НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В.о. завідувача кафедри
_____ Артем ВОЛОКИТА
(підпис)
“__” _____2026 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”

на тему: «Система 3D-виявлення об’єктів у природних середовищах»

Виконав : студент __4_ курсу, групи ___ІМ-21
(шифр групи)

_____Демчик Дмитро Сергійович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник_____Гордієнко Ю. Г._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант_____Коренко Д. В._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____Сергієнко П. А.____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2026 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)
Освітньо-професійна програма
“Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Артем ВОЛОКИТА
(підпис)
“__” _____2026 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Демчика Дмитра Сергійовича

1. Тема проєкту *Система 3D-виявлення об’єктів у природних середовищах*
керівник проєкту Гордієнко Ю. Г.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 03.06.2026 №2055-с
2. Термін здачі студентом закінченого проєкту *2 червня 2026 року*
3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Розділ 1. Огляд предметної області та аналіз існуючих рішень
Розділ 2. Огляд технологій та алгоритмів розробки
Розділ 3. Розробка системи постобробки детекцій

Київ – 2026 р.

Розділ 4. Дослідження та аналіз розробленої системи

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) схема архітектури системи постобробки, блок-схема алгоритму зіставлення об'єктів, схема формату збереження даних.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Коренко Д. В.		

7. Дата видачі завдання «28» листопада 2025 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Узгодження теми та аналіз предметної області*	*28.11.2025 – 01.12.2025*	
2.	*Огляд існуючих рішень та постановка задачі*	*02.12.2025 – 10.12.2025*	
3.	*Налаштування середовища та реалізація конвеєра обробки*	*15.12.2025 – 25.12.2025*	
4.	*Розробка модулів зіставлення та фільтрації детекцій*	*26.12.2025 – 22.02.2026*	
5.	*Розробка модулів згладжування та обчислення метрик*	*23.02.2026 – 30.03.2026*	
6.	*Експериментальне дослідження та аналіз результатів*	*01.04.2026 – 18.04.2026*	
7.	*Оформлення пояснювальної записки*	*19.04.2026 – 10.06.2026*	
8.	*Захист програмного продукту*	*04.06.2026*	
9.	*Передзахист*	*05.06.2026*	
10.	*Захист*	*16.06.2026*	

Студент-дипломник _____ Дмитро Демчик
(підпис)

Керівник проєкту _____ Гордієнко Ю. Г.
(підпис)

Київ – 2026 р.

АНОТАЦІЯ

Дипломна робота присвячена підвищенню часової стабільності тривимірного виявлення об’єктів у відеопослідовностях природних сцен на основі моделей типу WildDet3D.

Розроблено модульну систему постобробки результатів покадрового виявлення, що включає зіставлення об’єктів між кадрами за перетином над об’єднанням, фільтрацію шуму та згладжування координат рамок. Запропоновано метрику часової стабільності, що не потребує еталонної розмітки. Система зменшує тремтіння рамок на 85,3 % та усуває близько 65 % хибних детекцій.

Наукова новизна полягає у підвищенні часової стабільності виявлення як етапу постобробки без перенавчання моделі та у запропонованій метриці. Систему може бути використано на доступному апаратному забезпеченні.

Ключові слова: тривимірне виявлення об’єктів, комп’ютерний зір, часова стабільність, постобробка детекцій, відстеження об’єктів, WildDet3D.

ABSTRACT

The thesis addresses improving the temporal stability of 3D object detection in video sequences of natural scenes based on WildDet3D-type models.

A modular post-processing system for frame-by-frame detection results has been developed, comprising cross-frame object matching by Intersection over Union, noise filtering, and smoothing of bounding box coordinates. A temporal stability metric requiring no ground-truth annotation is proposed. The system reduces bounding box jitter by 85.3 percent and removes about 65 percent of false detections.

The scientific novelty lies in improving temporal stability of 3D detection as a post-processing stage without retraining the model and in the proposed metric. The system can be used on accessible hardware.

Keywords: 3D object detection, computer vision, temporal stability, detection post-processing, object tracking, WildDet3D.

Справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Система 3D-виявлення об’єктів у природних середовищах	3		
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Система 3D-виявлення об’єктів у природних середовищах	64		
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Система 3D-виявлення об’єктів у природних середовищах	1		
			Схема архітектури системи постобробки			
	А4	***ІАЛЦ.467200.005 Д2***	Система 3D-виявлення об’єктів у природних середовищах	1		
			Блок-схема алгоритму зіставлення об’єктів			
	А4	***ІАЛЦ.467200.006 Д3***	Система 3D-виявлення об’єктів у природних середовищах	1		
			Схема формату збереження даних			
	А4	***ІАЛЦ.467200.007 Д4***	Система 3D-виявлення об’єктів у природних середовищах	21		
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА
Зм	***Лист***	***№ докум.***	***Підп***	***Дата***	
Розроб		Демчик Д. С.			Система 3D-виявлення об’єктів у природних середовищах Опис альбому
Перев		Гордієнко Ю. Г.			

Літ.	Аркуш	Аркушів
	1	1

НТУУ “КПІ” ФІОТ
ІМ-21

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система 3D-виявлення об’єктів у природних середовищах*»

Київ – 2026 р.

ТЕХНІЧНЕ ЗАВДАННЯ

1. Підстава для виконання роботи

Підставою для виконання роботи є завдання на дипломне проєктування, видане кафедрою обчислювальної техніки.

2. Мета і призначення розробки

Метою роботи є підвищення стабільності та точності тривимірного виявлення об'єктів у складних природних умовах на основі моделей типу WildDet3D шляхом розроблення алгоритмів часової узгодженості детекцій, фільтрації шуму та постобробки результатів виявлення.

Призначення розробки — програмна система постобробки результатів покадрового тривимірного виявлення, що підвищує часову стабільність детекцій у відео без перенавчання базової нейромережевої моделі.

3. Вихідні дані для розроблення

– модель монокулярного тривимірного виявлення WildDet3D з відкритим програмним кодом та ваговими коефіцієнтами;

– відеозаписи природних сцен як вхідні дані для обробки;

– науково-технічна література з тематики тривимірного виявлення об'єктів та забезпечення часової узгодженості.

4. Вимоги до програмного продукту

Програмний продукт повинен забезпечувати:

– зчитування відеофайлів та покадрове виявлення об'єктів за допомогою моделі WildDet3D;

– збереження результатів виявлення у структурованому форматі;

– зіставлення об'єктів між кадрами та присвоєння стійких ідентифікаторів треків;

– фільтрацію хибних детекцій за часовими характеристиками;

– часове згладжування координат обмежувальних рамок;

– обчислення метрик часової стабільності;

– візуалізацію результатів виявлення на кадрах відео.

5. Вимоги до апаратного та програмного забезпечення

– операційна система Windows 11;

– графічний процесор із підтримкою технології CUDA;

– мова програмування Python версії 3.11;

– фреймворк PyTorch, бібліотеки OpenCV, NumPy, Matplotlib.

6. Етапи виконання роботи

Таблиця етапів виконання роботи

№	Назва етапу	Термін виконання
1	Аналіз предметної області та огляд існуючих рішень	28.11.2025 – 01.12.2025
2	Дослідження моделі WildDet3D, налаштування середовища	02.12.2025– 10.12.2025
3	Реалізація базового конвеєра обробки відео	15.12.2025 – 25.12.2025
4	Розроблення модулів зіставлення та фільтрації	26.12.2025 – 22.02.2026
5	Розроблення модуля згладжування та метрик	23.02.2026 – 30.03.2026
6	Експериментальне дослідження системи	01.04.2026 – 18.04.2026
7	Оформлення пояснювальної записки	19.04.2026-10.06.2026

7. Перелік графічного матеріалу

– схема архітектури системи постобробки;

– блок-схема алгоритму зіставлення об’єктів;

– схема формату збереження даних.

8. Очікувані результати

– програмна система постобробки результатів тривимірного виявлення;

– метрика часової стабільності детекцій;

– результати експериментального дослідження ефективності системи;

– пояснювальна записка до дипломного проєкту.

Завдання прийняв до виконання: _____ (підпис студента)

Керівник проєкту: _____ (підпис керівника)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

Скорочення та абревіатури

2D — двовимірний (two-dimensional)

3D — тривимірний (three-dimensional)

AP — середня точність (average precision)

API — програмний інтерфейс застосунку (application programming interface)

BGR — колірний простір «синій — зелений — червоний» (blue — green — red)

CUDA — програмно-апаратна платформа паралельних обчислень (Compute Unified Device Architecture)

EMA — експоненційне ковзне середнє (exponential moving average)

GPU — графічний процесор (graphics processing unit)

IoU — перетин над об'єднанням (Intersection over Union)

JSON — текстовий формат обміну даними (JavaScript Object Notation)

MA — ковзне середнє (moving average)

NMS — придушення немаксимумів (non-maximum suppression)

RGB — колірний простір «червоний — зелений — синій» (red — green — blue)

VRAM — відеопам'ять (video random access memory)

Терміни

Детекція — окремий результат виявлення об'єкта, що містить обмежувальну рамку, клас та оцінку впевненості.

Трек — послідовність детекцій, що належать одному об'єкту в різних кадрах відео.

Тремтіння (jitter) — дрібні випадкові коливання координат обмежувальної рамки між кадрами, не зумовлені реальним рухом об'єкта.

Промпт (підказка) — вхідний сигнал, що задає моделі, які об'єкти потрібно знайти.

Монокулярне виявлення — виявлення об'єктів за єдиним зображенням від однієї камери.

ВСТУП

Актуальність теми. Виявлення об'єктів у тривимірному просторі є однією з фундаментальних задач сучасного комп'ютерного зору. Здатність автоматично визначати не лише наявність та клас об'єктів на зображенні, а й їх просторове розташування, розміри та орієнтацію є необхідною умовою роботи систем автономного транспорту, робототехніки, доповненої реальності та інтелектуального відеоспостереження. Серед підходів до тривимірного виявлення особливе місце посідає монокулярне виявлення — відновлення просторової інформації з єдиного зображення звичайної камери. Його практична привабливість зумовлена доступністю та поширеністю звичайних камер, які не потребують дорогого спеціалізованого обладнання, як-от лідарних сканерів.

Сучасні моделі монокулярного тривимірного виявлення, побудовані на основі глибоких нейронних мереж, досягли значного прогресу й демонструють високу якість при обробці окремих зображень. Прикладом такої моделі є WildDet3D — промптована модель з відкритим словником категорій, що показує провідні результати на низці контрольних наборів даних. Однак переважна більшість подібних моделей розроблена й оцінена саме для окремих зображень, тоді як значна частина практичних застосувань має справу з відеопотоком.

При застосуванні моделі виявлення до відео в покадровому режимі, коли кожен кадр обробляється незалежно від інших, виникає проблема часової неузгодженості результатів. Вона проявляється у мерехтінні детекцій, тремтінні обмежувальних рамок, нестабільності ідентифікації об'єктів та появі непостійних хибних спрацювань. Ці дефекти погіршують якість результату та обмежують практичне застосування моделей виявлення для аналізу відео. Усунення цих дефектів шляхом перенавчання моделі є непрактичним через її обчислювальну складність. Це зумовлює актуальність розроблення методів

постобробки, які підвищують часову стабільність виявлення без втручання в саму модель.

Мета і завдання дослідження. Метою роботи є підвищення стабільності та точності тривимірного виявлення об'єктів у складних природних умовах на основі моделей типу WildDet3D шляхом розроблення алгоритмів часової узгодженості детекцій, адаптивної фільтрації шуму та постобробки результатів виявлення.

Для досягнення поставленої мети визначено такі завдання:

– провести аналіз сучасних підходів до тривимірного виявлення об'єктів, виявлення на відео та методів забезпечення часової узгодженості;

– дослідити архітектуру, формат даних та обмеження моделі WildDet3D;

– реалізувати базовий конвеєр обробки відео зі збереженням результатів виявлення;

– розробити метод зіставлення об'єктів між кадрами та присвоєння стійких ідентифікаторів;

– розробити метод фільтрації шуму на основі часових характеристик треків;

– розробити метод часового згладжування координат обмежувальних рамок;

– реалізувати систему автоматизованого аналізу результатів та запропонувати метрику часової стабільності;

– провести експериментальне дослідження розробленої системи, зокрема абляційне порівняння методів згладжування.

Об'єкт дослідження — процес виявлення тривимірних об'єктів у відеопослідовностях природних сцен.

Предмет дослідження — методи та алгоритми постобробки результатів покадрового виявлення, спрямовані на підвищення часової стабільності тривимірної детекції.

Методи дослідження. У роботі використано методи комп'ютерного зору для виявлення об'єктів, методи обчислювальної геометрії для зіставлення

обмежувальних рамок, методи обробки часових рядів для згладжування траєкторій об'єктів, а також експериментальні методи для кількісної оцінки ефективності розробленої системи.

Наукова новизна одержаних результатів полягає в такому. Запропоновано підхід до підвищення часової стабільності тривимірного виявлення об'єктів у відео, який реалізується як етап постобробки й не потребує перенавчання базової моделі. Запропоновано метрику часової стабільності детекцій, що ґрунтується на вимірюванні тремтіння обмежувальних рамок і не потребує еталонної розмітки даних. Проведено абляційне порівняння методів згладжування траєкторій стосовно задачі усунення тремтіння тривимірних детекцій.

Практичне значення одержаних результатів. Розроблена система дозволяє підвищити якість тривимірного виявлення об'єктів у відео без перенавчання базової моделі, що робить її застосовною на доступному апаратному забезпеченні. Модульна архітектура системи дозволяє використовувати її з різними моделями виявлення. Запропонована метрика часової стабільності може застосовуватися для оцінювання якості систем виявлення на відео. Розроблене програмне забезпечення може бути використане як основа для подальших досліджень у галузі стабілізації відеодетекції.

Структура та обсяг роботи. Дипломна робота складається зі вступу, чотирьох розділів, висновків, переліку посилань та додатків. У першому розділі проведено аналіз предметної області та існуючих рішень. У другому розділі розглянуто технології та алгоритми, використані при розробленні. У третьому розділі описано розроблення програмної системи постобробки. У четвертому розділі наведено результати експериментального дослідження.

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Задача виявлення об'єктів: двовимірне та тривимірне виявлення

Виявлення об'єктів є однією з фундаментальних задач комп'ютерного зору. Її суть полягає в автоматичному визначенні наявності об'єктів певних категорій на зображенні, встановленні їх місцеположення та класифікації. Виявлення об'єктів лежить в основі численних прикладних систем — від відеоспостереження й аналізу дорожнього руху до робототехніки та доповненої реальності.

Класична постановка задачі стосується двовимірного виявлення. У цьому випадку місцеположення кожного об'єкта описується прямокутною обмежувальною рамкою, заданою на площині зображення чотирма координатами. Двовимірне виявлення досягло високого рівня зрілості завдяки появі великих розмічених наборів даних та розвитку згорткових нейронних мереж. Проте двовимірна рамка описує лише проєкцію об'єкта на площину зображення і не містить інформації про його реальне положення у просторі, розміри та орієнтацію.

Багато практичних застосувань потребують саме просторового розуміння сцени. Системі автономного керування транспортним засобом необхідно знати не лише те, що попереду є інший автомобіль, а й на якій він відстані, яких він розмірів і куди орієнтований. Робот-маніпулятор для захоплення предмета потребує його тривимірних координат та орієнтації. Системи доповненої реальності повинні коректно розташовувати віртуальні об'єкти відносно реальних. Ці потреби зумовлюють перехід від двовимірного до тривимірного виявлення.

Тривимірне виявлення об'єктів розширює задачу: замість плоскої рамки кожен об'єкт описується тривимірною обмежувальною рамкою. Така рамка задається положенням центру у тривимірному просторі, трьома розмірами — довжиною, шириною та висотою, — а також орієнтацією. Орієнтація може описуватися одним кутом повороту навколо вертикальної осі або повним набором із трьох кутів. Результатом тривимірного виявлення є, таким чином, повний просторовий опис об'єкта, придатний для подальшого геометричного аналізу сцени.

Тривимірне виявлення є істотно складнішою задачею, ніж двовимірне. Основна складність полягає у відновленні інформації про глибину, яка втрачається при проєктуванні тривимірної сцени на двовимірну матрицю зображення. Один і той самий об'єкт, розташований на різній відстані від камери, може давати однакову проєкцію, тому однозначне відновлення глибини лише за зображенням є принципово неоднозначною задачею. Залежно від того, які додаткові джерела інформації використовуються для подолання цієї неоднозначності, методи тривимірного виявлення поділяють на кілька категорій.

Методи, що спираються на дані лідара, використовують хмару точок — набір просторових координат, отриманих лазерним сканером. Хмара точок безпосередньо містить інформацію про глибину, що спрощує тривимірне виявлення, проте лідарне обладнання є дорогим і не завжди доступним. Стереоскопічні методи використовують пару зображень з рознесених камер і відновлюють глибину за паралаксом. Монокулярні методи працюють з одним зображенням від однієї камери — це найдоступніший, але й найскладніший варіант, оскільки глибину доводиться оцінювати непрямими ознаками. Саме монокулярне тривимірне виявлення є предметом розгляду цієї роботи, тому йому присвячено окремий підрозділ.

1.2. Монокулярна тривимірна детекція: підходи й виклики

Монокулярне тривимірне виявлення полягає у відновленні повного просторового опису об'єктів — їх положення, розмірів та орієнтації — з єдиного зображення, отриманого однією звичайною камерою. Привабливість цього напряму зумовлена доступністю обладнання: звичайні камери є дешевими, поширеними і вже вбудованими у безліч пристроїв. Це робить монокулярні методи потенційно застосовними там, де лідарні чи стереоскопічні системи економічно невиправдані.

Водночас монокулярне виявлення є найскладнішим серед усіх підходів через принципову неоднозначність відновлення глибини з одного зображення. Оскільки проєктування тривимірної сцени на площину незворотне, метод змушений спиратися на непрямі ознаки глибини: відомі типові розміри об'єктів певних класів, перспективні спотворення, взаємне розташування об'єктів, положення відносно поверхні землі. Унаслідок цього монокулярні методи традиційно поступаються за точністю лідарним, але активно розвиваються завдяки своїй практичній доступності.

Ранні методи монокулярного тривимірного виявлення розроблялися переважно для сценаріїв автономного водіння і були вузькоспеціалізованими. Вони працювали з обмеженим набором категорій, характерних для дорожнього руху — автомобіль, пішохід, велосипедист, — і покладалися на специфічні припущення, наприклад про те, що всі об'єкти стоять на спільній площині дороги. Такі методи показували прийнятну точність у своїй вузькій області, але не узагальнювалися на інші сцени та категорії об'єктів.

Істотним обмеженням розвитку галузі тривалий час була відсутність великих і різноманітних наборів даних. На відміну від двовимірного виявлення, де великі набори даних з'явилися рано і стимулювали швидкий прогрес, тривимірні набори залишалися малими й вузькоспеціалізованими. Створення тривимірної розмітки є набагато трудомісткішим: розмітник має визначити не плоску рамку, а повну просторову конфігурацію об'єкта.

Помітним кроком у подоланні цього обмеження став набір даних Omni3D, представлений у роботі [2]. Його автори об'єднали кілька наявних

наборів даних в один великий і різноманітний набір, що містить близько 234 тисяч зображень з понад трьома мільйонами розмічених об'єктів 98 категорій, які охоплюють як вуличні, так і внутрішні сцени. Разом з набором даних автори запропонували модель Cube R-CNN — універсальний детектор, побудований на основі архітектури двостадійного виявлення з доданою спеціалізованою головою для передбачення параметрів тривимірної рамки. Cube R-CNN продемонстрував здатність узагальнюватися на різні типи камер і сцен, що стало важливим зрушенням від вузькоспеціалізованих методів до універсальних.

Навіть універсальні детектори, навчені на фіксованому наборі категорій, мають принципове обмеження: вони здатні виявляти лише ті класи об'єктів, які були присутні у навчальних даних. У реальних умовах кількість можливих категорій об'єктів є практично необмеженою, тому виникає потреба у виявленні з відкритим словником — здатності системи знаходити об'єкти категорій, які не були заздалегідь визначені при навчанні. Для цього сучасні методи інтегрують у задачу виявлення текстову інформацію: категорія об'єкта задається не числовим індексом з фіксованого переліку, а текстовим описом природною мовою.

Подальший розвиток ідеї відкритого словника привів до концепції промптованого виявлення, за якої користувач задає об'єкт для пошуку за допомогою підказки — промпта. Підказка може мати різну природу: текстовий опис категорії, точка на зображенні, обмежувальна рамка. Така парадигма набула поширення завдяки моделі Segment Anything та її наступникам, які продемонстрували гнучкість керування процесом виявлення й сегментації через підказки різних типів [3].

