільшення кількості ключових точок збільшує час виконання програми (таблиця 3.7).

Оптимальним параметром є 500 ознак так як якість до та після є меншою хоча кількість ключових точок 100 має кращий час роботи.

2. Експеримент 2: Зміна кількості ітерацій:

Кількість ітерацій визначає, скільки разів алгоритм виконує спроби для пошуку найкращого рішення або моделі, що найбільше відповідає даним.

Для першого тесту були взяті параметри, які були визначені як початкові (Рисунок 3.10).

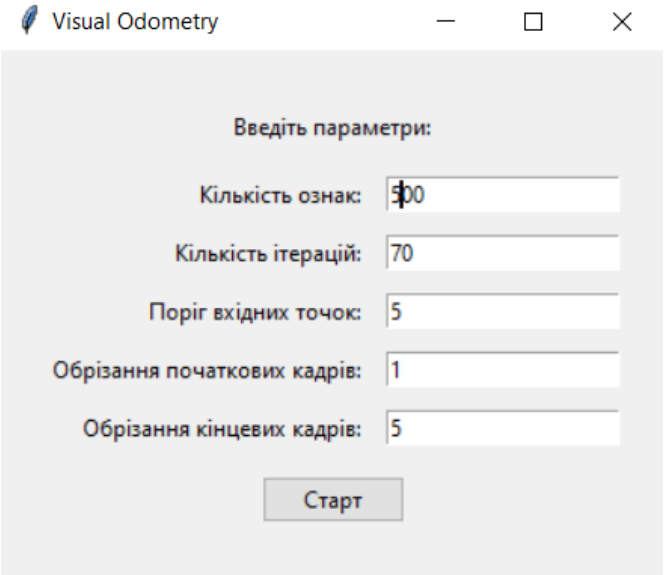

Рисунок 3.10 – Параметри для першого тесту

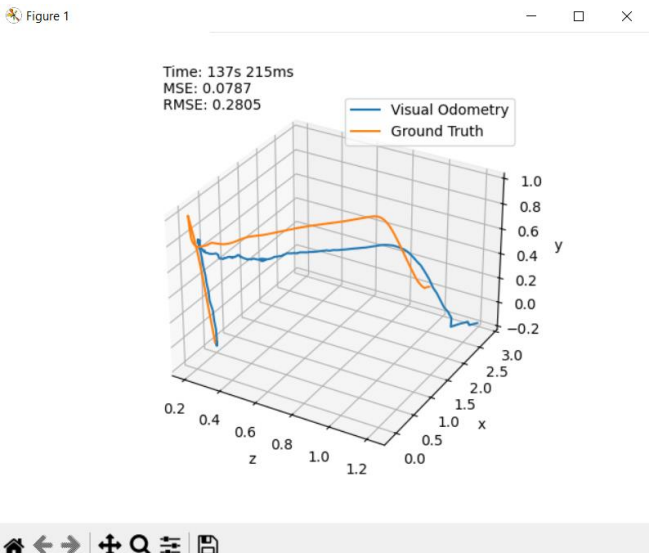

Рисунок 3.11 – Результат при кількості ітерацій 70

Аналіз рисунку 3.11 показує нам результат схожі до першого тесту першого експерименту, але точність впала приблизно на 25% але час роботи при цьому також незначно скоротився.

Для другого тесту кількість ітерацій була зменшена (Рисунок 3.12).

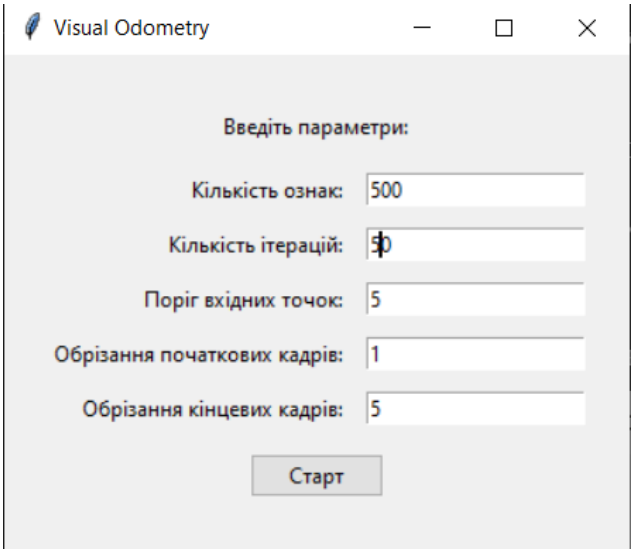

Рисунок 3.12 – Параметри для другого тесту

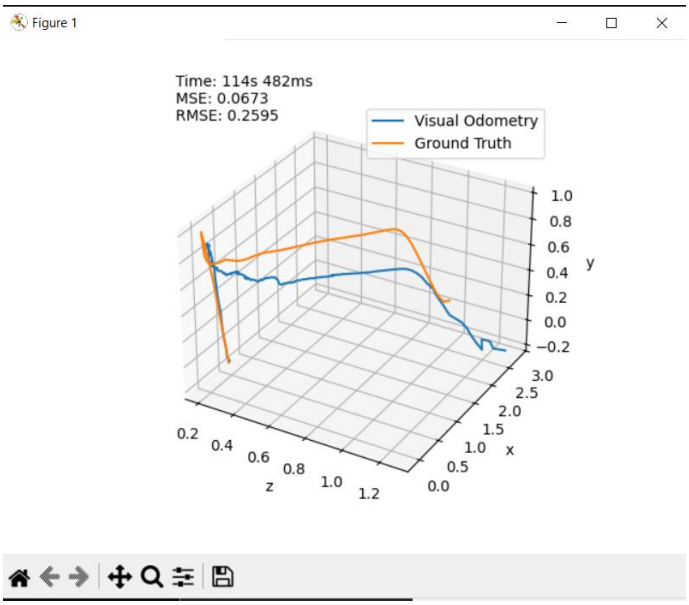

Рисунок 3.13 – Результат при кількості ітерацій 50

Аналіз рисунку 3.13 показує гарні результати MSE та RMSE при помітному скороченні часу роботи програми.

Для третього тесту кількість ітерацій була збільшена (Рисунок 3.14).

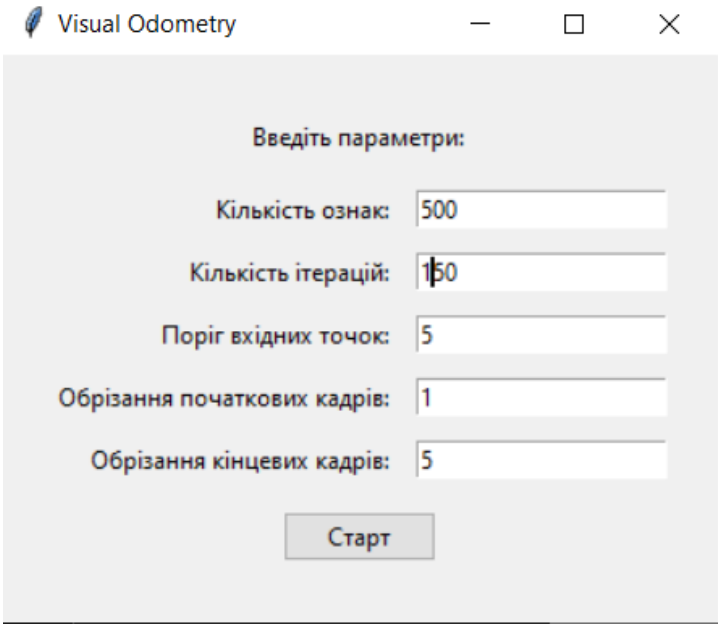

Рисунок 3.14 – Параметри для третього тесту

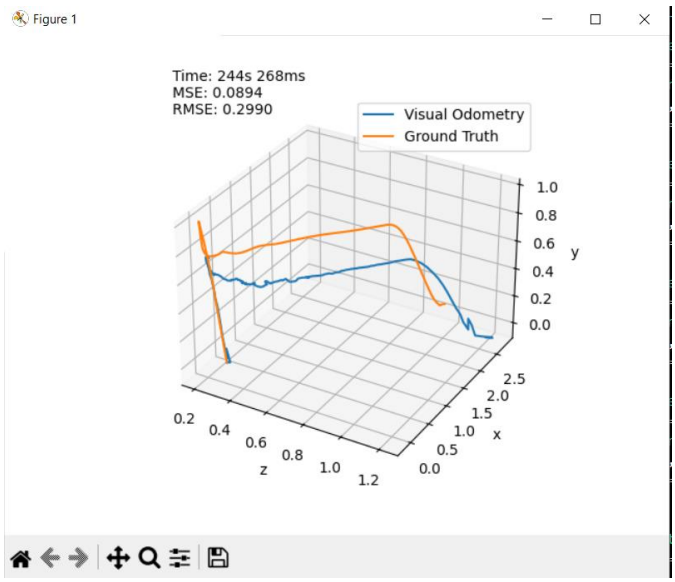

Рисунок 3.15 – Результат при кількості ітерацій 150

Аналіз рисунку 3.15 показує помірні результати mse при дуже великому часі роботи програми, що показує нам що параметр ітерацій не потрібно брати занадто великим.

Таблиця 3.8 – Результати експерименту 2

Кількість ознак	Кількість ітерацій	Поріг вхідних точок	Обрізання початкових кадрів	Обрізання кінцевих кадрів	MSE	RMSE	Час роботи програми
500	70	5	1	5	0,0787	0,2805	137,215 с.
500	50	5	1	5	0,0673	0,2595	114,482 с.
500	150	5	1	5	0,0894	0,299	244,268 с.

Однозначно найкращий результат показало значення 50 тому воно і ж оптимальним параметром, хоча значення між 70 були невеликими тому для подальших тестів було залишене це значеннння, так як незначне збільшення ітерацій може будувати більш стабільні візуальні траєкторії при невеликому збільшенні часу (таблиця 3.8).

3. Експеримент 3: Зміна кількості порогу вхідних точок:

Поріг вхідних точок визначає мінімальну допустиму відстань або величину, за якою оцінюється, чи належить точка до внутрішніх або зовнішніх точок. Цей параметр суттєво впливає на відбір "корисних" точок, що відповідають реальному руху камери, та відсіювання шуму або неправдивих точок, які можуть спотворювати кінцевий результат.

Для першого тесту були взяті параметри, які були визначені як початкові (Рисунок 3.16).

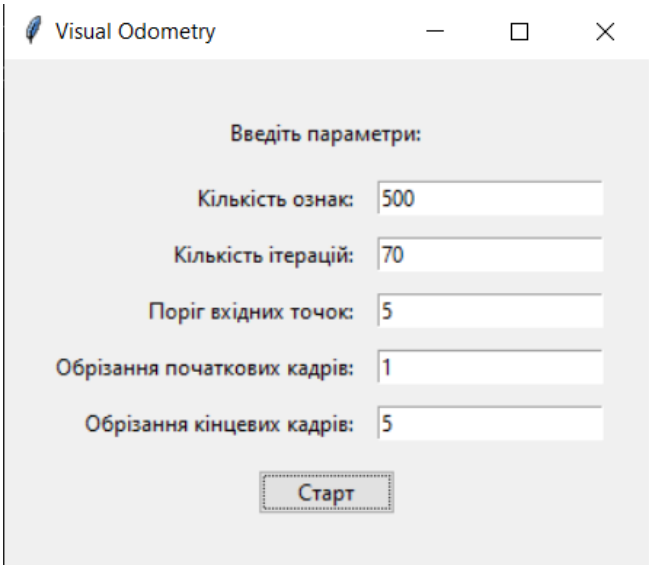

Рисунок 3.16 – Параметри для першого тесту

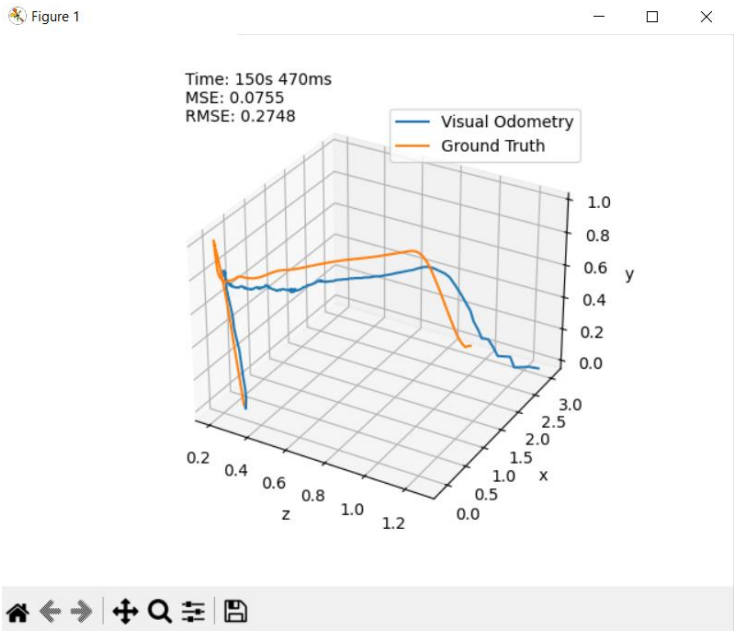

Рисунок 3.17 – Результат при кількості порогу вхідних точок – 5

Аналіз рисунку 3.17 показує результати дуже схожі до минулих тестів з початковими даними, значення MSE та RMSE є подібними до першого тесту другого експерименту з помірним збільшенням часу.

