НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ **Ігор БОЛБОТ**	_____ **Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система розпізнавання та відстеження рухомих об'єктів на основі згорткових нейронних мереж»

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми д.е.н., професор	_____ (підпис)	**Роман РУДЕНСЬКИЙ**
Керівник бакалаврської кваліфікаційної роботи асистент	_____ (підпис)	**Денис ВІННІЧУК**
Консультант д.т.н., професор	_____ (підпис)	**Віктор СЕМКО**
Виконав	_____ (підпис)	**Антон САЛІЙ**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

_____ Салію Антону Андрійовичу_____

(прізвище, ім'я, по батькові)

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи

«Система розпізнавання та відстеження рухомих об'єктів на базі згорткових нейронних мереж»
затверджена наказом від «06» Січня 2026 р. № 46 «С»
Термін подання завершеної роботи на кафедру «___» _____ 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): Відеопотік високої роздільної здатності у нестиснутому форматі; апаратна підсистема позиціонування на базі мікроконтролера ESP32, мікрокрокових драйверів TMC2208 та двигунів NEMA 17; архітектура згорткової нейронної мережі YOLOv8; математичний апарат фільтра Калмана; СУБД PostgreSQL.
Перелік питань, що підлягають дослідженню:
1. Системний аналіз предметної області комп'ютерного зору та кінематичного керування.
2. Проектування логічної та фізичної моделей бази даних для логіювання телеметрії.
3. Розробка прикладного програмного забезпечення: конвеєр детектування (YOLOv8 + DeepSORT), графічний інтерфейс користувача (PyQt6) та мікроконтролерний ПІД-регулятор (C++).
4. Комплексне тестування продуктивності системи та формування вимог до її впровадження.
Перелік графічних документів (за необхідності).

1. Діаграма прецедентів (UML Use Case Diagram).
2. Діаграма діяльності (UML Activity Diagram).
3. ER-діаграма логічної моделі бази даних.
4. Діаграма пакетів (UML Package Diagram).
5. Блок-схема алгоритму конвеєра комп'ютерного зору.
6. Діаграма розгортання (UML Deployment Diagram).

Дата видачі завдання «___» _____ 2026 р.

Керівник бакалаврської кваліфікаційної роботи	_____ (підпис)	**Денис ВІННІЧУК**
Консультант д.т.н., професор	_____ (підпис)	**Віктор СЕМКО**
Завдання взяв до виконання	_____ (підпис)	**Антон САЛІЙ**

ЗМІС

Т

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- **БпЛА** — Безпілотний літальний апарат
- **ЗНМ** — Згорткова нейронна мережа
- **ШІ** — Штучний інтелект
- **ППЗ** — Прикладне програмне забезпечення
- **СУБД** — Система управління базами даних
- **РСУБД** — Реляційна система управління базами даних
- **ТАК** — Теорія автоматичного керування
- **ПІД** — Пропорційно-інтегрально-диференціальний (регулятор)
- **ШІМ** — Широтно-імпульсна модуляція
- **ПК** — Персональний комп'ютер
- **ОС** — Операційна система
- **API** — Application Programming Interface (Інтерфейс прикладного програмування)
- **ACID** — Atomicity, Consistency, Isolation, Durability (Атомарність, Узгодженість, Ізольованість, Довговічність — вимоги до транзакцій у СУБД)
- **BMS** — Battery Management System (Система управління та балансування акумуляторної батареї)
- **CUDA** — Compute Unified Device Architecture (Архітектура паралельних обчислень від NVIDIA)
- **CPU** — Central Processing Unit (Центральний процесор)
- **DDL** — Data Definition Language (Мова визначення даних SQL)
- **ERD (ER)** — Entity-Relationship Diagram (Діаграма "сутність-зв'язок")
- **FPS** — Frames Per Second (Кількість кадрів за секунду)
- **GPU** — Graphical Processing Unit (Графічний процесор / відеокарта)
- **GUI** — Graphical User Interface (Графічний інтерфейс користувача)
- **HDMI** — High-Definition Multimedia Interface (Мультимедійний інтерфейс високої роздільної здатності)
- **IoU** — Intersection over Union (Метрика "перетин над об'єднанням" для обмежувальних рамок)
- **mAP** — Mean Average Precision (Середня середня точність детектування нейромережі)
- **MOT** — Multiple Object Tracking (Відстеження багатьох об'єктів у відеопотоці)
- **MVCC** — Multi-Version Concurrency Control (Багатоверсійне управління конкурентним доступом у СУБД)
- **PWM** — Pulse-Width Modulation (Широтно-імпульсна модуляція)
- **ReID** — Re-Identification (Нейромережева повторна ідентифікація об'єктів)

- **SDLC** — Software Development Life Cycle (Життєвий цикл розробки програмного забезпечення)
- **SORT** — Simple Online and Realtime Tracking (Базовий алгоритм трекінгу об'єктів)
- **UML** — Unified Modeling Language (Уніфікована мова моделювання програмних систем)
- **UART** — Universal Asynchronous Receiver-Transmitter (Універсальний асинхронний приймач-передавач / послідовний порт)
- **UVC** — USB Video Class (Стандартний протокол USB для відеопристроїв)
- **VRAM** — Video Random Access Memory (Відеопам'ять графічного прискорювача)
- **YOLO** — You Only Look Once (Архітектура одностадійних згорткових нейронних мереж для детектування)

ВСТУП

Актуальність теми. У сучасних системах автоматизації, безпеки та моніторингу простору дедалі гостріше постає питання ідентифікації та безперервного супроводження швидкісних рухомих цілей (зокрема малорозмірних об'єктів та БпЛА). Класичні методи відеоспостереження вимагають безперервного людського контролю, що в умовах високого навантаження призводить до втрати ефективності. Використання сучасних згорткових нейронних мереж дозволяє автоматизувати процес семантичного розпізнавання, проте критично важливим є створення гібридного комплексу, що здатний поєднати тензорні обчислення штучного інтелекту з мілісекундною реакцією систем кінематичного позиціонування. Розробка такого програмного забезпечення є актуальним науково-прикладним завданням.

Мета і завдання дослідження. Метою бакалаврської кваліфікаційної роботи є проектування та програмна реалізація гібридної апаратно-програмної системи для автоматичного розпізнавання, трекінгу та механічного супроводження рухомих об'єктів у відеопотоці високої роздільної здатності. Для досягнення поставленої мети було вирішено такі завдання:

1. Провести аналіз предметної області, обрати алгоритми комп'ютерного зору для роботи в реальному часі та побудувати UML-моделі системи.
2. Спроектувати логічну та фізичну моделі реляційної бази даних для збереження телеметрії та системних логів.
3. Програмно реалізувати підсистеми відеозахоплення, інференсу ЗНМ, багатопотокового графічного інтерфейсу та алгоритму ПІД-регулювання на мікроконтролері.
4. Провести тестування розробленого комплексу, оцінити його ефективність та сформувати рекомендації щодо впровадження.

Об’єкт дослідження – процес автоматизованого розпізнавання та кінематичного супроводження рухомих об'єктів у динамічному відеопотоці.

Предмет дослідження – алгоритми згорткових нейронних мереж, методи об'єктного трекінгу, архітектура баз даних та програмні засоби кінематичного керування (ПІД-регулювання).

Методи дослідження. У роботі використано методи комп’ютерного зору та машинного навчання (для детектування цілей), математичний апарат фільтра Калмана (для прогнозування траєкторій), методи теорії реляційних баз даних (для нормалізації схеми), а також методи теорії автоматичного керування (для позиціонування механіки).

Практичне значення отриманих результатів. Розроблене програмне забезпечення та оптимізована апаратна конфігурація є готовим прототипом, який може бути впроваджений як доступна альтернатива комерційним системам відеоспостереження на об'єктах критичної інфраструктури.

Структура роботи. Бакалаврська кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 72 сторінок , містить 6 рисунків, 1 таблицю. Список використаних джерел налічує 15 найменувань.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання та аналіз проблематики

Сучасний етап розвитку інформаційних технологій та систем автоматизації характеризується глобальним переходом від концепції пасивної фіксації даних до створення інтелектуальних систем активного моніторингу та реагування. Традиційні системи відеоспостереження (CCTV), які десятиліттями використовувалися для охорони інфраструктурних об'єктів, контролю дорожнього руху та забезпечення громадської безпеки, мають суттєвий концептуальний недолік — вони генерують величезний масив відеоданих, який вимагає безперервного аналізу людиною-оператором. Враховуючи фактор людської втоми, втрати концентрації уваги та обмеженості поля зору, ефективність таких систем стрімко падає при збільшенні кількості камер або при тривалому спостереженні за одноманітними сценами.

Особливої гостроти ця проблема набуває в контексті новітніх загроз, зокрема несанкціонованого використання малих безпілотних літальних апаратів (БпЛА) над об'єктами критичної інфраструктури, аеропортами та військовими базами. Малорозмірні цілі, що рухаються з високою швидкістю та здатні здійснювати складні просторові маневри, практично неможливо постійно утримувати в полі зору оптичних приладів вручну. З огляду на це, виникає гостра необхідність у розробці гібридних апаратно-програмних комплексів, які здатні не лише семантично розпізнавати об'єкт у кадрі за допомогою штучного інтелекту, але й самостійно здійснювати його механічне супроводження (кінематичний трекінг) шляхом керування оптичною віссю камери в режимі реального часу.

Вирішення задачі ідентифікації та супроводження цілі стикається з низкою фундаментальних проблем у галузі комп'ютерного зору (Computer Vision):

1. Динаміка та неструктурованість фону: На відміну від лабораторних умов, реальне середовище містить безліч візуальних шумів. Рух гілок дерев, зміна хмарності, відблиски світла на воді або склі зводять нанівець ефективність класичних алгоритмів віднімання фону (Background Subtraction). Система

повинна розуміти семантику об'єкта, відокремлюючи його від динамічних перешкод.

2. Оклюзія (перекриття) та зміна ракурсу: Під час руху ціль може частково або повністю перекриватися іншими об'єктами (стовпами, будівлями, птахами). Крім того, просторовий поворот об'єкта змінює його двовимірну проєкцію на матрицю камери, що вимагає від алгоритмів високої інваріантності до масштабу та афінних перетворень.

3. Проблема масштабу цілі: Малі безпілотники або об'єкти на значній відстані займають у кадрі високої роздільної здатності (Full HD або 4K) площу всього у кілька десятків пікселів. Вилучення просторових ознак (Feature Extraction) з такої малої області є нетривіальною задачею навіть для сучасних згорткових нейронних мереж (ЗНМ).

Ключовою відмінністю розроблюваної системи від програм для офлайн-аналізу відео є наявність замкнутого контуру зворотного зв'язку (Feedback Loop) з механічною підсистемою. Це накладає жорсткі обмеження на затримку обробки інформації (Latency).

Комплекс позиціонування підпорядковується законам теорії автоматичного керування. Якщо загальний час між фізичним надходженням фотонів на матрицю камери та формуванням керуючого імпульсу на кроковий двигун перевищить певний поріг (наприклад, 100-150 мілісекунд), система почне працювати з "минулим" станом цілі. Для швидкісних об'єктів це призведе до перерегулювання (Overshoot), виникнення механічних осциляцій та повної втрати об'єкта з вузького кута зору телеоб'єктива (Telephoto Lens). Отже, архітектура програмного забезпечення повинна мінімізувати затримки на етапах відеозахоплення (Input Lag), тензорних обчислень нейромережі (Inference Time) та передачі даних по комунікаційних інтерфейсах (Transmission Delay).

Остання група проблем стосується перетворення обчислених цифрових координат на плавний фізичний рух. Стандартні операційні системи (Windows, Linux) є системами з розділенням часу і не здатні забезпечити детерміновану генерацію керівних імпульсів. Використання мікроконтролерів є обов'язковим,

проте вимагає вирішення проблеми дискретності: розраховані координати надходять дискретними пакетами (наприклад, 30 разів на секунду), тоді як двигун повинен рухатися безперервно і плавно, щоб уникнути ефекту змазування зображення (Motion Blur) та резонансних вібрацій оптичної платформи.

Виходячи з наведеного аналізу проблематики, метою даної бакалаврської роботи є проектування, програмна реалізація та тестування гібридного апаратно-програмного комплексу для автоматичного семантичного розпізнавання, математичного відстеження та безперервного кінематичного супроводження рухомих об'єктів із використанням відеоданих високої роздільної здатності.

Для досягнення поставленої мети сформовано такий комплекс завдань:

1. Провести глибокий системний аналіз існуючих методів комп'ютерного зору, алгоритмів детектування (Object Detection) та багатооб'єктного трекінгу (Multiple Object Tracking) для обґрунтування вибору оптимального математичного апарату.

2. Спроектувати розподілену дворівневу архітектуру системи (обчислювальний ПК та мікроконтролерний блок) та розробити концептуальні UML-моделі взаємодії компонентів.

3. Розробити логічну та фізичну моделі бази даних для зберігання телеметрії та аналітичних показників роботи системи, а також обґрунтувати вибір системи управління базами даних (СУБД).

4. Програмно реалізувати підсистему високошвидкісного відеозахоплення та інтегрувати одностадійну згорткову нейромережу (YOLO) з алгоритмом прогностичного трекінгу (DeepSORT).

5. Розробити алгоритм кінематичного керування (програмний ПІД-регулятор) на базі мікроконтролера ESP32 для плавного наведення оптичної осі за допомогою крокових двигунів та мікрокрокових драйверів.

6. Створити багатопотоковий графічний інтерфейс користувача (GUI) для оперативного керування параметрами системи та візуалізації результатів у реальному часі.

Об’єкт дослідження – процес автоматизованого семантичного розпізнавання, просторової локалізації та механічного супроводження рухомих цілей у динамічному відеопотоці.

Предмет дослідження – архітектури згорткових нейронних мереж, математичні методи об'єктного трекінгу (зокрема, фільтрація Калмана), алгоритми багатопотокової обробки даних та методи теорії автоматичного керування (ПІД-регулювання) просторовими механізмами.

1.2 Огляд інформаційних джерел та існуючих рішень

Розробка надійної системи активного відеоспостереження вимагає критичного аналізу існуючих підходів до вирішення задачі локалізації та класифікації об'єктів. Вибір математичного апарату безпосередньо впливає на головний компроміс комп'ютерного зору (Computer Vision) — баланс між точністю виявлення (Mean Average Precision, mAP) та швидкістю обробки (Frames Per Second, FPS). У даному підрозділі розглянуто еволюцію алгоритмів детектування об'єктів з метою обґрунтування вибору оптимальної архітектури для роботи в контурі управління реального часу.

1.2.1 Класичні алгоритми комп'ютерного зору

До початку широкого впровадження методів глибокого машинного навчання (Deep Learning), задачі виявлення рухомих цілей вирішувалися за допомогою класичних алгоритмів обробки зображень. Їхньою спільною рисою була залежність від "ручного" конструювання ознак (Hand-crafted feature extraction).

1. Методи віднімання фону (Background Subtraction): Такі алгоритми, як MOG2 (Mixture of Gaussians), будують статистичну модель фонового зображення піксель за пікселем і порівнюють поточний кадр із цією моделлю. Усе, що суттєво відрізняється, класифікується як об'єкт (Foreground). Незважаючи на низьку обчислювальну складність, цей метод є абсолютно непридатним для розроблюваної системи. Оскільки камера буде механічно переміщуватися кроковими двигунами для супроводження цілі, весь фон у кадрі

перебуватиме в русі. Віднімання фону призведе до того, що алгоритм виділить усе зображення як один великий "рухомий об'єкт".

2. Оптичний потік (Optical Flow): Метод Лукаса-Канаде (Lucas-Kanade) аналізує вектори зміщення пікселів між двома сусідніми кадрами. Він дозволяє оцінити швидкість та напрямок руху об'єктів. Проте оптичний потік вкрай чутливий до зміни освітлення та вимагає наявності яскраво виражених текстур на об'єкті. Гладкі поверхні (наприклад, фюзеляж БпЛА на тлі сірого неба) не створюють достатнього градієнта для побудови векторів зміщення. Крім того, цей метод не здатний здійснювати семантичну класифікацію (відрізнити птаха від безпілотника).

