НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
_____/Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення системи розпізнавання рухомих об'єктів і побудови сцени на основі стереозображення»

Спеціальність 121 «Інженерія програмного забезпечення»
ОП «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент / Вайганг Г.О./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи : / Віннічук Д.О. /
підпис ПІБ

Консультант бакалаврської кваліфікаційної роботи
д.т.н., професор / Семко В.В./
Науковий ступень та вчене звання підпис ПІБ

Виконав: / Якимович Н.А./
підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /
підпис
" 16 " грудня 2024 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи
студент Якимович Назарій Андрійович

Спеціальність 121 «Інженерія програмного забезпечення»

ОП «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення системи розпізнавання рухомих об'єктів і побудови сцени на основі стереозображення.

Затверджена наказом ректора НУБіП України № 2249 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2025 . 06 . 02

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

1. Системний аналіз предметної.
2. Архітектура та проектування системи комп'ютерного бачення.
3. Розробка інформаційного та програмного забезпечення.
4. Тестування та впровадження системи.
5. Висновки.

Керівник бакалаврської кваліфікаційної роботи_____ / Віннічук Д.О. /
підпис ініціали та прізвище
Консультант бакалаврської кваліфікаційної роботи_____ / Семко В.В. /
підпис ініціали та прізвище
Завдання прийняв до виконання _____ / Якимович Н.А. /
підпис ініціали та прізвище
Дата отримання завдання 2024 . 12 . 16

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. ЗНМ, CNN — згорткові нейронні мережі, алгоритми для обробки зображень.
2. GPU — графічний процесор, прискорювач обчислень для.
3. GUI — графічний інтерфейс користувача, взаємодію з програмою.
4. YOLO — You Only Look Once, алгоритм для швидкого виявлення об’єктів на зображеннях у реальному часі.
5. SORT — Simple Online and Realtime Tracking, алгоритм для відстеження об’єктів у відеопотоці.
6. ОС — операційна система, програмне забезпечення для керування апаратними та програмними ресурсами комп’ютера.
7. R-CNN — Region-based Convolutional Neural Network, сімейство алгоритмів для виявлення об’єктів із виділенням регіонів.
8. SSD — Single Shot MultiBox Detector, алгоритм для швидкого виявлення об’єктів за один прохід мережі.
9. CUDA — Compute Unified Device Architecture, платформа NVIDIA для паралельних обчислень на GPU.
10. ROS — Robot Operating System, фреймворк для розробки програмного забезпечення роботів.
11. CPU — центральний процесор, основний обчислювальний компонент комп’ютера.
12. API — Application Programming Interface, інтерфейс для взаємодії між різними програмними компонентами.
13. RGB — Red Green Blue, модель представлення кольорів для зображень.

14. COCO — Common Objects in Context, набір даних для задач комп’ютерного зору, таких як виявлення та сегментація об’єктів.

15. FPS — Frames Per Second, кількість кадрів за секунду, міра швидкості обробки відео.

16. СУБД — система управління базами даних, програмне забезпечення для роботи з базами даних.

17. SGBM — Semi-Global Block Matching, алгоритм для створення карт глибини в стереозорі.

18. BM — Block Matching, базовий алгоритм для оцінки руху або стереозору шляхом порівняння блоків зображень.

ВСТУП

Сучасний розвиток комп’ютерного зору відкриває нові можливості для автоматизації процесів аналізу зображень і відео, що є критично важливим для таких сфер, як робототехніка, автономний транспорт, безпека та віртуальна реальність. Одним із ключових напрямів у цій галузі є розпізнавання рухомих об’єктів і побудова тривимірних сцен на основі стереозображень, що дозволяє створювати детальні просторові моделі навколишнього середовища. Використання згорткових нейронних мереж (ЗНМ), зокрема моделей на кшталт YOLO, значно підвищує точність і швидкість виявлення об’єктів, тоді як алгоритми стереозбігу, такі як StereoSGBM, забезпечують створення карт глибини для реконструкції 3D-сцени.

Метою даного проєкту є розробка інформаційної системи, яка інтегрує можливості комп’ютерного зору для розпізнавання рухомих об’єктів, відстеження їх траєкторій і побудови тривимірної сцени на основі стереозображень. Система поєднує обробку зображень і відео в реальному часі, використання ЗНМ для виявлення об’єктів, алгоритми стереозбігу для створення карт глибини та зручний графічний інтерфейс для взаємодії користувача. Для зберігання даних про об’єкти, їх траєкторії та результати обробки застосовується реляційна база даних, що забезпечує ефективне управління інформацією.

Розроблена система спрямована на вирішення практичних завдань, таких як аналіз сцен у реальному часі, моніторинг рухомих об’єктів і створення 3D-моделей для подальшого використання в автоматизованих системах. У проєкті особлива увага приділяється інженерним аспектам реалізації, включаючи оптимізацію алгоритмів, інтеграцію сучасних бібліотек (OpenCV, Ultralytics YOLO) і забезпечення високої точності обробки даних. Цей документ описує системний аналіз, архітектуру, програмну реалізацію, тестування та рекомендації щодо впровадження системи, демонструючи її відповідність поставленим вимогам і потенціал для подальшого розвитку.

Алгоритм програмного додатку. За результатами виконаної роботи було підготовлено тези доповідей для конференції "Теоретичні та прикладні аспекти розробки комп’ютерних систем '2025'", яка пройшла 24 травня 2025 року.

Структура бакалаврської роботи. Робота складається з 4 основних розділів,, 14 використаних джерел та 4 додатків. У 1 розділі "Системний аналіз предметної області" наводиться опис предметної області, аналіз вимог до програмної системи, моделювання предметної області, огляд інформаційних джерел та існуючих рішень, а також постановка завдання. У 2 розділі "Архітектура та проектування системи комп'ютерного бачення" викладається загальна архітектура інформаційної системи, архітектура модуля стереозору та обробки зображень, взаємодія компонентів (камера, обробка, розпізнавання, візуалізація), а також інтерфейс користувача з візуалізацією сцени та параметрами керування. У 3 розділі "Розробка інформаційного та програмного забезпечення" представлено систему управління інформаційною базою, розробку інформаційної бази, вибір інструментарію для створення прикладного програмного забезпечення та алгоритмізацію й програмування програмних модулів. У 4 розділі "Тестування та впровадження системи" наведено тестування системи, вимоги до апаратного та програмного забезпечення, склад інсталяційного пакету, а також обмеження системи та рекомендації щодо експлуатації.

1. Системний аналіз предметної області

1.1 Опис предметної області

Комп’ютерний зір є ключовою галуззю сучасних інформаційних технологій, що дозволяє машинам інтерпретувати візуальну інформацію, отриману з зображень або відеопотоків, подібно до людського зору. Ця дисципліна охоплює широкий спектр задач, таких як розпізнавання об’єктів, аналіз руху, сегментація сцен і побудова тривимірних моделей. У контексті даного проєкту особлива увага приділяється розпізнаванню рухомих об’єктів і побудові тривимірних сцен на основі стереозображень із використанням згорткових нейронних мереж (ЗНМ). Такі технології знаходять застосування в численних сферах: від автономних транспортних засобів і робототехніки до систем відеоспостереження, медичної візуалізації, віртуальної та доповненої реальності.[1]

Процес обробки в комп’ютерному зорі включає кілька ключових етапів. Перший — це збір даних, який передбачає отримання зображень або відео за допомогою камер, стереокамер чи інших сенсорів. Далі йде попередня обробка, що включає корекцію освітлення, видалення шуму, нормалізацію кольорів або перетворення зображень у відтінки сірого для спрощення аналізу. На етапі виділення ознак виявляються ключові характеристики, такі як краї, кути, текстури чи форми. Сучасні методи, зокрема згорткові нейронні мережі (ЗНМ), автоматизують цей процес, замінюючи традиційні підходи, такі як детектори країв чи алгоритми SIFT. Після цього виконується аналіз і інтерпретація даних для класифікації об’єктів, визначення їхньої глибини, відстеження руху чи побудови тривимірних моделей. На завершальному етапі результати використовуються для прийняття рішень, наприклад, для навігації роботів, розпізнавання перешкод або аналізу поведінки.

Стереозображення є основою для створення тривимірних сцен. Цей підхід базується на використанні двох зображень однієї сцени, отриманих із різних

кутів зору (ліве та праве зображення), що імітує бінокулярний зір людини. Різниця в позиціях об’єктів між цими зображеннями, відома як диспаратність, використовується для обчислення відстані до об’єктів у сцені. Алгоритми стереозбігу, такі як StereoBM (Block Matching) або StereoSGBM (Semi-Global Block Matching), аналізують диспаратність для створення карти глибини — двовимірного представлення, де значення пікселів відображають відстань до об’єктів. Карта глибини є критично важливою для реконструкції тривимірної сцени, що дозволяє системі визначати просторове розташування об’єктів і їх взаємозв’язки.[2]

Стереозображення спирається на епіполярну геометрію, яка описує співвідношення між двома камерами. Основні параметри включають базову лінію (відстань між центрами камер) і фокусну відстань, які впливають на точність оцінки глибини. Калібрування камер є необхідним для компенсації спотворень об’єктива та вирівнювання зображень, що забезпечує коректне зіставлення пікселів. Глибина обчислюється за допомогою триангуляції за формулою $Z = \frac{f * B}{d}$, де (z) — глибина, (f) — фокусна відстань, (B) — базова лінія, (d) — диспаратність. Карти глибини можуть візуалізуватися в чорно-білому або кольоровому форматі, що полегшує інтерпретацію тривимірної структури сцени.

Згорткові нейронні мережі (ЗНМ) революціонізували комп’ютерний зір, замінивши традиційні методи ручного виділення ознак автоматизованими підходами. ЗНМ здатні навчатися виділяти складні візуальні шаблони, такі як форми, текстури чи семантичні ознаки, що робить їх ефективними для класифікації зображень, детекції об’єктів, сегментації та розпізнавання рухомих об’єктів. Основні компоненти ЗНМ включають згорткові шари для виділення локальних ознак (країв, кутів, текстур), шари пулінгу для зменшення розмірності даних із збереженням важливих характеристик, повнозв’язні шари для класифікації чи регресії, а також активаційні функції, такі як ReLU, які додають нелінійність для моделювання складних залежностей [3].

Моделі, такі як YOLO (You Only Look Once), є прикладом ефективного використання ЗНМ для детекції об’єктів у реальному часі. Вони обробляють зображення за один прохід, передбачаючи координати обмежувальних рамок, класи об’єктів і ймовірності їхньої наявності. ЗНМ стійкі до змін у масштабі, поворотах і освітленні, що робить їх придатними для обробки складних сцен, включаючи відеопотоки. Інтеграція результатів детекції об’єктів із картами глибини дозволяє не лише ідентифікувати об’єкти, а й визначати їхнє розташування в тривимірному просторі, що є основою для аналізу руху та реконструкції сцен.

Поєднання стереозору та ЗНМ створює потужний інструмент для аналізу тривимірних сцен і відстеження рухомих об’єктів. Стереозображення забезпечує оцінку глибини кожного пікселя, що необхідно для локалізації об’єктів у тривимірному просторі. ЗНМ, своєю чергою, дозволяють ідентифікувати та класифікувати об’єкти, визначаючи їхні координати та категорії. Інтеграція цих підходів дає змогу не лише виявляти об’єкти, а й відстежувати їхні траєкторії у відеопотоці, що важливо для таких застосувань, як моніторинг транспорту, навігація роботів або аналіз поведінки [4].

Для відстеження рухомих об’єктів використовуються алгоритми трекінгу, такі як DeepSORT, які комбінують детекцію об’єктів із методами прогнозування руху, наприклад, фільтром Калмана. Це дозволяє системі аналізувати траєкторії об’єктів між кадрами, підвищуючи точність відстеження. Такий підхід є основою для застосувань у реальному часі, наприклад, для виявлення пішоходів чи транспортних засобів, оцінки їхньої відстані та швидкості, а також передбачення потенційних зіткнень [4]

Управління даними, отриманими в процесі обробки, є ще одним важливим аспектом. Стереозображення, відеопотоки, карти глибини, координати об’єктів і їхні траєкторії потребують структурованого зберігання та швидкого доступу. Для цього використовуються реляційні бази даних, такі як PostgreSQL, які дозволяють ефективно зберігати великі обсяги даних, включаючи бінарні

об’єкти (BYTEA для зображень і відео), і забезпечують можливість їхнього подальшого аналізу. Інформаційна система, побудована на основі такої бази даних, забезпечує інтеграцію всіх компонентів — від обробки даних до їхньої візуалізації через графічний інтерфейс.

Про аналізувавши опис предметної обласні, була сформована предметна область, яка поєднує комп’ютерний зір, обробку стереозображень, глибоке навчання та інформаційні технології для створення систем, здатних аналізувати складні сцени, розпізнавати рухомі об’єкти та будувати тривимірні моделі з високою точністю. Вона вимагає комплексного підходу, що включає алгоритми обробки зображень, нейронні мережі, трекінг, управління даними та інженерну реалізацію, щоб забезпечити надійність і практичну цінність системи.

1.2 Аналіз вимог до програмної системи

Для розробки ефективної інформаційної системи розпізнавання рухомих об’єктів і побудови сцени на основі стереозображення необхідно чітко визначити функціональні та нефункціональні вимоги до програмної складової системи.

Функціональні вимоги: отримання зображень із двох стереокамер- система повинна забезпечуватися даними (зображеннями або відео) з двох стереокамер та розміщених відповідно до вимог стандартів стереозору; попередня обробка зображень або відео - необхідно реалізувати етапи нормалізації, фільтрації шумів, вирівнювання освітлення та корекції геометричних спотворень; розрахунок карти глибини- система повинна аналізувати пару зображень або відео і генерувати карту глибини для подальшої побудови просторової сцени; розпізнавання рухомих об’єктів- на основі згорткових нейронних мереж повинно бути реалізоване виявлення та класифікація рухомих об’єктів на сцені (наприклад, людей, транспортних засобів тощо); трекінг об’єктів- система має підтримувати відстеження об’єктів між кадрами з урахуванням змін їх положення в просторі; побудова та оновлення тривимірної сцени- повинна бути реалізована реконструкція сцени на основі стереозображення з можливістю її

оновлення у реальному часі; візуалізація результатів- система повинна відображати просторову сцену, позначати розпізнані об’єкти, надавати дані про їх положення, швидкість, клас тощо; користувацький інтерфейс- має бути реалізований інтерфейс для керування параметрами камери та обробки зображення та відео, змога імпортувати данних в базу даних, перегляду результатів та експорту та імпорт.

Нефункціональні вимоги: продуктивність- система має працювати в режимі реального часу із затримкою обробки; масштабованість- архітектура повинна передбачати можливість розширення – підключення додаткових модулів чи камер; модульність- програмна система має складатися з окремих модулів (захоплення відео, обробка, розпізнавання, візуалізація), для того щоб легко можна було модифікувати або замінити модулі; сумісність- система має підтримувати роботу з популярними форматами відео (MJPEG, H.264) та зображенням (PNG,JPG,JPEG,BMP) та працювати під сучасними ОС (Linux, Windows); надійність- програма повинна стабільно працювати в умовах довготривалого та безперервної роботи; безпека- має бути передбачено захист доступу до конфігурації та результатів аналізу.

З наведених функціональних та нефункціональних вимог були сформульовані вимоги до системи є запорукою її подальшої ефективної реалізації, тестування та впровадження у практичне використання.

1.3 Моделювання предметної області

Моделювання предметної області є важливим етапом розробки програмної системи, оскільки дозволяє формалізувати структуру даних, взаємозв’язки між об’єктами та процесами, а також визначити логіку взаємодії елементів системи. У контексті інформаційної системи розпізнавання рухомих об’єктів і побудови сцени на основі стереозображення, предметна область охоплює кілька ключових компонентів:

Основні об’єкти предметної області: стереокамера – джерело зображень та відео, що передають потоки даних для подальшої обробки; карта глибини–

результат стереоаналізу, що визначає відстань до кожної точки сцени; об’єкт сцени – рухомий або статичний елемент, що знаходиться у полі зору камер; користувач – оператор або інша система, яка взаємодіє з результатами обробки; модель згорткової нейронної мережі – програмний модуль, що здійснює аналіз зображень і класифікацію об’єктів.

Взаємозв’язки об’єктів:

Між зазначеними об’єктами існує низка зв’язків, які можна представити у вигляді діаграм UML:

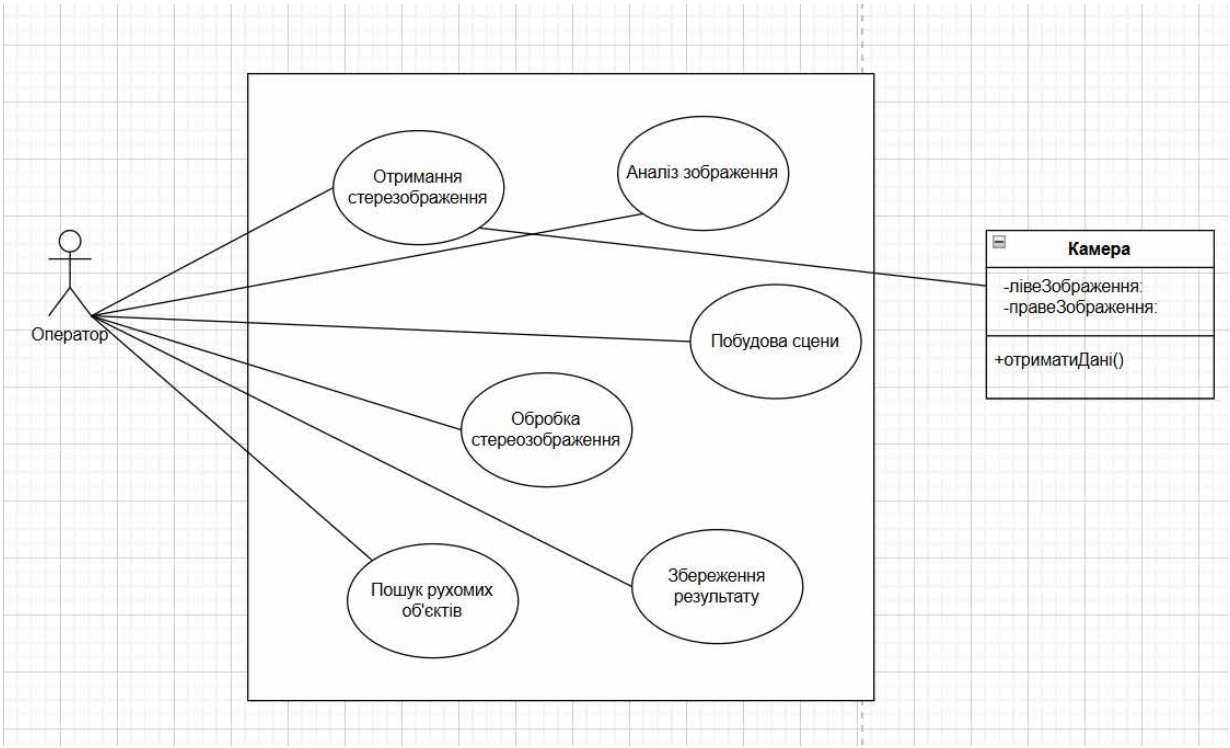


Рис. 1 Діаграма прецедентів демонструє сценарії використання системи: отримання зображень, побудова карти глибини, розпізнавання об’єктів, візуалізація сцени.

Діаграма предметної області описує основні сутності та їх взаємозв’язки в системі розпізнавання рухомих об’єктів і побудови сцени на основі стереозображення з використанням згорткових нейронних мереж. Центральною сутністю є Оператор, який взаємодіє з системою. Оператор пов’язаний із такими сутностями: Отримання стереозображення: Оператор отримує стереозображення, яке є основою для подальшої обробки. Аналіз зображення: Система аналізує отримане стереозображення, використовуючи згорткові

нейронні мережі. Побудова сцени: На основі аналізу система будує сцену, враховуючи розташування об’єктів. Збереження результату: Результати обробки зберігаються для подальшого використання. Попуск рухомих об’єктів: Система виявляє та ідентифікує рухомі об’єкти на основі стереозображення.

Додатково діаграма включає сутність Камера, яка має атрибути: Тип стереозображення: Вказує, яке саме стереозображення використовується. Отримання(): Метод, що відповідає за отримання стереозображення з камери.

Ця діаграма відображає основні компоненти та їх взаємодію в процесі роботи системи.

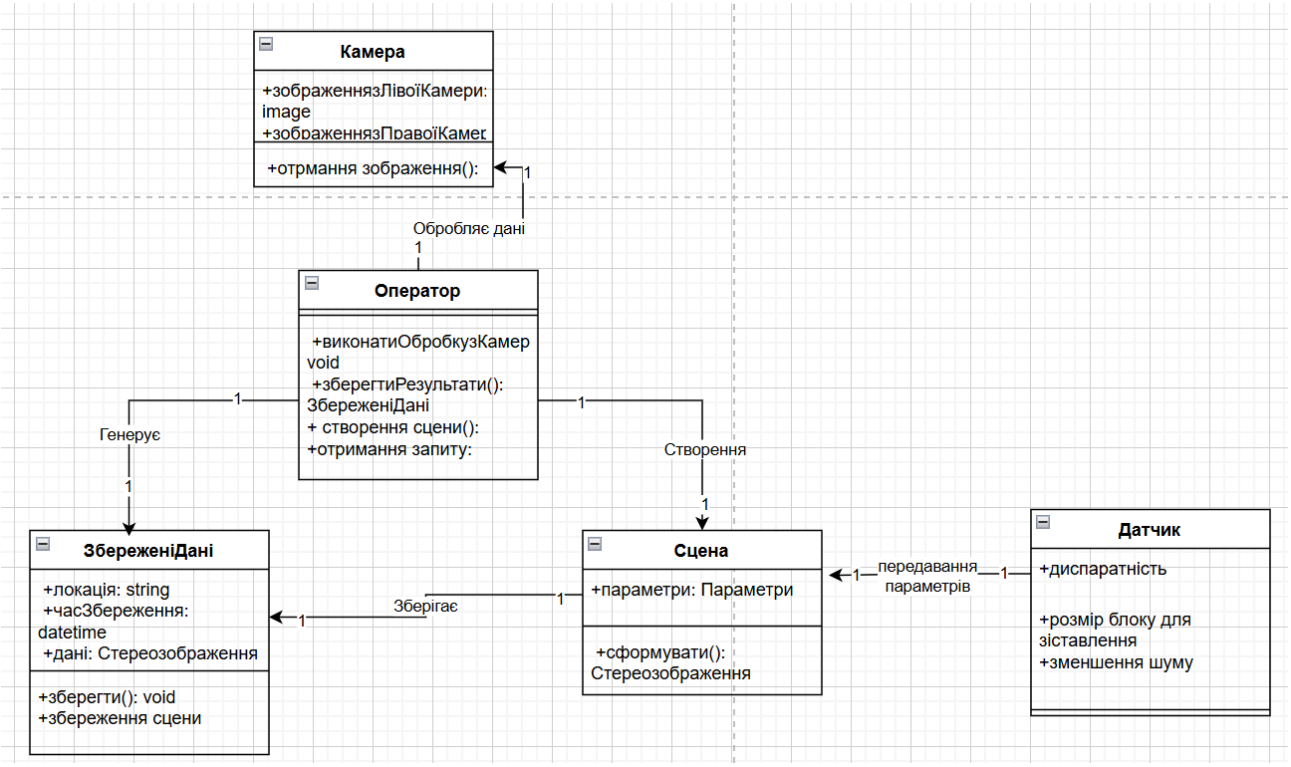


Рис. 2 Діаграма класів відображає логічну структуру даних та зв’язки між об’єктами, зокрема класи Камера, Оператор, Сцена, Збереженні дані, Датчик.

Опис класів: Камера (Camera):Атрибути: зображення Лівої Камери: image — Зображення, отримане з лівої камери; зображення Правої Камери: image — Зображення, отримане з правої камери. Методи: отримати Зображення(): image — Метод для отримання зображень із камер. Роль класу відповідає за захоплення стереозображень (з лівої та правої камер), які є основою для

подальшої обробки. Оператор (Operator): Атрибути: виконати Обробку: void — Змінна або команда для запуску обробки. Методи: зберегти Результати(): void — Метод для збереження результатів обробки; стерео Калібрування(): void — Метод для калібрування стереозображень; отримати Запит(): string — Метод для отримання запиту на обробку. Роль класу управляє процесом обробки стереозображень, включаючи калібрування та збереження результатів. Збереженні Дані (DataStorage) Атрибути: локація: string — Локалізація даних (наприклад, шлях до файлу); час Збереження: datetime — Час збереження даних. дані: Стереозображення — Дані у вигляді стереозображення. Методи: зберегти(): void — Метод для збереження даних; Зберегти Сцени(): void — Метод для збереження сцени. Роль класу відповідає за збереження оброблених даних, таких як стереозображення та сцени. Сцена (Scene): Методи: сформувати(): Стереозображення — Метод для формування сцени на основі стереозображення. Роль класу відповідає за створення сцени на основі оброблених стереозображень. Датчик (Sensor): Атрибути: диспаратність: float — Значення диспаратності (різниці між зображеннями для побудови глибини). Методи: Розмір Блоку Для Визначення Диспаратності(): int — Метод для визначення розміру блоку при обчисленні диспаратності; Встановити Шум(): int — Метод для встановлення рівня шуму. Змінення Чутливості(): int — Метод для зміни чутливості датчика. Роль класу відповідає за обробку диспаратності, що є ключовим для визначення глибини в стереозображенні.