Для другого тесту кількість порогу вхідних точок була зменшена (Рисунок 3.18).

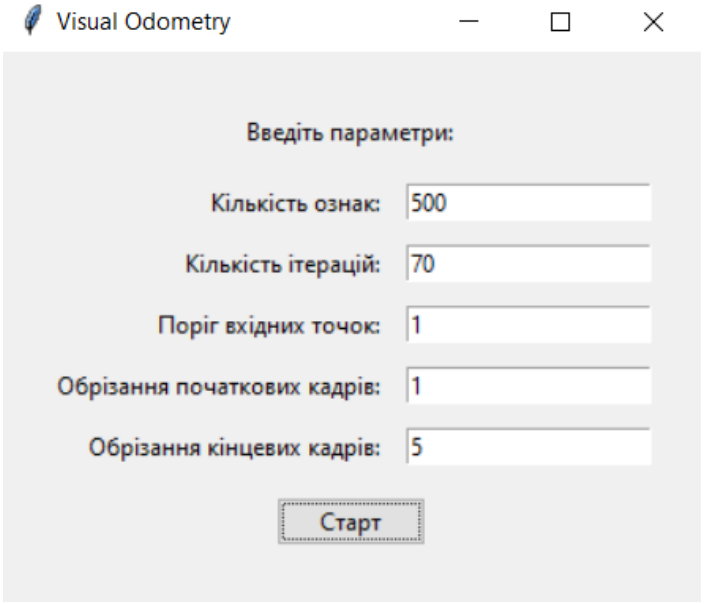

Рисунок 3.18 – Параметри для другого тесту

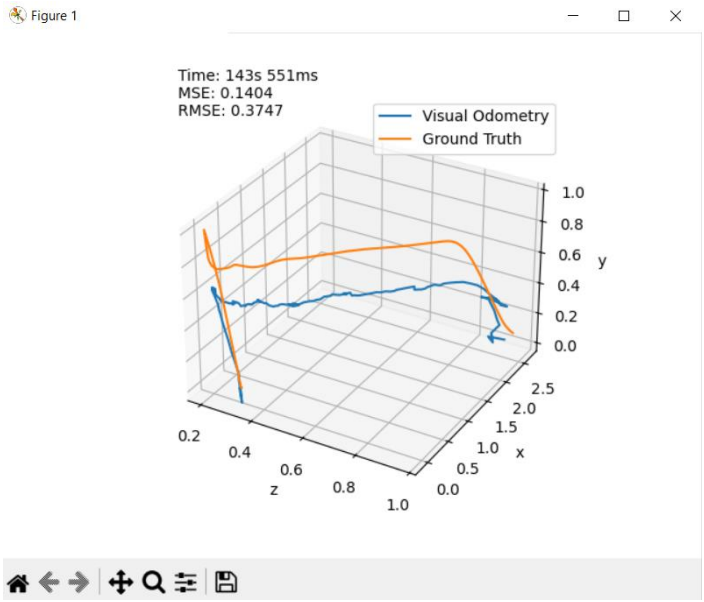

Рисунок 3.19 – Результат при кількості порогу вхідних точок – 1

Аналіз рисунку 3.19 показує значне зменшення точності траєкторії при зменшенні порогу вхідних точок приблизно у два рази, але на відміну від

інших параметрів погіршення точності MSE та RMSE не призвело до зменшення часу рогботи програми.

Для третього тесту кількість порогу вхідних точок була зменшена (Рисунок 3.20).

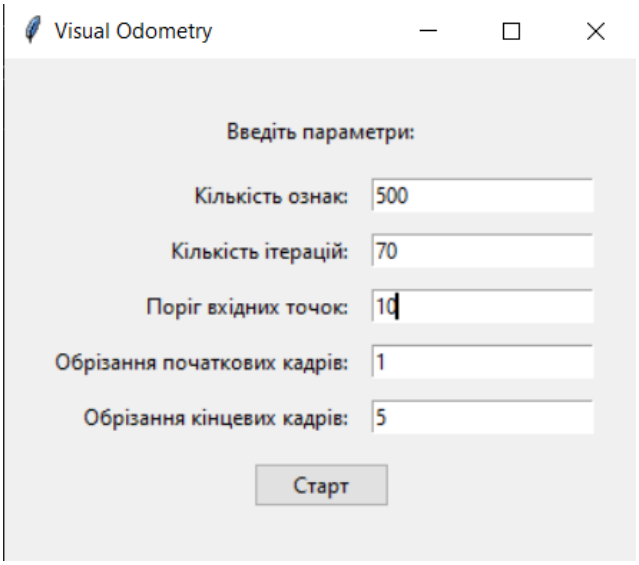

Рисунок 3.20 – Параметри для третього тесту

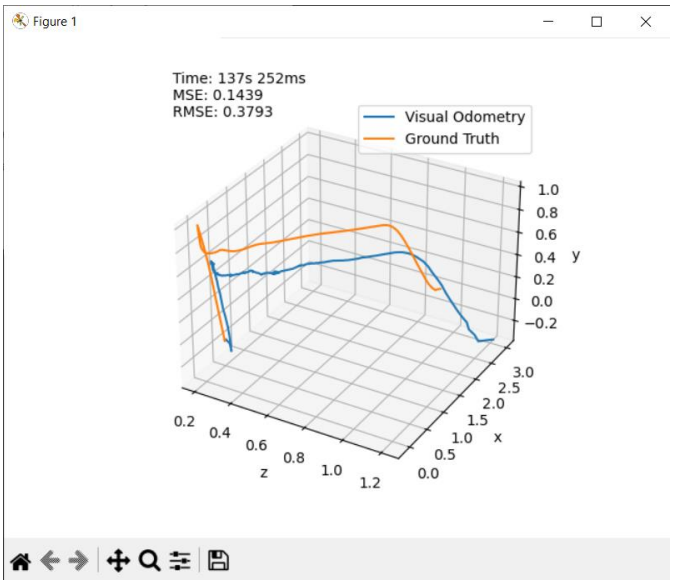

Рисунок 3.21 – Результат при кількості порогу вхідних точок – 10

Аналіз рисунку 3.21 показує значне зменшення точності траєкторії при збільшенні порогу вхідних точок приблизно у два рази, причина тому є те що такий фільтр є занадто жорстоким і він відкидає потенційно корисні точки.

Таблиця 3.9 – Результати експерименту 3

Кількість ознак	Кількість ітерацій	Поріг вхідних точок	Обрізання початкових кадрів	Обрізання кінцевих кадрів	MSE	RMSE	Час роботи програми
500	70	5	1	5	0,0755	0,2748	150,470 с.
500	50	1	1	5	0,1404	0,3747	143,551 с.
500	150	10	1	5	0,1439	0,3793	137,252 с.

Цей параметр найбільше впливає на точність траєкторії але майже не впливає на час. Оптимальним параметром виявляється 5, бо 1 пропускає занадто багато шуму, а 10 є занадто жорстким фільтром який фільтрує вже потенційно корисні точки (таблиця 3.9).

4. Експеримент 4: Робота програми при дуже низьких параметрах:

Наступний експеримент буде проведений з низькими параметрами кількості ознак, кількості ітерацій та порогу вхідних точок. Точне значення параметрів можете побачити на рисунку 3.22.

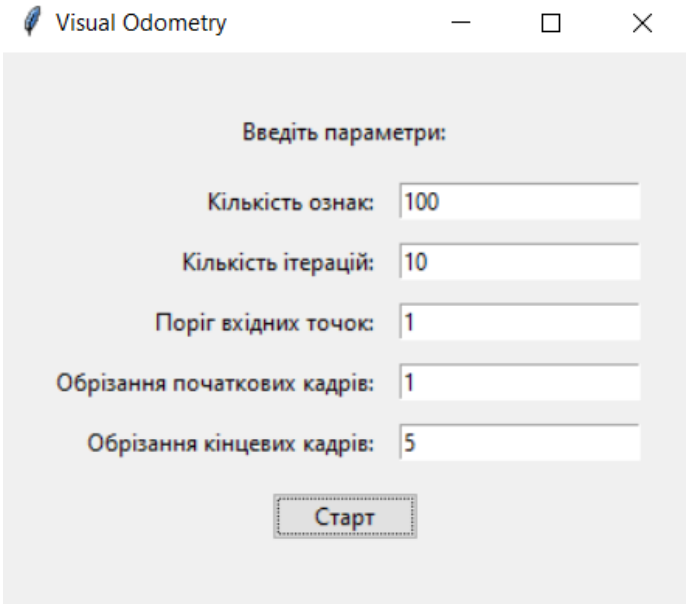

Рисунок 3.22 – Параметри для тесту

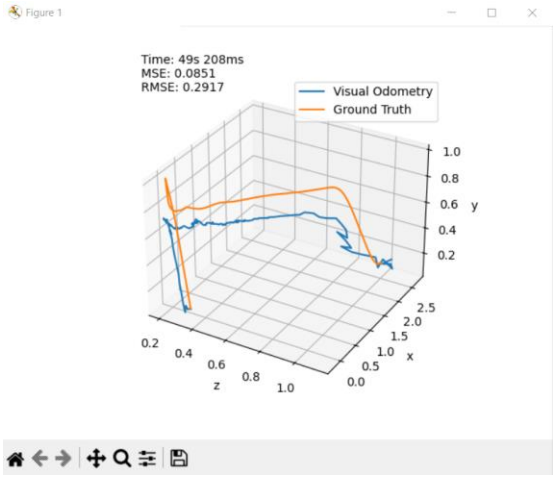

Рисунок 3.23 – Результат при низьких параметрах

Аналіз рисунку 3.23 показує нам не великі результати MSE та помірні результати RMSE при наймешому часі роботи програми на даному етапі. Ці результати роботи програми є найкращими, але потрібно відмітити нестабільність візуального представлення траєкторії при таких параметрах.

5. Експеримент 5: Робота програми при дуже високих параметрах:

Наступний екперимент буде проведений з високими параметрами кількості ознак, кількості ітерацій та порогу вхідних точок. Точне значення параметрів можете побачити на рисунку 3.24.

