Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ АВТОМАТИЧНОГО НАВЕДЕННЯ КАМЕРИ НА ОБ'ЄКТИ З ВИКОРИСТАННЯМ КОМП'ЮТЕРНОГО ЗОРУ

DESIGN AND IMPLEMENTATION OF A SYSTEM FOR AUTOMATICALLY POINTING A CAMERA AT OBJECTS USING COMPUTER VISION

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-44
Довгополюк Дмитро Сергійович

Керівник:
Ліщина Наталія Миколаївна

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна
_____

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«___» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Довгополюку Дмитру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «*Проєктування та реалізація системи автоматичного наведення камери на об'єкти з використанням комп'ютерного зору»*
Керівник роботи: к.т.н., доцент Ліщина Н. М.
затверджені наказом закладу вищої освіти від *«31» грудня 2025 р.* № *541/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра
«29» травня 2026 р.
3. Вихідні дані до роботи: *предметна область автоматичного виявлення та супроводу об'єктів у відеопотоці, технології розробки програмного забезпечення (Python, FastAPI), засоби комп'ютерного зору та ШІ детекції (YOLO, OpenCV, Roboflow Supervision), апаратні платформи та AI-прискорювачі (Raspberry Pi 5, Hailo-8L), протоколи передачі даних у реальному часі (WebRTC, WebSocket).*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз сучасного стану проблеми, постановка завдання, формування вимог до розроблюваної системи, обґрунтування вибору апаратної платформи та засобів розробки, підготовка датасету та навчання моделі детекції, практична реалізація програмно-апаратного комплексу, експорт та квантизація моделі для Hailo-8L, обґрунтування принципів захисту системи, тестування та налагодження програмного забезпечення.*
5. Перелік графічного матеріалу: *9 рисунків, 17 лістингів.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Ліщина Н. М.*		
Специфікація вимог до розробленої системи	*Ліщина Н. М*		
Розробка об’єкта проектування	*Ліщина Н. М*		
Нормоконтроль	*Суринович О. М.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Ліщина Н. М.*		

7. Дата видачі завдання *«3» лютого 2026 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 14.02.2026 р.	
2	Аналіз проблеми розробки та впровадження об’єкту проектування	до 03.03.2026 р.	
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 14.03.2026 р.	
4	Розробка функціонально-структурної схеми роботи об’єкта проектування та проектування бази даних	до 14.04.2026 р.	
5	Практична реалізація об’єкта проектування та розробка бази даних	до 20.04.2026 р.	
6	Тестування та налагодження об’єкта проектування	до 04.05.2026 р.	
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 29.05.2026 р.	

Здобувач вищої освіти _____ Дмитро ДОВГОПОЛЮК

Керівник кваліфікаційної роботи _____ Наталія ЛІЩИНА

АНОТАЦІЯ

Довгополюк Д. С. Проєктування та реалізація системи автоматичного наведення камери на об’єкти з використанням комп’ютерного зору. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі проведено дослідження предметної області, пов’язаної з автоматичним виявленням та супроводом об’єктів у відеопотоці, проаналізовано існуючі програмно-апаратні рішення та виявлено їхні сильні і слабкі сторони, а також обґрунтовано необхідність створення власної системи. У другому розділі сформовано вимоги до програмного забезпечення, здійснено проєктування системи із використанням UML-нотації та обґрунтовано вибір апаратної платформи, алгоритмів комп’ютерного зору і засобів розробки. У третьому розділі описано процес реалізації програмно-апаратного комплексу, підготовку власного датасету та навчання моделі детекції, експорт і квантизацію моделі для апаратного прискорювача, а також наведено результати тестування і налагодження системи. У висновках підсумовано отримані результати, підтверджено досягнення мети роботи та повне виконання поставлених завдань.

Ключові слова: комп’ютерний зір, автоматичне наведення камери, детекція об’єктів, трекінг, YOLO, ByteTrack, Hailo-8L, Raspberry Pi 5, WebRTC, FastAPI, Python.

ABSTRACT

Dovhopoliuk D. S. Design and Implementation of a System for Automatically Pointing a Camera at Objects Using Computer Vision. Manuscript. Qualification work for bachelor's degree in Software Engineering, specialty «Software Engineering». Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification work consists of an introduction, three chapters, conclusions, and a list of references.

In the first chapter, a study of the problem domain related to automatic detection and tracking of objects in a video stream was conducted. Existing hardware-software solutions were analyzed, their strengths and weaknesses were identified, and the necessity of developing a new system was substantiated. In the second chapter, the requirements for the software were defined, the system was designed using UML notation, and the choice of hardware platform, computer vision algorithms, and development tools was justified. The third chapter describes the implementation of the hardware-software complex, the preparation of a custom image dataset and training of the detection model, the export and quantization of the model for the hardware accelerator, as well as the results of system testing and debugging. The conclusions summarize the results obtained, confirm the achievement of the research objective, and verify the complete fulfillment of all defined tasks.

Keywords: computer vision, automatic camera pointing, object detection, tracking, YOLO, ByteTrack, Hailo-8L, Raspberry Pi 5, WebRTC, FastAPI, Python.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний розвиток інформаційних технологій, комп’ютерного зору та вбудованих обчислювальних систем сприяє активному впровадженню автоматизованих рішень у задачі спостереження, аналізу відеопотоку та керування технічними пристроями. Особливої актуальності набувають системи, здатні працювати в режимі реального часу, самостійно виявляти об’єкти на зображенні, супроводжувати їх між кадрами та формувати керуючі команди для зміни положення камери. Такі системи можуть використовуватися для моніторингу територій, робототехнічних платформ, дослідницьких стендів, систем відеоспостереження та інших прикладних задач, де необхідна швидка реакція на зміну положення об’єкта в кадрі.

Актуальність теми зумовлена необхідністю створення доступної програмно-апаратної системи, яка поєднує можливості детекції засобами комп’ютерного зору, супроводу об’єктів, локальної обробки відеопотоку та віддаленої взаємодії з оператором. Використання одноплатних комп’ютерів, апаратних AI-прискорювачів і сучасних вебтехнологій дозволяє реалізувати такі системи без залучення потужних серверів або хмарних сервісів. Це зменшує затримку обробки, підвищує автономність рішення та дає змогу застосовувати систему в умовах обмежених обчислювальних ресурсів.

Метою кваліфікаційної роботи бакалавра є розробка системи автоматичного наведення камери на об’єкти з використанням методів комп’ютерного зору, апаратного прискорення ШІ моделі та вебінтерфейсу для віддаленої взаємодії з оператором.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

– виконати аналіз сучасного стану проблеми та розглянути існуючі аналоги систем виявлення й супроводу об’єктів;

– сформулювати функціональні та нефункціональні вимоги до розроблюваної системи;

– обґрунтувати вибір апаратної платформи, програмних засобів, алгоритмів комп’ютерного зору, методів керування положенням камери та засобів передавання відеопотоку;

– сформувати власний датасет зображень, виконати його розмітку, поділ на навчальну, валідаційну й тестову вибірки та підготувати до навчання моделі;

– здійснити експорт, квантизацію та компіляцію моделі у формат HEF, придатний для виконання на апаратному прискорювачі Hailo-8L;

– реалізувати серверну частину системи для отримання відеопотоку, детекції об'єктів, трекінгу та формування керуючих команд;

– розробити вебінтерфейс оператора та організувати передавання відеопотоку з використанням WebRTC і обмін командами через WebSocket;

– провести тестування та налагодження розробленої інформаційно-комп'ютерної системи.

Об'єктом дослідження є процес автоматичного виявлення та супроводу об'єктів у відеопотоці з одночасним формуванням керуючих команд для зміни положення камери в режимі реального часу.

У роботі використано методи комп’ютерного зору, нейромережевої детекції, трекінгу об’єктів, програмної інженерії, моделювання UML, а також методи тестування та налагодження програмно-апаратних систем. Для реалізації системи використано мову програмування Python, FastAPI, Picamera2, OpenCV, Roboflow Supervision, модель сімейства YOLO, апаратний прискорювач Hailo-8L, WebSocket та WebRTC.

Практичне значення роботи полягає у створенні працездатного прототипу системи, яка забезпечує локальну обробку відеопотоку, виявлення та супровід об’єктів, формування керуючих команд і відображення результатів у вебінтерфейсі оператора. Розроблене рішення може бути використане як основа для подальшого вдосконалення систем відеоспостереження, роботизованих платформ або навчально-дослідних комплексів із комп’ютерного зору.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ ВИЯВЛЕННЯ ТА СУПРОВОДУ ОБ'ЄКТІВ У ВІДЕОПОТОЦІ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Системи автоматичного наведення камери на об'єкти з використанням комп'ютерного зору належать до програмно-апаратних рішень, у яких відеопотік аналізується в режимі реального часу, а результати цього аналізу використовуються для зміни положення камери. На відміну від звичайних систем відеоспостереження, де оператор самостійно стежить за зображенням і вручну керує напрямом огляду, система автоматичного наведення повинна самостійно виявляти об'єкт, визначати його координати в кадрі, супроводжувати його між послідовними кадрами та формувати керуючі команди для виконавчого механізму.

Актуальність таких систем зумовлена тим, що в багатьох практичних задачах пасивної передачі зображення вже недостатньо. Оператор може одночасно контролювати лише обмежену кількість відеопотоків, а за умов швидкого руху об'єктів, поганої видимості, великої відстані або тривалої роботи зростає ймовірність втрати цілі. Тому сучасні системи спостереження поступово переходять від простої трансляції відео до інтелектуальної обробки зображення, де камера не лише передає кадр, а й бере участь у процесі виявлення, супроводу та утримання об'єкта в полі зору.

Особливої актуальності ця задача набуває в сучасних умовах України, де зросла роль безпілотних літальних апаратів, мобільних засобів спостереження, наземних роботизованих платформ і систем локальної відеоаналітики. У таких умовах камера часто є частиною складнішого програмно-апаратного комплексу, який повинен працювати швидко, автономно та з мінімальною затримкою. Для таких систем важливими є не лише якість відеозображення, а й можливість

автоматичного виявлення об’єктів, супроводу вибраної цілі та формування команд для зміни положення камери.

Практична потреба в таких рішеннях безпосередньо пов’язана з необхідністю швидкого реагування на рухомі об’єкти. Якщо оператор самостійно виконує весь цикл роботи – спостереження, пошук об’єкта, вибір цілі, керування камерою та контроль її положення, – система значною мірою залежить від його уважності й швидкості реакції. У разі автоматизації частини цього процесу камера може самостійно утримувати об’єкт у полі зору, а оператор отримує змогу зосередитися на оцінюванні ситуації та прийнятті рішень. Саме тому задача автоматичного наведення камери має прикладне значення для охоронних систем, мобільних платформ, робототехнічних комплексів і систем спостереження за повітряними або наземними об’єктами.

З технічного погляду задача автоматичного наведення камери складається з кількох взаємопов’язаних етапів. Спочатку система повинна отримати відеопотік із камери та виконати виявлення об’єктів у кадрі. Далі необхідно визначити, який саме об’єкт потрібно супроводжувати, зберегти його ідентичність між послідовними кадрами, обчислити відхилення його положення від центра зображення та сформувати команди для виконавчого механізму. Якщо хоча б один із цих етапів працює нестабільно, система може втрачати ціль, перемикатися між різними об’єктами або формувати різкі й неточні рухи камери.

Першою важливою складовою є виявлення об’єкта у відеопотоці. Для автоматичного наведення недостатньо просто отримувати зображення з камери: система повинна визначати координати об’єкта, його клас і рівень упевненості розпізнавання. У сучасних дослідженнях для цього широко застосовуються ШІ моделі сімейства YOLO, оскільки вони забезпечують прийнятний баланс між точністю та швидкодією. Виявлення літальних об’єктів у режимі реального часу ускладнюється великою варіативністю просторових розмірів і пропорцій об’єктів, високою швидкістю руху, оклюзіями та складним фоном. У дослідженні також наведено результати, за яких модель досягає середньої швидкості близько 50 кадрів за секунду на відео роздільної здатності 1080p при

mAP50 на рівні 79,2 %, що підтверджує придатність подібних моделей для задач реального часу [1].

Для системи автоматичного наведення результати таких досліджень мають безпосереднє значення. Якщо модель детекції працює повільно, камера реагуватиме із затримкою. Якщо модель часто пропускає об’єкт або нестабільно визначає його координати, контур наведення не зможе надійно утримувати ціль у центрі кадру. Тому вибір моделі виявлення об’єктів є базовою умовою побудови всієї системи, а не окремим допоміжним елементом.