Сучасним прикладом застосування цієї парадигми до монокулярного тривимірного виявлення є модель WildDet3D, представлена у роботі [1]. Вона поєднує кілька згаданих напрямів: працює з одним зображенням, підтримує відкритий словник категорій, приймає підказки трьох типів — текст, точку й рамку, — а також може використовувати додаткову інформацію про глибину,

якщо така доступна. Модель навчена на великому наборі даних, що містить понад мільйон зображень з 13,5 тисячами категорій. Саме ця модель використовується як базовий компонент виявлення у цій роботі, тому її архітектура детально розглядається у другому розділі.

Попри значний прогрес, монокулярне тривимірне виявлення залишається складною задачею з відчутними обмеженнями за точністю. Окремою проблемою, яка майже не розглядається у роботах, орієнтованих на виявлення за одним зображенням, є поведінка таких методів при застосуванні до відео. Цій проблемі присвячено наступний підрозділ.

1.3. Виявлення об'єктів на відео та проблема часової неузгодженості

Переважна більшість методів виявлення об'єктів, зокрема й тривимірних, розроблялася й оцінювалася для окремих зображень. Проте значна частина практичних застосувань має справу не з окремими знімками, а з відеопотоком — неперервною послідовністю кадрів. Системи відеоспостереження, аналізу дорожнього руху, автономного керування транспортом, робототехніки працюють саме з відео. Це зумовлює потребу розглянути особливості застосування детекторів до відеопослідовностей.

Найпростіший спосіб обробити відео полягає у незалежному застосуванні детектора зображень до кожного кадру окремо. Такий підхід не потребує жодних модифікацій моделі й легко реалізується. Однак він має принциповий недолік: модель обробляє кожен кадр ізольовано, не використовуючи інформації про те, що сусідні кадри відеопотоку є тісно пов'язаними. Сусідні кадри відрізняються лише незначним рухом об'єктів і камери за короткий проміжок часу, тому результати виявлення на них мали б бути узгодженими. Покадровий підхід цієї узгодженості не забезпечує.

Відсутність врахування часового зв'язку між кадрами призводить до низки характерних дефектів, які узагальнено називають часовою неузгодженістю результатів виявлення.

Першим проявом є мерехтіння детекцій. Об'єкт, чітко присутній на сцені, може бути виявлений на одному кадрі, не виявлений на наступному і знову виявлений на подальшому. Об'єкт ніби блимає, то з'являючись, то зникаючи, хоча фізично він залишається на сцені незмінно. Причиною є те, що оцінка впевненості моделі коливається біля порогового значення, і незначні зміни зображення від кадру до кадру переводять детекцію то вище, то нижче порога.

Другим проявом є тремтіння обмежувальних рамок. Навіть для об'єкта, який стабільно виявляється на всіх кадрах, координати його рамки дрібно й хаотично коливаються від кадру до кадру. Ці коливання не відповідають реальному руху об'єкта, а є наслідком стохастичної природи нейромережевого виявлення: незначні зміни вхідного зображення спричиняють незначні зміни передбачених координат. На статичному кадрі тремтіння непомітне, але при відтворенні відео воно проявляється як помітне дрижання рамки навколо об'єкта.

Третім проявом є нестабільність ідентифікації. Оскільки детектор обробляє кожен кадр незалежно, він не присвоює об'єктам стійких ідентифікаторів. Той самий фізичний об'єкт у різних кадрах ніяк не пов'язаний з самим собою, що унеможливлює простеження його руху без додаткової обробки.

Окремим, особливо небажаним проявом часової неузгодженості є непостійність хибних спрацювань. Хибна детекція — фантом, породжений випадковим збігом візуальних ознак, — зазвичай виникає на окремому кадрі й зникає на наступному. На відміну від реального об'єкта, фантом не виявляє часової стійкості.

Остання властивість має важливий наслідок, який лежить в основі цієї роботи. Якщо хибні детекції непостійні, а реальні об'єкти стабільно простежуються впродовж багатьох кадрів, то сама часова поведінка детекції стає ознакою її достовірності. Аналіз результатів виявлення не покадрово, а вздовж часу дозволяє відрізнити корисний сигнал від шуму: те, що стабільно

повторюється, є реальним об'єктом, а те, що з'являється епізодично, є з високою ймовірністю хибним спрацюванням.

Таким чином, часова неузгодженість є одночасно і проблемою, і можливістю. Як проблема вона погіршує якість результату — спричиняє мерехтіння й тремтіння. Як можливість вона надає додаткову ознаку для фільтрації шуму. Обидва ці аспекти використовуються у розробленій системі: часова неузгодженість усувається там, де вона шкідлива, і використовується там, де вона корисна. Методи, які цілеспрямовано забезпечують узгодженість результатів виявлення у часі, об'єднують під назвою методів часової узгодженості. Їх огляд наведено в наступному підрозділі.

1.4. Огляд методів часової узгодженості в комп'ютерному зорі

Задача забезпечення часової узгодженості результатів обробки відео не є новою для комп'ютерного зору. Існує низка підходів, які різняться за тим, на якому етапі та яким чином враховується часовий зв'язок між кадрами. Їх можна умовно поділити на дві великі групи: методи, вбудовані в модель, і методи постобробки.

Перша група об'єднує підходи, у яких часова узгодженість забезпечується самою архітектурою нейромережевої моделі. Модель у цьому випадку приймає на вхід не окремий кадр, а кілька послідовних кадрів, і має у своїй структурі компоненти, здатні поєднувати інформацію з різних моментів часу. Для цього застосовують агрегацію ознак сусідніх кадрів, рекурентні зв'язки, що передають внутрішній стан моделі від кадру до кадру, механізми уваги вздовж часової осі.

Перевагою цих методів є те, що модель навчається враховувати час безпосередньо під час тренування й може досягати високої якості. Недоліки, однак, суттєві. Такі методи потребують спеціально спроєктованої архітектури та навчання на відеоданих з відповідною розміткою, що є дорогим. Вони прив'язані до конкретної моделі: забезпечити часову узгодженість для вже наявного детектора зображень у такий спосіб неможливо без його

перепроєктування й перенавчання. Нарешті, обробка кількох кадрів одночасно підвищує обчислювальні вимоги.

Друга група об'єднує підходи, у яких часова узгодженість забезпечується окремим етапом обробки, що застосовується до вже готових результатів покадрового виявлення. Модель виявлення при цьому залишається незмінною й використовується як є; уся часова логіка винесена в окремий компонент.

Центральним поняттям для методів постобробки є відстеження об'єктів. Задача відстеження полягає у встановленні відповідності між детекціями, що належать одному об'єкту, у послідовних кадрах, та присвоєнні об'єктам стійких ідентифікаторів. Поширений підхід до відстеження відомий як відстеження через виявлення: спершу детектор незалежно знаходить об'єкти на кожному кадрі, а потім окремий алгоритм пов'язує ці детекції у траєкторії — треки.

Класичним і широко застосовуваним прикладом такого підходу є метод SORT та його розвиток DeepSORT [4]. У цих методах зіставлення детекцій між кадрами виконується на основі двох складників: геометричної близькості рамок, яка оцінюється перекриттям або прогнозованим положенням, та, у випадку DeepSORT, схожості зовнішнього вигляду об'єктів. Прогнозування положення об'єкта в наступному кадрі здійснюється фільтром Калмана, а оптимальне зіставлення детекцій з наявними треками — угорським алгоритмом. Ці методи показали, що проста й обчислювально ефективна постобробка здатна надати покадровому детектору властивість стійкого відстеження.

Після того як детекції об'єднані у треки, стають можливими подальші етапи постобробки, спрямовані на усунення конкретних проявів часової неузгодженості. Фільтрація треків за їх характеристиками — довжиною, стабільністю, середньою впевненістю — дозволяє відсіяти короткі нестабільні треки, що відповідають хибним спрацюванням. Згладжування траєкторій усуває тремтіння: координати рамок усереднюються вздовж треку методами

обробки часових рядів. Інтерполяція пропущених детекцій відновлює об'єкт у кадрах, де детектор його не виявив, усуваючи мерехтіння.

Кожна з двох груп має свою область доцільного застосування. Вбудовані в модель методи здатні досягати вищої якості, але потребують перепроєктування й перенавчання моделі. Методи постобробки програють у потенційній якості, проте мають вирішальні практичні переваги: вони не потребують перенавчання, застосовні до будь-якого наявного детектора й допускають незалежне налаштування та дослідження кожного етапу.

Для цієї роботи вирішальними є саме практичні міркування. Базова модель WildDet3D є великою моделлю з понад мільярдом параметрів; її перенавчання виходить за межі обчислювальних можливостей типового персонального комп'ютера. Крім того, модель є моделлю виявлення за одним зображенням і не призначена для обробки відеопослідовностей. За таких умов підхід постобробки є єдиним практично доцільним. Тому в розробленій системі забезпечення часової стабільності реалізовано як окремий етап постобробки за схемою відстеження через виявлення: покадрові детекції моделі об'єднуються у треки зіставленням за перекриттям рамок, після чого застосовуються фільтрація треків та згладжування траєкторій.

Слід зазначити, що завдання цієї роботи не полягає у створенні нового алгоритму відстеження, який конкурував би з SORT чи DeepSORT за повнотою. Метою є побудова цілісного конвеєра постобробки, спеціалізованого на підвищенні часової стабільності тривимірного виявлення, з прозорими, придатними для дослідження компонентами та з кількісною метрикою оцінки досягнутої стабільності.

1.5. Постановка задачі дослідження

На основі проведеного аналізу предметної області можна сформулювати постановку задачі дослідження.

Сучасні моделі монокулярного тривимірного виявлення об'єктів, такі як WildDet3D, досягають високої якості при обробці окремих зображень, проте

розроблені й оцінені саме для окремих зображень, а не для відеопослідовностей. При покадровому застосуванні таких моделей до відео проявляється часова неузгодженість результатів: мерехтіння детекцій, тремтіння обмежувальних рамок, нестабільність ідентифікації об'єктів та поява непостійних хибних спрацювань. Це погіршує якість результату і обмежує практичне застосування таких моделей для аналізу відео. Перенавчання моделі для усунення цих дефектів є непрактичним через її обчислювальну складність.

Об'єктом дослідження є процес виявлення тривимірних об'єктів у відеопослідовностях природних сцен. Предметом дослідження є методи та алгоритми постобробки результатів покадрового виявлення, спрямовані на підвищення часової стабільності й точності тривимірної детекції.

Метою роботи є підвищення стабільності та точності тривимірного виявлення об'єктів у складних природних умовах на основі моделей типу WildDet3D шляхом розроблення алгоритмів часової узгодженості детекцій, адаптивної фільтрації шуму та постобробки результатів виявлення.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. Провести аналіз сучасних підходів до тривимірного виявлення об'єктів, виявлення на відео та методів забезпечення часової узгодженості в комп'ютерному зорі.
2. Дослідити архітектуру моделі WildDet3D: формат вхідних і вихідних даних, конвеєр виявлення, обмеження моделі.
3. Реалізувати базовий програмний конвеєр: запуск моделі на відеопослідовностях, збереження результатів виявлення у структурованому форматі.
4. Розробити метод часової агрегації детекцій: зіставлення об'єктів між кадрами, присвоєння стійких ідентифікаторів, усунення мерехтіння.
5. Розробити метод фільтрації шуму на основі часових характеристик треків та оцінок впевненості.

6. Розробити метод часового згладжування координат для усунення тремтіння обмежувальних рамок.

7. Реалізувати систему автоматизованого аналізу результатів: обчислення метрик часової стабільності, порівняння результатів до й після обробки.

8. Провести експериментальне дослідження розробленої системи, зокрема абляційне порівняння методів згладжування.

У роботі використано методи комп'ютерного зору для виявлення об'єктів, методи обчислювальної геометрії для зіставлення обмежувальних рамок, методи обробки часових рядів для згладжування траєкторій, а також експериментальні методи для кількісної оцінки ефективності розробленої системи.

Розроблена система дозволяє підвищити якість тривимірного виявлення об'єктів у відео без перенавчання базової моделі, що робить її застосовною на доступному апаратному забезпеченні. Запропонована метрика часової стабільності може використовуватися для оцінки якості будь-яких систем виявлення на відео.

Висновок до розділу 1

У розділі проведено аналіз предметної області та існуючих рішень. Розглянуто задачу виявлення об'єктів та її ускладнення при переході від двовимірного до тривимірного виявлення. Показано, що монокулярне тривимірне виявлення є найдоступнішим за обладнанням, але найскладнішим підходом через принципову неоднозначність відновлення глибини з одного зображення, та окреслено еволюцію методів від вузькоспеціалізованих детекторів до сучасних промптованих моделей з відкритим словником, таких як WildDet3D.

Проаналізовано проблему застосування детекторів зображень до відео. Встановлено, що покадрове виявлення спричиняє часову неузгодженість результатів — мерехтіння, тремтіння та нестабільність ідентифікації, — а також показано, що непостійність хибних спрацювань може слугувати ознакою для їх відсіювання. Розглянуто дві групи методів забезпечення часової узгодженості — вбудовані в модель та постобробку — і обґрунтовано, що для умов цієї роботи єдиним практично доцільним є підхід постобробки за схемою відстеження через виявлення.

На основі проведеного аналізу сформульовано постановку задачі дослідження: визначено проблему, об'єкт, предмет, мету, завдання та методи дослідження, які реалізуються у наступних розділах роботи.

РОЗДІЛ 2
ОГЛЯД ТЕХНОЛОГІЙ ТА АЛГОРИТМІВ РОЗРОБКИ

2.1. Модель WildDet3D: архітектура та принцип роботи

Базовим компонентом виявлення у розробленій системі є модель WildDet3D, представлена дослідниками Інституту штучного інтелекту Аллена спільно з Вашингтонським і Корнельським університетами [1]. Модель призначена для промптованого монокулярного тривимірного виявлення об'єктів — вона визначає й локалізує об'єкти у тривимірному просторі за єдиним зображенням RGB.

Вибір саме цієї моделі для роботи зумовлений кількома її властивостями. По-перше, вона працює з одним зображенням від звичайної камери, що відповідає найбільш доступному й найпоширенішому сценарію застосування. По-друге, вона підтримує відкритий словник категорій і керування через підказки, що робить її гнучкою. По-третє, на момент виконання роботи вона є однією з найсучасніших моделей у своїй області, демонструючи провідні результати на низці контрольних наборів даних. По-четверте, її програмний код і вагові коефіцієнти є відкритими, що уможливлює практичне використання.

WildDet3D побудована як єдина геометрично-обізнана архітектура. Її структуру можна умовно поділити на чотири функціональні частини: кодувальник зображення, модуль обробки підказок, модуль урахування геометрії та каскад голів виявлення.

Кодувальник зображення перетворює вхідне зображення на внутрішнє подання — набір ознак, що описують візуальний зміст сцени. Як основу кодувальника WildDet3D використовує архітектуру, запозичену з моделі сегментації Segment Anything, яка зарекомендувала себе як потужний універсальний кодувальник зображень [3].

Модуль обробки підказок дозволяє керувати тим, які саме об'єкти має знайти модель. Підказка перетворюється на внутрішнє подання, сумісне з ознаками зображення, після чого процес виявлення спрямовується на пошук об'єктів, що відповідають підказці.

Модуль урахування геометрії відповідає за роботу з інформацією про глибину. Особливістю WildDet3D є здатність використовувати додаткові геометричні дані, якщо вони доступні. Коли разом із зображенням подається карта глибини — наприклад, отримана від лідара чи стереокамери, — модуль інтегрує цю інформацію, що підвищує точність тривимірної локалізації. Якщо ж дані про глибину відсутні, модель переходить у суто монокулярний режим і оцінює глибину самостійно за допомогою вбудованого компонента оцінювання глибини. Така здатність працювати як з додатковою геометрією, так і без неї робить модель універсальною щодо доступного обладнання.

Каскад голів виявлення формує остаточний результат. Він складається з голови двовимірного виявлення, яка визначає положення об'єктів на площині зображення, та голови тривимірного виявлення, яка на основі двовимірного результату та геометричних ознак передбачає повну тривимірну рамку — метричну глибину, розміри об'єкта та його орієнтацію.

WildDet3D підтримує три типи підказок, кожен з яких задає задачу виявлення по-різному. Текстова підказка задає категорії об'єктів для пошуку у вигляді переліку назв природною мовою; модель знаходить усі екземпляри вказаних категорій. Підказка у вигляді рамки задає двовимірну рамку, для якої модель відновлює відповідну тривимірну. Точкова підказка задає об'єкт вказівкою точки на зображенні. У розробленій системі використовується текстовий режим підказки, оскільки він найкраще відповідає задачі автоматичної обробки відео, де потрібно знаходити всі об'єкти заданих категорій без втручання користувача в кожен кадр.

Слід зазначити, що промптована природа моделі є водночас і її перевагою, і обмеженням, яке враховується при побудові системи: модель не

виявляє об'єкти взагалі, а потребує явного задання цільових категорій перед обробкою.

2.2. Формат вхідних і вихідних даних, обмеження моделі

Для коректної інтеграції моделі WildDet3D у систему постобробки необхідно чітко визначити формат даних, з якими вона працює, та враховувати притаманні їй обмеження.

Основним входом моделі є зображення RGB — кольоровий знімок сцени. Перед подачею до моделі зображення проходить етап попередньої обробки, під час якого масштабується та доповнюється до фіксованих розмірів, очікуваних кодувальником. Окрім зображення, модель приймає кілька допоміжних входів.

Першим є параметри калібрування камери — матриця внутрішніх параметрів, яка описує фокусну відстань і положення оптичного центру. Ці параметри потрібні для коректного переходу між координатами зображення та тривимірними координатами простору. Особливістю WildDet3D є те, що за відсутності відомих параметрів калібрування — а для довільного відео з невідомого джерела вони, як правило, невідомі — модель здатна або використати типові значення за замовчуванням, або оцінити параметри самостійно. Це робить модель застосовною до відео з невідомою камерою, що є важливим для умов цієї роботи.

Другим допоміжним входом є підказка, яка в обраному текстовому режимі є переліком назв категорій. Третім, необов'язковим, входом є карта глибини, використання якої описано в попередньому підрозділі.

Результатом роботи моделі для одного зображення є набір детекцій. Кожна детекція містить двовимірну обмежувальну рамку у вигляді чотирьох координат на площині зображення; тривимірну обмежувальну рамку, що включає координати центру у просторі, три розміри та орієнтацію, подану кватерніоном; ідентифікатор класу об'єкта згідно з переліком категорій підказки; а також оцінки впевненості — окремо для двовимірного й

тривимірного виявлення. Окрім детекцій, модель повертає оцінену карту глибини сцени.

Саме цей формат вихідних даних визначає структуру даних, з якою працюють усі модулі постобробки. Кожна детекція в системі зберігається як запис із переліченими полями, що безпосередньо відповідають виходу моделі.

При побудові системи враховано низку обмежень моделі WildDet3D, які безпосередньо вплинули на проєктні рішення.

Найважливішим обмеженням є те, що WildDet3D є моделлю виявлення за одним зображенням. Вона обробляє кожен кадр незалежно й не має жодного механізму врахування часового зв'язку між кадрами. Саме це обмеження є першопричиною часової неузгодженості, описаної в першому розділі, і саме на його подолання спрямована вся розроблена система постобробки.

Другим обмеженням є промптована природа моделі: вона потребує явного задання цільових категорій і не виявляє об'єкти поза заданим переліком. Унаслідок цього система вимагає визначення переліку цільових класів перед обробкою відео.

Третім обмеженням є значна обчислювальна складність моделі. WildDet3D містить понад мільярд параметрів, що зумовлює високі вимоги до обсягу відеопам'яті та тривалий час обробки одного кадру. Це обмеження визначило два проєктні рішення системи: застосування проріджування кадрів для скорочення загального часу обробки та відмову від будь-яких варіантів, що передбачають перенавчання моделі.

Четвертим обмеженням є залежність точності тривимірної локалізації від наявності параметрів калібрування камери. За їх відсутності модель використовує наближені значення, що знижує точність відновлення абсолютних просторових координат. Це обмеження враховується при інтерпретації результатів: основна увага в роботі приділяється часовій стабільності виявлення, а не абсолютній точності тривимірної локалізації, яка за невідомої камери є принципово обмеженою.

Усвідомлення цих обмежень є необхідним для коректної побудови системи постобробки: система не намагається їх усунути всередині моделі, а компенсує їхні наслідки на окремому етапі обробки.

2.3. Мова Python та екосистема для комп'ютерного зору

Для реалізації системи обрано мову програмування Python. Цей вибір зумовлений як вимогами задачі, так і усталеною практикою галузі.

Python є мовою високого рівня з лаконічним та зрозумілим синтаксисом, що пришвидшує розробку та полегшує читання коду. Для дослідницької роботи, у якій код часто змінюється в ході експериментів, ця властивість є суттєвою перевагою. Проте вирішальним чинником вибору є не сам синтаксис мови, а її екосистема.