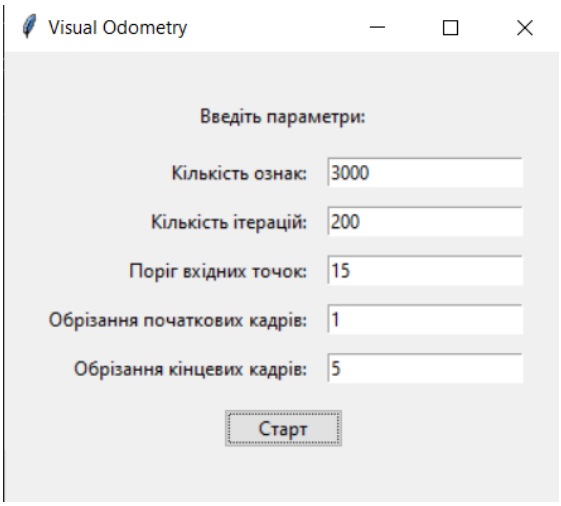

Рисунок 3.24 – Параметри для тесту

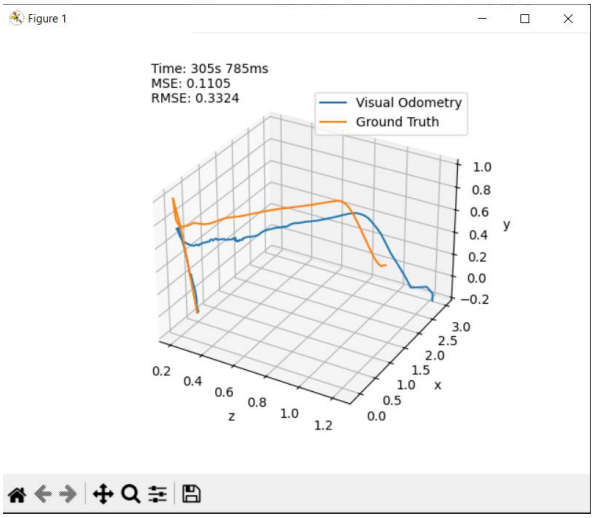

Рисунок 3.25 – Результат при високих параметрах

Аналіз рисунку 3.25 показує нам помірні результати MSE та помірні результати RMSE при дуже довгому часі роботи програми.

Таблиця 3.10 – Результати експериментів 4-5

Кількість ознак	Кількість ітерацій	Поріг вхідних точок	Обрізання початкових кадрів	Обрізання кінцевих кадрів	MSE	RMSE	Час роботи програми
100	10	1	1	5	0,0851	0,2917	49,208 с.
3000	200	15	1	5	0,1105	0,3324	305,785 с.

Результати при малих параметрах є кращими ніж при великих є такими бо на них впливає менше шуму чим при великих значеннях і алгоритм не є перевантаженим що покращує не тільки якість, а і швидкість (таблиця 3.10).

6. Експеримент 6: Робота програми при змішаних параметрах:

Наступний тест буде проведений з малою кількістю ознак, великою кількістю ітерацій та високим порогом вхідних точок. Точне значення параметрів можете побачити на рисунку 3.26.

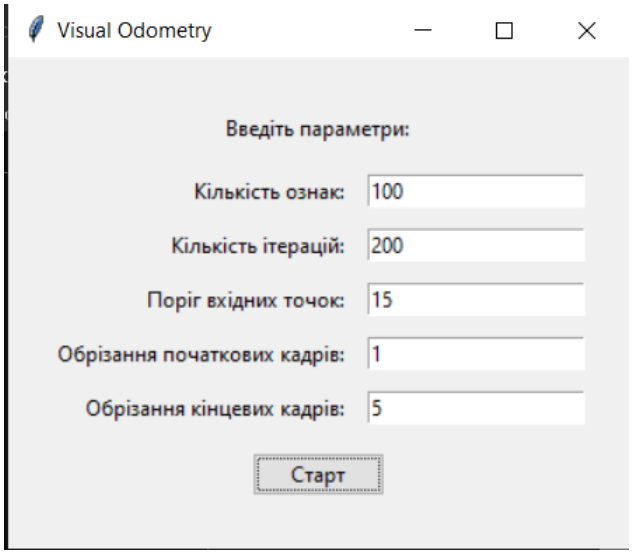

Рисунок 3.26 – Параметри для тесту з малою кількістю ознак, великою кількістю ітерацій та високим порогом вхідних точок

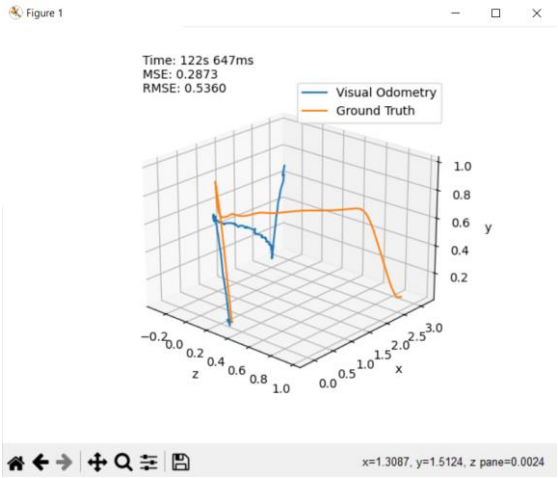

Рисунок 3.27 – Результат при малій кількісті ознак, великою кількістю ітерацій та високим порогом вхідних точок

Аналіз рисунку 3.27 показує нам найгірші результати MSE та RMSE на даному етапі, але час роботи є гарним. Потрібно відмітити також що результат візуального представлення траєкторії є найменш схожим на ground truth особливо по осях Z та Y.

Наступний тест буде проведений з великою кількістю ознак, малою кількістю ітерацій та малим порогом вхідних точок. Точне значення параметрів можете побачити на рисунку 3.28.

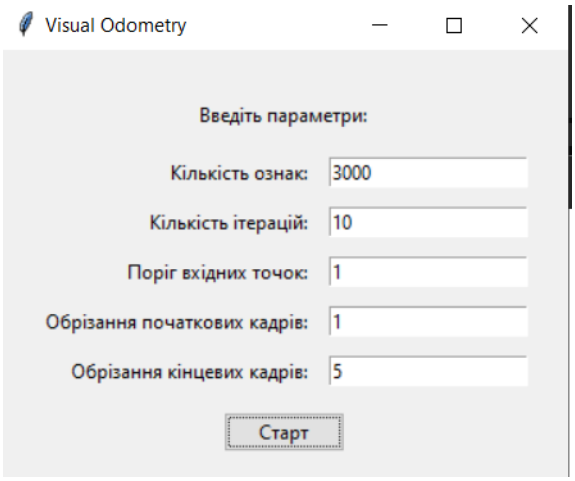

Рисунок 3.28 – Параметри для тесту з великою кількістю ознак, малою кількістю ітерацій та малим порогом вхідних точок

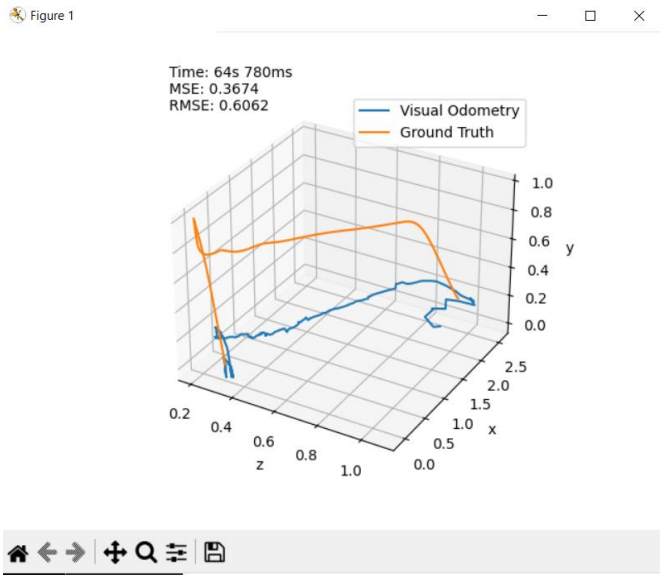

Рисунок 3.29 – Результат при великій кількісті ознак, малій кількістю ітерацій та малим порогом вхідних точок

Аналіз рисунку 3.29 показує нам ще гірші результати MSE та RMSE чим при минулому тесті, але з дуже малим часом роботи. Знову результат

візуального представлення траєкторії є дуже відміним від ground truth але у цьому випадку лише по осі Y.

Таблиця 3.11 – Результати експерименту 6

Кількість ознак	Кількість ітерацій	Поріг вхідних точок	Обрізання початкових кадрів	Обрізання кінцевих кадрів	MSE	RMSE	Час роботи програми
100	200	15	1	5	0,2873	0,534	122,647 с.
3000	10	1	1	5	0,3674	0,6062	64,780 с.

Таким чином були отримані найгірші параметри, бо у першому випадку при дуже малій кількості ключових точок був застосований занадто суворий фільтр який відкинув потенційно корисні точки, а у другому це сталося через зворотню причину надто слабкий фільтр при надто високій кількості ключових точок що пропустило багато шуму (таблиця 3.11).

7. Експеримент 7: Робота програми при обрізанні кадрів:

Перший тест буде стосуватися впливу шуму у перших та останніх кадрах набору даних, тут обрізання кадрів було зменшено до 1 на початку та у кінці, бо ці кадри завжди мають найменшу кількість корисної інформації. Точне значення параметрів можете побачити на рисунку 3.30.

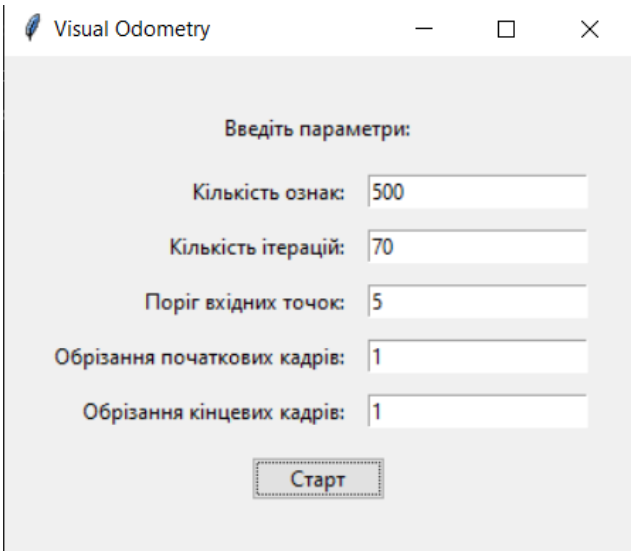

Рисунок 3.30 – Параметри для тесту з обрізанням мінімальної кількості кадрів

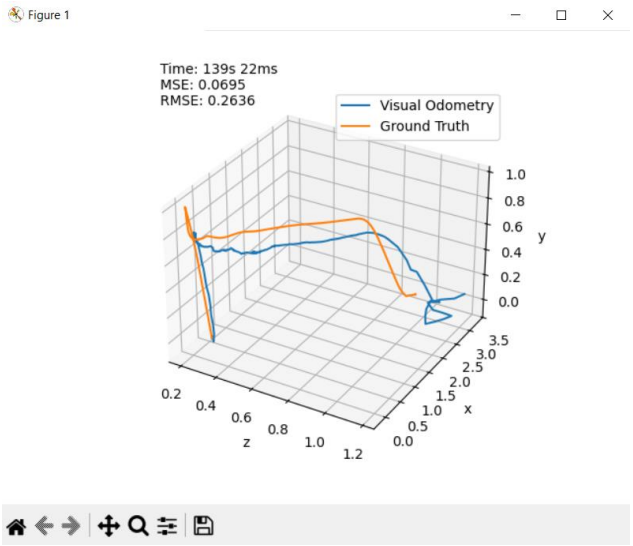

Рисунок 3.31 – Результат з обрізанням мінімальної кількості кадрів

Аналіз рисунку 3.31 показує нам високу кількість шуму в останніх кадрах набору даних, але на загальну точність MSE та RMSE це немає високого впливу.

Другий тест покаже вплив шуму останніх кадрів при малих параметрах. Точне значення параметрів можете побачити на рисунку 3.32.