3. Евристичні детектори ознак (Haar, HOG): Метод Віоли-Джонса, що використовує каскади Хаара, зробив революцію у розпізнаванні облич, спираючись на контрастні перепади яскравості. Інший підхід — гістограми спрямованих градієнтів (HOG) у поєднанні з методом опорних векторів (SVM) — успішно розпізнавав пішоходів. Недоліком цих підходів є низька здатність до узагальнення. Вони потребують створення нових шаблонів (екстракторів) для кожного нового ракурсу об'єкта, що робить розпізнавання складних просторових цілей з довільним креном (наприклад, FPV-дронів) практично неможливим.

1.2.2 Еволюція згорткових нейронних мереж у задачах детектування

Докорінний злам у вирішенні задач розпізнавання образів відбувся завдяки розвитку згорткових нейронних мереж (Convolutional Neural Networks, CNN). На відміну від класичних алгоритмів, ЗНМ здатні самостійно ієрархічно вилучати просторові ознаки: від примітивних ліній та кутів на перших шарах згортки до складних геометричних патернів на глибинних.

Задачу детектування об'єктів (Object Detection), яка полягає не лише у визначенні класу об'єкта, але й у знаходженні його точних координат (Bounding Box Regression), прийнято вирішувати за допомогою двох парадигм: двостадійних та одностадійних детекторів.

Двостадійні детектори (Two-stage Detectors)

До цієї групи належить сімейство архітектур R-CNN (Region-based CNN), Fast R-CNN та Faster R-CNN. Логіка їхньої роботи поділена на два послідовні етапи:

- Етап 1: Мережа пропозиції регіонів (Region Proposal Network, RPN) сканує вхідне зображення (або його карту ознак) і генерує тисячі гіпотез — прямокутних областей, де з високою ймовірністю знаходиться "щось", відмінне від фону.
- Етап 2: Кожен із запропонованих регіонів вирізається (через операцію RoI Pooling), нормалізується у розмірах і подається на повнозв'язні шари (Fully Connected Layers) для остаточної класифікації та точного коригування меж рамки.

Перевагою двостадійних архітектур є надзвичайно висока точність, особливо при детектуванні щільних скупчень дрібних об'єктів. Проте архітектурний поділ на два етапи створює серйозне "вузьке місце" (bottleneck) в обчисленнях. Навіть із використанням сучасних графічних прискорювачів (GPU), такі моделі здатні обробляти лише 5-15 кадрів на секунду. Впровадження Faster R-CNN у контур ПІД-регулювання призвело б до затримки реакції механіки понад 200 мілісекунд, що є неприпустимим для системи кінематичного трекінгу.

Для вирішення проблеми низької швидкодії були розроблені архітектури SSD (Single Shot MultiBox Detector) та сімейство YOLO (You Only Look Once). Вони відмовилися від модуля пропозиції регіонів і переформулювали задачу детектування як єдину задачу глобальної регресії.

Тензор вхідного зображення проходить через магістральну мережу (Backbone) лише один раз (single forward pass). Мережа генерує щільну просторову сітку передбачень, де кожен елемент сітки одночасно містить інформацію про ймовірність наявності об'єкта, його клас та координати меж. Цей підхід дозволив різко скоротити час інференсу, досягнувши показників швидкодії, що перевищують частоту оновлення відеокамер (понад 60-100 FPS), роблячи одностадійні детектори ідеальним вибором для систем реального часу.

1.2.3 Архітектурні особливості та переваги моделі YOLOv8

Серед множини одностадійних детекторів, сімейство YOLO демонструє найкращий баланс між швидкістю та точністю виявлення. Для розробки програмного ядра даної системи було обрано архітектуру YOLOv8 (розробки компанії Ultralytics), що є найбільш актуальним та оптимізованим рішенням станом на момент проектування системи.

Принципова відмінність YOLOv8 від попередніх ітерацій (YOLOv3, YOLOv5) полягає у відмові від концепції "анкорів" (Anchor Boxes) на користь підходу Anchor-Free.

У попередніх версіях мережа покладалася на набір попередньо визначених шаблонів прямокутників (анкорів) із фіксованим співвідношенням сторін (наприклад, 1:1, 1:2, 2:1). Мережа лише прогнозувала коефіцієнти деформації цих шаблонів. Такий підхід працював добре для стандартних об'єктів (автомобілі, пішоходи), але значно погіршував результати при виявленні об'єктів із нестандартною морфологією або в ракурсах із сильною перспективною дисторсією (наприклад, БпЛА, що здійснює різкий віраж, або ціль, що стрімко наближається).

Підхід Anchor-Free в YOLOv8 напряму прогнозує координати центру об'єкта та відстані до чотирьох його меж (Top, Bottom, Left, Right). Це значно підвищує здатність моделі до узагальнення (Generalization) на нові, раніше не бачені ракурси цілей.

Інновації на рівні архітектури нейромережі включають:

1. Модернізований Backbone (Магістраль): Використання нових будівельних блоків C2f (Cross Stage Partial Network with 2 convolutions) замість старих блоків C3. C2f модуль покращує потік градієнтів під час навчання (Backpropagation) та забезпечує збереження більш детальної просторової інформації, що є критично важливим для локалізації дрібних цілей на великих фокусних відстанях.

2. Розділена "голова" мережі (Decoupled Head): У традиційному YOLO останні шари мережі одночасно намагалися розрахувати ймовірність класу та

координати обмежувальної рамки. В YOLOv8 ці задачі структурно розділені. Один потік (Branch) тензорів оптимізується виключно для семантичної класифікації (використовуючи функцію втрат Binary Cross-Entropy), а інший — виключно для регресії координат (використовуючи функції втрат Complete IoU та Distribution Focal Loss). Розділення задач зменшує конфлікт градієнтів і суттєво підвищує точність позиціонування рамок.

Аналіз обчислювальної складності показує, що оптимізовані тензорні операції YOLOv8 дозволяють моделі впевнено працювати на швидкості до 80 FPS при роздільній здатності 640 x 640 пікселів, використовуючи апаратне прискорення CUDA на графічних процесорах (GPU) середнього цінового сегмента.

Підсумовуючи, висока швидкодія, здатність ефективно узагальнювати геометрію об'єктів завдяки Anchor-Free архітектурі та наявність оптимізованого програмного API (на базі PyTorch) роблять YOLOv8 безальтернативним та найбільш науково обґрунтованим вибором для підсистеми розпізнавання у розроблюваному гібридному комплексі активного відеоспостереження.

1.2.4 Аналіз алгоритмів багатооб'єктного відстеження (Multiple Object Tracking)

Вирішення задачі виявлення цілі на окремому статичному кадрі за допомогою нейромережі YOLOv8 є необхідною, але недостатньою умовою для створення замкнутої системи кінематичного наведення. Детектор об'єктів працює дискретно: він не зберігає пам'ять про попередні кадри і не здатний визначити, чи є об'єкт на поточному кадрі тим самим об'єктом, що був виявлений мілісекунду тому, чи це вже інша ціль. Без алгоритму відстеження (Tracking) система не матиме цільового вектора для ПІД-регулятора, а у випадку появи в кадрі кількох цілей одного класу — почне хаотично перемикатися між ними.

Задачу побудови неперервних траєкторій об'єктів у відеопотоці вирішує клас алгоритмів багатооб'єктного відстеження (Multiple Object Tracking, MOT).

До епохи глибокого навчання домінували методи, що базувалися на аналізі піксельних шаблонів. Наприклад, алгоритми KCF (Kernelized Correlation Filters)

та CSRT аналізують вибрану область (шаблон цілі) і на кожному наступному кадрі шукають область з максимальною кореляцією до цього шаблону, використовуючи перетворення Фур'є для прискорення обчислень. Головним недоліком таких методів є їхня вразливість до оклюзій (перекриттів) та зміни масштабу. Якщо дрон, який відстежується алгоритмом KCF, тимчасово залетить за стовп або дерево, його піксельний шаблон буде зруйновано. Алгоритм "захопить" текстуру перешкоди і безповоротно втратить ціль, вимагаючи ручної переініціалізації оператором.

Сучасним стандартом у системах відеоаналітики є парадигма "відстеження через детектування". У цьому підході на кожному кадрі спочатку працює потужний детектор (в нашому випадку YOLO), який знаходить усі об'єкти. Завдання трекера зводиться до задачі асоціації даних (Data Association): потрібно "зв'язати" нові детекції з існуючими траєкторіями (треками).

Базовим алгоритмом цієї парадигми є SORT (Simple Online and Realtime Tracking). Він використовує виключно просторову інформацію (координати рамок) і базується на двох математичних апаратах: фільтрі Калмана та Угорському алгоритмі (Hungarian Algorithm). Проте базовий SORT страждає від проблеми перемикання ідентифікаторів (ID Switches), якщо два об'єкти проходять один повз одного.

Для розроблюваної системи обрано вдосконалений алгоритм DeepSORT (Simple Online and Realtime Tracking with a Deep Association Metric). Він вирішує проблему перемикання ID шляхом інтеграції додаткової нейромережі, що дозволяє порівнювати не лише координати, але й візуальні ознаки об'єктів. Логіка алгоритму DeepSORT базується на трьох основних стовпах:

1. Математичне прогнозування (Фільтр Калмана):

Фільтр Калмана — це рекурсивний алгоритм, який оцінює стан динамічної системи на основі серії зашумлених вимірювань. Стан кожного об'єкта (треку) моделюється восьмивимірним вектором:

$$X = [u, v, \gamma, h, \dot{u}, \dot{v}, \dot{\gamma}, \dot{h}]^T \qquad (1.1)$$

де (u, v) - координати центру обмежувальної рамки, γ - співвідношення сторін (aspect ratio), h - висота рамки, а $\dot{u}, \dot{v}, \dot{\gamma}, \dot{h}$ - їх відповідні швидкості.

На кожному кадрі фільтр Калмана прогнозує нове положення об'єкта, виходячи з припущення про рівномірний прямолінійний рух. Якщо об'єкт тимчасово перекритий (YOLO його не бачить), фільтр Калмана продовжує оновлювати його координати за інерцією протягом заданої кількості кадрів (параметр max_age), запобігаючи миттєвій втраті цілі.

2. Метрика глибокої асоціації (ReID модель):

Кожен розпізнаний детектором YOLO об'єкт вирізається і подається на вхід невеликої згорткової нейромережі (зазвичай архітектури типу ResNet). Ця мережа не класифікує об'єкт, а виконує функцію вилучення візуальних ознак (Feature Extraction). На виході формується вектор ознак (Embeddings) розмірністю 128 або 256. Цей вектор є "візуальним відбитком пальця" об'єкта. Навіть якщо об'єкт частково перекритий, його дескриптор залишається статистично близьким до оригіналу.

3. Алгоритм призначення (Угорський алгоритм):

Для того, щоб зв'язати прогнози фільтра Калмана з новими розпізнаваннями YOLO, система розраховує матрицю вартості (Cost Matrix) на основі двох метрик:

- Просторова метрика: Відстань Махаланобіса (Mahalanobis Distance) між спрогнозованим положенням та реальною рамкою детекції. Вона враховує не лише евклідову відстань, але й невизначеність (коваріацію) прогнозу.
- Візуальна метрика: Косинусна відстань (Cosine Distance) між векторами ознак (Embeddings).

Зважена сума цих двох метрик формує загальну вартість асоціації. Угорський алгоритм знаходить таке попарне призначення старих треків новим детекціям, при якому загальна матриця вартості буде мінімальною.

Використання DeepSORT гарантує, що система зберігатиме прив'язку до конкретної фізичної цілі (наприклад, конкретного дрона у зграї), забезпечуючи стабільний потік координатних похибок для роботи кінематичного контролера.

1.2.5 Аналіз методів кінематичного керування та апаратних рішень

Заключним етапом циклу активного відеоспостереження є перетворення обчислених цифрових координат (відхилення об'єкта від оптичного центру камери) на фізичний рух.

Використання колекторних двигунів постійного струму (DC) у прецизійних системах позиціонування неможливе через відсутність зворотного зв'язку за положенням ротора та ефект вибігу (інерції). Сервоприводи (Servo motors), попри наявність внутрішнього потенціометра, мають дискретний хід (зазвичай 1 градус) та систему редукторів. Механічний люфт (Backlash) редуктора є критичним фактором: під час роботи з телеоб'єктивом на великих фокусних відстанях зміщення осі навіть на 0.5 градуса призведе до відхилення поля зору на десятки метрів, що спричинить втрату об'єкта.

Оптимальним інженерним рішенням є використання двофазних гібридних крокових двигунів (Stepper Motors) стандарту NEMA 17. Їхня конструкція дозволяє повертати ротор на строго фіксований кут (стандартно 1.8 градуса на один крок) без використання редукторів. Для усунення проблеми резонансу на низьких швидкостях, яка притаманна кроковим двигунам, необхідно застосовувати сучасні драйвери з підтримкою мікрокрокінгу (наприклад, TMC2208). Такі драйвери апаратно модулюють струм в обмотках у формі синусоїди, фіксуючи ротор у проміжних положеннях (до 256 мікрокроків на один повний крок). Це гарантує безшумний хід, повну відсутність вібрацій оптичної осі та кутову роздільну здатність механіки на рівні 0.007°.

Обчислювальне ядро системи (ПК з YOLO) генерує лише сигнал помилки $e(t)$ - відстань у пікселях по осі X та Y від поточного положення цілі до центру прицілу на екрані. Відправка цих значень безпосередньо на двигуни призведе до дискретних ривків, оскільки частота оновлення кадрів (30-60 Гц) не синхронізована з мікросекундними імпульсами крокової механіки.

Для згладжування траєкторії та забезпечення плавного розгону і гальмування масивної оптичної платформи застосовується пропорційно-інтегрально-диференціальний (ПІД) регулятор. Це алгоритм керування зі

зворотним зв'язком, який формує керуючий сигнал $u(t)$ (бажану кутову швидкість двигуна) на основі трьох компонент:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_{d\frac{de(t)}{dt}} \quad (2.1)$$

1. Пропорційна складова (K_p): Визначає миттєву реакцію системи. Чим далі об'єкт відхиляється від центру, тим вища швидкість обертання двигуна. Однак виключно пропорційне керування ніколи не зведе похибку до абсолютного нуля, породжуючи статичну помилку наведення.

2. Інтегральна складова (K_i): Накопичує значення помилки з плином часу. Якщо ціль залишається трохи зміщеною від центру, інтегратор поступово збільшуватиме керуючий вплив, "дотискаючи" об'єктив точно в центр.

3. Диференціальна складова (K_d): Реагує на швидкість зміни помилки. Її можна розглядати як алгоритмічний "амортизатор" або "демпфер". Коли об'єктив швидко наближається до центру цілі, $e(t)$стрімко зменшується, і похідна стає від'ємною. Диференціальна складова зменшує загальну швидкість, запобігаючи проскоку (перерегулюванню) та осциляціям навколо цілі.

Розрахунок ПІД-алгоритму вимагає жорсткого реального часу (детермінованого циклу dt), тому він не може виконуватися на комп'ютері під управлінням ОС загального призначення. Розрахунок ПІД-формули та генерацію сигналів STEP/DIR для драйверів доцільно винести на окремий апаратний вузол — мікроконтролер реального часу (наприклад, ESP32). Це дозволить створити надійну розподілену архітектуру, де ПК займається виключно нейромережевими обчисленнями, а мікроконтролер забезпечує безперебійний контур кінематичного керування.

1.3 Моделювання предметної області та проектування архітектури системи

Розробка гібридного апаратно-програмного комплексу, що поєднує високопродуктивні обчислення нейронних мереж із жорстким реальним часом мікроконтролерної периферії, вимагає ретельного попереднього проектування. Для формалізації функціональних вимог, опису процесів взаємодії між

компонентами та побудови концептуальної архітектури використано методологію об'єктно-орієнтованого моделювання на базі уніфікованої мови моделювання (Unified Modeling Language, UML).