Python став фактичним стандартом у галузях машинного навчання, комп'ютерного зору та аналізу даних. Навколо мови сформувалася велика екосистема спеціалізованих бібліотек, які покривають усі потреби розробки подібних систем — від роботи із зображеннями до навчання нейронних мереж. Усі сучасні фреймворки глибокого навчання надають Python як основний інтерфейс. Модель WildDet3D, що є базовим компонентом системи, також реалізована мовою Python і розрахована на використання з відповідної екосистеми. Це робить Python природним і фактично безальтернативним вибором: використання іншої мови вимагало б побудови проміжного інтерфейсу до моделі й позбавило б доступу до готових бібліотек обробки зображень.

Важливою особливістю Python є його роль мови-зв'язки. Сам інтерпретатор Python є відносно повільним, тому обчислювально складні операції — матричні обчислення, обробка зображень, виконання нейронних мереж — реалізовані не на чистому Python, а на компільованих мовах, як-от C++ та CUDA. Python у цьому випадку виступає зручним інтерфейсом керування, тоді як власне обчислення виконуються високопродуктивним

кодом. Завдяки цьому поєднанню досягається і зручність розробки, і прийнятна швидкодія.

Для розробки системи використано низку бібліотек екосистеми. Бібліотека NumPy надає ефективні структури для роботи з багатовимірними числовими масивами й використовується для представлення зображень та координат. Бібліотека OpenCV застосовується для роботи з відео та зображеннями. Фреймворк PyTorch забезпечує виконання нейромережевої моделі. Бібліотека Matplotlib використовується для побудови графіків при аналізі результатів. Стандартний для Python формат JSON застосовується для збереження проміжних даних. Найважливіші з цих технологій детальніше розглянуто в наступному підрозділі.

Окремо слід зазначити інструменти організації середовища розробки. Через те що різні проєкти потребують різних, інколи несумісних версій бібліотек, для ізоляції середовища використано систему керування віртуальними середовищами conda. Вона дозволяє створити для проєкту окреме середовище з власним набором бібліотек фіксованих версій, що забезпечує відтворюваність і запобігає конфліктам залежностей. Це особливо важливо з огляду на те, що модель WildDet3D вимагає суворо визначених версій ключових бібліотек.

2.4. Технології виконання нейромережевих обчислень

Розглянемо детальніше три ключові технології, на яких ґрунтується обчислювальна частина системи, — фреймворк PyTorch, програмно-апаратну платформу CUDA та бібліотеку OpenCV.

PyTorch є фреймворком глибокого навчання з відкритим кодом, який надає засоби для побудови, навчання та виконання нейронних мереж. У цій роботі PyTorch використовується не для навчання, а для виконання вже навченої моделі WildDet3D.

Основною структурою даних PyTorch є тензор — багатовимірний масив чисел, аналогічний масиву NumPy, але з двома ключовими додатковими

можливостями. По-перше, тензор може розміщуватися й оброблятися не лише на центральному, а й на графічному процесорі, що багаторазово пришвидшує обчислення. По-друге, PyTorch підтримує автоматичне диференціювання, необхідне для навчання, яке в задачі цієї роботи не використовується. Модель WildDet3D приймає вхідні дані у вигляді тензорів PyTorch і повертає результат також у тензорах, тому модуль обробки відео взаємодіє з цими структурами безпосередньо, перетворюючи зображення на тензори перед подачею до моделі та результати моделі на звичайні структури даних після виявлення.

CUDA є програмно-апаратною платформою компанії NVIDIA, яка дозволяє виконувати обчислення загального призначення на графічному процесорі. Графічний процесор, на відміну від центрального, містить тисячі простих обчислювальних ядер, здатних виконувати операції паралельно. Нейромережеві обчислення складаються переважно з матричних операцій, які добре піддаються розпаралелюванню, тому їх виконання на графічному процесорі є на порядки швидшим, ніж на центральному.

PyTorch використовує CUDA як основний механізм виконання обчислень на графічному процесорі. Для роботи системи необхідна узгодженість трьох компонентів — версії PyTorch, версії інструментарію CUDA та апаратного графічного процесора з підтримкою CUDA. У роботі використовувався графічний процесор NVIDIA GeForce RTX 2060 з обсягом відеопам'яті 6 гігабайтів. Обмежений обсяг відеопам'яті є суттєвим чинником: модель WildDet3D є великою, тому вимоги до пам'яті при її виконанні наближаються до доступного обсягу, що враховувалося при налаштуванні параметрів обробки.

OpenCV є бібліотекою комп'ютерного зору з відкритим кодом, яка надає велику кількість функцій для роботи із зображеннями та відео. У розробленій системі OpenCV виконує дві основні функції. Перша — декодування відео: бібліотека дозволяє відкрити відеофайл, зчитати його метадані та отримати окремі кадри як числові масиви. Друга — операції із зображеннями: перетворення між колірними просторами, нанесення графічних елементів —

обмежувальних рамок і текстових підписів — на кадри при візуалізації результатів. Таким чином, OpenCV забезпечує роботу системи на вході, відповідаючи за отримання кадрів з відео, та на виході, відповідаючи за візуальне подання результатів.

2.5. Алгоритми зіставлення об'єктів

Завдання зіставлення об'єктів між кадрами вимагає двох складників: міри подібності двох детекцій та алгоритму встановлення відповідності між наборами детекцій. Розглянемо теоретичні основи обох.

Стандартною мірою подібності двох обмежувальних рамок у задачах виявлення об'єктів є перетин над об'єднанням. Ця міра визначається як відношення площі перетину двох прямокутників до площі їх об'єднання. Площа перетину — це площа області, спільної для обох рамок. Площа об'єднання — це площа області, покритої хоча б однією з рамок; вона обчислюється як сума площ двох рамок за відрахуванням площі їх перетину, що враховує спільну область лише один раз.

Значення перетину над об'єднанням змінюється в межах від нуля до одиниці. Нульове значення відповідає рамкам, які не мають спільної площі. Одиничне значення відповідає рамкам, які повністю збігаються. Проміжні значення характеризують ступінь перекриття. Перевага цієї міри полягає в тому, що вона одночасно враховує і взаємне розташування рамок, і їх розміри, причому є безрозмірною величиною, нечутливою до абсолютного масштабу зображення.

У контексті зіставлення об'єктів між кадрами перетин над об'єднанням застосовується так: дві детекції з сусідніх кадрів вважаються тим імовірніше відповідними одному об'єкту, чим вище значення цієї міри для їх рамок. Це спирається на припущення, що за короткий інтервал між кадрами об'єкт зміщується незначно, тому його рамки у сусідніх кадрах суттєво перекриваються.

Встановлення відповідності між детекціями двох кадрів є окремим випадком класичної задачі про призначення. У загальній постановці задача формулюється так: дано дві множини елементів і матрицю вартостей, що задає вартість зіставлення кожного елемента першої множини з кожним елементом другої; потрібно знайти таке взаємно однозначне зіставлення, яке оптимізує сумарну вартість. У випадку зіставлення детекцій елементами є детекції двох кадрів, а вартістю — міра їх подібності.

Задача про призначення має точний розв'язок — угорський алгоритм, який гарантує знаходження глобально оптимального зіставлення. Саме він застосовується у класичних системах відстеження, таких як SORT і DeepSORT [4]. Проте угорський алгоритм є відносно складним у реалізації.

Альтернативою є жадібний алгоритм зіставлення. Він працює простіше: усі можливі пари детекцій упорядковуються за спаданням подібності, після чого пари переглядаються по черзі, і кожна пара приймається, якщо жодна з її детекцій ще не зіставлена. Жадібний алгоритм не гарантує глобально оптимального розв'язку, проте має суттєві переваги — простоту реалізації та прозорість роботи. Крім того, для випадку зіставлення детекцій у сусідніх кадрах різниця між жадібним і оптимальним розв'язками на практиці є незначною: коли об'єкти зміщуються мало, найкращі пари є очевидними й не конкурують між собою за спільні детекції. З огляду на це у розробленій системі застосовано жадібний алгоритм.

2.6. Алгоритми згладжування часових рядів

Координати обмежувальної рамки об'єкта, простежені вздовж послідовності кадрів, утворюють часовий ряд — послідовність значень, упорядкованих у часі. Завдання усунення тремтіння рамки зводиться до згладжування цього часового ряду — придушення дрібних випадкових коливань зі збереженням загальної тенденції. У роботі застосовано два класичні методи згладжування часових рядів.

Метод ковзного середнього полягає в заміні кожного значення ряду на середнє арифметичне значень у вікні, що охоплює певну кількість сусідніх елементів. У застосованому варіанті вікно є симетричним: для обчислення згладженого значення в певній точці беруться елементи як до неї, так і після неї. Для точки з індексом i та вікна розміру 2k+1 згладжене значення обчислюється як середнє елементів з індексами від i−k до i+k.

Розмір вікна є параметром методу, що визначає силу згладжування. Ширше вікно усереднює більшу кількість значень і сильніше придушує коливання, але водночас сильніше згладжує й реальні зміни сигналу. Симетричність вікна забезпечує те, що згладжене значення не запізнюється відносно справжнього й що випадкові відхилення в обидва боки взаємно компенсуються. Платою за симетричність є потреба у знанні майбутніх значень ряду, через що метод застосовний лише до повністю записаного ряду, а не в режимі реального часу.

Метод експоненційного згладжування обчислює згладжене значення рекурентно. Кожне нове згладжене значення є зваженою сумою поточного значення ряду та попереднього згладженого значення за формулою: поточне згладжене значення дорівнює добутку коефіцієнта згладжування на поточне значення ряду плюс добуток різниці між одиницею та коефіцієнтом на попереднє згладжене значення. Коефіцієнт згладжування лежить у межах від нуля до одиниці. Згладжене значення першого елемента приймається рівним самому елементу.

Коефіцієнт згладжування керує силою згладжування. Значення, близьке до одиниці, надає більшу вагу поточному значенню, унаслідок чого згладжування слабке, а реакція на зміни швидка. Значення, близьке до нуля, надає більшу вагу накопиченій історії, унаслідок чого згладжування сильне, а реакція повільна. Назва методу пояснюється тим, що вплив кожного попереднього значення на поточну оцінку спадає експоненційно з віддаленням у минуле.

На відміну від ковзного середнього, експоненційне згладжування використовує лише поточне й попередні значення і не потребує знання майбутніх. Це робить його придатним для застосування в режимі реального часу. Водночас несиметричність методу зумовлює два наслідки: згладжене значення дещо запізнюється відносно справжнього, а одиничні викиди придушуються слабше, ніж симетричним ковзним середнім. Порівняння ефективності обох методів стосовно задачі усунення тремтіння є предметом абляційного дослідження четвертого розділу.

Висновок до розділу 2

У розділі розглянуто технології та алгоритми, використані для розроблення системи. Описано модель WildDet3D — промптовану модель монокулярного тривимірного виявлення, що слугує базовим компонентом системи: розглянуто її архітектуру, формат вхідних і вихідних даних та обмеження, найважливішим з яких є обробка кожного кадру незалежно, що й зумовлює потребу в постобробці. Обґрунтовано вибір мови Python та її екосистеми, а також розглянуто ключові технології виконання нейромережевих обчислень — фреймворк PyTorch, платформу CUDA та бібліотеку OpenCV. Викладено теоретичні основи алгоритмів, застосованих у системі: міру подібності рамок на основі перетину над об'єднанням і жадібний алгоритм зіставлення для задачі про призначення, а також методи ковзного середнього й експоненційного згладжування для усунення тремтіння. Розглянуті технології й алгоритми становлять основу для опису практичної реалізації системи у наступному розділі.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ ПОСТОБРОБКИ ДЕТЕКЦІЙ

3.1. Загальна архітектура системи

Розроблена система є програмним комплексом постобробки результатів 3D-виявлення об'єктів, що надбудовується над попередньо натренованою моделлю WildDet3D. Ключова ідея архітектурного рішення полягає у відокремленні етапу нейромережевого виявлення від етапу часової обробки. Модель WildDet3D застосовується як чорна скринька, що для кожного окремого кадру повертає набір незалежних детекцій, тоді як уся логіка забезпечення часової стабільності реалізована окремими програмними модулями, які не змінюють саму модель.

Такий підхід має кілька переваг. По-перше, він не потребує перенавчання нейромережевої моделі, що є ресурсомісткою операцією і виходить за межі апаратних можливостей типового персонального комп'ютера. По-друге, відокремлення дозволяє замінити базову модель виявлення на будь-яку іншу без переписування решти системи — достатньо, щоб нова модель повертала детекції у сумісному форматі. По-третє, кожен етап обробки можна досліджувати, налаштовувати та оцінювати незалежно від інших, що є важливим для проведення абляційного дослідження.

Система побудована за принципом конвеєра, у якому дані послідовно проходять через ряд етапів обробки, і вихід кожного етапу слугує входом для наступного. Загальну схему потоку даних наведено на рисунку 3.1.

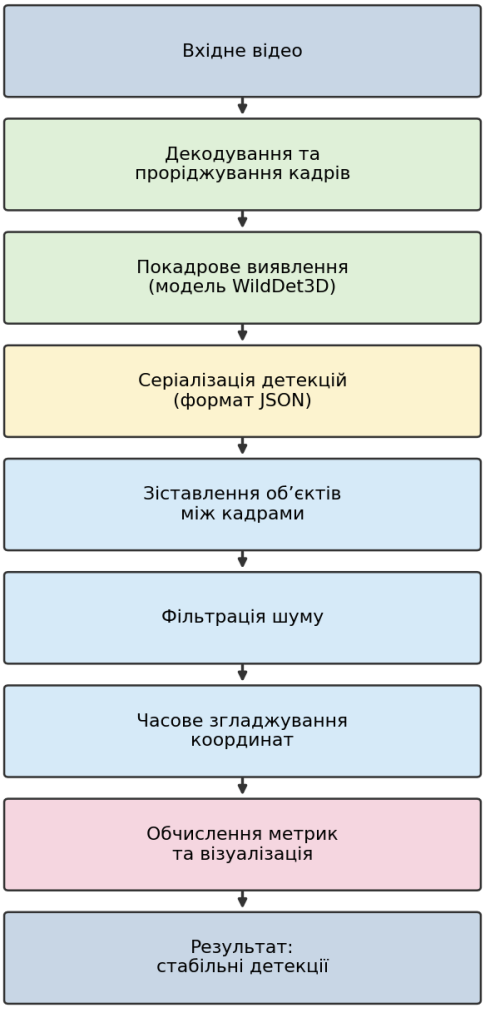


Рисунок 3.1 — Схема конвеєра обробки даних

Конвеєр складається з таких послідовних етапів:

1. Декодування відео. Вхідний відеофайл розбивається на послідовність окремих кадрів. Для скорочення обчислювального навантаження застосовується проріджування кадрів — обробляється не кожен кадр, а кожен N-ий.

2. Покадрове виявлення. Кожен відібраний кадр передається до моделі WildDet3D, яка повертає список детекцій. Кожна детекція містить клас об'єкта, оцінки впевненості, двовимірну та тривимірну обмежувальні рамки.

3. Серіалізація проміжних результатів. Сирі детекції зберігаються у структурованому файлі формату JSON. Цей файл є точкою інтеграції між модулем виявлення та модулями часової обробки.

4. Зіставлення об'єктів між кадрами. Детекціям, що належать одному фізичному об'єкту в різних кадрах, присвоюється спільний ідентифікатор треку.

5. Фільтрація шуму. Треки, які з високою ймовірністю є хибними спрацюваннями моделі, вилучаються з результату.

6. Часове згладжування. Координати обмежувальних рамок усереднюються вздовж треку для усунення тремтіння.

7. Обчислення метрик та візуалізація. Розраховуються кількісні показники якості, а результати відображаються на кадрах відео для візуального аналізу.

Архітектурно перший етап реалізовано у модулі обробки відео, а етапи з четвертого по сьомий — в окремих модулях зіставлення, фільтрації, згладжування, обчислення метрик та візуалізації відповідно. Така організація відповідає принципу єдиної відповідальності, згідно з яким кожен модуль розв'язує одну чітко окреслену задачу.

Важливою архітектурною особливістю є те, що модулі часової обробки не зберігають внутрішнього стану між запусками і не звертаються до нейромережевої моделі. Вони є детермінованими функціями перетворення даних: отримують на вхід структуру з детекціями, повертають оновлену структуру. Це робить систему передбачуваною, спрощує тестування і дозволяє багаторазово застосовувати різні комбінації параметрів обробки до одних і тих самих сирих детекцій.

Програмний код системи організовано у дві директорії. Директорія src містить функціональні модулі — багаторазово використовувані компоненти, які реалізують основну логіку. Директорія scripts містить виконавчі скрипти — обгортки з інтерфейсом командного рядка, які викликають функції модулів із конкретними параметрами. Такий поділ є усталеною практикою організації Python-проєктів і відокремлює логіку від способів її запуску.

3.2. Модуль обробки відео

Модуль обробки відео, реалізований у файлі pipeline.py, відповідає за перші три етапи конвеєра: декодування відео, покадрове виявлення об'єктів та серіалізацію сирих детекцій. Це єдиний модуль системи, який безпосередньо взаємодіє з нейромережевою моделлю WildDet3D.

Функція завантаження моделі отримує шлях до файлу вагових коефіцієнтів моделі та повертає готовий до використання об'єкт моделі. Завантаження моделі є тривалою операцією, яка триває близько сорока секунд, оскільки передбачає зчитування файлу вагових коефіцієнтів обсягом понад чотири гігабайти та побудову структури нейромережі. Через це функція завантаження свідомо відокремлена від функції обробки відео: модель завантажується один раз, а потім передається як параметр. Якби модель завантажувалася всередині обробки кожного відео, то при пакетному опрацюванні кількох відеофайлів час завантаження множився б на їхню кількість.

Функція виявлення на окремому кадрі інкапсулює взаємодію з моделлю для одного кадру. Вона приймає кадр у вигляді тривимірного масиву пікселів та список класів об'єктів, які потрібно знайти. Перед передачею до моделі кадр проходить етап попередньої обробки, під час якого зображення масштабується та доповнюється до розмірів, очікуваних моделлю. Результатом роботи моделі є набір тензорів, які містять обмежувальні рамки, оцінки впевненості та ідентифікатори класів. Ці тензори перетворюються на список словників — структуру даних, зручну для подальшої обробки та серіалізації.

Кожна детекція описується такими полями: назва класу об'єкта згідно з переданим списком класів; числовий індекс класу; оцінка впевненості двовимірного виявлення у діапазоні від нуля до одиниці; оцінка впевненості тривимірного виявлення; двовимірна обмежувальна рамка у форматі чотирьох координат — ліва, верхня, права та нижня межі у пікселях; тривимірна обмежувальна рамка у вигляді десяти чисел — три координати центру, три розміри та чотири компоненти кватерніона орієнтації.

Важливою особливістю моделі WildDet3D є те, що вона працює за принципом керованого підказкою виявлення. На відміну від класичних детекторів, які знаходять усі об'єкти відомих їм категорій, WildDet3D потребує явної вказівки, які саме об'єкти шукати. Така вказівка передається у вигляді списку текстових назв класів. Це дозволяє гнучко налаштовувати систему під конкретну задачу, проте вимагає, щоб перелік цільових класів був визначений заздалегідь.

Функція обробки цілого відео реалізує основний цикл обробки. Вона відкриває відеофайл за допомогою бібліотеки OpenCV, зчитує його метадані — частоту кадрів, загальну кількість кадрів та роздільну здатність, після чого формує перелік індексів кадрів, які підлягають обробці.

Для формування цього переліку застосовується механізм проріджування кадрів, керований параметром кроку. Замість обробки кожного кадру система обробляє лише кожен N-ий. Необхідність такого рішення зумовлена обчислювальною вартістю виявлення: на наявному апаратному забезпеченні обробка одного кадру триває близько п'ятдесяти секунд. Для відео тривалістю кілька секунд при частоті двадцять чотири кадри на секунду повна обробка всіх кадрів зайняла б години. Проріджування дозволяє скоротити час обробки до прийнятного, зберігаючи при цьому достатню для аналізу часової стабільності щільність кадрів. Сусідні кадри відео є висококорельованими, тому пропуск частини з них не призводить до втрати суттєвої інформації про рух об'єктів.

Додатково передбачено параметр обмеження кількості оброблюваних кадрів. Він призначений для швидкої перевірки працездатності системи: під час налагодження можна обробити лише кілька кадрів замість усього відео, що скорочує час очікування з хвилин до десятків секунд.

Цикл обробки послідовно перемотує відео до кожного відібраного кадру, зчитує його, перетворює з кольорового простору BGR, у якому кадри повертає OpenCV, до простору RGB, очікуваного моделлю, та викликає функцію виявлення. Хід обробки відображається індикатором прогресу. Для кожного

кадру зберігається його порядковий номер, часова позначка у секундах та список виявлених детекцій.