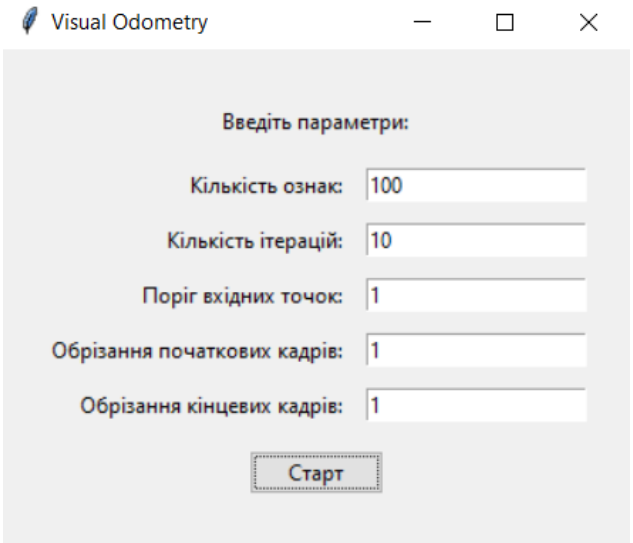

Рисунок 3.32 – Параметри для тесту з обрізанням мінімальної кількості кадрів і малими параметрами

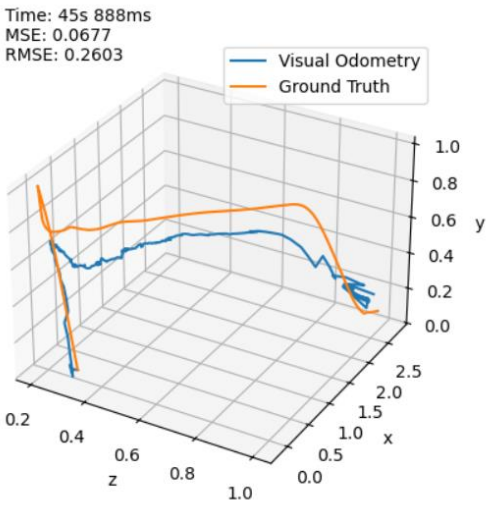

Рисунок 3.33 – Результати для тесту з обрізанням мінімальної кількості кадрів і малими параметрами

Аналіз рисунку 3.33 показує нам вплив шуму на точність візуального представлення траєкторії при малих параметрах, але на загальну точність MSE та RMSE це немає високого впливу.

Третій тест покаже вплив шуму останніх кадрів при великих параметрах. Точне значення параметрів можете побачити на рисунку 3.34.

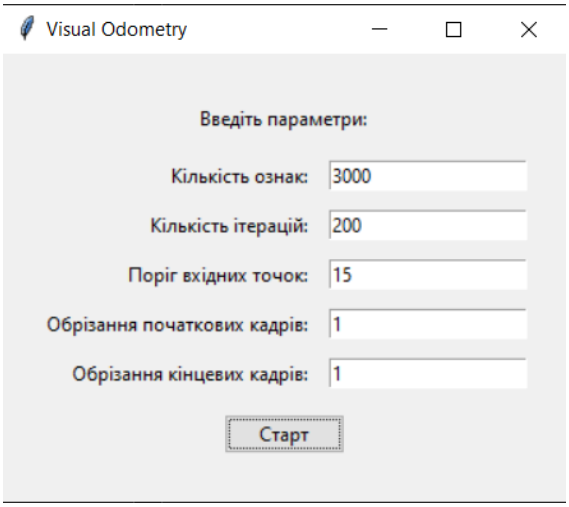

Рисунок 3.34 – Параметри для тесту з обрізанням мінімальної кількості кадрів і великими параметрами

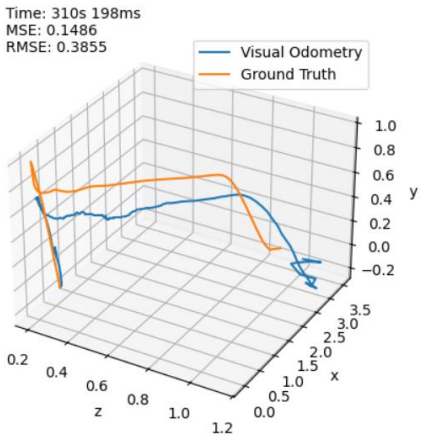

Рисунок 3.35 – Результати для тесту з обрізанням мінімальної кількості кадрів і великими параметрами

Аналіз рисунку 3.35 показує нам вплив шуму на точність візуального представлення траєкторії при великих параметрах, але на загальну точність MSE та RMSE це немає високого впливу.

Насупний тест покаже вплив шуму у перших та останніх кадрах набору даних на точність траєкторії. Точне значення параметрів можете побачити на рисунку 3.36.

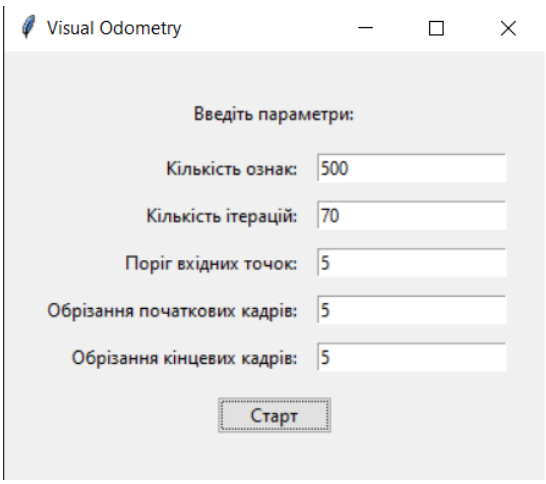

Рисунок 3.36 – Параметри для тесту з обрізанням перших 5 та останніх 5 кадрів

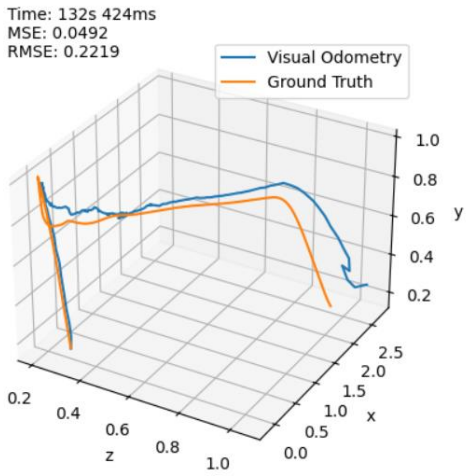

Рисунок 3.37 – Результати для тесту з обрізанням перших 5 та останніх 5 кадрів

Аналіз рисунку 3.37 показує нам найкращі результати MSE та RMSE серед усіх тестів, і як можна помітити зменшення кількості кадрів зменшило час виконання часу програми.

Наступний тест покаже вплив шуму у перших та останніх кадрах набору даних на точність траєкторії при малих параметрах. Точне значення параметрів можете побачити на рисунку 3.38.

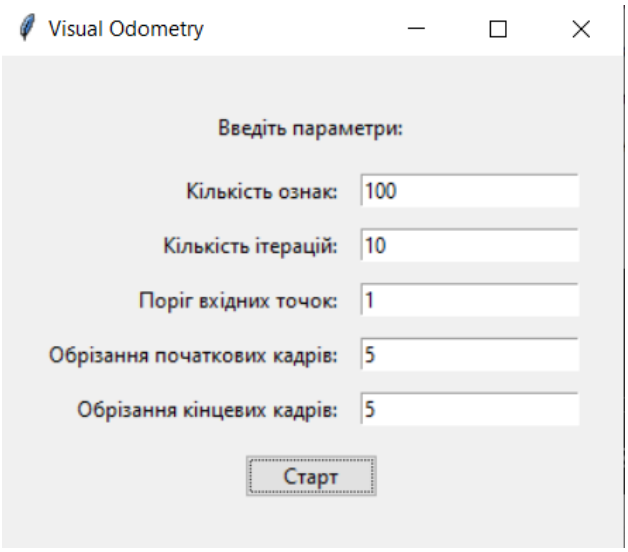

Рисунок 3.38 – Параметри для тесту з обрізанням перших 5 та останніх 5 кадрів та малими параметрами

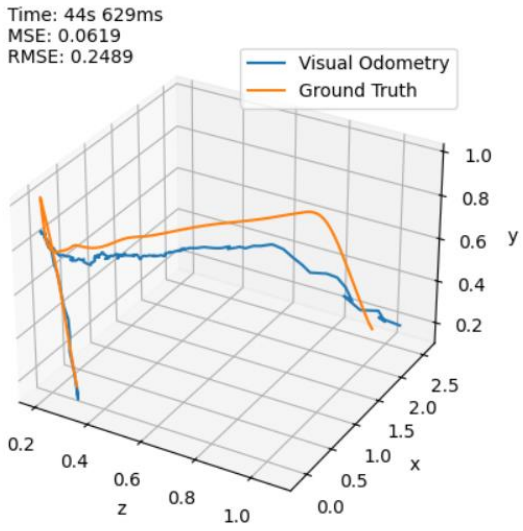

Рисунок 3.39 – Результати для тесту з обрізанням перших 5 та останніх 5 кадрів та малими параметрами

Аналіз рисунку 3.39 показує нам дуже гарні результати MSE та RMSE, і найкращій час виконання програми через зменшення кількості кадрів.

Наступний тест покаже вплив шуму у перших та останніх кадрах набору даних на точність траєкторії при великих параметрах. Точне значення параметрів можете побачити на рисунку 3.40.

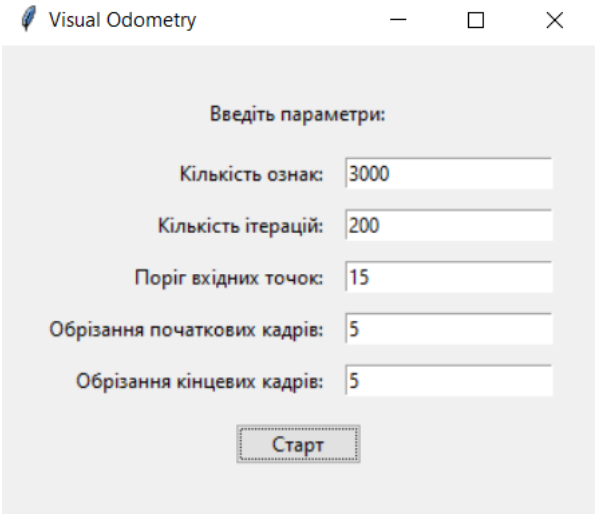

Рисунок 3.40 – Параметри для тесту з обрізанням перших 5 та останніх 5 кадрів та великими параметрами

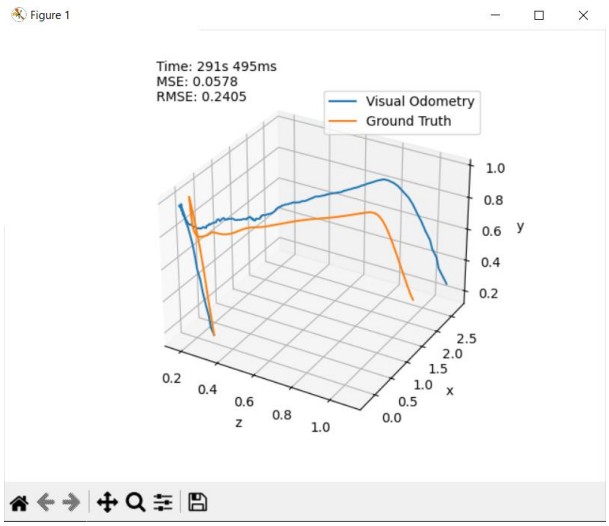

Рисунок 3.41 – Результати для тесту з обрізанням перших 5 та останніх 5 кадрів та великими параметрами

Аналіз рисунку 3.41 показує нам дуже гарні результати MSE та RMSE, і але час роботи програми не все ще дуже великий.

Таблиця 3.12 – Результати експерименту 7

Кількість ознак	Кількість ітерацій	Поріг вхідних точок	Обрізання початкових кадрів	Обрізання кінцевих кадрів	MSE	RMSE	Час роботи програми
500	70	5	1	1	0,0695	0,2636	139,22 с.
100	10	1	1	1	0,0677	0,2603	45,888 с.
3000	200	10	1	1	0,1486	0,3855	310,198 с.
500	70	5	5	5	0,0492	0,2219	132,424 с.
100	10	1	5	5	0,0619	0,2489	44,629 с.
3000	200	10	5	5	0,0578	0,2405	291,495 с.

Експеримент показав, що перші та останні кадри мають у собі шум і позбуття цього шуму покращує точність траєкторії (таблиця 3.12).

Повний список тестів з результатами можна побачити у таблиці 3.13.

