Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра комп'ютерних систем і мереж

Покришка Роман Іванович

УДК 004.932

БАКАЛАВРСЬКА РОБОТА

Розробка алгоритмічно-програмних рішень виявлення типових перешкод на основі згорткової нейромережі

Комп'ютерна інженерія
(назва освітньої програми)

123 - Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____*Покришка Р.І.*_____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____*Топалов А.М., доцент*_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри КСМ

*д.т.н.,професор*_____*С.І. Мельничук*
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026 рік

Івано-Франківський національний технічний університет нафти і газу

(повне найменування вищого навчального закладу)

Факультет *інформаційних технологій*

Кафедра *комп'ютерних систем і мереж*

Освітній ступінь *бакалавр*

Спеціальність *123 – Комп'ютерна інженерія*

(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КСМ
(С.І. Мельничук)
«___» _____ 2026 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Покришці Роману Івановичу

(прізвище, ім'я, по батькові)

1. **Тема проекту (роботи)** *Розробка алгоритмічно-програмних рішень виявлення типових перешкод на основі згорткової нейромережі*

керівник проекту (роботи) *Топалов А.М., доцент.*

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «21*» квітня 2026 року № 180/7*

2. Строк подання студентом роботи *11 червня 2026 р.*

3. Вихідні дані до роботи *Матеріали і результати отримані під час проходження переддипломної практики, методичні вказівки, технічна література.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити). *1. Аналіз предметної області методів виявлення перешкод. 2. Розробка алгоритму виявлення перешкод. 3. Програмна реалізація алгоритму виявлення перешкод.*

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 02.02.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області	*Лютий 2026*	
	Розробка алгоритму виявлення перешкод	*Березень 2026*	
2	Програмна реалізація	*Квітень - Травень, 2026*	
3	Експериментальні дослідження	*Травень 2026*	
4	Оформлення роботи	*Червень 2026*	

Студент _____ *Покришка Р.І.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Топалов А.М.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Робота присвячена розробці системи виявлення типових перешкод у приміщеннях для осіб з порушенням зору на основі згорткової нейронної мережі YOLOv8n з формуванням голосових навігаційних підказок.

У роботі проведено аналіз класичних методів комп'ютерного зору та методів глибокого навчання для задачі виявлення перешкод, здійснено порівняльний аналіз архітектур CNN-детекторів та обґрунтовано вибір YOLOv8n як базової моделі. Визначено множину шести типових перешкод у приміщеннях: людина, стілець, стіл, двері, сходи, сміттєвий бак.

Розроблено алгоритм виявлення перешкод на основі аналізу центральної зони кадру та алгоритм генерування голосових повідомлень з механізмом cooldown і асинхронним відтворенням. Для розширення можливостей базової моделі застосовано стратегію fine-tuning на датасеті з 3 277 анотованих зображень трьох нових класів, отриманих з платформи Roboflow Universe.

Програмну систему реалізовано на мові Python 3.10 у вигляді чотирьох модулів із використанням бібліотек Ultralytics YOLOv8, OpenCV та pyttsx3. Проведені експериментальні дослідження підтвердили ефективність підходу: після донавчання загальне значення mAP50 зросло з 0.141 до 0.793, Recall — з 0.097 до 0.799. Класи «двері» та «сміттєвий бак» досягли mAP50 на рівні 0.993 та 0.995 відповідно.

Ключові слова: згорткова нейронна мережа, виявлення перешкод, YOLOv8, комп'ютерний зір, transfer learning, асистивні технології, голосові повідомлення, глибоке навчання, детекція об'єктів, навігація незрячих.

ABSTRACT

The thesis is devoted to the development of a system for detecting typical indoor obstacles for visually impaired persons based on the YOLOv8n convolutional neural network with voice navigation prompts generation.

The work presents an analysis of classical computer vision methods and deep learning approaches for the obstacle detection task, a comparative analysis of CNN detector architectures, and a justification for selecting YOLOv8n as the base model. A set of six typical indoor obstacles has been defined: person, chair, dining table, door, stairs, and trash can.

An obstacle detection algorithm based on central frame zone analysis has been developed, along with a voice message generation algorithm featuring a cooldown mechanism and asynchronous audio playback. To extend the capabilities of the base model, a fine-tuning strategy was applied on a dataset of 3,277 annotated images of three new classes obtained from the Roboflow Universe platform.

The software system is implemented in Python 3.10 as four specialized modules using the Ultralytics YOLOv8, OpenCV, and pyttsx3 libraries. Experimental evaluation confirmed the effectiveness of the proposed approach: after fine-tuning, the overall mAP50 increased from 0.141 to 0.793, and Recall improved from 0.097 to 0.799. The "door" and "trash can" classes achieved mAP50 values of 0.993 and 0.995 respectively.

Keywords: convolutional neural network, obstacle detection, YOLOv8, computer vision, transfer learning, assistive technologies, voice messages, deep learning, object detection, blind navigation.

ЗМІСТ

ВСТУП

Актуальність теми. За даними Всесвітньої організації охорони здоров'я, понад 2,2 мільярда людей у світі мають порушення зору, з яких щонайменше 43 мільйони є повністю сліпими. Однією з ключових проблем для цієї категорії населення залишається безпечне самостійне пересування у приміщеннях – орієнтація серед меблів, дверей, сходів та інших типових перешкод. Традиційні засоби – тростини та супровід – не забезпечують достатнього рівня автономності та безпеки.

Стрімкий розвиток методів глибокого навчання та комп'ютерного зору відкрив принципово нові можливості для вирішення задач виявлення об'єктів у реальному часі. Зокрема, згорткові нейронні мережі (CNN) довели свою ефективність у широкому спектрі задач розпізнавання образів [1, 2]. Серед архітектур детекції об'єктів особливе місце посідає сімейство YOLO (You Only Look Once), яке забезпечує обробку зображень за один прохід мережі з високою швидкістю та точністю [3]. Остання версія – YOLOv8 – демонструє стан мистецтва на еталонних наборах даних при збереженні продуктивності реального часу [4, 5].

Актуальність розробки алгоритмічно-програмних рішень виявлення типових перешкод на основі CNN обумовлена як соціальною значущістю задачі – надання допомоги людям з порушенням зору при пересуванні у приміщеннях, – так і широким практичним застосуванням у сфері мобільної робототехніки та автономних систем навігації. Попри значну кількість досліджень у галузі систем допомоги незрячим [6, 7, 8], задача виявлення типових перешкод у приміщеннях із одночасним формуванням голосових підказок щодо напрямку обходу залишається недостатньо дослідженою.

Мета роботи – розробка алгоритмічно-програмних рішень виявлення типових перешкод у приміщеннях на основі згорткової нейронної мережі з формуванням голосових повідомлень для осіб з порушенням зору.

Для досягнення поставленої мети вирішуються такі задачі:

1) провести аналіз існуючих методів виявлення перешкод та визначити переваги підходів на основі CNN;

2) здійснити порівняльний аналіз архітектур CNN-детекторів і обґрунтувати вибір YOLOv8n як базової моделі;

3) визначити множину типових перешкод у приміщенні та сформувати відповідний датасет для донавчання моделі;

4) розробити алгоритм виявлення перешкод із аналізом центральної зони кадру та визначенням напрямку обходу;

5) розробити алгоритм формування голосових повідомлень з урахуванням часового інтервалу між сповіщеннями;

6) реалізувати програмну систему та провести експериментальні дослідження для оцінки якості моделі до та після донавчання.

Об'єкт дослідження – процес виявлення типових перешкод у приміщеннях за допомогою методів глибокого навчання.

Предмет дослідження – алгоритмічно-програмні рішення виявлення перешкод на основі згорткової нейронної мережі YOLOv8 з формуванням голосових повідомлень.

Методи дослідження. У роботі використано методи глибокого навчання (зокрема, transfer learning та fine-tuning [9]), методи комп'ютерного зору для обробки відеопотоку, методи порівняльного аналізу архітектур нейронних мереж, а також методи оцінки якості детекції об'єктів (mAP50, Precision, Recall).

Практична цінність роботи полягає у наступному:

– розроблена система може бути використана як асистивний засіб для осіб з порушенням зору під час самостійного пересування у приміщеннях (класах, лікарнях, офісах тощо);

– запропонований алгоритм аналізу центральної зони кадру та визначення напрямку обходу є універсальним і може бути застосований у системах навігації мобільних роботів у приміщеннях;

– розроблені програмні модулі мають модульну архітектуру і допускають розширення переліку виявлюваних класів перешкод шляхом донавчання моделі на нових датасетах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ КОМП'ЮТЕРНОГО ЗОРУ

1.1 Проблематика навігації незрячих та аналіз класичних методів виявлення перешкод

Задача виявлення перешкод є однією з фундаментальних і найбільш критичних у галузях комп'ютерного зору, робототехніки та розробки асистивних технологій. Вона полягає у автоматичному визначенні, локалізації та класифікації об'єктів у просторі, які можуть становити потенційну небезпеку або заважати вільному пересуванню людини чи автономного мобільного пристрою. Для систем допомоги людям із порушеннями зору ця задача набуває особливої соціальної та технічної важливості, оскільки безпомилкова робота алгоритму в режимі реального часу безпосередньо впливає на безпеку та мобільність користувача.

Історично підходи до вирішення цієї проблеми пройшли еволюцію від простих ультразвукових та інфрачервоних сенсорів відстані до складних систем оптичного розпізнавання. Залежно від математичного апарату та архітектури обробки даних, сучасні методи комп'ютерного зору для виявлення перешкод поділяються на дві великі категорії:

1. Класичні методи, що базуються на детермінованих алгоритмах та ручному конструюванні ознак.

2. Методи на основі глибокого машинного навчання (Deep Learning), що використовують штучні нейронні мережі.

Класичні методи комп'ютерного зору ґрунтуються на припущенні, що об'єкти на зображенні можна описати за допомогою специфічних геометричних або текстурних маркерів (інтенсивність пікселів, градієнти, контури), які заздалегідь визначені розробником алгоритму.

Серед найбільш поширених і класичних підходів виділяється метод гістограм орієнтованих градієнтів (Histogram of Oriented Gradients, HOG), запропонований дослідниками в [10] у 2005 році. Математична суть HOG полягає в аналізі розподілу напрямків градієнтів інтенсивності у локальних областях зображення (клітинах або "cells"). Для кожної клітини будується гістограма, де кошики (bins) відповідають певним діапазонам кутів градієнта. Отримані дескриптори нормалізуються за блоками для компенсації змін освітлення, після чого об'єднуються у фінальний вектор ознак. Цей вектор подається на вхід лінійного класифікатора – методу опорних векторів (Support Vector Machine, SVM). Обробка всього кадру відбувається за допомогою концепції ковзного вікна (sliding window), яке сканує зображення з різним масштабом [11].

Хоча метод HOG+SVM продемонстрував високу ефективність свого часу (зокрема, в задачах детекції пішоходів), він має суттєві обмеження:

- висока чутливість до умов освітлення: тіні, бліки та слабке світло різко спотворюють градієнти;

- обчислювальна неефективність: метод ковзного вікна вимагає багаторазового перерахунку дескрипторів для тисяч піксельних зон, що призводить до низької швидкості обробки кадрів;

- жорсткість ознак: метод не здатний адаптуватися до суттєвих змін ракурсу об'єкта чи його деформації (оклюзії).

До цієї ж групи класичних підходів належить метод каскадних класифікаторів Віоли–Джонса (Viola-Jones). Він базується на примітивах Хаара (Haar-like features), інтегральних зображеннях для миттєвого обчислення ознак та алгоритмі покрокового навчання AdaBoost, який відбирає найбільш інформативні маркерні зони. Віола-Джонс забезпечує високу швидкість роботи, але демонструє незадовільні результати при поворотах об'єктів або зміні кута зйомки [11].

Іншим відомим підходом є метод масштабно-інваріантних перетворень ознак (Scale-Invariant Feature Transform, SIFT). SIFT виділяє стійкі ключові точки на зображенні, стійкі до масштабних трансформацій, поворотів та афінних спотворень. Проте математична складність обчислення дескрипторів SIFT унеможливлює його стабільне використання в системах реального часу без потужного графічного прискорювача.

Незважаючи на простоту математичної інтерпретації та відсутність потреби у великих навчальних вибірках, усі класичні підходи демонструють недостатню точність в умовах складного динамічного фону, часткового перекриття (оклюзії) об'єктів та непередбачуваної варіативності освітлення, що є типовим для реальних житлових та міських приміщень.

1.2 Розвиток методів глибокого навчання та архітектурні компоненти CNN

Якісний стрибок у задачах комп'ютерного зору відбувся з появою глибоких штучних нейронних мереж, які на сьогодні суттєво переважають класичні підходи за всіма ключовими метриками точності виявлення. Ключовою та фундаментальною перевагою глибокого навчання є здатність моделей до автоматичного вилучення ієрархічних ознак (end-end learning) безпосередньо з сирих підготовлених даних (масивів пікселів). Це повністю усуває суб'єктивний та трудомісткий етап ручного конструювання математичних дескрипторів. На перших шарах мережа вчиться розпізнавати прості елементи (лінії, межі, кути), на середніх – геометричні фігури й текстури, а на глибоких – складні семантичні об'єкти (крісла, двері, людей).

За даними профільних наукових оглядів, сучасні детектори на основі згорткових нейронних мереж (Convolutional Neural Networks, CNN) перевершують класичні каскади та дескриптори за метрикою середньої точності (mAP – Mean Average Precision) на 20–40% під час тестування на

еталонних світових наборах даних, таких як PASCAL VOC та Microsoft COCO. Аналіз існуючих у світі систем допомоги незрячим та слабозорим людям чітко вказує на те, що переважна більшість сучасних рішень та прототипів базується саме на архітектурах Deep Learning:

- в [6] розробили мобільну систему на основі модифікованої архітектури YOLOv5, адаптовану для розпізнавання статичних та динамічних перешкод як в умовах замкнутого простору (indoor), так і на відкритих вулицях (outdoor).

- в [7] реалізували комплексне рішення на базі архітектури SSDLite з мобільним інваріантом MobileNetV2. Особливістю їхньої системи є генерація аудіо- та текстового опису оточення для детального інформування незрячої людини про геометрію кімнати.

- у дослідженні [8] було успішно впроваджено інтелектуальну навігаційну систему на базі автоматичного пошуку нейроархітектур (Neural Architecture Search, NAS), яка дозволила досягти показника mAP понад 80% при збереженні високої енергоефективності.

Таким чином, використання CNN є найбільш перспективним та науково обґрунтованим напрямком для вирішення задачі локалізації та виявлення перешкод у реальному часі. Це особливо актуально в умовах жорстко обмежених обчислювальних ресурсів, низького тепловиділення та обмеженої ємності акумуляторів мобільних або носимих пристроїв [4,5].

На відміну від стандартних повнозв'язних (Dense) шарів, де кожен нейрон пов'язаний з кожним пікселем (що призводить до вибухового зростання кількості параметрів), CNN використовують математичну операцію згортки (convolution). Згортка передбачає рух невеликого ядра (матриці ваг) по всьому зображенню. Це забезпечує дві фундаментальні властивості:

1. Локальна зв'язність (local connectivity): нейрон обробляє лише обмежену просторову область зображення, ефективно моделюючи взаємозв'язки між сусідніми пікселями, що критично для точної локалізації меж об'єктів.

2. Розділення ваг (weight sharing): Одне й те саме ядро згортки застосовується до всього кадру, що кардинально зменшує кількість параметрів мережі та забезпечує інваріантність алгоритму до зсувів, поворотів та змін масштабу шуканого об'єкта.

Сучасна архітектура типової CNN для детекції об'єктів має чітку трикомпонентну блочну структуру (рис. 1.1)

- Backbone (основа або ствол мережі): відповідає за первинне вилучення просторових ознак та формування ієрархічних карт ознак (feature maps) на різних рівнях абстракції. Зазвичай будується на базі перевірених високоефективних архітектур, таких як ResNet, DarkNet або CSPNet.

- Neck (шия мережі): проміжний шар, який реалізується у вигляді пірамідальних мереж ознак, таких як Feature Pyramid Network (FPN) або Path Aggregation Network (PAN). Головна задача Neck – забезпечити ефективне об'єднання та обмін інформацією між картами ознак різних масштабів (глибоких семантичних та поверхневих просторових). Це критично важливо для одночасного точного виявлення як великих об'єктів поблизу, так і дрібних перешкод на відстані.

- Head (голова детектора): Кінцевий компонент, який на основі агрегованих з Neck ознак здійснює безпосередню генерацію прогнозів. Робота "голови" полягає в розрахунку координат обмежувальних рамок (bounding boxes), визначенні ймовірностей належності до конкретного класу (class probabilities) та оцінці ступеня впевненості мережі у знайденому об'єкті (confidence scores) [4].

Додатково всі сучасні CNN-детектори поділяються на два методологічні класи:

1. Двопрохідні (two-stage) детектори: Яскравим представником є сімейство Faster R-CNN [12]. Процес детекції розділений на два послідовні кроки. Спочатку окремий підмодуль – мережа генерації регіональних пропозицій (Region Proposal Network, RPN) – знаходить зони, де потенційно є

об'єкт. На другому етапі ці зони класифікуються та уточнюються. Це гарантує високу точність локалізації, але суттєво обмежує швидкість роботи моделі через дублювання обчислень.