Результатом роботи модуля є структурований словник, який містить три групи даних: шлях до вихідного відеофайлу, метадані відео та конфігурацію запуску, а також список покадрових детекцій. Функція збереження записує цей словник у файл формату JSON. Конфігурація запуску включає використаний крок проріджування, перелік цільових класів, кількість оброблених кадрів та витрачений час, що забезпечує відтворюваність експериментів.

3.3. Модуль зіставлення об'єктів між кадрами

Модуль зіставлення, реалізований у файлі matching.py, розв'язує задачу ідентифікації об'єктів у часі. Необхідність цього модуля зумовлена принциповою особливістю роботи моделі WildDet3D: вона обробляє кожен кадр незалежно від інших і не зберігає відомостей про те, які об'єкти були виявлені в попередніх кадрах. Унаслідок цього одна й та сама машина, присутня у кадрах послідовності, отримує в кожному з них власний порядковий номер у списку детекцій, ніяк не пов'язаний з номерами в сусідніх кадрах. Для подальшої часової обробки необхідно встановити відповідність між детекціями, що належать одному фізичному об'єкту. Розв'язання цієї задачі полягає у присвоєнні кожній детекції ідентифікатора треку, спільного для всіх появ об'єкта впродовж відео.

Для визначення того, чи належать дві детекції з сусідніх кадрів одному об'єкту, потрібна кількісна міра подібності їхніх обмежувальних рамок. У роботі використано показник перетину над об'єднанням — стандартну метрику в задачах виявлення об'єктів. Обчислення цього показника реалізовано у відповідній функції. Спочатку визначаються координати прямокутника перетину як максимуми лівих і верхніх меж та мінімуми правих і нижніх меж двох рамок. Якщо отримані розміри перетину виявляються недодатними, прямокутники не перетинаються і метрика дорівнює нулю. В

іншому разі обчислюється площа перетину, після чого площа об'єднання знаходиться як сума площ обох рамок за відрахуванням площі перетину.

Вибір цього показника як міри подібності обґрунтовано кількома міркуваннями. По-перше, метрика одночасно враховує і положення об'єкта, і його розмір, тоді як, наприклад, відстань між центрами рамок ігнорує розмірну складову. По-друге, двовимірні обмежувальні рамки модель WildDet3D повертає стабільніше, ніж тривимірні, тому зіставлення за двовимірними рамками є надійнішим. По-третє, для близьких за часом кадрів зміщення об'єкта незначне, а отже рамки одного об'єкта у сусідніх кадрах сильно перекриваються і мають високе значення показника.

Функція зіставлення детекцій двох кадрів встановлює відповідність між списками детекцій двох сусідніх кадрів. Алгоритм працює у три кроки. Спочатку формується перелік усіх можливих пар детекцій із попереднього та поточного кадрів, для кожної з яких обчислюється показник перетину над об'єднанням. Пари, у яких детекції належать різним класам або значення показника нижче встановленого порога, відкидаються. Потім перелік пар-кандидатів сортується за спаданням показника, унаслідок чого найкращі відповідності опиняються на початку. Нарешті виконується жадібний відбір: пари переглядаються по черзі, і пара приймається лише тоді, коли жодна з її детекцій ще не була використана в іншій парі.

Жадібна стратегія відбору є простим і обчислювально ефективним наближенням до задачі оптимального паросполучення у дводольному графі. На відміну від точних методів, як-от угорський алгоритм, жадібний підхід не гарантує глобально оптимального розподілу, проте на практиці для близьких кадрів дає той самий результат: коли об'єкти зміщуються незначно, найкращі пари є очевидними і не конкурують між собою. Простота реалізації та прозорість роботи алгоритму в цьому випадку переважають теоретичну перевагу точних методів.

Поріг показника перетину над об'єднанням встановлено на рівні 0,3. Це значення є компромісом: надто високий поріг призвів би до розриву треків при

швидкому русі об'єкта, коли перекриття рамок зменшується, а надто низький — до хибного об'єднання різних об'єктів, розташованих поряд. Додатково передбачено можливість зіставляти лише детекції одного класу, що запобігає, наприклад, помилковому ототожненню машини з пішоходом.

Функція присвоєння ідентифікаторів треків застосовує попарне зіставлення до всієї послідовності кадрів. Детекціям першого кадру присвоюються початкові ідентифікатори треків. Для кожного наступного кадру виконується зіставлення з попереднім: детекції, для яких знайдено відповідність, успадковують ідентифікатор треку від відповідної детекції попереднього кадру, а детекції, що залишилися без пари, отримують нові, раніше не використані ідентифікатори.

Поява детекції без відповідності має дві можливі причини. Перша — це справді новий об'єкт, який щойно з'явився у полі зору камери. Друга — хибне спрацювання моделі, тобто фантомна детекція, якої не існувало в попередньому кадрі. Розрізнити ці два випадки на етапі зіставлення неможливо, проте подальша обробка дозволяє їх відокремити: фантомні детекції утворюють короткі треки завдовжки один-два кадри, тоді як реальні об'єкти простежуються впродовж багатьох кадрів. Ця властивість є основою для роботи модуля фільтрації.

Функція реалізована так, що не змінює вхідних даних, а повертає їхню оновлену копію. Це дозволяє багаторазово застосовувати зіставлення з різними значеннями порога до одних і тих самих сирих детекцій.

3.4. Модуль фільтрації шуму

Модуль фільтрації, реалізований у файлі filtering.py, призначений для вилучення з результатів виявлення хибних спрацювань моделі. Робота модуля ґрунтується на спостереженні, сформульованому в попередньому підрозділі: фантомні детекції, на відміну від реальних об'єктів, не виявляють часової стійкості. Хибне спрацювання моделі зазвичай виникає на окремому кадрі через випадковий збіг текстур або особливості освітлення і зникає в

наступному кадрі, тоді як справжній об'єкт послідовно виявляється впродовж тривалого проміжку часу. Отже, маючи присвоєні ідентифікатори треків, можна відрізнити шум від корисного сигналу за характеристиками треку загалом, а не окремої детекції.

Передумовою фільтрації є обчислення зведених характеристик кожного треку. Відповідна функція групує всі детекції за ідентифікаторами треків і для кожного треку обчислює три показники: довжину треку, тобто кількість кадрів, у яких присутній об'єкт; середню оцінку впевненості двовимірного виявлення; середню оцінку впевненості тривимірного виявлення. Окремо зберігається клас об'єкта, який у межах одного треку залишається незмінним.

У роботі реалізовано три стратегії фільтрації, які відрізняються критерієм відбору треків.

Перша стратегія — фільтрація за мінімальною довжиною треку — вилучає всі детекції, чий трек коротший за задану мінімальну довжину. Ця стратегія безпосередньо реалізує ідею часової стійкості: об'єкт, виявлений лише в одному чи двох кадрах, з високою ймовірністю є фантомом. Вибір порогової довжини є компромісом. Поріг у два кадри є м'яким і зберігає короткі треки, які можуть відповідати реальним об'єктам, що ненадовго з'явилися у кадрі. Поріг у три кадри є суворішим і вимагає, щоб об'єкт був підтверджений щонайменше трьома послідовними виявленнями.

Друга стратегія — фільтрація за середньою впевненістю — вилучає треки, у яких середня оцінка впевненості нижча за заданий поріг. Ця стратегія доповнює першу, оскільки охоплює випадок, який фільтрація за довжиною не виявляє: інколи модель стабільно бачить неіснуючий об'єкт на одному й тому самому місці впродовж багатьох кадрів — наприклад, на елементі фасаду будівлі або дорожньому знаку. Такий фантом утворює довгий трек і проходить фільтр довжини, проте його оцінки впевненості залишаються низькими. Усереднення оцінки за всім треком робить критерій стійкішим, ніж перевірка впевненості окремої детекції.

Третя стратегія — комбінована фільтрація — поєднує обидва критерії: трек зберігається лише тоді, коли він одночасно задовольняє вимогу мінімальної довжини і вимогу мінімальної середньої впевненості. Ця стратегія є основною робочою стратегією системи. Її перевага полягає в тому, що вона усуває фантоми обох типів — і короткочасні, і стійкі низькоймовірні, — тоді як кожен з окремих фільтрів охоплює лише один тип. Значення порогів комбінованого фільтра підбираються емпірично і є предметом абляційного дослідження, описаного в четвертому розділі.

Усі функції фільтрації побудовані за єдиним принципом: спочатку обчислюється статистика треків, потім формується множина ідентифікаторів треків, що задовольняють критерій, після чого з кожного кадру вилучаються детекції, чиї треки до цієї множини не входять. Як і модуль зіставлення, функції фільтрації не змінюють вхідних даних, а повертають оновлену копію.

3.5. Модуль часового згладжування

Модуль часового згладжування, реалізований у файлі smoothing.py, призначений для зменшення тремтіння обмежувальних рамок. Тремтінням називають дрібні випадкові коливання координат рамки від кадру до кадру, не зумовлені реальним рухом об'єкта. Це явище є наслідком того, що модель обробляє кожен кадр незалежно: навіть для нерухомого об'єкта оцінки координат у сусідніх кадрах дещо відрізняються через стохастичну природу нейромережевого виявлення. На статичному зображенні таке коливання непомітне, проте при відтворенні відео воно проявляється як помітне дрижання рамки навколо об'єкта і погіршує якість сприйняття результату. Окрім візуального ефекту, тремтіння спотворює оцінки руху об'єкта, якщо координати треку використовувати для обчислення швидкості чи траєкторії.

Ідея усунення тремтіння полягає в усередненні координат рамки вздовж треку. Оскільки тремтіння є випадковим коливанням навколо справжнього положення об'єкта, усереднення кількох послідовних значень взаємно компенсує ці коливання і наближає оцінку до дійсного положення. У модулі

реалізовано два класичні методи згладжування часових рядів — ковзне середнє та експоненційне згладжування.

Метод ковзного середнього обчислює для кожної детекції треку згладжене значення координат як середнє арифметичне координат детекцій у вікні, що охоплює задану кількість сусідніх кадрів симетрично — частину до поточного кадру і частину після нього. На краях треку, де повне вікно не вміщується, воно усікається до наявних детекцій. Розмір вікна є параметром методу: більше вікно дає сильніше згладжування, проте знижує чутливість до реального руху об'єкта.

Суттєвою властивістю ковзного середнього у такій реалізації є симетричність: при обчисленні згладженого значення враховуються як попередні, так і наступні кадри. Це забезпечує якісне усунення коливань, оскільки випадкові відхилення в обидва боки взаємно компенсуються. Платою за це є те, що метод потребує наявності всієї послідовності детекцій треку до початку обробки, тобто не може застосовуватися у режимі реального часу.

Метод експоненційного згладжування, на відміну від ковзного середнього, не використовує фіксованого вікна, а обчислює згладжене значення рекурентно: кожне нове значення є зваженою сумою поточного сирого значення координат і попереднього згладженого значення. Вага розподіляється коефіцієнтом згладжування у діапазоні від нуля до одиниці. Коефіцієнт, близький до одиниці, надає перевагу поточному значенню і дає слабке згладжування; коефіцієнт, близький до нуля, надає перевагу накопиченій історії і дає сильне згладжування. Згладжене значення для першої детекції треку приймається рівним її сирому значенню, оскільки попередньої історії ще немає.

Експоненційне згладжування залежить виключно від поточного та попередніх кадрів і не потребує знання майбутніх. Завдяки цьому воно придатне для застосування в режимі реального часу. Водночас через відсутність симетрії воно слабше згладжує одиничні викиди, ніж ковзне

середнє, і вносить незначне запізнення згладженого значення відносно реального руху.

Функція застосування згладжування застосовує обраний метод до всіх треків послідовності. Вона групує детекції за ідентифікаторами треків і обробляє кожен трек незалежно від інших, оскільки згладжування має сенс лише в межах одного об'єкта. Координати обмежувальних рамок — ліва, верхня, права та нижня межі — згладжуються незалежно одна від одної.

Принципово важливою рисою реалізації є те, що згладжені координати записуються в окреме поле детекції, а не замінюють вихідні. Завдяки цьому кожна детекція одночасно зберігає і сирі, і згладжені координати. Це рішення є необхідним для проведення дослідження: воно дозволяє безпосередньо порівнювати стан до і після згладжування на одних і тих самих даних, обчислювати метрики тремтіння для обох варіантів та оцінювати ефект кожного методу й набору параметрів.

3.6. Модуль обчислення метрик

Модуль обчислення метрик, реалізований у файлі metrics.py, призначений для кількісного оцінювання часової стабільності детекцій. Якщо попередні модулі перетворюють дані, то цей модуль вимірює якість результату, надаючи об'єктивну числову основу для порівняння різних методів і параметрів обробки.

Основною метрикою, запропонованою в роботі, є показник тремтіння, який кількісно характеризує нестабільність положення обмежувальної рамки вздовж треку. Метрику обчислює відповідна функція. Для кожного треку розглядаються всі пари сусідніх кадрів, у яких присутній об'єкт. Для кожної такої пари обчислюється середнє абсолютне зміщення чотирьох координат рамки між кадрами. Тремтіння треку визначається як середнє цих зміщень за всіма парами сусідніх кадрів. Підсумкове значення виражається в пікселях і має наочну інтерпретацію: воно показує, наскільки в середньому зсувається межа рамки від кадру до кадру.

Логіка метрики спирається на таке міркування. Для нерухомого або повільного об'єкта справжнє зміщення між сусідніми кадрами близьке до нуля, тому будь-яке виміряне зміщення координат є переважно тремтінням. Чим менше значення метрики, тим стабільніший трек. Застосування згладжування має зменшувати це значення, і величина зменшення є прямою кількісною оцінкою ефективності згладжування.

Функція побудована так, що приймає назву поля з координатами рамки як параметр. Це дозволяє обчислювати метрику як для сирих координат, так і для згладжених, використовуючи той самий код. Саме на цій можливості ґрунтується методика порівняння до і після: тремтіння обчислюється двічі — для вихідних і для згладжених координат, — а їх відношення дає відсоток зменшення тремтіння. Треки завдовжки менше двох кадрів при обчисленні пропускаються, оскільки для них поняття зміщення між кадрами не визначене.

Допоміжна функція обчислює зведені характеристики за всіма треками — кількість треків, а також середнє, максимальне й мінімальне значення тремтіння. Ці агреговані показники використовуються для підсумкового порівняння методів обробки в межах усього відео.

3.7. Модуль візуалізації

Модуль візуалізації, реалізований у файлі visualizer.py, призначений для відображення результатів виявлення на кадрах відео. Числові метрики дають об'єктивну оцінку якості, проте візуальне подання необхідне для якісного аналізу — воно дозволяє безпосередньо побачити, які об'єкти виявлено, наскільки точно розташовані рамки та як фільтрація змінює результат.

Модуль реалізує власну візуалізацію, незалежну від засобів, наявних у складі WildDet3D. Таке рішення зумовлене тим, що штатна функція візуалізації моделі потребує справжніх параметрів калібрування камери для коректної проєкції тривимірних рамок, які для довільного відео невідомі. Для задач цієї роботи достатньо відображення двовимірних рамок із підписами, що робить власну реалізацію простішою та більш керованою.

Функція витягування кадру дістає з відеофайлу кадр за його порядковим номером, що потрібно для накладання детекцій на оригінальне зображення. Функція нанесення детекцій наносить на копію кадру обмежувальні рамки та текстові підписи. Кожна рамка супроводжується підписом, який може містити ідентифікатор треку, назву класу та оцінку впевненості. Для забезпечення читабельності підпис розміщується на кольоровій підкладці.

Для розрізнення треків застосовано колірне кодування: колір рамки визначається ідентифікатором треку, причому той самий трек зберігає той самий колір упродовж усього відео. Це дозволяє візуально простежити об'єкт між кадрами і перевірити коректність роботи модуля зіставлення. Кольори обираються з наперед визначеної палітри.

Функція об'єднання кадрів з'єднує два кадри в одне зображення для безпосереднього порівняння. Вона використовується для побудови ілюстрацій у форматі до і після: на одній половині відображаються сирі детекції, на іншій — результат після фільтрації, що наочно демонструє ефект обробки.

3.8. Формат збереження даних

Усі модулі системи обмінюються даними через єдиний формат — структуру у форматі JSON. Вибір цього формату зумовлений його текстовою природою, придатністю для читання людиною, незалежністю від мови програмування та наявністю вбудованої підтримки в Python.

Структура файлу складається з трьох частин. Перша містить метадані відео — частоту кадрів, роздільну здатність, загальну кількість кадрів і тривалість. Друга містить конфігурацію обробки: на цьому рівні послідовно накопичуються параметри всіх застосованих етапів — крок проріджування і перелік класів від модуля обробки відео, поріг зіставлення від модуля зіставлення, пороги фільтрації, метод і параметри згладжування. Завдяки цьому за готовим файлом завжди можна відновити повну історію його обробки. Третя частина — масив кадрів, кожен з яких містить порядковий номер, часову позначку і список детекцій.

Ключовою властивістю формату є його інкрементність: кожен модуль не перезаписує файл, а доповнює структуру детекцій новими полями. Сирі детекції містять клас, оцінки впевненості та координати рамок. Модуль зіставлення додає до кожної детекції ідентифікатор треку. Модуль згладжування додає поле згладжених координат, зберігаючи вихідні. Така організація забезпечує простежуваність обробки та можливість порівняння проміжних результатів на будь-якому етапі.

Висновок до розділу 3

У розділі описано розробку програмної системи постобробки результатів 3D-виявлення об'єктів. Систему побудовано за модульним принципом у вигляді конвеєра обробки даних, у якому етап нейромережевого виявлення відокремлено від етапів забезпечення часової стабільності. Реалізовано шість функціональних модулів: модуль обробки відео, що виконує покадрове виявлення за допомогою моделі WildDet3D; модуль зіставлення, що ідентифікує об'єкти у часі за метрикою перетину над об'єднанням; модуль фільтрації, що вилучає хибні детекції за характеристиками треків; модуль згладжування, що зменшує тремтіння рамок методами ковзного середнього та експоненційного згладжування; модуль обчислення метрик, що кількісно оцінює часову стабільність; модуль візуалізації для якісного аналізу результатів. Обмін даними між модулями організовано через єдиний інкрементний формат JSON. Запропонована архітектура забезпечує незалежність етапів обробки, відтворюваність експериментів і можливість дослідження впливу параметрів, що використовується в наступному розділі.

РОЗДІЛ 4
ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1. Методика експериментів і тестові дані

Метою експериментальної частини роботи є кількісна та якісна оцінка ефективності розробленої системи постобробки. Дослідження має відповісти на три питання: наскільки ефективно система вилучає хибні детекції; наскільки зменшується тремтіння обмежувальних рамок після згладжування; який із двох реалізованих методів згладжування є кращим для поставленої задачі.

Експерименти проведено на відеозаписі типової міської сцени. Відео має тривалість 5,4 секунди, частоту 24 кадри на секунду та роздільну здатність 1280 на 720 пікселів, що відповідає 129 кадрам загалом. На сцені присутні припарковані та рухомі автомобілі, а також пішоходи на тротуарі й пішохідному переході. Така сцена є репрезентативною для задачі виявлення об'єктів у природному середовищі: вона містить кілька об'єктів цільових класів, розташованих на різній відстані від камери, часткові перекриття об'єктів та неоднорідний фон із елементів міської забудови.

Дослідження має характер пілотного експерименту: оцінювання виконано на одному відеозаписі з метою детального аналізу поведінки системи. Розширення експериментальної вибірки на сцени з різними умовами освітлення, щільністю об'єктів та характером руху камери є окремим напрямом подальшої роботи, що обговорюється в підрозділі 4.6.

З огляду на обчислювальну вартість виявлення застосовано проріджування кадрів із кроком п'ять, унаслідок чого з відео відібрано та оброблено 26 кадрів. Цей крок забезпечує компроміс між часом обробки та щільністю часової вибірки: інтервал між сусідніми обробленими кадрами становить близько 0,21 секунди, що достатньо для простеження руху об'єктів і водночас зменшує загальний час обробки приблизно у п'ять разів. Як цільові

класи задано дві категорії — автомобіль та пішохід, що відповідають об'єктам, наявним на сцені.

Зіставлення об'єктів між кадрами виконувалося з порогом перетину над об'єднанням 0,3. Фільтрація застосовувалася у комбінованому режимі з мінімальною довжиною треку три кадри та мінімальною середньою впевненістю 0,4. Згладжування досліджувалося для обох реалізованих методів: ковзного середнього з розміром вікна п'ять кадрів та експоненційного згладжування з коефіцієнтом 0,4.

Експерименти виконано на персональному комп'ютері під керуванням операційної системи Windows 11 із графічним процесором NVIDIA GeForce RTX 2060 обсягом відеопам'яті 6 гігабайтів. Виявлення об'єктів моделлю WildDet3D на одному кадрі тривало в середньому близько 50 секунд. Етапи часової обробки — зіставлення, фільтрація, згладжування та обчислення метрик — виконуються на центральному процесорі і потребують часток секунди для всього відео, оскільки оперують уже готовими детекціями, а не зображеннями.