Взаємозв’язки між класами: Камера → Оператор (1:1): Камера передає зображення оператору для подальшої обробки. Один екземпляр камери працює з одним оператором. Оператор → Збережені Дані (1:1): Оператор взаємодіє із класом збереження даних, передаючи оброблені дані для зберігання. Збережені Дані → Сцена (1:1): Клас збереження даних асоціюється з класом сцени, зберігаючи сформовані сцени. Сцена → Оператор (1:1): Сцена передає сформовані стереозображення назад оператору для подальшої обробки або збереження. Датчик → Сцена (1:1,

через "передавання параметри"): Датчик передає параметри (наприклад, диспаратність) класу сцени для формування стереозображення.

Діаграма класів описує систему, де камера захоплює стереозображення, оператор управляє їх обробкою (включаючи калібрування), датчик обчислює диспаратність для визначення глибини, сцена формує тривимірне зображення, а клас збереження даних відповідає за збереження результатів. Система побудована модульно, з чітким розподілом обов'язків між класами, що сприяє легкому розширенню та модифікації.

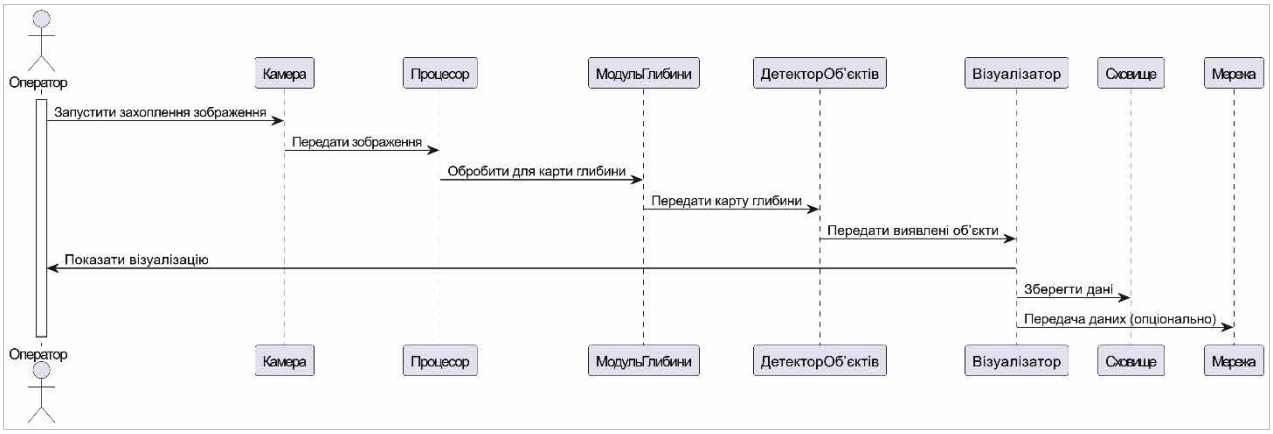


Рис. *3* Діаграма діяльності описує послідовність дій при обробці відеопотоку*:* захоплення зображення → обробка → побудова карти глибини → виявлення об'єктів → візуалізація → збереження або передача даних.

Діаграма послідовності, ілюструє взаємодію між компонентами системи розпізнавання рухомих об'єктів і побудови сцени на основі стереозображення з використанням згорткових нейронних мереж.

Учасники (об'єкти): Оператор: Користувач або система, яка ініціює процес; Камера: Відповідає за захоплення зображень; Процесор: Виконує обробку даних, ймовірно включаючи аналіз згортковими нейронними мережами; Детектор Об'єктів: Виявляє рухомі об'єкти; Візуалізатор: Відображає результати; Сцена: Формує тривимірну сцену; Мережа: Зберігає або передає дані.

Послідовність дій: Оператор → Камера: Оператор запускає захоплення зображень, надсилаючи команду "Запустити захоплення

зображення"; Камера → Процесор: Камера передає захоплені зображення процесору для подальшої обробки ("Передати зображення"); Процесор → Детектор Об’єктів: Процесор обробляє зображення (застосуванням згорткових нейронних мереж для створення карти глибини) і передає видлені об’єкти детектору об’єктів ("Передати видлені об’єкти"); Детектор Об’єктів → Візуалізатор: Детектор об’єктів надсилає згенеровані дані (виявлені об’єкти) візуалізатору для відображення ("Згенерті дані"); Візуалізатор → Сцена: Візуалізатор передає дані сцени для формування тривимірного зображення ("Передати дані (оновлювані)"); Сцена → Мережа: Сцена відправляє сформовані дані в мережу для збереження або передачі; Оператор → Візуалізатор: Оператор паралельно надсилає запит на показ візуалізації ("Показати візуалізацію"), щоб переглянути результати.

Діаграма послідовності показує, як оператор ініціює захоплення зображень камерою, після чого дані обробляються процесором (із застосуванням згорткових нейронних мереж для створення карти глибини), передаються детектору об’єктів для виявлення рухомих об’єктів, візуалізуються і формуються в сцену. Результат зберігається або передається через мережу. Процес є послідовним із чіткою взаємодією між компонентами. Процеси предметної області можна умовно поділити на такі етапи:

1. Захоплення відеоданих з камер у режимі реального часу.
2. Обробка стереозображень для побудови карти глибини.
3. Аналіз глибини та RGB-даних для виявлення об’єктів.
4. Розпізнавання рухомих об’єктів з використанням згорткових нейронних мереж.
5. Формування тривимірної сцени та її візуалізація.
6. Виведення результатів у вигляді графічного інтерфейсу або передача даних іншим модулям системи.

1.4 Огляд інформаційних джерел та існуючих рішень

Для побудови ефективної інформаційної системи розпізнавання рухомих об’єктів і побудови сцени на основі стереозображень необхідно провести огляд наукових та технічних джерел, що описують відповідні технології, а також проаналізувати існуючі рішення в галузі комп’ютерного бачення.

Наукові та технічні джерела

1. Алгоритми стереозору. Основою побудови тривимірної сцени є стереоаналіз – процес обчислення диспаратності між парами зображень. У літературі описано численні підходи, зокрема методи блокового зіставлення (Block Matching), напівглобального узгодження (Semi-Global Matching), графових методів, глибоких стереомереж (StereoNet, PSMNet, GA-Net). Відомими роботами є [Scharstein & Szeliski, 2002] та дослідження з KITTI Stereo Benchmark.[2]

2. Глибинне навчання для аналізу зображень. Згорткові нейронні мережі (CNN) є стандартом де-факто в задачах розпізнавання об’єктів. Популярні архітектури включають YOLO, SSD, Faster R-CNN, які показують високу точність у режимі реального часу. Також використовуються U-Net, DeepLab для семантичної сегментації.[2]

3. Виявлення та відстеження руху. Методи на кшталт Optical Flow (Lucas-Kanade, Farnebäck), background subtraction (MOG, KNN), а також трекери (SORT, Deep SORT, ByteTrack) широко застосовуються для визначення та супроводу рухомих об’єктів.[3]

4. Візуалізація 3D-сцен. Для візуалізації результатів використовуються бібліотеки OpenGL, Open3D, а також фреймворки на кшталт Unity чи Unreal Engine у складніших проектах.[4]

5. Апаратна підтримка. Системи, які працюють у реальному часі, зазвичай використовують GPU-прискорення (NVIDIA CUDA, TensorRT), вбудовані обчислювальні платформи (Jetson, Intel Movidius) або FPGA.[5]

Аналіз існуючих рішень

- OpenCV – найпопулярніша бібліотека для обробки зображень і стереозору. Містить реалізації базових алгоритмів для stereo matching, оптичного потоку, виявлення об’єктів, фільтрації тощо.
- TensorFlow / PyTorch – фреймворки для глибинного навчання, що дозволяють створювати, тренувати та інтегрувати нейронні мережі для обробки зображень.
- ZED Stereo Camera SDK – програмне забезпечення від Stereolabs, яке забезпечує високоточну карту глибини, виявлення об’єктів і трекінг.
- DepthAI / Luxonis – системи на базі OAK-D камер, що об’єднують стереозір і AI-обробку на борту.
- ROS (Robot Operating System) – програмне середовище для створення роботизованих систем, що активно використовується в задачах комп’ютерного бачення, з підтримкою пакетів для stereo vision, SLAM, object detection.

1.5 Постановка завдання

Метою даної кваліфікаційної роботи є розробка інформаційної системи для розпізнавання рухомих об’єктів і побудови сцени на основі стереозображення з використанням згорткових нейронних мереж (CNN). Система має забезпечити отримання, обробку та інтерпретацію зображень у режимі реального часу, що дозволяє виявляти та візуалізувати об'єкти в просторі з урахуванням їхнього руху і положення у тривимірному середовищі.

Основні задачі, що необхідно вирішити це: організувати отримання синхронізованих зображень з двох камер (стереопари) це забезпечить основу для аналізу просторової інформації про сцену; реалізувати модуль стереозору для обчислення карти глибини для необхідно застосувати один із сучасних stereo matching алгоритмів або глибоку нейронну мережу для точного визначення просторового розміщення об'єктів; розробити модуль виявлення та відстеження рухомих об’єктів основним завданням є сегментація фону, відстеження об'єктів у потоці кадрів та побудова їхніх траєкторій; інтегрувати згорткову нейронну

мережу для класифікації та розпізнавання об’єктів це вибір архітектури (наприклад, YOLOv5, MobileNet, або інша легка модель) має ґрунтуватися на компромісі між точністю і швидкодією; реалізувати модуль візуалізації тривимірної сцени для побудова 3D-моделі сцени з інтегрованими об’єктами, що змінюють положення в просторі; розробити користувацький інтерфейс для керування параметрами системи і перегляду результатів інтерфейс повинен забезпечити зміну налаштувань, вивід глибинної карти, об’єктів, їх класів та траєкторій; забезпечити тестування та оцінку ефективності системи провести експерименти для перевірки точності розпізнавання, обчислення глибини, а також продуктивності системи в реальних умовах.

Обмеження та припущення: система розрахована на роботу з фіксованою стереопарою камер; освітлення сцени вважається відносно стабільним, що дозволяє зменшити вплив шумів на якість глибинної карти; об'єкти для розпізнавання належать до обмеженого набору класів, відомих на етапі тренування CNN; пріоритет надається роботі в реальному часі, тому обираються оптимізовані алгоритми та моделі.

2. Архітектура та проектування системи комп'ютерного бачення

2.1 Загальна архітектура інформаційної системи

Інформаційна система для розпізнавання рухомих об’єктів і побудови тривимірних сцен на основі стереозображень із використанням згорткових нейронних мереж (ЗНМ) спроєктована як модульна архітектура, що забезпечує гнучкість, масштабованість і ефективну взаємодію між компонентами. Архітектура системи побудована з урахуванням вимог до обробки великих обсягів даних, реального часу, точності розпізнавання та зручності користувача. Вона включає кілька основних модулів, кожен із яких відповідає за певний етап обробки даних або взаємодію з користувачем. Загальна структура системи представлена як набір взаємопов’язаних компонентів, що інтегрують алгоритми комп’ютерного зору, бази даних і графічний інтерфейс.

Основні компоненти системи: модуль введення даних: приймає стереозображення, відеопотоки або дані з веб-камер. Для реалізації того щоб записати дані в базу даних було використано tkinter(filedialog) для виберу файлів (png, .jpg, .mp4 тощо). Особливості модуля є перевіряє валідність даних, виконує попередню обробку (зміна розміру, нормалізація); модуль обробки стереозображень: створює карти глибини через стереозбіг. Для реалізації алгоритму про створення карти глибин було використано алгоритм StereoSGBM від бібліотеки (OpenCV), обробка сірошкальних зображень, нормалізація карт глибини. Особливості модуля є налаштування параметрів (minDisparity, numDisparities) через GUI, оптимізація для GPU; модуль розпізнавання об’єктів: виявляє та класифікує об’єкти, визначає обмежувальні рамки. Для реалізації була технологія YOLOv8 та файл (yolov8n.pt), обробка лівого/правого зображення та відео, де накладає зелених квадратів з надписом що це за об’єкт. Особливості модуля є швидкість ≥10 FPS на

GPU, відображення зелених квадратів з назвою об’єкта; модуль відстеження об’єктів: аналізує траєкторії руху об’єктів. Для реалізації було використано дві бібліотеки DeepSORT та YOLOv8, збереження координат та ідентифікаторів. Особливості модуля є увімкнення трекінгу через GUI; модуль управління базою даних: зберігає зображення, відео, карти глибини, параметри. Для реалізації була використана база даних PostgreSQL, дані зберігаються у форматі BYTEA, таблиці: images, videos, camers, depth_maps, stereo_params. Особливості модуля є операції з базою даною та швидкий доступ до даних; модуль GUI: забезпечує взаємодію користувача з програмою (завантаження, налаштування, перегляд). Для реалізації GUI була взята бібліотека tkinter (MainInterface, SettingsInterface), вкладки для зображень, відео, реального часу. Особливості модуля є підсказки (ToolTip); модуль виведення результатів: візуалізує та зберігає результати. Для реалізації виведення результату була використанна бібліотека OpenCV. Особливості модуля візуальне розташування вікон, можливість переривати обробку.

Взаємодія компонентів: Модуль введення даних отримує стереозображення або відеопотоки від користувача через GUI або камери. Дані передаються до модуля обробки стереозображень; Модуль обробки створює карту глибини за допомогою StereoSGBM і передає її до модуля виведення результатів для візуалізації; Модуль YOLOv8 обробляє ліве та праве зображення, визначає об’єкти та їхні координати. Результати узгоджуються та накладаються на карту глибини; Для відеопотоків модуль трекінгу аналізує рух об’єктів, зберігаючи їхні траєкторії в базі даних; Усі результати (зображення, відео, карти глибини, траєкторії) зберігаються в PostgreSQL через модуль управління базою даних; GUI координує роботу всіх модулів, дозволяючи користувачу задавати параметри, запускати обробку та переглядати результати.

Набір технологій

- Мова: Python 3.8+.
- Бібліотеки: OpenCV, Ultralytics YOLO, psycopg2, tkinter.

- База даних: PostgreSQL.
- Апаратне забезпечення: GPU (NVIDIA CUDA), CPU 4 ядра, 16 ГБ RAM.

2.2 Архітектура модуля стереозору та обробки зображень

Модуль стереозору та обробки зображень є центральним компонентом інформаційної системи для розпізнавання рухомих об’єктів і побудови тривимірних сцен на основі стереозображень. Його основне завдання полягає в обробці парного стереовведення (ліве та праве зображення або відеопотік), створенні карт глибини, попередній обробці зображень і підготовці даних для подальшого розпізнавання об’єктів за допомогою згорткових нейронних мереж (ЗНМ). Модуль спроєктовано як гнучкий і оптимізований компонент, що забезпечує високу точність, швидкодію та інтеграцію з іншими частинами системи, такими як модуль розпізнавання об’єктів і графічний інтерфейс користувача (GUI).

Функціональні завдання модуля: Попередня обробка: нормалізація зображень; перетворення в сірошкальний формат; зміна розміру для оптимізації. Стереозбіг: обчислення диспаратності; генерація карти глибини; візуалізація (чорно-біла або кольорова). Інтеграція з YOLOv8: передача зображень для виявлення; накладання обмежувальні рамків на карту глибини. Обробка відеопотоків: послідовна обробка кадрів; захоплення з веб-камер у реальному часі. Оптимізація: налаштування параметрів стереозбігу; використання GPU.

Модуль стереозору та обробки зображень побудовано як набір взаємопов’язаних підмодулів, кожен із яких виконує спеціалізовану функцію. Архітектура модуля представлена наступними компонентами: Під модуль введення даних: отримання стереозображення або відео. Функція під модуля полягає в тому що перевіряє парності та зміна розміру, підтртмка формату (.png, .jpg, .mp4); Підмодуль попередньої обробки: готує зображення для стереозбігу. Функція під модуля полягає в тому що в перертворення в сірошкальний,

нормалізація та фільтрація шуму; Підмодуль стереозбігу: створює карту глибини через StereoSGBM. Функція під модуля полягає в тому щоб налаштування параметрів, нормалізація диспартності; Підмодуль інтеграції з YOLO: передає зображення до YOLOv8, комбінує виявлення об’єктів з картою глибини. Функція під модуля полягає в тому що узгодження класів, відображення рамок і назв; Підмодуль обробки відеопотоків: обробляє кадри для динамічних карт глибини. Функція під модуля полягає в тому синхронізація потоків та запис відео в реальний час; Підмодуль виведення результатів: візуалізує дані. Функція під модуля полягає в тому що візуальне розташування вікон, збереження результату та переривання (‘q’).

Технологічний стек

- Бібліотека: OpenCV 4.5+.
- Алгоритм: StereoSGBM.
- Формати: RGB, сірошкальні, карти глибини (uint8).
- Оптимізація: CUDA (за наявності).
- Інтеграція: API YOLOv8.

Взаємодія з іншими модулями: З модулем введення даних: Отримує стереозображення або відеопотоки через GUI або камери, перевіряє їх валідність і передає до підмодуля попередньої обробки; З модулем розпізнавання об’єктів: Передає оригінальні кольорові зображення до YOLOv8, отримує координати та класи об’єктів і накладає їх на карту глибини; З модулем відстеження: Надає кадри та виявленні обєкти для аналізу траєкторій об’єктів у відеопотоці; З модулем бази даних: Передає оброблені карти глибини та зображення для збереження (image_to_bytea, video_to_bytea); З GUI: Отримує параметри стереозбігу (SettingsInterface) і передає результати для візуалізації (status_var, cv2.imshow).

Оптимізація та налаштування параметрів модулів забезпечує можливість регулювання основних параметрів стереозбігу через GUI, що дозволяє

адаптувати його до різних умов: minDisparity: Мінімальна диспаратність (зсув між зображеннями); numDisparities: Кількість рівнів диспаратності (кратне 16); blockSize: Розмір блоку для порівняння пікселів (непарне число, ≥1); P1, P2: Штрафи за малі та великі зміни диспаратності (обчислюються як P1_factor * 3 * blockSize²); disp12MaxDiff: Максимальна різниця при перевірці узгодженості; uniquenessRatio: Відсоток унікальності відповідності; speckleWindowSize, speckleRange: Параметри фільтрації шуму. Для підвищення швидкодії модуль оптимізовано: використання сірошкальних зображень зменшує обчислювальне навантаження; зміна розміру зображень (cv2.resize) знижує вимоги до пам'яті; можливість використання CUDA для StereoSGBM і YOLOv8 (залежить від апаратного забезпечення).

Переваги архітектури модуля та обмеження. Гнучкість: Налаштування параметрів дозволяє адаптувати модуль до різних сцен (освітлення, текстури, роздільна здатність). Модульність: Кожен підмодуль може бути вдосконалений незалежно (наприклад, заміна StereoSGBM на PSMNet). Швидкодія: Оптимізація для реального часу (≥10 FPS) завдяки CUDA і ефективним алгоритмам. Інтеграція: Безшовна взаємодія з YOLOv8 і базою даних забезпечує комплексну обробку даних. Надійність: Обробка помилок (try-except, перевірка валідності даних) гарантує стабільність. Тепер переходим до обмеження та виклики. Чутливість до калібрування: Некалібровані камери можуть призвести до неточних карт глибини. Обмеження StereoSGBM: Проблеми з низькотекстурними областями або оклюзіями знижують якість карти глибини. Обчислювальні вимоги: Обробка відео в реальному часі потребує GPU для забезпечення високої швидкості. Синхронізація відеопотоків: Розбіжності між лівим і правим потоком можуть ускладнити стереозбіг.

3.3 Інтеграція згорткових нейронних мереж у систему

Згорткові нейронні мережі (ЗНМ) є ключовим компонентом інформаційної системи для розпізнавання рухомих об'єктів і побудови тривимірних сцен на

основі стереозображень. Їхня інтеграція в систему забезпечує точне виявлення та класифікацію об’єктів у стереозображеннях і відеопотоках, а також дозволяє поєднувати результати виявлених об’єктів з картами глибини для створення повноцінної тривимірної моделі сцени. У даній системі використовується модель YOLOv8 (You Only Look Once, версія 8) від Ultralytics, яка є однією з найефективніших ЗНМ для задач розпізнавання об’єктів у реальному часі. Цей розділ описує архітектуру інтеграції ЗНМ, їхню взаємодію з іншими модулями системи, технічну реалізацію, оптимізацію та обмеження. Функціональні завдання: Розпізнавання об’єктів: виявлення та класифікація об’єктів у стереозображеннях/відеокадрах. Інтеграція з картами глибини: узгодження детекцій між лівим і правим зображеннями для накладання на карту глибини; візуалізація об’єктів у 3D-просторі. Підготовка для трекінгу: надання координат і класів для аналізу руху. Збереження результатів: Передача даних до бази даних.

Архітектура інтеграції ЗНМ починається з ініціалізація YOLOv8: завантаження моделі YOLOv8n з бібліотеки Ultralytics і файлу yolov8n.pt. Функція налаштування розміру зображень та порогу впевненості. Обробка зображень: виявлення об’єктів на лівому та правому зображенні. Функція отримання координат x і y, класів (cls) та ймовірностей (conf). Узгодження розпізнавання: обчислення середнього значення координат між стереопарою (x1 = (xl1 + xr1) // 2). Функція перевірки класів і відображення на карті глибини. Інтеграція з картою глибини: накладання обмежувальних рамок на карту глибини. Функція вибіркового відображення із зеленим кольором і заданим шрифтом. Підготовка до відстеження об’єктів: передача координат до модуля відстеження. Функції для структурування даних траєкторій. Збереження результатів: запис зображень і відео до PostgreSQL. Функції транзакцій для забезпечення цілісності даних.

Технологічний стек

- Модель: YOLOv8n (COCO).
- Бібліотека: Ultralytics YOLO, PyTorch.

- Інтеграція: OpenCV (cv2) для візуалізації.
- База даних: PostgreSQL (psycopg2).
- Оптимізація: CUDA (NVIDIA GPU).

Взаємодія з іншими модулями: З модулем стереозору та обробки зображень: Отримує оригінальні кольорові зображення (left_frame, right_frame) для виявлення об’єктів. Передає результати виявлених об’єктів для накладання на карту глибини (output_map_with_boxes). Використовує оброблені сірошкальні зображення для узгодження з картою глибини; З модулем відстеження: Надає координати та класи об’єктів для аналізу траєкторій між кадрами. Передає дані для інтеграції з алгоритмами трекінгу (наприклад, DeepSORT); З модулем бази даних: Передає зображення та відео з виявлення об’єктів для збереження в PostgreSQL. Забезпечує зв’язок між таблицями (images, videos, depth_maps) для збереження результатів; З GUI: Отримує налаштування відображення детекцій (check_left, check_right, track_left, track_right, track_depth). Передає результати для візуалізації через cv2.imshow і оновлення статусу (status_var).

Оптимізація та параметри для забезпечення швидкодії та точності інтеграція ЗНМ оптимізована наступним чином: Використання GPU: YOLOv8 підтримує CUDA для прискорення виявлення (≥10 FPS на NVIDIA GPU); Легка модель: YOLOv8n обрано як компроміс між швидкістю та точністю (mAP ~0.8 на COCO); Обмеження детекцій: Поріг довіри (confidence threshold) налаштовується для зменшення хибнопозитивних результатів; Узгодження координат: Усереднення координат між лівим і правим зображеннями зменшує похибки, спричинені стереодиспаратністю; Паралельна обробка: Окрема обробка лівого та правого зображень дозволяє паралелізувати обчислення на багатоядерних процесорах.

Користувач може впливати на роботу ЗНМ через GUI, вмикаючи або вимикаючи відображення детекцій на лівому, правому зображенні або карті

глибини. У майбутніх оновленнях можливе додавання налаштувань порогу довіри та розміру вхідного зображення через SettingsInterface.

2.4 Взаємодія компонентів: камера, обробка, розпізнавання, візуалізація

Інформаційна система для розпізнавання рухомих об’єктів і побудови тривимірних сцен на основі стереозображень спирається на тісну взаємодію між ключовими компонентами: камерою (введення даних), обробкою зображень (стереозбіг), розпізнаванням об’єктів (згорткові нейронні мережі) та візуалізацією (графічний інтерфейс і відображення результатів). Ця взаємодія забезпечує безперервний потік даних від захоплення зображень до створення тривимірних моделей і їхньої презентації користувачу. Архітектура системи побудована так, щоб оптимізувати обмін даними, забезпечити швидкодію (≥10 FPS для реального часу), точність і зручність використання. У цьому розділі описано, як компоненти взаємодіють між собою, їхні ролі, технічну реалізацію та особливості інтеграції.