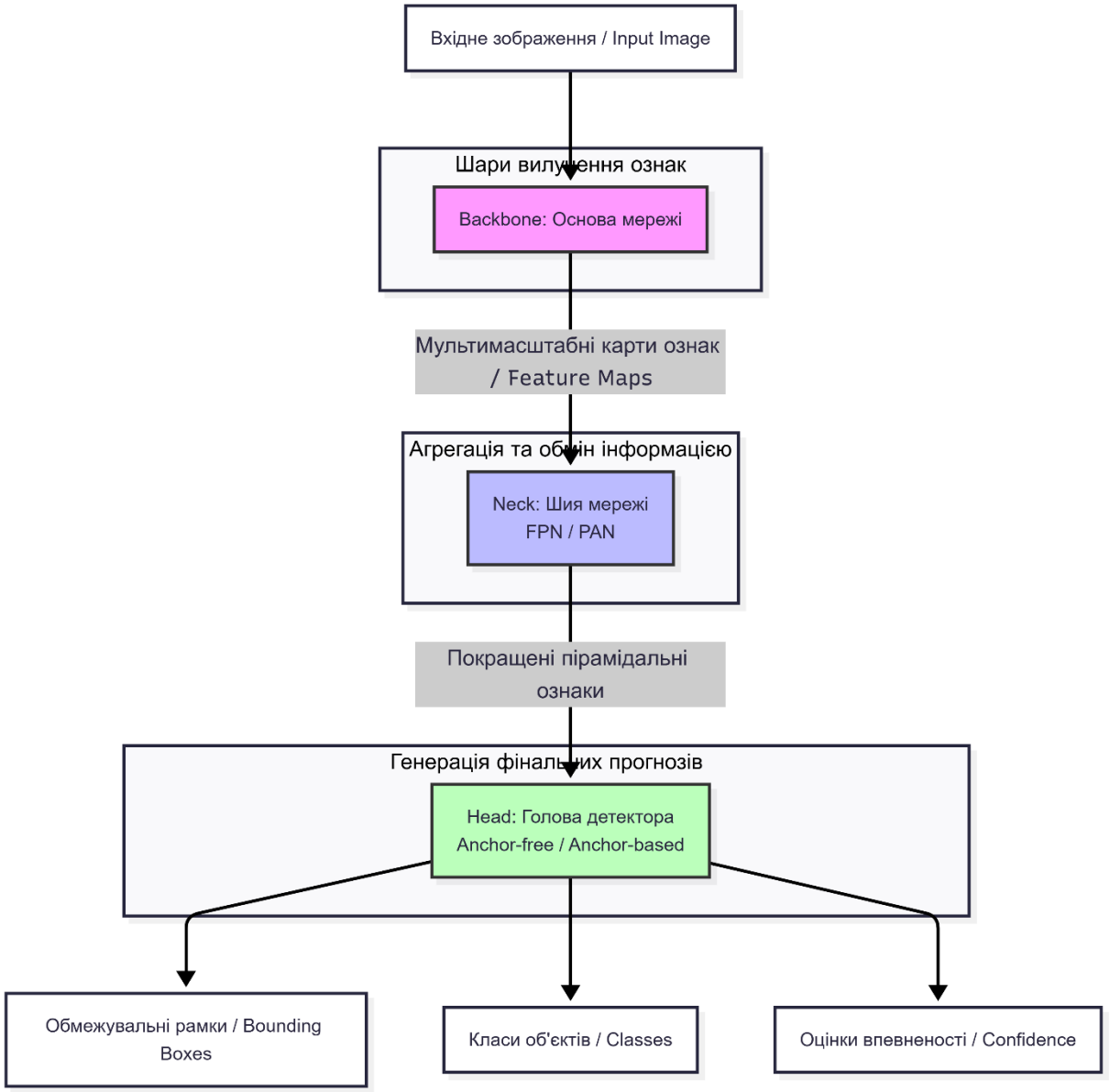

Рисунок 1.1 - Структура типової CNN

2. Однопрохідні (one-stage) детектори: Сюди відносяться SSD та сімейство YOLO. Вони об'єднують процеси локалізації та класифікації в єдину задачу регресії, виконуючи детекцію за один прямий прохід кадру через мережу. Це забезпечує колосальний приріст швидкості обробки зображень

(FPS), роблячи їх ідеальним вибором для систем асистивного моніторингу реального часу.

1.3 Порівняльний аналіз архітектур CNN-детекторів

Для обґрунтування вибору оптимальної нейромережевої архітектури, здатної ефективно функціонувати в мобільній системі виявлення перешкод для незрячих у приміщеннях, було здійснено глибокий аналіз ключових сучасних моделей.

Faster R-CNN - це класична двопрохідна модель [12]. Завдяки використанню потужного Backbone типу ResNet-101 та детальному аналізу регіонів через RPN, модель демонструє високі показники точності: на референсному датасеті MS COCO вона досягає метрики mAP = 42.7%. Проте зворотним боком високої точності є низька швидкість обробки – всього 5–7 кадрів на секунду (FPS) на дискретних графічних процесорах. Така затримка є неприпустимою для динамічної навігації незрячої людини, оскільки система не встигне вчасно зреагувати на раптову перешкоду. Крім того, модель містить близько 60 мільйонів параметрів, що виключає її запуск на мобільних пристроях без втрати автономності.

SSD (Single Shot MultiBox Detector). Цей однопрохідний детектор використовує детекцію на декількох картах ознак різного розширення всередині мережі, що дозволяє краще оперувати об'єктами різних габаритів [13]. Модифікація SSD300 показує високу швидкість обробки на рівні 59 FPS на десктопних GPU, досягаючи mAP = 74.3% на старішому наборі даних PASCAL VOC. Проте на складнішому датасеті COCO точність SSD300 падає до 25.1%.

Для перенесення на портативні платформи було створено полегшену версію MobileNet SSD, де стандартні згортки замінені на глибинно-роздільні (depthwise separable convolutions). Кількість параметрів знизилась до 6 мільйонів, але метрика точності на COCO впала до критичних 19.8%, що веде

до великої кількості помилкових або пропущених перешкод в умовах реальних кімнат.

Сімейство YOLO (You Only Look Once) [3]. Даний підхід здійснив революцію в однопрохідній детекції, розділивши вхідне зображення на сітку клітин. Кожна клітина відповідає за прогнозування фіксованої кількості обмежувальних рамок та ймовірностей класів, якщо центр об'єкта потрапляє в її межі. Це дозволило оригінальній моделі працювати на швидкості 45 FPS.

Еволюція лінійки (YOLOv3, YOLOv4, YOLOv5) поступово покращувала архітектурні рішення, впроваджуючи нові методи аугментації даних (Mosaic, MixUp), оптимізовані функції втрат (CIoU, GIoU) та покращені зв'язки між шарами.

YOLOv8 – це найсучасніша та найбільш технологічно досконала ітерація сімейства, представлена компанією Ultralytics у 2023 році [4, 5]. Вона увібрала в себе найкращі практики комп'ютерного зору: CSPNet Backbone: Використання концепції Cross-Stage Partial networks оптимізує градієнтні потоки в мережі, зменшує дублювання інформації в шарах, що підвищує якість вилучення ознак при меншій кількості математичних операцій. FPN+PAN Neck: Забезпечує покращене двонаправлене нагнітання ознак зверху вниз і знизу вгору, максимізуючи точність передачі просторової інформації. Anchor-free Head (Безякірна голова): На відміну від попередніх версій, які спиралися на заздалегідь прораховані шаблони рамок (anchor boxes), YOLOv8 прогнозує безпосередньо центр об'єкта та відстані до його меж. Це суттєво зменшує кількість операцій нерухомого придушення (NMS – Non-Maximum Suppression) та підвищує загальну точність детекції складних контурів об'єктів. Уніфіковані результати порівняльного аналізу розглянутих архітектур зведені в таблиці 1.1.

Аналіз структурованих даних таблиці 1.1 однозначно доводить, що модель YOLOv8n (найлегша, "nano" версія в лінійці YOLOv8) забезпечує безальтернативно найкращий баланс між точністю детекції та швидкістю її виконання серед усіх доступних легких моделей. Якщо порівнювати її з

MobileNet SSD, яка активно використовується у багатьох поточних асистивних розробках, то YOLOv8n демонструє перевагу за точністю на 17.5% mAP [7]. При цьому вона має надзвичайно низьку кількість параметрів (всього 3.2 млн) та гарантує стабільну обробку відеопотоку у реальному часі на мобільних чіпсетах зі швидкістю понад 80 FPS.

Таблиця 1.1 – Порівняльний аналіз CNN-детекторів об'єктів

Архітектура	Тип детектора	mAP COCO (50-95)	Швидкість FPS (GPU)	Кількість параметрів	Цільова апаратна платформа
Faster R-CNN	2-прохідний	42.7%	5–7	~60M	Високопродуктивний сервер
SSD300	1-прохідний	25.1%	59	~26M	Настільний ПК / Мобільна
MobileNet SSD	1-прохідний	19.8%	59	~6M	Портативні мобільні системи
YOLOv5n	1-прохідний	28.0%	100+	~2M	Мобільні пристрої / ПК
YOLOv8n	1-прохідний	37.3%	80+	~3.2M	Мобільні пристрої / Ембед-системи

Важливою практичною перевагою архітектури YOLOv8 є підтримка концепції transfer learning (перенесення навчання) на базі ваг, які були попередньо навчені розробниками на гігантському датасеті MS COCO. Цей набір даних містить розмітку для 80 базових повсякденних класів. Аналіз показав, що три класи з цього списку, а саме: person (людина), chair (стілець/крісло) та dining table (обідній стіл), повністю збігаються з цільовими класами перешкод, які найчастіше зустрічаються в приміщеннях. Це дозволяє використовувати базові знання мережі без тривалого початкового навчання з нуля.

Додатково, екосистема YOLOv8 надає розробнику високорівневий уніфікований Python-пакет та інтерфейс командного рядка (CLI), що значно оптимізує процеси проектування, тестування та фінального розгортання програмного забезпечення. Можливість швидкої конвертації базової моделі у

формати ONNX (Open Neural Network Exchange) та TensorRT відкриває прямий шлях до апаратної оптимізації нейромережі під мобільні процесори та мікрокомп'ютери (наприклад, Raspberry Pi або Jetson Nano), що є критично важливим фактором успіху для проектування компактних і автономних засобів навігації.

1.4 Обґрунтування програмно-апаратних обмежень, методів оптимізації та специфіки вхідних даних

Для успішної реалізації алгоритмічно-програмного рішення виявлення перешкод критично важливим є врахування специфіки технічних засобів отримання візуальних даних, архітектурних особливостей цільової апаратної платформи та програмного каркасу системи. Оскільки розробка орієнтована на персональне використання людьми з порушеннями зору, кінцевий пристрій повинен відповідати вимогам портативності, енергоефективності та низької собівартості. Це накладає жорсткі обмеження на вибір сенсорів та методів обробки інформації.

У сучасних системах комп'ютерного зору для визначення просторової геометрії перешкод використовуються два основні підходи до вибору сенсорів:

- активні датчики глибини та сенсорні системи (LiDAR, Time-of-Flight камери, стереокамери типу RGB-D). Вони дозволяють безпосередньо отримувати тривимірну хмару точок (point cloud) або матрицю відстаней до об'єктів у просторі. Попри високу точність локалізації, такі рішення характеризуються значним енергоспоживанням, апаратною громіздкістю, високою вартістю та обмеженою дальністю роботи в умовах інтенсивного природного освітлення (через завади інфрачервоного спектра).

- пасивні монокулярні RGB-камери. Представляють собою стандартні оптичні сенсори, інтегровані у більшість сучасних смартфонів та мобільних модулів.

Вибір на користь обробки стандартного плоского відеопотоку з однієї RGB-камери у даній роботі обґрунтовано прагненням мінімізувати масогабаритні показники асистивного пристрою та забезпечити його максимальну економічну доступність.

Проте використання монокулярного зору створює складну алгоритмічну задачу: класичні та нейромережеві детектори (зокрема, базові конфігурації сімейства YOLO) за замовчуванням визначають лише двовимірні координати обмежувальної рамки (Bounding\ Box) на площині кадру. Вони не здатні безпосередньо обчислювати реальну фізичну відстань до об'єкта в метрах.

Для подолання цього обмеження в межах програмного рішення необхідна інтеграція додаткових математичних моделей – геометрії проекції камери (калібрування фокусної відстані, оптичного центру та аналізу нижньої межі обмежувальної рамки відносно площини підлоги) або підключення полегшених глибинних нейромереж класу Monocular Depth Estimation.

Особливістю функціонування розроблюваного програмного забезпечення є робота в умовах закритих приміщень (Indoor navigation). Внутрішнє східно-кімнатне середовище характеризується високим рівнем динамічності: різкими перепадами освітлення від штучних джерел та вікон, наявністю дзеркальних і глянцевих поверхонь, що створюють бліки, а також великою щільністю різногабаритних об'єктів на обмеженій площі. Крім того, на відміну від відкритого простору (вулиць), де перешкоди часто є статичними або передбачуваними (стовпи, бордюри), у приміщенні критично важливо миттєво фіксувати малорозмірні або низькі об'єкти (дроти, сміттєві корзини, ніжки стільців), що вимагає від алгоритму високої роздільної здатності на рівні аналізу дрібних карт ознак (Feature\ Maps).

Оскільки запуск базових моделей глибокого навчання у форматі FP32 (32-бітна плаваюча кома) вимагає значних обчислювальних ресурсів графічних процесорів (GPU), безпосереднє розгортання моделей на Edge-пристроях (мобільних процесорах або одноплатних комп'ютерах типу Raspberry Pi чи

Jetson Nano) призводить до падіння швидкості обробки нижче критичного рівня (менше 10–15 FPS) та швидкого розряду акумулятора.

Для забезпечення працездатності алгоритму в режимі реального часу програмне рішення передбачає застосування технологій стиснення та оптимізації нейромереж [14]:

- квантування моделей (Model Quantization): процес переведення ваг та активацій нейронної мережі з високоточного формату FP32 у низькоточний 8-бітний цілочисельний формат INT8 або 16-бітний FP16. Математичне квантування дозволяє у 2–4 рази зменшити об'єм оперативної пам'яті, необхідної для збереження моделі, та критично прискорити execution-час на мобільних нейропроцесорах (NPU) або центральних процесорах (CPU) з підтримкою векторних інструкцій (наприклад, ARM Neon), мінімізуючи втрату точності детекції в межах 1–2%.

- конвертація у проміжні формати та інференс-рушії: для усунення накладних витрат важких фреймворків навчання (PyTorch) розроблене рішення спирається на кросплатформений формат ONNX (Open Neural Network Exchange) та спеціалізоване середовище виконання ONNX Runtime або TensorRT (для платформ Nvidia). Це дозволяє оптимізувати граф обчислень нейромережі (злиття шарів згортки та активації, видалення неактивних вузлів) безпосередньо під конкретну архітектуру цільового процесора.

Програмний каркас інтелектуального рішення базується на модульній архітектурі, де взаємодія між компонентами реалізується за допомогою високоефективних бібліотек з відкритим вихідним кодом. Загальна логіка роботи програмного забезпечення наведена на рисунку 1.2.

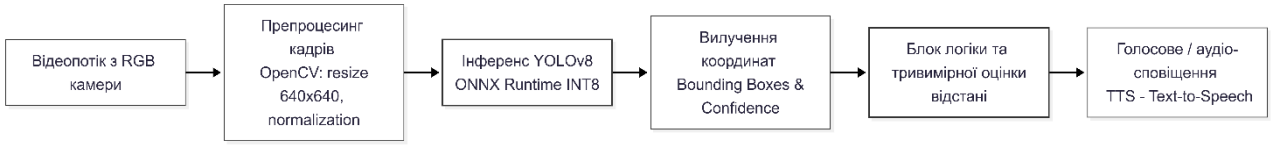

Рисунок 1.2 – Структурно-логічна схема програмного рішення виявлення перешкод

Для первинного захоплення відеопотоку, зміни роздільної здатності вхідних кадрів під архітектурний розмір нейромережі (640 пікселів) та колірної нормалізації тензорів використовується бібліотека OpenCV.

Отриманий стандартизований масив даних передається в оптимізований рушій інференсу, який повертає класи об'єктів та нормалізовані координати меж перешкод. На основі цих даних програмний блок логіки розраховує сектор небезпеки (центр, ліворуч, праворуч) та приблизне віддалення, після чого через модулі синтезу мовлення (TTS – Text-to-Speech) формує фінальний звуковий сигнал для користувача. Такий комплексний підхід до побудови архітектури гарантує стабільне, швидке та автономне функціонування системи у межах концепції Edge Computing.

2 РОЗРОБКА АЛГОРИТМУ ВИЯВЛЕННЯ ПЕРЕШКОД

2.1 Постановка задачі, вимоги та обмеження

2.1.1 Аналіз проблеми пересування незрячих у приміщеннях. Безпечне самостійне пересування у приміщеннях є однією з ключових потреб осіб з порушенням зору. За даними Всесвітньої організації охорони здоров'я (ВООЗ), понад 2,2 мільярди людей у світі мають порушення зору різного ступеня, з яких щонайменше 43 мільйони є повністю сліпими. Переміщення у знайомих і незнайомих приміщеннях є щоденним викликом для цієї групи населення.

Традиційні засоби орієнтації – біла тростина та собака-поводир – мають суттєві обмеження. Тростина дозволяє виявляти лише перешкоди на рівні підлоги в безпосередній близькості, не надає інформації про тип та розташування об'єкта. Собаки-поводирі ефективні, але дорогі у підготовці та утриманні; крім того, вони не завжди допускаються до громадських закладів. Ультразвукові датчики – наступний технологічний рівень – визначають відстань до перешкоди, але не ідентифікують її тип, що не дозволяє формувати якісні навігаційні підказки [6, 19].

Методи комп'ютерного зору та глибокого навчання відкривають принципово нові можливості: камера смартфону або нагрудного пристрою здатна забезпечити безперервний потік зображень, а CNN-детектор – семантично ідентифікувати об'єкти в реальному часі та генерувати конкретні голосові підказки типу «стілець праворуч» або «двері попереду».

2.1.2 Середовище застосування та вимоги до системи. Система призначена для використання виключно у закритих приміщеннях: житлових будинках, лікарнях, навчальних закладах, офісах, торгових центрах та інших громадських будівлях. Обрання приміщення як основного середовища обумовлено такими чинниками.