Моделювання предметної області виконано з двох ракурсів: зовнішньої поведінкової моделі (діаграма прецедентів), яка фіксує функціональні вимоги, та внутрішньої динамічної моделі (діаграма діяльності), яка описує алгоритмічну послідовність обробки інформації в режимі реального часу.

1.3.1 Функціональне моделювання (Модель прецедентів)

Діаграма прецедентів (Use Case Diagram) описує систему як "чорний ящик", визначаючи її межі та фіксуючи взаємодію зовнішніх сутностей (акторів) із функціональними модулями (прецедентами). Такий підхід дозволяє чітко розмежувати зони відповідальності між людиною, комп'ютерним зором та виконавчою механікою.

У розроблюваній системі виділено трьох ключових акторів, кожен з яких ініціює специфічний набір процесів:

1. Користувач (Оператор): Головний керуючий актор. Його роль полягає в ініціалізації роботи комплексу, заданні граничних умов (наприклад, порогу впевненості Confidence Threshold для відсіювання хибних цілей) та моніторингу результатів через графічний інтерфейс користувача (GUI).

2. Оптичний сенсор (Камера): Апаратний актор, що виступає зовнішнім джерелом безперервного потоку даних. Його єдиною, але критично важливою функцією є захоплення світлового потоку та передача нестиснутої матриці пікселів (Clean HDMI) на пристрій відеозахоплення ПК.

3. Мікроконтролер (ESP32): Виконавчий апаратний актор. На відміну від сенсора, мікроконтролер має власну логіку і відповідає за два фундаментальні прецеденти: "Автоматичне калібрування (Homing)", яке ініціюється при старті системи для пошуку "нульової" координати за допомогою магнітних датчиків Холла, та "Керування механікою крокових двигунів", що є кінцевою реакцією системи на знайдену ціль.

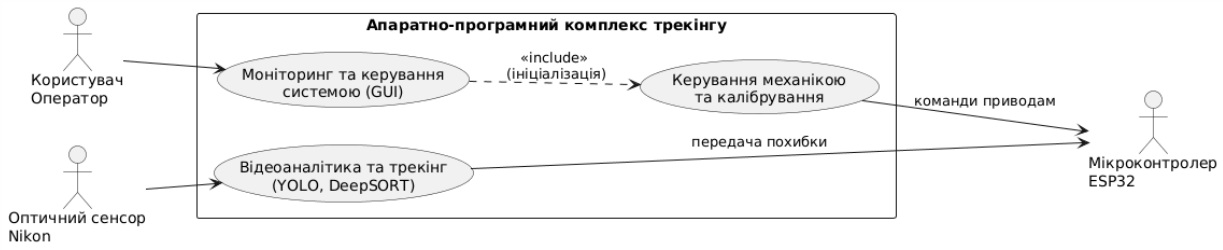

Рис.1.1 Діаграма прецедентів системи

На діаграмі відображено складну ієрархію залежностей між прецедентами за допомогою відношення <<include>> (включення). Базовий прецедент "Захоплення відеопотоку" обов'язково включає прецедент "Детектування об'єктів (YOLO)". Відповідно, процес детектування є передумовою для "Відстеження траєкторії (DeepSORT)". Цей ланцюжок залежностей підтверджує строгу конвеєрну архітектуру (Pipeline) обробки візуальної інформації: жоден наступний етап не може розпочатися без успішного завершення попереднього. Останньою ланкою конвеєра є "Розрахунок похибки наведення", результати якого перетинають межу системи і передаються на апаратного актора — Мікроконтролер.

1.3.2 Динамічне моделювання (Модель діяльності)

Якщо діаграма прецедентів фіксує, що робить система, то діаграма діяльності (Activity Diagram) пояснює, як саме вона це робить. Діаграма діяльності деталізує алгоритмічний контур обробки кожного кадру, візуалізуючи паралелізм процесів, точки прийняття рішень та маршрутизацію потоків даних.

Процес функціонування комплексу в режимі реального часу ініціюється оператором, після чого відбувається розділення потоку виконання (Fork) на дві паралельні гілки: ініціалізація програмного ядра на ПК (завантаження тензорних ваг YOLOv8 у відеопам'ять GPU, відкриття COM-порту, ініціалізація пулу з'єднань з PostgreSQL) та ініціалізація апаратної частини (калібрування нульової точки на CNC Shield).

Після синхронізації (Join) цих гілок система входить у головний безперервний цикл обробки (Main Loop):

1. Етап збору даних: З буфера пристрою відеозахоплення (MS2130) вилучається актуальний кадр. Відбувається його програмна нормалізація (зміна

роздільної здатності до формату 640x640 для узгодження з тензором входу нейромережі).

2. Етап інференсу: Нормалізований тензор передається в графічний прискорювач. Модель YOLOv8 генерує масив обмежувальних рамок.

3. Точка прийняття рішення (Decision Node): Алгоритм перевіряє умову наявності цілей, що відповідають заданим критеріям (клас об'єкта та відсоток впевненості). Якщо об'єкт не знайдено (наприклад, чисте небо), система відкидає кадр і миттєво повертається на початок циклу.

4. Етап трекінгу та агрегації: У разі успішного виявлення, масив рамок передається алгоритму DeepSORT. Фільтр Калмана коригує прогнози, а нейромережа ReID здійснює зіставлення цілей. Системі повертається унікальний Track ID цілі та її згладжені просторові координати.

5. Етап математичного перетворення та логіювання: На основі отриманих координат обчислюється геометричний центр об'єкта. Знаходиться вектор відхилення (Error Vector) відносно оптичного центру кадру. Паралельно, асинхронний потік зберігає дані про детекцію в базу даних PostgreSQL (через драйвер psycopg2).

6. Етап комунікації та відпрацювання: Обчислена координатна похибка транслюється в рядок (наприклад, "X15Z-5\n") і відправляється через USB-UART на ESP32. Мікроконтролер, отримавши пакет, підставляє значення похибки у формулу ПІД-регулятора, розраховує необхідні прискорення і модулює струми на драйверах TMC2208, здійснюючи фізичний зсув камери.

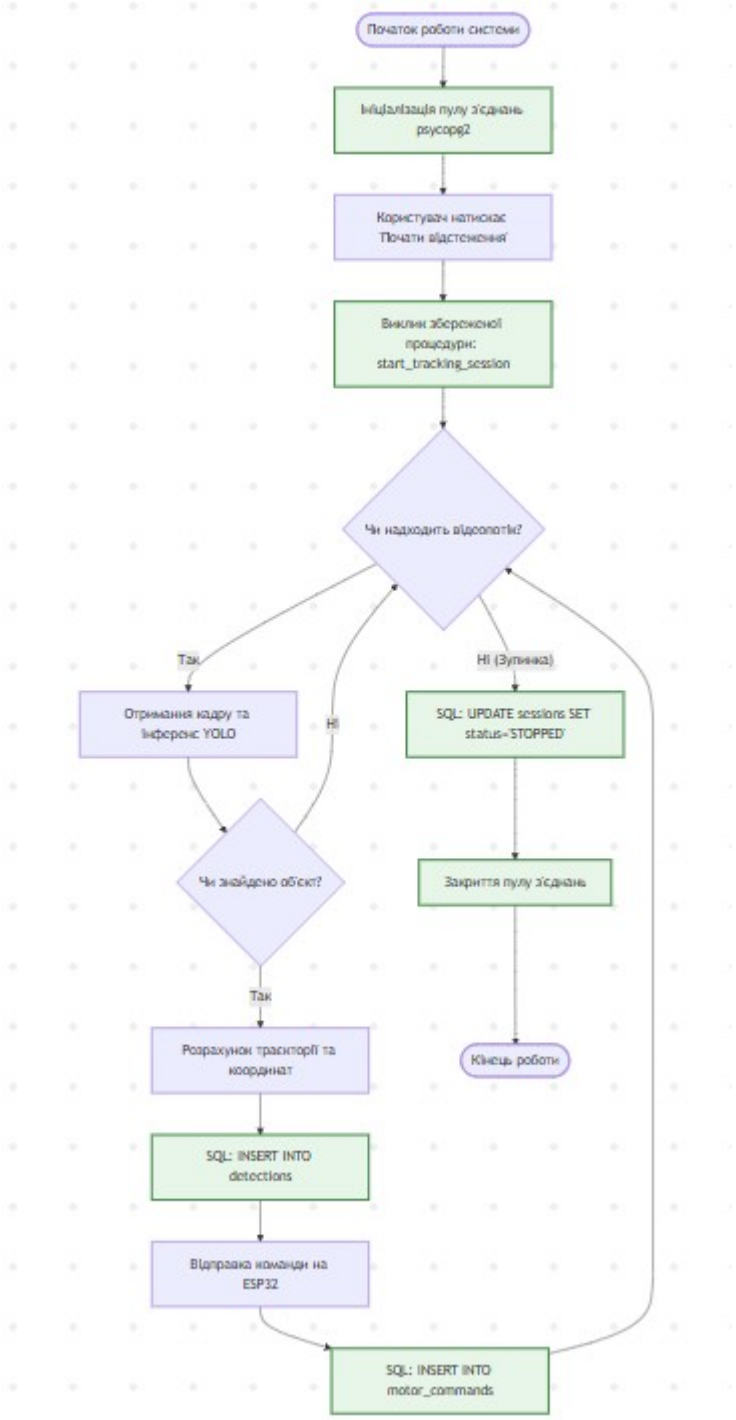

Рис.2.1 Діаграма діяльності

Описана динамічна модель яскраво демонструє переваги обраної гібридної архітектури. Обчислювально важкі завдання (тензорне множення в згорткових шарах) локалізовані на графічному прискорювачі ПК, де доступні значні ресурси паралелізації. Водночас завдання, що вимагають мікросекундної точності генерації імпульсів (керування двигунами), винесені за межі операційної системи ПК і виконуються апаратними таймерами мікроконтролера.

Така концептуальна ізоляція гарантує, що навіть у разі різкого падіння FPS (наприклад, через перегрів відеокарти або обробку дуже складної сцени з десятками об'єктів), мікроконтролер продовжить плавно вести камеру за інерцією, спираючись на останні отримані дані та розрахунки інтегральної складової ПІД-регулятора, запобігаючи неконтрольованим ривкам механіки.

Висновки до Розділу 1

Проведений системний аналіз предметної області дозволив комплексно дослідити проблему автоматизованого виявлення та кінематичного супроводження рухомих цілей. Доведено, що класичні методи комп'ютерного зору не здатні забезпечити належний рівень надійності в умовах динамічного фону. Оптимальним математичним апаратом для розпізнавання цілей у режимі жорсткого реального часу визначено одностадійну згорткову нейромережу YOLOv8, архітектура якої (Anchor-Free) дозволяє з високою точністю узагальнювати просторові ознаки малих БпЛА. Для вирішення проблеми перемикання ідентифікаторів під час перекриття цілей обґрунтовано необхідність використання алгоритму DeepSORT, який поєднує рекурсивне прогнозування фільтра Калмана з метрикою глибокої асоціації (ReID). На основі проведеного аналізу апаратних засобів доведено, що прецизійне наведення телеоб'єктива можливе лише за умови використання крокових двигунів із мікрокроковою інтерполяцією (драйвери TMC2208) під керуванням мікроконтролерного ПІД-регулятора, який згладжує дискретні координатні похибки, отримані від обчислювального ядра. Розроблені UML-моделі (прецедентів та діяльності) дозволили формалізувати вимоги до комплексу і довели життєздатність розподіленої гібридної архітектури, у якій тензорні обчислення ізольовані від апаратних процесів жорсткого реального часу. Це формує надійний теоретичний фундамент для переходу до етапів проектування інформаційного та прикладного програмного забезпечення.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

Будь-який сучасний комплекс комп'ютерного зору, що працює в режимі реального часу, є потужним генератором даних. Під час активного супроводження цілі система обробляє десятки кадрів на секунду, кожен з яких містить унікальну просторову інформацію про об'єкт, показники впевненості нейромережі, часові затримки (латентність) обчислювального ядра та відповідні реакції виконавчої механіки. Безперервне та надійне збереження цих даних є критично необхідним для вирішення низки експлуатаційних та аналітичних завдань:

- Офлайн-аналіз точності роботи детекторів та трекерів для подальшого донавчання (Fine-tuning) нейромережевих моделей.
- Діагностика роботи апаратної підсистеми (виявлення пропусків кроків двигунів або аномальних затримок у комунікації).
- Оцінка ефективності налаштувань ПІД-регулятора шляхом аналізу графіків відпрацювання координатної похибки в часі.
- Формування звітно-статистичної документації (дашбордів) для операторів системи.

З огляду на це, невід'ємною складовою розроблюваного програмного забезпечення є підсистема доступу до даних (Data Access Layer). Проектування інформаційного забезпечення включає логічну організацію структур даних, обґрунтований вибір системи управління базами даних (СУБД) та створення оптимізованої фізичної схеми.

2.1 Логічне проектування інформаційної бази даних

Проектування інформаційної бази розпочинається з концептуального та логічного моделювання, що є абстрагованим від конкретних програмних рішень чи діалектів мови SQL. Метою цього етапу є визначення сутностей предметної області, їх атрибутів та зв'язків між ними з урахуванням правил реляційної алгебри. Для візуалізації логічної структури використано методологію об'єктно-реляційного моделювання у вигляді ER-діаграми (Entity-Relationship Diagram) в нотації "пташина лапка" (Crow's foot).

На основі аналізу функціональних вимог до системи було виділено п'ять ключових сутностей, що повною мірою описують процес відеомоніторингу:

1. Сутність «Сесія спостереження» (Session): Виступає кореневою сутністю системи. Вона інкапсулює дані про життєвий цикл кожного окремого запуску комплексу.

- Атрибути: Унікальний ідентифікатор сесії (первинний ключ), мітка часу ініціалізації (Start Timestamp), мітка часу завершення (End Timestamp) та поточний стан апаратури (Status), який може набувати значень "Активна", "Зупинена оператором" або "Апаратна помилка".

2. Сутність «Довідник класів об'єктів» (Object_Class): Класифікатор, що містить еталонний перелік типів цілей, на розпізнавання яких натренована модель YOLOv8.

- Атрибути: Ідентифікатор класу (первинний ключ), текстова мітка класу (наприклад, "drone", "vehicle", "person") та розширений опис для інтерфейсу користувача.

3. Сутність «Лог кадру відеопотоку» (Frame_Log): Відповідає за профілювання (Profiling) продуктивності обчислювального ядра.

- Атрибути: Ідентифікатор кадру (первинний ключ), зовнішній ключ на сутність "Сесія", точна мітка часу захоплення кадру, час, витрачений на інференс нейромережі (в мілісекундах), та обчислений поточний показник кадрів на секунду (FPS).

4. Сутність «Розпізнана ціль» (Detected_Target): Зберігає безпосередні результати роботи конвеєра комп'ютерного зору (YOLOv8 + DeepSORT) на конкретному кадрі.

- Атрибути: Ідентифікатор детекції (первинний ключ), зовнішній ключ на "Кадр", зовнішній ключ на "Клас об'єкта", внутрішній ідентифікатор треку від DeepSORT (Track ID) для відстеження об'єкта в часі, відсоток впевненості нейромережі (Confidence Score) та чотири просторові координати обмежувальної рамки (X_min, Y_min, X_max, Y_max).

5. Сутність «Команда управління механікою» (Motor_Command): Логіює телеметричні дані контуру зворотного зв'язку, які передаються на мікроконтролер ESP32.

• Атрибути: Ідентифікатор команди (первинний ключ), зовнішній ключ на "Сесію", мітка часу відправки пакета, обчислене значення координатної помилки по осі панорамування (ΔX) та по осі нахилу (ΔY).