Другою складовою є супровід об’єкта між кадрами. У реальних умовах у полі зору камери може одночасно перебувати кілька об’єктів, частина з них може перекривати один одного, а впевненість детекції може змінюватися від кадру до кадру. Якщо система орієнтуватиметься лише на окремі результати детекції, вона може хаотично перемикатися між різними об’єктами або втрачати вибрану ціль. Саме тому після етапу виявлення необхідний етап трекінгу, який пов’язує результати детекції у часі та надає об’єктам сталі ідентифікатори. Для вирішення цієї проблеми було запропоновано алгоритм ByteTrack. Його особливість полягає в тому, що для асоціації використовуються не лише рамки з високою впевненістю, а майже всі виявлені рамки. Для рамок із низькою оцінкою впевненості автори використовують їхню схожість із наявними треклетами, що дає змогу відновлювати справжні об’єкти та відсіювати фонові детекції. Завдяки цьому система краще зберігає супровід об’єкта навіть у ситуаціях, коли ціль частково перекривається іншими об’єктами, на короткий час погіршується якість її розпізнавання або вона змінює положення в кадрі [2].

Для системи автоматичного наведення камери це має практичне значення, оскільки камера повинна стежити за конкретною вибраною ціллю, а не щоразу реагувати на новий результат детекції. Завдяки цьому зменшується ймовірність хаотичного перемикання між об’єктами та підвищується стабільність супроводу.

Третьою складовою є перетворення результатів комп’ютерного зору на команди керування камерою. Після того як система виявила об’єкт і визначила, яку саме ціль потрібно супроводжувати, необхідно обчислити її зміщення

відносно центра кадру. Це зміщення використовується як основа для формування команд повороту й нахилу камери. Такий підхід відповідає загальній логіці visual servoing, коли зображення з камери використовується як джерело зворотного зв’язку для керування виконавчим механізмом.

Було розглянуто систему візуального керування для супроводу рухомих цілей безпілотним літальним апаратом, у якій ознаки цілі на зображенні використовуються для наведення камери на об’єкт із заздалегідь невідомим характером руху, а детектор YOLO застосовується для отримання обмежувальної рамки та орієнтації цілі. Для розроблюваної системи такий принцип також є ключовим: координати центра рамки детекції порівнюються з центром кадру, після чого формується керуючий вплив для виконавчого механізму [3].

Однак на практиці керування камерою не може зводитися лише до прямого передавання координат об’єкта на приводи. Координати детекції можуть коливатися між кадрами, модель може давати незначні похибки, а сервоприводи мають власну інерційність і обмеження швидкості. Якщо не враховувати ці фактори, рух камери може стати різким, нестабільним або надмірно чутливим до дрібних змін у відеопотоці. Тому в системах автоматичного наведення доцільно використовувати згладжування сигналу, мертві зони, затримки утримання цілі та логіку повторного пошуку після втрати об’єкта. У розроблюваній системі ця ідея реалізується через згладжування команд керування, мертву зону та стани супроводу, які дають змогу уникати хаотичного перемикання між цілями.

Четвертою складовою є локальне виконання обробки відео. Для системи автоматичного наведення затримка між появою об’єкта в кадрі, його виявленням, супроводом і зміною положення камери має критичне значення. Якщо обробка відео виконується на віддаленому сервері, додаткова мережева затримка може погіршити якість супроводу. Саме тому сучасні технічні рішення дедалі частіше орієнтуються на edge-AI, тобто виконання нейромережевого інференсу безпосередньо на пристрої або поруч із камерою.

На ринку вже існують готові рішення, які частково реалізують подібний принцип. Найближчими аналогами є PTZ-камери з функцією автоматичного супроводу. Стандарт ONVIF PTZ Service Specification [4] визначає вебсервісний інтерфейс для конфігурації та роботи pan-tilt-zoom контролерів, а також пов’язаних із ними подій. Це дає змогу інтегрувати PTZ-камери в мережеві системи відеоспостереження, однак сам стандарт не визначає алгоритми виявлення об’єктів, вибору цілі або супроводу.

Прикладом готового промислового рішення є AXIS Perimeter Defender PTZ Autotracking [5]. Це програмне забезпечення дає змогу PTZ-камері автоматично наближати зображення та супроводжувати об’єкти тривоги, виявлені фіксованою камерою з відеоаналітикою AXIS Perimeter Defender. Перевагою такого рішення є готовність до використання, інтеграція з обладнанням виробника та орієнтація на реальні задачі охоронного відеоспостереження. Водночас воно є частиною закритої екосистеми, тому користувач обмежений підтримуваними камерами, ліцензійними умовами та внутрішньою логікою виробника.

Схожий підхід реалізовано в технології Dahua Auto Tracking 3.0 [6]. Вона призначена для PTZ-камер і передбачає автоматичне супроводження рухомої цілі після спрацювання правила інтелектуальної відеоаналітики (IVS). Камера використовує горизонтальне й вертикальне обертання, а також масштабування для утримання цілі в центрі екрана. Крім того, виробник зазначає використання AI-алгоритму для прогнозування напряму та швидкості руху об’єкта, а також роботу адаптивних алгоритмів швидкості й фокусування. Таке рішення добре демонструє практичну затребуваність автоматичного наведення камери, однак також має закриту архітектуру та не передбачає повної зміни алгоритмів детекції, супроводу чи керування.

Окремо доцільно згадати спеціалізовані промислові системи, у яких принцип автоматичного виявлення та супроводу об’єктів поєднується з керуванням виконавчим механізмом. Прикладом такого рішення є українська система Sky Sentinel [7]. Відкрита кампанія UNITED24 описує її як AI-керовану

систему протиповітряної оборони та повідомляє, що в межах початкового збору було акумульовано 1 500 000 доларів на перші 10 таких систем. У відкритих матеріалах також зазначається, що Sky Sentinel використовує штучний інтелект для виявлення, супроводу й наведення на повітряні цілі, зокрема ударні безпілотники типу Shahed. Перевагами такого підходу є висока автономність, інтеграція засобів виявлення, супроводу й керування, а також практична орієнтація на роботу в складних умовах. Водночас Sky Sentinel є спеціалізованим закритим промисловим рішенням, яке не призначене для вільної модифікації алгоритмів, навчання власних моделей або використання як навчально-дослідницька платформа. Тому в межах цієї роботи така система розглядається не як безпосередній аналог, а як приклад того, що задачі автоматичного виявлення, супроводу та наведення є актуальними й практично затребуваними в сучасних умовах України.

Таким чином, аналіз готових PTZ-рішень і спеціалізованих промислових систем показує, що функції автоматичного супроводу та наведення вже активно використовуються на практиці. Проте більшість таких рішень має закриту архітектуру, залежить від конкретного виробника та не дає змоги повністю контролювати всі етапи роботи системи – від отримання кадру й детекції об’єкта до формування керуючої команди. Саме це обґрунтовує доцільність розроблення власної системи автоматичного наведення камери у якій можна реалізувати власну логіку вибору цілі, супроводу, згладжування керуючих сигналів і віддаленої взаємодії з оператором.

Проведений аналіз показує, що сучасний стан проблеми складається з кількох взаємопов’язаних напрямів. Наукові дослідження підтверджують ефективність детекції та трекінгу об’єктів з використанням ШІ і керування pan-tilt платформами, але здебільшого розглядають ці складові окремо. Готові PTZ-камери забезпечують базове або розширене автостеження, однак мають закриту архітектуру та обмежені можливості адаптації. Спеціалізовані промислові системи демонструють практичну затребуваність автоматичного наведення, однак є дорогими та закритими. Навчальні та прототипні рішення є доступними,

проте часто не забезпечують достатньої стабільності, швидкодії та функціональної завершеності.

Отже, на сьогодні є доцільним розроблення власної системи автоматичного наведення камери на об’єкти з використанням комп’ютерного зору. Така система повинна поєднувати локальне виявлення об’єктів у відеопотоці, супровід вибраної цілі, формування команд для зміни положення камери, згладжування керуючих сигналів і віддалену взаємодію з оператором. Запропонований підхід дає змогу створити гнучке програмно-апаратне рішення, яке поєднує переваги сучасних методів комп’ютерного зору, edge-AI обробки та відкритішої архітектури, придатної для подальшого вдосконалення.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

На підставі проведеного у попередньому підрозділі аналізу сучасного стану проблеми та огляду існуючих аналогів у межах даної кваліфікаційної роботи бакалавра ставиться задача розробити систему автоматичного наведення камери на об’єкти з використанням комп’ютерного зору. Розроблювана система повинна виконувати виявлення й супровід об’єктів різного типу (зокрема людей, транспортних засобів і безпілотних літальних апаратів) у режимі реального часу, керувати власною електромеханічною платформою та забезпечувати віддалений доступ для оператора.

Для досягнення поставленої мети необхідно розв’язати такі конкретні завдання:

– виконати аналіз сучасного стану проблеми, огляд існуючих аналогів та обґрунтувати доцільність розробки власної системи;

– сформулювати функціональні та нефункціональні вимоги до системи, скласти специфікацію вимог і побудувати моделі, що відображають її структуру та поведінку;

– обґрунтувати вибір апаратної платформи, програмних засобів, алгоритмів комп’ютерного зору та методів керування положенням камери;

– спроєктувати архітектуру програмно-апаратного комплексу, визначити основні модулі системи та порядок їхньої взаємодії;

– підготувати власний набір даних для навчання моделі виявлення об’єктів, виконати його маркування, навчання моделі та оцінювання її точності;

– реалізувати програмні модулі отримання відеопотоку, детекції об’єктів, супроводу цілі та формування команд керування;

– реалізувати режими роботи системи та вебінтерфейс оператора для тестування, демонстрації й віддаленої взаємодії з системою;

– провести тестування розробленої системи, оцінити якість виявлення й супроводу об’єктів, стабільність наведення та навантаження на апаратну платформу.

У результаті виконання перелічених завдань повинна бути отримана працездатна система, придатна до демонстрації та подальшого доопрацювання. Технічні характеристики, структурні рішення й алгоритмічне забезпечення системи, а також результати тестування детально розглянуто у наступних розділах кваліфікаційної роботи бакалавра.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

На початковому етапі розробки програмного забезпечення системи автоматичного наведення камери було проведено аналіз предметної області, пов’язаної з обробкою відеопотоку, комп’ютерним зором, супроводом об’єктів та дистанційним керуванням положенням камери. Аналіз показав, що розроблювана система повинна забезпечувати отримання зображення з камери, виявлення об’єктів у кадрі, супровід вибраної цілі та формування команд для зміни положення камери відповідно до положення об’єкта у відеопотоці.

Основна мета програмного забезпечення – створити систему, здатну працювати в режимі реального часу, виконувати локальну обробку відеоданих і надавати оператору зручний інтерфейс для контролю та керування. Система повинна функціонувати на одноплатному комп’ютері Raspberry Pi 5 з використанням апаратного прискорення для задач комп’ютерного зору. Такий підхід зменшує залежність від зовнішніх серверів, скорочує затримку передавання даних і робить роботу системи автономнішою [8].

У процесі аналізу було визначено, що програмне забезпечення складається з кількох взаємопов’язаних функціональних частин:

– модуль отримання відеопотоку;

– модуль виявлення об’єктів;

– модуль супроводу цілі;

– модуль вибору активного об’єкта;

– модуль формування керуючих команд;

– модуль взаємодії з платою керування;

– модуль трансляції відеопотоку у вебінтерфейс;

– модуль авторизації користувача.

Кожен із цих модулів виконує окрему задачу, однак разом вони формують повний цикл роботи системи – від отримання кадру до передавання команди на виконавчі механізми та відображення результату оператору.

До основних функціональних вимог розроблюваного програмного забезпечення належать:

– отримання відеопотоку з камери у заданій роздільній здатності;

– обробка кадрів у режимі реального часу;

– виявлення об’єктів у кадрі за допомогою ШІ моделі;

– супровід виявлених об’єктів між послідовними кадрами;

– присвоєння об’єктам сталих ідентифікаторів;

– вибір активної цілі для подальшого супроводу;

– підтримка ручного, напівавтоматичного та автоматичного режимів роботи;

– формування команд повороту та нахилу камери;

– передавання керуючих команд на плату керування;

– трансляція відеопотоку до вебінтерфейсу оператора;

– відображення результатів обробки відео у вигляді рамок, ідентифікаторів і позначки активної цілі;

– забезпечення авторизованого доступу до системи;

– відображення службового стану системи та журналу подій.

Поряд із функціональними було сформульовано і нефункціональні вимоги. Система повинна забезпечувати низьку затримку між отриманням кадру та формуванням керуючої команди, стабільну роботу під час тривалого використання, коректне відображення інтерфейсу в сучасних браузерах, захищеність доступу, модульність програмної архітектури та можливість подальшого розширення. Окремо важливою є зручність інтерфейсу: оператор має швидко оцінювати поточний стан системи, бачити відеопотік, обирати режим роботи та за потреби втручатися в процес наведення.