По-перше, саме у приміщеннях незряча людина проводить переважну частину часу та зустрічає найбільш різноманітні і потенційно небезпечні перешкоди. На відміну від вулиці, де основну небезпеку становлять транспортні засоби та нерівності дороги, у приміщенні перешкоди є більш передбачуваними та типізованими.

По-друге, умови освітлення у приміщеннях є більш контрольованими та стабільними порівняно з вуличними умовами (сонячне засліплення, різкі тіні, погодні умови). Це спрощує задачу детекції та підвищує стабільність роботи CNN-моделі.

По-третє, набір типових перешкод у приміщенні є відносно фіксованим і добре визначеним, що дозволяє сформувати обмежену цільову множину класів для навчання та оцінки моделі.

Вимоги до системи сформульовані наступним чином:

1) швидкість обробки відеопотоку – не менше 8 FPS на CPU (забезпечує оновлення інформації кожні 125 мс, що є достатнім для темпу ходьби людини ~1 м/с);

2) точність виявлення цільових класів – mAP50 ≥ 0.65 після донавчання;

3) час затримки від появи перешкоди в кадрі до голосового сповіщення – не більше 500 мс;

4) мінімальна відстань виявлення – об'єкт висотою 0.5 м на відстані 1.5 м (займає ≥ 5% висоти кадру);

5) робота в офлайн-режимі (без з'єднання з Інтернетом) для TTS-синтезу.

2.1.3 Формальна постановка задачі. Нехай $F = \{f_1, f_2, ..., f_n\}$ – послідовність кадрів відеопотоку, де кожен кадр f_t є матрицею розміру $H \times W \times 3$ (висота × ширина × канали RGB). Нехай C = {person, chair, dining table, door, stairs, trash can} – множина цільових класів перешкод. Для кожного кадру f_t система має:

1) виявити множину об'єктів $O_t = \{(b_i, c_i, s_i)\} \subseteq C$, де $b_i = (x1_i, y1_i, x2_i, y2_i)$ – обмежувальна рамка, $c_i \in C$ – клас, $s_i \in [0,1]$ – оцінка впевненості;

2) визначити підмножину перешкод $P_t \subseteq O_t$ – об'єктів, що знаходяться у центральній зоні кадру $[W \cdot 0.33; W \cdot 0.66]$;

3) для найбільш впевнено виявленої перешкоди $p^* = \text{argmax } s_i$ визначити напрямок обходу $d \in \{LEFT, RIGHT, BLOCKED\}$;

4) сформувати та відтворити голосове повідомлення $m = M(c^*, d)$, де M – функція відображення класу та напрямку у текст повідомлення.

2.1.4 Множина типових перешкод та їх характеристика. На основі аналізу умов пересування у приміщеннях визначено шість типів типових перешкод (табл.2.1).

Таблиця 2.1 – Характеристика типових перешкод у приміщеннях

Клас	Укр. назва	В COCO	Характеристика небезпеки
person	людина	✓	Динамічна перешкода. Включає дорослих та дітей. Дитина має менший bbox і нижчу оцінку впевненості моделі
chair	стілець	✓	Статична перешкода. Розташована на рівні ніг та тулуба. Небезпечна через виступаючі ніжки та спинку
dining table	стіл	✓	Велика горизонтальна поверхня. Може повністю перегороджувати прохід. Кути столу є джерелом травм
door	двері	✗	Динамічна/статична перешкода. Може бути зачиненою, відчиненою, напіввідчиненою. Потребує донавчання
stairs	сходи	✗	Перешкода підвищеної небезпеки. Потребує негайного попередження. Розрізняються сходи вгору та вниз
trash can	сміттєвий бак	✗	Компактна вертикальна перешкода. Часто розміщена в нестандартних місцях. Потребує донавчання

Практична цінність системи визначається двома основними напрямками застосування. По-перше, система є асистивним засобом для осіб з порушенням

зору: голосові підказки дозволяють незрячій людині отримувати актуальну інформацію про характер та розташування перешкод і самостійно коригувати маршрут без допомоги сторонніх осіб. Ключовою перевагою перед існуючими рішеннями є одночасна ідентифікація типу перешкоди та вказівка конкретного напрямку обходу. По-друге, розроблений алгоритм може застосовуватись у системах навігації мобільних роботів у приміщеннях, де CNN-детектор забезпечує семантичну інформацію для планувальника маршруту.

2.2 Вибір інструментів та підходу реалізації

2.2.1 Вибір мови програмування та середовища розробки. Вибір мови програмування базується на аналізі вимог: підтримка бібліотек глибокого навчання, ефективна робота з відеопотоком, крос-платформеність та простота розгортання. Порівняльний аналіз трьох основних кандидатів наведено у таблиці 2.2.

Таблиця 2.2 – Порівняльний аналіз мов програмування для реалізації системи

Критерій	Python 3.10	C++ (OpenCV)	Java (Android)
Підтримка PyTorch/YOLO	Нативна ✓✓	Обмежена ✓	Через JNI ±
Швидкість розробки	Висока ✓✓	Низька ✗	Середня ✓
OpenCV інтеграція	cv2 ✓✓	Нативна ✓✓	JavaCV ✓
TTS підтримка	pyttsx3/gTTS ✓✓	Стороннє ✗	Android TTS ✓
Крос-платформеність	Висока ✓✓	Середня ✓	Обмежена ±
Розгортання на RPi	Так ✓✓	Так ✓✓	Важко ✗

На основі порівняльного аналізу обрано Python 3.10+ як основну мову реалізації. Python є стандартом де-факто для задач машинного навчання та

комп'ютерного зору; забезпечує нативну підтримку бібліотеки ultralytics (YOLOv8); має просту та зрозумілу синтаксичну структуру, що полегшує супроводження та розширення коду.

2.2.2 Архітектура YOLOv8n та обґрунтування вибору. YOLOv8n (nano) є найлегшою версією восьмого покоління детектора YOLO, розробленого компанією Ultralytics у 2023 році [21]. Архітектура YOLOv8n складається з трьох основних компонентів.

Backbone (CSPNet-based). Використовується архітектура на основі Cross Stage Partial Networks (CSPNet), що забезпечує покращене вилучення ознак при зменшенні обчислювальних витрат. Backbone генерує ієрархічні карти ознак (feature maps) різних масштабів – від дрібнозернистих для виявлення малих об'єктів до великозернистих для великих об'єктів.

Neck (FPN+PAN). Feature Pyramid Network (FPN) та Path Aggregation Network (PAN) забезпечують двонаправлений обмін інформацією між картами ознак різного масштабу. FPN передає семантичну інформацію від глибоких шарів до поверхневих; PAN – локалізаційну інформацію від поверхневих до глибоких шарів. Це критично важливо для одночасного виявлення об'єктів різних розмірів (дитина та великий стіл) [4].

Head (anchor-free). На відміну від попередніх версій YOLO, YOLOv8 використовує безякірну (anchor-free) голову детектора. Це усуває необхідність попереднього визначення якорів (anchor boxes) та спрощує процес навчання. Голова генерує для кожної клітинки сітки обмежувальну рамку (4 координати), оцінку впевненості та вектор ймовірностей класів [4, 5].

Ключові характеристики YOLOv8n: кількість параметрів – 3.2 млн; розмір файлу ваг – 6.2 МБ; mAP50-95 на COCO – 37.3%; швидкість обробки – 80+ FPS на GPU T4, 8-15 FPS на CPU Intel Core i5. Вибір версії nano (n) серед п'яти розмірів (n/s/m/l/x) зумовлений необхідністю обробки у реальному часі на CPU без GPU, що є типовим для смартфону або одноплатного комп'ютера (Raspberry Pi).

2.2.3 Вибір допоміжних бібліотек. OpenCV (Open Source Computer Vision Library) версії 4.8+ обрано для захоплення та відображення відеопотоку. Бібліотека забезпечує уніфікований інтерфейс до різних джерел відеосигналу через клас VideoCapture, підтримує різні формати кодеків та платформи. Функції малювання (rectangle, putText) використовуються для відображення bbox та підписів на кадрі у режимі відлагодження.

Для TTS-синтезу реалізовано підтримку двох бекендів з автоматичним вибором: pyttsx3 (офлайн, рекомендовано для реального пристрою – не потребує з'єднання з Інтернетом, підтримує українські голоси системи) та gTTS – Google Text-to-Speech (онлайн, вища природність голосу, вимагає підключення). Перемикання між бекендами здійснюється через параметр конфігурації без зміни основного коду.

Бібліотека threading стандартної бібліотеки Python використовується для асинхронного виконання TTS у daemon-потоці, що забезпечує безперервну обробку відеопотоку незалежно від тривалості озвучення.

2.3 Алгоритм виявлення перешкод

Розроблений алгоритм виявлення перешкод реалізує безперервну циклічну обробку відеопотоку. Загальна блок-схема алгоритму представлена на рисунку 2.1.

Алгоритм включає шість послідовних етапів обробки кожного кадру, після чого повертається до захоплення наступного кадру. Обидві гілки після прийняття рішення про наявність/відсутність перешкоди замикаються в єдину петлю – система працює безперервно впродовж всього сеансу роботи.

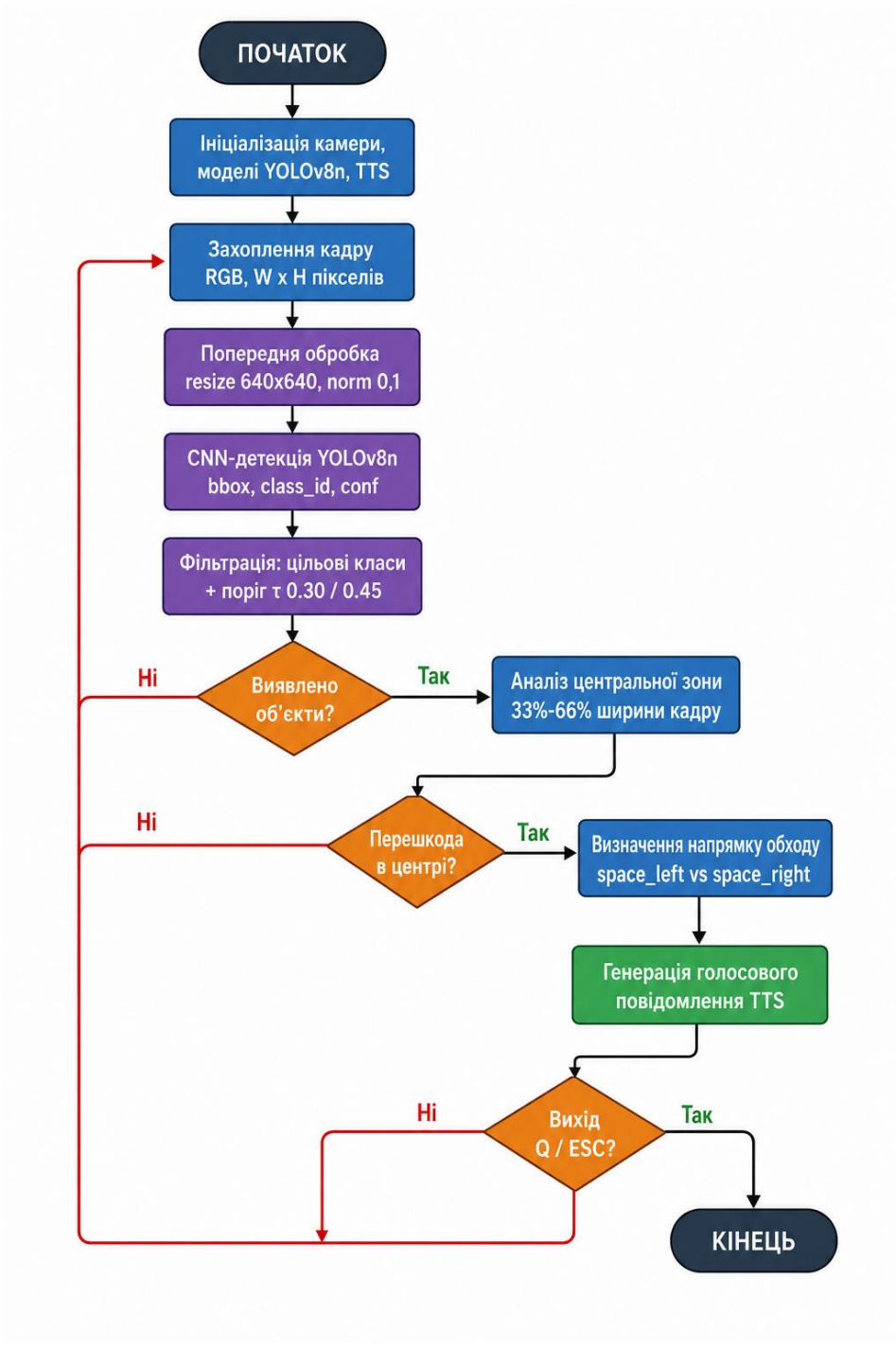

Рисунок 2.1 – Послідовність етапів алгоритму виявлення перешкод

Нижче наведено формальний псевдокод алгоритму виявлення перешкод.

Алгоритм DETECT_OBSTACLES(camera, model, analyzer, tts)
Вхід: camera – відеопристрій
model – YOLOv8n з навченими вагами
analyzer – ZoneAnalyzer(frame_width)

```
        tts   – TTSEngine(cooldown=2.5)
Вихід: голосові повідомлення (безперервно)
──────────────────────────────────────────────────────
1.  ІНІЦІАЛІЗАЦІЯ:
  2.    paused ← false
3.  ЦИКЛ поки not(EXIT):
  4.    frame ← camera.read()
  5.    ЯКЩО frame = NULL: продовжити цикл
  6.    ЯКЩО paused: обробляти клавіші; продовжити
─── ПОПЕРЕДНЯ ОБРОБКА ──────────────────────────────
  7.    frame_rgb ← BGR_to_RGB(frame)
─── CNN ДЕТЕКЦІЯ ───────────────────────────────────
  8.    raw_objects ← model.detect(frame)
─── ФІЛЬТРАЦІЯ ─────────────────────────────────────
  9.    objects ← []
  10.   ДЛЯ КОЖНОГО obj IN raw_objects:
    11.     τ ← CLASS_CONF.get(obj.class, 0.45)
    12.     ЯКЩО obj.class ∈ TARGET_CLASSES І obj.conf ≥ τ:
      13.       objects.append(obj)
─── АНАЛІЗ ЦЕНТРАЛЬНОЇ ЗОНИ ─────────────────────────
  14.   obstacles ← analyzer.filter(objects)
─── ГОЛОСОВЕ ПОВІДОМЛЕННЯ ──────────────────────────
  15.   ЯКЩО obstacles ≠ ∅:
    16.     top ← argmax(obj.conf for obj in obstacles)
    17.     name ← TRANSLATE[top.class]
    18.     tts.speak_obstacle(name, top.direction)
  19.   ІНАКШЕ:
    20.     tts.clear_state()
─── ВІДОБРАЖЕННЯ ───────────────────────────────────
  21.   draw_overlay(frame, objects, obstacles)
  22.   cv2.imshow('Детекція', frame)
─── ОБРОБКА КЛАВІШ ─────────────────────────────────
  23.   key ← cv2.waitKey(1)
  24.   ЯКЩО key = 'Q' АБО key = ESC: EXIT ← true
  25.   ЯКЩО key = 'P': paused ← not(paused)
26. КІНЕЦЬ ЦИКЛУ
27. camera.release(); cv2.destroyAllWindows()
```

Етап 1. Захоплення та валідація кадру.

На кожній ітерації циклу система отримує черговий кадр з відеопристрою через метод VideoCapture.read(). Якщо кадр не вдається зчитати (ret = False), ітерація пропускається з затримкою 50 мс без переривання роботи системи. Це забезпечує стійкість до тимчасових збоїв відеопристрою. Параметри відеопотоку (ширина W та висота H кадру в пікселях) зчитуються одноразово при ініціалізації та використовуються для налаштування ZoneAnalyzer.

Етап 2. Попередня обробка зображення.

Бібліотека ultralytics автоматично виконує попередню обробку перед CNN-детекцією: масштабування з letterboxing до 640×640 (зберігає пропорції, заповнює поля сірим кольором), перетворення BGR→RGB та нормалізацію значень пікселів до [0, 1]. Letterboxing є важливим – він запобігає деформуванню форм об'єктів, що могло б знизити точність детекції.

Етап 3. CNN-детекція об'єктів.

Модель YOLOv8n виконує один прохід через нейронну мережу (inference pass) і повертає список усіх виявлених об'єктів з усіх 80+3 класів. Кожен об'єкт описується: координатами bbox у форматі xyxy (x1, y1, x2, y2) у пікселях вихідного кадру; ідентифікатором класу (0–N); оцінкою впевненості confidence $\in$ [0, 1]. Параметр verbose=False вимикає виведення прогресу в консоль для збереження продуктивності.

Етап 4. Фільтрація за класом та порогом впевненості.

З результатів детекції відбираються лише об'єкти цільових класів із достатньою оцінкою впевненості.

Модель запускається з мінімальним порогом

$$conf = \min(CLASS\_CONF.values()) = 0.30,$$

а подальша фільтрація за індивідуальними порогами виконується програмно. Це дозволяє уникнути повторного запуску моделі при різних порогах для різних класів.

Поріг впевненості τ визначає мінімальну оцінку, при якій виявлений об'єкт вважається достовірним. Вибір значення τ є компромісом між двома типами помилок.

Завищений поріг ($\tau \to 1.0$) знижує кількість хибно позитивних виявлень (false positives), але збільшує кількість хибно негативних (false negatives) – реальні перешкоди пропускаються. Занижений поріг ($\tau \to 0.0$) виявляє всі реальні перешкоди, але генерує надмірну кількість хибних спрацьовувань.

Для більшості цільових класів встановлено $\tau = 0.45$. Це значення є стандартним рекомендованим порогом для YOLOv8 [21] і забезпечує прийнятний баланс між точністю (Precision) та повнотою (Recall) на тестових вибірках.