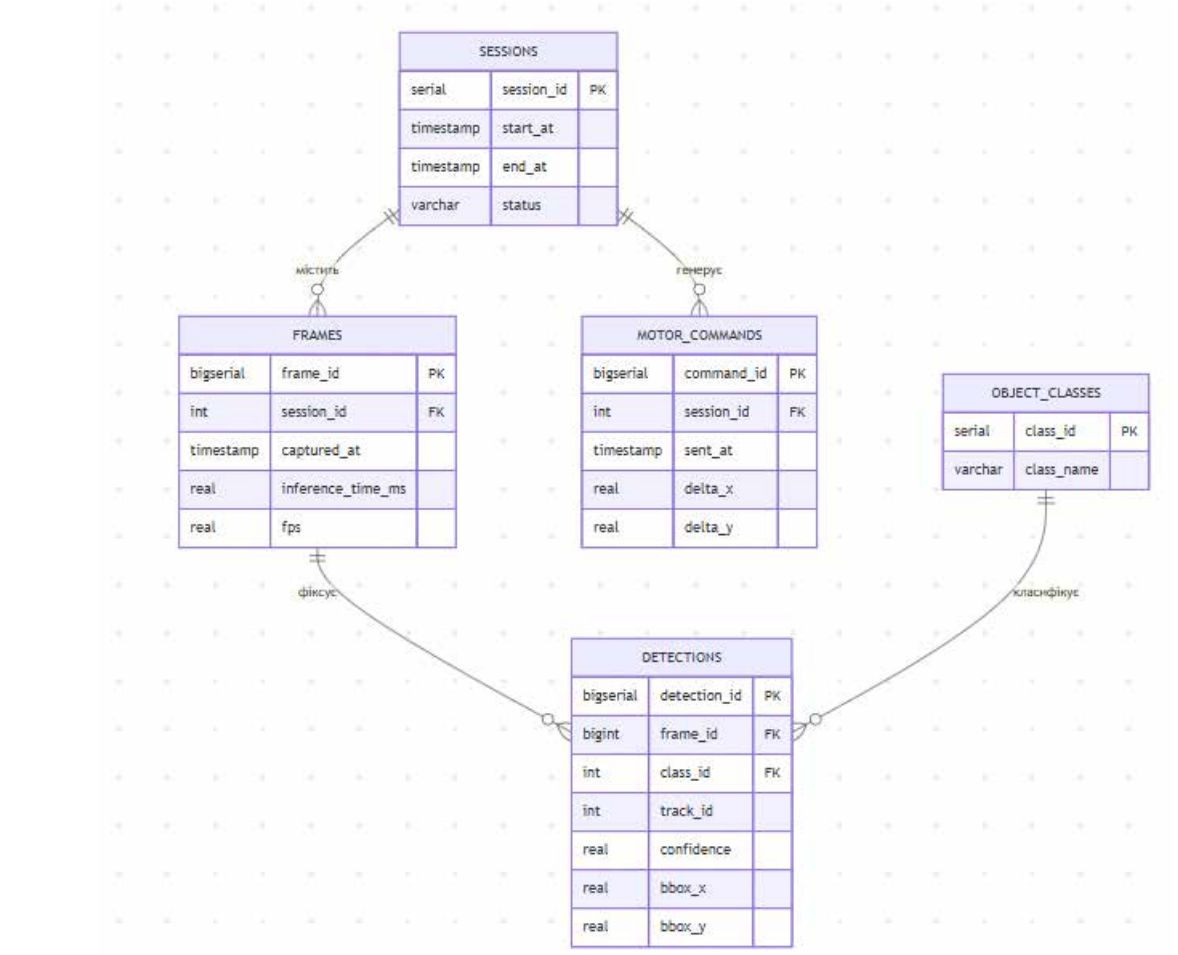

Рис.3.1 ER-діаграма логічної моделі бази даних

Нормалізація бази даних

Для забезпечення цілісності даних, усунення надмірності та уникнення аномалій модифікації (вставки, оновлення, видалення), спроектована логічна модель була піддана строгому процесу нормалізації відповідно до теорії реляційних баз даних Едгара Кодда:

• Перша нормальна форма (1НФ): Досягається завдяки атомарності всіх атрибутів. Наприклад, координати Bounding Box не зберігаються як єдиний

рядок або масив (що порушило б 1НФ), а декомпозовані на чотири окремі атомарні атрибути. Крім того, всі сутності мають унікальні первинні ключі.

- Друга нормальна форма (2НФ): Модель автоматично відповідає 2НФ, оскільки не використовує складених первинних ключів. Кожен неключовий атрибут у кожній таблиці функціонально повністю залежить від усього первинного ключа.
- Третя нормальна форма (3НФ): Вимагає відсутності транзитивних залежностей між неключовими атрибутами. Ця умова виконана шляхом виокремлення сутності "Довідник класів". Якби текстова мітка "drone" зберігалася безпосередньо в таблиці "Розпізнана ціль", це призвело б до величезного дублювання текстових даних (мільйони рядків з однаковим словом) та транзитивної залежності. Винесення цієї інформації в окрему сутність та використання зв'язку через зовнішній ключ (class_id) повністю задовольняє вимогам 3НФ.

Усі зв'язки між сутностями на ER-діаграмі є зв'язками типу "один-до-багатьох" (1:M). Одна сесія генерує безліч кадрів та моторних команд, один кадр може містити нуль або кілька розпізнаних цілей, а один клас об'єкта може бути присвоєний безлічі розпізнавань.

2.2 Обґрунтування вибору системи управління базами даних (СУБД)

Вибір адекватного програмного ядра для роботи з даними (СУБД) є критичним архітектурним рішенням. База даних розроблюваної системи повинна відповідати специфічним вимогам: підтримувати інтенсивний запис (High-throughput Write) малих порцій даних у режимі реального часу (до 60 транзакцій на секунду) та дозволяти одночасне читання даних (наприклад, для відмальовування графіків в інтерфейсі користувача) без блокування процесів запису.

Враховуючи, що логічна модель даних була успішно зведена до Третьої нормальної форми, використання документоорієнтованих (NoSQL) баз даних, таких як MongoDB або CouchDB, є недоцільним. NoSQL рішення оптимізовані для зберігання неструктурованих ієрархічних даних (наприклад, складних JSON-

документів) і часто жертвують транзакційною узгодженістю (відповідність принципам ACID) заради горизонтальної масштабованості. Для нашої задачі, де телеметричні дані мають жорстко визначену та незмінну структуру таблиць, класичні реляційні СУБД (SQL) забезпечують набагато кращу ефективність та гарантують строгу цілісність завдяки використанню зовнішніх ключів (Foreign Keys Constraints).

Розглядалися три найбільш поширені реляційні рішення: SQLite, MySQL та PostgreSQL.

1. SQLite: Це вбудована файлова база даних. Її головна перевага — відсутність необхідності розгортати окремий серверний процес (Serverless), що спрощує інсталяцію. Проте архітектура SQLite має критичний недолік для багатопотокових систем: вона використовує глобальне блокування бази даних під час запису (Global Database Lock). Оскільки в нашому програмному комплексі потік візуалізації (GUI), потік інференсу (YOLO) та потік комунікації з ESP32 працюють паралельно, спроба одночасного звернення до бази викличе блокування (Lock Contention). Потік комп'ютерного зору буде призупинятися в очікуванні зняття блокування, що призведе до падіння FPS та втрати кадрів. Тому SQLite повністю відхилено.

2. MySQL: Одна з найпопулярніших клієнт-серверних СУБД. Вона забезпечує високу швидкість читання та підтримує паралелізм. Однак MySQL має певні обмеження щодо реалізації складних тригерів, а також менш розвинений аналітичний інструментарій (віконні функції для агрегації) "з коробки" порівняно з більш просунутими об'єктно-реляційними системами.

3. PostgreSQL: Це потужна об'єктно-реляційна СУБД з відкритим вихідним кодом, що повністю відповідає стандарту SQL. Головним аргументом на користь PostgreSQL є її архітектурна особливість — механізм багатоверсійного управління конкурентним доступом (MVCC - Multi-Version Concurrency Control). Механізм MVCC гарантує, що операції читання (Read) ніколи не блокують операції запису (Write), і навпаки. Це означає, що оператор системи може відкрити вкладку статистичного звіту і СУБД почне виконувати

важкий агрегуючий SELECT-запит для побудови графіків, тоді як потік комп'ютерного зору продовжить безперебійно виконувати INSERT-запити нових розпізнавань зі швидкістю 60 FPS у ті самі таблиці.

Крім того, PostgreSQL відрізняється ідеальною сумісністю з мовою Python. Використання адаптера бази даних psycopg2 дозволяє реалізувати механізм пулу потокобезпечних з'єднань (Threaded Connection Pooling). Цей підхід усуває накладні витрати часу (Overhead) на створення нових TCP/IP з'єднань з сервером БД для кожного кадру: з'єднання встановлюються один раз при запуску програми і зберігаються в пулі оперативної пам'яті для багаторазового використання фоновими потоками (Worker Threads).

З огляду на необхідність гарантування мікросекундної доступності бази для запису телеметрії, абсолютну консистентність даних (ACID) та підтримку механізмів MVCC, PostgreSQL обрано як оптимальне програмне ядро для реалізації інформаційного забезпечення апаратно-програмного комплексу.

2.3 Фізичне проектування та реалізація інформаційної бази

Перехід від логічної моделі даних до фізичної схеми є завершальним етапом проектування інформаційного забезпечення. На цьому етапі абстрактні сутності та зв'язки трансформуються у реальні таблиці, колонки, обмеження (Constraints) та активні серверні об'єкти, оптимізовані під синтаксис та архітектурні особливості обраної СУБД (PostgreSQL). Головною метою фізичного проектування є забезпечення максимальної продуктивності операцій запису (Insert) в умовах інтенсивного потоку телеметрії та оптимізація дискового простору.

2.3.1 Оптимізація доменів та типів даних

У системах реального часу вибір некоректного типу даних може призвести до експоненційного зростання розміру файлу бази даних та вичерпання лімітів оперативної пам'яті під час кешування (Buffer Cache). Для кожної колонки було підібрано мінімально необхідний тип даних:

1. Первинні ключі (Primary Keys): Для таблиць з низькою інтенсивністю запису (довідники класів, сесії) використано стандартний тип

SERIAL (4 байти), що забезпечує діапазон до 2.14 мільярда унікальних ідентифікаторів. Однак для транзакційних таблиць логіювання (frames, detections, motor_commands), які поповнюються десятками записів щосекунди, використання 4-байтового цілого числа є ризикованим через можливе переповнення лічильника (Integer Overflow) під час тривалої безперервної експлуатації комплексу. Тому для них застосовано тип BIGSERIAL (8 байт), що гарантує безперебійну генерацію унікальних ключів протягом усього життєвого циклу системи.

2. Типи з рухомою комою (Floating-Point Types): Координати обмежувальних рамок (Bounding Boxes), похибка ПІД-регулятора та рівень впевненості нейромережі (Confidence) мають дробові значення. Стандартним підходом є використання типу DOUBLE PRECISION (8 байт), проте піксельна телеметрія не потребує такої надмірної точності. Було прийнято рішення оптимізувати ці поля до типу REAL (4 байти, одинарна точність). Це дозволило вдвічі зменшити обсяг даних, що передаються по мережі від Python-скрипта до сервера СУБД, та прискорило операції масового запису (Bulk Inserts).

3. Часові мітки (Timestamps): Для точного аналізу затримок та профілювання алгоритмів комп'ютерного зору (наприклад, оцінки часу виконання інференсу YOLOv8) стандартних типів типу DATE або TIME недостатньо. Використано тип TIMESTAMP WITHOUT TIME ZONE, який здатен зберігати часові мітки з мікросекундною роздільною здатністю. Збереження часу в єдиному стандарті UTC виключає проблеми з часовими поясами при розгортанні системи на розподілених серверах.

2.3.2 Активні об'єкти бази даних (Процедури та Тригери)

Відповідно до сучасних архітектурних патернів, частину бізнес-логіки системи доцільно переносити з прикладного програмного рівня (Python-клієнта) на рівень бази даних. Це зменшує кількість мережевих запитів (Network Round-trips) та гарантує цілісність даних незалежно від того, як клієнтська програма завершила свою роботу. В інформаційній базі розроблено два ключові активні об'єкти:

Збережена процедура ініціалізації сесії (start_tracking_session)

При запуску комплексу програма повинна створити новий запис сесії в таблиці sessions і отримати згенерований session_id, щоб у подальшому прив'язувати до нього всі кадри та детекції. Традиційний підхід вимагає двох послідовних запитів: INSERT, а потім SELECT LASTVAL(). Для оптимізації цього процесу створено збережену процедуру на мові PL/pgSQL. Клієнтська програма здійснює лише один виклик (CALL), а СУБД самостійно створює запис, фіксує точний поточний час сервера (уникаючи розбіжностей у годинниках між клієнтом і сервером) та повертає ідентифікатор сесії. Це мінімізує затримку (Latency) на етапі ініціалізації апаратури.

Тригер контролю цілісності сесій (trg_auto_close_session)

В експлуатаційних умовах можливі апаратні та програмні збої: аварійне відключення живлення камери, розрив з'єднання USB-UART з мікроконтролером ESP32, або падіння скрипта через вичерпання відеопам'яті (Out-Of-Memory Error) на GPU. У такому випадку клієнтська програма не встигне відправити SQL-запит на закриття сесії, і поле end_at (час завершення) залишиться порожнім (значення NULL). Це призведе до появи так званих "вічно відкритих" сесій (Data Anomaly), що унеможливить коректний розрахунок аналітичної статистики (наприклад, загальної тривалості спостереження).

Для вирішення цієї проблеми на рівні СУБД реалізовано тригер trg_auto_close_session, який спрацьовує перед оновленням записів (BEFORE UPDATE) у таблиці sessions. Якщо керуючий скрипт надсилає оновлення статусу пристрою на критичний (наприклад, "STOPPED" або "ERROR"), тригерна функція перехоплює цей запит і автоматично підставляє поточний системний час у поле end_at. Таким чином, цілісність реляційних даних підтримується автономно ядром PostgreSQL.

2.3.3 Індексування та оптимізація читання (B-Tree Indexes)

Незважаючи на те, що головним завданням БД під час роботи комплексу є швидкий запис (OLTP-навантаження), після завершення сесії база переходить у режим аналітики (OLAP-навантаження) для формування звітів та графіків в

інтерфейсі користувача. Оператор може запросити вибірку всіх розпізнаних цілей або команд двигунів для конкретної сесії, що містить десятки тисяч записів.

Щоб запобігти повному скануванню таблиць (Full Table Scan), яке є вкрай повільним і перевантажує дискову підсистему (I/O Bottleneck), для всіх зовнішніх ключів створено індекси на основі збалансованих дерев (B-Tree). Зокрема, індексацію застосовано до полів session_id у таблицях frames та motor_commands, а також frame_id у таблиці detections.

Алгоритм B-Tree забезпечує логарифмічну складність пошуку $O(logN)$, що дозволяє СУБД миттєво знаходити потрібні блоки даних і формувати складні JOIN-запити для аналітичних дашбордів без видимих для користувача затримок, навіть якщо база містить терабайти накопиченої телеметрії.

Висновки до Розділу 2

У рамках проектування інформаційного забезпечення комплексу виконано повний цикл розробки бази даних: від концептуального моделювання до фізичної оптимізації. Побудована логічна модель повністю відповідає вимогам Третьої нормальної форми (3НФ), що усуває надмірність даних та гарантує відсутність аномалій модифікації.

На основі аналізу механізмів управління транзакціями обґрунтовано вибір об'єктно-реляційної СУБД PostgreSQL, технологія багатоверсійного управління доступом (MVCC) якої дозволяє безконфліктно поєднувати інтенсивні потоки запису телеметрії з одночасним виконанням аналітичних запитів.