Таблиця 3.13 – Загальна таблиця тестів

Кількість ознак	Кількість ітерацій	Поріг вхідних точок	Обрізання початкових кадрів	Обрізання кінцевих кадрів	MSE	RMSE	Час роботи програми
Експеримент 1: Зміна кількості ознак							
500	70	5	1	5	0,057	0,2388	147,56 с.
100	70	5	1	5	0,1506	0,388	72,496 с.
1000	70	5	1	5	0,0763	0,2762	141,547 с.
Експеримент 2: Зміна кількості ітерацій							
500	70	5	1	5	0,0787	0,2805	137,215 с.
500	50	5	1	5	0,0673	0,2595	114,482 с.
500	150	5	1	5	0,0894	0,299	244,268 с.
Експеримент 3: Зміна кількості порогу вхідних точок							
500	70	5	1	5	0,0755	0,2748	150,470 с.
500	50	1	1	5	0,1404	0,3747	143,551 с.
500	150	10	1	5	0,1439	0,3793	137,252 с.
Експеримент 4-5: Робота програми при дуже низьких та дуже високих параметрах							
100	10	1	1	5	0,0851	0,2917	49,208 с.
3000	200	15	1	5	0,1105	0,3324	305,785 с.
Експеримент 6: Робота програми при змішаних параметрах							
100	200	15	1	5	0,2873	0,534	122,647 с.
3000	10	1	1	5	0,3674	0,6062	64,780 с.
Експеримент 7: Робота програми при обрізанні кадрів							
500	70	5	1	1	0,0695	0,2636	139,22 с.
100	10	1	1	1	0,0677	0,2603	45,888 с.
3000	200	10	1	1	0,1486	0,3855	310,198 с.
500	70	5	5	5	0,0492	0,2219	132,424 с.
100	10	1	5	5	0,0619	0,2489	44,629 с.
3000	200	10	5	5	0,0578	0,2405	291,495 с.

Список результатів які показують значення MSE від кращого до гіршого можна побачити у таблиці 3.14.

Таблиця 3.14 – Таблиця відфільтрована за найкращими значеннями MSE та RMSE

Кількість ознак	Кількість ітерацій	Поріг вхідних точок	Обрізання початкових кадрів	Обрізання кінцевих кадрів	MSE	RMSE	Час роботи програми
500	70	5	5	5	0,0492	0,2219	132,424 с.
500	70	5	1	5	0,057	0,2388	147,56 с.
3000	200	10	5	5	0,0578	0,2405	291,495 с.
100	10	1	5	5	0,0619	0,2489	44,629 с.
500	50	5	1	5	0,0673	0,2595	114,482 с.
100	10	1	1	1	0,0677	0,2603	45,888 с.
500	70	5	1	1	0,0695	0,2636	139,22 с.
500	70	5	1	5	0,0755	0,2748	150,470 с.
1000	70	5	1	5	0,0763	0,2762	141,547 с.
500	70	5	1	5	0,0787	0,2805	137,215 с.
100	10	1	1	5	0,0851	0,2917	49,208 с.
500	150	5	1	5	0,0894	0,299	244,268 с.
3000	200	15	1	5	0,1105	0,3324	305,785 с.
500	50	1	1	5	0,1404	0,3747	143,551 с.
500	150	10	1	5	0,1439	0,3793	137,252 с.
3000	200	10	1	1	0,1486	0,3855	310,198 с.
100	70	5	1	5	0,1506	0,388	72,496 с.
100	200	15	1	5	0,2873	0,534	122,647 с.
3000	10	1	1	5	0,3674	0,6062	64,780 с.

Список результатів які показують значення часу роботи програми від кращого до гіршого можна побачити у таблиці 3.15.

Таблиця 3.15 – Таблиця відфільтрована за найкращими значеннями часу роботи

Кількість ознак	Кількість ітерацій	Поріг вхідних точок	Обрізання початкових кадрів	Обрізання кінцевих кадрів	MSE	RMSE	Час роботи програми
100	10	1	5	5	0,0619	0,2489	44,629
100	10	1	1	1	0,0677	0,2603	45,888
100	10	1	1	5	0,0851	0,2917	49,208
3000	10	1	1	5	0,3674	0,6062	64,78
100	70	5	1	5	0,1506	0,388	72,496
500	50	5	1	5	0,0673	0,2595	114,482
100	200	15	1	5	0,2873	0,534	122,647
500	70	5	5	5	0,0492	0,2219	132,424
500	70	5	1	5	0,0787	0,2805	137,215
500	150	10	1	5	0,1439	0,3793	137,252
500	70	5	1	1	0,0695	0,2636	139,22
1000	70	5	1	5	0,0763	0,2762	141,547
500	50	1	1	5	0,1404	0,3747	143,551
500	70	5	1	5	0,057	0,2388	147,56
500	70	5	1	5	0,0755	0,2748	150,47
500	150	5	1	5	0,0894	0,299	244,268
3000	200	10	5	5	0,0578	0,2405	291,495
3000	200	15	1	5	0,1105	0,3324	305,785
3000	200	10	1	1	0,1486	0,3855	310,198

ВИСНОВОК

У магістерській кваліфікаційній роботі було досліджено проблему навігації малогабаритних безпілотних літальних апаратів в умовах обмеженого доступу до традиційних навігаційних систем, таких як GPS. Проведено огляд сучасних технологій навігації, акцентуючи увагу на викликах, пов’язаних із забезпеченням точності, адаптивності до змінних умов експлуатації та економічної ефективності.

Особливу увагу приділено методам візуальної одометрії, які передбачають використання монокулярного та стереоскопічного бачення для оцінки траєкторії руху, а також комбінування цих методів із IMU задля покращення стабільності та точності алгоритмів. Виявлено перспективність використання алгоритму Лукас-Канаде для обчислення оптичного потоку як основи для аналізу переміщення ключових точок між кадрами.

Розроблено інформаційну технологію для налаштування параметрів візуальної одометрії, яка дозволяє проводити серію експериментів із варіацією ключових параметрів і оцінювати вплив цих змін на точність побудови траєкторії. Проведені експерименти підтвердили ефективність запропонованих алгоритмів у моделюванні траєкторії камери, проте виявили також обмеження, пов’язані із чутливістю до шумів.

Результати досліджень створюють основу для подальших робіт, спрямованих на оптимізацію алгоритмів для апаратних платформ із обмеженими ресурсами, інтеграцію з реальними навігаційними системами та розширення функціональних можливостей розробленого програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Different Types of Drones and Uses (2024 Full Guide). *JOUAV*. Режим доступу: https://www.jouav.com/blog/drone-types.html (дата звернення: 26.09.2024).
2. Arafat, M. Y., Alam, M. M., & Moh, S. (2023). Vision-Based Navigation Techniques for Unmanned Aerial Vehicles: Review and Challenges. B Drones (Вип. 7, Issue 2, с. 89). MDPI AG. Doi: https://doi.org/10.3390/drones7020089. Режим доступу: https://www.mdpi.com/2504-446X/7/2/89 (дата звернення: 26.09.2024).
3. Ahmed, F., Mohanta, J. C., Keshari, A., & Yadav, P. S. (2022). Recent Advances in Unmanned Aerial Vehicles: A Review. B Arabian Journal for Science and Engineering (Вип. 47, Issue 7, с. 7963–7984). Springer Science and Business Media LLC. Doi: https://doi.org/10.1007/s13369-022-06738-0. Режим доступу: https://link.springer.com/article/10.1007/s13369-022-06738-0 (дата звернення: 20.10.2024).
4. А.В. Рудик. Методи оцінки просторового положення об’єктів. *Nuwm.edu*. Режим доступу: https://cutt.ly/4eHVhZ6H (дата звернення: 28.09.2024).
5. Oliver Öhrstam Lindström. O. Deep Monocular Visual Odometry for fixed-winged Aircraft. С. 8-10. *Diva-portal*. Режим доступу: https://www.diva-portal.org/smash/get/diva2:1750631/FULLTEXT01.pdf (дата звернення: 20.10.2024).
6. Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: part I. B IEEE Robotics & Automation Magazine (Вип. 13, Issue 2, с. 99–110). Institute of Electrical and Electronics Engineers (IEEE). Doi: https://doi.org/10.1109/mra.2006.1638022. Режим доступу: https://ieeexplore.ieee.org/document/1638022 (дата звернення: 26.09.2024).
7. Zhuang, L., Zhong, X., Xu, L., Tian, C., & Yu, W. (2024). Visual SLAM for Unmanned Aerial Vehicles: Localization and Perception. B Sensors (Вип. 24,

Issue 10, c. 2980). MDPI AG. Doi: https://doi.org/10.3390/s24102980. Режим доступу: https://www.mdpi.com/1424-8220/24/10/2980 (дата звернення: 29.10.2024).

8. Бондар Д.В. Застосування безпілотних авіаційних систем у сфері цивільного захисту: монографія / Д.В. Бондар, А.В. Гурник, А.О. Литовченко, В.В. Хижняк, В.Л. Шевченко, Д.М. Ядченко. Київ, 2022, 312 с. / ISBN 978-617-8015-16-9. Режим доступу: https://idundcz.dsns.gov.ua/upload/1/0/7/9/0/0/2/ljQ2PLY10IvuTptkblpjCibdMBdVh3TutyE5LM9Z.pdf (дата звернення: 30.09.2024).
9. Kristen D. Thompson. How the Drone War in Ukraine Is Transforming Conflict. *Council on Foreign Relations*. Режим доступу: https://www.cfr.org/article/how-drone-war-ukraine-transforming-conflict (дата звернення: 22.10.2024).
10. Kulbacki, M., Segen, J., Chaczko, Z., Rozenblit, J. W., Kulbacki, M., Klempous, R., & Wojciechowski, K. (2023). Intelligent Video Analytics for Human Action Recognition: The State of Knowledge. B Sensors (Вип. 23, Issue 9, c. 4258). MDPI AG. Doi:https://doi.org/10.3390/s23094258. Режим доступу: https://www.mdpi.com/1424-8220/23/9/4258 (дата звернення: 22.10.2024).
11. Sreenu, G., & Saleem Durai, M. A. (2019). Intelligent video surveillance: a review through deep learning techniques for crowd analysis. B Journal of Big Data (Вип. 6, Issue 1). Springer Science and Business Media LLC. https://doi.org/10.1186/s40537-019-0212-5. Режим доступу: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0212-5 (дата звернення: 28.09.2024).
12. Contributors to Wikimedia projects. Lucas–Kanade method - Wikipedia. *Wikipedia, the free encyclopedia*. Режим доступу: https://en.wikipedia.org/wiki/Lucas–Kanade_method (дата звернення: 30.10.2024).

13. R. Fisher, S. Perkins, A. Walker and E. Wolfart. Spatial Filters - Gaussian Smoothing. Informatics Homepages Server. Режим доступу: https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm (дата звернення: 30.10.2024).