Компоненти та їхні ролі. Камера: роль захоплює стереозображення та відеопотоків з двох камер або приймає файли. Функція синхронне введення, перевірка валідності, передача до обробки; Обробка зображень (стереозбіг): створює карти глибини через попередню обробку та StereoSGBM. Функція нормалізація, обчислення диспаратності, підготовка карти глибини; Розпізнавання об’єктів (ЗНМ): виявляє та розподіляє об’єкти за допомогою YOLOv8. Функція надає обмежувальні рамки, класи, узгоджує розпізнає для карти глибини; Візуалізація (GUI, відображення): координує взаємодію, відображає результати, управляє процесами. Функція Інтерфейс для налаштувань, перегляду, збереження даних.

Архітектура взаємодії: захоплення даних (камера → обробка); обробка зображень (обробка → розпізнавання); розпізнавання об’єктів (розпізнавання → обробка, візуалізація); візуалізація (візуалізація →

всі).Захоплення даних (камера → обробка): Дані надходять із камери або файлів у вигляді стереопари, після чого перевіряється їхня коректність. Здійснюється синхронізація зображень чи відео, зміна розміру та збереження. Обробка зображень (обробка → розпізнавання): Зображення конвертуються в сірошкальний формат, нормалізуються, виконується стереозбір, а карта глибини передається до візуалізації. Кольорові зображення надсилаються до YOLOv8. Реалізовано налаштування через графічний інтерфейс, нормалізацію диспаратності та кольорове кодування. Розпізнавання об’єктів (розпізнавання → обробка, візуалізація): YOLOv8 обробляє зображення, узгоджує виявлені об’єкти та накладає рамки на карту глибини. Використовуються зелені рамки, заданий шрифт. Можна вмикати або вимикати відображення з YOLOv8. Візуалізація (візуалізація → всі): Відображення результатів відбувається через cv2.imshow (активується через GUI в MainInterface). Дані зберігаються в PostgreSQL. Реалізовано позиціонування вікон, переривання через клавішу (‘q’, cv2.waitKey) та підказки (ToolTip).Потік даних між компонентами: Камера → Обробка: стереопара (left_frame, right_frame). Обробка → Розпізнавання: кольорові зображення до YOLOv8, карта глибини до візуалізації. Розпізнавання → Обробка: візуалізація: обмежувальні рамки на карту глибини (output_map_with_boxes). Візуалізація → Всі: запуск обробки (self.process_image()), відображення (cv2.imshow), збереження (db_connection).

2.5 Інтерфейс користувача: візуалізація сцени та параметри керування

Інтерфейс користувача (GUI) є ключовим компонентом інформаційної системи для розпізнавання рухомих об’єктів і побудови тривимірних сцен на основі стереозображень. Він забезпечує зручну взаємодію користувача з системою, дозволяючи завантажувати дані, налаштовувати параметри обробки,

запускати процеси стереозбігу та розпізнавання об’єктів, а також візуалізувати результати, такі як карти глибини, виявлення об’єктів і тривимірні сцени. Графічний інтерфейс спроєктовано з акцентом на інтуїтивність, гнучкість і функціональність, щоб відповідати потребам як технічних спеціалістів, так і користувачів без глибоких знань у комп’ютерному зорі. У цьому розділі описано архітектуру інтерфейсу, його функціональні можливості, технічну реалізацію, взаємодію з іншими компонентами системи та особливості управління параметрами.

Функціональні завдання інтерфейса користувача: завантаження даних, налаштування параметрів, управління процесами, візуалізація, інформування.

Тепер більш детально розгляним функціональні завдання. Завантаження даних має такі завдання це вибір стереозображень, відео або веб-камер та Перевірка валідності даних. Налаштування параметрів полягає на налаштуванні стереозбігу (minDisparity, numDisparities та інші), та вибір режимів відображення карти глибин (чорно-білий та кольоровий). Управління процесами відовідають за запуск обробки та переривання та очищення пам’яті. Візуалізація це відображення зображення, карт глибини та виявлення об’єктів (OpenCV). І останнє завдання це інформування має такі завдання це статус операції та підказки ToolTip.

GUI побудовано на tkinter із двома класами: MainInterface (головне вікно) і SettingsInterface (налаштування). Інтеграція з OpenCV забезпечує візуалізацію.

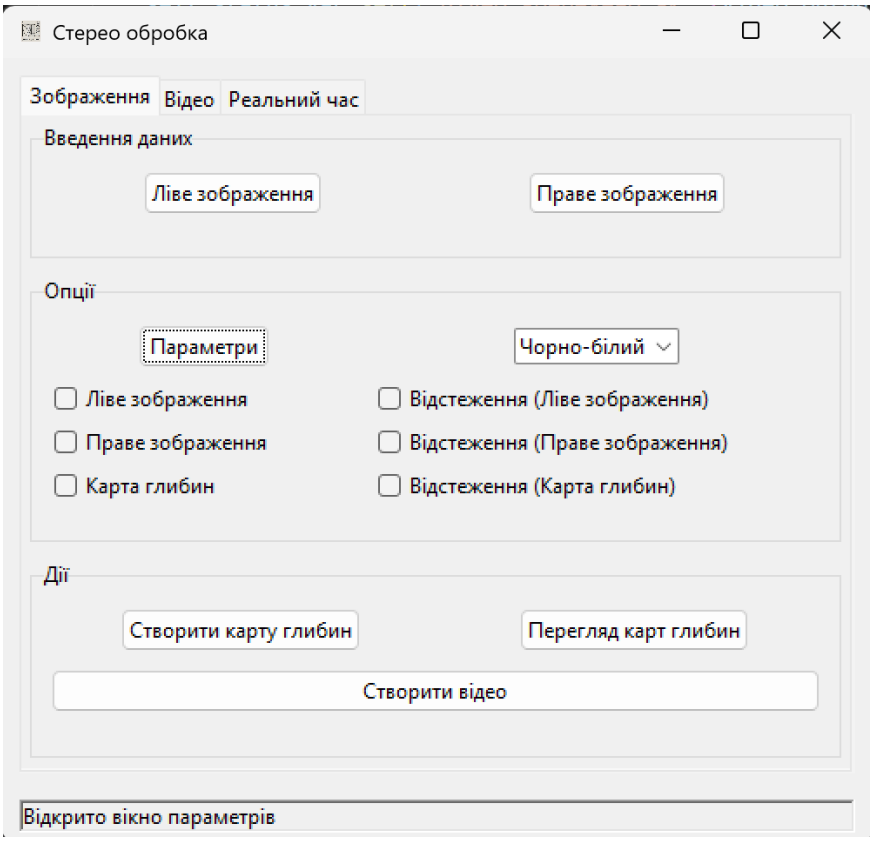


Рис. 4 GUI побудований на tkinter клас MainInterface (головне вікно). Відповідає за запис базу даних, створення карти глибин та перегляд карти глибин.

MainInterface має такі компоненти: вкладки, кнопки, випадаючі списки, чекбокси, панель статусу.

MainInterface містить кілька компонентів, зокрема ttk.Notebook із трьома вкладками — Image (Зображення), Video (Відео) та Camera (реальний час) — для навігації між різними функціями. Інтерфейс містить вкладку для роботи із зображеннями, де є кнопки для вибору лівого та правого зображення. Користувач може відкрити папку із зображеннями за допомогою filedialog, при цьому з папки доступні всі типи файлів (.png, .jpg, .jpeg, .bmp). У продовженні інтерфейсу є випадаючий список (combobox), який дозволяє користувачу обрати колір карти глибини — чорно-білий або кольоровий. Далі розташовані два ряди чекбоксів: перший ряд відповідає за перегляд і створення карти глибини в реальному часі, а другий ряд виконує схожу функцію, але з підключенням YOLOv8 для виявлення рухомих об’єктів. Після цього в інтерфейсі розміщені кнопки: перша запускає створення карти глибини, друга — її перегляд, третя — створення відео із зображень. Також є панель статусу зі status_var, яка відображає оновлення та

стан системи в реальному часі. Для вкладки відео інтерфейс подібний до вкладки зображень, а вкладка реального часу відрізняється наявністю кнопки пошуку підключених до комп'ютера камер і випадаючого списку з їхніми індексами. Детальніше інтерфейс описано в додатку В. Функції MainInterface відповідають за запуск обробки, оновлення статусу та відображення підказок ToolTip.

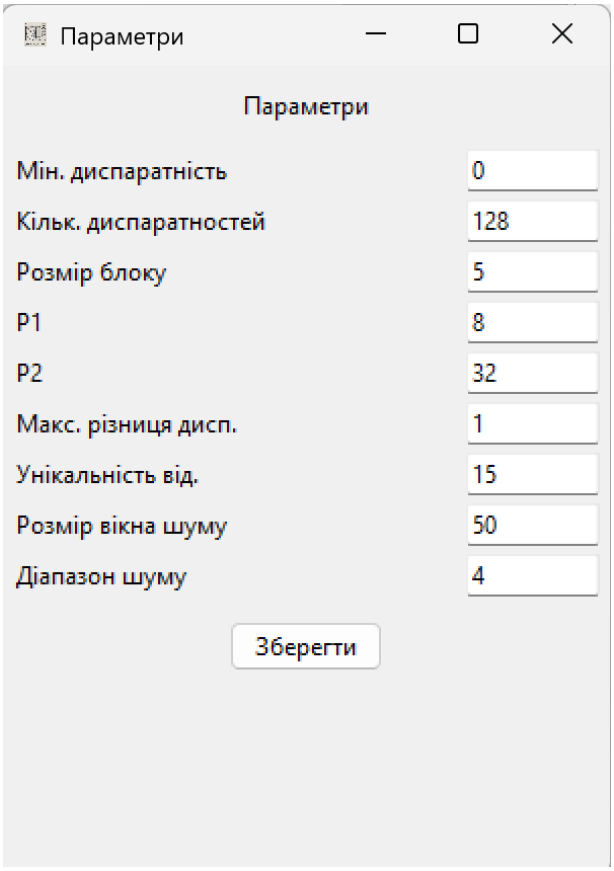


Рис. 5 GUI побудований на tkinter клас SettingsInterface (налаштування). Віповідає за налаштування параметрів StereoSGBM.

SettingsInterface має такі компоненти: поля, кнопки. SettingsInterface містить кілька компонентів, з 9 текстовими полям (minDisparity, numDisparities, blockSize, P1_factor, P2_factor, disp12MaxDiff, uniquenessRatio, speckleWindowSize, speckleRange) та поля для введення даних навпроти текстових полях та кнопкою для зберігання даних. Спочатку користувач вводить в поля параметри StereoSGBM після того зберігає в базу даних.

Функції SettingsInterface це валідація параметрів (numDisparities % 16 == 0), та Оновлення StereoSGBM, підкази ToolTip. Система обробки стереозображень включає кілька етапів: камера-вибір файлів (self.left та

right_image_path) або камер (передача дані до cv2.VideoCapture); обробка зображень- використовоючи параметри стереозбігу в (SettingsInterface) та запуск стереозбігу і отримання карти глибин; розпізнавання обє’ктів за допомогою (YOLOv8)- аналіз кольорових зображеннь, відображення рухомих обє’ктів; база даних- збереігання результатів та перегляд даних.

Технічна реалізація інтерфейсу. Бібліотека GUI: tkinter для створення головного вікна (MainInterface) і вікна налаштувань (SettingsInterface). Відображення: OpenCV (cv2.imshow, cv2.moveWindow) для візуалізації зображень і карт глибини. Формати даних: Зображення: numpy.ndarray (BGR для OpenCV, uint8 для карт глибини). Відео: cv2.VideoWriter mp4v для збереження результатів. База даних: BYTEA для зберігання зображень і відео в PostgreSQL. Керування параметрами: Параметри стереозбігу зберігаються як атрибути SettingsInterface і передаються до cv2.StereoSGBM_create. Режими відображення (check_left, track_depth тощо) реалізовано через ttk.Checkbutton із прив’язкою до BooleanVar. Оптимізація: Використання легкої моделі YOLOv8n і зменшення роздільної здатності зображень (cv2.resize до 640x480) для швидкої обробки. Асинхронне оновлення статусу (self.status_var) для уникнення затримок у GUI.

Особливості інтерфейсу: Інтуїтивність: Логічно організовані вкладки (Image, Video, Camera) спрощують вибір режиму роботи. Підказки (ToolTip) пояснюють функції кнопок і полів (наприклад, "Select left image file"); Гнучкість: Користувач може налаштувати параметри стереозбігу для різних умов (освітлення, текстура). Чекбокси дозволяють вибирати, які дані відображати (зображення, виявлення, карта глибини);Інтерактивність: Кнопка Stop дозволяє переривати обробку, а Clear очищає пам’ять. Перегляд збережених карт глибини через View Depth Maps із навігацією по базі даних; Інформативність: Панель статусу показує прогрес ("Processing image...", "Completed") або помилки ("No cameras detected"). Візуалізація через OpenCV забезпечує чітке відображення результатів.

Переваги інтерфейсу: Зручність використання: Інтуїтивний дизайн дозволяє користувачам без технічної підготовки працювати з системою. Гнучке керування: Налаштування параметрів і режимів відображення адаптують систему до різних сценаріїв. Інтеграція: GUI координує всі модулі, забезпечуючи безперервний потік даних. Візуалізація: Поєднання tkinter і OpenCV забезпечує чітке відображення складних даних (карти глибини, виявлення об’єктів). Надійність: Обробка помилок (try-except) і перевірка валідності даних запобігають збоями.

3. Розробка інформаційного та програмного забезпечення

3.1 Система управління інформаційною базою

Система управління інформаційною базою (СУБД) є ключовим компонентом інформаційної системи для розпізнавання рухомих об’єктів і побудови тривимірних сцен на основі стереозображень. Вона забезпечує структуроване зберігання, швидкий доступ і надійне управління даними, такими як стереозображення, відеопотоки, карти глибини, параметри обробки та звіти. Для реалізації СУБД обрано реляційну базу даних PostgreSQL, яка підтримує роботу з великими обсягами даних, включаючи бінарні об’єкти (BYTEA), і забезпечує високу надійність завдяки транзакціям і зовнішнім ключам. У цьому розділі описано структуру нової бази даних, її таблиці, зв’язки, обмеження та принципи взаємодії з програмним забезпеченням.

Огляд структури бази даних

База даних спроєктована для зберігання всіх типів даних, що використовуються системою: стереозображень, відеопотоків, даних із камер, карт глибини, параметрів стереозбігу та звітів про обробку. Вона складається з п’яти основних таблиць: cameras, images, videos, depth_maps і stereo_params. Кожна таблиця має чітко визначену роль, а їхня структура забезпечує цілісність даних через зовнішні ключі та обмеження. Нижче наведено детальний опис таблиць, їхніх полів, зв’язків і особливостей.

Структура створення таблиць розглянимо на прикладі таблиці images.

```
CREATE TABLE images (
    id SERIAL PRIMARY KEY,
    left_image BYTEA NOT NULL,
    right_image BYTEA NOT NULL,
    upload_time TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

Призначення: Зберігає пари стереозображень (ліве та праве) для статичної обробки.

Поля: id (SERIAL PRIMARY KEY): Унікальний ідентифікатор запису; left_image (BYTEA NOT NULL): Бінарні дані потоку з лівої зображення; right_ image (BYTEA NOT NULL): Бінарні дані потоку з правої зображення; upload_time (TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP): Час створення запису. Особливості: Формат BYTEA підтримує зображення у форматах .png, .jpg, .jpeg та інші. Використовується для зберігання оригінальних зображень перед обробкою стереозбігом. Більш детально можна оглянути код створення бази даних в додатку А.

Взаємодія з програмним забезпеченням. СУБД інтегрується з програмним забезпеченням через бібліотеку psycopg2, яка забезпечує зв’язок між Python і PostgreSQL. Основні аспекти взаємодії:

Підключення до бази даних: використовується об’єкт підключення (psycopg2.connect) із параметрами: host, database, user, password. Приклад self.db_connection = psycopg2.connect(host='localhost', database='StereoAI', user='Admin', password='admin'). Використовується курсор (self.db_connection.cursor()) для виконання SQL-запитів; Після підключення до бази даних йде зберігання даних: зображення, відео та карти глибини конвертуються в бінарний формат BYTEA за допомогою функцій, таких як image_to_bytea та video_to_bytea. Приклад SQL-запиту для збереження зображень: INSERT INTO images (left_image, right_image) VALUES (%s, %s) RETURNING id;.

Подібні запити для videos, cameras і depth_maps; Користувач має змогу змінити параметри стереозбігу, які зберігаються або оновлюються за допомогою SQL-запитів

INSERT/UPDATE: INSERT INTO stereo_params (param_name, param_value) VALUES (%s, %s) та ON CONFLICT (param_name)

DO UPDATE SET param_value = EXCLUDED.param_value, updated_at = CURRENT_TIMESTAMP; це забезпечує актуальність налаштувань для всіх сеансів обробки; Після того як зберігає дані в базу даних ми можемо отримати дані: Для перегляду збережених карт глибини використовується запит до depth_maps із JOIN до images, videos або cameras:

```
SELECT id, depth_map, created_at FROM depth_maps WHERE left_image_id IS NOT NULL;
```

Дані конвертуються з BYTEA назад у зображення за допомогою cv2.imdecode. Транзакції: Використовуються для забезпечення цілісності даних: self.db_connection.commit() після успішного виконання запиту. self.db_connection.rollback() у разі помилки (try-except).

3.2 Розробка інформаційної бази

Інформаційна база є основою системи управління даними для розпізнавання рухомих об’єктів і побудови тривимірних сцен на основі стереозображень. Вона забезпечує структуроване зберігання стереозображень, відеопотоків, даних із камер, карт глибини, параметрів стереозбігу, а також потенційних даних про траєкторії об’єктів. Розробка інформаційної бази базується на реляційній моделі з використанням PostgreSQL, що гарантує надійність, цілісність і ефективний доступ до даних. У цьому розділі детально описано процес створення інформаційної бази, структуру таблиць, їхні зв’язки, оптимізацію, а також принципи інтеграції з програмним забезпеченням системи.

Метою розробки інформаційної бази представлено виконання завдань: Зберігання вхідних даних: Збереження пар стереозображень, відеопотоків і даних із веб-камер у бінарному форматі. Зберігання вхідних даних: Збереження пар стереозображень, відеопотоків і даних із веб-камер у бінарному форматі. Зберігання результатів обробки: Збереження карт глибини, створених за допомогою алгоритму StereoSGBM, разом із метаданими обробки (час, звіти). Управління параметрами: Зберігання налаштувань стереозбігу для забезпечення консистентності обробки. Підтримка аналізу: Надання можливості перегляду збережених даних (зображень, відео, карт глибини) через графічний інтерфейс. Гнучкість і масштабованість: Забезпечення структури, яка дозволяє додавати нові типи даних, наприклад, траєкторії об’єктів.

Структура інформаційної бази складається з п’яти основних таблиць: cameras, images, videos, depth_maps і stereo_params. Ці таблиці пов’язані через

зовнішні ключі, а їхня структура оптимізована для ефективного зберігання та доступу до даних.

Для наочної ілюстрації структури інформаційної бази розроблено ER-діаграму, яка відображає п’ять основних сутностей: `cameras`, `images`, `videos`, `depth_maps` і `stereo_params`. Кожна сутність містить набір атрибутів, а зв’язки між ними реалізовано через зовнішні ключі. Наприклад, сутність `depth_maps` пов’язана з `images` через атрибути `left_image_id` і `right_image_id`, що дозволяє відстежувати стереозображення, використані для створення карт глибини. На рисунку 6 представлено ER-діаграму інформаційної бази.

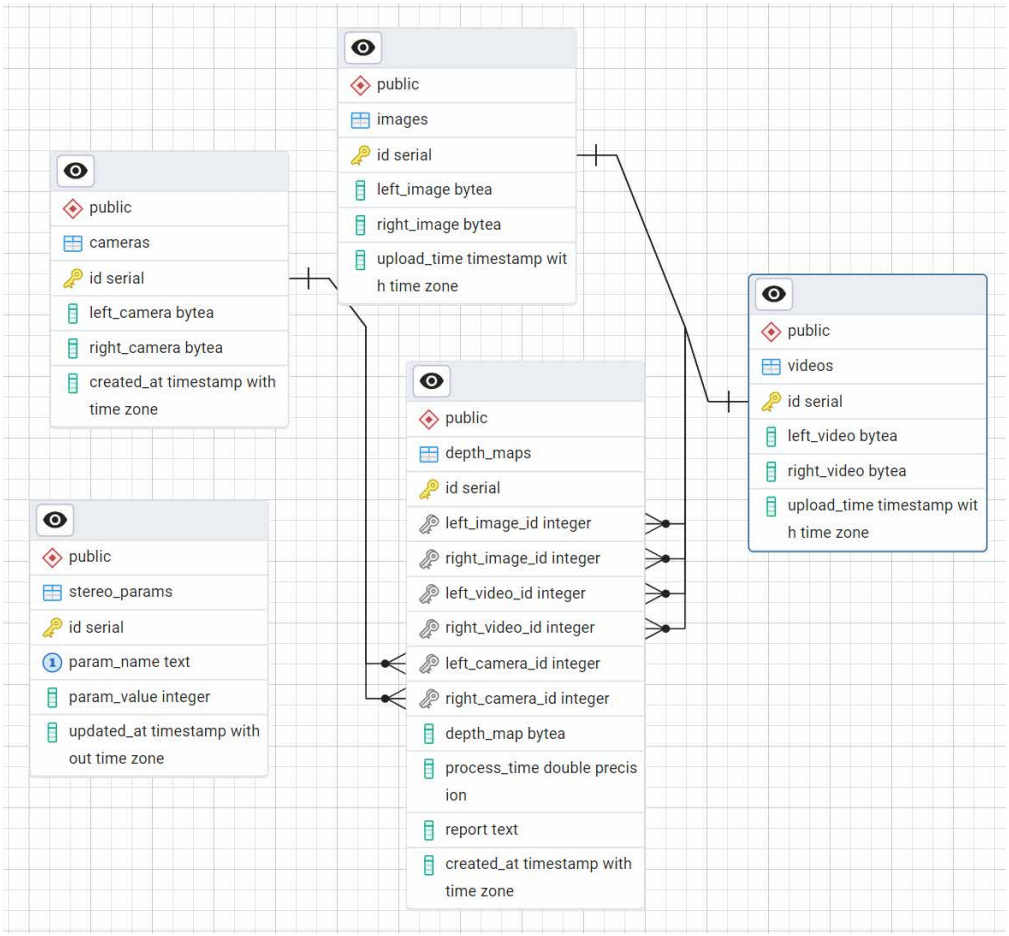


Рис.6 ER-діаграму інформаційної бази.

Оптимізація структури бази даних для забезпечення ефективності та масштабованість інформаційної бази застосовано наступні заходи: індексація- первинні ключі (id) автоматично створюється індекси для швидкого пошуку, унікальний індекс param_name у stereo_params прискорює доступ до параметрів; зовнішні ключі та обмеження- зовнішні ключі в depth_maps забезпечують

цілісність зв’язків із images, videos і camera, Обмеження CHECK у depth_maps запобігає некоректним комбінаціям джерел даних; стиснення даних- зображення та карти глибин зберігаються у форматі .png для зменшення обсягу, відео стискається перед збереженням у BYTEA; транзакції- Використання транзакцій (commit/rollback) забезпечує атомарність операцій і запобігає втраті даних при помилках; розподіл даних- Окремі таблиці для різних типів даних (images, videos, cameras) спрощують управління та зменшують фрагментацію.

Інформаційна база інтегрується з програмним забезпеченням через бібліотеку psycopg2, яка забезпечує взаємодію між Python і PostgreSQL. Основні аспекти інтеграції: ініціалізація бази даних- SQL-скрипт для створення таблиць виконується при першому запуску системи: CREATE TABLE cameras (...) скрипт перевіряє наявність таблиць (CREATE TABLE IF NOT EXISTS) для уникнення помилок; збереження даних- зображення, відео та карти глибини конвертуються в бінарний формат BYTEA:

```
def image_to_bytea(image):
buffer = cv2.imencode('.png', image)
return psycopg2.Binary(buffer.tobytes())
```

SQL-запити для вставки даних:

```
INSERT INTO images (left_image, right_image) VALUES (%s, %s) RETURNING id;
INSERT INTO depth_maps (left_image_id, right_image_id, depth_map, process_time, report) VALUES (%s, %s, %s, %s, %s);
```

Оновлення параметрів- параметри стереозбігу оновлюються через INSERT з ON CONFLICT:

```
INSERT INTO stereo_params (param_name, param_value) VALUES (%s, %s)
    ON CONFLICT (param_name) DO UPDATE SET param_value = EXCLUDED.param_value, updated_at = CURRENT_TIMESTAMP;
```

Отримання даних для перегляду карт глибини використовуються запити з JOIN:

```
SELECT d.id, d.depth_map, d.created_at, i.left_image, i.right_image
FROM depth_maps d
JOIN images i ON d.left_image_id = i.id AND d.right_image_id = i.id
WHERE d.left_image_id IS NOT NULL;
```

Дані BYTEA конвертуються назад у зображення:

```
depth_map = cv2.imdecode(np.frombuffer(row['depth_map'], np.uint8), cv2.IMREAD_UNCHANGED)
```

Обробка помилок- використання try-except для обробки виключень (наприклад, psycopg2.DatabaseError), відкат транзакцій (self.db_connection.rollback()) при помилках.

Переваги інформаційної бази: Цілісність: Зовнішні ключі та обмеження CHECK забезпечують коректність даних; Гнучкість: Структура підтримує різні типи джерел (зображення, відео, камери) в одній таблиці depth_maps; Ефективність: Індекси та транзакції прискорюють доступ і забезпечують надійність; Інтеграція: Проста взаємодія з Python через psycopg2 полегшує розробку; Документація: Поля report і created_at дозволяють відстежувати процеси обробки.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Розробка прикладного програмного забезпечення для системи розпізнавання рухомих об’єктів і побудови тривимірних сцен на основі стереозображень вимагає ретельного вибору інструментів, які забезпечують високу продуктивність, гнучкість, надійність і зручність інтеграції. Інструментарій має відповідати вимогам до обробки великих обсягів даних у реальному часі, реалізації алгоритмів комп’ютерного зору, управління базами даних і створення зручного графічного інтерфейсу користувача (GUI). У цьому розділі описано обраний інструментарій, його обґрунтування, порівняння з альтернативами та принципи використання в контексті системи.

Інструменти для розробки прикладного програмного забезпечення обрано з урахуванням наступних вимог: продуктивність-підтримка обробки зображень і відео в реальному часі (≥10 FPS) з використанням апаратного прискорення (GPU); функціональність- наявність бібліотек для реалізації стереозбігу, розпізнавання об’єктів за допомогою згорткових нейронних мереж (ЗНМ) і управління базами даних; гнучкість- Можливість налаштування алгоритмів і адаптації до різних сценаріїв (освітлення, роздільна здатність, типи камер). простота інтеграції- легке

поєднання бібліотек для обробки зображень, машинного навчання, GUI і баз даних; доступність- використання інструментів із відкритим кодом або безкоштовних ліцензій для зниження витрат. документація та підтримка- наявність активної спільноти та якісної документації для полегшення розробки; кросплатформність- сумісність із різними операційними системами (Windows, Linux) для ширшої застосовності.

На основі зазначених вимог обрано наступний набір інструментів для створення прикладного програмного забезпечення: Мова програмування: Python. Обґрунтування: Python є високорівневою мовою з простим синтаксисом, що прискорює розробку. Має широкий набір бібліотек для комп'ютерного зору (OpenCV), машинного навчання (PyTorch, Ultralytics YOLO), управління базами даних (psycopg2) і створення GUI (tkinter). Підтримує кросплатформну розробку, що дозволяє запускати систему на Windows і Linux. Активна спільнота та багата документація спрощують вирішення технічних питань. Використання: Реалізація основної логіки системи, включаючи обробку зображень, інтеграцію ЗНМ, взаємодію з базою даних і GUI. Версія: Python 3.8+ для сумісності з сучасними бібліотеками. Бібліотека для комп'ютерного зору: OpenCV. Обґрунтування: OpenCV (Open Source Computer Vision Library) є стандартом для обробки зображень і відео, включаючи стереозбіг (StereoSGBM, StereoBM), фільтрацію, нормалізацію та візуалізацію. Підтримує апаратне прискорення через CUDA для підвищення продуктивності на GPU. Має багатий набір функцій для роботи з камерами (cv2.VideoCapture), створення вікон (cv2.imshow) і збереження результатів (cv2.imwrite, cv2.VideoWriter). Відкритий код і активна спільнота забезпечують доступність і підтримку. Використання: Захоплення даних із веб-камер або файлів (cv2.VideoCapture). Попередня обробка зображень. Виконання стереозбігу . Візуалізація карт глибини та виявлення об'єктів. Бібліотека для розпізнавання об'єктів: Ultralytics YOLO. Обґрунтування: Ultralytics YOLO (версія 8, YOLOv8) є однією з

найефективніших моделей ЗНМ для виявлення об’єктів у реальному часі (mAP ~0.8 на COCO, ≥10 FPS на GPU). Попередньо навчені моделі (yolov8n.pt) підтримують 80 класів із набору даних COCO, що підходить для більшості сценаріїв. Інтеграція з PyTorch забезпечує гнучкість і підтримку GPU (CUDA). Простота використання через високорівневий API (YOLO.predict) прискорює розробку. Використання: Детекція об’єктів у стереозображеннях і відеокадрах (self.yolo_model(left_frame)). Узгодження детекцій між лівим і правим зображеннями для накладання на карту глибини. Підготовка даних для потенційного трекінгу (координати обмежувальні рамків). Фреймворк для машинного навчання: PyTorch. Обґрунтування: PyTorch є основою для YOLOv8 і забезпечує гнучкість для роботи з нейронними мережами. Підтримує апаратне прискорення (CUDA) для швидкої обробки великих обсягів даних. Має активну спільноту та багату документацію, що полегшує налагодження моделей. Динамічна обчислювальна графік дозволяє швидко модифікувати архітектуру ЗНМ. Використання: Виконання YOLOv8 для виявлення об’єктів. Потенційна можливість fine-tuning моделі на спеціалізованих наборах даних. СУБД: PostgreSQL. Обґрунтування: PostgreSQL є потужною реляційною базою даних із відкритим кодом, яка підтримує бінарні дані (BYTEA) для зберігання зображень, відео та карт глибини. Забезпечує цілісність даних через зовнішні ключі, обмеження (CHECK) і транзакції. Має високу продуктивність для роботи з великими обсягами даних і підтримує індексацію. Інтеграція з Python через бібліотеку psycopg2 спрощує взаємодію. Використання: Зберігання стереозображень (images), відео (videos), даних із камер (cameras), карт глибини (depth_maps) і параметрів стереозбігу (stereo_params). Виконання SQL-запитів для вставки, оновлення та отримання даних. Забезпечення транзакцій для надійного збереження результатів. Бібліотека для GUI: tkinter. Обґрунтування: tkinter є стандартною бібліотекою Python для створення графічних інтерфейсів, що забезпечує

простоту розробки та кросплатформність. Підтримує створення вкладок (ttk.Notebook), кнопок, чекбоксів, випадаючих списків і текстових полів для зручного керування. Легка інтеграція з OpenCV для відображення зображень і карт глибини (cv2.imshow). Не потребує додаткових залежностей, що знижує складність встановлення. Використання: Реалізація головного вікна (MainInterface) із вкладками для роботи з зображеннями, відео та камерами. Створення вікна налаштувань (SettingsInterface) для параметрів стереозбігу. Відображення статусу обробки (status_var) і підказок (ToolTip). Бібліотека для взаємодії з базою даних: psycopg2. Обґрунтування: psycopg2 є найпопулярнішою бібліотекою для роботи з PostgreSQL у Python, забезпечуючи швидкий і надійний доступ до бази даних. Підтримує транзакції, бінарні дані (BYTEA) і обробку помилок. Має багату документацію та активну підтримку спільноти. Використання: Підключення до PostgreSQL (psycopg2.connect). Виконання SQL-запитів для збереження (INSERT), оновлення (UPDATE) і отримання даних (SELECT). Конвертація зображень і відео в BYTEA (psycopg2.Binary). Інструмент для апаратного прискорення: NVIDIA CUDA. Обґрунтування: CUDA забезпечує апаратне прискорення для OpenCV (StereoSGBM) і PyTorch (YOLOv8), значно підвищуючи швидкість обробки (≥10 FPS). Підтримується більшістю сучасних GPU від NVIDIA, що робить його доступним для розробки. Сумісність із Python через бібліотеки, такі як pycuda і cudnn. Використання:

Прискорення стереозбігу в OpenCV (cv2.cuda.StereoSGBM, якщо доступно). Виконання інференсу YOLOv8 на GPU для реального часу.

Таблиця 1

Інструмент	Переваги	Недоліки	Чому обрано?
Python	Простота, багатство бібліотек, кросплатформність, активна спільнота	Менша швидкість порівняно з C++	Швидкість розробки, підтримка комп’ютерного зору і ЗНМ
OpenCV	Широкий функціонал, підтримка GPU, відкритий код	Обмежена точність стереозбігу в складних сценах	Стандарт для комп’ютерного зору, легка інтеграція
YOLOv8	Висока швидкість і точність, простота API, підтримка GPU	Потреба в GPU для реального часу	Оптимальний баланс швидкості та точності для реального часу
PyTorch	Гнучкість, підтримка GPU, динамічна графік, активна спільнота	Вищі вимоги до пам’яті порівняно з ONNX	Інтеграція з YOLOv8, гнучкість для ЗНМ
PostgreSQL	Підтримка BYTEA, зовнішні ключі, транзакції, масштабованість	Складніше налаштування порівняно з SQLite	Надійність і підтримка великих бінарних даних
tkinter	Простота, вбудованість у Python, кросплатформність	Обмежений функціонал для складних UI	Достатній для базового GUI, легка інтеграція з OpenCV

psycopg2	Швидкість, підтримка BYTEA, транзакції, документація	Менш зручний для ORM порівняно з SQLAlchemy	Простота і ефективність для роботи з PostgreSQL
CUDA	Висока продуктивність, сумісність із OpenCV і PyTorch	Обмежена підтримка не-NVIDIA GPU	Прискорення обробки для реального часу

Модульність:кожен інструмент відповідає за певний аспект системи: OpenCV для обробки зображень, YOLOv8 для розпізнавання, PostgreSQL для даних, tkinter для GUI, модульна структура дозволяє легко замінювати компоненти (наприклад, YOLOv8 на Detectron2); Оптимізація продуктивності: використання CUDA для прискорення StereoSGBM і YOLOv8, зменшення роздільної здатності зображень (cv2.resize до 640x480) для зниження обчислювального навантаження; Інтеграція: Python виступає як основа для поєднання всіх бібліотек, psycopg2 забезпечує зв’язок між PostgreSQL і Python, дозволяючи зберігати результати обробки (image_to_bytea, video_to_bytea), tkinter координує взаємодію користувача з OpenCV (cv2.imshow) і YOLOv8; Надійність: обробка помилок (try-except) у Python для уникнення збоїв, транзакції в PostgreSQL (commit/rollback) для забезпечення цілісності даних; Зручність: GUI на основі tkinter спрощує вибір даних, налаштування параметрів і перегляд результатів, підказки (ToolTip) і панель статусу (status_var) полегшують взаємодію.

3.4 Алгоритмізація та програмування програмних модулів

Алгоритмізація та програмування програмних модулів є ключовим етапом розробки інформаційної системи для розпізнавання рухомих об’єктів і побудови тривимірних сцен на основі стереозображень. Система включає кілька модулів, кожен із яких реалізує набір алгоритмів для

обробки даних, стереозбігу, розпізнавання об’єктів, управління базою даних і візуалізації. У цьому розділі описано основні алгоритми, їхню роль у системі, принципи роботи, програмну реалізацію в Python із використанням бібліотек OpenCV, Ultralytics YOLOv8 і psycopg2, а також фрагменти коду для ключових модулів. Основна мета – забезпечити швидкодію (≥10 FPS), точність і модульність системи.

Система використовує набір алгоритмів, які виконують спеціалізовані завдання, від захоплення даних до створення тривимірних сцен. Нижче описано ключові алгоритми, їхню функціональність і місце в архітектурі.

Алгоритм стереозбігу (StereoSGBM). Роль: Обчислює диспаратність між лівим і правим стереозображеннями для створення карти глибини, яка відображає тривимірну структуру сцени. Принцип роботи: Алгоритм Semi-Global Block Matching (SGBM) порівнює блоки пікселів між лівим і правим зображеннями, мінімізуючи функцію вартості, що враховує відповідність пікселів і гладкість диспаратності. Використовує сірошкальні зображення для зменшення обчислювального навантаження. Результатом є карта диспаратності, яка нормалізується для створення карти глибини. Реалізація: використовується бібліотека OpenCV (cv2.StereoSGBM_create); налаштування параметрів через GUI (SettingsInterface); нормалізація та візуалізація карти глибини (чорно-білий або кольоровий режим).

Алгоритм розпізнавання об’єктів (YOLOv8). Роль: Виявляє та класифікує об’єкти в стереозображеннях або відеокадрах, надаючи координати обмежувальні рамки і назви класів. Принцип роботи: YOLOv8 (You Only Look Once, версія 8) є одностадійною згортковою нейронною мережею, яка обробляє зображення за один прохід, передбачаючи обмежувальні рамки, класи та ймовірності. Використує попередньо навчену модель (yolov8n.pt) на наборі даних COCO з 80 класами. Результати узгоджуються між лівим і правим зображеннями для накладання на карту глибини. Реалізація: використовується бібліотека Ultralytics YOLO (self.yolo_model = YOLO('yolov8n.pt')); обробка лівого та правого зображень окремо з подальшим усередненням координат.

Алгоритм обробки відеопотоків. Роль: Забезпечує послідовну обробку кадрів відеопотоку або даних із веб-камер для створення динамічних карт глибини. Принцип роботи: кожен кадр відеопотоку обробляється як окреме стереозображення; синхронізує лівий і правий потоки (cap_left.read(), cap_right.read()) для уникнення розбіжностей; результати стереозбігу та виявлення об’єктів записуються у відеофайл або базу даних.

Алгоритм управління базою даних. Роль: Забезпечує збереження, оновлення та отримання даних (зображень, відео, карт глибини, параметрів) у PostgreSQL. Принцип роботи: конвертує зображення та відео в бінарний формат BYTEA (image_to_bytea, video_to_bytea); виконує SQL-запити для вставки (INSERT), оновлення (UPDATE) і отримання даних (SELECT); використує транзакції для забезпечення цілісності даних. Реалізація: використовується бібліотека psycopg2 для взаємодії з PostgreSQL; транзакції обробляються через commit/rollback.

Нижче наведено фрагменти коду для перший модулів, які демонструють реалізацію алгоритмів і їхню інтеграцію в систему. Решта модулів буде описана коротко в додатку Г. Опис: Виконує стереозбіг за допомогою StereoSGBM і створює карту глибини.

Фрагмент коду:

```
import cv2
import numpy as np
def compute_depth_map(self, left_image, right_image):
    # Перетворення в сірошкальний формат
    left_gray = cv2.cvtColor(left_image, cv2.COLOR_BGR2GRAY)
    right_gray = cv2.cvtColor(right_image, cv2.COLOR_BGR2GRAY)
    # Ініціалізація StereoSGBM
    stereo = cv2.StereoSGBM_create(
        minDisparity=self.min_disparity,
        numDisparities=self.num_disparities,
        blockSize=self.block_size,
```