Для формалізації функціональних можливостей системи було побудовано діаграму прецедентів, наведену на рисунку 2.1. Вона відображає основні сценарії взаємодії зовнішніх користувачів із програмним забезпеченням.

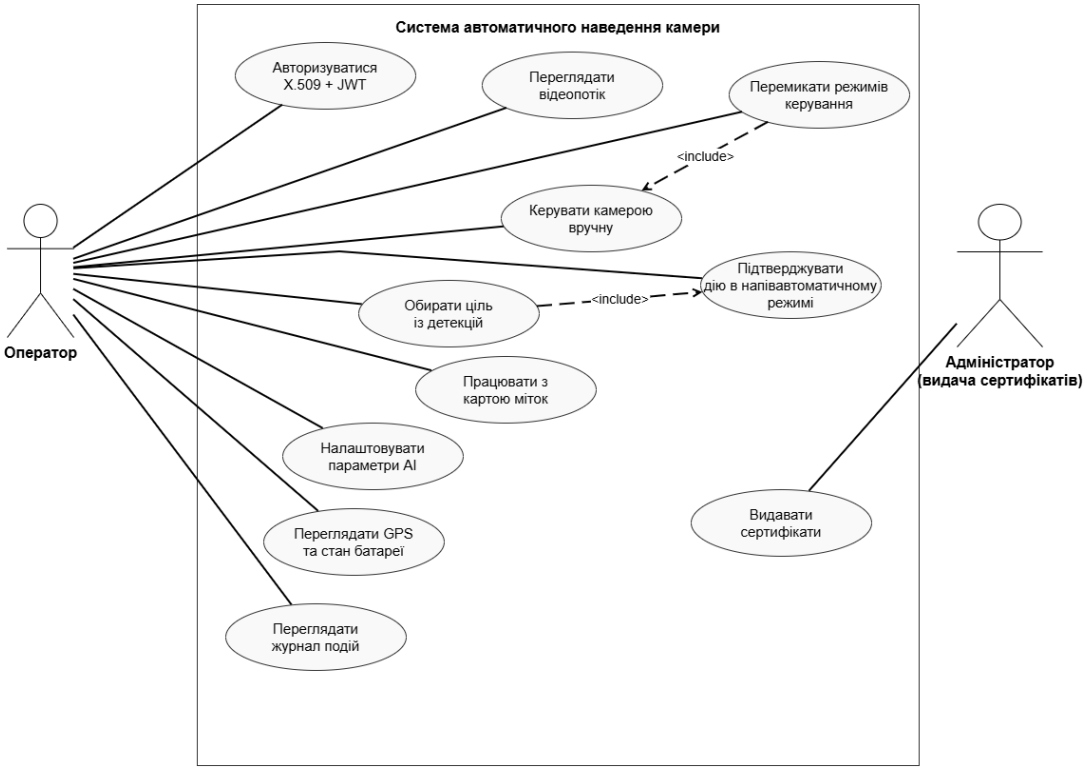

Рисунок 2.1 – Діаграма прецедентів системи

Діаграма демонструє, які функції доступні оператору під час роботи із системою. Основним актором є оператор, який проходить авторизацію, переглядає відеопотік, перемикає режими роботи, обирає ціль, здійснює ручне керування положенням камери, переглядає телеметричні дані та журнал подій. Окремо виділено адміністратора, який відповідає за керування доступом і видачу сертифікатів. Такий поділ дозволяє розмежувати функції безпосереднього керування системою та функції адміністрування доступу.

Особливістю системи є наявність кількох режимів роботи. У ручному режимі оператор самостійно керує положенням камери. У напівавтоматичному режимі система може пропонувати або супроводжувати ціль, однак остаточне підтвердження дії залишається за оператором. В автоматичному режимі система самостійно визначає активну ціль і формує керуючі команди на основі її

положення у кадрі. Такий підхід забезпечує гнучкість використання системи в різних умовах.

Окрему увагу під час проєктування приділено інформаційним потокам між компонентами системи. Основними потоками є відеопотік від камери до вебінтерфейсу, дані детекції та супроводу об’єктів, команди керування положенням камери, а також службові дані про стан системи. Відеопотік використовується для візуального контролю, тоді як команди керування та службові повідомлення забезпечують інтерактивну взаємодію оператора із системою.

Під час проєктування інтерфейсу користувача враховувалося, що оператор повинен одночасно бачити відеопотік, стан системи, список виявлених цілей та елементи керування. Тому інтерфейс організовано як єдиний робочий екран без необхідності переходу між великою кількістю сторінок. Схематичне розташування основних блоків інтерфейсу показано на рисунку 2.2.

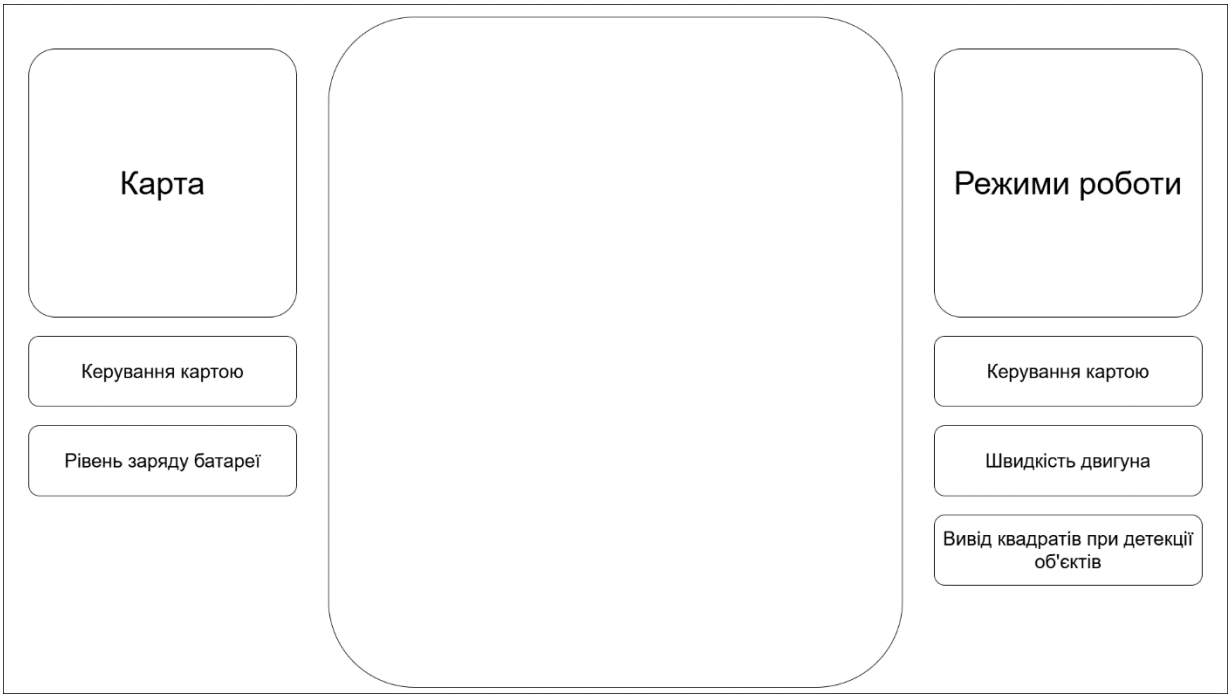

Рисунок 2.2 – Схематичне зображення вебінтерфейсу оператора

Центральну частину екрана займає область відеопотоку, у якій виводиться зображення з камери з накладеними результатами обробки: рамками виявлених

об’єктів, ідентифікаторами та позначкою активної цілі. У верхній частині інтерфейсу розміщуються індикатори стану системи – поточний режим роботи, частота кадрів, стан мережевих підключень і службові повідомлення. Панель керування містить елементи вибору режиму роботи, список доступних цілей, засоби ручного керування поворотом і нахилом камери, а також кнопку підтвердження дії для напівавтоматичного режиму. Нижня частина інтерфейсу використовується для відображення журналу подій, у якому фіксуються зміни режимів, вибір цілі, втрата супроводу, помилки з’єднання та інші важливі події. Така структура дозволяє оператору швидко отримувати інформацію про стан системи та оперативно виконувати необхідні дії.

Робота з вебінтерфейсом передбачає послідовну взаємодію оператора з основними елементами системи. Після запуску оператор спостерігає за відеопотоком, оцінює наявні цілі в кадрі та за потреби обирає одну з них для супроводу. Залежно від обраного режиму система або лише відображає результати обробки, або додатково формує керуючі команди для зміни положення камери. У разі виникнення помилок або зміни стану системи відповідна інформація відображається через службові повідомлення та журнал подій.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Після формування вимог до системи постало завдання обрати конкретні засоби розробки, методи обробки даних та алгоритми, на основі яких ці вимоги можна реалізувати. Система автоматичного наведення камери поєднує одразу кілька задач – обробку відеопотоку, детекцію та класифікацію об’єктів, супровід цілі, передавання відео у браузер користувача та керування апаратною частиною. Тому під час вибору інструментів враховувалися такі фактори як швидкодія, затримка обробки, сумісність з Raspberry Pi 5, можливість виконувати обчислення локально, а також зручність подальшого розширення коду.

Для серверної частини системи було обрано мову Python. Вибір зумовлений тим, що Python має широкий набір бібліотек для роботи з комп’ютерним зором, ШІ моделями, асинхронними вебсервісами та апаратними інтерфейсами. Він повноцінно підтримується на Raspberry Pi OS, що спрощує розгортання на цільовій платформі. Як альтернативи розглядалися C++ і JavaScript/Node.js, однак C++ значно ускладнює розробку, а Node.js менш зручний у поєднанні з AI-бібліотеками та засобами обробки зображень. На їхньому фоні Python виглядає оптимальним компромісом між продуктивністю, швидкістю розробки та доступністю необхідних інструментів.

Серверне API та взаємодію з вебінтерфейсом побудовано на фреймворку FastAPI. Він підтримує асинхронну обробку запитів і WebSocket з’єднання, що добре підходить для систем реального часу. FastAPI дозволяє одночасно обслуговувати HTTP-запити, канали керування та службові повідомлення в межах одного процесу. Запуск серверу здійснюється через ASGI-сервер uvicorn, який забезпечує ефективну роботу асинхронного застосунку. Порівняно з Flask, FastAPI краще пасує до цієї задачі завдяки вбудованій асинхронності та зручнішій організації API [9].

Кадри з камери отримуються за допомогою бібліотеки Picamera2 – основного інструменту роботи з камерами Raspberry Pi. Вона передає кадри безпосередньо в конвеєр обробки, без зайвих проміжних перетворень, що критично для системи реального часу: будь-яке додаткове копіювання чи перекодування збільшує загальну затримку. Picamera2 також дає змогу гнучко налаштовувати роздільну здатність, частоту кадрів і формат зображення [10].

Для виявлення об’єктів використовується модель сімейства YOLO. Вона добре підходить для роботи в реальному часі, оскільки виконує детекцію за один прохід та одразу повертає координати обмежувальних рамок, клас об’єкта і рівень достовірності. Це дозволяє швидко визначити положення об’єктів у кадрі й передати ці дані далі – до модуля супроводу. Порівняно з більш важкими підходами до детекції, YOLO забезпечує кращий баланс між швидкодією та якістю розпізнавання [11].

Запуск YOLO безпосередньо на центральному процесорі Raspberry Pi 5 створював би значне навантаження і помітно збільшував затримку. Тому для прискорення обчислень застосовується апаратний AI-прискорювач Hailo-8L. Модель попередньо компілюється у формат HEF і виконується саме на прискорювачі. Завдяки цьому центральний процесор залишається вільним для роботи вебсервера, обробки команд, авторизації та взаємодії з платою керування. Hailo-8L орієнтований саме на ШІ задачі і працює локально, без потреби у підключенні до зовнішніх серверів, що додатково підвищує автономність системи [12].

Для супроводу об’єктів між кадрами обрано алгоритм ByteTrack у реалізації бібліотеки Roboflow supervision. Його ключова перевага полягає в тому, що він підтримує сталі ідентифікатори об’єктів між послідовними кадрами. Для системи наведення це принципово: важливо супроводжувати не просто будь-який об’єкт у кадрі, а саме ту ціль, яку було обрано як активну. Інтеграція ByteTrack через Roboflow supervision спрощує зв’язок трекера з результатами YOLO-детекції та зменшує обсяг власного коду, потрібного для підтримки супроводу [13].

Вибір активної цілі реалізовано через метод пріоритетного оцінювання. Кожен виявлений і супроводжуваний об’єкт оцінюється за кількома ознаками: відстанню до центра кадру, достовірністю детекції, розміром обмежувальної рамки та класом об’єкта. Це дає змогу обрати найбільш релевантну ціль серед кількох кандидатів. Метод простий у реалізації, його логіка прозора, а вагові коефіцієнти можна змінювати залежно від умов експлуатації системи.