ДОДАТКИ

ДОДАТОК А. ФАЙЛ GUI.PY

```
import tkinter as tk
import cv2
from scipy.spatial.transform import Rotation
from tkinter import ttk, messagebox
import numpy as np
import queue
import threading
import matplotlib.pyplot as plt
import visual_odometry as visual_odometry
from mpl_toolkits.mplot3d import Axes3D
import time

class VOApp:
    def __init__(self, root):
        """
        Ініціалізація кореневого вікна додатку.

        Параметри:
        root (Tk): Кореневий елемент вікна додатку.
        """
        self.root = root
        self.root.title("Visual Odometry")

        # Початкові значення параметрів
        self.initial_param1 = 500  # кількість ознак
        self.initial_param2 = 70   # кількість ітерацій
        self.initial_param3 = 5    # поріг вхідних точок
```

```
        self.initial_param4 = 1    # обрізання початкових кадрів
        self.initial_param5 = 5    # обрізання кінцевих кадрів
         self.optical_flow_window_size = 5  # Розмір вікна для оптичного потоку
        self.pyramid_level = 5  # Рівень піраміди

        # Створення елементів інтерфейсу
        self.setup_ui()

    def setup_ui(self):
        """
         Створює графічний інтерфейс користувача для налаштування параметрів візуальної одометрії.
        Цей метод ініціалізує фрейм для введення параметрів, створює мітки та поля вводу для кожного
         параметра, включаючи кількість ознак, кількість ітерацій, поріг вхідних точок,
        обрізання початкових кадрів та обрізання кінцевих кадрів.
         Також додається кнопка "Старт" для запуску процесу візуальної одометрії.
        """
        # Створення фрейму для вибору параметрів
        self.control_frame = ttk.Frame(self.root)
            self.control_frame.pack(side="top", padx=20, pady=20, fill='y', expand=True)

        # Мітка для параметрів введення
             self.param_label = tk.Label(self.control_frame, text="Введіть параметри:")
        self.param_label.grid(row=0, column=0, pady=10, columnspan=2)

        # Створення полів для вводу параметрів
```

```
        self.param1_label = tk.Label(self.control_frame, text="Кількість ознак:")
        self.param1_label.grid(row=1, column=0, padx=5, pady=5, sticky="e")
        self.param1_entry = tk.Entry(self.control_frame)
        self.param1_entry.insert(0, self.initial_param1)  # Початкове значення
        self.param1_entry.grid(row=1, column=1, padx=5, pady=5, sticky="w")

        self.param2_label = tk.Label(self.control_frame, text="Кількість ітерацій:")
        self.param2_label.grid(row=2, column=0, padx=5, pady=5, sticky="e")
        self.param2_entry = tk.Entry(self.control_frame)
        self.param2_entry.insert(0, self.initial_param2)
        self.param2_entry.grid(row=2, column=1, padx=5, pady=5, sticky="w")

        self.param3_label = tk.Label(self.control_frame, text="Поріг вхідних точок:")
        self.param3_label.grid(row=3, column=0, padx=5, pady=5, sticky="e")
        self.param3_entry = tk.Entry(self.control_frame)
        self.param3_entry.insert(0, self.initial_param3)
        self.param3_entry.grid(row=3, column=1, padx=5, pady=5, sticky="w")

        self.param4_label = tk.Label(self.control_frame, text="Обрізання початкових кадрів:")
        self.param4_label.grid(row=4, column=0, padx=5, pady=5, sticky="e")
        self.param4_entry = tk.Entry(self.control_frame)
        self.param4_entry.insert(0, self.initial_param4)
        self.param4_entry.grid(row=4, column=1, padx=5, pady=5, sticky="w")

        self.param5_label = tk.Label(self.control_frame, text="Обрізання кінцевих кадрів:")
        self.param5_label.grid(row=5, column=0, padx=5, pady=5, sticky="e")
```

```
        self.param5_entry = tk.Entry(self.control_frame)
        self.param5_entry.insert(0, self.initial_param5)
        self.param5_entry.grid(row=5, column=1, padx=5, pady=5, sticky="w")

        # Створення кнопки "Старт"
        self.start_button = ttk.Button(self.control_frame, text="Старт", command=self.start_vo)
        self.start_button.grid(row=6, column=0, columnspan=2, pady=10)

        # Ініціалізація змінних для потоку
        self.vo_thread = None
        self.folder_path = "dataset"  # Шлях до папки з даними
        self.queue = queue.Queue()  # Черга для обміну даними між потоками

    def start_vo(self):
        """
        Запускає процес візуальної одометрії у новому потоці, використовуючи параметри, введені користувачем.

        Цей метод перевіряє, чи вже працює потік візуальної одометрії. Якщо потік не працює, то він ініціалізує
        необхідні параметри, такі як кількість ознак, розмір вікна для оптичного потоку, рівень піраміди, кількість
        ітерацій,поріг вхідних точок обрізання початкових кадрів та обрізання кінцевих кадрів,
        отримані з полів вводу користувача. Потім він створює новий потік, в якому
        виконується метод `run_vo`, передаючи йому ці параметри разом зі шляхом до даних і списками для збереження результатів.
        """
        try:
            # Кількість ознак
```

```
                param1 = int(self.param1_entry.get())
                if param1 <= 0:
                    raise ValueError("Кількість ознак повинна бути додатнім цілим числом.")

                # Кількість ітерацій
                param2 = int(self.param2_entry.get())
                if param2 <= 0:
                    raise ValueError("Кількість ітерацій повинна бути додатнім цілим числом.")

                # Поріг вхідних точок
                param3 = float(self.param3_entry.get())
                if param3 <= 0:
                    raise ValueError("Поріг вхідних точок повинен бути додатнім числом.")

                # Обрізання початкових кадрів
                param4 = int(self.param4_entry.get())
                if param4 < 0:
                    raise ValueError("Обрізання початкових кадрів повинно бути додатнім цілим числом.")

                # Обрізання кінцевих кадрів
                param5 = int(self.param5_entry.get())
                if param5 < 0:
                    raise ValueError("Обрізання кінцевих кадрів повинно бути додатнім цілим числом.")

            except ValueError as e:
                messagebox.showerror("Помилка вводу", str(e))
```

```
            return None

        # Перевірка, чи потік вже не працює
        if self.vo_thread is None or not self.vo_thread.is_alive():
            ground_truth = []  # Список для істинних значень
            monocular_vo = []  # Список для результатів візуальної одометрії

            # Запуск візуальної одометрії в окремому потоці з параметрами
            self.vo_thread = threading.Thread(target=self.run_vo, args=(
                ground_truth, monocular_vo, param1,
                self.optical_flow_window_size, self.pyramid_level,
                param2, param3, param4, param5, self.folder_path
            ))
            self.vo_thread.start()

    def compute_mse(self, ground_truth, estimated):
        """
        Обчислює середньоквадратичну помилку (MSE) та її корінь (RMSE) між істинними
        та оціненими траєкторіями.

        Параметри:
        - ground_truth (numpy.ndarray): Массив із 3D-координатами істинної траєкторії.
        - estimated (numpy.ndarray): Массив із 3D-координатами оціненої траєкторії.

        Повертає:
        - mse (float): Середньоквадратична помилка.
        - rmse (float): Корінь із середньоквадратичної помилки.
        """
```

```
        mse = 0.0
        N = len(ground_truth)

        # Ітерація по всіх кадрах
        for i in range(N):
            # Обчислення MSE для кожного кадру
                x_gt, y_gt, z_gt = ground_truth[i][0], ground_truth[i][1], ground_truth[i][2]
            x_est, y_est, z_est = estimated[i][0], estimated[i][1], estimated[i][2]

            frame_mse = (x_gt - x_est) ** 2 + (y_gt - y_est) ** 2 + (z_gt - z_est) ** 2
            mse += frame_mse

        mse /= N
        rmse = np.sqrt(mse)

        return mse, rmse

        def run_vo(self, ground_truth, monocular_vo, param1, optical_flow_window_size, pyramid_level, param2, param3, param4, param5, folder_path):
        """
        Виконує основний процес візуальної одометрії, обробляючи результати та візуалізуючи порівняння шляху візуальної одометрії
        з істинними даними та виводить час роботи, mse та rmse.

        Цей метод викликає основну функцію візуальної одометрії, передаючи їй необхідні параметри, включаючи список істинних значень
        та результатів візуальної одометрії. Потім відбувається обробка результатів, включаючи ротацію шляху візуальної одометрії
```

для вирівнювання з істинними даними, застосування корекції через матрицю обертання, та візуалізація результатів на 3D графіку.

Параметри:

- ground_truth (list): Список істинних значень 3D координат (шлях).

- monocular_vo (list): Список результатів візуальної одометрії (шлях).

- param1 (int): Кількість ознак.

- param2 (int): Кількість ітерацій.

- param3 (float): Поріг вхідних точок.

- param4 (int): Обрізання початкових кадрів.

- param5 (int): Обрізання кінцевих кадрів.

- folder_path (str): Шлях до папки з даними.

"""

Початок вимірювання часу

start_time = time.time()

visual_odometry.main(self.queue, folder_path, ground_truth, monocular_vo, param1, optical_flow_window_size, pyramid_level, param2, param3, param4, param5)

Завершення вимірювання часу

end_time = time.time()

Обчислення часу виконання в секундах та мілісекундах

elapsed_time = end_time - start_time

seconds = int(elapsed_time)

milliseconds = int((elapsed_time - seconds) * 1000)

cv2.destroyAllWindows()

```
gt_3d = np.array(ground_truth)
gt_path_3d = gt_3d[:, 3:]
gt_rot_vec = gt_3d[:, :3]
estimated_3d = np.array(monocular_vo)
estimated_path_3d = estimated_3d[:, 3:]
estimated_rot_vec = estimated_3d[:, :3]
vo_to_rotate = np.copy(estimated_path_3d - gt_path_3d[0])

# Ротація шляху візуальної одометрії для вирівнювання з істинними даними
rotate_matrix = []
for i in range(len(estimated_rot_vec)):
    R, _ = cv2.Rodrigues(estimated_rot_vec[i] - gt_rot_vec[0])
    rotate_matrix.append(R)
rotate_matrix = np.array(rotate_matrix)

# Оборотна матриця для корекції
    hec_rotation_matrix = Rotation.from_euler('xyz', [0, 190, 170], degrees=True)
hec_rotation_matrix = hec_rotation_matrix.as_matrix()
vo_to_rotate = vo_to_rotate[:, :, np.newaxis]
vo_rotated = hec_rotation_matrix @ vo_to_rotate
estimated_path_3d = np.squeeze(vo_rotated, axis=2)
estimated_path_3d = np.copy(estimated_path_3d + gt_path_3d[0])

mse, rmse = self.compute_mse(gt_path_3d, estimated_path_3d)

# Візуалізація результатів на 3D графіку
ax = plt.figure().add_subplot(111, projection='3d')
        ax.plot(estimated_path_3d[:, 2], estimated_path_3d[:, 0], estimated_path_3d[:, 1], label='Visual Odometry')
```

```
                ax.plot(gt_path_3d[:, 2], gt_path_3d[:, 0], gt_path_3d[:, 1], label='Ground Truth')
                ax.text2D(0.05, 0.95, f"Time: {seconds}s {milliseconds}ms\nMSE: {mse:.4f}\nRMSE: {rmse:.4f}", transform=ax.transAxes)
            ax.legend()
            ax.set_xlabel("z")
            ax.set_ylabel("x")
            ax.set_zlabel("y")
            plt.show()


    if __name__ == "__main__":
        # Ініціалізація головного вікна та запуск додатку
        root = tk.Tk()
        app = VOApp(root)
        root.mainloop()
```