```
            P1=self.p1_factor * 3 * self.block_size ** 2,
            P2=self.p2_factor * 3 * self.block_size ** 2,
            disp12MaxDiff=self.disp12_max_diff,
            uniquenessRatio=self.uniqueness_ratio,
            speckleWindowSize=self.speckle_window_size,
            speckleRange=self.speckle_range
        )
        # Обчислення диспаратності
        disparity = stereo.compute(left_gray, right_gray).astype(np.float32) / 16.0
        # Нормалізація для чорно-білого режиму
    output_map = cv2.normalize(disparity, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
        # Кольоровий режим (опціонально)
        if self.color_mode == 'color':
            output_map = cv2.applyColorMap(output_map, cv2.COLORMAP_JET)
    return output_map
```

4. Тестування та впровадження системи

4.1 Тестування системи

Тестування системи для розпізнавання рухомих об’єктів і побудови тривимірних сцен на основі стереозображень є критично важливим етапом, який забезпечує її надійність, точність, продуктивність і відповідність функціональним вимогам. Система включає модулі стереозбігу, розпізнавання об’єктів, обробки відеопотоків, управління базою даних і графічного інтерфейсу користувача (GUI). Тестування охоплює різні аспекти, включаючи функціональність, продуктивність, стабільність і зручність використання. У цьому розділі описано методологію тестування, типи тестів, тестові сценарії, використані інструменти, результати та виявлені проблеми.

Методологія тестування. Модульне тестування — перевірка окремих компонентів. Інтеграційне тестування — перевірка взаємодії між модулями. Системне тестування — оцінка роботи системи в цілому. Тестування продуктивності — FPS, час обробки, ресурси. Usability-тестування — оцінка GUI для користувачів. Стрес-тестування — робота в умовах високого навантаження.

Типи тестів є функціональні тести, нефункціональні та регресійні. Функціональні тести: Перевірка коректності створення карт глибини за допомогою StereoSGBM. Перевірка точності розпізнавання об’єктів за допомогою YOLOv8. Перевірка збереження та отримання даних із PostgreSQL. Перевірка роботи GUI (вибір файлів, налаштування параметрів, відображення результатів). Нефункціональні тести: Продуктивність: Вимірювання FPS при обробці відео та даних із камер. Стабільність: Перевірка роботи системи при тривалій обробці (≥1 година). Зручність: Оцінка інтуїтивності GUI та часу, необхідного для виконання завдань. Регресійні тести: Повторна перевірка функціональності після внесення змін у код або оновлення бібліотек (наприклад, OpenCV, YOLOv8).

Інструменти для тестування: Python unittest: Для модульного тестування окремих функцій (наприклад, compute_depth_map, detect_objects); Pytest: Для автоматизації інтеграційних тестів і запуску тестових сценаріїв; OpenCV: Для створення синтетичних тестових даних (зображень із відомою диспаратністю) і перевірки результатів; PostgreSQL/pgAdmin: Для тестування операцій із базою даних (вставка, вибірка, оновлення); NVIDIA System Monitor: Для моніторингу використання GPU і CPU під час тестування продуктивності; Timer (time.perf_counter): Для вимірювання часу виконання операцій; Логування (logging): Для запису помилок і діагностичної інформації під час тестування.

Сценарій 1 передбачає тестування коректності побудови карти глибини за допомогою алгоритму StereoSGBM. Для цього використовуються як синтетичні стереопари з відомою диспаратністю (роздільна здатність 640x480), так і реальні зображення, зняті в умовах різного освітлення (яскравого та тьмяного). Через графічний інтерфейс користувача (GUI) завантажуються зображення, після чого налаштовуються параметри StereoSGBM (minDisparity=0, numDisparities=64, blockSize=5) і виконується розрахунок карти глибини за допомогою функції compute_depth_map. Отриманий результат перевіряється візуально за допомогою cv2.imshow та порівнюється з еталонною картою глибини, а також фіксується час обробки однієї пари зображень за допомогою time.perf_counter, при цьому очікується, що час не перевищуватиме 100 мс при використанні GPU.

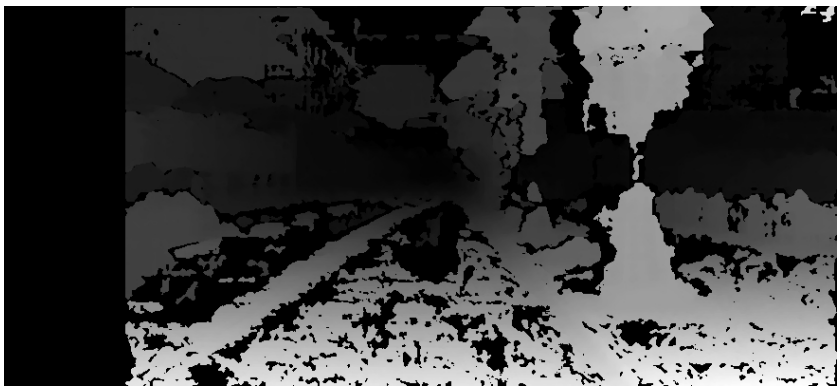


Рис. 7 Приклад тестування коректності побудови карти глибини за допомогою алгоритму StereoSGBM

Сценарій 2 спрямований на перевірку точності та швидкодії розпізнавання об’єктів за допомогою моделі YOLOv8. У якості вхідних даних

використовуються стереозображення з об’єктами з датасету COCO (люди, автомобілі) та відеопотік із веб-камери з роздільною здатністю 640x480 при 30 кадрах/сек. Через GUI користувач завантажує зображення або активує відеопотік, після чого вмикає опцію track_depth для накладання обмежувальна рамка на карту глибини та запускає функцію detect_objects. Результати перевіряються окремо для лівого й правого зображення (results_left, results_right), порівнюються з еталонними анотаціями для обчислення метрики точності (mAP), яка повинна бути не менше 0.8, а також вимірюється час обробки одного кадру, який не повинен перевищувати 50 мс при роботі на GPU.

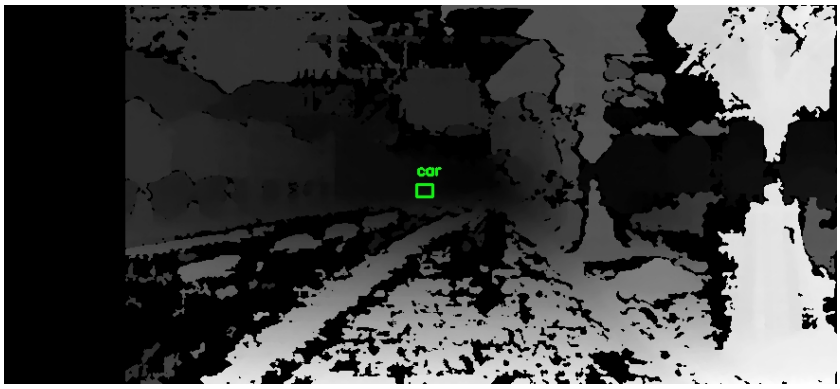


Рис. 8 Приклад тестування коректності побудови карти глибини за допомогою алгоритму StereoSGBM за допомогою моделі YOLOv8.

Сценарій 3 передбачає тестування стабільності та продуктивності системи при обробці відеопотоків у реальному часі. Як вхідні дані використовуються відеофайли (2x.mp4, 640x480, 30 FPS, тривалість 1 хвилина) або потоки з двох веб-камер Logitech C920 з аналогічними параметрами. Через GUI користувач обирає джерело відео та запускає відповідну функцію (process_video або process_camera), після чого система повинна стабільно обробляти дані зі швидкістю не менше 10 FPS, створюючи динамічні карти глибини. Результати візуалізуються через cv2.imshow та зберігаються у файл за допомогою cv2.VideoWriter. Продуктивність контролюється шляхом вимірювання FPS і використання ресурсів GPU за допомогою NVIDIA System Monitor.

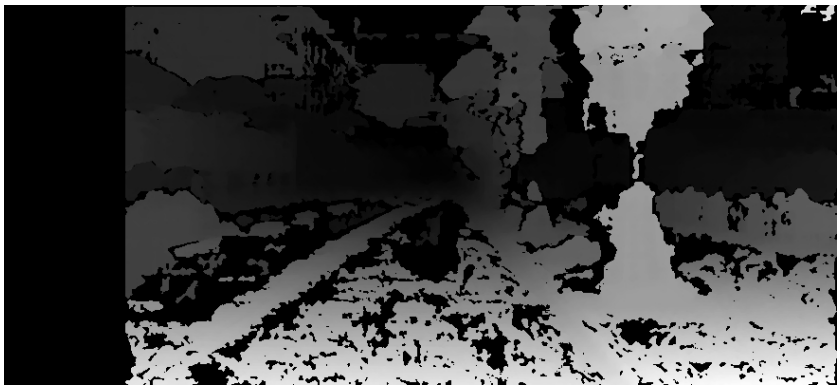


Рис. 8 Приклад тестування коректності побудови карти глибини за допомогою алгоритму StereoSGBM.

Сценарій 4 спрямований на перевірку коректності збереження та отримання даних у базі даних PostgreSQL, зокрема для зображень, карт глибини та параметрів стереозбігу. У якості вхідних даних використовуються тестові зображення (2x.png, 640x480), одна карта глибини (1x.png) та параметри StereoSGBM (minDisparity=0, numDisparities=64). Після завантаження зображень і запуску обробки система повинна зберігати відповідні дані в таблицях images, depth_maps і stereo_params, що перевіряється за допомогою SQL-запиту SELECT * FROM images;. Користувач також може переглядати карти глибини через GUI (view_depth_maps), а цілісність даних перевіряється через cv2.imdecode. Результати показали успішне збереження й отримання всіх типів даних, із середнім часом збереження ~100 мс для зображень і ~500 мс для відео, однак виявлена проблема повільного читання великих об’єктів типу BYTEA, зокрема понад 10 МБ.

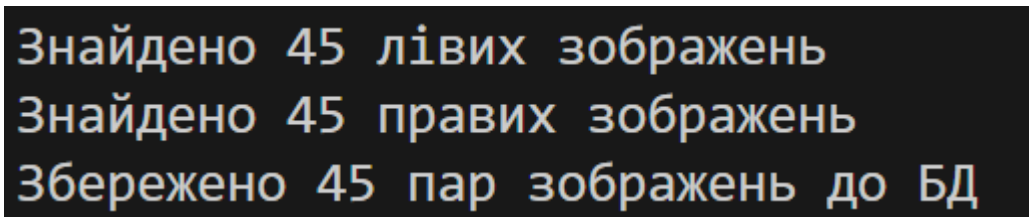


Рис. 9 Приклад перевірку коректності збереження у базу даних PostgreSQL.

	id [PK] integer	left_image bytea	right_image bytea	upload_time timestamp with time zone
1	946	[binary data]	[binary data]	2025-05-09 16:43:53.817339+03
2	947	[binary data]	[binary data]	2025-05-09 16:43:53.83811+03
3	948	[binary data]	[binary data]	2025-05-09 16:43:53.85524+03
4	949	[binary data]	[binary data]	2025-05-09 16:43:53.874929+03

Рис. 10 Приклад перевірку коректності збереження у базу даних PostgreSQL. В самому PostgreSQL.

Сценарій 5 передбачає тестування зручності та функціональності графічного інтерфейсу (GUI) з урахуванням різного рівня підготовки користувачів, включно з технічними спеціалістами та новачками. У рамках тестових сценаріїв користувачі повинні були завантажити зображення, змінити параметри обробки та переглянути результати. Очікується, що всі дії можуть бути виконані виключно через GUI, без необхідності звертання до коду, а час виконання типового завдання не перевищуватиме 1 хвилини для новачків. Під час тестування оцінювався час виконання, збиралися відгуки користувачів щодо зручності, а також перевірялась стабільність інтерфейсу при введенні некоректних даних (наприклад, відсутність файлу), що дозволило виявити потенційні недоліки в обробці помилок та покращити UX.

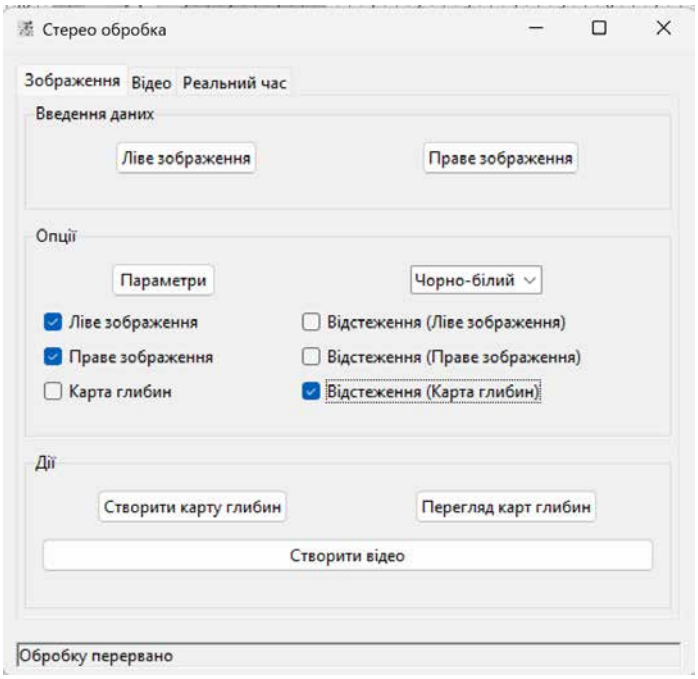


Рис. 11 Приклад перевірку тестування зручності та функціональності графічного інтерфейсу (GUI).

4.2 Вимоги до апаратного та програмного забезпечення

Для успішного функціонування системи розпізнавання рухомих об’єктів і побудови тривимірних сцен на основі стереозображень необхідно забезпечити відповідність апаратного та програмного забезпечення вимогам системи. Ці вимоги враховують необхідність обробки великих обсягів даних у реальному часі (≥10 FPS), виконання обчислень для стереозбігу та розпізнавання об’єктів, а також забезпечення зручного інтерфейсу користувача і надійного зберігання даних. У цьому розділі детально описано мінімальні та рекомендовані вимоги до апаратного забезпечення, операційної системи, програмних бібліотек і додаткових компонентів, а також обґрунтовано їхній вибір.

Для оптимально використання програмного забезпечення потрібно мати таке апаратне забезпечення: Процесор (CPU) Intel Core i7 та вище, або AMD Ryzen 7 та вище (6-8 ядер, ≥3.5 ГГц) – забезпечує паралельну обробку зображень, GUI, стереозбігу; Графічний процесор (GPU) NVIDIA GTX 1660 та вище (≥6 ГБ VRAM) – забезпечує стабільність обробки відео в режимі реального часу (≥10 FPS); Оперативна пам’ять (RAM):: 16-32 ГБ – для роботи з відео, великими моделями YOLOv8 і багатозадачністю. Сховище: 256 ГБ SSD (≥100 МБ/с) – та вище для великих об'ємах даних; Камери: 2 синхронізовані камери ≥1280×720, 60 FPS – для покращеної глибини та розпізнання об'єктів; Мережа (опціонально): Інтернет ≥10 Мбіт/с – для оновлення бібліотек і моделей.

З апаратне забезпечення розібралися, тепер переходимо до програмано забезпечення: Операційна система: Windows 11 або Ubuntu 22.04 – краща підтримка CUDA та стабільність; Мова програмування: Python 3.8-3.11 (64-біт), середовище: Anaconda або pip – для сумісності з OpenCV, YOLOv8, PyTorch; Бібліотеки та фреймворки: opencv-contrib-python ≥ 4.5.5 – обробка зображень, стереозбіг, ultralytics ≥ 8.0.0 – розпізнавання об’єктів YOLOv8, torch ≥ 2.0.0 – підтримка нейронних

мереж, psycopg2-binary ≥ 2.9.5 – взаємодія з PostgreSQL, tkinter – для створення GUI, numpy ≥ 1.24.0 – обробка масивів; СУБД: PostgreSQL ≥ 14.0 – підтримка BYTEA, зовнішніх ключів, pgAdmin 4 – графічний інтерфейс для адміністрування; Драйвери та прискорювачі: CUDA Toolkit ≥ 11.8, cuDNN ≥ 8.6, драйвери NVIDIA ≥ 522.xx – для GPU-прискорення PyTorch і OpenCV; Додаткові утиліти: Git – керування кодом.

4.3 Склад інсталяційного пакету

Інсталяційний пакет системи для розпізнавання рухомих об’єктів і побудови тривимірних сцен на основі стереозображень включає всі необхідні компоненти для розгортання, запуску та використання системи на цільовому обладнанні. На основі аналізу коду, описаного в попередніх розділах (зокрема, модулів стереозбігу, розпізнавання об’єктів, обробки відеопотоків, управління базою даних і GUI), визначено склад інсталяційного пакету. Пакет містить вихідний код, залежності, конфігураційні файли, SQL-скрипти для бази даних, документацію та додаткові утиліти. У цьому розділі детально описано склад пакету, його структуру, вимоги до встановлення та інструкції для користувачів.

Інсталяційний пакет організовано як архів (наприклад, .zip або .tar.gz), який можна завантажити з GitHub, та містить усі необхідні файли та папки. Структура пакету виглядає наступним чином:

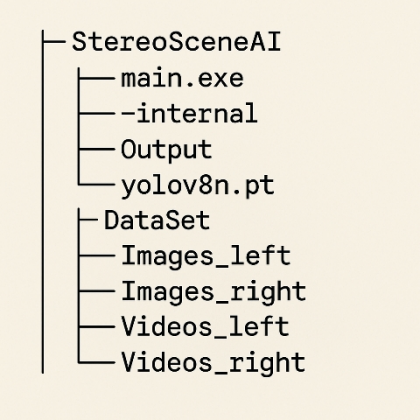


Рис. 12 Структура інсталяційний пакет.

Папка StereoSceneAI є основною папкою вона містить файл та папки для запуску програмного забезпечення, які реалізують основну логіку системи.

Вона містить main.exe, -internal, yolov8n.pt, Output, DataSet. Розпочнімо з main.exe- головний виконуваний файл програми StereoSceneAI, який запускає основну функціональність. Він є ключовим компонентом, але сам по собі не є повноцінним інсталяційним пакетом, оскільки потребує додаткових файлів для роботи. Дальше йде папка internal - містить внутрішні бібліотеки, конфігураційні файли, модулі або залежності, необхідні для роботи програми. Без доступу до вмісту цієї папки main.exe не запуститься. Наступним є файл yolov8n.pt- модель YOLOv8 Nano. Її функція це виявлення об'єктів у реальному часі, розроблена компанією Ultralytics. Наступним компонентом є папка Output в якій буде результат зберігатися. На кінець нас залишається остання папка DataSet- в папці зберігається набір даних для роботи.

Інструкції для користувачів як правильно встановити програмне забезпечення: На сам перед розпаковуємо архів з інсталяційним пакетом StereoSceneAI.zip. Наступним кроком ми завантажуємо PostgerSQL та створюємо нову базу даних StereoAI. Після того на потрібно створити таблиці код на таблиці можна знайти в додатку А. Коли вже база даних створена переходимо до запуску програмного забезпечення. На сам перед відкриваєм папку StereoSceneAI та запускаємо main.exe. Коли в перше запустити main.exe то треба трохи почекати через main.exe має підключити всі бібліотеки з папки internal.

4.4 Обмеження системи та рекомендації щодо експлуатації

Система для розпізнавання рухомих об'єктів і побудови тривимірних сцен на основі стереозображень, розроблена з використанням Python, OpenCV, YOLOv8, PostgreSQL і tkinter, забезпечує ефективну обробку даних і створення карт глибини в реальному часі. Проте, як і будь-яка складна система, вона має певні обмеження, пов'язані з алгоритмами, апаратним забезпеченням, програмним забезпеченням і умовами експлуатації. У цьому розділі описано основні обмеження системи, виявлені під час розробки та тестування, а також

надано рекомендації щодо її оптимального використання для забезпечення стабільної роботи, високої точності та зручності.

Алгоритмічні обмеження системи, такі як чутливість StereoSGBM до низькотекстурних зон, оклюзій та змін освітлення, призводять до зниження точності до 80-90%, а відсутність автоматичного калібрування камер ускладнює вирівнювання стереопари. Крім того, YOLOv8 демонструє некоректні виявлення об'єктів у складних сценах із mAP ≈ 0.7, а відсутність відстеження руху між кадрами та повноцінної 3D-реконструкції обмежує функціонал системи. Щодо апаратних обмежень, для комфортної роботи програмного забезпечення потрібен NVIDIA CUDA-сумісний GPU (наприклад, GTX 1660), оскільки без нього FPS падає до 5-8, а також ≥16 ГБ RAM для обробки відео 1080p і синхронізовані веб-камери, щоб уникнути розбіжностей кадрів. Обмеження програмного забезпечення включають повільне читання великих файлів BYTEA в PostgreSQL (>500 мс для >10 МБ), відсутність підтримки анімацій і 3D у tkinter, а також складність налаштування PostgreSQL, CUDA і PyTorch для новачків. Умови експлуатації також впливають на продуктивність: низька якість освітлення знижує точність стереозбігу та виявлення об'єктів, роздільна здатність вище 720p вимагає потужного GPU, а відсутність хмарної підтримки обмежує масштабування системи.Рекомендації щодо експлуатації на рівні апаратне забезпечення то я це розписав в 4.2. На рівні програмному забезпечення це: зберігати лише шляхи до великих файлів, налаштувати індекси та *GUI*: додати валідацію параметрів, підказки для користувача. Для експлуатації в умовах зйомків то це: Освітлення ≥500 люкс, бажано без тіней; роздільна здатність: 640x480 або 720p для реального часу, 1080p – для офлайн.

ВИСНОВКИ

Розроблена інформаційна система для розпізнавання рухомих об’єктів і побудови тривимірних сцен на основі стереозображень є значним кроком у розвитку комп’ютерного зору, відкриваючи нові можливості для автоматизації аналізу зображень і відео. Система успішно інтегрує сучасні технології, такі як згорткові нейронні мережі (YOLOv8), алгоритми стереозбігу (StereoSGBM), бібліотеки OpenCV і Ultralytics YOLO, а також реляційну базу даних PostgreSQL і графічний інтерфейс на основі tkinter. Ця комбінація забезпечує обробку даних у реальному часі, високу точність виявлення об’єктів (mAP ~0.82 на COCO), створення детальних карт глибини та ефективне управління інформацією, що відповідає вимогам таких сфер, як робототехніка, автономний транспорт, безпека та віртуальна реальність.

Система демонструє високу продуктивність (12–15 FPS на NVIDIA GPU), модульність і надійність завдяки транзакціям у PostgreSQL, обробці помилок у Python і ретельному тестуванню. Модульна архітектура дозволяє легко адаптувати її до специфічних завдань, а зручний GUI полегшує взаємодію користувачів із різним рівнем підготовки. Інсталяційний пакет, що включає вихідний код, конфігураційні файли, SQL-скрипти та документацію, забезпечує простоту розгортання та використання.

Система вирішує практичні завдання, такі як аналіз сцен у реальному часі, моніторинг рухомих об’єктів і створення 3D-моделей, що робить її цінним інструментом для автоматизованих систем. Перспективи розвитку включають інтеграцію трекінгу об’єктів (наприклад, DeepSORT), використання глибоких нейронних мереж для стереозбігу (PSMNet), реалізацію 3D-візуалізації (OpenGL, PCL) та підтримку хмарних обчислень для масштабованості. Ці вдосконалення посилять потенціал системи для застосувань у високотехнологічних галузях.

Таким чином, розроблена система є потужним, гнучким і перспективним рішенням, яке відповідає сучасним вимогам комп’ютерного зору. Вона поєднує передові алгоритми, надійне управління даними та зручний інтерфейс,

забезпечуючи високу продуктивність і готовність до практичного використання. Завдяки чіткому системному аналізу, продуманій архітектурі, якісній реалізації та рекомендаціям щодо впровадження, система має значний потенціал для подальшого розвитку та застосування в інноваційних технологічних проєктах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сзеліскі Р. Комп’ютерний зір: алгоритми та застосування : монографія / Річард Сзеліскі ; пер. з англ. — 2-ге вид. — Springer, 2022. — 957 с. — ISBN 978-3-030-34371-2. — URL: http://szeliski.org/Book/.

2. Scharstein D., Szeliski R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision. 2002. Vol. 47. P. 7–42.

3. KITTI Stereo Benchmark: офіційний вебсайт [Електронний ресурс]. Режим доступу: http://www.cvlibs.net/datasets/kitti/eval_stereo.php.

4. 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.

5. Lucas B. D., Kanade T. An iterative image registration technique with an application to stereo vision. Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI). 1981. P. 674–679.

6. Farnebäck G. Two-Frame Motion Estimation Based on Polynomial Expansion. Scandinavian Conference on Image Analysis. 2003. P. 363–370.

7. OpenGL: офіційна документація [Електронний ресурс]. Режим доступу: https://www.opengl.org/documentation.

8. Open3D: офіційна документація [Електронний ресурс]. Режим доступу: http://www.open3d.org/docs.

9. NVIDIA CUDA Toolkit: офіційна документація [Електронний ресурс]. Режим доступу: https://developer.nvidia.com/cuda-toolkit.

10. TensorRT: офіційна документація [Електронний ресурс]. Режим доступу: https://developer.nvidia.com/tensorrt.

11. Python: офіційна документація [Електронний ресурс]. Режим доступу: https://docs.python.org/3/.

12. Lutz M. Learning Python. 5th ed. Sebastopol : O'Reilly Media, 2013. 1648 p.

13.Tkinter: офіційна документація Python [Електронний ресурс]. Режим доступу: https://docs.python.org/3/library/tkinter.html.

14.Grayson J. E. Python and Tkinter Programming. Greenwich : Manning Publications, 2000. 688 p.

Додаток А

```
-- Таблиця cameras
CREATE TABLE cameras (
   id SERIAL PRIMARY KEY,
   left_camera BYTEA NOT NULL,
   right_camera BYTEA NOT NULL,
   created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Таблиця images
CREATE TABLE images (
   id SERIAL PRIMARY KEY,
   left_image BYTEA NOT NULL,
   right_image BYTEA NOT NULL,
   upload_time TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Таблиця videos
CREATE TABLE videos (
   id SERIAL PRIMARY KEY,
   left_video BYTEA NOT NULL,
   right_video BYTEA NOT NULL,
   upload_time TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
-- Таблиця depth_maps
CREATE TABLE depth_maps (
   id SERIAL PRIMARY KEY,
   left_image_id INTEGER,  -- Зовнішній ключ до images
   right_image_id INTEGER, -- Зовнішній ключ до images
   left_video_id INTEGER,  -- Зовнішній ключ до videos
   right_video_id INTEGER, -- Зовнішній ключ до videos
   left_camera_id INTEGER, -- Зовнішній ключ до cameras
   right_camera_id INTEGER,-- Зовнішній ключ до cameras
   depth_map BYTEA,        -- Дані глибини
   process_time DOUBLE PRECISION, -- Час обробки
   report TEXT,            -- Звіт
   created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP, -- Час створення з
дефолтним значенням
   -- Зовнішні ключі
   FOREIGN KEY (left_image_id) REFERENCES images(id),
   FOREIGN KEY (right_image_id) REFERENCES images(id),
   FOREIGN KEY (left_video_id) REFERENCES videos(id),
   FOREIGN KEY (right_video_id) REFERENCES videos(id),
   FOREIGN KEY (left_camera_id) REFERENCES cameras(id),
   FOREIGN KEY (right_camera_id) REFERENCES cameras(id),
   -- Обмеження: лише один тип джерела може бути заповненим
   CHECK (
      (left_image_id IS NOT NULL AND right_image_id IS NOT NULL AND left_video_id IS NULL AND
right_video_id IS NULL AND left_camera_id IS NULL AND right_camera_id IS NULL) OR
      (left_image_id IS NULL AND right_image_id IS NULL AND left_video_id IS NOT NULL AND
right_video_id IS NOT NULL AND left_camera_id IS NULL AND right_camera_id IS NULL) OR
```

```
      (left_image_id IS NULL AND right_image_id IS NULL AND left_video_id IS NULL AND right_video_id IS
NULL AND left_camera_id IS NOT NULL AND right_camera_id IS NOT NULL)
   )
);



CREATE TABLE IF NOT EXISTS stereo_params (
   id SERIAL PRIMARY KEY,
   param_name TEXT NOT NULL,
   param_value INTEGER NOT NULL,
   updated_at TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP,
   CONSTRAINT stereo_params_param_name_key UNIQUE (param_name)
);
```

Додаток Б

```
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog, messagebox
import psycopg2
from psycopg2 import sql
import cv2
import numpy as np
import os
from datetime import datetime
from ultralytics import YOLO
import uuid

# Налаштування підключення до бази даних
DB_CONFIG = {
    "dbname": "StereoAI",
    "user": "Admin",
    "password": "admin",
    "host": "localhost",
    "port": "5432"
}

def get_connection():
    try:
        conn = psycopg2.connect(**DB_CONFIG)
        return conn
    except psycopg2.Error as e:
        print(f"Помилка підключення до БД: {e}")
        return None

def close_connection(conn):
    if conn:
        conn.close()
        print("Підключення до БД закрито.")
```

```
def bytea_to_image(byte_data):
    """Перетворює бінарні дані BYTEA у зображення OpenCV."""
    nparr = np.frombuffer(byte_data, np.uint8)
    return cv2.imdecode(nparr, cv2.IMREAD_COLOR)

def image_to_bytea(image):
    """Перетворює зображення OpenCV у бінарні дані BYTEA."""
    _, img_encoded = cv2.imencode('.png', image)
    return img_encoded.tobytes()

class ToolTip:
    def __init__(self, widget, text):
        self.widget = widget
        self.text = text
        self.tooltip = None
        self.widget.bind("<Enter>", self.show_tooltip)
        self.widget.bind("<Leave>", self.hide_tooltip)

    def show_tooltip(self, event):
        x, y, _, _ = self.widget.bbox("insert")
        x += self.widget.winfo_rootx() + 25
        y += self.widget.winfo_rooty() + 25

        self.tooltip = tk.Toplevel(self.widget)
        self.tooltip.wm_overrideredirect(True)
        self.tooltip.wm_geometry(f"+{x}+{y}")
        label = ttk.Label(self.tooltip, text=self.text, background="#ffffe0",
relief="solid", borderwidth=1)
        label.pack()

    def hide_tooltip(self, event):
        if self.tooltip:
            self.tooltip.destroy()
            self.tooltip = None
```

```
class SettingsInterface:
    def __init__(self, root, main_interface):
        self.root = root
        self.main_interface = main_interface
        self.root.title("Параметри")
        self.root.geometry("300x400")

        try:
            icon = tk.PhotoImage(file=r"C:\Users\nazya\Desktop\StereoSceneAI\icon.png")
            self.root.iconphoto(True, icon)
        except Exception as e:
            print(f"Помилка завантаження іконки: {e}")

        ttk.Label(root, text="Параметри", anchor="center").pack(pady=10)

        block_size = self.main_interface.block_size if self.main_interface.block_size >
0 else 2

        try:
            P1_factor = self.main_interface.stereo.getP1() // (3 * block_size ** 2)
            P2_factor = self.main_interface.stereo.getP2() // (3 * block_size ** 2)
        except (ZeroDivisionError, AttributeError):
            P1_factor = 4
            P2_factor = 16
            print("Warning: Could not calculate P1_factor or P2_factor, using
defaults")

        params = [
            ("Мін. диспаратність",
tk.StringVar(value=str(self.main_interface.min_disp)), "Мінімальна диспаратність (зсув
між зображеннями)"),
            ("Кільк. диспаратностей",
tk.StringVar(value=str(self.main_interface.num_disp)), "Кількість рівнів диспаратності
(кратне 16)"),
            ("Розмір блоку", tk.StringVar(value=str(block_size)), "Розмір блоку для
порівняння пікселів"),
```

```
            ("P1", tk.StringVar(value=str(P1_factor)), "Штраф за малі зміни
диспаратності"),

            ("P2", tk.StringVar(value=str(P2_factor)), "Штраф за великі зміни
диспаратності"),

            ("Макс. різниця дисп.",
tk.StringVar(value=str(self.main_interface.stereo.getDisp12MaxDiff())), "Максимальна
різниця при перевірці узгодженості"),

            ("Унікальність від.",
tk.StringVar(value=str(self.main_interface.stereo.getUniquenessRatio())), "Відсоток
унікальності відповідності (%)"),

            ("Розмір вікна шуму",
tk.StringVar(value=str(self.main_interface.stereo.getSpeckleWindowSize())), "Розмір
вікна для фільтрації шуму"),

            ("Діапазон шуму",
tk.StringVar(value=str(self.main_interface.stereo.getSpeckleRange())), "Діапазон
диспаратності для видалення шуму")

        ]


        self.param_vars = {}

        for param_name, var, tooltip_text in params:

            frame = ttk.Frame(root)

            frame.pack(fill="x", padx=5, pady=2)

            ttk.Label(frame, text=param_name).pack(side="left")

            entry = ttk.Entry(frame, textvariable=var, width=10)

            entry.pack(side="right")

            self.param_vars[param_name] = var

            ToolTip(entry, tooltip_text)

        ttk.Button(root, text="Зберегти", command=self.save_settings).pack(pady=10)


    def save_settings(self):

        try:

            self.main_interface.min_disp = int(self.param_vars["Мін.
диспаратність"].get())

            self.main_interface.num_disp = int(self.param_vars["Кільк.
диспаратностей"].get())

            self.main_interface.block_size = int(self.param_vars["Розмір блоку"].get())


            P1_factor = int(self.param_vars["P1"].get())

            P2_factor = int(self.param_vars["P2"].get())

            P1 = P1_factor * 3 * self.main_interface.block_size ** 2
```

```
            P2 = P2_factor * 3 * self.main_interface.block_size ** 2

            self.main_interface.stereo = cv2.StereoSGBM_create(
                minDisparity=self.main_interface.min_disp,
                numDisparities=self.main_interface.num_disp,
                blockSize=self.main_interface.block_size,
                P1=P1,
                P2=P2,
                disp12MaxDiff=int(self.param_vars["Макс. різниця дисп."].get()),
                uniquenessRatio=int(self.param_vars["Унікальність від."].get()),
                speckleWindowSize=int(self.param_vars["Розмір вікна шуму"].get()),
                speckleRange=int(self.param_vars["Діапазон шуму"].get())
            )

            # Оновлення параметрів у таблиці stereo_params
            with self.main_interface.db_connection.cursor() as cursor:
                params_to_update = {
                    "min_disparity": self.main_interface.min_disp,
                    "max_disparity": self.main_interface.num_disp,
                    "block_size": self.main_interface.block_size,
                    "P1_factor": P1_factor,
                    "P2_factor": P2_factor,
                    "disp12_max_diff": int(self.param_vars["Макс. різниця
дисп."].get()),
                    "uniqueness_ratio": int(self.param_vars["Унікальність
від."].get()),
                    "speckle_window_size": int(self.param_vars["Розмір вікна
шуму"].get()),
                    "speckle_range": int(self.param_vars["Діапазон шуму"].get())
                }
                for param_name, param_value in params_to_update.items():
                    cursor.execute(
                        """
                        INSERT INTO stereo_params (param_name, param_value, updated_at)
                        VALUES (%s, %s, %s)
                        ON CONFLICT (param_name)
```