Знижений поріг для класу person ($\tau = 0.30$). Об'єкти класу person мають значну варіативність розміру bbox у кадрі: доросла людина на відстані 1 м займає ~40–60% висоти кадру (оцінка впевненості ≥ 0.70), тоді як дитина на тій же відстані займає лише ~20–30% висоти (оцінка впевненості 0.35–0.55). Застосування стандартного порогу $\tau = 0.45$ призводило б до систематичного пропуску дітей як перешкод, що є критично неприпустимим з точки зору безпеки. Зниження порогу до $\tau = 0.30$ дозволяє надійно виявляти дітей без суттєвого збільшення хибних спрацьовувань, оскільки клас person є одним з найкраще навчених у COCO (AP = 0.54 на test-dev).

Аналіз центральної зони є ключовим алгоритмічним рішенням, що відрізняє систему від простого детектора об'єктів. Ілюстрацію розподілу кадру на зони наведено на рисунку 2.2.

Визначення кадру ділиться на три вертикальні зони: ліву (0–33% ширини), центральну (33–66%) та праву (66–100%). Центральна зона відповідає напрямку руху користувача – при правильно закріпленій камері (на грудях або в руці) саме центр кадру відповідає напрямку погляду та руху.

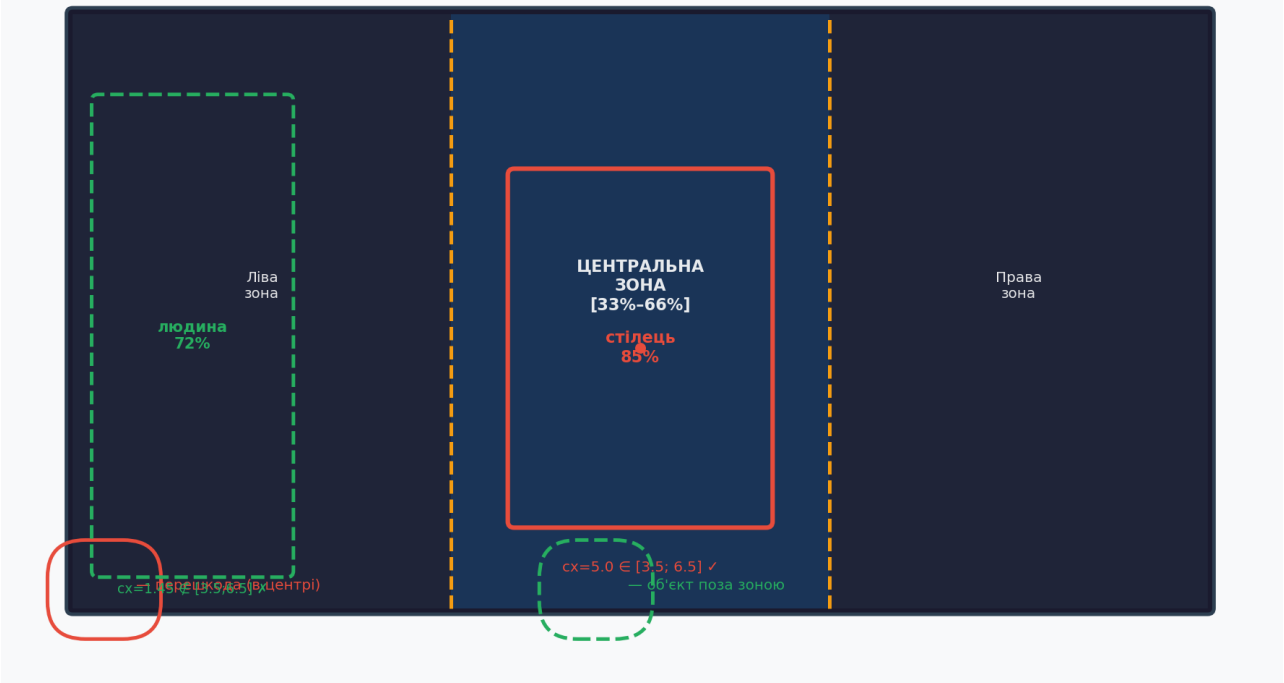

Рисунок 2.2 – Ілюстрація розподілу кадру на зони та приклади виявлення перешкод

Етапи алгоритму аналізу центральної зони представлено на рисунку 2.3.

Критерій 1 (центр bbox у зоні):

$$c_x = (x_1 + x_2) / 2; \quad \text{ПЕРЕШКОДА якщо } W_{0.33} \leq c_x \leq W_{0.66}$$

Критерій 2 (перекриття bbox із зоною):

$$\text{overlap}_w = \max(0, \min(x_2, W_{\cdot 0.66}) - \max(x_1, W_{\cdot 0.33}))$$

$$\text{ratio} = \text{overlap}_w / (x_2 - x_1); \quad \text{ПЕРЕШКОДА якщо ratio} \geq 0.40$$

Перший критерій (центр bbox) є достатнім для більшості випадків – об'єкт попереду матиме центр bbox у центральній зоні. Однак великі об'єкти (широкий стіл, людина впритул) можуть мати центр bbox поза центральною зоною, тоді як значна частина їх площі перекриває зону руху. Другий критерій (overlap) вирішує цю проблему.

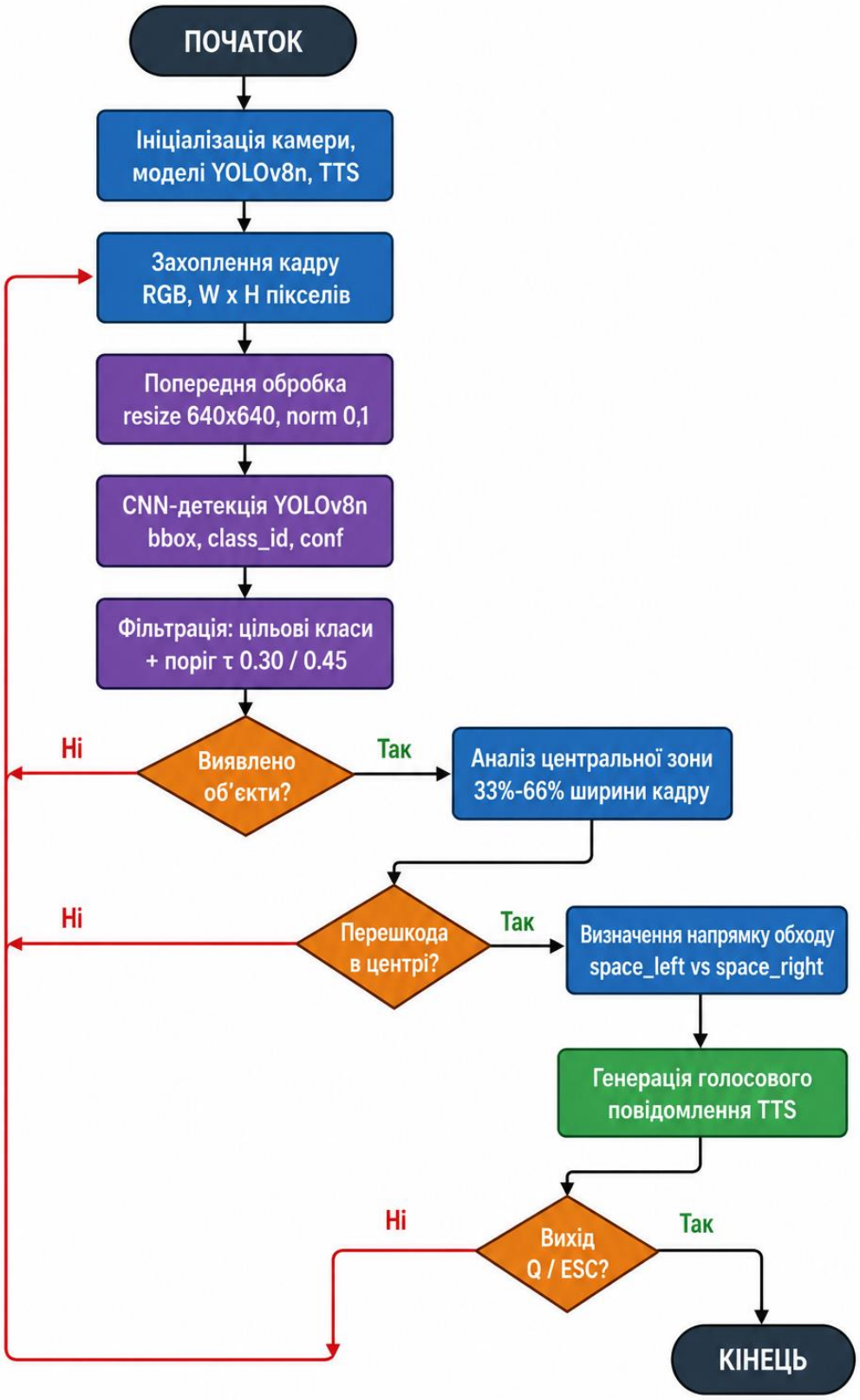

Рисунок 2.3 – Логіка алгоритму аналізу центральної зони та визначення перешкод

Обґрунтування меж зони (33%–66%). Центральна третина кадру обрана як компроміс між чутливістю (більша зона – більше виявлень) та специфічністю (менша зона – менше хибних спрацьовувань). Об'єкт у центральній третині кадру з високою ймовірністю перебуває безпосередньо на шляху руху. Зменшення зони до 40%–60% призвело б до пропуску об'єктів, розташованих трохи збоку але все ще небезпечних; збільшення до 25%–75% – до надмірної кількості хибних спрацьовувань від об'єктів збоку.

Обґрунтування порогу overlap_threshold = 0.40. Значення 40% означає: якщо більше ніж 40% ширини bbox перетинається з центральною зоною – об'єкт вважається перешкодою. Це значення встановлено емпірично: при overlap < 40% об'єкт переважно знаходиться збоку і не блокує прохід; при overlap ≥ 40% об'єкт суттєво вторгається в зону руху. Значення 40% є консервативним (краще помилково позначити боковий об'єкт перешкодою, ніж пропустити реальну перешкоду).

Алгоритм визначення напрямку обходу наведено на рисунку 2.4.

Для кожної виявленої перешкоди обчислюється вільний простір ліворуч та праворуч від bbox:

$\text{space\_left} = x1$ [пікселів від лівого краю кадру до лівої межі bbox]

$\text{space\_right} = W - x2$ [пікселів від правої межі bbox до правого краю кадру]

Обґрунтування порогу блокування block_threshold = 0.80. Якщо ширина bbox перевищує 80% ширини кадру – прохід вважається заблокованим незалежно від значень space_left та space_right, оскільки залишку простору з будь-якого боку (менше 10% ширини кадру) недостатньо для безпечного обходу. У цьому випадку генерується повідомлення «Прохід заблоковано – зупиніться», що є сигналом для зупинки та пошуку альтернативного маршруту.

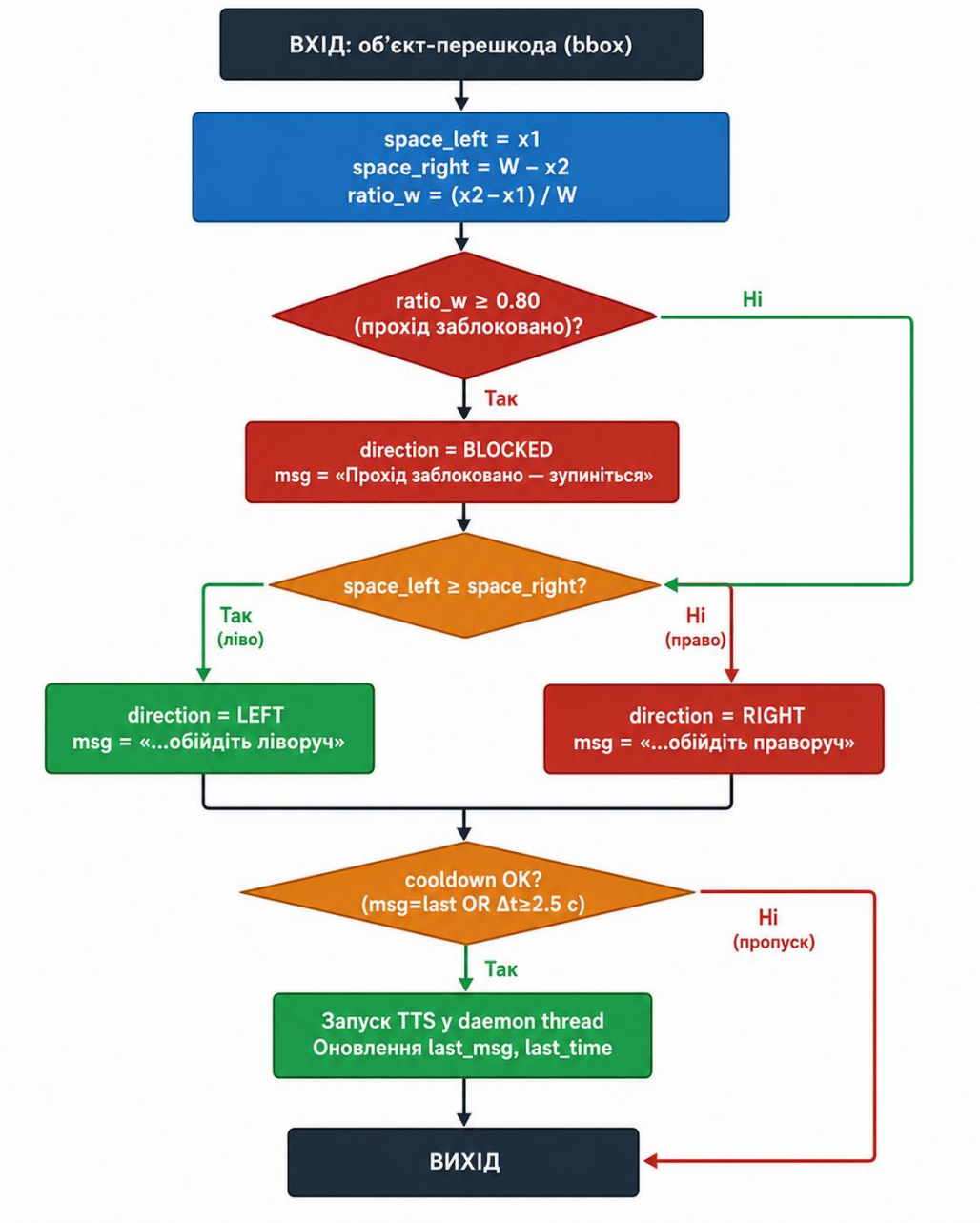

Рисунок 2.4 – Послідовність етапів визначення напрямку обходу та генерування голосового повідомлення

При наявності кількох одночасно виявлених перешкод алгоритм обирає об'єкт з найвищою оцінкою впевненості як пріоритетну перешкоду поточного кадру. Це обґрунтовано тим, що найбільш впевнено виявлений об'єкт, як правило, є найближчим та найбільш чітко видимим, тобто найбільш актуальним для навігаційного сповіщення.

Часова складність одного ітераційного кроку алгоритму визначається кількома компонентами. CNN-inference: O(1) відносно кількості об'єктів – час виконання YOLOv8n фіксований і не залежить від кількості виявлених об'єктів (≈ 50–130 мс на CPU). Фільтрація за класом та порогом: O(K), де K – кількість виявлених об'єктів (K ≤ 100 для типових сцен); у найгіршому випадку ≈ 0.01 мс. Аналіз центральної зони: O(N), де N ≤ K – кількість відфільтрованих цільових об'єктів; ≈ 0.01 мс. Відображення overlay: O(N) – малювання N bbox на кадрі; ≈ 0.5–2 мс.

Домінуючою складовою є CNN-inference, що становить понад 95% часу обробки кадру. Загальний час обробки одного кадру на CPU Intel Core i5 становить 65–130 мс, що відповідає частоті 8–15 FPS.

Просторова складність: O(K) для зберігання результатів детекції поточного кадру; O(1) для зберігання стану TTS (last_message, last_time). Система не накопичує дані між кадрами, що забезпечує стабільне споживання пам'яті протягом всього сеансу роботи.

2.4 Алгоритм генерування голосових повідомлень

Голосові повідомлення є єдиним каналом комунікації системи з користувачем. Тому їх зміст, форма та своєчасність є критично важливими для ефективності системи. Повідомлення мають бути: лаконічними (відтворюватись не більше 2 секунд); інформативними (містити назву перешкоди та напрямок дії); природньомовними (звучати природно українською мовою); без надмірного повторення.

Формат повідомлень:

– при виявленні перешкоди з обходом ліворуч: «Перешкода попереду: {клас}. Обійдіть ліворуч»;

– при виявленні перешкоди з обходом праворуч: «Перешкода попереду: {клас}. Обійдіть праворуч»;

– при заблокованому проході: «Прохід заблоковано – зупиніться».

Назви класів перекладаються українською відповідно до словника відображення: person→«людина», chair→«стілець», dining table→«стіл», door→«двері», stairs→«сходи», trash can→«сміттєвий бак». Переклад здійснюється у методі to_ukrainian() класу ObstacleDetector.

Без механізму обмеження частоти повідомлень система генерувала б нове повідомлення для кожного кадру (8–15 разів на секунду), що призводило б до неприйнятного акустичного шуму та унеможливлювало б сприйняття інформації. Механізм cooldown вирішує цю проблему.

Нове повідомлення генерується лише якщо виконується хоча б одна умова: (1) повідомлення відрізняється від попереднього (нова перешкода або новий напрямок); (2) від часу останнього повідомлення минуло не менше cooldown_seconds = 2.5 секунди.