Для стабільності супроводу побудовано кінцевий автомат станів, який описує переходи між трьома станами: SEARCHING, LOCKED і HOLDING. У стані SEARCHING система шукає придатну ціль. У стані LOCKED – супроводжує вже обраний об’єкт. У стані HOLDING – тимчасово утримує останнє відоме положення цілі на випадок короткочасної втрати детекції. Така модель захищає систему від випадкових перемикань між об’єктами та підвищує

її стійкість до шуму в результатах детекції. Логіку переходів між станами наведено на рисунку 2.3.

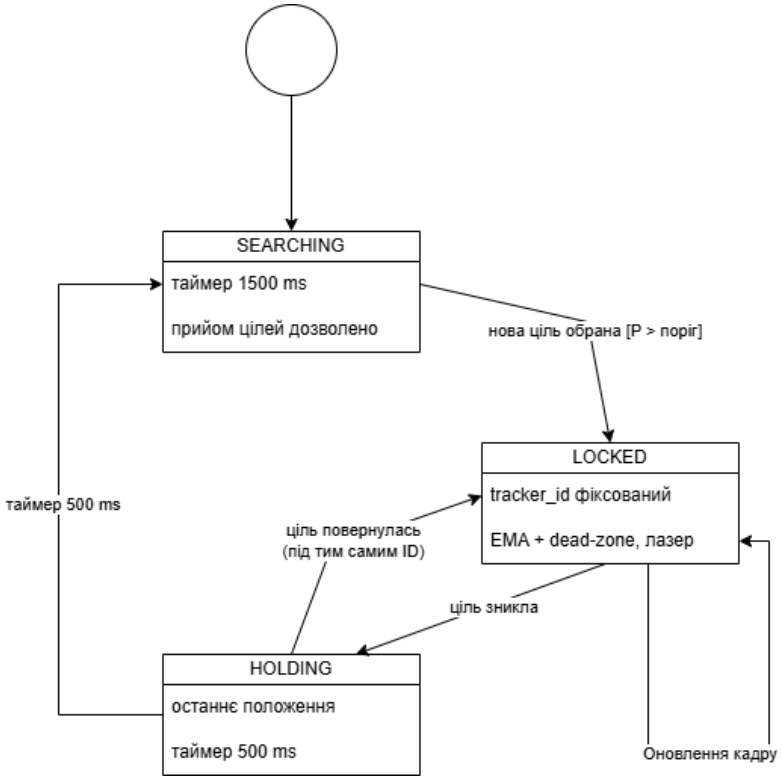

Рисунок 2.3 – Діаграма станів супроводу активної цілі

Перехід зі стану SEARCHING до LOCKED відбувається, коли вибрана ціль відповідає заданим умовам пріоритету. Якщо активна ціль тимчасово зникає з кадру, система переходить у HOLDING і продовжує орієнтуватися на її останнє відоме положення. У разі повторного виявлення цілі система повертається до LOCKED; якщо ж ціль так і не з’явилася – переходить назад у SEARCHING. Такий підхід забезпечує плавніше наведення і запобігає різким перемиканням між цілями.

Для зменшення коливань керуючого сигналу додатково застосовуються згладжування координат цілі та мертва зона. Згладжування усуває різкі стрибки координат, які можуть виникати через шум детекції або неточність обмежувальної рамки. Мертва зона запобігає формуванню нових команд, якщо зміщення цілі від центра кадру є незначним. У сукупності це зменшує кількість зайвих рухів сервоприводів і робить роботу системи плавнішою.

Відеопотік до браузера оператора передається через WebRTC. Ця технологія створювалася саме для передачі аудіо та відео у реальному часі й підтримується сучасними браузерами без додаткового програмного забезпечення. У контексті системи WebRTC дає меншу затримку порівняно з класичним HTTP запитами, що важливо для оператора – він повинен бачити сцену та результати обробки практично без відставання [14].

Для команд керування, перемикання режимів і службових повідомлень використовується WebSocket. На відміну від звичайних HTTP запитів, він підтримує постійне двостороннє з’єднання між браузером і сервером. Завдяки цьому команди оператора надходять оперативно, поточний стан системи можна отримувати без додаткових запитів, а керуючі сигнали – швидко передавати на плату Arduino. Розподіл каналів є логічним: WebRTC відповідає за відео, WebSocket – за команди та службові дані.

Безпосереднім керуванням сервоприводами займається плата Arduino. Вона приймає команди від серверної частини і формує сигнали для виконавчих механізмів. Такий розподіл дає змогу винести низькорівневу роботу з апаратурою на окремий контролер, а на Raspberry Pi залишити задачі вищого рівня – обробку відео, прийняття рішень, роботу вебсервера та взаємодію з оператором.

Доступ до системи захищено авторизацією на основі цифрового сертифіката та токена доступу. Цей підхід надійніший за звичайну авторизацію: доступ отримує лише користувач із відповідним ключем або сертифікатом. Після успішної перевірки користувачу видається токен, який далі використовується для роботи з вебінтерфейсом і мережевими каналами.

Внутрішню структуру серверної частини описано за допомогою діаграми класів, наведеної на рисунку 2.4.

Центральним є клас VideoProcessingPipeline – він поєднує отримання кадрів, детекцію об’єктів і супровід між кадрами. Клас TargetController відповідає за вибір активної цілі, обчислення її зміщення від центра кадру та оновлення стану супроводу. Клас HardwareController формує команди керування

і передає їх на плату Arduino. Клас WebInterfaceController забезпечує взаємодію з вебінтерфейсом: приймає команди оператора, передає відеопотік і публікує стан системи. Клас AuthService відповідає за перевірку доступу користувача. Такий поділ дозволяє розмежувати обробку відео, керування ціллю, апаратну взаємодію, вебкомунікацію та авторизацію між окремими класами, що підвищує модульність програмного забезпечення.

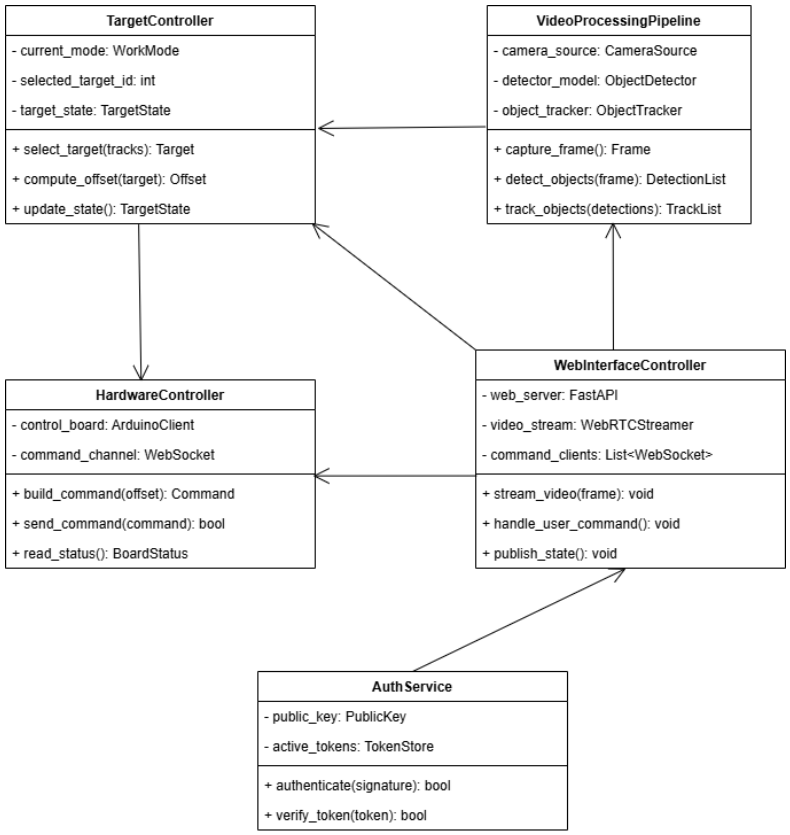

Рисунок 2.4 – Діаграма класів серверної частини системи

На стороні оператора розташовується вебінтерфейс, через який здійснюється перегляд відеопотоку та надсилання команд. На стороні Raspberry Pi 5 працюють модулі отримання відео, детекції, супроводу об’єктів, авторизації, вебсервер і модуль формування команд керування. Окремо виділено апаратні елементи – камеру, AI-прискорювач Hailo-8L, плату Arduino та сервоприводи. Камера передає кадри до серверної частини, Hailo-8L виконує прискорений інференс моделі, Arduino приймає команди і керує сервоприводами. Така

структура дозволяє чітко розділити обчислювальні задачі, AI-обробку та виконавчу частину системи.

Динаміку роботи системи відображає діаграма послідовності одного циклу автоматичного наведення, наведена на рисунку 2.5. На ній показано порядок взаємодії компонентів під час роботи системи.

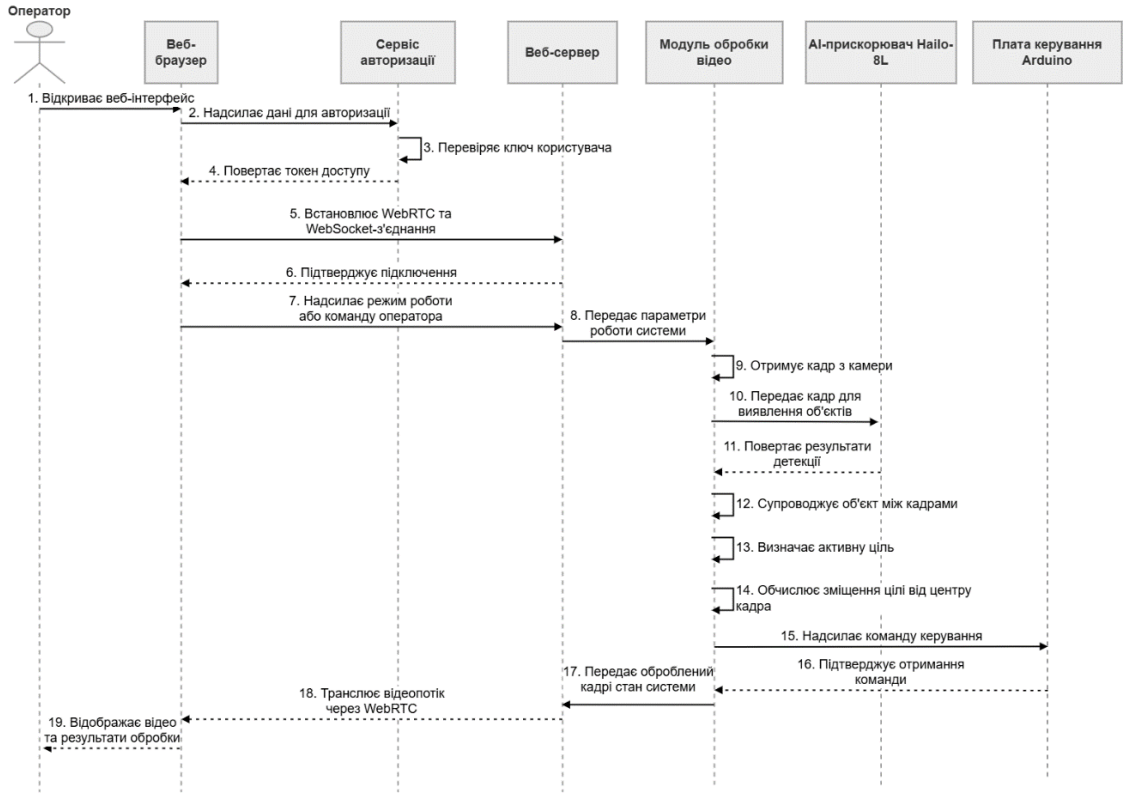

Рисунок 2.5 – Діаграма послідовності одного циклу автоматичного наведення

Спочатку оператор відкриває вебінтерфейс, після чого браузер надсилає дані для авторизації. Сервіс авторизації перевіряє користувача і повертає токен доступу. Далі встановлюються канали зв’язку – WebRTC для відеопотоку та WebSocket для команд керування. В основному циклі модуль обробки відео отримує кадр з камери і передає його на AI-прискорювач для виконання детекції. Отримавши результати, система супроводжує об’єкти між кадрами, визначає активну ціль, обчислює її зміщення відносно центра кадру та формує відповідну команду керування. Команда надсилається на плату Arduino, а оброблений кадр

і поточний стан системи повертаються до вебсервера й транслюються у браузер оператора.

Під час такого циклу обробки кожен компонент виконує окрему функцію та передає результат наступному елементу системи. Камера забезпечує надходження кадрів, AI-прискорювач виконує детекцію об’єктів, модуль супроводу зберігає зв’язок між об’єктами у послідовних кадрах, а контролер цілі визначає необхідне зміщення камери. Після цього апаратний контролер формує команду для сервоприводів, а вебінтерфейс відображає оператору оновлений відеопотік і службову інформацію.