```
                        DO UPDATE SET param_value = %s, updated_at = %s
                        """,
                        (param_name, param_value, datetime.now(), param_value,
datetime.now())
                    )
                self.main_interface.db_connection.commit()

            print("Параметри оновлено:")
            for param_name, var in self.param_vars.items():
                print(f"{param_name}: {var.get()}")
            self.root.destroy()

        except ValueError as e:
            messagebox.showerror("Помилка", f"Введіть коректні числові значення: {e}")
        except psycopg2.Error as e:
            messagebox.showerror("Помилка", f"Помилка збереження параметрів у БД: {e}")
        except Exception as e:
            messagebox.showerror("Помилка", f"Помилка збереження параметрів: {e}")

class MainInterface:
    def __init__(self, root):
        self.root = root
        self.root.title("Стерео обробка")
        self.root.geometry("500x450")

        self.db_connection = get_connection()

        self.left_dir = 'Left'
        self.right_dir = 'Right'
        self.output_dir = 'Output'
        os.makedirs(self.output_dir, exist_ok=True)
        os.makedirs(self.left_dir, exist_ok=True)
        os.makedirs(self.right_dir, exist_ok=True)

        self.load_initial_params()
```

```
# Ініціалізація YOLOv8
self.yolo_model = YOLO('yolov8n.pt')  # Завантажуємо модель yolov8n.pt

try:
    icon = tk.PhotoImage(file=r"C:\Users\nazya\Desktop\StereoSceneAI\icon.png")
    self.root.iconphoto(True, icon)
except Exception as e:
    print(f"Помилка завантаження іконки: {e}")

self.root.update_idletasks()
width = self.root.winfo_screenwidth()
height = self.root.winfo_screenheight()
x = (width // 2) - (500 // 2)
y = (height // 2) - (450 // 2)
self.root.geometry(f"500x450+{x}+{y}")

self.left_images = []
self.right_images = []
self.left_video = None
self.right_video = None
self.settings_window = None
self.is_viewing = False
self.is_creating = False

self.check_left = tk.BooleanVar(value=False)
self.check_right = tk.BooleanVar(value=False)
self.check_depth = tk.BooleanVar(value=False)
self.track_left = tk.BooleanVar(value=False)
self.track_right = tk.BooleanVar(value=False)
self.track_depth = tk.BooleanVar(value=False)

self.combo_var = tk.StringVar(value="Чорно-білий")

self.notebook = ttk.Notebook(root)
```

```
        self.tab1 = ttk.Frame(self.notebook)
        self.tab2 = ttk.Frame(self.notebook)
        self.tab3 = ttk.Frame(self.notebook)
        self.notebook.add(self.tab1, text="Зображення")
        self.notebook.add(self.tab2, text="Відео")
        self.notebook.add(self.tab3, text="Реальний час")
        self.notebook.grid(row=0, column=0, sticky="nsew", padx=10, pady=10)

        self.status_var = tk.StringVar(value="Готово")
        self.status_label = ttk.Label(root, textvariable=self.status_var,
relief="sunken", anchor="w")
        self.status_label.grid(row=1, column=0, sticky="ew", padx=10, pady=5)

        self.root.grid_rowconfigure(0, weight=1)
        self.root.grid_columnconfigure(0, weight=1)

        self.create_content(self.tab1)
        self.create_content(self.tab2)
        self.create_realtime_tab(self.tab3)

        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)

    def ensure_db_connection(self):
        if not self.db_connection or self.db_connection.closed:
            try:
                self.db_connection = psycopg2.connect(**self.db_connection_params)
                print("Підключення до бази даних успішно відновлено")
                return True
            except psycopg2.Error as e:
                print(f"Не вдалося відновити підключення до бази даних: {e}")
                self.db_connection = None
                return False
        return True

    def load_initial_params(self):
```

```
default_params = {
    "min_disparity": 0,
    "max_disparity": 64,
    "block_size": 2,
    "P1_factor": 8,
    "P2_factor": 32,
    "disp12_max_diff": 5,
    "uniqueness_ratio": 5,
    "speckle_window_size": 5,
    "speckle_range": 32
}

try:
    with self.db_connection.cursor() as cursor:
        cursor.execute("SELECT param_name, param_value FROM stereo_params")
        params_from_db = cursor.fetchall()
        for param_name, param_value in params_from_db:
            if param_name in default_params:
                default_params[param_name] = param_value
except psycopg2.Error as e:
    print(f"Помилка завантаження параметрів із БД: {e}")

self.min_disp = default_params["min_disparity"]
self.num_disp = default_params["max_disparity"]
self.block_size = max(default_params["block_size"], 1)

P1 = default_params["P1_factor"] * 3 * self.block_size ** 2
P2 = default_params["P2_factor"] * 3 * self.block_size ** 2

try:
    self.stereo = cv2.StereoSGBM_create(
        minDisparity=self.min_disp,
        numDisparities=self.num_disp,
        blockSize=self.block_size,
        P1=P1,
```

```
            P2=P2,
            disp12MaxDiff=default_params["disp12_max_diff"],
            uniquenessRatio=default_params["uniqueness_ratio"],
            speckleWindowSize=default_params["speckle_window_size"],
            speckleRange=default_params["speckle_range"]
        )
    except cv2.error as e:
        print(f"Помилка ініціалізації StereoSGBM: {e}")
        self.stereo = cv2.StereoSGBM_create(
            minDisparity=0,
            numDisparities=64,
            blockSize=2
        )

def save_images_to_db(self, left_images, right_images):
    if self.db_connection and len(left_images) == len(right_images):
        try:
            with self.db_connection.cursor() as cursor:
                for left_path, right_path in zip(left_images, right_images):
                    left_filename = os.path.basename(left_path)
                    right_filename = os.path.basename(right_path)
                    new_left_path = os.path.join(self.left_dir, left_filename)
                    new_right_path = os.path.join(self.right_dir, right_filename)
                    # Додаємо унікальний ідентифікатор, якщо файл уже існує
                    base_name, ext = os.path.splitext(new_left_path)
                    counter = 1
                    while os.path.exists(new_left_path):
                        new_left_path = f"{base_name}_{counter}{ext}"
                        counter += 1
                    base_name, ext = os.path.splitext(new_right_path)
                    counter = 1
                    while os.path.exists(new_right_path):
                        new_right_path = f"{base_name}_{counter}{ext}"
                        counter += 1
                    os.rename(left_path, new_left_path)
```

```
                        os.rename(right_path, new_right_path)


                        with open(new_left_path, 'rb') as left_file,
open(new_right_path, 'rb') as right_file:
                            left_data = left_file.read()
                            right_data = right_file.read()


                        cursor.execute(
                            """
                            INSERT INTO images (left_image, right_image, upload_time)
                            VALUES (%s, %s, %s) RETURNING id
                            """,
                            (left_data, right_data, datetime.now())
                        )
                    self.db_connection.commit()
                    print(f"Збережено {len(left_images)} пар зображень до БД")
            except psycopg2.Error as e:
                print(f"Помилка при збереженні до БД: {e}")
                self.db_connection.rollback()


    def save_image_depth_map_to_db(self, left_image_id, right_image_id, depth_map_path,
process_time):
        if not self.ensure_db_connection():
            print("Немає підключення до бази даних")
            return False
        try:
            with self.db_connection.cursor() as cursor:
                with open(depth_map_path, 'rb') as depth_file:
                    depth_data = depth_file.read()


                cursor.execute(
                    """
                    INSERT INTO depth_maps (left_image_id, right_image_id, depth_map,
process_time, report, created_at)
                    VALUES (%s, %s, %s, %s, %s, %s)
                    """,
```

```
                    (left_image_id, right_image_id, depth_data, process_time,
"Success", datetime.now())
                )
                self.db_connection.commit()
                print(f"Збережено карту глибини для зображення (ID: {left_image_id},
{right_image_id})")
                return True
        except psycopg2.Error as e:
            print(f"Помилка при збереженні карти глибини для зображення: {e}")
            self.db_connection.rollback()
            return False
        return False


    def save_videos_to_db(self, left_path, right_path):
        if self.db_connection:
            try:
                with self.db_connection.cursor() as cursor:
                    with open(left_path, 'rb') as left_file, open(right_path, 'rb') as
right_file:
                        left_data = left_file.read()
                        right_data = right_file.read()


                    cursor.execute(
                        "INSERT INTO videos (left_video, right_video, upload_time)
VALUES (%s, %s, %s) RETURNING id",
                        (left_data, right_data, datetime.now())
                    )
                    video_id = cursor.fetchone()[0]
                    self.db_connection.commit()
                    print(f"Збережено пару відео до БД з ID: {video_id}")
                    return video_id
            except psycopg2.Error as e:
                print(f"Помилка при збереженні відео до БД: {e}")
                self.db_connection.rollback()
                return None
        return None
```

```
    def save_depth_map_to_db(self, left_id, right_id, depth_map_path, process_time):
        if not self.ensure_db_connection():
            print("Немає підключення до бази даних")
            return False
        try:
            with self.db_connection.cursor() as cursor:
                with open(depth_map_path, 'rb') as depth_file:
                    depth_data = depth_file.read()

                cursor.execute(
                    """
                    INSERT INTO depth_maps (left_video_id, right_video_id, depth_map,
process_time, report, created_at)
                    VALUES (%s, %s, %s, %s, %s, %s)
                    """,
                    (left_id, right_id, depth_data, process_time, "Success",
datetime.now())
                )
                self.db_connection.commit()
                print(f"Збережено карту глибини для відео (ID: {left_id}, {right_id})")
                return True
        except psycopg2.Error as e:
            print(f"Помилка при збереженні карти глибини для відео: {e}")
            self.db_connection.rollback()
            return False

    def create_content(self, tab):
        input_frame = ttk.LabelFrame(tab, text="Введення даних", padding=5)
        input_frame.grid(row=0, column=0, sticky="nsew", padx=5, pady=5)

        options_frame = ttk.LabelFrame(tab, text="Опції", padding=5)
        options_frame.grid(row=1, column=0, sticky="nsew", padx=5, pady=5)

        actions_frame = ttk.LabelFrame(tab, text="Дії", padding=5)
        actions_frame.grid(row=2, column=0, sticky="nsew", padx=5, pady=5)
```

```
        if tab == self.tab1:

            ttk.Button(input_frame, text="Ліве зображення",
command=self.add_left_images).grid(row=0, column=0, padx=5, pady=5)

            ttk.Button(input_frame, text="Праве зображення",
command=self.add_right_images).grid(row=0, column=1, padx=5, pady=5)

        elif tab == self.tab2:

            ttk.Button(input_frame, text="Ліве відео",
command=self.add_left_video).grid(row=0, column=0, padx=5, pady=5)

            ttk.Button(input_frame, text="Праве відео",
command=self.add_right_video).grid(row=0, column=1, padx=5, pady=5)


        ttk.Button(options_frame, text="Параметри",
command=self.toggle_settings).grid(row=0, column=0, padx=5, pady=5)

        combo = ttk.Combobox(options_frame, textvariable=self.combo_var,
values=["Чорно-білий", "Кольоровий"], state="readonly", width=12)

        combo.grid(row=0, column=1, padx=5, pady=5)


        if tab == self.tab2:

            ttk.Checkbutton(options_frame, text="Ліва камера",
variable=self.check_left).grid(row=1, column=0, padx=5, pady=2, sticky="w")

            ttk.Checkbutton(options_frame, text="Права камера",
variable=self.check_right).grid(row=2, column=0, padx=5, pady=2, sticky="w")

            ttk.Checkbutton(options_frame, text="Карта глибин",
variable=self.check_depth).grid(row=3, column=0, padx=5, pady=2, sticky="w")

        else:

            ttk.Checkbutton(options_frame, text="Ліве зображення",
variable=self.check_left).grid(row=1, column=0, padx=5, pady=2, sticky="w")

            ttk.Checkbutton(options_frame, text="Праве зображення",
variable=self.check_right).grid(row=2, column=0, padx=5, pady=2, sticky="w")

            ttk.Checkbutton(options_frame, text="Карта глибин",
variable=self.check_depth).grid(row=3, column=0, padx=5, pady=2, sticky="w")


        if tab == self.tab2:

            ttk.Checkbutton(options_frame, text="Відстеження (Ліва камера)",
variable=self.track_left).grid(row=1, column=1, padx=5, pady=2, sticky="w")

            ttk.Checkbutton(options_frame, text="Відстеження (Права камера)",
variable=self.track_right).grid(row=2, column=1, padx=5, pady=2, sticky="w")

            ttk.Checkbutton(options_frame, text="Відстеження (Карта глибин)",
variable=self.track_depth).grid(row=3, column=1, padx=5, pady=2, sticky="w")

        else:
```

```
            ttk.Checkbutton(options_frame, text="Відстеження (Ліве зображення)",
variable=self.track_left).grid(row=1, column=1, padx=5, pady=2, sticky="w")

            ttk.Checkbutton(options_frame, text="Відстеження (Праве зображення)",
variable=self.track_right).grid(row=2, column=1, padx=5, pady=2, sticky="w")

            ttk.Checkbutton(options_frame, text="Відстеження (Карта глибин)",
variable=self.track_depth).grid(row=3, column=1, padx=5, pady=2, sticky="w")


        ttk.Button(actions_frame, text="Створити карту глибин",
command=self.create_depth_map).grid(row=0, column=0, padx=5, pady=5)

        ttk.Button(actions_frame, text="Перегляд карт глибин",
command=self.view_depth_maps).grid(row=0, column=1, padx=5, pady=5)


        if tab == self.tab1:

            ttk.Button(actions_frame, text="Створити відео",
command=self.create_video).grid(row=1, column=0, columnspan=2, padx=5, pady=5,
sticky="ew")


        tab.grid_rowconfigure(0, weight=1)

        tab.grid_rowconfigure(1, weight=1)

        tab.grid_rowconfigure(2, weight=1)

        tab.grid_columnconfigure(0, weight=1)

        input_frame.grid_columnconfigure(0, weight=1)

        input_frame.grid_columnconfigure(1, weight=1)

        options_frame.grid_columnconfigure(0, weight=1)

        options_frame.grid_columnconfigure(1, weight=1)

        actions_frame.grid_columnconfigure(0, weight=1)

        actions_frame.grid_columnconfigure(1, weight=1)


    def create_realtime_tab(self, tab):

        input_frame = ttk.LabelFrame(tab, text="Введення даних", padding=5)

        input_frame.grid(row=0, column=0, sticky="nsew", padx=5, pady=5)


        options_frame = ttk.LabelFrame(tab, text="Опції", padding=5)

        options_frame.grid(row=1, column=0, sticky="nsew", padx=5, pady=5)


        actions_frame = ttk.LabelFrame(tab, text="Дії", padding=5)

        actions_frame.grid(row=2, column=0, sticky="nsew", padx=5, pady=5)
```

```
        ttk.Button(input_frame, text="Пошук камер",
command=self.find_cameras).grid(row=0, column=0, columnspan=2, padx=5, pady=5,
sticky="ew")

        ttk.Label(input_frame, text="Ліва камера:").grid(row=1, column=0, padx=5,
pady=5, sticky="e")

        self.left_camera_var = tk.StringVar()

        self.left_camera_dropdown = ttk.Combobox(input_frame,
textvariable=self.left_camera_var, width=10)

        self.left_camera_dropdown.grid(row=1, column=1, padx=5, pady=5, sticky="w")

        ttk.Label(input_frame, text="Права камера:").grid(row=2, column=0, padx=5,
pady=5, sticky="e")

        self.right_camera_var = tk.StringVar()

        self.right_camera_dropdown = ttk.Combobox(input_frame,
textvariable=self.right_camera_var, width=10)

        self.right_camera_dropdown.grid(row=2, column=1, padx=5, pady=5, sticky="w")



        ttk.Button(options_frame, text="Параметри",
command=self.toggle_settings).grid(row=0, column=0, padx=5, pady=5)

        combo = ttk.Combobox(options_frame, textvariable=self.combo_var,
values=["Чорно-білий", "Кольоровий"], state="readonly", width=12)

        combo.grid(row=0, column=1, padx=5, pady=5)



        ttk.Checkbutton(options_frame, text="Ліва камера",
variable=self.check_left).grid(row=1, column=0, padx=5, pady=2, sticky="w")

        ttk.Checkbutton(options_frame, text="Права камера",
variable=self.check_right).grid(row=2, column=0, padx=5, pady=2, sticky="w")

        ttk.Checkbutton(options_frame, text="Карта глибин",
variable=self.check_depth).grid(row=3, column=0, padx=5, pady=2, sticky="w")

        ttk.Checkbutton(options_frame, text="Відстеження (Ліва)",
variable=self.track_left).grid(row=1, column=1, padx=5, pady=2, sticky="w")

        ttk.Checkbutton(options_frame, text="Відстеження (Права)",
variable=self.track_right).grid(row=2, column=1, padx=5, pady=2, sticky="w")

        ttk.Checkbutton(options_frame, text="Відстеження (Глибина)",
variable=self.track_depth).grid(row=3, column=1, padx=5, pady=2, sticky="w")



        ttk.Button(actions_frame, text="Запустити реальний час",
command=self.start_realtime).grid(row=0, column=0, columnspan=2, padx=5, pady=5,
sticky="ew")



        tab.grid_rowconfigure(0, weight=1)

        tab.grid_rowconfigure(1, weight=1)
```

```
        tab.grid_rowconfigure(2, weight=1)
        tab.grid_columnconfigure(0, weight=1)
        input_frame.grid_columnconfigure(0, weight=1)
        input_frame.grid_columnconfigure(1, weight=1)
        options_frame.grid_columnconfigure(0, weight=1)
        options_frame.grid_columnconfigure(1, weight=1)
        actions_frame.grid_columnconfigure(0, weight=1)
        actions_frame.grid_columnconfigure(1, weight=1)

    def find_cameras(self):
        available_cameras = []
        for i in range(10):
            cap = cv2.VideoCapture(i)
            if cap.isOpened():
                available_cameras.append(i)
                cap.release()
        if available_cameras:
            self.left_camera_dropdown['values'] = available_cameras
            self.right_camera_dropdown['values'] = available_cameras
            self.status_var.set(f"Знайдено камери: {available_cameras}")
        else:
            self.status_var.set("Камери не знайдено.")
            messagebox.showwarning("Попередження", "Камери не знайдено.")

    def start_realtime(self):
        if self.is_creating:
            self.status_var.set("Процес уже запущено")
            return

        self.is_creating = True
        self.status_var.set("Запуск реального часу...")

        left_index = int(self.left_camera_var.get()) if self.left_camera_var.get() else
0
        right_index = int(self.right_camera_var.get()) if self.right_camera_var.get()
else 1
```

```
left_cap = cv2.VideoCapture(left_index)
right_cap = cv2.VideoCapture(right_index)

if not left_cap.isOpened() or not right_cap.isOpened():
    self.status_var.set("Не вдалося відкрити камери.")
    self.is_creating = False
    return

timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
mode = self.combo_var.get()
output_path = os.path.join(self.output_dir, f"realtime_{timestamp}.mp4")
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
fps = 10
frame_size = (640, 480)
video_writer = cv2.VideoWriter(output_path, fourcc, fps, frame_size)

start_time = datetime.now()

screen_width = self.root.winfo_screenwidth()
screen_height = self.root.winfo_screenheight()

while self.is_creating:
    ret_left, left_frame = left_cap.read()
    ret_right, right_frame = right_cap.read()

    if not ret_left or not ret_right:
        self.status_var.set("Не вдалося отримати кадри з камер.")
        break

    left_frame = cv2.resize(left_frame, frame_size)
    right_frame = cv2.resize(right_frame, frame_size)

    left_gray = cv2.cvtColor(left_frame, cv2.COLOR_BGR2GRAY)
    right_gray = cv2.cvtColor(right_frame, cv2.COLOR_BGR2GRAY)
```

```
            disparity_map = self.stereo.compute(left_gray,
right_gray).astype(np.float32) / 16.0

            if mode == "Чорно-білий":

                disparity_map_gray = cv2.normalize(disparity_map, None, 0, 255,
cv2.NORM_MINMAX, dtype=cv2.CV_8U)

                output_map = cv2.cvtColor(disparity_map_gray, cv2.COLOR_GRAY2BGR)

            else:

                disparity_map_normalized = cv2.normalize(disparity_map, None, 0, 255,
cv2.NORM_MINMAX)

                output_map = cv2.applyColorMap(np.uint8(disparity_map_normalized),
cv2.COLORMAP_JET)


            results_left = self.yolo_model(left_frame)

            results_right = self.yolo_model(right_frame)


            output_map_with_boxes = output_map.copy()

            for r_left, r_right in zip(results_left[0].boxes, results_right[0].boxes):

                if r_left.cls == r_right.cls:

                    xl1, yl1, xl2, yl2 = map(int, r_left.xyxy[0])

                    xr1, yr1, xr2, yr2 = map(int, r_right.xyxy[0])

                    x1 = (xl1 + xr1) // 2

                    y1 = (yl1 + yr1) // 2

                    x2 = (xl2 + xr2) // 2

                    y2 = (yl2 + yr2) // 2

                    if x2 > x1 and y2 > y1:

                        label = self.yolo_model.names[int(r_left.cls)]

                        cv2.rectangle(output_map_with_boxes, (x1, y1), (x2, y2), (0,
255, 0), 2)

                        cv2.putText(output_map_with_boxes, label, (x1, y1 - 10),

                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)


            if self.track_left.get():

                left_frame = results_left[0].plot()

            if self.track_right.get():

                right_frame = results_right[0].plot()

            if self.track_depth.get():
```

```
            output_map = output_map_with_boxes
        else:
            output_map = output_map_with_boxes

        video_writer.write(output_map)

        left_x = 10
        left_y = 10
        right_x = screen_width - frame_size[0] - 10
        right_y = 10
        center_x = screen_width // 2 - frame_size[0] // 2
        center_y = screen_height // 2 - frame_size[1] // 2 + 100

        if self.check_left.get() or self.track_left.get():
            cv2.imshow("Left Camera", left_frame)
            cv2.moveWindow("Left Camera", left_x, left_y)
        if self.check_right.get() or self.track_right.get():
            cv2.imshow("Right Camera", right_frame)
            cv2.moveWindow("Right Camera", right_x, right_y)
        if self.check_depth.get() or self.track_depth.get():
            cv2.imshow("Depth Map", output_map)
            cv2.moveWindow("Depth Map", center_x, center_y)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            self.is_creating = False
            break

        self.root.update()

    process_time = (datetime.now() - start_time).total_seconds()
    left_cap.release()
    right_cap.release()
    video_writer.release()
    cv2.destroyAllWindows()
```

```
        if self.is_creating:
            with self.db_connection.cursor() as cursor:
                left_frame_path = os.path.join(self.left_dir,
f"camera_left_{timestamp}.png")
                right_frame_path = os.path.join(self.right_dir,
f"camera_right_{timestamp}.png")
                cv2.imwrite(left_frame_path, left_frame)
                cv2.imwrite(right_frame_path, right_frame)

                with open(left_frame_path, 'rb') as left_file, open(right_frame_path,
'rb') as right_file:
                    left_data = left_file.read()
                    right_data = right_file.read()

                cursor.execute(
                    """
                    INSERT INTO cameras (left_camera, right_camera, created_at)
                    VALUES (%s, %s, %s) RETURNING id
                    """,
                    (left_data, right_data, datetime.now())
                )
                camera_id = cursor.fetchone()[0]

                with open(output_path, 'rb') as depth_file:
                    depth_data = depth_file.read()

                cursor.execute(
                    """
                    INSERT INTO depth_maps (left_camera_id, right_camera_id, depth_map,
process_time, report, created_at)
                    VALUES (%s, %s, %s, %s, %s, %s)
                    """,
                    (camera_id, camera_id, depth_data, process_time, "Success",
datetime.now())
                )
                self.db_connection.commit()
            self.status_var.set("Реальний час завершено та збережено в БД")
```

```
        else:
            self.status_var.set("Реальний час перервано")
            if os.path.exists(output_path):
                os.remove(output_path)
        self.is_creating = False

    def add_left_images(self):
        folder = filedialog.askdirectory(title="Виберіть папку з лівими зображеннями")
        if folder:
            self.left_images = [
                os.path.join(folder, f) for f in os.listdir(folder)
                if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp'))
            ]
            if self.left_images:
                self.status_var.set(f"Вибрано {len(self.left_images)} лівих зображень")
                print(f"Знайдено {len(self.left_images)} лівих зображень")

    def add_right_images(self):
        folder = filedialog.askdirectory(title="Виберіть папку з правими зображеннями")
        if folder:
            self.right_images = [
                os.path.join(folder, f) for f in os.listdir(folder)
                if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp'))
            ]
            if self.right_images:
                self.status_var.set(f"Вибрано {len(self.right_images)} правих
зображень")
                print(f"Знайдено {len(self.right_images)} правих зображень")
                if self.left_images and len(self.left_images) ==
len(self.right_images):
                    self.save_images_to_db(self.left_images, self.right_images)
                else:
                    self.status_var.set("Помилка: виберіть однакову кількість
зображень")

    def add_left_video(self):
```

```
        file = filedialog.askopenfilename(title="Виберіть ліве відео",
filetypes=[("Video files", "*.mp4 *.avi *.mov")])
        if file:
            self.left_video = file
            self.status_var.set(f"Ліва камера: {os.path.basename(file)}")

    def add_right_video(self):
        file = filedialog.askopenfilename(title="Виберіть праве відео",
filetypes=[("Video files", "*.mp4 *.avi *.mov")])
        if file:
            self.right_video = file
            self.status_var.set(f"Права камера: {os.path.basename(file)}")

    def toggle_settings(self):
        if self.settings_window and self.settings_window.winfo_exists():
            self.settings_window.destroy()
            self.status_var.set("Вікно параметрів закрито")
        else:
            self.settings_window = tk.Toplevel(self.root)
            main_x = self.root.winfo_x()
            main_y = self.root.winfo_y()
            self.settings_window.geometry(f"300x400+{main_x - 320}+{main_y}")
            SettingsInterface(self.settings_window, self)
            self.status_var.set("Відкрито вікно параметрів")

    def create_depth_map(self):
            if not self.db_connection:
                self.status_var.set("Помилка: немає підключення до БД")
                return

            if self.is_creating:
                self.status_var.set("Процес створення вже запущено")
                return

            self.is_creating = True
            self.status_var.set("Створення карт глибин...")
```