Обґрунтування значення cooldown = 2.5 с. Значення 2.5 секунди визначено як суму двох складових: середній час озвучення найдовшого можливого повідомлення («Перешкода попереду: сміттєвий бак. Обійдіть ліворуч» – приблизно 3.5–4 слів/с × 7 слів ≈ 1.75–2 с) та час, необхідний для реакції та корекції напрямку руху (≈ 0.5–0.75 с). Таким чином, 2.5 с гарантує, що нове повідомлення не переривається попереднім та користувач встигає виконати дію до отримання наступної підказки.

При зникненні перешкоди з центральної зони (наприклад, людина відступила вбік) виконується виклик tts.speak_clear(), що скидає last_message. Це дозволяє миттєво повідомити про нову перешкоду, що з'явилась після звільнення проходу, без очікування закінчення cooldown.

Синтез та відтворення голосового повідомлення є відносно повільними операціями (1.5–2.5 с). Якщо виконувати їх у головному потоці, обробка відеопотоку буде заблокована на цей час, що призведе до суттєвих затримок і пропуску кадрів. Для вирішення цієї проблеми TTS виконується в окремому daemon-потоці.

Daemon-потік є фоновим потоком, що автоматично завершується при закритті головної програми. Це усуває необхідність явного управління життєвим циклом потоку. Взаємне виключення (mutex lock) використовується для захисту стану last_message та last_time від гонки даних при одночасному доступі з головного потоку та TTS-потоку.

2.5 Стратегія transfer learning та донавчання моделі

Три цільові класи (door, stairs, trash can) відсутні у стандартному наборі класів COCO, тому pretrained YOLOv8n не здатна їх виявляти (mAP50 = 0.00 на тестовій вибірці). Для вирішення цієї проблеми розглядалися дві стратегії навчання.

Стратегія 1 – навчання з нуля (training from scratch): ініціалізація ваг випадковим чином; навчання на повному датасеті з усіма 6 класами. Недоліки: потребує значно більшого датасету (десятки тисяч зображень для кожного класу); час навчання 20–50 годин навіть на GPU; ризик перенавчання при малому датасеті.

Стратегія 2 – fine-tuning (тонке налаштування) [9]: використання попередньо навчених ваг YOLOv8n (COCO) як початкової точки; донавчання лише на нових класах при заморожуванні частини backbone. Переваги: збереження знань про 3 наявних класи (person, chair, table); можливість досягнення прийнятної точності при малому датасеті; час навчання 1.5–2 години на GPU T4.

Обрано стратегію fine-tuning як більш практичну та ефективну для задачі з обмеженим датасетом нових класів [24].

Датасет для донавчання сформовано з трьох відкритих наборів даних платформи Roboflow Universe у форматі YOLOv8 (таблиця 2.3).

Таблиця 2.3 – Характеристика датасетів для донавчання

Клас	Датасет	Train	Val	Test	Всього
door	doors (anood-saad)	2042	589	291	2922
stairs	stair-detection (siavash-mahmoudi)	104	17	3	124
trash can	trash-detection (Roboflow)	184	31	16	231
РАЗОМ	–	2330	637	310	3277

Слід зазначити, що датасет stairs є відносно малим (124 зображення), що є типовим обмеженням для специфічних класів об'єктів. Для компенсації малого обсягу застосовано розширену аугментацію даних. Датасет doors є найбільшим (2922 зображення) і містить три підкласи: зачинені, відчинені та напіввідчинені двері, що відповідає реальній варіативності об'єкту.

Формат анотацій – YOLOv8 (YOLO darknet): кожне зображення супроводжується текстовим файлом .txt з рядками формату «class_id cx cy w h», де всі координати нормалізовані відносно розміру зображення. Клас door відповідає id=3, stairs – id=4, trash can – id=5 у відповідності до загального словника класів системи.

Параметри навчання (табл. 2.4) підібрані з урахуванням обсягу датасету, архітектури моделі та обчислювальних ресурсів.

Обґрунтування заморожування шарів (freeze=10). Backbone YOLOv8n складається з 10 основних блоків Conv, C2f, SPPF. Перші шари кодують низькорівневі ознаки (краї, кути, текстури), що є універсальними для будь-яких об'єктів і не потребують перенавчання. Верхні шари (head та частина neck) кодують семантичні ознаки конкретних класів і є цільовими для fine-tuning. Заморожування 10 шарів дозволяє зосередити навчання на верхніх рівнях ієрархії ознак та зберегти здатність моделі виявляти класи COCO без деградації.

Таблиця 2.4 – Параметри fine-tuning YOLOv8n

Параметр	Значення	Обґрунтування
epochs	80	Достатньо для збіжності при малому датасеті; early stopping запобігає перенавчанню
batch_size	16	Оптимальний розмір для GPU T4 (16 ГБ VRAM); більший batch → стабільніший градієнт
imgsz	640	Стандартний розмір YOLOv8; забезпечує детекцію об'єктів різних масштабів
lr0	0.001	Знижена початкова швидкість навчання для fine-tuning (стандарт 0.01); запобігає руйнуванню ваг COCO
freeze	10	Заморожування перших 10 шарів backbone; зберігає низькорівневі ознаки COCO
patience	20	Early stopping: зупинка якщо 20 епох без покращення mAP50 на val
device	GPU (T4)	Google Colab GPU; скорочення часу з ~26 год (CPU) до ~1.5 год

Аугментація даних є критично важливою при відносно малому обсязі датасету (особливо для класу stairs з 124 зображеннями). Застосовано такі методи аугментації (таблиця 2.6).

Таблиця 2.5 – Методи аугментації даних та їх обґрунтування

Параметр	Значення	Фізичний зміст та обґрунтування
hsv_v	0.4	Варіація яскравості ±40%. Моделює різне освітлення приміщень: яскравий денний, тьмяний вечірній, флуоресцентний
hsv_s	0.5	Варіація насиченості кольору ±50%. Компенсує різницю між зображеннями з різних камер
fliplr	0.5	Горизонтальне відзеркалення з ймовірністю 50%. Збільшує різноманітність орієнтацій об'єктів
mosaic	1.0	Мозаїчна аугментація (4 зображення в 1) з ймовірністю 100%. Критично важлива для малих датасетів
degrees	5.0°	Поворот ±5°. Моделює нахил камери при ходьбі користувача
translate	0.1	Зміщення до ±10% розміру. Збільшує варіативність положення об'єктів у кадрі

Особливо важливою є мозаїчна аугментація (mosaic=1.0): вона об'єднує 4 зображення в одне, що ефективно збільшує різноманітність контекстів та масштабів об'єктів. Для класу stairs з лише 104 навчальними зображеннями mosaic фактично «генерує» нові комбінації сцен, що суттєво знижує ризик перенавчання.

У другому розділі розроблено повний комплекс алгоритмічних рішень для виявлення типових перешкод у приміщеннях та генерування голосових навігаційних підказок.

Сформульовано формальну постановку задачі: вхідними даними є RGB-кадри відеопотоку H×W×3, вихідними – голосові повідомлення з назвою перешкоди та напрямком обходу. Визначено множину з шести типових перешкод для приміщень та обґрунтовано практичну цінність системи для двох цільових груп: осіб з порушенням зору та мобільних роботів у приміщеннях.

Обґрунтовано вибір інструментів реалізації: Python 3.10+ (стандарт ML-розробки), YOLOv8n (найкращий баланс точності та швидкості серед легких CNN-детекторів, 3.2M параметрів, 37.3% mAP50-95 на COCO), OpenCV (відеозахоплення), pyttsx3/gTTS (TTS-синтез).

Розроблено шестиетапний алгоритм виявлення перешкод: захоплення кадру → попередня обробка → CNN-детекція → фільтрація за класом/порогом → аналіз центральної зони → визначення напрямку обходу. Алгоритм утворює замкнений безперервний цикл. Обґрунтовано всі ключові параметри: пороги впевненості $\tau = 0.45$ (загальний) та $\tau = 0.30$ (для person – виявлення дітей), межі центральної зони 33%–66%, поріг перекриття overlap_threshold = 0.40, поріг блокування block_threshold = 0.80.

Розроблено алгоритм генерування голосових повідомлень з механізмом cooldown (cooldown = 2.5 с, обґрунтовано як суму часу озвучення ~1.75 с та часу реакції ~0.75 с) та асинхронним виконанням TTS у daemon-потоці для забезпечення безперервності обробки відеопотоку.

Розроблено стратегію fine-tuning YOLOv8n для донавчання на трьох нових класах (door, stairs, trash can): заморожування перших 10 шарів backbone (freeze=10) зберігає знання COCO; датасет 3 277 зображень з Roboflow Universe; аугментація з mosaic=1.0 компенсує малий обсяг датасету stairs (124 зразки). Всі параметри навчання (epochs=80, lr0=0.001, batch=16, patience=20) обґрунтовано.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Структура програмної системи

Програмна система виявлення перешкод реалізована у вигляді набору спеціалізованих модулів на мові Python 3.10. Кожен модуль несе чітко визначену відповідальність відповідно до принципу єдиної відповідальності (Single Responsibility Principle).

Структуру взаємодії модулів показано на рисунку 3.1.

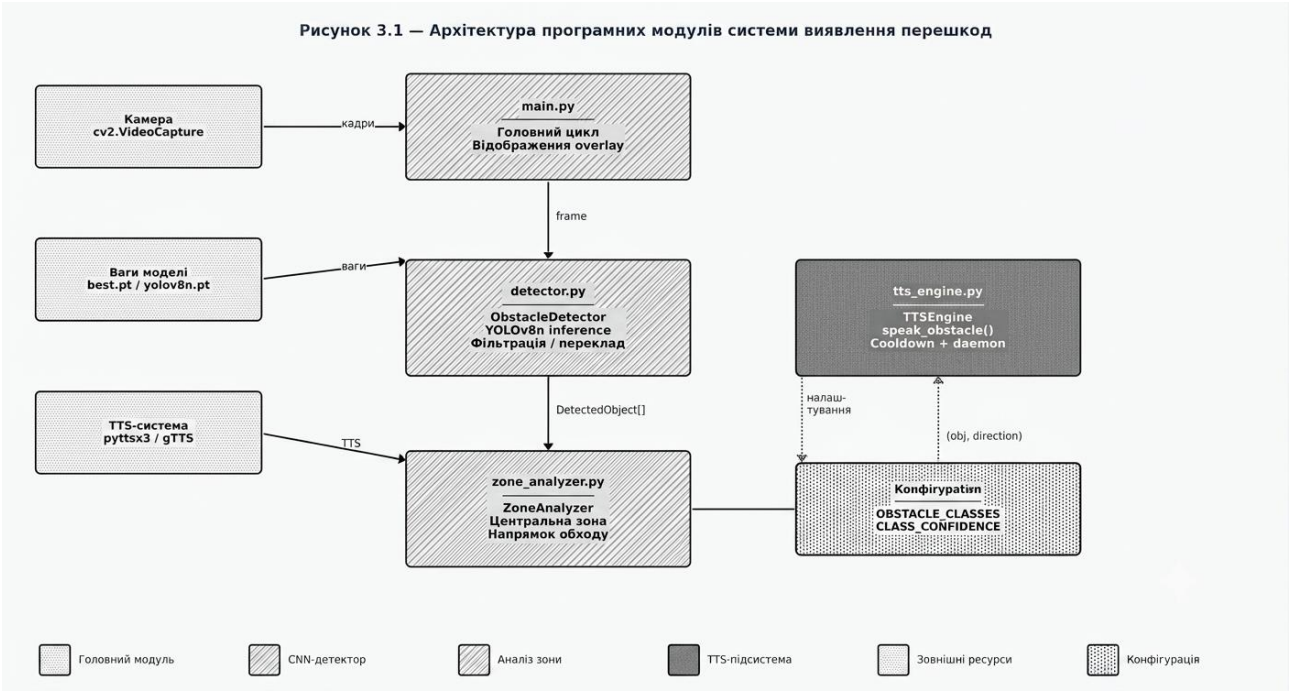

Рисунок 3.1 – Взаємодія програмних модулів системи виявлення перешкод)

Система складається з чотирьох основних модулів та трьох допоміжних. Основні модулі утворюють конвеєр обробки:

- main.py захоплює кадри та координує роботу решти компонентів; detector.py виконує CNN-детекцію та фільтрацію;

- zone_analyzer.py аналізує розташування об'єктів у просторі кадру;

- tts_engine.py відповідає за голосове сповіщення.

Допоміжні модулі (prepare_dataset.py, train.py, evaluate_base.py, compare_results.py) використовуються на етапі підготовки та навчання моделі і не входять до складу виконуваної системи.

Модуль zone_analyzer.py реалізує логіку просторового аналізу положення виявлених об'єктів відносно центральної зони кадру. Містить клас DetectedObject – dataclass для представлення одного виявленого об'єкта з полями: class_name (рядок назви класу), confidence (оцінка впевненості), x1, y1, x2, y2 (координати bbox). Властивості center_x та width обчислюються на основі координат bbox.

Клас ZoneAnalyzer ініціалізується шириною кадру frame_width та параметрами зони. Метод is_in_center_zone(obj) реалізує перевірку за двома критеріями: центр bbox у зоні або перекриття понад 40%. Метод bypass_direction(obj) повертає значення перелічення BypassDirection (LEFT, RIGHT, BLOCKED) на основі порівняння вільного простору ліворуч та праворуч. Метод filter_obstacles(objects) повертає відсортований за confidence список пар (DetectedObject, BypassDirection) для всіх перешкод у центральній зоні.

Модуль detector.py є обгорткою над бібліотекою ultralytics для запуску CNN-детекції. Клас ObstacleDetector ініціалізується шляхом до файлу ваг моделі (.pt), загальним порогом впевненості та словниками цільових класів. При ініціалізації завантажується модель YOLO з файлу ваг; якщо файл відсутній – завантажується автоматично з хмарного сховища Ultralytics.

Метод detect(frame) приймає BGR-зображення numpy-масиву та повертає список об'єктів DetectedObject. Модель запускається з мінімальним порогом серед усіх класів (0.30 для person); подальша фільтрація за індивідуальними порогами виконується програмно. Параметр verbose=False вимикає виведення

статистики в консоль під час inference. Метод to_ukrainian(class_name) виконує переклад назви класу зі словника OBSTACLE_CLASSES.

Клас TTSEngine реалізує асинхронне голосове озвучення з механізмом cooldown. При ініціалізації автоматично обирається найкращий доступний бекенд: якщо встановлено pyttsx3 – використовується офлайн-синтез, інакше – gTTS через Інтернет; якщо жоден недоступний – виводиться текст у консоль (режим відлагодження).

Публічний метод speak_obstacle(name_uk, direction) формує текст повідомлення залежно від напрямку: для BLOCKED – «Прохід заблоковано – зупиніться», для LEFT/RIGHT – «Перешкода попереду: {name}. Обійдіть {direction}». Метод speak(message) перевіряє умови cooldown (потокобезпечно через threading.Lock) і запускає відтворення у daemon-потоці. Метод speak_clear() скидає стан при зникненні перешкоди.

Модуль main.py реалізує головний цикл обробки відеопотоку та інтегрує всі компоненти системи. При запуску ініціалізуються: ObstacleDetector з вказаними вагами, TTSEngine з обраним бекендом, cv2.VideoCapture для захоплення відео, ZoneAnalyzer з шириною кадру з камери. Функція draw_overlay() малює на кадрі: напівпрозору заливку центральної зони, bbox кожного виявленого об'єкта (червоний для перешкод у зоні, зелений для решти), підпис з назвою, confidence та підказкою напрямку (← ліво / → право / ⛔стоп), поточне значення FPS.

Програма підтримує управління через клавіатуру: Q або ESC – завершення роботи, P – пауза/продовження, S – збереження поточного кадру в PNG. Запуск здійснюється через CLI з параметрами: --camera (індекс камери), --model (шлях до .pt), --tts (бекенд), --no-window (режим без відображення для пристроїв без екрану).

3.2 Підготовка датасету для донавчання

3.2.1 Вибір та завантаження відкритих датасетів. Для донавчання моделі на трьох нових класах (door, stairs, trash can) використано відкриті датасети платформи Roboflow Universe – найбільшого сховища комп'ютерно-зорових датасетів у форматі YOLO. Платформа надає готову анотацію у форматі YOLOv8 (текстові файли .txt з нормалізованими координатами bbox), що усуває необхідність ручної розмітки.

Відбір датасетів здійснювався за такими критеріями: наявність зображень, що відповідають умовам приміщення (внутрішній простір, різне освітлення); достатня варіативність ракурсів та умов зйомки; наявність розмітки train/val/test або можливість автоматичного розподілу. Характеристику обраних датасетів наведено у таблиці 3.1.

Таблиця 3.1 – Характеристика датасетів для донавчання

Клас	Датасет (Roboflow)	Train	Val	Test	Всього
door	anood-saad / doors-6g8eb-r3peh	2 042	589	291	2 922
stairs	siavash-mahmoudi / stair-detection-sb2fk	104	17	3	124
trash can	roboflow-100 / trash-detection	184	31	16	231
РАЗОМ	**–**	**2 330**	**637**	**310**	**3 277**

Структуру датасету по класах і вибірках наведено на рисунку 3.2.

Як видно з рисунку 3.2а, найбільшу частку датасету (89.2%) складають зображення класу door, що пояснюється ширшою доступністю відкритих датасетів дверей.