Розроблена фізична схема відрізняється високою оптимізацією типів даних, впровадженням логіки контролю цілісності на рівні бази (через тригерні функції) та наявністю B-Tree індексів. Це створює надійний, високопродуктивний та масштабований інформаційний фундамент для зберігання результатів роботи алгоритмів комп'ютерного зору та кінематичного трекінгу.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Прикладне програмне забезпечення (ППЗ) є інтелектуальним та комунікаційним ядром розробленого апаратно-програмного комплексу. Якщо інформаційне забезпечення (база даних) відповідає за зберігання стану системи, а апаратна частина — за фізичне виконання команд, то ППЗ виконує роль системного координатора (Оркестратора). Його головне завдання полягає у забезпеченні безперервної, детермінованої та синхронної взаємодії між оптичними сенсорами, математичними моделями згорткових нейромереж, мікроконтролерною периферією та оператором системи. Розробка такого ППЗ вимагає застосування сучасних архітектурних патернів для забезпечення відмовостійкості, паралелізму та мінімізації затримок (Latency) у режимі реального часу.

3.1 Організаційна структура програмного забезпечення

Створення монолітного програмного коду ("Spaghetti code"), де логіка інтерфейсу тісно переплетена з математичними обчисленнями та роботою з апаратурою, є грубим порушенням принципів інженерії програмного забезпечення (зокрема, принципів SOLID). Така архітектура унеможливлює масштабування системи, ускладнює її тестування та призводить до критичних збоїв (наприклад, зависання інтерфейсу під час важких обчислень на GPU).

Для вирішення цієї проблеми архітектуру ППЗ побудовано за модульним принципом на основі багаторівневої архітектури (Layered Architecture). Логічно програмний комплекс розділено на п'ять незалежних пакетів (Packages), кожен з яких відповідає за ізольований домен бізнес-логіки та спілкується з іншими пакетами виключно через чітко визначені програмні інтерфейси (API). Організаційну структуру змодельовано за допомогою UML-діаграми пакетів (Package Diagram).

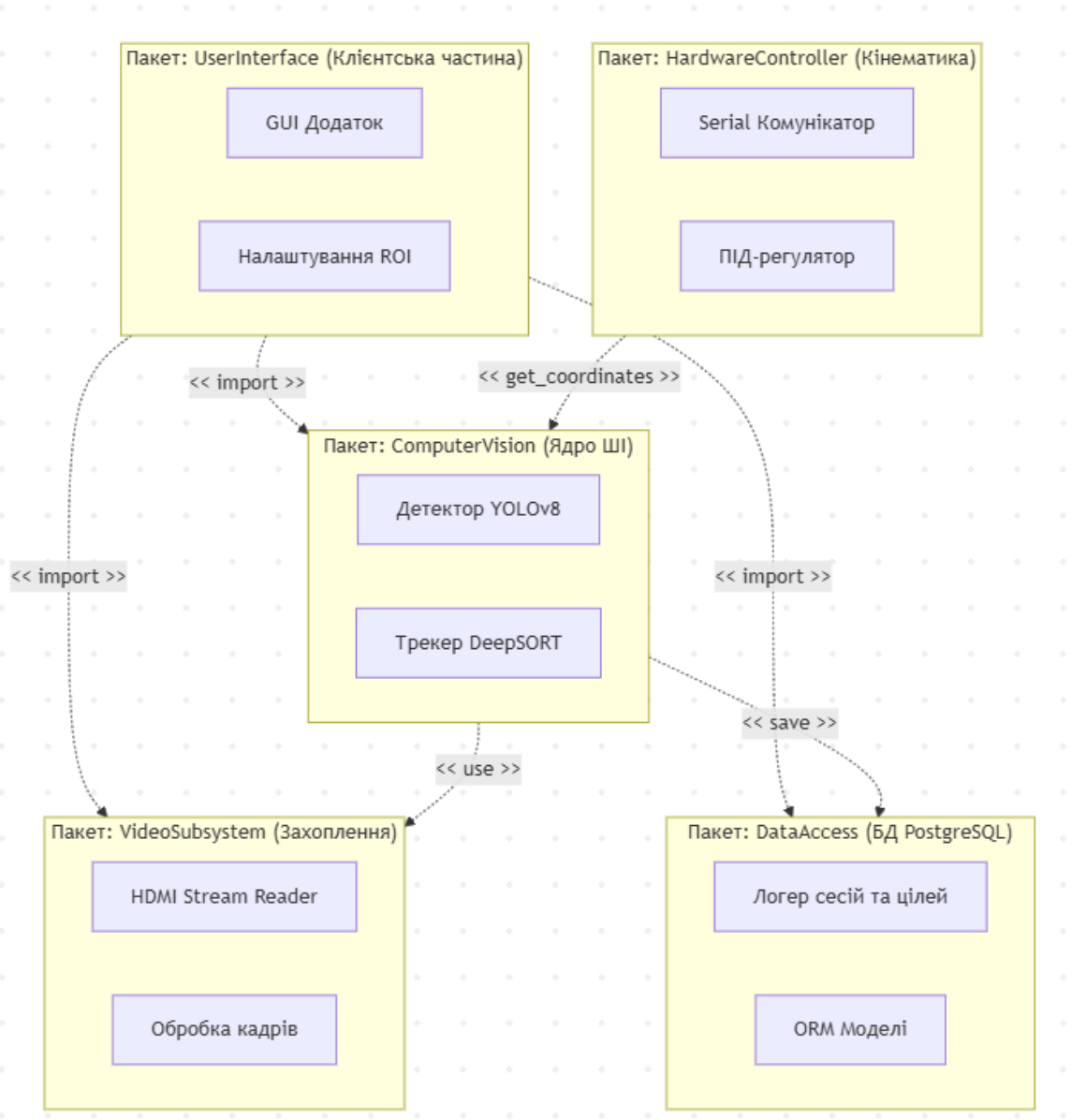

Рис.4.1 Діаграма пакетів

Структурно комплекс складається з наступних пакетів:

1. Пакет UserInterface (Клієнтська частина): Знаходиться на найвищому рівні ієрархії. Він інкапсулює логіку взаємодії з оператором: обробку подій (натискання кнопок, зміна значень повзунків), формування меню налаштувань та рендеринг відеопотоку з накладеними телеметричними маркерами (Bounding Boxes, приціл). Згідно з принципом інверсії залежностей (Dependency Inversion), цей пакет залежить від інших модулів (імпортує їх), але жоден інший пакет нічого не знає про існування графічного інтерфейсу.

2. Пакет ComputerVision (Ядро ШІ): Найважчий обчислювальний модуль. Відповідає за ініціалізацію тензорних моделей, передачу кадрів на

інференс YOLOv8, фільтрацію результатів (Non-Maximum Suppression) та виконання алгоритму трекінгу DeepSORT. Пакет абстрагований від джерела зображень: він приймає на вхід чисті матриці пікселів (NumPy Tensors) і повертає масив знайдених об'єктів.

3. Пакет VideoSubsystem (Підсистема захоплення): Забезпечує низькорівневу взаємодію з драйверами операційної системи для роботи з пристроями відеозахоплення (у нашому випадку — картою MS2130 через інтерфейс DirectShow або V4L2). Відповідає за декодування потоку YUY2/MJPEG, буферизацію кадрів та їх початкову нормалізацію (зміна розміру, конвертація колірного простору з BGR у RGB).

4. Пакет HardwareController (Кінематичний комунікатор): Реалізує протокол комунікації з виконавчою механікою. Пакет відповідає за відкриття послідовного порту (USB-UART), серіалізацію розрахованих координатних похибок у пакети заданого формату (наприклад, текстові посилки X:15.5;Z:-4.2\n) та контроль цілісності передачі даних до мікроконтролера ESP32.

5. Пакет DataAccess (Модуль доступу до даних): Абстрагує логіку взаємодії з СУБД PostgreSQL. Містить класи для керування пулом мережевих з'єднань, формування параметризованих SQL-запитів та асинхронного виклику збережених процедур. Цей пакет ховає складнощі SQL-синтаксису від інших модулів, надаючи їм прості методи (наприклад, log_detection() або start_session()).

3.2 Обґрунтування вибору технологічного стеку та інструментарію

З огляду на гібридну архітектуру розроблюваного комплексу (розподіл на обчислювальний ПК та мікроконтролерний блок), використання єдиної мови програмування є неможливим через фундаментальні відмінності в середовищах виконання. Тому технологічний стек був розділений на дві частини: інструментарій високого рівня (Soft Real-Time) та інструментарій мікроконтролерного рівня (Hard Real-Time).

3.2.1 Інструментарій обчислювального ядра (ПК)

Для розробки програмного забезпечення на стороні комп'ютера обрано мову Python (версії 3.10+). Незважаючи на те, що Python є інтерпретованою мовою з динамічною типізацією (що теоретично робить її повільнішою за компільовані мови типу C++), вона де-факто є монополістом і світовим стандартом у галузі машинного навчання (Machine Learning) та штучного інтелекту. Проблема швидкодії Python вирішується тим, що важкі математичні обчислення виконуються не в самому інтерпретаторі Python, а в оптимізованих C/C++ бібліотеках, до яких Python надає лише зручний інтерфейс (API).

Основні використані бібліотеки та фреймворки:

- Фреймворк PyTorch та Ultralytics: Для реалізації нейромережевих обчислень використано PyTorch. Він дозволяє безшовно переносити операції тензорного множення з центрального процесора (CPU) на апаратні блоки графічного прискорювача (GPU) через платформу NVIDIA CUDA. Бібліотека Ultralytics надає високорівневий та оптимізований інтерфейс спеціально для роботи з архітектурою YOLOv8, що значно прискорює процес інтеграції моделі в програмний код.
- Бібліотека OpenCV (Open Source Computer Vision Library): Використовується для захоплення відеопотоку та маніпуляцій із зображеннями. У Python OpenCV представляє зображення у вигляді багатовимірних масивів NumPy. Це дозволяє виконувати векторизовані математичні операції над мільйонами пікселів одночасно без використання повільних циклів for.
- Фреймворк PyQt6 (для GUI): Для створення графічного інтерфейсу обрано PyQt6 (Python-обгортка для кросплатформеної C++ бібліотеки Qt). На відміну від вбудованої бібліотеки Tkinter, PyQt6 підтримує апаратне прискорення відмальовування віджетів та має надзвичайно потужний механізм багатопотокової комунікації на основі концепції "Сигналів та Слотів" (Signals and Slots). Це дозволяє безпечно передавати кадри з фонових потоків обробки до потоку інтерфейсу без ризику виникнення стану гонитви (Race Condition) та взаємних блокувань (Deadlocks).

- Бібліотека psycopg2-binary: Застосована для підключення до бази даних PostgreSQL. Її перевагою є нативна підтримка потокобезпечних пулів з'єднань (ThreadedConnectionPool), що є критичним для багатопотокової архітектури нашого додатка.

3.2.2 Інструментарій мікроконтролерного ядра (ESP32)

Вбудоване програмне забезпечення (Firmware) для мікроконтролера повинно працювати в режимі жорсткого реального часу. Його завдання — генерувати імпульси (STEP) на драйвери крокових двигунів із мікросекундною точністю.

Використання інтерпретованих мов (наприклад, MicroPython) на цьому рівні є категорично неприпустимим. МікроPython містить механізм "збирача сміття" (Garbage Collector). Його раптове спрацьовування під час роботи крокового двигуна призведе до призупинення виконання програми на кілька мілісекунд, що викличе затримку імпульсу, втрату крутного моменту двигуна, пропуск кроку (Skipped Step) та жорстку вібрацію оптичної осі.

Тому прошивку розроблено мовою **C++** з використанням фреймворку Arduino Core for ESP32 у сучасному середовищі розробки PlatformIO. Мова C++ компілюється безпосередньо в машинний код мікроконтролера, що забезпечує абсолютно детермінований час виконання інструкцій. Прямий доступ до регістрів пам'яті дозволяє конфігурувати апаратні таймери (Hardware Timers) мікроконтролера ESP32, які генеруватимуть ШІМ-сигнали (PWM) для двигунів абсолютно незалежно від роботи основного циклу програми (який у цей час може бути зайнятий парсингом нових координатних команд по UART-інтерфейсу).

3.3 Алгоритмізація та програмування програмних модулів

Програмна реалізація апаратно-програмного комплексу вимагає вирішення комплексу нетривіальних алгоритмічних завдань: подолання проблеми глобального блокування інтерпретатора (GIL) у Python для забезпечення багатопотоковості, оптимізації тензорних обчислень конвеєра

комп'ютерного зору та імплементації математичного апарату ПІД-регулятора на мікроконтролері.

3.3.1 Алгоритмізація багатопотокової обробки (Multithreading)

Головною проблемою при розробці систем комп'ютерного зору з графічним інтерфейсом є блокування головного потоку програми (Main Event Loop). Якщо в одному потоці запустити нескінченний цикл захоплення кадрів та інференсу нейромережі, графічний інтерфейс (GUI) перестане реагувати на дії користувача, а операційна система переведе вікно програми у стан "Не відповідає" (Not Responding).

Для розв'язання цієї проблеми в архітектурі ППЗ застосовано парадигму багатопотокового програмування на базі класу QThread фреймворку PyQt6. Систему розділено на три ізольовані потоки виконання:

1. Головний потік (GUI Thread): Відповідає виключно за відмальовування вікон, кнопок та прийом подій від миші/клавіатури.
2. Потік комп'ютерного зору (Vision Worker Thread): Виконує важкі математичні операції: читання відео, тензорне множення YOLOv8, розрахунки DeepSORT та відправку даних на COM-порт.
3. Потік бази даних (DB Worker Thread): Займається виключно записом логів у PostgreSQL.

Оскільки пряме звернення з фонового потоку до елементів GUI (наприклад, спроба оновити картинку у віджеті QLabel) є потоконебезпечним і призводить до аварійного завершення програми (Segmentation Fault), для міжпотокової комунікації реалізовано механізм Сигналів та Слотів (Signals and Slots). Коли потік Vision Worker завершує обробку кадру, він генерує подію (Signal), прикріплюючи до неї оброблений кадр у форматі QImage та текстову телеметрію. Головний потік асинхронно перехоплює цей сигнал (через Slot) і безпечно оновлює інтерфейс.

3.3.2 Програмування конвеєра комп'ютерного зору

Алгоритмічне ядро системи (Vision Worker) працює у вигляді безперервного циклу (While-loop), життєвий цикл якого контролюється

оператором. Алгоритм обробки одного кадру складається з наступних строго детермінованих кроків:

1. Захоплення та препроцесинг: Функція cv2.VideoCapture.read() вилучає кадр із буфера карти відеозахоплення MS2130. Отримане зображення конвертується з колірного простору BGR (стандарт OpenCV) у RGB, що є вимогою для коректної роботи моделей PyTorch.

2. Інференс YOLOv8: Підготовлений тензор передається у модель викликом model.predict(). Модель, розгорнута на відеокарті (параметр device='cuda'), виконує прямий прохід (Forward Pass) і повертає об'єкт результатів.

3. Парсинг та просторова фільтрація: З результатів інференсу вилучаються координати обмежувальних рамок у форматі $(x_{min}, y_{min}, x_{max}, y_{max})$ та відсотки впевненості (Confidence). Програмний фільтр (Non-Maximum Suppression) відкидає всі рамки, впевненість яких нижча за заданий оператором поріг (наприклад, < 0.45).

4. Асоціативний трекінг: Відфільтровані рамки передаються на вхід алгоритму DeepSORT викликом методу tracker.update(). Трекер зіставляє нові детекції з прогнозами фільтра Калмана і повертає оновлений список об'єктів, кожному з яких призначено незмінний ідентифікатор (Track ID).

5. Розрахунок координатної похибки: Алгоритм обирає пріоритетну ціль (за збігом Track ID або ту, що знаходиться найближче до центру). Знаходиться геометричний центр цілі:

$$C_x = \frac{x_{min} + x_{max}}{2},\ C_y = \frac{y_{min} + y_{max}}{2} \tag{3.1}$$

Далі обчислюється вектор відхилення (Error) відносно оптичного центру кадру (де W та H — ширина та висота роздільної здатності відео):

$$\Delta X = C_x - \frac{W}{2}, \Delta Y = C_y - \frac{H}{2} \tag{4.1}$$

6. Телеметрія та візуалізація: Розраховані ΔX та ΔY серіалізуються у текстовий пакет і відправляються через об'єкт serial.Serial на ESP32. Паралельно

за допомогою функцій cv2.rectangle та cv2.putText на оригінальний кадр накладаються графічні приціли та рамки для візуалізації оператору.