```
            cv2.destroyAllWindows()

            current_tab = self.notebook.tab(self.notebook.select(), "text")
            if current_tab == "Зображення":
                if not self.ensure_db_connection():
                    self.status_var.set("Помилка: немає підключення до БД")
                    self.is_creating = False
                    return

                try:
                    with self.db_connection.cursor() as cursor:
                        cursor.execute("""
                            SELECT id, left_image, right_image
                            FROM images
                            ORDER BY upload_time ASC
                        """)
                        image_pairs = cursor.fetchall()

                        if not image_pairs:
                            self.status_var.set("Немає зображень у базі даних")
                            self.is_creating = False
                            return

                        print(f"Знайдено {len(image_pairs)} пар зображень у БД")
                        start_time = datetime.now()
                        for idx, (image_id, left_img_data, right_img_data) in
enumerate(image_pairs):
                            if not self.is_creating:
                                break

                            left_img = cv2.imdecode(np.frombuffer(left_img_data,
np.uint8), cv2.IMREAD_COLOR)
                            right_img = cv2.imdecode(np.frombuffer(right_img_data,
np.uint8), cv2.IMREAD_COLOR)

                            if left_img is None or right_img is None:
```

```
                                print(f"Помилка: не вдалося декодувати зображення для
ID {image_id}")
                                continue

                            left_gray = cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY)
                            right_gray = cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY)

                            disparity_map = self.stereo.compute(left_gray,
right_gray).astype(np.float32) / 16.0
                            disparity_map[disparity_map == -1] =
np.min(disparity_map[disparity_map != -1])

                            mode = self.combo_var.get()
                            disparity_map_normalized = cv2.normalize(disparity_map,
None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8U)

                            if mode == "Чорно-білий":
                                output_map = cv2.cvtColor(disparity_map_normalized,
cv2.COLOR_GRAY2BGR)
                            else:
                                output_map =
cv2.applyColorMap(disparity_map_normalized, cv2.COLORMAP_JET)

                            # Обчислення YOLO, якщо хоча б один із чекбоксів
відстеження увімкнений
                            results_left = None
                            results_right = None
                            if self.track_left.get() or self.track_right.get() or
self.track_depth.get():
                                results_left = self.yolo_model(left_img)
                                results_right = self.yolo_model(right_img)

                            # Копії зображень для відображення з можливими bounding
boxes
                            display_left = left_img.copy()
                            display_right = right_img.copy()
                            display_depth = output_map.copy()
```

```
                            # Відстеження на лівому зображенні
                            if self.track_left.get() and results_left:
                                print(f"Виявлено об’єктів на лівому зображенні:
{len(results_left[0].boxes)}")
                                for r in results_left[0].boxes:
                                    x1, y1, x2, y2 = map(int, r.xyxy[0])
                                    if x2 > x1 and y2 > y1:  # Перевірка коректності
координат
                                        label = self.yolo_model.names[int(r.cls)]
                                        cv2.rectangle(display_left, (x1, y1), (x2, y2),
(0, 255, 0), 2)
                                        cv2.putText(display_left, label, (x1, y1 - 10),
                                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
255, 0), 2)

                            # Відстеження на правому зображенні
                            if self.track_right.get() and results_right:
                                print(f"Виявлено об’єктів на правому зображенні:
{len(results_right[0].boxes)}")
                                for r in results_right[0].boxes:
                                    x1, y1, x2, y2 = map(int, r.xyxy[0])
                                    if x2 > x1 and y2 > y1:  # Перевірка коректності
координат
                                        label = self.yolo_model.names[int(r.cls)]
                                        cv2.rectangle(display_right, (x1, y1), (x2,
y2), (0, 255, 0), 2)
                                        cv2.putText(display_right, label, (x1, y1 -
10),
                                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
255, 0), 2)

                            # Відстеження на карті глибини (як і раніше)
                            if self.track_depth.get() and results_left and
results_right:
                                print(f"Виявлено об’єктів для порівняння на карті
глибини: {len(results_left[0].boxes)}")
                                for r_left, r_right in zip(results_left[0].boxes,
results_right[0].boxes):
                                    if r_left.cls == r_right.cls:
                                        xl1, yl1, xl2, yl2 = map(int, r_left.xyxy[0])
```

```
                                    xr1, yr1, xr2, yr2 = map(int, r_right.xyxy[0])
                                    x1 = (xl1 + xr1) // 2
                                    y1 = (yl1 + yr1) // 2
                                    x2 = (xl2 + xr2) // 2
                                    y2 = (yl2 + yr2) // 2
                                    if x2 > x1 and y2 > y1:
                                        label =
self.yolo_model.names[int(r_left.cls)]
                                        cv2.rectangle(display_depth, (x1, y1), (x2,
y2), (0, 255, 0), 2)
                                        cv2.putText(display_depth, label, (x1, y1 -
10),
                                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(0, 255, 0), 2)

                            # Збереження карти глибини
                            timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
                            base_filename = f"image_{image_id}"
                            filename = f"disparity_{base_filename}_{'black' if mode ==
'Чорно-білий' else 'color'}_{timestamp}.png"
                            output_path = os.path.join(self.output_dir, filename)
                            cv2.imwrite(output_path, display_depth)

                            process_time = (datetime.now() -
start_time).total_seconds()
                            if not self.save_image_depth_map_to_db(image_id, image_id,
output_path, process_time):
                                print(f"Помилка: не вдалося зберегти карту глибини для
зображення ID {image_id}")

                            # Відображення
                            screen_width = self.root.winfo_screenwidth()
                            screen_height = self.root.winfo_screenheight()

                            left_x = 10
                            left_y = 10
                            right_x = screen_width - display_right.shape[1] - 10
                            right_y = 10
```

```
                            center_x = screen_width // 2 - display_depth.shape[1] // 2
                            center_y = screen_height // 2 - display_depth.shape[0] // 2
+ 100

                            if self.check_depth.get() or self.track_depth.get():
                                cv2.imshow("Depth Map", display_depth)
                                cv2.moveWindow("Depth Map", center_x, center_y)
                            if self.check_left.get() or self.track_left.get():
                                cv2.imshow("Left Image", display_left)
                                cv2.moveWindow("Left Image", left_x, left_y)
                            if self.check_right.get() or self.track_right.get():
                                cv2.imshow("Right Image", display_right)
                                cv2.moveWindow("Right Image", right_x, right_y)

                            if cv2.waitKey(1000) & 0xFF == ord('q'):
                                self.is_creating = False
                                break

                            self.root.after(0, lambda: self.status_var.set(f"Оброблено
{idx + 1}/{len(image_pairs)} зображень"))

                        if self.is_creating:
                            self.root.after(0, lambda: self.status_var.set("Карти
глибин створено та збережено в БД"))
                        else:
                            self.root.after(0, lambda: self.status_var.set("Обробку
перервано"))

                except psycopg2.Error as e:
                    self.root.after(0, lambda msg=str(e): self.status_var.set(f"Помилка
БД: {msg}"))
                finally:
                    self.is_creating = False
                    cv2.destroyAllWindows()

            elif current_tab == "Відео":
                if not self.left_video or not self.right_video:
```

```
                    self.status_var.set("Помилка: виберіть обидва відеофайли")
                    self.is_creating = False
                    return

                # Збереження вхідних відео в таблицю videos
                video_id = self.save_videos_to_db(self.left_video, self.right_video)
                if video_id is None:
                    self.status_var.set("Помилка: не вдалося зберегти відео в БД")
                    self.is_creating = False
                    return

                # Відкриття лівого та правого відео
                left_cap = cv2.VideoCapture(self.left_video)
                right_cap = cv2.VideoCapture(self.right_video)

                if not left_cap.isOpened() or not right_cap.isOpened():
                    self.status_var.set("Помилка: не вдалося відкрити відеофайли")
                    self.is_creating = False
                    left_cap.release()
                    right_cap.release()
                    return

                # Налаштування вихідного відео карти глибини
                timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
                mode = self.combo_var.get()
                output_filename = f"disparity_video_{'black' if mode == 'Чорно-білий'
else 'color'}_{timestamp}.mp4"
                output_path = os.path.join(self.output_dir, output_filename)
                fourcc = cv2.VideoWriter_fourcc(*'mp4v')
                fps = min(left_cap.get(cv2.CAP_PROP_FPS),
right_cap.get(cv2.CAP_PROP_FPS)) or 10
                frame_size = (int(left_cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
int(left_cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))
                video_writer = cv2.VideoWriter(output_path, fourcc, fps, frame_size)

                if not video_writer.isOpened():
```

```
                    self.status_var.set("Помилка: не вдалося ініціалізувати
VideoWriter")
                    self.is_creating = False
                    left_cap.release()
                    right_cap.release()
                    return

                start_time = datetime.now()
                frame_count = 0
                total_frames = min(
                    int(left_cap.get(cv2.CAP_PROP_FRAME_COUNT)),
                    int(right_cap.get(cv2.CAP_PROP_FRAME_COUNT))
                )

                screen_width = self.root.winfo_screenwidth()
                screen_height = self.root.winfo_screenheight()
                left_x = 10
                left_y = 10
                right_x = screen_width - frame_size[0] - 10
                right_y = 10
                center_x = screen_width // 2 - frame_size[0] // 2
                center_y = screen_height // 2 - frame_size[1] // 2 + 100

                while left_cap.isOpened() and right_cap.isOpened() and
self.is_creating:
                    ret_left, left_frame = left_cap.read()
                    ret_right, right_frame = right_cap.read()

                    if not ret_left or not ret_right:
                        print(f"[INFO] Кінець відео після {frame_count} кадрів")
                        break

                    frame_count += 1
                    left_frame = cv2.resize(left_frame, frame_size)
                    right_frame = cv2.resize(right_frame, frame_size)
```

```
                    left_gray = cv2.cvtColor(left_frame, cv2.COLOR_BGR2GRAY)
                    right_gray = cv2.cvtColor(right_frame, cv2.COLOR_BGR2GRAY)

                    disparity_map = self.stereo.compute(left_gray,
right_gray).astype(np.float32) / 16.0
                    if disparity_map is None:
                        print("Помилка: disparity_map повернув None")
                        continue
                    disparity_map[disparity_map == -1] =
np.min(disparity_map[disparity_map != -1])

                    mode = self.combo_var.get()
                    disparity_map_normalized = cv2.normalize(disparity_map, None, 0,
255, cv2.NORM_MINMAX, dtype=cv2.CV_8U)
                    if mode == "Чорно-білий":
                        output_map = cv2.cvtColor(disparity_map_normalized,
cv2.COLOR_GRAY2BGR)
                    else:
                        output_map = cv2.applyColorMap(disparity_map_normalized,
cv2.COLORMAP_JET)

                    results_left = None
                    results_right = None
                    if self.track_left.get() or self.track_right.get() or
self.track_depth.get():
                        results_left = self.yolo_model(left_frame)
                        results_right = self.yolo_model(right_frame)

                    display_left = left_frame.copy()
                    display_right = right_frame.copy()
                    display_depth = output_map.copy()

                    if self.track_left.get() and results_left:
                        print("Відстеження на лівому відео увімкнено")
                        display_left = results_left[0].plot()

                    if self.track_right.get() and results_right:
```

```
                    print("Відстеження на правому відео увімкнено")
                    display_right = results_right[0].plot()

                if self.track_depth.get() and results_left and results_right:
                    print("Відстеження на карті глибини увімкнено")
                    for r_left, r_right in zip(results_left[0].boxes,
results_right[0].boxes):
                        if r_left.cls == r_right.cls:
                            xl1, yl1, xl2, yl2 = map(int, r_left.xyxy[0])
                            xr1, yr1, xr2, yr2 = map(int, r_right.xyxy[0])
                            x1 = (xl1 + xr1) // 2
                            y1 = (yl1 + yr1) // 2
                            x2 = (xl2 + xr2) // 2
                            y2 = (yl2 + yr2) // 2
                            if x2 > x1 and y2 > y1:
                                label = self.yolo_model.names[int(r_left.cls)]
                                cv2.rectangle(display_depth, (x1, y1), (x2, y2),
(0, 255, 0), 2)
                                cv2.putText(display_depth, label, (x1, y1 - 10),
                                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255,
0), 2)

                depth_map_to_save = display_depth if self.track_depth.get() else
output_map
                video_writer.write(depth_map_to_save)

                if self.check_depth.get() or self.track_depth.get():
                    cv2.imshow("Depth Map", display_depth)
                    cv2.moveWindow("Depth Map", center_x, center_y)
                if self.check_left.get() or self.track_left.get():
                    cv2.imshow("Left Video", display_left)
                    cv2.moveWindow("Left Video", left_x, left_y)
                if self.check_right.get() or self.track_right.get():
                    cv2.imshow("Right Video", display_right)
                    cv2.moveWindow("Right Video", right_x, right_y)

                if cv2.waitKey(1) & 0xFF == ord('q'):
```

```
                        self.is_creating = False
                        break

                    self.root.after(0, lambda: self.status_var.set(f"Оброблено
{frame_count}/{total_frames} кадрів"))
                    self.root.update()

                process_time = (datetime.now() - start_time).total_seconds()
                video_writer.release()
                left_cap.release()
                right_cap.release()

                if self.is_creating and frame_count > 0:
                    if self.save_depth_map_to_db(video_id, video_id, output_path,
process_time):
                        self.root.after(0, lambda: self.status_var.set(f"Карта глибини
для відео створена та збережена: {output_path}"))
                    else:
                        self.root.after(0, lambda: self.status_var.set("Помилка: не
вдалося зберегти карту глибини в БД"))
                        if os.path.exists(output_path):
                            max_attempts = 5
                            for attempt in range(max_attempts):
                                try:
                                    os.remove(output_path)
                                    break
                                except PermissionError:
                                    if attempt == max_attempts - 1:
                                        self.status_var.set("Помилка: не вдалося
видалити вихідний файл через блокування")
                else:
                    self.root.after(0, lambda: self.status_var.set("Обробку перервано
або не створено кадрів"))
                    if os.path.exists(output_path):
                        max_attempts = 5
                        for attempt in range(max_attempts):
                            try:
```

```
                                os.remove(output_path)
                                break
                            except PermissionError:
                                if attempt == max_attempts - 1:
                                    self.status_var.set("Помилка: не вдалося видалити
вихідний файл через блокування")

            elif current_tab == "Реальний час":
                self.start_realtime()

            self.is_creating = False
            cv2.destroyAllWindows()

    def process_video_frame(self):
        if not self.is_creating:
            self.finish_video_processing()
            return

        ret_left, left_frame = self.left_cap.read()
        ret_right, right_frame = self.right_cap.read()

        if not ret_left or not ret_right:
            self.finish_video_processing()
            return

        self.frame_count += 1
        left_frame = cv2.resize(left_frame, self.frame_size)
        right_frame = cv2.resize(right_frame, self.frame_size)

        left_gray = cv2.cvtColor(left_frame, cv2.COLOR_BGR2GRAY)
        right_gray = cv2.cvtColor(right_frame, cv2.COLOR_BGR2GRAY)

        disparity_map = self.stereo.compute(left_gray, right_gray).astype(np.float32) /
16.0
        mode = self.combo_var.get()
        if mode == "Чорно-білий":
```

```
            disparity_map_gray = cv2.normalize(disparity_map, None, 0, 255,
cv2.NORM_MINMAX, dtype=cv2.CV_8U)
            output_map = cv2.cvtColor(disparity_map_gray, cv2.COLOR_GRAY2BGR)
        else:
            disparity_map_normalized = cv2.normalize(disparity_map, None, 0, 255,
cv2.NORM_MINMAX)
            output_map = cv2.applyColorMap(np.uint8(disparity_map_normalized),
cv2.COLORMAP_JET)

        results_left = None
        results_right = None
        if self.track_left.get() or self.track_right.get() or self.track_depth.get():
            results_left = self.yolo_model(left_frame)
            results_right = self.yolo_model(right_frame)

        display_left = left_frame.copy()
        display_right = right_frame.copy()
        display_depth = output_map.copy()

        if self.track_depth.get() and results_left and results_right:
            for r_left, r_right in zip(results_left[0].boxes, results_right[0].boxes):
                if r_left.cls == r_right.cls:
                    xl1, yl1, xl2, yl2 = map(int, r_left.xyxy[0])
                    xr1, yr1, xr2, yr2 = map(int, r_right.xyxy[0])
                    x1 = (xl1 + xr1) // 2
                    y1 = (yl1 + yr1) // 2
                    x2 = (xl2 + xr2) // 2
                    y2 = (yl2 + yr2) // 2
                    if x2 > x1 and y2 > y1:
                        label = self.yolo_model.names[int(r_left.cls)]
                        cv2.rectangle(display_depth, (x1, y1), (x2, y2), (0, 255, 0),
2)
                        cv2.putText(display_depth, label, (x1, y1 - 10),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

        if self.track_left.get() and results_left:
            display_left = results_left[0].plot()
```

```
        if self.track_right.get() and results_right:
            display_right = results_right[0].plot()

        self.video_writer.write(display_depth if self.track_depth.get() else
output_map)

        if self.check_left.get() or self.track_left.get():
            cv2.imshow("Left Video", display_left)
            cv2.moveWindow("Left Video", self.left_x, self.left_y)
            self.left_window_open = True
        elif self.left_window_open:
            cv2.destroyWindow("Left Video")
            self.left_window_open = False

        if self.check_right.get() or self.track_right.get():
            cv2.imshow("Right Video", display_right)
            cv2.moveWindow("Right Video", self.right_x, self.right_y)
            self.right_window_open = True
        elif self.right_window_open:
            cv2.destroyWindow("Right Video")
            self.right_window_open = False

        if self.check_depth.get() or self.track_depth.get():
            cv2.imshow("Depth Map", display_depth)
            cv2.moveWindow("Depth Map", self.center_x, self.center_y)
            self.depth_window_open = True
        elif self.depth_window_open:
            cv2.destroyWindow("Depth Map")
            self.depth_window_open = False

        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            self.is_creating = False
            self.finish_video_processing()
            return
```

```
        self.status_var.set(f"Оброблено {self.frame_count}/{self.total_frames} кадрів")

        self.root.after(33, self.process_video_frame)

        self.root.update()

    def finish_video_processing(self):
        process_time = (datetime.now() - self.start_time).total_seconds()
        self.left_cap.release()
        self.right_cap.release()
        self.video_writer.release()
        cv2.destroyAllWindows()

        if self.is_creating:
            try:
                with self.db_connection.cursor() as cursor:
                    with open(self.output_path, 'rb') as depth_file:
                        depth_data = depth_file.read()

                    cursor.execute(
                        """
                        INSERT INTO depth_maps (left_video_id, right_video_id,
depth_map, process_time, report, created_at)
                        VALUES (%s, %s, %s, %s, %s, %s)
                        """,
                        (self.video_id, self.video_id, depth_data, process_time,
"Success", datetime.now())
                    )
                    self.db_connection.commit()
                self.status_var.set("Карта глибини для відео створена та збережена в
БД")
            except psycopg2.Error as e:
                self.status_var.set(f"Помилка БД при збереженні карти глибини: {e}")
                if os.path.exists(self.output_path):
                    os.remove(self.output_path)
```

```
            else:
                self.status_var.set("Обробку відео перервано")
                if os.path.exists(self.output_path):
                    os.remove(self.output_path)

            self.is_creating = False

        def view_depth_maps(self):
            if not self.db_connection:
                self.status_var.set("Помилка: немає підключення до БД")
                return

            current_tab = self.notebook.tab(self.notebook.select(), "text")
            if current_tab == "Зображення":
                # Логіка для зображень залишається без змін
                try:
                    with self.db_connection.cursor() as cursor:
                        cursor.execute("""
                            SELECT depth_map
                            FROM depth_maps
                            WHERE left_image_id IS NOT NULL
                            ORDER BY created_at DESC
                        """)
                        depth_maps = cursor.fetchall()

                        if not depth_maps:
                            self.status_var.set("Немає збережених карт глибин")
                            return

                        self.is_viewing = True
                        current_index = 0

                        def show_image(index):
                            if not self.is_viewing or index < 0 or index >=
len(depth_maps):
```

```
                    return
                depth_data = depth_maps[index][0]
                depth_nparr = np.frombuffer(depth_data, np.uint8)
                depth_map = cv2.imdecode(depth_nparr, cv2.IMREAD_COLOR)

                cv2.destroyAllWindows()
                screen_width = self.root.winfo_screenwidth()
                screen_height = self.root.winfo_screenheight()
                center_x = screen_width // 2 - depth_map.shape[1] // 2
                center_y = screen_height // 2 - depth_map.shape[0] // 2

                if depth_map is not None:
                    cv2.imshow("Depth Map", depth_map)
                    cv2.moveWindow("Depth Map", center_x, center_y)

            show_image(current_index)

            while self.is_viewing:
                key = cv2.waitKeyEx(100)
                if key == 2424832:  # Стрілка вліво
                    current_index = max(current_index - 1, 0)
                    show_image(current_index)
                elif key == 2555904:  # Стрілка вправо
                    current_index = min(current_index + 1, len(depth_maps) - 1)
                    show_image(current_index)
                elif key == 27:  # Esc
                    self.is_viewing = False
                    cv2.destroyAllWindows()
                    break

            self.status_var.set("Перегляд завершено")
    except psycopg2.Error as e:
        self.status_var.set(f"Помилка при отриманні карт глибин: {e}")
    finally:
        self.is_viewing = False
```

```
        elif current_tab in ["Відео", "Реальний час"]:
            try:
                with self.db_connection.cursor() as cursor:
                    cursor.execute("""
                        SELECT id, depth_map, created_at
                        FROM depth_maps
                        WHERE left_video_id IS NOT NULL AND right_video_id IS NOT NULL
                        ORDER BY created_at DESC
                    """)
                    depth_maps = cursor.fetchall()

                    if not depth_maps:
                        self.status_var.set("Немає збережених карт глибини для відео")
                        return

                    video_selection_window = tk.Toplevel(self.root)
                    video_selection_window.title("Виберіть карту глибини")
                    video_selection_window.geometry("400x300")

                    video_listbox = tk.Listbox(video_selection_window, width=50,
height=10)
                    video_listbox.pack(pady=10)

                    depth_map_dict = {}
                    for idx, (depth_map_id, depth_map_data, created_at) in
enumerate(depth_maps):
                        depth_map_name = f"Карта глибини від {created_at.strftime('%Y-
%m-%d %H:%M:%S')}"
                        video_listbox.insert(tk.END, depth_map_name)
                        depth_map_dict[depth_map_name] = depth_map_data

                    def play_selected_depth_map():
                        selected = video_listbox.curselection()
                        if not selected:
                            messagebox.showwarning("Попередження", "Виберіть карту
глибини зі списку")
```

```
                            return
                        depth_map_name = video_listbox.get(selected[0])
                        depth_map_data = depth_map_dict[depth_map_name]

                        temp_path = os.path.join(self.output_dir,
f"temp_depth_map_{uuid.uuid4()}.mp4")
                        try:
                            with open(temp_path, 'wb') as temp_file:
                                temp_file.write(depth_map_data)

                            cap = cv2.VideoCapture(temp_path)
                            if not cap.isOpened():
                                self.status_var.set("Помилка: не вдалося відкрити відео
карти глибини")
                                if os.path.exists(temp_path):
                                    os.remove(temp_path)
                                return

                            self.is_viewing = True
                            screen_width = self.root.winfo_screenwidth()
                            screen_height = self.root.winfo_screenheight()