Оцінювання системи побудоване на принципі порівняння стану до і після обробки. Сирі детекції, отримані від моделі, розглядаються як базовий рівень. До них послідовно застосовуються етапи обробки, і результат кожного етапу порівнюється з базовим рівнем за відповідними показниками. Для етапу фільтрації показником є кількість детекцій і треків, що залишилися. Для етапу згладжування показником є метрика тремтіння. Така методика дозволяє ізолювати внесок кожного етапу і отримати кількісну оцінку його ефективності.

4.2. Метрика часової стабільності

Для кількісної оцінки часової стабільності детекцій у роботі запроваджено метрику тремтіння. Необхідність власної метрики зумовлена тим, що стандартні метрики виявлення об'єктів, такі як середня точність, оцінюють правильність виявлення на окремих кадрах, але не враховують

узгодженості результатів у часі. Виявлення може бути точним на кожному окремому кадрі і водночас нестабільним у послідовності, якщо положення рамки коливається від кадру до кадру. Саме цю властивість — часову нестабільність — і покликана вимірювати запропонована метрика.

Тремтіння визначається в межах окремого треку як середня величина зміщення координат обмежувальної рамки між сусідніми кадрами. Для пари сусідніх кадрів треку обчислюється середнє абсолютне зміщення чотирьох координат рамки — лівої, верхньої, правої та нижньої меж. Тремтіння треку є середнім таких зміщень за всіма парами сусідніх кадрів, у яких присутній об'єкт. Значення метрики виражається в пікселях.

Метрика має наочну інтерпретацію: вона показує, наскільки в середньому зміщується межа обмежувальної рамки від одного кадру до наступного. Чим менше значення, тим стабільніший трек і тим менш помітне дрижання рамки при відтворенні відео.

В основі метрики лежить таке припущення. Реальне зміщення об'єкта між двома кадрами, відібраними з невеликим інтервалом, є незначним і плавним. Якщо ж виміряне зміщення координат рамки велике або хаотичне, воно зумовлене переважно не рухом об'єкта, а нестабільністю виявлення. Для повністю нерухомого об'єкта будь-яке ненульове зміщення координат є виключно тремтінням. Для повільно рухомого об'єкта виміряне зміщення є сумою невеликого плавного руху і випадкового тремтіння, причому згладжування зменшує саме другу складову, майже не впливаючи на першу.

Слід зазначити обмеження метрики: для об'єкта, що швидко рухається, велике значення метрики є очікуваним і не свідчить про нестабільність. Тому метрика коректно відображає саме тремтіння лише для нерухомих або повільних об'єктів, яких на типовій міській сцені більшість — припарковані автомобілі, пішоходи, що йдуть. Це обмеження враховується при інтерпретації результатів.

Метрика обчислюється модулем метрик. Важливою рисою реалізації є те, що функція приймає назву поля з координатами як параметр і тому може

обчислювати тремтіння як для сирих, так і для згладжених координат. Завдяки цьому метрика застосовується двічі — до базового рівня і до результату згладжування, — а відношення двох значень дає кількісну оцінку ефективності згладжування у відсотках.

Запропонована метрика є простою в обчисленні, прозорою в інтерпретації і не потребує еталонної розмітки даних, оскільки оцінює внутрішню узгодженість результатів, а не їх відповідність істинним значенням. Це робить її зручним інструментом для порівняння конфігурацій обробки в умовах, коли еталонна тривимірна розмітка відео відсутня.

4.3. Дослідження ефективності фільтрації шуму

Дослідження фільтрації має на меті встановити, якою мірою система здатна вилучати хибні детекції моделі, зберігаючи при цьому реальні об'єкти.

На першому етапі до 26 оброблених кадрів застосовано модуль зіставлення. Загалом модель WildDet3D повернула на цих кадрах 281 детекцію. Після присвоєння ідентифікаторів треків ці детекції об'єдналися у 57 унікальних треків. Уже сам цей факт є показовим: середня довжина треку становить менше п'яти кадрів, тоді як відео містить 26 кадрів, отже багато треків охоплюють лише незначну частину послідовності.

Розподіл треків за довжиною наведено на рисунку 4.1. Розподіл є вираженим чином двомодальним. Переважна частина треків — 22 з 57 — має довжину лише один кадр. Це означає, що відповідна детекція з'явилася в одному кадрі і не була підтверджена в жодному сусідньому. Ще сім треків мають довжину два кадри. Водночас на протилежному кінці розподілу присутня група з шести треків довжиною 16 кадрів і семи треків довжиною дев'ять кадрів — це об'єкти, що стабільно простежуються впродовж значної частини відео.

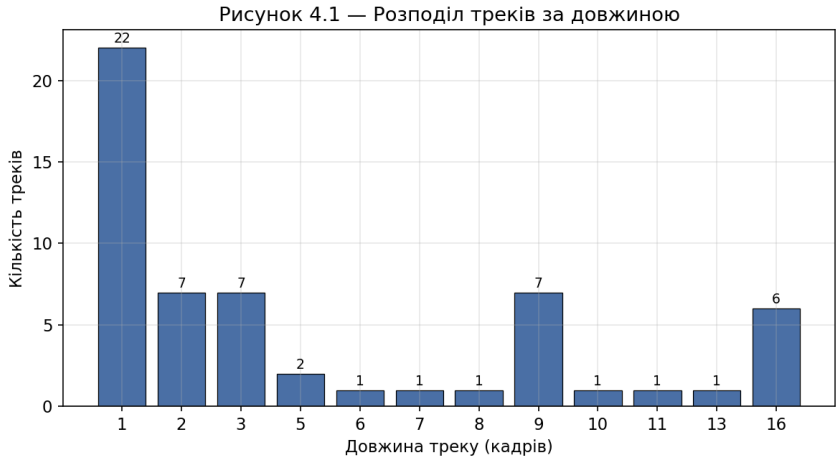


Рисунок 4.1 — Розподіл треків за довжиною

Така структура розподілу безпосередньо підтверджує припущення, покладене в основу модуля фільтрації. Реальні об'єкти сцени — припарковані автомобілі та пішоходи — утворюють довгі треки, оскільки послідовно виявляються кадр за кадром. Хибні спрацювання моделі натомість утворюють однокадрові й двокадрові треки, бо виникають на окремому кадрі через випадковий збіг візуальних ознак і зникають у наступному. Однокадрові й двокадрові треки складають 29 із 57, тобто більше половини всіх треків, що дає кількісне уявлення про рівень шуму у сирому виході моделі.

До зіставлених детекцій застосовано комбінований фільтр із порогами: мінімальна довжина треку — три кадри, мінімальна середня впевненість — 0,4. Результати наведено в таблиці 4.1.

Таблиця 4.1 — Результати фільтрації шуму

Показник	**До фільтрації**	**Після фільтрації**	**Зміна**
Кількість треків	57	20	−64,9 %
Кількість детекцій	281	191	−32,0 %

Фільтрація вилучила 37 із 57 треків, тобто майже дві третини. При цьому в перерахунку на окремі детекції усунено 90 із 281, тобто 32 відсотки. Різниця

між цими відсотками пояснюється тим, що вилучені треки є переважно короткими: 37 вилучених треків містять загалом лише 90 детекцій, тоді як 20 треків, що залишилися, містять 191 детекцію. Іншими словами, фільтрація усунула велику кількість треків, кожен з яких робив незначний внесок у загальну кількість детекцій, і зберегла невелику кількість треків, кожен з яких є насиченим детекціями.

Серед 20 треків, що пройшли фільтр, переважають довгі треки — шість завдовжки 16 кадрів та сім завдовжки дев'ять кадрів, що відповідають основним об'єктам сцени. Збереглося також кілька коротших треків довжиною три-п'ять кадрів із високою середньою впевненістю — це реальні об'єкти, що перебували в полі зору камери нетривалий час. Жоден однокадровий чи двокадровий трек фільтр не подолав, що відповідає очікуваній поведінці.

Якісну ілюстрацію результату фільтрації наведено на рисунку 4.2, де для одного з кадрів відео зіставлено сирі детекції та детекції після фільтрації. На зображенні з сирими детекціями присутні рамки з низькою впевненістю, розташовані на ділянках фону, а також дублюючі рамки на одному об'єкті. На зображенні після фільтрації залишилися лише рамки, що відповідають реальним автомобілям і пішоходам. Кількість детекцій на цьому кадрі зменшилася з 10 до 5.

Рисунок 4.2 — Порівняння детекцій до та після фільтрації на окремому кадрі

Отримані результати свідчать, що запропонований підхід до фільтрації ефективно вилучає хибні детекції. Усунення майже двох третин треків при збереженні всіх довгих треків, що відповідають реальним об'єктам, підтверджує коректність припущення про зв'язок між часовою стійкістю треку та достовірністю детекції.

4.4. Дослідження ефективності згладжування

Дослідження згладжування має на меті оцінити, наскільки застосування часового згладжування зменшує тремтіння обмежувальних рамок. На цьому етапі обробці підлягали зіставлені детекції до застосування фільтрації, оскільки метрика тремтіння обчислюється для треків незалежно від того, чи є вони реальними об'єктами, чи фантомами, а більша кількість треків дає повніше уявлення про поведінку методу.

Спершу метрику тремтіння обчислено для сирих координат, отриманих безпосередньо від моделі. З 57 треків метрика визначена для 35 — це треки завдовжки щонайменше два кадри. Середнє тремтіння за цими треками склало 8,97 пікселя. Це означає, що в середньому кожна межа обмежувальної рамки зміщується майже на дев'ять пікселів від кадру до кадру. Максимальне значення тремтіння серед треків досягло 86,92 пікселя, мінімальне — 1,70 пікселя.

Великий розкид значень пояснюється неоднорідністю треків. Мінімальне тремтіння відповідає стабільно виявленому нерухомому об'єкту. Максимальне значення належить короткому треку, що, найімовірніше, є наслідком хибного зіставлення двох різних фантомних детекцій або різкого стрибка рамки — такі треки згодом вилучаються фільтрацією, але на етапі вимірювання базового рівня вони враховуються.

До зіставлених детекцій застосовано згладжування методом ковзного середнього з розміром вікна п'ять кадрів. Після згладжування метрику тремтіння обчислено повторно — цього разу для згладжених координат. Середнє тремтіння зменшилося з 8,97 до 1,32 пікселя, що відповідає скороченню на 85,3 відсотка. Максимальне тремтіння зменшилося з 86,92 до 7,22 пікселя.

Порівняння тремтіння для окремих треків до і після згладжування наведено на рисунку 4.3. Для кожного з треків з найбільшим початковим тремтінням показано два значення — до і після обробки. З діаграми видно, що

згладжування зменшує тремтіння для всіх без винятку треків, причому найсильніший ефект досягається саме на треках із найбільшим початковим тремтінням.

Рисунок 4.3 — Тремтіння по треках до та після згладжування

Для наочної ілюстрації ефекту згладжування на рисунку 4.4 показано зміну однієї координати обмежувальної рамки — лівої межі — вздовж одного з довгих треків. Сира координата зображена ламаною лінією з помітними стрибками: між окремими кадрами її значення змінюється на десятки пікселів, хоча об'єкт рухається плавно. Згладжена координата зображена лінією, яка зберігає загальну тенденцію руху, але позбавлена різких стрибків. Особливо виразно це видно на перших кадрах треку, де сира координата стрибкоподібно змінюється, а згладжена змінюється поступово.

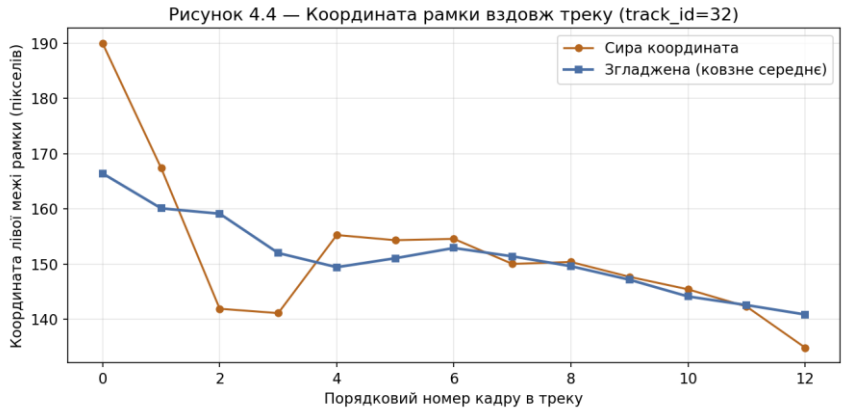


Рисунок 4.4 — Зміна координати рамки вздовж треку до та після згладжування

При аналізі результатів виявлено особливість, пов'язану зі взаємодією розміру вікна і довжини треку. Якщо довжина треку менша або дорівнює розміру вікна згладжування, то вікно охоплює весь трек, і всі згладжені значення координат стають однаковими — рівними середньому за всім треком. Метрика тремтіння для такого треку набуває нульового значення. Це нульове значення є артефактом методу, а не свідченням ідеальної стабільності: воно означає лише, що короткий трек повністю усереднено в одну точку. При інтерпретації результатів та при обчисленні зведених показників цю особливість слід враховувати, а як напрям удосконалення доцільно обмежити обчислення метрики треками, довжина яких перевищує розмір вікна.

Отримані результати свідчать, що часове згладжування методом ковзного середнього суттєво — більш ніж у шість разів — зменшує тремтіння обмежувальних рамок. Це підтверджує ефективність запропонованого підходу до підвищення часової стабільності детекцій.

4.5. Абляційне дослідження методів згладжування

Абляційне дослідження має на меті порівняти два реалізовані методи згладжування — ковзне середнє та експоненційне згладжування — і визначити, який із них краще підходить для задачі підвищення часової стабільності детекцій. Суть абляційного підходу полягає в тому, щоб за інших однакових умов змінювати лише один компонент системи і спостерігати за зміною результату. У цьому випадку незмінними залишаються тестові дані та результат зіставлення, а змінюється метод згладжування.

Обидва методи застосовано до одного й того самого набору зіставлених детекцій. Для ковзного середнього використано вікно розміром п'ять кадрів, для експоненційного згладжування — коефіцієнт згладжування 0,4. Ефективність кожного методу оцінено за середнім тремтінням, обчисленим за тими самими 35 треками, що й базовий рівень. Як базовий рівень для порівняння взято тремтіння сирих координат без згладжування. Результати наведено в таблиці 4.2 та на рисунку 4.5.

Таблиця 4.2 — Порівняння методів згладжування

Метод	Середнє тремтіння, пікс.	Максимальне тремтіння, пікс.	Зменшення
Без обробки (базовий рівень)	8,97	86,92	—
Експоненційне згладжування	4,41	43,80	50,9 %
Ковзне середнє	1,32	7,22	85,3 %

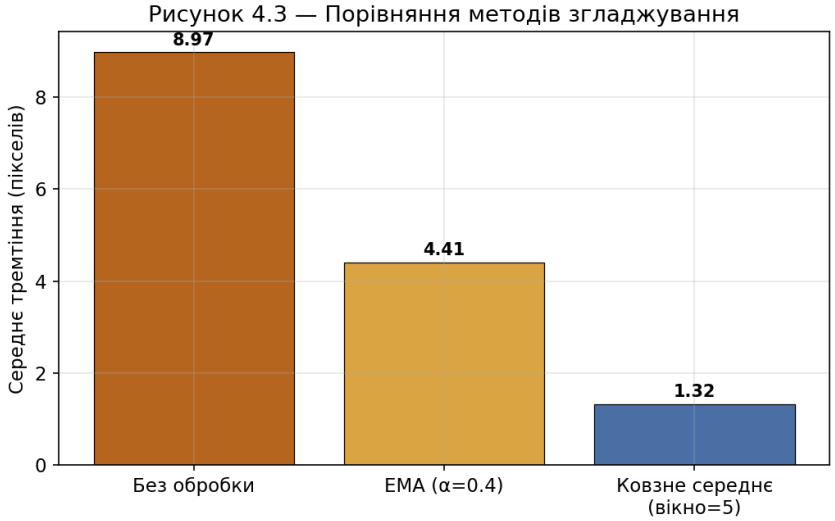


Рисунок 4.3 — Порівняння методів згладжування

Рисунок 4.5 — Порівняння середнього тремтіння для різних методів згладжування

Обидва методи зменшують тремтіння, проте їхня ефективність суттєво різниться. Експоненційне згладжування зменшило середнє тремтіння приблизно вдвічі — на 50,9 відсотка. Ковзне середнє виявилося значно ефективнішим, зменшивши тремтіння на 85,3 відсотка. Подібну перевагу ковзного середнього видно і за максимальним тремтінням: воно скоротилося до 7,22 пікселя проти 43,80 пікселя для експоненційного згладжування.

Така різниця в ефективності пояснюється принциповою відмінністю двох методів. Ковзне середнє у застосованій реалізації є симетричним: при обчисленні згладженого значення для кадру воно враховує як попередні, так і наступні кадри. Завдяки цьому випадкові відхилення координат в обидва боки

від справжнього положення взаємно компенсуються, і одиничний стрибок координати усереднюється з сусідніми коректними значеннями з обох сторін.

Експоненційне згладжування натомість є несиметричним: воно враховує лише поточний і попередні кадри. Згладжене значення фактично наздоганяє сирий сигнал із певним запізненням. Одиничний стрибок координати таке згладжування лише послаблює, але не компенсує повністю, оскільки наступних кадрів, які могли б врівноважити стрибок, метод не бачить.

Отже, перевага ковзного середнього зумовлена доступом до майбутніх кадрів. Це водночас вказує на межу застосовності методу: ковзне середнє потребує наявності всієї послідовності детекцій до початку обробки і тому не може застосовуватися в режимі реального часу. Експоненційне згладжування, попри слабший результат, такого обмеження не має і залишається єдиним з двох методів, придатним для онлайн-обробки відеопотоку.

Для задачі цієї роботи, у якій обробка виконується в пакетному режимі над заздалегідь записаним відео, обмеження ковзного середнього не є суттєвим, а його значно вища ефективність робить його кращим вибором. Тому ковзне середнє з вікном п'ять кадрів прийнято як основний метод згладжування системи. Експоненційне згладжування зберігає своє значення як альтернатива для сценаріїв обробки в реальному часі, що є можливим напрямом подальшого розвитку системи.

4.6. Аналіз обмежень і напрямки вдосконалення

Проведене дослідження підтвердило працездатність та ефективність розробленої системи, проте водночас виявило низку обмежень, які слід враховувати при оцінці результатів і які визначають напрямки подальшої роботи.

Оцінювання системи проведено на одному відеозаписі. Цього достатньо для детального аналізу поведінки системи та демонстрації принципової ефективності запропонованого підходу, проте недостатньо для статистично обґрунтованих висновків про її роботу в усьому розмаїтті умов. Різні сцени

відрізняються освітленням, щільністю об'єктів, характером і швидкістю руху об'єктів та камери. Розширення експериментальної вибірки на репрезентативний набір відео з різними умовами є першочерговим напрямом подальшої роботи.

Запропонована метрика тремтіння оцінює внутрішню узгодженість детекцій у часі, але не їх відповідність істинному положенню об'єктів. Тому вона здатна виявити нестабільність, але не здатна виявити систематичну похибку: якщо рамка стабільно зміщена відносно об'єкта, тремтіння буде низьким, попри неточність виявлення. Повноцінна оцінка точності потребує відео з еталонною тривимірною розміткою, створення або залучення якої є окремою трудомісткою задачею.

Як зазначено в підрозділі 4.4, для треків, коротших за розмір вікна згладжування, метрика тремтіння набуває нульового значення внаслідок повного усереднення треку. Це спотворює зведені показники у бік заниження середнього тремтіння. У поточному дослідженні вплив артефакту обмежений, оскільки короткі треки переважно вилучаються фільтрацією, проте коректним удосконаленням було б обчислювати метрику лише для треків, довжина яких перевищує розмір вікна.

Фільтрація за довжиною треку ґрунтується на припущенні, що короткий трек є фантомом. Це припущення справедливе для більшості випадків, але не для всіх: реальний об'єкт, що лише ненадовго з'явився в полі зору камери або був більшу частину часу перекритий іншим об'єктом, також утворює короткий трек і може бути помилково вилучений. Таким чином, фільтрація має компромісний характер між усуненням шуму і збереженням рідкісних коректних детекцій. Підбір порогів фільтрації, що мінімізує цей компроміс, та дослідження адаптивних порогів є напрямом удосконалення.

Застосований метод зіставлення спирається на перекриття обмежувальних рамок у сусідніх кадрах. Якщо об'єкт на якомусь кадрі не був виявлений моделлю або зазнав сильного перекриття, трек розривається, і після відновлення виявлення об'єкт отримує новий ідентифікатор. Це призводить до

фрагментації треків і завищення їх кількості. Удосконаленням є введення механізму продовження треку через пропущені кадри, а також застосування прогнозування положення об'єкта, наприклад за допомогою фільтра Калмана.