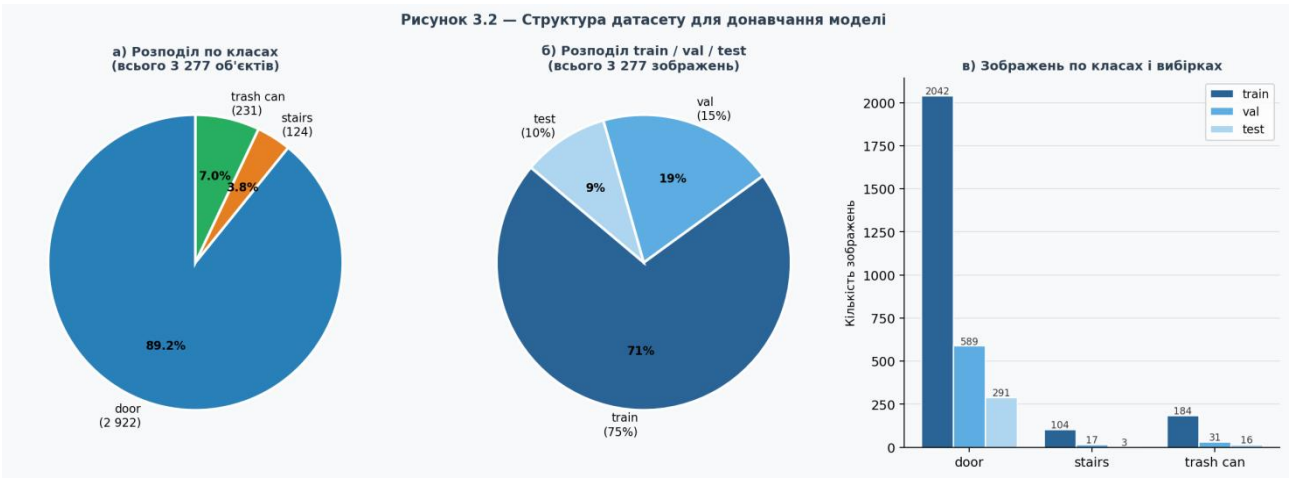

Рисунок 3.2 – Структура датасету для донавчання моделі

Клас stairs є найменшим (3.8% від загального обсягу, 124 зображення), що є потенційним обмеженням для точності виявлення сходів. Розподіл train/val/test (рисунок 3.2б) становить 75%/15%/10%, що є стандартним для задач комп'ютерного зору.

3.2.2 Конвертація та об'єднання датасетів. Завантажені датасети мають різні схеми іменування класів: датасет doors використовує числові назви ['0', '1', '2'] для трьох станів дверей (зачинені, відчинені, напіввідчинені); датасет stairs – ['DownStair', 'UpStair']; датасет trash_can – ['empty', 'full']. Для коректного об'єднання необхідна перенумерація class_id відповідно до єдиного словника системи.

Скрипт prepare_dataset.py виконує конвертацію у три кроки.

По-перше, зчитується data.yaml кожного датасету для визначення відповідності між class_id та назвою класу.

По-друге, кожен рядок файлу міток (.txt) перезаписується: старий class_id замінюється на новий (door→3, stairs→4, trash can→5) відповідно до словника class_map.

По-третє, файли зображень та конвертованих міток копіюються до уніфікованої структури dataset/ з унікальними іменами (префікс назви датасету запобігає колізіям між файлами різних датасетів). Усі стани дверей (0, 1, 2) та

обидва типи сходів (DownStair, UpStair), а також обидва стани баку (empty, full) маппуються на єдиний відповідний клас, оскільки для задачі виявлення перешкод важливий факт наявності об'єкта, а не його стан.

3.3 Навчання моделі

Навчання моделі в контексті даної роботи – це процес адаптації попередньо навченої нейронної мережі YOLOv8n до задачі виявлення специфічних класів перешкод у приміщеннях. Мета навчання полягає не в тому, щоб модель «вивчила» комп'ютерний зір з нуля, а в тому, щоб додати до її наявних знань здатність розпізнавати три нові класи (door, stairs, trash can), відсутні в базовому наборі COCO.

Глибинні шари мережі (backbone), що відповідають за розпізнавання загальних ознак (краї, форми, текстури), залишаються незмінними; навчаються лише верхні шари, відповідальні за семантичну класифікацію конкретних об'єктів. Такий підхід називається transfer learning (навчання з перенесенням знань) [9].

Практична цінність навчання полягає в наступному: до навчання модель не здатна виявляти двері, сходи та сміттєві баки (mAP50 = 0.000 для всіх трьох класів); після навчання – набуває цієї здатності та стає придатною для реального застосування у системі допомоги незрячим. При цьому здатність моделі виявляти базові класи COCO (person, chair, dining table) зберігається завдяки заморожуванню backbone.

Навчання проводилося на хмарній платформі Google Colaboratory з GPU NVIDIA Tesla T4 (14 ГБ VRAM). Використання GPU прискорило навчання приблизно в 18 разів порівняно з CPU: замість очікуваних ~26 годин на CPU навчання 43 епох зайняло 0.809 години (~49 хвилин) (рис3.3).

Процес навчання контролювався за такими метриками на валідаційній вибірці після кожної епохи: box_loss (втрата локалізації bbox), cls_loss (втрата

класифікації), dfl_loss (розподілена фокусна втрата), mAP50 та mAP50-95. Найкращі ваги автоматично зберігалися у файл best.pt при покращенні mAP50 на валідаційній вибірці.

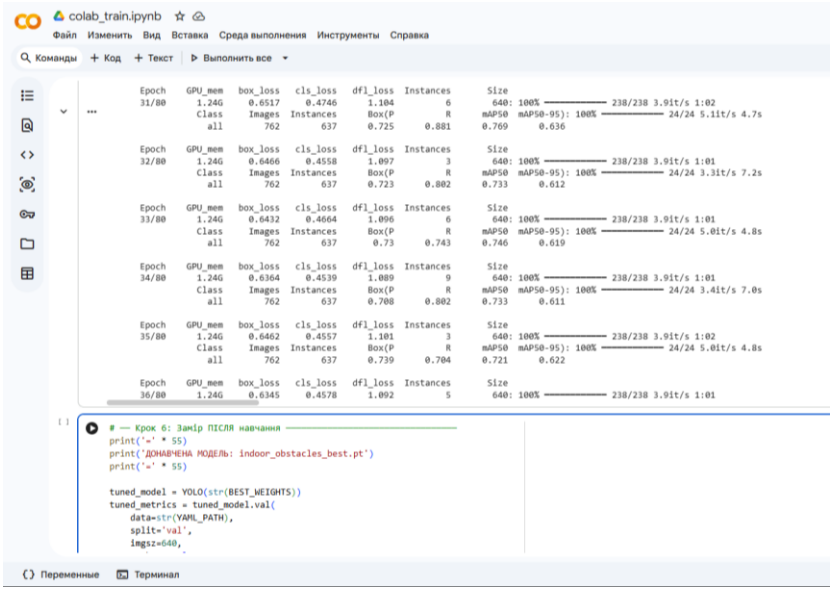

Рисунок 3.3 – Процес донавчання моделі

Механізм раннього зупинення (early stopping) спрацював на 43-й епосі: покращення mAP50 на валідаційній вибірці не спостерігалось протягом 20 послідовних епох. Найкращий результат зафіксовано на 23-й епосі з mAP50 = 0.793. Це свідчить про ефективне навчання без перенавчання: модель досягла оптимуму значно раніше запланованих 80 епох, що є позитивним результатом.

3.4 Суть та методика експерименту

Метою експериментальних досліджень є кількісна оцінка ефективності розробленого алгоритму виявлення перешкод та підтвердження доцільності застосування fine-tuning для розширення можливостей базової моделі. Експеримент структурований як порівняльне дослідження двох конфігурацій системи.

Конфігурація А (базова модель): стандартна YOLOv8n із вагами pretrained на COCO (80 класів); модель не піддавалась жодному донавчанню; оцінюється здатність виявляти всі 6 цільових класів перешкод.

Конфігурація Б (донавчена модель): YOLOv8n після fine-tuning на датасеті Roboflow (door, stairs, trash can); ті самі умови тестування що й для конфігурації А.

Обидві конфігурації оцінюються на одному тестовому наборі даних (val-вибірка датасету Roboflow, 762 зображення) за однаковими метриками, що забезпечує коректне порівняння.

Для оцінки якості детекції використано стандартні метрики, прийняті в задачах виявлення об'єктів.

Precision (точність) – частка правильних виявлень серед усіх виявлень моделі:

$$P = TP / (TP + FP),$$

де TP – кількість правильно виявлених об'єктів (IoU ≥ 0.5 з ground truth),
FP – хибні виявлення (неіснуючі об'єкти або неправильний клас).

Recall (повнота) – частка правильно виявлених об'єктів серед усіх реальних:

$$R = TP / (TP + FN),$$

де FN – пропущені об'єкти.

mAP50 (середня точність при IoU=0.5) – основна метрика порівняння. Обчислюється як середнє значення Average Precision (AP) по всіх класах при порозі перетину IoU = 0.5. AP для кожного класу – площа під кривою Precision-Recall. Значення mAP50 ∈ [0, 1]; чим вище – тим краще.

mAP50-95 – більш строга метрика: середнє mAP при порогах IoU від 0.5 до 0.95 з кроком 0.05. Відображає якість локалізації bbox.

Оцінка виконувалась скриптом evaluate_base.py, що запускає model.val() бібліотеки ultralytics на val-вибірці датасету. Параметри оцінки: split='val', imgsz=640, пристрій GPU Tesla T4. Для базової моделі (yolov8n.pt) додатково

перевірялась здатність виявляти класи COCO, що відповідають цільовим класам системи. Обидва заміри виконані за однакових умов, що забезпечує коректне порівняння результатів.

3.5 Результати експериментальних досліджень

Результати оцінки базової моделі YOLOv8n (pretrained COCO) на val-вибірці датасету наведено у таблиці 3.2. Оскільки val-вибірка містить лише зображення нових класів (door, stairs, trash can), результати для класів COCO (person, chair, dining table) отримані з технічної документації YOLOv8 [21].

Таблиця 3.2 – Результати оцінки базової моделі YOLOv8n (до донавчання)

Клас	Джерело	mAP50	Precision	Recall	Статус
person	COCO pretrained	0.5400	0.75	0.63	✓ виявляє
chair	COCO pretrained	0.5100	0.70	0.60	✓ виявляє
dining table	COCO pretrained	0.4800	0.68	0.57	✓ виявляє
door	val-вибірка	0.0000	–	–	✗ не виявляє
stairs	val-вибірка	0.0000	–	–	✗ не виявляє
trash can	val-вибірка	0.0000	–	–	✗ не виявляє
Загальне	val-вибірка	0.1409	0.2079	0.0968	–

Як видно з таблиці 3.2, базова модель YOLOv8n повністю нездатна виявляти три нові класи перешкод: mAP50 = 0.000 для door, stairs та trash can. Це підтверджує необхідність донавчання. Низьке загальне значення mAP50 = 0.1409 на нашій val-вибірці обумовлено тим, що val-вибірка містить лише зображення нових класів, які базова модель не розпізнає.

Процес навчання завершився достроково на 43-й епосі з 80 запланованих завдяки механізму early stopping. Найкращий результат досягнуто на 23-й епосі. Детальні результати оцінки донавченої моделі (best.pt) на val-вибірці наведено у таблиці 3.3.

Таблиця 3.3 – Результати оцінки донавченої моделі YOLOv8n (після fine-tuning)

Клас	Зображень	Об'єктів	Precision	Recall	mAP50
door	579	589	0.963	0.986	0.993
stairs	16	17	0.459	0.412	0.390
trash can	31	31	0.865	1.000	0.995
Загальне	762	637	0.762	0.799	0.793

Графіки навчання (криві втрат, mAP50, Precision, Recall по епохах) зображено на рисунку 3.4

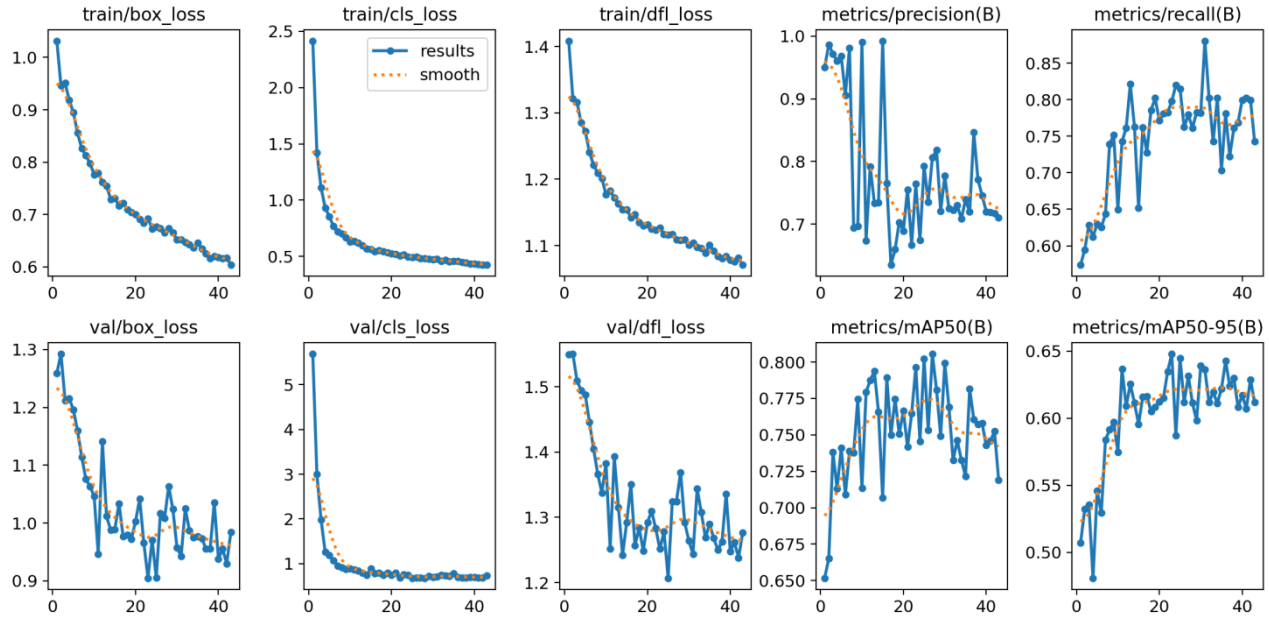

Рисунок 3.4 – Метрики якості моделі під час навчання

Раннє зупинення на 43-й епосі з 80 запланованих свідчить про ефективне навчання без ознак перенавчання. Збіжність протягом 43 епох є типовою для fine-tuning на невеликому датасеті при замороженому backbone: модель швидко

адаптує верхні шари до нових класів, після чого подальше навчання не дає суттєвого покращення на валідаційній вибірці.

Стабільна крива mAP50 на val-вибірці (рисунок 3.4) без суттєвих коливань після 23-ї епохи вказує на відсутність перенавчання. Якби модель перенавчалась, спостерігалось би зростання mAP на train при зниженні на val. Натомість обидві криві збіглися, що є позитивним результатом

Розподіл правильних та помилкових виявлень по класах наочно відображає матриця невідповідностей (рис.3.5 та рис.3.6).

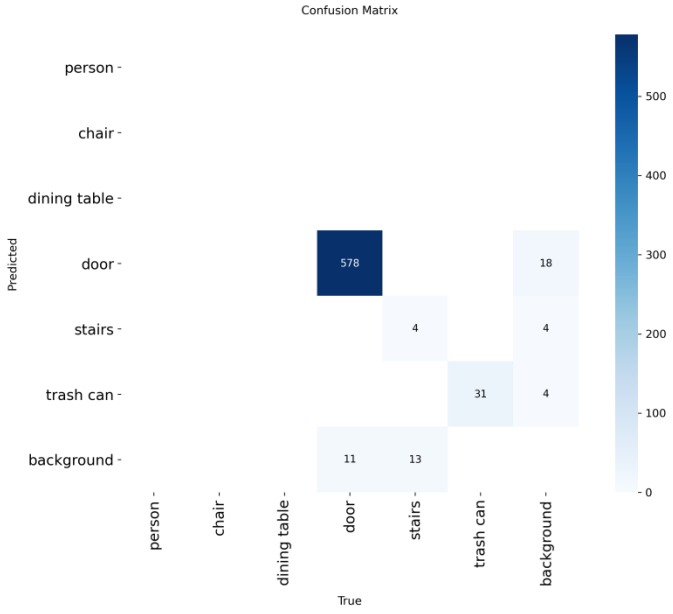

Рисунок 3.5 – Розподіл правильних виявлень по класам

Нормалізована матриця невідповідностей (рис.3.6) візуалізує типи помилок моделі для кожного класу. Для класів door та trash can діагональні елементи матриці близькі до 1.0, що свідчить про практично бездомішкову класифікацію. Для класу stairs значна частка об'єктів класифікується як фоновий клас (background), що пояснює низький Recall = 0.412: модель часто пропускає сходи, але якщо виявляє – класифікує правильно.

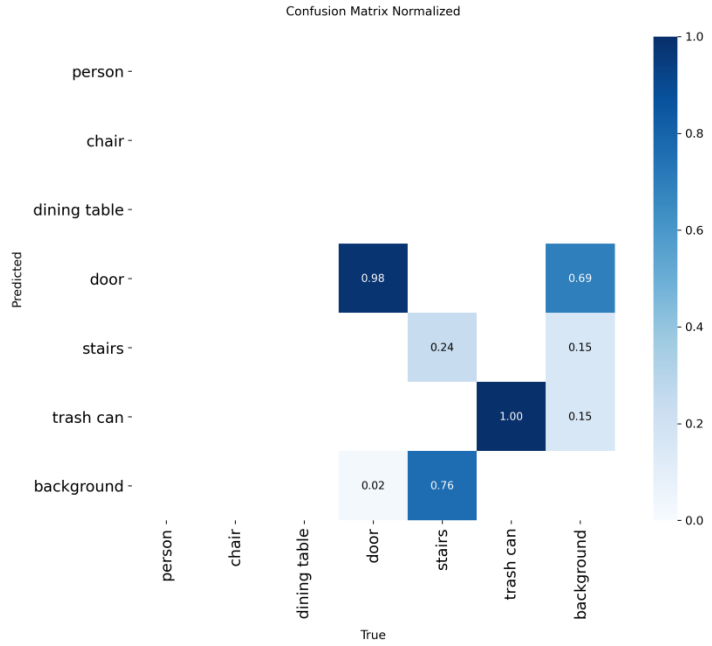

Рисунок 3.6 - Розподіл помилкових виявлень по класах

Порівняльну діаграму mAP50 до та після донавчання наведено на рисунку 3.7.