РОЗДІЛ 3
РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ АВТОМАТИЧНОГО НАВЕДЕННЯ КАМЕРИ

3.1 Практична реалізація об’єкта проєктування

Після формування усіх функціональних та нефункціональних вимог, а також вибору засобів проєктування, було розпочато реалізацію програмного забезпечення системи. Розробка проводилась в інтегрованому середовищі PyCharm, компанії JetBrains, яке надає зручні інструменти для роботи з проєктами на Python – підсвічування синтаксису, автодоповнення коду, інтегрований відладник коду, керування віртуальними оточеннями та підтримки системи контролю версій. Дані можливості суттєво полегшують роботу над великим проєктом, що складається з декількох взаємопов’язаних модулів.

На початковому етапі було сформовано базову структуру серверного застосунку. Вона містить головний файл запуску FastAPI сервісу, модулі маршрутизації HTTP запитів, обробники WebSocket повідомлень, модуль авторизації, модуль обробки відеопотоку та окремі сервіси для взаємодії з апаратною частиною. Такий поділ дозволяє розмежувати відповідальність між частинами програми та спрощує подальше тестування й супровід коду.

У лістингу 3.1 наведено приклад ініціалізації власного вебсервісу за допомогою FastApi та uvicorn та створення декілька тестових ендпоінтів для перевірки валідності роботи коду.

Лістинг 3.1 – Ініціалізація FastAPI сервісу

```
from fastapi import FastAPI
import uvicorn

app = FastAPI()
@app.get("/health")
async def health_check():
    return {"status": "ok"}
if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

кінець лістингу 3.1

Подальша реалізація вимагала підключення джерела відеопотоку, оскільки саме отриманий з камери кадр є вхідними даними для всіх наступних етапів обробки. Для роботи з камерою використано бібліотеку Picamera2, яка забезпечує доступ до камери Raspberry Pi та дозволяє налаштовувати параметри відеопотоку: роздільну здатність, частоту кадрів, режим фокусування, експозицію, контрастність і різкість зображення. Коректне налаштування цих параметрів є важливим, оскільки якість вхідного кадру безпосередньо впливає на точність подальшої детекції.

У лістингу 3.2 наведений приклад ініціалізації камери, та налаштування конфігурації, для коректного відображення відеопотоку.

Лістинг 3.2 – Ініціалізація камери за допомогою бібліотеки Picamera2

```
from picamera2 import Picamera2

picam2 = Picamera2()
config = picam2.create_video_configuration(
    main={"size": (1280, 1280), "format": "RGB888"}
)

picam2.configure(config)
picam2.start()

frame = picam2.capture_array()
```

кінець лістингу 3.2

Після запуску камери метод capture_frame() повертає кадр у вигляді масиву, який надалі може бути переданий до модуля ШІ детекції. Отримання кадру в такому форматі є зручним для подальшої обробки, оскільки більшість бібліотек комп’ютерного зору та ШІ працюють саме з масивами зображень.

Наступним етапом було ініціалізація та конфігурація ШІ моделі, яка буде виконувати детекцію, і класифікацію об’єктів. Для цього буде використовуватись, експортована модель сімейства YOLO, яка буде виконуватись на апаратному прискорювачі Hailo-8L. Оскільки даний прискорювач використовує власний формат виконання моделей, перед запуском модель має бути скомпільована у формат HEF. У серверному застосунку

передбачено завантаження цієї моделі під час старту системи. Це дозволяє не виконувати повторну ініціалізацію моделі для кожного кадру, що зменшує затримку обробки відеопотоку.

У лістингу 3.3 наведено детальну ініціалізацію класу HailoDetector, який відповідає за завантаження моделі у форматі HEF, налаштування параметрів роботи апаратного прискорювача Hailo-8L та підготовку середовища для подальшого виконання детекції об’єктів на вхідних зображеннях.

Лістинг 3.3 – Ініціалізація та налаштування детекції та класифікації об’єктів

```
class HailoDetector:
    def __init__(self, hef_path: str):
        self.hef_path = hef_path
        self.model = None

    def load_model(self):
        self.model = self.hef_path

    def infer(self, frame):
        raw_result = self.run_inference(frame)
        return self.parse_output(raw_result)
```

кінець лістингу 3.3

Під час ініціалізації задаються шлях до моделі, параметри вхідного зображення, список класів для фільтрації результатів та допоміжні об’єкти, необхідні для виконання. Такий підхід дозволяє виконати всі підготовчі операції один раз під час запуску системи, після чого кожен отриманий кадр може одразу передаватися на обробку ШІ моделлю.

Оскільки мережа обробляє кожен кадр незалежно від минулого кадру, то виникає проблема в присвоєні кожному ідентифікованому об’єкті, унікального ідентифікатора, який допоможе оператору розрізняти цілі і вже від цього відштовхуватись в вирішенні відслідковування. Для вирішення цієї проблеми використовується бібліотека Roboflow Supervision, яка реалізує алгоритм ByteTrack, але більш оптимізовано ніж якби ці розрахунки виконувались саме через офіційну бібліотеку ByteTrack. Він отримує на вхід результати детекції поточного кадру та зіставляє їх із об’єктами, виявленими на попередніх кадрах.

У процесі такого зіставлення кадрів кожному об’єкту присвоюється унікальний ідентифікатор, який зберігається протягом усього часу його присутності на кадрі. Завдяки цьому модулю оператор може обирати конкретну ціль за її ідентифікатором, а система й надалі буде супроводжувати саме цей об’єкт, навіть якщо поруч знаходяться інші об’єкти.

У лістингу 3.4 наведена інціалізація та базова конфігурація бібліотеки Roboflow Supervision.

Лістинг 3.4 – Ініціалізація модуля для трекінгу об’єктів

```
import supervision as sv


class RoboflowTracker:
    def __init__(self):
        self.tracker = sv.ByteTrack()

    def update_tracks(self, detections: sv.Detections) ->
sv.Detections:
        tracked_detections =
self.tracker.update_with_detections(detections)
        return tracked_detections
```

кінець лістингу 3.4

У наведеному фрагменті створюється клас RoboflowTracker, який ініціалізує роботу трекера sv.ByteTrack(). Метод update_tracks() приймає містить результати детекції поточного кадру, та передає його до методу update_with_detections(). Після оновлення трекера повертаються ті самі детекції, але вже з доданими ідентифікаторами супроводу. Це дозволяє пов’язувати результати детекції між кадрами.

Після отримання результатів трекінгу необхідно підготувати кадр до відображення оператору. Для цього з результатів витягуються координати обмежувальних рамок і відповідні ідентифікатори об’єктів. Підготовка цих даних винесена в окрему функцію, що робить код візуалізації простішим і зрозумілішим (ліст. 3.5).

Лістинг 3.5 – Підготовка даних для анотації кадру

```
def annotate_frame(frame, tracked_detections):
    boxes = tracked_detections.xyxy
    tracker_ids = tracked_detections.tracker_id
    return draw_tracked_objects(frame.copy(), boxes, tracker_ids)
```

кінець лістингу 3.5

Реалізована функція відповідає за підготовку даних для візуалізації результатів трекінгу. На вхід функція отримує поточний кадр і результати роботи трекера. З об’єкта tracked_detections витягуються координати обмежувальних рамок у форматі xyxy та список унікальних ідентифікаторів tracker_id. Після цього ці дані передаються у функцію draw_tracked_objects(), яка виконує безпосереднє нанесення рамок та ідентифікаторів на кадр.

Для безпосереднього відмальовування графічних елементів на кадрі використовується бібліотека OpenCV. Вона дозволяє накладати на зображення прямокутники, текстові підписи та інші елементи візуалізації. У даній системі OpenCV використовується для обведення кожного супроводжуваного об’єкта рамкою та виведення його ідентифікатора поруч із рамкою (ліст. 3.6) [15].

Лістинг 3.6 – Підготовка аннотації кадру

```
import cv2

def draw_tracked_objects(frame, boxes, tracker_ids):
    for box, tracker_id in zip(boxes, tracker_ids):
        x1, y1, x2, y2 = map(int, box)
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(frame, f"ID {tracker_id}",
(x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
    return frame
```

кінець лістингу 3.6

Дана функція виконує відмальовування рамок і підписів на кадрі. Функція отримує зображення, координати рамок і список ідентифікаторів об’єктів. Для кожної пари координат і ідентифікатор об’єкта за допомогою cv2.rectangle() малюється прямокутна рамка, а за допомогою cv2.putText() додається текстовий

підпис із номером об’єкта. Отриманий кадр із нанесеними елементами використовується для подальшої трансляції у вебінтерфейс оператора.

Після візуалізації результатів трекінгу наступним етапом є визначення активної цілі серед супроводжуваних об’єктів. Оскільки в одному кадрі може бути виявлено декілька об’єктів, система повинна вибрати один із них для подальшого аналізу та формування керуючих команд. У межах реалізації активною ціллю вважається перший доступний об’єкт зі списку супроводжуваних детекцій. Такий підхід дозволяє перевірити роботу повного циклу системи: від отримання кадру та детекції до визначення положення об’єкта і формування команди керування (ліст. 3.7).

Лістинг 3.7 – Вибір активної цілі серед об’єктів на камері

```
def select_target(tracked_detections):
    if tracked_detections.tracker_id is None:
        return None

    if len(tracked_detections.xyxy) > 0:
        return tracked_detections.xyxy[0]

    return None
```

кінець лістингу 3.7

Дана функція відповідає за вибір активної цілі серед об’єктів, отриманих після роботи трекера. Спочатку виконується перевірка наявності ідентифікаторів супроводу. Якщо список об’єктів не порожній, функція повертає координати першого об’єкта зі списку. Якщо ж супроводжувані об’єкти відсутні, повертається пусте значення. Отримані координати активної цілі надалі використовуються для обчислення її зміщення відносно центра кадру.

Фінальним етапом є реалізація визначення активної цілі система обчислює її положення відносно центра зображення. Для цього визначається центр обмежувальної рамки об’єкта та порівнюється з геометричним центром кадру. Результатом такого обчислення є два значення: зміщення по горизонталі та

зміщення по вертикалі, які надалі використовуються під час формування керуючої команди (ліст. 3.8).

Лістинг 3.8 – Обчислення зміщення активної цілі відносно центра кадру

```
def compute_offset(target_box, frame_width: int, frame_height:
int):
    x1, y1, x2, y2 = map(int, target_box)
    target_x = (x1 + x2) // 2
    target_y = (y1 + y2) // 2