Поточна реалізація згладжує чотири координати рамки незалежно одна від одної. Це спрощення, яке не зберігає узгодженості розмірів рамки: ширина і висота об'єкта фізично змінюються плавно, але незалежне згладжування меж цього явно не враховує. Згладжування у параметризації центр, ширина, висота замість меж краще відповідало б фізичній природі руху об'єкта.

Виявлення об'єктів моделлю триває близько 50 секунд на кадр на наявному апаратному забезпеченні, що унеможливлює обробку в реальному часі і обмежує практичне застосування системи пакетним режимом. Напрямами пом'якшення є застосування потужнішого апаратного забезпечення, зменшення вхідної роздільної здатності, а також застосування полегшених варіантів моделі виявлення.

Перелічені обмеження не спростовують отриманих результатів, а окреслюють межі їх застосовності та формують програму подальшого розвитку системи.

Висновок до розділу 4

У розділі наведено результати експериментального дослідження розробленої системи. Дослідження проведено на відеозаписі типової міської сцени як пілотний експеримент. Для кількісної оцінки часової стабільності запропоновано метрику тремтіння, яка вимірює середнє зміщення координат обмежувальної рамки між сусідніми кадрами і не потребує еталонної розмітки.

Дослідження фільтрації показало, що з 281 сирої детекції модуль зіставлення утворив 57 треків, з яких понад половина — короткі треки завдовжки один-два кадри, що відповідають хибним спрацюванням моделі. Комбінована фільтрація вилучила 64,9 відсотка треків, зберігши при цьому всі довгі треки реальних об'єктів. Дослідження згладжування показало, що метод ковзного середнього зменшує середнє тремтіння обмежувальних рамок на 85,3 відсотка — з 8,97 до 1,32 пікселя. Абляційне порівняння двох методів згладжування встановило перевагу ковзного середнього над експоненційним згладжуванням, яке зменшило тремтіння лише на 50,9 відсотка; перевагу пояснено симетричністю ковзного середнього.

Виявлені обмеження дослідження — насамперед обмежений обсяг експериментальної вибірки та відсутність еталонної розмітки — окреслюють напрямки подальшої роботи. Загалом результати підтверджують, що запропонована система ефективно підвищує часову стабільність 3D-виявлення об'єктів у відео.

ВИСНОВКИ

У дипломній роботі розв'язано актуальну задачу підвищення часової стабільності тривимірного виявлення об'єктів у відеопослідовностях природних сцен. Поставлену мету досягнуто, усі визначені завдання виконано.

1. Проведено аналіз предметної області. Встановлено, що монокулярне тривимірне виявлення об'єктів є практично важливим, але складним напрямом комп'ютерного зору, а сучасні моделі, такі як WildDet3D, попри високу якість на окремих зображеннях, розроблені без урахування специфіки відео. Показано, що покадрове застосування таких моделей до відео спричиняє часову неузгодженість результатів — мерехтіння детекцій, тремтіння обмежувальних рамок та нестабільність ідентифікації об'єктів. Обґрунтовано, що для умов роботи доцільним є підхід постобробки за схемою відстеження через виявлення, який не потребує перенавчання моделі.

2. Досліджено модель WildDet3D як базовий компонент виявлення: розглянуто її архітектуру, формат вхідних і вихідних даних та обмеження. Визначено, що ключовим обмеженням, яке зумовлює потребу в постобробці, є незалежна обробка кожного кадру без урахування часового зв'язку між кадрами.

3. Реалізовано базовий конвеєр обробки відео, який виконує покадрове виявлення об'єктів та зберігає результати у структурованому форматі JSON. Запропоновано інкрементний формат збереження даних, у якому кожен етап обробки доповнює структуру детекцій новими полями, що забезпечує простежуваність обробки та відтворюваність експериментів.

4. Розроблено метод зіставлення об'єктів між кадрами на основі перетину над об'єднанням обмежувальних рамок та жадібного алгоритму зіставлення. Метод присвоює детекціям стійкі ідентифікатори треків, що дозволяє простежувати об'єкти у часі.

5. Розроблено метод фільтрації шуму, який вилучає хибні детекції на основі часових характеристик треків — їх довжини та середньої впевненості. Експериментально встановлено, що метод усуває близько 65 відсотків треків,

які відповідають хибним спрацюванням моделі, зберігаючи при цьому треки реальних об'єктів.

6. Розроблено метод часового згладжування координат обмежувальних рамок, реалізований у двох варіантах — ковзного середнього та експоненційного згладжування. Метод зменшує тремтіння рамок, спричинене стохастичною природою покадрового виявлення.

7. Реалізовано систему автоматизованого аналізу результатів та запропоновано метрику часової стабільності — показник тремтіння, який вимірює середнє зміщення координат обмежувальної рамки між сусідніми кадрами і не потребує еталонної розмітки даних.

8. Проведено експериментальне дослідження розробленої системи на відеозаписі типової міської сцени. Встановлено, що часове згладжування методом ковзного середнього зменшує середнє тремтіння обмежувальних рамок на 85,3 відсотка — з 8,97 до 1,32 пікселя. Абляційне порівняння методів згладжування показало перевагу ковзного середнього над експоненційним згладжуванням, яке зменшує тремтіння на 50,9 відсотка; перевагу пояснено симетричністю ковзного середнього.

Наукова новизна роботи полягає у запропонованому підході до підвищення часової стабільності тривимірного виявлення як етапу постобробки, що не потребує перенавчання моделі, та у запропонованій метриці часової стабільності, яка не потребує еталонної розмітки.

Практичне значення роботи полягає в тому, що розроблена система підвищує якість тривимірного виявлення об'єктів у відео на доступному апаратному забезпеченні, має модульну архітектуру, придатну для використання з різними моделями виявлення, і може слугувати основою для подальших досліджень.

Визначено напрями подальшого розвитку системи: розширення експериментальної вибірки на відео з різними умовами; запровадження механізму продовження треків через пропущені кадри та прогнозування положення об'єктів; дослідження адаптивних порогів фільтрації;

згладжування рамок у параметризації центр — розміри; доповнення метрики часової стабільності метриками точності на основі еталонної розмітки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Huang W., Zhang J., Li S. et al. WildDet3D: Scaling Promptable 3D Detection in the Wild. arXiv preprint arXiv:2604.08626. 2026.

2. Brazil G., Kumar A., Straub J., Ravi N., Johnson J., Gkioxari G. Omni3D: A Large Benchmark and Model for 3D Object Detection in the Wild // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2023. P. 13154–13164.

3. Kirillov A., Mintun E., Ravi N. et al. Segment Anything // Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). 2023. P. 4015–4026.

4. Wojke N., Bewley A., Paulus D. Simple Online and Realtime Tracking with a Deep Association Metric // Proceedings of the IEEE International Conference on Image Processing (ICIP). 2017. P. 3645–3649.

5. Bewley A., Ge Z., Ott L., Ramos F., Upcroft B. Simple Online and Realtime Tracking // Proceedings of the IEEE International Conference on Image Processing (ICIP). 2016. P. 3464–3468.

6. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. P. 779–788.

7. Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2017. Vol. 39, No. 6. P. 1137–1149.

8. Geiger A., Lenz P., Urtasun R. Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2012. P. 3354–3361.

9. Caesar H., Bankiti V., Lang A. H. et al. nuScenes: A Multimodal Dataset for Autonomous Driving // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2020. P. 11621–11631.

10. Lin T.-Y., Maire M., Belongie S. et al. Microsoft COCO: Common Objects in Context // Proceedings of the European Conference on Computer Vision (ECCV). 2014. P. 740–755.

11. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. P. 770–778.

12. Vaswani A., Shazeer N., Parmar N. et al. Attention Is All You Need // Advances in Neural Information Processing Systems (NeurIPS). 2017. P. 5998–6008.

13. Dosovitskiy A., Beyer L., Kolesnikov A. et al. An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale // International Conference on Learning Representations (ICLR). 2021.

14. Paszke A., Gross S., Massa F. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library // Advances in Neural Information Processing Systems (NeurIPS). 2019. P. 8024–8035.

15. Bradski G. The OpenCV Library // Dr. Dobb's Journal of Software Tools. 2000. Vol. 25, No. 11. P. 120–125.

16. Harris C. R., Millman K. J., van der Walt S. J. et al. Array Programming with NumPy // Nature. 2020. Vol. 585. P. 357–362.

17. Hunter J. D. Matplotlib: A 2D Graphics Environment // Computing in Science & Engineering. 2007. Vol. 9, No. 3. P. 90–95.

18. Kalman R. E. A New Approach to Linear Filtering and Prediction Problems // Journal of Basic Engineering. 1960. Vol. 82, No. 1. P. 35–45.

19. Kuhn H. W. The Hungarian Method for the Assignment Problem // Naval Research Logistics Quarterly. 1955. Vol. 2, No. 1–2. P. 83–97.

20. Mousavian A., Anguelov D., Flynn J., Košecká J. 3D Bounding Box Estimation Using Deep Learning and Geometry // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. P. 7074–7082.

21. Wang T., Zhu X., Pang J., Lin D. FCOS3D: Fully Convolutional One-Stage Monocular 3D Object Detection // Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (ICCVW). 2021. P. 913–922.

22. Zhu X., Wang Y., Dai J., Yuan L., Wei Y. Flow-Guided Feature Aggregation for Video Object Detection // Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2017. P. 408–417.
23. Liu W., Anguelov D., Erhan D. et al. SSD: Single Shot MultiBox Detector // Proceedings of the European Conference on Computer Vision (ECCV). 2016. P. 21–37.
24. Zhou X., Koltun V., Krähenbühl P. Tracking Objects as Points // Proceedings of the European Conference on Computer Vision (ECCV). 2020. P. 474–490.

ДОДАТОК А

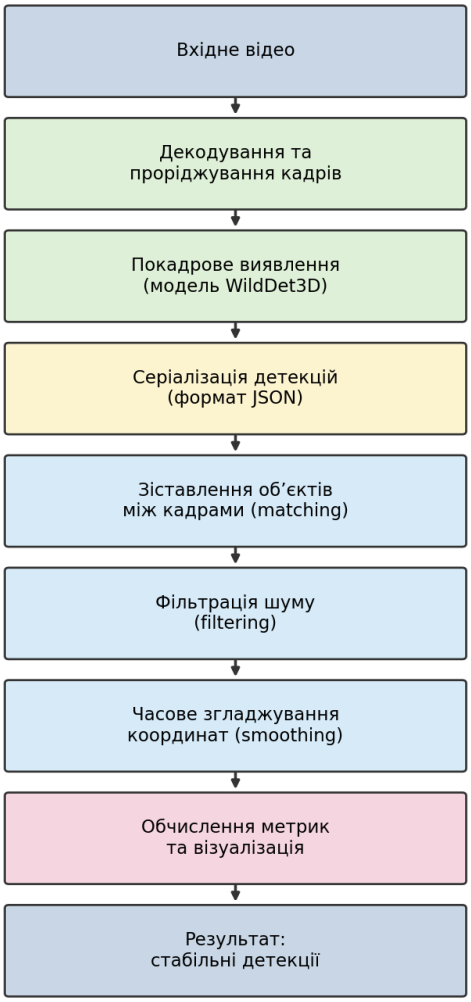
Система 3D-виявлення об’єктів у природних середовищах

Схема архітектури системи постобробки

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ – 2026 р.



Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467200.004 Д1			
Розробив		Демчик Д. С.						
Перевірив		Гордієнко Ю. Г.			Схема архітектури системи постобробки	Літ.	Аркуш	Аркушів
Реценз.		Сергієнко П. А.					1	1
Н. Контр.		Коренко Д. В.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21		
Затвердив		Волокита А. М.						

ДОДАТОК Б

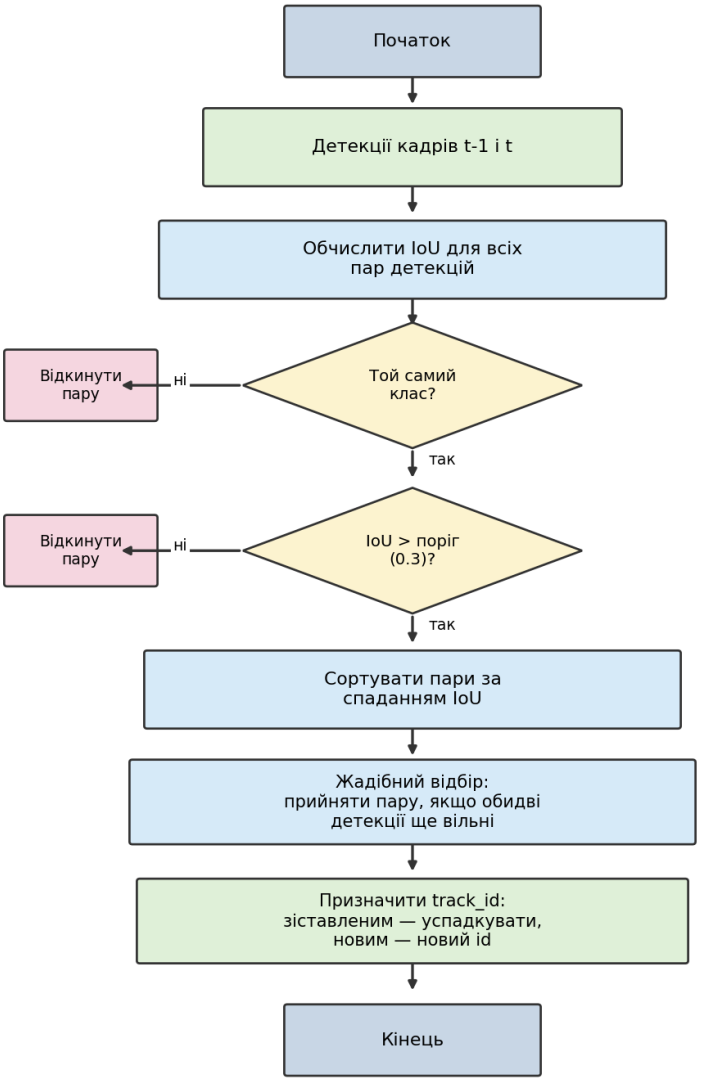
Система 3D-виявлення об’єктів у природних середовищах

Блок-схема алгоритму зіставлення об’єктів

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ – 2026 р.



Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467200.005 Д2		
Розробив		Демчик Д. С.					
Перевірив		Гордієнко Ю. Г.			Блок-схема алгоритму зіставлення об'єктів	Літ.	Аркуш 1 / Аркушів 1
Реценз.		Сергієнко П. А.					
Н. Контр.		Коренко Д. В.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21	
Затвердив		Волокита А. М.					

ДОДАТОК В

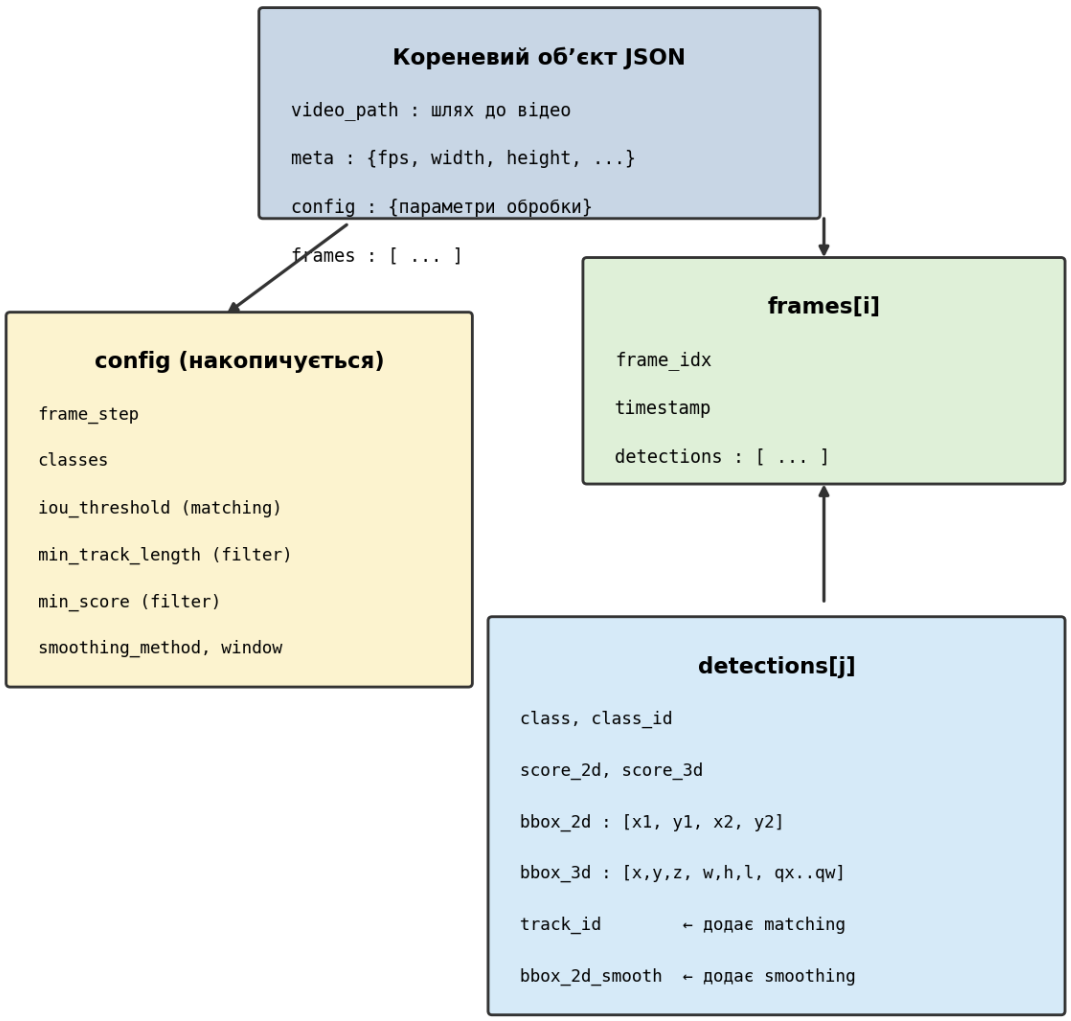
Система 3D-виявлення об’єктів у природних середовищах

Схема формату збереження даних

ІАЛЦ.467200.006 ДЗ

Аркушів 1

Київ – 2026 р.



Кожен етап постобробки доповнює структуру новими полями, не змінюючи попередніх (інкрементність)

Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467200.006 Д3			
Розробив		Демчик Д. С.						
Перевірив		Гордієнко Ю. Г.			Схема формату збереження даних	Літ.	Аркуш	Аркушів
Реценз.		Сергієнко П. А.					1	1
Н. Контр.		Коренко Д. В.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21		
Затвердив		Волокита А. М.						

ДОДАТОК Г

Система 3D-виявлення об’єктів у природних середовищах

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 22

Київ – 2026 р.