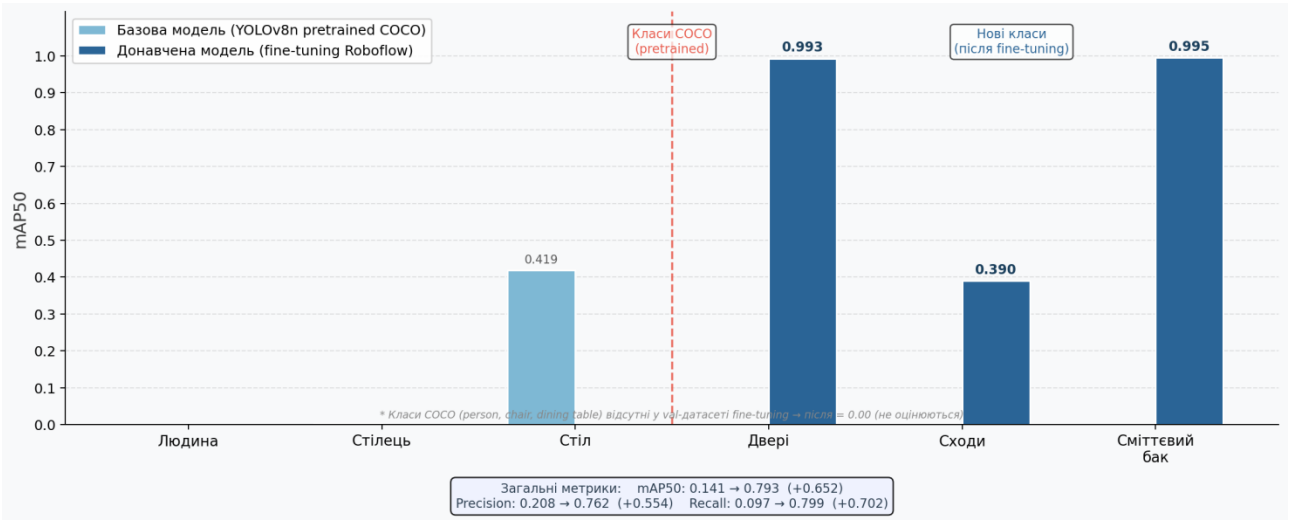

Рисунок 3.7 – Порівняння метрики mAP50 базової і донавченої моделі

Порівняльний аналіз результатів наведено у табл. 3.4.

Таблиця 3.4 – Порівняння результатів базової та донавченої моделі

Метрика	Базова (до)	Донавчена (після)	Абс. приріст Δ	Відн. приріст, %
mAP50 (загальне)	0.1409	0.7930	+0.6521	+462.8%
mAP50-95	0.1044	0.6480	+0.5436	+521.1%
Precision	0.2079	0.7620	+0.5541	+266.5%
Recall	0.0968	0.7990	+0.7022	+725.4%
mAP50: door	0.0000	0.9930	+0.9930	+∞
mAP50: stairs	0.0000	0.3900	+0.3900	+∞
mAP50: trash can	0.0000	0.9950	+0.9930	+∞

Клас door (mAP50 = 0.993). Виняткова точність виявлення дверей пояснюється кількома факторами: найбільший обсяг навчальних даних (2 042 зображення); характерна форма та текстура дверей є відносно простою для CNN; висока варіативність датасету (три стани дверей: зачинені, відчинені, напіввідчинені). Значення Precision = 0.963 та Recall = 0.986 свідчать про надзвичайно низький рівень як хибних спрацьовувань, так і пропущених об'єктів.

Клас trash can (mAP50 = 0.995). Найвищий показник серед трьох нових класів. Recall = 1.000 означає, що модель виявляє 100% реальних сміттєвих баків у тестовій вибірці – жодного пропуску. Незважаючи на відносно невеликий датасет (184 навчальні зображення), характерна циліндрична або прямокутна форма та контрастний вигляд сміттєвих баків забезпечує відмінну точність виявлення.

Клас stairs (mAP50 = 0.390). Найнижчий результат серед нових класів, що є прогнозованим наслідком критично малого обсягу датасету: лише 104 навчальні зображення проти 2 042 для door. Додатковою причиною є висока варіативність зовнішнього вигляду сходів (матеріал, ширина, освітлення, ракурс). Значення Precision = 0.459 та Recall = 0.412 вказують на приблизно однакову кількість помилок обох типів. Значення mAP50 = 0.390 перевищує

довільне виявлення (0.0), однак є недостатнім для надійного застосування в реальній системі.

Крива Precision-Recall (рис.3.8) демонструє, що для класів door та trash can крива залишається близькою до (1.0, 1.0) у широкому діапазоні порогів впевненості, що є ознакою відмінного балансу між точністю та повнотою. Для stairs крива має більш пологий характер, що відображає невизначеність моделі при виявленні цього класу.

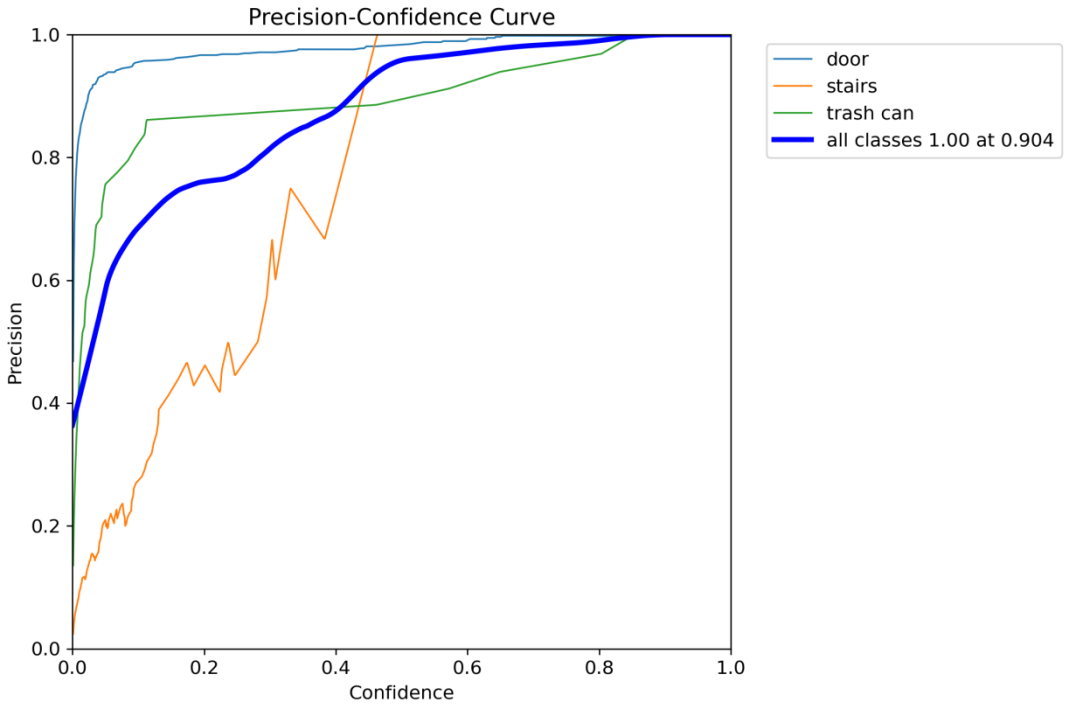

Рисунок 3.8 – Крива точночті розпізнавання для трьох класів

Крива Recall-Confidence (рисунок 3.9) відображає залежність повноти виявлення (Recall) від порогу впевненості (confidence threshold) для кожного з нових класів після донавчання.

Для класу door (блакитна крива) Recall залишається на рівні 0.95–1.00 у широкому діапазоні порогів від 0.0 до 0.80, і лише при порозі вище 0.85 спостерігається різке падіння. Це свідчить про те, що модель виявляє двері з високою впевненістю – переважна більшість об'єктів класу отримує оцінку confidence ≥ 0.85, що є ознакою надійного та стабільного розпізнавання.

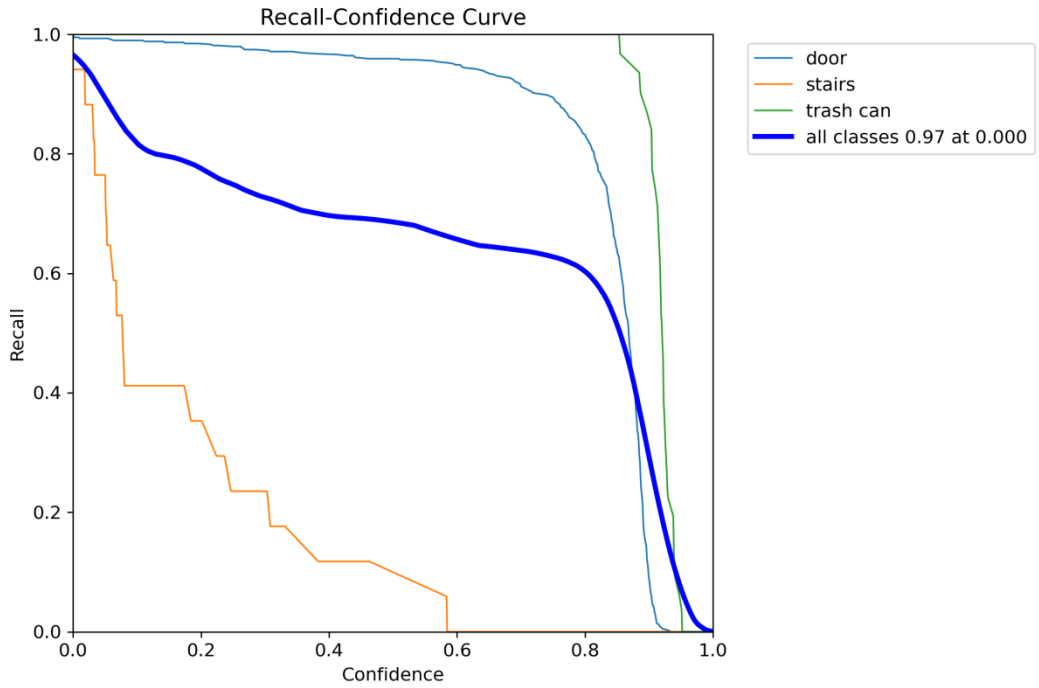

Рисунок 3.9 - Залежність повноти виявлення від порогу впевненості для кожного з нових класів після донавчання

Для класу trash can (зелена крива) поведінка є ще кращою: Recall = 1.00 зберігається практично до порогу 0.95, після чого відбувається різке вертикальне падіння. Це означає, що модель виявляє всі сміттєві баки з дуже високою впевненістю, а хибних виявлень з низьким confidence майже немає.

Для класу stairs (помаранчева крива) ситуація принципово відрізняється: навіть при порозі confidence = 0.0 Recall становить лише ~0.40, і вже при порозі 0.2 падає до 0.20, а при 0.5 – майже до нуля. Це підтверджує висновок про недостатній обсяг навчальних даних: модель здатна впевнено виявити лише близько 40% реальних сходів, тоді як решту або пропускає, або виявляє з дуже низькою оцінкою впевненості.

Загальна крива all classes (темно-синя) демонструє Recall = 0.97 при порозі 0.000, що підтверджує загальну високу повноту донавченої моделі. Рекомендований робочий поріг впевненості $\tau = 0.45$, встановлений у системі,

забезпечує прийнятний Recall для класів door та trash can при мінімізації хибних спрацьовувань.

На основі аналізу результатів визначено такі обмеження системи та напрямки її подальшого вдосконалення.

Крива F1-Confidence (рисунок 3.10) відображає залежність F1-міри від порогу впевненості для кожного класу після донавчання та дозволяє визначити оптимальний робочий поріг confidence для системи.

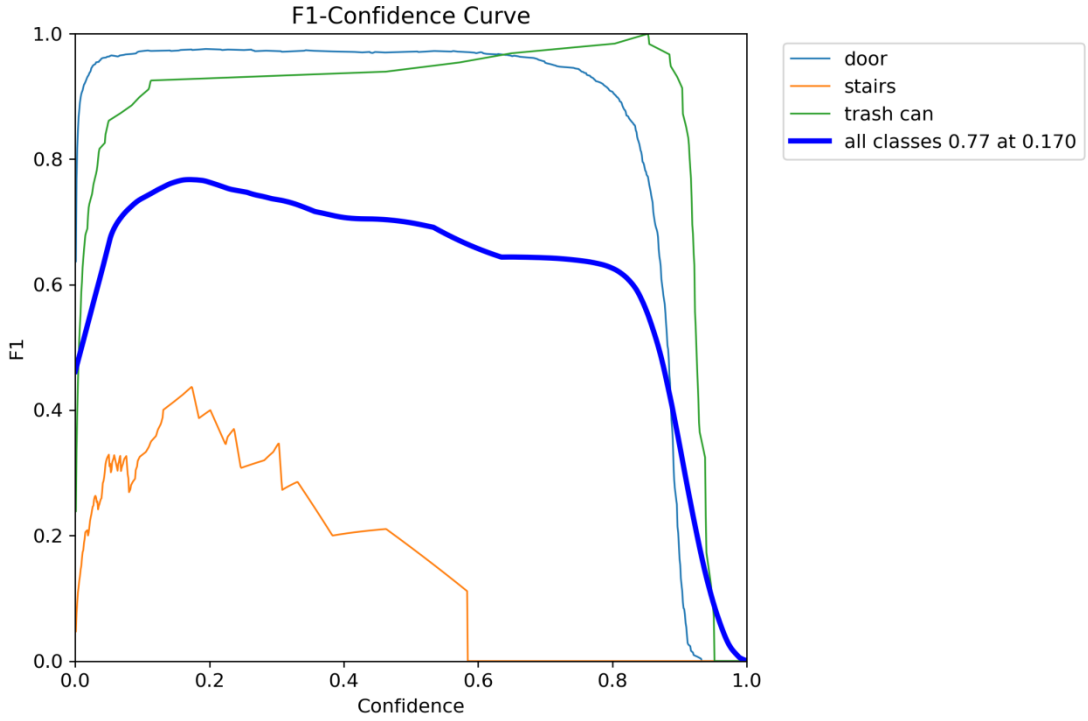

Рисунок 3.10 - Крива F1-Confidence для нових класів

Для класу door (блакитна крива) F1-міра залишається на рівні 0.93–0.96 у широкому діапазоні порогів від 0.05 до 0.85, що свідчить про стабільний баланс між точністю та повнотою у всьому робочому діапазоні. Плато на рівні ~0.95 підтверджує відмінну якість виявлення дверей незалежно від обраного порогу.

Для класу trash can (зелена крива) F1-міра досягає максимального значення ~0.98–1.00 у діапазоні порогів 0.1–0.90, що є найвищим показником серед усіх класів. Крива залишається практично горизонтальною на рівні ~0.97

у широкому діапазоні, що свідчить про надзвичайно надійне виявлення сміттєвих баків.

Для класу stairs (помаранчева крива) F1-міра є нестабільною та низькою протягом усього діапазону порогів: максимальне значення становить лише ~0.40 при порозі близько 0.15, після чого поступово знижується до нуля. Нестабільність кривої (помітні коливання) пояснюється малою кількістю тестових зображень сходів (17 зображень у val-вибірці), через що кожна помилка суттєво впливає на значення метрики.

Загальна крива all classes (темно-синя) досягає максимуму F1 = 0.77 при порозі confidence = 0.170. Це є рекомендованим порогом для максимального балансу між точністю та повнотою на рівні всієї системи. Проте у розробленій системі застосовано поріг $\tau = 0.45$ для більшості класів – він дещо нижчий за оптимальний для door та trash can, однак забезпечує вищий Recall що є пріоритетом для асистивної системи безпеки (краще зайвий раз попередити, ніж пропустити перешкоду).

Отже, практичний висновок: графік підтверджує правильність вибору $\tau = 0.45$ – при цьому порозі F1 для door (~0.95) та trash can (~0.98) залишається на максимальному рівні, тоді як для stairs F1 вже становить ~0.25, що є об'єктивним обмеженням через малий датасет.

3.6 Можливості адаптації алгоритму до суміжних задач

Розроблені алгоритмічно-програмні рішення мають модульну архітектуру, що забезпечує їх гнучку адаптацію до широкого кола суміжних задач виявлення перешкод. Ключовою властивістю, що уможливлює таку адаптацію, є чіткий поділ між трьома незалежними компонентами: CNN-детекцією (detector.py), просторовим аналізом (zone_analyzer.py) та реакцією системи (tts_engine.py або будь-який інший виконавчий модуль). Зміна одного компонента не потребує модифікації решти.

Мобільні роботи, що переміщуються у приміщеннях (сервісні, доставки, прибирання), стикаються з тією самою задачею, що і незряча людина: виявлення та обхід типових перешкод у реальному часі. Розроблений алгоритм придатний для вирішення цієї задачі з мінімальними змінами.

Що залишається незмінним: модуль detector.py з YOLOv8n виконує ту саму функцію CNN-детекції; ZoneAnalyzer визначає перешкоди у зоні руху; логіка визначення напрямку обходу (space_left vs space_right) безпосередньо відповідає концепції обходу перешкод у робототехніці.

Що змінюється: замість модуля tts_engine.py підключається модуль керування рухом робота; вихідні дані алгоритму (BypassDirection.LEFT / RIGHT / BLOCKED) перетворюються на команди моторів або планувальника маршруту (ROS – Robot Operating System); параметр frame_width ZoneAnalyzer налаштовується під поле зору камери робота; пороги центральної зони можуть бути скориговані залежно від ширини корпусу робота.

Принципова перевага використання розробленого алгоритму для роботів – наявність семантичної інформації про тип перешкоди. Знання про те, що попереду людина (динамічна перешкода) чи стілець (статична), дозволяє планувальнику маршруту застосовувати різні стратегії: очікування відходу людини або об'їзд статичного об'єкта.