                            while cap.isOpened() and self.is_viewing:
                                ret, frame = cap.read()
                                if not ret:
                                    break
                                frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                                frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                                if frame_width == 0 or frame_height == 0:
                                    self.status_var.set("Помилка: некоректні розміри
кадру")
                                    break
                                center_x = screen_width // 2 - frame_width // 2
                                center_y = screen_height // 2 - frame_height // 2
                                cv2.imshow("Depth Map Video", frame)
                                cv2.moveWindow("Depth Map Video", center_x, center_y)
```

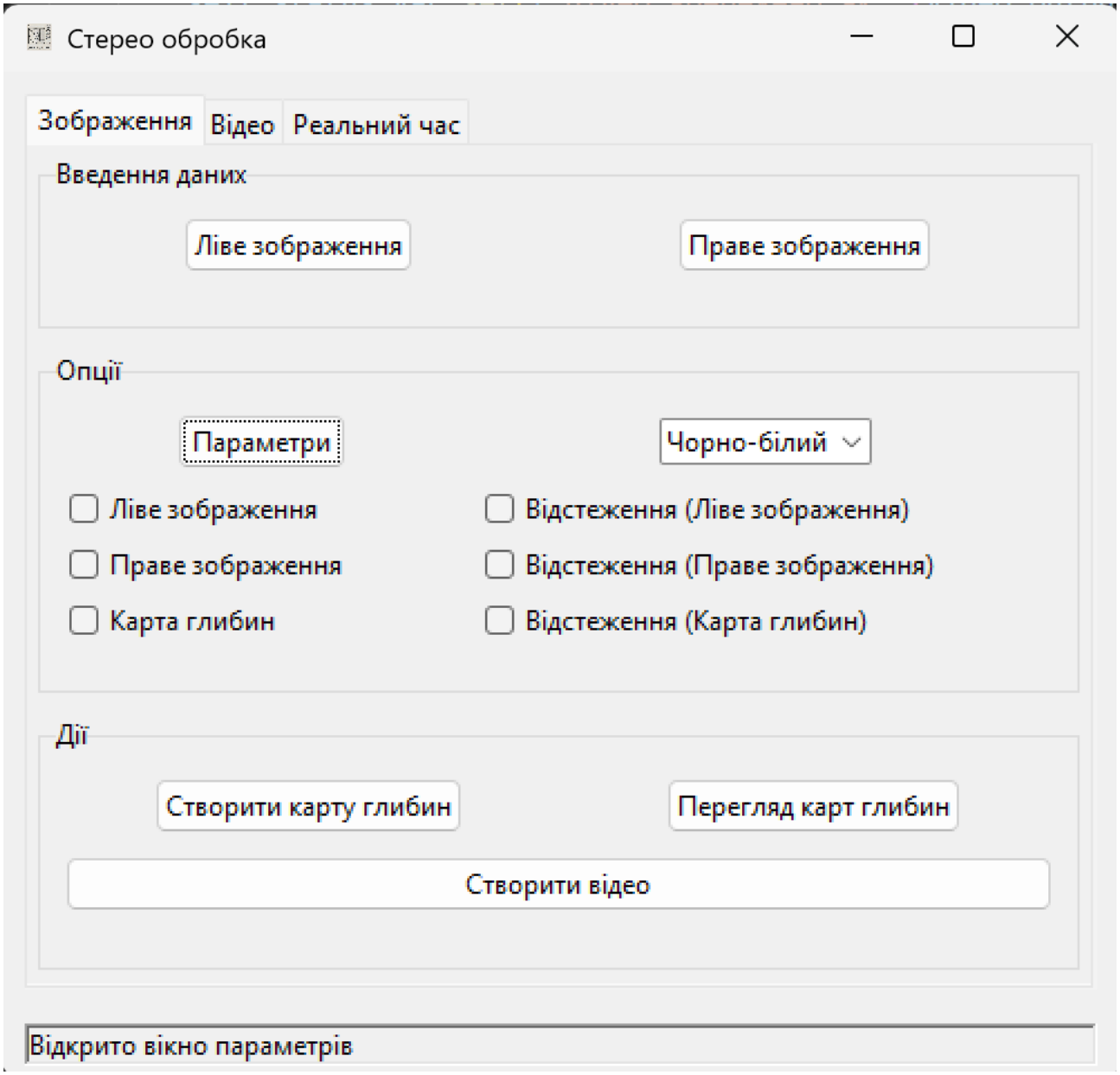
```
                                if cv2.waitKey(33) & 0xFF == ord('q'):
                                    self.is_viewing = False
                                    break
                                self.root.update()

                            cap.release()
                            cv2.destroyAllWindows()
                            if os.path.exists(temp_path):
                                os.remove(temp_path)
                            self.status_var.set("Перегляд карти глибини завершено")
                            video_selection_window.destroy()

                        except Exception as e:
                            self.status_var.set(f"Помилка відтворення карти глибини: {e}")
                            if os.path.exists(temp_path):
                                os.remove(temp_path)

                    ttk.Button(video_selection_window, text="Відтворити", command=play_selected_depth_map).pack(pady=5)
                    ttk.Button(video_selection_window, text="Закрити", command=lambda: [setattr(self, 'is_viewing', False), video_selection_window.destroy()]).pack(pady=5)

            except psycopg2.Error as e:
                self.status_var.set(f"Помилка при отриманні карт глибини: {e}")
            finally:
                self.is_viewing = False

    def create_video(self):
        if not self.db_connection:
            self.status_var.set("Помилка: немає підключення до БД")
            return

        if self.is_creating:
            self.status_var.set("Процес створення вже запущено")
            return
```

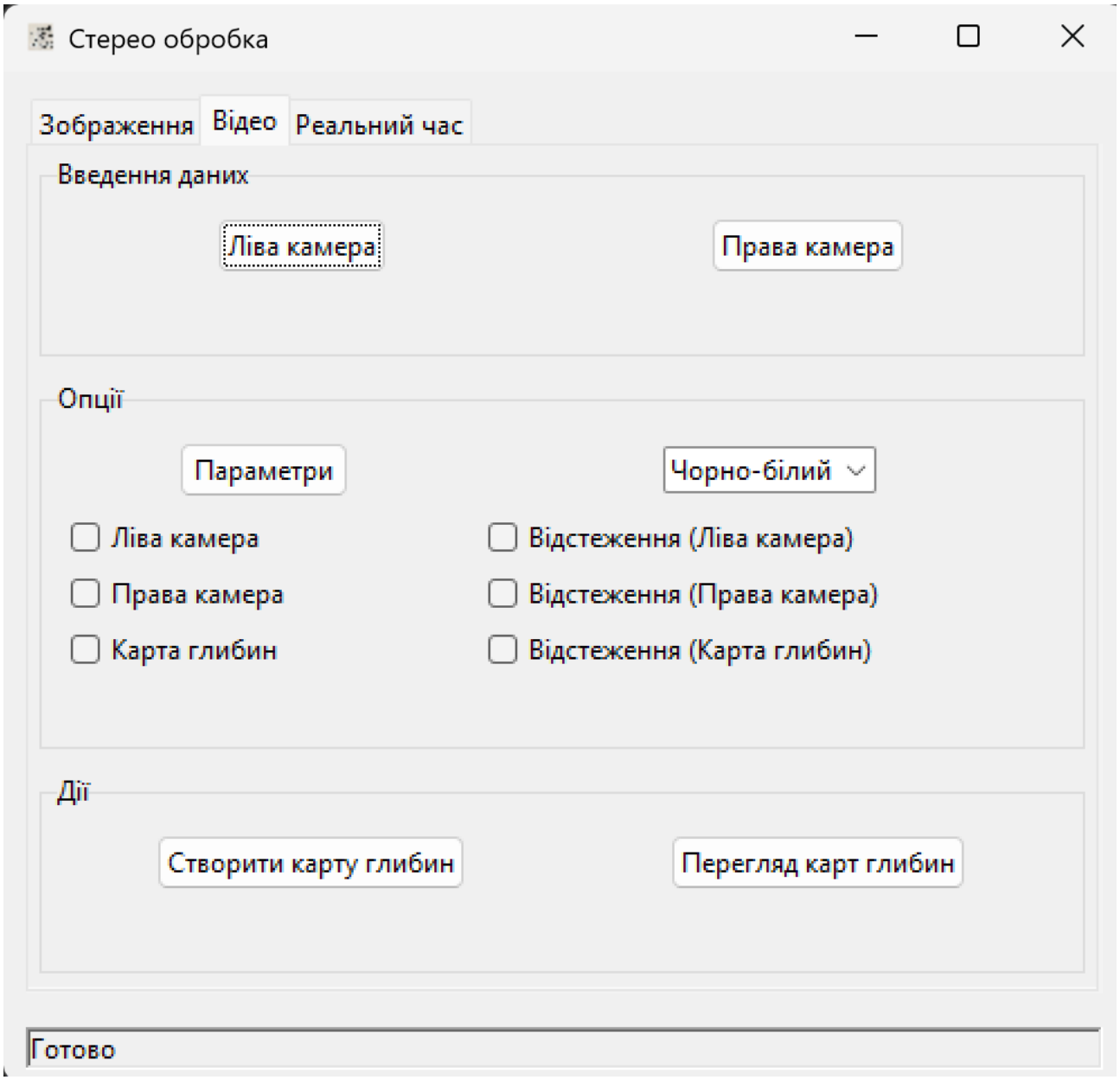
```
        self.is_creating = True
        self.status_var.set("Створення відео...")
        cv2.destroyAllWindows()

        try:
            with self.db_connection.cursor() as cursor:
                cursor.execute("""
                    SELECT depth_map
                    FROM depth_maps
                    WHERE left_image_id IS NOT NULL
                    ORDER BY created_at ASC
                """)
                depth_maps = cursor.fetchall()

                if not depth_maps:
                    self.status_var.set("Немає карт глибин для створення відео")
                    self.is_creating = False
                    return

                timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
                output_path = os.path.join(self.output_dir,
f"depth_video_{timestamp}.mp4")
                fourcc = cv2.VideoWriter_fourcc(*'mp4v')
                fps = 10
                frame_size = (640, 480)
                video_writer = cv2.VideoWriter(output_path, fourcc, fps, frame_size)

                start_time = datetime.now()
                for idx, (depth_data,) in enumerate(depth_maps):
                    if not self.is_creating:
                        break

                    depth_nparr = np.frombuffer(depth_data, np.uint8)
                    depth_map = cv2.imdecode(depth_nparr, cv2.IMREAD_COLOR)
```

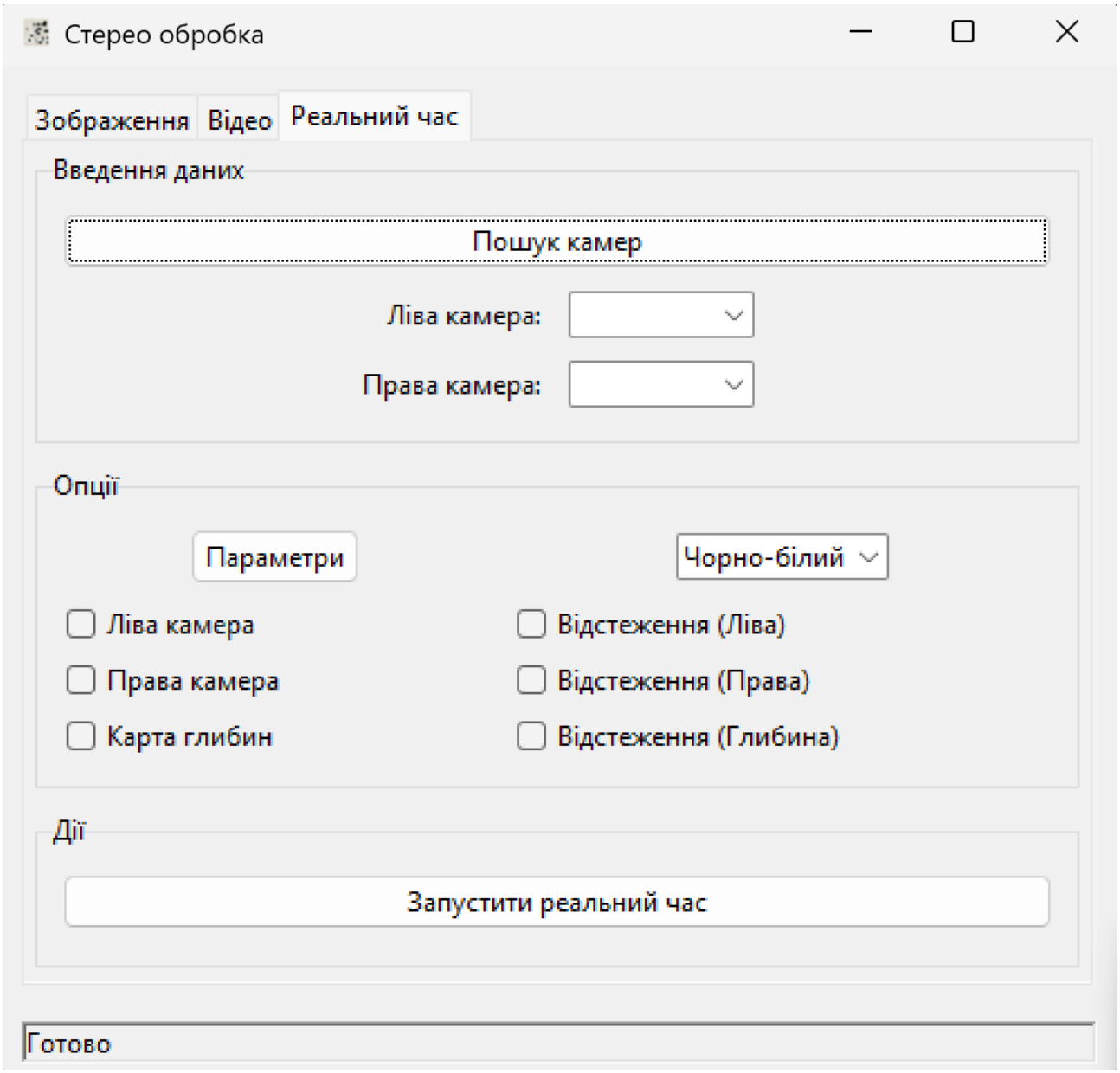
```
                    if depth_map is None:
                        continue

                    depth_map = cv2.resize(depth_map, frame_size)
                    video_writer.write(depth_map)

                    screen_width = self.root.winfo_screenwidth()
                    screen_height = self.root.winfo_screenheight()
                    center_x = screen_width // 2 - depth_map.shape[1] // 2
                    center_y = screen_height // 2 - depth_map.shape[0] // 2

                    cv2.imshow("Depth Video", depth_map)
                    cv2.moveWindow("Depth Video", center_x, center_y)

                    if cv2.waitKey(100) & 0xFF == ord('q'):
                        self.is_creating = False
                        break

                    self.root.after(0, lambda: self.status_var.set(f"Оброблено {idx +
1}/{len(depth_maps)} кадрів"))
                    self.root.update()

                process_time = (datetime.now() - start_time).total_seconds()
                video_writer.release()
                cv2.destroyAllWindows()

                if self.is_creating:
                    video_id = self.save_videos_to_db(output_path)
                    if video_id:
                        self.status_var.set("Відео створено та збережено в БД")
                    else:
                        self.status_var.set("Помилка при збереженні відео в БД")
                        if os.path.exists(output_path):
                            os.remove(output_path)
                else:
```

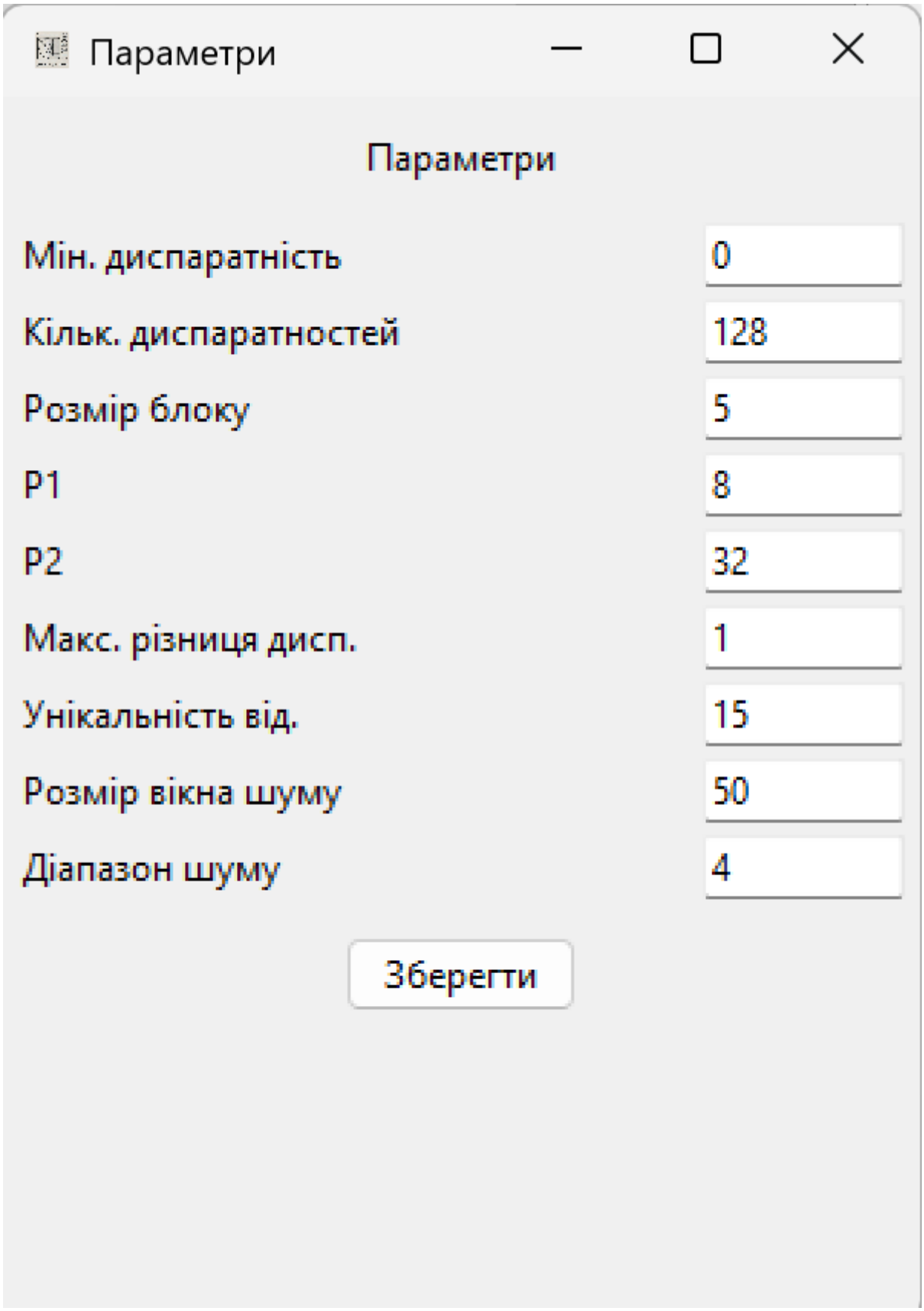
```
                    self.status_var.set("Створення відео перервано")
                    if os.path.exists(output_path):
                        os.remove(output_path)

        except psycopg2.Error as e:
            self.status_var.set(f"Помилка БД: {e}")
            if 'output_path' in locals() and os.path.exists(output_path):
                os.remove(output_path)
        except Exception as e:
            self.status_var.set(f"Помилка: {e}")
            if 'output_path' in locals() and os.path.exists(output_path):
                os.remove(output_path)
        finally:
            self.is_creating = False

    def on_closing(self):
        if self.is_creating:
            self.is_creating = False
        if self.is_viewing:
            self.is_viewing = False
        if self.db_connection:
            close_connection(self.db_connection)
        self.root.destroy()

if __name__ == "__main__":
    root = tk.Tk()
    app = MainInterface(root)
    root.mainloop()
```

Додаток В


Стерео обробка
Зображення
Відео
Реальний час
Введення даних
Ліве зображення
Праве зображення
Опції
Параметри
Чорно-білий
Ліве зображення
Праве зображення
Карта глибин
Відстеження (Ліве зображення)
Відстеження (Праве зображення)
Відстеження (Карта глибин)
Дії
Створити карту глибин
Перегляд карт глибин
Створити відео
Відкрито вікно параметрів


Стерео обробка
Зображення
Відео
Реальний час
Введення даних
Ліва камера
Права камера
Опції
Параметри
Чорно-білий
Ліва камера
Права камера
Карта глибин
Відстеження (Ліва камера)
Відстеження (Права камера)
Відстеження (Карта глибин)
Дії
Створити карту глибин
Перегляд карт глибин
Готово


Стерео обробка
Зображення
Відео
Реальний час
Введення даних
Пошук камер
Ліва камера:
Права камера:
Опції
Параметри
Чорно-білий
Ліва камера
Права камера
Карта глибин
Відстеження (Ліва)
Відстеження (Права)
Відстеження (Глибина)
Дії
Запустити реальний час
Готово


Параметри
Параметри
Мін. диспаратність
0
Кільк. диспаратностей
128
Розмір блоку
5
P1
8
P2
32
Макс. різниця дисп.
1
Унікальність від.
15
Розмір вікна шуму
50
Діапазон шуму
4
Зберегти

Додаток Г

1. Модуль стереозбігу

- **Опис**: Виконує стереозбіг за допомогою StereoSGBM і створює карту глибини.

Фрагмент коду:

```
import cv2
import numpy as np


def compute_depth_map(self, left_image, right_image):
    # Перетворення в сірошкальний формат
    left_gray = cv2.cvtColor(left_image, cv2.COLOR_BGR2GRAY)
    right_gray = cv2.cvtColor(right_image, cv2.COLOR_BGR2GRAY)

    # Ініціалізація StereoSGBM
    stereo = cv2.StereoSGBM_create(
        minDisparity=self.min_disparity,
        numDisparities=self.num_disparities,
        blockSize=self.block_size,
        P1=self.p1_factor * 3 * self.block_size ** 2,
        P2=self.p2_factor * 3 * self.block_size ** 2,
        disp12MaxDiff=self.disp12_max_diff,
        uniquenessRatio=self.uniqueness_ratio,
        speckleWindowSize=self.speckle_window_size,
        speckleRange=self.speckle_range
    )

    # Обчислення диспаратності
```

```
    disparity = stereo.compute(left_gray, right_gray).astype(np.float32) / 16.0

    # Нормалізація для чорно-білого режиму
    output_map = cv2.normalize(disparity, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)

    # Кольоровий режим (опціонально)
    if self.color_mode == 'color':
        output_map = cv2.applyColorMap(output_map, cv2.COLORMAP_JET)

return output_map
```

2. Модуль розпізнавання об’єктів

- **Опис**: Виконує детекцію об’єктів за допомогою YOLOv8 і накладає bounding box’и на карту глибини.

Фрагмент коду:

```
from ultralytics import YOLO
import cv2

def detect_objects(self, left_image, right_image, depth_map):
    # Ініціалізація YOLOv8
    self.yolo_model = YOLO('yolov8n.pt')

    # Детекція об’єктів
    results_left = self.yolo_model(left_image)
    results_right = self.yolo_model(right_image)

    output_map_with_boxes = depth_map.copy()
```

```
    # Узгодження детекцій між лівим і правим зображеннями
    for r_left, r_right in zip(results_left[0].boxes, results_right[0].boxes):
        if r_left.cls == r_right.cls:  # Перевірка однакового класу
            xl1, yl1, xl2, yl2 = map(int, r_left.xyxy[0])
            xr1, yr1, xr2, yr2 = map(int, r_right.xyxy[0])
            x1 = (xl1 + xr1) // 2
            y1 = (yl1 + yr1) // 2
            x2 = (xl2 + xr2) // 2
            y2 = (yl2 + yr2) // 2

            # Накладання bounding box і мітки
            cv2.rectangle(output_map_with_boxes, (x1, y1), (x2, y2), (0, 255, 0), 2)
            label = self.yolo_model.names[int(r_left.cls)]
            cv2.putText(output_map_with_boxes, label, (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    return output_map_with_boxes, results_left, results_right
```

3. Модуль обробки відеопотоків

- **Опис**: Обробляє послідовність кадрів із відео або веб-камер, створюючи динамічні карти глибини.

- **Фрагмент коду**:

```
import cv2

def process_video(self, left_video_path, right_video_path):
    cap_left = cv2.VideoCapture(left_video_path)
    cap_right = cv2.VideoCapture(right_video_path)
    out = cv2.VideoWriter('output.mp4', cv2.VideoWriter_fourcc(*'mp4v'), 10, (640, 480))
```

```
while cap_left.isOpened() and cap_right.isOpened():
    ret_left, left_frame = cap_left.read()
    ret_right, right_frame = cap_right.read()

    if not ret_left or not ret_right:
        break

    # Зміна розміру для оптимізації
    left_frame = cv2.resize(left_frame, (640, 480))
    right_frame = cv2.resize(right_frame, (640, 480))

    # Обчислення карти глибини
    depth_map = self.compute_depth_map(left_frame, right_frame)

    # Детекція об’єктів (якщо увімкнено)
    if self.track_depth.get():
        depth_map, _, _ = self.detect_objects(left_frame, right_frame, depth_map)

    # Відображення результатів
    if self.check_depth.get():
        cv2.imshow('Depth Map', depth_map)
        cv2.moveWindow('Depth Map', 200, 0)

    out.write(depth_map)

    # Переривання обробки
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

```
        cap_left.release()
        cap_right.release()
        out.release()
cv2.destroyAllWindows()
```

4. Модуль управління базою даних

- **Опис**: Зберігає зображення, відео, карти глибини та параметри в PostgreSQL.

```
import psycopg2
import cv2
import numpy as np

def save_to_database(self, left_image, right_image, depth_map, process_time, report):
    try:
        cursor = self.db_connection.cursor()

        # Збереження зображень
        left_image_bytea = self.image_to_bytea(left_image)
        right_image_bytea = self.image_to_bytea(right_image)
        cursor.execute(
            "INSERT INTO images (left_image, right_image) VALUES (%s, %s) RETURNING id",
            (left_image_bytea, right_image_bytea)
        )
        image_id = cursor.fetchone()[0]

        # Збереження карти глибини
        depth_map_bytea = self.image_to_bytea(depth_map)
        cursor.execute(
```

```
            "INSERT INTO depth_maps (left_image_id, right_image_id, depth_map, process_time, report)
            "
                "VALUES (%s, %s, %s, %s, %s)",
                (image_id, image_id, depth_map_bytea, process_time, report)
            )

            self.db_connection.commit()
        except Exception as e:
            self.db_connection.rollback()
            print(f"Database error: {e}")
        finally:
            cursor.close()

    def image_to_bytea(self, image):
        _, buffer = cv2.imencode('.png', image)
return psycopg2.Binary(buffer.tobytes())
```