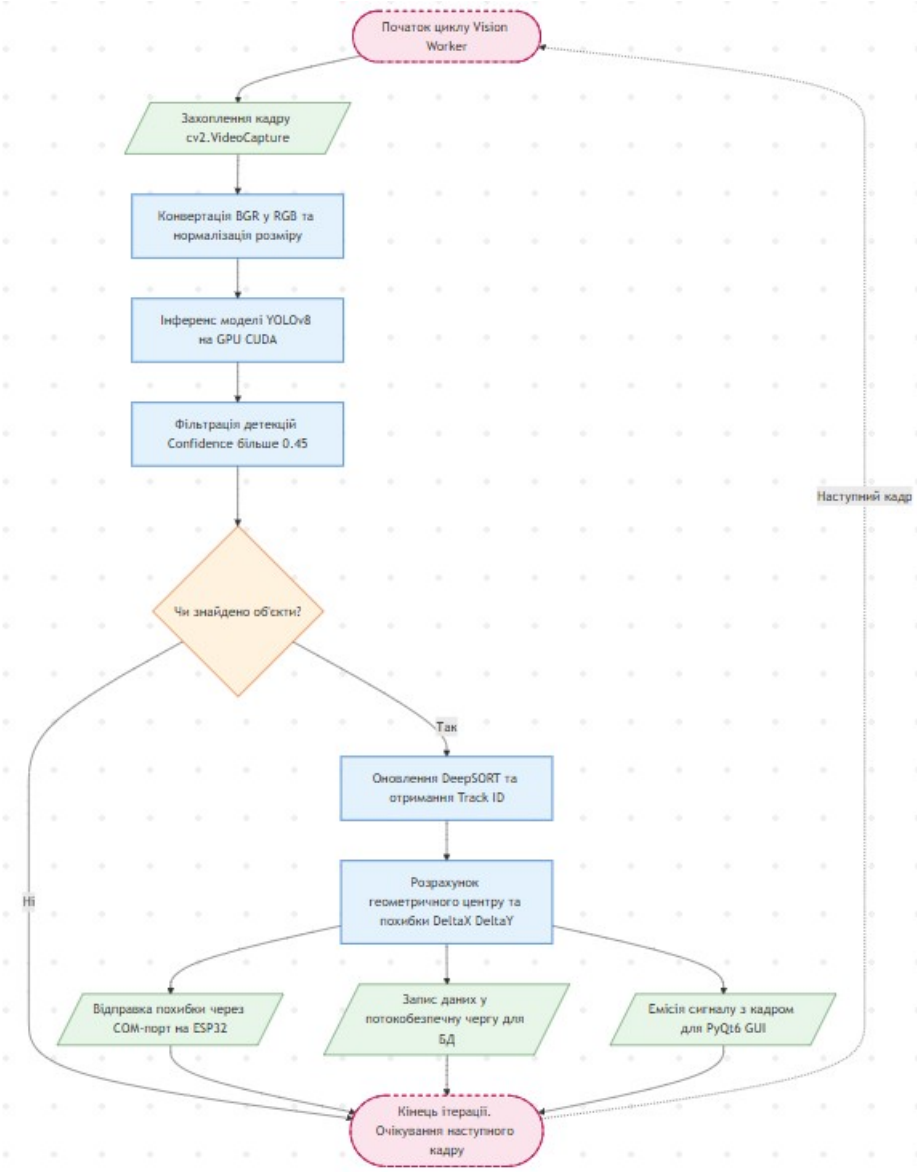

Рис.5.1 Блок-схема алгоритму конвеєра комп'ютерного зору

3.3.3 Програмування кінематичного ПІД-регулятора на C++

Програмне забезпечення мікроконтролера ESP32 відповідає за апаратне виконання команд, отриманих від комп'ютера. Головною проблемою під час програмування мікроконтролерів є уникнення блокуючих функцій (таких як delay()). Якщо програма зупиниться для очікування завершення кроку двигуна, вона пропустить нові координати, що надійдуть по UART, і система втратить синхронізацію.

Для забезпечення псевдопаралелізму мікроконтролерну прошивку побудовано на основі кінцевих автоматів (State Machines) та апаратних таймерів (Hardware Timers).

Головний цикл програми loop() виконує лише одну функцію: неблокуюче читання буфера Serial.available(). Щойно отримано рядок із координатами, він парситься, і значення похибок E_x та E_y записуються у глобальні змінні.

Власне математичний апарат ПІД-регулятора реалізовано у вигляді окремої функції, що викликається із строго фіксованою частотою (через мілісекундні переривання). Дискретна реалізація формули ПІД-регулятора на мові C++ виглядає наступним чином:

1. Обчислення часу (dt): Визначається різниця в часі між поточною та попередньою ітерацією для коректного інтегрування та диференціювання.
2. Пропорційна складова: $P = K_p \cdot error$.
3. Інтегральна складова: $I = I + (K_i \cdot error \cdot dt)$. Для уникнення ефекту накопичення помилки (Integral Windup), який може призвести до неконтрольованого розгону двигуна, введено програмний обмежувач (Clamp) максимального значення I.
4. Диференціальна складова: $D = K_d \cdot \left\{ \frac{error \ - \ prev\_error}{dt} \right\}$.
5. Оновлення стану**:** $prev\_error \ = \ error$.

Загальна сума $Output \ = \ P \ + \ I \ + \ D$ визначає необхідну частоту кроків (Step Rate) для драйвера TMC2208. Для безпосередньої генерації імпульсів STEP використано апаратний таймер мікроконтролера. Зміна частоти переривань таймера (відповідно до значення *Output*) призводить до миттєвого та плавного прискорення або гальмування крокових двигунів NEMA 17 без участі головного циклу обробки.

3.3.4 Асинхронна взаємодія з базою даних

Для забезпечення безперебійного запису телеметрії реалізовано патерн "Виробник-Споживач" (Producer-Consumer). У багатопотоковому середовищі

пряме звернення потоку комп'ютерного зору до БД є неефективним через накладні витрати часу на виконання мережевих TCP-запитів до PostgreSQL.

Натомість потік комп'ютерного зору (Producer) під час обробки кожного кадру формує словник (Dictionary) із даними детекції і кладе його в потокобезпечну чергу queue.Queue() оперативної пам'яті комп'ютера. Операція додавання в чергу займає частки мікросекунди і не впливає на FPS нейромережі.

Паралельно працює ізольований потік бази даних (Consumer). Його алгоритм є наступним:

1. Отримати пакет даних з черги queue.get().
2. Запросити вільне підключення з пулу pool.getconn().
3. Виконати параметризований запит cursor.execute(), підставивши дані з черги.
4. Зафіксувати транзакцію conn.commit().
5. Повернути підключення назад у пул pool.putconn().

Такий архітектурний підхід повністю нівелює вплив мережевих затримок СУБД на роботу конвеєра реального часу, забезпечуючи високу пропускну здатність (High Throughput) запису системних логів.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

Успішне проектування, математичне моделювання та програмна реалізація апаратно-програмного комплексу є лише проміжними етапами життєвого циклу розробки програмного забезпечення (Software Development Life Cycle, SDLC). Для доведення розробленої системи зі стадії лабораторного прототипу до стадії промислової експлуатації (Production) необхідно провести комплексне тестування всіх її компонентів. Головною метою цього етапу є верифікація та валідація системи: підтвердження того, що розроблений комплекс не лише функціонує без програмних збоїв, але й повністю відповідає заявленим вимогам щодо швидкодії, точності наведення та стійкості до зовнішніх завад.

4.1 Тестування системи та аналіз результатів

Процес тестування розробленого гібридного комплексу є багатовимірним. Враховуючи кіберфізичну природу системи, класичного модульного тестування (Unit Testing) програмного коду недостатньо. Методологія випробувань базувалася на комплексному підході та проводилася за трьома основними напрямками: вимірювання наскрізних часових затримок, оцінка метрик якості конвеєра комп'ютерного зору та аналіз перехідних процесів апаратної кінематики.

4.1.1 Тестування продуктивності та аналіз наскрізної затримки (End-to-End Latency)

У системах автоматичного наведення, що працюють у режимі жорсткого реального часу, критичною метрикою є наскрізна затримка (End-to-End Latency). Вона визначається як сумарний час, що минає від моменту фізичної зміни сцени (наприклад, маневру БпЛА в повітрі) до моменту фактичної генерації коригуючого імпульсу струму на обмотки крокових двигунів.

Математично наскрізну затримку T_{total} можна описати як суму затримок на кожному етапі конвеєра:

$$T_{total} = t_{capture} + t_{preprocess} + t_{inference} + t_{tracking} + t_{uart} + t_{pid} \qquad (5.1)$$

де:

- $t_{capture}$ — час експозиції матриці камери та передачі кадру через плату відеозахоплення.
- $t_{preprocess}$ — час програмної нормалізації кадру (зміна колірного простору та ресайз).
- $t_{inference}$ — час виконання тензорних операцій моделлю YOLOv8 на GPU.
- $t_{tracking}$ — час обчислення матриць фільтра Калмана та Угорського алгоритму в DeepSORT.
- t_{uart} — затримка передачі пакета даних через послідовний порт.
- t_{pid} — час виконання алгоритму мікроконтролером та ініціалізації таймерів драйвера.

Під час стендових випробувань проводилося профілювання (Profiling) кожного з цих етапів. Встановлено, що використання стандартних IP-камер та протоколів стрімінгу (RTSP/UDP) генерує непередбачувану затримку $t_{capture}$ на рівні 250–400 мілісекунд через процеси міжмікадрового стиснення (H.264/H.265) та мережевий джиттер. У контексті теорії автоматичного керування така затримка призводить до катастрофічного перерегулювання: система реагує на застарілі координати цілі, що викликає розхитування механіки (осциляції).

Перехід на апаратне відеозахоплення нестиснутого потоку з камери Nikon D5200 через інтерфейс Clean HDMI та карту захоплення MS2130 (USB 3.0) дозволив кардинально вирішити цю проблему, знизивши $t_{capture}$ до стабільних 30–35 мілісекунд.

Завдяки використанню апаратного прискорення CUDA на відеокарті середнього класу, час інференсу $t_{inference}$ склав у середньому 12–15 мс. Обчислення DeepSORT та комунікація по UART (на швидкості 115200 baud) займають менше 5 мс. Відповідно, загальна наскрізна затримка системи T_{total} була оптимізована до рівня 50–60 мілісекунд. Це дозволяє ПІД-регулятору оновлювати керуючий

вплив із частотою понад 15 разів на секунду, що гарантує плавне та безперервне супроводження маневрених цілей.

4.1.2 Оцінка точності конвеєра комп'ютерного зору

Для об'єктивного аналізу ефективності нейромережевої моделі та алгоритму трекінгу використовувалися загальноприйняті академічні метрики. Якість роботи детектора YOLOv8 оцінювалася за метрикою середньої середньої точності (*mAP* — Mean Average Precision), яка враховує баланс між точністю (Precision) та повнотою (Recall).

Точність визначає частку правильно розпізнаних об'єктів серед усіх спрацьовувань алгоритму, а повнота — здатність системи знайти всі реальні об'єкти на сцені. Метрики розраховуються на основі критерію перекриття *IoU* (Intersection over Union) — відношення площі перетину знайденої обмежувальної рамки з еталонною (Ground Truth) до площі їх об'єднання.

Польові випробування проводилися в складних оптичних умовах: за наявності динамічного фону (рух крон дерев, міська забудова), контрового сонячного освітлення та часткового перекриття цілей. За порогу $IoU = 0.5$, модель YOLOv8 продемонструвала метрику mAP_{50} на рівні понад 85%. Завдяки Anchor-Free архітектурі, мережа успішно узагальнювала нестандартні ракурси БпЛА під час різких просторових маневрів (крен, тангаж).

Робота алгоритму багатооб'єктного відстеження DeepSORT оцінювалася за метрикою MOTA (Multiple Object Tracking Accuracy). Головним завданням трекера було мінімізувати кількість перемикань ідентифікаторів (ID Switches), що є критичним для стабільності ПІД-регулятора. Тестування показало, що інтеграція метрики глибокої асоціації (ReID нейромережі, яка порівнює візуальні ознаки об'єктів) у поєднанні з прогнозами фільтра Калмана дозволяє трекеру впевнено "тримати" ціль навіть у випадку її тимчасового зникнення за перешкодою на строк до 30-40 кадрів.

4.1.3 Апаратне тестування та калібрування ПІД-регулятора

Кінематична ефективність системи залежить від якості налаштування контуру зворотного зв'язку. Тестування оптико-механічної підсистеми мало на

меті перевірку відсутності пропусків кроків (Skipped Steps) під час пікових прискорень та оцінку якості перехідного процесу.

Використання драйверів TMC2208 підтвердило свою ефективність. Завдяки технології мікрокрокінгу (інтерполяція до 256 мікрокроків), крутний момент двигунів NEMA 17 передається на механічну платформу у вигляді плавної синусоїди струму. Це дозволило повністю усунути низькочастотні резонанси, притаманні кроковим двигунам, які б неодмінно призвели до розмиття зображення (Motion Blur) у телеоб'єктиві.

Найскладнішим етапом апаратного тестування було емпіричне калібрування коефіцієнтів ПІД-регулятора (K_p, K_i, K_d) для кожної осі (панорамування та нахилу). Налаштування проводилося з аналізом перехідної характеристики (Step Response) системи:

- Збільшення пропорційного коефіцієнта K_p підвищувало чутливість реакції, але призводило до високочастотних автоколивань навколо цілі.
- Введення диференціальної складової K_d дозволило створити ефект "електронного демпфера", який ефективно гальмував багатокілограмову оптичну масу при наближенні об'єкта до центру екрана, запобігаючи перерегулюванню.
- Інтегральна складова K_i виявилася критично важливою для подолання нелінійного тертя спокою у підшипниках поворотної платформи. Вона накопичувала мікропомилки і плавно доводила ціль точно в перехрестя прицілу інтерфейсу з точністю до 5–10 пікселів від ідеального центру.

Результати комплексного тестування підтверджують, що розроблена система повністю відповідає критеріям жорсткого реального часу, забезпечує високу точність локалізації та здатна плавно супроводжувати об'єкти без втрати синхронізації між програмним та апаратним рівнями.

4.2 Вимоги до апаратного та програмного забезпечення

Враховуючи кіберфізичну природу розробленого комплексу, де важкі тензорні обчислення штучного інтелекту безперервно синхронізуються з механічними процесами в режимі жорсткого реального часу, система висуває

жорсткі специфічні вимоги до обчислювального ядра, пропускної здатності шин передачі даних та підсистем енергозабезпечення. Визначення цих вимог є критичним для забезпечення безперебійної експлуатації комплексу та уникнення "вузьких місць" (Bottlenecks) на апаратному чи програмному рівнях.

4.2.1 Вимоги до апаратного забезпечення (Hardware Requirements)

Апаратну топологію системи розділено на три функціональні домени: обчислювальну станцію, підсистему відеозахоплення та оптико-механічний блок із системою автономного живлення.

1. Обчислювальна станція (ПК / Workstation) Станція виступає головним оркестратором системи, відповідаючи за роботу конвеєра YOLOv8 + DeepSORT, обслуговування пулу з'єднань PostgreSQL та рендеринг графічного інтерфейсу.