Файл src/pipeline.py

```
"""
src/pipeline.py
================
Базовий пайплайн для прогону WildDet3D на відео.

Що робить:
  1. Відкриває відео через OpenCV
  2. Семплює кожен N-ий кадр
  3. Прогоняє кадр через модель WildDet3D
  4. Серіалізує детекції у структурований словник (готовий до JSON)

Модель завантажується ОДИН РАЗ ззовні (передається в process_video як параметр).
Це важливо: завантаження ~50 секунд, тож для батч-обробки кількох відео
ми не хочемо платити цю ціну для кожного відео.
"""
from __future__ import annotations

import json
import sys
import time
from pathlib import Path
from typing import Any

import cv2
import numpy as np
import torch
from tqdm import tqdm

# Додаємо шлях до WildDet3D у sys.path, оскільки він живе поза нашим src/.
# Робиться один раз при імпорті модуля.
_WILDDET3D_PATH = Path(__file__).resolve().parent.parent / "WildDet3D"
if str(_WILDDET3D_PATH) not in sys.path:
    sys.path.insert(0, str(_WILDDET3D_PATH))

from wilddet3d import build_model, preprocess  # noqa: E402


def load_model(
    checkpoint_path: str | Path,
    score_threshold: float = 0.3,
) -> Any:
    """Завантажує модель WildDet3D з чекпойнта. Викликається один раз."""
    checkpoint_path = Path(checkpoint_path)
    if not checkpoint_path.exists():
        raise FileNotFoundError(f"Чекпойнт не знайдено: {checkpoint_path}")

    print(f"[load_model] Завантаження моделі з {checkpoint_path}...")
    t0 = time.time()
    model = build_model(
        checkpoint=str(checkpoint_path),
        score_threshold=score_threshold,
        skip_pretrained=True,
    )
    print(f"[load_model] Готово за {time.time() - t0:.1f}с")
    return model
```

```
def detect_on_frame(
    model: Any,
    frame_rgb: np.ndarray,
    input_texts: list[str],
) -> list[dict[str, Any]]:
    """
    Запускає інференс на одному кадрі.

    Параметри:
        model: завантажена модель WildDet3D
        frame_rgb: кадр у форматі RGB (H, W, 3), uint8 або float32
        input_texts: список класів для пошуку, напр. ["car", "person"]

    Повертає:
        Список детекцій, кожна — словник з полями:
            - class: ім'я класу (з input_texts)
            - score_2d, score_3d: впевненості
            - bbox_2d: [x1, y1, x2, y2] у пікселях
            - bbox_3d: [x, y, z, w, h, d, qx, qy, qz, qw]
                (центр + розміри + кватерніон орієнтації)
    """
    # Препроцесинг (ресайз, паддінг тощо)
    image = frame_rgb.astype(np.float32)
    data = preprocess(image)

    # Forward pass
    results = model(
        images=data["images"].cuda(),
        intrinsics=data["intrinsics"].cuda()[None],
        input_hw=[data["input_hw"]],
        original_hw=[data["original_hw"]],
        padding=[data["padding"]],
        input_texts=input_texts,
    )
    boxes_2d, boxes_3d, scores, scores_2d, scores_3d, class_ids, _ = results

    # Розпаковуємо результати в список словників.
    # results — це списки довжини B (batch=1), беремо [0].
    detections: list[dict[str, Any]] = []
    if class_ids[0] is None or len(class_ids[0]) == 0:
        return detections

    for i in range(len(class_ids[0])):
        cls_idx = int(class_ids[0][i].item())
        detections.append({
            "class": input_texts[cls_idx],
            "class_id": cls_idx,
            "score_2d": float(scores_2d[0][i].item()),
            "score_3d": float(scores_3d[0][i].item()),
            "bbox_2d": boxes_2d[0][i].cpu().tolist(),
            "bbox_3d": boxes_3d[0][i].cpu().tolist(),
        })
    return detections



def process_video(
    model: Any,
    video_path: str | Path,
    input_texts: list[str],
    frame_step: int = 10,
    max_frames: int | None = None,
) -> dict[str, Any]:
```

```
    """
    Прогоняє pipeline на цілому відео.

    Параметри:
        model: завантажена модель WildDet3D
        video_path: шлях до .mp4 (або іншого формату, що підтримує OpenCV)
        input_texts: список класів для пошуку
        frame_step: брати кожен N-ий кадр (1 = всі, 10 = кожен 10-й)
        max_frames: обмеження на кількість оброблених кадрів (None = без обмеження).
            Корисно для дебагу: можна швидко перевірити що pipeline працює,
            обробивши лише 3-5 кадрів.

    Повертає:
        Словник з метаданими відео та списком детекцій по кадрах,
        придатний для серіалізації в JSON.
    """
    video_path = Path(video_path)
    if not video_path.exists():
        raise FileNotFoundError(f"Відео не знайдено: {video_path}")

    cap = cv2.VideoCapture(str(video_path))
    fps = cap.get(cv2.CAP_PROP_FPS)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Збираємо індекси кадрів, які будемо обробляти
    frame_indices = list(range(0, total_frames, frame_step))
    if max_frames is not None:
        frame_indices = frame_indices[:max_frames]

    print(f"[process_video] {video_path.name}: {total_frames} кадрів @ {fps:.1f} FPS")
    print(f"[process_video] Оброблятимемо {len(frame_indices)} кадрів (step={frame_step})")

    frames_data: list[dict[str, Any]] = []
    t0 = time.time()

    for frame_idx in tqdm(frame_indices, desc="Інференс", unit="кадр"):
        # Перемотуємо до потрібного кадру
        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_idx)
        ok, frame_bgr = cap.read()
        if not ok:
            print(f"[WARN] Не вдалося прочитати кадр {frame_idx}, пропускаємо")
            continue

        # OpenCV повертає BGR — конвертуємо в RGB для моделі
        frame_rgb = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2RGB)

        # Запускаємо інференс
        detections = detect_on_frame(model, frame_rgb, input_texts)

        frames_data.append({
            "frame_idx": frame_idx,
            "timestamp_sec": frame_idx / fps if fps > 0 else None,
            "detections": detections,
        })

    cap.release()
    elapsed = time.time() - t0

    result: dict[str, Any] = {
```

```
        "video": str(video_path),
        "video_metadata": {
            "fps": float(fps),
            "total_frames": total_frames,
            "width": width,
            "height": height,
            "duration_sec": total_frames / fps if fps > 0 else None,
        },
        "pipeline_config": {
            "frame_step": frame_step,
            "max_frames": max_frames,
            "input_texts": input_texts,
            "processed_frames": len(frames_data),
            "elapsed_sec": elapsed,
            "avg_sec_per_frame": elapsed / max(1, len(frames_data)),
        },
        "frames": frames_data,
    }
    return result



def save_detections_json(result: dict[str, Any], output_path: str | Path) -> None:
    """Зберігає результат у JSON."""
    output_path = Path(output_path)
    output_path.parent.mkdir(parents=True, exist_ok=True)
    with open(output_path, "w", encoding="utf-8") as f:
        json.dump(result, f, indent=2, ensure_ascii=False)
    print(f"[save_detections_json] Збережено: {output_path}")
```

Файл src/matching.py

```
"""
src/matching.py
================
IoU-based matching детекцій між сусідніми кадрами.

Призначення:
    Кожен кадр модель обробляє незалежно, тому одна й та сама машина
    у кадрах t, t+1, t+2 отримує різні "індекси" в списку детекцій.
    Нам же потрібен спільний track_id, щоб ідентифікувати об'єкт
    у часі (для smoothing, filtering, метрики stability).

Алгоритм:
    1. Для кожного кадру t (починаючи з 1) рахуємо матрицю IoU
       між детекціями кадру t-1 та кадру t.
    2. Жадібно матчимо пари з найбільшим IoU > порогу:
        - найвищий IoU → пара зматчена → видаляємо рядок і стовпець
        - повторюємо, поки є пари з IoU > порогу
    3. Зматчені детекції наслідують track_id від попередніх.
    4. Незматчені — отримують новий track_id (це або новий об'єкт,
       або фантом, який зникне в наступному кадрі).

Зауваження щодо вибору IoU:
    Альтернативи — center distance або 3D IoU. Ми вибрали 2D IoU,
    бо: (а) 2D-боксі модель видає стабільніше за 3D,
    (б) IoU природно враховує і позицію, і розмір об'єкта,
    (в) для близьких послідовних кадрів зрушення малі → IoU високий.
"""
from __future__ import annotations

from typing import Any



def compute_iou(box_a: list[float], box_b: list[float]) -> float:
```

```
    """
    Обчислює Intersection over Union для двох 2D-боксів у форматі [x1, y1, x2, y2].

    Повертає значення від 0.0 (немає перетину) до 1.0 (повний збіг).
    """
    ax1, ay1, ax2, ay2 = box_a
    bx1, by1, bx2, by2 = box_b

    # Координати прямокутника перетину
    inter_x1 = max(ax1, bx1)
    inter_y1 = max(ay1, by1)
    inter_x2 = min(ax2, bx2)
    inter_y2 = min(ay2, by2)

    # Якщо прямокутники не перетинаються — IoU = 0
    inter_w = max(0.0, inter_x2 - inter_x1)
    inter_h = max(0.0, inter_y2 - inter_y1)
    inter_area = inter_w * inter_h

    if inter_area == 0.0:
        return 0.0

    # Площі обох боксів та об'єднання
    area_a = max(0.0, ax2 - ax1) * max(0.0, ay2 - ay1)
    area_b = max(0.0, bx2 - bx1) * max(0.0, by2 - by1)
    union_area = area_a + area_b - inter_area

    if union_area <= 0.0:
        return 0.0
    return inter_area / union_area


def match_frames(
    prev_detections: list[dict[str, Any]],
    curr_detections: list[dict[str, Any]],
    iou_threshold: float = 0.3,
    same_class_only: bool = True,
) -> list[tuple[int, int, float]]:
    """
    Жадібний matching між двома списками детекцій.

    Параметри:
        prev_detections: детекції кадру t-1
        curr_detections: детекції кадру t
        iou_threshold: мінімальний IoU для матчу (детекції з IoU нижче не матчаться).
            0.3 — компроміс: достатньо м'яко, щоб дозволити рух між кадрами,
            але достатньо суворо, щоб не плутати різні об'єкти поряд.
        same_class_only: матчити тільки детекції одного класу
            (машина-машина, людина-людина, але не машина-людина).

    Повертає:
        Список пар (i, j, iou), де i — індекс у prev_detections,
        j — індекс у curr_detections, iou — їх IoU.
    """
    matches: list[tuple[int, int, float]] = []
    if not prev_detections or not curr_detections:
        return matches

    # Будуємо список усіх можливих пар із їх IoU
    candidates: list[tuple[float, int, int]] = []
    for i, prev in enumerate(prev_detections):
        for j, curr in enumerate(curr_detections):
```

```
            if same_class_only and prev["class"] != curr["class"]:
                continue
            iou = compute_iou(prev["bbox_2d"], curr["bbox_2d"])
            if iou >= iou_threshold:
                candidates.append((iou, i, j))

    # Сортуємо за IoU спадаюче (найкращі пари — першими)
    candidates.sort(reverse=True)

    used_prev: set[int] = set()
    used_curr: set[int] = set()

    for iou, i, j in candidates:
        # Якщо один з боків уже зайнятий — пропускаємо
        if i in used_prev or j in used_curr:
            continue
        matches.append((i, j, iou))
        used_prev.add(i)
        used_curr.add(j)

    return matches


def assign_track_ids(
    detections_per_frame: list[list[dict[str, Any]]],
    iou_threshold: float = 0.3,
    same_class_only: bool = True,
) -> list[list[dict[str, Any]]]:
    """
    Призначає track_id усім детекціям у послідовності кадрів.

    Параметри:
        detections_per_frame: список списків детекцій (один список на кадр)
        iou_threshold, same_class_only: див. match_frames

    Повертає:
        Той самий список, але до кожної детекції додано поле "track_id" (int).
        НЕ модифікує оригінальний вхід (повертає копію).
    """
    # Робимо глибоку копію, щоб не псувати вхідні дані
    out: list[list[dict[str, Any]]] = [
        [dict(det) for det in frame] for frame in detections_per_frame
    ]

    next_track_id = 0

    # Кадр 0: всім детекціям присвоюємо нові track_id
    if out:
        for det in out[0]:
            det["track_id"] = next_track_id
            next_track_id += 1

    # Кадри 1..N: матчимо з попереднім, наслідуємо track_id
    for t in range(1, len(out)):
        prev_frame = out[t - 1]
        curr_frame = out[t]

        matches = match_frames(
            prev_frame, curr_frame,
            iou_threshold=iou_threshold,
            same_class_only=same_class_only,
```

```
        )

        # Зматчені — наслідують track_id
        matched_curr_indices: set[int] = set()
        for i, j, _iou in matches:
            curr_frame[j]["track_id"] = prev_frame[i]["track_id"]
            matched_curr_indices.add(j)

        # Незматчені — нові track_id
        for j, det in enumerate(curr_frame):
            if j not in matched_curr_indices:
                det["track_id"] = next_track_id
                next_track_id += 1

    return out
```

Файл src/filtering.py

```
"""
src/filtering.py
================
Фільтрація шуму у tracked-детекціях.

Передумова: на вхід вже надходять детекції з присвоєними track_id
(результат src.matching.assign_track_ids).

Підходи до фільтрації:
    1. Мінімальна довжина треку: викидаємо треки коротші за N кадрів
       (типові фантоми з'являються на 1-2 кадрах).
    2. Середній score: викидаємо треки з низькою впевненістю
       (модель може стабільно "бачити" фантом на текстурі стіни — це не довжина,
       а середня впевненість).
    3. Комбінований: і довжина, і score одночасно.

Кожен фільтр повертає НОВУ структуру (не модифікує вхідну),
з тими ж кадрами, але без відфільтрованих детекцій.
"""
from __future__ import annotations

from collections import defaultdict
from statistics import mean
from typing import Any



def compute_track_stats(
    detections_per_frame: list[list[dict[str, Any]]],
) -> dict[int, dict[str, Any]]:
    """
    Підраховує статистику по кожному треку.

    Параметри:
        detections_per_frame: список списків детекцій з track_id (вихід assign_track_ids).

    Повертає:
        Словник {track_id: {"length": int, "mean_score_2d": float,
                            "mean_score_3d": float, "class": str}}
    """
    by_track: dict[int, list[dict[str, Any]]] = defaultdict(list)
    for frame in detections_per_frame:
        for det in frame:
            by_track[det["track_id"]].append(det)
```

```
    stats: dict[int, dict[str, Any]] = {}
    for tid, dets in by_track.items():
        stats[tid] = {
            "length": len(dets),
            "mean_score_2d": mean(d["score_2d"] for d in dets),
            "mean_score_3d": mean(d["score_3d"] for d in dets),
            "class": dets[0]["class"],  # клас стабільний у межах треку (matching
same_class_only)
        }
    return stats



def filter_by_min_length(
    detections_per_frame: list[list[dict[str, Any]]],
    min_length: int = 2,
) -> list[list[dict[str, Any]]]:
    """
    Викидає всі детекції, чий трек має довжину менше min_length.

    Параметри:
        min_length: мінімальна довжина треку (у кадрах).
            2 — м'яко (зберігає двокадрові, які могли б бути реальними).
            3 — суворіше (треки мають з'явитись хоча б у 3 кадрах).

    Повертає:
        Новий список кадрів з відфільтрованими детекціями.
    """
    stats = compute_track_stats(detections_per_frame)
    valid_ids = {tid for tid, s in stats.items() if s["length"] >= min_length}

    return [
        [det for det in frame if det["track_id"] in valid_ids]
        for frame in detections_per_frame
    ]



def filter_by_mean_score(
    detections_per_frame: list[list[dict[str, Any]]],
    min_mean_score_2d: float = 0.5,
) -> list[list[dict[str, Any]]]:
    """
    Викидає треки з низьким середнім score_2d.

    Параметри:
        min_mean_score_2d: поріг для середнього 2D-score треку.

    Повертає:
        Новий список кадрів.
    """
    stats = compute_track_stats(detections_per_frame)
    valid_ids = {
        tid for tid, s in stats.items()
        if s["mean_score_2d"] >= min_mean_score_2d
    }

    return [
        [det for det in frame if det["track_id"] in valid_ids]
        for frame in detections_per_frame
    ]
```

```
def filter_combined(
    detections_per_frame: list[list[dict[str, Any]]],
    min_length: int = 2,
    min_mean_score_2d: float = 0.4,
) -> list[list[dict[str, Any]]]:
    """
    Комбінований фільтр: трек має пройти ОБИДВА критерії —
    і за довжиною, і за середнім score.


    Це наш основний робочий фільтр. Параметри підбираються емпірично
    на основі експериментів (це частина ablation study).
    """
    stats = compute_track_stats(detections_per_frame)
    valid_ids = {
        tid for tid, s in stats.items()
        if s["length"] >= min_length and s["mean_score_2d"] >= min_mean_score_2d
    }


    return [
        [det for det in frame if det["track_id"] in valid_ids]
        for frame in detections_per_frame
    ]
```

Файл src/smoothing.py

```
"""
src/smoothing.py
=================
Згладжування координат боксів у межах треку для зменшення jitter.


Передумова: на вхід надходять детекції з присвоєними track_id
(вихід src.matching.assign_track_ids).


Підходи:
    1. Moving Average (MA): для кадру t середнє з вікна [t-w/2 .. t+w/2].
       Симетричне, сильніше згладжує, потребує всю послідовність.
    2. Exponential Moving Average (EMA): рекурсивне з параметром alpha.
       Тільки минуле — підходить для realtime.


Обидва згладжують координати bbox_2d (x1, y1, x2, y2) НЕЗАЛЕЖНО.
Це спрощення — нормально для дипломної, але є альтернатива:
згладжувати (center_x, center_y, width, height) — це краще збереже форму,
бо ширина/висота фізично мало змінюються між кадрами.
"""
from __future__ import annotations


from collections import defaultdict
from typing import Any



def _smooth_track_moving_average(
    track_detections: list[tuple[int, int, dict[str, Any]]],
    window: int = 3,
) -> None:
    """
    Згладжує координати bbox_2d одного треку методом moving average.


    Параметри:
        track_detections: список (frame_idx_in_seq, original_idx, det)
            відсортований за frame_idx_in_seq у зростаючому порядку.
            det — словник детекції (модифікується IN-PLACE).
        window: розмір вікна (непарне, типово 3 або 5).
```

```
    Логіка вікна:
        Для детекції на позиції i беремо детекції з позицій
        [i - window//2 .. i + window//2] (clamped до меж треку).
        Тобто на краях треку вікно стає несиметричним.
    """
    if not track_detections:
        return
    half = window // 2
    n = len(track_detections)


    # Спочатку зберігаємо оригінальні bbox, бо ми не можемо
    # читати "сирі" значення з det одразу після того як їх замінили.
    raw_bboxes: list[list[float]] = [
        list(d["bbox_2d"]) for _, _, d in track_detections
    ]


    for i in range(n):
        lo = max(0, i - half)
        hi = min(n - 1, i + half)
        # Середнє координат у вікні
        window_bboxes = raw_bboxes[lo:hi + 1]
        smoothed = [
            sum(b[k] for b in window_bboxes) / len(window_bboxes)
            for k in range(4)
        ]
        track_detections[i][2]["bbox_2d_smoothed"] = smoothed



def _smooth_track_ema(
    track_detections: list[tuple[int, int, dict[str, Any]]],
    alpha: float = 0.5,
) -> None:
    """
    Згладжує координати bbox_2d одного треку методом EMA.


    Параметри:
        track_detections: див. _smooth_track_moving_average
        alpha: коефіцієнт EMA, 0 < alpha <= 1
            alpha=1.0 — без згладжування (повторює raw)
            alpha=0.5 — баланс минуле/нове
            alpha=0.2 — сильне згладжування


    Рекурентне рівняння:
        smoothed[0] = raw[0]
        smoothed[i] = alpha * raw[i] + (1 - alpha) * smoothed[i-1]
    """
    if not track_detections:
        return


    prev_smoothed: list[float] | None = None
    for _, _, det in track_detections:
        raw = det["bbox_2d"]
        if prev_smoothed is None:
            smoothed = list(raw)
        else:
            smoothed = [
                alpha * raw[k] + (1.0 - alpha) * prev_smoothed[k]
                for k in range(4)
            ]
        det["bbox_2d_smoothed"] = smoothed
        prev_smoothed = smoothed
```

```
def apply_smoothing(
    detections_per_frame: list[list[dict[str, Any]]],
    method: str = "ma",
    window: int = 3,
    alpha: float = 0.5,
) -> list[list[dict[str, Any]]]:
    """
    Застосовує згладжування до всіх треків.

    Параметри:
        detections_per_frame: tracked детекції з track_id
        method: "ma" (moving average) або "ema" (exponential)
        window: для "ma" — розмір вікна
        alpha: для "ema" — коефіцієнт згладжування

    Повертає:
        Новий список кадрів (глибока копія).
        Кожна детекція має нове поле "bbox_2d_smoothed" — згладжені координати.
        Оригінальне поле "bbox_2d" не змінюється — це важливо для
        ablation: можемо порівнювати raw vs smoothed на одних і тих же даних.
    """
    if method not in ("ma", "ema"):
        raise ValueError(f"Невідомий метод: {method!r}. Очікую 'ma' або 'ema'.")

    # Глибока копія, щоб не псувати вхідні дані
    out: list[list[dict[str, Any]]] = [
        [dict(det) for det in frame] for frame in detections_per_frame
    ]

    # Групуємо детекції по track_id, зберігаючи (frame_seq_idx, det_idx_in_frame, det)
    tracks: dict[int, list[tuple[int, int, dict[str, Any]]]] = defaultdict(list)
    for seq_idx, frame in enumerate(out):
        for det_idx, det in enumerate(frame):
            if "track_id" not in det:
                continue
            tracks[det["track_id"]].append((seq_idx, det_idx, det))

    # Згладжуємо кожен трек незалежно
    for tid, track in tracks.items():
        # Вже відсортовано: ми йшли по кадрах послідовно вище
        if method == "ma":
            _smooth_track_moving_average(track, window=window)
        else:
            _smooth_track_ema(track, alpha=alpha)

    return out
```