    dx = target_x - frame_width // 2
    dy = target_y - frame_height // 2

    return dx, dy
```

кінець лістингу 3.8

Дана функція відповідає за обчислення зміщення активної цілі, відносно центру кадру. На вході функція отримує координати знайденого об’єкта та розмір кадру. Спочатку визначається координата центра об’єкта, після чого вони порівнюються з координатами центра зображення. А на виході функція повертає зміщення по горизонталі та вертикалі відносно центра зображення. Саме дана функція буде використовуватись для подальшого формування керуючої команди.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Після реалізації основних модулів програмного забезпечення було виконано тестування та налагодження інформаційно-комп’ютерної системи. Метою цього етапу була перевірка працездатності окремих компонентів, коректності їхньої взаємодії між собою, а також стабільності роботи системи під час обробки відеопотоку в режимі реального часу.

Тестування проводилося поетапно. Спочатку було перевірено запуск серверної частини, реалізованої на основі FastAPI. Для цього використовувався тестовий маршрут, який повертав службову відповідь про стан застосунку. У

результаті перевірки було встановлено, що сервер запускається без помилок, приймає HTTP-запити та коректно повертає відповідь клієнту. Це підтвердило можливість подальшого підключення до серверної частини інших модулів системи.

Наступним етапом було протестовано роботу камери. Перевірялося, чи правильно ініціалізується бібліотека Picamera2, чи застосовуються задані параметри конфігурації та чи стабільно система отримує кадри у потрібній роздільній здатності. Під час налагодження було встановлено, що якість кадру суттєво залежить від параметрів фокусування, експозиції, контрастності та різкості. Тому ці налаштування були скориговані таким чином, щоб отримане зображення було достатньо чітким для подальшої детекції об’єктів.

Після перевірки джерела відеопотоку було протестовано модуль детекції об’єктів. На цьому етапі кадри передавалися до ШІ моделі, яка виконувалася на апаратному прискорювачі Hailo-8L. Перевірялося, чи коректно завантажується модель у форматі HEF, чи правильно виконується обробка кадру та чи повертаються результати у вигляді координат обмежувальних рамок, класів об’єктів і рівня достовірності. У процесі налагодження було перевірено відповідність розміру вхідного зображення вимогам моделі та коректність обробки результатів інференсу.

Окремо було протестовано механізм трекінгу об’єктів. Після отримання результатів детекції вони передавалися до трекера Roboflow Supervision, який присвоює об’єктам унікальні ідентифікатори. Під час перевірки було підтверджено, що один і той самий об’єкт у послідовних кадрах зберігає свій tracker_id, що є необхідним для стабільного супроводу. Також було перевірено поведінку системи у випадках, коли в кадрі немає об’єктів або коли об’єкт тимчасово зникає з поля зору.

Під час тестування візуалізації перевірялося нанесення рамок та ідентифікаторів на кадр. Було встановлено, що координати рамок у форматі хуху коректно передаються до функції відмальовування, а бібліотека OpenCV правильно наносить прямокутні рамки та текстові підписи з номерами об’єктів.

Отримані анотовані кадри використовувалися для подальшого відображення у вебінтерфейсі оператора.

Наступним етапом було протестовано обчислення зміщення активної цілі відносно центра кадру. Для цього перевірялося визначення центра обмежувальної рамки об’єкта та порівняння його з центром зображення. У результаті система формувала значення зміщення по горизонталі та вертикалі, які використовуються для подальшого створення керуючої команди. Додатково було перевірено роботу мертвої зони, яка дозволяє ігнорувати незначні зміщення та зменшити кількість зайвих коригувань.

Після цього було виконано тестування каналу передавання керуючих команд до плати керування. Перевірялося, чи правильно формується команда, чи відповідає вона заданому формату та чи стабільно передається до апаратної частини. Під час налагодження було перевірено швидкість обміну, правильність кодування повідомлень та реакцію системи на втрату або відсутність активної цілі.

Також було протестовано роботу WebSocket з’єднання, яке використовується для передавання команд оператора та службових повідомлень. Перевірялося встановлення з’єднання між браузером і сервером, приймання JSON-повідомлень, обробка типів команд та надсилання відповіді клієнту. У результаті було підтверджено, що WebSocket канал дозволяє оперативно передавати команди без необхідності створення окремого HTTP-запиту для кожної дії.

Окремо перевірялася робота відеотрансляції у вебінтерфейсі. Оброблений кадр із нанесеними рамками та ідентифікаторами передавався до клієнтської частини, де відображався у браузері оператора. Під час тестування оцінювалася затримка від моменту отримання кадру до його відображення у вебінтерфейсі, а також стабільність передавання відеопотоку під час тривалої роботи системи.

На завершальному етапі було виконано інтеграційне тестування всієї системи. У цьому режимі послідовно перевірялася робота повного циклу: отримання кадру з камери, виконання детекції, оновлення трекера, нанесення

рамок і ідентифікаторів, вибір активної цілі, обчислення зміщення, формування керуючої команди та передавання результатів у вебінтерфейс. Такий підхід дозволив переконатися, що окремі модулі коректно взаємодіють між собою та не виникає помилок під час передавання даних між етапами обробки.

Для стабільної роботи системи було визначено рекомендовані системні вимоги:

– одноплатний комп’ютер Raspberry Pi 5;

– накопичувач microSD або SSD обсягом від 64 ГБ;

– мова програмування Python 3.10 або новіша;

– камера CSI з підтримкою налаштування роздільної здатності, фокусування та експозиції;

– апаратний AI-прискорювач Hailo-8L;

– сучасний браузер із підтримкою WebRTC та WebSocket, наприклад Chrome, Chromium або Microsoft Edge;

– плата керування для взаємодії з механізмом наведення.

Після визначення рекомендованих параметрів було проведено перевірку роботи системи саме в такій конфігурації. У результаті тестування підтверджено, що обрана апаратна та програмна платформа забезпечує коректне отримання кадрів, виконання детекції, супровід об’єктів, формування керуючих команд і передавання обробленого відеопотоку у вебінтерфейс оператора.

3.3 Формування та тестування власного датасету зображень

Для коректної роботи модуля детекції об’єктів необхідно підготувати набір зображень, на основі якого буде виконуватися навчання та перевірка моделі. Хоча моделі сімейства YOLO мають готові попередньо навчені версії, які здатні розпізнавати велику кількість загальних класів об’єктів, їх використання не завжди є достатнім для конкретної прикладної задачі. Такі моделі навчені на універсальних наборах даних і добре підходять для базового розпізнавання поширених об’єктів.

Основним завданням цього етапу є підготовка даних у форматі, придатному для навчання моделі сімейства YOLO. Для цього зображення були зібрані, відфільтровані, розмічені за класами та поділені на навчальну, валідаційну й тестову вибірки. Навчальна вибірка використовується для безпосереднього тренування моделі, валідаційна – для контролю якості під час навчання, а тестова – для перевірки результатів після завершення тренування.

Під час формування датасету особлива увага приділялася коректності розмітки об’єктів. Для кожного зображення необхідно було вказати межі об’єкта за допомогою обмежувальної рамки та призначити відповідний клас. Саме ці дані надалі використовуються моделлю для вивчення ознак об’єктів і формування прогнозів під час роботи системи.

Для підготовки власного датасету було використано вебплатформу Roboflow, яка надає зручний інтерфейс для завантаження зображень, створення класів, ручної розмітки об’єктів, перевірки анотацій та експорту датасету у форматі, сумісному з моделями YOLO. Використання цієї платформи дозволяє виконати більшість етапів підготовки даних без написання додаткового коду, що значно спрощує процес створення навчального набору [16].

Після створення нового проєкту в Roboflow було обрано тип задачі Object Detection, оскільки система повинна не лише визначати клас об’єкта, а й знаходити його положення на зображенні. Для цього кожен об’єкт розмічається за допомогою обмежувальної рамки, яка задає координати його розташування.

На рисунку 3.1 зображено інтерфейс проєкту в середовищі Roboflow, у якому здійснюється підготовка власного датасету. У цьому вікні доступні інструменти для перегляду завантажених зображень, переходу до їх розмітки, перевірки анотацій, формування версій датасету та експорту даних у форматі, сумісному з YOLO. Надалі саме цей проєкт використовувався для створення навчальної, валідаційної та тестової вибірок.

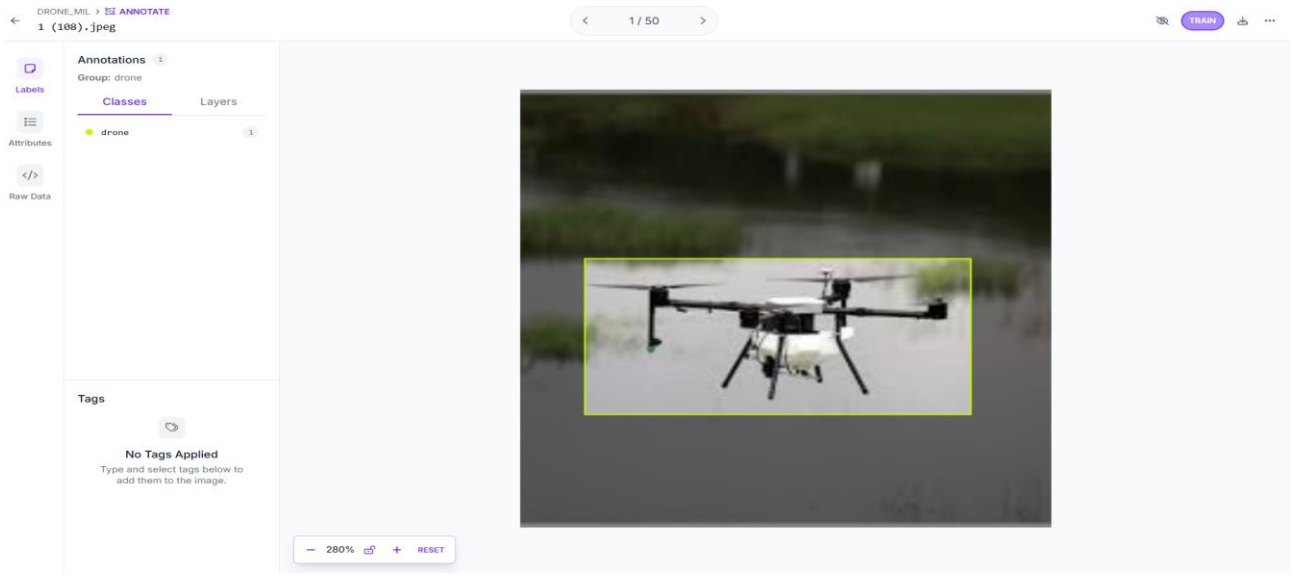

Рисунок 3.1 – Інтерфейс створеного проєкту датасету в Roboflow

Після завершення розмітки всіх зображень датасет необхідно підготувати до завантаження та подальшого використання під час тренування моделі. Для цього в Roboflow формується окрема версія датасету, у якій фіксуються всі розмічені зображення, класи об’єктів, анотації та налаштування поділу на навчальну, валідаційну й тестову вибірки. Перед завантаженням датасету Roboflow пропонує вибрати формат експорту, кодом чи за допомогою архіву, та для якої моделі буде використовуватись даний датасет (рис. 3.2).

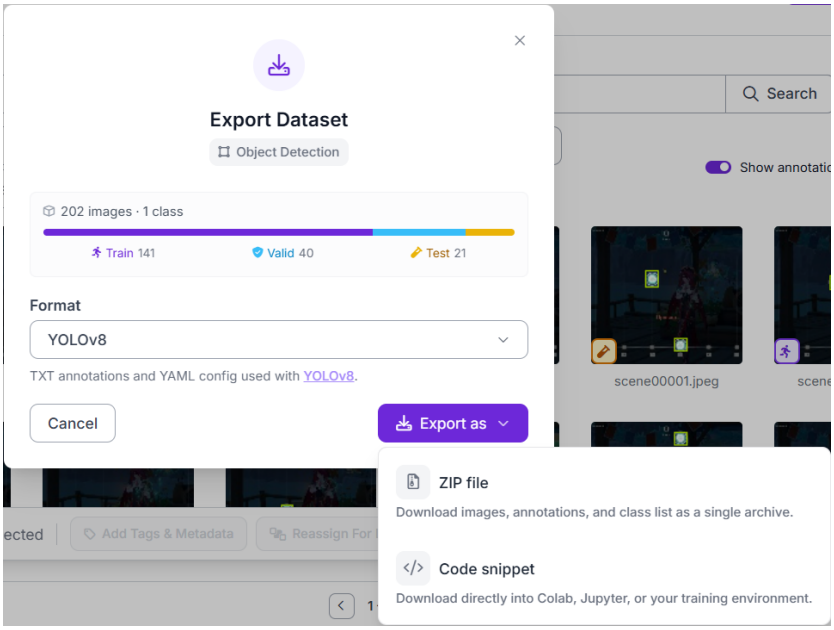

Рисунок 3.2 – Завантаження датасету

Після завершення розмітки зображень наступним етапом є експорт підготовленого датасету для подальшого використання під час навчання моделі. У середовищі Roboflow це можна виконати кількома способами: завантажити архів із файлами датасету, отримати пряме посилання або скористатися готовим фрагментом коду для інтеграції в проєкт чи Jupyter Notebook. У межах цієї роботи було обрано варіант експорту через код, оскільки він дозволяє швидко підключити датасет до проєкту без ручного завантаження та розпакування файлів. Вікно автоматично формує готовий фрагмент коду для завантаження даних (рис. 3.3).

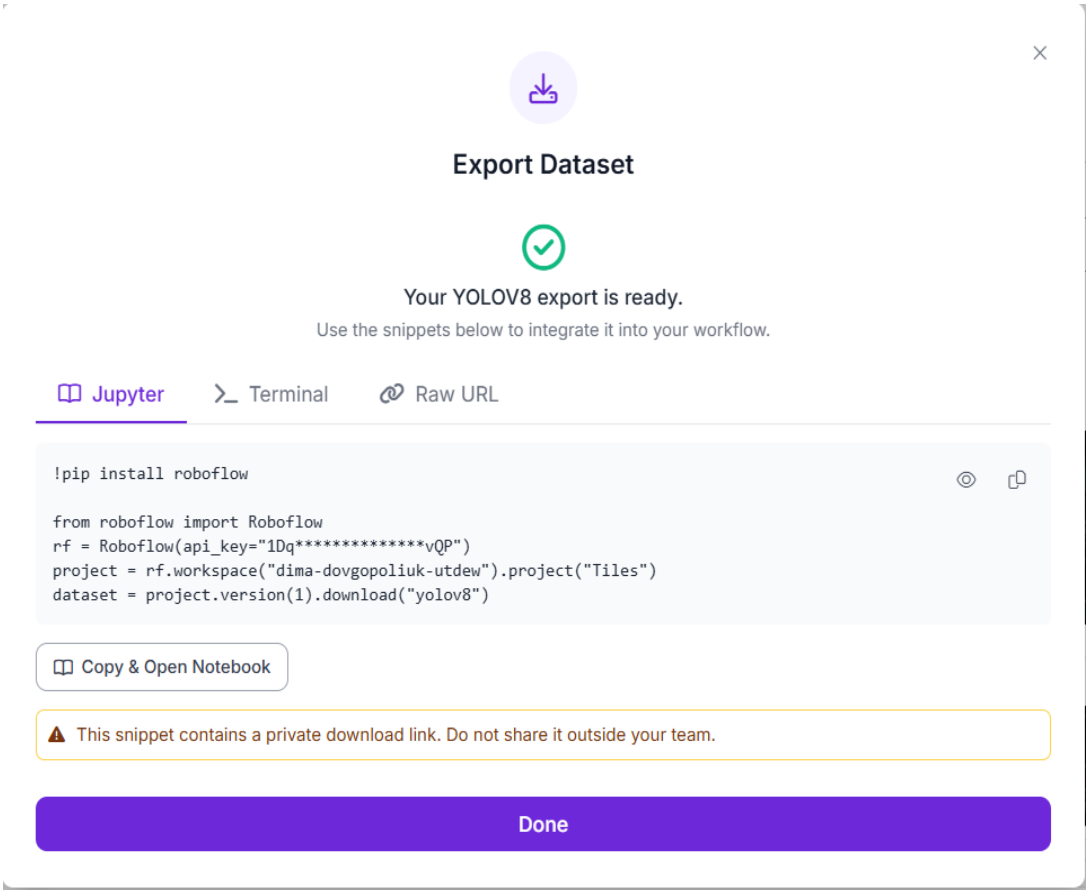

Рисунок 3.3 – Експорт датасету у форматі YOLOv8 з платформи Roboflow

Після завантаження датасету у структуру проєкту формується конфігураційний файл data.yaml (ліст. 3.9). Він є одним із основних файлів під час навчання моделі, оскільки саме в ньому вказуються шляхи до зображень навчальної, валідаційної та тестової вибірок, кількість класів і їхні назви.

Лістинг 3.9 – Приклад конфігураційного файлу