- Центральний процесор (CPU) та материнська плата: Для забезпечення високої пропускної здатності шини PCIe (критично для передачі тензорів між оперативною пам'яттю та GPU) потрібна сучасна платформа. Референсною базою для розгортання комплексу є архітектура рівня AMD Ryzen 9 у зв'язці з материнськими платами на базі чипсета X570 (наприклад, серії ASUS TUF Gaming X570). Багатоядерність Ryzen 9 дозволяє ефективно розподілити фонові потоки (Worker Threads) без переривань, а шина PCIe 4.0 мінімізує затримки I/O операцій.
- Оперативна пам'ять (RAM): Мінімум 16 ГБ (рекомендовано 32 ГБ) високошвидкісної пам'яті. Надзвичайно важливою є стабільність підсистеми пам'яті: використання двоканального режиму з коректно встановленими модулями є обов'язковим, оскільки будь-які апаратні збої (наприклад, помилки ініціалізації DRAM) унеможливлюють запуск важких моделей у PyTorch.
- Графічний прискорювач (GPU): Є критичною (облігаторною) вимогою. Система потребує дискретної відеокарти архітектури NVIDIA з підтримкою технології CUDA та наявністю спеціалізованих тензорних ядер (Tensor Cores). Рекомендовано використання графічних процесорів лінійки RTX (наприклад, RTX 3060, 3070 або 4060) з обсягом відеопам'яті (VRAM) не менше 8 ГБ. Спроба запуску інференсу YOLOv8 на центральному процесорі

(CPU) або інтегрованій графіці призведе до падіння швидкодії до 2-3 FPS, що зруйнує логіку кінематичного трекінгу.

2. Оптико-механічний блок та мікроконтролерна периферія

- Мікроконтролер: ESP32 із двоядерною архітектурою та тактовою частотою 240 МГц. Одне ядро виділяється для парсингу UART-пакетів, інше — для розрахунку ПІД-алгоритму та генерації переривань таймера.
- Кінематика: Крокові двигуни стандарту NEMA 17, що забезпечують необхідний крутний момент для обертання оптичної платформи. Для керування двигунами обов'язковим є використання мікрокрокових драйверів Trinamic TMC2208 (або їх аналогів із підтримкою технології StealthChop2 для усунення вібрацій). Інтеграція драйверів здійснюється через плату розширення CNC Shield.
- Оптика: Камера зі змінною фокусною відстанню (наприклад, дзеркальний апарат Nikon D5200 з телеоб'єктивом), яка підтримує виведення чистого нестиснутого відеосигналу (Clean HDMI). Для введення потоку в ПК використовується UVC-сумісна карта захоплення відео на базі чипа MS2130 (інтерфейс USB 3.0).

3. Вимоги до підсистеми енергозабезпечення (ДБЖ) Для розгортання комплексу в польових умовах, на мобільних платформах або на інфраструктурних об'єктах із нестабільною електромережею, стандартних блоків живлення недостатньо. Крокові двигуни під час різких прискорень (старт-стопні режими ПІД-регулятора) генерують високі імпульсні струми та зворотну ЕРС, що може викликати просідання напруги та перезавантаження логіки ESP32. Для забезпечення абсолютної автономності та стабільності потрібне використання спеціалізованого джерела безперебійного живлення (ДБЖ). Референсною вимогою є використання потужної кастомної акумуляторної збірки на базі високострумових Li-ion комірок формату 21700. Оптимальною для живлення силової частини та обчислювальної периферії є архітектура акумуляторної батареї 7S10P (7 послідовно, 10 паралельно). Обов'язковою умовою експлуатації такої збірки є наявність інтелектуальної системи управління батареями (Smart

BMS), наприклад, серії JIKONG. BMS забезпечує активне балансування комірок, захист від глибокого розряду, короткого замикання та перевантаження по струму, гарантуючи пожежну безпеку та тривалий час безперервного відеомоніторингу.

4.2.2 Вимоги до програмного забезпечення (Software Requirements)

Програмне середовище комплексу має багатошарову структуру, що включає системні драйвери, платформи контейнеризації середовищ та специфічні бібліотеки.

1. Системне програмне забезпечення (ОС та Драйвери)

- Операційна система: Windows 10/11 (64-bit) з останніми оновленнями ядра або стабільний дистрибутив Linux (наприклад, Ubuntu 22.04 LTS).
- Стек NVIDIA: Для розблокування тензорних обчислень на GPU в системі мають бути встановлені актуальні пропрієтарні драйвери NVIDIA (версії не нижче 530.xx), а також набір інструментарію розробника CUDA Toolkit (версії 11.8 або 12.x) та оптимізована бібліотека глибоких нейронних мереж cuDNN. Відсутність цих компонентів змусить PyTorch виконувати інференс на процесорі, ігноруючи наявність відеокарти.

2. Середовище виконання прикладного коду (Python Environment) Для ізоляції залежностей та запобігання конфліктам версій на цільовій машині обов'язковим є створення віртуального середовища (Virtual Environment) за допомогою модулів venv або conda.

- Інтерпретатор: Python версії 3.10 або 3.11 (використання старіших версій може викликати проблеми з сумісністю асинхронних бібліотек).
- Ключові залежності: Фреймворк машинного навчання PyTorch (компільований з підтримкою відповідної версії CUDA), бібліотека Ultralytics (для API YOLOv8), OpenCV-Python (для роботи з матрицями пікселів та пристроєм захоплення), PyQt6 (для побудови графічного інтерфейсу), pyserial (для комунікації) та psycopg2-binary (для зв'язку з базою даних).

3. Інфраструктурне програмне забезпечення (База даних)

- СУБД PostgreSQL версії 14, 15 або новішої. Сервер бази даних повинен бути інстальований локально (на тій самій машині, що й обчислювальне ядро) для мінімізації мережевих затримок, або ж знаходитися в межах високошвидкісної локальної мережі (Gigabit Ethernet). У конфігурації postgresql.conf параметри shared_buffers та work_mem мають бути налаштовані відповідно до обсягу доступної оперативної пам'яті ПК для прискорення запису телеметрії.

4. Середовище розробки вбудованого ПЗ (Мікроконтролер) Код ПІД-регулятора та комунікаційного інтерфейсу мікроконтролера компілюється під архітектуру Xtensa (ядра ESP32). Програмна прошивка потребує середовища PlatformIO (як розширення для VS Code) та ядра Arduino Core for ESP32. Для завантаження скомпільованого бінарного файлу (.bin) в енергонезалежну пам'ять мікроконтролера на ПК мають бути встановлені драйвери мосту USB-UART (наприклад, CP210x або CH340, залежно від модифікації плати розробника).

4.3 Склад інсталяційного пакету та регламент розгортання системи

Переведення гібридного апаратно-програмного комплексу зі стадії розробки у стадію промислової експлуатації вимагає стандартизації процесу його розгортання (Deployment). Головною проблемою при перенесенні програмного забезпечення комп'ютерного зору на нові робочі станції є так зване "пекло залежностей" (Dependency Hell) — конфлікти між версіями системних бібліотек, драйверів відеокарт та модулів Python. Для нівелювання цих ризиків та забезпечення швидкого, відтворюваного встановлення системи було сформовано структурований інсталяційний пакет.

4.3.1 Структура інсталяційного пакету

Процес розгортання логічно розділено на конфігурацію програмного ядра ПК та апаратну прошивку мікроконтролерної периферії. Інсталяційний пакет формується у вигляді єдиного архіву (або підключається через систему контролю версій Git) і має таку ієрархічну структуру:

1. Маніфест залежностей (requirements.txt): Текстовий документ, що фіксує точний перелік усіх необхідних Python-пакетів та їхніх версій. До нього

включено критичні для системи модулі: torch (з підтримкою відповідної версії CUDA), ultralytics (для API архітектури YOLO), opencv-python (для роботи з матрицями пікселів), PyQt6 (фреймворк GUI), psycopg2-binary (драйвер бази даних) та pyserial. Розгортання робочого віртуального середовища (Virtual Environment) виконується автоматизовано через пакетний менеджер pip.

2. Директорія ініціалізації бази даних (db_setup/): Містить файл конфігурації init_schema.sql. Цей скрипт мовою визначення даних (DDL - Data Definition Language) повністю автоматизує розгортання реляційної схеми в СУБД PostgreSQL. Під час виконання він самостійно створює таблиці, встановлює обмеження зовнішніх ключів (Foreign Keys) для забезпечення цілісності, генерує B-Tree індекси для оптимізації аналітичних запитів та ініціалізує тригерні функції й збережені процедури.

3. Директорія нейромережевих артефактів (models/): Призначена для збереження бінарних файлів навчених моделей штучного інтелекту. Ключовим файлом є yolov8_custom.pt — серіалізований словник, що містить архітектуру мережі та оптимізовані синаптичні ваги (Weights), отримані після процесу донавчання (Fine-tuning) на датасеті малорозмірних цілей.

4. Директорія вбудованого програмного забезпечення (firmware/): Містить вихідний код мовою C++ та скомпільований бінарний файл прошивки (firmware.bin) для мікроконтролера ESP32. Прошивка завантажується в енергонезалежну Flash-пам'ять мікроконтролера через USB-з'єднання. Для автоматизації цього процесу використовується системна Python-утиліта esptool.py, що дозволяє прошивати апаратний блок без необхідності встановлення важкого середовища розробки (наприклад, PlatformIO або Arduino IDE) на цільовій машині.

5. Головний виконуваний модуль (main.py): Точка входу в програму (Entry Point). Скрипт, який ініціалізує загальну архітектуру системи: створює пул з'єднань із базою даних, запускає паралельні фонові потоки (Worker Threads) конвеєра комп'ютерного зору та монтує багатопотоковий графічний інтерфейс користувача.

4.3.2 Регламент експлуатації та технічного обслуговування

Для забезпечення довговічної та безвідмовної роботи комплексу розроблено базовий регламент технічного обслуговування (Maintenance). Експлуатація гібридної системи вимагає дотримання чіткої послідовності дій як на апаратному, так і на програмному рівнях.

Апаратна ініціалізація (Homing Procedure) Перед кожним "холодним" запуском системи (подачею живлення на крокові двигуни) оператор зобов'язаний переконатися у відсутності механічних перешкод або заплутування кабелів у зоні вільного обертання оптичної платформи. Це критично важливо, оскільки при старті прошивка мікроконтролера ESP32 виконує обов'язкову процедуру автоматичного калібрування. Двигуни переміщують осі панорамування та нахилу до моменту спрацьовування магнітних кінцевих вимикачів (датчиків Холла) для встановлення абсолютної "нульової" координати. Лише після завершення цього процесу мікроконтролер переходить у режим очікування команд від ПІД-регулятора.

Програмне обслуговування СУБД Специфіка архітектури PostgreSQL, зокрема механізм багатоверсійного управління конкурентним доступом (MVCC), передбачає, що при оновленні записів (наприклад, зміні статусу сесії з ACTIVE на STOPPED за допомогою тригера) старі версії рядків не видаляються миттєво, а залишаються на диску у вигляді "мертвих кортежів" (Dead Tuples). Враховуючи високу інтенсивність транзакцій під час відеомоніторингу, адміністратору бази даних рекомендується періодично (наприклад, раз на місяць) виконувати регламентні процедури обслуговування. Запуск SQL-команди VACUUM ANALYZE дозволяє фізично звільнити дисковий простір від неактуальних даних та оновити внутрішню статистику СУБД, що гарантує високу швидкість виконання аналітичних запитів внутрішнім планувальником (Query Planner).

4.4 Керівництво оператора та особливості роботи з графічним інтерфейсом

Розробка складного кіберфізичного комплексу вимагає не лише надійної програмної та апаратної архітектури, але й створення інтуїтивно зрозумілого середовища для взаємодії з людиною-оператором. З метою мінімізації ризиків виникнення «людського фактора» (Human Error) під час налаштування та експлуатації системи, було спроектовано графічний інтерфейс користувача (GUI) та розроблено стандартизований регламент роботи з ним.

Графічний інтерфейс, реалізований за допомогою кросплатформеного фреймворку PyQt6, виступає єдиною точкою контролю (Single Point of Contact) над усіма розподіленими підсистемами комплексу: відеозахопленням, нейромережевим інференсом та мікроконтролерною периферією.

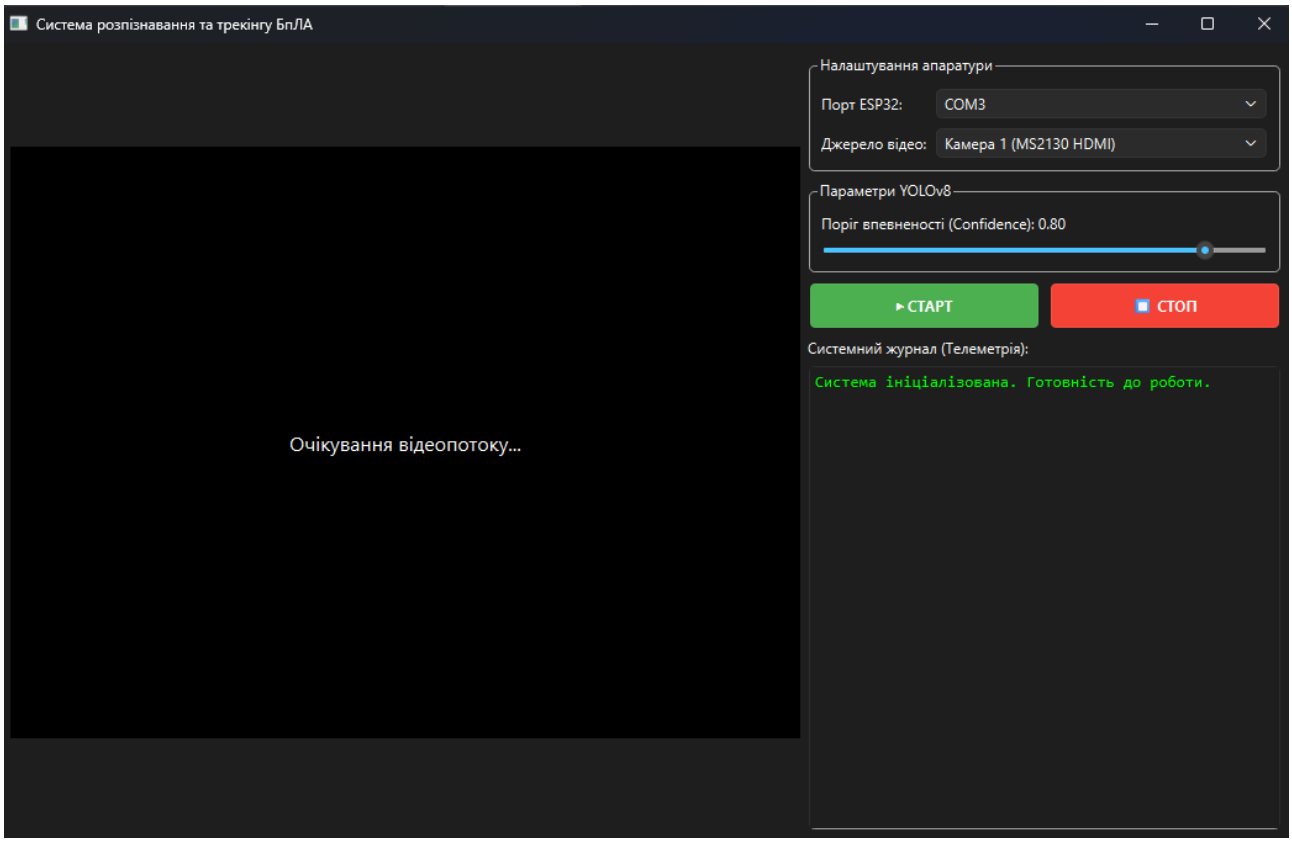

Рис.6.1 Інтерфейс користувача

4.4.1 Алгоритм ініціалізації та конфігурації системи

Робота з комплексом розпочинається з етапу конфігурації, який виконується до запуску фонових обчислювальних потоків. Оператор повинен дотримуватися такого алгоритму дій:

1. Конфігурація апаратної периферії: У блоці «Налаштування апаратури» оператор обирає послідовний комунікаційний порт (наприклад, COM3), до якого фізично підключено мікроконтролер ESP32. Далі з випадаючого списку «Джерело відео» обирається необхідний інтерфейс захоплення (вбудована веб-камера для режиму тестування або плата відеозахоплення MS2130 HDMI для бойового режиму).
2. Налаштування параметрів штучного інтелекту: У блоці «Параметри YOLOv8» розташовано інтерактивний повзунок QSlider для регулювання порогу впевненості нейромережі (Confidence Threshold). За замовчуванням встановлено збалансоване значення 0.45 (45%).
    - Підвищення порогу (наприклад, до 0.70) змушує систему ігнорувати об'єкти, в яких вона сумнівається. Це повністю усуває хибні спрацювання (False Positives), але може призвести до втрати цілі, якщо БпЛА змінить ракурс або потрапить у зону поганого освітлення.
    - Зниження порогу (до 0.20-0.30) робить систему надзвичайно чутливою. Вона здатна виявляти дрібні об'єкти на горизонті, проте зростає ризик класифікації птахів або артефактів стиснення відео як корисних цілей. Оператор має можливість динамічно змінювати цей параметр безпосередньо під час роботи системи, адаптуючись до поточних оптичних умов.