Файл src/metrics.py

```
"""
src/metrics.py
================
Метрики для оцінки якості tracked + smoothed детекцій.

Найважливіша — jitter: наскільки стабільні координати об'єкта між кадрами.
"""
from __future__ import annotations

from collections import defaultdict
from statistics import mean
from typing import Any
```

```
def compute_jitter_per_track(
    detections_per_frame: list[list[dict[str, Any]]],
    bbox_field: str = "bbox_2d",
) -> dict[int, float]:
    """
    Для кожного треку рахує середній абсолютний зсув координат
    між сусідніми кадрами того треку.


    Параметри:
        detections_per_frame: tracked детекції
        bbox_field: яке поле bbox використовувати ("bbox_2d" для сирих,
            "bbox_2d_smoothed" для згладжених)


    Повертає:
        {track_id: average_jitter_in_pixels}
        Треки довжиною 1 (немає сусідніх кадрів) пропускаються.


    Як рахується:
        Для пари сусідніх кадрів (i, i+1) одного треку:
            shift = mean(|x1_i+1 - x1_i|, |y1_i+1 - y1_i|,
                         |x2_i+1 - x2_i|, |y2_i+1 - y2_i|)
        Jitter треку = середнє таких shift по всіх парах.
    """
    # Групуємо детекції по track_id, зберігаючи порядок кадрів
    tracks: dict[int, list[list[float]]] = defaultdict(list)
    for frame in detections_per_frame:
        for det in frame:
            if "track_id" not in det or bbox_field not in det:
                continue
            tracks[det["track_id"]].append(list(det[bbox_field]))


    result: dict[int, float] = {}
    for tid, bboxes in tracks.items():
        if len(bboxes) < 2:
            continue
        # Зсув для кожної пари (i, i+1)
        shifts: list[float] = []
        for i in range(len(bboxes) - 1):
            a, b = bboxes[i], bboxes[i + 1]
            pair_shift = mean(abs(b[k] - a[k]) for k in range(4))
            shifts.append(pair_shift)
        result[tid] = mean(shifts)
    return result



def summarize_jitter(jitter_per_track: dict[int, float]) -> dict[str, float]:
    """
    Підсумкові статистики по jitter всіх треків.
    """
    values = list(jitter_per_track.values())
    if not values:
        return {"count": 0, "mean": 0.0, "max": 0.0, "min": 0.0}
    return {
        "count": len(values),
        "mean": mean(values),
        "max": max(values),
        "min": min(values),
    }
```

Файл src/visualizer.py

```
"""
src/visualizer.py
==================
Власна візуалізація tracked-детекцій на кадрах відео.
```

```
Чому не використовуємо draw_3d_boxes з wilddet3d:
    1. Вона потребує реальні intrinsics камери, яких у нас немає.
    2. Нам зараз важливі 2D-бокси з track_id (для перевірки matching).
    3. Хочемо легку незалежну реалізацію, щоб контролювати вигляд.


Функції:
    draw_detections_on_frame() — малює список детекцій на одному кадрі
    extract_frame_from_video() — дістає конкретний кадр з відео як numpy array
"""
from __future__ import annotations


from pathlib import Path
from typing import Any


import cv2
import numpy as np



# Палітра кольорів для track_id (BGR, бо OpenCV).
# Кольори свідомо різноманітні, щоб сусідні track_id не зливались.
_PALETTE_BGR: list[tuple[int, int, int]] = [
    (0, 200, 255),    # помаранчевий
    (0, 255, 0),      # зелений
    (255, 100, 0),    # синій-зелений
    (255, 0, 200),    # фіолетовий
    (0, 255, 255),    # жовтий
    (200, 0, 255),    # рожевий
    (255, 200, 0),    # циан
    (100, 255, 100),  # світло-зелений
    (255, 100, 255),  # світло-фіолетовий
    (50, 50, 255),    # червоний
]



def color_for_track_id(track_id: int) -> tuple[int, int, int]:
    """Стабільний колір для track_id (один колір = один трек на всьому відео)."""
    return _PALETTE_BGR[track_id % len(_PALETTE_BGR)]



def extract_frame_from_video(video_path: str | Path, frame_idx: int) -> np.ndarray:
    """
    Дістає кадр з відео за індексом.


    Повертає кадр у форматі BGR (як OpenCV).
    """
    video_path = Path(video_path)
    cap = cv2.VideoCapture(str(video_path))
    cap.set(cv2.CAP_PROP_POS_FRAMES, frame_idx)
    ok, frame = cap.read()
    cap.release()
    if not ok:
        raise RuntimeError(f"Не вдалося прочитати кадр {frame_idx} з {video_path}")
    return frame



def draw_detections_on_frame(
    frame_bgr: np.ndarray,
    detections: list[dict[str, Any]],
    show_track_id: bool = True,
    show_class: bool = True,
```

```
    show_score: bool = True,
    box_thickness: int = 2,
) -> np.ndarray:
    """
    Малює детекції на копії кадру.

    Параметри:
        frame_bgr: кадр у форматі BGR (як з OpenCV)
        detections: список детекцій (формат — той що зі src.pipeline.detect_on_frame)
        show_track_id: чи показувати "T<id>" у підписі (для tracked даних)
        show_class: чи показувати клас
        show_score: чи показувати score_2d
        box_thickness: товщина рамок у пікселях

    Повертає:
        Новий кадр з намальованими боксами і підписами.
    """
    out = frame_bgr.copy()

    for det in detections:
        x1, y1, x2, y2 = (int(v) for v in det["bbox_2d"])

        # Колір залежить від track_id, якщо є. Інакше — нейтральний зелений.
        if "track_id" in det:
            color = color_for_track_id(det["track_id"])
        else:
            color = (0, 255, 0)  # зелений

        # Рамка
        cv2.rectangle(out, (x1, y1), (x2, y2), color, box_thickness)

        # Підпис
        parts: list[str] = []
        if show_track_id and "track_id" in det:
            parts.append(f"T{det['track_id']}")
        if show_class:
            parts.append(det["class"])
        if show_score:
            parts.append(f"{det['score_2d']:.2f}")
        label = " | ".join(parts)

        if label:
            # Фон під текстом — для читабельності на будь-якій картинці
            (text_w, text_h), baseline = cv2.getTextSize(
                label, cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1
            )
            label_y = max(0, y1 - 6)
            cv2.rectangle(
                out,
                (x1, label_y - text_h - baseline),
                (x1 + text_w + 4, label_y + baseline),
                color, -1,  # заповнений
            )
            cv2.putText(
                out, label, (x1 + 2, label_y - 2),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA,
            )

    return out


def stack_horizontal(
```

```
    left: np.ndarray,
    right: np.ndarray,
    left_label: str = "",
    right_label: str = "",
) -> np.ndarray:
    """
    З'єднує два кадри по горизонталі (для порівняння "до/після").

    Якщо передано left_label/right_label — малює їх угорі кадру.
    """
    assert left.shape == right.shape, "Кадри мають бути однакового розміру"
    h, w = left.shape[:2]

    def add_banner(img: np.ndarray, text: str) -> np.ndarray:
        if not text:
            return img
        out = img.copy()
        banner_h = 40
        # Чорний прямокутник зверху
        cv2.rectangle(out, (0, 0), (w, banner_h), (0, 0, 0), -1)
        cv2.putText(
            out, text, (10, 28),
            cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2, cv2.LINE_AA,
        )
        return out

    left_l = add_banner(left, left_label)
    right_l = add_banner(right, right_label)
    return np.hstack([left_l, right_l])
```

Файл scripts/run_pipeline.py

```
"""
scripts/run_pipeline.py
=======================
CLI-обгортка для запуску pipeline на відео.

Приклад використання:
    python scripts/run_pipeline.py data/street.mp4 --classes car person --frame-step 10 --
max-frames 5

Аргументи:
    video_path            — шлях до відео
    --classes             — список класів для пошуку (через пробіл)
    --frame-step          — брати кожен N-ий кадр (default 10)
    --max-frames          — обмеження кадрів для дебагу (default None)
    --score-threshold     — мін. впевненість моделі (default 0.3)
    --checkpoint          — шлях до .pt (default ckpt/wilddet3d_alldata_all_prompt_v1.0.pt)
    --output              — шлях до результуючого JSON (default
outputs/<video_name>_detections.json)
"""
from __future__ import annotations

import argparse
import sys
from pathlib import Path

# Додаємо корінь проекту в sys.path, щоб імпорт `src.pipeline` працював
# незалежно від того, звідки запущено скрипт.
_PROJECT_ROOT = Path(__file__).resolve().parent.parent
if str(_PROJECT_ROOT) not in sys.path:
    sys.path.insert(0, str(_PROJECT_ROOT))

from src.pipeline import load_model, process_video, save_detections_json
```

```
def main() -> None:
    parser = argparse.ArgumentParser(description="Прогон WildDet3D на відео.")
    parser.add_argument("video_path", type=str, help="Шлях до відео")
    parser.add_argument(
        "--classes", nargs="+", default=["car", "person"],
        help="Класи для пошуку (через пробіл)",
    )
    parser.add_argument("--frame-step", type=int, default=10)
    parser.add_argument("--max-frames", type=int, default=None)
    parser.add_argument("--score-threshold", type=float, default=0.3)
    parser.add_argument(
        "--checkpoint", type=str,
        default="WildDet3D/ckpt/wilddet3d_alldata_all_prompt_v1.0.pt",
    )
    parser.add_argument("--output", type=str, default=None)
    args = parser.parse_args()


    video_path = Path(args.video_path)
    output_path = (
        Path(args.output) if args.output
        else Path("outputs") / f"{video_path.stem}_detections.json"
    )


    # Завантажуємо модель один раз
    model = load_model(args.checkpoint, score_threshold=args.score_threshold)


    # Прогоняємо pipeline
    result = process_video(
        model=model,
        video_path=video_path,
        input_texts=args.classes,
        frame_step=args.frame_step,
        max_frames=args.max_frames,
    )


    # Зберігаємо
    save_detections_json(result, output_path)


    n_det = sum(len(f["detections"]) for f in result["frames"])
    print(f"\nГотово. Оброблено кадрів: {len(result['frames'])}, всього детекцій: {n_det}")



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

Файл scripts/apply_matching.py

```
"""
scripts/apply_matching.py
=========================
Бере JSON з детекціями (вихід run_pipeline.py),
прогоняє IoU-matching між кадрами,
зберігає новий JSON з присвоєними track_id.


Приклад:
    python scripts/apply_matching.py outputs/street_detections.json
    python scripts/apply_matching.py outputs/street_dense.json --iou-threshold 0.3
"""
from __future__ import annotations
```

```
import argparse
import json
import sys
from pathlib import Path
from collections import defaultdict


# Додаємо корінь проекту в sys.path
_PROJECT_ROOT = Path(__file__).resolve().parent.parent
if str(_PROJECT_ROOT) not in sys.path:
    sys.path.insert(0, str(_PROJECT_ROOT))

from src.matching import assign_track_ids



def main() -> None:
    parser = argparse.ArgumentParser(
        description="Застосовує IoU-matching до JSON з детекціями."
    )
    parser.add_argument("input_json", type=str, help="Шлях до JSON з детекціями")
    parser.add_argument(
        "--iou-threshold", type=float, default=0.3,
        help="Мінімальний IoU для матчу (default 0.3)",
    )
    parser.add_argument(
        "--output", type=str, default=None,
        help="Шлях до вихідного JSON (default <input>_tracked.json)",
    )
    parser.add_argument(
        "--ignore-class", action="store_true",
        help="Якщо вказано — матчити детекції незалежно від класу",
    )
    args = parser.parse_args()


    input_path = Path(args.input_json)
    output_path = (
        Path(args.output) if args.output
        else input_path.with_name(input_path.stem + "_tracked.json")
    )


    # Читаємо JSON
    print(f"[1/3] Читаємо: {input_path}")
    with open(input_path, "r", encoding="utf-8") as f:
        data = json.load(f)


    # Витягуємо детекції по кадрах
    detections_per_frame = [frame["detections"] for frame in data["frames"]]
    n_detections_total = sum(len(f) for f in detections_per_frame)
    print(f"      Кадрів: {len(detections_per_frame)}, детекцій усього:
{n_detections_total}")


    # Прогоняємо matching
    print(f"[2/3] Виконуємо matching (iou_threshold={args.iou_threshold}, "
          f"same_class_only={not args.ignore_class})")
    tracked = assign_track_ids(
        detections_per_frame,
        iou_threshold=args.iou_threshold,
        same_class_only=not args.ignore_class,
    )


    # Підраховуємо статистику по треках
    track_lengths: dict[int, int] = defaultdict(int)
    track_class: dict[int, str] = {}
    for frame in tracked:
```

```
        for det in frame:
            tid = det["track_id"]
            track_lengths[tid] += 1
            track_class[tid] = det["class"]

    n_tracks = len(track_lengths)
    n_single_frame = sum(1 for length in track_lengths.values() if length == 1)
    n_stable = sum(1 for length in track_lengths.values() if length >= 3)

    print(f"      Унікальних треків: {n_tracks}")
    print(f"      Однокадрові треки (потенційні фантоми): {n_single_frame}")
    print(f"      Стабільні треки (>=3 кадрів): {n_stable}")

    # Топ-10 найдовших треків
    longest = sorted(track_lengths.items(), key=lambda x: -x[1])[:10]
    print("      Топ-10 найдовших треків:")
    for tid, length in longest:
        print(f"        track_id={tid:3d} class={track_class[tid]:8s} length={length}
кадрів")

    # Записуємо результат
    data["frames"] = [
        {**old_frame, "detections": new_dets}
        for old_frame, new_dets in zip(data["frames"], tracked)
    ]
    data["matching_config"] = {
        "iou_threshold": args.iou_threshold,
        "same_class_only": not args.ignore_class,
        "n_tracks": n_tracks,
        "n_single_frame_tracks": n_single_frame,
        "n_stable_tracks": n_stable,
    }

    print(f"[3/3] Зберігаємо: {output_path}")
    output_path.parent.mkdir(parents=True, exist_ok=True)
    with open(output_path, "w", encoding="utf-8") as f:
        json.dump(data, f, indent=2, ensure_ascii=False)
    print("[OK]")



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

Файл scripts/apply_filtering.py

```
"""
scripts/apply_filtering.py
==========================
Бере tracked JSON (вихід apply_matching.py),
прогоняє фільтрацію треків,
зберігає JSON з очищеним списком детекцій.


Приклад:
    python scripts/apply_filtering.py outputs/street_detections_tracked.json --min-length 2
    python scripts/apply_filtering.py outputs/street_dense_tracked.json --min-length 3 --
min-score 0.4
"""
from __future__ import annotations


import argparse
import json
import sys
from pathlib import Path
```

```
_PROJECT_ROOT = Path(__file__).resolve().parent.parent
if str(_PROJECT_ROOT) not in sys.path:
    sys.path.insert(0, str(_PROJECT_ROOT))

from src.filtering import compute_track_stats, filter_combined



def main() -> None:
    parser = argparse.ArgumentParser(
        description="Фільтрує треки за довжиною та середнім score."
    )
    parser.add_argument("input_json", type=str, help="Шлях до tracked JSON")
    parser.add_argument(
        "--min-length", type=int, default=2,
        help="Мінімальна довжина треку (default 2)",
    )
    parser.add_argument(
        "--min-score", type=float, default=0.4,
        help="Мінімальний середній score_2d треку (default 0.4)",
    )
    parser.add_argument(
        "--output", type=str, default=None,
        help="Шлях до вихідного JSON (default <input>_filtered.json)",
    )
    args = parser.parse_args()

    input_path = Path(args.input_json)
    output_path = (
        Path(args.output) if args.output
        else input_path.with_name(input_path.stem + "_filtered.json")
    )

    # Читаємо JSON
    print(f"[1/4] Читаємо: {input_path}")
    with open(input_path, "r", encoding="utf-8") as f:
        data = json.load(f)

    detections_per_frame = [frame["detections"] for frame in data["frames"]]
    n_before = sum(len(f) for f in detections_per_frame)
    stats_before = compute_track_stats(detections_per_frame)
    print(f"      ДО: {len(stats_before)} треків, {n_before} детекцій")

    # Прогоняємо фільтр
    print(f"[2/4] Фільтр: min_length={args.min_length}, min_mean_score_2d={args.min_score}")
    filtered = filter_combined(
        detections_per_frame,
        min_length=args.min_length,
        min_mean_score_2d=args.min_score,
    )

    n_after = sum(len(f) for f in filtered)
    stats_after = compute_track_stats(filtered)
    print(f"      ПІСЛЯ: {len(stats_after)} треків, {n_after} детекцій")
    if n_before > 0:
        reduction = (1 - n_after / n_before) * 100
        print(f"      Відсоток усунутого шуму: {reduction:.1f}%")

    # Які саме треки лишились
    print("[3/4] Треки що залишились:")
    for tid in sorted(stats_after.keys()):
        s = stats_after[tid]
```

```
            print(f"       track_id={tid:3d} class={s['class']:8s} "
                  f"length={s['length']} mean_score_2d={s['mean_score_2d']:.3f}")

        # Записуємо результат
        data["frames"] = [
            {**old_frame, "detections": new_dets}
            for old_frame, new_dets in zip(data["frames"], filtered)
        ]
        data["filtering_config"] = {
            "min_length": args.min_length,
            "min_mean_score_2d": args.min_score,
            "n_tracks_before": len(stats_before),
            "n_tracks_after": len(stats_after),
            "n_detections_before": n_before,
            "n_detections_after": n_after,
        }

        print(f"[4/4] Зберігаємо: {output_path}")
        output_path.parent.mkdir(parents=True, exist_ok=True)
        with open(output_path, "w", encoding="utf-8") as f:
            json.dump(data, f, indent=2, ensure_ascii=False)
        print("[OK]")



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

Файл scripts/apply_smoothing.py

```
    """
    scripts/apply_smoothing.py
    ==========================
    Застосовує smoothing до tracked-детекцій і порівнює jitter "до vs після".
    """
    from __future__ import annotations

    import argparse
    import json
    import sys
    from pathlib import Path

    _PROJECT_ROOT = Path(__file__).resolve().parent.parent
    if str(_PROJECT_ROOT) not in sys.path:
        sys.path.insert(0, str(_PROJECT_ROOT))

    from src.smoothing import apply_smoothing
    from src.metrics import compute_jitter_per_track, summarize_jitter



    def main() -> None:
        parser = argparse.ArgumentParser(description="Smoothing + jitter analysis.")
        parser.add_argument("input_json", type=str, help="Tracked JSON")
        parser.add_argument("--method", choices=["ma", "ema"], default="ma")
        parser.add_argument("--window", type=int, default=3, help="Для method=ma")
        parser.add_argument("--alpha", type=float, default=0.5, help="Для method=ema")
        parser.add_argument("--output", type=str, default=None)
        args = parser.parse_args()

        input_path = Path(args.input_json)
        output_path = (
            Path(args.output) if args.output
            else input_path.with_name(input_path.stem + "_smoothed.json")
        )
```

```
print(f"[1/4] Читаємо: {input_path}")
with open(input_path, "r", encoding="utf-8") as f:
    data = json.load(f)


detections_per_frame = [frame["detections"] for frame in data["frames"]]
n_det = sum(len(f) for f in detections_per_frame)
print(f"      {len(detections_per_frame)} кадрів, {n_det} детекцій")


jitter_before = compute_jitter_per_track(detections_per_frame, bbox_field="bbox_2d")
stats_before = summarize_jitter(jitter_before)
print(f"[2/4] Jitter ДО згладжування:")
print(f"      треків з jitter: {stats_before['count']}, "
      f"mean={stats_before['mean']:.2f}px, "
      f"max={stats_before['max']:.2f}px, "
      f"min={stats_before['min']:.2f}px")


print(f"[3/4] Smoothing: method={args.method}, "
      + (f"window={args.window}" if args.method == "ma" else f"alpha={args.alpha}"))
smoothed = apply_smoothing(
    detections_per_frame,
    method=args.method,
    window=args.window,
    alpha=args.alpha,
)


jitter_after = compute_jitter_per_track(smoothed, bbox_field="bbox_2d_smoothed")
stats_after = summarize_jitter(jitter_after)
print(f"      Jitter ПІСЛЯ згладжування:")
print(f"      треків з jitter: {stats_after['count']}, "
      f"mean={stats_after['mean']:.2f}px, "
      f"max={stats_after['max']:.2f}px, "
      f"min={stats_after['min']:.2f}px")


if stats_before["mean"] > 0:
    reduction = (1 - stats_after["mean"] / stats_before["mean"]) * 100
    print(f"      Зменшення jitter: {reduction:.1f}%")


print(f"      Per-track деталі (top 10):")
for tid in sorted(jitter_before.keys())[:10]:
    before = jitter_before[tid]
    after = jitter_after.get(tid, 0.0)
    print(f"        track_id={tid:3d}  before={before:6.2f}px  after={after:6.2f}px")


data["frames"] = [
    {**old_frame, "detections": new_dets}
    for old_frame, new_dets in zip(data["frames"], smoothed)
]
data["smoothing_config"] = {
    "method": args.method,
    "window": args.window if args.method == "ma" else None,
    "alpha": args.alpha if args.method == "ema" else None,
    "jitter_mean_before": stats_before["mean"],
    "jitter_mean_after": stats_after["mean"],
}


print(f"[4/4] Зберігаємо: {output_path}")
output_path.parent.mkdir(parents=True, exist_ok=True)
with open(output_path, "w", encoding="utf-8") as f:
    json.dump(data, f, indent=2, ensure_ascii=False)
print("[OK]")
```

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