```
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 1
names: ['tiles']

roboflow:
  workspace: dima-dovgopoliuk-utdew
  project: dima-dovgopoliuk-utdew
  version: dataset
  license: Private
```

кінець лістингу 3.9

Під час запуску тренування YOLO модель зчитує цей файл і на його основі визначає, звідки брати зображення та які класи потрібно розпізнавати.

Після перевірки структури датасету та конфігураційного файлу можна переходити до запуску тренування моделі на власних даних. Для цього використовується попередньо навчена модель YOLO, яка донавчається на підготовленому датасеті. Такий підхід дозволяє не навчати модель з нуля, а використати вже сформовані базові ознаки попередньо навченої моделі та адаптувати її до конкретної задачі. Для керування процесом навчання було створено окремий скрипт, у якому задаються основні параметри тренування: шлях до моделі, шлях до конфігураційного файлу датасету, кількість епох, розмір зображення, розмір пакета даних, оптимізатор, швидкість навчання (ліст. 3.10).

Лістинг 3.10 – Скрипт тренування YOLO моделі

```
from ultralytics import YOLO

model = YOLO("yolo8s.pt")

model.train(
    data="data.yaml",
    epochs=100,
    imgsz=640,
    batch=16,
    device=0,
```

```
    optimizer="AdamW",
    lr0=0.001,
    weight_decay=0.0005,
    patience=20,
    project="camera_guidance_training",
    name="custom_yolo_model",
    pretrained=True,
    plots=True
)
```

кінець лістингу 3.10

Даний фрагмент коду запускає тренування YOLO моделі з використанням усіх основних параметрів, заданих у конфігурації. У ньому вказується шлях до файлу data.yaml, кількість епох навчання, розмір вхідного зображення, розмір пакета даних, оптимізатор, швидкість навчання та параметри збереження результатів. Після виконання скрипта створюється окрема папка, у якій зберігаються навчені ваги моделі, графіки метрик та службові файли тренування.

3.4 Експорт та квантизація моделі у формат HEF для Hailo-8L

Після тренування моделі на власному датасеті отримані ваги у форматі best.pt не можуть безпосередньо виконуватися на апаратному прискорювачі Hailo-8L. Для запуску моделі на цьому пристрої її потрібно перетворити у формат, сумісний із середовищем виконання Hailo. Для цього виконується кілька послідовних етапів: експорт моделі з формату PyTorch у формат ONNX, квантизація з використанням калібрувального набору зображень та компіляція у файл HEF. Саме HEF-файл надалі завантажується серверною частиною системи для виконання інференсу на Hailo-8L.

У лістингу 3.11 наведена команда, яка експортує навчену модель в формат ONNX.

Лістинг 3.11 – Експорт навченої моделі у формат ONNX

```
yolo export model=best.pt format=onnx imgsz=640 opset=11
```

кінець лістингу 3.11

Після експорту навченої YOLO моделі у формат ONNX наступним етапом є її парсинг засобами Hailo Dataflow Compiler. На цьому етапі ONNX модель аналізується та перетворюється у внутрішнє представлення Hailo, яке надалі використовується для оптимізації, квантизації та компіляції. Результатом парсингу є проміжний файл у форматі HAR, що містить структуру моделі у вигляді, придатному для подальшої роботи інструментів Hailo (ліст. 3.12).

Лістинг 3.12 – Парсинг ONNX моделі засобами Hailo Dataflow Compiler

```
from hailo_sdk_client import ClientRunner

ONNX_MODEL_PATH = "best.onnx"
MODEL_NAME = "custom_yolo_model"
HW_ARCH = "hailo8l"

runner = ClientRunner(hw_arch=HW_ARCH)

hn, npz = runner.translate_onnx_model(
    ONNX_MODEL_PATH,
    MODEL_NAME
)

runner.save_har(f"{MODEL_NAME}_parsed.har")
```

кінець лістингу 3.12

Для виконання парсингу використовується клас ClientRunner, у якому задається цільова апаратна архітектура hailo8l. Далі за допомогою методу translate_onnx_model() виконується зчитування ONNX файлу та його перетворення у формат Hailo Archive.

Після успішного парсингу ONNX моделі необхідно виконати її оптимізацію та квантизацію. Цей етап потрібний для адаптації моделі до виконання на апаратному прискорювачі Hailo-8L, оскільки модель у початковому вигляді використовує числове представлення, яке не є оптимальним для AI пристрою. Для квантизації використовується калібрувальний набір зображень, що дозволяє підібрати параметри перетворення ваг і активацій моделі без суттєвої втрати точності.

Приклад квантинізації та оптимізації наведений в лістингу 3.13.

Лістинг 3.13 – Оптимізація та квантизація моделі

```
import os
import numpy as np
from PIL import Image
from hailo_sdk_client import ClientRunner

MODEL_NAME = "custom_yolo_model"
CALIBRATION_DIR = "calibration/images"
INPUT_SIZE = (640, 640)

runner = ClientRunner(har=f"{MODEL_NAME}_parsed.har")

def load_calibration_data(path):
    images = []

    for file_name in os.listdir(path):
        if file_name.endswith((".jpg", ".png", ".jpeg")):
            image = Image.open(os.path.join(path,
file_name)).convert("RGB")
            image = image.resize(INPUT_SIZE)
            images.append(np.asarray(image, dtype=np.float32))

    return np.array(images)

calib_data = load_calibration_data(CALIBRATION_DIR)

runner.optimize(calib_data)
runner.save_har(f"{MODEL_NAME}_optimized.har")
```

кінець лістингу 3.13

І фінальним етапом є компіляція моделі в той тип, який якраз потрібний для запуску моделі на прискорювачі Hailo-8L. На цьому етапі модель перетворюється у виконуваний файл, який містить у собі дані необхідні для роботи на апаратному прискорювачі (ліст. 3.14).

Лістинг 3.14 – Компіляція моделі у фінальний формат

```
from hailo_sdk_client import ClientRunner

MODEL_NAME = "custom_yolo_model"
runner = ClientRunner(har=f"{MODEL_NAME}_optimized.har")
hef = runner.compile()

with open(f"{MODEL_NAME}.hef", "wb") as hef_file:
    hef_file.write(hef)
```

кінець лістингу 3.14

Після отримання HEF-файлу модель можна інтегрувати у серверну частину системи. Для цього шлях до файлу передається у модуль HailoDetector, який завантажує модель під час запуску програми та використовує її для обробки вхідних кадрів. Таким чином, після проходження всіх етапів підготовки модель стає придатною для виконання на апаратному прискорювачі Hailo-8L у складі розроблюваної системи.

У результаті виконання цього підпункту було отримано модель у форматі HEF, оптимізовану для запуску на Hailo-8L. Послідовне виконання експорту, парсингу, квантизації та компіляції дозволило адаптувати навчену YOLO модель до апаратного прискорювача та підготувати її до використання в режимі реального часу.

3.5 Розробка вебінтерфейсу оператора з використанням WebRTC і WebSocket

Після реалізації серверної частини, підготовки моделі детекції та її експорту для виконання на апаратному прискорювачі було розроблено вебінтерфейс оператора. Його основним призначенням є відображення відеопотоку з камери, перегляд результатів детекції та трекінгу, контроль поточного стану системи, а також передавання команд керування до серверної частини.

Вебінтерфейс було реалізовано у вигляді односторінкового клієнтського застосунку, який відкривається у браузері (рис. 3.4). Такий підхід є зручним, оскільки оператору не потрібно встановлювати додаткове програмне забезпечення на клієнтський пристрій. Для доступу до системи достатньо відкрити відповідну адресу сервера у браузері, після чого користувач отримує доступ до відеопотоку, панелі керування та службової інформації про стан системи. Інтерфейс побудовано таким чином, щоб основні елементи керування були доступні з одного екрана, без необхідності переходу між окремими сторінками.

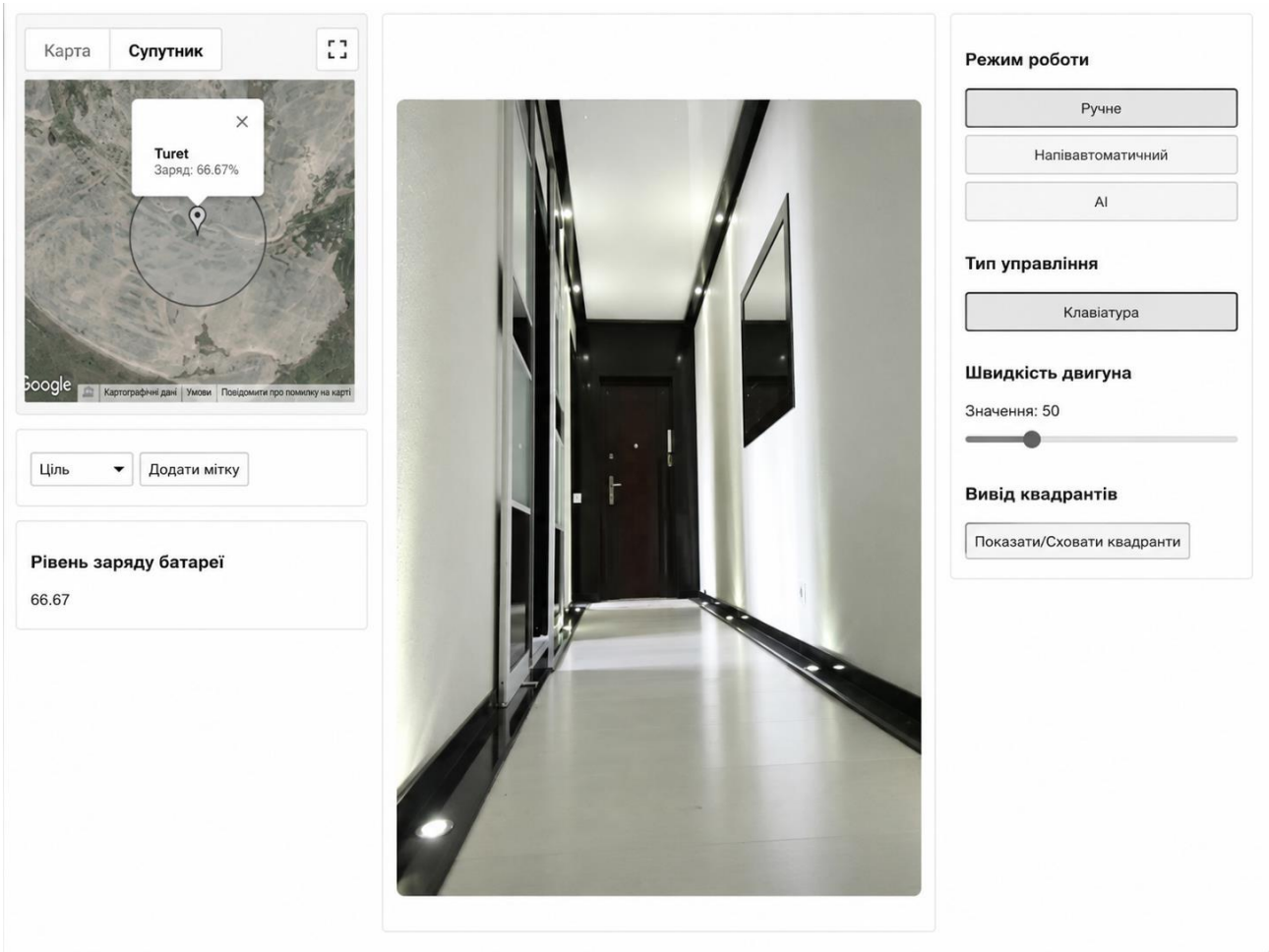

Рисунок 3.4 – Вебінтерфейс реалізованої системи

Основна область інтерфейсу призначена для відображення відеопотоку. У цьому блоці виводиться зображення з камери, на яке серверна частина накладає рамки виявлених об’єктів та їхні ідентифікатори. Завдяки цьому оператор може бачити не лише початкове зображення, а й результати роботи модуля комп’ютерного зору. Додатково в інтерфейсі передбачено панель керування, де відображаються режими роботи системи, стан підключення та елементи для взаємодії із системою.

Для передавання відеопотоку в браузер було використано технологію WebRTC. Вона дозволяє організувати передавання відео з низькою затримкою, що є важливим для систем, які працюють у режимі реального часу. На відміну від передавання відео через звичайні HTTP запити, WebRTC створює окреме з’єднання між сервером і браузером, через яке кадри можуть передаватися без

значних затримок. Це дозволяє оператору бачити актуальний стан сцени та швидше реагувати на зміни у відеопотоці.

Передавання службових команд було реалізовано окремо від відеопотоку. Для цього використовується WebSocket з'єднання, яке забезпечує постійний двосторонній канал зв'язку між вебінтерфейсом і серверною частиною. Через цей канал можуть передаватися команди зміни режиму роботи, службові повідомлення, інформація про стан підключення та інші дані, які не належать безпосередньо до відеопотоку. Такий поділ є доцільним, оскільки WebRTC використовується для відео, а WebSocket – для команд і стану системи.

Загальна структура вебінтерфейсу містить HTML елемент для відеопотоку, блок кнопок для вибору режиму роботи та область для відображення службового стану системи.

Приклад базової структури сторінки наведено у лістингу 3.15.

Лістинг 3.15 – Спрощена структура вебінтерфейсу