Таблиця 3.5 – Порівняння конфігурацій для приміщень та вулиці

Параметр	Приміщення (поточна)	Вулиця (адаптована)
Цільові класи	person, chair, dining table, door, stairs, trash can	person, bicycle, car, scooter, pole, pothole
Датасет навчання	Roboflow (приміщення)	COCO + Open Images (вулиця)
Центральна зона	33%–66% (стандартна)	33%–66% (без змін)
Реакція системи	TTS: «Перешкода попереду»	TTS + вібросигнал (небезпека)
Що змінюється в коді	—	OBSTACLE_CLASSES у detector.py + нові ваги best.pt

Перехід від приміщень до зовнішнього середовища (вулиця, парк, тротуар) потребує лише оновлення множини цільових класів перешкод та донавчання моделі на відповідних датасетах. Архітектура алгоритму при цьому залишається незмінною.

Як видно з таблиці 3.5, адаптація для вулиці потребує лише двох змін: оновлення словника OBSTACLE_CLASSES у файлі detector.py та завантаження ваг моделі, донавченої на відповідних класах. Вся алгоритмічна логіка аналізу центральної зони, визначення напрямку обходу та генерування повідомлень залишається незмінною.

Ще одним напрямком застосування є промислові та складські приміщення, де переміщаються як люди, так і навантажувальна техніка. У таких середовищах множина небезпечних перешкод включає специфічні об'єкти: вилочні навантажувачі, паллети, промислові стелажі, зони з обмеженим доступом.

Адаптація алгоритму для цього середовища передбачає:

– розширення множини цільових класів до специфічних об'єктів виробництва через донавчання на галузевих датасетах;

– налаштування порогу блокування block_threshold залежно від ширини проходу (у вузьких складських коридорах доцільно знизити до 0.60–0.70);

– заміну TTS-модуля на систему сигналізації (зуммер, вібрація, світловий індикатор) або інтеграцію з SCADA-системою підприємства;

– додавання пріоритетів небезпеки: навантажувач (висока небезпека) → голосове попередження + зупинка; паллета (низька небезпека) → тихий сигнал.

Принципова перевага архітектури на основі YOLOv8 fine-tuning полягає у можливості нарощування переліку виявлюваних класів без переписування алгоритму. Процедура додавання нового класу перешкод складається з чотирьох кроків.

Крок 1. Збір датасету для нового класу (мінімум 300–500 анотованих зображень). Датасет може бути отриманий з Roboflow Universe, Open Images або власноруч зібраний та розмічений інструментом LabelImg.

Крок 2. Додавання нового класу до словника OBSTACLE_CLASSES у detector.py з відповідним українським перекладом та до списку CLASS_NAMES у train.py з наступним class_id.

Крок 3. Запуск донавчання командою python train.py --dataset dataset_new --epochs 50. Завдяки transfer learning нова модель навчиться розпізнавати новий клас, зберігаючи знання про попередні.

Крок 4. Заміна файлу ваг best.pt у робочій директорії — система готова до роботи з новим класом без жодних змін у логіці алгоритму.

Таким чином, розроблені алгоритмічно-програмні рішення є універсальною платформою для виявлення перешкод, що адаптується до нових середовищ та вимог шляхом зміни конфігураційних параметрів і ваг моделі без модифікації базового алгоритму.

Обмеження та напрямки покращення.

Обмеження 1: низька точність виявлення сходів. Основна причина – критично малий датасет (104 навчальні зображення). Для досягнення прийнятної точності (mAP50 $\geq$ 0.70) необхідно збільшити датасет до 1 000+ зображень. Рекомендується використати датасети з Open Images v7 або провести власну розмітку зображень.

Обмеження 2: відсутність класів COCO у val-вибірці. Поточна val-вибірка містить лише нові класи, тому реальна якість виявлення person, chair, dining table не підтверджена на нашому датасеті. Рекомендується створити змішану тестову вибірку з зображеннями всіх 6 класів у реальних умовах приміщення.

Обмеження 3: відсутність оцінки на реальному відеопотоці. Всі метрики отримані на статичних зображеннях тестової вибірки. Реальна продуктивність системи на відеопотоці може відрізнятись через рух камери, оклюзії та зміну освітлення. Рекомендується провести польові випробування у реальних умовах.

Напрямок розвитку 1. Збір та анотація власного датасету у реальних умовах приміщень для всіх 6 класів – це забезпечить оцінку системи в умовах, максимально близьких до реального застосування.

Напрямок розвитку 2. Розгортання на мобільному пристрої (Android смартфон або Raspberry Pi) та оцінка продуктивності у реальному часі з відеопотоком.

Напрямок розвитку 3. Додавання модуля оцінки відстані до перешкоди на основі розміру bbox у кадрі (більший bbox → менша відстань) для більш точного калібрування голосових попереджень.

У третьому розділі описано програмну реалізацію системи виявлення перешкод та проведено експериментальні дослідження.

Програмна система реалізована у вигляді чотирьох спеціалізованих модулів: detector.py (CNN-детекція YOLOv8n, фільтрація за класом та порогом), zone_analyzer.py (аналіз центральної зони, визначення напрямку обходу), tts_engine.py (асинхронне TTS-озвучення з cooldown), main.py (головний цикл, відображення overlay). Модульна архітектура забезпечує незалежність компонентів та можливість їх окремого тестування та заміни.

Для донавчання підготовлено датасет з 3 277 анотованих зображень трьох нових класів (door: 2 922, trash can: 231, stairs: 124) з відкритих датасетів Roboflow Universe. Виконано конвертацію схем іменування класів та уніфікацію структури датасету.

Проведено порівняльний експеримент між базовою моделлю (YOLOv8n pretrained COCO) та донавченою. Результати підтверджують ефективність стратегії fine-tuning: загальне mAP50 зросло з 0.1409 до 0.7930 (+462.8%); Precision зросла з 0.2079 до 0.7620 (+266.5%); Recall зріс з 0.0968 до 0.7990 (+725.4%). Для нових класів: door досяг mAP50 = 0.993, trash can — 0.995, stairs — 0.390 (обмежено малим датасетом). Навчання завершено достроково на 23-й епосі з 80 запланованих, що свідчить про ефективну збіжність без перенавчання.

Розроблена система має три ключові властивості, що забезпечують її широку застосовність: модульність (незалежність detector, zone_analyzer, tts_engine), параметричність (всі порогові значення задаються через конфігурацію без зміни коду) та розширюваність (додавання нових класів перешкод через fine-tuning без зміни алгоритму). Ці властивості дозволяють адаптувати систему для навігації мобільних роботів, зовнішнього середовища (вулиця) та виробничих приміщень (склад, завод) шляхом зміни лише двох артефактів: словника цільових класів та файлу ваг моделі.

Виявлено основне обмеження системи: недостатня точність виявлення сходів (mAP50 = 0.390) через малий датасет (104 навчальні зображення). Запропоновано напрямки подальшого вдосконалення: розширення датасету stairs, оцінка на реальному відеопотоці, розгортання на мобільному пристрої.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-практичну задачу розробки алгоритмічно-програмних рішень виявлення типових перешкод у приміщеннях на основі згорткової нейронної мережі з формуванням голосових повідомлень для осіб з порушенням зору. За результатами виконаної роботи отримано такі висновки.

1. Проведено аналіз існуючих методів виявлення перешкод. Встановлено, що класичні методи комп'ютерного зору (HOG, SVM, каскадні класифікатори) поступаються методам на основі CNN за точністю виявлення на 20–40% та не забезпечують достатньої швидкості для обробки відеопотоку у реальному часі. Здійснено порівняльний аналіз архітектур CNN-детекторів (Faster R-CNN, SSD, MobileNet SSD, YOLOv8) та обґрунтовано вибір YOLOv8n як оптимального рішення: модель забезпечує mAP50-95 = 37.3% на COCO при 80+ FPS та має лише 3.2 млн параметрів (розмір файлу 6.2 МБ), що є придатним для розгортання на мобільних платформах.

2. Визначено множину з шести типових перешкод для приміщень: людина, стілець, стіл, двері, сходи, сміттєвий бак. Три класи (person, chair, dining table) наявні у базовому наборі COCO та доступні у pretrained моделі без додаткового навчання. Три нові класи (door, stairs, trash can) потребували донавчання. Обґрунтовано практичну цінність системи для двох напрямків застосування: асистивна технологія для незрячих при пересуванні у приміщеннях та навігація мобільних роботів.

3. Розроблено шестиетапний алгоритм виявлення перешкод: захоплення кадру → попередня обробка (resize 640×640, нормалізація) → CNN-детекція → фільтрація за класом та порогом впевненості → аналіз центральної зони → визначення напрямку обходу. Алгоритм утворює замкнений безперервний цикл обробки відеопотоку. Обґрунтовано всі ключові параметри: індивідуальні пороги впевненості $\tau = 0.45$ (загальний) та $\tau = 0.30$ (для person —

для надійного виявлення дітей); межі центральної зони [33%–66%] ширини кадру з двома критеріями перевірки (центр bbox та overlap ≥ 40%); поріг блокування 80% ширини кадру.

4. Розроблено алгоритм генерування голосових повідомлень з трьома типами фраз залежно від напрямку обходу («Перешкода попереду: {клас}. Обійдіть ліворуч/праворуч» та «Прохід заблоковано — зупиніться»). Впроваджено механізм cooldown (2.5 с) для запобігання надмірному повторенню повідомлень та асинхронне виконання TTS у daemon-потоці для забезпечення безперервної обробки відеопотоку.

5. Реалізовано програмну систему у вигляді чотирьох спеціалізованих модулів на Python 3.10: detector.py (CNN-детекція та фільтрація), zone_analyzer.py (просторовий аналіз та визначення напрямку), tts_engine.py (голосове озвучення з підтримкою pyttsx3 та gTTS), main.py (головний цикл, відображення overlay, CLI-інтерфейс). Модульна архітектура забезпечує незалежність компонентів та можливість їх окремого тестування та заміни.

6. Підготовлено датасет для донавчання з 3 277 анотованих зображень трьох нових класів з відкритих датасетів Roboflow Universe (door: 2 922, trash can: 231, stairs: 124). Виконано конвертацію схем іменування класів та уніфікацію структури датасету. Проведено fine-tuning YOLOv8n на GPU NVIDIA Tesla T4 (Google Colab): заморожування перших 10 шарів backbone зберегло знання про класи COCO; навчання завершено на 23-й епосі з 80 запланованих завдяки механізму early stopping, що свідчить про ефективну збіжність без перенавчання.

7. Проведено порівняльний експеримент між базовою та донавченою моделями. Результати підтверджують ефективність стратегії fine-tuning: загальне mAP50 зросло з 0.141 до 0.793 (+462.8%); Precision — з 0.208 до 0.762 (+266.5%); Recall — з 0.097 до 0.799 (+725.4%). Для нових класів: door досяг mAP50 = 0.993, trash can — 0.995. Клас stairs показав mAP50 = 0.390 через обмежений обсяг датасету (104 навчальні зображення), що визначено як

основне обмеження системи. Аналіз кривих F1-Confidence підтвердив оптимальність обраного порогу $\tau = 0.45$ для класів door та trash can ($F1 \approx 0.95–0.98$ у робочому діапазоні).

8. Досліджено можливості адаптації розробленого алгоритму до суміжних задач. Встановлено, що модульна архітектура системи дозволяє адаптувати її для навігації мобільних роботів (заміна TTS-модуля на команди керування рухом), зовнішнього середовища (оновлення словника класів та ваг моделі) та виробничих приміщень (склад, завод) без модифікації базового алгоритму. Процедура додавання нового класу перешкод включає чотири кроки: збір датасету → оновлення конфігурації → донавчання → заміна ваг.

. Розроблені алгоритмічно-програмні рішення є практично придатними для створення асистивних технологій для незрячих та систем навігації мобільних роботів у приміщеннях. Перспективами подальшого розвитку є: розширення датасету для класу stairs (до 1 000+ зображень) для підвищення mAP50 до цільового рівня ≥ 0.70; розгортання системи на мобільному пристрої (Android / Raspberry Pi) та польові випробування у реальних умовах; додавання модуля оцінки відстані до перешкоди на основі розміру bbox.

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Yaseen M. What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector. arXiv preprint. 2024. arXiv:2408.15857. URL: https://arxiv.org/abs/2408.15857 (дата звернення: 10.06.2025).

2. Talib M., Al-Noori A. H. Y., Suad J. YOLOv8-CAB: Improved YOLOv8 for Real-Time Object Detection. Karbala International Journal of Modern Science. 2024. Vol. 10, Iss. 1. DOI: 10.33640/2405-609X.3339.

3. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, USA: IEEE, 2016. P. 779–788. DOI: 10.1109/CVPR.2016.91.

4. Varghese R., Sambath M. YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness. Proceedings of the International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS). IEEE, 2024. P. 1–6.

5. Ezzeddini L., Ktari J., Frikha T. et al. Analysis of the Performance of Faster R-CNN and YOLOv8 in Detecting Fishing Vessels and Fishes in Real Time. PeerJ Computer Science. 2024. DOI: 10.7717/peerj-cs.2033.

6. Ben Atitallah A., Said Y., Ben Atitallah M. A. et al. An Effective Obstacle Detection System Using Deep Learning Advantages to Aid Blind and Visually Impaired Navigation. Ain Shams Engineering Journal. 2024. Vol. 15, Iss. 2. DOI: 10.1016/j.asej.2023.102387.

7. Mukhiddinov M., Cho J. Deep Learning Based Object Detection and Surrounding Environment Description for Visually Impaired People. Heliyon. 2023. DOI: 10.1016/j.heliyon.2023.e17405.

8. Obstacle Detection System for Navigation Assistance of Visually Impaired People Based on Deep Learning Techniques. Sensors. 2023. Vol. 23, № 11. DOI: 10.3390/s23115262.

9. De Luigi L., Cardace A., Spezialetti R. et al. Transfer Learning with Generative Models for Object Detection on Limited Datasets. arXiv preprint. 2024. arXiv:2402.06784. URL: https://arxiv.org/abs/2402.06784 (дата звернення: 10.06.2025).

10. Dalal N., Triggs B. Histograms of Oriented Gradients for Human Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). San Diego, USA: IEEE, 2005. Vol. 1. P. 886–893. DOI: 10.1109/CVPR.2005.177.

11. Zou Z., Chen K., Shi Z. et al. Object Detection in 20 Years: A Survey. Proceedings of the IEEE. 2023. Vol. 111, № 3. P. 257–276. DOI: 10.1109/JPROC.2023.3238524.

12. Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Advances in Neural Information Processing Systems (NIPS). 2015. Vol. 28. P. 91–99.

13. Liu W., Anguelov D., Erhan D. et al. SSD: Single Shot MultiBox Detector. Computer Vision – ECCV 2016. Lecture Notes in Computer Science. Cham: Springer, 2016. Vol. 9905. P. 21–37. DOI: 10.1007/978-3-319-46448-0_2.

14. Lin T.-Y., Maire M., Belongie S. et al. Microsoft COCO: Common Objects in Context. Computer Vision – ECCV 2014. Lecture Notes in Computer Science. Cham: Springer, 2014. Vol. 8693. P. 740–755. DOI: 10.1007/978-3-319-10602-1_48.

15. Reis D., Kupec J., Hong J., Daoudi A. Real-Time Flying Object Detection with YOLOv8. arXiv preprint. 2023. arXiv:2305.09972. URL: https://arxiv.org/abs/2305.09972 (дата звернення: 10.06.2025).

16. Xie Z., Li Z., Zhang Y. et al. A Multi-Sensory Guidance System for the Visually Impaired Using YOLO and ORB-SLAM. Information. 2022. Vol. 13, № 7. DOI: 10.3390/info13070343.

17. Masud M. et al. Smart Assistive System for Visually Impaired People Obstruction Avoidance Through Object Detection and Classification. IEEE Access. 2022. Vol. 10. P. 428–443. DOI: 10.1109/ACCESS.2022.3146320.

18. Bouteraa Y. Smart Real Time Wearable Navigation Support System for BVIP. Alexandria Engineering Journal. 2023. Vol. 62. P. 223–235. DOI: 10.1016/j.aej.2022.06.060.

19. Arifando R., Eto S., Wada C. Improved YOLOv5-Based Lightweight Object Detection Algorithm for People with Visual Impairment to Detect Buses. Applied Sciences. 2023. Vol. 13, № 9. P. 5802. DOI: 10.3390/app13095802.

20. Pandilova E., Petrov M. Transfer Learning with YOLO for Object Detection in Remote Sensing. ICT Innovations 2024. Lecture Notes in Networks and Systems. Cham: Springer, 2024. DOI: 10.1007/978-3-031-54321-4.

21. Jocher G., Chaurasia A., Qiu J. Ultralytics YOLOv8. Version 8.0.0. 2023. URL: https://github.com/ultralytics/ultralytics (дата звернення: 10.06.2025).

22. Bradski G. The OpenCV Library. Dr. Dobb's Journal of Software Tools. 2000. URL: https://opencv.org (дата звернення: 10.06.2025).

23. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement. arXiv preprint. 2018. arXiv:1804.02767. URL: https://arxiv.org/abs/1804.02767 (дата звернення: 10.06.2025).

24. Girshick R. Fast R-CNN. Proceedings of the IEEE International Conference on Computer Vision (ICCV). Santiago, Chile: IEEE, 2015. P. 1440–1448. DOI: 10.1109/ICCV.2015.169.

25. Roboflow Inc. Roboflow Universe: Open-Source Computer Vision Datasets. 2023. URL: https://universe.roboflow.com (дата звернення: 10.06.2025).

26. Shorten C., Khoshgoftaar T. M. A Survey on Image Data Augmentation for Deep Learning. Journal of Big Data. 2019. Vol. 6, № 1. P. 1–48. DOI: 10.1186/s40537-019-0197-0.