ДОДАТОК Б. ФАЙЛ VISUAL_ODOMETRY.PY

```
import os
import numpy as np
import cv2
from tqdm import tqdm
from lib.lucas_kanade_flow import lucas_kanade_optical_flow
from lib.imu_integration import load_optitrak_data


class VisualOdometry:
    def __init__(self, data_directory):
        """
        Ініціалізує об'єкт VisualOdometry та завантажує необхідні дані з вказаного каталогу.

        Параметри:
        data_directory (str): Шлях до каталогу, що містить файли для калібрування, ground truth, IMU та зображення.
        """
                                self.intrinsic_matrix, self.projection_matrix = self.load_calibration(os.path.join(data_directory, "calib.txt"))
                                self.ground_truth_poses, self.imu_distances = self.load_ground_truth_and_imu(
            [os.path.join(data_directory, file) for file in os.listdir(data_directory) if file.startswith("ground_truth")],
            [os.path.join(data_directory, file) for file in os.listdir(data_directory) if file.startswith("imu")]
        )
            self.image_sequences = self.load_images(os.path.join(data_directory, "images"))

    @staticmethod
```

```
def load_calibration(filepath):
    """
    Завантажує калібрувальні параметри з вказаного файлу.

    Параметри:
    filepath (str): Шлях до файлу, що містить калібрувальні параметри.

    Повертає:
    tuple: Внутрішню матрицю та проекційну матрицю.
    """
    with open(filepath, 'r') as file:
        params = np.fromstring(file.readline(), dtype=np.float64, sep=' ')
        projection_matrix = np.reshape(params, (3, 4))
        intrinsic_matrix = projection_matrix[0:3, 0:3]
    return intrinsic_matrix, projection_matrix

@staticmethod
def load_ground_truth_and_imu(filepath, timestamp):
    """
    Завантажує дані про позу та відстані з файлів OptiTrak та IMU.

    Параметри:
    filepath (list): Список шляхів до файлів з даними про позу.
    timestamp (list): Список шляхів до файлів з часовими мітками.

    Повертає:
    tuple: Трансформаційні матриці поз та масив відстаней.
    """
    poses, distances = load_optitrak_data(filepath[0], timestamp[0])
    return poses, distances
```

```
    @staticmethod
    def load_images(filepath):
        """
        Завантажує зображення з вказаного каталогу.

        Параметри:
        filepath (str): Шлях до каталогу, який містить зображення.

        Повертає:
        list: Список зображень, завантажених за допомогою OpenCV.
        """
        image_paths = [os.path.join(filepath, file) for file in sorted(os.listdir(filepath))]
        return [cv2.imread(path) for path in image_paths]

    @staticmethod
    def create_transformation_matrix(R, t):
        """
        Створює матрицю трансформації на основі матриці обертання та вектора переміщення.

        Параметри:
        R (ndarray): Матриця обертання розміру (3, 3).
        t (ndarray): Вектор переміщення розміру (3,).

        Повертає:
            ndarray: 4x4 матриця трансформації, яка комбінує обертання та переміщення.
        """
        transformation_matrix = np.eye(4, dtype=np.float64)
```

```
        transformation_matrix[:3, :3] = R
        transformation_matrix[:3, 3] = t
        return transformation_matrix


    def get_feature_matches(self, index, step, num_features=500, window_size=5, pyramid_level=5, num_iterations=70, inlier_threshold=3):
        """
        Знаходить відповідні точки між двома зображеннями за допомогою методу оптичного потоку Лукас-Канаде.

        Параметри:
        index (int): Поточний індекс зображення в послідовності.
        step (int): Крок між поточним і наступним зображенням для порівняння.
        num_features (int): Кількість характеристик для відслідковування.
        window_size (int): Розмір вікна для алгоритму Лукас-Канаде.
        pyramid_level (int): Кількість рівнів віртуальної піраміди.
        num_iterations (int): Кількість ітерацій для оцінки оптичного потоку.
        inlier_threshold (float): Поріг для визначення коректних точок.

        Повертає:
        tuple: Два масиви точок (q1, q2) з координатами відповідних точок на двох зображеннях.
        """
        q1, q2, focus_of_expansion = lucas_kanade_optical_flow(
            self.image_sequences[index - step],
            self.image_sequences[index + 1 - step],
            num_features,
            window_size,
            pyramid_level,
            num_iterations,
            inlier_threshold,
```

```
        )
        image = self.image_sequences[index - step].copy()
        self.plot_optical_flow(image, q1, q2, focus_of_expansion)
        return q1, q2

    def compute_pose(self, q1, q2):
        """
            Обчислює матрицю обертання та вектор переміщення між двома зображеннями
        за допомогою есеціальної матриці.

        Параметри:
        q1 (array): Масив координат точок на першому зображенні.
        q2 (array): Масив координат точок на другому зображенні.

        Повертає:
        tuple: Матриця обертання (R) та вектор переміщення (t).
        """
        if q1 is None or q2 is None or len(q1) == 0 or len(q2) == 0:
            print("Error: One or both point sets are empty.")
            return None, None
            essential_matrix, _ = cv2.findEssentialMat(q1, q2, self.intrinsic_matrix, method=0, threshold=0.1)
        if essential_matrix is None or essential_matrix.size == 0:
            print("Error: Essential matrix is None or empty. Terminating the process.")
            return None, None
        R, t = self.decompose_essential_matrix(essential_matrix, q1, q2)
        return R, t

    def compute_positive_z_count(self, R, t, q1, q2):
```

```
"""
```

Обчислює кількість точок, що знаходяться перед камерою (мають додатну координату Z)

у двох системах координат: до і після трансформації.

Параметри:

R (array): Матриця обертання.

t (array): Вектор переміщення.

q1 (array): Масив координат точок на першому зображенні.

q2 (array): Масив координат точок на другому зображенні.

Повертає:

int: Сума кількості точок із додатною координатою Z у двох системах координат.

```
"""
transformation_matrix = self.create_transformation_matrix(R, t)

# Формування проєкційної матриці для другої камери
projection_matrix = np.matmul(
    np.concatenate((self.intrinsic_matrix, np.zeros((3, 1))), axis=1),
    transformation_matrix
)

# Триангуляція точок
homogeneous_Q1 = cv2.triangulatePoints(
    np.float32(self.projection_matrix),
    np.float32(projection_matrix),
    np.float32(q1.T),
    np.float32(q2.T)
```

```
        )

        # Перетворення точок до другої системи координат
        homogeneous_Q2 = np.matmul(transformation_matrix, homogeneous_Q1)

        # Нормалізація гомогенних координат
        un_homogeneous_Q1 = homogeneous_Q1[:3, :] / homogeneous_Q1[3, :]
        un_homogeneous_Q2 = homogeneous_Q2[:3, :] / homogeneous_Q2[3, :]

        # Підрахунок точок із додатною Z-координатою
        count_positive_z_Q1 = np.sum(un_homogeneous_Q1[2, :] > 0)
        count_positive_z_Q2 = np.sum(un_homogeneous_Q2[2, :] > 0)

        return count_positive_z_Q1 + count_positive_z_Q2

    def decompose_essential_matrix(self, essential_matrix, q1, q2):
        """
            Розкладає есенціальну матрицю на матриці обертання та вектор переміщення.

        Параметри:
        essential_matrix (array): Есенціальна матриця.
        q1 (array): Масив координат точок на першому зображенні.
        q2 (array): Масив координат точок на другому зображенні.

        Повертає:
        tuple:
            - Найкраща матриця обертання (R).
            - Найкращий вектор переміщення (t).
        """
```

```
        try:
            # Перевірка на коректність essential_matrix перед розкладом
            if essential_matrix is None or essential_matrix.size == 0:
                raise ValueError("Essential matrix is None or empty")

            R1, R2, t = cv2.decomposeEssentialMat(essential_matrix)
            t = np.squeeze(t)

            pairs = [[R1, -t], [R1, t], [R2, t], [R2, -t]]

            positive_z_counts = [self.compute_positive_z_count(R, t, q1, q2) for R, t in
pairs]

            best_pair_index = np.argmax(positive_z_counts)
            best_R, best_t = pairs[best_pair_index]

            return best_R, best_t
        except cv2.error as e:
            print(f"OpenCV error during essential matrix decomposition: {e}")
            return None, None
        except ValueError as e:
            print(f"ValueError: {e}")
            return None, None

    def plot_optical_flow(self, image, points1, points2, focus_of_expansion):
        """
        Візуалізує оптичний потік між двома наборами точок на зображенні у
вигляді стрілок та фокусу розширення.

        Параметри:
```

image (array): Зображення, на якому відображатиметься оптичний потік.

points1 (array): Масив точок на першому зображенні.

points2 (array): Масив точок на другому зображенні.

focus_of_expansion (tuple): Координати фокусу розширення (точка, де потік рухається).

"""

features = np.intp(points1)

depths = np.intp(points2)

for i in range(len(depths)):

cv2.arrowedLine(image, (features[i, 0], features[i, 1]), (depths[i, 0], depths[i, 1]), [255, 150, 0], 1, tipLength=0.2)

cv2.circle(image, (np.intp(focus_of_expansion[0]), np.intp(focus_of_expansion[1])), 7, (0, 0, 255), -1)

cv2.imshow('Optical Flow Frame', image)

cv2.waitKey(10)

def main(queued_data, data_directory, gt_path_3d, estimated_path_3d, num_features=500, window_size=5, pyramid_level=5, num_iterations=70, inlier_threshold=3, start_frame=1, end_frame=5):

"""

Головна функція для виконання візуальної одометрії.

Функція ініціалізує об'єкт візуальної одометрії, обробляє кадри з даними про позу та IMU, отримує відповідні точки за допомогою оптичного потоку Лукас-Канаде,

а також обчислює трансформаційні матриці для кожного кадру. Крім того, зберігаються та оновлюються орієнтації і пози для оціненого та істинного шляху.

Параметри:

queued_data (queue): Черга для обміну даними між процесами.

data_directory (str): Шлях до каталогу з вхідними даними.

gt_path_3d (list): Список для зберігання оціненого 3D шляху.

estimated_path_3d (list): Список для зберігання оцінених 3D шляхів.

num_features (int): Кількість особливостей для пошуку (за замовчуванням 200).

window_size (int): Розмір вікна для оптичного потоку (за замовчуванням 5).

pyramid_level (int): Кількість рівнів піраміди для оптичного потоку (за замовчуванням 5).

num_iterations (int): Кількість ітерацій для оптичного потоку (за замовчуванням 70).

inlier_threshold (int): Поріг для верифікації інлайнів (за замовчуванням 3).

is_static (bool): Прапорець для визначення, чи є сцена статичною (за замовчуванням False).

"""

```
# Ініціалізація об'єкта VisualOdometry
vo = VisualOdometry(data_directory)
imu_distances = vo.imu_distances
ground_truth_poses = vo.ground_truth_poses
current_poses = np.copy(ground_truth_poses)
step = 1


current_poses = np.array(current_poses)[start_frame:-(end_frame):step, :, :]


estimated_rotations = []
ground_truth_rotations = []
current_pose = current_poses[0]


# Проходимо через усі кадри
for i, gt_pose in zip(range(start_frame, len(current_poses) - end_frame, step), tqdm(current_poses, unit=" Frames", smoothing=0)):
    shared_data = [i - start_frame, len(current_poses), 1]
    queued_data.put(shared_data)
```

```
        if i > start_frame:
            # Отримуємо відповідні точки для оптичного потоку
            q1, q2 = vo.get_feature_matches(i, step, num_features, window_size, pyramid_level, num_iterations, inlier_threshold)
            R, t = vo.compute_pose(q1, q2)
            if R is None or t is None:
                print(f"Помилка: Неможливо обчислити позу для кадру {i}. Завершення процесу.")
                break
            t *= imu_distances[i]
            transformation = vo.create_transformation_matrix(R, np.squeeze(t))
            current_pose = np.matmul(current_pose, np.linalg.inv(transformation))

        # Зберігаємо обертання та пози
        gt_rotation, _ = cv2.Rodrigues(gt_pose[:3, :3])
        estimated_rotation, _ = cv2.Rodrigues(current_pose[:3, :3])
        ground_truth_rotations.append(gt_rotation)
        estimated_rotations.append(estimated_rotation)
        estimated_path_3d.append((estimated_rotation[0][0], estimated_rotation[1][0], estimated_rotation[2][0], current_pose[0, 3], current_pose[1, 3], current_pose[2, 3]))
        gt_path_3d.append((gt_rotation[0][0], gt_rotation[1][0], gt_rotation[2][0], gt_pose[0, 3], gt_pose[1, 3], gt_pose[2, 3]))

if __name__ == "__main__":
    main()
```