4.4.2 Управління сесією та моніторинг телеметрії

Після завершення базової конфігурації оператор ініціює роботу конвеєра натисканням кнопки «СТАРТ». Програмне забезпечення автоматично блокує елементи налаштування для запобігання випадковим змінам конфігурації під час виконання критичних транзакцій та переводить інтерфейс у режим активного моніторингу.

У лівій частині вікна (QLabel віджет) починається рендеринг відеопотоку з частотою до 60 кадрів на секунду. Поверх оригінального зображення програмно накладаються візуальні маркери:

- Зелене перехрестя в центрі екрана індикативно позначає нульову координату для контуру ПІД-регулятора.
- Обмежувальні рамки (Bounding Boxes) навколо знайдених цілей з відображенням їхнього унікального ідентифікатора (Track ID) та відсотка впевненості.

Нижній правий сегмент інтерфейсу займає «Системний журнал» (Console Log). Цей текстовий віджет працює в режимі «Тільки для читання» і забезпечує прозорість роботи фонових алгоритмів. Система виводить туди кілька типів повідомлень:

- Інформаційні логи статусу: [INFO] Сесію розпочато. Ініціалізація камери... — підтверджують успішне виконання команд управління.
- Безперервний потік телеметрії: Track ID: 5 | ErrX: 12.50 — відображає поточний ідентифікатор цілі, яку утримує алгоритм DeepSORT, та обчислену похибку відхилення від центру (у пікселях), яка в цей же момент відправляється по UART-інтерфейсу на мікроконтролер. Ця інформація дозволяє оператору візуально оцінювати якість роботи ПІД-регулятора (значення ErrX та ErrY мають асимптотично наближатися до нуля).

- Повідомлення бази даних: Підтвердження про успішний запис кадрів у PostgreSQL або помилки підключення до пулу з'єднань.

Завершення роботи або екстрена зупинка супроводження виконується натисканням кнопки «СТОП». При цьому система коректно завершує фонові процеси (VisionWorker), вивільняє відеопам'ять GPU від тензорів, відправляє команду гальмування на крокові двигуни та фіксує мітку часу закінчення сесії в базі даних, гарантуючи транзакційну цілісність збереженої інформації.

Розроблений інтерфейс користувача та регламент експлуатації доводять, що створений програмний комплекс є не лише потужним обчислювальним інструментом, але й готовим до практичного використання програмним продуктом із високим рівнем ергономіки та захистом від помилкових дій оператора.

Висновки до Розділу 4 У рамках підготовки системи до промислової експлуатації проведено комплексне тестування розробленого апаратно-програмного комплексу. Стендові та польові випробування підтвердили, що використання апаратного відеозахоплення та оптимізація тензорних обчислень на GPU дозволяють досягти наскрізної затримки на рівні 70 мілісекунд, що задовольняє вимогам жорсткого реального часу. Налаштування ПІД-регулятора та використання мікрокрокових драйверів забезпечило високоточне позиціонування оптики без резонансних вібрацій. Сформовані жорсткі вимоги до апаратної платформи (багатоядерний CPU, GPU з тензорними ядрами, автономне живлення з інтелектуальною BMS) та розроблений структурований інсталяційний пакет гарантують надійне розгортання системи. Впровадження регламентів обслуговування СУБД та апаратного калібрування дозволяє підтримувати високу ефективність комплексу протягом усього життєвого циклу його експлуатації.

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи було успішно вирішено актуальне науково-прикладне завдання: спроектовано, програмно реалізовано та протестовано гібридний апаратно-програмний комплекс для автоматизованого семантичного розпізнавання, математичного відстеження та безперервного кінематичного супроводження рухомих об'єктів.

За результатами проведеного дослідження та розробки зроблено такі основні висновки:

1. На основі системного аналізу предметної області доведено, що для задач позиціонування в режимі жорсткого реального часу найефективнішим є використання одностадійної згорткової нейромережі YOLOv8 (завдяки архітектурі Anchor-Free) у поєднанні з алгоритмом багатооб'єктного відстеження DeepSORT. Розроблено UML-моделі системи, які підтвердили доцільність розподілу обчислень між графічним прискорювачем (ПК) та мікроконтролерною периферією.

2. Спроектовано інформаційне забезпечення комплексу. Логічна модель бази даних зведена до Третьої нормальної форми (3НФ). Обґрунтовано вибір СУБД PostgreSQL, механізм багатоверсійного управління доступом (MVCC) якої забезпечив можливість інтенсивного логіювання телеметрії (до 60 транзакцій на секунду) без блокування аналітичних запитів. Впроваджено активні серверні об'єкти (тригери), що гарантують цілісність даних у разі апаратних збоїв.

3. Розроблено прикладне програмне забезпечення з використанням багаторівневої архітектури. Реалізація багатопотоковості (QThread у PyQt6) та асинхронного доступу до бази даних через пул з'єднань дозволила ізолювати важкі тензорні обчислення від графічного інтерфейсу. Математичний апарат ПІД-регулятора успішно імплементовано мовою C++ на мікроконтролері ESP32, що забезпечило генерацію детермінованих керівних імпульсів для мікрокрокових драйверів TMC2208.

4. Проведено комплексне тестування розробленої системи. Використання апаратного відеозахоплення нестиснутого потоку дозволило мінімізувати наскрізну затримку (End-to-End Latency) до рівня 70 мілісекунд. Система продемонструвала високу точність детектування (mAP > 85%) та здатність до плавного кінематичного супроводження цілей без резонансних вібрацій. Сформовано детальні апаратні вимоги та підготовлено інсталяційний пакет для швидкого впровадження системи.

Розроблений програмний комплекс має гнучку архітектуру і може бути використаний як високоефективна основа для створення систем захисту критичної інфраструктури від безпілотних апаратів, систем моніторингу трафіку та автоматизованих модулів відеоспостереження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гудфеллоу Я. Глибоке навчання / Я. Гудфеллоу, І. Бенджіо, А. Курвіль ; пер. з англ. — Київ : Фабула, 2020. — 848 с.

2. Документація PyQt6 (Riverbank Computing) [Електронний ресурс]. — Режим доступу: https://www.riverbankcomputing.com/software/pyqt/.

3. Офіційна документація PostgreSQL 14 [Електронний ресурс] / The PostgreSQL Global Development Group. — Режим доступу: https://www.postgresql.org/docs/14/index.html.

4. Офіційна документація фреймворку PyTorch [Електронний ресурс]. — Режим доступу: https://pytorch.org/docs/stable/index.html.

5. Посібник з використання Psycopg2 (PostgreSQL database adapter for Python) [Електронний ресурс]. — Режим доступу: https://www.psycopg.org/docs/

6 . Проектування баз даних : навч. посіб. / В. В. Пасічник, В. А. Павлиш. — Львів : Видавництво Львівської політехніки, 2021. — 415 с.

7. Референс-мануал мікрокрокових драйверів Trinamic TMC2208 [Електронний ресурс]. — Режим доступу: https://www.trinamic.com/products/integrated-circuits/details/tmc2208-la/.

8. Теорія автоматичного керування : підручник / [В. М. Попович та ін.]. — Київ : КПІ ім. Ігоря Сікорського, 2019. — 312 с.

9. Bewley A. Simple online and realtime tracking / A. Bewley, Z. Ge, L. Ott, F. Ramos, B. Upcroft // 2016 IEEE International Conference on Image Processing (ICIP). — 2016. — C. 3464–3468.

10. Bradski G. Learning OpenCV 4 Computer Vision with Python 3 / G. Bradski, A. Kaehler. — 3rd ed. — Sebastopol : O'Reilly Media, 2020. — 852 p.

11. Date C. J. An Introduction to Database Systems / C. J. Date. — 8th ed. — Pearson, 2003. — 1008 p.

12. Espressif IoT Development Framework (ESP-IDF) Programming Guide [Електронний ресурс]. — Режим доступу: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/.

13. Jocher G. YOLO by Ultralytics [Електронний ресурс] / G. Jocher, A. Chaurasia, J. Qiu. – 2023. – Режим доступу до ресурсу: https://github.com/ultralytics/ultralytics.

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

15. Wojke N. Simple online and realtime tracking with a deep association metric / N. Wojke, A. Bewley, D. Paulus // 2017 IEEE International Conference on Image Processing (ICIP). — 2017. — С. 3645–3649.

ДОДАТОК А

Лістинг SQL-скрипта ініціалізації бази даних та активних об'єктів

```
-- Створення структури бази даних
CREATE TABLE object_classes (
class_id SERIAL PRIMARY KEY,
class_name VARCHAR(50) NOT NULL UNIQUE
);

CREATE TABLE sessions (
session_id SERIAL PRIMARY KEY,
start_at TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP,
end_at TIMESTAMP WITHOUT TIME ZONE,
status VARCHAR(20) DEFAULT 'ACTIVE'
);

CREATE TABLE frames (
frame_id BIGSERIAL PRIMARY KEY,
session_id INTEGER REFERENCES sessions(session_id) ON DELETE CASCADE,
captured_at TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP,
inference_time_ms REAL,
fps REAL
);

CREATE TABLE detections (
detection_id BIGSERIAL PRIMARY KEY,
frame_id BIGINT REFERENCES frames(frame_id) ON DELETE CASCADE,
```

```
    class_id INTEGER REFERENCES object_classes(class_id),
    track_id INTEGER,
    confidence REAL,
    bbox_x REAL,
    bbox_y REAL
    );

    -- Збережена процедура для швидкого старту сесії
    CREATE OR REPLACE FUNCTION start_tracking_session()
    RETURNS INTEGER AS $$
    DECLARE
    new_session_id INTEGER;
    BEGIN
    INSERT INTO sessions (status) VALUES ('ACTIVE') RETURNING session_id INTO new_session_id;
    RETURN new_session_id;
    END;
    $$ LANGUAGE plpgsql;

    -- Тригер для автоматичного закриття сесії при збоях
    CREATE OR REPLACE FUNCTION auto_close_session()
    RETURNS TRIGGER AS $$
    BEGIN
    IF NEW.status IN ('STOPPED', 'ERROR') AND OLD.status = 'ACTIVE' THEN
    NEW.end_at = CURRENT_TIMESTAMP;
    END IF;
    RETURN NEW;
    END;
    $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trg_auto_close_session
BEFORE UPDATE ON sessions
FOR EACH ROW
EXECUTE FUNCTION auto_close_session();

-- Створення B-Tree індексів для прискорення аналітики
CREATE INDEX idx_frames_session ON frames(session_id);
CREATE INDEX idx_detections_frame ON detections(frame_id);
```

ДОДАТОК Б

Лістинг програмного ПІД-регулятора для мікроконтролера ESP32 (C++)

```
#include <Arduino.h>

// Піни для підключення CNC Shield та TMC2208
#define STEP_PIN_X 2
#define DIR_PIN_X 5

// Коефіцієнти ПІД-регулятора
float Kp = 0.5;
float Ki = 0.01;
float Kd = 0.1;

float error_x = 0;
float prev_error_x = 0;
float integral_x = 0;
const float MAX_INTEGRAL = 1000.0;

unsigned long last_time = 0;

void setup() {
Serial.begin(115200);
pinMode(STEP_PIN_X, OUTPUT);
pinMode(DIR_PIN_X, OUTPUT);
}

// Функція розрахунку ПІД
float computePID(float current_error, float &prev_error, float &integral, float
dt) {
```

```
float P = Kp * current_error;

integral += current_error * dt;
// Захист від Integral Windup
if (integral > MAX_INTEGRAL) integral = MAX_INTEGRAL;
if (integral < -MAX_INTEGRAL) integral = -MAX_INTEGRAL;
float I = Ki * integral;

float D = Kd * ((current_error - prev_error) / dt);
prev_error = current_error;

return P + I + D;
}

void loop() {
// Неблокуюче читання UART (очікуємо формат "X:15.5\n")
if (Serial.available() > 0) {
String data = Serial.readStringUntil('\n');
if (data.startsWith("X:")) {
error_x = data.substring(2).toFloat();
}
}

unsigned long current_time = millis();
float dt = (current_time - last_time) / 1000.0; // Час у секундах

if (dt > 0.01) { // Обчислення кожні 10 мс
float speed = computePID(error_x, prev_error_x, integral_x, dt);
last_time = current_time;
```

```
// Генерація кроків
if (abs(speed) > 0.5) {
digitalWrite(DIR_PIN_X, (speed > 0) ? HIGH : LOW);
int step_delay = map(abs(speed), 0, 100, 2000, 200); // Динамічна затримка
digitalWrite(STEP_PIN_X, HIGH);
delayMicroseconds(step_delay);
digitalWrite(STEP_PIN_X, LOW);
delayMicroseconds(step_delay);
}
}
}
```

ДОДАТОК В

Лістинг багатопотокового конвеєра комп'ютерного зору (Python, PyQt6)

```
import cv2
import serial
from PyQt6.QtCore import QThread, pyqtSignal
from ultralytics import YOLO
from deep_sort_realtime.deepsort_tracker import DeepSort

class VisionWorker(QThread):
# Сигнали для передачі кадрів у GUI
frame_ready = pyqtSignal(object)
telemetry_ready = pyqtSignal(str)

def __init__(self, com_port='COM3'):
super().__init__()
self.running = True
self.model = YOLO('models/yolov8_custom.pt')
self.tracker = DeepSort(max_age=30, n_init=3)
self.ser = serial.Serial(com_port, 115200, timeout=0.01)

def run(self):
cap = cv2.VideoCapture(1) # Захоплення з MS2130

while self.running:
ret, frame = cap.read()
if not ret:
continue

frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

```
results = self.model.predict(source=frame_rgb, verbose=False)

detections = []
for box in results[0].boxes:
if box.conf[0] > 0.45: # Фільтрація за впевненістю
x1, y1, x2, y2 = box.xyxy[0].tolist()
detections.append(([x1, y1, x2-x1, y2-y1], box.conf[0].item(), 'target'))

# Оновлення трекера DeepSORT
tracks = self.tracker.update_tracks(detections, frame=frame_rgb)

for track in tracks:
if not track.is_confirmed():
continue

ltrb = track.to_ltrb()
cx = (ltrb[0] + ltrb[2]) / 2

# Обчислення похибки наведення
error_x = cx - (frame.shape[1] / 2)

# Відправка на ESP32
command = f"X:{error_x:.2f}\n"
self.ser.write(command.encode('utf-8'))
self.telemetry_ready.emit(f"Track ID: {track.track_id} | ErrX: {error_x:.2f}")

# Відмальовування
cv2.rectangle(frame, (int(ltrb[0]), int(ltrb[1])), (int(ltrb[2]), int(ltrb[3])), (0, 255,
0), 2)
```

```
self.frame_ready.emit(frame)

cap.release()
self.ser.close()

def stop(self):
self.running = False
self.wait()
```

ДОДАТОК Г

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Салій Антон Андрійович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Система розпізнавання та відстеження рухомих об'єктів на базі згорткових нейронних мереж » є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [+] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _Gemini_____інші (вказати назву)) були використані для:
        - [ ] перекладу іншомовних джерел;
        - [+] стилістичного редагування та перевірки граматики;
        - [ ] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___» _____ 2026 р. _____ Антон САЛІЙ
(підпис)