```
<main>
    <section class="video-area">
        <video id="video" autoplay playsinline></video>
    </section>

    <section class="control-panel">
        <button onclick="setMode('MANUAL')">Manual</button>
        <button onclick="setMode('SEMI_AUTO')">Semi</button>
        <button onclick="setMode('AUTO')">AI</button>

        <div id="system-status">Статус: очікування
підключення</div>
    </section>
</main>
```

кінець лістингу 3.15

Елемент video використовується для відображення відеопотоку, отриманого через WebRTC. Блок control-panel містить кнопки для перемикання режимів роботи системи та службову область для відображення стану підключення. У реальній реалізації цей інтерфейс буде доповнений списком

виявлених об’єктів, показниками частоти кадрів, повідомленнями про стан системи та іншими елементами.

Для передавання команд із вебінтерфейсу до серверної частини було реалізовано клієнтський WebSocket. Він встановлює з’єднання з відповідним маршрутом сервера та надсилає повідомлення у форматі JSON. Приклад такого підключення наведено у лістингу 3.16.

Лістинг 3.16 – Передавання команд з вебінтерфейсу через WebSocket

```
const s = new WebSocket("ws://localhost:8000/ws/control");
function setMode(mode) {
    s.send(JSON.stringify({
        type: "set_mode",
        mode: mode}));}

function selectTarget(id) {
    socket.send(JSON.stringify({
        type: "select_target",
        target_id: id
    }));}
```

кінець лістингу 3.16

Для підключення відеопотоку у вебінтерфейсі використовується об’єкт RTCPeerConnection. Він відповідає за створення WebRTC з’єднання між браузером і серверною частиною. Після отримання відеотреку браузер підключає його до HTML елемента video, де потік відображається оператору. Приклад клієнтської частини підключення WebRTC наведено у лістингу 3.17.

Лістинг 3.17 – Підключення WebRTC потоку у браузері

```
const pc = new RTCPeerConnection();

pc.ontrack = (event) => {
    const video = document.getElementById("video");
    video.srcObject = event.streams[0];
};

async function startVideo() {
    const offer = await pc.createOffer();
    await pc.setLocalDescription(offer);
```

```
    const response = await fetch("/offer", {
        method: "POST",
        body: JSON.stringify(pc.localDescription),
        headers: {"Content-Type": "application/json"}
    });

    const answer = await response.json();
    await pc.setRemoteDescription(answer);
}
```

кінець лістингу 3.17

Даний код відповідає за створення WebRTC з’єднання на стороні браузера. Спочатку створюється об’єкт RTCPeerConnectio, після чого визначається обробник, який підключає отриманий потік до елемента який відповідає за відео.

У результаті реалізації вебінтерфейсу оператор отримує можливість переглядати відеопотік у браузері, бачити результати детекції та трекінгу, а також передавати команди до серверної частини. Використання WebRTC забезпечує передавання відео з низькою затримкою, а WebSocket дозволяє організувати швидкий обмін командами та службовими повідомленнями. Такий підхід забезпечує зручну взаємодію користувача із системою та відповідає вимогам до роботи в режимі реального часу.

ВИСНОВКИ

У межах кваліфікаційної роботи бакалавра було розроблено програмно-апаратну систему автоматичного наведення камери на об’єкти з використанням методів комп'ютерного зору. Поставлену мету роботи досягнуто, усі сформульовані завдання виконано в повному обсязі.

Виконано аналіз сучасного стану проблеми та розглянуто існуючі аналоги: наукові публікації з детекції й трекінгу об’єктів із використанням методів комп’ютерного зору, готові PTZ-камери з функцією автостеження (Axis, Dahua), а також спеціалізовані промислові системи (Sky Sentinel). Аналіз показав, що більшість готових рішень має закриту архітектуру та обмежені можливості адаптації, що обґрунтовує доцільність розробки власної відкритої системи.

Сформульовано функціональні та нефункціональні вимоги до розроблюваної системи. Визначено, що система повинна забезпечувати отримання відеопотоку з камери, виявлення об'єктів, присвоєння їм унікальних ідентифікаторів, супровід активної цілі, формування керуючих команд, передавання відеопотоку у вебінтерфейс. Складено специфікацію вимог і побудовано UML-моделі: діаграму прецедентів, діаграму класів, діаграму компонентів, діаграму станів і діаграму послідовності.

Обґрунтовано вибір апаратної та програмної платформи. Як основний обчислювальний пристрій обрано Raspberry Pi 5, а для прискорення ші детекції – Hailo-8L. Серверну частину реалізовано мовою Python із використанням FastAPI, для отримання кадрів застосовано Picamera2, для детекції об'єктів – модель сімейства YOLO, для трекінгу – бібліотеку Roboflow Supervision на основі алгоритму ByteTrack. Передавання відеопотоку організовано засобами WebRTC, обмін командами й службовими повідомленнями – через WebSocket.

Сформовано власний датасет зображень для навчання моделі детекції. За допомогою платформи Roboflow виконано завантаження зображень, ручне маркування об’єктів обмежувальними рамками, поділ даних на навчальну, валідаційну й тестову вибірки та експорт датасету у форматі, сумісному з YOLO.

Спроєктовано архітектуру програмно-апаратного комплексу, що ґрунтується на розподілі функцій між клієнтською, серверною та апаратною частинами. Визначено основні модулі системи – отримання відеопотоку, детекції об’єктів, трекінгу, вибору активної цілі, формування керуючих команд, авторизації та вебкомунікації – а також встановлено порядок їхньої взаємодії від моменту отримання кадру до передавання команди на сервоприводи. Такий поділ забезпечує модульність системи та спрощує її подальше доопрацювання.

Реалізовано програмні модулі серверної частини системи: модуль отримання відеопотоку з камери на основі бібліотеки Picamera2, модуль детекції об'єктів із виконанням інференсу YOLO моделі на апаратному прискорювачі Hailo-8L, модуль супроводу цілі на основі алгоритму ByteTrack із присвоєнням об’єктам сталих ідентифікаторів, а також модуль формування керуючих команд, який обчислює зміщення активної цілі відносно центра кадру та передає відповідні команди на плату Arduino. Взаємодію між модулями організовано у вигляді конвеєра обробки кадру, що забезпечує безперервну роботу системи в режимі реального часу.

Реалізовано вебінтерфейс оператора у вигляді односторінкового клієнтського застосунку, який забезпечує перегляд відеопотоку з накладеними рамками виявлених об’єктів, перемикання режимів роботи системи та передавання команд керування до серверної частини. Відеопотік передається засобами WebRTC для низької затримки, а команди й службові повідомлення – через WebSocket з’єднання.

Проведено тестування розробленої інформаційно-комп’ютерної системи, у ході якого перевірено роботу всіх основних модулів – отримання відеопотоку, детекції об’єктів, трекінгу з присвоєнням ідентифікаторів, формування керуючих команд і передавання відео у вебінтерфейс. Результати тестування підтвердили коректну взаємодію модулів між собою, стабільність роботисистеми в режимі реального часу та відповідність її поведінки сформульованим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Reis D., Kupec J., Hong J., Daoudi A. Real-Time Flying Object Detection with YOLOv8. arXiv:2305.09972, 2023. URL: https://arxiv.org/abs/2305.09972 (дата звернення: 20.04.2026).

2. Zhang Y., Sun P., Jiang Y., Yu D., Weng F., Yuan Z., Luo P., Liu W., Wang X. ByteTrack: Multi-Object Tracking by Associating Every Detection Box. arXiv:2110.06864, 2021. URL: https://arxiv.org/abs/2110.06864 (дата звернення: 20.04.2026).

3. Chen C. W., Hung H. A., Yang P. H., Cheng T. H. Visual Servoing of a Moving Target by an Unmanned Aerial Vehicle, 2021. vol. 21. № 17. 5708. URL: https://www.mdpi.com/1424-8220/21/17/5708 (дата звернення: 20.04.2026).

4. ONVIF. ONVIF PTZ Service Specification. Ver. 25.12. ONVIF, 2025. URL: https://www.onvif.org/specs/srv/ptz/ONVIF-PTZ-Service-Spec.pdf (дата звернення: 22.04.2026).

5. Axis Communications. AXIS Perimeter Defender PTZ Autotracking. URL: https://www.axis.com/products/axis-perimeter-defender-ptz-autotracking (дата звернення: 22.04.2026).

6. Dahua Technology. Auto Tracking 3.0. URL: https://www.dahuasecurity.com/products/key-technologies/937 (дата звернення: 27.04.2026).

7. UNITED24. Sky Sentinel fundraising campaign. URL: https://u24.gov.ua/uk/news/sky-sentinel-fundraiser-completed-3300000-raised-for-22-air-defense-units (дата звернення: 27.04.2026).

8. Raspberry Pi Foundation. Raspberry Pi 5 Product Brief. URL: https://www.raspberrypi.com/products/raspberry-pi-5/ (дата звернення: 05.05.2026).

9. Ramírez S. FastAPI Documentation. 2024. URL: https://fastapi.tiangolo.com (дата звернення: 27.04.2026).

10. Raspberry Pi Foundation. Picamera2 Library Documentation. 2023. URL: https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf (дата звернення: 03.05.2026).

11. Ultralytics. YOLOv8 Documentation. 2023. URL: https://docs.ultralytics.com (дата звернення: 03.05.2026).

12. Hailo Technologies. Hailo-8L AI Accelerator. URL: https://hailo.ai/products/ai-accelerators/hailo-8l-m-2-ai-acceleration-module-for-ai-light-applications/ (дата звернення: 06.05.2026).

13. Roboflow. Supervision – Computer Vision Toolkit. 2023. URL: https://supervision.roboflow.com (дата звернення: 07.05.2026).

14. W3C. WebRTC 1.0: Real-Time Communication Between Browsers. 2021. URL: https://www.w3.org/TR/webrtc/ (дата звернення: 08.05.2026).

15. OpenCV Team. OpenCV Documentation. 2024. URL: https://docs.opencv.org (дата звернення: 10.05.2026).

16. Roboflow. Computer Vision Platform. URL: https://roboflow.com (дата звернення: 13.05.2026).