ДОДАТОК В. ФАЙЛ IMU_INTEGRATION.PY

```
import csv
import numpy as np
import cv2
from scipy.integrate import cumulative_trapezoid


def load_optitrak_data(filename, timestamps):
    """
    Завантажує дані з файлів OptiTrak і обробляє їх для створення трансформаційних матриць.

    Параметри:
    filename (str): Шлях до файлу з даними OptiTrak.
    timestamps (str): Шлях до файлу з часовими мітками.

    Повертає:
    tuple: Список трансформаційних матриць і масив відстаней.
    """
    timestamp = []

    # Завантаження часових міток з файлу
    with open(timestamps, newline='\n', encoding='utf-8') as csvfile:
        file = csv.reader(csvfile, delimiter=',', quotechar='|')
        k = 0
        for time_stamp in file:
            if k > 0:
                timestamp.append((time_stamp[1], time_stamp[2], time_stamp[6], time_stamp[7],
                                  time_stamp[8], time_stamp[9], time_stamp[10], time_stamp[11], time_stamp[12]))
```

```
            k += 1
    timestamp = np.array(timestamp, dtype=np.float64)
    timestamp[:,-2] += 981  # Коригуємо координати
    poses = []
    j = 0
    i = 0

    # Завантажуємо дані позицій з файлу
    with open(filename, newline='\n', encoding='utf-8') as csvfile:
        rec = csv.reader(csvfile, delimiter=',', quotechar='|')
        for row in rec:
            if i >= 7 and j < len(timestamp):
                # Пошук найближчої часової мітки
                if abs(float(row[1]) - float(timestamp[j, 0])) < 0.01:
                    if '' not in row[2:8]:  # Перевірка на порожні значення
                        row = np.array(row[2:8], dtype=np.float32)
                        x, y, z, t_x, t_y, t_z = row

                        # Перетворення даних на радіани для обертання
                        optitrak_rotation = np.array([np.radians(x), np.radians(y),
np.radians(z)])
                        optitrak_translation_vector = [t_x, t_y, t_z]

                        # Обчислення матриці обертання і вектору трансляції
                        rotation_matrix_opencv, _ = cv2.Rodrigues(optitrak_rotation)
                        translation_vector_opencv = np.array(optitrak_translation_vector)

                        # Створення коригуючої матриці для орієнтації
                        R = np.array([[0,0,1], [0,1,0], [1,0,0]])
                        transformation_matrix_opencv = np.eye(4)
```

```
                transformation_matrix_opencv[:3, :3] = rotation_matrix_opencv
                transformation_matrix_opencv[:3, 3] = translation_vector_opencv
                        transformation_matrix_opencv[:3, :] = np.matmul(R, transformation_matrix_opencv[:3, :])

                # Додаємо отриману матрицю в список
                poses.append(transformation_matrix_opencv)

                j += 1
        i += 1  # Крок індексу рядка

    # Обчислення відстані
    dist = integrate((timestamp[:, 2:8]) / 100, timestamp[:, 0])

    return poses, dist

def integrate(a, t):
    """
     Обчислює відстань на основі прискорень і часових міток за допомогою чисельного інтегрування.

    Параметри:
    a (numpy.ndarray): Масив прискорень для кожної осі.
    t (numpy.ndarray): Часові мітки.

    Повертає:
    list: Список відстаней між точками.
    """
    # Розрахунок швидкості по осях X, Y, Z
    vx = cumulative_trapezoid(a[:, 3] + a[:, 4], t, initial=0)  # Швидкість по X
```

```
    vy = cumulative_trapezoid(a[:, 4] - a[:, 4], t, initial=0)  # Швидкість по Y
    vz = cumulative_trapezoid(-a[:, 5] - a[:, 4], t, initial=0) # Швидкість по Z

    # Розрахунок переміщення по кожній осі
    dx = cumulative_trapezoid(vx, t, initial=0)
    dy = cumulative_trapezoid(vy, t, initial=0)
    dz = cumulative_trapezoid(vz, t, initial=0)

    # Створення масиву швидкостей
    vo = np.zeros((len(dx), 3))
    vo[:, 0], vo[:, 1], vo[:, 2] = dx, dy, dz

    # Обчислення відстані між точками
    d_vo = []
    for i in range(1, len(vo)):
        d_vo.append(np.linalg.norm(vo[i] - vo[i - 1]))  # Евклідова відстань між точками

    return d_vo
```

ДОДАТОК Г. ФАЙЛ FOE_RANSAC.PY

```
import numpy as np

def estimate_foe(flow_vectors, inlier_threshold=3, num_iterations=100):
    """
    Оцінює точку сходження (FOE) за допомогою методу RANSAC для векторів потоку.

    Параметри:
    flow_vectors (list or ndarray): Список/масив векторів потоку (x, y).
    inlier_threshold (float): Поріг для визначення внутрішніх точок.
    num_iterations (int): Кількість ітерацій для виконання RANSAC.

    Повертає:
    best_foe (tuple): Краща оцінка точки сходження (x, y).
    best_k (list): Список індексів внутрішніх точок для кращої моделі.
    """
    # Ініціалізація кращої точки сходження та внутрішніх точок
    best_foe = (0, 0)
    best_inliers = 0
    best_k = []

    for _ in range(num_iterations):
        # Перевірка на мінімальну кількість точок
        if len(flow_vectors) < 2:
            continue

        # Випадковий вибір двох точок для розрахунку FOE
        indices = np.random.choice(len(flow_vectors), 2, replace=False)
```

```
        vector1, vector2 = flow_vectors[indices[0]], flow_vectors[indices[1]]

        # Розрахунок коефіцієнтів для лінії (FOE)
        x1, y1 = vector1
        x2, y2 = vector2
        a, b, c = y1 - y2, x2 - x1, x1 * y2 - x2 * y1

        # Розрахунок кількості внутрішніх точок
        inliers = 0
        inlier_indices = []
        for i, (x, y) in enumerate(flow_vectors):
            if abs(a * x + b * y + c) < inlier_threshold:
                inliers += 1
                inlier_indices.append(i)

        # Оновлення кращої моделі, якщо знайдено більше внутрішніх точок
        if inliers > best_inliers:
            best_foe = (-c / (a + 1e-10), -c / (b + 1e-10))  # Уникнення ділення на нуль
            best_inliers = inliers
            best_k = inlier_indices

        # Якщо виявлено достатньо внутрішніх точок (поріг 90%), припинити ітерації
        if best_inliers > 0.9 * len(flow_vectors):
            break

    return best_foe, best_k

def inlier_static(q2, d, S, inlier_threshold):
    """
```

Визначає внутрішні точки для статичної сцени за допомогою методу RANSAC

та фільтрації точок на основі їх відстані від середнього значення.

Параметри:

q2 (numpy.ndarray): Координати точок на зображенні.

d (numpy.ndarray): Векторні напрямки для кожної точки.

S (tuple): Розміри зображення у вигляді (висота, ширина).

inlier_threshold (float): Поріг для визначення внутрішніх точок за допомогою RANSAC.

Повертає:

tuple: Відфільтровані координати точок, відфільтровані напрямки, а також точка сходження (FOE), коригована на розмір зображення.

"""

```
# Оцінка точки сходження (FOE) за допомогою методу RANSAC
foe, inliers = estimate_foe(d, inlier_threshold)

# Фільтрація точок за допомогою внутрішніх індексів
q2, d = q2[inliers], d[inliers]

# Обчислення евклідової відстані для кожної точки
a = np.linalg.norm(d, axis=1)

# Обчислення медіани відстаней
dm = np.median(a)

# Вибір точок, відстань яких менша за подвоєну медіану
inlier_indices = np.where(a < 2 * dm)

# Отримання точок, які відповідають фільтру
```

```
    qr, dr = q2[inlier_indices], d[inlier_indices]


     # Повернення відфільтрованих точок, напрямків та точки сходження з коригуванням на розмір зображення
    return qr, dr, foe + np.array([S[1] / 2, S[0] / 2])  # FOE коригується на розміри зображення
```

ДОДАТОУ Д. ФАЙЛ LUCAS_KANADE_FLOW.PY

```
import cv2
import numpy as np
from lib.foe_ransac import inlier_static


def calculate_window_indices(points, window_size, image_shape, level):


    x_range = [np.arange(val - window_size, val + window_size + 1) for val in points[:, 0]]
    y_range = [np.arange(val - window_size, val + window_size + 1) for val in points[:, 1]]


    # Обмеження індексів з урахуванням розмірів зображення
    x = np.minimum(np.array(x_range), image_shape[level, 1] - 1)
    y = np.minimum(np.array(y_range), image_shape[level, 0] - 1)


    # Обмеження індексів до 0
    x[x < 0] = 0
    y[y < 0] = 0


    # Створення комбінованих індексів для кожного пікселя
    x_combined = np.repeat(x, window_size, axis=1)
    y_combined = np.tile(y, window_size)


    # Пошук попередніх індексів
    x_prev = np.maximum(x_combined - 1, 0)
    y_prev = np.maximum(y_combined - 1, 0)


        return np.intp(x_combined), np.intp(y_combined), np.intp(x_prev), np.intp(y_prev)
```

```
def calculate_gradients(image, window_shape, level, x, y, x_prev, y_prev):

    I_x = (image[y, np.minimum(x + 1, window_shape[level, 1] - 1)] - image[y, x_prev]) / 2
    I_y = (image[np.minimum(y + 1, window_shape[level, 0] - 1), x] - image[y_prev, x]) / 2
    return I_x, I_y

def compute_intensity_difference(I_L, J_L, x, y, velocity, gradient, level, shape):

    vy = velocity[:, 1].reshape(len(velocity), 1)
    vx = velocity[:, 0].reshape(len(velocity), 1)
    gy = gradient[:, 1].reshape(len(gradient), 1)
    gx = gradient[:, 0].reshape(len(gradient), 1)

    k = np.clip(np.intp(np.round(y + vy + gy)), 0, shape[level, 0] - 1)
    m = np.clip(np.intp(np.round(x + vx + gx)), 0, shape[level, 1] - 1)

    intensity_diff = I_L[y, x] - J_L[k, m]
    return intensity_diff

def lucas_kanade_optical_flow(img1_, img2_, number_features, window_size, pyramid_levels, iterations, inlier_threshold):

    # Перетворення в відтінки сірого
    img1 = cv2.cvtColor(img1_, cv2.COLOR_BGR2GRAY)
    img2 = cv2.cvtColor(img2_, cv2.COLOR_BGR2GRAY)

    # Створення пірамід зображень
```

```
        I_L = np.empty((img1.shape[0], img1.shape[1], pyramid_levels), dtype=np.float32)
        J_L = np.empty((img1.shape[0], img1.shape[1], pyramid_levels), dtype=np.float32)
    I_L[:, :, 0] = img1
    J_L[:, :, 0] = img2


    shape = np.empty((pyramid_levels, 2), dtype=int)
    shape[0, :] = img1.shape


    # Створення рівнів піраміди
    for i in range(1, pyramid_levels):
      shape[i, :] = np.shape(cv2.resize(I_L[0:shape[i-1, 0], 0:shape[i-1, 1], i-1], None, fx=0.5, fy=0.5))
       I_L[0:shape[i, 0], 0:shape[i, 1], i] = cv2.resize(I_L[0:shape[i-1, 0], 0:shape[i-1, 1], i-1], None, fx=0.5, fy=0.5)
       J_L[0:shape[i, 0], 0:shape[i, 1], i] = cv2.resize(J_L[0:shape[i-1, 0], 0:shape[i-1, 1], i-1], None, fx=0.5, fy=0.5)


    # Знаходимо риси для відстеження
    features = cv2.goodFeaturesToTrack(I_L[:, :, 0], number_features, 0.01, 10)
    points = np.intp(np.array(features).reshape(len(features), -1))


    # Ініціалізація швидкостей
    g_Lm = np.zeros((len(points), 2), dtype=np.float32)
    velocities = []


    # Оптичний потік для кожного рівня
    for level in range(pyramid_levels - 1, -1, -1):
       q = np.intp(points / (2 ** level))
      x, y, x_prev, y_prev = calculate_window_indices(q, window_size, shape, level)
```

```
I_x, I_y = calculate_gradients(I_L[:, :, level], shape, level, x, y, x_prev, y_prev)

# Обчислення матриці градієнтів
I2x = I_x * I_x
I2y = I_y * I_y
Ixy = I_x * I_y

I_x2_sum = np.sum(I2x, axis=1)
I_y2_sum = np.sum(I2y, axis=1)
I_xy_sum = np.sum(Ixy, axis=1)

G = np.dstack((I_x2_sum, I_xy_sum, I_xy_sum, I_y2_sum)).reshape(len(I_x), 2, 2)
G_inv = np.linalg.pinv(G)

# Ініціалізація швидкості
velocity = np.zeros((len(x), 2), dtype=np.float32)

# Виконання ітерацій для поліпшення результату
for _ in range(iterations):
    dIk = compute_intensity_difference(I_L[:, :, level], J_L[:, :, level], x, y, velocity, g_Lm, level, shape)
    dIk_x = dIk * I_x
    dIk_y = dIk * I_y
    b = np.sum(np.dstack((dIk_x, dIk_y)), axis=1).reshape(len(I_x), 2, 1)

    # Обчислення оптичного потоку за методом Лукас-Канаде
    delta_v = np.matmul(G_inv, b).reshape(velocity.shape)
    velocity += delta_v
```

```
        g_Lm = 2 * (velocity + g_Lm)
        velocities.append(np.median(velocity, axis=0))

    # Розрахунок остаточного руху
    final_displacement = g_Lm / 2
     points, final_displacement, outliers = inlier_static(points, final_displacement, img1.shape, inlier_threshold)

    # Обчислення нових координат
    new_points = points + final_displacement
    return points, new_points, outliers
```